28th International Conference on
Automated Planning and Scheduling

June 24-29, 2018, Delft, the Netherlands

PlanRob 2018

Proceedings of the 6th Workshop on

Planning and Robotics

Edited by:

Alberto Finzi, Erez Karpas, Goldie Nejat,
AndreA Orlandini, Siddharth Srivastava

Organization

Alberto Finzi
Università di Napoli "Federico II", Italy

Erez Karpas
Technion - Israel Institute of Technology, Israel

Goldie Nejat
University of Toronto, Canada

AndreA Orlandini
ISTC-CNR, Italy

Siddharth Srivastava
Arizona State University, Arizona, USA.

Program Committee

Chris Beck (University of Toronto, Canada)
Arthur Bit-Monnot (LAAS, France)
Amedeo Cesta (ISTC-CNR, Italy)
Marcello Cirillo (Scania, Sweden)
Patrick Doherty (Linkoping University, Sweden)
Alberto Finzi (Universita' di Napoli Federico II, Italy)
Robert Fitch (University of Sydney, Australia)
Nick Hawes (University of Oxford, UK)
Felix Ingrand (LAAS-CNRS, France)
Erez Karpas (Technion, Israel)
Sven Koenig (University of Southern California, USA)
Daniele Magazzeni (King's College, UK)
Karen Myers (SRI, USA)
Daniele Nardi (Sapienza Universià di Roma, Italy)
Goldie Nejat (University of Toronto)
AndreA Orlandini (ISTC-CNR, Italy)
Frederic Py (University of Porto, Portugal)
Marco Roveri (FBK, Italy)
Enrico Scala (FBK, Italy)
Siddharth Srivastava (University of Arizona, USA)
Sebastian Stock (DFKI, Germany)
Alessandro Umbrico (ISTC-CNR, Italy)
Tiago Stegun Vaquero (JPL, USA)

Additional Reviewers

Jiaoyang Li (University of Southern California, USA)
Hang Ma (University of Southern California, USA)

Foreword

AI Planning & Scheduling (P&S) methods are crucial to enable intelligent robots to perform autonomous, flexible, and interactive behaviors, but a deep integration into robotic architectures is needed for their effective deployment. This requires a close collaboration between researchers from both the AI and the Robotics communities. In this direction, the PlanRob workshop aims at constituting a stable, long-term forum where researchers from both the P&S and the Robotics communities can openly discuss about relevant issues, research and development progress, future directions and open challenges related to P&S when applied to Robotics.

Started during ICAPS 2013 in Rome (Italy), followed by several editions at ICAPS 2014 in Portsmouth (NH, USA), ICAPS 2015 in Jerusalem (Israel), ICAPS 2016 in London (UK), and ICAPS 2017 in Pittsburgh, the PlanRob WS series[1] has gathered excellent feedback from the P&S community which is also confirmed by the organisation of a specific Robotics Track starting (since ICAPS 2014) and a Dagstuhl seminar on Planning and Robotics in 2017. This sixth edition of the PlanRob workshop has been proposed in synergy with the Robotics Track to further enforce its original goals and to maintain an informal forum where more preliminary/visionary works can be discussed. PlanRob 2018 succeeded in achieving these objectives providing a rich and articulated program. Indeed, 19 papers have been accepted for oral presentation covering many relevant topics in Planning and Robotics such as mission planning, multi-robot collaboration, activity/goal recognition, task and motion planning, integrated planning and execution, interactive plan execution, real applications and case studies. The workshop program is completed by an invited talk and a discussion panel.

The varieties of research topics and results collected in these proceedings reflect a stimulating and intense research activity along with a growing interest for a forum where the Planning and Robotics communities can find a common ground. Among the numerous people that contribute to the success of PlanRob 2018, we would first of all like to thank the authors of the submitted papers and all workshop attendees. Moreover, we sincerely thank the program committee for their important work on the reviewing process.

Alberto Finzi, Erez Karpas, AndreA Orlandini, Goldie Nejat and Siddharth Srivastava
June 2018

[1] http://pst.istc.cnr.it/planrob/

Contents

Self-Reliant Rover Design for Increasing Mission Productivity
Daniel Gaines, Joseph Russino, Gary Doran, Ryan Mackey, Michael Paton, Brandon Rothrock, Steve Schaffer, Ali-Akbar Agha-Mohammadi, Chet Joswig, Heather Justice, Ksenia Kolcio, Jacek Sawoniewicz, Vincent Wong, Kathryn Yu, Gregg Rabideau, Robert Anderson and Ashwin Vasavada **1**

Dynamic Shared Computing Resources for Multi-Robot Mars Exploration
Joshua Vander Hook, Tiago Stegun Vaquero, Martina Troesch, Jean-Pierre de La Croix, Joshua Schoolcraft, Saptarshi Bandyopadhyay and Steve Chien **11**

An Approach for Autonomous Multi-rover Collaboration for Mars Cave Exploration: Preliminary Results
Tiago Vaquero, Martina Troesch and Steve Chien **19**

Using Squeaky Wheel Optimization to Derive Problem Specific Control Information for a One Shot Scheduler for a Planetary Rover
Wayne Chi, Jagriti Agrawal and Steve Chien **27**

Front Delineation and Tracking with Multiple Underwater Vehicles
Andrew Branch, Mar M. Flexas, Brian Claus, Andrew F. Thompson, Evan B. Clark, Yanwu Zhang, James C. Kinsey, Steve Chien, David M. Fratantoni, Brett Hobson, Brian Kieft and Francisco P. Chavez **36**

Autonomous Nested Search for Hydrothermal Venting
Andrew Branch, Guangyu Xu, Michael V. Jakuba, Christopher R. German, Steve Chien, James C. Kinsey, Andrew D. Bowen, Kevin P. Hand and Jeffrey S. Seewald **46**

Using a Hybrid AI-Planner to Plan Feasible Flight Paths for HAPS-Like UAVs
Jane Jean Kiam, Enrico Scala, Miquel Ramirez and Axel Schulte **55**

A Dynamic Task Planning System for Advanced Manufacturing Scenarios
Amedeo Cesta, Andrea Orlandini and Alessandro Umbrico **65**

Integrating Classical Planning and Real Robots in Industrial and Service Robotics Domains
Oscar Lima, Rodrigo Ventura and Iman Awaad **75**

Interactive Plan Execution during Human-Robot Cooperative Manipulation
Jonathan Cacace, Riccardo Caccavale, Alberto Finzi and Vincenzo Lippiello **82**

Action trees for scalable goal recognition in robotic applications
Helen Harman, Keshav Chintamani and Pieter Simoens **89**

Robust Human Activity Monitoring Using Qualitative Spatial Representation and Reasoning
Sang Uk Lee, Ashkan Jasour, Andreas Hofmann and Brian Williams **94**

Domain Reasoning for Robot Task Planning - A Position Paper
Uwe K öckemann, Ali Abdul Khaliq, Federico Pecora and Alessandro Saffiotti **102**

Improving Trajectory Optimization using a Roadmap Framework
Siyu Dai, Matthew Orton, Shawn Schaffert, Andreas Hofmann and Brian Williams **106**

An Anytime Algorithm for Task and Motion MDPs
Siddharth Srivastava, Nishant Desai, Richard Freedman and Shlomo Zilberstein **115**

A Unified Framework for Planning in Adversarial and Cooperative Environments
Anagha Kulkarni, Siddharth Srivastava and Subbarao Kambhampati **123**

What do you really want to do? Representing and Reasoning with Intentional Actions on a Robot
Rocio Gomez, Mohan Sridharan and Heather Riley **133**

Sound and Complete Reactive UAV Behavior using Constraint Programming
Hoang Tung Dinh, Mario Henrique Cruz Torres and Tom Holvoet **143**

Combining Planning and Model Checking to Get Guarantees on the Behavior of Safety-Critical UAV Systems
Hoang Tung Dinh, Mario Henrique Cruz Torres and Tom Holvoet **152**

Self-Reliant Rover Design for Increasing Mission Productivity

Daniel Gaines, Joseph Russino, Gary Doran, Ryan Mackey, Michael Paton, Brandon Rothrock, Steve Schaffer, Ali-akbar Agha-mohammadi, Chet Joswig, Heather Justice, Ksenia Kolcio, Jacek Sawoniewicz, Vincent Wong, Kathryn Yu, Gregg Rabideau, Robert Anderson, Ashwin Vasavada

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, California 91109
{*firstname.lastname*}@jpl.nasa.gov

Okean Solutions
1463 E Republican St 32A
Seattle, Washington 98112
ksenia@okeansolutions.com

Abstract

Achieving consistently high levels of productivity has been a challenge for Mars surface missions. While the rovers have made major discoveries and dramatically increased our understanding of Mars, they often require a great deal of effort from the operations teams, and achieving mission objectives can take longer than anticipated. The objective of this work is to identify changes to flight software and ground operations that enable high levels of productivity with reduced reliance on ground interactions. This will enable the development of Self-Reliant Rovers: rovers that make use of high-level guidance from operators to select their own situational activities and respond to unexpected conditions, all without dependence on ground intervention. In this paper we describe the system we are developing and illustrate how it enables increased mission productivity.

Introduction

Maintaining high productivity for the Mars exploration rover missions is very challenging. While the operations teams have achieved impressive accomplishments with the rovers, doing so often requires significant human effort in planning, coordinating, sequencing, and validating command products for the robots. A primary reason for these productivity challenges is the heavy reliance on interaction between the rovers and ground operators in order to accomplish mission objectives. For example, prior rovers depend on operators to provide a detailed schedule of activities, select science targets, navigate around slip hazards, and recover from anomalies. When combined with the limited communication opportunities between the rovers and human operators, this reliance on ground interaction results in under-utilization of vehicle resources and increased days on Mars to accomplish mission objectives.

The objective of our work is to identify changes to flight software and mission operations that improve rover efficiency and reduce dependency on ground interactions. This will facilitate the development of Self-Reliant Rovers: rovers that make use of high-level guidance from operators to select their own situational activities and respond to unexpected conditions, all with reduced reliance on human intervention.

Although our objective is to reduce the reliance on ground support in order to promote productivity, we are by no means attempting to remove human operator involvement. To the contrary, our objective is to increase the scope of operator input so that operators can effectively guide rover activity without requiring up to date knowledge of the rover and its environment.

This paper will present the Self-Reliant Rover design and illustrate how it enables rovers to maintain high levels of productivity. In this paper, we will highlight four main components of the design:

Campaign Intent: Allows operators to provide the rover with high-level guidance over the rover's activity planning and autonomous science

Slip-aware navigation: Enables the rover to assess the amount of predicted slip in its environment and plan safe paths to avoid both geometric and slip hazards.

Model-based health assessment: Improves the rover's ability to detect and isolate problems, and increases the range of problems from which it can recover on its own

Global localization: Enables the rover to remove positional knowledge error that accumulates during navigation

Overview of the Self-Reliant Rover Design

We are designing the Self-Reliant Rover system within the context of the Jet Propulsion Laboratory flight software architecture (Weiss 2013). Figure 1 provides an overview of this architecture and the changes we are introducing.

The Jet Propulsion Laboratory (JPL) architecture consists of components organized into three layers: behaviors, activities, and functions. Each successive layer has a reduced degree of autonomy, fewer interactions with other components, and a narrower scope of system knowledge.

Behavior: Collection of autonomously scheduled activities in service of an over-arching mission goal. Contains broad system knowledge.

Activity: Coordinates function invocations to achieve some high-level spacecraft task. Encompasses knowledge local to the activity being managed.

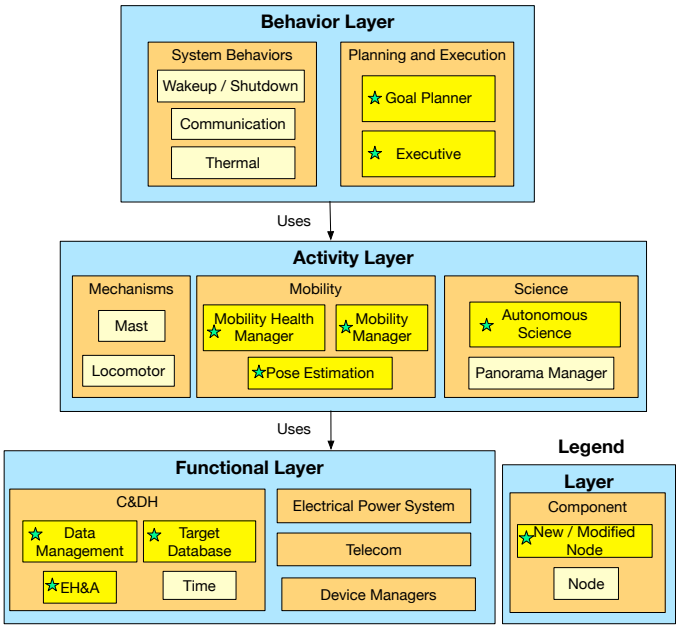


Figure 1: Self-Reliant Rover flight software architecture.

Function: Primitive action required to achieve a single well-delineated spacecraft objective. Contemplates only highly-localized function-specific knowledge.

Following is a summary of the changes we are introducing for the Self-Reliant Rover approach. Subsequent sections will provide more details on the most significant changes.

Goal Planner: Generates onboard activity plans to accomplish mission goals. Improves resource utilization by synchronizing plans with in-situ vehicle resource knowledge. Responds to new goals identified by onboard autonomous science.

Executive: Executes plans generated by the Goal Planner and provides updates to facilitate re-planning.

Autonomous Science: Identifies science targets when the rover enters an unexplored area. Increases the scope of guidance that scientists can provide and deepens the integration with onboard planning, as compared with previous autonomous science on MSL (Francis *et al.* 2017).

Mobility Manager: Improves navigation by reasoning about terrain-dependent slip.

Mobility Health Manager: Increases the robustness of mobility activities and the scope of faults from which the rover can autonomously recover by leveraging model-based fault detection and isolation.

Pose Estimation: Maintains high quality position knowledge over long traverse distances via onboard global localization (a technique that previously required ground operator support).

Target Database: Facilitates communication about targets of interest among scientists, engineers, and onboard autonomous components by leveraging previous ground operations tools onboard.

Data Management: Provides queryable onboard data product access to autonomous components such as onboard science analysis.

EH&A: Provides onboard access to engineering, housekeeping, and accountability telemetry for use by autonomous reasoning components.

Campaign Intent for Operator Guidance

A significant challenge to maintaining high rover productivity under reduced operator interaction is conveying operator guidance and objectives without requiring operators have up to date knowledge of the rover and its environment. Our approach is motivated by prior operations practice. In traditional operations, each planning cycle begins with a recapitulation of the current long term objectives of the mission presented in the context of the latest available rover state data (Chattopadhyay *et al.* 2014). The human operators assimilate all the various objectives, state data, and mission knowledge in order to synthesize a high quality plan that makes progress toward the goals while respecting limited rover resources such as time, energy, and data volume.

The team will typically have several high-level objectives to pursue. For example, during MSL's Pahrump Hills Walkabout campaign, the primary focus of the mission was to collect observations of exposed outcrop forming the basal layer of Mount Sharp (Gaines *et al.* 2016). This required driving the rover to several locations and acquiring high quality Mastcam and ChemCam observations selected locally at each stop.

Concurrently, the team also pursued a variety of supplementary objectives. During this campaign, Siding Spring (Comet C/2013 A1) would pass Mars closer than any other known comet flyby of Earth or Mars. The operations team thus incorporated comet observations into the rover plans. In addition, the team planned ongoing periodic observations to study clouds, dust devils, and atmospheric opacity. A wide range of recurring engineering activities also had to be included: instrument calibrations, telemetry collection, and system configuration management.

Importantly, the quality of the plan is not just a function of what activities are scheduled; it depends on how well they relate to the current objectives and to each other. Each individual outcrop observation was valuable, but understanding the geology of the region required accumulation of a variety of observations that were spatially distributed throughout the area. Periodic tasks such as atmospheric measurements and engineering activities had similar preferred temporal patterns that the team must try to match.

We developed the concept of *campaign intent* to convey such information to the rover so that it may generate its own prudent in-situ plans when human guidance is prohibitively delayed. Campaign intent specifies a set of goals for the rover and the relationships among those goals. We gleaned three initial types of campaign intent from MSL scenarios, as summarized in Figure 2:

Class sampling: Choose observation targets that best exemplify a particular feature (e.g. layering). Once identified, the targets form a goal set. Value typically accumu-

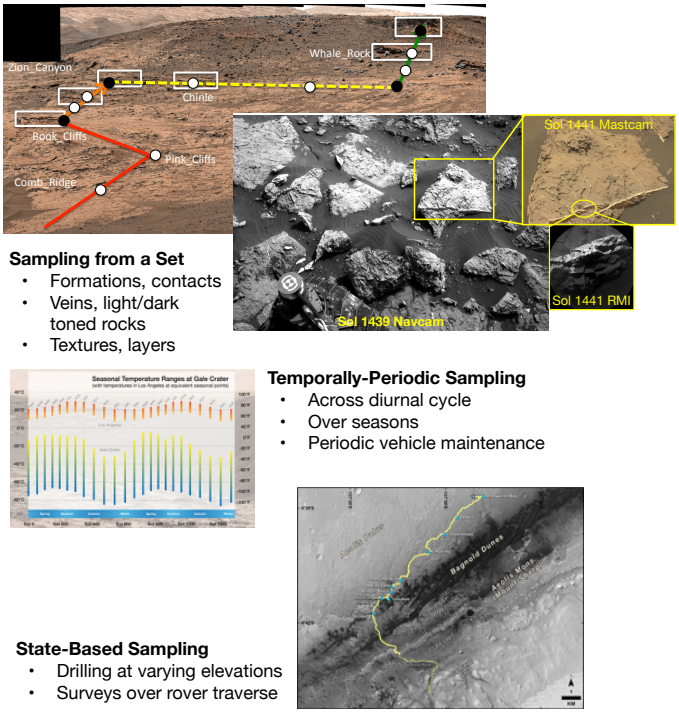


Figure 2: Summary of campaign intent types.

lates with additional samples from the set, but eventually reaches a point of diminishing returns.

Temporally-Periodic sampling: Schedule goals to match a repeating temporal pattern (e.g. hourly). The preferred goal cadence typically allows at least some timing flexibility.

State-based sampling: Trigger goals based on the evolution of the rover/terrain state (e.g. at every 50m traveled). The state criteria is typically expressed as a preferred cadence with some flexibility.

Using Campaign Intent to Guide Planning

Our approach to plan generation is based on branch-and-bound search. Starting from the empty plan, each iteration of search expands a chosen partial plan into many possible successor plans (the branches). Each potential successor is scored and must exceed a running threshold of plan quality (the bound) in order to be retained for future expansion; otherwise it is pruned (along with all its descendants). Specifically, the optimistic maximum quality of any plan based on the candidate partial plan must exceed the pessimistic minimum quality prediction of all other candidates already considered. Plan quality is evaluated as the degree of satisfaction of the campaign intents, which may be both priority tiered and utility weighted by the user. The frontier of unexpanded partial plans is periodically sorted by estimated final plan quality, yielding a hybrid of depth-first and best-first expansion order.

Partial plans are always expanded forward in time by appending one of the possible subsequent actions to the growing plan. The possible actions include mandatory goals (such as communication passes), auxiliary actions (such as sleep periods), as well as all the possible goals introduced by campaign intents. For temporal and state-based campaigns, this is just the next instance of the periodic goal, timed within its allowed cadence. For unordered goal set campaigns, each remaining un-attempted goal becomes a possible addition. In the limit, the search will thus evaluate (or justifiably prune) all possible combinations and orderings of campaign goals.

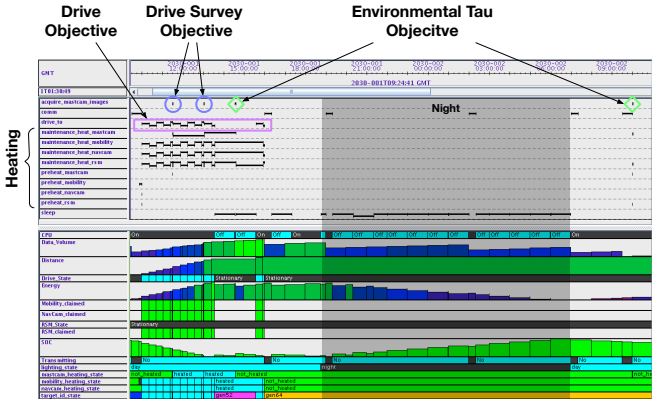


Figure 3: Example generated plan illustrating a long-range drive objective that was split up to support two different types of campaign objectives.

The complete search can be very time intensive, but is guaranteed to return an optimal plan according to the expressed campaign preferences. Even without running to completion, the search can return the best plan encountered so far. This anytime algorithm feature allows the rover to limit its planning time and proceed to be productive with a reasonable (but not provably optimal) plan. Minor plan perturbations during execution are accommodated by time-efficient repair strategies (for example, to shift actions forward after a small driving delay), while major disruptions (such as an insurmountable obstacle in a drive, or the injection of an entirely new goal) invoke a full replanning cycle so that all goals are reconsidered.

Figure 3 shows an example plan generated by the search algorithm. The planning model derives from the operational MSL activity model and features important mission aspects such as science campaign activities, communication windows, regenerative sleeping, and device heating.

The campaign objectives provided to the rover in this example include: a goal set campaign with a distant MastCam target (entailing a long-range traverse), a temporal campaign with recurring atmospheric opacity (tau) measurements every 3 hours, and state-based campaign with mid-drive survey actions after every 75 meters traveled. The resultant plan demonstrates how the planner synthesizes the campaign relationships to coordinate rover activity, including pausing the ongoing drive action to interleave other objectives.

Using Campaign Intent to Guide Autonomous Science

The system also leverages high-level campaign objectives to introduce additional in-situ goals based on scientist guidance. This improves rover productivity when the operations team does receive data about the rover's environment in time to select their own local targets for that day.

For example, scientists may be interested in remote-sensing composition measurements of a rock formation encountered previously and known to exist in a region the rover is approaching. The scientists can train a TextureCam (Thompson *et al.* 2012) model to detect that rock formation by labeling examples in previous navigation camera images (Figure 4, left). The rover then runs that TextureCam model onboard to compute a probability map of locations in the new region that likely contain the rock formation of interest (Figure 4, center). The probability map can be used to select the best targets for measurement, as well as the likelihood that each measurement satisfies the scientific intent of characterizing the rock formation (Figure 4, right). Each proposed target becomes a new goal in the campaign set for the planner. The planner may also use the probability information to reason about the trade offs between the various generated goals.

Slip-Aware Navigation

The Navigation systems equipped on the Mars rover missions, Mars Exploration Rover (MER) and Mars Science Laboratory (MSL), rely on the Grid-based Estimation of Surface Traversability Applied to Local Terrain (GESTALT) algorithm (Goldberg, Maimone, & Matthies 2002) to detect and avoid geometric hazards and the $\mathbf{D}^*$ algorithm (Stentz & Mellon 1993) to plan global paths to goals. These methods have enabled operators to provide high-level autonomy goals to the rovers, increasing mission efficiency.

However, geometry alone is not sufficient to guarantee safe traverses on the surface of Mars in every environment. Both MER and MSL operators have experienced hazardous conditions due to otherwise geometrically benign terrain such as sand dunes, and small rocks. These hazards can create adverse conditions such as wheel slip, sinkage, and damage. When current rovers pass through these hazardous environments, operators control the rovers manually with slow, deliberate commands, resulting in a loss in efficiency. In response, this paper proposes a navigation system that can reason about geometry *and* terrain type to plan safe reliable paths to science targets and enable a larger role in autonomy for future Mars Rovers.

System Overview The slip-aware navigation system, highlighted in Figure 5, is built upon the GESTALT system (Goldberg, Maimone, & Matthies 2002) and contains the following components: i) stereo vision, ii) visual odometry, iii) traversability assessment, iv) terrain classification, and v) path planning. The input to system is a synchronized pair of stereo images from the rover's navigation cameras. Image data is sent to the OpenCV (Bradski 2000) block matching algorithm to obtain dense 3D information about the environment. In parallel, the left stereo image is sent to a speeded-up version of the Soil Property and Object Clasification (SPOC) (Rothrock *et al.* 2016) terrain classifier more suited for on-board computation requirements. This segments the image into three classes: i) sand, ii) soil, iii) flagstone. Both texture and depth information are then sent to the JPL Visual Odometry (VO) method detailed in (Howard 2008) to compute the relative motion between images. This information is the incorporated into the 3D map and assessed for both geometric and slip hazards in the traversability-assessment module. Geometric Hazards are assessed and mapped using the Morphin algorithm (Goldberg, Maimone, & Matthies 2002), a predecessor to the GESTALT method running on the Mars rovers To plan safe paths around geometry- and terrain-based hazards, we employ the RRT# sample-based planner (Arslan & Tsiotras 2016) to make informed decisions on adding new samples using the computed geometry, terrain, and rover motion information.

Slip-Aware Planning Our navigation system plans paths on a map that builds upon the data structure detailed in (Goldberg, Maimone, & Matthies 2002)—an occupancy-grid map fitted to a local ground plane with point-cloud statistics. The slip-aware navigation system improves on this map structure by adding terrain information information for each point in the stereo point cloud. Point clouds are accumulated to compute geometry and terrain statistics at each cell in the map. To assess the traversability of the map at each cell, a plane the size of the rover is centered and fitted to the containing points. Each cell in the map contains the following information: i) maximum step-size, ii) roughness, iii) slope, and iv) terrain information. Terrain information comes in the form of a discrete probability distribution for the three terrain types of interest: soil, sand, and flagstone.

The slip-aware navigation system plans safe paths that avoids geometric- and terrain-based hazards by employing the sample-based planner, RRT# (Arslan & Tsiotras 2016) and the traversability map to make informed decisions on expected wheel slippage. The sample-based planner constructs a random graph where vertices contain robot poses and edges link poses by vehicle-constrained motion primitives (Pivtoraiko, Nesnas, & Kelly 2009). During planning, new vertices are considered as viable if they do not intersect with any geometric obstacles in the map (step-size or roughness). The cost of edges in the graph is a function of the motion primitive distance weighted by an expected slip profile for each terrain type. Terrain slip profiles map slope to expected rover slip for a given terrain type. This planner furthermore takes into account direction of travel when adding a new sample.

Model-Based Health Assessment

The autonomous science scenario discussed in previous sections is only practical under two strong assumptions: First, that the rover protects itself from any problems during auto-generated activities; and second, that the rover can reliably detect and recover from problems that are routine but

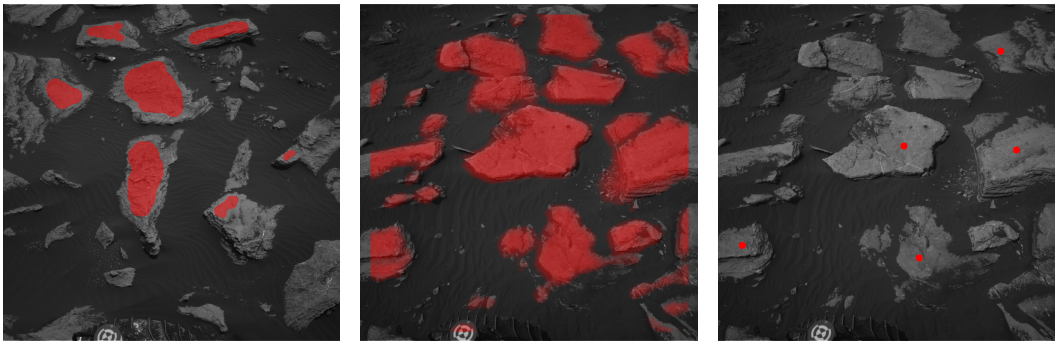


Figure 4: An example showing how scientists can use TextureCam to express intent to autonomously generate new goals on board. The left image shows hand-labeled regions of a geological formation of interest. The center image shows the estimated probabilities that regions in a new image are of the same formation, given a model trained from labels. The right image shows the top five software-selected locations for diverse observations of the rock formation, each corresponding to a new goal for the planning system.

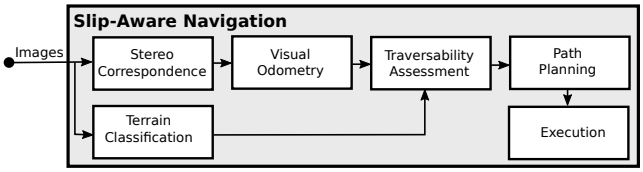


Figure 5: Illustration of the slip-aware navigation pipeline. This navigation system uses both geometry and texture from stereo images to map and assess hazards to the rover and plan safe paths in challenging environments with high slip risks. This will allow rover operators to plan longer autonomous traverses in difficult terrain.

currently require ground-in-the-loop resolution. Both challenges exceed the current state of practice for rover health assessment. We must assume an autonomous rover will have only limited knowledge of the terrain due to its more aggressive exploration, and thus the autonomous rover will be unable to avoid some hazards that a traditional rover would bypass. Existing rover fault protection can be elaborate – Mars Science Laboratory (MSL), for instance, has over 1,000 distinct fault monitors – but is defensive in nature, aimed at maintaining rover safety and preserving capability rather than ensuring efficiency of operations.

In current operations it is not unusual for even a routine activity (such as a drive) to "fault out," i.e., halt prematurely or exit with errors that requires ground analysis before resuming operations. These are typically benign, caused by factors such as slower progress than expected or incorrect assumptions about activity timing. Nonetheless, the Self-Reliant Rover must be able to discriminate recoverable problems from more serious ones, it must solve the simpler problems without ground input, and it must do both reliably.

It is clear that we are unlikely to achieve this by simply expanding the scope of fault protection. Instead, to compensate for uncertainty in the environment, the plan, and in the rover performance itself, we turn to model-based methods and reasoning systems. The Self-Reliant Rover prototype incorporates Model-based Off-Nominal State Identification

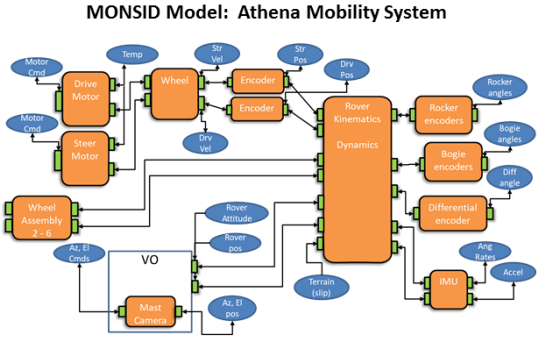


Figure 6: MONSID model of Athena mobility components

and Detection (MONSID) (Kolcio & Fesq 2016), which analyzes command and sensor data in real-time to construct an estimate of system health.

MONSID utilizes a simplified physics model comprised of a network of numerical constraints, describing the physical laws and relationships between sensed and internally-computed parameters. The model also relates these constraints to physical or logical components of the host system, allowing inconsistencies found to be linked to their root cause. MONSID was applied to an example rover electrical power subsystem in a previous experiment (Kolcio, Fesq, & Mackey 2017), illustrating the suitability and unique advantages of the approach while exposing model details and algorithm behavior.

For integrated testing with Athena, MONSID concentrates on the mobility systems and associated sensors. A summary of the MONSID model is shown below in 6.

In the diagram above, orange boxes represent rover components or pseudocomponents that aggregate different state variables. Blue ovals indicate command or sensor values, used to enable or verify computation of system state variables. Connections between components as indicated by ports (green boxes) represent constraints, evaluated sepa-

rately in each direction.

Each of the six wheels incorporates separate steering and drive motors, while the wheel assembly interacts with the controller via a pair of position encoders. Rover position and orientation is provided by visual odometry, analyzing images captured by Athena's mast-mounted cameras, and a notional Inertial Reference Unit (IMU). Additionally, the model supports variation in rocker and bogie position, as reported by four angle sensors. Note that in the current implementation both IMU and suspension sensors are not present.

During execution MONSID must detect faults and distinguish which are autonomously recoverable in a manner that other autonomy components can interpret. To support our evaluation, the Athena team has developed a fault injection capability enabling us to simulate drive and steering motor failure, failure of on-board controllers, and failure of mobility sensors. The most relevant cases to the autonomous rover science scenario can be summarized as follows:

Detect and classify recoverable mobility faults: If a drive is interrupted by an unexpected event, determine whether this is terrain-induced or caused by mechanical failure, and whether the rover should autonomously retreat and avoid the problematic terrain.

Recognize errors in terrain knowledge: If drives complete but leave the rover far from its expected position, determine whether the problem is caused by mechanical failure, sensor failure, or incorrect assessment of terrain. In the latter case, attempt to recover terrain knowledge by comparing to alternate models of terrain behavior.

Identify emergent, unknown, or surprise behavior: A significant hurdle to adoption of autonomy technologies in general is the persistent risk of unexpected behavior in the system leading to an unpredictable response. However, due to its reliance upon physical principles instead of purpose- built monitors, model-based health assessment is often capable of detecting and correctly classifying even novel system behavior.

An example of the last class of behavior was observed by the Athena team in early 2018, when driving up a steep slope led unexpectedly to one of the front wheels rising off the surface. We quickly replicated this behavior in our testing, finding it was caused by unexpectedly high traction in Athena's center wheel coupled with slippage of the rear wheel. This resulted in the center wheel driving forward relative to the rover as a whole, rotating the bogie in the process. A brief summary of this behavior is shown in 7.

This behavior is interesting because, while undesirable, all individual rover components are operating in familiar and acceptable ways. The root cause is instead a violation of a more fundamental assumption about the rover, namely the rover wheel geometry is changing while on flat terrain. These assumptions are incorporated into the MONSID constraints, and as a result, the novelty of this situation is detected without difficulty, despite the fact that this behavior had gone unnoticed after years of testing and experience with Athena.

Unlike the other types of faults, it is likely that we would halt operations after observing this for the first time in flight

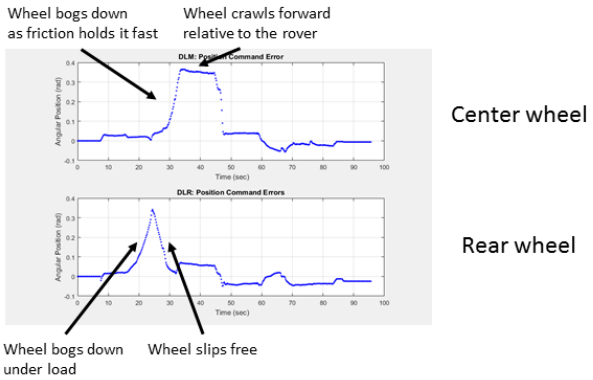


Figure 7: Athena drive position error during unusual "wheelie" behavior

to permit thorough analysis of newly revealed design vulnerabilities. MONSID's responsibility in this case ends with detection and classification as a non-recoverable event, however MONSID also provides diagnostic information to assist in event analysis. In this case the fault is correctly isolated to the center and rear wheels instead of any control fault, or any fault in the wheel that actually rises from the surface.

Global Localization

One of the key goals in improving autonomy for mobility is extending the distance the rover can drive per sol. Localization errors accumulate as a function of driving distance, however, due to drift in visual odometry and the integration of inertial measurements. The magnitude of this error depends on the terrain, but can be on the order of 5% of the drive distance. For MSL operations, the typical drive distance is on the order of 30m, resulting in fairly small drift in position estimate that could be several meters.

For MSL, this drift is corrected manually by visual alignment of navcam imagery to orbital HiRISE imagery. To estimate the alignment, a mosaic of navcam stereo images are taken to cover a full panoramic around the rover. These images are then orthographically projected and salient surface features are manually tie-pointed to compute a correction offset.

The self-reliant rover design utilizes both longer drives as well as multi-sol operations without the involvement of ground operators to perform these corrections. To achieve this, a similar alignment method is used in an automated manner onboard the rover. Instead of keypoint tie-pointing, the images are aligned using a matching criteria on both the image intensities and the elevation map. Both the surface and orbital images are orthographic projections, created by projecting the image onto the elevation map using HiRISE DEMs (digital elevation models) for the orbital images, and stereo disparity from the surface navcam images. The orbital image products are georeferenced and stored on the rover.

The matching criteria for the imagery uses mutual-information, or relative entropy, between the images (Ansar & Matthies 2009). This measures the statistical dependence

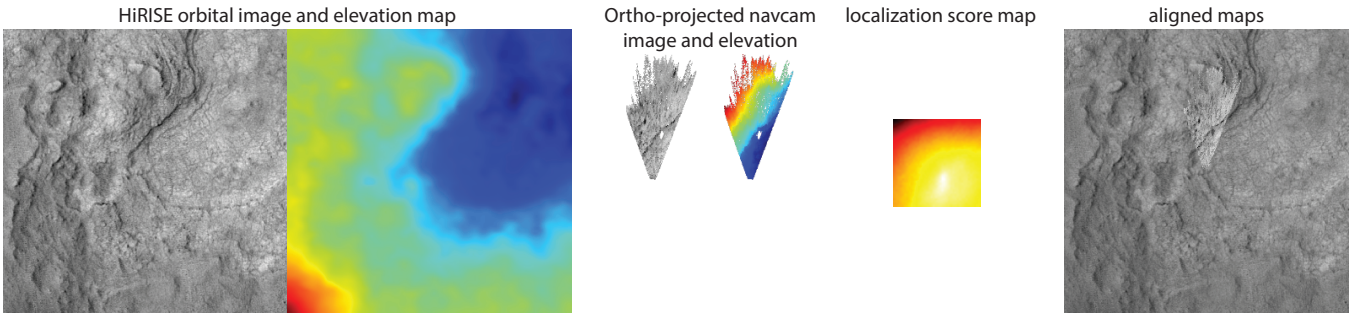


Figure 8: Global localization utilizes automated alignment of navcam image and elevation maps to onboard orbital maps.

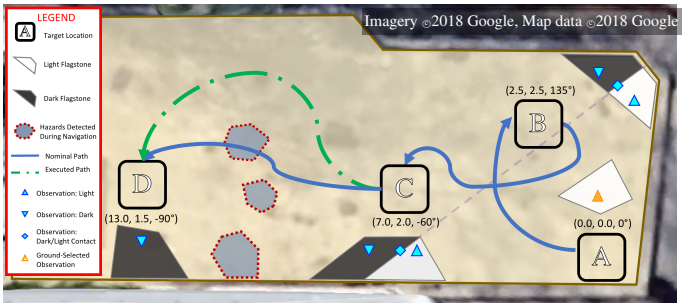


Figure 9: Overview of simulated mission area. Operator inputs include a specific target selection (orange) near starting area A along with only high-level campaign guidance for areas B, C, and D. Automated science analysis injects additional targets (cyan) during execution. The initial planned route (blue) is dynamically adjusted (green) to avoid unanticipated terrain hazards (red).

between corresponding pixels of a candidate alignment. Mutual information is used instead of more conventional correlators such as SAD or SSD for robustness to varying conditions from when the orbital image was acquired such as lighting or surface changes. The elevation map alignment uses a conventional SSD correlator. The overall matching score is simply a weighted sum between the image and elevation scares. The maximum drift of the position estimate is largely bounded, and the alignment search can be performed using a conventional sliding window approach.

Illustrative Scenario

The Self-Reliant Rovers system was demonstrated on the JPL Athena test rover within a mission scenario that explores the JPL mini-Mars Yard robotic testing facility. The primary science objective was to characterize the rock outcrop materials embedded in the sandy soil using the rover's mast-mounted cameras. The mission spans a period of limited communication with operators, so the rover must operate almost entirely autonomously in order to remain productive toward its high-level goals.

Figure 9 shows the overhead layout of the mission area, as might be available to mission planners from orbital imagery. The operations team selects several regions of interest (indicated by letters) from this coarse data, but is unable to identify specific targets or terrain obstacles beyond a few meters from the rover, for which the team has local imagery obtained from the rover. Previous local imagery allows the operators to set one precise outcrop target nearby the starting location at A. In prior operations, the team would have to be satisfied with filling the rest of the communication-limited period with various in-place tasks and perhaps one drive attempt toward the next area. Instead, using the Self-Reliant Rover system, the team can entrust the rover with enough campaign intent to continue conducting detailed science on its own.

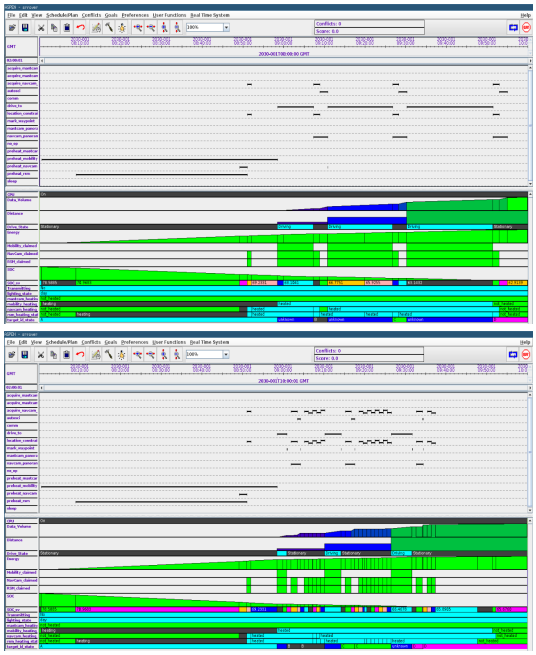


Figure 10: Initial generated plan and final as-executed plan for the simulated mission scenario. Many new targeted science goals are suggested at run-time by automated image analysis and then integrated into the schedule in service of science campaigns. Drive estimates are also updated during execution, thus correcting initial approximations.

First, the operators create a goal for each area of interest

that entails driving to a specified vantage point in that area, acquiring a contextual wide-angle image, and then running the appropriate automated science algorithms. These survey goals become part of their own goal set campaign, and the planner will stitch together an optimal drive ordering to achieve as many as possible. In addition, the scientists create initially empty goal set campaigns for each of the desired outcrop observations (light flagstone, dark flagstone, and multiple-contact) at each area. The campaign intents provide guidance for the rover's autonomous science behavior by indicating the algorithms to perform and the types of follow-up observations to suggested based on the results. During subsequent automated analysis, the previously trained onboard science classifiers will inject their newly identified follow-up targets as goals into these campaign containers for consideration during replanning.

Scalable campaign satisfaction criteria are described as a utility scored range over the number of observations desired. The planner and automated science cooperate to identify the best candidate targets to include in the plan so as to maximize expected utility score. When a campaign cannot be minimally satisfied with available targets, it may be skipped over in order to include lower priority campaigns. Likewise, only the best observation targets up to the desired maximum for a campaign will be scheduled. In this demonstration scenario, campaigns request follow-up mast camera imaging of the 2-5 best outcrop specimens in each category at each location.

Several additional relevant campaign types were demonstrated in separate scenarios. The operators can specify ongoing temporal periodic campaigns; for example, visual atmospheric opacity (τ) measurements every 20 ± 2 minutes. Mandatory downlink relay communication passes can also be enforced at specific times in the schedule, representing a exogenous orbiter overflights.

All of the various goals are provided to the rover at its morning communication pass at the start of the mission scenario. Thereupon, the onboard planner generates a plan to image the specifically requested target near A, and then travel in turn to B, C, and D to conduct survey observations (Figure 10, top, and Figure 9, blue path). The plan adheres to all standing rover resource limits (such as battery energy and data volume), as well as incorporating any required heating (such as needed for instruments or mobility mechanisms).

The actual path driven by the rover undergoes refinement by the onboard terrain classification and autonomous navigation so as to best avoid geometric obstacles. Due to a lack of terrain diversity and slopes in the testing environment, the slip avoidance aspect of the planner was disabled.

Depending on terrain, drives may also perform better than expected by the initial approximation. Diversion delays and expeditious travel cause minor perturbations to the plan, which are accommodated by an agile plan repair strategy that shifts actions within some threshold as long as they still meet their requirements.

On arriving at B, and later C, the rover acquires the requested contextual images and analyzes them using the onboard science detectors. In turn, the analysis software identifies both light and dark flagstone outcrops, as well as contact

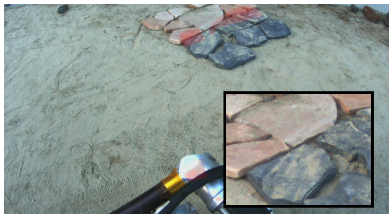


Figure 11: Automated detection of geologic formation contact in a survey image (top, contacts highlighted in red) triggers follow-up detailed imagery of the contact area (inset).

between the two (Figure 11, top, with contact areas highlighted in red.) These specific follow-up targets are then automatically injected as new goals in their respective campaigns, and a replanning cycle is initiated. The planner's updated solution includes each of the newly suggested observations, which are duly collected (Figure 11, inset) before proceeding to the next area.

Upon driving toward D, the rover's automated terrain classification identifies a major obstacle, and the navigation system must divert significantly. The planner assimilates updated drive estimates from the navigation engine to ensure that the plan can accommodate the delay without conflict. After planning a safe path around the observed obstacles and eventually reaching D, the system once again identifies flagstone features and conducts the requested follow-up observation. At this point the mission period ends.

As seen in the final plan (Figure 10, bottom), the productivity benefits of additional onboard rover autonomy are evident even within the limited scope of this demonstration scenario. Traditional operations would have accomplished just one initial outcrop observation and a first drive.The combined autonomy of the Self-Reliant Rover system produced three survey panorama images throughout the mission area, toured several unexpectedly difficult terrain routes, and accrued fifteen additional targeted outcrop observations. The Self-Reliant Rover system also allows the rover to incorporate periodic objectives into its generated activity plans. Overall, the scenario demonstrates the ability of the Self-Reliant Rover approach to increase mission productivity.

Related Work

Shalin, Wales, & Bass, (2005) conducted a study of Mars Exploration Rovers operations to design a framework for expressing the intent for observations requested by the science teams. Their focus was the use of intent to coordinate planning among human operators and the resulting intent was not captured in a manner that would be conducive for machine interpretation. Our approach codifies some of the fields in their framework in a way suitable for the rover. In particular, the authors defined a "Related Observations" field as a way for scientists to identify relationships among different observations, which need not be in the same plan. Our work on campaign intent can be seen as a way of defining a specific semantics to these types of relationships to facilitate reasoning about these relationships by the rover.

Their framework also includes information that we agree is essential for effective communication among operators

but that we do not currently express to the rover. For example, the "Scientific Hypotheses" field is used to indicate what high-level campaign objective is being accomplished by the requested observation. We are not yet providing these higher-level campaign objectives to the rover, though it is an interesting area of future research.

Mali (2016) views intent as a means for a user to place constraints on the types of plans a planner is allowed to produce such as only generating plans that have at most one instance of a class of actions or that plans must limit the use of a particular action. The primary role of our use of intent is to allow the planner to assess the value of achieving a given set of goals. However, some of our campaign intent does imply constraints and preferences on how, or more specifically, when goals are accomplished. For example, the periodic campaign intent specifies a timing relationship among goals and a preference on how close to comply with the desired timing.

There are some similarities between our campaign definitions and those used for Rosetta science planning (Chien *et al.* 2015). Both use campaigns to express requests for variable-sized groups of observations with relationships and priorities. Rosetta plans covered much longer time periods (e.g. weeks) and required more complex temporal patterns, such as repeating groups of observations. But observation patterns were primarily driven by the predictable trajectory of the spacecraft, allowing relationships to be expressed as temporal constraints. This is not sufficient for rovers, where many observations are dictated by the rover location and surrounding terrain, and the duration of many activities cannot be accurately predicted. State-based and goal set relationships more accurately represent some of the science intent found on surface missions.

There have been a variety of autonomous science systems deployed or proposed for rovers including the AEGIS system running on the Opportunity and Curiosity rovers (Francis *et al.* 2017), and the SARA component proposed for an ExoMars rover (Woods *et al.* 2009). These systems allow the rover to identify targets in its surroundings that match scientist-provided criteria. The introduction of campaign relationships broadens the scope of the type of guidance that scientists can provide these systems, allowing scientists to express the amount of observations they would like for their different objectives along with the relative priorities of the high-level objectives.

There have been several integrated rover systems with similar objectives to our work including PRoViScout (Paar *et al.* 2012), Zoe (Wettergreen *et al.* 2014) and OASIS (Castano *et al.* 2007). The PRoViScout project has similar objectives to our work (Paar *et al.* 2012). These systems It include autonomous science capabilities to enable onboard identification of science targets. Similar to our approach, they select follow-up observations for identified targets and submits these requests to an onboard planner to determine if there are sufficient resources to accomplish these new objectives. The campaign intent concepts we have developed would also be applicable to PRoViScout as a way to increase the expressivity for providing scientist intent to the rover.

There is an active area of research in intent recognition (Sukthankar *et al.* 2014). The general goal of this area is to identify the objectives of other agents (human or otherwise) from observations of the agents' actions. In contrast, in our work, it is acceptable for users to explicitly identify their intent, rather than require the system to attempt to infer intent. Indeed, there is interest in the operations team to clearly document their intent for the purpose of communication among teams and as a record of what activity was planned for the rover and why. As such, rather than try to infer user intent, our objective is to increase the expressivity of the rover's interface in order to more closely reflect mission intent.

The Mars 2020 mission is planning to incorporate onboard scheduling to improve resource utilization of the rover (Rabideau & Benowitz 2017). Similar to the Self-Reliant Rover approach, the use of onboard scheduling is intended to allow the Mars 2020 rover to use current vehicle knowledge when generating schedules to accomplish mission objectives. This will reduce the loss of productivity that results from the difficulty in predicting how much resources (e.g. time and energy) activities will consume. The Self-Reliant Rover approach is addressing additional productivity challenges by improving the ability of rovers to identify their own objectives, to incorporate a richer set of guidance from operators and to reason about slip hazards as it navigates.

The navigation system presented in this paper is most similar to the system presented in (Helmick, Angelova, & Matthies 2009). They propose a system with the same high-level machinery: i) a GESTALT-based vision pipeline, ii) a terrain classifier, and iii) a slip-aware planner. However, their system is not capable of making decisions based on direction of travel. When direction of travel is not considered, then the system is forced to make more conservative plans. An example is if the rover is planning a path on a steep slope containing soil, it might be too dangerous to drive up the slope due to expected slippage, but driving downhill would be safe.

Conclusions

We have presented an approach for increasing the authority of autonomous rovers to increase mission productivity. Our approach includes the ability for ground operators to provide guidance to the system without requiring up to date knowledge of the rover's state and its surroundings.

We have implemented a prototype of this approach on the Athena test rover. Over the next year we will be conducting mission-relevant, multi-sol scenarios with the rover at the JPL Mars Yard to evaluate its ability to support productive operations with limited ground-in-the-loop interactions.

Acknowledgments

This research was conducted at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. This work was funded by the Jet Propulsion Laboratory Research and Technology Development program.

References

[Ansar & Matthies 2009] Ansar, A., and Matthies, L. 2009. Multi-modal image registration for localization in titan's atmosphere. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, 3349–3354. IEEE.

[Arslan & Tsiotras 2016] Arslan, O., and Tsiotras, P. 2016. Incremental sampling-based motion planners using policy iteration methods. *CoRR* abs/1609.05960.

[Bradski 2000] Bradski, G. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

[Castano *et al.* 2007] Castano, R.; Estlin, T.; Anderson, R. C.; Gaines, D. M.; Castano, A.; Bornstein, B.; Chouinard, C.; and Judd, M. 2007. OASIS: Onboard Autonomous Science Investigation System for opportunistic rover science. *Journal of Field Robotics* 24(5):379–397.

[Chattopadhyay *et al.* 2014] Chattopadhyay, D.; Mishkin, A.; Allbaugh, A.; Cox, Z. N.; Lee, S. W.; Tan-Wang, G.; and Pyrzak, G. 2014. The Mars Science Laboratory supratactical process. In *Proceedings of the SpaceOps 2014 Conference*.

[Chien *et al.* 2015] Chien, S.; Rabideau, G.; Tran, D.; Doubleday, J.; Nespoli, F.; Ayucar, M.; Sitje, M.; Vallat, C.; Geiger, B.; Altobelli, N.; Fernandez, M.; Vallejo, F.; Andres, R.; and Kueppers, M. 2015. Activity-based scheduling of science campaigns for the rosetta orbiter. In *Proceedings of IJCAI 2015*.

[Francis *et al.* 2017] Francis, R.; Estlin, T.; Doran, G.; Johnstone, S.; Gaines, D.; Verma, V.; Burl, M.; Frydenvang, J.; Montano, S.; Wiens, R.; Schaffer, S.; Gasnault, O.; DeFlores, L.; Blaney, D.; and Bornstein, B. 2017. AEGIS autonomous targeting for ChemCam on Mars Science Laboratory: Deployment and results of initial science team use. *Science Robotics* 2(7).

[Gaines *et al.* 2016] Gaines, D.; Doran, G.; Justice, H.; Rabideau, G.; Schaffer, S.; Verma, V.; Wagstaff, K.; Vasavada, A.; Huffman, W.; Anderson, R.; Mackey, R.; and Estlin, T. 2016. Productivity challenges for Mars rover operations: A case study of Mars Science Laboratory operations. Technical Report D-97908, Jet Propulsion Laboratory.

[Goldberg, Maimone, & Matthies 2002] Goldberg, S. B.; Maimone, M. W.; and Matthies, L. 2002. Stereo vision and rover navigation software for planetary exploration. In *Proc., IEEE Aerospace Conference*, volume 5, 5–2025–5–2036 vol.5.

[Helmick, Angelova, & Matthies 2009] Helmick, D.; Angelova, A.; and Matthies, L. 2009. Terrain adaptive navigation for planetary rovers. *Journal of Field Robotics* 26(4):391–410.

[Howard 2008] Howard, A. 2008. Real-time stereo visual odometry for autonomous ground vehicles. In *Proc. of the Int. Conf. on Intelligent Robots and Systems (IROS)*, 3946–3952.

[Kolcio & Fesq 2016] Kolcio, K., and Fesq, L. 2016. Model-based off-nominal state identification and detection for autonomous fault management. In *Proceedings of the 2016 IEEE Aerospace Conference*. IEEE.

[Kolcio, Fesq, & Mackey 2017] Kolcio, K.; Fesq, L.; and Mackey, R. 2017. Model-based approach to rover health assessment for increased productivity. In *Proceedings of the 2016 I7EE Aerospace Conference*. IEEE.

[Mali 2016] Mali, A. D. 2016. Expressing user intent in planning by instance rewriting. In *Proceedings of the 2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI 2016)*.

[Paar *et al.* 2012] Paar, G.; Woods, M.; Gimkiewicz, C.; Labrosse, F.; Medina, A.; Tyler, L.; Barnes, D. P.; Fritz, G.; and Kapellos, K. 2012. PRoViScout: a planetary scouting rover demonstrator. In *Proceedings of SPIE Vol. 8301 ntelligent Robots and Computer Vision XXIX: Algorithms and Techniques*.

[Pivtoraiko, Nesnas, & Kelly 2009] Pivtoraiko, M.; Nesnas, I. A. D.; and Kelly, A. 2009. Autonomous robot navigation using advanced motion primitives. In *2009 IEEE Aerospace conference*, 1–7.

[Rabideau & Benowitz 2017] Rabideau, G., and Benowitz, E. 2017. Prototyping an onboard scheduler for the Mars 2020 rover. In *Proceedings of the International Workshop on Planning and Scheduling for Space*.

[Rothrock *et al.* 2016] Rothrock, B.; Kennedy, R.; Cunningham, C.; Papon, J.; Heverly, M.; and Ono, M. 2016. Spoc: Deep learning-based terrain classification for mars rover missions. In *Proc. of the AIAA Space Forum and Exposition*.

[Shalin, Wales, & Bass 2005] Shalin, V. L.; Wales, R. C.; and Bass, D. S. 2005. Communicating intent for planning and scheduling tasks. In *Proceedings of HCI International*.

[Stentz & Mellon 1993] Stentz, A., and Mellon, I. C. 1993. Optimal and efficient path planning for unknown and dynamic environments. *Int. Journal of Robotics and Automation* 10:89–100.

[Sukthankar *et al.* 2014] Sukthankar, G.; Goldman, R. P.; Geib, C.; Pynadath, D. V.; and Bui, H. H. 2014. An introduction to plan, activity, and intent recognition. In Sukthankar, G.; Goldman, R.; Geib, C.; Pynadath, D.; and Bui, H. H., eds., *Plan, Activity, and Intent Recognition,*. Elsevier.

[Thompson *et al.* 2012] Thompson, D. R.; Abbey, W.; Allwood, A.; Bekker, D.; Bornstein, B.; Cabrol, N. A.; Castano, R.; Estlin, T.; Fuchs, T.; and Wagstaff, K. L. 2012. Smart cameras for remote science survey. In *Proceedings of the International Symposium on Artificial Intelligence, Robotics, and Automation in Space*.

[Weiss 2013] Weiss, K. 2013. An introduction to the jpl flight software product line. In *Proceedings of the 2013 Workshop on Spacecraft Flight Software (FSW-13)*.

[Wettergreen *et al.* 2014] Wettergreen, D.; Foil, G.; Furlong, M.; and Thompson, D. 2014. Science autonomy for rover subsurface exploration of the Atacama desert. *AI Magazine* 35(4).

[Woods *et al.* 2009] Woods, M.; Shaw, A.; Barnes, D.; Price, D.; Long, D.; and Pullan, D. 2009. Autonomous science for an ExoMars roverlike mission. *Journal of Field Robotics* 26(4):358–390.

In Proceedings of the 14th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS), Madrid, Spain, June 2018.

Dynamic Shared Computing Resources for Multi-Robot Mars Exploration

Joshua Vander Hook, Tiago Vaquero, Martina Troesch,
Jean-Pierre de la Croix, Joshua Schoolcraft, Saptarshi Bandyopadhyay, Steve Chien
Jet Propulsion Laboratory, California Institute of Technology

Abstract

The NASA roadmap for 2020 and beyond includes several key technologies which will have a game-changing impact on planetary exploration. The first of these is High Performance Spaceflight Computing (HPSC), which will provide orders of magnitude increases in processing power for next-generation rovers and orbiters (Doyle et al. 2013). The second is Delay Tolerant Networking, which overlays the Deep Space Network, providing internet-like abstractions and store-forward to route data through intermittent delays in connectivity. The third is a trend toward small, co-dependent robots included in flagship missions (MarCO, PUFFER, and Mars Heli). Taken together, these imply an increasing amount of communication and computing heterogeneity on Mars in coming decades.

Motivated by these technological trends, we study the concept of Mars on-site shared analysis, information, and communication (MOSAIC) for Mars exploration. The key algorithmic problem associated with MOSAIC networks is simultaneous scheduling of computation, communication, and caching of data, which we illustrate using the three scenarios. We present models, preliminary solutions, and simulation results for two scenarios, showing how mission efficiency relates to communication bandwidth, processing power, geography of the environment, and optimal scheduling of computation, communication, and data caching. The third scenario illustrates future directions of this work.

1 Introduction and Related Work

Three trends are poised to significantly change mission concepts for future NASA planetary exploration. While previous missions involved single robots with limited processing capability, the combination of new networking technology, advanced computation hardware, and small-bodied robot designs is making multi-robot missions more attractive.

In an effort to modernize the flight computing hardware available for NASA missions, the High Performance Spaceflight Computing (HPSC) initiative was announced in 2013 (Doyle et al. 2013; Powell et al. 2011; Mounce et al. 2016). Unlike the current generation of computing, this program aims to keep NASA computing technologies at most one generation behind commercial technologies. HPSC is expected to become a mainstay in post-2020 deployments.

The second key emerging technology is Delay or Disruption Tolerant Networks (DTNs). DTNs span communications links in an overlay architecture, enabling connectivity across network boundaries in a transparent manner, regardless of multiple potentially disparate network link layer protocols. A core principle of this overlay quality is the ability of individual nodes to store network data for possibly long durations before forwarding it to another node. This store-and-forward paradigm is central to DTNs. Many features of Delay Tolerant Networking architectures are of particular utility in the deep space interplanetary communications realm, where a multitude of link layers, bandwidth constraints, and disruptions are expected during end-to-end transfer of mission commands and data (Wyatt et al. 2017).

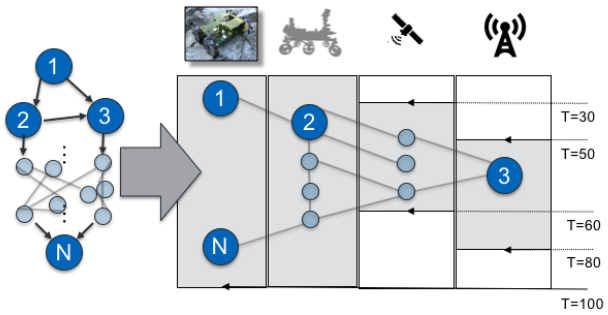


Figure 1: Illustrative MOSAIC scenario. A set of processing and data-driven tasks (left as dependency graph) must be mapped to multiple assets with heterogeneous computing, communication, and energy capacities. Each asset is also available over a fixed time window due to terrain effects or orbital parameters. The goal is to compute all the required tasks as quickly as possible.

Finally, an interest in multi-robot systems re-emerged. Currently planetary exploration is limited to benign operating areas due to the inability to land, traverse challenging terrain, or generally too great a risk for the primary mission asset. Unfortunately, the most compelling locations are often in these extreme terrains. Small, low cost, expendable rovers could transport key sensors and instruments to locations considered too risky for the primary lander, rover, or astronaut. Also due to the high communications latencies of deep space missions these expendable rovers must minimize their dependence on ground control and be able to operate primarily autonomously. These small craft can be released from par-

ent rovers and guided toward sampling targets which may be out of reach of the main craft, either because of risk, or simply to avoid delays from stopping. The "daughter-craft" do not have advanced processing capabilities due to weight, power, and cost constraints, but are attractive for a number of science targets, such as being left behind to investigate transient detections, risky exploration areas such as Recurring Slope Lineae, or wide-area sampling for In-Situ Resource Utilization. Two examples of such potential future systems now being considered for development are the Mars Helicopter, and the "PUFFER" rover (Pop-Up Flat-Folding Explorer Robots) (Karras et al. 2017).

Combining these three trends, we envision scenarios in which a system containing two or more robotic agents with large discrepancies in processing power, communication bandwidths, data capacities, and energy storage must collaborate to achieve a variety of realistic remote science missions.

The concept of study in all three scenarios is that advanced, software-driven robotic capabilities can be realized on small, resource-constrained, high-risk "edge" devices by optimizing data flows and processing assignments among all the devices. In this paper we formalize this problem and present preliminary results in modelling and analyzing Mars exploration missions. Because data and computation are shared among many devices, we dub a local computation-sharing network a MOSAIC (Multi-robot On-site Shared Analytics Information and Computing) network.

Our paper is organized as follows. First, we derive our problem statement in Section 2. We decompose objectives into a set of computing tasks, each of which generates data products which must be fed into subsequent tasks (possibly by transmitting between agents). Each task may be conducted by humans or robots. In Section 3 we discuss a search routine which can identify how the computational load can be distributed over the network.

We describe our study scenarios in Section 4. In the first illustration, (Section 4.1) we consider a single, cpu-bereft asset which can request computation from a nearby base station or visible orbiter. We study both a single PUFFER released from a base station (first scenario), and the Mars 2020 rover assisted by a hypothetical HPSC (second scenario). Planning for a potential Mars Sample Return campaign is dominated by the need for autonomous traversals of increasingly fast speeds, and we show an analysis of impact that computation sharing can have on mission success.

The third scenario (Section 4.3) is a mother / daughter craft design consisting of a large centralized asset (a human or flagship rover) controlling one or more agile, but less-capable "scouts" for an area search task. In this regime we discuss some emergent behavior like data relay and automatic choice of a centralized computing agent.

2 Problem Description

In this section, we describe how we frame the problem of dynamic shared computation for Mars exploration. We define a data communication and processing workflow that represents the mission objectives and intermediate goals at a high level.

Our primary abstraction is that of a Server Graph. Let there be $N \in \mathbb{Z}^+$ agents in the network, where $\mathbb{Z}^+$ denotes positive integers. The robot agents are denoted by $A_1, A_2, \ldots, A_N$. Each agent has on-board processing, memory, and communication links.

2.1 Computation

The agents perform $M \in \mathbb{Z}^+$ data-driven tasks. The set of M tasks is denoted $\mathbb{T}$. We consider heterogeneous processing times, so the time cost of executing task T on agent i is given by: $C_i^t(T)$. The model represents, e.g., the worst-case, expected, or bounded computation time, and so all the times are deterministic. In addition, program outputs are the same irrespective of the agent doing the computing (or are just as useful). Task T performed by robotic agent i may also include an energy cost, $C_i^e(T)$. If an agent has access to two or more *different* processing units, we model those as two co-located agents. If an agent has access to two or more *similar* processing units, we adjust the costs of each task to reflect its level of parallelization, but otherwise consider them the same processor.

Tasks produce data products. Data products for task T are denoted $d(T)$. If a task produces more than one data product, we model it as multiple tasks, one per produced data product. The size of the data products are known a-priori, and labelled as $s(T)$ for task T.

Let P_T be a set of predecessor tasks for T. Then $j \in P_T$ means task T depends on the output of task j. A task may have multiple prerequisite sets, one of which must be satisfied entirely. That is, a task must have only one of its prerequisite sets satisfied.

The static software network SN captures dependencies as the flow of information through various individual programs to solve the complex computing task.

Finally, we allow some of the tasks to be required and some to be optional. Optional tasks have a reward score ($r(T)$). The set of required tasks is denoted $\mathbb{R} \subseteq \mathbb{T}$.

Assumptions: The software network SN does not have any cycles. The mission statement for each problem/scenario can be stated as a software network SN.

A *solution* is a mapping of tasks to servers (agents) and start-times denoted

$$\mathbb{S} : i \rightarrow (A_j, t) \tag{1}$$

where (2)

$$j \in [1, \ldots, N] \tag{3}$$

$$t \geq 0 \tag{4}$$

Each agent's computing schedule in a solution is denoted

$$\mathbb{S}_i = j \rightarrow_i (t) \tag{5}$$

and has cost equal to the time required to complete the last task in the agent's queue,

$$C(\mathbb{S}) = \max_i C(\mathbb{S}_i) \tag{6}$$

where (7)

$$C(\mathbb{S}_i) = \max_j \mathbb{S}_i(j) + C_i^t(j) \tag{8}$$

To execute a specific task in the software network, an agent must have all the data products from one of the tasks predecessor sets, either by computing them directly, or by receiving them by communication from other agents. To communicate, we model each agent as having a Delay Tolerant Networking (DTN) stack to enable communication, as defined next.

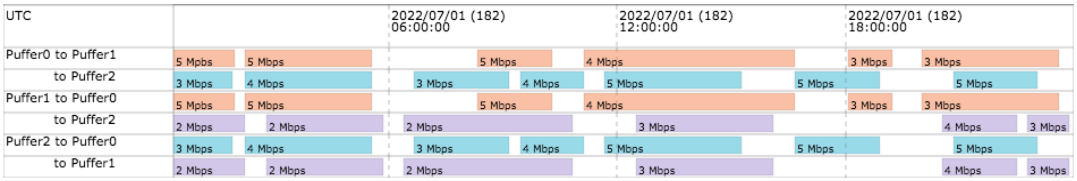


Figure 2: Contact graph for 3 agents showing times and bandwidths available

2.2 Communication

A key feature of DTN-based networking is Contact Graph Routing (CGR) (Wyatt et al. 2017). CGR takes into account predictable link schedules and bandwidth limits to automate data delivery and optimize the use of network resources. The contact graph describing a network's links over time is distributed to participating DTN nodes, allowing each node a clear picture of how to route data in an optimal manner. Each scenario is complicated by the relative geography of the agents which may affect communication rates, their motion plans through the environment, and the nature of long-distance communications such as light delays or degraded signal strengths. The practical effect of incorporating DTN's store-forward mechanism into the scheduling problem is that it is possible to use mobile agents as *robotic routers* to ferry data packets past communication interference.

The time-varying contact graph CG captures the communication network topology between agents. For each agent, the graph provides a list of all the time intervals during which it can establish a directed communication link with another agent. An example timeline representation of a contact graph for 3 agents showing available bandwidths can be seen in Figure 2.

Links have a time varying data rate from 0 (not connected) to ∞ (communicating to self), denoted by $r_{ij}(t)$ for the rate from A_i to A_j at time t. Thus, communication links are directed.

At any time k, let $\mathcal{G}_k$ be the graph representing the set of agents it can send to or receive from. Vertices $\mathcal{V} = \{1, \ldots, N\}$ and the directed edges $\mathcal{E}_k$ along which communication is possible. That is, if information can flow from the i^{th} agent to the j^{th} agent at the k^{th} time instant (where $i, j \in \mathcal{V}$), then the edge $\overrightarrow{ij} \in \mathcal{E}_k$.

Communication between agents is a task with cost determined by the size of the data product and the current data rate between agents. The task of communicating the data product $d(T)$ from A_i to A_j at time t requires time $C^t_{ij}(T) \propto s(T)/r_{ij}(t)$ for *both* agents and energy equal to $C^e_{ij}(T)$ on the *sending* agent.

Assumptions: Agents take 0 time to communicate the solution to themselves. Intervals with non-zero data rates are sufficiently long to transmit any data product (or they would be "effectively zero").

Problem 1 (Distributed Computation)**.** *Given a set of tasks modelled as a software network SN, a list of computational agents A_i $i \in [1 \ldots N]$, a contact graph CG, and a maximum schedule length $C^\star$, find a solution which is a mapping of tasks to servers (agents) and start times, $\mathbb{S} = f(i) :\rightarrow (A_j, t)$, such that:*

- *The maximum server cost, $C(\mathbb{S}) = \max_j C(\mathbb{S}_j)$ is no more than $C^\star$;*
- *All required tasks are scheduled;*
- *At least one of the prerequisites for all required tasks are scheduled.*

3 Scheduler Implementation

To study the role of optimal distributed computing in our mission concepts, we implemented a scheduler which uses a simple state space search to satisfy Problem 1.

We use a simple solution-space search. Conceptually, a priority queue of solutions is maintained, sorted by cost. The set of acceptable end states are those which contain all the required tasks. The starting state is an empty schedule. At each iteration, a new, partial solution is constructed by adding a new task to one of the servers. If the requisite data products were not previously calculated on that server, then the solution is first augmented with communication tasks to gather the missing pre-requisites. The cost of the communication task depends on the transmission rate between the agents, and the earliest time that the agents can communicate. The agent which ensures earliest arrival time is chosen for each prerequisite data product. Thus, at each iteration either the solution is augmented by one task, or by a set of transmissions to retrieve missing data followed by the task itself. During search, a list of feasible solutions is kept for each reward value. If the resulting solution contains all required tasks, it is stored, indexed by reward. The search continues until the priority queue contains only solutions exceeding the maximum cost. Then, the maximum reward solution is returned.

The performance of the implemented scheduler is suitable for trade studies and ground-side assignment of computing duties. However, since we can solve the optimal distribution of tasks a-priori given a communications regime, it is simple to provide an onboard scheduler as a lookup table of pre-verified assignments.

In what follows we describe three scenarios where scheduling shared computing resources is key and would have impact. We use agent's tasks taken from literature and from future exploration missions to Mars.

4 Scenario Descriptions

Given the problem description and scheduler implementation from previous sections, we now describe the mission scenarios we consider. The scenarios were chosen to be realistic enough for meaningful analysis, and to stress different aspects of the computation and communication scheduling.

The costs for transmissions vary by data product size and transmission speeds (e.g., the contact graph data rates). Thus, we vary the data rates and maximum time to explore the trade space of solutions. The resulting set of solutions could be pre-calculated for quick look-up in a real mission if the device was too resource-constrained to run a full scheduler. However, we leave the onboard scheduler implementation to future work and instead explore the "tipping points"

between schedules and data rate thresholds at which interesting transitions between scheduler regimes occur.

In what follows we describe three scenarios. For each scenario we determine which set of agent capabilities is relevant, and compose them into a software network. For two of them we provide initial results of simulations, which illuminate the benefits of a MOSAIC-like architecture. The last scenarios illustrate more complex missions in which the architecture would provide a promising impact in future work.

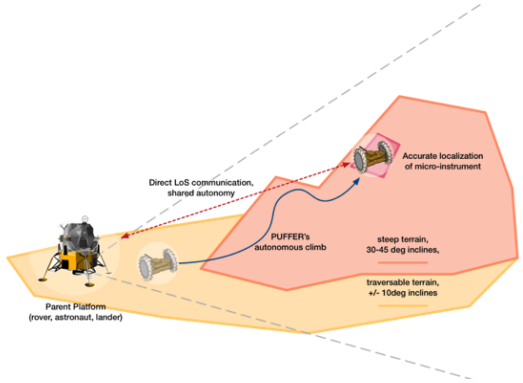


Figure 3: The "assisted drive" scenario.

4.1 Mars Drives

The first conceptual mission (Figure 3) is based on a single PUFFER combined with a parent platform (e.g., base station or flagship rover) to accurately place a PUFFER's instrument (microimager) on a terrain feature. This operation occurs within the parent platform's direct communication and sensing line-of-sight (LoS). PUFFER must be capable of autonomously navigating the environment homing in on the feature. It may leverage the better computation capabilities of the parent platform, as well as its sensors that offer a more advantageous perspective of the drive to improve its placement accuracy.

Each PUFFER is equipped with two STM32F4 microprocessors clocked at 180MHz and 168MHz with 256KB and 192KB of SRAM (Static Random Access Memory). Current versions of PUFFER utilize a Bluetooth radio with up to 2.1 Mbit/s data rates at approximately 1 W. Future versions of PUFFER may use a mesh radio, such as ZigBee, with data rates up to 250 kbits/s with approximately 100 mW power draw.

The parent platform, representing either a lander or rover, would include more significant computational resources, such as the HPSC. It would also have more power (e.g., MSL's radioisotope thermoelectric generator produces 2.5 kWh), and the communication equipment to communicate with an orbiter or directly to Earth (e.g., MSL has X-band for direct communication with Earth at 32kbit/s with 15 W, and UHF for communication to the orbiter at 2Mbit/s with 9 W) (Edwards et al. 2014).

We assume the parent platform can image the surrounding environment and locate the puffer to provide terrain-relative localization. We also assume the PUFFER can estimate it's own position using visual odometery (VO) and inertial measurements. We assume onboard state estimation using VO requires an image from the PUFFER's onboard camera system.

Since the PUFFER is equipped with a small scientific instrument, we assume that the puffer can acquire measurements from the instrument during its drive, but that this requires time, such as focusing, deploying, and pre-processing an image from a microscope. Thus, in process of navigating to its destination, a PUFFER has to sense the environment, plan its path and act (dispatch and execute low level tasks). During that process, a PUFFER might choose to perform science (microimager) and transmit the science data product to the lander for further use.

Figure 4a shows a data flow diagram to represent the software network associated with the aforementioned processes. The diagrams model the options available to the PUFFER to execute and share tasks in this scenario. Depending on the bandwidth and contact graph, the vehicle might choose to request the lander to take a long range image, localize the vehicle, perform path planning and then send the plan back to the PUFFER to execute the plan. That would potentially allow the PUFFER to use the spare time to take a microscope image and send it to the lander to archive it for further data fusion. The PUFFER might choose, as an alternative, to take an image from its camera system, use visual odometry to localize, and perform path planning all onboard; however, given that VO is less accurate than the terrain-relative localization from the lander, the PUFFER would have a higher uncertainty level about its position which would be carried to path planning.

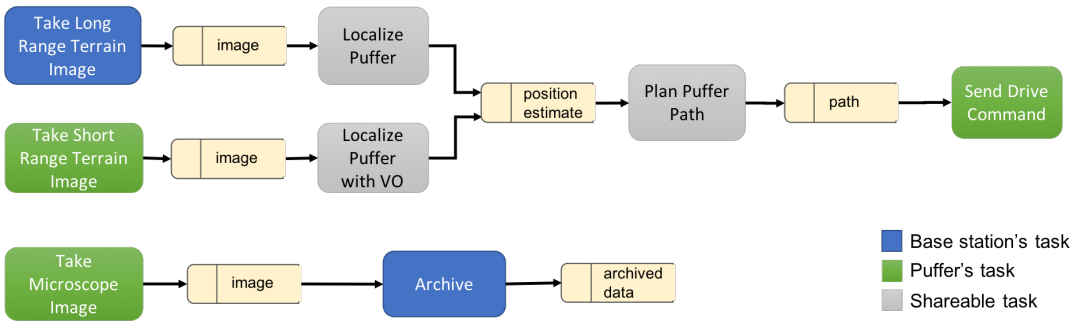
If both images (from lander and PUFFER) are taken and both localization processed are performed, the resulting position estimate is more accurate and so is the resulting trajectory from the path planning process. Figure 4a illustrates the tasks that can be shared and executed either onboard the lander or the PUFFER. Specifically, localization tasks and path planning are example of computational capabilities that can be scheduled and placed either onboard the lander or the PUFFER itself. Colored tasks mark the required vehicle for those capabilities.

We analyzed this software network for a variety of time limits and bandwidths between parent and PUFFER. The analysis is summarized in Table 4b. Figure 4c shows an example activity timeline for the puffer and base station from one of the resulting regimes. We find that high bandwidths are required to show preference towards off-board computing, at least for this scenario.

As shown in Table 4b, the long delays of taking microscopic images can be offset by requesting computational aid from the base state for planning paths. Alternatively, optimizations to program runtimes could have greater impact than bandwidth increases. Analyzing the *sensitivity* of these scheduler regimes with respect to runtimes, environment such as bandwidth distributions, and hardware choices is a key future direction to enable quick hardware and mission trade studies for distributed systems.

4.2 Mars 2020 Assisted Drive

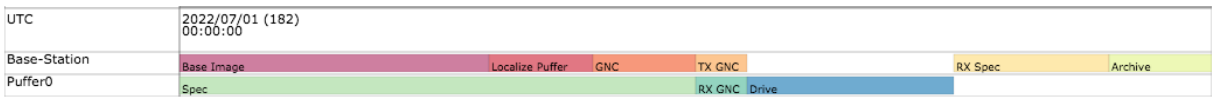
Note, the single-PUFFER scenario closely mimics the Mars 2020 mission with only minor changes. One defining feature of Mars Sample Return mission concepts is the likliihood of re-visiting the same area with subsequent launches to fetch, retrieve, and launch the samples (Mattingly and May 2011). If an on-site computing asset were available to multiple rovers in the area, they could make use of it for



(a) Data flow diagram representing software network for the single PUFFER scenario.

b/w (Mbps)	Base Cam	Puffer Cam	VO	Locate Puffer	GNC	Drive	Microscope	Store Img
≤ 0.05	Base	PUFFER	PUFFER	Base	PUFFER	PUFFER	N/A	N/A
$(0.05, .5]$	Base	PUFFER	PUFFER	Base	Base	PUFFER	N/A	N/A
$(0.5, 2.5)$	Base	N/A	N/A	Base	Base	PUFFER	PUFFER	Base
≥ 2.5	Base	PUFFER	Base	Base	Base	PUFFER	PUFFER	Base

(b) Distributed processing regimes for a single PUFFER and base station.



(c) Example activity timeline for the base station and puffer for a resulting regime.

Figure 4: Single-PUFFER scenario. The software network (4a) was analyzed as a function of bandwidth between the base station and rover to produce different processing regimes (4b. The rate of data transfer between the two uniquely determines what processes are possible, and where they are executed. Each data point is a timeline as shown in (4c. The roll-up shows aggregation of thousands of timelines produced by a solution-space search routine.

off-loading their required engineering tasks, in order to take advantage of opportunistic science processing and sensing. Thus, the assisting asset(s) could provide an "infrastructure upgrade" and could remain on-site, providing communication, computation, and data analysis services for all subsequent phases of the campaign. An interesting direction for future research would be to identify the requirements of such an asset. The asset could be embedded in a CubeSat network, and "piggy back" on the 2020 launch, similar to the MarCO CubeSats (Hodges et al. 2016). Alternatively, it could be embedded in the "skycrane" lander and dropped during the "flyaway" phase (Korzun et al. 2010; Sell et al. 2013). Finally, it could be a tethered balloon configuration (Kerzhanovich et al. 2004).

To explore any potential benefit, we next consider a strategic drive campaign by a Mars 2020 rover. In this case, we used information about the intended Mars 2020 drive pipeline from a talk given by Richard Rieber (Rieber 2017). The Mars 2020 conceptual path-planning pipeline, presented in (Rieber 2017) is simplified for our use in Figure 5a. The randomized time associated with Select Path is understandable given the mission analysis from (Ono et al. 2015). This data used in simulation is adapted from (Ono et al. 2016).

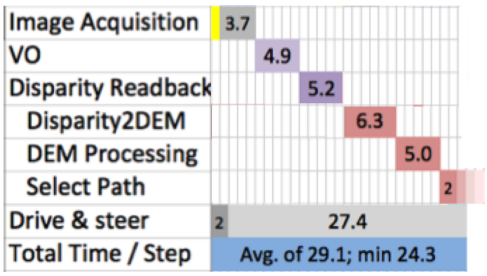
We created the model software network for Mars 2020 illustrated in Figure 5b. The required tasks are constructed to model the timings given in Figure 5a. From (Ono et al. 2016), we also included the ability for the rover to use imagers to classify the terrain, but only as an optional algorithm, since the current Mars 2020 pipeline does not include it.

To model the terrain in our simulations, we use terrain data classified from HiRISE imagery from (Ono et al. 2016). Multiple terrain types are grouped into different classes or as obstacles (terrain that cannot be traversed). We do not currently take slope into account, therefore we model the velocity of a rover in a given terrain class based on the average speed over multiple slopes for that classification.

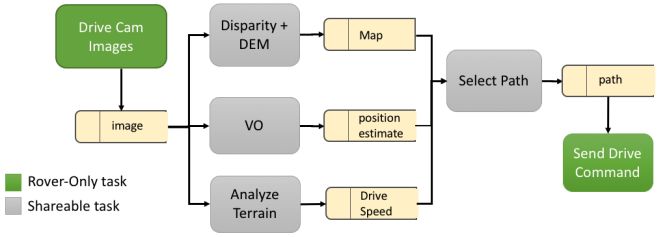
In order to model the different fidelity of data obtained in orbit and on the ground by the rover, we assume certain terrain types as unknown. When a rover is in an unknown terrain type, it will move at the velocity of the real terrain class; however, it will plan a path assuming a terrain with the fastest traverse velocity. Nevertheless, if a rover is able to perform terrain classification, we assume it will be able to correctly classify the terrain within a given radius.

Repeating the analysis of the software network produced the data shown in Table 6a. From this analysis, we isolated four operating regimes for the rover. In the first regime, the rover has no access to the assisting resource (regime 0). Regimes 1-4 represent increasing bandwidth, and therefore increasing savings from assisted computation. To reveal the *strategic* benefits of these computational regimes, we simulate the four rover regimes across a Mars-like strategic drive.

To test the different communication and computation regimes, simulations for 4 different regimes were run on 3 different terrain subsections 10 times each (resulting in 30 total runs) using stochastic durations for the path planning and terrain analysis activities. The assumed stochastic activity times are shown in Table 1. It is further assumed that the



(a) Simplified model of Mars 2020 path planning.



(b) A corresponding software network.

Figure 5: A model for the timing of Mars 2020 as discussed. The Select-path task is modelled as a random process taking a minimum of 2 seconds, but widely varying. The over-runs associated with any runtime longer than 30 seconds is the primary contributor to lost drive distance. The secondary contributor was a lack of terrain awareness, caused by insufficient processing power to run onboard terrain analysis.

duration to communicate the data to the balloon is approximately 3 seconds, and that the duration to communicate a response back to the rover is approximately 1 second. The distance between the start and end points for each traversal was approximately 93 meters.

Table 1: Duration of activities in seconds on-board the rover and on the balloon.

	On-Board	Balloon
Path Planning	$\mathcal{N}(8,4)$	$\mathcal{N}(0.5,0.0001)$
Terrain Analysis	$\mathcal{N}(4,4)$	$\mathcal{N}(0.5,0)$

The baseline regime is Regime 1, where the rover performs all path planning on-board and does not perform any terrain analysis. In Regime 2, the rover sends data to a balloon where the path planning algorithm is performed and the results sent back to the rover. Regime 3 is the same as Regime 2, except that with the extra time, the rover performs terrain analysis on-board, which can be used for the next planning cycle. In Regime 4, terrain analysis is also performed on the balloon and the results communicated back to the rover.

Figure 6b shows an example of the different paths that are taken for the different regimes when some of the terrain is unknown without terrain analysis. The yellow terrain requires terrain analysis to be identified and is also slower to traverse. From this example, it is shown that with the terrain identification knowledge, Regime 3 and Regime 4 are able to come up with more efficient paths.

Since it is assumed that the rover must operate on a fixed 30 second cycle, if the path planning and/or terrain analysis are not completed within the allotted 8 seconds, an overrun will occur, causing the rover to stop until computation is completed. The distribution of percentage of overruns are shown as box plots in Figure 6d. As expected, Regime 2 and Regime 4 result in no overruns, whereas Regime 1 and Regime 3 have overruns around 50% of the time.

Another metric for the improvement of the rover performance is in the time it takes to traverse a terrain. Figure 6c shows the time to traverse a terrain for each regime compared to the baseline (Regime 1). From these results, it is shown that being able to perform terrain analysis, and therefore being able to plan a path with better terrain knowledge, improves the time to travel between two points.

We note a measurable increase in strategic drive efficiency using this limited study technique. Future work can focus on a more realistic terrain model, including that of the intended landing site. In addition, we can more realistically model the communication network. Intermittent loss of connectivity and varying data rates are significant impediments to this approach over long dries. Finally, modelling multiple assets would involve not only competing for the computational resources, but forwarding terrain classifications and drive rates between rovers.

4.3 Cooperative Exploration

The next conceptual mission (Figure 7) is based on multiple PUFFERs cooperatively (i.e., their autonomous operations are coordinated by sharing information) expanding science and exploration footprints into areas not within direct line-of-sight of the parent platform. The team of PUFFERs will maintain a communication network while exploring an environment with limited direct line of site (e.g., rubble fields, caves, lava tubes).

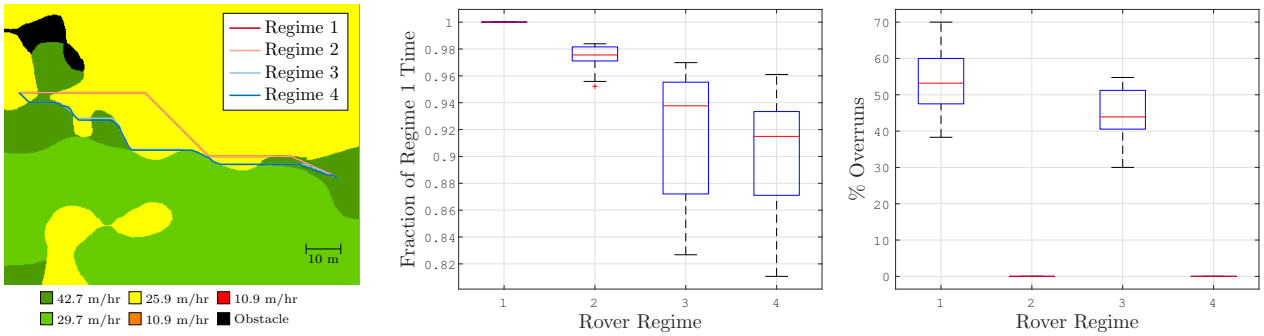
We assume the PUFFERS are exploring a distributed, but spatially-correlated phenomena, such as water moisture levels. We model the sampling and estimation on a similar terrestrial process used in farms(Tokekar et al. 2016). The point samples of moisture levels are gathered by spectroscopy or dipole measurements, and are incorporated into a spatial-estimation technique called Kriging (Brdossy and Lehmann 1998). Kriging is computationally expensive, and requires storage of all measurements. Therefore, it is not suitable for computationally-constrained devices like PUFFERS, but can be performed on the base station, orbiter, or on Earth.

In this scenario, each PUFFER operates under the same condition and software network as those used in the single vehicle scenario (Figure 4a), except that herein the lander becomes a shared resource for computation requests. Moreover, the team of PUFFERs provides a larger mesh-based communication network, allowing data to be sent across vehicles to reach the lander.

As before, a PUFFER can perform localization and path planning onboard, request the lander for support on those tasks, or even both while navigating the environment. In those cases, the computation sharing has to be coordinated among the vehicles since the lander has limited computational resource. In such coordination, PUFFERs can reason

b/w (Mbps)	Time	Image	Mapping	Extra Observations	Plan Path	Confirm / Drive	SPOC-lite
$(0-.1]$	27	Rover	Rover	N/A	Rover	Rover	N/A
$(0.1-.3]$	29.3	Rover	Rover	N/A	Assist	Rover	Rover
$(0.3-1]$	$(29.7-28.2]$	Rover	Rover	Rover	Assist	Rover	Rover
$(1,100]$	$(27.3-15.3]$	Rover	Assist	Assist	Assist	Rover	Rover

(a) A Mars 2020 rover adaptation of assisted drive. The adaptation was made using the pipeline information given in Figure 5a.



(b) Example paths (c) Time to traverse waypoints per regime (d) Overruns per traverse for each regime

Figure 6: Effect of computing regimes on a Mars 2020-like mission. 6b shows the path choices. The main effects of addition computation assistance is reduced planner overrun and better terrain classification, resulting in more efficient paths, as shown in 6c and 6d. Terrain types are designated as different colors and the darker terrain (darkest except for black) can only be identified using terrain analysis.

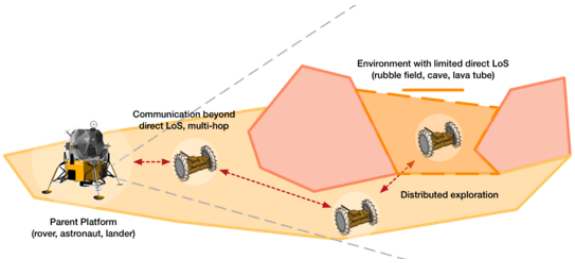


Figure 7: Multi-Robot Scenario 2

about routing their data products to the lander. For example, it might detect that a direct communication link to the lander is poor due to the current terrain features, but routing data through one of the other PUFFERS would work better. That would allow the scheduler to potentially add both microscope image and archive tasks to the regime, along with transferring the data through the vehicles network.

This scenario presents significant challenges to distributed computation because of the combination of roles a PUFFER may take. First, it may be purely sensory, taking images and then moving while sending those images to the base station. Second, it may be able to position itself as a relay node, spending all its time ferrying data between other assets. Alternatively, it could be a combination of the two, depending on its location, the plans of assets around it, and the motion around intervening terrain which may affect bandwidth. Finally, the motion planning problem in this context is critical. How are sample locations chosen for the PUFFERs? How does the motion and location of the PUFFERs affect data rate, and can paths be chosen to maximize information flow? These questions are good directions for future work.

5 Conclusion

In this paper we described the MOSAIC concept for Mars exploration in which simultaneous scheduling of computation, communication, and caching of data across different networked assets becomes increasingly essential. We presented a series of scenarios to illustrate how MOSAIC networks can impact science utility, vehicle performance and would enable an optimal distribution of computational loads, specially in multi-asset scenarios - a natural progression of future missions to Mars and other planets.

The cooperative exploration scenario in Section 4.3 represents our major next hurdle. A comprehensive solution would include role assignment (relays versus sensors), position and path assignments to maintain connectivity, and response to changing communication networks, including momentary breakage of links to gain greater sensing data. We will proceed first with role assignments for data routing.

The preliminary study of optimal processing distribution is useful as feedback into hardware design. The methods of this paper can be used to optimize the hardware of the PUFFER design, or design communication networks for future Mars exploration missions. In this direction, determining the "tipping points" between different processing regimes is most important. The differences in efficiency between regimes can be very large. A schedule sensitivity analysis is required to determine the optimal schedule's response to perturbations to e.g., bandwidth. We have conducted this analysis by using a "brute-force" search routine, but producing analytical and algorithmic results which are quick are more capable are a primary next step for research. We expect this analysis will fold nicely into a framework similar to (Herzig et al. 2017) which provides a hardware-space expansion for designing multi-asset missions.

The initial results and envisioned scenarios described in this paper brings interesting next steps and promising re-

search efforts in the MOSAIC project. We will study in more depth the multi-vehicle scenarios presented in this paper and identify the key algorithmic requirements for those cases. In these cases we will investigate on different scheduling techniques and formalisms that could be utilized onboard the assets to allocate computation load, considering vehicle with both low and high CPU capabilities, and manage connectivity fluctuations. Our framework is designed to be responsive to loss of connectivity by re-scheduling tasks based on a new communications graph using a set of pre-verified distribution of tasks. In particular, we have studied the change in optimal computing distribution due to bandwidth fluctuations, but more research is necessary to fully evaluate risk of connectivity variations and provide an onboard scheduler which can accommodate unlikely but impactful changes. Moreover, we will also incorporate the multi-agent coordination aspect to the target scenarios, in which agents have to negotiate the distribution of computation, data flow and utilization of resources. Agents might have different utility functions and goals that will add an interesting element to our network problem.

Finally, uncertainty and risk management is a key aspect of realistic assets networks for planetary exploration. Several aspects of exploration mission have uncertainty and can potentially be represented with stochastic models, such as task outcome and duration, vehicle failure, connectivity, bandwidth variations, and others. One promising research avenue is to incorporate probabilistic planning and scheduling approaches (Santana et al. 2016) to the computation sharing problem, as well risk-bounded techniques to provide guarantees that the network and the vehicles are able to operate within user specified bounds.

Acknowledgements

The research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

[Brdossy and Lehmann 1998] Brdossy, A., and Lehmann, W. 1998. Spatial distribution of soil moisture in a small catchment. part 1: geostatistical analysis. *Journal of Hydrology* 206(1):1 – 15.

[Doyle et al. 2013] Doyle, R.; Some, R.; Powell, W.; Mounce, G.; Goforth, M.; Horan, S.; and Lowry, M. 2013. High performance spaceflight computing (hpsc) next-generation space processor (ngsp): a joint investment of nasa and afrl. In *Proceedings of the Workshop on Spacecraft Flight Software*.

[Edwards et al. 2014] Edwards, C. D.; Barela, P. R.; Gladden, R. E.; Lee, C. H.; and Paula, R. D. 2014. Replenishing the mars relay network. In *2014 IEEE Aerospace Conference*, 1–13.

[Herzig et al. 2017] Herzig, S. J. I.; Mandutianu, S.; Kim, H.; Hernandez, S.; and Imken, T. 2017. Model-transformation-based computational design synthesis for mission architecture optimization. In *2017 IEEE Aerospace Conference*, 1–15.

[Hodges et al. 2016] Hodges, R. E.; Chahat, N. E.; Hoppe, D. J.; and Vacchione, J. D. 2016. The mars cube one deployable high gain antenna. In *2016 IEEE International Symposium on Antennas and Propagation (APSURSI)*, 1533–1534.

[Karras et al. 2017] Karras, J. T.; Fuller, C. L.; Carpenter, K. C.; Buscicchio, A.; McKeeby, D.; Norman, C. J.; Parcheta, C. E.; Davydychev, I.; and Fearing, R. S. 2017. Pop-up mars rover with textile-enhanced rigid-flex pcb body. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, 5459–5466. IEEE.

[Kerzhanovich et al. 2004] Kerzhanovich, V.; Cutts, J.; Cooper, H.; Hall, J.; McDonald, B.; Pauken, M.; White, C.; Yavrouian, A.; Castano, A.; Cathey, H.; Fairbrother, D.; Smith, I.; Shreves, C.; Lachenmeier, T.; Rainwater, E.; and Smith, M. 2004. Breakthrough in mars balloon technology. *Advances in Space Research* 33(10):1836 – 1841. The Next Generation in Scientific Ballooning.

[Korzun et al. 2010] Korzun, A. M.; Dubos, G. F.; Iwata, C. K.; Stahl, B. A.; and Quicksall, J. J. 2010. A concept for the entry, descent, and landing of high-mass payloads at mars. *Acta Astronautica* 66(7):1146 – 1159.

[Mattingly and May 2011] Mattingly, R., and May, L. 2011. Mars sample return as a campaign. In *2011 Aerospace Conference*, 1–13.

[Mounce et al. 2016] Mounce, G.; Lyke, J.; Horan, S.; Powell, W.; Doyle, R.; and Some, R. 2016. Chiplet based approach for heterogeneous processing and packaging architectures. In *2016 IEEE Aerospace Conference*, 1–12.

[Ono et al. 2015] Ono, M.; Fuchs, T. J.; Steffy, A.; Maimone, M.; and Yen, J. 2015. Risk-aware planetary rover operation: Autonomous terrain classification and path planning. In *Aerospace Conference, 2015 IEEE*, 1–10. IEEE.

[Ono et al. 2016] Ono, M.; Rothrock, B.; Almeida, E.; Ansar, A.; Otero, R.; Huertas, A.; and Heverly, M. 2016. Data-driven surface traversability analysis for mars 2020 landing site selection. In *Aerospace Conference, 2016 IEEE*, 1–12. IEEE.

[Powell et al. 2011] Powell, W.; Johnson, M.; Some, R.; Wilmot, J.; Gostelow, K.; Reeves, G.; and Doyle, R. 2011. Enabling future robotic missions with multicore processors. In *Infotech@ Aerospace 2011*. 1447.

[Rieber 2017] Rieber, R. R. 2017. Designing for a martian road trip: The mobility system for mars-2020. Keynote Talk: Mars Forum (URS: URS270204, CL17-5707).

[Santana et al. 2016] Santana, P.; Vaquero, T.; Toledo, C.; Wang, A.; Fang, C.; and Williams, B. 2016. Paris: A polynomial-time, risk-sensitive scheduling algorithm for probabilistic simple temporal networks with uncertainty. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

[Sell et al. 2013] Sell, S.; Chen, A.; Davis, J.; San Martin, M.; Serricchio, F.; and Singh, G. 2013. Powered flight design and reconstructed performance summary for the mars science laboratory mission. Technical report, Jet Propulsion Laboratory, National Aeronautics and Space Administration.

[Tokekar et al. 2016] Tokekar, P.; Hook, J. V.; Mulla, D.; and Isler, V. 2016. Sensor planning for a symbiotic uav and ugv system for precision agriculture. *IEEE Transactions on Robotics* 32(6):1498–1511.

[Wyatt et al. 2017] Wyatt, E. J.; Belov, K.; Burleigh, S.; Castillo-Rogez, J.; Chien, S.; Clare, L.; and Lazio, J. 2017. New capabilities for deep space robotic exploration enabled by disruption tolerant networking. In *2017 6th International Conference on Space Mission Challenges for Information Technology (SMC-IT)*, 1–6.

In Proceedings of the 14th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS), Madrid, Spain, June 2018.

An Approach for Autonomous Multi-rover Collaboration for Mars Cave Exploration: Preliminary Results

Tiago Vaquero and **Martina Troesch** and **Steve Chien**
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109

Abstract

Mars caves are promising targets for planetary science and human shelter. Exploring these environments would pose several challenges, including limited communication, lack of sunlight, limited vehicles lifetime that would not allow humans in the loop, and a totally unknown environment. Mission to these underground environments would required levels of autonomy, coordination and collaboration never been deployed before in rovers. In this paper we propose a multi-rover coordination algorithm and experimental framework for cave exploration missions. We describe preliminary experimental results with this coordination algorithm in a realistic simulated cave. We analyze rover coordination performance in different environmental settings and provide insights on potential opportunities for enhanced autonomy with AI planning and scheduling.

1. Introduction

Exploration of planetary caves is becoming an active research topic in the planetary science community and a promising scientific target for autonomous robotic explorers. Mars in particular offers exciting opportunities for (1) human settlements, (2) understanding the planet's evolution, and (3) the search of extraterrestrial life. Caves present the most mission effective habitat alternative for future human exploration, offering a stable, UV-shielding, meteoric-shielding environment (Boston et al. 2003), as well as access to minerals, gases and ice. Equally important, caves may preserve valuable information about the planet's history and evolution. Specifically, they offer stable physio-chemical environments, trapped volatiles, secondary mineral precipitation and microbial growth, which are expected to preserve bio-signatures and provide a record of past climate (Boston et al. 2005; 2004). Moreover, caves can potentially host water deposits which, through interaction with volcanic heat and minerals, could have created a favorable environment to microbial life preservation. What makes planetary caves even more attractive is that they are quite abundant. Mars for example has more than 2000 cave-related features identified, commonly associated with lava tubes, which provides a variety of promising targets for future exploration missions.

Robotic exploration missions on Mars would provide unique science opportunities for the cognitive and robotics communities, however, they present several challenges. Communicating with a rover into any of these caves and transmitting science data out is in itself a hard technical problem. Without a link to the surface, a rover would not be able to go far into the cave without losing contact with a base station. Moreover, because sunlight is not available in the cave, a mission is likely to last only a few days since the rovers will rely exclusively on battery power. Given limited communication, power and mission duration (just days), it is impractical to wait for humans' commands and feedback like in current Mars operations. For example, current MSL operations requires humans in the loop to plan sequences of actions for each sol based on downlinked data (Gaines et al. 2016). Those challenges alone require rovers far more autonomous than the existing surface rovers, for their environment is quite unknown and their communication with Earth is extremely limited, if at all.

Autonomy in multi-rover coordination is a key mission enabler that would help rovers to map and explore as much of the cave as efficiently as possible. With their very limited lifetime, rovers cannot wait for large parts of each day to receive directions from ground/Earth. The need for such multi-asset coordination was identified in recent studies in Mars cave exploration (Dubowsky et al. 2005; Kesner et al. 2007; Husain et al. 2013; Thangavelautham et al. 2014) and in Mars surface exploration (Clement and Barrett 2003; Yliniemi, Agogino, and Tumer 2014). The AI community has recently started to look into coordination techniques to map and explore Mars cave environments (Husain et al. 2013). One traditional approach would be to use a centralized task allocation and communication architecture to coordinate the rovers during exploration (Chien et al. 2000; Clement, Durfee, and Barrett 2007). However, this approach becomes unfeasible in a realistic cave environment due to intermittent, unreliable communication, as well as the high cost of communication power associated with the centralized scheme. Some existing work explores distributed techniques to coordinate vehicles to maintain connectivity between a base robot and a mobile explorer at all times in more controlled environments (auf der Heide and Schneider 2008; Stump, Jadbabaie, and Kumar 2008). These approaches can be leveraged to address subsurface missions, but they would need to be contextualized to environments with high likelihood of connectivity loss between rovers (sometimes done proactively by rovers to increase science utility) and unknown density and geometry of obstacles. Research on multi-rover coordination under these challenging constraints is in its infancy.

In this work, we propose a multi-rover coordination strat-

egy for cave exploration that aims to send rovers as deep into the cave as possible while also maximizing science data sent out to a surface base station. The proposed *Dynamic Zonal Relay with Sneakernet Relay Algorithm* is a two step algorithm. The first phase of the algorithm (Dynamic Zonal) drives each rover to a designated zone along the length of the cave, while maintaining communication distance between neighboring rovers. Each rover only takes science data in its designated zone and transmits it to the neighboring rover in the direction of the base station. Once at the end of its zone, the rover stops and becomes a relay point. The dynamic part of this algorithm is that if a rover is no longer operable, the other rovers would re-distribute the zones to maintain communication and characterization of the environment. The next step of the algorithm (Sneakernet Relay) would allow the rovers to acquire science data further in the cave by driving beyond the communication distance (intentional communication lost) and driving between neighboring rovers to relay the data out of the cave. We implement a simulation framework that (1) allows different multi-rover mission configurations, as well as (2) supports the measurement, evaluation, visualization and analysis of rover performance. We present preliminary results on the rovers and algorithm performance in a realistic simulated cave environment and rover configuration, including power and communication constraints, and science instruments and navigation system specification. The results provide initial insights to future mission design space, direction for algorithm improvements, as well as interesting opportunities for task planning and scheduling that would improve rover coordination, operation, communication and science return.

This paper is organized as follows. We first describe the multi-rover coordination problem for Mars cave exploration we address in this work and the particular elements of the mission. We then present the Dynamic Zonal Relay with Sneakernet Relay Algorithm in more detail. Next, we provide experimental results from a set of simulated Mars Cave exploration scenarios, in which a team of four rovers explore a realistic-size cave with varying obstacle densities. We analyze the performance of the coordination algorithms with respect to a score based on cave coverage transmitted out of the cave, mission life span, distribution of energy and time spent in different rover activities. Finally, we discuss the results, potential roles for AI planners and schedulers, as well as future directions.

2. Example Problem Definition

Among the several mission challenges related to deploying and controlling a set of rovers in a Mars cave, in this work we focus on the hypothetical problem of autonomously coordinating multiple rovers to (1) map and characterize a Martian cave as far into the cave as possible from the entrance, and (2) to transmit as much science data collected by the rovers' instruments as possible out of the cave to a lander (base station), which will then take care of transmitting it to scientists on Earth. Figure 1 illustrate a Martian lave tube structure, with the lander positioned at the entrance and a set of science rovers exploring the cave interiors. We provide more details and constraints on the cave environment and rover platform in what follows.

2.1 Cave Environment

In this paper we focus on Martian caves associated with lava tubes. Due to Mars' lower gravity, Martian lava tubes are much larger than Earth lava tubes. Herein we target caves that are approximately 100 meters wide and potentially hundreds of meters deep, with a skylight entrance formed from a collapsed cave ceiling as illustrated in Figure 1. We assume that the terrain in the interior of the cave is unknown a priori.

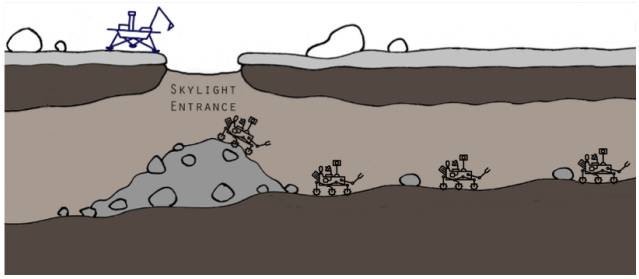


Figure 1: Illustration of a hypothetical multi-rover coordination problem in Mars cave exploration. Rovers not to scale. Credit: Figure adapted from the Wikimedia Commons, Longitudinal cross-section of a martian lava tube with skylight.

Cave walls are a quite interesting science target for NASA/JPL scientists. They can provide critical constraints on lava temperature and cooling history, leading to insights into Martian magmatic processes and differentiation. Thus, in our coordination problem the rovers should try to safely remain as close to the walls as possible to characterize wall properties and facets.

2.2 Conceptual Autonomous Rovers

We consider a set of homogenous rovers that are assumed to be successfully deployed at the bottom of the cave through the skylight entrance. The problem of deploying the rovers into the cave, although interesting, is not in the scope of this work. The focus is in the exploration and coordination problem while in the cave where communication is limited.

The rovers are equipped with a battery module, mobility components, a communication component (antennas), and a science component with a set of key science instruments. Those components allow each rover to perform the following actions:

- **Ping** (communication component): a rover can send a ping to all rovers within communication range to detect the vehicles around it. Rovers (including the base rover) in the communication range respond with their position and status update. A ping process has a specific duration (e.g., 2 seconds) and also a power consumption rate known a priori. Communication range and ping duration are provided in the antenna specs.
- **Drive** (mobility components): to navigate the environment safely, each rover is able to detect obstacles within a radius (e.g. 5 meters) in 360 degrees. The cave map is stored during exploration - given that the focus of this work is not on mapping and localization per se, we assume that the knowledge about the map and coverage becomes available to all the rovers as they explore the cave. The velocity of the vehicle and power consumption during driving is known and given by the mobility specification.
- **Science** (science component): each rover has the same set of science instruments partitioned in three categories: *primary instruments*, *secondary instruments*, and *periodic*

instruments. Each one of these instruments has its own specs for power consumption, data volume generated by each reading and the sensing duration.

- **Transfer** (communication component): transfer is a collaboration task in which the sender first sends a transfer request to a target/receiver rover. The receiver then informs the sender when it is available to receive data. Once that confirmation is received, the sender transfers the data to the target rover. When the data is successfully transferred, the receiving rover sends a confirmation and the task is completed. The duration of the actual data transfer between two rovers (lander and/or science rover) is determined by the antenna specs, the data volume and the distance from each other (bandwidth). The bandwidth can be modeled with an arbitrary function (see Discussion). For the simulations presented in this paper, we model bandwidth as a step-wise function of distance between communicating vehicles. For example between 0-5 meters rovers can transfer data at 11.0 Mbps, between 5-10 meters at 5.5 Mbps, between 10-15 meters at 2.0 Mbps, and between 15-25 meters at 1.0 Mbps. Power consummation rates are also known and are constant during communication, regardless of distance.

In additional to the above action specification, we list below some of the key assumptions on the exploration problem:

1. All actions consume energy from the battery component, which is a limited resource. If the battery drains out, the rover becomes non-operational.
2. We consider a constant hotel load that represents the energy consumption to keep the rover operational. We frame any cognitive process (e.g., decision making, path planning computation) as part of this constant consumption.
3. Each rover can only execute one action at a time, except sending and responding to pings. In the science case, only one instrument can be used at a time.
4. Communication model does not consider the shape, texture, material of the cave or proximity to walls. (This is actually already being incorporated in our models, but will be left for future publications.)
5. Communication is possible only up to a fixed distance between rovers, where the lander has a longer fixed range.
6. Rover can fail during exploration, which means that the coordination has to account for reconfiguration.
7. In this work we are not modeling acceleration or slippage in the motion model.
8. Finally, each rover does have a memory component for data science storage, but the memory capacity is large enough to handle days or weeks worth of data.

3. Approach

We propose a multi-rover coordination strategy for cave exploration that aims to send rovers as deep into the cave as possible while also maximizing data sent out to a surface base station.

The rovers explore the cave using the *Dynamic Zonal Relay with Sneakernet Relay Algorithm*, which is a two phase algorithm, starting with (1) *Dynamic Zonal Relay* and expanding with (2) *Sneakernet Relay*. One of the main aspects of this algorithm is the use of spatial zones to determine the state of the rover. Each zone is a distinct section of the cave based on distance from the lander, as shown in Figure 2.

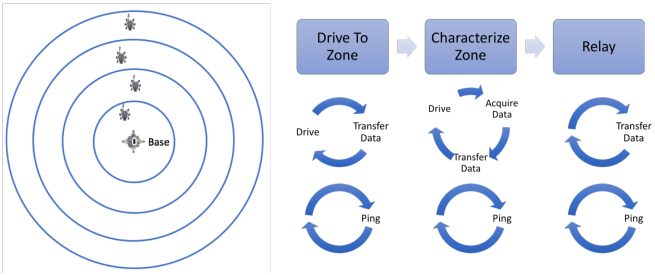


Figure 2: Zones based on the distance from the conceptual lander or base station (left). Nominal state transitions for the Dynamic Zonal Relay phase (right).

3.1 Dynamic Zonal Relay

The first phase, *Dynamic Zonal Relay*, assigns the rovers to designated, adjacent zones that keep the rovers within communication range of their immediate neighbors. The algorithm is *dynamic* in that if any rover becomes inactive (i.e., no longer communicating due to some kind of failure or running out of battery), the other rovers dynamically readjusts the zone assignments.

While *driving* to its assigned zone, the rover maintains a safe communication distance with its neighboring rovers and relays any science data that has been transferred to it to its neighbor in the direction of the lander. When in its zone, the rover moves along the length of the cave, continuing to maintain communication distance, while *characterizing* the cave. The rover sends acquired science data to the neighboring rover closest to the lander. Once at the end of its zone, the rover becomes a *relay* point. In this state, the rover transfers any remaining science data that it has collected, as well as any science data that has been transferred to it, to its neighbor closest to the lander.

A diagram of the nominal state transitions for the Dynamic Zonal Relay phase is shown in Figure 2. The diagram also shows that the rovers perform periodic pings to the other rovers to share status information, such as position, and to keep track of which rovers are still active.

In the case that a rover becomes inactive, the surrounding rovers readjust depending on their position relative to the inactive rover. Rovers closer to the lander would not need to adjust their zones; however, they need to be made aware of the new configuration. Rovers deeper into the cave need to adjust their zone closer to the other rovers to re-establish a continuous line of communication across all rovers. Since the rovers do not know how much science data the inactive rover was able to acquire and transfer out of its zone (if it was already characterizing its zone), all rovers that moves into a new zone re-characterize the entire zone.

3.2 Sneakernet Relay

Once all of the data that was collected during the Dynamic Zonal phase has been passed to the lander, the rovers transition to the *Sneakernet Relay* phase. During this phase, the rover furthest into the cave is designated as the lead rover (e.g., *Rover4* in a team of four rovers) and the others are designated as relayers (e.g., *Rover1*, *Rover2* and *Rover3* in the

team of four rovers). The lead rover is now tasked with characterizing the next zone, which means that one of the relayers is no longer in communication range of one of its neighbors, meaning that it must *sneakernet*. Increased sneakernet distance is assigned in order, starting with the rover closest to the lander (e.g., *Rover1* in our example), as the lead rover characterizes more zones.

The sneakernetting process is composed of cycles, where a sneakernet cycle consists of each rover incrementally increasing its sneakernetting distance. A cycle is further broken down into stages that are repeated with each assignment of increased distance: (1) extension/replacement and characterization, (2) relay, and (3) confirmation. Except at the beginning of the Sneakernet Relay phase, stage (1) and stage (3) occur simultaneously. Figure 3 helps to illustrate the evolution of the Sneakernet Relay phase, with line 1 showing the positions of the rovers for a three rover mission configuration at the end of the Dynamic Zonal Relay phase.

The beginning of a cycle is triggered by the rover closest to the lander (*Rover1*) beginning the *extension/replacement and characterization* stage, as shown in Figure 3 line 2. The initiator of this stage (in this case, the rover closest to the lander) moves forward to the relay position of its neighbor. This triggers the neighbor rover to move forward to the relay position of the rover in front of it, and so on, until the lead rover. When the lead rover is triggered, it moves forward and characterizes the next section of the cave, which is the same distance as that of the relay distance of the previous rover (the distance between the leader's neighbor and the neighbor's neighbor).

When the lead rover has finished collecting new data (line 3), the *relay* stage is initiated. The lead rover begins by moving within communication range of the rover following it and transferring all of its data. After the transfer, the transferring rover remains where it is while the receiving rover moves to communication range of its neighbor in the direction of the lander and transfers all of the data, and so on, until the rover closest to the lander transfers all of the data out of the cave. In Figure 3, line 4 shows the first rover requiring to move in order to transfer the data to the lander.

The transfer of all of the data to the lander triggers the next stage, *confirmation*. The rover closest to the lander now moves back to its neighbor inside the cave, confirms that the transfer was successful, and returns to its previous relay position (line 5). The next rover then moves deeper into the cave to its neighbor and reports the confirmation and returns to its relay position, and so on for all of the rovers until the confirmation reaches the lead rover. During this stage, the next rover to initiate extension moves to its next relay position during the confirmation process, triggering all rovers to move deeper into the cave as a cascading sequence of extension and confirmation, such as on line 6. In line 7, we see the lead rover moving ahead and characterizing a zone the same distance as that of the relay distance of the previous rover (distance between *Rover2* and *Rover1*, as described previously, requiring the lead rover to sneakernet on line 8.

The remainder of Figure 3 shows the repetition of these stages, until line 13, which shows the positions of the rovers after the second cycle is initiated by the first rover (*Rover1*).

To remain robust to rover failures during the Sneakernet Relay phase when the rovers are no longer in communication range, the rovers rely on timeouts to estimate how long they should wait for a neighbor to initiate the next phase. If a timeout is reached, they will try to find its neighbor in the direction of the lander to re-establish the relay chain. If a relayer reaches a timeout waiting for its peer deeper in the cave, it will then act as the leader.

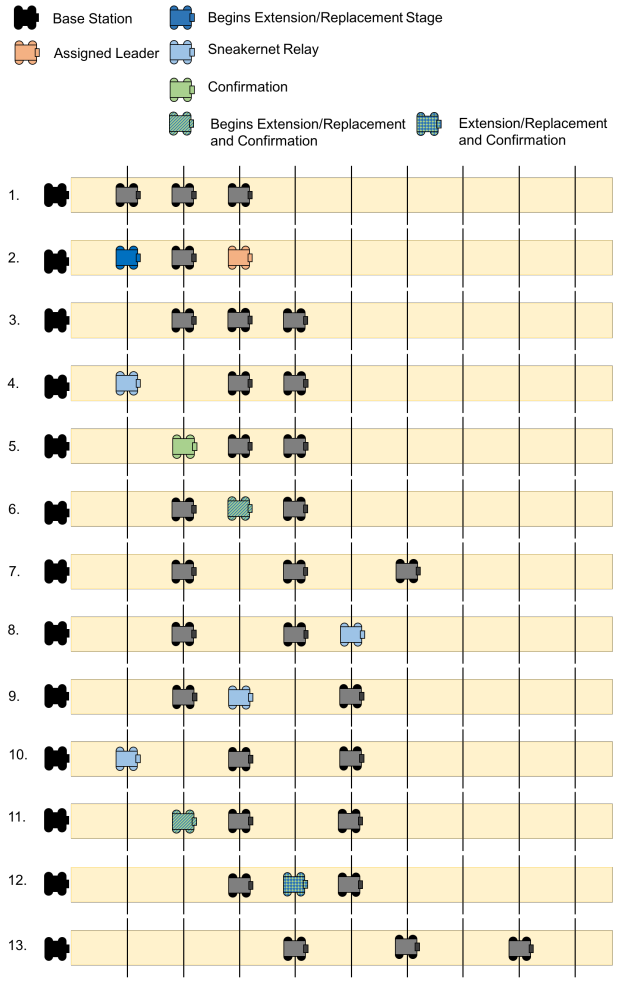


Figure 3: The movement of the rovers during the Sneakernet Relay phase of the algorithm for the first cycle.

4. Experiments in Simulation

The Dynamic Zonal Relay with Sneakernet Relay Algorithm was tested in a simulation framework using the Robot Operating System (ROS) to model the communication between the rovers as well as to model the different rover components (e.g. the science instruments, driving and navigation, etc.) and the cave.

A configuration with four rovers (*Rover1* through *Rover4*) and a base station was used, as illustrated in Figure 1, for the experiments. This configuration is based on preliminary cost and payload analysis of similar classes of missions. The cave model used is a model of the Cassone Cave (Santagata), scaled approximately twelve times so that the width is around 70 m, which is shown in Figure 4. The cave model is made up of approximately 350,000 triangular facets, with an average size 1.16 m^2.

Each rover is assumed to have an identical suite of instruments, partitioned in the three aforementioned categories: primary, consisting of a LiDAR to characterize the walls, facets and structure of the cave; secondary, including a color

camera and a spectrometer; and periodic instruments, including a thermometer, radiation detector, and hygrometer. Primary and secondary instruments are used based on movement of the rover, whereas the periodic instruments are used based on a regular, timed cadence (in this case, every 60 minutes). A summary of the assumed instrument parameters is shown in Table 1.

Table 1: Assumed Instrument parameters

	Power (Watts)	Data Volume (Mb)	Sensing Duration (s)
LiDAR	10.0	1344	5.0
Color Camera	5.0	150.0	1.0
Spectrometer	10.0	14.4	660.0
Thermometer	1.0	0.0008	5.0
Radiation Detector	1.0	0.0008	5.0
Hygrometer	2.0	0.0008	5.0

The communication range was limited to 25 meters between rovers and 75 meters to the lander. Pings to communicate position and status were performed once per minute. The assumed power for communication was 4.0 Watts.

In this work we incorporate a simple approach for rover navigation and selection of the region to be explored in the cave. Each rover computes its path through the cave map using the A* algorithm, where *Rover1* through *Rover3* move toward the rovers ahead of them, while the leader, *Rover4*, uses a frontier detection algorithm (Yamauchi 1997) to move towards unexplored regions of the map into the cave. In this experiment, rovers traverse the environment at 0.005 m/s and a 5-meter range is used for obstacle detection and mapping. It is assumed that driving requires 14.0 Watts of power.

We also model a hotel load (the amount of power required for a rover to remain operational, such as basic heating and health monitoring) of 5.0 Watts.

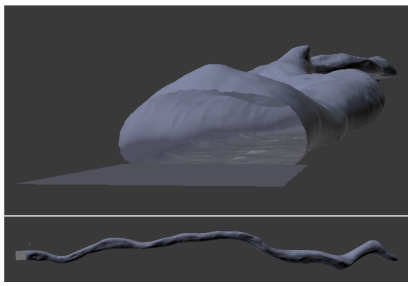


Figure 4: Simulated cave front and top views. Model of the Cassone Cave (Santagata), scaled approximately twelve times.

To test a perfect scenario where the cave has no obstacles and the rovers are able to function until they run out of energy, one experiment was performed with zero obstacles in the cave and no random dying of the rovers. Two further experiments were performed, again with rovers able to function until they run out of energy, with random obstacle densities of 10% and 20% to show how the simulation can evaluate the success of a mission where there are rocks and debris throughout the cave. In order to show the robustness of the algorithm to loss of rovers, another experiment was run with zero obstacles and a random chance of rovers dying during the run.

As a comparison, an experiment was performed with a single rover using the Dynamic Zonal Relay with Sneakernet Relay algorithm, where the rover extends by a single zone (in this case 20 m) at each step. No obstacles were used for this experiment.

To evaluate the different runs, a scoring function based on the cave characterization data transferred out of the cave was used. For remote instruments (such as cameras), the score, s_{remote}, is the area of the triangular facets covered in the cave model based on the position of the rover, the field of view of the instrument, and any restrictions on far and near clip planes or normal angle of the facet, which is summarized in Eq. 1 for a data acquisition instance data_i.

$$s_{remote}(\text{data}_i) = \sum_{\text{cave facets, } f} \text{area}(f), \text{if } f \text{ visible} \quad (1)$$

For in-situ instruments (e.g. temperature sensors), the score, $s_{in-situ}$, depends on both position and time of the data. For these types of measurements, the value of the data decreases if it is taken at almost the same position and time, therefore the score is a function that decays based on the position and time of any previously taken data of that type. Given a max time of T before a facet can receive a full score again, $s_{in-situ}$ is determined by Eq. 2 for a data acquisition instance data_i.

$$s_{in-situ}(\text{data}_i) = \sum_{\text{cave facets, } f} \text{area}(f) * d, \text{if } f \text{ visible}, \quad (2)$$

where,

$$d = \begin{cases} 1 & \text{if } f \text{ last measured } > T \text{ seconds ago} \\ e^{-(T-\Delta t)/T} & \text{otherwise} \end{cases}$$

and visibility is based on a sphere with a fixed radius instead of a field of view and clip planes.

This results in a total score, $score$, defined by Eq. 3, where only data that is transferred out of the cave is scored.

$$score = \sum_{\text{remote data acquisitions, } i} s_{remote}(\text{data}_i) + \sum_{\text{in-situ data acquisitions, } j} s_{in-situ}(\text{data}_j) \quad (3)$$

5. Simulation Results

In what follows we present the results from the single rover and the multi-rover experiments using the simulator. A comparison of the results are shown in Table 2.

5.1 Single Rover

The single rover was able to explore up to 100 meters into the cave; however it was only able to transfer data from up to 80 meters. This can be seen from Figure 5, which shows the depth into the cave that the rover travelled with respect to time; the rover was not able to make it back close enough to the lander after characterizing up to 100 meters to transfer its most recently collected data (i.e., data collected between 80-100 meters).

The percentage of time that the rover spent performing different activities is show in Figure 6, which demonstrates that the amount of time required to drive and transfer the data is quite significant, especially with respect to the amount of

Table 2: Comparison of simulated experiments

	Max Lifetime (days)	Max Transferred Data Distance (m)	Score	Data Volume Transferred (GB)	Data Volume Un-Transferred (GB)	Rover Death	Death Time (days)
Single Rover	1.59	80	1122.76	4.70	1.39		
0% Obstacles	2.99	100	3828.27	6.44	1.02		
10% Obstacles	3.00	100	4285.45	6.76	1.39		
20% Obstacles	3.41	45	3347.68	4.34	0.0000077		
Random Death	2.69	75	2452.43	5.20	2.41	Rover4, Rover1	0.19, 1.94

time spent acquiring the data (labeled "Science"). However, when looking at the percentage of energy required, Figure 7, the transfers make less of an impact, whereas driving continues to have the greatest impact.

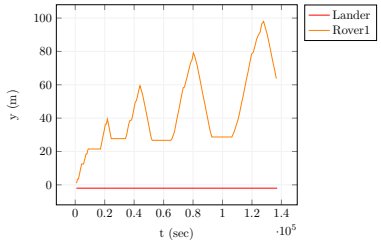


Figure 5: Simulated depth of the rover into the cave (y position) with respect to time.

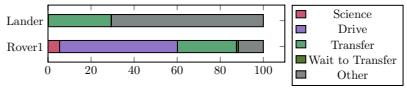


Figure 6: Percentage of time spent on different activities.

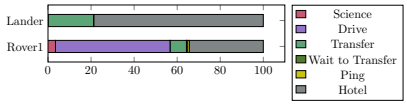


Figure 7: Percentage of energy required to perform different activities.

5.2 Four Rovers

Figure 8 shows an example of the motion of the rovers expanding and sneakernetting in the four rover configuration with zero obstacles and no random death.

The percent breakdown of activities in terms of time is displayed in Figure 9, where it is shown that transferring data (either receiving or sending) takes a large portion of a rover's time, increasing for the rovers closer to the lander. In fact, comparing Figure 6 and Figure 9, *Rover1* spends approximately as much time transferring as the single rover. However, the largest amount of time is spent performing "other" activities, which includes pings and idle time. Unlike in the single rover scenario, with multiple rovers there are times that the state transition of a rover depends on the actions and states of the surrounding rovers, meaning that the rover must wait idly, which is why the *Other* time is so high in Figure 9 compared to Figure 6.

Figure 10 shows the energy distribution for the four rovers. Like the single rover scenario, in terms of specific activities, driving takes the most amount of energy. However, in the multi-rover scenario, each rover spends much less energy performing science activities, with *Rover4* using the most energy on science, as expected since it characterized more zones. Although a small percentage of the overall energy required by the rovers, in Figure 10, it can be seen that it is not an insignificant source of energy usage.

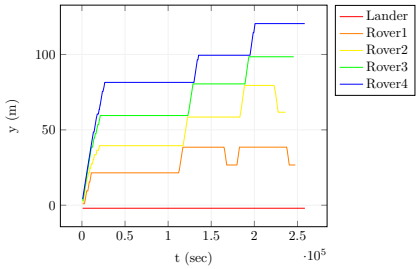


Figure 8: Simulated depth of the rovers into the cave (y position) with respect to time for a four rover sneakernet configuration with zero obstacles and no chance of random death.

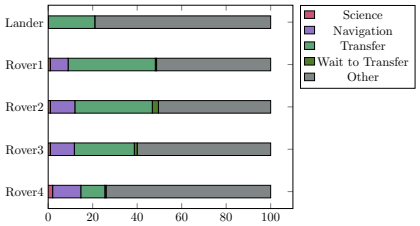


Figure 9: Percentage of time spent on different activities for the four rovers and the lander in the simulation.

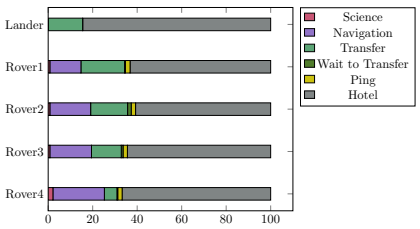


Figure 10: Percentage of energy required to perform different activities for the four rovers and the lander in the simulation.

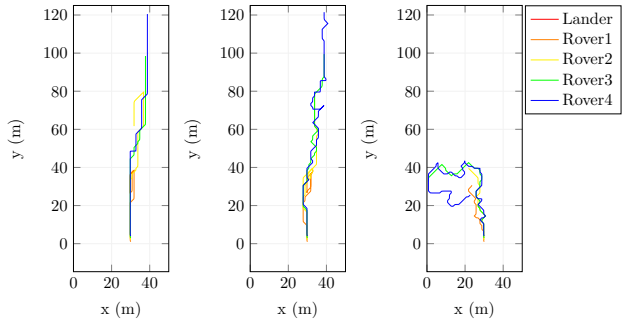


Figure 11: Paths of the rovers with 0% obstacle density (left), 10% obstacle density (center) and 20% obstacle density (right) with no random rover death.

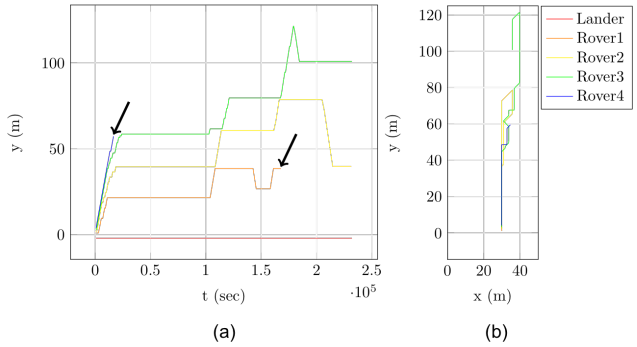


Figure 12: a) Simulated depth of the rover into the cave (y position) with respect to time. The arrows point out times and locations of rover deaths. b) Paths of the rovers in the random dying rovers experiment.

With more obstacles, the paths of the rovers, shown in Figure 11 for 0%, 10%, and 20% obstacle densities and no random death, become less straight and aligned with the cave wall. In fact, with 20% obstacle density, the rovers are not able to find a path that stretches beyond 45 meters into the cave, and the rovers begin exploring farther from the cave wall. This means that *Rover4* (the leader) never reaches its zone and does not take any primary or secondary data.

The experiment with random rover death shows a scenario where *Rover4* dies just before *Rover3*, its immediate follower, finishes characterizing its zone, and *Rover1* dies after the first transfer of the first sneakernet expansion data. From Figure 12 (a), we see that *Rover3* seemlessly becomes the new leader and leads the way during the algorithm's Sneakernet Relay phase. As shown in Figure 3, *Rover3* expands by its neighbors relay distance.

We also see a timeout begin in Figure 12 with *Rover2*. When *Rover2* sneakernets back toward *Rover1* to relay the data, it is not able to locate *Rover1*, therefore *Rover2* waits at the location it last saw *Rover1* for a timeout duration (which in this scenario, *Rover2* does not live long enough to finish). Although there are only three rovers, they are able to collect data beyond 100 meters, but are only able to live long enough to transfer out data up to 75 meters into the cave.

Figure 12 (b) shows the paths of the rovers for the experiment with random rover deaths, which looks very much like the 0% obstacle paths in Figure 11, as expected.

6. Discussion

The simulation shows that a single rover can successfully characterize up to 80 meters along a cave wall (given no obstacles) if it does not encounter any problems before running out of battery; however, this is a large assumption given the unknown environment of the cave. The sneakernet results with randomly dying rovers shows the robustness of the Dynamic Zonal Relay with Sneakernet Relay algorithm to rover loss. Furthermore, with the survival of all rovers for the duration of the battery charge, science data from deeper into the cave can be transferred out to the lander than in the single rover case.

Although the experiment with 20% obstacle density showed the rovers unable to reach as deep into the cave as other scenarios, it is interesting to note the large score. This is due to the fact that as the rovers move farther away from the cave wall, the field of view of the remote instruments is able to capture a larger section of the cave model facets. This shows an interesting trade-off that can be made between remaining close to the wall, and therefore characterizing smaller features of the cave and reaching deeper distances, versus moving away from the wall and characterizing larger sections.

In our exploration approach, the sequencing of science actions is predefined based on scientist team inputs. Nevertheless, an automated and opportunistic sequencing of science actions could provide a higher science utility. Onboard data analysis and science goals and instrument prioritization techniques are described in (Chien et al. 2016). Castano et al. (2007) describes the Onboard Autonomous Science Investigation System (OASIS), an autonomous system that is capable of analyzing imagery to generate new science tasks for execution both in simulation and on a test rover. Wettergreen et al. (2014) shows the capability of autonomous sample location selection and adaptive path planning on a rover in a deployment to the Atacama Desert. Woods et al. (2009) demonstrates the feasability of autonomous opportunistic science with autonomous instrument placement for contact science. All of these algorithms and techniques would support desirable autonomous behavior. Moreover, our proposed approach has room for improvement with respect to the rovers responsible for relaying data. More opportunistic decision making approaches would allow relayers to potentially perform additional science tasks while also managing the task of relaying data out of the cave.

Coordination of data transfer is also an opportunity for cognitive systems. As opposed to waiting for a target rover to be available to receive data, a scheduling system could support a more efficient data transfer coordination between rovers (Clement and Barrett 2003) - assuming they can share their status and activities. The communication model and respective ranges have a great impact on this coordination. We are working on incorporating a stochastic communication model in which bandwidth degrades as a function of distance and does not have a hard constraint on the maximum distance (e.g., 25-meter max range). That provides opportunities for rovers to establish a comm link in greater distance and provides options to route data science out of the cave through different rovers. A more realistic package management during communication would make the routing problem even more interesting, in which science data could be partitioned into smaller pieces and sent to different rovers over time depending on bandwidth variations. Here we assume that data packages would be able to be prop-

erly combined at the target asset (e.g., lander or an orbiter). Such stochastic models would also create scenarios in which rovers are physically close but with a poor or unexisting communication link.

7. Conclusion

In this paper we proposed a multi-rover coordination algorithm for Mars Cave exploration. A simulation framework was created to evaluate the performance of the algorithm and to study mission configurations to explore design options for future missions to underground cavities in other planets and moons. We utilized realistic cave settings and vehicle specs to generate an initial evaluation of the feasibility of the multi-rover approach for science data collection. We also discussed opportunities for AI planning and scheduling techniques to augment rover autonomy and efficiency with respect to science utility.

This is an ongoing research project with several promising immediate next steps and future directions. In the short-term we will investigate the impact on rover performance when increasing action concurrency. More specifically, in the simulation we will allow rovers to transfer data while navigating the environment and doing science. We will also incorporate data routing techniques with the aforementioned stochastic communication model we are integrating. We are also interested in augmenting the proposed algorithm to help rovers to better coordinate data transfer and to balance data relay and science tasks.

8. Acknowledgments

Portions of this work were performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration. We thank the valuable inputs from other JPL team members, including Jay Wyatt, Joseph Lazio, Jay Gao, Abigail Fraeman, Julie Castillo-Rogez, Sebastian Herzig, Konstantin Belov, and Amos Byon.

References

auf der Heide, F. M., and Schneider, B. 2008. Local strategies for connecting stations by small robotic networks. In Hinchey, M.; Pagnoni, A.; Rammig, F. J.; and Schmeck, H., eds., *Biologically-Inspired Collaborative Computing*, 95–104. Boston, MA: Springer US.

Boston, P.; Frederick, R.; Welch, S.; Werker, J.; Meyer, T.; Sprungman, B.; Hildreth-Werker, V.; Thompson, L.; and Murphy, D. 2003. Human utilization of subsurface extraterrestrial environments. *Gravitational and Space Biology Bulletin* 26(2).

Boston, P.; Frederick, G.; Welch, S.; Werker, J.; Meyer, T.; Sprungman, B.; Hildreth-Werker, V.; Murphy, D.; and Thompson, S. 2004. System Feasibility Demonstrations of Caves and Subsurface Constructed for Mars Habitation and Scientific Exploration. Technical report, USRA Reports, NASA Institute for Advanced Concepts.

Boston, P.; Spilde, M.; Northup, D.; Melim, L.; Soroka, D.; Kleina, L.; Lavoie, K.; Hose, L.; Mallory, L.; Dahm, C.; Crossey, L.; and Schelble, R. 2005. Cave biosignature suites: microbes, minerals, and mars. *Astrobiology* 1(1):25–55.

Castano, R.; Estlin, T.; Anderson, R. C.; Gaines, D. M.; Castano, A.; Bornstein, B.; Chouinard, C.; and Judd, M. 2007. Oasis: Onboard autonomous science investigation system for opportunistic rover science. *Journal of Field Robotics* 24(5):379–397.

Chien, S.; Barrett, A.; Estlin, T.; and Rabideau, G. 2000. A comparison of coordinated planning methods for cooperating rovers. In *International Conference on Autonomous Agents (Agents 2000)*.

Chien, S.; Thompson, D. R.; Castillo-Rogez, J.; Rabideau, G.; Bue, B.; Knight, R.; Schaffer, S.; Huffman, W.; and Wagstaff, K. L. 2016. Agile science - a new paradigm for missions and flight software.

Clement, B. J., and Barrett, A. C. 2003. Continual coordination through shared activities. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '03, 57–64. New York, NY, USA: ACM.

Clement, B.; Durfee, E.; and Barrett, A. 2007. Abstract reasoning for planning and coordination. *Journal of Artificial Intelligence Research* 28:453–515.

Dubowsky, S.; Iagnemma, K.; Liberatore, S.; Lambeth, D. M.; Plante, J. S.; and Boston, P. J. 2005. A concept mission: Microbots for largescale planetary surface and subsurface exploration. *AIP Conference Proceedings* 746(1):1449–1458.

Gaines, D.; Anderson, R.; Doran, G.; Huffman, W.; Justice, H.; Mackey, R.; Rabideau, G.; Vasavada, A.; Verma, V.; Estlin, T.; et al. 2016. Productivity challenges for mars rover operations. In *Proceedings of 4th Workshop on Planning and Robotics (PlanRob)*, 115–125. London, UK.

Husain, A.; Jones, H.; Kannan, B.; Wong, U.; Pimentel, T.; Tang, S.; Daftry, S.; Huber, S.; and Whittaker, W. L. 2013. Mapping planetary caves with an autonomous, heterogeneous robot team. In *IEEE Aerospace Conference*, 1–13.

Kesner, S. B.; Plante, J. S.; Boston, P. J.; Fabian, T.; and Dubowsky, S. 2007. Mobility and power feasibility of a microbot team system for extraterrestrial cave exploration. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, 4893–4898.

Santagata, T. Università degli Studi di Modena e Reggio Emilia. Inside the Glacier Project.

Stump, E.; Jadbabaie, A.; and Kumar, V. 2008. Connectivity management in mobile robot teams. In *ICRA*, 1525–1530. IEEE.

Thangavelautham, J.; Robinson, M. S.; Taits, A.; McKinney, T.; Amidan, S.; and Polak, A. 2014. Flying, hopping pit-bots for cave and lava tube exploration on the moon and mars. In *The 2nd International Workshop on Instrumentation for Planetary Missions*.

Wettergreen, D.; Foil, G.; Furlong, M.; and Thompson, D. R. 2014. Science autonomy for rover subsurface exploration of the atacama desert. *AI Magazine* 35(4):47–60.

Woods, M.; Shaw, A.; Barnes, D.; Price, D.; Long, D.; and Pullan, D. 2009. Autonomous science for an exomars rover–like mission. *Journal of Field Robotics* 26(4):358–390.

Yamauchi, B. 1997. A frontier-based approach for autonomous exploration. In *IEEE International Symposium on Computational Intelligence in Robotics and Automation, CIRA'97*, 146–151.

Yliniemi, L.; Agogino, A.; and Tumer, K. 2014. Multi-robot coordination for space exploration. *AI Magazine* 35(4):61–74.

Using Squeaky Wheel Optimization to Derive Problem Specific Control Information for a One Shot Scheduler for a Planetary Rover

Wayne Chi, Steve Chien, Jagriti Agrawal
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109
{firstname.lastname}@jpl.nasa.gov

Abstract

We describe the application of using Monte Carlo simulation to optimize a schedule for execution and rescheduling robustness and activity score in the face of execution uncertainties. We apply these techniques to the problem of optimizing a schedule for a planetary rover with very limited onboard computation. We search in the schedule activity priority space - where the onboard scheduler is (a) a one shot non-backtracking scheduler in which (b) the activity priority determines the order in which activities are considered for placement in the schedule and (c) once an activity is placed it is never moved or deleted. We show that simulation driven search outperforms a number of alternative proposed heuristic static priority assignment schemes. Our approach can be viewed using simulation feedback to determine problem specific heuristics much like squeaky wheel optimization.

Introduction

Embedded schedulers must often perform within very limited computational resources. We describe an approach to automatically deriving problem specific control knowledge for a one-shot (non-backtracking) scheduler intended for a planetary rover with very limited computing. In this application, the onboard scheduler is intended to make the rover more robust to run-time variations (e.g., execution durations) by rescheduling. Because the general structure of the schedule is known a priori on the ground before uplink, we use both analysis of the schedule dependencies and simulation feedback to derive problem specific control knowledge to improve the onboard scheduler performance.

The target onboard scheduler is a one-shot limited search scheduler. Because the scheduler does not backtrack across activity placements, the order in which it considers activities heavily influences generated schedule quality. In our approach, we search the space of activity priorities which determine the order in which the scheduler considers activity placement. At each step in the priority search, a Monte Carlo simulation is conducted to assess the likelihood of an activity being executed. Using an approach analogous to squeaky wheel optimization, these runs are automatically analyzed and used to feed back into adjustments to the activity priorities (and hence the order in which they are considered for inclusion in the schedule for both initial schedule generation and rescheduling). This search in the activity priority space continues until all requested activities are included or a resource bound is exceeded. We call this method *Priority Search* and we present empirical results that show that *Priority Search* outperforms several static priority assignment methods (those that do not use Monte Carlo feedback) *including manual expert derived priority setting*.

We study this problem in the context of setting activity priorities as part of the ground operations process for a one-shot, non-backtracking scheduler (Rabideau and Benowitz 2017) designed to run onboard NASA's next planetary rover, the Mars 2020 (M2020) rover (Jet Propulsion Laboratory 2017a). For our problem, the onboard scheduler is treated as a predetermined "black box".

The remainder of the paper is organized as follows. First we describe our formulation of the scheduling problem, metrics for schedule goodness, and the onboard scheduling algorithm. Second, we describe several static approaches to priority assignment as well as our *priority search* approach that leverages Monte Carlo simulation feedback. Third, we describe empirical results demonstrating the efficacy of *priority search* over static methods, evaluating on *sol types*, the best available anticipated operations plans for the M2020 planetary rover mission. Finally, we describe related and future work and conclusions.

Problem Definition

For our defined scheduling problem (Rabideau and Benowitz 2017), the scheduler is given

- a list of activities $A_i\langle p, R, e, dv, \Gamma, T, D\rangle \ldots A_n\langle p, R, e, dv, \Gamma, T, D\rangle$
- where p is the scheduling priority of the activity, and
- R is the set of unit resources $R_1 \ldots R_m$ that the activity will use (up to project limitations - 128 for M2020), and
- e and dv are the rate at which the consumable resources energy and data volume respectively are consumed by the activity, and
- Γ are non-depletable resources used such as sequence engines available or peak power, and

- T is a set of the activity's optional a) start time windows $T_{i\_start} \dots T_{i\_end}$ and b) preferred schedule time $T_{i\_preferred}$, and
- D is a set of the activity's dependency constraints from $A_j \rightarrow A_k$ [1]

All activities are *Mandatory Activities*. These are activities, $m_1 \dots m_j \subseteq A$, that must be scheduled as long as the given set of inputs are *valid*. In order for a set of inputs to be considered *valid*, there must exist a valid (e.g. constraint satisfying) schedule - in the context of the scheduler - that includes all of the mandatory activities. Note that the M2020 Onboard Scheduler is an incomplete algorithm. As a result, there could be a set of inputs where valid schedule exists and a complete scheduler would place all mandatory activities, but the Onboard scheduler would not. Since not all input sets will be *valid*, it is important for us to modify the input sets (e.g. changing priorities) to allow all mandatory activities to be scheduled.

In addition, activities can be grouped into *Switch Groups*. A *Switch Group* is a set of activities where exactly one of the activities in the set must be scheduled. The activities within a switch group are called *switch cases* and vary only by how many resources (time, energy, and data volume) they consume. Switch groups allow us to schedule a more resource-consuming activity if it will fit in the schedule. For example, one of the M2020 instruments takes images to fill mosaics which can vary in size; for instance we might consider $1x5$, $3x5$, or $5x5$ mosaics. Taking larger mosaics might be preferable, but taking a larger mosaic takes more time, takes more energy, and produces more data volume. These alternatives would be modeled by a switch group that might be as follows:

$$SwitchGroup = \begin{cases} Mosaic_{1x5} & \text{Duration=100 sec} \\ Mosaic_{3x5} & \text{Duration=200 sec} \\ Mosaic_{5x5} & \text{Duration=400 sec} \end{cases} \quad (1)$$

In the above example, the scheduling priority order would be $Mosaic_{1x5}$ the lowest of the three, then $Mosaic_{3x5}$, and $Mosaic_{5x5}$ the highest. The desire is for the scheduler to schedule the activity $Mosaic_{5x5}$ but if it does not fit then try scheduling $Mosaic_{3x5}$, and eventually try $Mosaic_{1x5}$ if the other two fail to schedule. The challenge for the scheduler is that getting a preferred switch case is not deemed worth forcing out another mandatory activity from the schedule. Because the normal approach to handling such interactions is to search, this introduces complications into the scheduling algorithms but these are the subject of a different paper.

The charter of the scheduler is to produce a grounded time schedule that satisfies all of the above constraints.

We also make the following assumptions:

1. There exists a set of activity scheduling priorities that would allow all mandatory activities to be scheduled by the scheduler [2].
2. The prior schedule is executed while the scheduler is running (Chi et al. 2018).
3. Activities do not fail.
4. No preemption (activities are only preempted as a major failure case for M2020).
5. The onboard scheduler is a "black box" - the onboard scheduler algorithm (Algorithm 1) is fixed.

The goal of the scheduler is to schedule all mandatory activities and better switch cases [3] while respecting individual and plan-wide constraints.

The goal of the priority setting algorithm is to derive a set of priorities that will best allow the scheduler to achieve that goal. Not only that, but we must derive that set of priorities in the shortest amount of time possible in order to satisfy daily mission time constraints.

Scheduler Design

Algorithm 1 Onboard Scheduler

Input:
$A\langle p, R, e, dv, \Gamma, T, D\rangle$: List of activities with their individual constraints
C: Constraints for the whole plan (e.g. available cumulative resources)
S: Current state of the spacecraft (state of charge, data volume, activity status)
Output:
U: Resulting schedule

Sort(A) ▷ Sorted by highest to lowest priority.
for each $a \in A$ **do**
  $P \leftarrow \emptyset$ ▷ Some activities may require automatically generated preheats
  $M \leftarrow \emptyset$ ▷ Some activities may require automatically generated maintenances
  $I \leftarrow [a.earliest\_start\_time, a.latest\_start\_time] \bigcap find\_valid\_intervals(a.unit\_resources) \bigcap find\_valid\_intervals(a.activity\_status) \bigcap find\_valid\_intervals(a.data\_volume)$
  if $requires\_preheat(a)$ **then**
    $P \leftarrow generate\_preheat\_activities(a)$
    $M \leftarrow generate\_maintenance\_activities(a)$
  end if
  $I \leftarrow I \bigcap find\_valid\_intervals(a.energy, P, M) \bigcap find\_valid\_intervals(a.peak\_power, P, M)$
  $awake \leftarrow generate\_awake\_activity(a, I)$
  if $I \neq \emptyset$ **then**
    $schedule\_activity(a, I)$
    $schedule\_activity(awake, I)$
    for each $p \in P$ **do**
      $schedule\_activity(p, I)$
    end for
    for each $m \in M$ **do**
      $schedule\_activity(m, I)$
    end for
  end if
end for

The Mars 2020 onboard scheduler (Algorithm 1) is a single shot, non-backtracking scheduler that schedules (consid-

[1] $A_j \rightarrow A_k$ means the scheduled end time of A_k must be before the scheduled start time of A_j.

[2] Since our algorithm includes an incomplete scheduler, our assumption of a valid set of inputs can only hold true for our particular scheduler

[3] See Evaluating a Schedule for more information

ers activities) priority first order and never removes or moves an activity after it is placed during a single scheduler run. It does not search except when considering valid intervals for a single activity placement and when scheduling sleep and preheats [4] (Rabideau and Benowitz 2017).

Due to the greedy, non-backtracking nature of the onboard scheduler, the order in which activities are scheduled can greatly impact the quality of the schedule.

Evaluating a Schedule

In order to evaluate the goodness of a particular priority assignment, we have developed a scoring method based on how many and what type of mandatory and switch group activities are able to be scheduled successfully by the scheduler. The score is such that the value of any single mandatory activity being scheduled is much greater than that of any combination of switch cases (at most one activity from each switch group can be scheduled). This ensures the following strict ordering:

$$V(m \in M) \gg \sum_{i=1}^{n_S} V(s \in S_i) \tag{2}$$

where $V(x)$ is the value of activity x being scheduled, M is the set of all mandatory activities, n_S is the number of switch groups, S_i is switch group i, and s is a switch case in switch group S_i.

Static Algorithms for Activity Priority Assignment

We have developed several static algorithms which set the priorities of activities based on various activity ordering criteria. These algorithms do not consider Monte Carlo simulations of plan execution where activities may end early or late while determining priorities, unlike our Priority Search approach. We will later compare these to our Priority Search approach to gain a better understanding of how well it performs. Activities which must begin at a particular time (e.g. data downlink) are always given the highest priority and thus are not affected by the static algorithms described.

The following four methods are used to initialize activity priorities:

- *Equal Priorities.* All activities have equal priorities.
- *Random Assignment.* Each activity is given a random priority.
- *Latest Start Time.* The activity priorities are ordered by the latest time they are allowed to start. The activity with the earliest such time has the highest priority.
- *Human Expert.* Each activity is assigned a priority based on the start time of the activity in a schedule constructed by a human expert. The activity with the earliest start time in this schedule has the highest priority.

The following two methods are applied to the priorities after they have been initialized in one of the four ways described above:

[4] Sleep and preheats are activities automatically generated and scheduled by the scheduler.

- *Dependencies.* $A \rightarrow B$ means that B must execute successfully before A can start. To generate a schedule that respects this,

$$A \rightarrow B \Rightarrow priority_A < priority_B \tag{3}$$

where higher priority means an activity is considered for scheduling earlier.

- *Tie Breaker.* If activities have the same priority assignment the activity with earliest latest allowed start time is of higher priority. If they also have the same latest allowed start time then the longer activity has the higher priority. If all of these attributes are equal then the higher priority activity is chosen lexicographically based on each activity's unique identifier.

Priority Search

In order to determine a set of priorities which will allow the scheduler to generate a schedule better than our static heuristics, we attempt to search the priority space in an approach similar to Squeaky Wheel Optimization (SWO) as described in Joslin and Clements 1999 (Joslin and Clements 1999). Squeaky Wheel Optimization usually involves a constructor, an analyzer, and a prioritizer. The constructor generates a schedule, the analyzer determines problem areas and assigns "blame" to certain elements in the schedule, and the prioritizer modifies the order in which the elements are considered. This process repeats until a satisfactory result is reached or allotted time runs out. However, our scheduling problem is intrinsically tied to execution and analyzing the initial schedule generated by itself is not satisfactory. Our approach (Figure 1) builds upon the usual SWO approach by incorporating a simulation of execution and Monte Carlo to build an execution sensitive result. We call our approach Priority Search as it searches the priority space using Monte Carlo simulation feedback to find a good set of priorities, unlike the static algorithms.

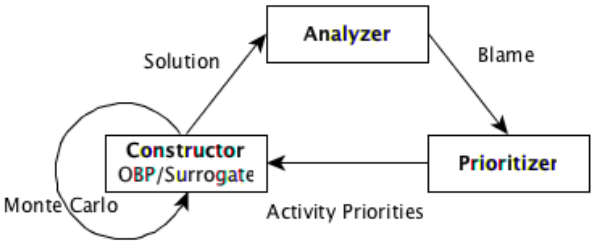


Figure 1: Squeaky Wheel accounting for Execution

Constructor

Typically, the constructor generates a schedule as the solution, which is then fed into the analyzer. However, our scheduling problem must be taken in context with execution. Activities may finish early or late which affect how many and which activities can be scheduled. In order to take this into account, we generate the final schedule of a

(lightweight) simulation of the entire plan execution. This is simulated by letting activities finish early or late by a variable amount based on a probabilistic model of plan execution [5]. However, the probabilistic model may promote misleading results if only sampled once. As a result, our constructor (Algorithm 2) runs a Monte Carlo and simulates multiple plan executions, passing on all of the executed plans as the solution to the analyzer.

Algorithm 2 Monte Carlo Constructor

Input:
$A\langle p, R, e, dv, \Gamma, T, D\rangle$: List of activities with their individual constraints
C: Constraints for the whole plan (e.g. available cumulative resources)
N: Number of runs in the Monte Carlo
Output:
S: List of all final schedules after simulating execution
1: $i \leftarrow 0$
2: **while** $i < N$ **do**
3: $schedule \leftarrow$ simulation(A, C)[6]
4: $S_i \leftarrow schedule$
5: $i \leftarrow i + 1$
6: **end while**

Priority Analyzer

The analyzer (Algorithm 3) takes the solution and assigns blame to problem areas. Since our objective is to schedule all mandatory activities and better switch cases, we blame all activities that are not scheduled. Since the solution is multiple schedules, there may be some Monte Carlo runs where the activities do not succeed or fail to be scheduled. For simplicity, we chose to blame any activity that was unscheduled in any of the schedules, but other approaches may assign blame according to how many times an activity was not scheduled.

Algorithm 3 Monte Carlo Analyzer

Input:
$A\langle p\rangle$: List of activities with priorities
S: List of all final schedules after simulating execution
Output:
U: List of all unscheduled activities
$score$: Score (objective function)
1: **for each** $S_i \in S$ **do**
2: $U \leftarrow U \bigcup \{\forall a \in A | a \notin S_i\}$
3: $score \leftarrow score +$ get_score(S_i)
4: **end for**

Constant Step Prioritizer

A simple way to re-prioritize is to increase the blamed (unscheduled) activities' priorities by a constant step size s.

Typically, activities have varying degrees of flexibility due to their constraints (resources, dependencies, time, etc.).

[5]See Empirical Results for how that probabilistic model was generated.

[6]The final schedule after simulating execution.

Algorithm 4 Constant Step Reprioritization

Input:
$A\langle p\rangle$: List of activities with priorities
U: List of all unscheduled activities (from analyzer)
$step$: Constant step size
Output:
A: Best relative ordering of activities found
1: **for each** $a \in U$ **do**
2: incrementRelativePriority(a, $step$, A)
3: **for each** $d \in a.dependents$ **do**
4: incrementRelativePriority(d, $step$, A)
5: **end for**
6: **for each** $sg \in a.switchGroup$ **do**
7: incrementRelativePriority(sg, $step$, A)
8: **end for**
9: **end for**

Higher priority activities can consume resources (unit resources, energy, and data volume) or change state in a way that prevents lower priority activities from scheduling such that their constraints are satisfied. Increasing the blamed activities' priorities allows them to schedule earlier (scheduling order) which means they have more "slack" to satisfy their constraints. The goal is that the algorithm will slowly promote less flexible activities to the top so that their constraints can be satisfied, and demoted activities are flexible enough to be scheduled in a more constrained plan.

When increasing the relative priorities of blamed activities, the existing relative priorities between certain groups of activities must remain enforced.

First, each switch group must maintain the relative priorities between each activity in the grouping. For each switch group, the activities $(s_1, \ldots, s_n)$ must be ordered such that those with higher resource consumption (time, energy, and data volume) have higher priorities as well.

Second, dependency relationships must be enforced such that (3) is held true.

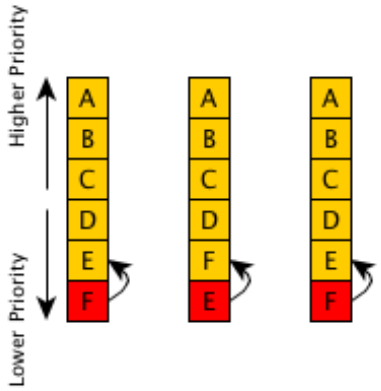


Figure 2: Cycle in the Constant Step approach. Red activities were unable to be scheduled and assigned blamed.

There is one main issue with the Constant Step approach

- it is extremely susceptible to cycles. One common cause for cycles is that a set of activities needs to be promoted beyond a particular activity together, but the constant step size prevents this from ever occurring. For example, in Figure 2 activity F is unschedulable and assigned blame. Its priority is increased, but this causes activity E to fail to schedule. Activity E is then promoted in the next iteration, causing F to fail to schedule and the process repeats. In reality, both E and F have to be promoted above D, but because the step size is constant, they will never achieve that and form a cycle. The situation where activities are unable to be promoted above an activity blocking it can be extended to any constant step size less than the maximum step size [7].

Stochastic Step Reprioritization

Algorithm 5 Stochastic Step Reprioritization

Input:
$A\langle p\rangle$: List of activities with priorities
U: List of all unscheduled activities (from analyzer)
Output:
A: Best relative ordering of activities found

```
step ← random(1, A.length)
for each a ∈ U do
    incrementRelativePriority(a, step, A)
    for each d ∈ a.dependents do
        incrementRelativePriority(d, step, A)
    end for
    for each sg ∈ a.switchGroup do
        incrementRelativePriority(sg, step, A)
    end for
end for
```

Injecting randomness to the step size allows the algorithm to become robust to cycles. In each iteration of the priority setting algorithm, a random step distance between 1 and N, where N is the number of activities in the plan, is assigned to all of the blamed activities. This lets the scheduler always have the possibility of being promoted above a resource constraining activity, while still allowing smaller step size priority permutations.

The main issue that lies with a random approach is that empirically [8] it finds the global maximum score slower than desired. This is further exacerbated by the fact that each iteration of our SWO cycle takes a non-negligible amount of time (a few seconds) due to the need to run a lightweight simulation and Monte Carlo.

Max Step Reprioritization

Stochastic Step Reprioritization (empirically) produced results slower than desired. Max Step Reprioritization seeks to solve both of those issues by always promoting blamed activities to have the highest scheduling priorities. The earlier an activity is considered for scheduling, the more flexibility that activity has to be scheduled. Therefore, if an activity is first to be considered for scheduling, but still cannot be successfully scheduled, there is no other scheduling priority that would allow the activity to be scheduled. Knowing this, by promoting blamed activities to have the highest scheduling priorities we can attempt to avoid iterations that fail to schedule the same blamed activities, thereby speeding up the overall algorithm.

[7] See section Max Step Reprioritization

[8] More information can be found in Empirical Evaluation.

Algorithm 6 Max Step Reprioritization

Input:
$A\langle p\rangle$: List of activities with priorities
U: List of all unscheduled activities (from analyzer)
Output:
A: Best relative ordering of activities found

```
for each a ∈ U do
    step ← A.length
    incrementRelativePriority(a, step, A)
    for each d ∈ a.dependents do
        incrementRelativePriority(d, step, A)
    end for
    for each sg ∈ a.switchGroup do
        incrementRelativePriority(sg, step, A)
    end for
end for
```

Since the blamed activities will have the highest scheduling priorities, cycles such as those seen in Figure 2 can be avoided. However, Max Step Reprioritization doesn't prevent cycles entirely and they still pose an issue when encountered.

Empirical Evaluation

In order to evaluate how well our Priority Search algorithm is able to generate a priority assignment which results in an optimal schedule, we have applied the algorithm to various sets of inputs comprised of activities with their constraints and priorities and compared against various static algorithms. The inputs are derived from *sol types*. *Sol types* are currently the best available data on expected Mars 2020 rover operations (Jet Propulsion Laboratory 2017a). In order to construct a schedule and simulate plan execution, we use the *M2020 surrogate scheduler* - an implementation of the same algorithm as the M2020 onboard scheduler (Rabideau and Benowitz 2017), but implemented for a Linux workstation environment. As such, it is expected to produce the same schedules as the operational scheduler but runs much faster in a workstation environment. The surrogate scheduler is expected to assist in validating the flight scheduler implementation and also in ground operations for the mission (Chi et al. 2018).

Each input file contains approximately 40 activities. We use a probabilistic execution model based on operations data from the Mars Science Laboratory Mission (Jet Propulsion Laboratory 2017b; Gaines et al. 2016a; 2016b) in order to simulate activities completing early by a reasonable amount. In our model to determine activity execution durations, each of the actual execution durations provided in MSL data is first divided by the corresponding predicted execution dura-

tion. Then, we use a linear regression on the scaled values to obtain a mean and standard deviation presuming the ratio of predicted to actual execution times is normally distributed. The value representing the actual execution duration on the regression line for the given conservative duration is used as the mean. A scaled prediction of the actual duration is generated from a a normal distribution using the derived mean and standard deviation. Finally, this value is scaled back by multiplying by the given conservative duration. Note that we do not explicitly change other activity resources such as energy and data volume since they are generally modeled as rates and changing activity durations implicitly changes energy and data volume as well.

Using each of the sol types, we create variants by adding two switch groups to a set of inputs. Each switch group contains three switch cases where the switch cases differ in duration in a manner similar to the one described in (1). Each of the two switch groups are as follows:

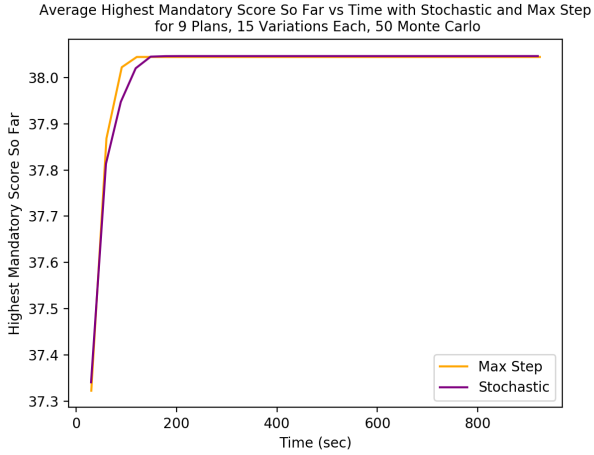
$$SwitchGroup = \begin{cases} Activity_{original} & \text{Duration=}x \text{ sec} \\ Activity_{2x} & \text{Duration=}2x \text{ sec} \\ Activity_{4x} & \text{Duration=}4x \text{ sec} \end{cases} \tag{4}$$

Due to the inequality in (2), a successfully scheduled mandatory activity is of much higher value than a successfully scheduled longer switch case. Therefore, the mandatory activity score is weighted at a much larger value then the switch group score. Each mandatory activity that is successfully scheduled is given one point which contributes to the mandatory score. If a switch case with a duration that is 2 times that of the original activity is able to be scheduled, then it contributes $1/5$ to the switch group score. If a switch case that is 4 times the original duration is able to be scheduled, then it contributes $2/5$ to the switch group. Since there are two switch groups in each variant, the maximum switch group score for a variant is $2 * (2/5) = 4/5$. In the following empirical results, we average the mandatory and switch groups scores over all Monte Carlo runs of execution.

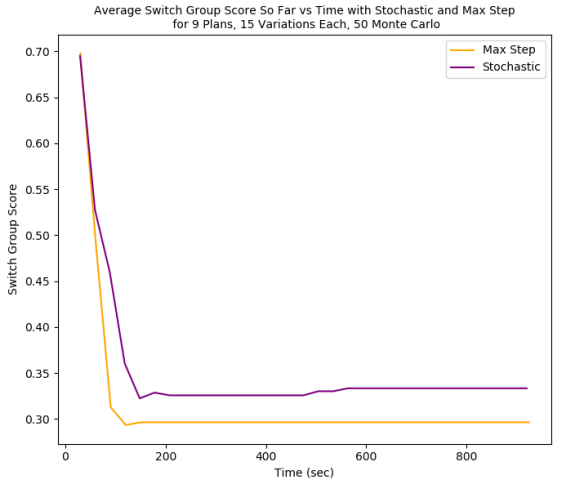
Also, in each of our variants we set the preferred schedule time of each activity to the earliest time the activity is allowed to start.

We first compare the different approaches to implementing Priority Search to understand which performs better.

The highest score so far is a combination of the mandatory score and the switch group score where the mandatory score is weighted at a much higher value than the switch group score. In Figure 3 we plot how the mandatory and switch case components of the highest score achieved up to the current time change over time using both the Stochastic method and the Max Step method. We do not consider the Constant Step method since it is so highly susceptible to cycles. For both methods, as the score for mandatory activities increases, the score for switch groups largely decreases until the highest mandatory score is reached. This is a reasonable outcome because as more mandatory activities are scheduled, the schedule likely becomes more constricted, thus making it more difficult to schedule longer switch cases. Since the mandatory score contributes much more to the total score than the switch group score and the mandatory sore



(a) Mandatory score component of highest score so far vs Time averaged across sol type variants using both priority search methods.



(b) Switch group score component of highest score so far vs Time averaged across sol type variants using both priority search methods.

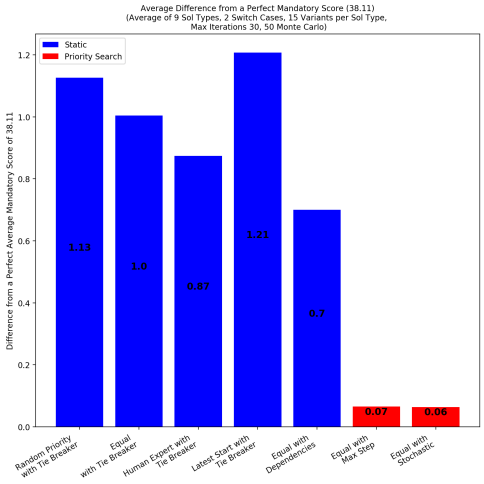
.

Figure 3: Plot of the highest score so far separated by mandatory score (3a) and switch group score (3b) over time using the Stochastic Step method and the Max Step method averaged over 9 sol types, each with 10 variants each containing 2 switch groups. Each iteration of Priority Search was run with 10 Monte Carlo runs and with 30 iterations of Priority Search alloted for each run of the algorithm.

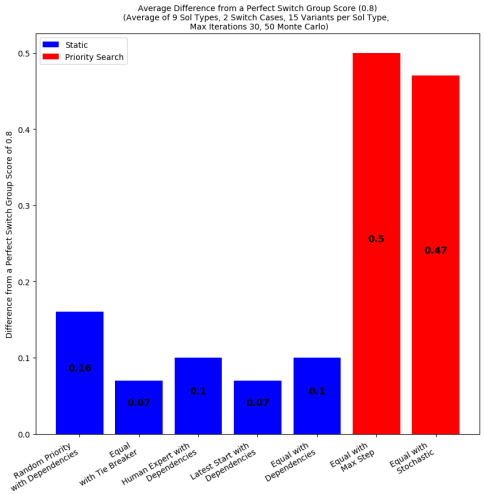
is increasing in both figures, the total highest score so far is always increasing over time, as it should be.

Figure 3a shows that Stochastic Step reaches its highest mandatory score that is ever achieved over the time span of approximately 920 seconds (30 iterations of the priority search algorithm) in 207.58 seconds. The highest mandatory score achieved at this time and onwards is 38.047. The high-

est mandatory score using the Max Step method is reached at 120.59 seconds and has a value of 38.044. Figure 3b shows that the highest switch group score after the point at which the highest mandatory score is reached is 1.67 at 568.16 seconds using the Stochastic method and 1.48 at 150.87 seconds using the Max Step method. Therefore, we conclude that using the stochastic method results in a marginally higher total highest score but it takes less time to reach the highest score using the Max Step method.



(a) Difference from perfect mandatory score averaged across sol type variants for various scheduling methods.



(b) Difference from perfect switch group score averaged across sol type variants for various scheduling methods.

Figure 4: The difference from a perfect mandatory score of 38.11 and perfect switch group score of 1.0 using various scheduling methods is averaged over 9 sol types where 15 variants are derived from each sol type and each variant contains 2 switch cases. Each iteration of the Priority Search algorithm is run with 50 Monte Carlo runs of execution

Figure 4 shows the results of comparisons between Priority Search and other static priority setting algorithms. Since the scheduling of mandatory activities and switch groups are not weighted equally, we have constructed two separate plots to show the results for each. Both methods of Priority Search, in red, result in fewer unscheduled mandatory activities and consequently a lower difference from the perfect mandatory score. This implies they set the priorities such that more mandatory activities are able to be scheduled over multiple Monte Carlo runs compared to how the static algorithms set the activity priorities. As shown in 4b, they result in a higher number of unscheduled switch cases, likely because if more mandatory activities were scheduled it becomes more difficult to schedule longer switch cases. Due to the strict inequality described in (2), even though fewer longer switch cases are scheduled, the total scheduling score is still higher when using Priority Search. Thus, we conclude that both Priority Search methods outperform the static algorithms. Among the static algorithms, running the Dependencies algorithm with Tie Breaker on equal priorities performs the best as it results in the highest mandatory score while running Tie Breaker after setting the priorities based on the latest start time performs the worst.

Related Work

Our Priority Search approach is inspired by Squeaky Wheel Optimization (SWO). Typically, SWO uses a constructor and analyzer, and prioritizer for the next iteration of schedule generation (Joslin and Clements 1999). Priority Search differs in that the intent is not to generate a good schedule but rather to set priorities that perform well in execution and rescheduling. Therefore the Priority Search constructor must use the scheduler through multiple runs of execution (where each run of execution incurs multiple scheduler invocations) to assess priority assignment performance.

Generating schedules that are robust to execution run time variations (Leon, Wu, and Storer 1994) is a mature area of work. However, the topic usually revolves around developing a scheduler that can generate robust schedules. In our case, the scheduler is a) a fixed ”black box” that we have no control over and b) robust to execution run time variations mainly through rescheduling (Chi et al. 2018). As a result, rather than developing a scheduler itself, we’re developing a methodology that is able to generate a set of priorities for a fixed scheduler that enables it to be robust to rescheduling due to runtime variations.

Other approaches (Drummond, Bresina, and Swanson 1994; Washington, Golden, and Bresina 2000) use branching to increase robustness - these differ from our work that adjusts priorities and allows rescheduling.

A number of other spacecraft (Muscettola et al. 1998; Pell et al. 1997; Chien et al. 2005; 2016) and rover (Woods et al. 2009; Gregory et al. 2002) autonomy systems have included planning, but these differ in that we are deriving control information specific to scheduling for a limited context - e.g. one rover sol. temporal schedule.

Discussion and Future Work

While we have focused on the impact of activity priority on the scheduler (and hence rescheduling during execution), there is often an execution system that may also have some flexibility to add robustness to the overall system (Chi et al. 2018). For the empirical evaluation described above, we ran without such an execution system. In the future, we could consider the execution system in the schedule and Monte Carlo analysis and potentially derive information usable by the execution system (e.g. allow an activity to run late but only until time T). This paper describes initial work to determine priorities for scheduler activity consideration ordering to optimize scheduler execution results for an embedded scheduler. However, this work is still preliminary with many other ideas to be explored as described below.

First, more sophisticated critique/blame assignment methods should be explored. Currently, priorities of activities that are not executed are modified, but more sophisticated analysis of scheduler runs could provide greater insight into how the priorities should be modified. Prior work in Process Chronologies (Biefeld and Cooper 1991) has been used to focus scheduling tactics by finding regions where time constraints or high demand for some resource results in conflict. By evaluating which periods of time or what resources are over-subscribed using Capacity/Over-Subscription Analysis, we can pinpoint which activities are more tightly constrained and increase their priorities. Prior work in Oversubscribed Scheduling Problems (Kramer and Smith 2006) show that scheduling according to maximum-availability (least subscribed) allows a suitable schedule to be generated. Similar analysis could be used to determine which activities to assign blame to and by how much to promote the blamed activities. We can also consider precedence constraints when deciding by how much to promote activity priorities. For every blamed activity, there is likely a scheduled activity that is using resources needed by the blamed activity. Precedence constraints could help discern which activity is using those resources. The blamed activity could then be promoted only as much as is necessary in order to be scheduled before the offending activity.

We can also implement several methods to help us explore different search spaces. Priority Search only adjusts priorities to improve execution and rescheduling performance. We could also add new activity precedence constraints (e.g. A must end before B starts) or enforce partitions in the schedule (e.g. all of these activities must be scheduled to end prior to 11 am). These types of constraints could drive the scheduler towards subsets of the schedule search space. Randomized restart can allow our priority search algorithm to better explore the global space rather than searching locally. Another alternative would be to keep a list of promising schedule priority assignments and backtrack to those randomly, allowing us to better explore the search space.

We can also make improvements to our Monte Carlo method and use the resulting simulations for further analysis of the scheduler. In order to build a model of run time variations that is not overly skewed, we use Monte Carlo to repeatedly sample a variety of execution run time results. Standard Monte Carlo simulations tend to focus most runs on the nominal cases, but a more effective methodology samples edge cases but weighs the cases by their likelihood to increase coverage of the variability in the space (in this case variable activity execution times). The Monte Carlo of execution run time variations can provide valuable information for why activities fail to schedule, what input plans are best suited for the current scheduler design, and how the current input could end up executing. We are working on visualizing this information to better inform those working with the scheduler.

Currently, we only test with mandatory activities. In the future, we will extend our approach to include optional activities, which will add further complexity to the algorithm and analysis. Optional activities are lower priority activities what are nice to have scheduled, but not necessary. They are generally only able to be scheduled if mandatory activities end early or consume less resources than expected. We also plan to use an activity's actual scheduled preferred time while testing.

Cycles pose an issue to both Constant Step Reprioritization and Max Step Reprioritization. Better cycle detection would allow us to not only overcome the issues presented, but also provide additional information on how to permute the priority set for the next iteration. For example, cycle detection could allow us to not only detect the cycle in Figure 2, but know that both E and F should be incremented together.

While we have established a few methods to improve the prioritizer and decide on the next permutation of activity priorities, we have utilized the same objective function to determine the success of our algorithm. However, our objective function is simple and coarse; oftentimes, the same score will appear repeatedly in multiple consecutive. As a result, the algorithm often travels swaths of plateaus before sharply improving. This choppiness is suboptimal for Squeaky Wheel Optimization and gradient descent problems in general. Some potential additions to the objective function could be how much energy is leftover in the plan or how close an activity is to their preferred scheduling time. Evaluating a more precise objective function can reduce choppiness and better steer the algorithm towards a more optimal solution.

Conclusion

We have presented a study of methods to assign activity priorities to control a limited, embedded scheduler to optimize rescheduling for a specific problem. We first define a set of static methods that assign activity priorities based on heuristics and schedule dependencies. We then describe how these priorities can be further adjusted based on feedback from simulated execution and rescheduling using Monte Carlo methods to perform Priority Search. We present an empirical evaluation of several static and priority search methods using best available planetary rover operations data. This empirical evaluation shows that Priority Search outperforms static methods including human expert derived priorities. Finally we describe a number of promising areas for future improvements to our algorithms.

Acknowledgments

This work was performed at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

Biefeld, E., and Cooper, L. 1991. Bottleneck identification using process chronologies. In *IJCAI*, 218–224.

Chi, W.; Chien, S.; Agrawal, J.; Rabideau, G.; Benowitz, E.; Gaines, D.; Fosse, E.; Kuhn, S.; and Biehl, J. 2018. Embedding a scheduler in execution for a planetary rover. In *ICAPS*.

Chien, S. A.; Sherwood, R.; Tran, D.; Cichy, B.; Rabideau, G.; Castano, R.; Davies, A.; Mandl, D.; Trout, B.; Shulman, S.; et al. 2005. Using autonomy flight software to improve science return on earth observing one. *Journal of Aerospace Computing Information and Communication* 2(4):196–216.

Chien, S.; Doubleday, J.; Thompson, D. R.; Wagstaff, K. L.; Bellardo, J.; Francis, C.; Baumgarten, E.; Williams, A.; Yee, E.; Stanton, E.; et al. 2016. Onboard autonomy on the intelligent payload experiment cubesat mission. *Journal of Aerospace Information Systems*.

Drummond, M.; Bresina, J.; and Swanson, K. 1994. Just-in-case scheduling. In *AAAI*, volume 94, 1098–1104.

Gaines, D.; Anderson, R.; Doran, G.; Huffman, W.; Justice, H.; Mackey, R.; Rabideau, G.; Vasavada, A.; Verma, V.; Estlin, T.; et al. 2016a. Productivity challenges for mars rover operations. In *Proceedings of 4th Workshop on Planning and Robotics (PlanRob)*, 115–125. London, UK.

Gaines, D.; Doran, G.; Justice, H.; Rabideau, G.; Schaffer, S.; Verma, V.; Wagstaff, K.; Vasavada, A.; Huffman, W.; Anderson, R.; et al. 2016b. Productivity challenges for mars rover operations: A case study of mars science laboratory operations. Technical report, Technical Report D-97908, Jet Propulsion Laboratory.

Gregory, N. M.; Dorais, G. A.; Fry, C.; Levinson, R.; and Plaunt, C. 2002. Idea: Planning at the core of autonomous reactive agents. In *Proceedings of the 3rd International Workshop on Planning and Scheduling for Space*. Citeseer.

Jet Propulsion Laboratory. 2017a. Mars 2020 rover mission https://mars.nasa.gov/mars2020/ retrieved 2017-11-13.

Jet Propulsion Laboratory. 2017b. Mars science laboratory mission https://mars.nasa.gov/msl/ 2017-11-13.

Joslin, D. E., and Clements, D. P. 1999. Squeaky wheel optimization. *Journal of Artificial Intelligence Research* 10:353–373.

Kramer, L. A., and Smith, S. F. 2006. Resource contention metrics for oversubscribed scheduling problems. In *ICAPS*, 406–409.

Leon, V. J.; Wu, S. D.; and Storer, R. H. 1994. Robustness measures and robust scheduling for job shops. *IIE transactions* 26(5):32–43.

Muscettola, N.; Nayak, P. P.; Pell, B.; and Williams, B. C. 1998. Remote agent: To boldly go where no ai system has gone before. *Artificial Intelligence* 103(1-2):5–47.

Pell, B.; Gat, E.; Keesing, R.; Muscettola, N.; and Smith, B. 1997. Robust periodic planning and execution for autonomous spacecraft. In *International Joint Conference on Artificial Intelligence*, 1234–1239.

Rabideau, G., and Benowitz, E. 2017. Prototyping an onboard scheduler for the mars 2020 rover. In *International Workshop on Planning and Scheduling for Space*.

Washington, R.; Golden, K.; and Bresina, J. 2000. Plan execution, monitoring, and adaptation for planetary rovers. *Electron. Trans. Artif. Intell.*

Woods, M.; Shaw, A.; Barnes, D.; Price, D.; Long, D.; and Pullan, D. 2009. Autonomous science for an exomars rover–like mission. *Journal of Field Robotics* 26(4):358–390.

Front Delineation and Tracking with Multiple Underwater Vehicles

Andrew Branch[1], Mar M. Flexas[2], Brian Claus[3], Andrew F. Thompson[2], Evan B. Clark[1], Yanwu Zhang[4], James C. Kinsey[3], Steve Chien[1], David M. Fratantoni[5], Brett Hobson[4], Brian Kieft[4], Francisco P. Chavez[4]

[1]Jet Propulsion Laboratory, California Institute of Technology
[2]California Institute of Technology
[3]Woods Hole Oceanographic Institution
[4]Monterey Bay Aquarium Research Institute
[5]Remote Sensing Solutions
Correspondence Author: andrew.branch@jpl.nasa.gov

Abstract

This work describes a method for detecting and tracking ocean fronts using multiple autonomous underwater vehicles. Multiple vehicles — equally-spaced along the expected frontal boundary — complete near parallel transects orthogonal to the front. Lateral gradients are used to determine the location of the front crossing from each individual vehicle transect by detecting a change in the observed water property. Adaptive control of the vehicles ensure they remain perpendicular to the estimated front boundary as it evolves over time. This method was demonstrated in and around Monterey Bay, California in May of 2017. We compare the front detection method to previously used methods. We introduce a metric in order to evaluate the adaptive control techniques presented. We show the capability of this method for repeated sampling across a dynamic two-dimensional ocean front using short-range Iver AUVs. This method extends to tracking gradients of different properties using a variety of vehicles.

Introduction

Space-based remote sensing can provide extensive information about ocean dynamics. However, remote sensing information is generally limited to measuring the ocean surface. To probe the ocean interior efficiently requires marine vehicles such as autonomous underwater vehicles (AUVs), gliders, profiling buoys, surface vehicles, and ships sampling in situ. Unfortunately, building, deploying and operating these in situ marine robotic explorers is expensive. As a result, any actual study involves a limited number of marine vehicles, especially when compared to the vast expanse of the ocean. Determining where to deploy and operate marine assets is a challenging problem given the 4D spatiotemporal variations in oceanographic phenomena.

The use of autonomous marine vehicles will increase as the size of ocean observing systems expand in order to study the impact of the oceans on Earth's climate and ecosystems. The day-to-day operations of these systems will become increasingly difficult if human intervention is required. In order to enable large observing systems to operate, techniques for autonomous control of assets based on science goals and data sources such as in situ measurements, remote-sensing, and model-derived data need to be developed. The Keck Institute for Space Studies (KISS) Satellites to Seafloor project works towards this goal of fully autonomous sampling [Thompson et al., 2017]. Previous ocean observing systems have relied on substantial human intervention or non-adaptive sampling strategies, including the Autonomous Ocean Sampling Networks (AOSN) [Curtin and Bellingham, 2009; Curtin et al., 1993; Haley et al., 2009; Leonard et al., 2007; Ramp et al., 2009] and the Adaptive Sampling and Prediction (ASAP) [Leonard et al., 2010] projects.

One approach is to deploy in situ assets to study coherent scientific features such as fronts, eddies, upwelling events, and harmful algal blooms. A typical strategy would be to deploy marine assets to measure transects across the feature of interest at a scale that covers the feature, as well as a baseline signal around the feature. However, asset capabilities (e.g. mobility, endurance) and prevailing ocean currents may render these science goals unachievable. Our project targets automatic generation of coordinated mission plans for teams of assets to follow these science derived observation policies (e.g. the use of multiple vehicles to perform transects orthogonal to a front). This paper specifically describes an approach using multiple vehicles to make a linear estimation of an ocean front's geometry and to continuously direct a team of marine robotic vehicles to perform orthogonal transects with the midpoint of the transect roughly centered on the target front. We describe both the general approach for front-crossing detection, front-geometry estimation, and multi-asset control as well as results from deployment and testing of this approach using short-range Iver Autonomous Underwater Vehicles in Monterey Bay in late spring 2017. The full results from the deployment, including the use of underwater gliders and Long-Range AUVs, are presented in Branch et al. [2018]. This paper focuses on the results using the Iver AUVs. This deployment was the result of a team effort between the KISS project members and the MBARI Spring 2017 CANON participants [Monterey Bay Aquarium Research Institute, 2017]. The method presented here represents significant

steps towards the fully-autonomous adaptive sampling framework as envisioned in Thompson et al. [2017].

The remainder of this paper is organized as follows. First, we provide science context to the target problem of front tracking. Second, we describe the front-crossing detection method. Third we describe the front-geometry estimation and tracking algorithm used to estimate a linear front across multiple vehicles and produce resultant vehicle transects. Fourth, we describe the experimental setup with the Iver AUVs. Fifth, we describe the results from the field deployments. Finally, we discuss related and future work and summarize the results of the experiment.

Science Context

Coherent fronts are ubiquitous features of the ocean circulation. Fronts, defined as regions of enhanced gradients in water mass or tracer properties, can occur across different scales spanning many hundreds of kilometers, such as the strong western boundary currents (e.g. the Gulf Stream), to smaller-scale filamentary features which are often associated with the fringes of coherent mesoscale eddies, but may cascade down to the meter scale [D'Asaro et al., 2017]. Due to the earth's rotation, lateral gradients in density at ocean fronts can generate strong (and strongly-sheared) along-front velocities. These velocities can, in various scenarios, act to both enhance the front by suppressing mixing or enhance mixing due to the generation of flow instabilities [Bower, Rossby, and Lillibridge, 1985]. Fronts may also be regions of intense vertical velocities and vertical fluxes. This may occur either because density surfaces tilt across strong fronts, leading to strong vertical, but still largely along-isopycnal advection. Alternatively, at sharp fronts, relative vorticity may be enhanced such that the Rossby number, defined as the ratio of the vertical relative vorticity $\zeta = \partial v/\partial x - \partial u/\partial y$ to the Coriolis frequency f, becomes comparable to or greater than 1. In this regime, the effects of rotation that constrain the velocity field to be largely horizontal begin to break down and vertical velocities can become enhanced. This dynamical regime is known as the submesocale [McWilliams, 2016; Thomas, Tandon, and Mahadevan, 2008] and can generate vertical velocities on the order of hundreds of meters per day over most of the ocean [Su et al., 2018].

This enhancement of vertical velocities at the submesoscale has important implications for the coupling between the physical circulation and ocean biogeochemistry. Primary production in the ocean is characterized by a "patchiness" implying a large degree of spatial and temporal intermittency [Martin et al., 2002]. Ocean fronts have been identified as locations where primary production may be transiently enhanced, especially in oligotrophic waters due to the injection of nutrient-rich waters to the surface ocean [Brannigan, 2016; Lévy, Klein, and Treguier, 2001; Mahadevan, 2016]. More recent work has shown that frontal instabilities can rapidly shoal the mixed layer and lead to phytoplankton blooms due to a relaxation of light limitations [Mahadevan et al., 2012; Taylor and Ferrari, 2011]. These latter studies imply that ocean fronts can also regulate carbon cycling in subpolar latitudes. Ocean fronts have also been shown to have a large impact on large marine ecosystem [Belkin, Cornillon, and Sherman, 2009].

Therefore, for both physical and biogeochemical reasons, ocean fronts tend to be hotspots of turbulent mixing, ventilation (the transfer of near-surface water properties into the ocean interior) and subduction. Furthermore, the implication is that a significant portion of the exchange between the near-surface ocean and the ocean interior occurs over a relatively small fraction of the surface ocean. Thus there is a need to dedicate greater resources to the study of these frontal features, both to improve our mechanistic understanding of how these fronts develop, evolve and impact transport properties, but also so that they can be effectively represented in data-assimilating numerical simulations of the ocean circulation.

Front-Crossing Detection

Lateral Gradient Front-Crossing Detection

The KISS team developed an algorithm to identify a subsurface oceanic fronts based on lateral gradients of a given hydrographic property. This could be temperature, buoyancy or density (if salinity data is available), or any available biogeochemical property such as dissolved oxygen or chlorophyll.

When in situ data is received in near real time, the algorithm grids the field, smooths it by applying a simple linear weighted average of immediate neighboring measured data points, and calculates the lateral gradients (Figure 1). Smoothing parameters must be selected before using this algorithm in a near real time application. The algorithm uses temporal gradients, and assumes that time can be linearly related to distance. The algorithm then calculates the lateral gradients along the transect within the layer of interest (defined beforehand by the user) as well as the mean value, and the standard deviation. The user also defines beforehand the number of standard deviations used to declare a front-crossing detection. All points above this threshold are considered potential front crossings (Figure 2). To qualify for a frontal crossing, it is required that the threshold is crossed twice (once entering and once leaving the high gradient region). Half-crossings do not qualify. The width of the front is used to choose the front crossing of interest if more than one is present. The front location, width, and time of crossing is then output for later use in vehicle tasking. An example is shown in Figure 1 and Figure 2. Time, as apposed to distance, is plotted on the x-axis as that is what the algorithm uses. Using real time data from May 4, 2017 (Figure 1d) the algorithm detects five narrow subsurface fronts from 10 to 15 m deep (Figure 2a), and selects the widest front (Figure 2d).

The front-crossing detector can be customized, for ex-

ample, to select only positive or negative frontal crossings. This could be useful in the event of targeting a cold or warm eddy. It can also be modified to select, instead of the widest front, the front corresponding to the maximum lateral gradient, if desired. The capability of selecting the depths over which the lateral gradients were to be evaluated allows the user to target surface fronts, or instead, focus only on deeper fronts.

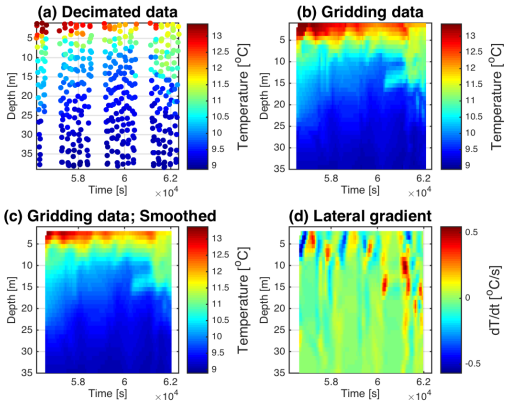


Figure 1: Lateral gradient front-crossing detector. For this example we use data obtained on May 4, 2017 from Iver 136 (segment 000). Real-time in situ temperature data (shown in scatter plot in panel a) is gridded (panel b) and smoothed (panel c). Then, lateral gradients are calculated (panel d). When used in real time, the algorithm uses temporal gradients, and assumes that time can be linearly related to distance.

Comparison to previous methods

Next we briefly compare the front-crossing detection technique presented above to a previous upwelling front detection technique developed by Monterey Bay Aquarium Research Institute (MBARI) [Zhang et al., 2012a,b, 2013]. This previous method is based on the vertical temperature structure measured on the AUV's sawtooth (i.e., yo-yo) trajectory. In stratified water, the vertical temperature difference is large: warm at surface and cold at depth. The upwelling process breaks down stratification and makes water properties more vertically homogeneous. Consequently, the vertical temperature difference between shallow and deep depths is smaller in upwelling regions. To enable an AUV to autonomously differentiate between upwelling and stratified water columns, Zhang et al. used a classification metric — the vertical temperature homogeneity index (VTHI) [Zhang et al., 2012b]

For this comparison, we use data obtained on May 1, 2017. Vertical sections in Figure 3 show the presence of a front at longitude ∼122.25°W. The front separates warm, fresh water to the west, from cold, salty water to the east (Figure3a-d). The maximum lateral gradients of these properties are clearly observed at longitude

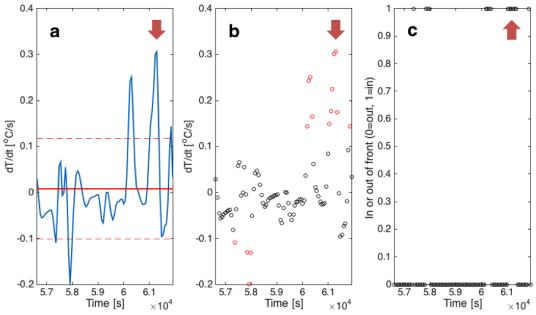


Figure 2: (Continues from Figure 1) The algorithm calculates the mean value of the lateral gradients over the layer of interest. In this example, we use data from 10m to 15m. The algorithm calculates the mean value (bold red line in panel a) and the n-standard deviation (in this case, n=1.2; red broken lines in panel a). All points above the n-value standard deviation are considered potential fronts (red circles in panel b). A boolean is used to isolate the front crossings (panel c). The width of the front is used to choose the front crossing when more than one front is present. The crossing chosen by the algorithm is marked with a red arrow.

∼122.25°W (Figure3e-i). We apply the two methods described above to the upper 30 m of the water column.

The VTHI method detects a decrease of VTHI value (note that a lower VTHI value means the observed water column is more homogeneous vertically) between ∼122.25-122.28°W, which corresponds to the maximum lateral gradient of buoyancy (Figure 3h). If we calculate the lateral gradient of VTHI we find agreement with the maximum lateral gradient of buoyancy (Figure 3i). Small differences are attributed to the role of salinity in the buoyancy values, which is not accounted for in VTHI.

Although both techniques give basically the same result, VTHI only captures upwelling fronts and so is specifically designed with Monterey Bay hydrography/circulation in mind. Our algorithm would be more general for detecting fronts throughout the ocean. We acknowledge that the need for interpolation in the lateral gradient method presented in this paper may pose difficulties for the implementation of this method onboard underwater vehicles. In the future, an onboard method of calculating lateral gradients without interpolation would be required.

Autonomous Control of Underwater Vehicles for Front Tracking

A technique was developed to control a group of vehicle to repeatedly sample across a dynamic ocean as it evolves over time. Vehicles must be able to modify their transects in order to adapt to the changing ocean conditions. The control algorithm (Algorithm

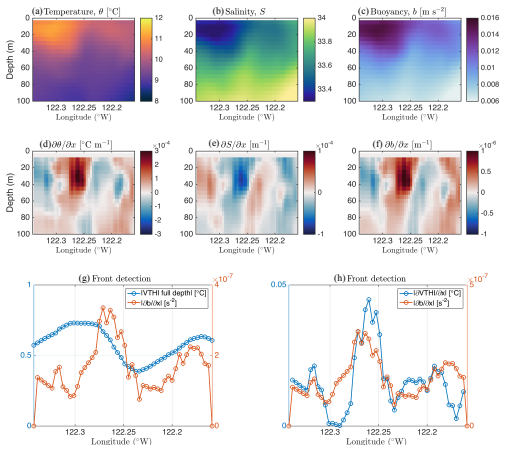


Figure 3: Comparison between lateral gradient detection method and vertical temperature homogeneity index (VTHI). (a-c) Vertical sections of (a) Temperature, θ, (b) Salinity, S, and (c) Buoyancy, b. (d-f) Lateral gradients of θ, S and b. (g) Front detection using absolute values of VTHI (blue) and lateral gradient of b (red). (h) Front detection using lateral gradients of VTHI (blue) and lateral gradient of b (red).

1) operates as follows. When first deployed, an initial estimated front location and orientation is manually provided based on available data from other assets. The vehicles are equally spaced along this estimated front. Each vehicle is commanded on an initial transect orthogonal to the provided estimated front. When the vehicle surfaces to plan, Algorithm 1 is executed. The vehicle location and the scientific data from the current transect are provided as $vehicle\_location$ and $transect\_data$ respectively. The vehicles location along the transect is calculated as $location_p$ by projecting the vehicles current location onto the commanded transect. If the vehicle has traveled a minimum distance along the commanded transect, specified by $transect\_dist_{min}$, then the front-crossing detection algorithm is run on the data from this transect. The resulting front-crossing is defined as $new\_front\_crossing$. If the vehicle is a specified distance past this new front detection, then the front is re-estimated using linear regression on front detections from all vehicles, otherwise the transect is continued. When re-estimating, only certain front detections from each vehicle are considered, specified by $valid\_front\_detections$. We used two methods when selecting the subset of detections used in the linear regression: a time based approach where detections from the last N hours were considered and a latest detection approach where only the last detection from each vehicle was considered. These two approaches are defined in the procedure $get\_estimation\_crossings$. The new $transect_p$ is calculated such that it is orthogonal to $estimated\_front$. The vehicle is then commanded on this new transect. In order to prevent the vehicle from leaving the study area, $transect\_dist_{max}$ is defined. If a transect has reached this length the front is re-estimated, a transect orthogonal to this is defined, and the vehicle is commanded on this new transect.

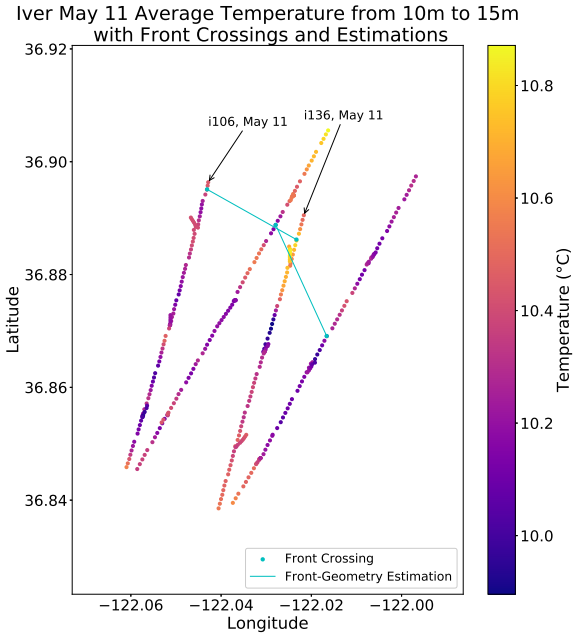


Figure 4: Iver transects on May 11 with temperature averaged from 10 meters to 15 meters plotted. Front crossings are shown as blue dots and estimated fronts are shown as blue lines. Each vehicle starting location is labeled with the vehicle name and the date. The second transect for each vehicle is orthogonal to the estimated front from the front crossings on the first transect.

Pilot Experiment

Experiment Site

The pilot experiment took place in Monterey Bay, California (36.80°N, 121.90°W) from May to June 2017.

The circulation in Monterey Bay is characterized by a persistent coastal upwelling, in response to prevalent northerly winds, which generates highly-productive cold coastal regions [Hickey, 1979; Lynn and Simpson, 1987]. Physical–biological coupling at the edges of mesoscale eddies, and turbidity plumes resulting from the interaction of the flow with topography, influence the phytoplankton ecology [Ryan, Chavez, and Bellingham, 2005]. Offshore (>150 km), the California Current (CC) flows southward with surface speeds of ∼0.25 m s^{-1} [Hickey, 1979; Lynn and Simpson, 1987]. Near the coast (<150 km), the surface flow varies seasonally, flowing northward in fall and winter [Reid and Schwartzlose, 1962], and receiving the name of the Inshore

Algorithm 1 Linear Front Delineation and Tracking

procedure VEHICLE_RETASKING($vehicle\_location$, $transect\_data$) ▷ Run this procedure when a vehicle surfaces to plan
  $location_p \leftarrow project\,(transect, vehicle\_location)$
  if $dist\left(transect\_start, location_p\right) >= transect\_dist_{min}$ **then**
    $new\_crossing \leftarrow detect\_crossings\,(transect\_data)$
    if $new\_crossing$ was detected **then**
      $crossings \leftarrow crossings \bigcup \{new\_crossing\}$
      $valid\_crossings \leftarrow get\_estimation\_crossings(crossings)$
      $estimated\_front \leftarrow linear\_regression\,(valid\_crossings)$
      $location_f \leftarrow project\,(transect, new\_front\_crossing)$
      if $dist\left(location_p, location_f\right) > \varepsilon_{past\_front}$ km **then**
        Calculate $transect_p$ **s.t.** $transect_p \perp estimated\_front$
        Command vehicle on $transect_p$
      else
        Continue on current transect
  else if $dist\left(transect\_start, location_p\right) <= transect\_dist_{max}$ **then**
    $valid\_crossings \leftarrow get\_estimation\_crossings(crossings)$
    $estimated\_front \leftarrow linear\_regression\,(valid\_crossings)$
    Calculate $transect_p$ **s.t.** $transect_p \perp estimated\_front$
    Command vehicle on $transect_p$

procedure GET_ESTIMATION_CROSSINGS($crossings$) ▷ First of two options for this procedure
  return Latest front crossing for each vehicle.

procedure GET_ESTIMATION_CROSSINGS($crossings$) ▷ Second of two options for this procedure
  return $\{crossing \in crossings \mid crossing.time > current\_time - \varepsilon_{time}\}$

Countercurrent (IC) [Lynn and Simpson, 1987]. The IC is intermittent in space and time. Below, the subsurface California Undercurrent (CU) flows northward. South of Monterey Bay, at Point Sur (36.31°N, 121.90°W), the CU separates from the coast due to topographic curvature and flow inertia [Molemaker, McWilliams, and Dewar, 2015] and forms mesoscale anticyclonic eddies whose inner edge reaches the shelf break off Monterey Bay.

In May 2017, an intensive upwelling plume spread southeastward across the mouth of Monterey Bay. A fleet of AUVs were deployed to detect and track the fronts between the upwelling plume and the stratified inner bay water. Over the shelf, KISS IVERs were set to detect lateral gradients of temperature from 10m to 15m. Over the slope, temperature in the upwelling water column was remarkably homogeneous in the vertical dimension. The operations region of the Iver AUVs are shown in Figure 5.

Iver AUVs

This work was demonstrated on two OceanServer Iver2 AUVs, shown in Figure 6. The method is extensible to other platforms and indeed other domains where the vehicles are able to at least intermittently transmit collected data and receive new instructions mid-deployment. Both of the vehicles were equipped with a hull-mounted Neil Brown conductivity/temperature sensor (Ocean Sensors Inc.) which served as the primary scientific payload for this work. Additionally, one of these vehicles, Iver-106, was an Ecomapper variant equipped with a SonTek Doppler velocity log (DVL), an Ocean-Server compass for attitude estimation, a WHOI micro-modem 2 and a depth sensor. The other Iver2 vehicle, Iver-136, was similarly equipped with the WHOI

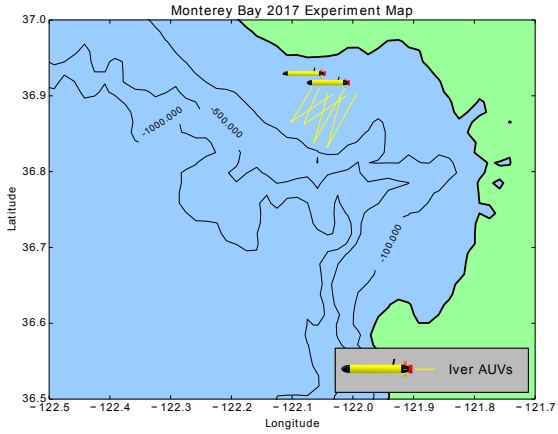


Figure 5: Map of the 2017 pilot experiment region near Monterey Bay, California. The operation region of the Iver AUVs are shown.

micro-modem 2, compass and depth sensor as well as a dual upward, downward facing 600 kHz RDI phased array DVL, a Microstrain 3DM-GX3-25 and an APS-1540 fluxgate magnetometer. The Iver2 AUVs have an approximate maximum horizontal velocity of 2 m s^{-1} and were operated at a speed of 1.5 m s^{-1} for these trials. These vehicles, are shown on board the R/V *Shana Rae* in Figure 6 during operations in August 2016.

Figure 6: OceanServer Technology, Inc. Iver2 AUVs onboard the R/V *Shana Rae*

Prior to shipping the vehicles were cross-calibrated against a Seabird SBE49 in a tank to get the relative sensor offsets. These offsets seemed to drift during shipping and the collocated measurements taken in the harbour and during deployment. In post-processing, Iver-106 was corrected for a salinity offset of 0.5180 practical salinity units.

Vehicle control

The Iver AUVs required some modifications to enable the transmission of data and receiving of new instructions during operations. Four communication modalities are available to the Iver, Iridium short burst data (SBD), Wi-Fi, 900 MHz RF, and acoustic modem. Sci-

entific data such as position, conductivity, temperature, and timestamps can be received and new commands can be sent over any of these four available communication links. Possible commands include stopping a mission, starting a mission already loaded on the vehicle, parking the vehicle and inserting segments of waypoints into the already running mission. Initially, it was planned to use the segment insertion to facilitate the retasking of the vehicles. While these commands were successfully received and interpreted by the vehicle, some unexplained behaviors while using this command precluded its ongoing use. As a temporary work around for the 2017 field trials in Monterey we used the outputs of the planning software to manually program a new mission which was then loaded onto the AUV over the RF link.

Results

We introduce a metric in order to quantify the performance of the front tracking control techniques presented here. For a given transect N, the front location, as predicted by transect $N-1$, and the front location observed on transect N are compared. As a baseline, the observed front location for transect N is also compared to the initial front-geometry estimation provided manually at the beginning of each experiment. More specifically, the metric is defined as follows. For a given transect N, define the initial front-geometry estimation manually provided at the beginning of the experiment as *initial estimation*, the front-geometry estimation used to create transect N as *predicted estimation* and the front-geometry estimation after transect N as *observed estimation*. Calculate the intersection point of transect N and the *predicted estimation* as well as the the intersection point of transect N and the *observed estimation*. The front tracking metric is defined as the distance between these two intersection points. The intersection point of transect N and *initial estimation* is also calculated. The baseline metric is defined as the distance between this intersection point and the intersection of transect N and the *observed estimation*. These two metrics are calculated for each transect. An example of the calculation for this metric can be seen in Figure 7.

The time between front crossings has an important role in the performance of this metric. Longer time between front crossings allows for a larger change in the ocean conditions. This time is a function of the speed of the vehicle and the length of a transect. The dynamism of the experiment region also affects this metric as this determines how much one might expect the front to evolve between two crossings. Due to this, direct comparisons of this metric between vehicles and operations areas are questionable, however, it can be used to assess the performance of the front tracking algorithm as well as indicate the suitability of a vehicle to a specific operating environment.

Iver AUV Results

Two Iver AUVs were operated on three days, 4 May, 9 May, and 11 May 2017. They are limited to single

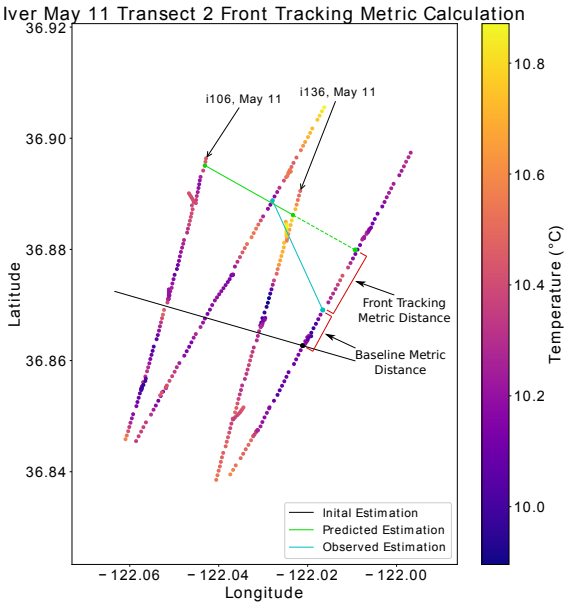


Figure 7: An example calculation of the front tracking metric. We use the May 11 i136 transect 2 for the example. The initial manually provided transect (i.e. *initial estimation*) is plotted a black line. The front-geometry estimation from the previous transect (i.e. *predicted estimation*) is plotted as a green line. The front-geometry estimation after transect 2 (i.e. *observed estimation*) is plotted as a blue line. The intersection of these three lines and the transect in question are plotted as dots of their respective colors. The distances used for the baseline and front tracking metric are shown in red.

day deployments due to the short range of the vehicles. Some operational constraints required modifications to the outlined front tracking control method. The range limitation associated with acoustic communication and the desire to have the ability for quick vehicle recovery required the two Iver AUVs to remain in close proximity to each other. The front tracking algorithm as presented does not guarantee any vehicle synchronization with regards to position. In order to solve this issue the vehicles pause at any point in which a new transect could start and waits for every other vehicle to reach their respective decision points. Once all vehicles have paused, the front-crossing detection algorithms are executed for each vehicle. If at least one vehicle has detected a front crossing, a new linear front estimation well be generated and all vehicles will be commanded orthogonal to it. If no front crossings are detected then all vehicles will continue on the current transect.

In this experiment the minimum transect distance was set at 3 km past the current estimated front. The minimum distance required for a vehicle to go past the

	Baseline Metric (m)	Front Tracking Metric (m
Average	1619.598	839.393
Std Dev	943.674	523.301

Table 1: Baseline and Front Tracking metric for the Iver Experiment on 9 and 11 May 2017

front-crossing detection on a given transect was set to 0 km, this results in the vehicle turning around at the first decision point after a front crossing is detected. The first decision point can be significantly past the detected front crossing due the minimum transect length. Ideally this would be set to a longer distance to insure that the vehicle has crossed the entire front before calculating a new transect, however due to software constraints during this deployment this was not possible. Front-geometry estimation was performed with the latest front crossing from each vehicle. The lateral gradient front-crossing detection algorithm was used with the Iver AUVs. Figure 8 shows the results of the Iver experiment on 9 and 11 May, 2017. Two transects were completed per vehicle per day. The starting locations for each vehicle on each day are labeled. Temperature averaged from 10 meters to 15 meters is plotted. All front crossing and front-geometry estimations used during the deployment are shown as blue dots and blue lines respectively. A number of different depth intervals for front-crossing detection were used during the deployment in order to examine the sensitivity of the algorithm. For reference, the front crossings and front-geometry estimations for 10 meter to 15 meter depth range are also plotted in green.

The baseline and front tracking metric for the Iver experiment is presented in Table 1. These values are calculated with the transects from both vehicles on 09 May and 11 May. We see a lower average distance with the front tracking metric compared to the baseline metric, indicating an improvement in the ability to tracking a front when using the method presented here. This result is also indicative of the suitability of the Iver platform for this specific region. Iver AUVs are fast moving vehicles with relatively short transects operating in a region where the front is mainly bathymetry driven, resulting in smaller changes in ocean conditions between front crossings. The dataset presented here is limited. It is an an initial step towards understanding the performance of the front estimation and tracking algorithm, however more data is necessary to make conclusions.

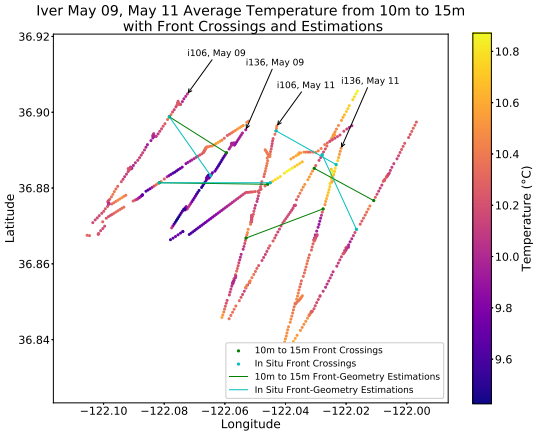


Figure 8: Map view of the temperature averaged from 10 to 15 meters for the Iver transects on 09 and 11 May, 2017. Front crossings and front-geometry estimations used during the experiment are indicated with a blue dot and blue line respectively. Front crossings and front-geometry estimations using data from 10 meters to 15 meters during the experiment are indicated with a green dot and green line respectively. The start location for each vehicle for each day is labeled.

Discussion

Related Work

Adaptive sampling and control of autonomous underwater vehicles has been extensively studied, including foundational work with the Autonomous Ocean Sampling Network [Curtin and Bellingham, 2009; Curtin et al., 1993; Haley et al., 2009; Leonard et al., 2007; Ramp et al., 2009]. Much of this work focuses on spatially adapting the control strategy in order to optimally sample a fixed region. Our method instead performs repeated focused sampling across a single front as it evolves over time.

Other work focused on control strategies that adapt to the current conditions. the Adaptive Sampling and Prediction project [Leonard et al., 2010] used adaptive control in order to coordinate 6 gliders to fly in loops at fixed spacing. Troesch et al. [2016] uses an ocean model in order to improve the station keeping ability of vertically profiling floats. Eriksen et al. [2001] describes the capabilities of a Seaglider to compensate for drift from currents using depth averaged currents over multiple dives. Those important works focus on adaptive control of vehicles based on the current conditions they are in, in order to improve sampling. We instead look at other hydrographic properties in order to optimize sampling of a specific feature.

A number of different near real-time feature tracking methods exist for applications such as thermoclines [Cruz and Matos, 2010; Sun et al., 2016; Zhang et al., 2010], and oil spills [Zhang et al., 2011]. These approaches focus on tracking a one-dimensional feature using a single vehicle, while we utilize multiple vehicles to track a two-dimensional feature. Flexas et al. [2018] uses an ocean model and autonomous planning to optimize sampling of submesoscale structures. Our approach focuses on frontal tracking using trailing in-situ vehicle data as apposed to an ocean model.

Other work has investigated two-dimensional feature tracking. Zhang et al. [2013, 2016] utilize the VTHI

front detection method on a single vehicle to detect and track an upwelling front on a zig-zag track with a fixed turn angle. Cruz and Matos [2014] tracks any gradient boundary using a single vehicle following a dynamic zig-zag pattern and a lateral gradient detection algorithm to estimate the gradient boundary using an arc whose curvature is defined by the last three front-crossing locations. Kularatne, Smith, and Hsieh [2015] tests a method in a tank to perform a zig-zag across a front using an autonomous surface vehicle. A similar method can also be applied to tracking the center of a phytoplankton bloom patch [Godin et al., 2011]. Machine learning, in the form of policy learning, has also been applied to the problem of tracking the edge of a harmful algal bloom [Magazzeni et al., 2014]. Other work focuses on tracking algal blooms by flying formations relative to the bloom as tracked by a drifter [Das et al., 2012]. Petillo, Schmidt, and Balasuriya [2012] uses a simulated network of AUVs in order to estimate the boundary of a simulated plume. These all differ from our approach in that we are using multiple vehicles in order to estimate the position and orientation of an ocean front using a method of gridded front detections as well as a linear front model.

Issues and Future Work

The front-crossing detection method is key in order for the front-geometry estimation and autonomous control portions of this method to work correctly. Throughout this experiment multiple points of improvement were identified in regards to the lateral gradient front detection. Front detection could be improved by gridding data based on distance traveled as apposed to time. This is particularly important for slower moving vehicles. The gridding process itself could also be improved by using objective mapping. In this experiment temperature was used, other ocean properties such as, buoyancy could also be used. The lateral gradient front detection method consists of many parameters, a more in-depth analysis of the effects of these parameters would be beneficial.

One of the issues encountered in the experiments was determining that the sampled front was the same as previously sampled fronts. Crossing multiple fronts would result in erroneous front-geometry estimations. In order to handle this situation our front-crossing detection technique would need to be extended in order to select a crossing based on a set of criteria such as front direction (i.e. cold-to-warm versus warm-to-cold), gradient strength, and front size. By using these different properties a specific front can be targeted.

The communication paradigms of the vehicles used is important as our technique was implemented off-board. Data decimation is an issue with vehicles that are unable to send all the available data to the planner. A data decimation scheme must be selected that allows for the front detection algorithms to perform well. These issues could be avoided by bringing the front tracking algorithm onboard the vehicles, however this introduces a number of different issues such as limited computing capabilities and inter-vehicle communication.

Conclusion

This work presents a method of adaptive control of multiple autonomous underwater vehicles in order to track an ocean front evolving over time. This method utilizes a near real-time front detection method, and an off-board planner doing front estimation using a linear model and vehicle retasking. This method builds upon the prior efforts of the AOSN deployments and takes a further step towards a fully-autonomous adaptive sampling framework [Thompson et al., 2017].

The experiment was conducted in May, 2017 in Monterey Bay, California using two short-range Iver AUVs. A front detection technique based on lateral gradients with gridded and interpolated data was used. During this experiment we demonstrated the performance of the front detection method on data from the vehicles. We also demonstrated the capability of the autonomous control method for front tracking. In doing this we introduced a metric which allows for a quantitative comparison of the front tracking algorithms performance as well as an indication of the suitability of a platform in a specific operating environment. The multi-vehicle front tracking approach allows for improved synopticity over a zig-zag method when sampling a front. While the use of off-board front detection, estimation, and retasking algorithms provided more processing power and allowed for flexible implementation for different platforms.

Acknowledgments The following work was done under the framework of the Keck Institute for Space Studies (KISS)-funded project "Science-driven Autonomous and Heterogeneous Robotic Networks: A Vision for Future Ocean Observations"[Thompson et al., 2017]. Portions of this work were funded by the Keck Institute and Woods Hole Oceanographic Institution. Portions of this work were performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

References

Belkin, I.; Cornillon, P.; and Sherman, K. 2009. Fronts in large marine ecosystems. *Prog. Oceanogr.* 81:223–236.

Bower, A.; Rossby, H. T.; and Lillibridge, J. L. 1985. The Gulf Stream–barrier or blender. *J. Phys. Oceangr.* 15:24–32.

Branch, A.; Flexas, M. M.; Claus, B.; Clark, E. B.; Thompson, A. F.; Chien, S.; Kinsey, J. C.; Fratantoni, D. M.; Zhang, Y.; Kieft, B.; Hobson, B.; and Chavez, F. P. 2018. Front delineation and tracking with multiple underwater vehicles. *J. Field Robotics* in review.

Brannigan, L. 2016. Intense submesoscale upwelling

in anticyclonic eddies. *Geophys. Res. Lett.* 43:3360–3369.

Cruz, N. A., and Matos, A. C. 2010. Adaptive sampling of thermoclines with autonomous underwater vehicles. In *OCEANS 2010*, 1–6. IEEE.

Cruz, N. A., and Matos, A. C. 2014. Autonomous tracking of a horizontal boundary. In *Oceans-St. John's, 2014*, 1–6. IEEE.

Curtin, T. B., and Bellingham, J. G. 2009. Progress toward autonomous ocean sampling networks. *Deep Sea Research Part II: Topical Studies in Oceanography* 56(3):62 – 67. AOSN II: The Science and Technology of an Autonomous Ocean Sampling Network.

Curtin, T. B.; Bellingham, J. G.; Catipovic, J.; and Webb, D. 1993. Autonomous oceanographic sampling networks. *Oceanography* 6(3):86–94.

Das, J.; Py, F.; Maughan, T.; OReilly, T.; Messié, M.; Ryan, J.; Sukhatme, G. S.; and Rajan, K. 2012. Coordinated sampling of dynamic oceanographic features with underwater vehicles and drifters. *The International Journal of Robotics Research* 31(5):626–646.

D'Asaro, E.; Shcherbina, A.; Klymak, J.; Molemaker, J.; Novelli, G.; Guigand, C.; Haza, A.; Haus, B.; Ryan, E.; Jacobs, G.; Huntley, H.; Laxague, N.; Chen, S.; F. Judt, J. W.; Barkan, R.; Kirwan, A.; Poje, A.; and Özgökmen, T. 2017. Ocean convergence an dispersion of flotsam. *Proc. Nat. Ac. Sci.* in press.

Eriksen, C. C.; Osse, T. J.; Light, R. D.; Wen, T.; Lehman, T. W.; Sabin, P. L.; Ballard, J. W.; and Chiodi, A. M. 2001. Seaglider: A long-range autonomous underwater vehicle for oceanographic research. *IEEE J. Oceanic Eng.* 26:424436.

Flexas, M. M.; Troesch, M. I.; Chien, S.; Thompson, A. F.; Chu, S.; Branch, A.; Farrara, J. D.; and Chao, Y. 2018. Autonomous sampling of ocean submesoscale fronts with ocean gliders and numerical model forecasting. *Journal of Atmospheric and Oceanic Technology* 35(3):503–521.

Godin, M. A.; Zhang, Y.; Ryan, J. P.; Hoover, T. T.; and Bellingham, J. G. 2011. Phytoplankton bloom patch center localization by the tethys autonomous underwater vehicle. In *OCEANS'11 MTS/IEEE KONA*, 1–6.

Haley, P.; Lermusiaux, P.; Robinson, A.; Leslie, W.; Logoutov, O.; Cossarini, G.; Liang, X.; Moreno, P.; Ramp, S.; Doyle, J.; Bellingham, J.; Chavez, F.; and Johnston, S. 2009. Forecasting and reanalysis in the monterey bay/california current region for the autonomous ocean sampling network-ii experiment. *Deep Sea Research Part II: Topical Studies in Oceanography* 56(3):127 – 148. AOSN II: The Science and Technology of an Autonomous Ocean Sampling Network.

Hickey, B. M. 1979. The california current system-hypotheses and facts. *Prog. Oceanogr.* 8:191–279.

Kularatne, D.; Smith, R. N.; and Hsieh, M. A. 2015. Zig-zag wanderer: Towards adaptive tracking of time-varying coherent structures in the ocean. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 3253–3258.

Leonard, N. E.; Paley, D. A.; Lekien, F.; Sepulchre, R.; Fratantoni, D. M.; and Davis, R. E. 2007. Collective motion, sensor networks, and ocean sampling. *Proceedings of the IEEE* 95(1):48–74.

Leonard, N. E.; Paley, D. A.; Davis, R. E.; Fratantoni, D. M.; Lekien, F.; and Zhang, F. 2010. Coordinated control of an underwater glider fleet in an adaptive ocean sampling field experiment in monterey bay. *Journal of Field Robotics* 27(6):718–740.

Lévy, M.; Klein, P.; and Treguier, A.-M. 2001. Impact of sub-mesoscale physics on production and subduction of phytoplankton in an oligotrophic regime. *J. Mar. Res.* 59(4):535–565.

Lynn, R. J., and Simpson, J. J. 1987. The california current system: The seasonal variability of its physical characteristics. *J. Geophys. Res.* 92:12947–12966.

Magazzeni, D.; Py, F.; Fox, M.; Long, D.; and Rajan, K. 2014. Policy learning for autonomous feature tracking. *Autonomous Robots* 37(1):47–69.

Mahadevan, A.; D'Asaro, E.; Lee, C.; and Perry, M. J. 2012. Eddy-driven stratification initiates North Atlantic spring phytoplankton blooms. *Science* 337(6090):54–58.

Mahadevan, A. 2016. The impact of submesoscale physics on primary productivity of plankton. *Ann. Rev. Mar. Sci.* 8(17.1–17.24).

Martin, A. P.; Richards, K. J.; Bracco, A.; and Provenzale, A. 2002. Patchy productivity in the open ocean. *Global Biogeochemical Cycles* 16:1025.

McWilliams, J. C. 2016. Submesoscale currents in the ocean. *Proceedings of the Royal Society A* 472:20160117.

Molemaker, M. J.; McWilliams, J. C.; and Dewar, W. K. 2015. Submesoscale instability and generation of mesoscale anticyclones near a separation of the california undercurrent. *J. Phys. Oc.* 45:613–629.

Monterey Bay Aquarium Research Institute. 2017. Canon spring 2017 expedition.

Petillo, S.; Schmidt, H.; and Balasuriya, A. 2012. Constructing a distributed auv network for underwater plume-tracking operations. *International Journal of Distributed Sensor Networks* 2012:Article ID 191235, 12pp.

Ramp, S.; Davis, R.; Leonard, N.; Shulman, I.; Chao, Y.; Robinson, A.; Marsden, J.; Lermusiaux, P.; Fratantoni, D.; Paduan, J.; Chavez, F.; Bahr, F.; Liang, S.; Leslie, W.; and Li, Z. 2009. Preparing to

predict: The second autonomous ocean sampling network (aosn-ii) experiment in the monterey bay. *Deep Sea Research Part II: Topical Studies in Oceanography* 56(3):68 – 86. AOSN II: The Science and Technology of an Autonomous Ocean Sampling Network.

Reid, J. L., and Schwartzlose, R. A. 1962. Direct measurements of the davidson current off central california. *J. Geophys. Res.* 67:2491–2497.

Ryan, J. P.; Chavez, F. P.; and Bellingham, J. G. 2005. Physicalbiological coupling in monterey bay, california: topographic influences on phytoplankton ecology. *Mar. Ecol. Prog. Ser.* 287:23–32.

Su, Z.; Wang, J.; Klein, P.; Thompson, A. F.; and Menemenlis, D. 2018. Ocean submesoscales as a key component of the global heat budget. *Nat. Comm.* accepted.

Sun, L.; Li, Y.; Yan, S.; Wang, J.; and Chen, Z. 2016. Thermocline tracking using a portable autonomous underwater vehicle based on adaptive threshold. In *OCEANS 2016-Shanghai*, 1–4. IEEE.

Taylor, J. R., and Ferrari, R. 2011. Ocean fronts trigger high latitude phytoplankton blooms. *Geophys. Res. Lett.* 38:L23601.

Thomas, L. N.; Tandon, A.; and Mahadevan, A. 2008. Sub-mesoscale processes and dynamics. In Hecht, M. W., and Hasumi, H., eds., *Ocean Modeling in an Eddying Regime*, volume 177 of *Geophysical Monograph Series*. Washington DC: American Geophysical Union. 17–38.

Thompson, A. F.; Chao, Y.; Chien, S.; Kinsey, J.; Flexas, M. M.; Erickson, Z. K.; Farrara, J.; Fratantoni, D.; Branch, A.; Chu, S.; Troesch, M.; Claus, B.; and Kepper, J. 2017. Satellites to seafloor: Toward fully autonomous ocean sampling. *Oceanography* 30(2):160–168.

Troesch, M.; Chien, S. A.; Chao, Y.; and Farrara, J. D. 2016. Planning and control of marine floats in the presence of dynamic, uncertain currents. In *International Conference on Automated Planning and Scheduling*, 431–440.

Zhang, Y.; Bellingham, J. G.; Godin, M.; Ryan, J. P.; McEwen, R. S.; Kieft, B.; Hobson, B.; and Hoover, T. 2010. Thermocline tracking based on peak-gradient detection by an autonomous underwater vehicle. In *OCEANS 2010*, 1–4. IEEE.

Zhang, Y.; McEwen, R. S.; Ryan, J. P.; Bellingham, J. G.; Thomas, H.; Thompson, C. H.; and Rienecker, E. 2011. A peak-capture algorithm used on an autonomous underwater vehicle in the 2010 gulf of mexico oil spill response scientific survey. *Journal of Field Robotics* 28(4):484–496.

Zhang, Y.; Godin, M. A.; Bellingham, J. G.; and Ryan, J. P. 2012a. Using an autonomous underwater vehicle to track a coastal upwelling front. *IEEE Journal of Oceanic Engineering* 37(3):338–347.

Zhang, Y.; Ryan, J. P.; Bellingham, J. G.; Harvey, J. B. J.; and McEwen, R. S. 2012b. Autonomous detection and sampling of water types and fronts in a coastal upwelling system by an autonomous underwater vehicle. *Limnology and Oceanography: Methods* 10:934–951.

Zhang, Y.; Bellingham, J. G.; Ryan, J. P.; Kieft, B.; and Stanway, M. J. 2013. Two-dimensional mapping and tracking of a coastal upwelling front by an autonomous underwater vehicle. *Proc. MTS/IEEE Oceans'13* 1–4.

Zhang, Y.; Bellingham, J. G.; Ryan, J. P.; Kieft, B.; and Stanway, M. J. 2016. Autonomous four-dimensional mapping and tracking of a coastal upwelling front by an autonomous underwater vehicle. *Journal of Field Robotics* 33(1):67–81.

Autonomous Nested Search for Hydrothermal Venting

Andrew Branch[1], Guangyu Xu[2], Michael V. Jakuba[2], Christopher R. German[2], Steve Chien[1], James C. Kinsey[2], Andrew D. Bowen[2], Kevin P. Hand[1], Jeffrey S. Seewald[2]

[1]Jet Propulsion Laboratory, California Institute of Technology

[2]Woods Hole Oceanographic Institution

Correspondence Author: andrew.branch@jpl.nasa.gov

Abstract

Ocean Worlds represent one of the best chances for the discovery of extra-terrestrial life within our own solar system. Liquid oceans are thought to exist on these celestial bodies, often encased in a thick icy shell. In order to investigate these oceans, a new mission concept utilizing a submersible craft must be developed. This vehicle would be required to traverse the icy shell and travel hundreds or even thousands of kilometers to survey the ocean below. In doing this, the vehicle might be out of contact for weeks or months at a time, requiring it to autonomously detect, locate, and study features of interest. Hydrothermal venting is one potential target, due to the unique ecosystems it supports on Earth. We have developed an autonomous, nested search strategy to locate sources of hydrothermal venting based on currently used methods. To test this search technique a simulation environment was developed using a hydrothermal plume dispersion simulation and a vehicle model. We show the effectiveness of the search method in this environment.

Introduction

At least eight bodies in our solar system are thought to harbor liquid oceans. In some cases, such as Europa and Enceladus, this ocean is perhaps habitable and encased in an icy shell kilometers thick [National Aeronautics and Space Administration 2018]. To explore these worlds new mission concepts must be developed using penetrating, submersible vehicles. A notional mission concept for such a submersible, outlined in Figure 1, contains four main components, an orbiting communications relay, a surface antenna, an under-ice base station, and a submersible vehicle. In order to facilitate ice shell transit, the vehicle needs to be small (particularly in cross sectional area). The long mission duration — potentially over a year to melt through the icy shell and a one year exploration mission — requires a low power vehicle, limiting the types of instruments on board. While the vehicle would ideally travel hundreds to thousands of kilometers distant from the base station, the submersible would need to return close to the base station to transfer data – with data subsequently relayed from the base station, through the surface antenna to the orbiter for eventual return to Earth. The radiation environment near the target body could preclude the use of an orbiting communication relay, instead relying on a relay in an eccentric Jovian orbit, in the case of Europa, increasing the time between communication windows from daily to monthly. When the submersible is away from the base station it would be unable to communicate with Earth. Therefore, while making journeys further and further away from the base station, the submersible might be operating days or weeks without contact. During this time the submersible would be required to autonomously detect, locate, and study a specific feature of interest.

Hydrothermal venting is one potential target for a submersible mission. Evidence for hydrothermal activity has been found on one Ocean World, Enceladus [Hsu et al. 2015; Waite et al. 2017]. On Earth, these geological phenomena harbor unique ecosystems and are potentially critical to the origin of life. Similar vents on Ocean Worlds could be the best chance at extra-terrestrial life in our Solar System. We have developed a fully autonomous nested search strategy for the localization of hydrothermal vents based on a manual three-phase nested search commonly used in the field [German et al. 2008]. In order to test this approach we have developed a simulation environment using FVCOM [Chen, Liu, and Beardsley 2003] — an existing ocean circulation model — and a vehicle model. Due to the resolution of the simulation environment, we focus on search in the non-buoyant plume. This corresponds to the ship based CTD casts and the phase 1 survey of the method presented in [German et al. 2008].

The rest of the paper is organized as follows. First we discuss the structure of hydrothermal venting. Then we discuss the simulation environment used to test our approach. We outline the approach itself and the experimental setup. Finally we discuss the results and future work.

Related Work

Adaptive sampling and control of autonomous underwater vehicles has been extensively studied, including foundational work with the Autonomous Ocean Sampling Network [Curtin et al. 1993; Curtin and Bellingham 2009; Ramp et al. 2009; Haley et al. 2009; Leonard et al. 2007].

Hydrothermal vent localization on Earth is often done with a non-autonomous three-phase nested search [German et al. 2008]. [Yoerger et al. 2007a] demonstrates this method in a number of cruises. [Yoerger et al. 2007b] presents a method to autonomously revisit areas of interest after the primary mission is completed, however this requires humans to develop the primary mission. This method was used in the

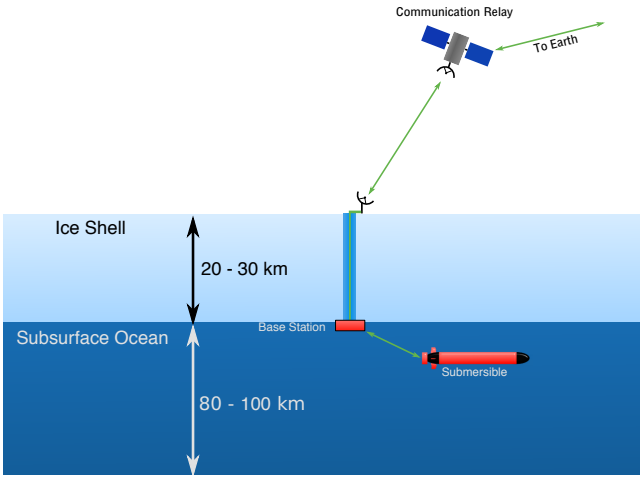


Figure 1: Notional Europa submersible mission showing the communication pathway from the submersible vehicle to Earth. Approximate ice thickness and ocean depth are labeled.

field multiple times. [Farrell, Pang, and Li 2005] field tests a strategy inspired by moths in order to trace chemical plumes.

Many approaches have been tested in idealized simulation environments or with deployment data, which does not allow for testing of fully autonomous planning algorithms. [Pang 2010] and [Tian et al. 2014] use moth based strategies in order to localize hydrothermal venting. [Jakuba and Yoerger 2008] uses occupancy grid mapping in order to localize vents. [Saigol et al. 2010] uses a belief-maximization algorithm to find a target of interest in simulation. [Ferri, Jakuba, and Yoerger 2010] uses a trigger based approach in order to gather higher resolution data in areas of strong sensor readings.

Hydrothermal venting is not the only target of interest. While not all ocean processes on Earth are expected to recur on other ocean worlds distant from the sun, we have a wealth of experience studying thermoclines, ocean fronts, and other structures in Earth's oceans. A number of different near real-time feature tracking methods exist for thermoclines [Cruz and Matos 2010; Zhang et al. 2010; Sun et al. 2016]. [Zhang et al. 2013; 2016] tracks upwelling fronts using a zig-zag pattern. [Cruz and Matos 2014] tracks any gradient boundary using a single vehicle following a dynamic zig-zag pattern and a lateral gradient detection algorithm to estimate the gradient boundary using an arc. A similar method can also be applied to tracking the center of a phytoplankton bloom patch [Godin et al. 2011]. [Branch et al. 2018] uses near real-time data to autonomously retask a set of vehicles to repeatedly sample an ocean front. Machine learning, in the form of policy learning, has been applied to the problem of tracking the edge of a harmful algal bloom [Magazzeni et al. 2014]. Other work focuses on tracking algal blooms by flying formations relative to the bloom as tracked by a drifter [Das et al. 2012]. [Petillo, Schmidt, and Balasuriya 2012] uses a simulated network of AUVs in order to estimate the boundary of a simulated plume. [Flexas et al. 2018] uses an ocean model and autonomous planning to optimize sampling of submesoscale structures.

Onboard autonomy has also been used to coordinate multiple vehicles and correct for ocean currents. The Adaptive Sampling and Prediction project [Leonard et al. 2010] used adaptive control to coordinate 6 gliders flying in loops at fixed spacing. [Troesch et al. 2016] uses an ocean model in order to improve the station keeping ability of vertically profiling floats. [Eriksen et al. 2001] describes the capabilities of a Seaglider to compensate for drift from currents using depth averaged currents over multiple dives.

Hydrothermal Venting

Hydrothermal venting produces a plume which can be traced back to the source. The structure of the plume is shown in Figure 2. Hydrothermal fluid exiting the vent is less dense than the surrounding water, resulting in the formation of a buoyant plume. Due to entrainment, the plume is continuously diluted by the ambient water column and expands from ~10 cm at the vent source to ~100 m at equilibrium. Upon reaching equilibrium, the plume expands horizontally — ten to hundreds of kilometers — to form the non-buoyant plume [German and Seyfried 2014]. The non-buoyant plume height is a function of the properties of the hydrothermal vent fluid as well as the surrounding water column [Turner 1979]. In the Pacific the non-buoyant plume is normally observed at 100-150 m above the seafloor, while in the Atlantic it is normally closer to 200-400 m [Speer and Rona 1989].

Hydrothermal plumes are the main source of information when localizing venting. However, tidal flows lead to local maxima [Veirs 2003], turbulent flow disrupting smooth gradients, differing vent types and strengths, and an unknown number of sources increase the difficulty of determining the plume source. [German et al. 2008] uses three primary sensors in the detection of hydrothermal plumes: temperature, optical backscatter [Baker, German, and Elderfield 1995; Baker and German 2004], and a chemical sensor such as oxidation-reduction potential [Nakamura et al. 2000]. These sensors may be good candidates for inclusion on a submersible mission to an Ocean World due to their compact form factor (100s of grams) and low power consumption (10s of milliwatts).

Simulation

A simulation environment was developed, using a hydrothermal plume dispersion simulation and a vehicle model. A numerical simulation of hydrothermal plume dispersion is performed using FVCOM, an ocean-circulation model, at Axial Seamount on the Juan de Fuca Ridge. The abundant lava supply to Axial supports vigorous hydrothermal systems and frequent volcanic activity, which have drawn extensive on-going scientific research that makes Axial one of the best-studied seamounts on this planet. A snapshot of this simulation is shown in Figures 3 and 4.

FVCOM is a finite-volume, time and density-dependent, three-dimensional, ocean circulation model [Chen, Liu, and

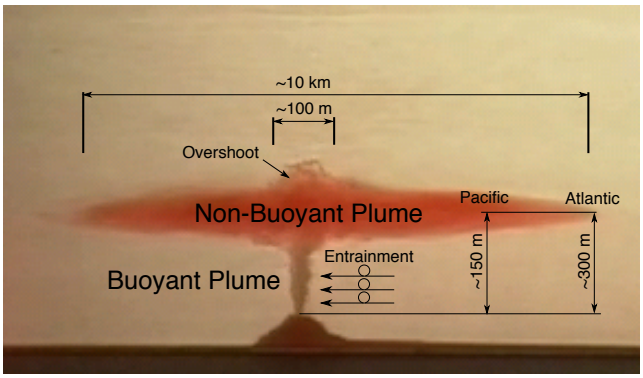


Figure 2: Demonstration of a hydrothermal plume performed in an aquarium tank. The buoyant and non-buoyant components of the hydrothermal vent plume are labeled with approximate scales. Image courtesy of C. German, WHOI

Beardsley 2003]. The unstructured grid employed in FVCOM supports grid size variation, therefore, proves efficient for the simulation of motion over a broad range of length scales. In addition, FVCOM supports the use of large-scale ocean circulation and tidal model outputs as open boundary forcing to drive flow across a broad range of frequencies inside the model domain [Zheng and Weisberg 2012].

Our model domain covers 300 by 300 km, centered on the Axial Seamount caldera and is open to flow across all four sides of that region. Horizontal resolution varies from 200 m within a 10 by 10km region enclosing Axial's caldera to 10km at the domain's boundary. The vertical dimension utilizes a uniform sigma-coordinate system with 127 layers, covering the full water column. This results in a ~12 m layer thickness above Axial's summit. The duration of the simulation is 58 days with model outputs sampled hourly. The 3-hourly sampled, 1/12.5° horizontal resolution, global reanalysis outputs of the HYbrid Coordinate Ocean Model (HYCOM) are used to construct the initial stratification profiles and open boundary forcing. Because HYCOM does not include ocean tides, we superimpose the tidal elevation and velocity predicted by the OSU Tidal Inversion onto the HYCOM outputs when constructing the open boundary forcing. We also add surface wind forcing and heat flux from 1-hourly sampled National Centers for Environmental Prediction (NCEP) Climate Forecast System Reanalysis (CFSR) outputs. We apply a linear ramp to bring open boundary and surface forcing from zero to full value over an initial four simulation days. Lastly, we add a seafloor heat source of 1 GW at the center (0,0) of the model domain inside Axial's caldera, which is turned on after the initial four simulation days. The model output consists of current, temperature, salinity, and a passive tracer, dye, which is released at the vent source. This tracer has a value range of $[0, 100]$. After 30 days the tracer content in a 20 by 20 km region surrounding the vent source reaches a quasi-steady state. In a 50 by 50 km region surrounding the vent source no quasi-steady state is reached before the end of the simulation.

The simulated vehicle uses a kinematic model and has three degrees-of-freedom: surge, heave, and yaw. A proportional controller allows the vehicle to navigate to a specified location. The nominal vehicle speed is set to 1 m/s. Simulated sensors are used to measure temperature, salinity, the passive tracer, vehicle depth, and distance to seafloor at a fixed interval. The position of the vehicle is assumed to be known at all times. Currently a chemical sensor, such as oxidation-reduction potential, and vehicle resources, such as energy and data capacity, are not modeled.

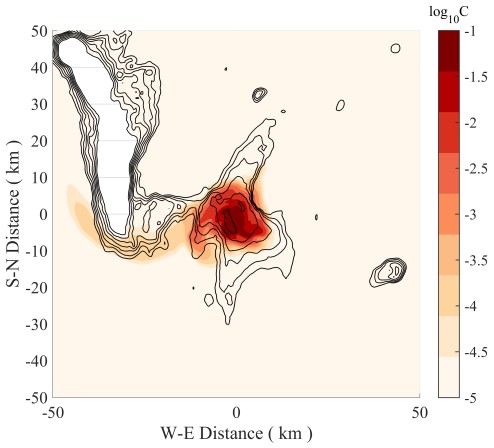


Figure 3: Snapshot taken at 1400 m depth on Mar 1, 2011 00:00 UTC of the simulated concentration (normalized by the source value) of a neutrally buoyant tracer originating from a hydrothermal vent source of 1 GW heat flux located inside the caldera of Axial Seamount at coordinate center. The global-simulation results of HYCOM and OSU Tidal Inversion for the period of Feb-Mar 2011 were used to drive flow inside the domain from its four boundaries.

Spatial Nested Search Strategy

Given a vehicle's starting location, the goal is to produce a control strategy that results in locating the vent source. The vent source is considered found when the region around the vent has been surveyed at a specified resolution. A resolution of 200 m was selected to match the resolution of the hydrothermal plume dispersion model at the vent source.

The strategy developed here addresses a number of issues. It mimics the field-proven methods of [German et al. 2008].

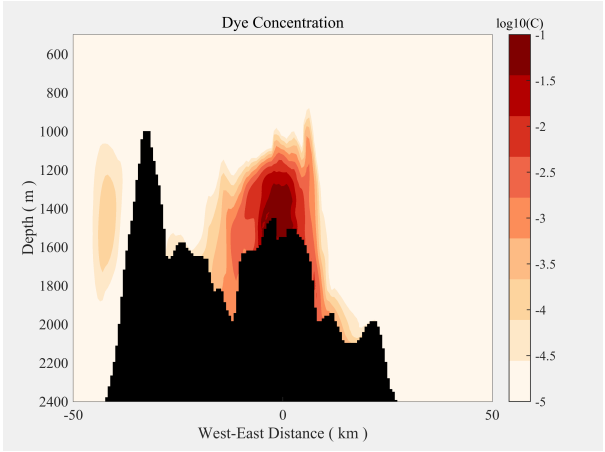


Figure 4: Snapshot taken along a W-E transect across the center of the model domain on Mar 1, 2011 00:00 UTC of the simulated concentration (normalized by the source value) of a neutrally buoyant tracer originating from a hydrothermal vent source of 1000 MW heat flux located inside the caldera of Axial Seamount at coordinate center. The global-simulation results of HYCOM and OSU Tidal Inversion for the period of Feb-Mar 2011 were used to drive flow inside the domain from its four boundaries.

Due to the limited resolution of the simulation environment, we focus specifically on search in the non-buoyant plume. (The buoyant plume is approximately 100 m, placing it below our 200 m resolution at the vent source.) This corresponds to the ship based CTD casts and — to some extent — phase 1 of the [German et al. 2008] method. Our strategy also allows for the localization of plume sources with differing strengths and maintains a robustness to local maxima in vent fluid concentrations and to small scale turbulence.

Before we can search for hydrothermal venting, we must have some method for detecting plumes. Ideally this would involve modeled sensors for temperature, optical backscatter, and oxidation reduction potential. However, currently we only use the passive tracer in the model as a direct measure of the hydrothermal plume. This is an area of future improvement.

The search algorithm is outlined in Algorithm 1 and operates as follows. A spiral is initiated at the start location. The horizontal spacing of the spiral is manually selected to be the expected size of the feature in question. This insures features of the expected size are seen during this initial survey. During this spiral the vehicle completes vertical profiles through the extent of the water column. When the max plume strength value of a single profile exceeds the specified threshold, $plume_t$ in Algorithm 1, the second phase of surveys begins. The height of the detected feature, p_h, is determined by binning the data from the vertical profile, p_d, at a 10 m resolution and selecting the bin with the largest average value. The subsequent surveys are performed at a depth of p_h. This is in contrast to the 3-phase strategy outlined in [German et al. 2008] because of our focus on search in the non-buoyant plume.

During the second phase of surveys, the search space is partitioned into bins, $survey\_bins$, of size $spacing_0$. These bins are separated into four quadrants centered on the corner of the bin closest to the location of the plume detection. A dynamic "lawnmower" survey is executed in each of the four quadrants. The dynamic lawnmower algorithm is outlined in Algorithm 2. The spacing of the lawnmower pattern, $track\_spacing$, is specified beforehand. The direction of the lawnmower pattern is defined by $along\_track$ and $across\_track$. Each track line of the lawnmower pattern consists of sections with length equal to the spacing. At least $min\_sections$ sections are be completed per track line. If $sections\_limit$ sections have average plume strengths below $plume_t$ and the sections have monotonically decreasing average plume strengths, then the track line is completed and the next track line is commenced. $min\_sections$ and $sections\_limit$ are manually specified search parameters. If the maximum value of an entire track line is less than $plume_t$ then the current lawnmower survey is ended and the next begins. The data from each dynamic lawnmower is binned into $survey\_bins$.

An example dynamic lawnmower is shown in Figure 5. The plot is subdivided into track line sections. The average plume strength is listed in each section; a green background indicates that the average plume strength is greater than the specified threshold, $plume_t$. Two boundaries to the survey are shown. Upon reaching the right-most boundary, the vehicle completes the current trackline. The boundaries correspond to the shared edges of the four quadrants defined during the search process.

Upon the completion of each dynamic lawnmower, local maxima of $survey\_bins$ are found. A maximum is declared when the 8 neighboring bins of the same resolution have a max plume detection value less than that of the center bin. Some a maximum has been found a nested "lawnmower" survey begins. An example of this process is shown in Figure 6. The local maximum — shown in green — and its neighbors are subdivided into smaller bins with one-third the side length of their parents. A lawnmower with spacing equal to one-third that of the previous lawnmower survey and with track lines centered on each row of nested bins is initiated. The new nested lawnmower survey covers the local maximum and all surrounding neighbors. If multiple local maxima have been found, they are prioritized on plume strength. This process repeats recursively until a survey spacing of $final\_spacing$, is reached. If no local

maxima are found during a dynamic lawnmower, or all local maxima have been exhausted before the survey spacing of $final\_spacing$ is reached, then the dynamic lawnmowers resume. After all dynamic lawnmower surveys are completed the spiral is resumed. Another set of dynamic lawnmowers is started if a plume is detected outside of the previously searched area.

Algorithm 1 Autonomous Nested Search

procedure NESTED_SEARCH
  $plans \leftarrow$ empty stack
  $visited \leftarrow$ empty set
  $plans.push(spiral)$
  $survey\_bins \leftarrow$ bins of size $spacing_0$
  while $plans.size > 0$ and not timed out **do**
    Execute or Continue $plans.top()$
    if executing spiral **then**
      Wait until end of vertical profile
      $p_d \leftarrow$ Get data from profile
      $d \leftarrow max(p_d)$
      if $d >= plume_t$ and $d.location$ not explored **then**
        $bins \leftarrow profile\_data$ binned at 10 meters and averaged
        $p_h \leftarrow max(bins).height$
        $(x, y) \leftarrow$ bin corner closest to $d.position$
        $plans.push(dynamic\_lawnmower(x, y, p_h, 90^\circ, 0^\circ, spacing_0))$
        $plans.push(dynamic\_lawnmower(x, y, p_h, -90^\circ, 0^\circ, spacing_0))$
        $plans.push(dynamic\_lawnmower(x, y, p_h, -90^\circ, 180^\circ, spacing_0))$
        $plans.push(dynamic\_lawnmower(x, y, p_h, 90^\circ, 180^\circ, spacing_0))$
        Execute $plans.top()$
    else
      while $plans.top()$ is not completed **do**
        Wait
      $survey\_data \leftarrow$ Get data from latest survey
      $survey\_bins.add\_data(survey\_data)$
      $maxima \leftarrow get\_bin\_maxima(survey\_bins)$
      sort $maxima$
      for bin in $maxima$ **do**
        if bin not in $visited$ **then**
          Partition bin and $bin.neighbors()$
          $visited.add(bin)$
          $plans.push(nested\_lawnmower(bin))$
          break
    while $plans.size > 0$ and $plans.top()$ is complete **do**
      $f \leftarrow plans.pop()$
      if $f.spacing < final\_spacing$ and f contains vent source **then**
        return Success
  return Failure

Algorithm 2 Execute Dynamic Lawnmower

procedure EXECUTE_DYNAMIC_LAWNMOWER(x, y, h , along_track, across_track, track_spacing)
  $start\_x \leftarrow x + cos(along\_track) * track\_spacing/2$
  $start\_y \leftarrow y + sin(across\_track) * track\_spacing/2$
  Go to $(start\_x, start\_y, h)$
  $curr\_track \leftarrow 0$
  $curr\_section \leftarrow 0$
  $completed \leftarrow False$
  $section\_data \leftarrow$ empty list
  Start current track line on heading $along\_track$
  while not $completed$ **do**
    Do next section on current track
    $section\_data[curr\_section] \leftarrow$ Get data from last section
    $curr\_section \leftarrow curr\_section + 1$
    if $curr\_section >= min\_sections$ or survey boundary reached **then**
      if $avg(section\_data[i]) < plume\_t$ for last $sections\_limit$ sections and monotonically decreasing **then**
        $curr\_track \leftarrow curr\_track + 1$
        if $max(section\_data) < plume\_thresh$ **then**
          $completed \leftarrow True$
          $section\_data \leftarrow$ empty list
          Travel $track\_spacing$ on heading $across\_track$
          if $curr\_track$ is even **then**
            Start next track line on heading $along\_track$
          else
            Start next track line on heading $-along\_track$

Experiment

121 scenarios were completed with the vehicle starting location uniformly varied between $x = [-30000, 30000]$ and $y = [-30000, 30000]$ at intervals of 6000m. Due to the nature of the algorithm and the location of the vent at $(0, 0)$ it is likely that the vehicle will pass directly over the vent source if the start location x and y are multiples of 1000. To mitigate this, a uniformly random value between $[-1500, 1500]$ was added to the x and y values of the starting location. The simulated vehicle has a horizontal and vertical velocity of 1 m/s. The vehicle samples the model at 0.2 hz. The plume detection threshold was set to 0.5. The initial spiral spacing was set to 5000 m and the initial dynamic lawnmower spacing was set to 4000 m. The dynamic lawnmower parameters $min\_sections$ and $sections\_limit$ are set to 4 and 2, respectively. The search parameters were selected based on preliminary results. More work investigating search parameters is necessary.

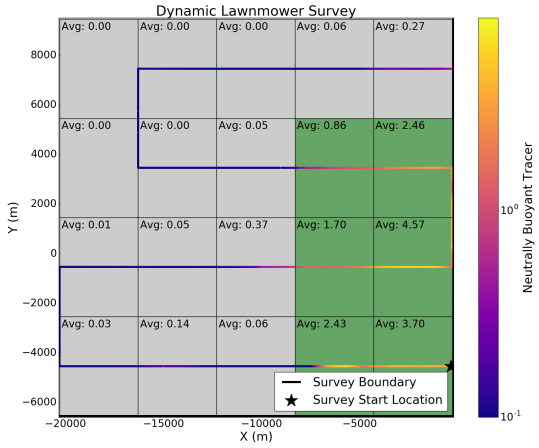


Figure 5: Plot showing an example dynamic lawnmower survey. The survey area is partitioned into regions representing sections of each track line. Regions shaded green have an average plume strength over the specified threshold. The average value is labeled in the upper left corner of each region. The two survey boundaries are shown as thick black lines on the right and bottom of the plot. The starting location is marked with a black star.

Results

87% of the simulation scenarios successfully found the vent location within 28 days. Figure 7 shows the time each run took to successfully find the vent in black. The runs that failed to find the vent are shown in red. Plot (a) shows the total time while plots (b), (c), and (d) show the time spent on the spiral survey, dynamic lawnmower surveys, and nested lawnmower surveys respectively. Figures 8, 9, and 10 show an example run plotting a top down view and a 3d view of the passive tracer (dye) value from the model, and a top down view of the survey types during the run, respectively.

We see a slight correlation between the distance and total time on successful runs. When this is decomposed into the different stages of the algorithm we see this correlation stronger within the spiral surveys while not at all in the lawn-

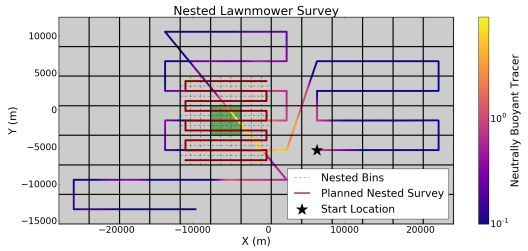


Figure 6: Plot showing an example of the planning process for a single nested lawnmower in one of the four quadrants. The search space is divided into square bins with sides equal to the lawnmower spacing. Upon finding a local maximum bin, the bin and all its neighbors are subdivided into nested bins of one-third the side length. A lawnmower pattern is then executed such that each track line is centered on a row of bins. The vehicle path and observed tracer is plotted. The planned nested lawnmower is show in dark red. The starting location is marked with a black star.
. Note that the measured passive tracer does not remain the same on subsequent measurements of the same location due to the temporal variation in fluid concentrations.

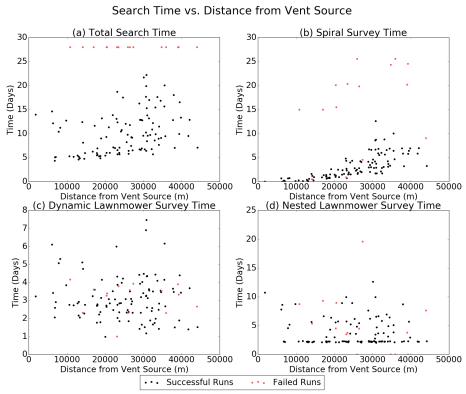


Figure 7: Plots showing the time to find the vent source compared to the distance from the vent source. The runs that successfully find the vent within 28 days are show in black. The failed runs are shown in red. Panel (a) plots the total time spent during the search. Panel (b), (c), and (d) decompose the time into the spiral survey, dynamic lawnmower surveys, and nested surveys respectively

mower surveys. No correlation is seen between the failed surveys and the distance from the vent, indicating that the cause of the failure is not related to distance. This method does not have a set distance in which it is feasible, starting further from the vent location would only require longer search times. Search times can be minimized by selecting appropriate values for the survey spacing parameters.

Upon initial investigation into the failed scenarios we see that the spiral surveys always detect the plume and initiate lawnmower surveys. Two failure modes are then observed in the lawnmower surveys. First, plume strength contours are not closed by the dynamic lawnmower survey. As such, they are not investigated by the nested lawnmower survey. Second, local maxima are not seen at the vent location. This could be caused by the temporal variation of the plume or from using constant depth, as apposed to constant density, lawnmower surveys.

Future Work

The planning method has many areas which could use further investigation. The lawnmower surveys could be improved by guaranteeing that contours will be closed, resulting in less failed searches. The non-buoyant plume is positioned at a constant density, not depth. As such, a fixed depth search is not ideal. In addition, the plume height can vary temporally on the order of 100 m over a tidal cycle on Earth [Rudnicki and German 2002]. A long duration search strategy, with respect to the tidal cycle, should be able to address this temporal variation. Improved search in the vertical direction would insure that the vehicle maintains contact with the strongest part of the plume. Temporal variations in the lateral direction should also be accounted for. This may be particularly important for slower vehicles, perhaps less so if they only move relative to the water, rather than relative to the ground or icy shell. Other geometric search patterns and other search strategies such as gradient search or biologically inspired approaches can be implemented and tested. Automated tuning of search parameters could improve results. Vehicle resource considerations can be incorporated into the planner. More intelligent path planning can be implemented to reduce resource consumption while performing multiple surveys. Hydrothermal activity is one potential target for a submersible; investigation into other targets and the development of a search approach capable of prioritizing multiple target types would be beneficial.

Currently, the vehicle simulation is rudimentary. Realistic models for sensors such as temperature, optical backscatter, and chemical sensors can be developed. Vehicle resources such as power and data capacity can be implemented. Finally, the vehicles motion model can be improved by advecting the vehicle according to the currents in the model.

The data volume collected by the vehicle far exceeds the communication throughput capabilities. Therefore, a method of summarizing the data collected needs to be developed. A number of spacecraft have implemented systems for this purpose. The Autonomous Sciencecraft Experiment used onboard science algorithms to summarize, delete, and prioritize data for downlink [Chien et al. 2005]. The onboard product generation for the Earth Observing-1 mission serves as a predecessor to the proposed HyspIRI Intelligent Payload Module [Chien et al. 2013]. The Mars Exploration Rover's (MER) WATCH system processes imagery to detect dust devils and send summarized data products to Earth [Castano et al. 2008]. The AEGIS system processes onboard imagery to autonomously retarget science instruments on the Mars Science Laboratory [Estlin et al. 2014] and MER [Estlin et al. 2012].

More simulation runs varying search parameters such as starting location, plume detection threshold, and survey

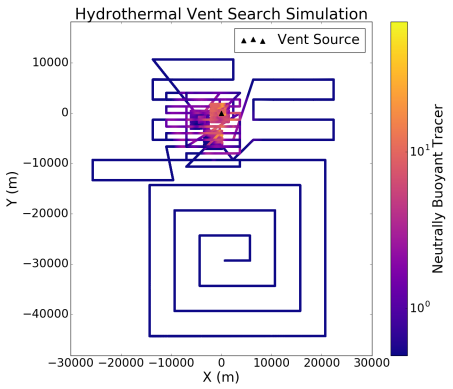


Figure 8: Top down plot showing the passive tracer (dye) as seen by the vehicle from a scenario starting at x=710, y=-29337. The vent source location is shown as a black triangle at (0,0).

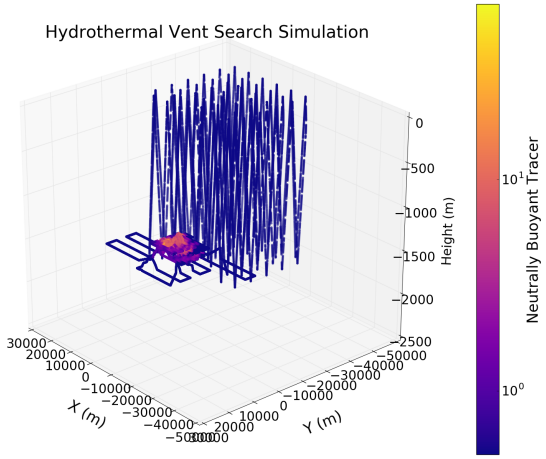


Figure 9: 3d view plot showing the passive tracer (dye) as seen by the vehicle from a scenario starting at x=710, y=-29337.

spacing would allow for a better understanding of the presented search strategy. Another plume dispersal model, either of a different region or with different plume parameters could be developed. Real world tests in well studied areas such as Axial Seamount would further validate the approach.

Conclusion

We developed an autonomous nested search based on the current manual three-phase search method [German et al. 2008], as well as a realistic simulation environment in which to test search strategies for the localization of hydrothermal venting. This simulation environment allows for testing at much larger spatial scales than has been investigated for other autonomous approaches. Search parameters, such as survey resolution and search location, allow for manual fine tuning of the search process based on the observed data, allowing for a human-in-the-loop model when possible. We

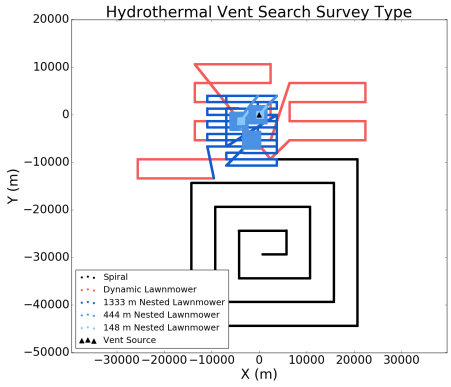


Figure 10: Plots showing the types of surveys performed on a scenario starting at x=710, y=-29337. The spiral survey is shown is black, the dynamic lawnmower surveys are red, and the nested lawnmower surveys are differing shades of blue with darker shades as surveys with larger spacing. The vent source location is shown as a black triangle at (0,0).

performed 121 scenarios with varying start locations, of which 87% were able to successfully find the hydrothermal vent within 28 days.

Acknowledgments

Portions of this work were performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

References

Baker, E. T., and German, C. R. 2004. On the global distribution of hydrothermal vent fields. *Mid-ocean ridges* 245–266.

Baker, E. T.; German, C. R.; and Elderfield, H. 1995. Hydrothermal plumes over spreading-center axes: Global distributions and geological inferences. *Seafloor hydrothermal systems: Physical, chemical, biological, and geological interactions* 47–71.

Branch, A.; Flexas, M. M.; Claus, B.; Clark, E. B.; Thompson, A. F.; Chien, S.; Kinsey, J. C.; Fratantoni, D. M.; Zhang, Y.; Kieft, B.; Hobson, B.; and Chavez, F. P. 2018. Front delineation and tracking with multiple underwater vehicles. *J. Field Robotics* submitted.

Castano, A.; Fukunaga, A.; Biesiadecki, J.; Neakrase, L.; Whelley, P.; Greeley, R.; Lemmon, M.; Castano, R.; and Chien, S. 2008. Automatic detection of dust devils and clouds on mars. *Machine Vision and Applications* 19(5-6):467–482.

Chen, C.; Liu, H.; and Beardsley, R. C. 2003. An unstructured grid, finite-volume, three-dimensional, primitive equations ocean model: Application to coastal ocean and estuaries. *Journal of Atmospheric and Oceanic Technology* 20(1):159–186.

Chien, S.; Sherwood, R.; Tran, D.; Cichy, B.; Rabideau, G.; Castano, R.; Davis, A.; Mandl, D.; Trout, B.; Shulman, S.; et al. 2005. Using autonomy flight software to improve science return on earth observing one. *Journal of Aerospace Computing, Information, and Communication* 2(4):196–216.

Chien, S.; Mclaren, D.; Tran, D.; Davies, A. G.; Doubleday, J.; and Mandl, D. 2013. Onboard product generation on earth observing one: a pathfinder for the proposed hyspiri mission intelligent payload module. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 6(2):257–264.

Cruz, N. A., and Matos, A. C. 2010. Adaptive sampling of thermoclines with autonomous underwater vehicles. In *OCEANS 2010*, 1–6. IEEE.

Cruz, N. A., and Matos, A. C. 2014. Autonomous tracking of a horizontal boundary. In *Oceans-St. John's, 2014*, 1–6. IEEE.

Curtin, T. B., and Bellingham, J. G. 2009. Progress toward autonomous ocean sampling networks. *Deep Sea Research Part II: Topical Studies in Oceanography* 56(3):62 – 67. AOSN II: The Science and Technology of an Autonomous Ocean Sampling Network.

Curtin, T. B.; Bellingham, J. G.; Catipovic, J.; and Webb, D. 1993. Autonomous oceanographic sampling networks. *Oceanography* 6(3):86–94.

Das, J.; Py, F.; Maughan, T.; OReilly, T.; Messié, M.; Ryan, J.; Sukhatme, G. S.; and Rajan, K. 2012. Coordinated sampling of dynamic oceanographic features with underwater vehicles and drifters. *The International Journal of Robotics Research* 31(5):626–646.

Eriksen, C. C.; Osse, T. J.; Light, R. D.; Wen, T.; Lehman, T. W.; Sabin, P. L.; Ballard, J. W.; and Chiodi, A. M. 2001. Seaglider: A long-range autonomous underwater vehicle for oceanographic research. *IEEE J. Oceanic Eng.* 26:424436.

Estlin, T. A.; Bornstein, B. J.; Gaines, D. M.; Anderson, R. C.; Thompson, D. R.; Burl, M.; Castano, R.; and Judd, M. 2012. Aegis automated science targeting for the mer opportunity rover. *ACM Transactions on Intelligent Systems and Technology (TIST)* 3(3):50.

Estlin, T.; Gaines, D.; Bornstein, B.; Schaffer, S.; Tompkins, V.; Thompson, D. R.; Altinok, A.; Anderson, R. C.; Burl, M.; Castaño, R.; et al. 2014. Automated targeting for the msl rover chemcam spectrometer. In *12th International Symposium on Artificial Intelligence, Robotics, and Automation in Space (i-SAIRAS)*, 17–19.

Farrell, J. A.; Pang, S.; and Li, W. 2005. Chemical plume tracing via an autonomous underwater vehicle. *IEEE Journal of Oceanic Engineering* 30(2):428–442.

Ferri, G.; Jakuba, M. V.; and Yoerger, D. R. 2010. A novel trigger-based method for hydrothermal vents prospecting using an autonomous underwater robot. *Autonomous Robots* 29(1):67–83.

Flexas, M. M.; Troesch, M. I.; Chien, S.; Thompson, A. F.; Chu, S.; Branch, A.; Farrara, J. D.; and Chao, Y. 2018. Autonomous sampling of ocean submesoscale fronts with ocean gliders and numerical model forecasting. *Journal of Atmospheric and Oceanic Technology* 35(3):503–521.

German, C., and Seyfried, W. 2014. Hydrothermal processes. *Treatise on geochemistry* 8:191–233.

German, C. R.; Yoerger, D. R.; Jakuba, M.; Shank, T. M.; Langmuir, C. H.; and Nakamura, K.-i. 2008. Hydrothermal exploration with the autonomous benthic explorer. *Deep Sea Research Part I: Oceanographic Research Papers* 55(2):203–219.

Godin, M. A.; Zhang, Y.; Ryan, J. P.; Hoover, T. T.; and Bellingham, J. G. 2011. Phytoplankton bloom patch center localization by the tethys autonomous underwater vehicle. In *OCEANS'11 MTS/IEEE KONA*, 1–6.

Haley, P.; Lermusiaux, P.; Robinson, A.; Leslie, W.; Logoutov, O.; Cossarini, G.; Liang, X.; Moreno, P.; Ramp, S.; Doyle, J.; Bellingham, J.; Chavez, F.; and Johnston, S. 2009. Forecasting and reanalysis in the monterey bay/california current region for the autonomous ocean sampling network-ii experiment. *Deep Sea Research Part II: Topical Studies in Oceanography* 56(3):127 – 148. AOSN II: The Science and Technology of an Autonomous Ocean Sampling Network.

Hsu, H.-W.; Postberg, F.; Sekine, Y.; Shibuya, T.; Kempf, S.; Horányi, M.; Juhász, A.; Altobelli, N.; Suzuki, K.; Masaki, Y.; et al. 2015. Ongoing hydrothermal activities within enceladus. *Nature* 519(7542):207.

Jakuba, M., and Yoerger, D. R. 2008. Autonomous search for hydrothermal vent fields with occupancy grid maps. In *Proc. of ACRA*, volume 8, 2008.

Leonard, N. E.; Paley, D. A.; Lekien, F.; Sepulchre, R.; Fratantoni, D. M.; and Davis, R. E. 2007. Collective motion, sensor networks, and ocean sampling. *Proceedings of the IEEE* 95(1):48–74.

Leonard, N. E.; Paley, D. A.; Davis, R. E.; Fratantoni, D. M.; Lekien, F.; and Zhang, F. 2010. Coordinated control of an underwater glider fleet in an adaptive ocean sampling field experiment in monterey bay. *Journal of Field Robotics* 27(6):718–740.

Magazzeni, D.; Py, F.; Fox, M.; Long, D.; and Rajan, K. 2014. Policy learning for autonomous feature tracking. *Autonomous Robots* 37(1):47–69.

Nakamura, K.; Veirs, S.; Sarason, C. P.; McDuff, R. E.; Stahr, F.; Yoerger, D. R.; and Bradley, A. M. 2000. Electrochemical signals in rising buoyant plumes and tidally oscillating plumes at the main endeavour vent field, juan de fuca ridge. *EOS, Transactions of the American Geophysical Union* 81(48).

National Aeronautics and Space Administration. 2018. Ocean worlds.

Pang, S. 2010. Plume source localization for auv based autonomous hydrothermal vent discovery. In *OCEANS 2010*, 1–8. IEEE.

Petillo, S.; Schmidt, H.; and Balasuriya, A. 2012. Constructing a distributed auv network for underwater plume-tracking operations. *International Journal of Distributed Sensor Networks* 2012:Article ID 191235, 12pp.

Ramp, S.; Davis, R.; Leonard, N.; Shulman, I.; Chao, Y.; Robinson, A.; Marsden, J.; Lermusiaux, P.; Fratantoni, D.; Paduan, J.; Chavez, F.; Bahr, F.; Liang, S.; Leslie, W.; and Li, Z. 2009. Preparing to predict: The second autonomous ocean sampling network (aosn-ii) experiment in the monterey bay. *Deep Sea Research Part II: Topical Studies in Oceanography* 56(3):68 – 86. AOSN II: The Science and Technology of an Autonomous Ocean Sampling Network.

Rudnicki, M. D., and German, C. R. 2002. Temporal variability of the hydrothermal plume above the kairei vent field, 25 s, central indian ridge. *Geochemistry, Geophysics, Geosystems* 3(2).

Saigol, Z.; Dearden, R.; Wyatt, J.; and Murton, B. 2010. Belief change maximisation for hydrothermal vent hunting using occupancy grids. In *Proceedings of the Eleventh Conference Towards Autonomous Robotic Systems (TAROS-10)*, 247–254.

Speer, K. G., and Rona, P. A. 1989. A model of an atlantic and pacific hydrothermal plume. *Journal of Geophysical Research: Oceans* 94(C5):6213–6220.

Sun, L.; Li, Y.; Yan, S.; Wang, J.; and Chen, Z. 2016. Thermocline tracking using a portable autonomous underwater vehicle based on adaptive threshold. In *OCEANS 2016-Shanghai*, 1–4. IEEE.

Tian, Y.; Zhang, A.; Li, W.; Yu, J.; Li, Y.; and Zeng, J. 2014. A behavior-based planning strategy for deep-sea hydrothermal plume tracing with autonomous underwater vehicles. In *OCEANS 2014-TAIPEI*, 1–10. IEEE.

Troesch, M.; Chien, S. A.; Chao, Y.; and Farrara, J. D. 2016. Planning and control of marine floats in the presence of dynamic, uncertain currents. In *International Conference on Automated Planning and Scheduling*, 431–440.

Turner, J. S. 1979. *Buoyancy effects in fluids*. Cambridge University Press.

Veirs, S. R. 2003. *Heat flux and hydrography at a submarine volcano: Observations and models of the Main Endeavour vent field in the northeast Pacific*. Ph.D. Dissertation, University of Washington.

Waite, J. H.; Glein, C. R.; Perryman, R. S.; Teolis, B. D.; Magee, B. A.; Miller, G.; Grimes, J.; Perry, M. E.; Miller, K. E.; Bouquet, A.; Lunine, J. I.; Brockwell, T.; and Bolton, S. J. 2017. Cassini finds molecular hydrogen in the enceladus plume: Evidence for hydrothermal processes. *Science* 356(6334):155–159.

Yoerger, D. R.; Bradley, A. M.; Jakuba, M. V.; Tivey, M. A.; German, C. R.; Shank, T. M.; and Embley, R. W. 2007a. Mid-ocean ridge exploration with an autonomous underwater vehicle.

Yoerger, D. R.; Jakuba, M.; Bradley, A. M.; and Bingham, B. 2007b. Techniques for deep sea near bottom survey using an autonomous underwater vehicle. *The International Journal of Robotics Research* 26(1):41–54.

Zhang, Y.; Bellingham, J. G.; Godin, M.; Ryan, J. P.; McEwen, R. S.; Kieft, B.; Hobson, B.; and Hoover, T. 2010. Thermocline tracking based on peak-gradient detection by an autonomous underwater vehicle. In *OCEANS 2010*, 1–4. IEEE.

Zhang, Y.; Bellingham, J. G.; Ryan, J. P.; Kieft, B.; and Stanway, M. J. 2013. Two-dimensional mapping and tracking of a coastal upwelling front by an autonomous underwater vehicle. *Proc. MTS/IEEE Oceans'13* 1–4.

Zhang, Y.; Bellingham, J. G.; Ryan, J. P.; Kieft, B.; and Stanway, M. J. 2016. Autonomous four-dimensional mapping and tracking of a coastal upwelling front by an autonomous underwater vehicle. *Journal of Field Robotics* 33(1):67–81.

Zheng, L., and Weisberg, R. H. 2012. Modeling the west florida coastal ocean by downscaling from the deep ocean, across the continental shelf and into the estuaries. *Ocean Modelling* 48:10 – 29.

Using a Hybrid AI-Planner to Plan Feasible Flight Paths for HAPS-Like UAVs

Jane Jean Kiam [1], **Enrico Scala**[2], **Miquel Ramirez**[3], **Axel Schulte**[1]
[1] University of the Bundeswehr, Munich, Institute of Flight Systems
[2] Fondazione Bruno Kessler
[3] The University of Melbourne, School of Computing and Information Systems

Abstract

Solar-powered, High-Altitude Long-Endurance (HALE) Unmanned Aerial Vehicles (UAVs) are a low cost alternative to fixed-orbit satellites providing surveillance and communications relay services. Such platforms are also often referred to as High-Altitude Pseudo-Satellite (HAPS). Flight planning for HAPS is challenging due to the inherent fragility of the light-weight materials used to construct their airframes. Adverse weather conditions pose a structural risk for the aircraft, and in the best case, can severely impair its performance. This paper discusses how HAPS flight path planning can be modeled with PDDL+, a declarative language that allows to specify the dynamics and constraints characterising complex hybrid control systems with ease. Flight plans, derived from PDDL+ descriptions of non–linear, non–homogeneous dynamical constraints that allow mobile obstacles, can be calculated efficiently with off-the-shelf, domain–independent hybrid planners. Albeit plans are generated on a more abstract model of the world, we show that these plans result executable when tested on a high fidelity simulator.

Remotely operated UAVs are nowadays regularly used to pursue tasks in which the presence of humans on board would result uneconomical, uncomfortable or hazardous. Increasing the degree of autonomy of UAVs is desirable not only for safety purposes, but also to improve economical viability (Johnson *et al.* 2017). Some types of UAVs present unique challenges when it comes to achieving higher levels of autonomy, this paper studies one.

Solar-powered, High-Altitude Low-Endurance (HALE) UAVs (Robert 1984) are a class of UAVs that can provide a viable alternative to fixed-orbit satellites in a number of applications (Klöckner 2016) due to their extreme endurance. These UAVs are also often referred to as High-Altitude Pseudo-Satellite (HAPS). Figure 1a depicts Zephyr 7, a HAPS that holds the world record for a continuous flight of 14 days at altitudes of 18 kms. On the other hand, HAPS pose a number of unique challenges in their operation. Their light-weight build ($\sim$100 kg), low airspeed ($\sim$30 m/s) and large wingspan ($\sim$30 m) result in platforms which are very sensitive to adverse atmospheric phenomena, which cannot be assumed to be static over the long periods of time that typical missions span. Their limited maneuverability, due mainly to the limited battery capacity ($\sim$15 kWh) and engine power (1.7 kW), further composes the challenges posed by weather conditions, and complicates navigation in strong wind fields. As recent tests (Araripe d'Oliveira *et al.* 2016; Morton *et al.* 2013) show that the technology readiness level (TRL) of similar platforms increases steadily, it is realistic to expect that frequent use in surveillance and mapping applications will be seen, since they are more flexible alternatives to low Earth orbit satellites and aircraft that require frequent refueling. However, the typical mission duration requires a substantial number of human operators to be readily available on a 24/7 basis. In order to improve economical viability of HAPS, increasing autonomy is essential to reduce manpower required in continuous operation. This paper proposes a flight-path planning approach using a hybrid AI-planner which is helpful to increase autonomy for offline mission planning, the context of which will be briefly described in the following paragraphs.

Exemplary Mission Scenario

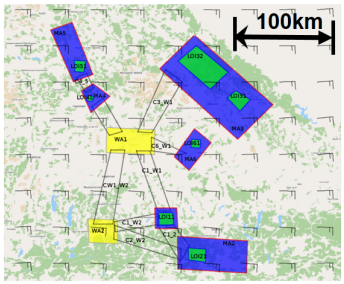
A typical realistic continuous surveillance and mapping mission is shown in Figure 1b, in which several Locations Of Interest (LOIs) marked with green polygons are landmarks to be continuously monitored. A well defined airspace is important so that the scenario is applicable also for more congested high-altitude airspace in the future (Johnson *et al.* 2017). The Mission Areas (MAs) in blue encompassing LOIs of the same client, denote the allocated airspace for carrying out the tasks at the operating altitude ($\sim$18 km). The Waiting Areas (WAs) represented in yellow are airspace in which the HAPS can loiter freely while not in mission execution, e.g. at night. A HAPS is allowed to move between MAs only through the designated Corridors (C).

Each mission element is available only within certain time windows as required by the mission or as according to airspace availability. Therefore, time of arrival at a mission element is closely relevant to the success of a mission plan.

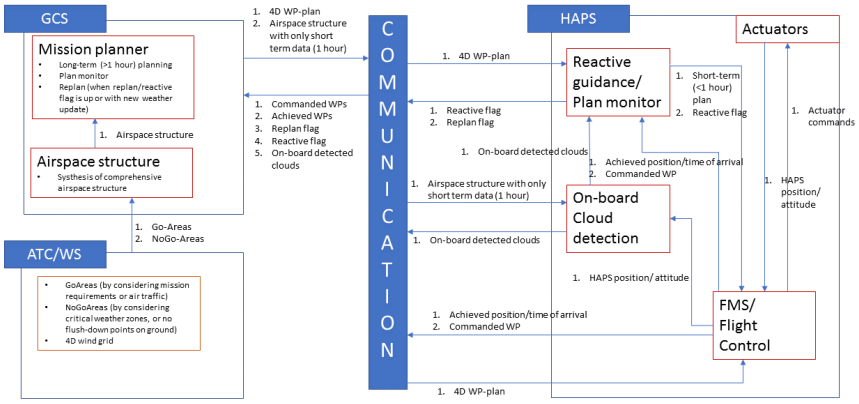
HAPS Mission Management System

The HAPS operates in a controlled airspace (Everaerts and Lewyckyj 2011); therefore the upcoming flight routes must be preplanned and communicated.

(a) Zephyr 7 ©Airbus Defence and Space GmbH



(b) A typical airspace structure defined for repetitive monitoring tasks



(c) HAPS mission management system

Figure 1: Zephyr 7: a solar-powered HALE, or rather HAPS as a satellite substitute for very long-term missions

Due to the aforementioned physical properties of a HAPS, their operation and planning is more challenging. To be taken into account in the mission planning are

- mission requirements (e.g. tasks, execution time)
- the allocated airspace for operation
- weather condition and the avoidance of dynamic critical weather zones, and
- flight dynamics (e.g. speed, turn rate etc.) in the time-varying wind field.

As illustrated in Fig. 1c, a Mission Management System (MMS) for HAPS consists of three major components (Müller *et al.* 2018): a mission planner to plan off-line for long-term tasks, a flight control system to guide the vehicle according to plans and a reactive guidance to steer the vehicle to safety in urgency.

Given the limited payload of the platform (5-20 kg), it is essential to limit the on-board equipments to only safety-critical real-time applications. Modules of MMS such as flight control and reactive guidance must be on-board, while long-term operational mission planning that works at fix intervals to plan or re-plan off-line for the tasks to execute in the next hours can be performed in the Ground Control Station (GCS). With this architecture, the ground-based mission planner will not be limited hardware-wise, as computation power is critical to process the weather data (Müller *et al.* 2018; Köhler *et al.* 2016) and plan accordingly.

Hierarchical Scheduling and Planning for Offline Mission Planning

Time of arrival is an important factor as the HAPS is bound to fulfill the tasks within the execution time windows as requested and while avoiding dynamic weather critical zones. Intuitive waypoint-planning or a geometric flight path planner is insufficient, since HAPS have rather limited airspeed range, therefore exerting the motor to compensate for the wind and reach a planned waypoint on time is not always an option.

LTL-based schedulers are efficient but the linearity assumption could affect the punctuality of the plan (i.e. the precision of the estimated time of arrival). A control-based/ action-based planner can better take into account the effect of wind on the flight dynamics but is computationally too expensive to solve the complete mission planning problem at once. A hierarchical planning architecture as illustrated in

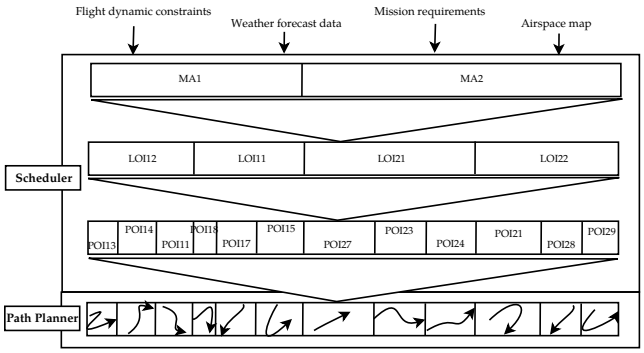


Figure 2: A hierarchical architecture for scheduling and planning

Figure 2 is adopted in our work in order to allow finer details to be successively considered down the hierarchy in a smaller and more isolated abstraction space (i.e. first the se-

quence of MAs which will be decomposed into sequences of LOIs and subsequently into sequences of Points-of-Interests (POIs), as represented by the red dots in Figure 13). The long-term mission planning is carried out strategically by the scheduler to decide for the sequence of points to visit, which is referred to as a "schedule" to be distinguished from the final "plan". The scheduler considers the airspace structure and the constant cruising airspeed of the HAPS ($\sim$28 m/s). More than one schedule can be found and they are ranked according to the expected probabilistic rewards, which decreases inversely proportional to the cloud coverage between the operation altitude ($\sim$18 km) and the landmarks, if an electro-optical mission camera is used. The LOI-scheduler in Figure 2 can be developed with a probabilistic approach as reported in (Kiam and Schulte 2017).

Tactically, the flight path planner refines the schedules by computing the point-to-point flight trajectories while considering the wind effect on the flight dynamics and while avoiding the static/dynamic no-go areas. The cascaded flight trajectories for a sequence of POIs is referred to as a "plan", which improves the time estimations of a schedule and hence the reward/cost estimations, thanks to the relief of linearity assumption.

Offline Flight Path Planning Problem

In this paper, we concentrate only on the offline flight path planning problem, i.e. the isolated planning problem between two POIs, as shown in Figure 2. Details on other parts of the MMS in Figure 1c can be found in (Müller *et al.* 2018; Kiam and Schulte 2017; Köhler *et al.* 2016; Klöckner 2016). We intend especially to demonstrate for the first time in this work the formulation in PDDL+ (Fox and Long 2006) for this class of planning problem, as well as the use of a domain-independent automated hybrid AI-planner ENHSP (Expressive Numeric Search Planner) (Scala *et al.* 2016a) as an off-the-shelf planner to solve our problem.

We first provide an insight of a domain-independent hybrid AI-planner and explain why it has considerable potential to solve the class of path planning problems in question. A typical kinematic model and the dynamic constraints of the HAPS is provided followed by its formulation in PDDL+. Subsequently, the various sophisticated platform-specific weather data are described and represented in PDDL+ so that the weather constraints can also be considered. Systematic tests were carried out to study the performance of the planner (and its heuristics). The results of the tests helps to fine-tune the implementation of the flight path planner using a planner. The planned flight paths are tested with a 6-DoF HAPS simulator built on realistic parameters of a HAPS (Müller *et al.* 2018). The results are shown and analyzed.

The PDDL+ formulation of the planning domain and problems will be made publicly available.

Domain-Independent Planners

Over the past 15 years, numerous domain-independent planners, that operate over descriptions of problems given in the standardised Problem Domain Definition Language (PDDL) (McDermott 2000), have been reported to scale up on huge discrete planning tasks (Richter and Westphal 2010), by exploiting the fact that logical dependencies between the cause and effects of an action can be encoded naturally in PDDL. Since the release of PDDL 2.1 (Fox and Long 2003) and subsequently of PDDL+ (Fox and Long 2006), it has become possible to represent compactly numeric effects and autonomous processes.

Definition 1 *(Hybrid Planning Problem) A planning problem H is given by the tuple $< X_p, X_n, A, P, X_0, G, C >$, where:*

- *X_p and X_n are the propositional and numeric state variables respectively,*
- *A is the set of instantaneous actions,*
- *P is the set of autonomous processes,*
- *X_0 is the initial state,*
- *G is the set of goal conditions, and*
- *C is the set of global constraints.*

Actions $a \in A$ are pairs $\langle pre(a), eff(a)\rangle$, where $pre(a)$ is a set (conjunction) of propositional and numeric *preconditions*, and $eff(a)$ is a set of *effects* boolean or numeric expressions indicating *instantaneous* changes of values in X_p and X_n. A more complete discussion of action preconditions and effects can be found in (Fox and Long 2006). An autonomous process $p \in P$ has a continuous effect on variables X_n over time. Like actions, they are a pair $\langle pre(p), eff(p)\rangle$ where preconditions $pre(p)$ are like those of actions, but effects $eff(p)$ are ordinary differential equations (ODE) $\dot{x} := exp(e)$, where $x \in X_n$ and $exp(e)$ is a well-formed arithmetic expression featuring standard mathematical operators, variables $y \in X_n$, constants and transcendental functions. While being syntactically equivalent to action precondition, a process precondition expresses an invariant condition along the execution of the process itself. Their violation causes the process to stop, so switching in what the hybrid automaton literature calls, another mode of execution. More details on the semantics aspects of PDDL+ can be found in Fox et al. (2006). Global constraints are arbitrary quantified-free formula over variables in $X_n \cup X_p$. They have to be satisfied by any state throughout the plan timeline. Solutions to H are plans, sequences of time-stamped actions a (Fox and Long 2006; Scala *et al.* 2016a).

While quite a number of domain-independent planners have been developed for some fragment of it (Hoffmann 2003; Gerevini *et al.* 2003; DellaPenna *et al.* 2009; Coles *et al.* 2012; Cashmore *et al.* 2016), only until recently domains with non-linear dynamics have been supported more effectively (Piotrowski *et al.* 2016; Scala *et al.* 2016a). In particular, ENHSP (Scala *et al.* 2016a) offers *support to trigonometric functions and global constraints, which are of critical importance to our application*.

Usability of a Domain-Independent Planner as a Flight Path Planner

Planning flight paths requires the consideration of three important factors: mission environment (wind and critical

zones), aircraft kinematics and mission requirements (goal, short travel time, etc.) (De Filippis and Guglieri 2012).

In the offline path planning problem for HAPS, since the mission environment varies with time and moving obstacles are to be avoided while considering flight dynamics, a control-based planner is most commonly used (LaValle 2006; Chakrabarty and Langelaan 2013; Doshi *et al.* 2013). Amongst the many available planning algorithms specifically relevant is Kinematic A* as presented in (De Filippis and Guglieri 2012), that relies on a simplified model of the aircraft in the planner: control inputs such as turn and climb rate are discretized, restricting the search space to those states reachable modulo discretization. Control-based planners publicly available on the Open Motion Planning Library (OMPL) (Sucan *et al.* 2012) are based for example on Rapidly exploring Random Tree (RRT) (LaValle and Kuffner 2001) and Kinodynamic Planning by Interior-Exterior Cell Exploration (KPIECE) (Sucan and Kavraki 2008).

Starting from the observation that in PDDL+, it is possible to separate the decisions of the actions to take from the dynamics of the system (by using actions and processes), whilst making sure that a set of global constraints remain satisfied along the resulting trajectory, next section explores for the first time a PDDL+ encoding of the HAPS flight path planning problem.

Defining the HAPS Movement Model

Similar to the kinematic model of a fixed-wing airplane in a wind field as described in (De Filippis and Guglieri 2012), used here are the equations of motion defined but with a spherical Earth assumption. The kinematics of the HAPS considered by the flight path planner are given by,

$$\begin{aligned} \dot{\lambda} &= (v_{\text{wind,E}} + v_{\text{TAS}} \cos\gamma \sin\chi)/(R+h)\cos\phi, \\ \dot{\phi} &= (v_{\text{wind,N}} + v_{\text{TAS}} \cos\gamma \cos\chi)/(R+h), \\ \dot{h} &= v_{\text{wind,U}} + v_{\text{TAS}} \sin\gamma. \end{aligned} \tag{1}$$

where λ, ϕ and h_j denote respectively the longitude, latitude and altitude, χ and γ denote the yaw and pitch angle, R denotes the radius of the Earth, $v_{\text{wind}} = (v_{\text{wind,E}}, v_{\text{wind,N}}, v_{\text{wind,U}})^{\text{T}}$ being the wind velocity in the East-North-Up coordinates, and v_{TAS} being the True Air Speed (TAS).

An action-based discrete path planner considers a set of actions while searching for a (sub-)optimal path. In our case, we use a set of feasible discrete turn rate $A_{\dot{\chi}} = \{-|\dot{\chi}_{\max}|, -|\dot{\chi}_{\max}| + \Delta\dot{\chi}, ..., |\dot{\chi}_{\max}| - \Delta\dot{\chi}, |\dot{\chi}_{\max}|\}$ and climb angles $A_\gamma = \{-|\gamma_{\max}|, -|\gamma_{\max}| + \Delta\gamma, ..., |\gamma_{\max}| - \Delta\gamma, |\gamma_{\max}|\}$ to allow for the dynamic constraints.

According to (McDermott 2003; Fox and Long 2006), actions in PDDL+ have instantaneous effects and are selected by the planner executive in the development of a plan while events are a control of the world. Processes, which run over time as long as the conditions are met, are independent of the planner's choice, and can be initiated by actions or events.

The range of turn rate and climb angle can be formulated as actions in PDDL+. Figure 3 shows the action to increase the turn rate `chi_rate ?uav`, while being subject to its limits, as described mathematically by Eq. 2.

$$\dot{\chi} := \dot{\chi} + \Delta\dot{\chi}, \quad \text{if} \quad \dot{\chi} < |\dot{\chi}_{\max}| - \Delta\dot{\chi}. \tag{2}$$

Similar formulations are application to the actions of decreasing the turn rate and selecting a climb angle.

```
(:action increase_turn_rate
    :parameters (?uav -uav)
    :precondition (and
        (< (chi_rate ?uav) (- (max_chi_rate ?uav)
                              (delta_chi_rate ?uav)))))
    :effect (and
        (increase (chi_rate ?uav) (delta_chi_rate ?uav)
        ) ))
```

Figure 3: PDDL+ snippet to show the action of increasing the turn rate `chi_rate ?uav`.

Subsequently, the position of the HAPS has to be updated using Equations 1 with processes formulated as shown in Figure 4. Similarly the attitude of the airplane can be updated using a process.

```
(:process update_latitude
    :parameters (?uav -uav)
    :precondition ()
    :effect (and
        (increase (phi ?uav)
            (* #t (/ (+ (* (v ?uav)
                           (* (cos (gamma ?uav))
                              (cos (chi ?uav))))
                        (north_wind ?uav))(+ R (h ?uav))
                  )))))

(:process update_longitude
    :parameters (?uav -uav)
    :precondition ()
    :effect (and
        (increase (lambda ?uav)
            (* #t (/ (+ (* (v ?uav)
                           (* (cos (gamma ?uav))
                              (sin (chi ?uav))))
                        (east_wind ?uav))
                     (* (cos (phi ?uav)) (+ R (h ?uav)))
                  ))))

(:process update_altitude
    :parameters (?uav -uav)
    :precondition ()
    :effect (and
        (increase (h ?uav)
             (* #t (+ (* (v ?uav) (sin (gamma ?uav)))
                      (up_wind ?uav))
             ))
```

Figure 4: Formulation in PDDL+ to update the WGS84 position of the HAPS

The `v ?uav` refers to the TAS of the HAPS. As a fixed-wing aircraft, it flies at an optimal equivalent airspeed (EAS) of $\sim$9 m/s (Müller *et al.* 2018), which can then be scaled

using the following equation to obtain the TAS at different altitude levels:

$$v_{\text{TAS}} = v_{\text{EAS}} \sqrt{\rho(h)/\rho_0}, \tag{3}$$

where $\rho(h)$ and ρ_0 are respectively the ambient and sea-level air densities given by the International Standard Atmosphere. Figure 5 shows the determination of TAS `v ?uav` as a continuous process. The used formulations in PDDL+,

```
(:process determine_airspeed
    :parameters (?uav -uav ?h_level -h_level)
    :precondition (and
        (< (h ?uav) (h_max ?h_level))
        (> (h ?uav) (h_min ?h_level)))
    :effect (and
        (assign (v ?uav)
            (* (v_eas ?uav) (^ (/ (rho ?h_level)
                                  (rho_0)) 0.5))) ))
```

Figure 5: PDDL+ formulation of the process to determine the true airspeed

involve more sophisticated algebraic operations like the exponential and trigonometric functions.

Figure 3 to 5 account respectively for the increase of the turn rate as described by Eq. 2 (similarly for the decrease of turn rate), the update of the position of the HAPS in WGS84 coordinates (latitude, longitude and altitude), which consists of integrating Equations 1 over time, and the determination of the altitude-dependent true airspeed of the platform as given by Equation 3.

Modeling Weather in PDDL+

More sophisticated weather data are becoming available for airborne vehicles as the conventional wide area weather data soon becomes insufficient, since the sensibility towards weather differs from aircraft to aircraft. HAPS are extremely fragile; furthermore, while planning for HAPS, long-term forecast is especially important since the vehicle cannot fly around hazardous zones so swiftly as most aircrafts. Several types of weather forecast suitable for a long-term planning for HAPS are used in our work:

- Cb-like thunderstorm forecast from the German Aerospace Center (DLR) (Köhler *et al.* 2016; 2017): it computes the likelihood of thunderstorms for the upcoming hours using fuzzy logic. The critical zones are summarized as ordered two-dimensional convex polygons. Due to the aggressiveness of thunderstorm clouds, these polygonal hazardous zones are identical for all flight altitudes.
- Polygonal NoGo-areas due to turbulences, strong wind, clouds etc. are also provided by the DLR (Köhler *et al.* 2017) as two-dimensional convex polygons which differ for each altitude level.
- COSMO-DE four dimensional wind data from the German Meteorological Office (DWD) (Baldauf *et al.* 2011) are delivered in GRIdded Binary (GRIB) [1].

[1]Daily weather data in GRIB-format can be downloaded from https://www.dwd.de/DE/leistungen/opendata/opendata.html

Formulating Convex Polygonal Obstacles in PDDL+

The aforementioned pieces of weather information constrain the validity of a flight plan in a number of ways. To capture such requirements in a concise way, we make use of global constraints, an extension of PDDL+ definition by Fox & Long (2006), which is considered for example by ENHSP. The representation expresses constraints that are applicable at any time (Scala *et al.* 2016b). Compared to the inclusion of the constraints individually as preconditions of each action or process, the formulation using global constraints appears to be more concise.

Algorithm 1 Determine the inclusion of a point $p = (\lambda, \phi)$ in a convex polygon

Require: V, an ordered set of vertices of a convex polygon
1: **for** each edge $v_i v_{\overline{i+1}}$, where $v_i, v_{\overline{i+1}} \in V$ **do**
2: % $\overline{*}$ indicates circular indexing
3: determine a_i, b_i, c_i such that
4: $a_i\lambda_i + b_i\phi_i == c_i$ and
5: $a_i\lambda_{\overline{i+1}} + b_i\phi_{\overline{i+1}} == c_i$ and
6: $a_i\lambda_{\overline{i+2}} + b_i\phi_{\overline{i+2}} \leq c_i$
7: **end for**
8: **if** $\wedge_i(a_i\lambda + b_i\phi \leq c_i)$ **then**
9: p is in the convex polygon described by V
10: **else**
11: p is **NOT** in the convex polygon V
12: **end if**

For a given flight level, critical weather zones can be formulated as 2D convex polygons. We can check, using linear inequalities, if a point p lies in a convex polygon. For each edge $v_i v_{i+1}$ of the polygon, where $v_i, v_{i+1} \in V$, a set of ordered vertices, if p lies on the same side of the edge $v_i v_{i+1}$ as an arbitrary interior point of the polygon, then p in included in the polygon. The verification method is recapitulated in Algorithm 1.

Determining the parameters of the inequalities (Line 6 to 7) can be preprocessed in a weather data parser and provided as inputs to our PDDL+ planning problem definition. Checking on which side p lies (Line 9 to 12) can be formulated as a global constraint as shown in Figure 6. The `exist` quantificator checks if there is one edge of the obstacle `?obs` where the condition in Line 8 is false (negation of the inequality). With the universal quantification, the convex polygon can have an *arbitrary number of edges* without complicating the PDDL+ representation.

```
(:constraint convex_Cb_like_obstacle
    :parameters (?obs -obstacle ?uav -uav)
    :condition (exists (?edge -edge )
        (< (c ?edge ?obs)
           (+ (* (a ?edge ?obs) (lambda ?uav))
              (* (b ?edge ?obs) (phi ?uav)))) ))
```

Figure 6: PDDL formulation to check if a HAPS lies within a convex polygonal hazardous zone

Figure 7 depicts a planned trajectory while avoiding two

static convex polygonal obstacles in a homogeneous wind field.

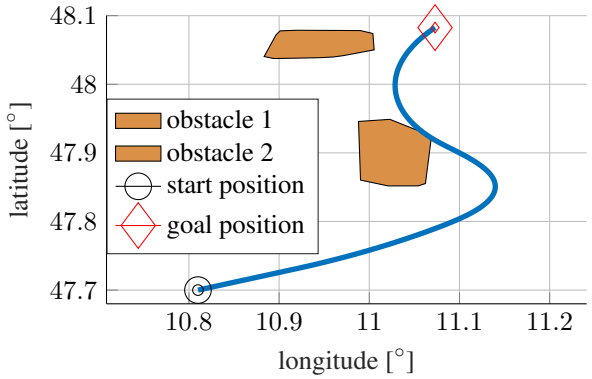


Figure 7: Avoiding convex polygonal obstacles in the presence of northwest wind

Some NoGo-areas, clouds for example, move along with the wind. In our work, this movement is assumed linear, with each vertex moving at the speed of the wind evaluated at the barycenter of the polygon. The variation over time of the inequality parameters determined with Algorithm 1 are given by

$$a(t+\delta t) = a(t) \quad , \quad b(t+\delta t) = b(t) \tag{4}$$
$$c(t+\delta t) = c(t) + v_{\phi,\text{wind}}(t) \cdot b(t) \cdot \Delta t \tag{5}$$
$$+ v_{\lambda,\text{wind}}(t) \cdot a(t) \cdot \Delta t \tag{6}$$

where $v_{\lambda,\text{wind}}$ and $v_{\phi,\text{wind}}$ are the zonal and meridional wind components in rad/s at the barycenter of the polygon. Although this assumption is simplified, it is practical, and necessary given the low airspeed of the HAPS and the wide mission areas. A safety margin can be added to the polygons to allow for deformation of the clouds and the nonlinear movements. The formulation of the above equations can be represented by processes in PDDL+ with basic algebraic functions.

Similarly, mission area (i.e. allocated airspace for a specific task as depicted in Figure 1b) can also be formulated using global constraints.

Formulating Discrete Wind Grid Data in PDDL+

The COSMO-DE wind data provides independently zonal wind u, meridional wind v, vertical wind w for a discrete 4D-grid (longitude-latitude-altitude-time). Although the altitude and time dimensions have regular discretization, the longitude-latitude 2D-grid has inhomogeneous spacing, i.e. four neighboring vertices form an arbitrary quadrilateral instead of a rectilinear shape (see Figure 8). The mean value of the wind components of each 4D-grid can be precomputed and parsed into the problem file in PDDL+.

Systematic Tests for Planning Performance

The main intention of this paper is to demonstrate, by using HAPS as an application example, that a hybrid heuristic AI-planner can be used as a full-fledged numeric planner to compute flight trajectories analytically by considering the dynamic behaviors of the vehicle in a vector field. Using the AI-planner comes with several advantages. First of all, the physical problem can be formally represented in a comprehensive way in PDDL+, as described in the previous sections. Secondly, the planner can be used *off-the-shelf*; in other words, the heuristic algorithms can be exploited blindly. Thirdly, the planner is less prone to error than a self-developed planner.

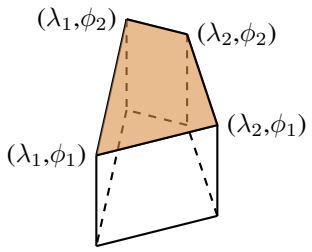


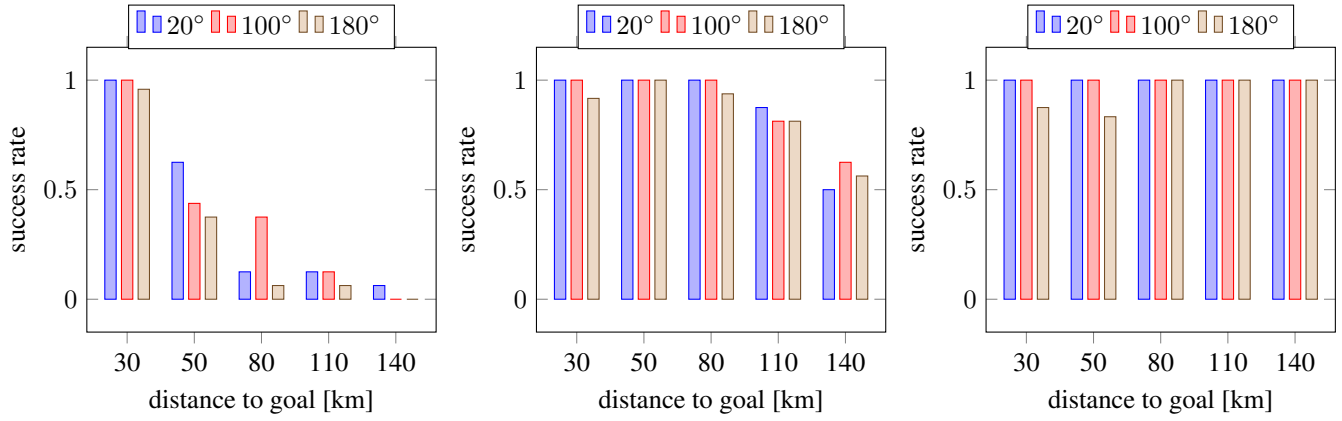
Figure 8: Visualisation in 3D (longitude-latitude-altitude) of the polytope

```
(:process determine_wind
    :parameters (?uav -uav ?grid -grid ?h_level -h_level
        ?time_interval -time_interval)
    :precondition (and
        (not (exists (?edge -edge )
            (< (c ?edge ?grid)
            (+ (* (a ?edge ?grid) (lambda ?uav))
        (* (b ?edge ?grid) (phi ?uav)))) ))
        (< (h ?uav) (h_max ?h_level))
        (>= (h ?uav) (h_min ?h_level))
        (< (t ?uav) (t_max ?time_interval))
        (>= (t ?uav) (t_min ?time_interval)) )
    :effect (and
        (assign (north_wind ?uav)
                (north_wind ?grid ?h_level ?time_interval))
        (assign (east_wind ?uav)
                (east_wind ?grid ?h_level ?time_interval))
        (assign (up_wind ?uav)
                (up_wind ?grid ?h_level ?time_interval)) ))
```

Figure 9: PDDL+ formulation to determine the wind components for a given grid in which the `?uav` is situated

We know that a point-to-point (sub-)optimal trajectory planning problem can be expressed using PDDL+ as shown in the previous sections. ENHSP[2] (Scala *et al.* 2016a) is capable of solving this problem class. ENHSP is a heuristic search forward state (Ghallab *et al.* 2004; Geffner and Bonet 2013) planner. It provides a front-end interface that is used to describe the planning problem H in a textual form. At the back-end, a search engine systematically and incrementally extends a search tree, rooted at X_0 with edges corresponding to spontaneous state transitions or instantaneous actions until a goal state or a fixed-point is reached. The planner includes a heuristic component, a general algorithm that computes automatically and efficiently a *relaxation* of H, H^+ for each state in the search tree. H^+ is then readily solved

[2]https://bitbucket.org/enricode/the-enhsp-planner



(a) $\Delta t = 30s$ (b) $\Delta t = 100s$ (c) $\Delta t = 150s$

Figure 10: Success rate to plan in a wide operation area from start to goal within 5s

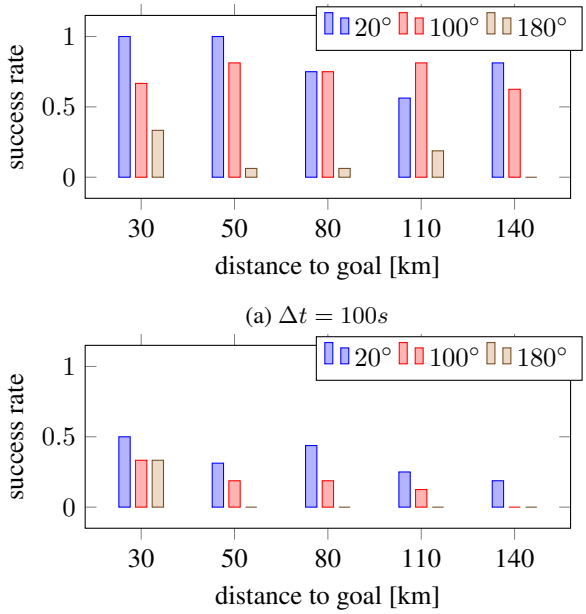
by whatever methods deemed suitable to produce a heuristic estimate of the sequence of transitions required to reach goal states. ENHSP heuristic component, the AIBR heuristic, has been shown experimentally to provide effective guidance, thus limiting the size of the search tree considered over a very diverse set of domains (Scala *et al.* 2016a).

To evaluate the robustness of ENHSP in handling the problem, as suggested by (Hooker 1995), we generate a variety of instances differing among each other for the wind magnitude, number of obstacles, obstacle occlusion ratio, distance from the goal, initial heading with respect to the goal etc.

Figure 10 shows the performance of the planner for a two-dimensional trajectory planning from a start to a goal position in a wide area polygon similar to the mission areas illustrated in Figure 1b with mild wind magnitude between 0 m/s and 5 m/s (which is usual at altitudes of ∼18 km). The varied parameters are the search step of the planner (Δt) and the distance between the start and goal positions. The distances selected are reasonable for our use case as shown in Figure 1b. The different colors indicate the initial angle difference between the initial course heading of the HAPS and the start-goal vector. The planning time out was set to 5 seconds. We notice that a bigger search step improves tremendously the planning efficiency, especially when the distance to goal is substantial.

However, if the authorized airspace is reduced to a narrow corridor, the planning performance is even more impaired with increasing search step if the initial angle difference between the course heading of the HAPS and the start-goal vector is substantial, as seen in Figure 11.

Another interesting test result with respect to the performance in the presence of obstacles can be viewed in Figure 12. The test was performed by varying the obstacle occlusion ratio in the search space with a fix number of obstacles (i.e. two or five obstacles) in each set of test. The planning success rate within one minute reduces with increasing obstacle occlusion. However, in the case of only two obstacles, the success rate decreases more than in the case of five



(a) $\Delta t = 100s$

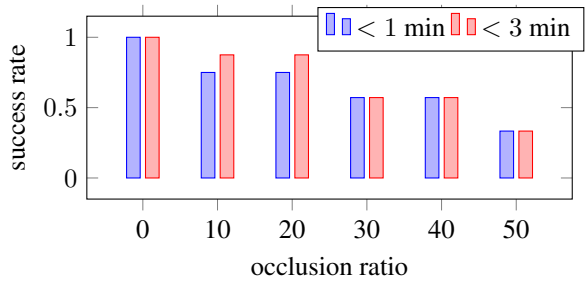
(b) $\Delta t = 150s$

Figure 11: Success rate to plan in a corridor-like narrow space from start to goal within 5s

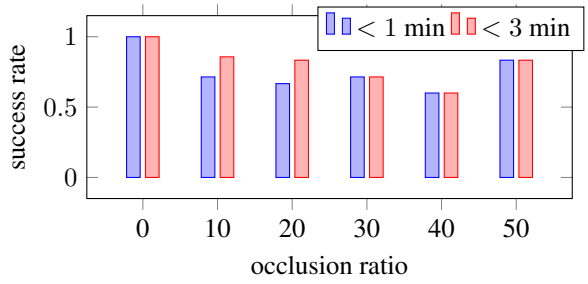
obstacles, mainly due to the the size of each obstacle. The heuristics of the planner guides the search toward the goal. However, if a huge obstacle happens to be in the way, it is harder for the planner to get round it.

Fine Tuning in the Implementation of Flight Path Planning using ENHSP

Due to the observations on the performance of the planner, the search step for ENHSP is set to 150 s if the distance to goal is larger than 80 km and 100 s otherwise. An advantage of using ENHSP is that the search step and the validation step can be set separately. Therefore even if explored



(a) 2 obstacles



(b) 5 obstacles

Figure 12: Performance of the planner with respect to obstacle occlusion ratio in the case of two and five obstacles respectively

Algorithm 2 Iterative Search with Relaxed Subgoals

Require: HAPS start position vector p_{start}, goal position vector p_{goal}

% assign initial position vector
$p_{\text{init}} = p_{\text{start}}$
% determine distance to goal
$d = |p_{\text{goal}} - p_{\text{init}}|$
determine bearing b, the angle difference between initial course heading and the heading between initial and goal positions
while $k = \lfloor b/20° \rfloor > 1$ **do**
    set subgoal conditions to:
    1) $|p_{\text{HAPS}} - p_{\text{goal}}| < (d - d/k)$
    2) $b < b - 20°$
    parse plan instance and call ENHSP
    $p_{\text{init}} = p_{\text{HAPS}}$
    $d = |p_{\text{goal}} - p_{\text{init}}|$
    determine bearing b
end while

nodes are spaced quite far apart, the smaller obstacles between nodes will not be missed since the plan validation is performed with a smaller step.

In the case where the search is to be performed within a narrow search space (e.g. a corridor), if the initial heading of the vehicle is too much deviated from the start-goal vector, the planner will be called iteratively by imposing subgoals placed between the start and goal positions, so that the course heading of the HAPS approaches the heading of the HAPS-goal vector. Algorithm 2 explains how ENHSP is called iteratively.

Plan Executability Validation

The development of HAPS is still at a Technology Readiness Level (TRL) of 2 to 3; not only that a real hardware test is financially costly, but is also rare due to difficulties to obtain a permission to fly (Everaerts and Lewyckyj 2011). In this work, to validate the generated paths, we use a six degrees of freedom (6-DoF) aircraft simulator provided by an external entity constructed based on a realistic HAPS model (Müller *et al.* 2018) coupled with a four dimensional flight controller (Müller and Looye 2013) to control the aircraft in the vertical and lateral directions so that the reference flight path as well as the airspeed are followed to keep track of the time of arrival at each point of the path. The mission scenario chosen is as described earlier: the HAPS is contracted to monitor the locations of interest at flight level (FL) 600 ($\sim$ 18 km) in the daytime. The weather forecast data used for planning as well as the nowcast data for simulation are historical data from the 27th June 2015. The weather forecast data is provided to the planner before the offline planning begins and a plan for the next hours is to be calculated and communicated so that the whereabouts and actions of the HAPS along the timeline can be predicted. Figure 13 shows partially the planned reference path from 06:30am local Bavarian time until noon. The six hours plan was computed offline within five minutes planning time with an Intel i7-6700K, 4GHz processor. In fact, flight paths that were successfully computed are feasible, except for when the forecasted weather is too different from the real weather, for instance if a huge Cumulonimbus cloud was not predicted in the weather forecast used for offline planning. In this case, a replanning or reactive avoidance is necessary, which is not the focus of this work. How the handover is achieved between different modules for flight guidance of HAPS in the event of urgency is reported in (Müller *et al.* 2018). We consider only scenarios in which the weather forecast is not too erroneous.

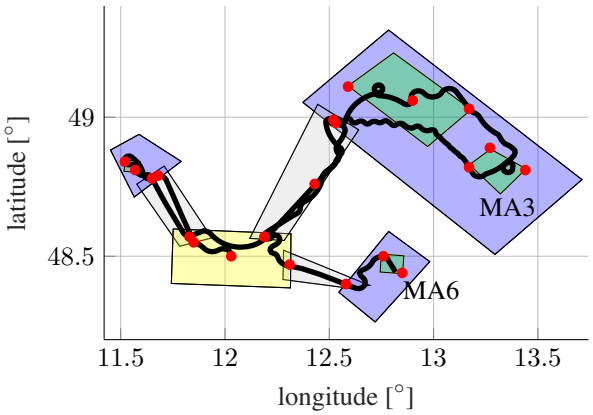


Figure 13: A typical mission plan of a HAPS performing surveillance tasks

The HAPS simulator can keep track with the planned path

while obeying its dynamic constraints. A couple of flight performances are however worth mentioning. Although the flight controller manages to follow the planned path, there is a slight deviation but acceptable between the planned and the flown paths, as seen in Figure 14.

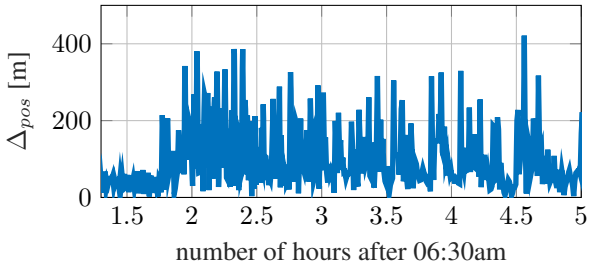


Figure 14: Deviation in position between the planned path and the simulated flight path

The deviation in position is maintained less than 420 m, which is acceptable for a HAPS (Müller *et al.* 2018). This experimental result is interesting as it provides a lower bound of the safety margin to critical zones that the planner should take into consideration while formulating the global constraints for the avoidance of critical weather zones. The deviation is due to several obvious reasons:

- the time discretization for plan validation ($\sim$10 s) is larger than that of the controller ($\sim$1 ms). Therefore, a flight management system (FMS) is integrated to parse and interpolate the planned path to force the reference trajectory to have the same time discretization step as the flight controller;
- the forecasted wind used by the planner differs from the nowcast wind data considered in the simulator.

However, due to the fact that the planner takes into consideration as much as possible the wind effect as well as the flight dynamics, the flight controller can follow the paths by maintaining an equivalent airspeed of around 9 - 10 m/s (see Figure 15), which is the optimal airspeed. It is hence more energy efficient and operationally safer since it is unlikely that the electro-motors are pushed to their power limit. Another cause that could lead to a deviation between the

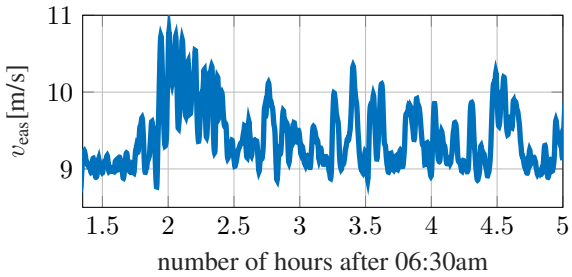


Figure 15: Equivalent airspeed during the test

planned and simulated paths is identified in Figure 16, the greater the turn rate is, the harder it is to follow the planned path. The planning model should be adapted so that turns can be penalized and avoided.

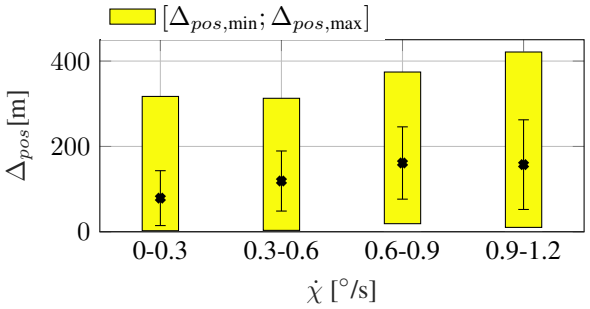


Figure 16: The relation between the error in position and the turn rate. The yellow bars indicate the range of the deviation while the black error bars indicate the standard deviations with the cross marking the mean error.

Conclusion

Automated motion planning for airborne vehicles has been a popular research topic. Many motion planning libraries can be used to solve the planning problem after some low-level adaptation. Automated planners from the domain-independent planning community (Fox and Long 2006) are often used for high-level logical planning. In this paper, we have provided the first proof that it is also possible to model in PDDL+ the notorious path planning problem of a HAPS-like light-weight UAV *traveling in a time-varying wind field while avoiding dynamic critical weather zones*. We have also identified a suitable domain-independent automated planner (ENHSP) to be used off-the-shelf to generate flight path plans. We used ENHSP because it is one of the few that remove several of the limitations of classical planners. Yet, it is wishful that more domain-independent planners can take interest in the class of planning problem we tackle in this work so that more planners can be chosen from. Some parameter testing was performed to help to fine-tune the implementation. A complex 6-DoF HAPS simulator and an exemplary mission scenario with real historical weather data were used to validate the feasibility of the generated paths.

Left for future works are the inclusion of more high-level logical actions in the planning problem definition, such as "turn on surveillance camera", "communicate with ground-control station" etc. It is also interesting to adapt and use a planner from the existing motion planning libraries such as OMPL to solve our problem and compare the planning runtime as well as the plan quality (e.g. trajectory feasibility, execution time, cost optimization etc.) with the domain-independent planners.

References

F. Araripe d'Oliveira, F. C. Lourenco de Melo, and T. C. Devezas. High-altitude platforms - present situation and technology trends. *Journal of Aerospace Technology and Management*, 8:249–262, July-September 2016.

M. Baldauf, A. Seifert, J. Förstner, D. Majewski, M. Raschendorfer, and T. Reinhardt. Operational convective-scale numerical weather prediction with the cosmo model: description and sensitivities. *Monthhly Weather Review*, 139:3887–3905, 2011.

M. Cashmore, M. Fox, D. Long, and D. Magazzeni. A compilation of the full PDDL+ language into SMT. In *Proc. ICAPS*, pages 79–87, 2016.

A. Chakrabarty and J. Langelaan. UAV flight path planning in time varying complex wind-fields. In *Proc. of ACC*, June 2013.

A. Coles, A. Coles, M. Fox, and D. Long. Colin: Planning with continuous linear numeric change. *JAIR*, 44:1–96, 2012.

L. De Filippis and G. Guglieri. *Advanced Graph Search Algorithms for Path Planning of Flight Vehicles, Recent Advances in Aircraft Technology, Dr. Ramesh Agarwal (Ed.)*. InTech, 2012.

G. DellaPenna, D. Magazzeni, F. Mercorio, and B. Intrigila. UPMurphi: a tool for universal planning on PDDL+ problems. In *Proc. ICAPS*, 2009.

A. A. Doshi, S. P. N. Singh, and A. J. Postula. An online motion planning and control strategy for UAVs in wind using reduced order forward models. In *Proceedings of Australasian Conference on Robotics and Automation*, December 2013.

J. Everaerts and N. Lewyckyj. Obtaining a permit-to-fly for a hale-uav in belgium. In *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Science, ISPRS Zurich 2011 Workshop,*, September 2011.

M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *JAIR*, 20:61–124, 2003.

M. Fox and D. Long. Modelling mixed discrete-continuous domains for planning. *JAIR*, 27:235–297, 2006.

H. Geffner and B. Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2013.

A. Gerevini, A. Saetti, and I Serina. Planning through stochastic local search and temporal action graphs in lpg. *JAIR*, 20:239–290, 2003.

M. Ghallab, D. S. Nau, and P. Traverso. *Automated planning - theory and practice*. Elsevier, 2004.

J. Hoffmann. The metric-ff planning system: Translating 'ignoring delete list' to numeric stats variables. *JAIR*, 20:291–341, 2003.

J.N. Hooker. Testing heuristics: We have it all wrong. *Journal of Heuristics*, 1:33–42, May 1995.

M. Johnson, J. Jung, J. Rios, J. Mercer, J. Homola, T. Prevot, D. Mulfinger, and P. Kopardekar. Flight test evaluation of an unmanned aircraft system traffic management (utm) concept for multiple beyond-visual-line-of-sight operations. In *Twelfth USA/Europe Air Traffic Management Research and Development Seminar (ATM 2017)*, October 2017.

J. J. Kiam and A. Schulte. Multilateral quality mission planning for solar-powered long-endurance uav. In *IEEE Aerospace Conference*, March 2017.

A. Klöckner. *Behavior Trees for Missions Management of High-Altitude Pseudo-Satellites*. Verlag Dr. Hut, Munich, 2016.

M. Köhler, T. Gerz, and A. Tafferner. Cb-like-cumulonimbus likelihood: Thunderstorm forecasting with fuzzy logic. *Meteorologische Zeitschrift*, 25:1–19, 2016.

M. Köhler, F. Funk, T. Gerz, F. Mothes, and E. Stenzel. Comprehensive weather situation map based on xml-format as decision support for uavs. *Journal of Unmanned System Technology*, 5:13–23, 2017.

S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20:378–400, May 2001.

S. M. LaValle. *Planning algorithms*. Cambridge university press, 2006.

D. McDermott. The 1998 ai planning systems competition. *AI Magazine*, 21(2), 2000.

D. McDermott. The formal semantics of processes in pddl. In *Proceedings of ICAPS'03 Workshop on PDDL*, Trento, Italy, June 2003.

S. Morton, L. Scharber, and N. Papanikolopoulos. Solar powered unmanned aerial vehicle for continuous flight: Conceptual overview and optimization. In *2013 IEEE International Conference on Robotics and Automation (ICRA)*, May 2013.

R. Müller and G. Looye. A constrained inverse modeling approach for trajectory optimization. In *AIAA Guidance Navigation and Control Conference*, 2013.

R. Müller, J. J. Kiam, and F. Mothes. Multiphysical simulation of a semi-autonomous solar powered high altitude pseudo-satellite. In *IEEE Aerospace Conference*, March 2018.

W. M. Piotrowski, M. Fox, D. Long, D. Magazzeni, and F. Mercorio. Heuristic planning for PDDL+ domains. In *Proc. of IJCAI*, 2016.

S. Richter and M. Westphal. The lama planner: Guiding cost-based anytime planning with landmarks. *JAIR*, 39(1):127–177, 2010.

J. B. Robert. History of solar flight. In *20th Joint Propulsion Conference*, June 1984.

E. Scala, P. Haslum, S. Thiebaux, and M. Ramirez. Interval-based relaxation for general numeric planning. In *European Conference on Artificial Intelligence*, August 2016.

E. Scala, M. Ramirez, P. Haslum, and S. Thiebaux. Numeric planning with disjunctive gobal constraints via smt. In *Proc. ICAPS*, June 2016.

I. A. Sucan and L. E. Kavraki. Kinodynamic motion planning by interior-exterior cell exploration. In *in Workshop on the Algorithmic Foundations of Robotics*, December 2008.

I. A. Sucan, M. Moll, and L. E. Kavraki. The open motion planning library. In *IEEE Robotics and Automation Magazine*, December 2012.

A Dynamic Task Planning System for Advanced Manufacturing Scenarios

Amedeo Cesta, Andrea Orlandini, and **Alessandro Umbrico**
Istituto di Scienze e Tecnologie della Cognizione,
Consiglio Nazionale delle Ricerche, Italy
Email: name.surname@istc.cnr.it

Abstract

Recent advances in Artificial Intelligence (AI) are facilitating the deployment of intelligent systems in manufacturing. In Human-Robot Collaboration (HRC), industrial robots offer accuracy and efficiency while humans guarantee both experience and specialized, not replaceable skills. The seamless coordination of such different abilities constitutes one of the current challenges. This paper presents a dynamic task planning system for robust HRC developed within an EU-funded project. The proposed solution uses Planning and Scheduling (P&S) techniques to deal with the temporal variance entailed by the active presence of humans as well as to dynamically adapt task plans according to actual behavior of the pair human-worker/robot. The tool has been deployed in a real pilot plant.

Introduction

During the last decade, *industrial robotic* systems have entered assembly cells supporting human workers in repetitive and physical demanding operations. The co-presence of robots and humans in a shared environment entails many issues to be properly addressed requiring *robust controllers* capable of preserving *productivity* and enforcing *human safety* (Freitag and Hildebrandt 2016). Human-Robot Collaboration (HRC) challenges concern both the *physical interactions*, guaranteeing *safety* of humans, and activities *coordination*, improving the productivity of the cell. From a *physical perspective*, an HRC task can be accomplished through many *robot trajectories* each of which could be executed concurrently to different human tasks. Motion controllers can modify the *speed* of robot motions for safety reasons and the time needed to perform robot tasks can vary significantly. From a *functional perspective*, HRC scenarios should exploit collaboration to enhance the efficiency of production processes (i.e., maximize the throughput) by means of proper assignment and coordination of human and robot tasks. Such HRC scenarios identify a task planning problem where coordination, the management of temporal uncertainty and robustness play an important role. It is worth noting that robustness here includes not only the safety of the worker but also flexibility and reliability of tasks coordination so as to actually take advantage of the capability of each of the interacting entities (human and robot).

Robustness indeed is a key enabling feature of controllers in HRC scenarios where robot motions must be continuously adapted for the presence of the human, which acts as an *uncontrollable agent* in the environment. This presence entails the ability of evaluating robot execution time variability a task where standard methods are not fully effective. In fact, current techniques are not able to foresee the actual time needed by robots to perform collaborative tasks (i.e., tasks that involve humans). Robot trajectories are computed *online* by taking into account the current position of the human and therefore it is not possible to know in advance the time needed by the robot to complete a task. Thus, it is not possible to plan robot and human tasks within a long production process and take into account performance issues at the same time.

Moreover, current solutions are not capable of dealing with task and motion planning in a uniform way without relying on limiting hypothesis (Michalos et al. 2014; Pellegrinelli et al. 2014). Some authors e.g., (Wolfe, Marthi, and Russell 2010; Srivastava et al. 2014; de Silva, Lallement, and Alami 2015), pursue a hierarchical integrated approach that rely on a clear distinction between task and motion planning features. In such cases, the task plan is built at an abstract and discrete level and is evaluated just before execution in order to verify the feasibility of the tasks. Additionally, these works do not consider *temporal information* and concurrent execution of human and robot tasks at planning time. Some plan-based controllers rely on temporal planning mechanisms capable of dealing with coordinated task actions and temporal flexibility (e.g., (Py, Rajan, and McGann 2010; Lemai and Ingrand 2004)) that rely respectively on temporal planners (e.g., (Barreiro et al. 2012; Ghallab and Laruelle 1994)). Unfortunately, these systems do not allow an explicit representation of *uncontrollability* features. Consequently, the resulting controllers are not endowed with the *robustness* needed to deal with the *temporal uncertainty* of HRC scenarios. These system usually rely on *replanning mechanisms* that may however strongly penalize the production performance. One additional aspect worth to be considered is the ability of planning software to support non specialist users for an easy integration of such solutions in different industrial settings.

Our long-term research goal is to realize a robust task planning system enabling flexible, safe and efficient HRC.

In (Cesta et al. 2016), the general pursued approach is presented aiming at realizing controllers capable to dynamically coordinate tasks according to the behaviors of human workers. This paper presents more recent results concerning the development of a task planning and execution technology deployed in realistic manufacturing scenarios. Specifically, the paper presents the FOURBYTHREE Engineering & Control Environment which integrates a task planning system with an *engineering environment* taylored to support robust human-robot collaboration.

The FOURBYTHREE Project

This work has been developed within FOURBYTHREE (FbT for short) project (Maurtua et al. 2016) that aims at realizing new robotic solutions to allow human operators to safely and efficiently collaborate with robots in manufacturing contexts. The project aims at addressing HRC challenges by creating a new generation of robotic solutions based on innovative hardware and software. The envisaged solutions present four main characteristics (*modularity*, *safety*, *usability* and *efficiency*) and take into account the co-presence of three different actors (the *human*, the *robot* and the *environment*). The resulting robotic solution of the project is tested in four pilot implementations representing two possible robot-human relationships in a given workplace without physical fences: (i) *coexistence* (the human and the robot conduct independent activities); (ii) *collaboration* (the human and the robot work collaboratively to achieve a given goal). FOURBYTHREE combines hardware and software advanced solutions for HRC scenarios: a brand new collaborative robotic arm has been designed and has been validated within the project; a set of software modules spanning from very high level features, such as, e.g., voice and gesture commands interaction, to low-level robot control have been developed. A complex integrated system has been produced to cope with a combination of HRC issues (Maurtua et al. 2016). The FOURBYTHREE Engineering & Control Environment is part of the software environment and is presented here. Its objective is to *support the design of plan-based control* for HRC scenarios as well as to *facilitate the access of manufacturing experts to AI planning technologies* for the synthesis of coordination and control strategies. The FOURBYTHREE Engineering & Control Environment integrates (i) planning and execution functionalities for coordination of robot control with the human activities, and (ii) representation features for the specification of production requirements.

The rest of the paper presents the general design of a dynamic task planning environment for facilitating both the specification of a subdivision of roles between robot and human in performing a task. Then, an example of the temporal planning of a flow of actions that guarantee safe coordination and an initial description of a run time support for the symbiotic execution of such a multi-agent plan is reported.

For the purpose of this paper, a human-robot collaboration workcell is a bounded connected space in which two agents (i.e., a human and a robotic system) collaborate (Marvel, Falco, and Marstio 2015). The robotic system consists of a robotic arm with a set of tools that can be either mounted on the arm or available within the workcell space as well as the workpieces and any other tool associated with the targeted task and dedicated safeguards (e.g., monitoring video cameras). In such workcell, different degrees of HRC interaction can be defined (Helms, Schraft, and Hagele 2002). In general, the robot and the human may need to occupy the same spatial location and interact according to different modalities. *Independent*, the human and the robot operate on separate workpieces without collaboration, i.e., independently from each other; *Synchronous*, the human and the robot operate on sequential components of the same workpiece, i.e., one can start a task only after the other has completed a preceding task; *Simultaneous*, the human and the robot operate on separate tasks on the same workpieces at the same time; *Supportive*, the human and the robot work cooperatively to complete the processing of a single workpiece, i.e., they work simultaneously on the same task. It is worth underscoring how different interaction modalities entails the robot to be endowed with different safety (hardware and control) settings while executing tasks.

In FOURBYTHREE, four pilot plants covering different types of production process have been considered for validation, i.e., assembly/disassembly of parts, welding operations, large parts management and machine tending. A general design approach has been pursued in order to elicit the relevant information concerning the following aspects: (i) *working procedure* describing production processes in terms of tasks to be performed and operational constraints; (ii) *human operator* describing the capabilities of the operator in terms of task that can perform and the related temporal features; (iii) *robot configuration* describing the available configurations of the robots and its capabilities; (iv) *human-robot collaboration* describing the envisaged interactions between humans and robots needed to successfully achieve production tasks. The reader may find a detailed pilots description in (Maurtua et al. 2016). Here, the *ALFA Precision Casting* pilot is briefly introduced. ALFA produces aluminum parts by means of "investment casting". This process is well suited for producing parts that require tight tolerances and dimensional precision but are produced in small size production batches. The considered process concerns collaborative assembly/disassembly operations. Specifically, the process consists in working a metal die which is used to produce a wax pattern and, then, after several processes, to obtain a mould for metallic components.

The considered process concerns *collaborative assembly/disassembly* operations. Specifically, the process consists in working a metal die which is used to produce a wax pattern in a injection machine. Once injected, the pattern is taken out the die. Several patterns are assembled to create a cluster. The wax assembly is covered with a refractory element, creating a shell (this process is called investing). The wax pattern material is removed by the thermal or chemical means. The mould is heated to a high temperature to eliminate any residual wax and to induce chemical and physical changes in the refractory cover. The metal is poured into the refractory mould. Once the mould has cooled down sufficiently, the refractory material is removed by impact, vibration, and high pressure water-blasting or chemical dissolu-

tion. The casting are then cut and separated from the runner system. Other post-casting operations (e.g. heat treatment, surface treatment or coating, hipping) can be carried out, according to customer demands. Due to the small size of the dies and the type of operations done by the worker to remove the metallic parts of the die, it is very complex for the robot and the worker to operate on the die simultaneously. However, both of them can cooperate in the assembly/disassembly operation. Specifically, once the injection process has finished, the die is taken to the workbench by the worker. Then, the robot and the worker can simultaneously *screw/unscrew* bolts on the different covers composing the die in order to disassembly/assembly the workpiece.

The FOURBYTHREE Engineering & Control Architecture

Given the HRC scenarios described above, there are many features and constraints that the envisaged control architecture must deal with in order to realize an effective, robust and safe collaboration. The architecture must capture and find a suited tradeoff among the requirements of the different stakeholders involved into the production process, i.e., a *Production Engineer*, a *Knowledge Engineer* and a *Human Worker* in addition to the specific *Robot* requirements. The *Production Engineer* is the expert of the production needs and specifies operational requirements of the different processes that can be performed. The *Knowledge Engineer* knows the features of the robot and of the specific working environment and, therefore, is responsible to model the production processes according to specified operational requirements. The *Human Worker* and the *Robot* are the main actors that actually carry out the production tasks to achieve the production process.

In general, several production processes can be performed within a factory. Each process consists of a set of *tasks* that must be executed according to some operational requirements. The perspective pursued here is the following: a *Worker* and a *Robot* represent two *autonomous agents* capable of executing different types of task. Some tasks can be executed only by the human, some tasks can be executed only by the robot and some tasks can be executed by both the human and the robot. Thus, given a particular process, the control system is responsible for synthesizing the set of needed tasks to complete the working process, assigning tasks to the human and to the robot and guaranteeing to *robustly and safely* executing them.

Figure 1 shows the FOURBYTHREE Engineering & Control Architecture developed for flexible human-robot collaboration. The architecture shows the elements and the actors involved within the control loop as well as their relationships. Specifically, the labeled arrows describe all the phases of the *control process* starting from domain modeling up to physical task execution. The *FbT Engineering Environment* relies on KEEN (*Knowledge Engineering ENvironment*) (Orlandini et al. 2014) to support domain experts in the design of the control model exploited by the *FbT Controller* to coordinate the human and the robot tasks. Specifically, the *FbT Engineering Environment* allows the *Production Engineer* and/or the *FbT Knowledge Engineer* to model the working environment and the production processes without knowing in details the specific planning and execution technology utilized. Once the model is defined, the *FbT Task Planner* synthesizes a temporal flexible plan assigning tasks to the human and to the robot and the *FbT Plan Executive* executes such plans in order to achieve the production goals. Both the *FbT Task Planner* and the *FbT Plan Executive* rely on PLATINUm (PLanning and Acting with TimelINes under Uncertainty) (Umbrico et al. 2018; 2017), a timeline-based planning and execution framework which complies with the formalization proposed in (Cialdea Mayer, Orlandini, and Umbrico 2016). Specifically, the developed task planner is capable of generating temporally robust plan by dealing with *temporal uncertainty* at solving time. This is crucial in the considered scenarios where a human must tightly cooperate with a robot. Indeed, a human is *uncontrollable* and his/her behavior may affect also the behavior of the robot from the control perspective. Thus, the *Human* is modeled as an autonomous and completely *uncontrollable* agent whose behavior may affect the behavior of the *Robot* which is modeled as a *partially controllable* agent.

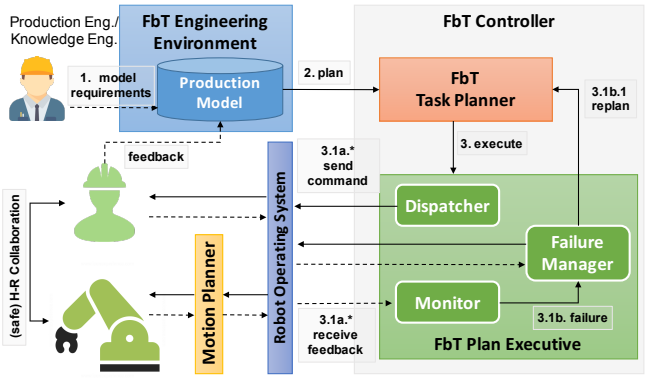


Figure 1: The FOURBYTHREE Engineering & Control architecture

A plan is executed by *dispatching* commands to the robot and to the human and by receiving *feedbacks* through dedicated communication channels implemented on ROS[1]. The *FbT Plan Executive* realizes a closed-loop control process which puts the human-in-the-loop. Broadly speaking, the executive is capable of dynamically adapting a task plan (i.e., robot task execution) according to the detected behavior of the human. Thus, the executive can temporally adapt a task plan by *absorbing* execution delays and generate a new plan through *replanning* only if strictly needed. Replanning allows the executive to manage exogenous events the plan cannot capture like e.g., a failure of a robot actuator or a human task whose duration is longer than expected and synthesize a new (adapted) plan which tries to complete the execution of the process. It is worth pointing out that the integration of *temporal uncertainty* at both planning and execution time makes the control process more *robust* than

[1] http://www.ros.org/

classical approaches in the literature e.g., T-REX (Py, Rajan, and McGann 2010) or IXTET-EXEC (Lemai and Ingrand 2004), limiting the need for replanning.

Planning and Execution with Uncertainty

The *FbT Controller* has been developed by following the *timeline-based planning approach*. This approach is a particular A.I. planning paradigm which has been introduced in early 90s (see for instance (Muscettola 1994)) and successfully applied in several real-world scenarios (mainly in space applications such as, e.g., (Cesta et al. 2007; Jonsson et al. 2000; Muscettola 1994)). This approach takes inspiration from the classical control theory and models a complex system by identifying a set of relevant features that must be controlled over time. Thus, a timeline-based application aims at controlling a system by synthesizing temporal behaviors of its features (i.e. *timelines*). Several timeline-based systems have been introduced in the literature (Barreiro et al. 2012; Chien et al. 2010; Ghallab and Laruelle 1994), each of which applies its own interpretation of this paradigm. The developed *FbT Controller* relies on the characterization of the timeline-based approach given in (Cialdea Mayer, Orlandini, and Umbrico 2016) which takes into account also *temporal uncertainty* and *controllability* features of the domains and plans (see the *controllability problem* (Vidal and Fargier 1999)). Indeed, *temporal uncertainty* and *temporal flexibility* play a key role in real-world applications, especially in HRC where a robot must cooperate at different interaction levels with a human which represents an autonomous and *uncontrollable* agent of the working environment. Thus, it is important to properly handle *temporal uncertainty* and *uncontrollable* events in order to synthesize robust and effective *control strategies*.

The Timeline-based planning formalism

According to (Cialdea Mayer, Orlandini, and Umbrico 2016), a timeline-based planning model is composed by multi-valued state variables, representing the set of features to be controlled over time and specifying causal and temporal constraints characterizing their allowed temporal behaviors. A state variable describes the set of values $v \in V$ the related feature may assume over time with flexible temporal duration. For each value $v \in V$, a transition function $T : V \rightarrow 2^V$ describes the set of values $v \in V$ that may follow v. A controllability function $\gamma(v) = \{c, u\}$ characterizes the controllability property. Namely, if a value $v \in V$ is tagged as controllable, i.e. $\gamma(v) = c$ then the system can decide the actual duration of the value. If a value $v \in V$ is tagged as uncontrollable, i.e. $\gamma(v) = u$, the system cannot decide the duration of the value. The state variables behavior may be further restricted by means of synchronization rules specifying temporal constraints (i.e., Allen's temporal constraints) among different values. Planning with timelines usually entails considering sequence of valued intervals and time flexibility is taken into account by requiring that the durations of valued intervals, called *tokens*, range within given bounds. In this regard, a plan represents a whole set of timelines each of which represents an envelop of possible behaviors that respect the duration constraints. However, a set of flexible timelines do not convey enough information to represent a flexible plan. Thus, plan representation must include also information about the relations that must hold between tokens in order to satisfy the synchronization rules of the planning domain.

According to (Cialdea Mayer, Orlandini, and Umbrico 2016), a timeline-based planning model is composed by *multi-valued state variables* representing the set of features to be controlled over time and specifying causal and temporal constraints that characterized the allowed temporal behaviors. A state variable describes the set of values $v \in V$ the related feature may assume over time together with the related temporal duration bounds. A For each value $v \in V$, a transition function $T : V \rightarrow 2^V$ describes the set of values $v \in V$ that may *follow* v. A *controllability function* $\gamma(v) = \{c, u\}$ specifies the controllability property. Namely, if a value $v \in V$ is tagged as *controllable* i.e., $\gamma(v) = c$, then the system can decide the actual duration of the value. If a value $v \in V$ is tagged as *uncontrollable* i.e., $\gamma(v) = u$, the system cannot decide the duration of the value. The allowed behaviors of state variables may be further restricted by means of *synchronization rules* specifying temporal constraints between different values of different variables.

Thus, while state variables specify *local* rules for the single features of the domain, synchronizations represent *global* rules specifying how different features of the domain must behave together. A formal definition of a synchronization rule is the following:

$$a_0[x_0 = v_0] \rightarrow \exists\, a_1[x_1 = v_1]...a_n[x_n = v_n].\, C$$

where (i) $a_0, ..., a_n$, called *token variables*, denote valued temporal intervals of state variables; (ii) for all $i = 0, ..., n$, x_i is a state variable and v_i is a value of x_i; and (iii) C is a *positive boolean formulae* (PBF) specifying temporal constraints among token variables and where only the token variables $a_0, ..., a_n$ occur. The left-hand part of the synchronization $a_0[x_0 = v_0]$, is called the *trigger* of the rule.

Planning with timelines usually entails considering sequence of valued intervals and time flexibility is taken into account by requiring that the durations of valued intervals, called *tokens*, range within given bounds. In this regard, a plan represents a whole set of timelines each of which represents an envelop of possible behaviors that respect the duration constraints. Specifically, a *timeline* for a state variable x in the temporal horizon H is finite sequence of tokens for x:

$$\begin{aligned} FTL_x = \quad & x^1 = (v_1, [e_1, e'_1], [d_1, d'_1]), \\ & ..., \\ & x^k = (v_n, [e_n, e'_n], [d_n, d'_n]) \end{aligned}$$

where the sequence of values $v_1, ..., v_n$ of the tokens satisfy the transition constraints of the state variable.

However, a set of flexible timelines do not convey enough information to represent a flexible plan. The representation of a plan must include also information about the relations that have to hold between tokens in order to satisfy the synchronization rules of the planning domain.

The Knowledge Engineering Environment

The *Knowledge Engineering ENvironment* (called KEEN) (Orlandini et al. 2014) is one of the software assets developed at ISTC-CNR to support the design and development of timeline-based Planning and Scheduling applications. The KEEN system is built around APSI-TRF (Cesta et al. 2009), a state of the art framework for P&S with timelines, and exploits the UPPAAL-TIGA verification tool[2] to perform Validation and Verification of plans (Bensalem, Havelund, and Orlandini 2014).

To support the knowledge engineering (KE) phase, KEEN is composed by a Domain/Problem Editing and Visualization module, providing user interaction functionality for creating planning domain models through textual and graphical (diagram) editors, and a set of V&V services taking advantage of the results presented in (Cesta et al. 2010; Orlandini et al. 2013). Additionally, plans can be generated by means of a planner in a continuous loop of usage. It also supports plan execution feature to send actual commands to a controlled system and allowing to receive the telemetry from the actual plan execution environment. The idea pursued is that KEEN can be connected to an accurate simulator of the real environment, to a real physical system (e.g., the FOURBYTHREE robotic arm) and be able to monitor the execution phase with visual tools.

The KEEN editing and visualization capabilities have been developed as an Eclipse plugin (see Figure 2), thus providing a graphical interface to model, visualize and analyze the P&S domains. The V&V functionality is based on Timed Game Automata (TGA) model checking and rely on UPPAAL-TIGA as verification engine. As a result, UPPAAL-TIGA constitutes an additional core engine for KEEN. In FOURBYTHREE, the KEEN system has been further developed in order to integrate also the EPSL system as well as support the specific operational requests related to the HRC context.

While KEEN was already capable of supporting the modeling of planning problems such as the one connected to the ALFA Pilot case, additional work was needed to integrate it with the FOURBYTHREE executor. The integration was needed to enable the engineer to see the results of his work in real time: just seconds after making a change to the domain or problem definition he could see the robotic arm move, or, for testing purposes, he could choose to run a version of the FOURBYTHREE executor implementing a simulator of the real robotic arm.

Figure 3 shows the KEEN environment during an editing session of the textual representation of the ALFA domain. At the same time, the FOURBYTHREE executor is being run and its output is visible in the Console window at the bottom. KEEN supports the installation of a number of different executors, or different configurations of the same executor. For instance, one might want to have a FOURBYTHREE executor installation targeted to "release mode" whose configuration instructs the executor to connect to the main robotic arm via ROS. Another instance could instead drive a different, test arm in the developer's room. And a third installation could instead use a simulator through the FOURBYTHREE executor. The developer could then choose different run configurations tied to different installations of the executor according to its needs: probably, while developing he will continuously perform simulation runs of the FOURBYTHREE executor, just to be quickly sure that everything works as expected. Then, every now and then, he could instead test his domain on a real robotic arm located in his room. Finally, before releasing his domain in the production environment, he might test everything in the laboratory where an environment similar to the real production site is set up.

[2] http://www.uppaal.org

The FOURBYTHREE Controller

The *FbT Controller* is the element responsible to actually carry out production processes and to coordinate the robot and the human. The synthesized tasks and the coordination of the human and the robot must follow the operational requirements specified by the *Production Engineer* and encoded into the *domain model* through KEEN. As Figure 1 shows, the controller is composed by the *FbT Task Planner* and the *FbT Plan Executive* both relying on the timeline-based formalisms. The *FbT Task Planner* is responsible for generating the set of tasks needed to perform the production processes according to the desired requirements. In HRC scenarios, it is necessary to guarantee the safety of the human without penalizing the productivity of the factory. The task planner is in charge of finding a tradeoff between performance and safety and therefore there are several features to take into account when synthesizing plans.

The planning model can be characterized according to three different levels of abstraction: (i) the *supervision level*; (ii) the *coordination level*; (iii) the *implementation level*. In the *supervision level*, the task planner has to decide the set of tasks needed to execute the production process by modeling the operational requirements specified by the *Production Engineer*. In the *coordination level*, the task planner has to decide who, between the human and the robot, must perform each task harmonizing the activities of both. In this context, the human and the robot are modeled as *two autonomous agents* capable of executing some types of task. Given a production process, some tasks can be performed only by the human, some tasks can be performed only by the robot and some tasks are *free* to be performed either by the human or by the robot. This choice-point represents the main *branching factor* of the task planning process. It can affect the *quality* of the collaboration and the efficiency of processes. Finally, in the *implementation level*, the task planner has to decide the operations the robot must perform in order to execute the assigned tasks. According to the particular type of collaboration decided at coordination level, the task planner decides the most appropriate *execution modality* of the tasks of the robot in order to preserve the safety of the human.

Figure 4 (automatically generated by KEEN) shows an example of a timeline-based planning model for the collaborative assembly scenario in the ALFA Pilot. The model is hierarchically organized according to the three levels of

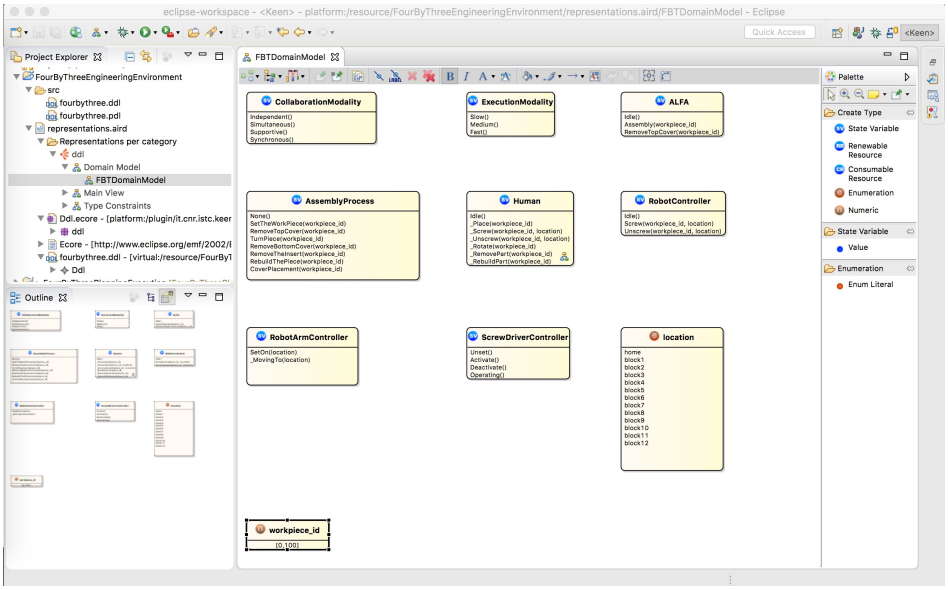


Figure 2: Visual modeling for FOURBYTHREE KEEN environment

abstraction identified (i.e., *supervision*, *coordination*, *implementation*). The *ALFA* and *AssemblyProcess* state variables compose the *supervision level* of the model. These variables characterize the considered production context in terms of tasks that can be executed. The *AssemblyProcess* specifies the set of *high-level tasks* needed to complete the process and the related operational requirements. For example, the *RemoveTopCover* and *RemoveBottomCover* values in *AssemblyProcess* represent high-level tasks modeling part of the assembly/disassembly procedure. Notice that no task assignment is performed at this level of abstraction.

The *Human*, *RobotController* and *CollaborationType* state variables compose the *coordination level* of the model. Specifically, the *Human* and *RobotController* state variables model the *low-level* tasks the human and the robot agents can perform over time. For example, the *Screw* or *Unscrew* values of *Human* and *RobotController* state variables model the capability of both *agents* of performing screwing operations. Instead, *RemovePart* or *Rotate* values of the *Human* state variables model *critical operations* that only the human is allowed to perform. The *CollaborationType* state variable models the possible types of human-robot collaboration within the execution of the tasks of the desired process. The supervision and coordination layers are connected by a set of synchronization rules that specify decomposition constraints of *high-level tasks* in terms of *low-level tasks*. These rules specify how the tasks composing the process can be performed in collaboration by the human and the robot. Namely, these rules describe the possible task assignments between the human and the robot and specify the collaboration modalities suited for human-robot interactions.

The *RobotArmController*, *ScrewDriverController* and *ExecutionModality* state variables constitute the *implementation level* of the model. These variables represent the physical and/or logical elements composing the production environment the system must directly interact with. The *RobotArmController* together with the *ExecutionModality* model the robotic arm. They represent the functional control interface of the robot provided by the integrated *motion planner* (see Figure 1). Specifically, the *RobotArmController* models the motion tasks the robot can perform while, the *ExecutionModality* models the type of trajectory that must be used to perform the motion (see section for further details). The coordination and implementation layers are connected by another set of synchronization rules that specify how the robot must execute the assigned tasks. A particular execution modality of robot motions is selected according to the expected collaboration modality in the coordination layer.

Planning with Temporal Uncertainty

The *FbT Task Planner* leverages PLATINUm and the related plan-refinement procedure which iteratively refines an *initial partial plan* (i.e., a given set of partially constrained timelines) until a *complete* and *valid* plan is found. Plan refinement consists in detecting and solving *flaws* of the plan that represent particular condtions affecting the *completeness* (e.g., a planning goal) or the *validity* (e.g., temporal overlaps of tokens of a same timeline) of the plan. In addition, the solving process iteratively checks the controllability features of the plan by taking into account temporal uncertainty. Specifically, the procedure verifies the *pseudo-controllability property* (Vidal and Fargier 1999) by analyzing the flexible durations of uncontrollable tokens. *Pseudo-controllability* is a necessary but not sufficient condition for *dynamic controllability* which is a desirable property to robustly cope with the *uncontrollable dynamics* in the real-world. Pseudo-controllability guarantees that a plan does not *restrict* the flexible duration of uncontrollable tasks. In this way, generated plans are more flexible and can better deal with the uncontrollable events.

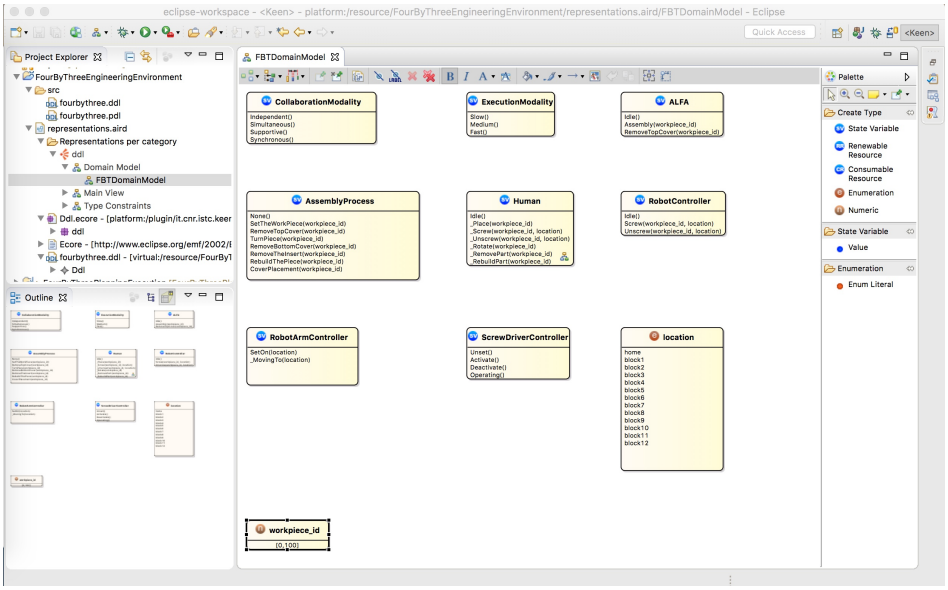


Figure 3: Executing plans through KEEN

The execution process consists of *control cycles* that iteratively execute a flexible (timeline-based) plan by sending commands to the robot and receiving *feedbacks* from the environment until all the tasks of the plan have been executed. A ROS-based middleware provides communication channels that allow the control system to exchange commands and feedbacks with the robotic platform and the human operator. Specifically, human feedbacks are received by several sensor devices like MyO[3] or video cameras that produce signals concerning the activities/operators of a human within the working cell

Control cycles are managed through a *clock* which determines the *frequency* and therefore the responsiveness of the executive. The clock "discretizes" the temporal axis by identifying temporal units called *ticks*. At each *tick* corresponds the execution of a control cycle which is composed by two distinct phases, (i) the *synchronization phase* and (ii) the *dispatching phase*. The *synchronization phase* manages the received execution feedbacks in order to *verify* the consistency of the ongoing plan with respect to the actual state of the working environment. If the plan is valid then the *dispatching phase* selects the next activities to be executed. Algorithm 1 describes the general control procedure of the executive and its related sub-procedures.

The procedure takes a plan Π to be executed and a clock $\mathcal{C}$ as input. The plan Π is analyzed to identify *start* and *end* execution *dependencies* between tokens of the timelines. This information is encapsulated by a dedicated structure π_{exec} (row 3), called *Execution Dependency Graph*. Then, the procedure iteratively executes the plan until all tokens have complete their execution (rows 5-12). The timing of the iterations of the procedure is determined by the clock $\mathcal{C}$ which continuously updates and signals the current execution time

[3]https://www.myo.com/

Algorithm 1 The executive control procedure

```
1:  function EXECUTE(Π, C)
2:      // initialize executive plan database
3:      π_exec ← Setup(Π)
4:      // check if execution is complete
5:      while ¬CanEndExecution(π_exec) do
6:          // wait a clock's signal
7:          τ ← WaitTick(C)
8:          // handle synchronization phase
9:          Synchronize(τ, π_exec)
10:         // handle dispatching phase
11:         Dispatch(τ, π_exec)
12:     end while
13: end function
```

τ (row 7). The *Synchronize* sub-procedure is in charge of handling the *synchronization phase* by managing execution feedbacks (row 9). It checks the validity of the expected status of the world (i.e. the task plan) with respect to the observed status of the world (i.e. the feedbacks). Similarly, the *Dispatch* sub-procedure is in charge of handling the *dispatching phase* by deciding the tokens of the plan that must start their execution (row 11).

The tokens of the plan may have different controllability properties that entail different managements of the related execution dependencies within the control loop. Specifically, it is possible to identify three types of token according to the different *controllability properties* in HRC scenarios. A token can be (i) *controllable*, (ii) *partially-controllable* or (iii) *uncontrollable*. *Controllable tokens* are completely under the control of the executive. The executive can decide the actual start and end times of the token and therefore its actual duration. *Partially-controllable tokens* represent tokens the executive cannot completely control. The executive can only *observe* the actual execution of this type of

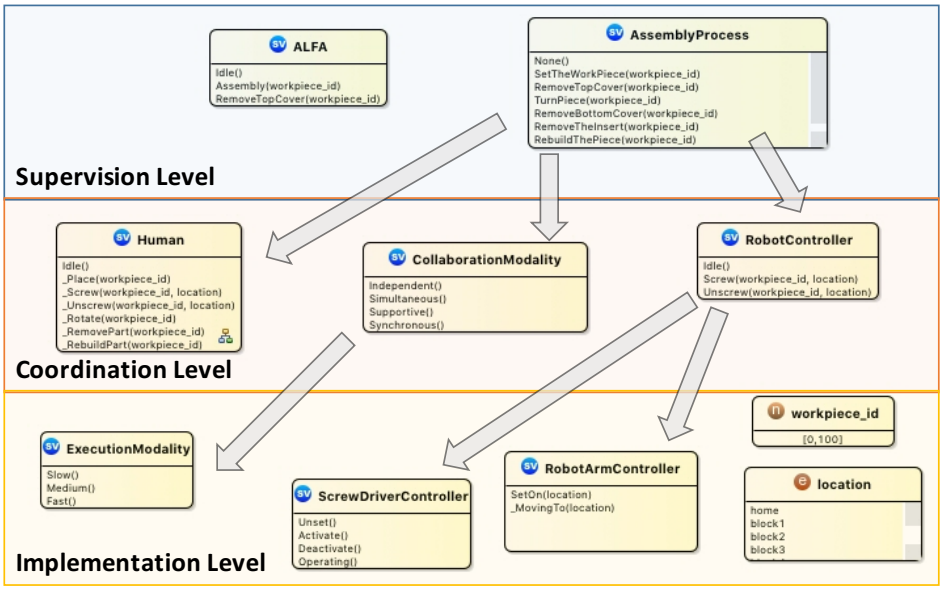


Figure 4: Hierarchical model for collaborative assembly in the ALFA pilot

tokens. Namely, the executive controls only the start of the execution and can assume that execution is ended only if a signal has been received. When the signal is received, the executive verifies the consistency of the plan with respect to the *observation* (i.e., the system verify whether the end conditions and the schedule of the token in the plan comply with the observed behavior). *Uncontrollable tokens* are completely outside the control of the executive. The executive may suppose when the token is about to start according to its schedule, but cannot decide its actual start time. It can only *observe* the start of the execution and check whether the received signal complies with the plan (i.e., whether the start execution dependencies and the schedule of the token are satisfied). Then, similarly to *partially-controllable tokens*, the executive waits a second signal concerning the end of the execution of the token. When the signal is received, again the executive checks the consistency with respect to the plan and the related (end) execution dependencies.

In such a context, it is not always possible to complete the execution without changing the plan. Indeed, temporal uncertainty and *uncontrollability features* of the environment may lead to uncontrollable behaviors the timeline-based plan cannot capture. The control system is forced to generate a new plan in such cases. For example, the execution of a human task may last longer than expected from the model, or it can start later than expected from the plan. If such an event is detected, the controller interrupts the plan execution and enters the *replanning phase*. The executed tokens and the last received observations determine the *initial situation* the task planner starts from to synthesize a new plan. If some uncontrollable tokens were in execution when the failure occurred (e.g., the robot arm was moving between two positions), then the controller, through the *Failure Manager* in Figure 1, waits the related feedbacks in order to "reset" a stable state. When the problem specification is complete, a new plan is generated and plan execution can be resumed.

Task and Motion Planning Integration & Deployment

Final goal of the FOURBYTHREE project is the full integration of technologies. In particular, the task planning technology has been integrated with a motion planning subsystem. The capability of selecting different *execution modalities* of robot tasks according to the expected *collaboration* with the human is the result also of a tight integration of the task planning and executive system with the motion planning approach described in (Pellegrinelli et al. 2016). The motion planning module is also part of the FOURBYTHREE general architecture and it is implemented as an extension of the ROS MoveIT library. Such approach realizes an *offline analysis* of the production scenarios in order to synthesize, for each collaborative task, a number of *robot motion trajectories* (three in the considered scenarios) with different level of *safety*. Each trajectory is associated with an expected temporal execution bound and represents a tradeoff between "speed" of the motion and "safety" of the human. *Slow trajectories* are considered the safest because they tend to move the arm far from the expected position of the human (low probability of collisions). Conversely, *fast trajectories* are considered the less safe because they tend to move the arm close to the expected position of the human (high probability of collisions). The task planner leverages this set of information to characterize the temporal behavior of the robot and coordinate tasks accordingly. During plan execution, the executive *online* communicates to the motion planner the particular trajectory selected to perform the collaborative tasks. The motion planner is in charge of safely executing the selected trajectories by avoiding collisions with the human.

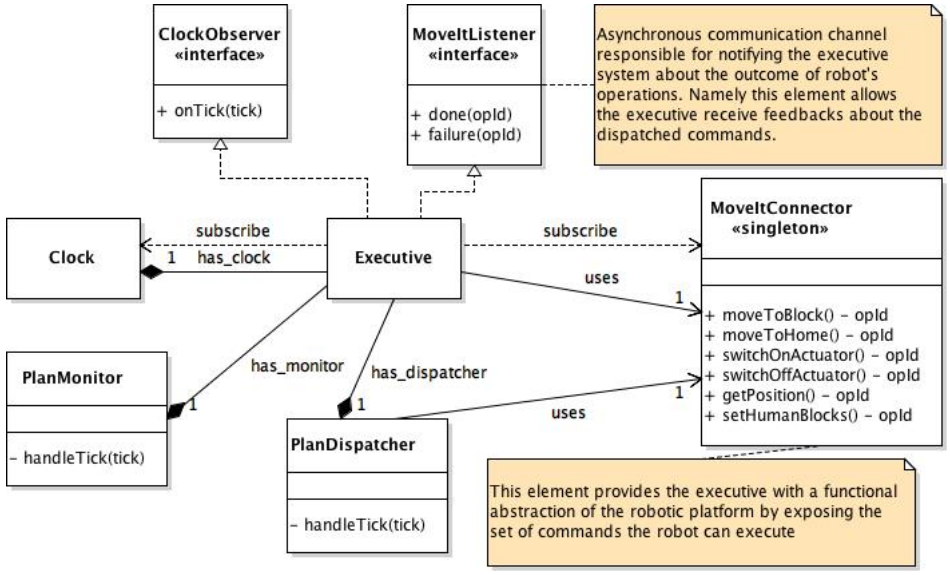


Figure 5: Executive ROS-based integration with the robot motion layer

Figure 5 shows the integration of the *Fbt Plan Executive* with a motion planning module through a ROS-based communication channel. Specifically, the *MoveItConnector* element in Figure 5 shows the *control interface* the executive needs to actually send commands to the robot and the human. The *executeTask(taskId, trajectoryId)* function makes the arm execute the task identified by *taskId* using the trajectory identified by *trajectoryId*. The *switchOnActuator(actuatorId)* and *switchOffActuator(actuatorId)* activates and deactivates respectively the tool (i.e., the screwdriver in the ALFA case study) identified by *actuatorId*. Finally, the *startHumanTask(taskId)* function asks the human to start the task specified by *taskId*. Everytime a task is completed, the executive *asynchronously* receives *feedbacks* through the *MoveItListener*, communicating the successful or unsuccessful execution of tasks. In the case of *startHumanTask*, two notifications are expected. A first notification is sent when the operator starts the assigned task. A second notification is sent when the operator completes the task. The rationale behind this decision is that the human is *uncontrollable* and therefore the system cannot expect a human being to immediately begin to execute an order, as it is the case with the robotic arm.

The integrated control architecture has been deployed and tested in laboratory on a realistic manufacturing scenario (Pellegrinelli et al. 2017) very similar to the assembly/disassembly case study of the ALFA pilot. The experimental evaluation has shown the capability of the proposed approach in realizing a productive and safe collaboration between humans and robots. Experimental results show how the system is able to find a well suited distribution of tasks capable of increasing the productivity of the production process without affecting the safety of the human operator.

Conclusions and Future Works

This paper presented the FOURBYTHREE Engineering & Control Environment, a novel software framework for dynamic task planning and execution for Human-Robot Collaboration aiming at being a robust facilitator in the share of work between humans and robots. The environment provides both *knowledge engineering* features, to support the development of task planning models, and *task planning and execution* capabilities, also able to deal with temporal uncertainty in the presence of human operators. Relevant is the ability to to model collaborative tasks considering a temporal variance of tasks duration, possibly entailed by the presence of a human operator, as well as dynamically adapting task plans according to the actual behavior of the human worker. The system has been validated in laboratory on a realistic industrial case study.

Acknowledgement

Authors have been supported by the European Commission within the H2020 FOURBYTHREE project (GA No. 637095).

References

Barreiro, J.; Boyce, M.; Do, M.; Frank, J.; Iatauro, M.; Kichkaylo, T.; Morris, P.; Ong, J.; Remolina, E.; Smith, T.; and Smith, D. 2012. EUROPA: A Platform for AI Planning, Scheduling, Constraint Programming, and Optimization. In *ICKEPS 2012: the 4th Int. Competition on Knowledge Engineering for Planning and Scheduling*.

Bensalem, S.; Havelund, K.; and Orlandini, A. 2014. Verification and validation meet planning and scheduling. *Software Tools for Technology Transfer* 16(1):1–12.

Cesta, A.; Cortellessa, G.; Fratini, S.; Oddi, A.; and Poli-

cella, N. 2007. An Innovative Product for Space Mission Planning: An A Posteriori Evaluation. In *ICAPS*, 57–64.

Cesta, A.; Cortellessa, G.; Fratini, S.; and Oddi, A. 2009. Developing an End-to-End Planning Application from a Timeline Representation Framework. In *IAAI-09. Proc. of the 21st Innovative Application of Artificial Intelligence Conference, Pasadena, CA, USA*.

Cesta, A.; Finzi, A.; Fratini, S.; Orlandini, A.; and Tronci, E. 2010. Analyzing Flexible Timeline Plan. In *ECAI 2010. Proceedings of the 19th European Conference on Artificial Intelligence*, volume 215. IOS Press.

Cesta, A.; Orlandini, A.; Bernardi, G.; and Umbrico, A. 2016. Towards a planning-based framework for symbiotic human-robot collaboration. In *21th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE.

Chien, S.; Tran, D.; Rabideau, G.; Schaffer, S.; Mandl, D.; and Frye, S. 2010. Timeline-Based Space Operations Scheduling with External Constraints. In *ICAPS-10. Proc. of the 20th Int. Conf. on Automated Planning and Scheduling*.

Cialdea Mayer, M.; Orlandini, A.; and Umbrico, A. 2016. Planning and execution with flexible timelines: a formal account. *Acta Inf.* 53(6-8):649–680.

de Silva, L.; Lallement, R.; and Alami, R. 2015. The hatp hierarchical planner: Formalisation and an initial study of its usability and practicality. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 6465–6472.

Freitag, M., and Hildebrandt, T. 2016. Automatic design of scheduling rules for complex manufacturing systems by multi-objective simulation-based optimization. *CIRP Annals - Manufacturing Technology* 65(1):433 – 436.

Ghallab, M., and Laruelle, H. 1994. Representation and control in ixtet, a temporal planner. In *2nd Int. Conf. on Artificial Intelligence Planning and Scheduling (AIPS)*, 61–67.

Helms, E.; Schraft, R. D.; and Hagele, M. 2002. rob@work: Robot assistant in industrial environments. In *Proceedings. 11th IEEE International Workshop on Robot and Human Interactive Communication*, 399–404.

Jonsson, A.; Morris, P.; Muscettola, N.; Rajan, K.; and Smith, B. 2000. Planning in Interplanetary Space: Theory and Practice. In *AIPS-00. Proceedings of the Fifth Int. Conf. on AI Planning and Scheduling*.

Lemai, S., and Ingrand, F. 2004. Interleaving Temporal Planning and Execution in Robotics Domains. In *AAAI-04*, 617–622.

Marvel, J. A.; Falco, J.; and Marstio, I. 2015. Characterizing task-based human #x2013;robot collaboration safety in manufacturing. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 45(2):260–275.

Maurtua, I.; Pedrocchi, N.; Orlandini, A.; d. G. Fernndez, J.; Vogel, C.; Geenen, A.; Althoefer, K.; and Shafti, A. 2016. Fourbythree: Imagine humans and robots working hand in hand. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, 1–8.

Michalos, G.; Kaltsoukalas, K.; Aivaliotis, P.; Sipsas, P.; Sardelis, A.; and Chryssolouris, G. 2014. Design and simulation of assembly systems with mobile robots. {*CIRP*} *Annals - Manufacturing Technology* 63(1):181 – 184.

Muscettola, N. 1994. HSTS: Integrating Planning and Scheduling. In *Intelligent Scheduling*. Morgan Kauffmann.

Orlandini, A.; Suriano, M.; Cesta, A.; and Finzi, A. 2013. Controller synthesis for safety critical planning. In *IEEE 25th International Conference on Tools with Artificial Intelligence (ICTAI 2013)*, 306–313. IEEE.

Orlandini, A.; Bernardi, G.; Cesta, A.; and Finzi, A. 2014. Planning meets verification and validation in a knowledge engineering environment. *Intelligenza Artificiale* 8(1):87–100.

Pellegrinelli, S.; Pedrocchi, N.; Tosatti, L. M.; Fischer, A.; and Tolio, T. 2014. Multi-robot spot-welding cells: An integrated approach to cell design and motion planning. {*CIRP*} *Annals - Manufacturing Technology* 63(1):17 – 20.

Pellegrinelli, S.; Moro, F. L.; Pedrocchi, N.; Tosatti, L. M.; and Tolio, T. 2016. A probabilistic approach to workspace sharing for human-robot cooperation in assembly tasks. {*CIRP*} *Annals - Manufacturing Technology* 65(1):57 – 60.

Pellegrinelli, S.; Orlandini, A.; Pedrocchi, N.; Umbrico, A.; and Tollio, T. 2017. Motion planning nad scheduling for human and industrial-robot collaboration. *CIRP Annals - Manufacturing Technology.* 66(1):1–4.

Py, F.; Rajan, K.; and McGann, C. 2010. A systematic agent framework for situated autonomous systems. In *AAMAS*, 583–590.

Srivastava, S.; Fang, E.; Riano, L.; Chitnis, R.; Russell, S.; and Abbeel, P. 2014. Combined task and motion planning through an extensible planner-independent interface layer. In *2014 IEEE International Conference on Robotics and Automation*, 639–646.

Umbrico, A.; Cesta, A.; Cialdea Mayer, M.; and Orlandini, A. 2017. Platinum: A new framework for planning and acting. In Esposito, F.; Basili, R.; Ferilli, S.; and Lisi, F. A., eds., *AI*IA 2017 Advances in Artificial Intelligence*, 498–512. Springer.

Umbrico, A.; Cesta, A.; Cialdea Mayer, M.; and Orlandini, A. 2018. Integrating resource management and timeline-based planning. In *The 28th International Conference on Automated Planning and Scheduling (ICAPS)*.

Vidal, T., and Fargier, H. 1999. Handling Contingency in Temporal Constraint Networks: From Consistency To Controllabilities. *JETAI* 11(1).

Wolfe, J.; Marthi, B.; and Russell, S. J. 2010. Combined task and motion planning for mobile manipulation. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010, Toronto, Ontario, Canada, May 12-16, 2010*, 254–258.

Integrating Classical Planning and Real Robots in Industrial and Service Robotics Domains

Oscar Lima, Rodrigo Ventura
Institute for Systems and Robotics
Instituto Superior Tecnico
Av. Rovisco Pais 1, 1049-001 Lisbon, Portugal
Email: {olima, rodrigo.ventura}@isr.tecnico.ulisboa.pt

Iman Awaad
Bonn-Rhein-Sieg University
Grantham-Allee 20, 53575
Sankt Augustin, Germany
Email: iman.awaad@h-brs.de

Abstract

In this paper we propose a case study where we integrate classical planning and real autonomous mobile robots. We start by describing all necessary components to automatically set required facts, generate plans, execute them on real robots to finally monitor their outcome. At the core of our method and to deal with the required complex execution in dynamic environments, we propose to encapsulate the agent high level actions with automatas. We prove the flexibility of the system by testing on two different domains: industrial (Basic Transportation Test) and domestic (General Purpose Service Robot) in the context of the international RoboCup competition. Additionally we benchmark the scalability of the selected planner in two domains on a set of problems with increasing complexity. The proposed framework is open source[1] and can be easily extended.

1 Introduction

Bridging robotics and classical AI planning poses several challenges to both areas. In robotics one faces continuous time, temporal actions, concurrency, in the presence of partial observability, time constraints, stochastic events, to name a few. Classical planning is often formulated as discrete time with instantaneous actions, plans are sequential, and the world is assumed to be both fully observable and deterministic. However, classical planning research is now a mature field, providing a broad range planning methods that the community can benefit from. This paper addresses the problem of integrating such planners into real robots, in particular when such domains are highly unstructured and stochastic. In this paper we present an approach to use a classical planner by integrating it with an execution layer based on finite automata. We have tested the approach in two example realistic scenarios: industrial and service robotics domains.

1.1 Motivation

Artificial intelligence and robotics are two research areas that have benefited significantly from cross-fertilization, however, each of them tend to have their own research agenda with few researchers working on the intersection between them[2]. The International Planning Competition (IPC) and RoboCup (Kitano et al. 1997) are major tournaments for AI planning and robotics both of them which offer a common ground for benchmarking and valuable knowledge sharing. In RoboCup most teams in industrial and domestic domains use finite automata to coordinate the execution of particular skills, such as navigation and manipulation. However, big state machines are hard to maintain, unintuitive for humans to program and hard to reuse. The use of classical planners allows the specification of a goal and domain in an high-level planning domain. However, classical planning algorithms tend to not scale well with the complexity of the domain: it is not reasonable for a robotics domain for a planner to take more than a few seconds to obtain a feasible plan.

1.2 Problem Description

The problem is described as the integration of classical planning and real robots and how to overcome the limitations of classical planning when dealing with unstructured, stochastic, real world domains. We assume the availability of a set of robot skills, namely autonomous navigation, mobile manipulation, perception, and natural language interaction. Given a task, requiring a subset of these skills, we aim at integrating a classical planning into an execution layer of a robot. We focus particularly on the problem of near real-time planning time and robustness to unexpected action effects.

1.3 Structure of this work

In section 2 we start highlighting some of the features that are needed in robotics from the planning community, then we present existing approaches that have merged classical planning and robotics, then we talk about ROSPlan (Cashmore et al. 2015) as one of our dependencies and a close related work, then we briefly describe our selected solver Mercury (Katz and Hoffmann 2014) a well-known planner from IPC 2014, that scored in second place in the deterministic track. In section 4 we briefly describe each of the required components for planning in robotics and how they interact with each other by using a planning coordinator automata. In section 5 we talk about the robots used, their hardware, a description of the industrial and domestic domains that were

[1] www.github.com/oscar-lima/isr_planning

[2] AI summer school 2017
www.lucia.isr.tecnico.ulisboa.pt

modeled and the experiments that we designed to investigate some of the Mercury planner features. Section 6 comments on the scalability and cost assignment results and sections 7 and 8 talk about the conclusions of this work.

2 Related Work

2.1 Planning under time constraints

In the international planning competition typically planners are provided with a 30 min timeout, however in service robotics (or @Home) for instance you can't afford having a robot thinking for more than e.g. 10 seconds, otherwise you deeply affect the human robot interaction process.

Fast approaches like real time (Korf 1990), deadline aware (Burns, Ruml, and Do 2013) or anytime search (Richter and Westphal 2010) are critical for service robotics.

2.2 Classical Planning in Robotics

One of the oldest examples of integration between AI and robotics is the Stanford Research Institute Problem Solver (STRIPS) (Fikes and Nilsson 1971). The famous automated planner was implemented on a real robot "Shakey" (Nilsson 1984). Without a doubt, a milestone in AI planning and robotics.

2.3 RoboCup Logistics League

In RoboCup (Kitano et al. 1997) Logistics league, the system used by the Carologistics (Niemueller et al. 2015) team (winner of 2014-2016) is based on the Fawkes Robot Software Framework (Niemueller, Reuter, and Ferrein 2015). The software stack contains components for localization, navigation, perception and basic behaviors using a Lua-base behavior engine and complete task-level executive based on the C Language Integrated Production System (CLIPS) (Wygant 1989). CLIPS is public domain software built for expert systems. It was initially developed in 1985 at the NASA Jhonson Space Center and presents an object-oriented language for writing expert systems. Like other languages CLIPS deals with rules and fact to operate. One of the main problems with ruled based systems is the amount of rules they require to work efficiently, making them hard to extend and maintain.

2.4 RoboCup at Work League

While most teams in the league use automatas for decision making, the team LUHbots @Work[3] from Hannover, uses a graph-based search algorithm (greedy) and a minimization cost function.

2.5 ROSPlan

Developed at King's College London University KCL ROSPlan (Cashmore et al. 2015) introduces a framework with a generic method for the integration of PDDL planning and the famous Robot Operating System (ROS) which is the standard middleware in robotics. It exposes a Knowledge base (KB), an automatic PDDL problem generator and plan dispatcher to interact with a robot.

Re-planning in ROSPlan is done based upon 3 different criteria: because the dispatcher reports failure on an action, the KB changed in such a way that invalidates the current plan or because the action has consumed to much resources (e.g. time or energy).

2.6 Mercury Planner

Mercury (Katz and Hoffmann 2014) is a sequential, classical, satisficing planner (no optimal solution is guaranteed) which won 2nd place at IPC 2014. The planner starts with an initial greedy best first search and once a solution has being found it performs multiple iterations of heuristic search with a weighted A* algorithm. Mercury uses a partial relaxation delete list heuristic called Red-Black. Red variables take the relaxed semantics (ignore the delete list) while black variables take the regular semantics (will not ignore the delete list) (Katz and Hoffmann 2014). The method used to find black variables is called the paint strategy and is domain dependent. The rationale behind having selected this planner was because it outperformed all other planners in the transport domain (in IPC 2014), which is similar to our industrial domain : basic robot transportation task.

3 Background

The domains were modeled based on the tasks proposed by the international RoboCup competition @Work and @Home leagues and in particular, the Basic Transportation Test (BTT) and General Purpose Service Robot (GPSR).

3.1 Industrial Domain

In the BTT domain, the industrial KUKA YouBot robot (Bischoff, Huggenberger, and Prassler 2011) (see Figure 1), the standard platform of the league, needs to transport objects between locations. The robot has a rear metal platform where a maximum of three objects can be stored and a robotic manipulator with a gripper that can fetch small objects. Additionally a RGBD camera was placed near the end effector to perceive the objects in the environment.

A diagram of the simplified environment is depicted in Figure 2.

From the planning perspective the domain is composed of the following actions: move to location, perceive, pick, place, stage and unstage object. The full domain PDDL definition can be consulted online[4].

3.2 Domestic Domain

The General Purpose Service Robot (GPSR) domain tests the ability of the robots to respond to various commands given from human beings. The robot is provided with a list of locations, items, persons and objects and is expected to execute flexible commands such as: guiding people from source to destination, answer questions, introduce himself, grasp and place objects (transportation) or telling something to someone.

[3] http://luhbots.de/wordpress/

[4] www.github.com/oscar-lima/mercury_planner_experiments

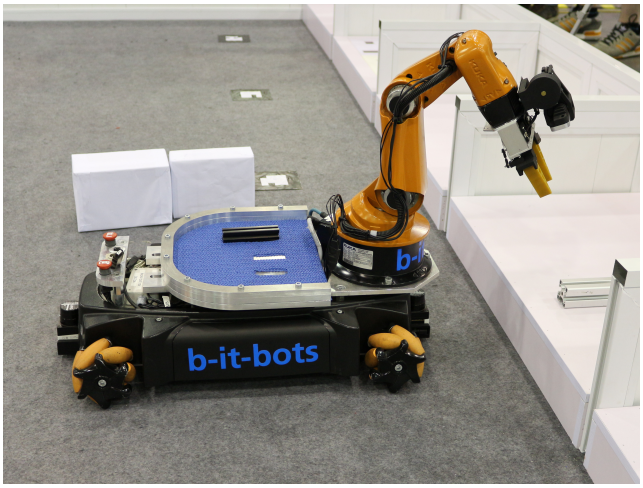


Figure 1: KUKA™ YouBot industrial robot at RoboCup world championship 2015 Hefei, China.

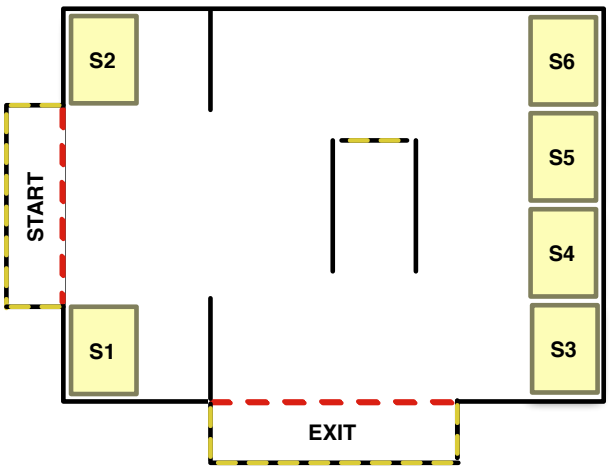


Figure 2: Basic transportation test domain simplified diagram.

The robot that we use for this purpose is the MOnarCH robot (see Figure 3) which is equipped with various sensors and actuators that allow him to interact in the home scenario.

4 System Architecture

Our integration strategy is to use commercial off-the-shelf software while developing custom components we are interested from a research perspective. We briefly describe them in section 4.1.

4.1 Component Description

Speech recognition, natural language understanding and intention to knowledge. In the domestic domain a human needs to interact with the robot, we do this via a 3 step pipeline (see Figure 4). The speech recognition module inputs audio stream from the robot's microphone into the computer and converts it into a sentence. The sentence is

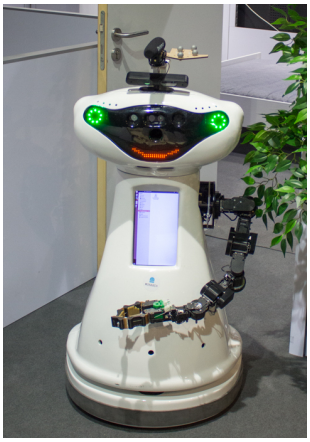


Figure 3: MOnarCH service robot.

input to the natural language understanding component that first divides the sentence into multiple phrases and recognizes the intention and arguments, e.g. sentence: "*go to the kitchen and pick the water bottle*", would get divided into 2 phrases: "*go to the kitchen*" and "*pick the water bottle*", from each phrase we extract the intentions: "*go, grasp*" with the arguments: "*kitchen, water bottle*". Finally we map the intention to knowledge, e.g. (at robot destination), (holding object robot), where destination is kitchen and object is water_bottle.

Knowledge Base. We reuse this components from ROSPlan (Cashmore et al. 2015) to store the instances, facts and goals that are required for the planning process. We define four possible ways to interact with the knowledge base.

1. At startup fixed initial conditions can be uploaded to KB, e.g. A robot is at the entrance and its gripper is empty.
2. A human can provide facts or goals through voice.
3. The dispatcher (based upon success or action failure) can update the KB with the world state.
4. The action automatas modify KB based upon expert knowledge that tries to best fit the current world state based for instance on perception input.

PDDL problem generator. We partially reuse this components from ROSPlan (Cashmore et al. 2015), fetch instances, facts and goals from KB to construct a PDDL problem definition. Only one thing is missing: cost. We have extended their PDDL problem generator component to accept and produce PDDL problems with cost information. The cost information is computed by calling a motion planning algorithm based upon a particular environment, calculating distance between locations and generating a distance matrix that increases the cost function as en effect of the navigation operator. A video showing this process can be seen here[5].

Knowledge base analyzer. This component answers the following questions: 1. are there unfinished goals in the KB? 2. Is there new knowledge in the KB? (w.r.t. the last query).

[5] www.youtube.com/watch?v=VPuuv7F1auw

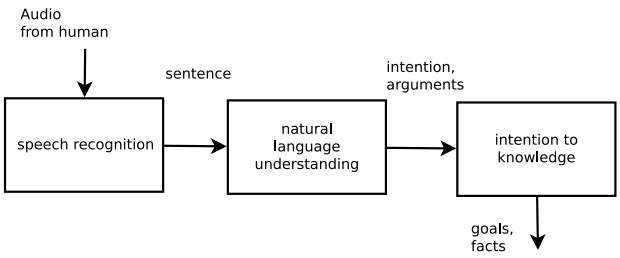


Figure 4: Human voice to knowledge pipeline.

Planner. This component currently calls Mercury (Katz and Hoffmann 2014), however other PDDL planners could be used due to the flexible architecture that we developed.

Plan Validation (VAL). The plan validation tool (Howey, Long, and Fox 2004) inputs the generated plan, the domain model and the problem definition and outputs a boolean response with information whether if the plan solves the imposed goals.

Plan parser. Currently structured to parse IPC formated plans, can be easily modified to adapt to other planners output. However most recent planners available in the community will produce the plan in the correct format.

Scheduler and execution layer. Receives the plan as a sequence of actions to be executed, iterates over each of them and updates the world state based on the action outcome (success or failure). The execution layer is based on high level actions logically structured by the domain model but on the inside each action is coded as an automata. This approach keeps each state machine well factored while maintaining the planner search space in control of the user (depends on how the domain is modeled).

4.2 Planning coordinator

In Figure 5 we present the planning architecture, it shows the interaction between components described in section 4.1.

The planning coordinator implementation itself is also an automata, depicted in Figure 6, notice that the framework allows you to implement your own re-planning strategy by creating your own automata.

The planning coordinator loop starts by uploading intrinsic or basic facts to the KB, then waits until unfinished goals are available in the KB, afterwards it checks if new knowledge is available. This step avoids loops when the planner fails to make a solution, preventing it from continuous failure and waiting until new knowledge arrives, before attempting to solve the problem again.

Next step is to automatically generate a PDDL problem instance from the Knowledge stored in the KB, then the produced plan is validated with the plan validation tool (VAL) (Howey, Long, and Fox 2004) to ensure that the solution solves the goal. Then we parse the planner output and convert it into a vector of actions that is sent for execution to individual action based automatas. Every time an actions is

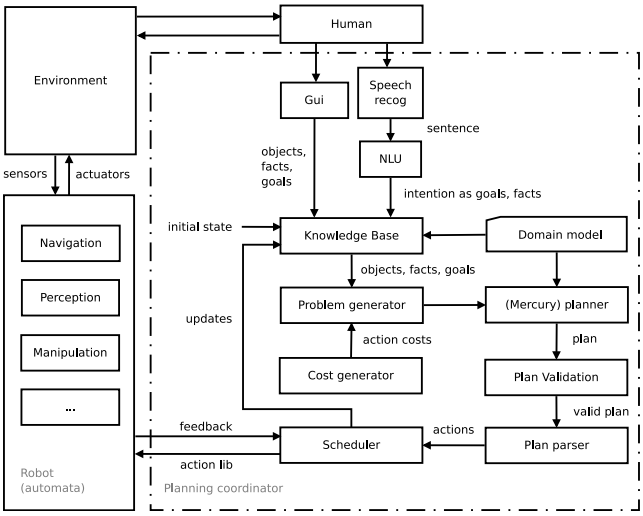


Figure 5: System architecture showing the specifics of the planning framework.

completed the dispatcher updates the KB. If one of the actions reports failure then we trigger re-planning.

4.3 Execution layer : State Machines

Planners are well known to suffer from curse of dimensionality, this usually leads to the advice: keep the domain as simple as possible. In our approach the domain expert has to balance this situation. There is always a trade off between flexibility and planning time.

Behind each high level planning operator there is an automata that deals partially with the complexity of the stochastic domain. In this approach there is a mutual benefit: the planning domain helps to refactor and logically organize each individual automata and they in return help the planner to keep its actions simple enough to be able to find a plan in real time.

The state machine refactoring was guided by the operators within the planning domain. This is one more example of how planning theory can help guide roboticists.

What existed before the integration of planning technology were monolithic state machines that would carry out the tasks seen in the domains. The burden of providing a state machine responsible for the whole plan fell to the developers. The use of automated planning technology enabled us to simply model the domains and create a state machine for each operator leaving the planner to decide which sequence of actions would constitute the plan. This refactoring was thoroughly explained in previous work (Lima 2016).

5 Experiment Setup

We performed three different experiments, in the first one we have participated in two RoboCup international scientific competition (China 2015, Leipzig 2016) to evaluate against other teams which are mostly using automatas (only two @Work teams used planning). Since this work presents results in planning, execution and monitoring on real robots

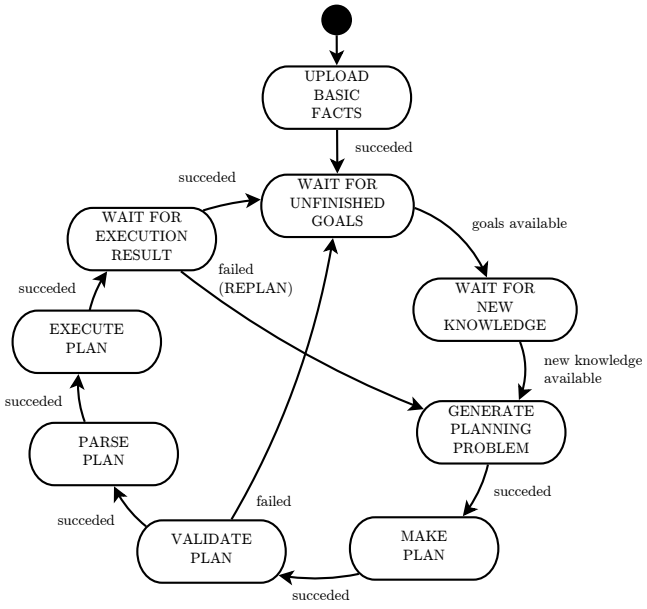


Figure 6: Planning coordinator automata.

we benchmark our robot with this real life experiment.

In the second and third experiment we focused on the planner. The first experiment aims to examine the scalability of the planner. We start giving a small problem to solve and we increase the amount of goals gradually. The experiments were performed on computers with 5 cores and 2-8 GB of RAM (2GB for @Work scenario and 8GB for @Home).

5.1 Mercury planner parameters

The planner has mainly the following search parameters to configure: timeout, cost type, Landmark (lm) cost type and heuristic weight. Since the planner is not anytime the timeout in this case will just basically interrupt the planning process without a solution. The cost type can be NORMAL, ONE and PLUSONE and it refers to the operator cost adjustment type. The lm cost type can be NORMAL, ONE, PLUSONE and it refers to the landmark action cost adjustment. 10 different parameter sets were selected for our experiments based on what the original authors have used in their planning scripts. Table 1 shows the selected parameters that from now on will be refereed as parameter set n.

5.2 Scalability experiment

The problem instance for the @Work scenario requires the robot to transport objects between locations (see Figure 2). We start with one object to be transported and we gradually increase the amount of objects one at a time until 25. A timeout of 1 minute was given to the planner to produce solution for all experiments.

For the @Home scenario we have generated an example problem instance where the robot has to guide a certain amount of people (1-25). Notice that the domain can do many other things (including the transportation of objects) and only one particular operator is being tested in this experiment (guide).

	cost type	lm cost type	w
parameter set 1	1	1	1
parameter set 2	1	1	2
parameter set 3	1	1	3
parameter set 4	1	1	4
parameter set 5	1	1	5
parameter set 6	2	2	1
parameter set 7	2	2	2
parameter set 8	2	2	3
parameter set 9	2	2	4
parameter set 10	2	2	5

Table 1: 10 different search parameters were used for the experiments.

The idea behind the scalability experiments is to investigate how the planning time and plan quality (cost) behave when the problem size grows.

The navigation cost information for the @Work domain obeys a distance matrix generated from an example scenario (see Figure 2) and was calculated by using a motion planning algorithm between all locations. All other actions (perceive, pick, place, stage, unstage) are having unit cost.

The cost information used for the @Home experiment is as follows: navigation action (2), guide action (500), all others (1).

6 Experiment Results

Automated scripts were used to create different solutions to the proposed PDDL problems in a 26 hour experiment run. The planner was asked to create over 40,000 plans.

6.1 Scalability Tests

In Figure 7 we present the results for the scalability experiment, we can see that while planning time grows exponentially the plan length grows linearly. Additionally each planner parameters produces different plan quality and time, e.g. parameter set 1 ”saturates” fast and is unable to handle problems which are bigger than 5 objects to transport for 1 min timeout.

In Figure 8 we present the results from the scalability test in the @Home scenario. We can observe a similar behavior compared to @Work, but this particular domain is more complex, therefore “saturates” earlier.

7 Discussion

We have seen that one possibility to reduce the search space of the planner is to interleave high level actions with the use of automatas. The domain model provides with a logical structure on how such automatas should be factored. Modeling real robot domains can be quite challenging as it requires experience on the domain. Typically roboticists are focused on specific areas, e.g. navigation, manipulation, perception, human-robot interaction, but rarely on all of them.

Modeling such domains is usually an iterating process, you start with a subset of actions and you try to scale up the

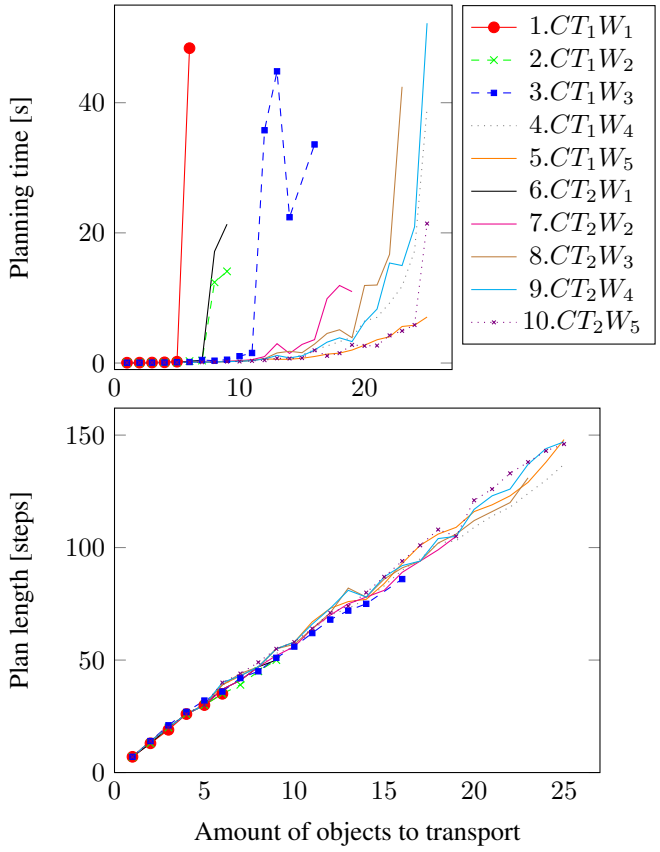


Figure 7: As problem increases in complexity, so does the planning time, 10 different parameters have been used.

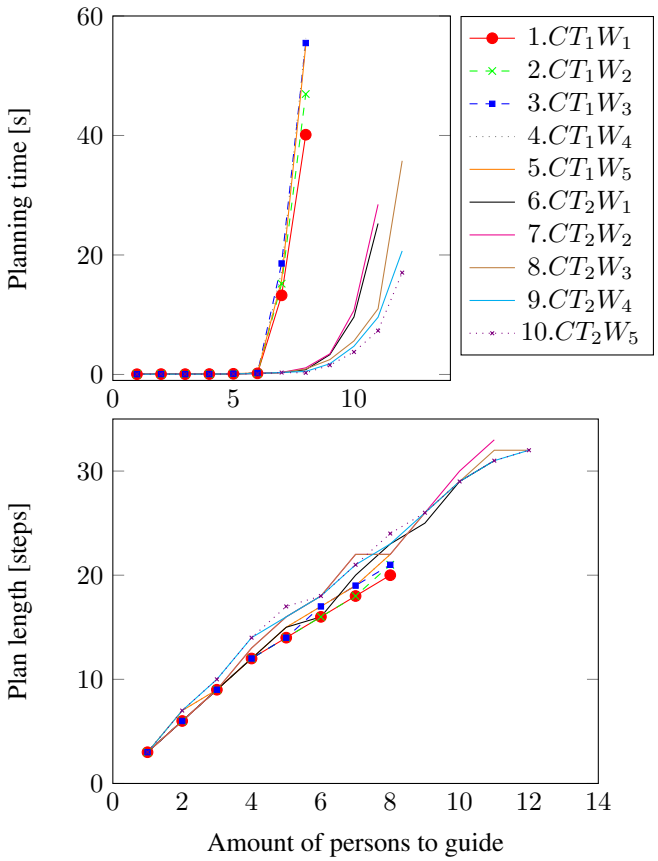


Figure 8: As problem increases in complexity, so does the planning time, 10 different parameters have been used.

domain by adding more actions on each iteration. It is important to notice that task planning software is not mature and is quite experimental, usually you don't get much feedback about syntax mistakes as you would get in programming languages such as c++ or python.

Without the first extension to the problem generator, we would not have been able to use the Mercury planner. The automata-based execution layer allows for complex execution of a plan that was generated with various assumptions to simplify the planning part. The use of refactored action-based automatas in the execution layer helps to balance the various robot skills by offloading the complexity either to the planner or to the automata (design choice).

Although any PDDL planner can be easily integrated in our architecture, we provide with an interface for the IPC 2014 Mercury planner (Katz and Hoffmann 2014). The reason behind it, being it's top performance in the IPC 2014 transport domain which was similar to our industrial @Work domain.

8 Conclusions

In this work which we consider a success story in planning we have presented with a use case where classical task planning and robotics were integrated. It shows the interaction between individual contributions that were published in the ICAPS domain, that can be used to build the overall system out of diverse but compatible components.

The system has proven to be a working solution and led the team to win 3rd place in 2015 and 2nd place in 2016 in the RoboCup @Work international scientific competition (Kitano et al. 1997). With regard to @Home the system is still in experimental phase and has only being tested in local demonstrations in our lab. A video showing the global robot performance can be seen here[6].

The complexity analysis that we have done is helpful to know the scalability of the planner and to balance the complexity between domain operators and the automata. Additionally our benchmark experiments provide an intuition of the problem size that a classical planner can handle.

Both competition domains include non-deterministic outcomes and incomplete information. The execution layer performed as expected and yielded good results[7] for the overall systems performance.

The representation of the execution layer as a state machine is an improvement over other implementations, e.g. ROSPlan, where it is difficult to analyze or change the re-

[6] `www.youtube.com/watch?v=7fvAQVNoKjo`

[7] Our team obtained 3rd place in 2015 and 2nd place in 2016 in the RoboCup @Work international scientific competition.

planning behavior. It is important to highlight that while this work uses some tools from ROSPlan (KB and part of the PDDL problem generator) the framework is entirely our own work. The changes we have made to the ROSPlan problem generator were necessary to enable it to cope with cost information however the system we have created can handle any PDDL planner with or without cost information.

One of the central execution issues is dealt with by coding actions as automatas.

Our experience shows that SMACH state machines meet our requirements and allows us to represent what is necessary to achieve the robustness that we have during the acting phase. This includes the action execution monitoring, handling failures that do not require replanning, updating the KB (based on whether actions were perceived by the robots sensors to have succeeded or failed during the monitoring phase) thereby providing up-to-date information for the replanning process when it is needed.

Contributions of this work are: The integration of various different planning components into real robot systems with two use cases: Industrial and Domestic service robots. Domain models that work in real scenarios. Experiments on a state of the art planner on scalability and cost assignment that provide with valuable guidelines on what problem size can the planner solve (and in which time), and cost assignment strategy, regarding the numerical values that can be set.

This work is relevant for roboticists that want to add planning capabilities to their robot systems.

Acknowledgment

Bonn-Rhein-Sieg University, as I did part of this work during my master thesis work in there.

This work was supported by the FCT project [UID/EEA/50009/2013].

This work was supported by project [PTDC/EEI-SII/4698/2014].

References

Bischoff, R.; Huggenberger, U.; and Prassler, E. 2011. Kuka youbot-a mobile manipulator for research and education. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 1–4. IEEE.

Burns, E.; Ruml, W.; and Do, M. B. 2013. Heuristic search when time matters. *Journal of Artificial Intelligence Research* 47:697–740.

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtós, N.; and Carreras, M. 2015. Rosplan: Planning in the robot operating system. In *ICAPS*, 333–341.

Fikes, R. E., and Nilsson, N. J. 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence* 2(3-4):189–208.

Howey, R.; Long, D.; and Fox, M. 2004. Val: Automatic plan validation, continuous effects and mixed initiative planning using pddl. In *Tools with Artificial Intelligence, 2004. ICTAI 2004. 16th IEEE International Conference on*, 294–301. IEEE.

Katz, M., and Hoffmann, J. 2014. Mercury planner: Pushing the limits of partial delete relaxation. *Proceedings of the 8th International Planning Competition (IPC-2014)*.

Kitano, H.; Asada, M.; Kuniyoshi, Y.; Noda, I.; Osawa, E.; and Matsubara, H. 1997. Robocup: A challenge problem for ai. *AI magazine* 18(1):73.

Korf, R. E. 1990. Real-time heuristic search. *Artificial intelligence* 42(2-3):189–211.

Lima, O. 2016. Task planning, execution and monitoring for mobile manipulators in industrial domains, (master dissertation). retrieved from: https://github.com/oscar-lima/isr_planning/blob/kinetic/oscar_lima_master_thesis.pdf.

Niemueller, T.; Reuter, S.; Ewert, D.; Ferrein, A.; Jeschke, S.; and Lakemeyer, G. 2015. The carologistics approach to cope with the increased complexity and new challenges of the robocup logistics league 2015. In *Robot Soccer World Cup*, 47–59. Springer.

Niemueller, T.; Reuter, S.; and Ferrein, A. 2015. Fawkes for the robocup logistics league. In *Robot Soccer World Cup*, 365–373. Springer.

Nilsson, N. J. 1984. Shakey the robot. Technical report, SRI INTERNATIONAL MENLO PARK CA.

Richter, S., and Westphal, M. 2010. The lama planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research* 39:127–177.

Wygant, R. M. 1989. Clipsa powerful development and delivery expert system tool. *Computers & Industrial Engineering* 17(1-4):546–549.

Interactive Plan Execution during Human-Robot Cooperative Manipulation

Jonathan Cacace, Riccardo Caccavale, Alberto Finzi and Vincenzo Lippiello
PRISMA Lab, Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione,
Università degli Studi di Napoli Federico II, via Claudio 21, 80125, Naples, Italy
e-mail: {jonathan.cacace, riccardo.caccavale, alberto.finzi, vincenzo.lippiello}@unina.it

Abstract

Collaborative robots (*Cobots*) are robotic systems designed to physically interact with humans during task execution in a shared industrial workspace. In this paper, we consider a scenario in which a human operator can be supported by a lightweight robotic arm in order to accomplish complex manipulation tasks. Specifically, we assume that manipulation tasks are explicitly represented as hierarchical task networks to be interactively executed exploiting the human physical guidance. In this setting, the human interventions are continuously interpreted by the robotic system in order to infer whether the human guidance is aligned or not with respect to the planned activities. The interpreted human interventions are then exploited by the robotic system to adapt its cooperative behavior during the execution of the shared plan. Depending on the estimated operator intentions, the robotic system can replan/adjust tasks or motions, while regulating the robot compliance in order to follow or lead the human coworker. The proposed approach is demonstrated in a testing scenario consisting of a human operator that interacts with a Kuka *LBR iiwa* arm in order to perform a cooperative manipulation task. The collected experimental results show the feasibility and the effectiveness of the approach.

Introduction

Collaborative robots (*cobots*) are robotic systems enabling physical human-robot interaction and collaborative task execution in a shared workspace (Colgate et al. 1996). In this setting, the cooperation between the human and the robotic co-worker should not only be safe, but also natural and effective. While robotic platforms suitable for a safe and compliant physical human-robot interaction are wide spreading in service robotics applications (De Santis et al. 2007), the collaborative execution of complex tasks still poses relevant research challenges (Johannsmeier and Haddadin 2017). Indeed, in these setting, the robotic control system should generate and flexibly orchestrate structured plans, taking into account the human intentions and interventions. These issues are particularly evident in industrial service robotics, where the tasks are usually well defined and explicitly formalized (Vernon and Vincze 2016), but their execution needs to be flexibly adapted to the co-workers behaviors in order to ensure a safe and natural human-robot collaboration. In this paper, we tackle this problem considering a scenario in which a human operator can physically interact with a lightweight robotic manipulator in order to accomplish complex collaborative activities represented as hierarchical task networks (HTNs). In this setting, we propose an approach to interactive plan execution based on a continuous interpretation of the human physical guidance. In the proposed framework, the operator physical interventions are assessed with respect to the planned activities and motions in order to estimate the human intentions and targets. These are then exploited by the robotic system in order to suitably adapt its collaborative behavior at different levels of abstraction. Specifically, when the human interventions are aligned with respect to the planned task, these are maintained and the robotic manipulator can proactively guide the user towards the estimated targets. Otherwise, depending on the assessment of the operator aims, the robotic system can change targets and tasks, while regulating the robot compliance in order to follow or lead the human guidance. The overall system has been demonstrated in a testing scenario in which a human operator interacts with a Kuka *LBR iiwa* arm in order to perform a simple assembly task. We assessed the system by comparing its performance with or without the proposed interactive plan execution mechanism. The collected results show the effectiveness of the approach in cooperative task execution and human intention estimation.

Related works

Collaborative task/plan execution is a very relevant topic in the human-robot interaction literature (Karpas et al. 2015; Shah et al. 2011; Clodic et al. 2008; Caccavale and Finzi 2017; Sisbot et al. 2007). In this paper, we focus on physical human-robot interaction during the execution of complex co-manipulation tasks. A framework suited for a similar scenario can be found in (Johannsmeier and Haddadin 2017), where a layered architecture is proposed to enable the execution of hierarchical assembly tasks. In this context, the main focus is on activity coordination and allocation, in contrast, we are interested in a natural and compliant collaboration between the human and the robot during the execution of the shared task, which is a less explored topic in physical human-robot interaction. In particular, we propose a framework that combines the human and the robotic guidance by means of a continuous interpreta-

tion of the human interventions. Estimating the human intentions during the interaction is considered as a very relevant topic in human-robot collaborative execution, indeed it is exploited in many frameworks (Hoffman and Breazeal 2004; 2007). For instance, similarly to our approach, in (Jlassi, Tliba, and Chitour 2014) a shared trajectory generator based on operator force contact is proposed to translate the human intentions into ideal trajectories for the robot. For this purpose, an on-line trajectory generator is combined with a classical impedance control system. In contrast, we propose an integrated framework that combines plan execution and intention estimation to enable a task-based interpretation of the human intentions and targets. Interestingly, in (Peternel et al. 2016) the authors propose to adapt the robot behavior with respect to the operator fatigue during a co-manipulation task, but intentions and targets are not inferred. Less related to the work presented in this paper, in other approaches, human intention estimation is also exploited to increase the efficiency of task planning algorithms (Hoang and Low 2013; Caccavale et al. 2016a). Intention recognition methods typically consider external forces excreted by the human operator on the robot side to regulate the low level control of the robot (Park, Park, and Manocha 2016; Peternel and Babic 2013; Gribovskaya, Kheddar, and Billard 2011; Li et al. 2015). In contrast, in our framework the assessed human intentions are exploited at different levels of abstraction. Indeed, not only they are used to adapt the robot role (from active to passive and vice versa) and compliance during the co-manipulation, but also to suitably modify the execution of a cooperative task when novel intended targets are estimated from the human physical interventions. Our approach to task/trajectory deviation can be related to the one proposed by (Cacace, Finzi, and Lippiello 2014) for shared teleoperation of an aerial vehicles, which is here suitably adapted to physical collaborative manipulation. Human motion estimation is also deployed in (Ge, Li, and He 2011), where the authors employ Neural Networks to extract human motion parameters and predict whether the human interventions are active or passive; in contrast, we assess human intentions and possible targets with respect to the activities of a cooperative task.

System Architecture

The proposed control architecture is illustrated in Figure 1. The *High-level control system* is responsible for task planning/replanning, plan monitoring, and plan execution, while the *Adaptive shared controller* module manages the robot motion according to the selected target and the human guidance. We assume that this controller can directly provide positions and orientations of the manipulator end effector, delegating inner control loops to solve the associated inverse kinematics. In addition, we assume that robotic system can directly estimate the forces F_t acting on the manipulator end effector and externally provided by the operator, who perceives an associated force feedback F_{ext}. These forces are continuously monitored during physical human-robot interaction in order to interpret the human guidance in the context of to the current plan. For this purpose, the *Operator Intention Estimation* module assesses whether the human interventions are aligned or not with respect to the robot activities and targets. This module is exploited by the *High-level Control System* to interact with the *Adaptive Shared Controller* in order to suitably estimate the robot targets and adapt its compliant behaviour during the execution of the cooperative task. Specifically, the *H-L Control System* selects and monitors the plan and the associated target points, while the selected target point X_t is provided to the *Adaptive Shared Controller* along with an associated control mode b. The *Adaptive Shared Controller* module not only generates and monitors the motion data X_d for each target X_t, but also regulates the robot behavior according to the operator guidance. For this purpose, we exploit an admittance controller that enables the robot to dynamically combine the forces applied to the robot and its displacement from its desired position (Hogan 1984). Finally, the end effector trajectories are converted into joint values q to be executed by the manipulator controller. These modules are further detailed in the following sections.

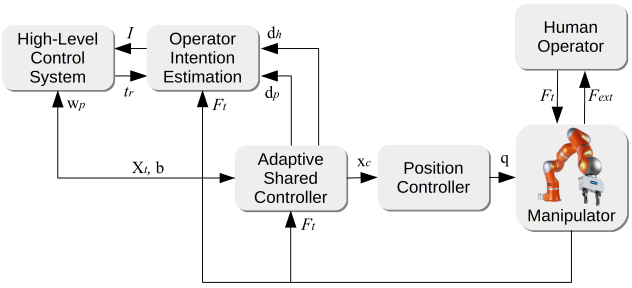


Figure 1: The overall architecture integrates an *High-Level Control System* that interacts with the *Operator Intention Estimation* module providing the estimated intention and target to the *Adaptive Shared Controller* and receiving the reached target points. The *Human Operator* can guide the task execution by physically interacting with the manipulator.

In this section, we describe the *High Level Control System* that integrates plan generation, plan monitoring, and execution. The overall system architecture is depicted in Figure 2. The proposed framework relies on an *Executive System* capable of continuously monitoring and orchestrating multiple hierarchical tasks, in order to adapt in real time plan selection/execution with respect to the recognized human intention. The *Executive System* can exploit a hierarchical *Task Planner* for plan generation and replanning, while a *Target Selector* is introduced to interpret the human guidance with respect to the current tasks providing targets and control modes for the *Adaptive Shared Control*. The proposed executive framework is inspired by the one proposed by (Caccavale et al. 2016b; Caccavale and Finzi 2017). It is based on a control cycle that involves an internal structure, called *Working Memory* (WM) and a plan library, called *Long Term Memory* (LTM). The LTM is a repository that contains a declarative representations of the tasks and the actions the robotic system can perform. In order to be executed, a task is to be allocated, hierarchically decomposed, and instantiated into the WM. This structure represents the executive state of the system and collects the set of activities currently

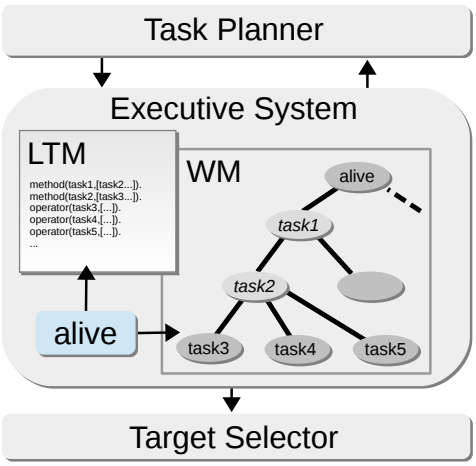


Figure 2: The *High Level Control System* relies on an *Executive System* that interacts with a *Task Planner* in order to expand and instantiate hierarchical tasks. Hierarchical task definitions are represented in the *Long Term Memory* (LTM), while the *Working Memory* (WM) collects the set of tasks currently under execution .

under execution, including both abstract and primitive tasks, representing, respectively, structured activities and concrete sensorimotor processes.

Task Representation

Each task is hierarchically represented in the LTM by means of methods and primitives actions. We deploy a representational framework, which is inspired by the one proposed for Hierarchical Task Networks (HTNs) (Nau et al. 2003). A method is represented by the quadruple (h, C, T, E), where h is a compound task (head of the clause), C are the preconditions, $T = (t_1, \ldots, t_n)$ is the list of subtasks, while E is the set of post-conditions. A primitive action is represented as a quadruple (h, A, D, E) where h is a primitive task, A and D are, respectively, the add and the delete lists of a STRIPS-like representation, while E is a postcondition. Notice that, in this task formulation, postconditions of methods and actions are introduced in order to enable plan monitor during plan execution; by neglecting postconditions, the proposed encoding can be also exploited for plan generation by a HTN-based planning process as in (Nau et al. 2003).

Working Memory

The WM collects the abstract and concrete processes allocated and instantiated for the execution. In our framework, these processes are represented by an annotated rooted tree $T = (r, B, E)$, whose nodes in B represent allocated processes, the root $r \in B$ is the process that bootstraps and manages the WM, while the edges E represent parental relations among sub-processes. These nodes can be either *concrete*, representing real sensorimotor processes, or *abstract*, which are for instantiated methods to be hierarchically decomposed according their definition in the LTM. Each node in WM is represented by a 5-tuple (m, q, p, x, μ), where m is the name of the allocated task, q and p represent the instantiated precondition and postcondition respectively, x is the set of sub-behaviors generated by m, while μ is an activation value for that node (which is not used in this work). Notice that multiple tasks can be allocated and executed in the WM. A node in the WM is considered enabled if all the preconditions along his branch are satisfied. A special process (*alive* in Figure 2) is the root of the WM tree and manages its updates by allocating and deallocating tasks.

Plan Execution

In this work, we focus on plan execution, therefore, we assume that the system is to execute a task already decomposed by a hierarchical planner. In particular, we assume that a task t is associated with set of alternatives $P_1, \ldots, P_k$ each representing a possible executable plan generated by a HTN planner. We also assume that each plan P_i is represented by a suitable node allocated in the WM and connected with a hierarchical structure H_i that contains the ground instances (either methods and operators) of the task network generated during the planning process. Each plan node is also associated with a process responsible for plan monitoring.

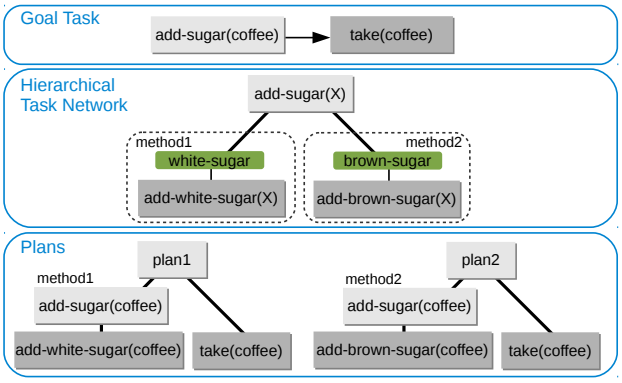


Figure 3: Hierarchical representation of plans. Starting from a list of goal tasks (upper-level), the planner exploits the HTN representation of the tasks (middle-level) in order to produce two plan sequences along with their hierarchy (lower-level). Notice that light and dark gray boxes stand for compound and primitive tasks respectively, while green boxes are preconditions.

An example of task decomposition is depicted in Figure 3. The goal tasks is composed of *add-sugar(coffee)* and *take(coffee)*. In this case, *add-sugar(coffee)* is a compound task that can be performed in two ways (methods) by adding either brown or white sugar to the coffee; instead, *take(coffee)* is already a primitive task, which can be directly executed with no further decompositions. If both types of sugar are available (brown and white) in the initial state, two plans can be generated (Figure 3, plans) and allocated in the WM for the execution. In Figure 4, we show how the two plans of Figure 3 are allocated in the WM. In this case, primitives and methods provide the following task-specific postconditions: the add-sugar tasks are associated with the *sugar.added* postcondition while *take(coffee)* is associated

with *coffee.taken*. Once allocated in the WM, the execution of one of these two plans is then decided at run-time depending on the WM updates due to the environmental changes and the operator guidance. In this setting, when multiple conflicting activities are enabled, the human operator guidance can be exploited to implicitly overcome the produced impasse pointing the system towards the desired target. During the interaction, the WM maintains the hierarchical structure of the allocated plans. It keeps track of the current state, including active tasks, sub-tasks, and the concrete sensorimotor processes that allow to continuously monitor both the environment and the human interventions. In this setting, in order to assess which task/action is accomplished among the active ones, we provide each method/primitive action with a specific *postcondition*. When a postcondition is satisfied the associated task is considered as accomplished, hence disabled.

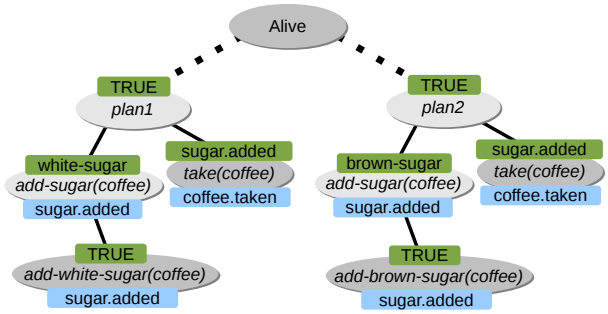


Figure 4: Representation of plans in WM. Light and dark gray ovals are for compound and primitive tasks. Each oval contains the ground instance of the related task, while green and blue boxes are preconditions and postconditions provided by the planned methods/actions. The root of the hierarchy, the *alive* node, is associated with the behavior that manages the WM.

Integrating Robot and Human Guidance

The human interactive physical interventions are continuously interpreted in order to estimate the associated intention and to accordingly adjust the robot collaborative behavior at different levels of abstraction: trajectories, targets, and tasks. The interpretation of the human intention is obtained by the interaction of the *High-level Control System* and the *Operator Intention Estimation* modules (see Fig. 1). The first one proposes possible targets for robotic manipulator depending on the planned and executed activities; each possible target is then evaluated by the *Operator Intention Estimation* with respect to the current human physical guidance. The interpreted targets are then provided to the *Target Selector*, whose outcome is sent to the *Adaptive Shared Controller*, which is to accordingly adapt the robot behavior. In particular, when the human guidance is coherent with respect to the planned task, hence a human-robot shared target clearly emerges, the *Adaptive Shared Controller* provides a robotic behavior which is compliant with the human action. Otherwise, if the assessed intention is misaligned with respect to the planned targets, or the current target is ambiguous (see *Target Selector* in the next section), the *Adaptive Shared Controller* switches in a passive mode in order to enable the human to easily guide the manipulator end-effector.

Intention Classification

Given a target (and the trajectory to reach it), we classify the human interventions into four possible main intentions. In a first case, the user guidance is coherent with both the target and trajectory (*Coinciding*). Instead, in a different scenario, the human aims at adjusting the robot motion (e.g. in order to avoid an obstacle) without changing the target point (*Deviating*). We also consider two cases in which the human intention is to contrast the robot motion (e.g. to stop or suspend the execution) (*Opposite*) or to switch towards a different task/target (*Opposite Deviating*). In our framework, intention classification is based on a three layered feed-forward Neural Network that classifies the aim of the human physical interventions from three input data: the magnitude of the contact forces $||F_t||$ provided by the operator on the manipulator end effector; the distance d_h between the current position of the end effector X_c and the closest point to the planned trajectory C_p, i.e. $||X_c - C_p||$; the deviation d_p between the planned and human motions, calculated as the angle between the two movement vectors, i.e $\angle(\overrightarrow{d_c}, \overrightarrow{d_d})$. The outcome of the network are the 4 classes described above (*Coinciding*, *Deviating*, *Opposite*, *Opposite Deviating*) associated with 4 nodes. The middle layer is composed of 24 nodes. The network has been trained by involving 10 testers (students and researchers), each physically interacting with the robotic arm for 10 minutes in order to obtain data about all the *intentions classes*. After this training phase, we tested again the classified with another group of 10 testers, obtaining an accuracy of about 92%, 70%, 82%, 72% for *Coinciding*, *Deviating*, *Opposite*, *Opposite Deviating*, respectively.

Target Selection

We now illustrate how the plans allocated in the WM and the human interventions are continuously monitored and integrated in order to define the target positions and the control modes for the *Adaptive Shared Controller*. As already explained above, the multiple plans allocated in the WM are decomposed into a set of concrete sensorimotor processes, each associated with a primitive operator. The execution cycle of a generic process is illustrated in Algorithm 1. For each allocated primitive task, if enabled, i.e. all the preconditions along the branch are satisfied (line 2), the estimated human intention (line 3) and the task are sent to the *Target Selector*, which is to produce the inputs for the *Adaptive Shared Control*. When the postconditions E are satisfied (line 5), the operator is assumed to be executed, hence the add and delete lists are exploited to check the state and accordingly update it (line 6). Finally, h is deallocated from the WM (along with all the ancestors with the same postconditions). For each time stamp, the enabled primitive tasks produce a list of couples (h, I_h) that is exploited by the *Target Selector* in order to define the target position X_t for the robot along with the interaction mode. From this list we can

Algorithm 1 Execution cycle of a monitoring-behavior associated with the operator (h, A, D, E).

```
1: while task h is allocated in WM do
2:   if h preconditions are satisfied then
3:     assess the human intention I_h for h
4:     send (h, I_h) to the Target Selector
5:     if E postcondition is satisfied then
6:       check and update the state according to A and D
7:       remove h from WM
8:     end if
9:   end if
10: end while
```

also extract the possible targets in the current executive state. For this purpose, we assume that each primitive task h, that determines a robot motion, is directly associated to a target position X_h for that motion. Target selection works as follows. We introduce an ordering on the intentions such that $Coinciding > Deviating > Opposite$, $> Opposite\ Deviating$ which induces a partial ordering on the set of targets (i.e. $X_{h_1} > X_{h_2}$ if $I_{h_1} > I_{h_2}$). Whenever only one target is enabled with the intention assessed as $Coinciding$ or $Deviating$, that target is selected. When multiple targets are enabled, if their ordering determines a unique best target (neither $Opposite$ or $Opposite\ Deviating$), this is selected. Otherwise, no target is selected. The *Target Selector* couples each target with an operation mode that coincides with the estimated intention in the case of $Coinciding$ or $Deviating$, otherwise, when no target is defined, the operation mode is set as $Passive$ leaving the lead to the human operator until a unique target is again available.

Adaptive Shared Controller

The *Adaptive Shared Controller* receives target positions $X_t = (x_t, y_t, z_t)$ along with the operation mode ($Coinciding$, $Deviating$, $Passive$) from the *High-Level Control System* in order to guide the manipulator towards that target in cooperation with the human exerting a force F_t on the end effector. The *Adaptive shared controller* is to generate the motion data X_d needed to reach the target X_t. In order to reach the target position, the controller generates a velocity reference $\dot{X}_d$ for each time stamp i, as follows:

$$\ddot{X}_{d_i} = \omega^2 e_p - 2\zeta \dot{X}_{d_{i-1}} \tag{1}$$

$$\dot{X}_{d_i} = \dot{X}_{d_{i-1}} + \ddot{X}_{d_i}\tau \tag{2}$$

where ω and ζ are gains representing frequency and damping of the system, τ is the sampling interval of the controller, while $e_p = (X_t - X_c)$ is the distance of the manipulator (X_c) from the target position (X_t). The velocity in Eq. 2 is then integrated to reach the desired position of the robot, that is:

$$X_{d_i} = X_{d_{i-1}} + \dot{X}_{d_{i+1}}\tau \tag{3}$$

Since the manipulator should be adaptive with respect to the operator physical guidance, we exploit an admittance controller, which is typically described by the second-order equation:

$$m\ddot{x} + d\dot{x} + kx = F \tag{4}$$

that in our case can be specialized as follows:

$$\ddot{X}_{c_{i+1}} = \frac{M\ddot{X}_{d_i} + D(\dot{X}_{d_i} - \dot{X}_{c_i}) + K(X_{d_i} - X_{c_i}) + F_t}{M}. \tag{5}$$

with M, D and K representing, respectively, the desired virtual inertia, the virtual damping and the virtual stiffness. The output of this module is the instant compliant position X_c, representing the control command for the *Position-Controlled System*. Depending on the estimated target and the human intention, the robotic manipulator may set a *passive* or an *active* mode. In the first case, the manipulator is fully compliant to the operator interaction without providing any contribution to the task execution. Instead, in the second case, the robot is to protectively assist the operator in reaching the execution of the cooperative task. In our framework, the system can switch from a passive to an active mode by removing the virtual stiffness from Eq. 5 and by setting to zero the desired acceleration and velocity. Instead, when the target is associated with a *Coinciding* or *Deviating* mode, the virtual stiffness is set to a value higher than zero. In particular, when the operator intention is interpreted as *Coinciding* the planned target point and the motion trajectories are maintained, along with the admittance parameters for cooperative manipulation. Instead, when the operation mode is *Deviating*, a more docile behavior for the robot is needed. For this purpose, we set different admittance parameters that enable the human to deviate, but can be also guided back towards the planned target. In order to achieve this effect, while the operation mode is $Deviating$, the *Adaptive Shared Controller* not only sets different admittance parameters, but also generates intermediate target points between the final target position X_t and the closest point to the planned path C_p in order to guide the user towards the planned trajectory. This intermediate target is updated until the operative mode changes. When the manipulator is guided back to the planned trajectory a *Coinciding* mode is activated; otherwise, the system switches to the $Passive$ mode, following the human guidance. It is worth noticing that, similarly to (Cacace, Finzi, and Lippiello 2014), as a side effect of the robot compliant behavior, the operator receives also a force feedback from the robotic manipulator that provides a haptic perception the displacement between the current robot state and the planned one.

Case Study

In this section, we describe a pilot test to assess the system capability of suitably executing planned activities under the human physical guidance. For this purpose, we introduce a simple assembly task where a human co-worker is to guide the robotic manipulator in order to built a small pyramid of objects. The experimental workspace is illustrated in Figure 6. In our set-up we have three colored blocks (two white and one red) that have to be stack in order to create a pyramid on the central support: the two white blocks should be used as the base of the pyramid, while the red one as the vertex (see Figure 6). This task can be executed in two ways, depending on which white block is picked at first. In this experiment, both these alternatives are considered by allocat-

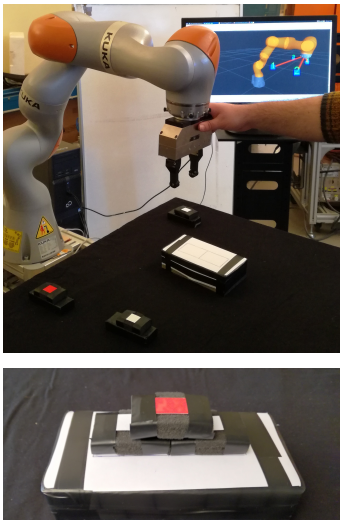


Figure 5: Experimental setup for the assembly-task: it comprises three colored blocks and a support (up). The blocks have to be composed on the support to create a pyramid (down).

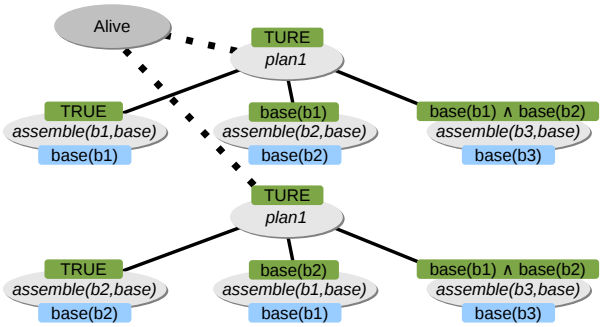


Figure 6: WM representation of the assembly-task: the robot is to cooperate with the human operator following two possible plans.

ing two alternative plans in WM (Figure 6), while the actual plan/action selection process depends on the users physical guidance. In these tests, we exploit as robotic manipulator a KUKA LWR IV+, equipped with a WSG50 2-fingers gripper. The workspace is a table of 50×70 cm. We involved 3 users in the experiments. Notice that in co-bots industrial scenarios the users are supposed to be expert and trained, hence the testers are students with robotics background provided with an informal description of the system features and the task. Each user performed 4 executions of the task in two different modalities, enabling and disabling the plan guidance. In the second case, we assume that all the possible targets points are always enabled (i.e. Target Selector does not filter out targets), hence not selected exploiting the plan. Our aim here is to test whether and how the proposed plan guidance supports cooperative task execution and enhances the accuracy of intention estimation.

Table 1 compares the system and the human performance

	Plan	no-Plan
Accuracy	0.986	0.639
Errors (avg $\pm$ std)	0.167 ± 0.408	3.332 ± 3.724
Times (avg $\pm$ std)	82"$\pm$14"	145"$\pm$38"
Speed-up	16.1% - 59.6%	

Table 1: Systems performance on assembly-task.

during the assembly task, with or without the plan guidance support. As for the system performance, in the upper part we report the accuracy of the intention and target recognition along with the average number of miss-recognitions per execution, with or without the plan. We can observe that the plan guidance clearly improves the system ability to recognize the human intentions and targets during the interaction. Here, as expected, the planned activities propose a reduced set of possible targets, filtered by the plan, which can be better evaluated by the *Operator Intention Estimation*. This way, the plan guidance is more effective, as we can observe in the in the lower part of Table 1 that illustrates the minimum and maximum improvement (speed-up) in terms of time to accomplish the task in the two testing modes. For each tester, this improvement is calculated as $(time_{np} - time_p)/time_{np}$, where $time_{np}$ and $time_p$ are for the time to accomplish the task with or without the plan guidance.

Conclusions and Future works

We presented a framework that integrates interactive plan execution and physical human-robot interaction in order to enable the execution of complex co-manipulation tasks. In the proposed approach, we assume that system is endowed with hierarchically represented tasks that can be executed exploiting the human physical guidance. In contrast with alternative approaches to physical human-robot interaction, in the proposed framework the operator physical guidance is interpreted in the context of a structured collaborative task. In this setting, during the interactive manipulation, the user interventions are continuously assessed with respect to the possible alternative tasks/activities proposed by the plan, in order to infer intentions and targets. These are then exploited both for task selection and to on-line regulate the robotic compliance with respect to the human interactive behavior. We described the overall architecture detailing the plan execution framework, the intention estimation system, the target selection mechanism, and the adaptive shared control system. The proposed framework has been demonstrated in a real world testing scenario in which a user interacts with a lightweight manipulator in order to accomplish a simple assembly tasks. In this context, we compared the performance of the system with or without the support of the plan guidance. The collected results suggest that the proposed approach effectively exploits the plan structure to enhance both intention/target estimation and cooperative task execution. These results encourage us to investigate the system behavior in more complex co-manipulation settings assessing additional parameters from the human side (e.g. human fa-

tigue, cognitive workload, situation awareness, etc.). We are also interested in the enhancement of the proposed plan execution framework by introducing additional constructs and constraints.

References

Cacace, J.; Finzi, A.; and Lippiello, V. 2014. A mixed-initiative control system for an aerial service vehicle supported by force feedback. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1230–1235.

Caccavale, R., and Finzi, A. 2017. Flexible task execution and attentional regulations in human-robot interaction. *IEEE Transactions on Cognitive and Developmental Systems* 9(1):68–79.

Caccavale, R.; Cacace, J.; Fiore, M.; Alami, R.; and Finzi, A. 2016a. Attentional supervision of human-robot collaborative plans. In *25th IEEE International Symposium on Robot and Human Interactive Communication, RO-MAN 2016*.

Caccavale, R.; Cacace, J.; Fiore, M.; Alami, R.; and Finzi, A. 2016b. Attentional supervision of human-robot collaborative plans. In *Robot and Human Interactive Communication (RO-MAN), 2016 25th IEEE International Symposium on*, 867–873. IEEE.

Clodic, A.; Cao, H.; Alili, S.; Montreuil, V.; Alami, R.; and Chatila, R. 2008. SHARY: A supervision system adapted to human-robot interaction. In *ISER*, volume 54 of *Springer Tracts in Advanced Robotics*, 229–238. Springer.

Colgate, J. E.; Edward, J.; Peshkin, M. A.; and Wannasuphoprasit, W. 1996. Cobots: Robots for collaboration with human operators. In *Proceedings of the ASME Dynamic Systems and Control Division*, 433–439.

De Santis, A.; Siciliano, B.; Luca, A.; and Bicchi, A. 2007. An atlas of physical human-robot interaction. *Mechanism and Machine Theory* 43(3):253–270.

Ge, S. S.; Li, Y.; and He, H. 2011. Neural-network-based human intention estimation for physical human-robot interaction. In *2011 8th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, 390–395.

Gribovskaya, E.; Kheddar, A.; and Billard, A. 2011. Motion learning and adaptive impedance for robot control during physical interaction with humans. In *2011 IEEE International Conference on Robotics and Automation*, 4326–4332.

Hoang, T. N., and Low, K. H. 2013. Interactive POMDP lite: Towards practical planning to predict and exploit intentions for interacting with self-interested agents. *CoRR* abs/1304.5159.

Hoffman, G., and Breazeal, C. 2004. Collaboration in human-robot teams. *Proceeding of the AIAA 1st Intelligent Systems Technical Conference* 1.

Hoffman, G., and Breazeal, C. 2007. Effects of anticipatory action on human-robot teamwork efficiency, fluency, and perception of team. *Proceeding of the ACM/IEEE international conference on Human-robot interaction - HRI '07* 1.

Hogan, N. 1984. Impedance Control: An Approach to Manipulation. *IEEE American Control Conference* 304–313.

Jlassi, S.; Tliba, S.; and Chitour, Y. 2014. An Online Trajectory generator-Based Impedance control for comanipulation tasks. *IEEE Haptics Symposium, HAPTICS* 391–396.

Johannsmeier, L., and Haddadin, S. 2017. A Hierarchical Human-Robot Interaction-Planning Framework for Task Allocation in Collaborative Industrial Assembly Processes. *IEEE Robotics and Automation Letters* 2(1):41–48.

Karpas, E.; Levine, S. J.; Yu, P.; and Williams, B. C. 2015. Robust execution of plans for human-robot teams. In *ICAPS*, 342–346. AAAI Press.

Li, Y.; Tee, K. P.; Chan, W. L.; Yan, R.; Chua, Y.; and Limbu, D. K. 2015. Continuous Role Adaptation for Human Robot Shared Control. *IEEE Transactions on Robotics* 31(3):672–681.

Nau, D. S.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. Shop2: An htn planning system. *Journal of artificial intelligence research* 20:379–404.

Park, J. S.; Park, C.; and Manocha, D. 2016. Intention-aware motion planning using learning based human motion prediction. *CoRR* abs/1608.04837.

Peternel, L., and Babic, J. 2013. Learning of compliant human-robot interaction using full-body haptic interface. *Advanced Robotics* 27:1003–1012.

Peternel, L.; Tsagarakis, N.; Caldwell, D.; and Ajoudani, A. 2016. Adaptation of robot physical behaviour to human fatigue in human-robot co-manipulation. *IEEE-RAS International Conference on Humanoid Robots* 489–494.

Shah, J.; Wiken, J.; Williams, B.; and Breazeal, C. 2011. Improved human-robot team performance using chaski, a human-inspired plan execution system. In *Proceedings of the 6th International Conference on Human-robot Interaction*, HRI '11, 29–36. New York, NY, USA: ACM.

Sisbot, E. A.; Marin-Urias, L. F.; Alami, R.; and Simeon, T. 2007. A human aware mobile robot motion planner. *IEEE Transactions on Robotics* 23(5):874–883.

Vernon, D., and Vincze, M. 2016. Industrial priorities for cognitive robotics. In *EUCognition*, volume 1855 of *CEUR Workshop Proceedings*, 6–9. CEUR-WS.org.

Action trees for scalable goal recognition in robotic applications

Helen Harman, Keshav Chintamani, Pieter Simoens
Department of Information Technology - IDLab,
Ghent University - imec,
Technologiepark 15, B-9052 Ghent, Belgium
{firstname.surname}@ugent.be

Abstract

Robots are being deployed in a wide range of environments to assist humans with their daily activities. To assist a person, and avoid obstructing them when executing a different task, a robot needs to know what the intentions of the people are. In this short paper we present an early version of our work, in which we focus on goal recognition using techniques from classical symbolic planning to form an *Action Tree*. We present results which show improved goal recognition times, without compromising on accuracy.

Introduction

Increasingly robots are being developed to work alongside and help humans, therefore it is essential for a robot to understand the intentions of the humans. In many situations there are multiple ways the same aim can be achieved. Thus, in order to assist the person a robot will need to recognise both the goal and the intended plan. However, humans are likely to switch between goals and leave goals partially completed causing real world intention recognition to be a challenging problem. In addition to this, noisy erroneous sensor observations may cause further problems. Our long-term aim is to enable robots to provide assistant to humans with their daily activities, by recognising a person's goal and how they intend to reach that goal (i.e. their plan).

In this paper we focus on single-goal recognition, as a first step. This is to show our algorithm's potential advantages for use in intention recognition in robotic applications. We propose transforming the planning problem into an *Action Tree* (i.e. AND-OR tree with some temporal constraints). This allows the dependencies between different actions to be represented, and the most likely plans as well as goals can be extracted.

A well-studied approach to intention recognition is searching through a dictionary/library of predefined plans (Zhuo and Li 2011). (Holtzen et al. 2016) take a similar approach to us, as they use a Temporal AND-OR tree. However, their probability update rules differ from ours, and we do not use a dictionary, as we aim to allow more flexibility in the way a person's intentions are modelled (Ramırez and Geffner 2010).

One approach is through training a model on humans' intentions using a set of training data, such as HMM (Singla, Cook, and Schmitter-Edgecombe 2010) and RNN (Bisson, Larochelle, and Kabanza 2015). These types of approaches can require a lot of time being spent on manually labelling data and can produce models which only work on data similar to the training set (Yordanova, Krüger, and Kirste 2012).

Due to the above disadvantages we have opted to use a classical planning approach to intention recognition, which in some literature is referred to as goal/plan recognition as planning (Sohrabi, Riabov, and Udrea 2016) or inverse planning (Ramırez and Geffner 2010). In (Ramırez and Geffner 2010), (Chen et al. 2013) and (Freedman and Zilberstein 2017) a planner must be called twice for every possible goal, which is unscalable to large state-spaces.

(Pereira, Oren, and Meneguzzi 2017) significantly reduce the recognition time through the use of landmarks, i.e. actions that must always be performed for a goal to be reached. We will compare our approach to this and show we have improved the scalability of goal recognition. In (Freedman et al. 2018) an algorithm to speed-up recognition time, by only using a single call to the planner, has been proposed but not yet implemented, therefore in future work we would look at also comparing to this approach.

We begin by presenting a brief description of the planning algorithm we have adapted. Then, we describe our approach to goal recognition. Finally, we give our preliminary result.

Background

Traditionally a planning problem is formally defined as $P = (F, I, A, G)$. Where F is a set of atoms, $I \subset F$ is the initial state, $G \subset F$ is a goal state, and A is a set of actions along with their preconditions and effects (Ramırez and Geffner 2010). A task planner is used to find the least costly sequence of actions required to reach the goal state. Often these planning problems are written in Planning Domain Definition Language (PDDL).

In Fast Downward (FD) (Helmert 2006) the planning problem is first translated into SAS+ (a "multivalued planning tasks" representation). Actions and states which are impossible to reach from the goal are removed during this translation. From this the causal relationships between state variables (i.e. causal graph) and how the variables change state i.e. Domain Transition Graphs (DTGs) are determined. Every variable has its own DTG. The causal graph and DTG are used during the search for a plan.

Goal recognition is often viewed as the inverse of planning i.e. $T = (F, I, A, O, \mathcal{G})$ where $\mathcal{G}$ is the set of all possible goals and O is the sequence of observed actions (Ramırez and Geffner 2010). In this paper, we aim to find the probability of each $G \in \mathcal{G}$.

Dataset

For evaluating our approach we use the datasets produced by (Pereira, Oren, and Meneguzzi 2017), which are based on the problems used for the International Planning Competition (IPC). Each dataset contains a PDDL domain and template (i.e. problem without a goal) file, a list of possible goals $\mathcal{G}$, and a sequence of observations $O \subset A$. To check the results produced by goal recognition the real goal is provided.

In Figure 2 we show the Action Tree for the *Kitchen* dataset. The next section describes how it has been created and how the probability of an action being performed is updated.

Method

Our software starts by creating a PDDL problem file, consisting of the template plus a goal state containing all of the possible goals $\mathcal{G}$ in an `or` statement. The PDDL domain and problem file is then transformed into a set of DTGs. Once the Action Tree has been created from the DTGs, the probability of each action $a \in A$ appearing in the person's plan is updated based on the observations O. The tree is then searched for the most likely actions which result in each of the possible goals being reached. The goal(s) with the highest probability are returned, i.e. the *candidate goals* $\mathcal{C} \subset \mathcal{G}$. Multiple candidate goals could be returned as several goals can be equally likely. An overview of our system is shown in Figure 1. We use the term *dependencies* to mean the actions that must be performed before another action can be performed.

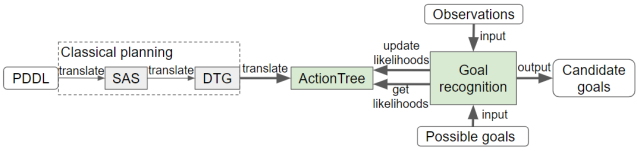


Figure 1: The most important classes in our system are depicted in green boxes; grey boxes show the steps performed by the original task planner FD (Helmert 2006).

To describe our method we use the *Kitchen* dataset. In this dataset there are 3 possible goals: `made_breakfast`, `lunch_packed` and `made_dinner`. For each of these goals there are multiple plans which can be used to reach that goal, e.g. for `lunch_packed` a person must always perform the `take(lunch_bag)` action and has the option of either perform the `activity-make-peanut-butter-sandwich` or `activity-make-cheese-sandwich` action.

Action tree creation

To perform goal recognition, we transform the DTGs into an Action Tree, in which leaf nodes are actions and all other nodes are: `OR` nodes in which one or more of the sub-trees must be performed; `UNORDERED-AND` nodes where all sub-trees are performed in any order, and `ORDERED-AND` nodes for which all sub-trees must be performed in order.. All examples used in this section are shown in the Action Tree depicted in Figure 2. Action (leaf) nodes and `ORDERED_AND` nodes can have multiple parent nodes, as the Action Tree only contains one action node per action. Unless otherwise stated, we always use the term parent(s) to refer to the direct parent(s) of a node. A tree is initialised with an `OR` node as the root, this root remains the same and will receive a new child for every action inserted into the tree.

Each DTG describes how a variable changes state. Multiple labels are given to transitions with multiple possible preconditions, e.g. the preconditions for `activity-Pack-lunch` require either `(made_cheese_sandwich)` or `(made_peanut_butter_sandwich)` to be true. `activity-Pack-lunch` has the effect `lunch_packed`, therefore the transition to `lunch_packed` being true will have at least two labels.

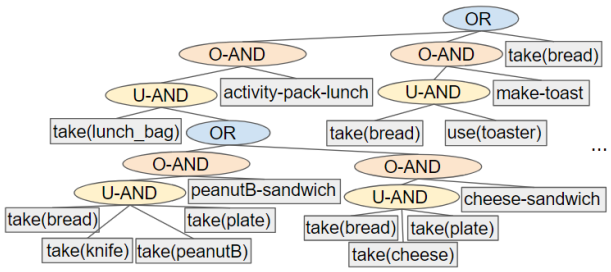


Figure 2: Small section of the action tree created from the kitchen domain. O-AND stands for `ORDERED-AND` and U-AND is `UNORDERED-AND`. For readability some action names have been shortened, e.g. activity-make-peanut-butter-sandwich has been shortened to peanutB-sandwich. Note, in this figure nodes have been repeated to represent that they have multiple parent nodes.

Our system iterates through all the DTG transitions for all of the variables and adds each transition (i.e. action) to the tree. Actions which do not have any preconditions are appended to the root node's children, e.g. `take(bread)`. Actions with dependencies/preconditions are added after all of their dependencies are, e.g. `take(lunch_bag)`, `activity-make-cheese-sandwich` and `activity-make-peanut-butter-sandwich` are added to the tree before `activity-pack-lunch` is.

When an action has dependencies, an `UNORDERED-AND` node is created containing all the dependencies as its children. These children will now have at least two parents, the `UNORDERED-AND` and the root node. The `UNORDERED-AND` node along with the action itself are added to a new `ORDERED-AND` node, which is appended to the root node's children. If the dependencies have dependencies, then a direct child of the `UNORDERED-AND` node will be an `ORDERED-AND` node.

For example, the action `activity-make-cheese-sandwich` requires the actions `take(bread)`, `take(cheese)` and `take(plate)` to be performed first, however it does not matter what order the required actions are performed, therefore they become the chil-

dren of an UNORDERED_AND node; which along with the activity-make-cheese-sandwich action is set as the ORDERED_AND node's children. Note, if a node has an ORDERED-AND node as its parent it can only have one parent.

When a PDDL action contains a precondition which has an or statement or multiple actions exist which result in the same state being reached, the DTG transition will have multiple labels. This results in OR nodes being inserted into the tree. For example, to complete the preconditions of the action activity-pack-lunch the take(lunch_bag) and either activity-make-peanut-butter-sandwich or activity-make-cheese-sandwich must have been performed.

Updating probabilities based on observations

All action nodes are initialised with a probability of 0.5, as they are all equally likely to appear or not appear in a person's plan. We experimented with different initial values but found this made little difference to our results. In future work we intend to experiment with multiple interleaving goals, which this value may have a greater impact on.

When an observation $o \in O \subset A$ is received that action's probability is set to 1, this is shown in line 3 of Algorithm 1. The action node's parents are then updated (lines 10-12). If a parent is an OR node its probability is set to the maximum probability of its children (line 5), otherwise it is set to the mean probability of its children (line 7). This algorithm recurses (line 11) until the root node is reached (line 13). It does not matter in which order a node's parents are updated.

Algorithm 1 Update node probability upwards

```
1:  function UPDATE_PROBABILITY_UPWARDS(node)
2:      if node is an action node then          ▷ The observed action
3:          node.probability = 1.0.
4:      else if node is an OR node then
5:          node.probability = max(children).
6:      else node is an AND node
7:          node.probability = mean(children)
8:      end if
9:      for each parent in node.parents do
10:         UPDATE_PROBABILITY_UPWARDS(parent)
11:     end for
12:     if node is an action node then          ▷ The observed action
13:         UPDATE_PROBABILITY_DOWNWARDS(root)
14:     end if
15: end function
```

We considered using product, rather than mean but found the size of the sub-trees had a much larger effect on the probability of a goal (i.e. strongly favours shorter plans), therefore we opted to use mean as this achieved better results. The maximum probability is used for OR nodes as it does not matter which one of its children have been (partially) executed.

To set the probability of an action appearing in the subsequently performed actions we then traverse down the tree (depth-first) using Algorithm 2. If the current node is an AND node (line 2) and its child's probability is lower, then the child's probability is assigned the AND node's probability (lines 3-5). The direct children of OR nodes are not updated.

Algorithm 2 Update node probability downwards

```
1:  function UPDATE_PROBABILITY_DOWNWARDS(node)
2:      if node is an AND node then
3:          for each child in node.children do
4:              child.probability = max(child, this)
5:          end for
6:      end if
7:      for each child in node.children do
8:          UPDATE_PROBABILITY_DOWNWARDS(child)
9:      end for
10: end function
```

Goal recognition

Each goal $G_i \in \mathcal{G}$ contains one or more atoms $G_i \subset F$. For each atom $f \in G_i$ we find the most likely action whose effects contain f; and find the average over all atoms in G_i. This is shown in Equation 1.

$$p(G_i) = \frac{\sum_{f \in G_i} \max(p(a_{1f \in eff}), ..., p(a_{Nf \in eff}))}{|f \in G_i|} \quad (1)$$

Where $p(G_i)$ is the probability of the i-th goal in $\mathcal{G}$ and $p(a_{1f \in eff})$ is the probability of an action $a_1 \in A$ whose effects contain f. If $p(G_i) \equiv \max(p(G_1), p(G_2), ..., p(G_N))$ then G_i is added to the set of candidate goals $\mathcal{C}$.

Preliminary results

We ran our goal recognition, and the goal completion heuristic from (Pereira, Oren, and Meneguzzi 2017), on a dataset they produced. For both approaches we only consider the most likely goals as being in the set of candidate goals (i.e. the threshold value described by Pereira et al. is set to 0).

The dataset consists of 15 domains and a total of 6313 goal recognition problems; which include problems where 10%, 30%, 50%, 70% and 100% of observations are provided. The goal recognition times for each domain are given in Table 1 and the accuracy is presented in Table 2. $|\mathcal{C}|$ is the number of candidate goals, and the accuracy A is determined by the number of times the correct goal appears in the list of candidate goals.

On all plan recognition problems our approach is faster than that of (Pereira, Oren, and Meneguzzi 2017). Overall our approach took 1727s to run on all plan recognition problems, whereas their approach took 7798s.

Table 1: Recognition times per domain. All times are in seconds. As planning problems can greatly vary in size we show the standard deviation.

		Ours		Pereira et al.	
Domain	$\|probs\|$	$\sum t$	$\overline{t} \pm$ std	$\sum t$	$\overline{t} \pm$ std
miconic	364	125.44	0.34 ± 0.26	546.90	1.50 ± 1.07
sokoban	364	140.09	0.38 ± 0.14	579.63	1.59 ± 0.49
satellite	364	127.15	0.35 ± 0.23	621.45	1.70 ± 1.19
logistics	673	170.24	0.25 ± 0.25	1089.15	1.61 ± 1.05
ferry	364	60.86	0.17 ± 0.07	258.40	0.71 ± 0.17
rovers	364	186.93	0.51 ± 0.33	545.47	1.49 ± 0.78
intrusion-detection	465	54.97	0.12 ± 0.01	331.35	0.71 ± 0.07
kitchen	75	8.10	0.11 ± 0.00	41.64	0.55 ± 0.07
easy-ipc-grid	673	127.58	0.19 ± 0.05	743.55	1.10 ± 0.37
blocks-world	1076	205.36	0.19 ± 0.07	941.76	0.88 ± 0.57
depots	364	134.22	0.37 ± 0.17	481.02	1.32 ± 0.30
zeno-travel	364	166.02	0.46 ± 0.16	615.76	1.69 ± 0.65
dwr	364	110.78	0.30 ± 0.06	517.21	1.42 ± 0.37
campus	75	8.74	0.12 ± 0.00	45.42	0.61 ± 0.06
driverlog	364	100.80	0.28 ± 0.17	438.87	1.21 ± 0.71
ALL	6313	1727.27	0.27 ± 0.20	7797.56	1.24 ± 0.84

On average when 10% of observations are provided our approach has more candidate goals and therefore a higher accuracy. As the number of observations increases the number of candidate goals decreases. For (Pereira, Oren, and Meneguzzi 2017) the number of candidate goals does not decrease by much, however the accuracy increases as the number of observations increases. There are some domains which are exceptions to this trend, such as the kitchen domain where our approach produces fewer candidate goals.

Conclusion and future work

In this paper we presented an early version of our intention recognition system, where we focus on single-goal recognition. DTGs are translated into an Action Tree which is used to predict the probability of a person performing an action. We compared our approach to (Pereira, Oren, and Meneguzzi 2017) and found our approach is quicker and performs equally well in terms of accuracy.

We intend to extract the most likely actions a human will perform from the tree. This will enable a robot to assist the person (e.g. open doors, fetch objects, provide instructions) and avoid obscuring the person when performing a different task within the same environment. In the case of assisting a person, we will investigate how confident the recognition should be before the robot attempts to give assistance.

In future work we will also provide experimentation results for datasets containing invalid and missing observations caused by noisy sensor readings. Additionally, we will test our approach on multiple interleaving and concurrent goals. In dynamically changing environments multiple humans act continuously, including leaving and returning to the environment. Therefore, rather than ending when a goal (or set of goals) is reached, we will investigate how the intentions of a human can be continuously updated. To do this we will experiment with decaying the probability of actions the human has performed, when they are no-longer part of the person's intended plan.

Acknowledgements

H. Harman is an SB fellow at FWO (prj. 1S40217N). Part of this research was funded via imec's RoboCure project.

Table 2: Accuracy for the different domains when different percentages of observations are known. To save space we do not show results for 100% of observations.

			Ours		Pereira et al.	
Domain	$\overline{\|\mathcal{G}\|}$	**Obs %**	$\|\mathcal{C}\|$	A	$\|\mathcal{C}\|$	A
miconic	6	10	4.40	0.96	1.46	0.69
		30	2.64	0.96	1.15	0.98
		50	1.88	0.96	1.02	0.99
		70	1.23	0.96	1.01	1.00
sokoban	7.14	10	5.35	0.93	2.10	0.55
		30	2.73	0.82	1.40	0.58
		50	2.26	0.86	1.35	0.71
		70	1.45	0.95	1.08	0.86
satellite	6.43	10	2.65	0.87	2.18	0.70
		30	1.51	0.87	1.45	0.86
		50	1.17	0.89	1.29	0.94
		70	1.11	0.96	1.05	0.99
logistics	10.46	10	6.74	0.96	2.01	0.63
		30	3.39	0.98	1.34	0.86
		50	1.91	0.97	1.21	0.95
		70	1.24	0.99	1.10	0.97
ferry	7.57	10	3.51	0.98	1.45	0.64
		30	1.50	0.88	1.15	0.86
		50	1.26	0.92	1.07	0.94
		70	1.12	1.00	1.00	0.96
rovers	6	10	2.73	0.87	1.82	0.67
		30	1.31	0.85	1.36	0.82
		50	1.12	0.96	1.12	0.89
		70	1.00	0.98	1.05	1.00
intrusion-detection	16.67	10	1.39	0.73	1.37	0.74
		30	1.05	0.96	1.03	0.95
		50	1.01	0.99	1.03	1.00
		70	1.00	0.99	1.00	1.00
kitchen	3	10	1.00	0.80	3.00	1.00
		30	1.00	0.93	2.60	1.00
		50	1.00	0.93	2.60	1.00
		70	1.00	0.93	2.33	1.00
easy-ipc-grid	8.66	10	7.67	1.00	2.58	0.67
		30	6.03	1.00	1.65	0.82
		50	3.97	1.00	1.18	0.91
		70	3.12	1.00	1.07	0.97
blocks-world	20.28	10	7.07	0.62	1.26	0.44
		30	1.65	0.62	1.17	0.56
		50	1.20	0.74	1.13	0.63
		70	1.19	0.91	1.15	0.84
depots	8.86	10	4.42	0.90	1.31	0.39
		30	2.44	0.93	1.15	0.67
		50	1.69	0.98	1.11	0.85
		70	1.44	0.99	1.01	0.94
zeno-travel	6.86	10	3.33	0.92	1.43	0.45
		30	2.11	0.90	1.40	0.79
		50	1.27	0.96	1.15	0.82
		70	1.05	1.00	1.10	0.98
dwr	7.29	10	2.55	0.61	1.20	0.38
		30	1.51	0.75	1.10	0.64
		50	1.38	0.85	1.06	0.73
		70	1.15	0.89	1.05	0.90
campus	2	10	1.93	0.93	1.13	0.87
		30	1.93	0.93	1.13	0.87
		50	1.80	0.87	1.13	0.93
		70	1.73	0.93	1.00	1.00
driverlog	7.14	10	3.17	0.83	1.29	0.45
		30	1.67	0.73	1.24	0.60
		50	1.40	0.90	1.29	0.77
		70	1.19	0.95	1.24	0.93
ALL	10.43	10	3.86	0.86	1.71	0.62
		30	2.16	0.87	1.36	0.79
		50	1.62	0.92	1.25	0.87
		70	1.33	0.96	1.15	0.96

References

Bisson, F.; Larochelle, H.; and Kabanza, F. 2015. Using a recursive neural network to learn an agent's decision model for plan recognition. In *IJCAI*, 918–924.

Chen, J.; Chen, Y.; Xu, Y.; Huang, R.; and Chen, Z. 2013. A planning approach to the recognition of multiple goals. *Intl Journal of Intelligent Systems* 28(3):203–216.

Freedman, R. G., and Zilberstein, S. 2017. Integration of planning with recognition for responsive interaction using classical planners. In *AAAI*, 4581–4588.

Freedman, R. G.; Fung, Y. R.; Ganchin, R.; and Zilberstein, S. 2018. Towards quicker probabilistic recognition with multiple goal heuristic search.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Holtzen, S.; Zhao, Y.; Gao, T.; Tenenbaum, J. B.; and Zhu, S.-C. 2016. Inferring human intent from video by sampling hierarchical plans. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ Intl Conf on*, 1489–1496. IEEE.

Pereira, R. F.; Oren, N.; and Meneguzzi, F. 2017. Landmark-based heuristics for goal recognition. In *Thirty-First AAAI Conf on Artificial Intelligence (AAAI-17)*. AAAI Press.

Ramırez, M., and Geffner, H. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of the Conf of the Association for the Advancement of Artificial Intelligence (AAAI 2010)*, 1121–1126.

Singla, G.; Cook, D. J.; and Schmitter-Edgecombe, M. 2010. Recognizing independent and joint activities among multiple residents in smart environments. *Journal of ambient intelligence and humanized computing* 1(1):57–63.

Sohrabi, S.; Riabov, A. V.; and Udrea, O. 2016. Plan recognition as planning revisited. In *IJCAI*, 3258–3264.

Yordanova, K.; Krüger, F.; and Kirste, T. 2012. Context aware approach for activity recognition based on precondition-effect rules. In *PERCOM Workshops, 2012 IEEE Intl Conf on*, 602–607. IEEE.

Zhuo, H. H., and Li, L. 2011. Multi-agent plan recognition with partial team traces and plan libraries. In *IJCAI*, volume 22, 484.

Robust Human Activity Monitoring Using Qualitative Spatial Representation and Reasoning

Sang Uk Lee, Ashkan Jasour, Andreas Hofmann, Brian Williams

Massachusetts Institute of Technology
77 Massachusetts Ave, Cambridge, MA, 02139, USA
sangukbo@mit.edu, jasour@mit.edu, hofma@csail.mit.edu, williams@csail.mit.edu

Abstract

This paper presents a robust human activity monitoring algorithm. We focus on a human activity monitoring problem with three distinct characteristics: i) there is significant sensing noise, ii) humans act according to a predefined abstract behavior model in the context of a plan, and iii) spatial relations between the objects involved are an important aspect. Our algorithm, called Logical Activity Recognition System (LCARS), has two components: i) the offline compilation component and ii) the online estimation component. The offline part autonomously generates the online part in offline, using common sense structural and logical knowledge. This knowledge is based on the abstract human behavior model, written in Planning Domain Definition Language (PDDL), and spatial relations between objects, represented using Region Connection Calculus-8 (RCC-8). Especially, use of PDDL allows this work to be fluently connected to other planning and execution works. The resulting online part performs online estimation with sensor measurements. It has a layered structure with a series of Hidden Markov Models (HMMs) coding common sense knowledge. Experimental result shows that LCARS is robust even under significant sensing noise.

Keywords

Human activity monitoring, Human robot collaboration, Planning Domain Definition Language (PDDL), predicate estimation, and Qualitative Spatial Representation and Reasoning (QSR)

Introduction

Applications where humans and robots need to cooperate are of increasing interest, a manufacturing environment for example. In human-robot interaction, a robot's ability to recognize and monitor which activity the human is performing is crucial to ensuring both safe and effective collaboration. In this paper, we focus on a human activity monitoring problem with three distinct characteristics. Firstly, there is a significant amount of noise in our sensing capability. This would require the solution to be robust to noise. Secondly, humans act according to a predefined abstract behavior model. Thirdly, the spatial relations between the objects are an important aspect of a human-robot interaction scenario.

We present Logical Activity Recognition System (LCARS) for the human activity monitoring problem. It has two components: i) the offline compilation component and ii) the online activity estimation component. The offline part builds the online part autonomously using common sense structural and logical knowledge. This knowledge is based on an abstract human behavior model and spatial relations between objects. For example, if a human is holding a red block, he/she must place it before picking up a green block, meaning that the hand, first in contact with a red block, should be away from it and then be in contact with a green block. The online part, which is the result from offline compilation, is the one that actually performs estimation over predicates and activities. It has a layered structure. Each layer is designed as a set of Hidden Markov Models (HMMs) (Murphy 2012), coding common sense knowledge. Using the probabilistic approach based on common sense information filters out noisy observations and ensures robustness. We explain LCARS using a general pick-and-place example.

The use of structural and logical information in the offline compilation process of LCARS is possible because we applied Planning Domain Definition Language (PDDL) (Fox and Long 2003) to code the abstract human behavior model, specifying conditions and effects of each activity. In this work, we assume that humans perform tasks based on a predefined PDDL code. PDDL is a predicate-based language (it has statements like $(empty\ manipulator)$, which is true if and only if the manipulator is empty) widely used in the artificial intelligence (AI) community for activity planning and execution. Using PDDL has the following advantages: i) we can use an existing and well-proven language, and ii) our work can be easily integrated with existing planning and execution work. Despite these advantages, PDDL has one downside. Extracting the common sense structural information we can use is not direct, due to its predicate-based nature. However, we apply a recently developed algorithm called invariant synthesis to solve this problem (Bernardini and Smith 2011).

In many cases, predicates in PDDL are closely related to qualitative spatial relations between objects. For instance, the $(empty\ manipulator)$ predicate is true if the manipulator is not in contact with any objects: $(empty\ manipulator) \coloneqq \forall object, \sim(in\_contact\ manip, object)$. Thus, we can represent predicates using primitive statements about spatial relations, such as $(in\_contact\ manip, obj)$. We use Region Connection Calculus (RCC) (Cohn et al. 1997), a calculus used in Qualitative Spatial Reasoning (QSR) (Freksa 1991),

for this purpose. The benefits of using QSR are: i) it can represent spatial predicates in PDDL easily; ii) it better matches intuitive thinking; and iii) it enables us to use qualitative reasoning to represent complex ideas.

The main contribution of this work is that it provides a robust human activity monitoring algorithm using PDDL and QSR. There has been other works that use QSR for human activity monitoring, such as (Schlenoff 2013; 2015). However, these works only provide deterministic approaches and lack robustness. Our online estimation component uses probabilistic approach through layers of HMMs, ensuring robustness to noisy observation. Experimental results compare the two. Also, to author's knowledge, this paper is the first to use PDDL, a well-known existing language, for human activity monitoring. Other human activity recognition works such as (Schlenoff 2013), (Awais and Henrich 2010), and (Schrempf and Hanebeck 2005) used a predefined human behavior model, written in languages of their own. This might make the human activity monitoring problem itself easier, since the language can be specialized to the activity monitoring problem. However, it would be more difficult to integrate the activity monitoring work with other planning and execution works. Especially, by using PDDL, LCARS estimates over not only human activities, but also predicates in PDDL, thus performing the role of predicate estimator as well.

This paper presents several supplementary novelties. Firstly, this paper provides how we can use RCC-8 for human activity monitoring problem. In previous human activity monitoring works using QSR, such as (Schlenoff 2013; 2015), used RCC-3D (Albath et al. 2010), a variant of RCC, instead of RCC-8. We state that RCC-3D only complicates the representation unnecessarily. We support this statement by providing a complete explanation on how to use RCC-8 for human activity monitoring, illustrating how basic spatial predicates, such as *on*, *in*, *empty (manipulator)* etc., can be represented in RCC-8. Secondly, LCARS' offline component can generate the online activity estimation unit for different PDDL scenarios autonomously. Autonomous compilation for other scenarios hasn't been discussed much in other works. This automation is possible thanks to the repetitive structure of online estimation component of LCARS and the invariant synthesis algorithm. Thirdly, this paper shows that we can use a collision detection algorithm to efficiently find which RCC-8 statements hold for any two given regions.

This paper is organized as follows. Section II provides a pick and place example used throughout the paper. Section III provides the formal problem statement and an overview of our solution. The background is presented in Section IV. Brief explanations of PDDL and RCC, in addition to our pick-and-place example are provided here. A detailed illustration of LCARS is presented in Section V and Section VI, with Section V for the online estimation component and Section VI for the offline compilation component. Section VII presents experimental results. Finally, the paper is concluded in Section VIII.

Pick and Place Example

In this paper, we use a pick and place example throughout. Experimental results are also based on this example. Our pick and place example has three actions; pick, place, and pass. In pick action, a manipulator picks up a block from a location. In place action, a manipulator places a block on a location. In pass action, a manipulator holding a block (a human hand) passes the block to another empty manipulator (a robot manipulator). We provide the PDDL code for our pick and place example in Table I. Our goal is to estimate which action human is performing, among actions given in the PDDL code. The meaning of each predicate and action is commented with //. Note that we use the term action instead of activity in PDDL, and two terms have the same meaning in this paper.

TABLE I. EXAMPLE PICK AND PLACE PDDL CODE

```
(define (domain PDDL-domain)
  (:requirements :strips :typing :durative-actions)
  (:types manipulator object location)
  (:predicates
    (on ?obj - object ?loc - location)  // true when ?obj is on ?loc
    (clear ?obj - object)
    // true when no manipulator is holding ?obj (clear to hold)
    (empty ?manip - manipulator)  // true when ?manip is empty
    (holding ?obj - object ?manip - manipulator)
    // true when ?manip holding ?obj
    (in-pass-region ?manip - manipulator))
    // true when ?manip is in predefined region used for passing
  (:durative-action pick  // ?manip picks up ?obj from ?loc
    :parameters (?obj - object ?manip - manipulator ?loc - location)
    :duration (= ?duration 20)
    :condition (and
      (at start (on ?obj ?loc))  (at start (clear ?obj))
      (at start (empty ?manip))  (at start (not (in-pass-region ?manip)))
      (at end (on ?obj ?loc))  (at end (not (clear ?obj))
      (at end (not (empty ?manip)))  (at end (holding ?obj ?manip))
      (at end (not (in-pass-region ?manip)))
      (over all (on ?obj ?loc))  (over all (not (clear ?obj))
      (over all (not (empty ?manip)))  (over all (holding ?obj ?manip))
      (over all (not (in-pass-region ?manip))))
    :effect (and
      (at start (not (clear ?obj)))  (at start (not (empty ?manip)))
      (at start (holding ?obj ?manip)))  (at end (not (on ?obj ?loc)))
  (:durative-action place  // ?manip places ?obj on ?loc
    :parameters  (?obj - object ?manip - manipulator ?loc - location)
    :duration (= ?duration 20)
    :condition (and
      (at start (not (on ?obj ?loc)))  (at start (not (clear ?obj)))
      (at start (holding ?obj ?manip))
      (at start (not (in-pass-region ?manip)))  (at end (on ?obj ?loc))
      (at end (not (clear ?obj)))  (at end (holding ?obj ?manip))
      (at end (not (in-pass-region ?manip)))  (over all (on ?obj ?loc))
      (over all (not (clear ?obj)))  (over all (holding ?obj ?manip))
      (over all (not (in-pass-region ?manip))))
    :effect (and
      (at start (on ?obj ?loc))  (at end (clear ?obj))
      (at end (empty ?manip))  (at end (not (holding ?obj ?manip)))))
  (:durative-action pass  // ?m1 passes ?obj to empty ?m2
    :parameters (?obj - object ?m1 - manipulator ?m2 - manipulator)
    :duration (= ?duration 30)
    :condition (and
      (at start (not (clear ?obj)))  (at start (not (empty ?m1)))
      (at start (holding ?obj ?m1))  (at start (in-pass-region ?m1))
      (at start (empty ?m2))  (at end (not (clear ?obj)))
      (at end (not (empty ?m1)))  (at end (holding ?obj ?m1))
```

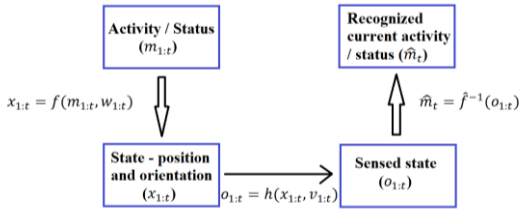
```
    (at end (in-pass-region ?m1))  (at end (not (empty ?m2)))
    (at end (holding ?obj ?m2))  (at end (in-pass-region ?m2))
    (over all (not (clear ?obj)))  (over all (not (empty ?m1)))
    (over all (holding ?obj ?m1))  (over all (in-pass-region ?m1))
    (over all (not (empty ?m2)))  (over all (holding ?obj ?m2))
    (over all (in-pass-region ?m2)))
  :effect (and
    (at start (not (empty ?m2)))  (at start (holding ?obj ?m2))
    (at start (in-pass-region ?m2))  (at end (empty ?m1))
    (at end (not (holding ?obj ?m1))))))
```

This example is written in PDDL version 2.1, where durative actions were first introduced to represent actions with a time duration. Note, definition of predicates and activities in the above PDDL code is ungrounded. In our experiment, we used one manipulator (a human hand), three blocks (red, green, and blue), and two locations (A and B). Thus, there were 14 grounded predicates (6 for *on*, 3 for *clear*, 1 for *empty*, 3 for *holding*, and 1 for *in-pass-region*), and 15 grounded activities (6 for *pick*, 6 for *pick*, and 3 for *pass*). For pass activity, we needed a second manipulator (a robot manipulator), but since we are performing human activity monitoring, the robot manipulator was preprogrammed and not part of estimation (it was assumed to be perfectly observable).

Problem Statement and Solution Overview

The task of human activity monitoring is to estimate the activities that a human is performing and their statuses (and also estimate the state of the world, given in predicate statements in many cases, if possible). Note that we are interested in activities with a temporal duration: thus, we have distinguished the activity (type, i.e. pick, place, etc.) from its status (the temporal stage, i.e. executing, finished, etc.). Three main characteristics of our problem are: i) sensors are assumed to be noisy, ii) humans act according to a predefined abstract behavior model, and iii) spatial relations between objects are of great interest. The human activity monitoring task can be visualized as in Figure 1 (Heinze 2004).

Figure 1. Human activity monitoring problem (Heinze 2004)



We assume that the true model of how a human's activity and activity status affect the true state $x_{1:t}$, positions and orientations of objects, can be represented as some function, $x_{1:t} = f(m_{1:t}, w_{1:t})$. $w_{1:t}$ represents probabilistic behaviors within the true model. We can sense $x_{1:t}$ using a noisy sensor, $o_{1:t} = h(x_{1:t}, v_{1:t})$, where $v_{1:t}$ is the sensor noise. Our job is to model the human activity monitoring system, $\hat{m}_t = \hat{f}^{-1}(o_{1:t})$, to estimate the current activity and activity status. We suggest that this can be modeled well using the abstract human behavior model and spatial relations between objects.

Figure 2. Graphical representation of LCARS

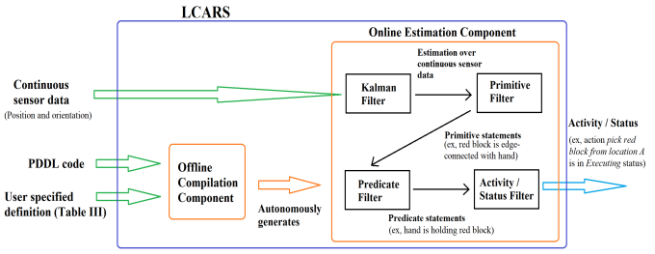


Figure 2 visualizes the structure of our solution called LCARS. It has two components: i) the offline compilation component and ii) the online estimation component. In the big picture, LCARS takes three inputs, continuous sensor data, PDDL code, and user specified definitions about how each predicate in PDDL is to be represented in RCC-8 statements, as in Table III. It estimates the current activity a human is performing and its status. To be specific, the offline part uses the PDDL code and user specified definitions to generate the online estimation algorithm autonomously. The resulting online part uses the continuous sensor data to estimate the current activity and status. The online estimation component has a layered structure with four filters. Each filter is designed as a set of HMMs. Each filter layer applies structural and logical relations, based on PDDL and RCC-8 formulation, in a series. This information might be hard to apply together at one time, so we apply it in a sequence, through a series of filters.

Algorithm Background

PDDL and Invariant Synthesis

Planning Domain Definition Language is widely used in activity planning (Fox and Long 2003). An example PDDL code is provided in Table I. PDDL is based on predicates that can be true or false. For example, a predicate $(on\ block_r\ location_A)$ is true if and only if the red block is at location A. Each durative action has the same structure. *condition* is a set of predicates that must hold to execute the action, and *effect* is a set of predicates that results from applying the action. *at start* indicates predicates related to the beginning of an action, and *at end* indicates predicates related to the end. *over all* indicates predicates related to the duration between start and end. A more detailed explanation of PDDL 2.1 is provided in (Fox and Long 2003).

PDDL is a planning language based on predicate statements; it is different from representations that use multi-valued state variables. Let's explain this further with PDDL predicates: $(empty\ hand)$, $(holding\ block_r\ hand)$, $(holding\ block_g\ hand)$, and $(holding\ block_b\ hand)$. We can think of them as separate statements, where each statement either can be true or false, which is how PDDL works. However, we can combine them and form a new multi-valued state variable, x_{hand}, with four possible states, stating that $hand$ must either be empty or holding one of the blocks. Transition between states can be accomplished through defined *pick* and *place* actions. This representation offers some advantages: i) it provides us with structural information

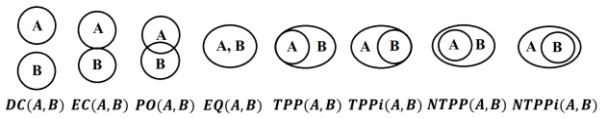
about the possible transitions between states, ii) it is more intuitive to human users, and iii) it reduces the size of the possible domain. An algorithm recently introduced in the activity planning community, called invariant synthesis, extracts multiple-valued state variables from PDDL. To be more specific, the invariant synthesis algorithm finds sets of predicates that are mutually exclusive to each other, meaning that only one predicate in the set can be true at any time. A more detailed explanation of invariant synthesis is in (Bernardini and Smith 2011).

QSR and RCC-8

Qualitative Spatial Reasoning (Freksa 1991) abstracts continuous spatial data on objects or regions (positions and orientations) into qualitative relations between them. Then, we can perform reasoning using the abstract spatial relations. Region Connection Calculus (Cohn et al. 1997) is a promising approach for this job.

In RCC, there is only a finite number of qualitative relations possible for any two given (regular (Cohn et al. 1997)) objects or regions. The number is 5 for RCC-5, 8 for RCC-8, and 23 for RCC-23 (Cohn et al. 1997), etc. RCC-8 is going to be used in this paper since it is rich enough. In RCC-8, the finite relations are i) A is disconnected from B ($DC(A,B)$), ii) A is edge-connected with B ($EC(A,B)$), iii) A is partially occluded by B ($PO(A,B)$), iv) A is identical with B ($EQ(A,B)$), v) and vi) A is a tangentially proper part of B, or the inverse ($TPP(A,B)$ or $TPPi(A,B)$), vii) and viii) A is a nontangentially proper part of B, or the inverse ($NTPP(A,B)$ or $NTPPi(A,B)$). These relations are visualized in Figure 3. We are going to use RCC-8 statements as primitives to express complex predicates in PDDL.

Figure 3. RCC-8 statements (Cohn et al. 1997)



Given two closed regular regions A and B in $\mathbb{R}^3$ space, we can acquire an RCC-8 relation using a collision detection algorithm. We emphasize that no other papers have discussed this potential. Since collision detection has been a massive research field with many efficient algorithms developed (Ericson 2004), we can find the RCC-8 statement very efficiently. Table II shows how. Here, $coll(A,B)$ is "True" if and only if A and B are in collision. $int(A)$, $ext(A)$, and $\partial(A)$ indicate interior, exterior, and boundary of region A respectively.

TABLE II. RCC-8 STATEMENTS USING COLLISION DETECTION

RCC-8	Representation using collision detection
$DC(A,B)$	$(\sim coll(int(A), int(B))) \wedge (\sim coll(\partial(A), \partial(B)))$
$EC(A,B)$	$(\sim coll(int(A), int(B))) \wedge (coll(\partial(A), \partial(B)))$
$PO(A,B)$	$(coll(int(A), int(B))) \wedge (\sim(EQ \vee TPP \vee TPPi \vee NPP \vee NPPi))$
$EQ(A,B)$	$(coll(int(A), int(B))) \wedge (\sim coll(ext(A), int(B))) \wedge (\sim coll(int(A), ext(B)))$
$TPP(A,B)$	$(coll(int(A), int(B))) \wedge (coll(\partial(A), \partial(B))) \wedge (\sim coll(int(A), ext(B)))$
$TPPi(A,B)$	$(coll(int(A), int(B))) \wedge (coll(\partial(A), \partial(B))) \wedge (\sim coll(ext(A), int(B)))$
$NTPP(A,B)$	$(coll(int(A), int(B))) \wedge (\sim coll(\partial(A), \partial(B))) \wedge (\sim coll(int(A), ext(B)))$
$NTPPi(A,B)$	$(coll(int(A), int(B))) \wedge (\sim coll(\partial(A), \partial(B))) \wedge (\sim coll(ext(A), int(B)))$

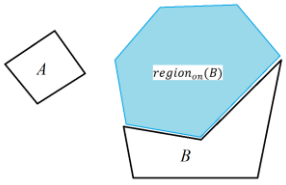
In several previous studies, RCC specialized for $\mathbb{R}^3$ space was introduced, namely RCC-3D (Albath et al. 2010). In addition, RCC-3D was used in human activity monitoring works such as (Schlenoff 2013; 2015). However, we would like to state that RCC-8 is rich enough to deal with $\mathbb{R}^3$ space as well. Using RCC-3D would only complicate the representation unnecessarily. Since the main purpose of using QSR in our work, via RCC-8, is to represent predicates in PDDL (which are defined over $\mathbb{R}^3$ space, of course), we support our argument by showing how some fundamental predicates can be translated using RCC-8 in Table III. This is based on (Aurnague and Vieu 1993).

TABLE III. FUNDAMENTAL PREDICATES USING RCC-8 PRIMITIVES

Predicates	Primitive representation
$(in\ A\ B)$	$NTPP(A,B) \vee TPP(A,B)$
$(on1\ A\ B)$	$EC(A,B) \wedge (in\ A\ region_{on}(B))$
$(on\ A\ B)$	$(on1\ A\ B) \vee \left(\exists obj\ s.t.\ ((on1\ obj\ B) \wedge (on1\ A\ obj))\right) \ldots$
$(holding\ hand\ A)$	$\sim(DC\ hand\ A)$
$(empty\ hand)$	$\forall obj, (DC\ A\ obj)$

$(in\ A\ B)$ means A is in B, and $(on\ A\ B)$ means A is on B. $(on1\ A\ B)$ is for when A is directly on top of B, while the two objects are in contact. $region_{on}(B)$ is the region around B that users can define for the *on* predicate. An example of this region is shown in Figure 4. $(on\ A\ B)$ is more complex; there can be other objects in between. We left ... since there can continue to be more than one object in between. This can be defined recursively. This is a good example of an advantage of how using QSR enables us to express complex ideas through reasoning. Predicates such as $(under\ A\ B)$, $(next\ A\ B)$, etc. can be defined similarly.

Figure 4. A, B, and $region_{on}(B)$



Note that Table III represents the user specified definitions shown in Figure 2. Other users are welcomed to use ours, but they need to provide such definitions if they want to use different predicates or change the definitions. A final comment is that if we want to use a deterministic approach to state predicates, as in (Schlenoff 2013), we apply relations in Table II and Table III directly, without combining them with HMMs. We provide an outline of a deterministic approach in Algorithm 2.

Online Estimation Component

The online estimation component takes the continuous sensor data and estimates the current activity statuses as well as primitives and predicates. It has four layers of HMMs performing filtering. The overall structure is shown in Algorithm 1. Though the offline compilation component operates first to make the online estimation component for given PDDL problem, we illustrate the online component first for easier explanation.

Algorithm 1 : Online layered HMM filtering

1 **Input** : $bel(pose_{t-1}), bel(primitive_{t-1}), bel(predicate_{t-1})$ $bel(activity_{t-1}), o_t^{pose}$
// $bel(s)$ is belief state over s, o_t^{pose} is vector of object poses
2 **Output** : $bel(pose_t), bel(primitive_t), bel(predicate_t)$, $bel(activity_t)$ // we can get most probable state from $bel(s)$
3 $bel(pose_t) \leftarrow$ Kalman_Filter$(bel(pose_{t-1}), o_t^{pose})$
4 $pose_t^{ml} \leftarrow \underset{pose_t}{\text{argmax}}(bel(pose_t))$ // most likely pose
5 $o_t^{primitive} \leftarrow RCC-8\_from\_Table\_II(pose_t^{ml})$
// $o_t^{primitive}$ is vector of relevant RCC-8 statements
6 $bel(primitive_t) \leftarrow$ Primitive_Filter$(bel(primitive_{t-1}), o_t^{primitive})$
7 $primitive_t^{ml} \leftarrow \underset{primitive_t}{\text{argmax}}(bel(primitive_t))$ // most likely primitive
8 $o_t^{predicate} \leftarrow Predicate\_Table\_III(primitive_t^{ml})$
// $o_t^{primitive}$ is vector of relevant predicate statements
9 $bel(predicate_t) \leftarrow$ Predicate_Filter$(bel(predicate_{t-1}), o_t^{predicate})$
10 $predicate_t^{ml} \leftarrow \underset{predicate_t}{\text{argmax}}(bel(predicate_t))$ // most likely predicate
11 $o_t^{activity} \leftarrow Activity\_Figure\_7(predicate_t^{ml})$
// $o_t^{activity}$ is vector of relevant activity statuses
12 $bel(activity_t) \leftarrow$ Activity_Filter$(bel(activity_{t-1}), o_t^{activity})$

Kalman Filter

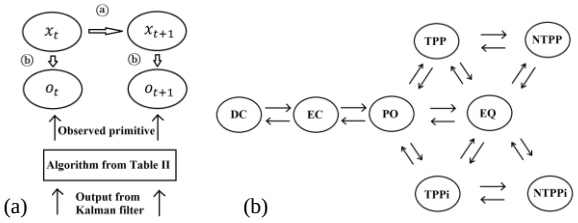
The Kalman filter (Vaseghi 2008) in our layered framework takes continuous sensor data (o_t^{pose}) as the input and outputs the filtered estimate of positions and orientations of objects. We use the transition function $x_{t+1} = x_t + w_t$, where x_t is a 6 by 1 vector of the object's position and orientation. We use this model because we assume that we have no information about what a human is going to do at this level. The observation function is $o_t = x_t + v_t$. It adds the additive sensor noise term. The Kalman filter is used to process the noise initially to some extent. w_t and v_t can be learned using training dataset.

Primitive Filter

Primitive filter converts the output from the Kalman filter into an estimate of currently true RCC-8 primitive statements, such as $(DC\ hand\ block_r)$, $(NTPP\ block_r\ location_A)$, etc. (line 4 ~ 6 in Algorithm 1). We design a distinct HMM for each combination of objects or regions, such as $\{hand, block_r\}$ and $\{hand, block_g\}$, to get RCC-8 statements for all combinations. Each combination is considered independently. The HMM in the primitive filter is graphically represented in Figure 5 (a).

For each HMM, we need transition ($P(x_{t+1}|x_t)$, ⓐ) and observation ($P(o_t|x_t)$, ⓑ) models. For the transition model, we use the continuity network for RCC-8 shown in Figure 5 (b) (Cohn et al. 1997). It specifies which transitions between RCC-8 primitive statements are possible. Self-transition is omitted. Note, for example, that DC cannot move to PO without visiting EC. The observation model is much simpler, it is just the matrix representing $P(o_t|x_t)$, where both x_t and o_t are in the RCC-8 statements. To perform HMM filtering, we need observation value o_t. To get o_t, we use the most probable estimate of pose from Kalman filter ($pose_t^{ml}$) and apply Table II, as if we were using a deterministic approach (line 4 - 5 in Algorithm 1). That is, if using $pose_t^{ml}$ for $block_r$ and $hand$ satisfies $(\sim coll(int(block_r), int(hand))) \wedge (coll(\partial(block_r), \partial(hand)))$ statement in Table II, $o_t^{primitive} = EC(block_r, hand)$ for combination $\{hand, block_r\}$. We would like to emphasize that, since x_t and o_t have the same domain, HMM filtering can be considered as a noise rejection process, as in control theory (Vaseghi 2008). HMMs in other filters have similar structure.

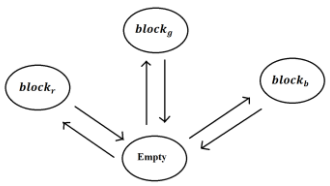
Figure 5. (a) The HMM for primitive filter, and (b) Continuity network (Cohn et al. 1997)



Predicate Filter

The predicate filter takes the output from the primitive filter as its input. It estimates the currently true predicates (line 7 ~ 9 in Algorithm 1). We design a distinct HMM for each set of mutually exclusive (grounded) predicates. We can find mutually exclusive predicates with invariant synthesis. In our pick and place example, we will use the predicate set $\{(empty\ hand), (holding\ block_r\ hand), (holding\ block_g\ hand),$ $(holding\ block_b\ hand)\}$ to demonstrate how an HMM is designed in predicate filter. If such a set has only one element, for example $\{(clear\ block_r)\}$, the predicate forms an individual HMM with two states, $(clear\ block_r)$ and $\neg(clear\ block_r)$. In our pick and place example, there are 11 individual HMMs for predicate filter (1 for $\{(empty\ hand), (holding\ block_r\ hand),$ $(holding\ block_g\ hand), (holding\ block_b\ hand)\}$ form and 10 for $\{(clear\ block_r), \neg(clear\ block_r)\}$ form). Note that using invariant synthesis reduced the number of HMMs needed from 14 to 11.

Figure 6. Example transition model



As for each HMM, an example transition model is given in

Figure 6. Note that if *hand* is initially holding $block_r$, it must place $block_r$ and be empty before picking up $block_g$. The observation model is a matrix representing $P(o_t|x_t)$, where x_t and o_t are both predicates in PDDL. To get o_t, we use the most probable estimate from primitive filter ($primitive_t^{ml}$) and apply Table III, as if we were using a deterministic approach (line 7 - 8 in Algorithm 1). That is, if using $primitive_t^{ml}$ for combination $\{hand, block_r\}$ satisfies $\sim(DC\ hand\ A)$ statement in Table III, $o_t^{predicate} = (holding\ block_r\ hand)$.

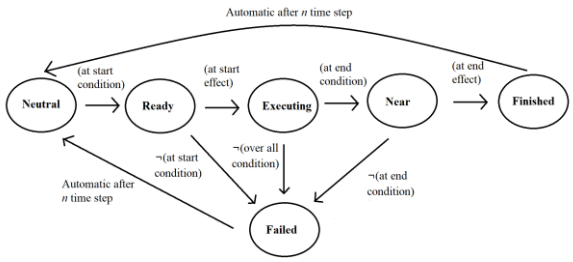
Note, using the output from predicate filter, LCARS can also be used to estimate over the predicates related to human activities.

Activity/Status Filter

The activity/status filter takes the output from the predicate filter. It estimates the current action and status (line 10 ~ 12 in Algorithm 1). Each (grounded) action is designed as a separate HMM. In our pick and place example, there are 15 individual HMMs for activity/status filter.

For each HMM, the transition model is given in Figure 7, which is based on the work in (Wang and Williams 2015) and (Lane 2016). It shows how six predefined statuses (circled in Figure 7) evolve. All guard conditions (indicated along each edge) are ignored since, we use a homogeneous HMM for simplicity. The observation model is a matrix representing $P(o_t|x_t)$, where x_t and o_t are both activity statuses. As for the previous filters, we use the most probable predicates from the predicate filter ($predicate_t^{ml}$) and apply the structure in Figure 7 (with guard conditions) to get o_t, as if we were using a deterministic approach (line 10 - 11 in Algorithm 1). That is, we assume that our previous most probable action status, $\hat{x}_{t-1} = \underset{x_{t-1}}{argmax}\, P(x_{t-1}|O_{1:t-1})$, is given. Then, we assume that we are at status $\hat{x}_{t-1}$ at time $t-1$. If any of the guard conditions for each edge that starts at $\hat{x}_{t-1}$ is satisfied, $o_t^{activity}$ is assigned as the destination status. If not, $o_t^{activity} = \hat{x}_{t-1}$. For example, if we start from $Neutral$, $\hat{x}_{t-1} = Neutral$, and the output from the predicate filter tells us that the *at start condition* is satisfied, then $o_t = Ready$. Here, we are assuming that $x_{t-1} \approx \hat{x}_{t-1}$.

Figure 7. Connection between statuses in each action



Offline Compilation Component

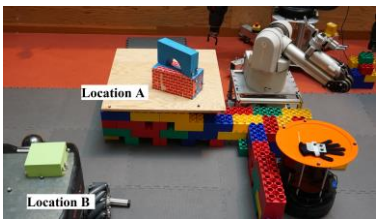
The offline compilation component takes given PDDL code (Table I) and user specified definitions over predicates (Table III) and generates the online estimation component autonomously. That is, even when a different human activity monitoring problem is given, we can autonomously generate online estimation component by modifying only Table I and Table III. The offline compilation process can be automated since filters in the online estimator have repetitive structure with three parts: i) a transition model, ii) an observation model (omitted here since it has a simple matrix form), and iii) an algorithm for obtaining observation o_t. The process is outlined in the following.

1. Inputs are PDDL code as in Table I, and user specified definitions about how each predicate in PDDL code is to be represented in RCC-8 primitives as in Table III. Note, user only need to provide these two inputs for autonomous generation of online estimator.
2. The Kalman filter is generated as in Section IV.
3. The primitive filter is generated using a continuity network for the transition model and Table II for the algorithm to find observation o_t. Note, continuity network structure and Table II don't change for different human activity monitoring problems.
4. The invariant synthesis algorithm is applied to find sets of mutually exclusive predicates.
5. The predicate filter is generated using the result from step 4 for the transition model and user specified definitions (Table III) for the algorithm to find observation o_t (line 8 in Algorithm 1). Note, once all user-required inputs (Table I and Table III) are given, the process can be automated since the invariant synthesis algorithm can extract sets of mutually exclusive predicates autonomously.
6. The activity/status filter is generated using Figure 7 for the transition model. The algorithm to find observation o_t is generated as in Section IV. Note, Figure 7 structure doesn't change for different human activity monitoring problems.

Experimental Results

We performed an experiment using the PDDL in Table I. Figure 8 shows the experiment environment. Again, we used three blocks (red, green, and blue), two locations (A and B), and one manipulators (a human hand). Thus, there were 14 grounded predicates and 15 grounded activities.

Figure 8. Experimental environment



We used a Vicon system to measure the position (global x, y, and z coordinates for the center of mass) and orientation (helical x, y, and z coordinates) of each object, while a person performing a pick, place, and pass action manually. Since the Vicon system is very accurate, we included random white Gaussian noise. For position, we added zero mean noise with covariance $\begin{bmatrix} 400 & 0 & 0 \\ 0 & 400 & 0 \\ 0 & 0 & 400 \end{bmatrix}$ (mm^2). For orientation, covariance was $\begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 0.01 \end{bmatrix}$ (rad^2). Note that the length of the blocks' sides ranged from 70 mm to 200 mm, so the added noise was relatively significant. Then, we applied both our probabilistic approach (online estimatior of LCARS) and a deterministic approach (Schlenoff 2013) to estimate activity statuses. The object pose measurement was performed for about 100 seconds, receiving 9848 sequential measurements. For the deterministic approach, we applied relations in Table II and Table III directly, which is equivalent to not using lines 3, 6, 9, and 12 in Algorithm 1, which is summarized in Algorithm 2.

Algorithm 2 : Deterministic approach (similar to (Schlenoff 2013))

1 **Input** : o_t^{pose} // o_t^{pose} is vector of object poses
2 **Output** : $pose_t, primitive_t, predicate_t, activity_t$
3 $pose_t \leftarrow o_t^{pose}$
4 $primitive_t \leftarrow RCC-8\_from\_Table\_II(pose_t)$
// $primitive_t$ is vector of relevant RCC-8 statements
5 $predicate_t \leftarrow Predicate\_Table\_III(primitive_t)$
// $predicate_t$ is vector of relevant predicate statements
6 $activity_t \leftarrow Activity\_Figure\_7(predicate_t)$
// $activity_t$ is vector of relevant activity statuses

In our work, the accuracy rate is the rate of estimating the most likely activity statuses (or primitives and predicates) correctly compared to the true activity statuses (or primitives and predicates), which was kept separately along the measurement (note the estimation problem in our work is fundamentally the same as the classification problem). Though the main purpose is to estimate the activity and its status, LCARS can also estimate the primitive and the predicate, since we also calculate the belief state over primitives can predicate in Algorithm 1. Thus, we also calculated the primitive and predicate accuracy rates to show LCARS estimate them correctly as well.

We calculated the accuracy rates for primitives. For the deterministic approach, the accuracy rate for all primitives being correct (estimating the whole vector of primitives correctly, for all combinations of objects) at a time was 19.82%. It was 65.86% for LCARS. The primitive filter reduces the noise to some degree before the predicate filter. Next, we calculated the accuracy rates for predicates. For the deterministic approach, the accuracy rate for all predicates being correct at a time was 21.66%. It was 94.53% for LCARS (using both primitive and predicate filters), ensuring robustness. In addition, when we used the probabilistic predicate filter only, without a probabilistic primitive filter (first running until line 5 of Algorithm 2, then running from line 9 of Algorithm 1), the accuracy rate was 63.18%. This shows that combining filters performs better than using only one of them. The results are summarized in Table IV.

TABLE IV. ACCURACY RATES FOR PRIMITIVES AND PREDICATES

Case	Primitives	Case	Predicates
Deterministic only	19.82 %	Deterministic only	21.66 %
Primitive filter	65.86 %	Predicate filter only	63.18 %
-	-	Layered structure	94.53 %

We also calculated the accuracy rates for individual mutually exclusive predicates (rather than as a whole vector). Table V shows the result for some. The result does not deviate much for omitted ones.

TABLE V. ACCURACY RATES FOR MUTUALLY EXCLUSIVE PREDICATES

Mutually Exclusive Predicates	Deterministic	LCARS
$\{(empty\ hand), (holding\ ?obj\ hand)\ \forall ?obj\}$	76.45 %	97.83 %
$\{(clear\ block_r), \neg(clear\ block_r)\}$	92.01 %	99.20 %
$\{(on\ block_r\ location_A), \neg(on\ block_r\ location_A)\}$	88.76 %	99.75 %
$\{(on\ block_r\ location_B), \neg(on\ block_r\ location_B)\}$	72.26 %	99.44 %

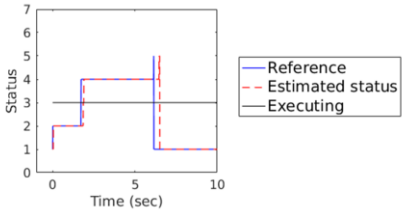
Finally, we checked if LCARS calculates the activity statuses correctly. We applied LCARS to other 60 datasets, performing one of three actions (20 datasets for each of *pick*, *place*, and *pass* action) with either red, green, or blue block and either location A or B. We also included occasional failure of actions (by intentionally violated guard conditions in Figure 7) to show LCARS detects the action failure correctly as well. For example, for pick action, we violated $(holding\ ?obj\ hand)$ in *at end condition*, simulating block slipping from hand. Table VI shows the accuracy rates for getting statuses for all grounded actions correctly (again, for whole vector of all grounded actions). Note, we assumed actions are not mutually exclusive and multiple actions can happen at the same time, for more general future extensions.

TABLE VI. ACCURACY RATES FOR ACTIVITY/STATUS

Action type	Average status accuracy over 20 datasets
Pick	93.42 %
Place	94.02 %
Pass	90.99 %

Figure 9 shows how LCARS estimated the activity statuses over time for an individual example action (for grounded action $(pick\ ?obj = red\ ?manip = hand\ ?loc = location\ A)$). The x axis represents the elapsed time; the y axis shows the status (integers from 1 to 6 indicate $\{Neutral, Ready, Executing, Near, Finished, Failed\}$, respectively). Note LCARS successfully detects the action being finished around 6 seconds (red dotted line reaches status 5). We would like to point out that LCARS always detected the success in finishing the action or failure of the action correctly, with slight delay due to the nature of HMM.

Figure 9. The progress graph for an example grounded action



From the above results, we can conclude that using LCARS reduces noise very effectively. In addition, combining filters as a layered structure performs much better.

Conclusion

This paper presents the LCARS algorithm for robust human activity monitoring tasks. The task has three interesting points: i) sensor noise, ii) a predefined abstract human behavior model and iii) the spatial relations between objects. LCARS has two components: i) the offline compilation component and ii) the online estimation component. The offline part autonomously generates the online part using the common sense structural and logical knowledge. The knowledge is based on the abstract human behavior model, written in PDDL, and spatial relations between objects, represented using RCC-8. The resulting online part has a layered structure, designed as a series of HMMs to estimate the current activity and its status. Experimental results show that our solution is robust.

Future efforts will focus on modeling a more sophisticated structure to capture the dependencies between primitives, predicates, and activities/statuses. Our model is a bottom-up model. That is, it captures well how primitives affect predicates and how predicates affect actions and statuses, compared to the other direction. We hope to model the other direction better by using a variant of an HMM, such as a hierarchical HMM (HHMM) (Murphy 2012). In addition, we will apply LCARS to other noisy sensing technologies. For example, cameras with neural network based object detection algorithms are widely used nowadays. The measurement is very noisy and we expect LCARS to be effective in this case as well.

References

Albath, J.; Leopold, J. L.; Sabharwal, C. L.; and Maglia, A. M. 2010. RCC-3D: Qualitative Spatial Reasoning in 3D. *International Conference on Computer Applications in Industry and Engineering (CAINE)*, 74-79. Las Vegas, NV, USA.

Aurnague, M. and Vieu, L. 1993. A Three-level Approach to the Semantics of Space. In *The Semantics of Prepositions: from Mental Processing to Natural Language Processing 3*, 393-440. Walter de Gruyter.

Awais, M. and Henrich, D. 2010. Human-robot Collaboration by Intention Recognition Using Probabilistic State Machines. *Robotics in Alpe-Adria-Danube Region (RAAD), 2010 IEEE 19th international Workshop on*, 75-80. Balantonfured, Hungary.

Bernardini, S. and Smith, D. E. 2011. Automatic Synthesis of Temporal Invariants. *Ninth Symposium on Abstraction, Reformulation and Approximation (SARA)*, 10-17. Parador de Cardona, Spain.

Cohn, A. G.; Bennett, B.; Gooday, J.; and Gotts, N. M. 1997. Qualitative Spatial Representation and Reasoning with the Region Connection Calculus. *GeoInformatica* 1(3): 275-316.

Ericson, C. 2004. *Real-Time Collision Detection*. CRC press.

Fox, M. and Long, D. 2003. PDDL 2.1: An extension to PDDL for expressing temporal planning domains, *Journal of Artificial Intelligence Research* Vol. 20: 61-124.

Freksa, C. 1991. Qualitative Spatial Reasoning. *Cognitive and Linguistic Aspects of Geographic Space* Vol. 63: 361-372.

Heinze, C. 2004. Modelling Intention Recognition for Intelligent Agent Systems, No. DSTO-RR-0286. Defence Science and Technology Organisation Salisbury (Australia) Systems Sciences Lab.

Lane, S. D. 2016. Propositional and Activity Monitoring Using Qualitative Spatial Reasoning. M.S. Dissertation, Massachusetts Institute of Technology.

Murphy, K. P. 2012. *Machine Learning: A Probabilistic Perspective*. MIT press.

Schlenoff, C.; Pietromartire, A.; Kootbally, Z.; Balakirsky, S.; and Foufou, S. 2013. Ontology-based State Representations for Intention Recognition in Human–robot Collaborative Environments. *Robotics and Autonomous Systems* 61(11): 1224-1234.

Schlenoff, C.; Kootbally, Z.; Pietromartire, A.; Franaszak, M.; and Foufou, S. 2015. Intent Recognition in Manufacturing Applications. *Robotics and Computer-Integrated Manufacturing* Vol. 33: 29-41.

Schrempf, O. C. and Hanebeck, U. D. 2005. A Generic Model for Estimating User Intentions in Human-robot Cooperation. *International Conference on Informatics in Control, Automation, and Robotics (ICINCO)*, 250-256. Barcelona, Spain.

Vaseghi, S. V. 2008. *Advanced Digital Signal Processing and Noise Reduction*. Wiley.

Wang, D. and Williams, B. 2015. tBurton: A Divide and Conquer Temporal Planner. *Association for the Advancement of Artificial Intelligence (AAAI)*, 3409-3417. Austin, TX, USA.

Domain Reasoning for Robot Task Planning — A Position Paper

Uwe Köckemann, Ali Abdul Khaliq, Federico Pecora, Alessandro Saffiotti
Center for Applied Autonomous Sensor Systems (AASS), Örebro University, Örebro, Sweden
firstname.lastname@oru.se

Abstract

In this position paper we argue for moving towards general purpose domains to promote the usage of task planning for real-world robot systems. Planning approaches should extract concrete domains based on their current context in order to solve problems. Towards this aim, we define the problem of domain reasoning, by which a planning domain is obtained from a more general, multi-purpose domain definition, given the current deployment and context of the robot system. We provide examples motivating the need for domain reasoning in robot task planning, as well as a discussion of potential solutions to the domain reasoning problem.

Introduction

In order to make automated planning ready for real-world robotic applications, we need to be able to specify general purpose domains for planning that can be adapted to a variety of different deployment contexts. In order to support this claim, we will employ three hypothetical applications of robot systems as running examples throughout this position paper.

Example 1. *A general-purpose personal robot assistant can be deployed in a variety of different cultural contexts. It should be possible to automatically adapt the domain formulation which drives the robot's task planning to account for background knowledge of the culture and of the personal perferences of the user(s) it is deployed with.*

Example 2. *A general purpose robotic assistant can be deployed either in an office environment to escort visitors or in a museum environment to give guided tours. We want the robot to rely on different context-based instantiations of the same operators. In the office case, movement may be focused on efficiently reaching the target office. In the museum context, movement should be slower and the robot should move with a crowd of visitors that following it.*

Example 3. *Human-aware planning for a robot companion for children and for elderly people might be structurally similar but have different goals, adjusted robotic behaviors, and different constraints on feasible plans. Here, the domain may include constraints for not disturbing school work and not disturbing when the nurse is present. Depending on the deployment context, the correct sub-set of rules is chosen. In case both children and elderly are present in the same household, the domain reasoner should dynamically choose which parts of the domain to apply depending on the current context (e.g., who is at home currently).*

In each of the applications above, the same robot can be used across different contexts. Indeed, we can easily envisage using the same robot across applications. In order to do this, we claim that some form of **domain reasoning** is necessary. Broadly speaking, domain reasoning is the problem of determining a specific domain to be used by a task planner given a *general purpose domain* and the current context. In this paper, we will attempt to provide a more concrete definition of this problem.

There is a variety of possible advantages for general purpose domains: Using such domains may increase the quality of plans, since domain reasoning would distill a purpose-built, highly contextual concrete domain to be used for task planning. There may also be benefits from the point of view of the efficiency of planning. Domain reasoning may filter out large portions of the general purpose domain, potentially leading to smaller problem instances, and in some cases changing the complexity of the problem. Realizing, for instance, that the temporal aspect of a domain is not important in a certain context, planning operators could be restricted to their non-temporal parts. Similarly, context may indicate whether probabilistic planning is necessary. The planning community is well aware of the fact that no planning method performs well across the board for all forms of planning, hence it seems reasonable to investigate AI methods for deciding when to use which fraction of a domain. This could, in principle, include knowledge about uncertainty, temporal relations, and other forms of knowledge at the same time.

There are various engineering benefits connected to the use of domain reasoning, such as re-usability, testing, and validation. Operators that are used in most domains (e.g., move the robot from one location to another) may be automatically adjusted rather than having to be manually reformulated. Finally, to extend an existing general purpose domain with a new context should be easier than creating a new domain from scratch, since many aspects are already present. This is especially true if the underlying robot remains the same.

In this paper we formalize the notion of context-adapted planning domains, and analyze different ways of extracting

the domain from the general purpose domain. All our considerations are independent of a specific planning approach, and should therefore be applicable to any domain definition language. In light of our problem definition, we discuss a series of examples in a more detailed way, drawing freely from the three application examples sketched above.

A Formal Model for Domain Reasoning

A general task planning problem can be expressed as a tuple $\Pi = (\Phi, \mathcal{O}, \delta, \theta)$, where Φ is a set of expressions describing knowledge about the environment. Φ may contain, for instance, current states, projected future states, as well as constraints or rules regulating how the environment works. For practical reasons, we consider the initial state $\mathcal{I} \subseteq \Phi$ and goals $\mathcal{G} \subseteq \Phi$. The set of operators $\mathcal{O}$ describes decisions that the planner can make to change aspects of the environment. Usually, each operator $o \in \mathcal{O}$ has conditions under which it can be applied and a description of how it changes Φ. The function $\delta : 2^{\Phi} \times \mathcal{O} \rightarrow 2^{\Phi}$ applies an operator and creates a new description of the environment. Finally, the function $\theta : 2^{\Phi} \times \mathcal{G} \rightarrow \{True, False\}$ tests if a goal has been achieved in a description of the environment. With this problem definition, planning is usually reduced to a search problem that finds a (partially or totally ordered) set of operators leading to some Φ' in which all goals are achieved.

Both Φ and $\mathcal{O}$ are expressed in terms of a *Domain Definition Language (DDL)* $\mathcal{L}_{\mathcal{D}}$. In addition, we assume a context definition language $\mathcal{L}_{\mathcal{C}}$. A planning domain $\mathcal{D}$ can now be defined as $\mathcal{D} = \Phi \cup \mathcal{O} \in \mathcal{L}_{\mathcal{D}}$. with these ingredients, we can now define four concepts that are not traditionally considered in automated planning:

Context $\mathcal{C} \in \mathcal{L}_{\mathcal{C}}$: circumstances that determine which subset of a domain has to be used. May be static or dynamic (or have both static and dynamic components)

General Purpose Domain $\mathcal{D}^* \in \mathcal{L}_{\mathcal{D}}$: an unfiltered domain that models all possible application domains (may not be consistent as is, due to mutually exclusive requirements for mutually exclusive contexts)

Context Reasoning $f_{\mathcal{C}} : \mathcal{L}_{\mathcal{D}} \rightarrow \mathcal{L}_{\mathcal{C}}$: to determine the relevant context, given all available information

Domain Reasoning $f_{\mathcal{D}} : \mathcal{L}_{\mathcal{D}} \times \mathcal{L}_{\mathcal{C}} \rightarrow \mathcal{L}_{\mathcal{D}}$: to generate a domain for a given context

Using this formal model we will now consider a series of questions that arise in practice.

What does the domain reasoning function do? There are several options here. One option is *variable substitution* (e.g., substitute cultural or personal preferences of the user in Example 1 into operators). Another option is *subset selection*, that is, removing unnecessary or unwanted operators, constraints, or other parts of the general purpose domain. Indeed, HTN planning can be seen as a form of subset selection. Finally, *structure generation/alteration* could be used to assemble operators and/or constraints dynamically for a given context. A move operator in a museum (see Example 2) may have different internal conditions (e.g., stay close to the visitor group) from those of a movement operator that does not include others. A temporally expressive operator could be reduced to a simpler operators if temporal information is not relevant in the current context.

When/how does the context change? *User driven* context changes are caused directly by user interactions with the robot. The *current goal* has a strong impact on which parts of a domain should be considered for planning. Escorting a visitor to an office or giving a guided tour to a visitor are different goals that should use a different domain. In the same way, goals can also limit the background knowledge used by the planner. For instance, going to the supermarket may require a map of an apartment building and part of a city; fetching an object from the kitchen, on the other hand, requires only a map of the apartment. In a similar way, *current/future states*, as well as *background knowledge* can determine the context. Depending on the culture of the owner (see Example 1), a robot may have to solve problems in a different way when the user is alone at home compared to when there are visitors. Cleaning an apartment in a noisy way is not a problem when the owner is not at home, but constraints may apply otherwise (e.g., robot should not clean the room in which the owner is present).

How is context expressed? Context could be expressed in a propositional way (similar to states in classical planning), or in a declarative languages such as Prolog (Bratko, 2000), or Constraint Processing (Dechter, 2003) languages. The latter becomes interesting when inferences about implied context need to be made. Separating $\mathcal{L}_{\mathcal{C}}$ from $\mathcal{L}_{\mathcal{D}}$ is useful conceptually, but in practice they may overlap, as task planners may also require context for decision making (Pecora et al., 2012). Coutaz et al. (2005) argue for treating context as a process rather than a state.

Examples of Domain Reasoning & Context

In this section we go through a detailed set of examples taken from of the possible content of the general purpose domain and how context could influence it. Consider that all of the following paragraphs describe parts of the same general purpose domain used by a robot for household and/or office applications across the globe. We will highlight how substitution, filtering, and structural changes could come into play for each of these aspects of the general purpose domain. Context in this domain may be determined by a multitude of factors such as relevant users, deployment focus (private or business), current and/or relevant locations, current goals or tasks, user background information (country, city, language, cultural background, or the user's role in the environment), and relevant events (e.g., visits, parties, birthdays, important deadlines).

Maps and Objects. The general purpose domain may include a set of maps of varying detail and objects within the environment (e.g., apartment, house, office building, city). Information about locations and objects can be filtered out when it is not relevant to the given context. Filtering could be done by analyzing the structure of the problem (as, e.g., described by Helmert, 2004) or maintaining a knowledge base that describes the relevance of parts of $\mathcal{D}^*$ depending on the context c. A query to this knowledge base for context c could be answered with a domain $\mathcal{D} \subseteq \mathcal{D}^*$. Depending on the chosen form of knowledge representation, the amount of

knowledge that has to be modeled here may vary. An ontology could capture a lot of information that is independent of concrete objects, and could be queried to retrieve the properties of groups of objects. This part of the general purpose domain may also lead to structural variations. There may be special preconditions that only apply to certain sub-classes of objects (e.g., handle with care); in such cases we could consider adapting the structure of a manipulation operator to the object being manipulated.

Information about users. Preferences, cultural background, or the role of a user can be substituted via domain reasoning into operators that involve human-robot interaction. Engaging in a conversation with a user, for instance, requires the robot to approach the user. The approach distance, as well as the topic of conversation and how the conversation is initiated, may depend on the user's cultural background, preferences, and role within the environment. As before, user preferences may lead to changes in operator structure in cases where users have special needs. A robot that would normally navigate between two people who are talking to each other may be considered very rude if the people were using sign language. As a result, the domain reasoner should adapt all operators that involve moving in these circumstances.

General Operators. A general purpose domain would likely include a default set of operators covering task-independent robot capabilities, such as movement and manipulation. Robot movement may change depending on the environment (inside or outside). These changes could be substituted into the existing movement operator. Some locations may have special conditions that restrict movement. The robot may need special permission to enter the archive of an office, some objects may need to be handled with care, or movement may need to be adapted considering who the robot is moving with (see examples above). Crossing a street may require the traffic light to be green, while moving from one room to another requires the door to be open. Considering these issues as structural changes made to the general purpose domain by the domain reasoner is interesting, because it may lead to a situation where the general purpose domain is actually more compact than the extracted domain. One general purpose movement operator, for instance, may lead to a set of concrete operators adapted to the context at hand. Modeling context dependence in the form of preconditions would lead to a large number of variations of the same operator(s), which quickly becomes hard to maintain.

Specialized Operators. As robots become more capable at executing everyday tasks, such as tidying, doing laundry, folding clothes, or shopping, a large number of specialized operators will be added to the general purpose domain. If not relevant, many of these operators can be filtered out. However, domain reasoning could also consider what is relevant for these operators. Cleaning a load of laundry and folding it afterwards, for instance, are local tasks. They should not involve leaving the house (unless some cleaning product needs to be bought or fetched from some other building). The locations involved in planning for shopping depend on the items on the shopping list. A general way to model this relevance (e.g., through an ontology) could be an interesting start for creating a domain reasoner that does not require explicit knowledge of all such details for every object.

Rules and Constraints. There are potentially many rules and constraints that planners should uphold only under specific circumstances. Rules for goal reasoning (Vattam et al., 2013) may cover under which circumstances a planner adds a new goal or changes an existing goal. These rules may be relevant only in specific situations. In a similar way, constraints for social acceptability (Köckemann, Pecora, and Karlsson, 2014) may apply only given a user's cultural background or personal preferences. Enforcing the full set of these rules and constraints may lead to a very difficult or unsolvable problem. If, for instance, in the current context the user is not at home for the entire day, there is no reason for the planner to take into account any constraints that depend on the location of the user. For the same reason, every rule for goal reasoning (e.g., describing when to formulate a goal) that involves the user performing an activity can be disregarded. The structure of these rules and constraints may change as well, depending on the circumstances. How goals are formulated, or which situation possibly violates social acceptability, may vary depending on the context.

Related Work

The problem that we wish to address here is quite general and related to many other aspects of planning research. We attempt here to provide a short overview of research in task planning that is related to domain reasoning. Given length restrictions, this section is necessarily incomplete.

The lack of automated planning in real robot deployments was also pointed out by Alterovitz, Koenig, and Likhachev (2016), who list several research challenges for robot planning that involve planning in the real world (with perception) and with humans (human-aware planning, predictability, understandability). We argue that for many of these issues, some form of domain reasoning as suggested here will be necessary in order to cope with the growing complexity of the robot systems and tasks.

Many planning systems already support some form of domain reasoning by removing unnecessary predicates or operators that cannot contribute to a goal.

Automatic domain abstraction through pre-compilation (Knoblock, 1994) is often used to reduce domain complexity before starting the search for a plan. A sub-set of operators may actually never contribute to achieving certain goals, and can safely be removed. These approaches can be considered as a form of domain reasoning that applies filtering.

There is also a relation between our suggestion and the approach taken by Hierarchical Task Network (HTN) planners (Nau et al., 2005). Methods in HTN planners describe how a combinations of actions can be used to achieve a task. This can also be seen as a form of domain reasoning, where task decomposition leads from a general purpose domain to a specific one. (Hartanto and Hertzberg, 2009) used a description logic reasoner as $f_{\mathcal{D}}$ to complile HTN domains.

Borgo et al. (2016) extend the DOLCE ontology to model global, local, and internal context for creating planning domains for a Reconfigurable Manufacturing System.

Planning for large-scale domains has been addressed by Galindo et al. (2008), who plan using several layers of abstraction. This approach incorporates domain reasoning directly into planning. Individual rooms, for instance, are only considered after planning on the level of abstraction of areas in a building.

The KnowRob project (Tenorth and Beetz, 2013) aimed to provide general robot skills that can be downloaded and executed by any capable platform. In a way, this project can be seen as an effort to bridge the task planning and robotics gap from the robotics side (by making robot capabilities transferable). In contrast, domain reasoning would reach out from the AI side (by dealing with real-world domain complexity).

Depending on what we consider relevant context for a planning system, there is related work to be found on the topics of context recognition and inference (Pecora et al., 2012), or activity recognition. Plan recognition (Carberry, 2001) could be used in a similar way.

Context for reasoning in first-order logic has been considered by McCarthy (1993) to provide a way of adjusting reasoning and the truth of sentences in first-order logic to context. This allows to perform reasoning about other agents' viewpoints, as well as about the implications of hypotheses. The problem we consider in this paper is more specific and driven by practical considerations.

In summary, while there is some work that goes in the direction of domain reasoning, we are not aware of any approach attempting to generalize this idea in order to push for task planners that are useful for robotics. The model for domain reasoning suggested in this paper is independent of the underlying planning approach. Our hope is that studying this general formulation of domain reasoning will contribute to bridging the gap between task planning and robotics.

Conclusion

In this position paper we have argued for general purpose domains and domain reasoning to address the lack of task planning technology in robot applications. Domain reasoning could allow us to address scalability by pruning domain subsets and by possibly limiting planning problems to a lower complexity class (e.g., from temporal planning to classical planning). In addition, maintaining a general purpose domain for a robotic deployment should have a variety of engineering benefits such as testability and extandability.

We have provided a simple formulation of the domain reasoning problem and related concepts, and discussed various questions that arise when introducing the idea of context-dependent domains to planning. We have discussed a series of examples and possible solutions to the context and domain reasoning functions $f_{\mathcal{C}}$ and $f_{\mathcal{D}}$.

We are currently using domain reasoning within the CARESSES[1] project, with the aim to design care robots that adapt to the culture of the person they assist. In future work, we intend to investigate alternative solutions to the domain reasoning problem, and move towards implementation and comparison of different approaches in terms of how well they enable the use of automated task planning for robots.

[1] `http://caressesrobot.org/`.

Acknowlegements. This work was supported by the European Commission Horizon2020 Research and Innovation Programme under grant agreement No. 737858 (CARESSES).

References

Alterovitz, R.; Koenig, S.; and Likhachev, M. 2016. Robot planning in the real world: Research challenges and opportunities. *AI Magazine* 37(2):76–84.

Borgo, S.; Cesta, A.; Orlandini, A.; and Umbrico, A. 2016. A planning-based architecture for a reconfigurable manufacturing system. In *Proc. of ICAPS*, 358–366.

Bratko, I. 2000. *Prolog Programming for Artificial Intelligence*. Addison Wesley.

Carberry, S. 2001. Techniques for plan recognition. *User Modeling and User-Adapted Interaction* 11:31–48.

Coutaz, J.; Crowley, J. L.; Dobson, S.; and Garlan, D. 2005. Context is key. *Commun. ACM* 48(3):49–53.

Dechter, R. 2003. *Constraint processing*. Elsevier Morgan Kaufmann.

Galindo, C.; Fernández-Madrigal, J.-A.; González, J.; and Saffiotti, A. 2008. Robot task planning using semantic maps. *Robotics and autonomous systems* 56(11):955–966.

Hartanto, R., and Hertzberg, J. 2009. On the benefit of fusing dl-reasoning with htn-planning. In Mertsching, B.; Hund, M.; and Aziz, Z., eds., *KI 2009: Advances in Artificial Intelligence*, 41–48. Berlin, Heidelberg: Springer Berlin Heidelberg.

Helmert, M. 2004. A Planning Heuristic Based on Causal Graph Analysis. In *Proc. of ICAPS*, 161–170.

Knoblock, C. A. 1994. Automatically generating abstractions for planning. *Artif. Intell.* 68(2):243–302.

Köckemann, U.; Pecora, F.; and Karlsson, L. 2014. Grandpa hates robots — interaction constraints for planning in inhabited environments. In *Proc. of AAAI*.

McCarthy, J. 1993. Notes on Formalizing Context. In *Proc. of IJCAI*, 555–562.

Nau, D.; Au, T.-C.; Ilghami, O.; Kuter, U.; Wu, D.; Yaman, F.; Munoz-Avila, H.; and Murdock, J. W. 2005. Applications of SHOP and SHOP2. *IEEE Intelligent Systems* 20(2):34–41.

Pecora, F.; Cirillo, M.; Dell'Osa, F.; Ullberg, J.; and Saffiotti, A. 2012. A Constraint-Based Approach for Proactive, Context-Aware Human Support. *Ambient Intelligence and Smart Environments* 4(2):347–367.

Tenorth, M., and Beetz, M. 2013. KnowRob: A knowledge processing infrastructure for cognition-enabled robots. *Int Journal of Robotics Research* 32(5):566–590.

Vattam, S.; Klenk, M.; Molineaux, M.; and Aha, D. W. 2013. Breadth of Approaches to Goal Reasoning : A Research Survey. *Goal Reasoning: Papers from the ACS Workshop* 111.

Improving Trajectory Optimization using a Roadmap Framework

Siyu Dai, Matthew Orton, Shawn Schaffert, Andreas Hofmann, Brian Williams
Massachusetts Institute of Technology
77 Massachusetts Avenue
Cambridge, MA 02139

Abstract

We present an evaluation of several representative sampling-based and optimization-based motion planners, and then introduce an integrated motion planning system which incorporates recent advances in trajectory optimization into a sparse roadmap framework. Through experiments in 4 common application scenarios with 5000 test cases each, we show that optimization-based or sampling-based planners alone are not effective for realistic problems where fast planning times are required. To the best of our knowledge, this is the first work that presents such a systematic and comprehensive evaluation of state-of-the-art motion planners, which are based on a significant amount of experiments. We then combine different stand-alone planners with trajectory optimization. The results show that the combination of our sparse roadmap and trajectory optimization provides superior performance over other standard sampling-based planners' combinations. By using a multi-query roadmap instead of generating completely new trajectories for each planning problem, our approach allows for extensions such as persistent control policy information associated with a trajectory across planning problems. Also, the sub-optimality resulting from the sparsity of roadmap, as well as the unexpected disturbances from the environment, can both be overcome by the real-time trajectory optimization process.

1 Introduction

Robotic systems deployed in the real world have to contend with a variety of challenges: light-weight arms or those with series elastic actuators shake when they move, wheels slip, IMUs drift, lidars do not reflect off glass doors, structure light sensors fail outdoors, body-mounted cameras get occluded by appendages, and humans in the environment move quickly and in unpredictable manners. These systems cannot spend an unbounded amount of time searching for an optimal motion plan – a plan that will ultimately be invalidated by the next sensor reading, a change in the environment, or a slipping wheel. Instead, a motion planner must find solutions rapidly even at the expense of optimality. A motion planner that operates quickly allows the robot to truly react to new information and to feel interactive to humans. In addition to quick generation, these plans need to account for the system's dynamics, be robust to disturbances, and operate faithfully within a higher-level task plan.

The problem of moving a robot safely and efficiently in uncertain environments, however, is a challenging one. Often, there is significant complexity with path planning alone, due to the robot and environment geometry. Coupled with dynamic obstacles and sensor noises, the planning problem only becomes more challenging. Additionally, accounting for dynamics and actuation limits becomes untenable within many frameworks.

Due to the complexity of the overall problem, current motion planning and execution systems do not adequately address all of these challenges simultaneously: they often assume the environment is static, or at least, predictable; many do not simultaneously support collision avoidance and complex dynamics; and many generate completely new trajectories for each planning problem instead of allowing for persistent control policy information associated with a trajectory across planning problems.

We have previously developed *Chekhov*, a reactive motion execution system that addresses these requirements (Hofmann et al. 2015). Chekhov avoids obstacles, incorporates dynamic models and control policies, and observes temporal constraints. However, because Chekhov uses a roadmap approach (Kavraki et al. 1996), and because robotic motion planning state spaces are typically very large, Chekhov's coverage of the operating workspace is very sparse. As a result, trajectories produced by Chekhov are sub-optimal. In this work, we address this limitation by leveraging recent advances in obstacle-aware trajectory optimization (Schulman et al. 2014). First, we show that recently developed trajectory optimization techniques, which include some capability to avoid obstacles, are not, by themselves adequate for typical problems. We then show that by formulating trajectory optimization problems based on the Chekhov roadmap, the problems associated with using trajectory optimization alone are solved. Further, we show that the optimized trajectory is superior to (more optimal than) the trajectory produced by the roadmap alone. Thus, the combination results in superior performance in terms of feasibility, optimality, and also planning time. Our future goal is to integrate trajectory optimization into the complete Chekhov motion execution system, so it is essential that the trajectory optimization approach is able to incorporate dynamics and temporal constraints, as well as being able to react quickly to disturbances in planning tasks.

2 Related Work

Optimization-based robotic motion planners are attracting more and more attention with the increasing complexity of robots and environments. Covariance Hamiltonian Optimization for Motion Planning (CHOMP) (Ratliff et al. 2009), Stochastic Trajectory Optimization for Motion Planning (STOMP) (Kalakrishnan et al. 2011), Incremental Trajectory Optimization for Real-time Replanning (ITOMP) (Park, Pan, and Manocha 2012) and TrajOpt (Schulman et al. 2013) are several state-of-the-art optimization-based planners. In this work, we focus on the TrajOpt planner for three reasons. First, the convex-convex collision checking method used in TrajOpt can take accurate object geometry into consideration, shaping the objective to enhance the ability of getting trajectories out of collision. In contrast, the distance field method used in CHOMP and STOMP consider the collision cost for each exterior point on a robot (Zucker et al. 2013), which means two points might drive the objective in opposite direction. Second, the sequential quadratic programming method used in TrajOpt can better handle deeply infeasible initial trajectories than the commonly used gradient descent method (Schulman et al. 2013). Third, customized differential constraints, for example velocity constraints and torque constraints, can be incorporated in TrajOpt. This is an important consideration for Chekhov which aims at building a motion execution system that incorporates system dynamics models and control policies, while respecting additional temporal constraints.

Despite the advantages of optimization-based planners, they are not stand-alone planners and their performance is very sensitive to the quality of initializations. Also, numerical trajectory optimization often suffers from the problem of getting stuck in high-cost local optima. Therefore, a natural thought to improve the performance of optimization-based planners is to combine them with global planners. Some existing work, for example Luna et al. (2013) and Campana et al. (2015), has proposed online path shortening methods for sampling-based planners. The effect of optimization in those approaches is mostly limited to trajectory smoothing and shortening, and can't account for real-time obstacle avoidance and dynamics constraints. Therefore, those modified sampling-based planners still share the typical slow planning times with other common sampling-based planners. Other researches (Park et al. 2015) have presented a combined roadmap and trajectory optimization planning algorithm. However, they additionally focused on avoiding singularities in redundant manipulators and meeting Cartesian constraints resulting in relatively long planning times. In comparison, our approach aims at fast reactive real-time planning in practical planning scenarios, and extensive experiment results in Section 5 show that our approach reaches this goal.

3 Problem Statement and Approach

The problem solved by Chekhov is to quickly plan and execute robot motions that accomplish a task specified by a set of temporal and spatial constraints. The inputs to Chekhov can change quickly and unexpectedly with time while the motion is being executed. For practical applications, changes fall into three categories: 1) the current state of the robot changes; 2) the goals to be achieved change; and 3) an environment obstacle moves in a way that affects the robot. Thus, we define a *disturbance* as such an unexpected change to task goals, environment, or robot state. The system we aim at achieving should react, effectively, instantaneously to disturbances; it should act as if it always, "instantly" knows what to do, for any combination of goals and circumstances. This fast reaction is key to providing robots the capability to operate effectively in unstructured, uncertain, fast-changing environments.

We make a number of key assumptions in our approach. Although these assumptions may seem restrictive, we believe that they are consistent with a large class of practical robotic manipulation problems. First, we assume that the manipulation workspace is characterized by a limited set of pre-grasp poses. Second, we assume that the pre-grasp to grasp motion is short, and is best handled by visual and force servoing loops, rather than open-loop planners. Third, we assume that the collision environments are not overly complex. We are not trying to solve "piano mover" problems like reaching into tunnels or through a maze of obstacles. Instead, we assume that there is a small set of potential obstacles, such as a workpiece, a table, another robot, or a human, but that some of these may move. The emphasis here is on achieving fast performance in typical, practical situations.

We endeavor to achieve a fast, reactive capability by using a roadmap-based approach. The roadmap represents the static collision-free space, and therefore, is re-used across planning instances. For each pair of nodes in the roadmap, k shortest paths ($k \geq 1$) are calculated and stored, so that when dynamic obstacles invalidate some of the edges in the roadmap, the probability of finding a collision-free path for the planning task can be improved as we increase k. Our approach features three key innovations from the previous Chekhov. First, as stated in Section 1, we extend the roadmap approach used previously in Chekhov by incorporating recent advances in obstacle-aware trajectory optimization (Schulman et al. 2014) in order to improve solution optimality and fast reaction to disturbances. Our goal here is to consider the entire solution space, rather than the very sparse one provided by the roadmap. Second, we use a set of practically relevant test environments, rather than random ones or ones that are artificially challenging. To this end, we have developed three new environments that represent typical motion planning scenarios. We have also included a fourth environment developed previously in the motion planning community. Third, we use semantic information about the environment to help guide the construction of the roadmap to favor inclusion of poses that are known to be useful. Utilizing semantic information includes making a basic distinction between static and dynamic obstacles. It also includes utilizing knowledge of objects in the environment in order to generate pre-grasp poses that will be useful for manipulating them.

4 Implementation

In order to test and compare the performance of different path planners, we use four representational environments: a "tabletop with a pole", a "tabletop with a container", a "kitchen" and a "shelf with boxes" environment. We choose environments that are representative of different application domains rather than using an environment with randomly-placed obstacles because our goal is to develop a path planner that operates quickly and provides short paths for real world applications. The kitchen environment comes from the TrajOpt package, whereas, we designed the remaining three. The "tabletop with a pole" environment, shown in Fig. 1, is a simple tabletop pick-and-place task environment, with a slender pole in the middle of the table and a box on each side of the pole. All the planners can easily handle most planning queries in this environment. The "tabletop with a container" environment is similar, but has a large container on the table with both boxes inside and outside of it. The "kitchen" environment models a typical kitchen scenario which is common in household domains. The "shelf with boxes" environment, shown in Fig. 2, is a 7-level shelf environment with boxes on each level of the shelf, which is a common scenario in the logistic application domain. This scenario is known to be hard because of the relatively large total number of obstacles and the narrow space between them.

For each environment, we generate 5000 feasible planning tests by randomly sampling 5000 start and target end-effector pose pairs that are collision-free and kinematically feasible. For each sampled point, both the joint-space position and the end-effector location and orientation are recorded. For each experiment trial, planners are provided with the starting joint-space position and the goal end-effector pose. We specify the goal in workspace to give planners the opportunity to find different joint-space solutions to the planning problem. We have ensured that all test cases have a solution by executing all the planners on each test case, and re-sampling start and goal points when no planners could find a solution. All the test cases, including the environment and poses, are saved so that they can easily be repeated in the future.

In our experiments, we use the Baxter robot (RethinkRobotics) with its 7-DOF left arm as the manipulator. Based on our initial tests, TrajOpt works quite similarly on other manipulators, so here we take the left arm as an example to implement the in-depth analysis.

In addition to the discrete-time collision costs approach, the TrajOpt algorithm also provides a "swept-out volume" method in order to ensure continuous-time collision checking (Schulman et al. 2013). However, during our experiments, we find that even when the continuous-time collision cost is utilized, collision can still occur in-between waypoints, and it is not obvious how to use TrajOpt's reported collision cost to detect collisions consistently since large cost values can indicate either a collision or just a waypoint close to an obstacle. Hence, rather than simply referring to cost values returned by TrajOpt, in our experiments we also implement an independent collision checking process for the returned trajectory to test continuous-time safety. In particular, we interpolate 100 intermediate waypoints between each pair of adjacent waypoints and collision check each point using the OpenRAVE collision checking. For our work, we consider this fine-grained discrete-time collision check to approximate a continuous-time collision check sufficiently well.

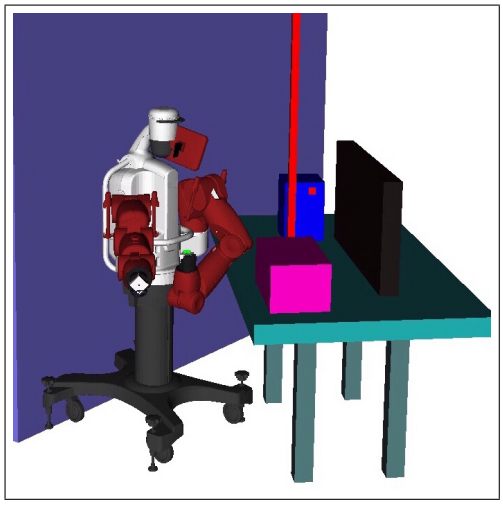


Figure 1: The "tabletop with a pole" environment

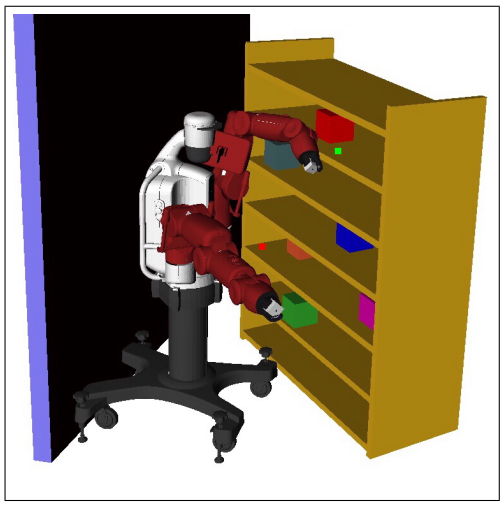


Figure 2: The "shelf with boxes" environment

As introduced previously, we can use semantic information about the environment to improve roadmap construction and thus, the motion planning result. *Semantic Object Maps* (SOM) (Pangercic et al. 2012) provide a representation of such information, including an overall ontology, and also part composition and articulation. This can be used, for example, to represent how a refrigerator door, or a desk drawer opens and closes, which can be used to generate precise pre-grasp poses for the open and closed positions. This is important as it guarantees that required poses will exist directly in the roadmap. Additionally, semantic information can be used to bias sampling of poses during roadmap construc-

tion to favor areas of interest. For example, the area above a desktop is more likely to contain objects of interest and hence should get more nodes than the (free) area under the desktop.

Although our tube-based roadmap architecture supports dynamics and temporal constraints (Hofmann and Williams 2017), our experiments here mainly focus on kinematic planning tasks for robot manipulation considering obstacle avoidance. We have already incorporated customized constraints into TrajOpt which respect system dynamics such as torque constraints, velocity limits and acceleration limits. Experiments on planning tasks with dynamics and temporal constraints are beyond the focus of this paper but will be further explored in our future research. Furthermore, for the purposes of evaluating key aspects of our approach, we have assumed that all obstacles in the test environments are static. We focus here on static rather than dynamic obstacles because static obstacles occupy the majority of the workspace in many practical applications. As stated in Section 3, we handle dynamic obstacles through storing redundant roadmap paths and by coupling these paths with fast optimization from TrajOpt. Therefore, experiments with dynamic obstacles can be straightforwardly extended from our current experiments.

5 Experiments and Results

In Section 5, we provide experiment results and performance evaluation of five standard path planners (OpenRAVE BasicRRT, OMPL LazyPRM (Bohlin and Kavraki 2000), OMPL PRM* (Karaman and Frazzoli 2011), OMPL RRT* (Karaman and Frazzoli 2011), and TrajOpt with a straight-line joint-space initialization).In Section 5, we show the results and evaluation of four combined planners which pass in a sampling-based planner solution as an initial path (or "seed path") to TrajOpt. Their performance is analyzed and compared in terms of failure-rate, average joint-space path length and average algorithm runtime. Additionally, we also implemented our own roadmap planner which can provide seed paths to TrajOpt – the results and evaluation of which is described in Section 5. Each of the experiments includes 5000 test queries and is conducted in all the four environments mentioned in Section 4, but for brevity, most of the tables only provide the results summary for the "tabletop with a pole" environment and the "shelf with boxes" environment, which qualitatively represent the easiest and hardest environments for the planners, respectively.

Limitation of current planners

Currently, popular path planners include sampling-based path planners, which can operate stand-alone, and trajectory-optimization type path planners, which modify a seed trajectory and return the optimized solution. However, in practical application scenarios, each of those planners has their own disadvantages. The sampling-based path planners are usually not fast enough for real-time planning tasks, and some of them (like PRM and PRM*) can not incorporate dynamic constraints. Meanwhile, trajectory-optimization type planners locally optimize a path, thus their performance depends much on the quality of seed trajectories. When provided with a bad seed, trajectory-optimization type planners can have high collision-rates or get stuck in local optima. This section provides a systematic empirical study on some sampling-based planners and a trajectory-optimization type planner, TrajOpt (Schulman et al. 2013), comparing their performance in terms of failure-rate, average joint-space path length, and average algorithm runtime.

We compared five off-the-shelf planners (OpenRAVE BasicRRT, OMPL LazyPRM, OMPL PRM*, OMPL RRT* and TrajOpt with straight-line joint-space initialization) on all 5000 cases for each environment. For the sampling-based planners, we set the runtime upper bound for generating a plan to 300s. The runtime upper bound was choosen, after initial testing, to reduce the failure rates of the optimal sample-based planners (RRT* and PRM*). For example, if we set the RRT* runtime bound to 60s, the failure rate for the "shelf with boxes" environment will be as high as 70%.

TrajOpt works by formulating the kinematic motion planning problem as a non-convex optimization problem over a $T \times K$-dimensional vector, where T is the number of timesteps and K is the number of degrees of freedom (Schulman et al. 2013). Hence every trajectory in TrajOpt is made up of T waypoints, where the number T is set by the user. We ran 16 sets of tests, each with an increasing total number of waypoints, and observed that TrajOpt runtime increased approximately linearly with number of waypoints while the collision rate dropped quickly with more waypoints. For our tests on TrajOpt with straight-line seed trajectories, we found that setting $T = 30$ provided a good balance between low collision rates and algorithm runtimes. Henceforth, in this subsection, we use 30 total waypoints (including the start and target waypoints).

Table 1 summarizes the experiment results in the easiest environment, "tabletop with a pole", and the hardest environment, "shelf with boxes", in terms of failure rate, average runtime and average joint-space path length. The reported failure rate encompasses all possible failure modality (i.e., not finding a solution or returning a solution in collision). Since TrajOpt will always return a "solution" even if the optimization fails, we log a failure when our (secondary) collision checker determines the solution to be in collision; for sampling-based planners, failure rate is represented by the percentage of cases where the planner failed to return a solution.

If we compare the failure rate of different planners in Table 1, we can see that, both in the relatively easy "tabletop with a pole" environment and in the relative hard "shelf with boxes" environment, TrajOpt fails more frequently to find collision-free solutions than any other planners. If we compare the four sampling-based planners, it can be observed that all the four planners find collision-free solutions for most of the cases in the simple "tabletop with a pole" environment. In contrast, in the complicated "shelf with boxes" environment, RRT and LazyPRM show relatively better solution-finding performance, whereas the optimal planners RRT* and PRM*, even though provided 300s runtime, still fail frequently. From the "average runtime" column in Table 1, it can be observed that the sampling-

Environments	Planners[1]	Failure Rate[2]	Average Runtime (s)[3]	Average Path Length (rad)
Tabletop with a Pole	RRT	2.30%	17.88	0.77
	LazyPRM	0.22%	7.32	1.76
	RRT*	5.32%	300.19	0.63
	PRM*	1.00%	300.71	0.79
	TrajOpt	17.38%	0.56	0.71
Shelf with Boxes	RRT	10.00%	63.86	1.06
	LazyPRM	16.94%	63.85	2.08
	RRT*	26.78%	300.37	0.93
	PRM*	24.34%	300.79	1.16
	TrajOpt	32.06%	1.59	1.51

[1] For each planner in each environment, 5000 planning tasks are tested and the data shown in this table are averaged from the 5000 results.

[2] For TrajOpt with a straight-line seed, failure rate is the percentage of cases where the solution is in collision; for sampling-based planners, failure rate is the percentage of cases where the sampling-based planner failed to find solution.

[3] The runtime upper-bound is set to 300s. RRT* and PRM* always use the full amount of time – the small deviation from 300s shown in the table is due to small timing errors during simulation.

Table 1: Evaluation of Current Sampling-based and Trajectory Optimization Planners

based planners require too much time for most practical path planning applications. In the case of the optimal planners (RRT* and PRM*), they take all the given time to approximate the optimal solution, therefore their average runtime is always around 300s. Even for LazyPRM, 7.32s in the simple environment and 63.85 in the complicated environment is infeasible for real-time reaction to disturbances in planning tasks. In terms of average path length, optimal planners have noticeable advantages in finding shorter solutions, especially in harder environments. Among the remaining planners, LazyPRM tends to return longer solutions, which is reasonable due to the intrinsic mechanism of lazy searching algorithms. TrajOpt performance in path length is comparable to sampling-based planners, especially in relatively easy environments.

In conclusion, although sampling-based planners are good at avoiding collision, they often take too long for practical application to find a solution. In contrast, TrajOpt shows good performance in terms of runtime, but the high collision-rate makes it an unsatisfactory practical planner.

TrajOpt performance with a collision-free seed

The way TrajOpt works indicates its sensitivity and dependency on the initialization condition (Schulman et al. 2013). Therefore, we propose that the performance of TrajOpt can be dramatically improved if we pass in a collision-free trajectory as a seed instead of using the joint-space straight-line seed. Based on the sampling-based planner experiment results from Section 5, we conduct systematic tests on TrajOpt's performance when provided with a sampling-based planner solution as a seed trajectory. For the cases where a sampling-based planner found a solution, we pass in the solution as the seed trajectory to TrajOpt and record the TrajOpt runtime, solution path length, and collision rate.

TrajOpt algorithm requires the number of waypoints in the solution trajectory to be the same as in the seed. Therefore, if we pass in seeds directly from sampling-based planners without any pre-processing, the number of waypoints in different cases will fluctuate drastically. As mentioned in Section 5, TrajOpt runtime increases approximately linearly as the number of waypoints increases, which means the variation of waypoint numbers will influence runtime. Additionally, seeds taken directly from the sampling-based planners with a fewer number of waypoints will results in higher collision rates after processing by TrajOpt than those with more waypoints. This is because such cases usually have longer edges in-between waypoints and are more likely to have seed paths that are very close to obstacles. Our tests show that TrajOpt has a much weaker ability to deal with edge collisions than with waypoint collisions, and it is likely to push path edges into obstacles when shortening and smoothing the trajectory. Hence, before passing the seed paths into TrajOpt, we sample them by setting a upper bound of 0.16 rad for the distance between adjacent waypoints. This preprocessing dramatically reduced the collision rate of TrajOpt solutions, as well as narrowing down the variance of TrajOpt's runtime among different cases. Inevitably, the average TrajOpt runtime is increased because of more waypoints after sampling the seed, but it is still generally under 1s, which is acceptable for real-time planning tasks.

The performance of this combined "seed + TrajOpt" planner is shown in Table 2. Comparing the TrajOpt runtime column in Table 2 and the straight-line seed TrajOpt runtime in Table 1, we see that when provided with a good seed, the TrajOpt runtime generally decreased. Specializing to the cases where TrajOpt with a straight-line seed failed to push the trajectory out of collision, we found a 50% - 70% runtime drop after provided with sampling-based planners' solutions as initializations. Although a small percentage of cases end up in collision when TrajOpt is smoothing and optimizing the seeds, if we compare the "average path length" column in Table 1 and Table 2, an obvious improvement in average joint-space path length is observed. After comprehensively comparing TrajOpt's performance with a sampling-based planner seed and with a straight-line seed, we see that TrajOpt's performance improves tremendously in terms of both success rate and optimization time when provided with a collision-free seed. However, according to the "average runtime" for combined planners shown in Table 2, it is not feasible to use sampling-based planners as seed planners for practical path planning tasks. Thus, the challenge becomes how to generate a good enough seed quickly.

TrajOpt with Standard Sampling-based Planner Seed and Roadmap Seed

The core of the roadmap framework for Chekhov is a simplified PRM variant combined with a cache of all-pair-shortest-paths (APSP) solutions. The roadmaps are constructed by randomly sampling points in joint space until a pre-defined number of collision-free points have been sampled. The

Environments	Seed Planners	Average TrajOpt Runtime (s)	Seed + TrajOpt Planner		
			Average Runtime (s)[1]	Average Path Length (rad)	Collision Rate[2]
Tabletop with a Pole	RRT	0.63	18.51	0.70	1.29%
	LazyPRM	0.98	8.30	1.28	0.12%
	RRT*	0.29	300.48	0.54	0.02%
	PRM*	0.36	301.07	0.64	0.10%
Shelf with Boxes	RRT	0.92	64.78	0.98	4.20%
	LazyPRM	1.36	65.21	1.60	1.57%
	RRT*	0.46	300.83	0.81	1.17%
	PRM*	0.67	301.46	0.95	1.98%

[1] Sum of sampling-based seed planner runtime (as shown in Table 1 column 4) and TrajOpt runtime averaged from 5000 test cases.
[2] Continuous-time collision rate.

Table 2: Performance of the Combined "Sampling-based Seed + TrajOpt" Planner

Environments[1]	Failure Rate[2]	Average Runtime (s)	Average Path Length (rad)	Best Average[3] (rad)
Tabletop with a Pole	0.18%	0.14	1.24	0.63
Tabletop with a Container	0.76%	0.18	1.32	0.80
Kitchen	1.92%	0.38	1.29	0.71
Shelf with Boxes	12.06%	0.39	1.30	0.93

[1] In each environment, roadmap performance is tested on 5000 planning tasks and the data shown in this table are averaged from the 5000 results.
[2] For these roadmaps, failure occurs when no collision-free straight-line connection was found to an existing point on the roadmap from the start or goal pose of a test case.
[3] Best average is the shortest average path length between all tested sampling-based planners in that environment. Shown here to provide context for the roadmap performance.

Table 3: Roadmap Performance in All Environments

sampling is uniform over the four most proximal joints of the robot, and fixed values are assigned to the remaining joints for all nodes. This approach is taken to more completely cover the workspace with random samples in joint space. For the tests in Table III and Table IV, the roadmaps start out with 1000 collision-free nodes. Then, each node is connected to the k nearest neighbors for which collision-free edges exist. For the tests below, $k = 10$ is used. The resulting graph is pruned of any nodes and edges disconnected from the largest subgraph. For the environments tested, no more than five of the 1000 points were disconnected from the main subgraph. Then an APSP solution set is constructed for the pruned roadmap and stored for rapid shortest path queries.

Table III shows the performance of the roadmap planner for all four tested environments. The remaining two environments omitted in Table 1 and Table 2 are also included to emphasize the difficulty of the "shelf with boxes" environment relative to realistic environments. It makes sense that it is difficult to establish collision-free straight-line connections to randomly sampled points in the roadmap when the environment contains narrow shelves with objects inside them. That being said, tests were conducted to observe the failure rates of roadmaps in different environments relative to the number of randomly sampled points in the roadmap. As the number of randomly sampled points increased, we observed significant improvement in how often the roadmap was connected to in all environments, particularly in the "shelf with boxes" environment. This leads us to believe that it will not be difficult to develop more intelligent sampling methods that allow roadmaps to more effectively cover all areas of interest within an environment.

If we compare the results in Table III to those in Table I, we can see that, in terms of failure rate, our roadmap planner performs comparably or better than all tested sampling-based planners. In the most difficult environment, only RRT was able to produce a solution more often than our roadmap planner. In addtion to failure rate, our roadmap planner's average runtime is substantially better than the sampling-based planners' in all cases. It is faster by more than an order of magnitude in most observed cases. This is a result of caching the APSP solution set for fast queries. Additionally, it should be noted that the roadmap planner constructs the roadmap for each environment a priori whereas LazyPRM constructs a new roadmap online for each case in our tests. For path length, the roadmap planner performs worse than the optimal planners and RRT, but better than LazyPRM. In general with roadmap based planners, the sparsity of the roadmap restricts ability to obtain short paths. With only 1000 nodes, we consider the roadmaps we are using to be relatively sparse for the workspace. That being said, the roadmap planner generates direct, collision-free paths compared to the off the shelf sampling-based planners. Since these paths are just seeds for TrajOpt and their lengths are well within an order of magnitude of one another, the discrepancies in path length are not a concern for us.

Table IV shows a comparison of solutions produced by TrajOpt when traditional sampling-based planners are used versus our roadmap planner. Many of the observations that can be made from this table reinforce observations made from comparing Table III to Table I. Something new to note is that when the roadmap planner produces a solution, TrajOpt in turn produces a collision-free trajectory more than 98% of the time. Additionally, these optimized trajectories are on average more than 10% shorter than their corresponding seed trajectories. Figure 3 shows the four proximal joints for three different trajectories to help visualize the improvments TrajOpt is making on the seed trajectories. The solid lines are the roadmap seeds and the dashed lines are the outputted trajectories by TrajOpt when provided those seeds. From Figure 3 we can see that TrajOpt fulfilled the task of smoothing and shortening the sub-optimal trajectories produced by the Chekhov roadmap. This result is significant because, as a start, it proves that TrajOpt can effectively optimize the roadmap solutions for kinematic planning prob-

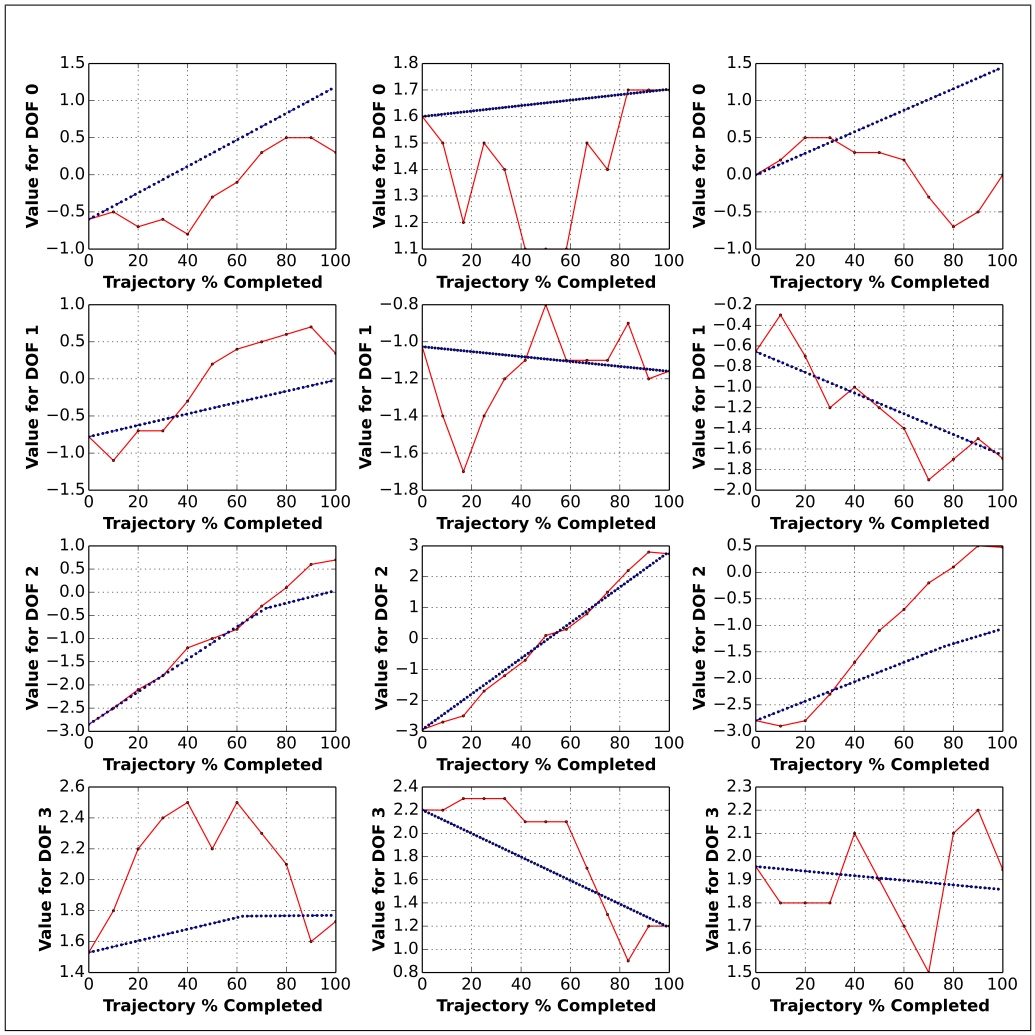


Figure 3: Roadmap seed trajectories shown with corresponding trajectories optimized by TrajOpt to illustrate improvement on the seed. The solid lines are the roadmap seeds and the dashed lines are the outputted trajectories by TrajOpt when provided those seeds.

lems. Therefore, when we fully incorporate all the dynamics and temporal constraints with TrajOpt, we are optimistic that TrajOpt can also fulfill the task of optimizing trajectories for the whole Chekhov motion and execution framework.

The difference in average runtime of the different seed planner coupled with TrajOpt is most notable for highlighting the performance improvements provided by our roadmap planner, but runtime as a metric does not reveal the whole picture for many of these planners. As noted earlier, the optimal planners like RRT* will always use the full allotted time but may have a good non-optimal solution far sooner than that. Also, in our test cases, LazyPRM constructs its roadmap online for one time use and then searches for a path in that roadmap. In general, a PRM does not lend itself to single-query problems. Our roadmap planner precomputes the roadmap and APSP solutions, but is also essentially a PRM. It would be interesting to compare the performance of our roadmap planner to faster RRT variants, but

Environments	Seed Planners	Average TrajOpt Runtime (s)	Average Seed Length (rad)	Seed + TrajOpt Planner		
				Average Runtime (s)[1]	Average Path Length (rad)	Collision Rate[2]
Tabletop with a Pole	RRT	0.63	0.77	18.51	0.70	1.29%
	LazyPRM	0.98	1.76	8.30	1.28	0.12%
	RRT*	0.29	0.63	300.48	0.54	0.02%
	Roadmap	0.45	1.24	0.59	0.82	0.06%
Shelf with Boxes	RRT	0.92	1.06	64.87	0.98	4.20%
	LazyPRM	1.36	2.08	65.21	1.60	1.57%
	RRT*	0.46	0.93	300.83	0.81	1.17%
	Roadmap	0.61	1.30	1.00	1.02	1.98%

[1] Sum of seed planner runtime and TrajOpt runtime averaged from 5000 test cases.
[2] Continuous-time collision rate.

Table 4: TrajOpt Seeded with Sampling-based Planner Solution compared to Roadmap Solution

it is clear to us that the speed provided by querying precomputed solutions from a PRM of some form outweighs any optimization to be had in online search, especially as system dynamics are factored in.

Overall, our roadmap planner performs as well as if not better than the off the shelf sampling based planners we tested. The performance metrics used are failure rate, average runtime, and average path length. Since one of our main goals is to develop a reactive motion execution system that can "instantly" replan when disturbances occur, average runtime is where we are most concerned with improvement. Fortunately, average runtime is where we saw the greatest improvement when using our roadmap planner to provide seed solutions rather than using other traditional sampling-based planners. Although we are currently not using dynamic obstacles in our experiments, our average online planning time leaves us optimistic that our planner will be able to handle disturbances in planning tasks with fast reaction.

6 Discussion

Our results show the benefit of extending the Chekhov roadmap approach with the TrajOpt algorithm. The speed of both approaches is preserved, and meanwhile the combination produces more optimal solutions than the roadmap approach alone and with less failure than the TrajOpt approach alone. The average runtime of under 1 sec and the success rate of above 98% in practical application scenarios show that our approach can handle practical planning tasks with fast reaction. We are currently distinguishing static from dynamic obstacles to the extent that the roadmap is constructed to not collide with the static obstacles in the environment, but dynamic obstacles introduced at runtime will likley obstruct nodes and edges in the roadmap. Incorporating incremental search algorithms to account for these obstructions is an active area of research in our group. We would also like to improve the our ability to connect to our roadmaps in difficult environments, but since there are already techniques that have been shown to improve roadmap coverage with sparse sampling (Siméon, Laumond, and Nissoux 2000), we are not currently researching new approaches to the problem.

Another active area of research in our group concerns the interaction of dynamics and temporal constraints in integrated motion and task planning problems. We have previously utilized Chekhov's roadmap framework to incorporate dynamics and temporal constraint information (Hofmann et al. 2015), (Hofmann and Williams 2017), and we plan to extend this work using recent advances in control theory such as Sum of Squares (Majumdar and Tedrake 2017) programming. This is important for challenging underactuated applications like underwater mobile manipulators operating in the proximity of reefs, and walking robots.

References

Bohlin, R., and Kavraki, L. E. 2000. Path planning using lazy prm. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 1, 521–528. IEEE.

Campana, M.; Lamiraux, F.; and Laumond, J.-P. 2015. A simple path optimization method for motion planning.

Hofmann, A. G., and Williams, B. C. 2017. Temporally and spatially flexible plan execution for dynamic hybrid systems. *Artificial Intelligence* 247:266–294.

Hofmann, A.; Fernandez, E.; Helbert, J.; Smith, S.; and Williams, B. 2015. Reactive integrated motion planning and execution. AAAI Press/International Joint Conferences on Artificial Intelligence.

Kalakrishnan, M.; Chitta, S.; Theodorou, E.; Pastor, P.; and Schaal, S. 2011. Stomp: Stochastic trajectory optimization for motion planning. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 4569–4574. IEEE.

Karaman, S., and Frazzoli, E. 2011. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research* 30(7):846–894.

Kavraki, L. E.; Svestka, P.; Latombe, J.-C.; and Overmars, M. H. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation* 12(4):566–580.

Luna, R.; Şucan, I. A.; Moll, M.; and Kavraki, L. E. 2013. Anytime solution optimization for sampling-based motion planning. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, 5068–5074. IEEE.

Majumdar, A., and Tedrake, R. 2017. Funnel libraries for real-time robust feedback motion planning. *The International Journal of Robotics Research* 36(8):947–982.

Pangercic, D.; Pitzer, B.; Tenorth, M.; and Beetz, M. 2012. Semantic object maps for robotic housework-representation, acquisition and use. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, 4644–4651. IEEE.

Park, C.; Rabe, F.; Sharma, S.; Scheurer, C.; Zimmermann, U. E.; and Manocha, D. 2015. Parallel cartesian planning in dynamic environments using constrained trajectory planning. In *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*, 983–990. IEEE.

Park, C.; Pan, J.; and Manocha, D. 2012. Itomp: Incremental trajectory optimization for real-time replanning in dynamic environments. In *ICAPS*.

Ratliff, N.; Zucker, M.; Bagnell, J. A.; and Srinivasa, S. 2009. Chomp: Gradient optimization techniques for efficient motion planning. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, 489–494. IEEE.

RethinkRobotics. Baxter. http://www.rethinkrobotics.com/baxter/.

Schulman, J.; Ho, J.; Lee, A. X.; Awwal, I.; Bradlow, H.; and Abbeel, P. 2013. Finding locally optimal, collision-free trajectories with sequential convex optimization. In *Robotics: science and systems*, volume 9, 1–10. Citeseer.

Schulman, J.; Duan, Y.; Ho, J.; Lee, A.; Awwal, I.; Bradlow, H.; Pan, J.; Patil, S.; Goldberg, K.; and Abbeel, P. 2014. Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research* 33(9):1251–1270.

Siméon, T.; Laumond, J.-P.; and Nissoux, C. 2000. Visibility-based probabilistic roadmaps for motion planning. *Advanced Robotics* 14(6):477–493.

Zucker, M.; Ratliff, N.; Dragan, A. D.; Pivtoraiko, M.; Klingensmith, M.; Dellin, C. M.; Bagnell, J. A.; and Srinivasa, S. S. 2013. Chomp: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research* 32(9-10):1164–1193.

An Anytime Algorithm for Task and Motion MDPs

Siddharth Srivastava[*‡], **Nishant Desai**[†], **Richard Freedman**[§], **Shlomo Zilberstein**[§]
[‡]Arizona State University, [†]University of California Berkeley, [§]University of Massachusetts

Abstract

Integrated task and motion planning has emerged as a challenging problem in sequential decision making, where a robot needs to compute high-level strategy and low-level motion plans for solving complex tasks. While high-level strategies require decision making over longer time-horizons and scales, their feasibility depends on low-level constraints based upon the geometries and continuous dynamics of the environment. The hybrid nature of this problem makes it difficult to scale; most existing approaches focus on deterministic, fully observable scenarios. We present the first approach for computing task and motion policies for settings where the high-level problem can be modeled as a Markov decision process. In contrast to prior efforts, we show that complete MDP policies, or contingent behaviors, can be computed effectively in an anytime fashion. Our algorithm continuously improves the quality of the solution and is probabilistically complete. We evaluate the performance of our approach on a challenging, realistic test problem: autonomous aircraft inspection. Our results show that we can effectively compute consistent task and motion policies for the most likely execution-time outcomes using only a fraction of the computation required to develop the complete task and motion policy.

1 Introduction

In order to be truly helpful, robots will need to be able to accept commands from humans at high-levels of abstraction, and autonomously execute them. Consider the problem of inspecting an aircraft (Fig. 1). In order to autonomously plan and execute such a task, the robots (UAVs in this case) will need to be able to make high-level inspection decisions on their own, while satisfying low-level constraints that arise from environment geometries and the limited capabilities of the UAVs. High-level decisions can include selecting where to go next, with whom to communicate, and what to inspect.

[*]Work done while this author was at the United Technologies Research Center

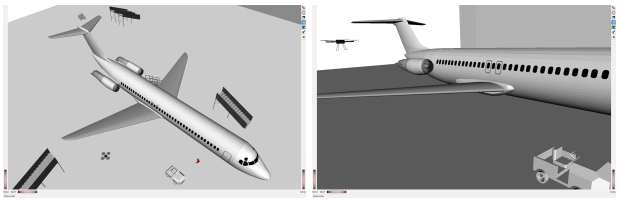


Figure 1: The aircraft inspection scenario

These decisions need to take into account the uncertainty in the UAV's actions.

For instance, at the start of an aircraft's inspection, one may know that the left wing has a structural problem, but the location of the fault may not be known precisely. When a UAV inspects the left wing, its sensors may succeed with probability 0.9, and so on. In order to solve this task autonomously, the UAV needs to select which pose to fly to next, which trajectory to use in order to do so, and the order in which to carry out inspections while making sure that it always has sufficient battery to return to the docking station and that it does not collide with any object in the environment. The feasibility of a high-level strategy for inspection therefore depends on the battery power required for each high-level operation such as "move to left wing"; "inspect left wing", etc., which in turn depends on the low-level motion plan selected, which in turn depends on the hangar's geometric layout and the physical geometry of the UAV. Throughout this paper, we will use the term "high-level" to represent a discrete MDP and "low-level" to refer to a motion planning problem.

The framework of Markov decision processes (MDPs) can express discrete sequential decision making (SDM) problems. Numerous advances have been made in solving MDPs [Russell *et al.*, 2015]. However, the scalability of these approaches relies upon a few key properties, including a bounded branching factor (or the set of possible actions) and the ability to express a problem accurately using discrete state variables. Both of these properties fail to hold in problems such as those described above. Recent work on deterministic, integrated task and motion planning [Kaelbling and Lozano-Pérez, 2011; Erdem *et al.*, 2011; Srivastava *et al.*, 2014; Dantam *et al.*, 2016] shows that hierarchical approaches are useful for such problems.

Computing task and motion policies for MDPs presents a

new set of challenges not encountered in computing task and motion plans for deterministic scenarios. In particular, selecting an action for a state while ensuring a feasible refinement requires knowing the history of actions used to reach that state, since effects on properties that were abstracted away (such as battery usage) cannot be modeled accurately at the high level. A direct application of classical task and motion planning techniques is further limited by the number of possible high-level action paths that can be taken during an execution. Indeed, the task and motion planning literature makes it clear that computing a single high-level sequence of actions that is feasible with low-level constraints is a challenge; the extension to MDPs expands the problem to computing a feasible high-level sequence of actions for every possible stochastic outcome of a high-level action.

In this paper, we present the first approach for computing branching task and motion policies for MDPs and show that principles of abstraction can be used to effectively model the problem, as well as to solve it by dynamically refining the abstraction used. We address the problem of computational complexity by developing an anytime algorithm that rapidly produces feasible policies for a high likelihood of scenarios that may be encountered during execution. Our methods can therefore be used to start the execution before the complete problem is solved; computation could continue during execution. The continual policy computation reports the probability of encountering situations which have not been resolved yet. This can be used to select the point at which execution is started in a manner appropriate to the application. In the worst case, if an unlikely event is encountered before the ongoing policy computation resolves it, execution could be brought to a safe state; in situations where this is not possible, one could wait for the entire policy to be computed with motion plans. In this way our approach offers a trade-off between pre-execution guarantees and pre-computation time requirements.

The rest of this paper is organized as follows. Sec. 2 introduces the main concepts that we draw upon from prior work. Sec. 3 presents our formalization of abstractions and representations. This is followed by a description of our algorithms (Sec. 4). Sec. 5 presents an empirical evaluation of our approach in a test scenario that we created using open-source 3D models of aircraft and various hangar components. Sec. 6 discusses the relationship of the presented work and contributions with prior work.

2 Background

A Markov decision process (MDP) $\langle S, A, T, R, \gamma \rangle$ is defined by a set of states S, a set of actions A, a transition function $T : S \times A \rightarrow \mu S$ that gives the probability distribution over result states upon the application of an action on a state; a reward function $R : S \rightarrow \mathbb{R}$; and a discounting factor $\gamma \leq 1$. We will use $T(s, a)$ as a function that maps a state to its probability. We will be particularly interested in MDPs with absorbing states and $\gamma = 1$, or, stochastic shortest path problems [Bertsekas and Tsitsiklis, 1991]. In this class of MDPs, the reward function yields negative values (action costs) for states except the absorbing states G. Absorbing states give zero rewards; once the agent reaches an absorbing state, it stays in it: $\forall a \in A, g \in G, R(g) = 0; T(g, a)(g) = 1$. We will consider SSPs that have a known initial state s_0 and a finite time horizon, h, which represents an uppper bound on the number of discrete decision making steps available to the agent.

Solutions to MDPs are represented as policies. A policy $\pi : S \times \{1, \ldots, h\} \rightarrow A$ maps a state and the timestep at which it is encountered, to the action that the agent should execute while following π. Given an MDP, the optimal "policy" of the agent is defined as one that maximizes the expected long-term reward $\sum_{i=1}^{h} r_i$, where r_i is the reward obtained at timestep i following the function R. Our notion of policies includes non-stationary policies since the optimal policy in a finite horizon MDP need not be stationary. In principle, dynamic programming can be used to compute the optimal policy in this setting just as in the infinite horizon setting:

$$V^0(s) = R(s) \quad (1)$$

$$V^i(s) = R(s) + max_a \sum_{s'} T(s, a)(s') V^{i-1}(s') \quad (2)$$

Here V^i is the i-step-to-go value function. Since we are given the initial state s_0, non-stationary policies can be expressed as finite state machines (FSMs). We will consider policies that are represented as tree-structured FSMs, also known as contingent plans. Several algorithms have been developed to solve SSPs. The LAO* algorithm [Hansen and Zilberstein, 2001] was developed to incorporate heuristics while computing solution policies for SSPs. Kolobov *et al.* [2011] developed general methods for solving SSPs in the presence of dead-ends.

Specifying real-world sequential decision making problems as MDPs using explicitly enumerated state lists usually results in large, unweildy formulations that are difficult to modify, maintain, or understand. Lifted, or parameterized representations for MDPs such as FOMDPs [Sanner and Boutilier, 2009], RDDL [Sanner, 2010] and PPDDL [Younes and Littman, 2004] have been developed for overcoming these limitations. Such languages separate an MDP *domain*, constituting parameterized actions, functions and predicate vocabularies, from an MDP *problem*, which expresses the specific objects of each type and a reward function. We refer to Helmert [2009] for a general introduction to the concepts of problems and domains in this context. W.l.o.g, we consider the vocabulary to consist of predicates alone, since functions can be represented as special predicates. A grounded predicate is a predicate whose parameters have been substituted by the objects in an MDP problem. For instance, Boolean valuations of the grounded predicate *faultLocated(LeftWing)* express whether the LeftWing's fault's precise location was identified. We use the symbolic formalization proposed by Srivastava *et al.* [2014] and represent function-calls that return continuous values as symbols. Thus the function call $f(o)$, where o is a constant, will be represented by the constant f_o in the high level SDM solver. For readability however, in this paper we will depict such function calls using the conventional syntax, as $f(o)$.

Action: *inspect(Structure s, Trajectory tr)*		
precond	*batterySufficient*$(tr)\wedge$ *inspects*$(tr, s)\wedge$ collisionFree(tr)	
effect	*faultLocated*(s)	0.8
	$\neg$*faultLocated*(s)	0.2
	decrease(batteryLevel, $c(tr)$*)*	

Figure 2: Specification of a stochastic action model

In our framework, states are defined as valuations of grounded predicates in a given problem. Although this framework usually expresses *discrete* properties, it can be extended naturally to model actions that have continuous action arguments and depend on and affect geometric properties of the environment.

Example 1. *Fig. 2 shows the specification for an* inspect *action in the aircraft inspection domain in a language similar to PPDDL (some syntactic elements have been simplified for readability). This action models the act of inspecting a structure s while following the path tr. We use* batterySufficient(tr) *as an abbreviation for* batteryLevel$-$batteryRequired(tr)*. Intuitively, the specification states that if this action is executed in a state where the battery is sufficient and the selected trajectory satisfies constraints for being an inspection trajectory (the* precondition *is satisfied), it will result in locating the fault with the probability 0.8. In any case, the battery's charge will be depleted by an amount depending on the trajectory used for inspection $c(tr)$. The* inspects(tr, s) *predicate is true if the trajectory tr "covers" the structure s. Different interpretations for such predicates would result in different classes of coverage patterns.*

3 Formal Framework

Let X be a set of states and S a set of abstract states. We define a **state abstraction** as a surjective function $\alpha : X \to S$. We focus on *predicate abstractions*, where the abstraction function effectively projects the state space into a space without a specified set of predicates. Given a set of predicates $\mathcal{P}$ that are retained by a predicate abstraction, the states of the abstract state space are equivalence classes defined by the equivalence relation $s_1 \sim s_2$ iff s_1 and s_2 agree on the valuations of every predicate in $\mathcal{P}$, grounded using the objects in the problem.

For any $s \in S$, the *concretization function* $\gamma_\alpha(s) = \{x \in X : \alpha(x) = s\}$ denotes the set of concrete states *represented by the abstract state* s. For a set $C \subseteq X$, $[C]_\alpha$ denotes the smallest set of abstract states representing C. Generating the complete concretization of an abstract state can be computationally intractable, especially in cases where the concrete state space is continuous and the abstract state space is discrete. In such situations, the concretization operation can be implemented as a *generator* that incrementally computes or samples elements from an abstract state's concretization.

Action abstraction functions can be defined similarly. The main form of an action abstraction function is to drop action arguments, which leads to predicate abstractions to eliminate all predicates that used the dropped arguments in the action's description. This process can also model non-recursive temporal abstractions since a macro or a high-level action with multiple implementations [Marthi *et al.*, 2007] can be modeled as an action whose arguments include the arguments of its possible implementations as well as an auxiliary argument for selecting the implementation. The concretization of an action abstraction function is the set of actions corresponding to different instantiations of the dropped action arguments. Concretization functions for action abstraction functions can also be implemented as generators.

Formally, the concretization of each high-level action corresponds to a set of motion planning problems. We will use the notation $a(x_1 \mapsto o_1)$ to denote a grounded action, whose x_1 argument has been instantiated with the element o_1 defined by the underlying MDP problem (Sec. 2). Let $a(\bar{x}, \bar{y})$ be a concrete action where $\bar{x}$ ($\bar{y}$) are ordered, typed discrete (continuous) arguments. The *concretization of the instantiated abstract action* $\gamma([a](\bar{x} \mapsto \bar{o}))$ is the set of actions $\{a(\bar{x} \mapsto \bar{o}, \bar{y} \mapsto \bar{o'}) : \bar{o'}$ *is a tuple of elements with types and arity specified by* $\bar{y}\}$. Predicates in action preconditions specify the constraints that these arguments need to satisfy. Common examples for continuous arguments include robot poses and motion plans; predicates about them may include *collisionFree(*tr*)*, which is true exactly when the trajectory tr has no collisions as well as *inspects* (Eg. 1).

Both state and action abstractions affect the transition function of the MDP. The actual transition probabilities of an abstract MDP depend on the policy being used and are therefore difficult to estimate accurately [Bai *et al.*, 2016; Li *et al.*, 2006; Singh *et al.*, 1995]. In this paper, we will use an optimistic estimate of the true transition probabilities when expressing the abstract MDP. Such estimates are related to upper bounds for reachability used in prior approaches for reasoning in the presence of hierarchical abstractions (e.g., [Marthi *et al.*, 2007; Ha and Haddawy, 1996]).

Example 2. *Consider the action presented in Eg. 1 Such actions are difficult to plan with however, since the tr argument is a high-dimensional real-valued vector. We can abstract away this argument to construct the following abstraction:*

Action: [inspect](Structure s)		
precond	batterySufficient	
effect	faultLocated(s)	0.8
	$\neg$faultLocated(s)	0.2
	⍰{batteryLevel, batterySufficient}	

Dropping the tr argument from each predicate that results in abstract predicates of lower arities. The zero-arity batterySufficient *becomes a Boolean state variable and* batteryLevel *becomes a numeric variable. The symbol ⍰ indicates that this action affects the predicates* batteryLevel *and* batterySufficient*, but its effects on these predicates cannot be determined due to abstraction.*

An optimistic representation of this abstract action would state that it does not reduce batteryLevel *and consequently, does not make* batterySufficient *false.*

This approach for abstraction is computationally better than a high-level representation that discretizes the continuous variables, as it does not require the addition of constants representing discrete pose or trajectory names to the vocab-

Algorithm 1: Anytime Task and Motion MDP (ATM-MDP)

Data: domain $\mathcal{D}$, problem $\mathcal{P}$, motionPlanner MP, SSP Solver SSP
Input: threshold t
Result: Task and motion policy for $\langle \mathcal{D}, \mathcal{P} \rangle$

```
1  policyTree ← SSP.getContingentPlan(P.f0, D, P);
2  currentState ← P.f0; proportionRefined ← 0.0; replanBias ←
     0.5;
3  partialTraj ← None;
4  leafQueue ← estimatePathCosts (policyTree,
     partialTraj) ;
5  while resource limit not reached and leafQueue.size() ≠ 0 and
     proportionRefined < t do
6      pathToRefine ← ancestors (leafQueue.pop()) ;
7      while resource limit not reached and pathToRefine.length()
         ≠ 0 do
8          (success, partialPathTraj, failureNode, failureReason)
             ← refinePath (pathToRefine, partialTraj,
             policyTree, MP) ;
9          if not success and failureReason ≠ None then
10             policyTree ← SSP.replan(failureNode,
                 failureReason);
11             break;
           else
12             for node ∈ partialPathTraj do
13                 partialTraj[node] ← partialPathTraj[node]
14     leafQueue ← estimatePathCosts (policyTree,
         partialTraj) ;
15     proportionRefined ←
         computeProportionRefined (policyTree,
         partialTraj)
```

ulary. This is desirable because the size of the state space would be exponential in the number of such discretized values that are included.

4 Overall Algorithmic Framework

The ATM-MDP algorithm (Alg. 1) presents the main outer loop of our approach for computing a task and motion policy. It assumes the availability of an SSP solver that can generate tree-structured policies (starting at a given initial state) for solving an SSP, a motion planner for refinement of actions within the policy, and a module that determines the reason for infeasibility of a given motion planning problem. The overall algorithm operates on root-to-leaf paths in the SSP solution.

The main computational problem is that the number of possible paths to refine grows exponentially with the time horizon. Waiting for a complete refinement would result in a lot of wasted time as most paths may correspond to outcomes that are unlikely to be encountered. Every path is associated with the probability p that an execution would follow that path; and a cost c of refining that pat. Ideally, we would like to compute an ordering of these paths so that at every time instant, we compute as many of the most likely paths as can be computed up to that time instant. Unfortunately, achieving this would be infeasible as it would require solving multiple *knapsack* problems. Instead, we order the paths by the ratio p/c for refinement (lines 4-15).

Theorem 1. *Let t be the time since the start of the algorithm at which the refinement of any root-to-leaf path is completed. If path costs are accurate and constant then the total probability of unrefined paths at time t is at most $1 - opt(t)/2$, where $opt(t)$ is the best possible refinement (in terms of the probability of outcomes covered) that could have been achieved in time t.*

The proof follows from the fact that the greedy algorithm achieves a 2-approximation for the knapsack problem. In practice, the true cost of refining a path cannot be determined prior to refinement. We therefore estimate the cost as the product of the parameter ranges covered by the generator of each action in the path. This results in lower bounds on the ratios p/c modulo constant factors, since a path could be refined before all the generator ranges are exhausted. In this way it doesn't over-estimate the relative value of refining a path. As we show in the empirical section, the resulting algorithm yields the concave performance profiles desired of anytime algorithms.

The *while* loop iterates over these paths while recomputing the priority queue keys after each iteration. Within each iteration, the algorithm tries to compute a full motion planning refinement of the path. First, the entire path (*pathToRefine*) is extracted from the leaf (line 6). The *refinePath* subroutine attempts to find a motion planning refinement (concretization) for *pathToRefine*. If it is unable to find a complete refinement for this path, it either (a) returns with a reason for failure along with a partial trajectory going up to the deepest node in the path for which it was able to compute a feasible motion plan, or (b) backtracks to return a partial trajectory that will result in a future refinePath call for a parent node of a node for which a motion planning refinement couldn't be found.

For partial trajectories under (a) (line 9), Alg. 1 calls an SSP solver after adjusting its initial state and domain definitions to include the *FailureReason*. The policy computed by the SSP solver is then merged with the existing policy and the while loop continues. For partial trajectories along case (b) (line 12), the path is added back to the queue with a partial, successful trajectory that results in backtracking.

If *refinePath* is successful in computing a full refinement, the while loop continues with an updated priority queue. In each iteration of the while loop, we compute the total probability of refined paths – this probability gives us the likelihood of being able to successfully execute the policy in its current state of refinement.

The *refinePath* subroutine (Alg. 2) attempts to compute a motion plan for each action in a given path. More precisely, it uses a generator to sample the possible concretizations for each action and test their feasibility. A feasible solution to any one of these motion planning problems is considered a feasible refinement of that abstract action. *refinePath* starts by selecting the first node in the path that needs to be refined in line 1 (Alg. 1 may result in situations where a prefix of a path has already been refined by a prior call to *refinePath*, due to line 14 in that algorithm).

It then iterates over possible target poses for the selected action (lines 8 through 11). If a feasible motion plan is found,

Algorithm 2: Subroutine refinePath

Input: pathToRefine, partialTraj, policyTree, motionPlanner
Output: success: indicator of successful refinement;
partialPathTraj: refined path up to the first failure;
failureNode, failureReason: failure information

```
1  node ← head(pathToRefine); partialPathTraj ← None;
2  for node ∈ pathToRefine do
3      a ← policyTree[node];
4      if partialTraj = None then
5          pose_1 ← InitialPose;
       else
6          pose_1 ← extractPose(partialTraj[parent(node)]);
7      while resource limit not reached and
         partialPathTraj[node] = None do
8          pose_2 = targetPoseGen(a);
9          if GetMotionPlan(pose_1, pose_2) succeeds then
10             partialPathTraj[node] ← ComputePath;
11             break;
12     if partialPathTraj[node] = None then
13         if Bernoulli(replanBias).sample() then
14             return (False, partialPathTraj, node,
                 FailureReason);
           else
15             partialPathTraj.remove(node.parent());
16             return (False, partialPathTraj, node.parent(),
                 None )
   return (True, partialPathTraj, None, None);
```

then the algorithm refines the next action in the path. If not, it stochastically chooses to either re-invoke the SSP by returning a *FailureReason*, or to backtrack by invalidating the current node's path (line 15) by removing it from *partialPathTraj* and returning to follow lines 12-13 in Alg. 1.

Though a backtracking search through all possible motion plans is required to guarantee the completeness of the algorithm, we find in practice that replanning with a new initial state and replacing the subtree rooted at a failed node with a new SSP solution is often more time efficient. This is because backtracking to an ancestor of the failed node invalidates the motion plans associated with all paths passing through that ancestor, often causing a large amount of previously completed work to be thrown out. This situation is illustrated in Figure 3. For this reason, we stochastically choose between backtracking and replanning and settle for *probabilistic* completeness of the search algorithm.

Properties of the Algorithm Our algorithm solves the dual problems of synthesizing a strategy as well as computing motion plans while ensuring that the computed strategy has a feasible motion plan. It factors a hybrid planning problem into a succession of discrete SSPs and motion planning problems. The algorithm can compute solutions even when most discrete strategies have no feasible refinements. A few additional salient features of the algorithm are:

- The representational mechanisms for encoding SSPs do not require discretization, thus providing scalability.

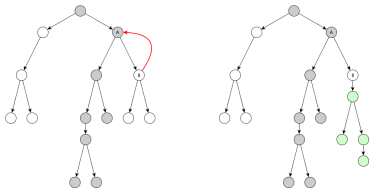


Figure 3: Left: Backtracking from node B invalidates the subtree rooted at A. In doing so, the work done in refining the node A's left child, in gray, is lost. Right: In some cases, replanning from node B requires less work than re-refining the invalidated subtree.

- The SSP model dynamically improves as the motion planning problems reveal errors in the high-level model in terms of *FailureReason*s.
- Prioritizing paths of relative value gives the algorithm a desirable anytime performance profile. This is further evaluated in the empirical section.

5 Empirical Evaluation

We implemented the algorithms presented in Sec. 4 using an implementation of LAO* [Hansen and Zilberstein, 2001] as the SSP solver. We used the OpenRAVE [Diankov, 2010] system for modeling and visualizing test environments and its collision checkers and RRT [LaValle and Kuffner Jr, 2000] implementation for motion planning. Since there has been very little research on the task and motion planning problem in stochastic settings, there are no standardized benchmarks. We evaluated our algorithms by creating a hangar model in OpenRAVE for the aircraft inspection problem (Fig. 1). UAV actions in this domain include actions for moving to various components of the aircraft, such as the left and right wings, nacelles, fuselage, etc. Each such action could result in the UAV reaching the specified component or a region around the component. The inspection action for a component had the stochastic effect of localizing a fault's location. The environment included docking stations that the UAV could reach and recharge on reserve battery power. Generators for concretizing all actions except the inspect action uniformly sampled poses in the target regions. Some of these poses naturally lead to shorter trajectories and therefore lower battery usage, depending on the UAV's current pose. However, we used uniform-random samples to evaluate the performance of the algorithm while avoiding domain-specific enhancements. The generator for *inspect(s)* simulated an inspection pattern by randomly sampling five waypoint poses in an envelope around s and ordering them along the medial axis of the component. We used a linear function of the trajectories to keep track of battery usage at the low level and to report insufficient battery as the *failureReason* when infeasibility was detected. This function was used to provide failure reasons to the high-level when the battery level was found to be insufficient.

Fig. 4 shows the performance of our approach for producing execution strategies with motion planning refinements as a function of the time for which the algorithm is allowed to run. The red lines show the number of nodes in the high-level policy that have been evaluated, refined, and potentially replaced with updated policies that permit low-level plans. The

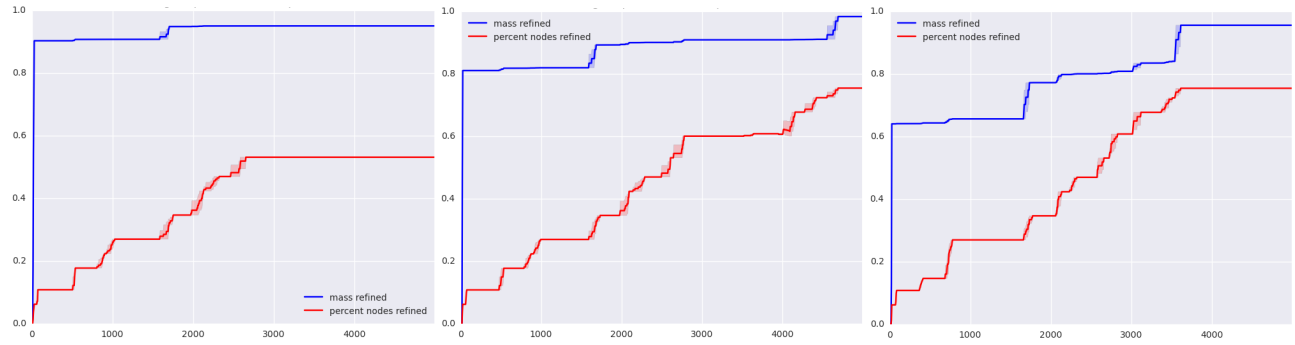


Figure 4: Performance of our anytime algorithm for solving MDPs using dynamic abstractions. The plots from left to right corresponds to formulation of the problem with 5%, 10%, and 20% rates of failure of the abstract actions described in the text. The blue lines (red lines) plot the probability mass of possible outcomes (proportion of nodes in the policy graph) that is covered by the partially computed policy as computation time (x axis, in seconds) evolves.

blue lines show the probability with which the policy available at any time during the algorithm's computation will be able to handle all possible execution-time outcomes. The different plots show how these relations change as we increase the level of uncertainty in the domain. The horizon is fixed at ten high-level decision epochs (each of which can involve arbitrarily long movements) and the number of parts with faults is fixed at two. The policy generated by LAO* is unrolled into a tree prior to the start of refinement. The reported times include the time taken for unrolling.

Our main result is that that our anytime algorithm balances complexity of computing task and motion policies with time very well and produces desirable concave anytime performance profiles. Fig. 4 shows that when noise in the agent's actuators and sensors is set at 5%, with 10% of computation our algorithm computes an executable policy that misses only the least likely 10% of the possible execution outcomes. This policy is computed in less than 10 seconds. In the worst case, with a 20% error rate in actuators and sensors (sensors used in practice are much more reliable), we miss only about 20% of the execution trajectories with 40% of the computation.

6 Other Related Work

There has been a renewed interest in integrated task and motion planning algorithms. Most research in this direction has been focused on deterministic environments [Cambon *et al.*, 2009; Plaku and Hager, 2010; Hertle *et al.*, 2012; Kaelbling and Lozano-Pérez, 2011; Garrett *et al.*, 2015; Dantam *et al.*, 2016]. Kaelbling and Lozano-Pérez [2013] consider a partially observable formulation of the problem. Their approach utilizes regression modules on belief fluents to develop a regression-based solution algorithm. While they address the more general class of partially observable problems, their approach follows a process of online, incremental discretization and does not address the computation of branching policies, which is the focus of this paper. Şucan and Kavraki [2012] use an explicit multigraph to represent the plan or policy for which motion planning refinements are desired. Hadfield-Menell *et al.* [2015] address problems where the high-level formulation is deterministic and the low-level is determinized using most likely observations. In contrast, our approach employs abstraction to bridge MDP solvers and motion planners to solve problems where the high-level model is stochastic. In addition, the transitions in our MDP formulation depend on properties of the refined motion planning trajectories (e.g., battery usage).

Principles of abstraction in MDPs have been well studied [Hostetler *et al.*, 2014; Bai *et al.*, 2016; Li *et al.*, 2006; Singh *et al.*, 1995]. However, these directions of work assume that the full, unabstracted MDP can be efficiently expressed as a discrete MDP. Marecki *et al.* [2006] consider continuous time MDPs with finite sets of states and actions. In contrast, our focus is on MDPs with high-dimensional, uncountable state and action spaces. Recent work on deep reinforcement learning (e.g., [Hausknecht and Stone, 2016; Mnih *et al.*, 2015]) presents approaches for using deep neural networks in conjunction with reinforcement learning to solve MDPs with continuous state spaces. We believe that these approaches can be used in a complementary fashion with our proposed approach. They could be used to learn maneuvers spanning shorter-time horizons, while our approach could be used to efficiently abstract their representations and to use them as actions or macros in longer-horizon tasks.

Efforts towards improved representation languages are orthogonal to our contributions [Fox and Long, 2002]. The fundamental computational complexity results indicating growth in complexity with increasing sizes of state spaces, branching factors, and time horizons remain true regardless of the solution approach taken. It is unlikely that a uniformly precise model, a simulator at the level of precision of individual atoms, or even circuit diagrams of every component used by the agent will help it solve the kind of complex tasks on which humans would appreciate assistance. On the other hand, not using any model at all would result in dangerous agents that would not be able to safely evaluate the possible outcomes of their actions. Our results show that these divides can be bridged using hierarchical modeling and solution approaches that simplify the representational requirements and offer computational advantages that could make autonomous robots feasible in the real world.

7 Conclusions

Our experiments showed that starting with an imprecise model, refining it based on the information required to evaluate different courses of action is an efficient approach for the synthesis of high-level policies that are consistent with constraints that may be imposed by aspects of the model that are more abstract or imprecise. While full models of realistic problems can overwhelm SDM solvers due to the uncountable branching factor and long time horizons, our hierarchical approach allows us to use SDM solvers while addressing more realistic problems involving physical agents.

Acknowledgments

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) and Space and Naval Warfare Systems Center Pacific (SSC Pacific) under Contract No. N66001-16-C-4050. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the DARPA or SSC Pacific.

References

Aijun Bai, Siddharth Srivastava, and Stuart J Russell. Markovian state and action abstractions for MDPs via hierarchical MCTS. In *Proc. IJCAI*, 2016.

Dimitri P Bertsekas and John N Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16(3):580–595, 1991.

Stephane Cambon, Rachid Alami, and Fabien Gravot. A hybrid approach to intricate motion, manipulation and task planning. *IJRR*, 28:104–126, 2009.

Neil T Dantam, Zachary K Kingston, Swarat Chaudhuri, and Lydia E Kavraki. Incremental task and motion planning: A constraint-based approach. In *Proc. RSS*, 2016.

Rosen Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, 2010.

Esra Erdem, Kadir Haspalamutgil, Can Palaz, Volkan Patoglu, and Tansel Uras. Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In *Proc. ICRA*, 2011.

Maria Fox and Derek Long. PDDL+: Modeling continuous time dependent effects. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*, 2002.

Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. FFrob: An efficient heuristic for task and motion planning. In *Proc. WAFR*. 2015.

Vu Ha and Peter Haddawy. Theoretical foundations for abstraction-based probabilistic planning. In *Proc. UAI*, 1996.

Dylan Hadfield-Menell, Edward Groshev, Rohan Chitnis, and Pieter Abbeel. Modular task and motion planning in belief space. In *Proc. IROS*, 2015.

Eric A Hansen and Shlomo Zilberstein. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1-2):35–62, 2001.

Matthew Hausknecht and Peter Stone. Deep reinforcement learning in parameterized action space. In *Proc. ICLR*, 2016.

Malte Helmert. Concise finite-domain representations for pddl planning tasks. *Artificial Intelligence*, 173(5):503 – 535, 2009.

Andreas Hertle, Christian Dornhege, Thomas Keller, and Bernhard Nebel. Planning with semantic attachments: An object-oriented view. In *Proc. ECAI*, 2012.

Jesse Hostetler, Alan Fern, and Tom Dietterich. State aggregation in monte carlo tree search. In *Proc. AAAI*, 2014.

Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *Proc. ICRA*, 2011.

Leslie Pack Kaelbling and Tomás Lozano-Pérez. Integrated task and motion planning in belief space. *The International Journal of Robotics Research*, 32(9-10):1194–1227, 2013.

A Kolobov, Mausam, DS Weld, and H Geffner. Heuristic search for generalized stochastic shortest path mdps. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2011.

S.M. LaValle and J.J. Kuffner Jr. Rapidly-exploring random trees: Progress and prospects. 2000.

Lihong Li, Thomas J Walsh, and Michael L Littman. Towards a unified theory of state abstraction for mdps. In *ISAIM*, 2006.

Janusz Marecki, Zvi Topol, Milind Tambe, et al. A fast analytical algorithm for mdps with continuous state spaces. In *AAMAS-06 Proceedings of 8th Workshop on Game Theoretic and Decision Theoretic Agents*, 2006.

Bhaskara Marthi, Stuart J Russell, and Jason Wolfe. Angelic semantics for high-level actions. In *Proc. ICAPS*, 2007.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

E. Plaku and G. D. Hager. Sampling-based motion and symbolic action planning with geometric and differential constraints. In *Proc. ICRA*, 2010.

Stuart Russell, Daniel Dewey, and Max Tegmark. Research priorities for robust and beneficial artificial intelligence. *AI Magazine*, 36(4):105–114, 2015.

Scott Sanner and Craig Boutilier. Practical solution techniques for first-order MDPs. *Artificial Intelligence*, 173(5-6):748–788, 2009.

Scott Sanner. Relational dynamic influence diagram language (rddl): Language description. `http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/RDDL.pdf`, 2010.

Satinder P Singh, Tommi Jaakkola, and Michael I Jordan. Reinforcement learning with soft state aggregation. In *Proc. NIPS*, pages 361–368, 1995.

Siddharth Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stuart Russell, and Pieter Abbeel. A modular approach to task and motion planning with an extensible planner-independent interface layer. In *Proc. ICRA*, 2014.

Ioan A Şucan and Lydia E Kavraki. Accounting for uncertainty in simultaneous task and motion planning using task motion multigraphs. In *Proc. ICRA*, 2012.

Håkan LS Younes and Michael L Littman. PPDDL 1.0: An extension to pddl for expressing planning domains with probabilistic effects. *Technical Report CMU-CS-04-162*, 2004.

A Unified Framework for Planning in Adversarial and Cooperative Environments

Anagha Kulkarni, Siddharth Srivastava and Subbarao Kambhampati
School of Computing, Informatics, and Decision Systems Engineering
Arizona State University, Tempe, AZ 85281 USA
{anaghak, siddharths, rao} @ asu.edu

Abstract

Users of AI systems may rely upon them to produce plans for achieving desired objectives. Such AI systems should be able to compute obfuscated plans whose execution in adversarial situations protects privacy, as well as legible plans which are easy for team members to understand in cooperative situations. We develop a unified framework that addresses these dual problems by computing plans with a desired level of comprehensibility from the point of view of a partially informed observer. For adversarial settings, our approach produces obfuscated plans with observations that are consistent with at least k goals from a set of decoy goals. By slightly varying our framework, we present an approach for goal legibility in cooperative settings which produces plans that achieve a goal while being consistent with at most j goals from a set of confounding goals. In addition, we show how the observability of the observer can be controlled to either obfuscate or clarify the next actions in a plan when the goal is known to the observer. We present theoretical results on the complexity analysis of our problems. We demonstrate the execution of obfuscated and legible plans in a cooking domain using a physical robot Fetch. We also provide an empirical evaluation to show the feasibility and usefulness of our approaches using IPC domains.

1 Introduction

AI systems have become quite ubiquitous. As users, we heavily rely on these systems to plan our day-to-day activities. Since all these systems have logging and tracking abilities, an observer can get access to our data and our actions. Such observers can be of two types: adversarial or cooperative. In adversarial settings, like mission planning, military intelligence, reconnaissance, counterintelligence, etc., protection of sensitive data can be of utmost importance to the agent. In such situations, it is necessary for an AI system to produce plans that reveal neither the intentions nor the activities of the agent. On the other hand, in case of a cooperative observer, the AI system should be able to produce plans that help clarify the intent of the agent. Therefore, it is desirable for an AI system to be capable of computing both obfuscated plans in adversarial settings and legible plans in cooperative settings.

In this work, we propose a new unifying formalization and algorithms for computing obfuscated plans as well as legible plans. In our framework, we consider two agents: an acting agent and an observer. The acting agent has full observability of its activities. The observer is aware of the agent's planning model but has partial observability of the agent's activities. The observations are emitted as a side effect of the agent's activities and are received by the observer. In the following example, we illustrate the influence of an observation model on the belief space of the observer.

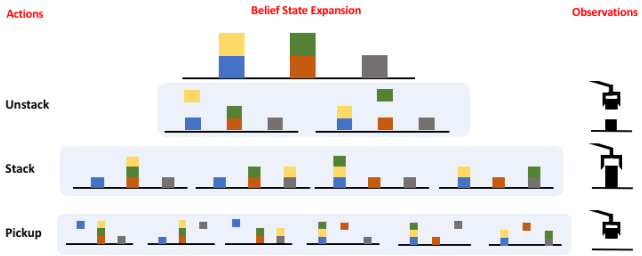


Figure 1: A `blocksworld` example illustrating belief expansion due to observation equivalence induced by the observation model of the observer.

Example 1. *Consider a* `blocksworld` *planning domain, where the actions* `unstack`, `stack`, `pickup` *and* `putdown` *are available. The observer can only receive partial observations using sensors on the table and on the gripper. An observation is emitted whenever the gripper is holding a block and indicates whether the block was lifted from the table or from the top of another block. This allows our observer to learn whether or not the current action has the agent holding a block (for example in* `unstack` *and* `pickup`*, it is holding a block) and whether or not the current action involves a block that was on the table (for example, in* `pickup` *and* `putdown`*, the blocks are picked up from / putdown on the table). Thus, our observer can distinguish between types of actions that were performed but is still unaware of which blocks were used. In Figure 1, we illustrate the progression of belief states from the observer's point of view based on the observation model defined above.*

An adversarial observer may be able to use the information gleaned from observations to interfere with or hamper the agent's activities. For example, consider the keystroke timing attack (Song, Wagner, and Tian 2001) where the observer retrieves observations about keystroke timing by

studying an agent's inter-keystroke timings. Through such traffic analysis attacks, the observer can learn the passwords typed by an agent while connecting to a remote machine. On the other hand, in cooperative scenarios, an agent is required to communicate its intentions to the observer as quickly and clearly as possible. For example, consider a robot who is capable of assembling either chairs or tables. A chair has three components: seat, back and legs; and a table has two components: surface and legs. Whenever the robot is holding a component, the observer receives an observation regarding the type of component. In order to notify about a task of say, assembling a chair, the robot can start with the seat or the back components rather than with the legs to make its objectives clearer to the observer.

In this work, we develop a coherent set of notions for goal obfuscation and goal legibility. Our approach computes the solutions for each of these problems using the variants of a common underlying algorithm. Our approach assumes offline settings, where the observer receives the observations after the agent has finished executing a plan. In the case of a goal obfuscation problem, there exist multiple decoy goals and one true goal. The observer is unaware of the agent's true goal, and the objective is to generate a plan solution without revealing it. Our solution ensures that *at least* k goals are possible at the end of the observation sequence. On the other hand, in the goal legibility problem, there exist multiple confounding goals and a true goal. Here the objective is to reveal *at most* j goals to the observer. Our solution ensures that at most j goals are possible at the end of the observation sequence. We also consider a variant of obfuscation and legibility where the adversary knows the goal of the agent and wants to obfuscate or reveal the next action in the plan to achieve that goal, we call these problems plan obfuscation and plan legibility respectively. For plan obfuscation, the objective is to generate a plan solution with an observation sequence that is consistent with at least ℓ diverse plans. On the other hand, for plan legibility, the objective is to generate a plan solution that is consistent with at least m similar plans.

In the following sections, we present a common framework that encapsulates the planning problems discussed above. And thereafter, we discuss each of the problems in detail. We also provide a theoretical and empirical analysis of the value and scope of our approaches.

2 Controlled Observability Planning Problem

2.1 Classical Planning

A classical planning problem can be defined as a tuple $\mathcal{P} = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, G \rangle$, where $\mathcal{F}$, is a set of fluents, $\mathcal{A}$, is a set of actions. A state s of the world is an instantiation, $\mathcal{F}^i$ of $\mathcal{F}$. The initial state $\mathcal{I}$ is the instantiation of all fluents in $\mathcal{F}$ and the goal G is a subset of instantiated fluents in $\mathcal{F}$. Each action $a \in \mathcal{A}$ is a tuple of the form $\langle pre(a), add(a), delete(a), c(a) \rangle$ where $c(a)$ denotes the cost of an action, $pre(a) \subseteq \mathcal{F}$ is a set of preconditions for the action a, $add(a) \subseteq \mathcal{F}$ is a set of positive effects and $delete(a) \subseteq \mathcal{F}$ is a set of negative effects, i.e., $\Gamma(s, a) \models \bot$ if $s \not\models pre(a)$; else $\Gamma(s, a) \models s \cup add(a) \setminus delete(a)$ where $\Gamma(\cdot)$ is the transition function. The solution to $\mathcal{P}$ is a *plan* or a sequence of actions $\pi = \langle a_1, a_2, \ldots, a_n \rangle$, such that, $\Gamma(\mathcal{I}, \pi) \models G$, i.e., starting from the initial state sequentially executing the actions lands the agent in a goal state. The cost of the plan, $c(\pi)$, is summation of the cost of all the actions in the plan π, $c(\pi) = \sum_{a_i \in \pi} c(a_i)$.

2.2 Problem Setting

We now introduce a general planning problem framework that will be used to define adversarial and cooperative cases in the following sections. The controlled observability problem involves an acting agent and an observer.

Definition 1. *A **controlled observability planning problem** is a tuple, $\mathcal{P}_{\mathcal{CO}} = \langle \mathcal{D}, \mathcal{G}, \Omega, \mathcal{O} \rangle$, where,*

- $\mathcal{D} = \langle \mathcal{F}, \mathcal{A}, \mathcal{I} \rangle$ *is the planning domain of the agent.*
- $\mathcal{G} = \{G_1 \cup G_2 \ldots \cup G_{n-1} \cup G_A\}$ *is a set of candidate goal conditions, each defined by subsets of fluent instantiations, where G_A is the true goal of the agent.*
- $\Omega = \{o_i | i = 1, \ldots, m\}$ *is a set of m observations that can be emitted as a result of the action taken and the state transition.*
- $\mathcal{O} : (\mathcal{A} \times \mathcal{S}) \rightarrow \Omega$ *is a many-to-one observation function which maps the action taken and the next state reached to an observation in Ω. That is to say, the observations are deterministic, each $\langle a, s' \rangle$ pair is associated with a single observation but multiple pairs can be mapped to the same observation.*

The observer has access to $\mathcal{P}_{\mathcal{CO}}$, but is unaware of the true goal of the agent. Also, the observer does not have access to the actions performed by the agent, instead receives the observations corresponding to the plan executed by the agent. The observation function can be seen as a sensor model, as modeled in several prior works (Geffner and Bonet 2013; Bonet and Geffner 2014; Keren, Gal, and Karpas 2016b). For every action taken by the agent and an associated state transition, the observer receives an observation. This observation might be consistent with multiple action-state pairs because of the many-to-one formulation of $\mathcal{O}$. Therefore, the observer operates in the belief space. The agent takes the belief space of the observer into account in its planning process, so as to control the observability of the observer. Our formulation defines an offline scenario where the observer receives all the observations once the plan has been executed by the agent.

2.3 Observer's Belief Space

The observer may use its observations of the agent's activity to maintain a *belief state*, or the set of possible states consistent with the observations.

Definition 2. *A **belief**, b_i, induced by observation, o_i, emitted by action, a_i, and resulting state, s_i, is, $b_i = \{\hat{s}_i \mid \exists \hat{a}_i,\ \mathcal{O}(\hat{a}_i, \hat{s}_i) = o_i \wedge \mathcal{O}(a_i, s_i) = o_i\}$.*

Whenever a new action is taken by the agent, the observer's belief can be updated as follows:

Definition 3. *A **belief update**, b_{i+1} for belief b_i is defined as, $b_{i+1} = update(b_i, o_{i+1}) = \{\hat{s}_{i+1} \mid \exists \hat{s}_i, \exists \hat{a}_{i+1},\ \Gamma(\hat{s}_i, \hat{a}_{i+1}) \models \hat{s}_{i+1} \wedge \hat{s}_i \in b_i \wedge \mathcal{O}(\hat{a}_{i+1}, \hat{s}_{i+1}) = o_{i+1}\}$.*

A sequence of such belief updates gives us the observer's belief sequence that is consistent with a sequence of observations emitted by the agent.

Definition 4. *A **belief sequence** induced by a plan p starting at state s_0, BS(p, s_0), is defined as a sequence of beliefs $\langle b_o, b_1, \ldots, b_n\rangle$ such that there exist $o_0, o_1, o_2, \ldots, o_n \in \Omega$ where,*

- $o_i = \mathcal{O}(a_i, s_i)$
- $b_0 = \{\hat{s}_0 | \mathcal{O}(\emptyset, s_0) = o_0 \wedge \mathcal{O}(\emptyset, \hat{s}_0) = o_0\}$
- $b_{i+1} = update(b_i, o_{i+1})$

The objective of the agent is to generate a desired belief in the observer's belief space, such that the last belief in the induced belief sequence satisfies goal conditions from the candidate goal set $\mathcal{G}$ including G_A.

2.4 Variants of $\mathcal{P}_{\mathcal{CO}}$

We now discuss the two major variants of $\mathcal{P}_{\mathcal{CO}}$ namely, goal obfuscation and goal legibility planning problems.

Goal Obfuscation The adversary is aware of agent's candidate goal set but is unaware of agent's true goal. The aim of goal obfuscation is to hide this true goal from the observer. This is done by taking actions towards agent's true goal, such that, the corresponding observation sequence exploits the observer's belief space in order to be consistent with multiple goals.

Definition 5. *A **goal obfuscation planning problem**, is a $\mathcal{P}_{\mathcal{CO}}$, where, $\mathcal{G} = \{G_A \cup G_1 \cup \ldots \cup G_{n-1}\}$, is the set of n goals where G_A is the true goal of the agent, and $G_1, \ldots, G_{n-1}$ are decoy goals.*

A solution to a goal obfuscation planning problem is a *k-ambiguous* plan. The objective here is to make the observation sequence consistent with at least k goals, out of which $k-1$ are decoy goals, such that, $k \leq n$. These $k-1$ goals can be chosen by the robot so as to maximize the obfuscation.

Definition 6. *A plan, π_k, is a **k-ambiguous plan**, if $\Gamma(\mathcal{I}, \pi_k) \models G_A$ and the last belief, $b_n \in BS(\pi_k, \mathcal{I})$, satisfies the following, $|b_n \cap \mathcal{G}| \geqslant k$, where $1 \geqslant k \geqslant n$.*

Definition 7. *An observation sequence $O_k = \langle o_1, \ldots, o_n\rangle$ is **k-ambiguous observation sequence** if it is an observation sequence emitted by a k-ambiguous plan.*

A *k-ambiguous* plan achieves at least k goals in the last belief of the observation sequence.

Goal Legibility The aim of goal legibility is to take goal-specific actions which help the observer in deducing the robot's goal. This can be useful in cooperative scenarios where the robot wants to notify the observer about its goal without explicit communication. This case is exactly opposite of the obfuscation case.

Definition 8. *A **goal legibility planning problem** is a $\mathcal{P}_{CO}$, where, $\mathcal{G} = \{G_A \cup G_1 \cup \ldots \cup G_{n-1}\}$ is the set of n goals where G_A is the true goal of the agent, and $G_1, \ldots, G_{n-1}$ are confounding goals.*

The objective here is to generate legible plans so as to reveal at most j goals. Here we ensure that the plans are consistent with *at most* j goals so as to minimize the number of goals in the observer's belief space.

Definition 9. *A plan, π_j, is a **j-legible plan**, if $\Gamma(\mathcal{I}, \pi_j) \models G_A$ and the last belief, $b_n \in BS(\pi_j, \mathcal{I})$, satisfies the following, $|b_n \cap \mathcal{G}| \leqslant j$, where $1 \geqslant j \geqslant n$.*

The definition of *j-legible* observation sequence follows that of *k-ambiguous* case.

2.5 Complexity Analysis

In this section, we discuss the complexity results for $\mathcal{P}_{\mathcal{CO}}$. Given the Definitions 6 and 9 of goal obfuscation and goal legibility plan solutions, we prove that the plan existence problem for $\mathcal{P}_{\mathcal{CO}}$ is EXPSPACE-complete.

Theorem 1. *The plan existence problem for a controlled observability planning problem is EXPSPACE-hard.*

Proof. To show that the plan existence problem for $\mathcal{P}_{\mathcal{CO}}$ is EXPSPACE-hard, we will show that the NOD (No-Observability Deterministic) planning problem is reducible to $\mathcal{P}_{\mathcal{CO}}$. The plan existence problem for NOD has been shown to be EXPSPACE-complete (Haslum and Jonsson 1999; Rintanen 2004).

Let $\mathcal{P}_N = \langle \mathcal{F}_N, \mathcal{A}_N, \mathcal{I}_N, G_N, \mathcal{V}\rangle$ be a NOD planning problem, where, $\mathcal{F}_N$ is the set of fluents (or Boolean state variables), such that, state s is an instantiation of $\mathcal{F}_N$. $\mathcal{A}_N$ is a set of actions, such that, when an action $a \in \mathcal{A}_N$ is applied to a state, s_i, a deterministic transition to the next state occurs, $\Gamma(s_i, a) \models s_{i+1}$. $\mathcal{I}$ and G are Boolean formulae that represent sets of initial and goal states. $\mathcal{V} = \emptyset$ is the set of observable state variables. Since the underlying system state is unknown, the deterministic transition function does not reveal the hidden state. $\mathcal{P}_N$ can be expressed as a $\mathcal{P}_{\mathcal{CO}}$ problem, $\mathcal{P}_C = \langle \mathcal{D}_C, G_C, \Omega_C, \mathcal{O}_C\rangle$, where, $\mathcal{D}_C = \{\mathcal{F}_C, \mathcal{A}_C, \mathcal{I}_C\}$, such that $\mathcal{I}_C$ is a set of possible initial states, G_C is a subset of instantiations in $\mathcal{F}_C$, $\Omega = \emptyset$ and $\mathcal{O} = \emptyset$.

Suppose $\pi_{\mathcal{P}_C} = \langle a_1, \ldots, a_r\rangle$ is a plan solution to $\mathcal{P}_C$, such that, $\Gamma(\mathcal{I}_C, \pi_{\mathcal{P}_C}) \models G_C$ and the last belief $b_r \in BS(\pi_{\mathcal{P}_C}, \mathcal{I}_C)$ satisfies $|b_r \cap G_C| = 1$. Then according to the definition of $\mathcal{P}_N$, the plan $\pi_{\mathcal{P}_C}$ has a last belief, such that, $\exists s_r \in b_r, s_r \models G_C$ and therefore solves $\mathcal{P}_N$.

Conversely, suppose $\pi_{\mathcal{P}_N} = \langle a_1, \ldots, a_q\rangle$ is a plan solution to $\mathcal{P}_N$, such that, $\Gamma(\mathcal{I}_N, \pi_{\mathcal{P}_N}) \models G_N$. Let B_q be the belief associated with the last action in $\pi_{\mathcal{P}_N}$. Since it achieves the goal, we can say that $|B_q \cap G| = 1$. According to Definitions 6, 9, for $k = j = 1$, B_q satisfies the condition. Therefore $\pi_{\mathcal{P}_N}$ is a solution to $\mathcal{P}_C$. □

Theorem 2. *The plan existence problem for a controlled observability planning problem is EXPSPACE-complete.*

Proof. In $\mathcal{P}_{\mathcal{CO}}$, the planner operates in belief space and the state space is bounded by $2^{2^{|\mathcal{F}|}}$, where $|\mathcal{F}|$ is the cardinality of the fluents (or Boolean state variables). If there exists a plan solution for $\mathcal{P}_{\mathcal{CO}}$, it must be bounded by $2^{2^{|\mathcal{F}|}}$ in length. Any solution longer in length must have loops, which can be removed. Therefore, by selecting actions non-deterministically, the solution can be found in at most $2^{2^{|\mathcal{F}|}}$ steps. Hence, the plan existence problem for $\mathcal{P}_{\mathcal{CO}}$ is in NEXPSPACE. By Savitch's theorem (Savitch 1970), NEXPSPACE = EXPSPACE. Therefore, the plan existence problem for $\mathcal{P}_{\mathcal{CO}}$ is EXPSPACE-complete. □

2.6 Algorithm for Plan Computation

We present the details of a common algorithm template used by our formulations in Algorithm 1. In Section 3, we show how we customize the goal-test (line 24) and the heuristic function (line 30) to suit the needs of each of our problem variants. There are two loops in the algorithm: the outer loop (line 3) runs for different values of $\Delta = \{1, 2, \ldots, |\mathcal{S}|\}$; while the inner loop (line 12) performs search over the state space of size $\binom{|\mathcal{S}|}{\Delta}$. These loops ensure the complete exploration of the belief space.

For each outer iteration, s_Δ is augmented with elements of the belief state until the cardinality of s_Δ is equal to the value of Δ. In the inner loop, we run GBFS over the state space of s_Δ. For each successor node in the open list, the belief induced by an observation is updated. The heuristic value of a state is computed using a plan graph (Blum and Furst 1997) level based heuristic, such as set-level heuristic (Nguyen, Kambhampati, and Nigenda 2002). The plan graph data structure contains information about the positive and the negative interactions between the sets of propositions and actions. We use set-level plan graph heuristic to guide the search. To get the set-level cost, the plan graph is populated with a state, s (search node), and it is expanded until one of the following holds $(\mathbf{1})$ the goal is reachable, that is, the goal propositions are present in a proposition layer and are mutex-free pairs, or $(\mathbf{2})$ the graph levels off, that is, it cannot be expanded further. If the goal is not reachable before the graph levels off then it cannot be achieved by any plan. In this case, the heuristic cost is ∞. Else, when the goal is reachable and the goal propositions are pairwise mutex-free, the heuristic value is the index of the first plan graph layer that contains it.

Proposition 1. *Algorithm 1 necessarily terminates in finite number of $|\mathcal{S}|$ iterations, such that, the following conditions hold:*

***(Completeness)** Algorithm 1 explores the complete solution space of $\mathcal{P}_{\mathcal{CO}}$, that is, if there exists a $\pi_{\mathcal{P}_{\mathcal{CO}}}$ that correctly solves $\mathcal{P}_{\mathcal{CO}}$, it will be found.*

***(Soundness)** The plan, $\pi_{\mathcal{P}_{\mathcal{CO}}}$, found by Algorithm 1 correctly solves $\mathcal{P}_{\mathcal{CO}}$ as ensured by the corresponding goal-test.*

Algorithm 1 terminates either when a plan is found or after running the outer loop for $|\mathcal{S}|$ iterations. The outer loop ensures that the all the paths in the search space are explored.

Algorithm 1: Plan Computation

Input: $\mathcal{P}_{\mathcal{CO}} = \langle \mathcal{D}, \mathcal{G}, \Omega, \mathcal{O} \rangle$
Output: plan solution $\pi_{\mathcal{P}_{\mathcal{CO}}}$, observation sequence, $O_{\mathcal{P}_{\mathcal{CO}}}$

1 $\Delta \leftarrow 1$ ▷ Counter
2 $\Delta\_limit \leftarrow False$ ▷ Delta cardinality flag
3 **while** $\Delta \leqslant |\mathcal{S}|$ **do**
4 $\quad s_\Delta \leftarrow \{\mathcal{I}\}$ ▷ Initial state
5 $\quad open \leftarrow$ Priority_Queue() ▷ Open list
6 $\quad closed \leftarrow \{\}$ ▷ Closed list
7 $\quad b_0 \leftarrow \{\mathcal{O}(\emptyset, s_\Delta)\}$ ▷ Initial belief
8 $\quad open.push(\langle \mathcal{I}, b_0 \rangle, priority = 0)$
9 $\quad$ **if** $|s_\Delta| = \Delta$ **then**
10 $\quad\quad \Delta\_limit \leftarrow True$
11 $\quad$ **end**
12 $\quad$ **while** $open \neq \emptyset$ **do**
13 $\quad\quad \langle s_\Delta, b \rangle \leftarrow open.pop()$
14 $\quad\quad$ **if** $\neg \Delta\_limit$ **then**
15 $\quad\quad\quad$ **for** $\hat{s} \in b \setminus s_\Delta$ **do**
16 $\quad\quad\quad\quad s_\Delta \leftarrow s_\Delta \cup \hat{s}$
17 $\quad\quad\quad\quad$ **if** $|s_\Delta| = \Delta$ **then**
18 $\quad\quad\quad\quad\quad \Delta\_limit \leftarrow True$
19 $\quad\quad\quad\quad\quad$ **break**
20 $\quad\quad\quad\quad$ **end**
21 $\quad\quad\quad$ **end**
22 $\quad\quad$ **end**
23 $\quad\quad closed \leftarrow closed \cup s_\Delta$
24 $\quad\quad$ **if** $\langle s_\Delta, b \rangle \models$ GOAL-TEST($\mathcal{G}$) **then**
25 $\quad\quad\quad$ **return** $\pi_{\mathcal{P}_{\mathcal{CO}}}, O_{\mathcal{P}_{\mathcal{CO}}}$
26 $\quad\quad$ **end**
27 $\quad\quad$ **for** $s'_\Delta \in successors(s_\Delta)$ **do**
28 $\quad\quad\quad o \leftarrow \mathcal{O}(a, s'_\Delta)$
29 $\quad\quad\quad b' \leftarrow$ Belief-Generation(b, a, o)
30 $\quad\quad\quad h(s'_\Delta) \leftarrow$ HEURISTIC-FUNCTION(s'_Δ, b')
31 $\quad\quad\quad$ **if** $s'_\Delta \notin$ *open* ***and*** $s'_\Delta \notin$ *closed* **then**
32 $\quad\quad\quad\quad open.push(\langle s'_\Delta, b' \rangle, h(s'_\Delta))$
33 $\quad\quad\quad$ **else if** $h(s'_\Delta) < h^{prev}(s'_\Delta)$ **then**
34 $\quad\quad\quad\quad$ **if** $s'_\Delta \notin$ *open* **then**
35 $\quad\quad\quad\quad\quad closed \leftarrow closed \setminus s'_\Delta$
36 $\quad\quad\quad\quad\quad open.push(\langle s'_\Delta, b' \rangle, h(s'_\Delta))$
37 $\quad\quad\quad\quad$ **else**
38 $\quad\quad\quad\quad\quad$ update priority from $h^{prev}(s'_\Delta)$ to $h(s'_\Delta)$
39 $\quad\quad\quad\quad$ **end**
40 $\quad\quad\quad$ **end**
41 $\quad\quad$ **end**
42 $\quad$ **end**
43 $\quad \Delta \leftarrow \Delta + 1$
44 $\quad \Delta\_limit \leftarrow False$
45 **end**
46 **procedure** Belief-Generation(b, a, o)
47 $b' \leftarrow \{\}$
48 **for** $\hat{s} \in b$ **do**
49 $\quad$ **for** $\hat{a} \in \mathcal{A}$ **do**
50 $\quad\quad$ **if** $\hat{s} \models pre(\hat{a})$ ***and*** $\mathcal{O}(\hat{a}, \Gamma(\hat{s}, \hat{a})) = o$ **then**
51 $\quad\quad\quad b' \leftarrow b' \cup \Gamma(\hat{s}, \hat{a})$
52 $\quad\quad$ **end**
53 $\quad$ **end**
54 **end**
55 **return** b'

And the goal tests of both of the problem variants ensure that the solutions are correct with respect to Definitions 6 and 9.

The increase in cardinality of s_Δ can lead to increase in the search overhead. In our implementation, we run only the first iteration of the outer loop. Most of the problem instances can be solved in the first iteration itself.

3 Plan Computation

In this section, we present instantiations of modules presented in Algorithm 1 for goal obfuscation and legibility.

3.1 Computing Goal Obfuscated Plans

Goal test We ensure that the solution plan does not end unless all k goals occur in the belief state. In order to achieve this, the goal condition checks whether the goal has been achieved in the agent's true state and also whether the $k-1$ goals have been achieved in the belief state. If there doesn't exist a plan that can achieve the true goal with the given $k-1$ goals, we restart the search with next combination of $k-1$ goals among the total $n-1$ decoy goals.

Heuristic function We now propose a heuristic strategy for generating a plan where the last belief state satisfies k goals, making it a *k-ambiguous* plan. We use two heuristic functions to estimate a node's cost:

$$h(s) = \big(\textit{set-level}_{G_A}(s) + \textit{set-level}_{\mathcal{G}_{k-1}}(b)\big) \qquad (1)$$

where the first heuristic computes the *setLevel* distance to the true goal from the agent's actual state, while the second heuristic computes the *setLevel* distance to $k-1$ goals from the belief induced by the emitted observation. This is computed by finding the max of the minimum set-level distance from belief to each of the $k-1$ goals. The heuristic value of a state is computed as the total of these two values. The heuristic ensures at least k goals occur in the last belief induced by the plan.

Note on deterministic output and use of noop The *k-ambiguous* algorithm maintains obfuscation assuming that the adversarial observer does not have access to the process of plan computation. If the observer has access to it then, a simple variation like the addition of random noise to the heuristic can thwart the attack.

Our formulation supports the use of noops for obfuscation. A noop action can be compiled to multiple noop actions, $noop = \{pretend_{o_i} | i = 1, \ldots, |\Omega|\}$ where each noop corresponds to an observation of a prior action. This would allow the agent to emit observations without doing any action. This way the agent can increase the chances of obfuscation while using the same approaches.

3.2 Computing Goal Legible Plans

Goal test In this case, we have to ensure that the computed plan is consistent with *at most* j true goals. To that end, we change our goal condition to additionally check whether at most $j-1$ confounding goals have been achieved in the last belief in BS, and whether at least $n-j$ goals are absent in the belief.

Heuristic function In this case, our objective is to avoid at least $n-j$ goals and at the same time be consistent with at most j goals. We achieve this by minimizing the set-level distance to true goal from the agent's actual state and to the $j-1$ confounding goals from the agent's belief state. However, we maximize the set-level distance to other $n-j$ goals in order to achieve at most j goals. This is written as,

$$h(s) = \big(\textit{set-level}_{G_A}(s) + \textit{set-level}_{\mathcal{G}_{j-1}}(b) - \textit{set-level}_{\mathcal{G}_{n-j}}(b)\big) \qquad (2)$$

4 Plan Obfuscation and Plan Legibility

We also discuss two other problem variants called plan obfuscation and plan legibility. These problems come into picture when the observer is aware of the agent's goal. However, the objective here is to either obfuscate or reveal the next steps in the plan.

4.1 Obfuscation

We achieve plan obfuscation by computing a plan whose observation sequence conforms to a set of diverse plans, making it hard to predict the next action in that plan.

Definition 10. *A **plan obfuscation planning problem** is a tuple, $\mathcal{P}_{PO} = \langle \mathcal{D}, \mathcal{G}_{PO}, \Omega, \mathcal{O} \rangle$, where, $\mathcal{G}_{PO} = \{G_A\}$, and G_A is the true goal of the agent.*

The solution to a plan obfuscation planning problem is an *ℓ-diverse* plan. An *ℓ-diverse* plan has an observation sequence that is consistent with ℓ plans that are at least d distance away. In order to compute an *ℓ-diverse* plan, we need to keep track of the paths that are consistent with the belief sequence of the given plan, we call the set of these paths as belief plan set.

Definition 11. *A **belief plan set**, BPS(p, s_0) = $\{p_1, \ldots, p_n\}$, induced by a plan p starting at s_0, is a set of plans that are formed by causally consistent chaining of state sequences in $BS(p, s_0)$, i.e., BPS(p, s_0) = $\{\langle \hat{s}_0, \hat{a}_1, \hat{s}_1, \ldots, \hat{s}_n \rangle \mid \exists\ \hat{a}_j,\ \hat{s}_{j-1} \models pre(\hat{a}_j) \wedge \hat{s}_{j-1} \in b_{j-1} \wedge \hat{s}_j \models \hat{s}_{j-1} \cup add(\hat{a}_j) \setminus delete(\hat{a}_j) \wedge \hat{s}_j \in b_j\}$.*

Our aim is to compute the diversity between all the pairs of plans in $BPS(p, s_0)$. The diversity between plans can be enforced by using plan distance measures.

Plan Distance Measures We will utilize the three plan distance measures introduced in Srivastava et al. (2007), and refined in Nguyen et al. (2012), namely action, causal link and state sequence distances. Our aim is to use these plan distance measures to measure the diversity of plans in a belief plan set.

Action distance We denote the set of unique actions in a plan π as $A(\pi) = \{a \mid a \in \pi\}$. Given the action sets $A(p_1)$ and $A(p_2)$ of two plans p_1 and p_2 respectively, the action distance is, $\delta_A(p_1, p_2) = 1 - \frac{|A(p_1) \cap A(p_2)|}{|A(p_1) \cup A(p_2)|}$.

Causal link distance A causal link represents a tuple of the form $\langle a_i, p_i, a_{i+1} \rangle$, where p_i is a predicate that is produced as an effect of action a_i and used as a precondition

for a_{i+1}. The causal link distance for the causal link sets $Cl(p_1)$ and $Cl(p_2)$ of plans p_1 and p_2 is, $\delta_C(p_1, p_2) = 1 - \frac{|Cl(p_1) \cap Cl(p_2)|}{|Cl(p_1) \cup Cl(p_2)|}$.

State sequence distance This distance measure takes the sequences of the states into consideration. Given two state sequence sets $S(p_1) = (s_0^{p_1}, \ldots, s_n^{p_1})$ and $S(p_2) = (s_0^{p_2}, \ldots, s_{n'}^{p_3})$ for p_1 and p_2 respectively, where $n \geq n'$ are the lengths of the plans, the state sequence distance is, $\delta_S(p_1, p_2) = \frac{1}{n}\big[\sum_{k=1}^{n'} d(s_k^{p_1}, s_k^{p_2}) + n - n'\big]$, where $d(s_k^{p_1}, s_k^{p_2}) = 1 - \frac{|s_k^{p_1} \cap s_k^{p_2}|}{|s_k^{p_1} \cup s_k^{p_2}|}$ represents the distance between two states (where $s_k^{p_1}$ is overloaded to denote the set of fluents in state $s_k^{p_1}$).
We now formally define ℓ-*diverse* plan and other terms.

Definition 12. *Two plans, p_1, p_2, are a **d-distant pair** with respect to distance function δ if, $\delta(p_1, p_2) = d$, where δ is a diversity measure.*

Definition 13. *A BPS induced by plan p starting at s_0 is **minimally d-distant**, $d_{min}(BPS(p, s_0))$, if $d = \min_{p1,p2 \in BPS(p,s_0)} \delta(p1, p2)$.*

Definition 14. *A plan, π_l, is an **ℓ-diverse plan**, if for a given value of d and distance function δ, $d_{min}(BPS(\pi_l, \mathcal{I})) \geq d$, $|BPS(\pi_l, \mathcal{I})| \geq \ell$, where $\ell \geq 2$ and every plan in $BPS(\pi_l, \mathcal{I})$ achieves the goal in $\mathcal{G}_{PO}$.*

Computing Obfuscated Plans Here we return a plan that is at least ℓ-*diverse* and that maximizes the plan distance between *BPS* induced by a plan.

Goal test To ensure the plans in BPS induced by ℓ-*diverse* plan can achieve the goal in $\mathcal{G}_{\mathcal{PO}}$, we change the goal condition to additionally check whether at least ℓ plans are reaching the goal or not. Also in order to ensure termination of the algorithm, there is a cost-bound given as input to the algorithm.

Heuristic function We now present our heuristic strategy to compute ℓ-*diverse* observation sequence. Our heuristic is a three-part function:

$$h(s) = (-d_{min}(BPS(p, s_0)), -|BPS(p, s_0)|, \textit{set-level}(s)) \quad (3)$$

where the primary heuristic maximizes the d of $d_{min}(BPS(p, s_0))$ induced by plan p starting at s_0, the second heuristic maximizes the cardinality of the set $BPS(p, s_0)$, while the third heuristic gives the set-level value of s. The cardinality of $BPS(p, s_0)$ is computed by counting the number of plans that have the same set-level value from their last states as that from the true state. The distance is computed using either of the three plan distances.

4.2 Plan Legibility

The definition of plan legibility planning problem is similar to that of plan obfuscation planning problem. Here the objective is to reveal the next action in the plan. We achieve this by making the observation sequence consistent with plans that are similar in terms of observation sequences. This case is opposite to that of the plan obfuscation case. We use the same three plan distance measures to define the similarity of the plans. We call the solution to this problem as an *m-similar* plan. For similar plans in BPS, we define the maximum distance between any two pairs of plans.

Definition 15. *A BPS induced by plan p starting at s_0 is **maximally d-distant**, $d_{max}(BPS(p, s_0))$, if $d = \max_{p1,p2 \in BPS(p,s_0)} \delta(p1, p2)$.*

Definition 16. *A plan, π_m, is a **m-similar plan**, if for a given value of d and distance function δ, $d_{max}(BPS(\pi_m, \mathcal{I})) \leq d$, $|BPS(\pi_m, \mathcal{I})| \geq m$, where $m \geq 2$ and every plan in $BPS(\pi_m, \mathcal{I})$ achieves the goal in $\mathcal{G}_{PL}$.*

Computing Legible Plans Here we return a plan that is at least m-*similar* and that minimizes the plan distance between *BPS* induced by a plan. The goal test for this case is similar to that of plan obfuscation case.

Heuristic function The heuristic function to compute m-*similar* observation sequence is a three-part function:

$$h(s) = (d_{max}(BPS(p, s_0)), -|BPS(p, s_0)|, \textit{set-level}(s)) \quad (4)$$

where the primary heuristic minimizes the d value of $d_{max}(BPS(p, s_0))$ induced by plan p starting at s_0, the remaining parts are same as those in plan obfuscation case.

5 Robot Demonstration

We now present a demonstration of *k-ambiguity* and *j-legibility* using a physical Fetch robot in a cooking domain. Here robot plays the role of an actor and a human plays the role of an observer. The goal of the robot is to add necessary ingredients to make a recipe. In the obfuscation setting, the robot's aim is to hide the ingredients from the human. This can be of use, especially, in scenarios where there is a need to preserve the secrecy of a recipe due to proprietary reasons. In contrast, in the legibility setting, the robot's aim is to inform the human about the ingredients being added. This could be seen from a household user's perspective, where the user needs to know what is being used in the recipe.

The domain consists of actions that allow the robot to communicate with the human, as well as pick-up and pour ingredients into a container. Each action can be performed in either an obfuscated or a legible way with different effects and different observations. We provide the action descriptions in Figures 3 and 4. From Figure 3, it can be seen that the action `ask-human-to-stir` distracts the human and allows the robot to obfuscate its goal. When the human is inattentive, the observation emitted for picking up any ingredient is generic. On the other hand, in Figure 4 the action `ask-human-to-pass-sugar` brings the human to attention. When the human is attentive, the observation emitted for picking up a container is specific for each ingredient. As shown in Figure 2, the scenario consists of a salt container and two sugar containers: one labeled as sugar, another without a label. The human is not aware of the contents of the unlabeled container.

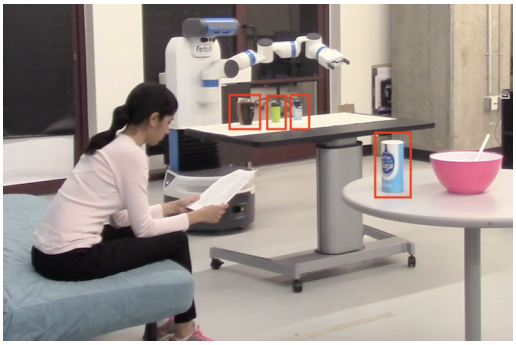


Figure 2: An illustration of the cooking domain. Here the robot is following a recipe and has to pick up and pour the ingredients into a cup. The robot has two containers available on its table, an unlabeled green colored container that contains sugar (unbeknownst to the human) and a labeled salt container. A labeled sugar container is available on the second table.

Table 1: Obfuscated and legible plans for cooking domain

True Goal	contents-added cup1 sugar
Decoy / Confounding	contents-added cup1 salt
Obfuscated Plan	ask-human-to-stir sugar container1, pickup-container-obfuscated sugar container1 table1, pour-ingredient-and-place sugar container1 cup1 table1
Obfuscation observation	human-inattentive, pickup-ingredient, pour-ingredient
Legible Plan	ask-human-to-pass-sugar sugar container2 table1, pickup-container-legible sugar container2 table1, pour-ingredient-and-place sugar container2 cup1 table1
Legible observation	human-attentive, pickup-sugar, pour-sugar

We used MoveIt (Sucan and Chitta 2013) software with `RRTConnectkConfigDefault` planner to perform motion planning. RViz was used to visualize the objects in the scene and the possible trajectories of the robot. AR markers were used to localize the objects in the scene. The candidate goal set along with *k-ambiguous* and *j-legible* plans are given in Table 1. Here $k = 2$ and $j = 1$. The videos for both obfuscation and legibility can be viewed at `https://youtu.be/mW7qCeggpUA` and `https://youtu.be/qzvTivKuszU`.

6 Empirical Evaluation

We now present an empirical analysis of all four approaches to demonstrate the performance and scope of our system. To this end, we use the following three IPC domains for our experiments: 15 problems from `BlocksWorld`, 10 from `Storage`, 10 from `Gripper`. We use grounded domains and problems in our experiments.

6.1 Observation Model Setup

We now discuss the design of $\mathcal{O}$ for the following domains:

BlocksWorld For blocksworld, the observation model is same as the one defined in Example 1. For the 15 problems used in this domain, the smallest problems had 26 fluents, 34 actions and the largest problems had 39 fluents, 74 actions.

```
(:action ask-human-to-stir
        :parameters (?x - ingredient ?y - container)
        :precondition (and
                (in ?x ?y)
                (not (obfuscated ?y))
        )
        :effect (and
                (human-inattentive)
                (increase (total-cost) 1)
        )
)

(:action pickup-container-obfuscated
        :parameters (?x - ingredient ?y - container
        ?z - table)
        :precondition (and
                (in ?x ?y)
                (handempty)
                (on ?y ?z)
                (accessible ?z)
                (human-inattentive)
        )
        :effect (and
                (not (handempty))
                (holding ?y)
                (not (on ?y ?z))
                (obfuscated ?y)
                (increase (total-cost) 1)
        )
)
```

Figure 3: Action description of obfuscating actions

Storage The storage domain has actions `move`, `drop`, `lift`, `go-out` and `go-in`. The observer cannot distinguish between lift and drop actions but receives an observation saying whether the hoist was used to perform an action. The observer can tell whether the agent is inside a particular storage area or outside. However, once the agent is inside a store area, the move actions do not reveal the agent's area. Therefore all move actions are of the same type. For the 10 test problems, the smallest problem had 12 fluents, 10 actions and the largest problem had 43 fluents, 78 actions.

Gripper The gripper domain has actions `move`, `drop` and `pickup`. The observer gets observation when the agent moves from one room to another. Also, the observer gets an observation regarding whether the gripper is holding something or not. Therefore in this domain, the observer can distinguish between all types of actions. However, the observer is not aware of the exact location of the agent. For the 10 problems used in this domain, the smallest problem had 21 fluents, 30 actions and the largest problem had 40 fluents, 96 actions after grounding.

6.2 Results

We provide evaluation of our approaches in Table 2 and 3. We wrote new planners from scratch for each of the the algorithms presented. We ran our experiments on Intel(R)

```
(:action ask-human-to-pass-sugar
        :parameters (?x - ingredient ?y - container
        ?z - table)
        :precondition (and
                (in ?x ?y)
                (is-type-labeled ?y)
                (accessible ?z)
                (not (legible ?y))
        )
        :effect (and
                (on ?y ?z)
                (human-attentive)
                (increase (total-cost) 1)
        )
)

(:action pickup-container-legible
        :parameters (?x - ingredient ?y - container
        ?z - table)
        :precondition (and
                (in ?x ?y)
                (handempty)
                (on ?y ?z)
                (accessible ?z)
                (is-type-labeled ?y)
                (human-attentive)
        )
        :effect (and
                (not (handempty))
                (holding ?y)
                (not (on ?y ?z))
                (legible ?y)
                (increase (total-cost) 1)
        )
)
```

Figure 4: Action description of legible actions

Domain	Metrics	*k-amb*	ℓ-*div* (action)	ℓ-*div* (causal)	ℓ-*div* (state)
Blocksworld	avg time	32.20	123.41	174.06	571.03
	sd time	82.15	155.72	210.49	169.37
	$\|O\|$	9.33	7.71	6.85	7.11
Storage	avg time	37.72	88.11	212.49	227.58
	sd time	35.80	90.38	374.14	250.79
	$\|O\|$	7.83	6.75	5.83	5.66
Gripper	avg time	56.49	175.56	592.94	149.63
	sd time	118.64	52.41	197.61	48.87
	$\|O\|$	6.88	4.3	5.12	4.55

Table 2: Empirical evaluation for goal obfuscation and plan obfuscation. We report average, standard deviation of time taken in seconds and the average observation sequence length of the obfuscated plans.

Xeon(R) CPU E5-2643v3, with a time out of 30 minutes. We created the planning problems in a randomized fashion. We report the performance of our approaches in terms of average and standard deviation for the time taken to run the problems in the given domain, and the average length of the observation sequence. For all the problems, the values used were $k = 5$, $\ell = 3$, $j = 3$ with $n - j = 2$, $m = 3$, $d_{min} = 0.25$ and $d_{max} = 0.50$.

Domain	Metrics	*j-leg*	*m-sim* (action)	*m-sim* (causal)	*m-sim* (state)
Blocksworld	avg time	204.12	59.63	73.56	81.07
	sd time	155.04	73.21	88.03	127.62
	$\|O\|$	6.9	6.93	7.14	6.85
Storage	avg time	14.21	36.34	31.97	38.79
	sd time	15.65	41.52	27.50	52.09
	$\|O\|$	5.27	9.8	9.66	10.12
Gripper	avg time	383.17	329.37	314.62	349.66
	sd time	178.14	131.70	112.64	159.65
	$\|O\|$	6.75	7.34	8.62	8.33

Table 3: Empirical evaluation for goal legibility and plan legibility. We report average, standard deviation of time taken in seconds and the average observation sequence length of the legible plans.

For all the domains, we note that the average length of *k-ambiguous* observation sequences is longer than that of *j-legible* observation sequences as obfuscated plans involve incorporating "time-wasting" actions. The computation of legible plans, in general, is a challenging task, as it is difficult to inform an observer about an agent's goals without any explicit communication. Moreover, the *j-legible* case limits the number of conforming goals to at most j. For *j-legible* case, $3/15$ problems in blocksworld and $1/10$ problems in gripper were unsolvable. We also note that the average observation length for all three distance measures is different, thus showing that they produce different plans. For ℓ-*diverse* and *m-similar* case, $2/10$ problems in storage domain, were unsolvable for all three distance measures.

6.3 Example

We illustrate the plans generated by our algorithms along with a default optimal plan generated by FD for two different observation models of `BlocksWorld`. $\mathcal{O}_1$ is same as the one discussed above with $|\Omega| = 4$. For $\mathcal{O}_2$, let's assume the agent can identify the block in its hand, then $|\Omega| = 16$. The initial state is, *on-B-C, on-C-A, on-A-D, ontable-D, clear-B, handempty*. The true goal is *on-A-B* and the decoy/confounding goals are *on-B-C, on-D-C*. We compute plans for $k = 3$, $\ell = 2$ with action distance and $j = 2$ in both the observation models. These plans along with accompanying observation sequences are given in Table 4. For the *j-legible* case, j consists of *on-B-C* and true goal *on-A-B*, while $n - j$ consists of *on-D-C*. We note that the obfuscated plans consist of "time-wasting" actions as can be seen in *k-amb* and ℓ-*div* examples. We can see that the *k-amb* and *m-sim* plans are affected by the observation model difference. However, the other two are not affected, this is mostly because this is a small problem for illustrative purposes. In general, plan computation particularly depends on the observation model.

Algo, $\mathcal{O}$	Plan	Observation Sequence
FD,$\{\mathcal{O}_1, \mathcal{O}_2\}$	unstack-B-C, putdown-B, unstack-C-A, putdown-C, unstack-A-D, stack-A-B	unstack, putdown, unstack, putdown, unstack, stack
k-amb, $\mathcal{O}_1$	unstack-B-C, putdown-B, unstack-C-A, putdown-C, unstack-A-D, stack-A-B, pickup-C, putdown-C, pickup-D, putdown-D, pickup-C, stack-C-D	unstack, putdown, unstack, putdown, unstack, stack, pickup, putdown, pickup, putdown, pickup, stack
k-amb, $\mathcal{O}_2$	unstack-B-C, putdown-B, unstack-C-A, putdown-C, unstack-A-D, stack-A-B, unstack-A-B, putdown-A, pickup-B, stack-B-C, pickup-A, stack-A-B	unstack-B, putdown-B, unstack-C, putdown-C, unstack-A, stack-A, unstack-A, putdown-A, pickup-B, stack-B, pickup-A, stack-A
ℓ-div, $\mathcal{O}_1$	unstack-B-C, putdown-B, unstack-C-A, stack-C-B, unstack-C-B, putdown-C, unstack-A-D, stack-A-B	unstack, putdown, unstack, stack, unstack, putdown, unstack, stack
ℓ-div, $\mathcal{O}_2$	unstack-B-C, putdown-B, unstack-C-A, stack-C-B, unstack-C-B, putdown-C, unstack-A-D, stack-A-B	unstack-B, putdown-B, unstack-C, stack-C, unstack-C, putdown-C, unstack-A, stack-A
j-leg, $\mathcal{O}_1$	unstack-B-C, putdown-B, unstack-C-A, putdown-C, pickup-B, stack-B-C, unstack-A-D, stack-A-B	unstack, putdown, unstack, putdown, pickup, stack, unstack, stack
j-leg, $\mathcal{O}_2$	unstack-B-C, putdown-B, unstack-C-A, putdown-C, pickup-B, stack-B-C, unstack-A-D, stack-A-B	unstack-B, putdown-B, unstack-C, putdown-C, pickup-B, stack-B, unstack-A, stack-A
m-sim, $\mathcal{O}_1$	unstack-B-C, putdown-B, unstack-C-A, putdown-C, unstack-A-D, stack-A-B	unstack, putdown, unstack, putdown, unstack, stack
m-sim, $\mathcal{O}_2$	unstack-B-C, putdown-B, unstack-C-A, putdown-C, unstack-A-D, putdown-A pickup-A, stack-A-B	unstack-B, putdown-B, unstack-C, putdown-C, unstack-A, putdown-A, pickup-A, stack-A

Table 4: Examples of plans generated for two different observation models

7 Related Work

There are prior works which discuss the problem of privacy preservation in distributed multi-agent systems (Brafman 2015; Luis and Borrajo 2014; Bonisoli et al. 2014). A recent work on privacy for multi-agents of Maliah, Shani, and Stern (2016) is complementary to our approach, as they consider problems where the model needs to be protected from the team members but goals and behavior are coordinated. In contrast, we consider problems where the models are public but goals and behavior need to be protected.

The problem of goal obfuscation is also related to plan recognition literature (Ramırez and Geffner 2009; 2010; E-Martin, R-Moreno, and Smith 2015; Sohrabi, Riabov, and Udrea 2016; Keren, Gal, and Karpas 2016a). Traditional plan recognition systems have focused on scenarios where actions being executed can be observed directly. In our case, observational equivalence due to the many-to-one formulation of $\mathcal{O}$ introduces, in effect, noisy action-state observations. This, in turn, complicates plan recognition. More crucially, the agent uses the observational equivalence to actively help or hinder the ease of plan recognition.

There are a few recent works which have explored the idea of obfuscation in adversarial settings from the goal recognition aspect (Keren, Gal, and Karpas 2016b; Masters and Sardina 2017). One of the closely related work is that of Keren, Gal, and Karpas (2016b) on privacy preservation, in which the authors propose a solution that obfuscates a goal by choosing one of the candidate goals that has the maximum non-distinct path in common with the true goal, which obfuscates part of the plan. In contrast, our plans are obfuscated for the entire length such that, at least k goals are consistent with the observations. Also, our framework supports the case of plan obfuscation which prevents the next step from being deciphered by making it consistent with ℓ diverse plans, and the case of a cooperative observer which make the agent's intentions legible to the observer by being consistent with at most j goals.

The notions of k-anonymity (Sweeney 2002) and l-diversity (Machanavajjhala et al. 2006) were originally developed in the literature on privacy and security for relational databases. In motion planning and robotics community, legibility (Dragan and Srinivasa 2013; Knepper et al. 2017) has been a well-studied topic. However, this has been mostly looked at from the motion planning perspective, and therefore the focus has been on optimizing the motion trajectories such that the goal is revealed. We borrow these notions and generalize it in a unified framework to provide obfuscated and legible plans from a task planning perspective.

7.1 Compilation to Model Uncertainty

In recent years, there has been some interesting research in the field of human aware planning. Especially the work on explainable AI and explanations (Fox, Long, and Magazzeni 2017; Zhang et al. 2017; Chakraborti et al. 2017) proposes modeling the human's understanding of a planning agent and introduces the notion of human-aware multi-model planning. Their framework consists of two models representing the planner's domain model and the observing or interacting human's understanding of the planning model. This setting captures the uncertainty of the observer in the form of human's partial or incorrect model of the agent. On the other hand, our setting also explores uncertainty of the observer's understanding of the plans computed by the planner. However, we capture the uncertainty in form of a partial observation model. We hypothesize that the two settings can be compiled from one formulation to another, and can be perceived as primal and dual problems. We intend to investigate this direction in future work.

8 Conclusion

We introduced a unified framework that gives a planner the capability of addressing both adversarial and cooperative situations. Our setting assumes that the observer has partial visibility of the agent's actions, but is aware of agent's planning capabilities. We define four problems: goal obfuscation and goal legibility when the agent's true goal is unknown and, plan obfuscation and plan legibility when the agent's true goal is known. We propose the following solutions to these problems: *k-ambiguous* plan which obfuscates the true goal with respect to at least k goals, *j-legible* plan which

enables an observer to quickly understand the j true goals of the agent, ℓ-*diverse* plan which obfuscates the next actions in a plan and, *m-similar* plan which reveals the next actions in the plan. We present different search techniques to achieve these solutions and evaluate the performance of our approaches using three IPC domains: `BlocksWorld`, `Storage` and `Gripper`. We also demonstrate the goal obfuscation and goal legibility problems using the Fetch robot in a cooking domain.

Acknowledgments

This research is supported in part by the AFOSR grant FA9550-18-1-0067, the ONR grants N00014-16-1-2892, N00014-13-1-0176, N00014-13- 1-0519, N00014-15-1-2027, N00014-18-1-2442 and the NASA grant NNX17AD06G.

References

Blum, A. L., and Furst, M. L. 1997. Fast planning through planning graph analysis. *Artificial intelligence* 90(1):281–300.

Bonet, B., and Geffner, H. 2014. Belief tracking for planning with sensing: Width, complexity and approximations. *Journal of Artificial Intelligence Research* 50:923–970.

Bonisoli, A.; Gerevini, A. E.; Saetti, A.; and Serina, I. 2014. A privacy-preserving model for the multi-agent propositional planning problem. In *Proceedings of the Twenty-first European Conference on Artificial Intelligence*, 973–974.

Brafman, R. I. 2015. A privacy preserving algorithm for multi-agent planning and search. In *IJCAI*, 1530–1536.

Chakraborti, T.; Sreedharan, S.; Zhang, Y.; and Kambhampati, S. 2017. Plan explanations as model reconciliation: Moving beyond explanation as soliloquy. In *IJCAI*.

Dragan, A., and Srinivasa, S. 2013. Generating legible motion. In *Proceedings of Robotics: Science and Systems*.

E-Martin, Y.; R-Moreno, M. D.; and Smith, D. E. 2015. A fast goal recognition technique based on interaction estimates. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Fox, M.; Long, D.; and Magazzeni, D. 2017. Explainable planning. *arXiv preprint arXiv:1709.10256*.

Geffner, H., and Bonet, B. 2013. A concise introduction to models and methods for automated planning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 8(1):1–141.

Haslum, P., and Jonsson, P. 1999. Some results on the complexity of planning with incomplete information. In *European Conference on Planning*, 308–318. Springer.

Keren, S.; Gal, A.; and Karpas, E. 2016a. Goal recognition design with non-observable actions. In *AAAI*, 3152–3158.

Keren, S.; Gal, A.; and Karpas, E. 2016b. Privacy preserving plans in partially observable environments. In *IJCAI*, 3170–3176.

Knepper, R. A.; Mavrogiannis, C. I.; Proft, J.; and Liang, C. 2017. Implicit communication in a joint action. In *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, 283–292. ACM.

Luis, N., and Borrajo, D. 2014. Plan merging by reuse for multi-agent planning. *Distributed and Multi-Agent Planning* 38.

Machanavajjhala, A.; Gehrke, J.; Kifer, D.; and Venkitasubramaniam, M. 2006. l-diversity: Privacy beyond k-anonymity. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, 24–24. IEEE.

Maliah, S.; Shani, G.; and Stern, R. 2016. Stronger privacy preserving projections for multi-agent planning. In *ICAPS*, 221–229.

Masters, P., and Sardina, S. 2017. Deceptive path-planning. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, 4368–4375.

Nguyen, T. A.; Do, M.; Gerevini, A. E.; Serina, I.; Srivastava, B.; and Kambhampati, S. 2012. Generating diverse plans to handle unknown and partially known user preferences. *Artificial Intelligence* 190(0):1 – 31.

Nguyen, X.; Kambhampati, S.; and Nigenda, R. S. 2002. Planning graph as the basis for deriving heuristics for plan synthesis by state space and csp search. *Artificial Intelligence* 135(1-2):73–123.

Ramırez, M., and Geffner, H. 2009. Plan recognition as planning. In *Proceedings of the 21st international joint conference on Artifical intelligence. Morgan Kaufmann Publishers Inc*, 1778–1783.

Ramırez, M., and Geffner, H. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI 2010)*.

Rintanen, J. 2004. Complexity of planning with partial observability. In *ICAPS*, 345–354.

Savitch, W. J. 1970. Relationships between nondeterministic and deterministic tape complexities. *Journal of computer and system sciences* 4(2):177–192.

Sohrabi, S.; Riabov, A. V.; and Udrea, O. 2016. Plan recognition as planning revisited. In *IJCAI*, 3258–3264.

Song, D. X.; Wagner, D.; and Tian, X. 2001. Timing analysis of keystrokes and timing attacks on ssh. In *USENIX Security Symposium*, volume 2001.

Srivastava, B.; Nguyen, T. A.; Gerevini, A.; Kambhampati, S.; Do, M. B.; and Serina, I. 2007. Domain independent approaches for finding diverse plans. In *IJCAI*, 2016–2022.

Sucan, I. A., and Chitta, S. 2013. Moveit! *Online at http://moveit. ros. org*.

Sweeney, L. 2002. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10(05):557–570.

Zhang, Y.; Sreedharan, S.; Kulkarni, A.; Chakraborti, T.; Zhuo, H. H.; and Kambhampati, S. 2017. Plan explicability and predictability for robot task planning. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*.

Representing and Reasoning with Intentional Actions on a Robot

Rocio Gomez
Electrical and Computer Engineering
The University of Auckland, NZ
m.gomez@auckland.ac.nz

Mohan Sridharan
School of Computer Science
University of Birmingham, UK
m.sridharan@bham.ac.uk

Heather Riley
Electrical and Computer Engineering
The University of Auckland, NZ
hril230@aucklanduni.ac.nz

Abstract

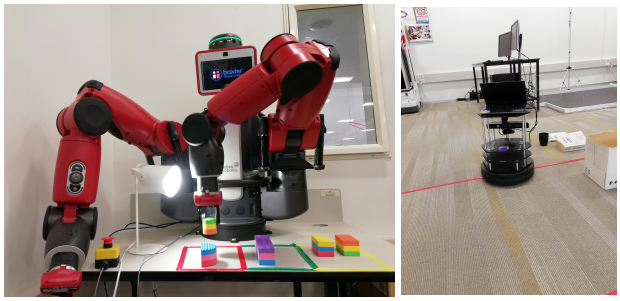
This paper describes a general architecture for robots to represent and reason with intentional actions. The architecture reasons with tightly-coupled transition diagrams of the domain at two different resolutions. Non-monotonic logical reasoning with a coarse-resolution transition diagram is used to compute a plan comprising intentional abstract actions for any given goal. Each such abstract action is implemented as a sequence of concrete actions by automatically zooming to and reasoning with the relevant part of a fine-resolution transition diagram that is defined as a refinement of the coarse-resolution transition diagram. The execution of each concrete action uses probabilistic models of uncertainty in sensing and actuation, and the outcomes of executing the sequence of concrete actions are added to the coarse-resolution history. The capabilities of this architecture are illustrated in the context of a simulated robot assisting humans in an office domain, on a physical robot (Baxter) manipulating tabletop objects, and on a wheeled robot (Turtlebot) moving objects to particular places or people in an office. We show that this architecture improves reliability and efficiency in comparison with a planning architecture that does not include intentional actions.

1 Introduction

Consider robots assisting humans in dynamic domains, e.g., a robot helping a human arrange objects in different configurations on a tabletop in Figure 1a, or a robot delivering objects to particular places or people in Figure 1b. These robots often have to reason with different descriptions of uncertainty and incomplete domain knowledge. This information about the domain often includes commonsense knowledge, especially default knowledge that holds in all but a few exceptional circumstances, e.g., "books are usually in the library but cookbooks may be in the kitchen". The robot also receives a lot more sensor data than it can process, and it is equipped with many algorithms that compute and use a probabilistic quantification of the uncertainty in sensing and actuation, e.g., "I am 90% certain the robotics book is on the table". Furthermore, while it is difficult to provide robots comprehensive domain knowledge or elaborate supervision, reasoning with incomplete or incorrect information can provide incorrect or suboptimal outcomes. This loss in performance is more pronounced in scenarios corresponding to unexpected success or failure, which are common in dynamic domains. For instance, consider a robot trying to move two books from an office to a library. After moving the first book to the library, if the robot observes the second book in the library, or if it observes the second book in the kitchen on the way back to the office, it should stop executing its plan, reason about what may have happened, and compute a new plan if necessary. One way to achieve this behavior is to augment a traditional planning approach with the ability to reason about observations of all domain objects and events during plan execution, but this approach is computationally intractable in complex domains. Instead, the architecture described in this paper seeks to enable a robot pursuing a particular goal to automatically reason about the underlying *intention* and related observations of its domain during planning and execution. It does so by building on an architecture that uses declarative programming to reason about intended actions to achieve a given goal (Blount, Gelfond, and Balduccini 2015), and on an architecture that reasons with tightly-coupled transition diagrams at different levels of abstraction (Sridharan et al. 2017). We describe the following characteristics of the architecture:

- An action language is used to describe the tightly-coupled transition diagrams of the domain at two different resolutions. At the coarse resolution, non-monotonic logical reasoning with commonsense knowledge, including default knowledge, produces a sequence of intentional abstract actions for any given goal.
- Each intended abstract action is implemented as a sequence of concrete actions by automatically zooming to and reasoning with the relevant part of the fine-resolution system description defined as a refinement of the coarse-resolution system description. The outcomes of executing the concrete actions using probabilistic models or uncertainty are added to the coarse-resolution history.

In this paper, the coarse-resolution and fine-resolution action language descriptions are translated to programs in CR-Prolog, an extension of Answer Set Prolog (ASP) (Gelfond and Kahl 2014), for commonsense reasoning. The execution of each concrete action using probabilistic models of uncertainty in sensing and actuation is achieved using existing algorithms. The architecture thus reasons about intentions and beliefs at different levels of resolution. We demonstrate the general applicability of our architecture in the context of (i) a simulated robot assisting humans in an office do-



(a) Baxter robot. (b) Turtlebot.

Figure 1: (a) Baxter robot manipulating objects on a tabletop; and (b) Turtlebot moving objects to particular locations in a lab.

main; (ii) a physical robot (Baxter) manipulating objects on a tabletop; and (iii) a wheeled robot (Turtlebot) moving objects to desired locations in an office domain. We show that the proposed architecture improves reliability and computational efficiency of planning and execution in dynamic domains in comparison with a planning architecture that does not support reasoning about intentional actions.

2 Related Work

There is much work in the modeling and recognition of intentions. Belief-desire-intention (BDI) architectures model the intentions of reasoning agents and guide reasoning by eliminating choices inconsistent with current intentions (Bratman 1987; Rao and Georgeff 1995). However, such architectures do not learn from past behavior, adapt to new situations, or include an explicit representation of (or reasoning about) goals. Other work has reasoned with domain knowledge or used models learned from training samples to recognize intentions (Kelley et al. 2008).

An architecture formalizing intentions based on declarative programming was described in (Baral and Gelfond 2005). It introduced an action language that can represent intentions based on two principles: (i) *non-procrastination*, i.e., intended actions are executed as soon as possible; and (ii) *persistence*, i.e., unfulfilled intentions persist. This architecture was also used to enable an external observer to recognize the activity of an observed agent, i.e., for determining what has happened and what the agent intends to do (Gabaldon 2009). However, this architecture did not support the modeling of agents that desire to achieve specific goals. The *Theory of Intentions* ($\mathcal{TI}$) (Blount, Gelfond, and Balduccini 2015; 2014) builds on (Baral and Gelfond 2005) to model the intentions of goal-driven agents. $\mathcal{TI}$ expanded transition diagrams that have physical states and physically executable actions to include mental fluents and mental actions. It associated a sequence of agent actions (called an "activity") with the goal it intended to achieve, and introduced an *intentional agent* that only performs actions that are intended to achieve a desired goal and does so without delay. This theory has been used to create a methodology for understanding of narratives of typical and exceptional restaurant scenarios (Zhang and Inclezan 2017), and goal-driven agents in dynamic domains have been modeled using such activities (Saribatur, Baral, and Eiter 2017). A common requirement of such theories and their use is that all the domain knowledge, including the preconditions and effects of actions and potential goals, be known and encoded in the knowledge base, which is difficult to do in robot domains. Also, the set of states (and actions, observations) to be considered can be large in robot domains, which makes efficient reasoning a challenging task. In recent work (Zhang and Inclezan 2017), the authors attempt to address this problem by clustering indistinguishable states (Saribatur and Eiter 2016) but these clusters need to be encoded in advance. Furthermore, these approaches do not consider the uncertainty in sensing and actuation.

Many logic-based methods have been used in robotics, including those that also support probabilistic reasoning (Hanheide et al. 2017; Zhang, Sridharan, and Wyatt 2015). Methods based on first-order logic do not support non-monotonic logical reasoning or the desired expressiveness for capabilities such as default reasoning, e.g., it is not always meaningful to express degrees of belief by attaching probabilities to logic statements. Non-monotonic logics such as ASP address these limitations and have been used in cognitive robotics applications (Erdem and Patoglu 2012), but classical ASP formulations do not support the probabilistic models of uncertainty that are used by algorithms for sensing and actuation. Approaches based on logic programming also do not support one or more of the capabilities such as incremental addition of probabilistic information or variables to reason about open worlds. Our prior refinement-based architecture reasoned with tightly-coupled transition diagrams at two resolutions; each abstract action in a coarse-resolution plan computed using ASP was executed as a sequence of concrete actions computed by probabilistic reasoning over the relevant part of the fine-resolution diagram (Sridharan et al. 2017; Sridharan and Gelfond 2016). This paper explores the combination of these ideas with those drawn from $\mathcal{TI}$; specific differences from prior work are described below.

3 Cognitive Architecture

Figure 2 presents a block diagram of the overall architecture. Similar to prior work (Sridharan et al. 2017), this architecture may be viewed as consisting of three components: controller, logician, and executor. In this paper, the controller is responsible for holding the overall beliefs regarding domain state, and for the transfer of control and information between all components. For any given goal, the logician performs non-monotonic logical reasoning with the coarse-resolution representation of commonsense knowledge to generate an activity, i.e., a sequence of intentional abstract actions. Each abstract action is implemented as a sequence of concrete actions by zooming to and reasoning with a fine-resolution representation defined as a refinement of the coarse-resolution representation. The executor uses probabilistic models of the uncertainty in sensing and actuation to execute each concrete action, with the outcomes being communicated to the controller and added to the coarse-resolution history of the logician. These components of the architecture are described below, along with differences from prior work, using variants of the following illustrative domain.

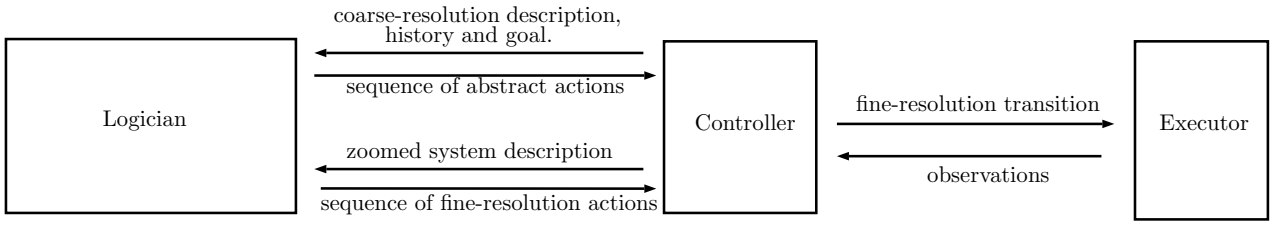


Figure 2: Architecture represents intentions and beliefs as tightly coupled transition diagrams at two different resolutions. It combines the complementary strengths of declarative programming and probabilistic reasoning, and may be viewed as interactions between a controller, logician, and executor.

Example 1 *[Robot Assistant (RA) Domain]* Consider a robot assisting humans in moving particular objects to desired locations in an indoor office domain with:

- Sorts such as place, thing, robot, object, and book, arranged hierarchically, e.g., object and robot are subsorts of thing. Sort names and constants are in lowercase, and variable names are in uppercase.
- Places: $\{office_1, office_2, kitchen, library\}$ with a door between neighboring places; only door between kitchen and library can be locked—Figure 3.
- Instances of sorts, e.g., rob_1, $book_1$, $book_2$.
- Static attributes such as color, size and different parts (e.g., base and handle) associated with objects.
- Other agents that may influence the domain, e.g., move a book or lock a door. These agents are not modeled.

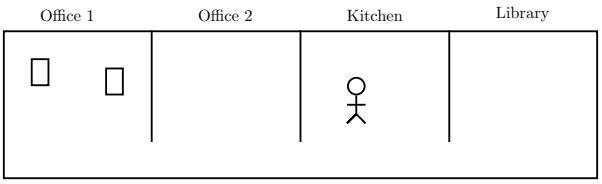


Figure 3: Four rooms considered in Example 1, with a human in the kitchen and two books in $office_1$. Only the library's door can be locked; all other rooms are open at all times.

3.1 Action Language and Domain Representation

We first describe the action language encoding of domain dynamics, and its translation to CR-Prolog programs for knowledge representation and reasoning.

Action Language Action languages are formal models of parts of natural language used for describing transition diagrams of dynamic systems. We use action language $\mathcal{AL}_d$ (Gelfond and Inclezan 2013) to describe the transition diagrams at different resolutions. $\mathcal{AL}_d$ has a sorted signature with *statics*, *fluents* and *actions*. Statics are domain attributes whose truth values cannot be changed by actions, whereas fluents are domain attributes whose truth values can be changed by actions. Fluents can be *basic* or *defined*. Basic fluents obey the laws of inertia and can be changed by actions. Defined fluents do not obey the laws of inertia and are not changed directly by actions—their values depend on other fluents. Actions are defined as a set of elementary operations. A domain attribute p or its negation $\neg p$ is a *literal*. $\mathcal{AL}_d$ allows three types of statements:

a **causes** l_b **if** $p_0, \ldots, p_m$ (Causal law)
l **if** $p_0, \ldots, p_m$ (State constraint)
impossible $a_0, \ldots, a_k$ **if** $p_0, \ldots, p_m$ (Executability condition)

where a is an action, l is a literal, l_b is a basic literal, and $p_0, \ldots, p_m$ are domain literals.

Knowledge Representation The domain representation consists of system description $\mathcal{D}$, a collection of statements of $\mathcal{AL}_d$, and history $\mathcal{H}$. System description $\mathcal{D}$ has a sorted signature Σ and axioms that describe the transition diagram τ. Σ defines the basic sorts, domain attributes and actions. Example 1 introduced some basic sorts and ground instances of the RA domain. Σ also includes the sort step for temporal reasoning. Domain attributes (i.e., statics and fluents) and actions are described in terms of their arguments' sorts. In the RA domain, statics include relations such as $next\_to(place, place)$, which describes the relative location of places in the domain; and relations representing object attributes such as color and size, e.g., $obj\_color(object, color)$. Fluents include $loc(thing, place)$, the location of the robot or domain objects; $in\_hand(robot, object)$, which denotes a particular object is in the robot's hand; and $locked(place)$, which implies a particular place is locked. The locations of other agents, if any, are not changed by the robot's actions; these locations are inferred from observations obtained from other sensors. The domain's actions include $move(robot, place)$, $pickup(robot, object)$, $putdown(robot, object)$, and $unlock(robot, place)$; we also consider exogenous actions $exo\_move(object, place)$ and $exo\_lock(place)$, which are used for diagnostic reasoning. Σ also includes the relation $holds(fluent, step)$ to imply that a particular fluent holds true at a particular time step.

Axioms for the RA domain include causal laws, state constraints and executability conditions such as:

$move(rob_1, P)$ **causes** $loc(rob_1, P)$
$pickup(rob_1, O)$ **causes** $in\_hand(rob_1, O)$
$\neg loc(Th, L_2)$ **if** $loc(Th, L_1)$, $L_1 \neq L_2$
$loc(O, P)$ **if** $loc(rob_1, P)$, $in\_hand(rob_1, O)$

impossible $\mathtt{pickup}(\mathtt{rob}_1, \mathtt{O})$ **if** $\mathtt{loc}(\mathtt{rob}_1, \mathtt{L}_1)$, $\mathtt{loc}(\mathtt{O}, \mathtt{L}_2)$, $\mathtt{L}_1 \neq \mathtt{L}_2$

The history $\mathcal{H}$ of a dynamic domain is usually a record of fluents observed to be true or false at a particular time step, i.e., $\mathtt{obs}(\mathtt{fluent}, \mathtt{boolean}, \mathtt{step})$, and the occurrence of an action at a particular time step, i.e., $\mathtt{occurs}(\mathtt{action}, \mathtt{step})$. In (Sridharan et al. 2017) this notion was expanded to represent defaults describing the values of fluents in the initial state, e.g., "books are usually in the library and if it not there, they are normally in the office" is encoded as:

initial default $\mathtt{loc}(\mathtt{X}, \mathtt{library})$ **if** $\mathtt{book}(\mathtt{X})$

initial default $\mathtt{loc}(\mathtt{X}, \mathtt{office}_1)$ **if** $\mathtt{book}(\mathtt{X})$, $\neg\mathtt{loc}(\mathtt{X}, \mathtt{library})$

We can also encode exceptions to these defaults, e.g., "cookbooks are in the kitchen". Such a representation, which does not use numerical values to model degrees of belief in these defaults, supports elegant reasoning with generic defaults and their specific exceptions.

Reasoning The domain representation is translated into a program $\Pi(\mathcal{D}, \mathcal{H})$ in CR-Prolog[1], a variant of ASP that incorporates consistency restoring (CR) rules (Balduccini and Gelfond 2003). ASP is based on stable model semantics and supports concepts such as *default negation* and *epistemic disjunction*, e.g., unlike "$\neg$a" that states *a is believed to be false*, "not a" only implies *a is not believed to be true*. ASP can represent recursive definitions and constructs that are difficult to express in classical logic formalisms, and it supports non-monotonic logical reasoning, i.e., it is able to revise previously held conclusions based on new evidence. An ASP program Π includes the signature and axioms of $\mathcal{D}$, inertia axioms, reality checks, and observations, actions, and defaults from $\mathcal{H}$. Every default also has a CR rule that allows the robot to assume the default's conclusion is false to restore consistency under exceptional circumstances. For instance, the following statement in the ASP program:

$$\neg\mathtt{loc}(\mathtt{X}, \mathtt{library}) \stackrel{+}{\leftarrow} \mathtt{book}(\mathtt{X})$$

is triggered under exceptional circumstances to consider the rare event of a book not being in the library, as a potential explanation of an unexpected observation. Each *answer set* of an ASP program represents the set of beliefs of an agent associated with the program. Algorithms for computing entailment, and for planning and diagnostics, reduce these tasks to computing answer sets of CR-Prolog programs. We compute answer sets of CR-Prolog programs using the system called SPARC (Balai, Gelfond, and Zhang 2013).

3.2 Adapted Theory of Intention

For any given goal, a robot using ASP-based reasoning will compute a plan and execute it until the goal is achieved or an action in the plan has an unexpected outcome; in the latter case, the robot will attempt to explain the outcome (i.e., perform diagnostics) and compute a new plan if necessary.

[1]We use the terms "ASP" and "CR-Prolog" interchangeably.

To motivate the need for a different approach in dynamic domains, consider the following scenarios in which the goal is to move $\mathtt{book}_1$ and $\mathtt{book}_2$ to the library; these scenarios have been adapted from scenarios considered in prior work (Blount, Gelfond, and Balduccini 2015):

- **Scenario 1 (planning):** Robot $\mathtt{rob}_1$ is in the kitchen holding $\mathtt{book}_1$, and believes $\mathtt{book}_2$ is in the kitchen and that the library is unlocked. The computed plan is: $\mathtt{move}(\mathtt{rob}_1, \mathtt{library})$, $\mathtt{put\_down}(\mathtt{rob}_1, \mathtt{book}_1)$, $\mathtt{move}(\mathtt{rob}_1, \mathtt{kitchen})$, $\mathtt{pickup}(\mathtt{rob}_1, \mathtt{book}_2)$, $\mathtt{move}(\mathtt{rob}_1, \mathtt{library})$, $\mathtt{put\_down}(\mathtt{rob}_1, \mathtt{book}_2)$.

- **Scenario 2 (unexpected success):** Assume that $\mathtt{rob}_1$ in Scenario-1 has moved to the library and put $\mathtt{book}_1$ down, and observes $\mathtt{book}_2$ there. The robot should be able to explain this observation (e.g., $\mathtt{book}_2$ was moved there) and realize the goal has been achieved.

- **Scenario 3 (not expected to achieve goal, diagnose and replan, case 1):** Assume rob1 in Scenario-1 starts moving $\mathtt{book}_1$ to library, but observes $\mathtt{book}_2$ is not in the kitchen. The robot should realize the plan will fail to achieve the overall goal, explain the unexpected observation, and compute a new plan.

- **Scenario 4 (not expected to achieve goal, diagnose and replan, case 2):** Assume rob1 is in the kitchen holding book1, and believes book2 is in $\mathtt{office}_2$ and library is unlocked. The robot plans to put $\mathtt{book}_1$ in the library before fetching $\mathtt{book}_2$ from $\mathtt{office}_2$. Before $\mathtt{rob}_1$ moves to the library, it suddenly observes $\mathtt{book}_2$ in the kitchen. The robot should realize the plan will fail, explain the observation, and compute a new plan.

- **Scenario 5 (failure to achieve the goal, diagnose and replan):** Assume $\mathtt{rob}_1$ in Scenario-1 is putting $\mathtt{book}_2$ in the library, after having put $\mathtt{book}_1$ in the library earlier, and observes that $\mathtt{book}_1$ is no longer there. The robot's intention should persist; it should explain the unexpected observation, replan if necessary, and execute actions until the goal is achieved.

One way to support the desired behavior in such scenarios is to reason with all possible observations of domain objects and events (e.g., observations of all objects in the sensor's field of view) during plan execution. However, such an approach would be computationally intractable in complex domains. Instead, we build on the principles of non-procrastination and persistence and the ideas from $\mathcal{TI}$. Our architecture enables the robot to compute actions that are intended for any given goal and current beliefs. As the robot attempts to implement each such action, *it obtains all observations relevant to this action and the intended goal*, and adds these observations to the recorded history. We will henceforth use $\mathcal{ATI}$ to refer to this adapted theory of intention that expands both the system description $\mathcal{D}$ and history $\mathcal{H}$ in the original program $\Pi(\mathcal{D}, \mathcal{H})$. First, the signature Σ is expanded to represent an *activity*, a triplet of a *goal*, a *plan* to achieve the goal, and a specific *name*, by introducing

relations such as:

$$\texttt{activity}(\texttt{name}),\ \ \texttt{activity\_goal}(\texttt{name},\texttt{goal})$$
$$\texttt{activity\_length}(\texttt{name},\texttt{length})$$
$$\texttt{activity\_component}(\texttt{name},\texttt{number},\texttt{action})$$

These relations represent each named activity, the goal and length of each activity, and the actions that are the components of the activity. Note that when these relations are ground, they are statics.

Next, the existing fluents of Σ are considered to be *physical fluents* and the set of fluents is expanded to include *mental fluents* such as:

$$\texttt{active\_activity}(\texttt{activity}),\ \texttt{in\_progress\_goal}(\texttt{goal})$$
$$\texttt{next\_action}(\texttt{activity},\texttt{action}),$$
$$\texttt{in\_progress\_activity}(\texttt{activity}),$$
$$\texttt{active\_goal}(\texttt{goal}),\ \texttt{next\_activity\_name}(\texttt{name})$$
$$\texttt{current\_action\_index}(\texttt{activity},\texttt{index})$$

where the relations in the first three lines are defined fluents, whereas the other relations correspond to basic fluents. These fluents represent the robot's belief about a particular activity, action or goal being active or in progress. None of these fluents' values are changed directly by executing any physical action. The value of *current_action_index* changes if the robot has completed an intended action or if a change in the domain makes it impossible for an activity to succeed. The values of the other mental fluents are changed directly or indirectly by expanding the set of existing *physical actions* of Σ to include *mental actions* such as:

$$\texttt{start}(\texttt{name}),\ \ \texttt{stop}(\texttt{name})$$
$$\texttt{select}(\texttt{goal}),\ \ \texttt{abandon}(\texttt{goal})$$

where the first two mental actions are used by the controller to start or stop a particular activity, and the other two action are exogenous actions used (e.g., by a human) to select or abandon a goal.

We also define new axioms in $\mathcal{AL}_{\mathrm{d}}$, e.g., to represent the effects of actions, prevent certain outcomes, and generate intentional actions—we do not describe these here due to space constraints. The notion of history is also expanded to include statements such as:

$$\texttt{attempt}(\texttt{action},\texttt{step}) \qquad \neg\, \texttt{hpd}(\texttt{action},\texttt{step})$$

which denote that a particular action was attempted at a particular time step, and that a particular action did not happen (i.e., was not executed successfully) at a particular time step. The revised system description $\mathcal{D}'$ and history $\mathcal{H}'$ are translated automatically to CR-Prolog program $\Pi(\mathcal{D}',\mathcal{H}')$ that is solved for planning or diagnostics. The complete program for the RA domain is available online (Software 2018).

Key differences between $\mathcal{ATI}$ and prior work on $\mathcal{TI}$ are:

- $\mathcal{TI}$ becomes computationally expensive, especially as the size of the plan or history increases. It also performs diagnostics and planning jointly, which allows it to consider different explanations during planning but increases computational cost in complex domains. $\mathcal{ATI}$, on the other hand, first builds a consistent model of history by considering different explanations, and *uses this model to guide planning*, significantly reducing computational cost in complex domains.
- $\mathcal{TI}$ assumes complete knowledge of the state of other agents (e.g., humans or other robots) that perform exogenous actions. In many robotics domains, this assumption is rather unrealistic. $\mathcal{ATI}$ instead makes the more realistic assumption that the robot can only infer exogenous actions by reasoning with the observations that it obtains from sensors.
- $\mathcal{ATI}$ does not include the notion of sub-goals and sub-activities (and associated relations) from $\mathcal{TI}$, as they were not necessary. Also, the sub-activities and sub-goals will need to be encoded in advance, and reasoning with these relations will also increase computational complexity in many situations. The inclusion of sub-activities and sub-goals will be explored in future work.

Any architecture with $\mathcal{ATI}$, $\mathcal{TI}$, or a different reasoning component based on logic-programming or classical first-order logic, has two key limitations. First, reasoning does not scale well to the finer resolution required for many tasks to be performed by the robot. For instance, the coarse-resolution representation discussed so far is not sufficient if the robot has to grasp and pickup a particular object from a particular location, and reasoning logically over a sufficiently fine-grained domain representation will be computationally expensive. Second, we have not yet modeled the actual sensor-level observations of the robot or the uncertainty in sensing and actuation. Section 2 further discusses the limitations of other approaches based on logical and/or probabilistic reasoning for robotics domains. Our architecture seeks to address these limitations by combining $\mathcal{ATI}$ with ideas drawn from work on a refinement-based architecture (Sridharan et al. 2017).

3.3 Refinement, Zooming and Execution

Consider a coarse-resolution system description $\mathcal{D}_{\mathrm{c}}$ of transition diagram τ_{c} that includes $\mathcal{ATI}$. For any given goal, reasoning with $\Pi(\mathcal{D}_{\mathrm{c}},\mathcal{H}_{\mathrm{c}})$ will provide an activity, i.e., a sequence of abstract intentional actions. In our architecture, the execution of the coarse-resolution transition corresponding to each such abstract action is based on a fine-resolution system description $\mathcal{D}_{\mathrm{f}}$ of transition diagram τ_{f}, which is a *refinement* of, and is tightly coupled to, $\mathcal{D}_{\mathrm{c}}$. We can imagine refinement as taking a closer look at the domain through a magnifying lens, potentially leading to the discovery of structures that were previously abstracted away by the designer (Sridharan et al. 2017). $\mathcal{D}_{\mathrm{f}}$ is constructed automatically as a step in the design methodology using $\mathcal{D}_{\mathrm{c}}$ and some domain-specific information provided by the designer.

First, the signature Σ_{f} of $\mathcal{D}_{\mathrm{f}}$ includes each basic sort of $\mathcal{D}_{\mathrm{c}}$ whose elements have not been *magnified* by the increase in resolution, or both the coarse-resolution copy and its fine-resolution *counterparts* for sorts with magnified elements.

For instance, sorts in the RA domain include:

$$\begin{aligned}
&place^* = \{office_1, office_2, kitchen, library\}\\
&place = \{c_1, \ldots, c_m\}\\
&cup^* = \{cup_1\}\\
&cup = \{cup_1_base, cup_1_handle\}\\
&book = \{book_1, book_2\}
\end{aligned}$$

where $\{c_1, \ldots, c_m\}$ are the cells that are the components of the original set of places, and any cup has a $base$ and $handle$ as components; any $book$, on the other hand, is not magnified and has no components. We also include domain-dependent statics relating the magnified objects and their counterparts, e.g., $component(cup\_base, cup)$. Next, domain attributes of Σ_f include the coarse-resolution version and fine-resolution counterparts (if any) of each domain attribute of Σ_c. For instance, in the RA domain, Σ_f will include domain attributes such as:

$$\begin{aligned}
&loc^*(thing^*, place^*),\ next_to^*(place^*, place^*)\\
&loc(thing, place),\ next_to(place, place)
\end{aligned}$$

where relations with and without the "*" represent the coarse-resolution counterparts and fine-resolution counterparts respectively. The specific relations listed above describe the location of each thing at two different resolutions, and describe two places or cells that are next to each other. Actions of Σ_f include (a) every action in Σ_c with its magnified parameters replaced by fine-resolution counterparts; and (b) knowledge-producing action $test(robot, fluent)$ that checks the value of a fluent in a given state. Finally, Σ_f includes *knowledge fluents* to describe observations of the environment and the axioms governing them, e.g., basic fluents to describe the direct (sensor-based) observation of the values of the fine-resolution fluents, and defined domain-dependent fluents that determine when the value of a particular fluent can be tested. The $test$ actions only change the values of knowledge fluents.

The axioms of $\mathcal{D}_f$ include (a) coarse-resolution and fine-resolution counterparts of all state constraints of $\mathcal{D}_c$, and fine-resolution counterparts of all other axioms of $\mathcal{D}_c$, with variables ranging over appropriate sorts from Σ_f; (b) general and domain-specific axioms for observing the domain through sensor inputs; and (c) axioms relating coarse-resolution domain attributes with their fine-resolution counterparts. For example:

$$\begin{aligned}
&test(rob_1, F)\ \textbf{causes}\ dir_obs(rob_1, F)\ \textbf{if}\ F = true\\
&\textbf{impossible}\ test(rob_1, F)\ \textbf{if}\ \neg can_test(rob_1, F)\\
&in_hand^*(rob_1, O)\ \textbf{if}\ component(O_base, O),\\
&\qquad\qquad in_hand(rob_1, O_base)
\end{aligned}$$

If certain conditions are met, e.g., each coarse-resolution domain attribute can be defined in terms of the fine-resolution attributes of the corresponding components, there is a path in τ_f for each transition in τ_c—see (Sridharan et al. 2017) for formal definitions and proofs.

Reasoning at fine resolution using $\mathcal{D}_f$ does not address the uncertainty in sensing and actuation, and becomes computationally intractable for complex domains. We address this problem by drawing on the principle of *zooming* introduced in (Sridharan et al. 2017). Specifically, for each abstract transition T to be implemented (i.e., executed) at fine resolution, we automatically determine the system description $\mathcal{D}_f(T)$ relevant to this transition; we do so by determining the relevant object constants and restricting $\mathcal{D}_f$ to these object constants. To implement T, we then use ASP-based reasoning with $\Pi(\mathcal{D}_f(T), \mathcal{H}_f)$ to plan a sequence of *concrete* (i.e., fine-resolution) actions. In what follows, we use "refinement and zooming" to refer to the use of both refinement and zooming as described above. Note that fine-resolution reasoning does not (need to) reason with activities or intentional actions.

The actual execution of the plan of concrete action is based on existing implementations of algorithms for common robotics tasks such as motion planning, object recognition, grasping and localization. These algorithms use probabilistic models of uncertainty in sensing and actuation. The high-probability outcomes of each action's execution are elevated to statements associated with complete certainty in $\mathcal{H}_f$ and used for subsequent reasoning. The outcomes from fine-resolution execution of each abstract transition, along with relevant observations, are added to $\mathcal{H}_c$ for subsequent reasoning using $\mathcal{ATI}$. The CR-Prolog programs for fine-resolution reasoning and the program for the overall control loop of the architecture are available online (Software 2018).

Key differences between the current representation and use of fine-resolution information, and the prior work on the refinement-based architecture (Sridharan et al. 2017) are:

- Prior work used a partially observable Markov decision process (POMDP) to reason probabilistically over the zoomed fine-resolution system description $\mathcal{D}_f(T)$ for any coarse-resolution transition T; this can be computationally expensive, especially when domain changes prevent reuse of POMDP policies (Sridharan et al. 2017). In this paper, CR-Prolog is used to compute a plan of concrete actions from $\mathcal{D}_f(T)$; each concrete action is executed using algorithms that incorporate probabilistic models of uncertainty, significantly reducing the computational costs of fine-resolution planning and execution. The disadvantage is that the uncertainty associated with each algorithm is not considered explicitly during planning at the fine-resolution.
- Prior work did not (a) reason about intentional actions; (b) maintain any fine-resolution history; or (c) extract and exploit all the information from fine-resolution observations. The architecture described in this paper keeps track of the relevant fine-resolution observations and adds appropriate statements to the coarse-resolution history to use all the relevant information. It also explicitly builds a consistent model of history at the finer resolution.

4 Experimental Setup and Results

This section reports the results of experimentally evaluating the capabilities of our architecture in different scenarios. We evaluated the following hypotheses:

- **H1:** using $\mathcal{ATI}$ improves the computational efficiency in comparison with not using it, especially in scenarios with

unexpected success.

- **H2:** using $\mathcal{ATI}$ improves the accuracy in comparison with not using it, especially in scenarios with unexpected goal-relevant observations.
- **H3:** the architecture that combines $\mathcal{ATI}$ with refinement and zooming supports reliable and efficient operation in complex robot domains.

We report results of evaluating these hypotheses experimentally: (a) in a simulated domain based on Example 1; (b) on a Baxter robot manipulating objects on a tabletop; and (c) on a Turtlebot finding and moving objects in an indoor domain. We also provide some execution traces as illustrative examples of the working of the architecture. In each trial, the robot's goal was to find and move one or more objects to particular locations. As a baseline for comparison, we used an ASP-based reasoner that does not include $\mathcal{ATI}$—we refer to this as the "traditional planning" ($\mathcal{TP}$) approach in which only the outcome of the action currently being executed is monitored. Note that this baseline still uses refinement and zoom, and probabilistic models of the uncertainty in sensing and actuation. Also, we do not use $\mathcal{TI}$ as the baseline because it includes components that make it much more computationally expensive than $\mathcal{ATI}$—see Section 3.2 for more details. To evaluate the hypotheses, we used one or more of the following performance measures: (i) total planning and execution time; (ii) number of plans computed; (iii) planning time; (iv) execution time; (v) number of actions executed; and (vi) accuracy.

4.1 Experimental Results (Simulation)

We first evaluated hypotheses H1 and H2 extensively in a simulated world that mimics Example 1, with four places and different objects. Please also note the following:

- To fully explore the effects of $\mathcal{ATI}$, the simulation-based trials did not include refinement, i.e., the robot only reasons with the coarse-resolution domain representation. We also temporarily abstracted away uncertainty in perception and actuation.
- We conducted paired trials and compared the results obtained with $\mathcal{TP}$ and $\mathcal{ATI}$ for the same initial conditions and for the same dynamic domain changes (when appropriate), e.g., a book is moved unknown to the robot and the robot obtains an unexpected observation.
- To measure execution time, we assumed a fixed execution time for each concrete action, e.g., 15 units for moving from a room to the neighboring room, 5 units to pick up an object or put it down; and 5 units to open a door. Ground truth is provided by a component that reasons with complete domain knowledge.

Table 1 summarizes the results of $\approx$ 800 paired trials in each scenario described in Section 3.2; all claims made below were tested for statistical significance. The initial conditions, e.g., starting location of the robot and objects' locations, and the goal were set randomly in each paired trial; the simulation ensures that the goal is reachable from the chosen initial conditions. Also, in suitable scenarios, a randomly-chosen, valid (unexpected) domain change is introduced in each paired trial. Given the differences between paired trials, it does not make sense to average the measured time or plan length across different trials. In each paired trial, the value of each performance measure (except accuracy) obtained with $\mathcal{TP}$ is thus expressed as a fraction of the value of the same performance measure obtained with $\mathcal{ATI}$; each value reported in Table 1 is the average of these computed ratios. We highlight some key results below.

Scenario-1 represents a standard planning task with no unexpected domain changes. Both $\mathcal{TP}$ and $\mathcal{ATI}$ provide the same accuracy (100%) and compute essentially the same plan, but computing plans comprising intentional actions takes longer. This explains the reported average values of 0.45 and 0.81 for planning time and total time (for $\mathcal{TP}$) in Table 1.

In Scenario-2 (unexpected success), both $\mathcal{TP}$ and $\mathcal{ATI}$ achieve 100% accuracy. Here, $\mathcal{ATI}$ stops reasoning and execution once it realizes the desired goal has been achieved unexpectedly. However, $\mathcal{TP}$ does not realize this because it does not consider observations not directly related to the action being executed; it keeps trying to find the objects of interest in different places. This explains why $\mathcal{TP}$ has a higher planning time and execution time, computes many more plans, and executes more plan steps than $\mathcal{ATI}$.

Scenarios 3–5 correspond to different kinds of unexpected failures. In all trials corresponding to these scenarios, $\mathcal{ATI}$ leads to successful achievement of the goal, but there are many instances in which $\mathcal{TP}$ is unable to recover from the unexpected observations and achieve the goal. For instance, if the goal is to move two books to the library, and one of the books is moved to an unexpected location when it is no longer part of an action in the robot's plan, the robot may not reason about this unexpected occurrence and thus not achieve the goal. This phenomenon is especially pronounced in Scenario-5 that represents an extreme case in which the robot using $\mathcal{TP}$ is never able to achieve the assigned goal because it never realizes that it has failed to achieve the goal. Notice that in the trials corresponding to all three scenarios, $\mathcal{ATI}$ takes more time than $\mathcal{TP}$ to plan and execute the plans for any given goal, but this increase in time is more than justified given the high accuracy and the desired behavior that the robot is able to achieve in these scenarios using $\mathcal{ATI}$.

The row labeled "All" in Table 1 shows the average of the results obtained in the different scenarios. The following three rows in Table 1 summarize results after removing from consideration all trials in which $\mathcal{TP}$ fails to achieve the assigned goal. We then notice that $\mathcal{ATI}$ is at least as fast as $\mathcal{TP}$ and is often faster, i.e., takes less time (overall) to plan and execute actions to achieve the desired goal. In summary, $\mathcal{TP}$ results in faster planning but results in lower accuracy and higher execution time than $\mathcal{ATI}$ in dynamic domains, especially in the presence of unexpected successes and failures that are common in dynamic domains. All these results provide evidence in support of hypotheses H1 and H2.

4.2 Execution traces

The following execution traces illustrate the differences in the decisions made by a robot using $\mathcal{ATI}$ in comparison

Scenarios	Average Ratios					Accuracy	
	Total Time	Number Plans	Planning Time	Exec. Time	Exec.Steps	$\mathcal{TP}$	$\mathcal{ATI}$
1	0.81	1.00	0.45	1.00	1.00	100%	100%
2	3.06	2.63	1.08	5.10	3.61	100%	100%
3	0.81	0.92	0.34	1.07	1.12	72%	100%
4	1.00	1.09	0.40	1.32	1.26	73%	100%
5	0.18	0.35	0.09	0.21	0.28	0%	100%
All	1.00	1.08	0.41	1.39	1.30	74%	100%
3 - no failures	1.00	1.11	0.42	1.32	1.39	100%	100%
4 - no failures	1.22	1.31	0.49	1.61	1.53	100%	100%
All - no failures	1.23	1.30	0.5	1.72	1.60	100%	100%

Table 1: Experimental results comparing $\mathcal{ATI}$ with $\mathcal{TP}$ in different scenarios. Values of all performance measures (except accuracy) for $\mathcal{TP}$ are expressed as a fraction of the values of the same measures for $\mathcal{ATI}$. $\mathcal{ATI}$ improves accuracy and computational efficiency, especially in dynamic domains.

with a robot using $\mathcal{TP}$. These traces correspond to scenarios in which the robot has to respond to the observed effects of an exogenous action.

Execution Example 1 *[Example of Scenario-2]*
Assume that robot $\mathtt{rob}_1$ is in the kitchen initially, holding $\mathtt{book}_1$ in its hand, and believes that $\mathtt{book}_2$ is in $\mathtt{office}_2$ and the library is unlocked.

- The goal is to have $\mathtt{book}_1$ and $\mathtt{book}_2$ in the library. The computed plan is the same for $\mathcal{ATI}$ and $\mathcal{TP}$, and consists of actions:

  $\mathtt{move}(\mathtt{rob}_1, \mathtt{library})$, $\mathtt{put\_down}(\mathtt{rob}_1, \mathtt{book}_1)$,
  $\mathtt{move}(\mathtt{rob}_1, \mathtt{kitchen})$, $\mathtt{move}(\mathtt{rob}_1, \mathtt{office}_2)$,
  $\mathtt{pickup}(\mathtt{rob}_1, \mathtt{book}_2)$, $\mathtt{move}(\mathtt{rob}_1, \mathtt{kitchen})$
  $\mathtt{move}(\mathtt{rob}_1, \mathtt{library})$, $\mathtt{putdown}(\mathtt{rob}_1, \mathtt{book}_2)$

- Assume that as the robot is putting $\mathtt{book}_1$ down in the library, someone has moved $\mathtt{book}_2$ to the library.
- With $\mathcal{ATI}$, the robot observes $\mathtt{book}_2$ in the library, reasons and explains the observation as the result of an exogenous action, realizes the goal has been achieved and stops further planning and execution.
- With $\mathcal{TP}$, the robot does not observe or does not use the information encoded in the observation of $\mathtt{book}_2$. It will thus waste time executing subsequent steps of the plan until it is unable to find or pickup $\mathtt{book}_2$ in the library. It will then replan (potentially including prior observation of $\mathtt{book}_2$) and eventually achieve the desired goal. It may also compute and pursue plans assuming $\mathtt{book}_2$ is in different places, and take more time to achieve the goal.

Execution Example 2 *[Example of Scenario-5]*
Assume that robot $\mathtt{rob}_1$ is in the kitchen initially, holding $\mathtt{book}_1$ in its hand, and believes that $\mathtt{book}_2$ is in kitchen and the library is unlocked.

- The goal is to have $\mathtt{book}_1$ and $\mathtt{book}_2$ in the library. The computed plan is the same for $\mathcal{ATI}$ and $\mathcal{TP}$, and consists of the actions:

  $\mathtt{move}(\mathtt{rob}_1, \mathtt{library})$, $\mathtt{put\_down}(\mathtt{rob}_1, \mathtt{book}_1)$,
  $\mathtt{move}(\mathtt{rob}_1, \mathtt{kitchen}), \mathtt{pickup}(\mathtt{rob}_1, \mathtt{book}_2)$,
  $\mathtt{move}(\mathtt{rob}_1, \mathtt{library})$, $\mathtt{putdown}(\mathtt{rob}_1, \mathtt{book}_2)$

- Assume the robot is in the act of putting $\mathtt{book}_2$ in the library, after having already put down $\mathtt{book}_1$ in the library earlier. However, someone has moved $\mathtt{book}_1$ to the kitchen while the robot was moving $\mathtt{book}_2$.
- With $\mathcal{ATI}$, the robot observes $\mathtt{book}_1$ in not in the library, realizes the goal has not been achieved, and continues to replan until it finds $\mathtt{book}_1$ and moves it to the library.
- With $\mathcal{TP}$, the robot puts $\mathtt{book}_2$ in the library and stops execution because it believes it has achieved the desired goal. In other words, it does not even realize that the goal has not been achieved.

4.3 Robot Experiments

We also ran experimental trials with the combined architecture, i.e., $\mathcal{ATI}$ with refinement and zoom, on two different robot platforms. These trials represented instances of the different scenarios (in Section 3.2) in domains that are variants of the domain in Example 1.

First, consider the experiments with the Baxter robot manipulating objects on a tabletop as shown in Figure 1a. Some other details of the domain include:

- The goal is to move particular objects between different "zones" (instead of places) or particular cell locations on a tabletop.
- After refinement, each zone is magnified to obtain grid cells. Also, each object is magnified into parts such as base and handle after refinement.
- Objects are characterized by color and size.
- The robot cannot move its body but it can use its arm to move objects between cells or zones.

Next, consider the experiments with the Turtlebot robot operating in an indoor domain as shown in Figure 1b. Some other details of the domain include:

- The goal is to find and move particular objects between places in an indoor domain.
- The robot does not have a manipulator arm; it solicits help from a human to pickup the desired object when it has reached the desired source location and found the object, and to put the object down when it has reached the desired target location.

- Objects are characterized by color and type.
- After refinement, each place or zone was magnified to obtain grid cells. Also, each object is magnified into parts such as base and handle after refinement.

Although the two domains differ significantly, e.g., in the domain attributes, actions and complexity, no change is required in the architecture or the underlying methodology. Other than providing the domain-specific information, no human supervision is necessary; most of the other steps are automated. In ≈ 50 experimental trials in each domain, the robot using the combined architecture is able to successfully achieve the assigned goal. The performance is similar to that observed in the simulation trials. For instance, if we do not include $\mathcal{ATI}$, the robot has lower accuracy or takes more time to achieve the goal in the presence of unexpected success or failure; in other scenarios, the performance with $\mathcal{ATI}$ and $\mathcal{TP}$ is comparable. Also, if we do not include zooming, the robot takes a significantly longer to plan and execute concrete, i.e., fine-resolution actions. In fact, as the domain becomes more complex, i.e., there are many objects and achieving the desired goal requires plans with multiple steps, there are instances when the planning starts becoming computationally intractable. All these results provide evidence in support of hypothesis H3.

Videos of the trials on the Baxter robot and Turtlebot corresponding to different scenarios can be viewed online[2]. For instance, in one trial involving the Turtlebot, the goal is to have both a cup and a bottle in the library, and these objects and the robot are initially in office_2. The computed plan has the robot pick up the bottle, move to the kitchen, move to the library, put the bottle down, move back to the kitchen and then to office_2, pick up the cup, move to the library through the kitchen, and put the cup down. When the Turtlebot is moving to the library holding the bottle, someone moves the cup to the library. With $\mathcal{ATI}$, the robot uses the observation of the cup, once it has put the bottle in the library, to infer the goal has been achieved and thus stops planning and execution. With just $\mathcal{TP}$, the robot continued with its initial plan and realized that there was a problem (unexpected position of the cup) only when it went back to office_2 and did not find the cup.

Similarly, in one trial with the Baxter, the goal is to have blue and green blocks in zone Y (right side of the screen) and these blocks are initially in zone R (left side of the screen). The computed plan has the Baxter move its arm to zone R, pick up a block, move to zone G (center) then to zone Y to put the block down, and repeat this process until it has moved all blocks. When the Baxter has moved one block and is moving back to pick up the second block from zone R, an exogenous action puts the first block in zone G (center). With $\mathcal{ATI}$, as the Baxter is moving over zone G on the way to zone R, it observes the block (it had previously put in zone Y), performs diagnostics and realizes his current activity will not achieve the goal. It then re-plans and manages to move all blocks to zone Y. With $\mathcal{TP}$, the robot is not able to use the observation of the first block in zone G, continues with the initial plan and never realizes that the goal has not been achieved.

[2] https://drive.google.com/drive/u/1/folders/1cWXVib82K7qVSIP5i_cT7HEBfE5cGB4G

5 Discussion and Future Work

In this paper we presented a general architecture that reasons with intentions and beliefs using transition diagrams at two different resolutions. Non-monotonic logical reasoning with a coarse-resolution domain representation containing commonsense knowledge is used to provide a plan of abstract intentional actions for any given goal. Each such abstract intentional action is implemented as a sequence of concrete actions by reasoning with the relevant part of a fine-resolution representation that is a refinement of the coarse-resolution representation. Also, the architecture allows the robot to automatically and elegantly consider the observations that are relevant to any given goal and the underlying intention. Experimental results in simulation and on different robot platforms indicate that this architecture improves the accuracy and computational efficiency of decision making in comparison with an architecture that does not reason with intentional actions and/or does not include refinement and zooming.

This architecture opens up directions for future research. First, we will explore and formally establish the relationship between the different transition diagrams in this architecture, along the lines of the analysis provided in (Sridharan et al. 2017). This will enable us to prove correctness and provide other guarantees about the robot's performance. We will also instantiate the architecture in different domains and to further demonstrate the applicability of the architecture. The long-term goal will be enable robots to represent and reason reliably and efficiently with different descriptions of knowledge and uncertainty.

Acknowledgments

The authors thank Michael Gelfond for discussions related to the $\mathcal{TI}$ architecture (Blount, Gelfond, and Balduccini 2015). The authors also thank Evgenii Balai for providing support with the SPARC software. This work was supported in part by the US Office of Naval Research Science of Autonomy award N00014-17-1-2434, and the Asian Office of Aerospace Research and Development award FA2386-16-1-4071. All opinions and conclusions described in this paper are those of the authors.

References

Balai, E.; Gelfond, M.; and Zhang, Y. 2013. Towards Answer Set Programming with Sorts. In *International Conference on Logic Programming and Nonmonotonic Reasoning*.

Balduccini, M., and Gelfond, M. 2003. Logic Programs with Consistency-Restoring Rules. In *AAAI Spring Symposium on Logical Formalization of Commonsense Reasoning*, 9–18.

Baral, C., and Gelfond, M. 2005. Reasoning about intended actions. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20, 689.

Blount, J.; Gelfond, M.; and Balduccini, M. 2014. Towards a theory of intentional agents. In *Knowledge Representation and Reasoning in Robotics. AAAI Spring Symp. Series*, 10–17.

Blount, J.; Gelfond, M.; and Balduccini, M. 2015. A theory of intentions for intelligent agents. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, 134–142. Springer.

Bratman, M. 1987. *Intention, Plans, and Practical Reason*. Center for the Study of Language and Information.

Erdem, E., and Patoglu, V. 2012. Applications of action languages in cognitive robotics. In *Correct Reasoning*. Springer. 229–246.

Gabaldon, A. 2009. Activity Recognition with Intended Actions. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Gelfond, M., and Inclezan, D. 2013. Some Properties of System Descriptions of $\mathrm{AL_d}$. *Journal of Applied Non-Classical Logics, Special Issue on Equilibrium Logic and Answer Set Programming* 23(1-2):105–120.

Gelfond, M., and Kahl, Y. 2014. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach*. Cambridge University Press.

Hanheide, M.; Gobelbecker, M.; Horn, G.; Pronobis, A.; Sjoo, K.; Jensfelt, P.; Gretton, C.; Dearden, R.; Janicek, M.; Zender, H.; Kruijff, G.-J.; Hawes, N.; and Wyatt, J. 2017. Robot Task Planning and Explanation in Open and Uncertain Worlds. *Artificial Intelligence* 247:119–150.

Kelley, R.; Tavakkoli, A.; King, C.; Nicolescu, M.; Nicolescu, M.; and Bebis, G. 2008. Understanding Human Intentions via Hidden Markov Models in Autonomous Mobile Robots. In *International Conference on Human-Robot Interaction (HRI)*.

Rao, A. S., and Georgeff, M. P. 1995. BDI Agents: From Theory to Practice. In *First International Conference on Multiagent Systems*, 312–319.

Saribatur, Z. G., and Eiter, T. 2016. Reactive policies with planning for action languages. In Michael, L., and Kakas, A., eds., *Logics in Artificial Intelligence*, 463–480. Springer International Publishing.

Saribatur, Z. G.; Baral, C.; and Eiter, T. 2017. Reactive maintenance policies over equalized states in dynamic environments. In Oliveira, E.; Gama, J.; Vale, Z.; and Lopes Cardoso, H., eds., *Progress in Artificial Intelligence*, 709–723. Cham: Springer International Publishing.

2018. Software and Results for Architecture combining Theory of Intentions and Refinement. Retrieved March 2018 from `https://github.com/hril230/theoryofintentions/tree/master/simmulation`.

Sridharan, M., and Gelfond, M. 2016. Using knowledge representation and reasoning tools in the design of robots. In *IJCAI*.

Sridharan, M.; Gelfond, M.; Zhang, S.; and Wyatt, J. L. 2017. A refinement-based architecture for knowledge representation and reasoning in robotics. *CoRR* abs/1508.03891.

Zhang, Q., and Inclezan, D. 2017. An application of asp theories of intentions to understanding restaurant scenarios. *International Workshop on Practical Aspects of Answer Set Programming*.

Zhang, S.; Sridharan, M.; and Wyatt, J. 2015. Mixed Logical Inference and Probabilistic Planning for Robots in Unreliable Worlds. *IEEE Transactions on Robotics* 31(3):699–713.

Sound and Complete Reactive UAV Behavior using Constraint Programming

Hoang Tung Dinh, Mario Henrique Cruz Torres, Tom Holvoet
imec-DistriNet, KU Leuven, 3001 Leuven, Belgium
{hoangtung.dinh, mariohenrique.cruztorres, tom.holvoet}@cs.kuleuven.be

Abstract

Realizing autonomous behavior of UAVs is a complex endeavor. The behavior needs to be goal-directed (1), while adhering to safety constraints (2), and while dealing with contingency situations (3). A purely imperative approach is unsuitable for this purpose, since such approach is unable to manage the sheer complexity of the behavior, leading to erroneous reactions to unforeseen situations in the environment, or situations that are simply not covered. In this paper, we present an innovative declarative approach to specify the autonomous behavior of UAVs. The approach combines a planning-based approach with constraint-programming for specifying safety and behavior constraints. The formal specification can be verified to be realizable. From the specification, we generate an execution policy, which can be proven to be sound and complete by construction if the specification is realizable. If the specification is not realizable, e.g., due to conflicting constraints, the generator indicates the source of the problem.

We illustrate our approach on a case study that uses a UAV for air quality monitoring and show how the approach assists the developer to specify correct autonomous behavior. The resulting behavior is deployed on an on-board computer of a UAV and tested on the field.

1 Introduction

Recent advances in both hardware and software for UAVs open the door to several applications. As UAV missions become more and more complicated, the demand for autonomous behavior is increasing. However, defining the autonomous behavior for UAVs is complex. First, this behavior needs to be goal-directed, i.e., given a set of actions that a UAV can perform (such as take-off, navigate to a waypoint, hover, capture images, etc.), the system must come up with a sound plan to achieve the mission goals, taking into account the situation in the environment. Second, safety requirements must be adhered to. Examples are collision avoidance, respecting no-fly zones, only fly if the battery level is sufficient, only land in safe spots, etc. Complex as the combination of both already is, the environment in which a UAV operates is typically dynamic and often unpredictable. The UAV must account for contingencies and adapt its behavior accordingly, always adhering to the stipulated safety constraints. This reaction must be fast and correct in terms of mission goals and safety requirements.

To accomplish this, developers must be able to specify and verify the behavior of an autonomous UAV. At run-time, the autonomous behavior must be sound and complete. The behavior is sound if the UAV behaves correctly according to the given specification. The behavior is complete if there is no situation where there are no actions the UAV can take to satisfy the specification. If there exists no sound and complete behavior that can satisfy a specification, we call the specification *unrealizable*.

Defining the autonomous behavior of UAVs imperatively, by specifying *how* the UAV should behave (e.g., by constructing Finite State Machines (Bohren et al. 2011; Nguyen et al. 2013)) is hard, not manageable or maintainable, and not verifiable, due to the explosion of possible states to be taken into account. In this paper we propose an approach to specify and generate the autonomous behavior of UAVs. In this approach, the actions, goals and constraints are specified in a declarative way, i.e., by defining rules about *what* the expected behavior of the UAV should accomplish. Such specification represents the minimal requirements. The actual behavior, i.e., the actions that are selected in a particular situation, is generated based on the specification.

In concrete, the behavior of a UAV is specified by a set of logical rules. The rules take into account mission goals, the actions a UAV is able to perform and the safety requirements that the UAV must conform to. This specification combines concepts from a classical planning approach - such as states, actions, goals - with constraint programming - imposing additional limitations on acceptable behavior. The behavior specification is then automatically translated to a set of constraint satisfaction problems (CSPs) (Tsang 1993) which are solved off-line. The solutions of the CSPs form an *execution policy* that maps each possible state of the world, that concerns the behavior specification, to a set of actions to be executed, which represents the *reactive* behavior of the UAV. The generated policy is sound because it satisfies the behavior specification, i.e., all the specified rules, by construction. On the one hand, if the specification is realizable, all the CSPs are feasible and the achieved policy is complete. On the other hand, if the specification is unrealizable, some CSPs are infeasible which indicates the situations leading to the unrealizability of the behavior specification.

The generated policy is used at runtime to achieve the autonomous behavior of the UAV. Whenever the state of the

world changes, the generated policy decides upon which actions to execute.

This paper brings two contributions to the autonomous vehicle research:

- We present a new formal way to specify the behavior of a UAV and its mission
- We present a technique to generate sound and complete execution policies from our behavior specifications.

This paper is organized as follows. Section 2 discusses related work. Section 3 presents our approach to formally specify the behavior of a UAV. Section 4 details our approach to generate an execution policy from the behavior specification. Section 5 describes a case study where the proposed behavior specification and generation approach is applied to an air quality monitoring mission. Finally, Section 6 draws conclusions and details possible future work.

2 Related work

Traditionally, robot behaviors are constructed manually by specifying *how* a robot should behave. Finite State Machines (FSMs) (Bohren et al. 2011; Nguyen et al. 2013) are the most commonly used model to represent robot behaviors. A well-known limitation of FSMs is that the number of states and transitions of a FSM exponentially grows as the complexity of the behavior arises. Recently, Behavior Trees (BTs) (Marzinotto et al. 2014; Colledanchise and Ögren 2016) received attention in the robotics community as an alternative to FSMs thanks to their modularity. However, different from FSMs, Behavior Trees are not event-driven, which may not be suitable for safety critical behaviors.

Hand-coding robot behaviors is hard and one has to rely on simulation to check if the constructed behavior meets the specified requirements. To address this problem, approaches to automatically generate robot behaviors from formal mathematical-based representations of requirements have been proposed.

Doherty et al. (Doherty, Heintz, and Kvarnström 2013) use Temporal Action Logic (TAL) (Doherty and Kvarnström 2008) as a specification language. Several planners have been developed to generate plans from specifications written in TAL (Doherty and Kvarnstram 2001; Kvarnström 2011). However, those planners assume that the environment is fully controllable and do not take into account contingencies. At runtime, if failures are detected during the plan execution, replanning is triggered (Doherty et al. 2014). This approach is similar to the use of the Planning Domain Definition Language (PDDL) (Gerevini et al. 2009) in the planning community, where generic planners are developed to solve planning problems described in PDDL. Since these planning approaches rely on replanning at runtime to deal with contingencies, they do not guarantee the completeness of the behavior. The replanning may fail due to an unrealizable specification and this failure can only be detected at runtime. Besides that, replanning can be computationally expensive and take too long to execute at runtime.

Linear Temporal Logic (LTL) (Emerson and others 1990) is another specification language often used in robotics.

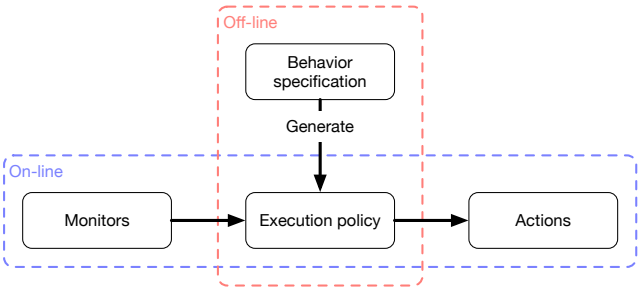


Figure 1: From a behavior specification, an execution policy is automatically generated off-line. At runtime, the execution policy maps the state vector value given by the monitors to a set of actions to execute.

There exists synthesis techniques to generate FSMs (Wongpiromsarn, Topcu, and Murray 2013; Maniatopoulos et al. 2016) or BTs (Colledanchise, Murray, and Ogren 2017) from specifications written in some fragments of LTL. Using these synthesis approaches, the behavior is guaranteed to be sound and complete since all contingencies are taken into account in the generated behavior. The main limitation of the LTL approaches is their computational complexity.

3 Behavior specification

In this section we present a new approach for specifying the behavior of an autonomous UAV in a mission. In our approach, a *behavior specification* is a formal representation of all the concerns relevant to the UAV's behavior in the mission. The specification aggregates information about how the UAV perceives the world, the actions it can perform, the safety aspects of a mission and its goals. Fig. 1 shows how the behavior specification fits in the whole system, being used off-line to generate an *execution policy* that maps each possible state of the world to a set of actions to execute. The execution policy is used at runtime by the UAV, so that it knows which actions have to be executed in the occurrence of any changes in the environment.

A behavior specification consists of four elements: (1) *state vector*, (2) *actions*, (3) *reaction rules* and (4) *goals*.

The *state vector* abstracts the world representation. It contains state variables about the environment and the system itself, which one is interested in.

An *action* represents a primitive behavior that the UAV can perform, e.g., take-off. An action is durative, i.e., it takes time. The specification of an action includes the preconditions for the activation and execution of the action, the desired effects the action will have and the controlled resources that the action requires exclusive control.

The *reaction rules* specify the reaction that the UAV has to guarantee during a mission.

The final element, *goals* represent the desired goals of the UAV's mission.

Each one of the above-mentioned elements is further explained in the following sub-sections.

State vector

A state vector $\mathbf{S}$ represents which states of the world and the UAV-system that the UAV is concerned with. Formally, a state vector is a set of measurable *discrete state variables* $\{S_0, S_1, ..., S_n\}$. Each state variable S_i represents a state that the UAV can measure. At runtime, the value of a state variable is updated by a *monitor*, which measures a specific aspect of the world or of the system. A monitor translates a measurement to a particular discrete value in a state variable.

It is the responsibility of developers to decide which aspects of the world and the UAV-system should be represented in the state vector and how they are discretized and monitored. E.g., to represent whether the battery level is below 5%, one can create the state variable $S_{battery} = \{bellow\_5, between\_5\_100\}$ and a monitor that constantly measures the battery level and updates the state variable to the corresponding value at runtime. A state variable could also represent aspects related to history. E.g., one can define the state variable $S_{a\_executed} = \{true, false\}$ to represent whether action $\mathbf{a}$ has ever been executed and finished.

Actions

In our approach, the behavior of autonomous UAVs is about deciding which *actions* to execute based on runtime feedback information. Given a set of actions $\mathbf{A} = \{\mathbf{a_0}, \mathbf{a_1}, ..., \mathbf{a_n}\}$, the semantic meaning of each action $\mathbf{a}$ is defined by three properties: *preconditions*, *desired effects* and *controlled resources*. An action's preconditions and desired effects define when the action can be executed and what is the expected state after such action's execution. We also formally specify the controlled resources, so that it is possible to guarantee that there are no conflicts in the system regarding multiple actions trying to control the same resources simultaneously.

Preconditions: $\text{Pre}(\mathbf{a}, \mathbf{S})$ are predicates on the state vector $\mathbf{S}$ that must hold on the activation and during the execution of the action $\mathbf{a}$. An action can only be started if its preconditions hold. During the execution, if the preconditions of an action change from "hold" to "not hold", the action will be stopped.

Desired effects: $\text{Eff}(\mathbf{a}, \mathbf{S})$ are predicates on the state vector $\mathbf{S}$ that will hold if the action $\mathbf{a}$ is executed successfully. Currently, our approach only allows desired effects to be represented as equality and conjunctive predicates. An action can still be executed after its desired effects have been achieved, as long as its preconditions still hold. If the desired effects have been achieved and the action is still executing, the desired effects will be maintained.

Controlled resources: $\text{R}(\mathbf{a})$ is a set of resources over which the action $\mathbf{a}$ requires exclusive control during its execution. If two actions have at least one controlled resource in common, they cannot be executed in parallel. Examples of controlled resources are the rotors, or an air quality sensor. We currently assume that an action has at most one controlled resource. This assumption may be relaxed in future studies.

The preconditions of an action are a set of necessary requirements for the action to achieve its desired effects. If the action cannot achieve the desired effects, we call the action *infeasible*. In that case, the observable condition for an action to be infeasible is represented by a predicate $\text{Infeasible}(\mathbf{a}, \mathbf{S})$ and the predicate $\neg\text{Infeasible}(\mathbf{a}, \mathbf{S})$ is added to the preconditions of the action. By doing so, we guarantee that an infeasible action will not be activated or if the action is already executing, it will be stopped. E.g., action $\mathbf{navigate\_to\_A}$ is infeasible if there exists no path from the current position of the UAV to point A, i.e., $S_{path\_to\_A} = not\_exist$. Therefore, the predicate $\neg(S_{path\_to\_A} = not\_exist)$ is included in the preconditions of the action. Note that, it is up to the developer to decide how $S_{path\_to\_A}$ should be monitored (e.g., by executing a path planner at a fixed frequency).

Given the assumption above, a developer must model all possible infeasible conditions of actions in their preconditions. If a contingency event leading to the infeasibility of an action is not modeled, the behavior of the UAV is not guaranteed to be correct. However, it is possible to define a high-level infeasible condition that can capture different unknown contingency events. E.g., an action that navigates the UAV to a checkpoint can be considered as infeasible if the UAV cannot reach the checkpoint in 5 minutes after the activation of the action or when a path planner cannot find a feasible path from the current position of the UAV to the checkpoint.

Compositions of actions can be defined by logical rules to constrain or enforce the parallel execution of actions. Logical rules about action compositions involve the predicate $\text{Exec}(\mathbf{a})$ which represents whether an action is executing. E.g., the rule in Eq. (1) requires action $\mathbf{a_0}$ and $\mathbf{a_1}$ to always be executing in parallel. Note that, actions executed in parallel must have no conflict in their preconditions and desired effects as well as no controlled resource in common. If there is a conflict, the behavior specification is unrealizable and the conflict situation will be detected during the policy generation (Section 4).

$$\text{Exec}(\mathbf{a_0}) \Leftrightarrow \text{Exec}(\mathbf{a_1}) \tag{1}$$

Reaction rules

Safety requirements related to the reaction of a UAV are represented as logical rules on the state vector and the executed actions as in Eq. (2). The predicate $\text{Cond}(\mathbf{S})$ represents whether a logical condition holds on the state vector. The rule in Eq. (2) states that if a condition on the state vector holds, action $\mathbf{a}$ must be executing.

$$\text{Cond}(\mathbf{S}) \Rightarrow \text{Exec}(\mathbf{a}) \tag{2}$$

This rule allows developers to specify the UAV's reactions to contingency events in a declarative way. E.g., one can require the UAV to go home or land if its battery level is low. At runtime, the UAV is guaranteed to handle contingency events correctly according to the reaction rules. Conflicts among reaction rules will be reported during the policy generation so that the developer can fine-tune the specification.

Goals

We represent the goals of the UAV's mission as rules concerning the values on the state vector $\mathbf{S^G}$, i.e., the state vector value after the UAV completed its mission. There are two typical types of goal rules.

$$\text{Goal}(\mathbf{S^G}) \tag{3}$$

$$\text{Cond}(\mathbf{S}) \Rightarrow \text{Goal}(\mathbf{S^G}) \tag{4}$$

The first type of goal rules (Eq. (3)) states that a condition must hold at the end of the mission. E.g., one may want the UAV to always be landed at the end of the mission. The second type of goal rules (Eq. (4)) represents *conditional goals* which only need to be achieved if the goals are feasible. The condition for a goal to be feasible is represented by the predicate $\text{Cond}(\mathbf{S})$ in Eq. (4). E.g., one may want the UAV to visit a checkpoint only if there exists a feasible path from the current position of the UAV to the checkpoint. At runtime, the UAV continually assesses the feasibility of the conditional goals to decide whether to pursue them or not.

Given the state vector, actions, reaction rules and goal rules, we generate an execution policy $\pi(\mathbf{S}) = \{\mathbf{a_i}\}$ mapping each state vector value to a set of actions to execute (Section 4). At runtime, whenever there is a change to a state variable in the state vector, the UAV looks up at the generated policy to select the actions to execute (as shown in Fig. 1).

Example

We present a simple example of a behavior specification. Assuming that a UAV needs to fly to a checkpoint and land there. However, we only allow the UAV to fly if the battery level is above a predefined threshold. If the battery level of the UAV is below the threshold, the UAV must stop flying and land immediately. We define the state vector as follows.

$$\begin{aligned} &\mathbf{S} = \{S_{flying}, S_{dest}, S_{battery}\} \\ &S_{flying} = \{flying, landed\} \\ &S_{dest} = \{not\_reached, reached\} \\ &S_{battery} = \{below, above\} \end{aligned} \tag{5}$$

The following actions are defined for the mission.

- $\mathbf{TakeOff}$:
  - Preconditions: $S_{flying} = landed$
  - Desired effects: $S_{flying} = flying$
  - Controlled resource: *rotors*
- $\mathbf{NavigateToPoint}$:
  - Preconditions: $S_{flying} = flying$
  - Desired effects: $S_{dest} = reached$
  - Controlled resource: *rotors*
- $\mathbf{Land}$:
  - Preconditions: $S_{flying} = flying$
  - Desired effects: $S_{flying} = landed$
  - Controlled resource: *rotors*

The following reaction rule states that if the battery level of the UAV is below a threshold, the UAV must land.

$$(S_{flying} = flying) \wedge (S_{battery} = below) \Rightarrow \text{Exec}(\mathbf{Land}) \tag{6}$$

The goals are specified in Eq. (7) and (8). The rule in Eq. (7) states that if the battery level is above the threshold, the UAV must try to reach the destination. The rule in Eq. (8) requires the UAV to land at the end of the mission.

$$S_{battery} = above \Rightarrow S^G_{dest} = reached \tag{7}$$

$$S^G_{flying} = landed \tag{8}$$

Table 1 shows a sound and complete execution policy generated from the specification. The notation **no-op** indicates that no action needs to be executed.

Table 1: A sound and complete execution policy

S_{flying}	S_{dest}	$S_{battery}$	Actions
$landed$	$not\_reached$	$above$	$\mathbf{TakeOff}$
$landed$	$not\_reached$	$below$	$\mathbf{no\text{-}op}$
$flying$	$not\_reached$	$above$	$\mathbf{NavigateToPoint}$
$flying$	$not\_reached$	$below$	$\mathbf{Land}$
$flying$	$reached$	$above$	$\mathbf{Land}$
$flying$	$reached$	$below$	$\mathbf{Land}$
$landed$	$reached$	$above$	$\mathbf{no\text{-}op}$
$landed$	$reached$	$below$	$\mathbf{no\text{-}op}$

4 Execution policy generation using constraint programming

In this section we present how to generate an execution policy from a behavior specification described in the previous section. The process of generating an execution policy from a behavior specification is illustrated in Fig. 2. Recall that an execution policy is a function mapping each state vector value to a set of actions. We calculate the mapping result of a state vector value by solving a classical planning problem. The solution of the planning problem is a sequence of *execution steps* that transform the given state vector value to a state vector value that satisfies all the goal rules (see Section 3). Each execution step is a set of actions to be executed in parallel. The first execution step of the solution is used as the mapping result from the given state vector value. The complete execution policy is generated by calculating the mapping results of all possible state vector values.

We model each classical planning problem mentioned above as a constraint satisfaction problem (CSP). In (Barták and Toropila 2008), Bartak et al. summarize several constraint satisfaction models for sequential planning problems, i.e., problems where only one action is allowed to execute at the same time. In this paper, we extend the constraint model described in (Barták and Toropila 2008) to support multiple actions to be executed in parallel.

Given a state vector value $\mathbf{S^0}$, we find a plan of length n, corresponding to n execution steps, transforming $\mathbf{S^0}$ to the goal state vector $\mathbf{S^n}$ which satisfies all the goal rules. To find the plan with the shortest length, we start solving the

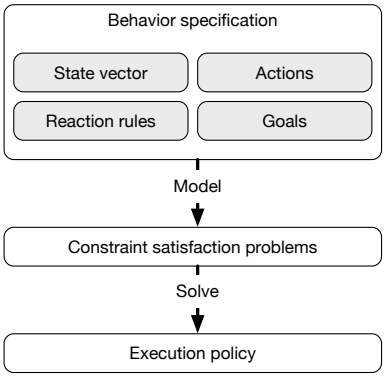


Figure 2: To generate an execution policy, a set of constraint satisfaction problems (CSPs) are constructed from the behavior specification. Each CSP models a mapping result of a state vector value. All the CSPs are solved off-line to achieve the complete execution policy.

CSP with $n = 1$ and then gradually increase n by 1, until a feasible plan is found or a maximal value of n is reached.

Fig. 3 shows the decision variables of a CSP. Since each action has either one or no controlled resource and actions with the same controlled resource cannot be executed together (see Section 3), maximally $k = k_0 + k_1$ actions can be executed in parallel , where k_0 is the total number of controlled resources and k_1 is the number of actions having no controlled resource. Therefore, we represent a plan by $k \times n$ action variables with k action variables at each execution step. Since the plan has length n, there are $n+1$ state vectors. Each state vector is represented by a set of state variables.

It is possible to define the domain $\mathbf{D}$ of each action variable as all the specified actions in $\mathbf{A}$ and add constraints stating that two actions having the same control resource cannot be in the same execution step. However, doing so leads to a big model which is computationally expensive to solve. We therefore reduce the computational complexity of the constraint model by narrowing down the domain of each action variable as follows.

For the first k_0 rows of action variables (different rows of action variables are represented by different colors in Fig. 3), the domain of action variables in each row consists of only actions having the same controlled resource. For the next k_1 rows of action variables, the domain of action variables in each row consists of a single action having no controlled resource. In other words, each row of action variables corresponds to one controlled resource or a single action with no controlled resource. Since it could be the case that not all execution steps have exactly k actions to execute in parallel, we define a *null* action with empty preconditions and desired effects to fill in the empty position at each execution step. The *null* action is added to the domain of each action variable. This way of modeling reduces the domains of the action variables while ensuring that actions controlling the same resource are not executed in parallel since they are represented by a single action variable at each execution step.

For each action variable A_r^i connecting the state vector

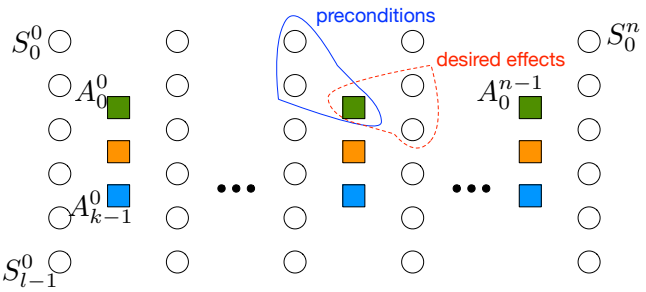


Figure 3: Decision variables in a planning problem modeled as a CSP. The preconditions and desired effects of an action (represented by a square) show the connection between two instances of the state vector (represented by circles). This figure is partly based on Figure 1 in (Barták and Toropila 2008).

$\mathbf{S^i}$ and $\mathbf{S^{i+1}}$, the following constraints are added to impose the preconditions and desired effects (Barták and Toropila 2008). The constraints are repeated for each action $\mathbf{a}$ in the domain of A_r^i.

$$\begin{aligned} A_r^i = \mathbf{a} &\Rightarrow \text{Pre}(\mathbf{a}, \mathbf{S^i}) \\ A_r^i = \mathbf{a} &\Rightarrow \text{Eff}(\mathbf{a}, \mathbf{S^{i+1}}) \end{aligned} \tag{9}$$

We assume that at each plan step, state variables which are not in the desired effects of A^i stay the same. This assumption, which is called the frame assumption (Barták and Toropila 2008), is necessary to formulate a classical planning problem. We deal with exogenous events and the undesired effects of actions by creating a plan for each possible value of the state vector. At runtime, every time the value of the state vector changes, the actions corresponding to the plan of the new state vector value will be executed.

Since in (Barták and Toropila 2008), at each execution step only one action is executed, the frame axioms can simply be defined as follows.

$$A_r^i = \mathbf{a} \Rightarrow S_j^i = S_j^{i+1}, \forall S_j \notin \text{Eff}(\mathbf{a}, \mathbf{S}) \tag{10}$$

However in our problem, k actions are executed in parallel at each step. Therefore, to represent the frame axioms, for each state vector S^i, we define k boolean vectors $\mathbf{B}^r, r \in \langle 0, k-1 \rangle$ having the same length v as the state vector. Each boolean vector $\mathbf{B}^r$ represents which state variables in the state vector $\mathbf{S^{i+1}}$ action A_r^i has effect on. Then, the frame axioms are defined as follows.

$$\begin{aligned} A_r^i = \mathbf{a} &\Rightarrow \mathbf{B}^r = \mathbf{B}(\mathbf{a}) \\ \left(\bigvee_r B_j^r\right) = false &\Rightarrow S_j^i = S_j^{i+1} \end{aligned} \tag{11}$$

where $\mathbf{B}(\mathbf{a})$ is a precomputed boolean vector. An element in $\mathbf{B}(\mathbf{a})$ is $true$ if $\mathbf{a}$ has effect and is $false$ if $\mathbf{a}$ does not have effect on the corresponding state variable. Eq. (11) states that a state variable must stay unchanged after an execution step if it does not belong to the desired effect of any action executed in that step.

Reaction rules (Section 3) are added to the state and action variables at each execution step. The predicate $\text{Exec}(\mathbf{a})$ is translated to the predicate $A_r^i = \mathbf{a}$ in the CSP, where r is the row where the action variable domain containing $\mathbf{a}$. E.g., the rule in Eq. (2) is added to the CSP as follows.

$$\bigwedge_i \big(\text{Cond}(S^i) \Rightarrow A_r^i = \mathbf{a}\big) \tag{12}$$

Rules representing goals (Section 3) are applied on the first state vector $\mathbf{S^0}$ and the final state vector $\mathbf{S^n}$. The rules in Eq. (3) and (4) are translated to the following constraints.

$$\text{Goal}(\mathbf{S^n}) \tag{13}$$

$$\text{Cond}(\mathbf{S^0}) \Rightarrow \text{Goal}(\mathbf{S^n}) \tag{14}$$

The resulting execution policy generated by solving all the CSPs is sound by construction, i.e., the policy is correct according to the behavior specification. It is because the policy satisfies all the constraints translated from the reaction rules and goal rules. The generated policy is complete if there exists a mapping result for each state vector value, i.e., all CSPs are feasible. The unrealizability of the specification can be detected when a CSP is infeasible. In that case, the corresponding state vector value will be reported so that one can refine the specification. Examples of an unrealizable behavior specification will be presented in Section 5.

At runtime, the following properties will be guaranteed by the generated policy. (1) An action is not executed if its preconditions do not hold. (2) The UAV never violates any reaction rules. (3) The UAV always actively tries its best, according to its belief, to pursue the goals.

Note that, the policy does not guarantee that the goals can always be reached. It is because the belief of the UAV contains the frame assumption. Depending on the degree of the assumption's violation in reality, the UAV may achieve the goals or not. Our future work will focus on revealing the exact situations where the UAV may fail to achieve the goals so that we can provide hard guarantees on goal achievement.

5 Case study - Air quality monitoring

We present a case study where our behavior specification and generation approach is applied to a mission taken from the SafeDroneWare[1] project. In the mission, the UAV has to monitor the air quality at three checkpoints A, B and C (can be in arbitrary order). The UAV takes off at $home$ location. To monitor the air quality at a checkpoint, the UAV needs to fly to the checkpoint and activate its air quality sensor. The UAV must hover while the air quality sensor is collecting data. If the UAV cannot reach a checkpoint (e.g., because the path planner cannot find a feasible path), it can ignore that checkpoint. After finishing taking samples at all possible checkpoints, the UAV should return to $home$. However, if the battery level of the UAV is below 20%, the UAV should abort the mission and return to $home$. In addition, if the battery level is below 5%, the UAV needs to land immediately, no matter whether it reached $home$ or not. The UAV also needs to support manual control mode if the boolean variable $manualcontrol$ is set to $true$.

In our notation in the following subsections, state variables and actions represented with a parameter, like $S_{sample}(x), x \in \{A, B, C\}$, represent multiple occurrences of such entities. E.g., the above state variable notation is a short version of three state variables S_{sample_A}, S_{sample_B} and S_{sample_C}.

State vector

We define the following state variables for the mission with their corresponding meanings.

$$\begin{aligned}
&\mathbf{S} = \{S_{flying}, S_{location}, S_{battery}, S_{sample}(x), S_{path}(y),\\
&\qquad S_{manualcontrol}\}, x \in \{A, B, C\}, y \in \{A, B, C, home\}\\
&S_{flying} = \{flying, landed\}\\
&S_{location} = \{home, A, B, C, other\}\\
&S_{battery} = \{above_20\%, from_5_to_20\%, below_5\%\}\\
&S_{sample}(x) = \{not_collected, collected\}, x \in \{A, B, C\}\\
&S_{path}(x) = \{not_exist, exist\}, x \in \{A, B, C, home\}\\
&S_{manualcontrol} = \{false, true\}
\end{aligned} \tag{15}$$

Actions

We define the following actions for the mission.

- **TakeOff**
  - Preconditions: $S_{flying} = landed$
  - Desired effects: $S_{flying} = flying$
  - Controlled resources: *rotors*
- **Land**
  - Preconditions: $S_{flying} = flying$
  - Desired effects: $S_{flying} = landed$
  - Controlled resources: *rotors*
- **Navigate**$(x), x \in \{home, A, B, C\}$
  - Preconditions: $S_{flying} = flying$, $S_{path}(x) = exist$
  - Desired effects: $S_{location} = x$
  - Controlled resources: *rotors*
- **Hover**$(x), x \in \{A, B, C\}$
  - Preconditions: $S_{flying} = flying$, $S_{location} = x$
  - Desired effects: $S_{flying} = flying$, $S_{location} = x$
  - Controlled resources: *rotors*
- **CollectSample**$(x), x \in \{A, B, C\}$
  - Preconditions: $S_{flying} = flying$, $S_{location} = x$, $S_{sample}(x) = not\_collected$
  - Desired effects: $S_{sample}(x) = collected$
  - Controlled resources: *air quality sensor*
- **ManuallyFly**
  - Preconditions: $S_{flying} = flying$, $S_{manualcontrol} = true$
  - Desired effects: $S_{manualcontrol} = false$

[1] https://www.imec-int.com/en/what-we-offer/research-portfolio/safedroneware

– Controlled resources: *rotors*

The following composition action rule is added to enforce the UAV to hover while collecting air samples.

$$\text{Exec}(\mathbf{CollectSample}(x)) \Rightarrow \text{Exec}(\mathbf{Hover}(x)) \quad (16)$$

Reaction rules and goals

We now translate the description of the mission to reaction rules and goals. Note that, the current specification is unrealizable. We will show how the unrealizability can be detected during the generation of the execution policy.

The first rule states that whenever the user wants to take control of the UAV and the UAV is flying, the action **ManuallyFly** must be executed.

$$(S_{manualcontrol} = true) \wedge (S_{flying} = flying) \Rightarrow \text{Exec}(\mathbf{ManuallyFly}) \quad (17)$$

The second rule states that if the battery level is between 5% and 20%, the UAV must navigate to $home$.

$$(S_{battery} = from\_5\%\_to\_20\%) \wedge (S\_flying = flying) \wedge \neg(S\_location = home) \Rightarrow \text{Exec}(\mathbf{Navigate}(home)) \quad (18)$$

The last rule requires the UAV to land if the battery level is below 5%.

$$(S_{battery} = below\_5\%) \wedge (S\_flying = flying) \Rightarrow \text{Exec}(\mathbf{Land}) \quad (19)$$

The goals of the mission are to collect air samples at A, B and C if there exists feasible paths to those checkpoints and then return to home and land. The goals are represented by the three following rules.

$$(S_{path}(x) = exist) \Rightarrow S^G_{sample}(x) = collected \quad (20)$$

$$S^G_{location} = home \quad (21)$$

$$S^G_{flying} = landed \quad (22)$$

Unrealizability detection

The behavior specification described in the previous section is translated to a set of CSPs. While solving the CSPs, the situation in Fig. 4 is reported to be infeasible.

We can see that the there is no solution for the situation where the battery level of the UAV is below 5% and the UAV has not collected all the air samples yet. In this case, the rule requiring the UAV to land conflicts with the rule requiring the UAV to collect the air samples. We also realize that the same will happen when the battery level of the UAV is between 5% and 20%. The question to the developer is: What should the UAV do in these situations? Since in this mission the safety rules are more important than the goal to collect air samples, we relax the rule in Eq. (20) by replacing it with the following rule.

$$(S_{path}(x) = exist) \wedge (S_{battery} = above\_20\%) \Rightarrow S^G_{sample}(x) = collected \quad (23)$$

However, after updating the rule, the above situation is still reported to be infeasible. It is because the rule of being home at the end of the mission conflicts with the rule stating that the UAV must land immediately if the battery level is below 5%. If the battery level is below 5% and the UAV is not at home yet, there is no solution that can satisfy both rules. In this case, we want the UAV to land instead of still trying to go home. We therefore replace the rule in Eq. (21) by the rule below. The new rule states that the UAV should only try to be at home if its battery level is more than or equal to 5%.

$$\neg(S^I_{battery} = below\_5\%) \Rightarrow S^G_{location} = home \quad (24)$$

The situation in Fig. 4 is now feasible. However, the specification is still not realizable yet. Another situation reported to be infeasible is shown in Fig. 5.

From this situation, we realize that we did not specify what the UAV should do if there exists no path to go home. The first situation where the UAV needs to go home is after it collected air samples at all checkpoints. If there is no path, we allow the UAV not to go home anymore, which means that the UAV can land at its current location. Therefore, we replace the rule in Eq. (24) by the rule below.

$$(S_{path}(home) = exist) \wedge \neg(S^I_{battery} = below\_5\%) \Rightarrow S^G_{location} = home \quad (25)$$

Note that, depending on the missions, one may define extra actions to deal with such situations. In this paper, for the ease of presentation, we simply allow the UAV to land at its current position.

The second situation where the UAV needs to go home is when its battery level is between 5% and 20%. We again allow the UAV to land at its current location if there exists no path to go home. Thus, we replace the rule in Eq. (18) by the following rules which explicitly expresses what the UAV

$S_{flying} = flying$	$S_{path}(A) = exist$
$S_{location} = other$	$S_{path}(B) = exist$
$S_{battery} = below\_5\%$	$S_{path}(C) = exist$
$S_{sample}(A) = collected$	$S_{path}(home) = exist$
$S_{sample}(B) = collected$	$S_{manualcontrol} = false$
$S_{sample}(C) = not\_collected$	

Figure 4: First infeasible situation.

$S_{flying} = flying$	$S_{path}(A) = exist$
$S_{location} = other$	$S_{path}(B) = exist$
$S_{battery} = from\_5\_to\_20\%$	$S_{path}(C) = exist$
$S_{sample}(A) = collected$	$S_{path}(home) = not\_exist$
$S_{sample}(B) = collected$	$S_{manualcontrol} = false$
$S_{sample}(C) = collected$	

Figure 5: Second infeasible situation.

$S_{flying} = flying$	$S_{path}(A) = exist$
$S_{location} = other$	$S_{path}(B) = exist$
$S_{battery} = from\_5\_to\_20\%$	$S_{path}(C) = exist$
$S_{sample}(A) = collected$	$S_{path}(home) = exist$
$S_{sample}(B) = collected$	$S_{manualcontrol} = true$
$S_{sample}(C) = collected$	

Figure 6: Third infeasible situation.

must do if there exists a path to go home and if there exists no path to go home.

$$(S_{battery} = from\_5\%\_to\_20\%) \wedge (S\_flying = flying) \wedge \neg(S\_location = home) \wedge (S_{path}(home) = exist) \Rightarrow \text{Exec}(\mathbf{Navigate}(home)) \quad (26)$$

$$(S_{battery} = from\_5\%\_to\_20\%) \wedge (S\_flying = flying) \wedge \neg(S\_location = home) \wedge (S_{path}(home) = not\_exist) \Rightarrow \text{Exec}(\mathbf{Land}) \quad (27)$$

The last conflict being detected is about the manual control mode. Thanks to the reported situation in Fig. 6, we realize that the rule about executing the $\mathbf{ManuallyFly}$ action in Eq. (17) conflicts with the rules requiring the UAV to return to home or land when the battery level is low. For example, if the UAV is requested to switch to the manual control mode while its battery level is below 5%, the UAV does not know what to do. Since in our scenario we prefer the UAV to land, we refine the rule about the manual control mode by only allowing the UAV to execute the $\mathbf{ManuallyFly}$ action if the battery level is above 20% as in Eq. (28).

$$(S_{manualcontrol} = true) \wedge (S_{flying} = flying) \wedge (S_{battery} = above\_20\%) \Rightarrow \text{Exec}(\mathbf{ManuallyFly}) \quad (28)$$

The examples above show the capability of our approach to detect the unrealizability of a behavior specification. This is especially useful since it is not trivial for humans to think about all situations while designing the UAV's behavior. Using our approach, one could gradually refine the behavior specification until it becomes realizable.

Policy generation

There are 7680 possible values for the state vector in the air quality monitoring mission, corresponding to 7680 CSPs. We used Choco (Prud'homme, Fages, and Lorca 2016), an open source constraint programming solver, to solve the CSPs. It took 94.7 seconds in total to solve all the CSPs and generate the execution policy on an Ubuntu computer with Intel i7- 7700K 4.20GHz processors.

Validation

The generated execution policy is validated on a DJI Matrice 100 UAV[2]. The execution policy is run on the DJI Manifold on-board computer[3] with a NVIDIA Tegra K1 processor. A video of the flight test is included in the submission[4]. Three scenarios are demonstrated in the flight test. The first scenario shows a normal flight where the UAV took off, visited three checkpoints and collected samples, then went home and landed. In the second scenario, while the UAV was navigating to the third checkpoint, a no-fly zone that covers the third checkpoint was added. The UAV then realized that there is no path to the last checkpoint and decided to go home. While the UAV was navigating to home, the human pilot requested for the manual control mode. The UAV therefore activated the $\mathbf{ManuallyFly}$ action. The human pilot kept controlling the UAV. When the battery level was below 20%, the UAV decided to terminate the $\mathbf{ManuallyFly}$ action, then went home and landed. The third scenario illustrates a situation where the battery level of the UAV decreases faster than normal. While the UAV was collecting samples at the second checkpoint, the battery level dropped below 20%. The UAV decided to stop collecting samples and go home. However, when the UAV was halfway to home, the battery level became less than 5%. Thus, it decided to land immediately.

[2]https://www.dji.com/matrice100

[3]https://store.dji.com/product/manifold

[4]https://youtu.be/S146DP1th0U

6 Conclusions

In this paper we present an approach to specify and generate reactive UAV behavior. The behavior is specified formally as a set of declarative logical rules. An execution policy is generated from the behavior specification by creating and solving a set of CSPs (Constraint Satisfaction Problems). Using constraint programming, reaction rules and goals can be easily taken into account by being translated to constraints of the CSPs. The CSPs can be solved efficiently using an existing open source solver. Any unrealizability in the behavior specification can be detected while solving the CSPs. This way, it is possible to guarantee that the generated policy is sound and complete by construction. During execution, our approach guarantees that the reactions rules are always satisfied and the UAV always actively pursues the goals.

While the potential of our behavior specification and policy generation approach has been demonstrated and validated on a real UAV, a number of problems remain open. While we guarantee that the UAV always actively pursues the goals, it is not clear to what extend the goals can be achieved in a non-deterministic environment. Future studies will concentrate on revealing the situations where the goals cannot be achieved and providing hard guarantees on goal achievement. Our future studies will also aim at developing software toolboxes and domain specific languages to ease the usage of our behavior specification and generation approach.

Acknowledgment

This research is partially funded by the Research Fund KU Leuven, and by the imec-ICON project SafeDroneWare.

References

Barták, R., and Toropila, D. 2008. Reformulating Constraint Models for Classical Planning. In *International Florida Artificial Intelligence Research Society Conference*, 525–530.

Bohren, J.; Rusu, R. B.; Jones, E. G.; Marder-Eppstein, E.; Pantofaru, C.; Wise, M.; Mösenlechner, L.; Meeussen, W.; and Holzer, S. 2011. Towards autonomous robotic butlers: Lessons learned with the pr2. In *International Conference on Robotics and Automation*, 5568–5575. IEEE.

Colledanchise, M., and Ögren, P. 2016. How Behavior Trees Generalize the Teleo-Reactive Paradigm and And-Or-Trees. In *International Conference on Intelligent Robots and Systems*, 424–429. IEEE.

Colledanchise, M.; Murray, R. M.; and Ogren, P. 2017. Synthesis of Correct-by-Construction Behavior Trees. In *International Conference on Intelligent Robots and Systems*.

Doherty, P., and Kvarnstram, J. 2001. TALplanner: A temporal logic-based planner. *AI Magazine* 22:95.

Doherty, P., and Kvarnström, J. 2008. Temporal action logics. *Foundations of Artificial Intelligence* 3:709–757.

Doherty, P.; Kvarnstrom, J.; Wzorek, M.; Heintz, F.; and Conte, G. 2014. HDRC3: A Distributed HybridDeliberative/Reactive Architecture for Unmanned Aircraft Systems. In *Handbook of Unmanned Aerial Vehicles*. Springer. 849–952.

Doherty, P.; Heintz, F.; and Kvarnström, J. 2013. High-level mission specification and planning for collaborative unmanned aircraft systems using delegation. *Unmanned Systems* 1:75–119.

Emerson, E. A., and others. 1990. Temporal and modal logic. *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)* 995:5.

Gerevini, A. E.; Haslum, P.; Long, D.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence* 173:619–668.

Kvarnström, J. 2011. Planning for Loosely Coupled Agents Using Partial Order Forward-Chaining. In *International Conference on Automated Planning and Scheduling*, 138–145.

Maniatopoulos, S.; Schillinger, P.; Pong, V.; Conner, D. C.; and Kress-Gazit, H. 2016. Reactive high-level behavior synthesis for an atlas humanoid robot. In *International Conference on Robotics and Automation*, 4192–4199. IEEE.

Marzinotto, A.; Colledanchise, M.; Smith, C.; and Ögren, P. 2014. Towards a unified behavior trees framework for robot control. In *International Conference on Robotics and Automation*, 5420–5427. IEEE.

Nguyen, H.; Ciocarlie, M.; Hsiao, K.; and Kemp, C. C. 2013. Ros commander (rosco): Behavior creation for home robots. In *International Conference on Robotics and Automation*, 467–474. IEEE.

Prud'homme, C.; Fages, J.-G.; and Lorca, X. 2016. Choco Documentation.

Tsang, E. 1993. *Foundations of Constraint Satisfaction*. Academic Press.

Wongpiromsarn, T.; Topcu, U.; and Murray, R. M. 2013. Synthesis of control protocols for autonomous systems. *Unmanned Systems* 1:21–39.

Combining Planning and Model Checking to Get Guarantees on the Behavior of Safety-Critical UAV Systems

Hoang Tung Dinh, Mario Henrique Cruz Torres, Tom Holvoet
imec-DistriNet, KU Leuven, 3001 Leuven, Belgium
{hoangtung.dinh, mariohenrique.cruztorres, tom.holvoet}@cs.kuleuven.be

Abstract

Until now, the application of planning on safety-critical UAV systems is limited. That is because planning techniques make unrealistic assumptions, such as the frame assumption or fairness assumption. It is not clear whether the desired properties of UAVs' behavior still hold during execution if these assumptions are violated. To overcome this issue, we promote the combination of planning and model checking techniques in developing the behavior of safety-critical UAV systems. We present an iterative approach where the behavior of a UAV is generated by planning and then verified by model checking. Constraint programming is used to solve both planning and model checking problems in our approach. The use of planning with constraint programming in specifying and generating UAV's behavior has been presented in our previous research (Dinh, Torres, and Holvoet 2018). In this paper, we focus on describing the use of model checking with constraint programming and its integration with planning. We also discuss the potential advantages of this approach over other traditional planning approaches.

1 Introduction

Since autonomous UAVs often operate in an open environment, they must continually observe and react to changes, exogenous events and failures. Besides that, the behavior of an autonomous UAV needs to be goal-directed. Given that, constructing the behavior of autonomous UAVs manually, for example, using Finite State Machine (FSM) (Nguyen et al. 2013), is not trivial.

Planning techniques can be used to automatically generate the behavior of a UAV (Dinh, Torres, and Holvoet 2018). However, this approach does not attract much attention in safety-critical applications since existing planning techniques make assumptions that often do not hold in reality (Ghallab, Nau, and Traverso 2014) such as the frame assumption and the fairness assumption (Sardina and D'Ippolito 2015). It is not clear whether the desired properties of the generated behavior still hold at execution time if the assumptions made by the planning techniques are violated. While it is always necessary to make assumptions about the environment to provide guarantees on the properties of the behavior, such assumptions can often be less restricted and more realistic than the assumptions made by planning techniques. To distinguish the assumptions made by planning techniques and the necessary environment assumptions to provide guarantees on the properties of the behavior, we call the former planning assumptions and the latter environment assumptions.

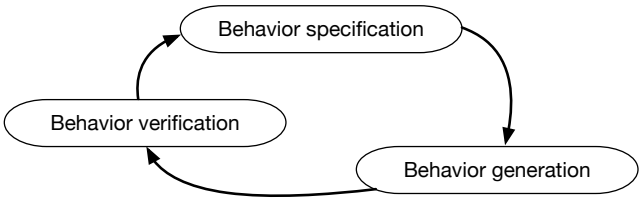


Figure 1: An iterative approach to specify, generate and verify the behavior of an autonomous UAV.

To advocate the use of planning in safety-critical UAV systems, we believe that it is necessary to combine planning with model checking in an iterative approach. In this paper, we introduce an innovative approach to develop the behavior of safety-critical UAV systems that uses techniques from both planning and model checking studies to complement each other. We find that both planning and model checking problems in our approach can be specified and solved efficiently by constraint programming technology.

In our approach, one develops the behavior of a UAV in three phases as illustrated in Figure 1. In the first phase, the behavior of the UAV is specified as a set of logical rules, which is then translated to a set of planning problems. In the second phase, a policy which represents the behavior of the UAV is generated by solving the planning problems. The first two phases are described in our previous research (Dinh, Torres, and Holvoet 2018). In the third phase, we use a technique inspired by Bounded Model Checking (Biere et al. 2003) to verify the desired properties of the generated policy with a set of environment assumptions. The assumptions and the properties are modeled in a Constraint Satisfaction Problem (CSP). Then, a constraint solver searches for a counter-example of maximum length k that falsifies the desired properties. If a counter-example is found, one can go back to the first phase to modify the behavior specification and generate another policy. If no counter-example is found, depending on the properties, we can guarantee that the properties hold either infinitely (for achievement properties) or

up to the given bound k (for safety properties).

Note that, our approach requires human in the loop to interpret counter-examples and modify the behavior specification. In the literature, controller synthesis (Wongpiromsarn, Topcu, and Murray 2013) studies techniques to automatically generate reactive behavior from a specification with assumptions written in formal languages such as Linear Temporal Logic (LTL). This approach is not pragmatic due to three reasons. First, it is computationally expensive and can only be used for small problems (while planning techniques are much more efficient). Second, it only works with a restricted set of LTL language which limits the expressibility. Third, this approach only generates a feasible behavior and does not try to generate the most compact one, leading to complex behavior which is difficult to understand.

This paper is organized as follows. Section 2 discusses the issues of planning techniques. Section 3 explains our approach to verify the behavior of a UAV using model checking and constraint satisfaction. Section 4 describes an example where the proposed approach is applied to verify the behavior of a UAV in a simplified scenario. Section 5 draws conclusions and details possible future work.

2 Issues with planning

Planning requires actions to be represented with preconditions and effects. While modeling preconditions is an easy problem, modeling effects is non-trivial. First, most of the time, it is impossible for one to think about all the possible effects of an action. Second, even when it is possible to model all possible effects, it is costly to reason over all the potential effects.

To overcome this problem, planning techniques make some assumptions on the effects of actions. Classical planning requires each action to have only one deterministic effect. Nondeterministic planning (Muise, Belle, and McIlraith 2014) allows an action to have nondeterministic effects. However, they make the fairness assumption (Sardina and D'Ippolito 2015) which states that if an action is executed infinitely, all effects will eventually occur. This assumption clearly does not hold in many situations. For example, the *open-door* action of a robot could result in two different effects: *door-open* or *door-closed*. We cannot expect that if the robot tries to open the door infinitely, the door will eventually open because someone may have locked the door.

Similar to nondeterministic planning, probabilistic planning such as MDPs and POMDPs also assumes the fairness in action's effects. Moreover, they require to model the probability that each effect happens. In many cases, it is difficult to know such probability. For example, it is not clear which probability one should assign to the effect *door-open* when the robot executes the *open-door* action.

In addition, all planning techniques rely on the closed-world assumption (CWA) which states that all information about the planning problem is explicitly stated. The CWA leads to the frame assumption, that is, if a state variable is not changed by the effects of the action(s) executed in the previous time step, it will remain unchanged. It also means that planning techniques do not allow exogenous events to be modeled. A state variable can only be changed by the execution of actions. In reality, since UAVs operate in an open environment, the occurrence of exogenous events is unavoidable.

Note that, a workaround to model exogenous events in nondeterministic and probabilistic planning is to replace the effects of each action by the joint effects of the action and exogenous events (Ghallab, Nau, and Traverso 2016). However, even with only a few exogenous events, the joint effects can be complex. In addition, it is error-prone and not intuitive for one to think about all exogenous events while modeling an action.

While the assumptions described above often do not hold in reality, they have advantages. First, they reduce the computational complexity of planning techniques. Thanks to that, planning problems can be solved efficiently by many existing algorithms. Controller synthesis techniques, in contrast, do not have these assumptions but their computational complexity is too high to be useful. Second, it is easy for one to describe a planning problem. This advantage does not hold for probabilistic planning because it is not easy to model correctly the probability. Third, one could easily interpret planning solutions.

3 Verify planning solutions

Thanks to the advantages of planning described in Section 2, an approach to specify and generate UAV's behavior has been proposed based on planning techniques (Dinh, Torres, and Holvoet 2018). In that approach, one specifies the behavior of a UAV as a set of logical rules, which is then translated to a set of planning problems. Then, an execution policy is generated by solving the planning problems using constraint satisfaction technology. The approach corresponds to the behavior specification and behavior generation phases in Figure 1.

While the generated policy guarantees some desired properties by construction (e.g., properties related to preconditions and reaction rules), it is not clear whether desired properties related to the effects of actions, such as whether the UAV will achieve a goal, will hold during the execution. More specifically, we want to verify that if the generated policy is executed in an open environment, given a set of environment assumptions which are less restricted than the planning assumptions, a set of desired properties hold. This corresponds to the behavior verification phase in Figure 1.

Constraint programming is a powerful technology thanks to its ease of modeling and efficiency. In (Dinh, Torres, and Holvoet 2018), constraint programming has been used to specify and solve planning problems to generate execution policies. Interestingly, constraint programming can also be used to solve the behavior verification problem.

Our proposed verification approach is inspired by Bounded Model Checking (Biere et al. 2003). We define a bounded number of steps k that the policy will be executing. The *system*, that is, the execution policy, the environment assumptions and the negations of the desired properties are modeled as constraints in a constraint satisfaction problem (CSP). Then, a constraint solver searches for a feasible solution for the CSP. The main idea is that if there exists a solution for the CSP, at least one of the desired properties is

falsified. The details of the approach are demonstrated via an example in Section 4.

If the CSP can falsify at least one of the desired properties, a counter-example leading to the invalidation of the property is generated. Based on the counter-example, one could go back to the behavior specification phase to modify the specification and then generate a new policy. The iterative process stops once no counter-example is found. For properties related to goal achievement, that is, a state has been visited, if there exists no counter-example within k steps, a hard guarantee can be provided since there will be no counter-example in k' steps, where $k' > k$. For other properties, depending on the application domains and the scenarios, a large enough k may be acceptable to provide hard guarantees.

4 Example

In this section, we describe a simple example to demonstrate the process illustrated in Figure 1.

Behavior specification and generation

Assuming a UAV moves in an environment containing four regions labeled A, B, C and D. All the regions are connected, that is, it is always possible to move from one region to another region. However, the structure of the environment is unknown. Figure 2 shows three examples of the environment structure. The goal of the UAV is to take one picture in region A and one in region B for inspection purposes. The UAV can observe the region it is currently in and it knows whether the pictures at A and B have been taken. Given that, we specify the following state variables.

$$\begin{aligned} &\mathbf{S} = \{S_{loc}, S_{pic}(x)\}, x \in \{A, B\} \\ &S_{loc} = \{A, B, C, D\} \\ &S_{pic}(x) = \{not\_taken, taken\}, x \in \{A, B\} \end{aligned} \tag{1}$$

The following actions are defined with their corresponding meanings. For simplicity, we assume that actions can only be executed sequentially.

- $\mathbf{TakePic}(x), x \in \{A, B\}$
  - Preconditions: $S_{loc} = x$
  - Desired effects: $S_{pic}(x) = taken$
- $\mathbf{GoTo}(x), x \in \{A, B\}$
  - Preconditions: $none$
  - Desired effects: $S_{loc} = x$

The goal of the UAV is specified as follows.

$$S^G_{pic}(A) = taken \wedge S^G_{pic}(B) = taken \tag{2}$$

Note that, at runtime, the UAV executes the $\mathbf{GoTo}(x)$ action by continuously moving from its current position to the x region. Figure 2 illustrates the execution of the $\mathbf{GoTo}(B)$ action in different environment structures. To go to B, the UAV may have to pass by other regions. Since the structure of the environment is unknown, it is not possible to model precisely the effects of the $\mathbf{GoTo}(x)$ actions. Therefore, we only model the desired effects of the actions, that is, the UAV will reach region x. Due to this incorrectness in modeling, it is not clear whether the UAV is always able to achieve the goal in Eq. (2).

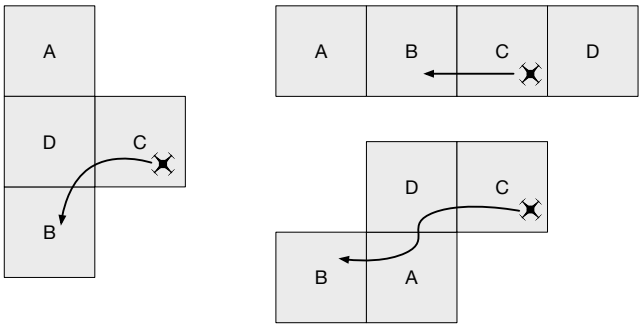


Figure 2: Three example structures of the environment and example executions of the $\mathbf{GoTo}(B)$ action.

An execution policy is then generated by creating and solving a planning problem from each possible state value (Dinh, Torres, and Holvoet 2018).

Behavior verification

Given the generated policy, we construct a CSP to verify whether the UAV is always able to achieve the goal in (2) given a set of environment assumptions. We define the number bounded steps $k = 10$. The policy is represented by the following constraints.

$$\mathbf{S^i} = s \Rightarrow \mathbf{A^i} = a,\ i \in \{0, \ldots, k-1\} \tag{3}$$

where $\mathbf{S^i}$ and $\mathbf{A^i}$ represent the state variables and the action to be executed at step i. The values s and a are given by the generated policy.

We represent the environment assumptions by the following constraints.

$$A^i = \mathbf{TakePic}(x) \wedge S^i_{loc} = x \Rightarrow S^{i+1}_{pic}(x) = taken \tag{4}$$

$$S^i_{pic}(x) = taken \Rightarrow S^{i+1}_{pic}(x) = taken \tag{5}$$

$$A^i = \mathbf{GoTo}(x) \wedge S^i_{loc} \neq x \Rightarrow S^{i+1}_{loc} \neq S^i_{loc} \tag{6}$$

$$\begin{aligned} &A^i = \mathbf{GoTo}(x) \wedge A^{i+1} = \mathbf{GoTo}(x) \wedge A^{i+2} = \mathbf{GoTo}(x) \\ &\Rightarrow S^{i+3}_{loc} = x \end{aligned} \tag{7}$$

$$S^0_{loc} = D \wedge S^0_{pic}(A) = not\_taken \wedge S^0_{pic}(B) = not\_taken \tag{8}$$

Assumptions (4) and (5) state the following. If the UAV executes the action $\mathbf{TakePic}(x)$ at location x, it will always get the picture. In other words, this action cannot result in an unexpected effect. If the picture has been taken at x, the picture will never be lost.

Assumptions (6) and (7) represent the effects of the $\mathbf{GoTo}(x)$ actions and are less restricted than the planning assumptions. Assumption (6) states that if the action $\mathbf{GoTo}(x)$ is executed and the UAV is not at x, it will move to another region. However, the exact next region is unknown. Because there are four different regions, if the UAV

Table 1: A counter example

Step	Action	$S_{location}$	$S_{picture}(A)$	$S_{picture}(B)$
0	$\mathbf{GoTo}(B)$	D	not_taken	not_taken
1	$\mathbf{GoTo}(A)$	C	not_taken	not_taken
2	$\mathbf{GoTo}(B)$	D	not_taken	not_taken
3	$\mathbf{GoTo}(A)$	C	not_taken	not_taken
4	$\mathbf{GoTo}(B)$	D	not_taken	not_taken
...	...	...	...	...

keep executing the action $\mathbf{GoTo}(x)$ in three consecutive steps, it will arrive at x. It is represented by assumption (7).

For simplicity, we also assume that at the beginning, the UAV is at region D and both the pictures at A and B have not been taken (assumption (8)). Note that, this assumption is not compulsory since without it, the CSP will search for a counter-example from all possible initial states.

We want to verify that the pictures at A and B will always be eventually taken given the set of environment assumptions above. To do so, we represent the negation of this property as constraints of the CSP. Constraint (9) enforces that there will never be a state where the pictures at A and B have been both taken.

$$\neg(S^i_{picture}(A) = taken \wedge S^i_{picture}(B) = taken) \quad \forall i \in \{0, \ldots, k\} \tag{9}$$

Our CSP model is written in the FO($\cdot$) language and is solved by the knowledge base system IDP (De Cat et al. 2014). The full model is available online[1]. The solver returned a counter-example presented in Table 1.

From the counter-example, we realize that the situation in Figure 4 may arise. Given that both pictures at A and B have not been taken, if the UAV is at D, the policy tells the UAV to go to B. To move to B, the UAV must pass by C. As soon as the UAV arrives at C, the policy tells it to go to A and then the UAV arrives at D again.

We fix this issue by adapting the goal in Eq. (2) to the following goals which state that the UAV should try to take a picture at B only after the picture at A has been taken.

$$S^G_{picture}(A) = taken \tag{10}$$

$$S_{picture}(A) = taken \Rightarrow S^G_{picture}(B) = taken \tag{11}$$

After the modification, we generate a new policy and search for a counter-example again using the same CSP described above. This time, no counter-example is found.

Note that, for achievement property (the UAV will eventually reach a state) such as the one demonstrated in this example, if there is no counter example with the length less than k steps, there will be no counter example with the length more than k steps. However, for safety property (e.g., the UAV will never arrive at a state), we cannot guarantee that after k steps, the UAV will not arrive at a to-be-avoided state. Depending on the concrete domains and scenarios, a large enough k may be acceptable.

[1] https://github.com/hoangtungdinh/idp-model-checking-example

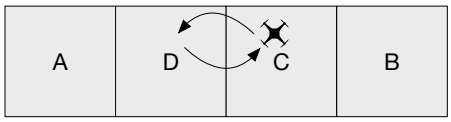


Figure 3: A situation preventing the UAV from achieving the goals.

5 Conclusions

We presented our view on how to advocate the use of planning in safety-critical UAV systems. Combining planning with model checking in an iterative approach is necessary to overcome the limitations in the assumptions of planning techniques. We also show that constraint programming can be used as an effective tool for modeling and solving both planning and model checking problems in our approach.

Several problems remain open. First, it is important to understand which kinds of desired properties we can verify using our model checking approach. Second, the scalability of the approach needs to be studied in depth. Finally, the approach needs to be tested with more scenarios from real-world applications.

Acknowledgment

This research is partially funded by the Research Fund KU Leuven, and by the imec-ICON project SafeDroneWare.

References

Biere, A.; Cimatti, A.; Clarke, E. M.; Strichman, O.; and Zhu, Y. 2003. Bounded model checking. *Advances in computers* 58:117–148.

De Cat, B.; Bogaerts, B.; Bruynooghe, M.; Janssens, G.; and Denecker, M. 2014. Predicate Logic as a Modelling Language: The IDP System.

Dinh, H. T.; Torres, M. H. C.; and Holvoet, T. 2018. Sound and Complete Reactive UAV Behavior using Constraint Programming. In *ICAPS Workshop on Planning and Robotics*.

Ghallab, M.; Nau, D.; and Traverso, P. 2014. The actor's view of automated planning and acting: A position paper. *Artificial Intelligence* 208:1–17.

Ghallab, M.; Nau, D.; and Traverso, P. 2016. *Automated Planning and Acting*. Cambridge University Press.

Muise, C. J.; Belle, V.; and McIlraith, S. A. 2014. Computing Contingent Plans via Fully Observable Non-Deterministic Planning. In *AAAI*, 2322–2329.

Nguyen, H.; Ciocarlie, M.; Hsiao, K.; and Kemp, C. C. 2013. Ros commander (rosco): Behavior creation for home robots. In *International Conference on Robotics and Automation*, 467–474. IEEE.

Sardina, S., and D'Ippolito, N. 2015. Towards Fully Observable Non-Deterministic Planning as Assumption-based Automatic Synthesis. In *IJCAI*, 3200–3206.

Wongpiromsarn, T.; Topcu, U.; and Murray, R. M. 2013. Synthesis of control protocols for autonomous systems. *Unmanned Systems* 1:21–39.