# Sistemi Per il Governo dei Robot
## Introduction to ROS and V-REP (CoppeliaSim)

CoppeliaSim

from the creators of V-REP

ROS.org

# Introduction to ROS and V-REP
## ROS: Introduction

What is ROS?

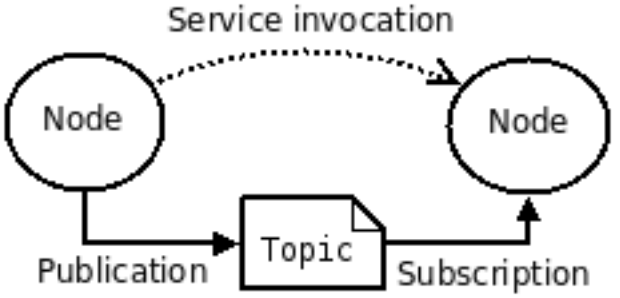
- ROS (Robot Operative System) is an open-source, meta-operating system (or middleware) for robots. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management.
- It provides tools and libraries for obtaining, building, writing, and running code across multiple computers.
- Code can be written in several languages like C++ or Python.

ROS mainly works on Linux!

[http://wiki.ros.org]

# Introduction to ROS and V-REP
## ROS: Key concepts

- **Nodes**: are processes that perform computation (programs). Nodes may control lasers or range-finders, joints or wheel motors, perform localization, path planning, and so on. Organized in **packages**.
- **Master**: provides name registration and lookup nodes. Without the Master, nodes would not be able to find each other or communicate (can be shared among different machines).
- **Communication**: is generally asynchronous using callbacks.
  - **Topics**: transport system with publish/subscribe semantics (similar to pipes).
  - **Services**: One-shot requests.
  - **Actions**: One-shot requests, with control.

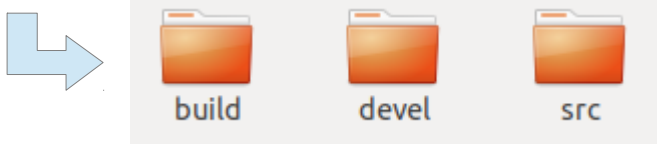


# Introduction to ROS and V-REP
## ROS: organization and custom packages

Install and configure ROS:

http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment


Workspace of ROS (catkin_ws or ros_ws)

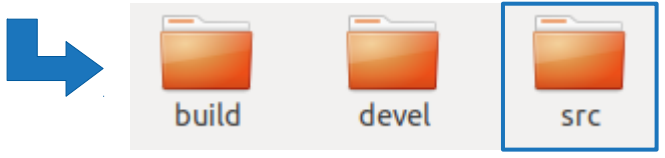


# Introduction to ROS and V-REP
## ROS: organization and custom packages

Install and configure ROS:

http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment


Workspace of ROS (catkin_ws or ros_ws)

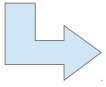


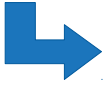

Packages

# Introduction to ROS and V-REP
## ROS: organization and custom packages

Create a custom ROS package:

http://wiki.ros.org/ROS/Tutorials/CreatingPackage

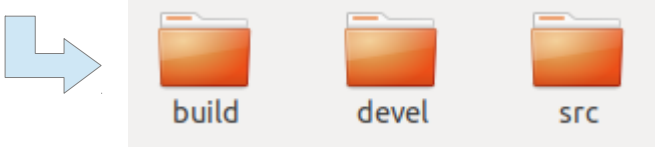




```
$ catkin_create_pkg my_first_pkg DEP1 DEP2
```

# Introduction to ROS and V-REP
## ROS: organization and custom packages

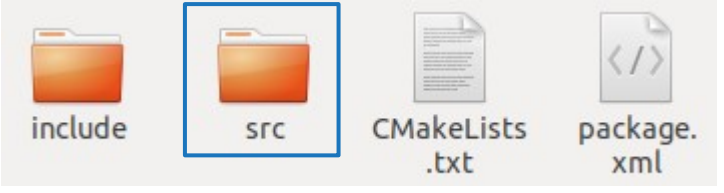
Create a ROS package:

http://wiki.ros.org/ROS/Tutorials/CreatingPackage


```
$ catkin_create_pkg my_first_pkg DEP1 DEP2
```


Nodes source files



# Introduction to ROS and V-REP
## ROS: node example (C++)

Publisher/subscriber example:

http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29

```
1   #include "ros/ros.h"
2   #include "std_msgs/String.h"
3
4   #include <sstream>
5
6   void myRosCallback(const std_msgs::String::ConstPtr& msg)
7   {
8     ROS_INFO("I heard: [%s]", msg->data.c_str());
9   }
10
11  int main(int argc, char **argv)
12  {
13    ros::init(argc, argv, "my_node");
14    ros::NodeHandle n;
15
16    ros::Publisher pub = n.advertise<std_msgs::String>("say_hello", 1000);
17    ros::Subscriber sub = n.subscribe("hear_hello", 1000, myRosCallback);
18
19    ros::Rate loop_rate(10);
```

Initialize ROS and enable publisher and subscriber (the latter with callback)

Publish into the main loop. The message is received asynchronously

```
21    while (ros::ok())
22    {
23      std_msgs::String msg;
24
25      std::stringstream ss;
26      ss << "hello world";
27      msg.data = ss.str();
28
29      pub.publish(msg);
30
31      ros::spinOnce();
32
33      loop_rate.sleep();
34    }
35
36    return 0;
37  }
```

# Introduction to ROS and V-REP
## ROS: packages from repository

ROS have a huge community that continuously develop packages and nodes (often implementing state-of-the-art algorithms).

There are thousands of packages and nodes!

```
$ sudo apt-get install ros-DISTRO-PKG_NAME
```

(e.g.)

```
$ sudo apt-get install ros-melodic-amcl
```

DISTRO: Melodic
PKG: Adaptive Monte Carlo Localization

# Introduction to ROS and V-REP
## ROS: useful commands (nodes)

| | |
|---|---|
| `$ roscore` | Starts the roscore, this have to be always running. |
| `$ rosrun PKG_NAME NODE_NAME` | Starts a ROS node. |
| `$ roslaunch PKG_NAME LAUNCH_FILE_NAME` | Starts a .launch file. This allows multiple nodes to be started. |
| `$ rosnode list` | Plots the list of all nodes running. |
| `$ catkin_make` | Compiles all C++ nodes and packages. |

# Introduction to ROS and V-REP
## ROS: useful commands (nodes)

| | |
|---|---|
| `$ roscore` | Starts the roscore, this have to be always running. |
| `$ rosrun PKG_NAME NODE_NAME` | Starts a ROS node. |
| `$ roslaunch PKG_NAME LAUNCH_FILE_NAME` | Starts a .launch file. This allows multiple nodes to be started. |
| `$ rosnode list` | Plots the list of all nodes running. |
| `$ catkin_make -DCATKIN_WHITELIST_PACKAGES="PKG_NAME"` | Compiles a single package |

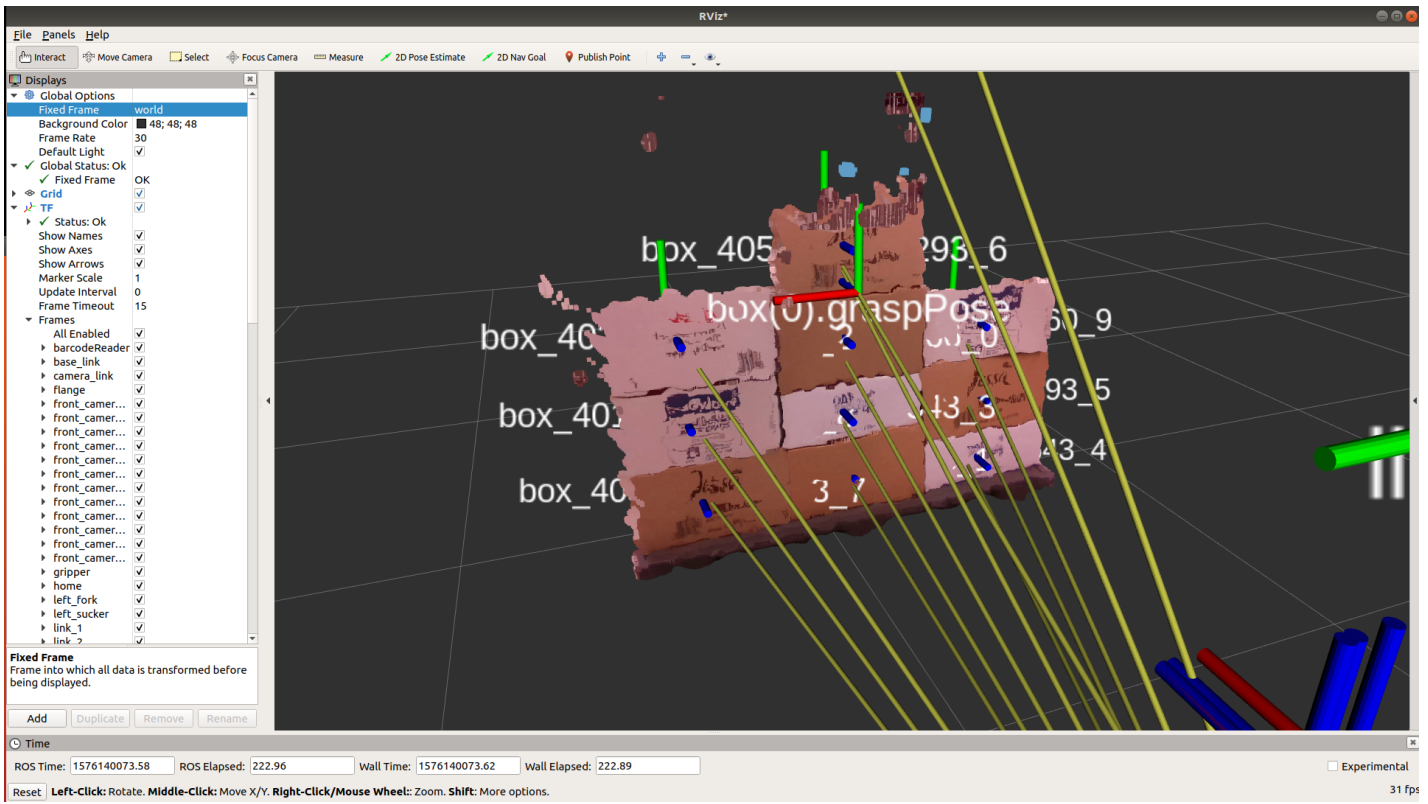
# Introduction to ROS and V-REP
## ROS: useful commands (topics)

| | |
|---|---|
| `$ rostopic list` | Lists all topics published or subscribed by running nodes. |
| `$ rostopic info TOPIC_NAME` | Plots the info of a topic like type or publishing subscribing nodes. |
| `$ rostopic echo TOPIC_NAME` | Plots the value of a topic in real-time. |
| `$ rostopic pub TOPIC_NAME TYPE VALUE` | Publish a value on a topic. |
| `$ rviz` | Graphical visualization of topics. |

# Introduction to ROS and V-REP
## ROS: rviz

Graphical visualization of topics.

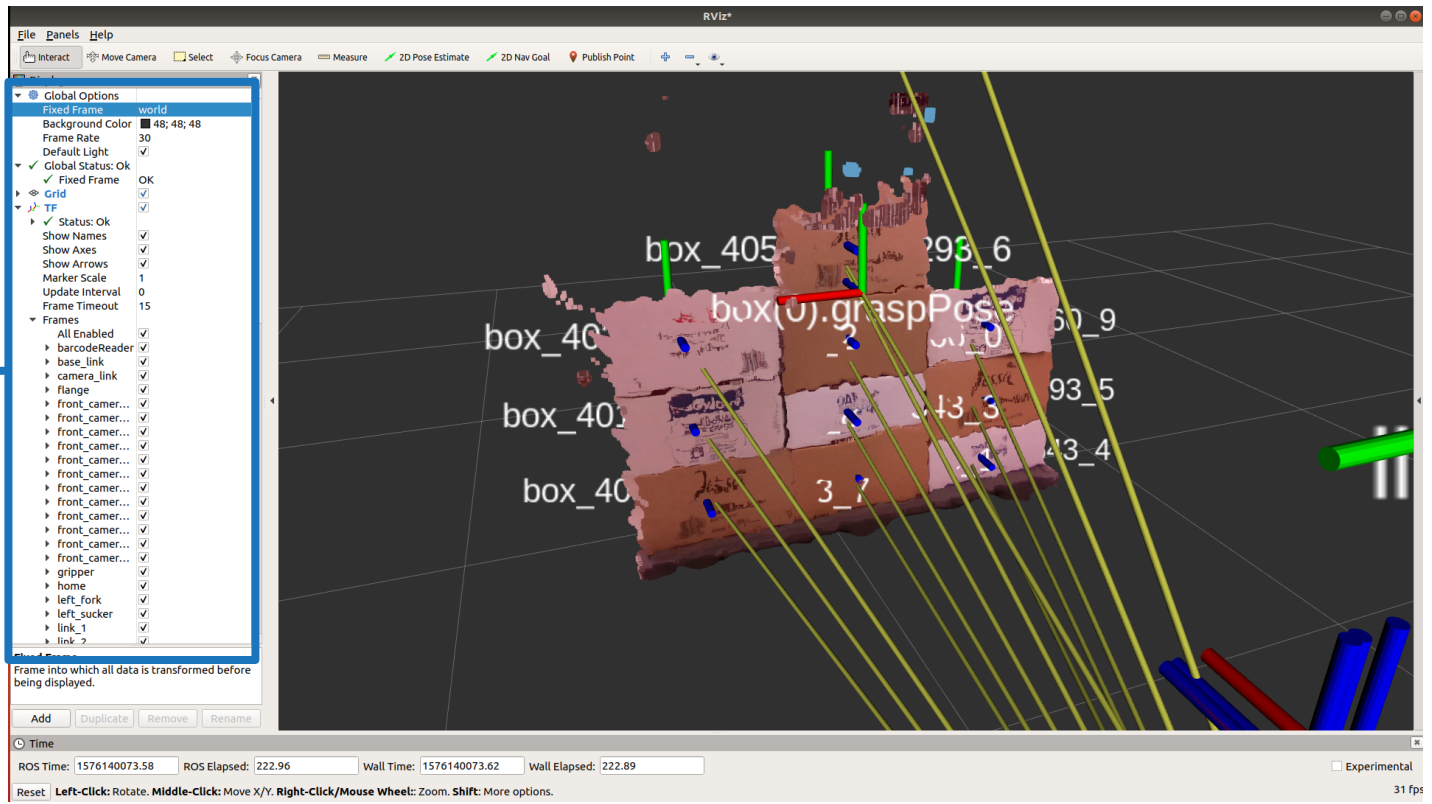
Rviz is a special node that subscribes to topics and provide to some of them a graphical visualization.



# Introduction to ROS and V-REP
## ROS: rviz

Graphical visualization of topics.

List of visualized objects and topics

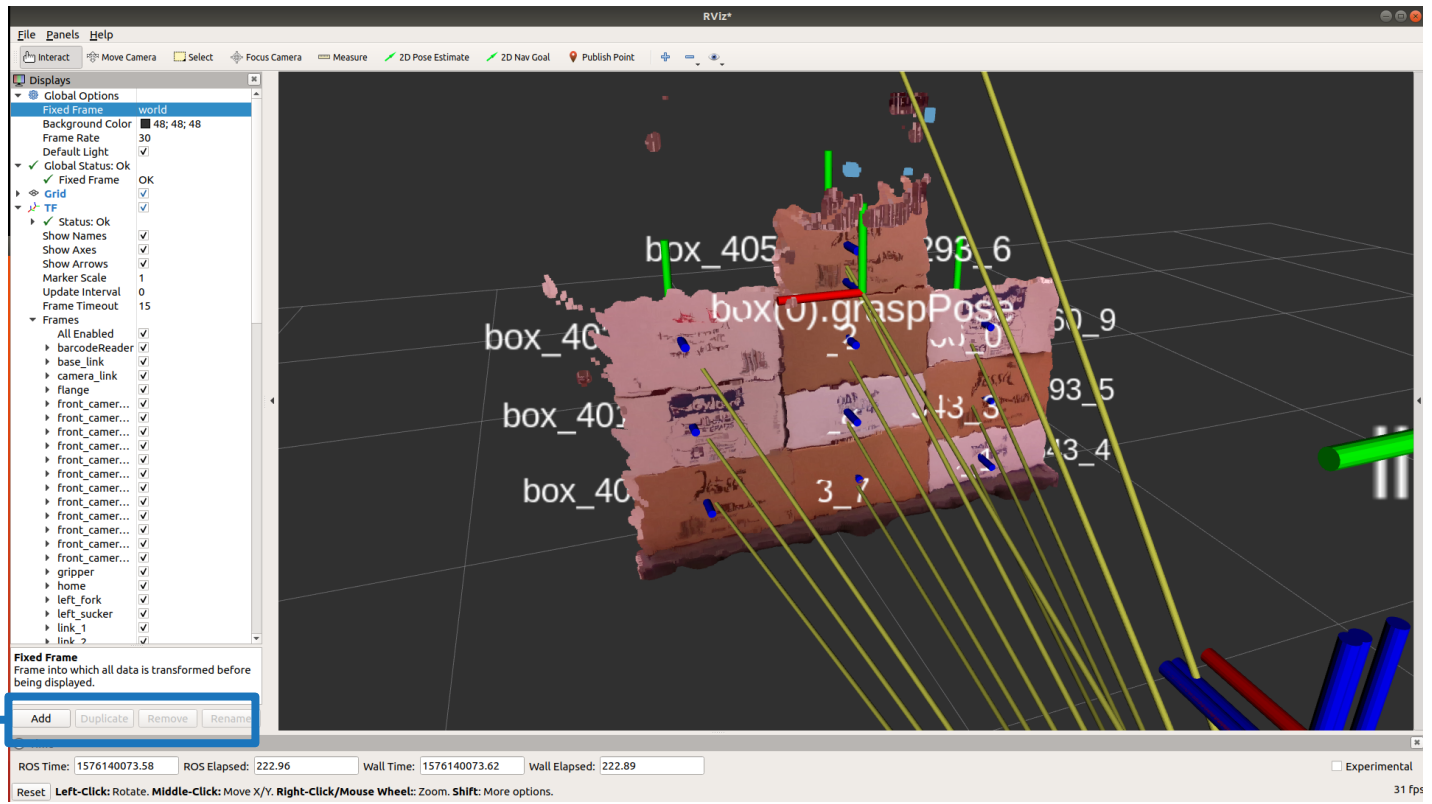


# Introduction to ROS and V-REP
## ROS: rviz

Graphical visualization of topics.

List of visualized objects and topics

Add/remove topics from visualization



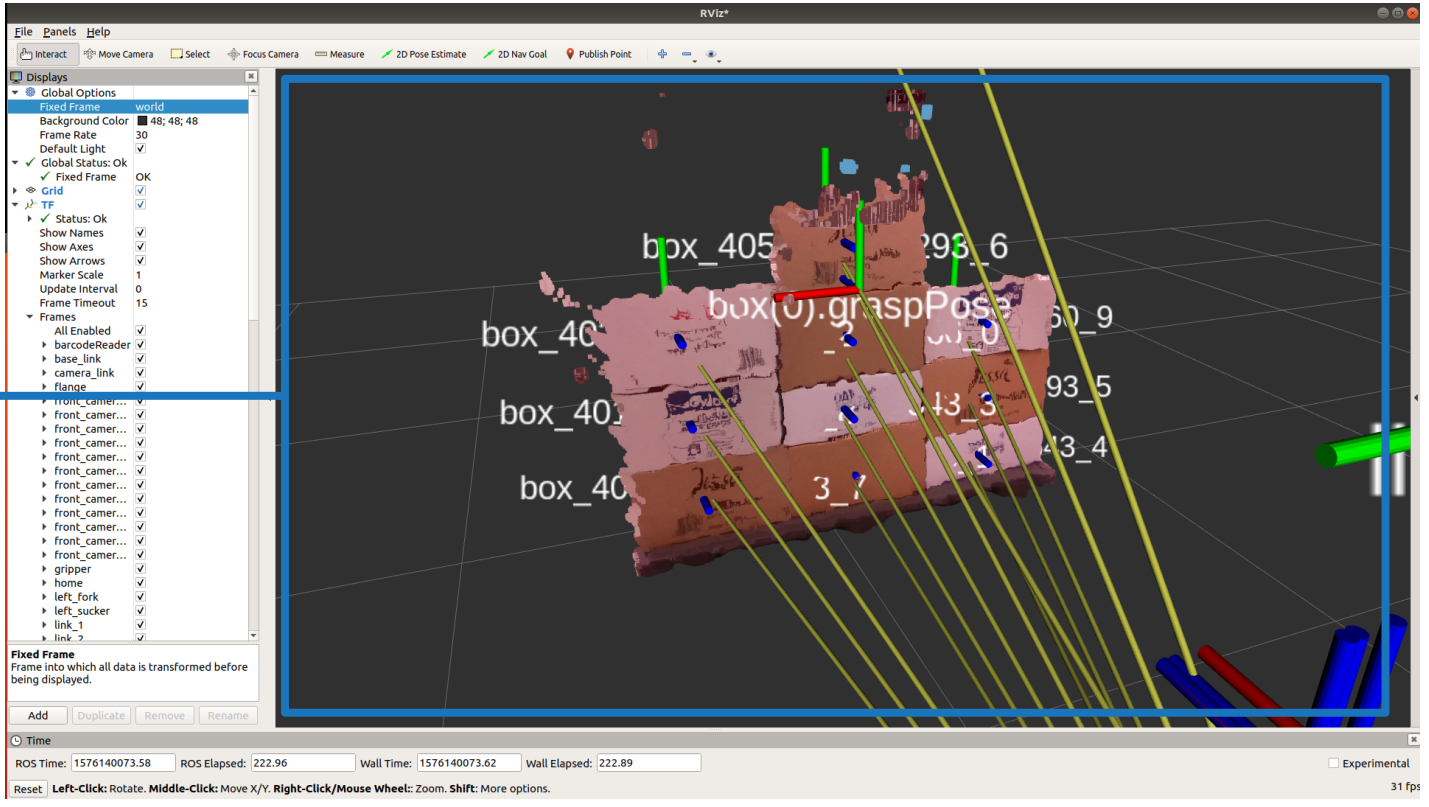
# Introduction to ROS and V-REP
## ROS: rviz

Graphical visualization of topics.

List of visualized objects and topics

Add/remove topics from visualization

Visualization environment



# Introduction to ROS and V-REP
## ROS: rviz

Graphical visualization of topics.

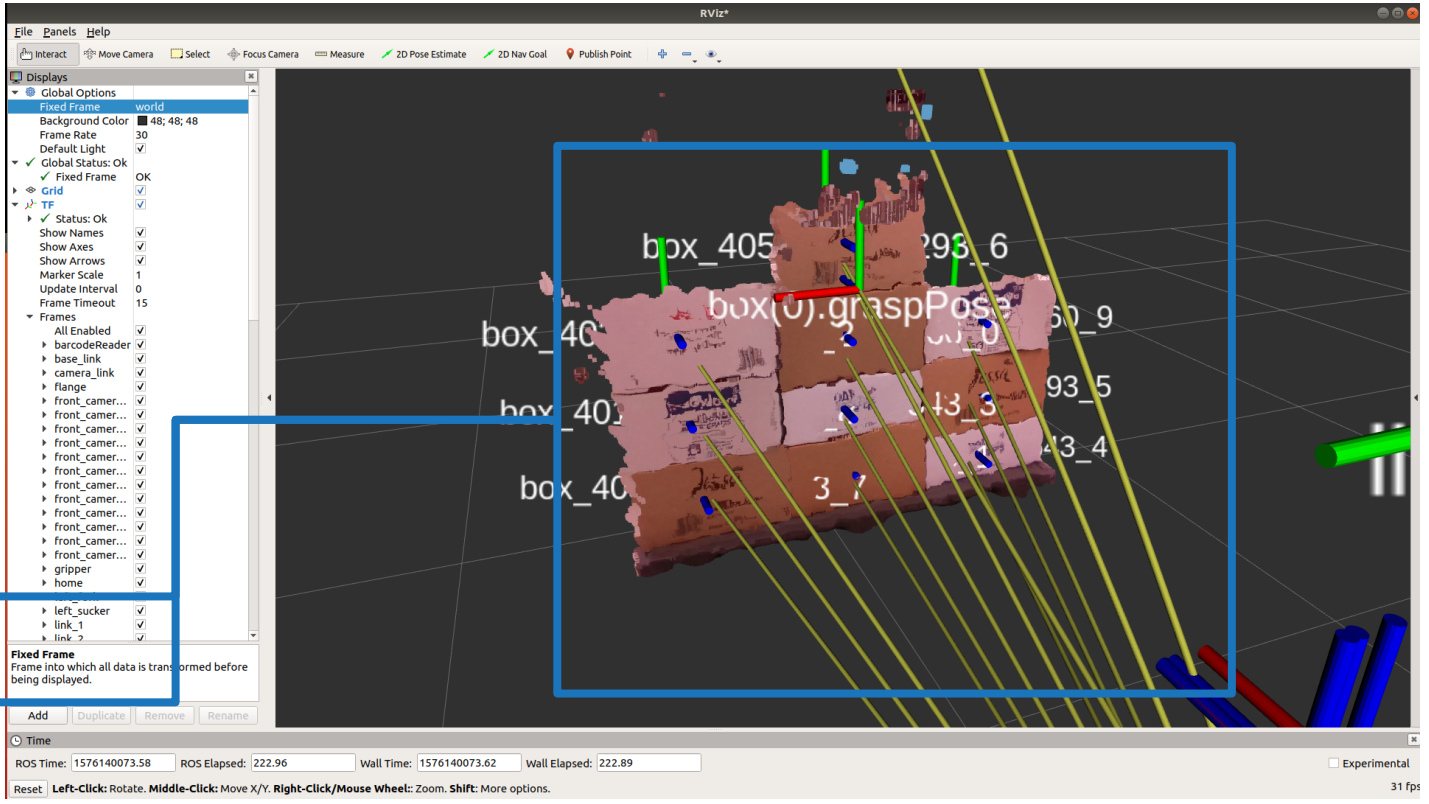
List of visualized objects and topics

Add/remove topics from visualization

Visualization environment

Colored pointcloud

Frames transforms



# Introduction to ROS and V-REP
## V-REP: Introduction

What is V-REP? To be precise, it is no more! From v4.0 it is CoppeliaSim.

- CoppeliaSim is a robot simulator, with integrated development environment. It is based on a distributed control architecture: each object/model can be individually controlled via an embedded script, a plugin, a ROS or BlueZero node, a remote API client, or a custom solution. This makes CoppeliaSim very versatile and ideal for multi-robot applications.
- Controllers can be written in C/C++, Python, Java, Lua, Matlab or Octave. **The integrated environment is Lua**.

CoppeliaSim works on Windows also!

[https://www.coppeliarobotics.com]

# Introduction to ROS and V-REP
## V-REP: Key concepts

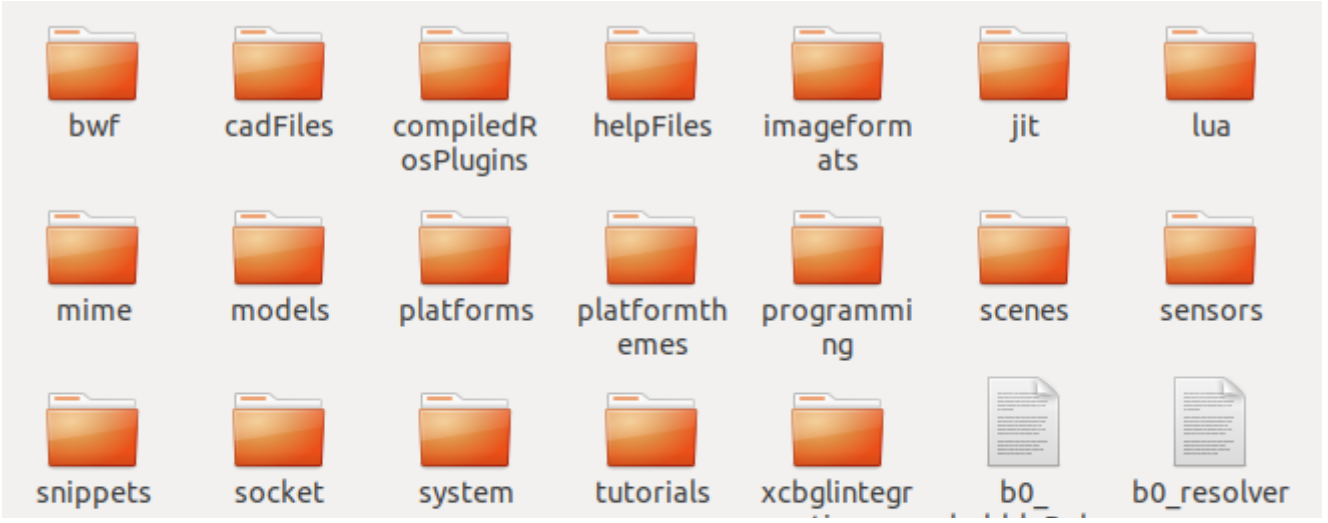
- **Scenes**: are simulated environments that contain static and dynamic objects (models) interacting each other. Specified in .ttt files.
- **Models**: are sub-elements of a scene that can be hierarchically defined by other sub-models. Specified in .ttm files.
- **Scripts**: are processes associated with models that runs during the simulation. Allow scene customization: reading sensors, moving joints, real-time setting poses of models, etc.
  - **Main**: contains the basic code that allows a simulation to run. Without main script, a simulation won't do anything.
  - **Non-threaded**: contain a collection of blocking functions and are called each step of the simulation by the main-script.
  - **Threaded**: are scripts that will launch in a thread.

# Introduction to ROS and V-REP
## V-REP: organization

Download CoppeliaSim:

https://www.coppeliarobotics.com/downloads (EDU version is free)


Workspace: just extract the folder into your home directory.



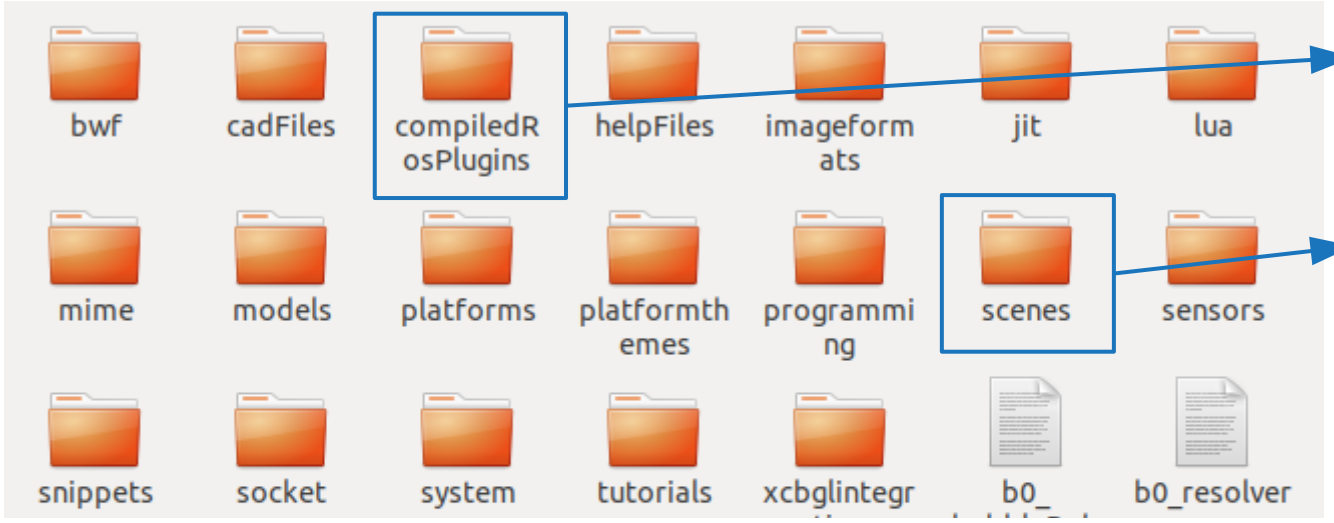

There are tons of files inside this folder.

# Introduction to ROS and V-REP
## V-REP: organization

Download CoppeliaSim:

https://www.coppeliarobotics.com/downloads (EDU version is free)


Workspace: just extract the folder into your home directory.




Enables ROS communication.

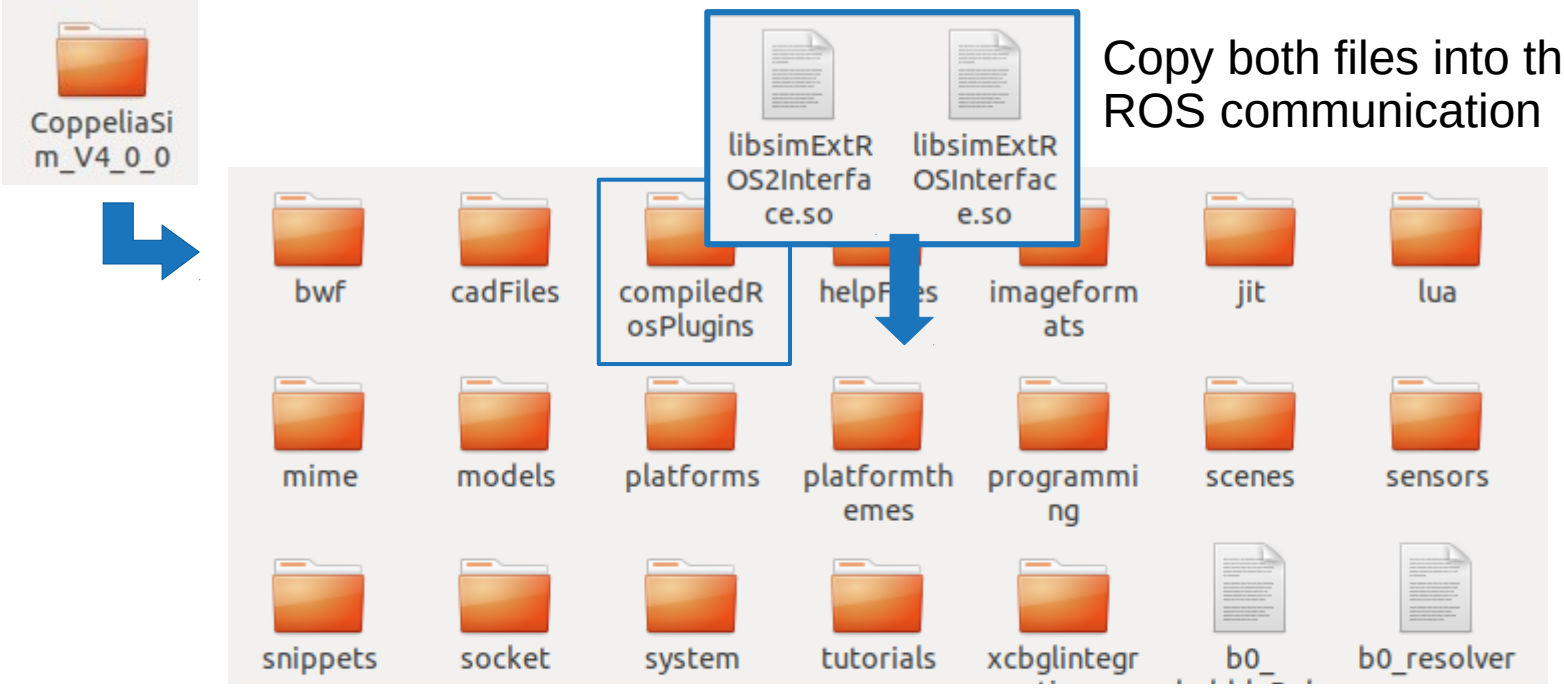
Contains scenes.

There are several useful built-in scenes!

# Introduction to ROS and V-REP
## V-REP: ROS interface

Enable ROS interface:

https://www.coppeliarobotics.com/helpFiles/index.html (ROS section)




Copy both files into the main dir to enable ROS communication

# Introduction to ROS and V-REP
## V-REP: Start simulator with ROS

The simulator is associated to a special node that subscribes or publishes user-defined topics. This node is started when the simulator starts. You can see it from rosnode list.

Terminal A

```
$ roscore
```

Terminal B

```
$ cd CoppeliaSim_V4_0_0/
$ ./coppeliaSim.sh
```

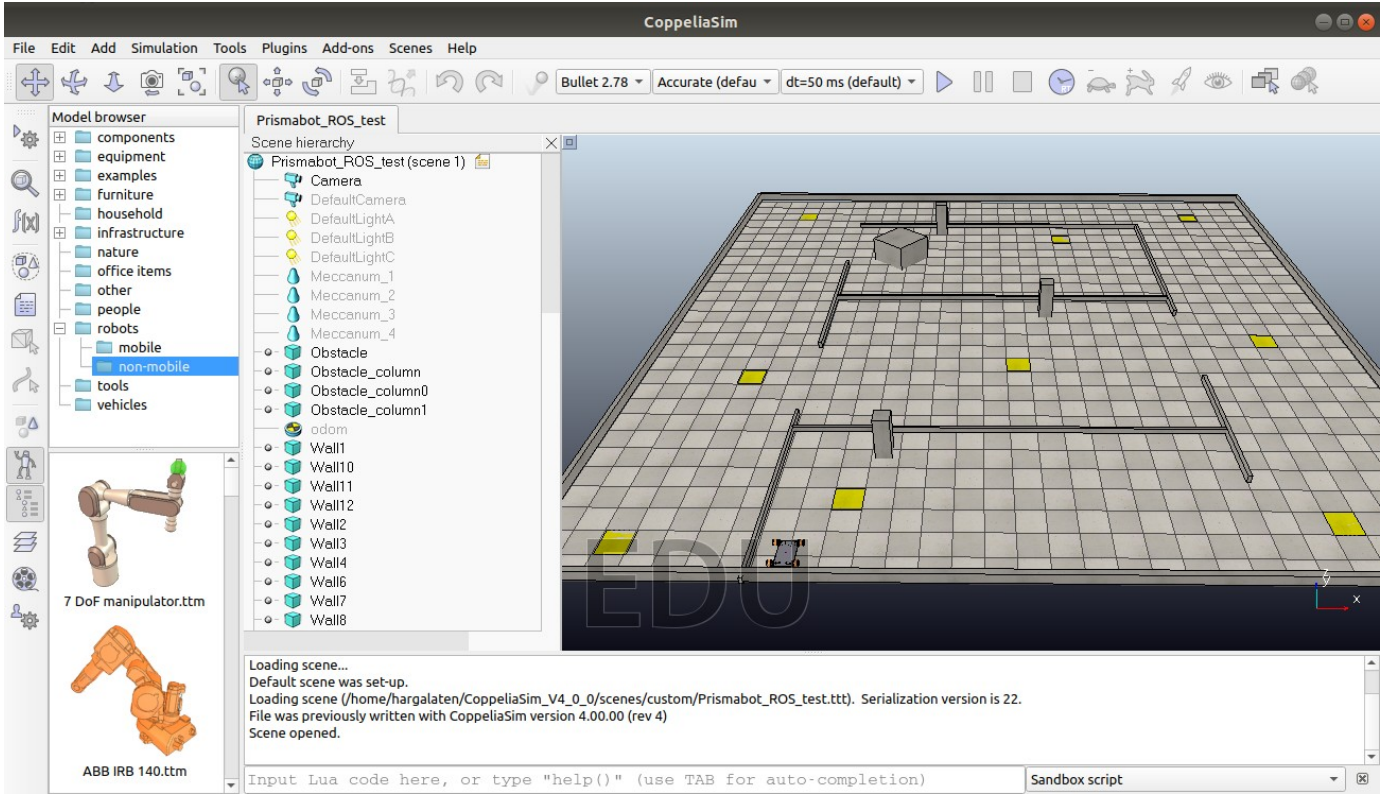
You should see some messages on start

```
Plugin 'ROSInterface': loading...
Plugin 'RosInterface': warning: replaced variable 'simROS'
Plugin 'ROSInterface': load succeeded.
```

# Introduction to ROS and V-REP
## V-REP: description

## CoppeliaSim GUI

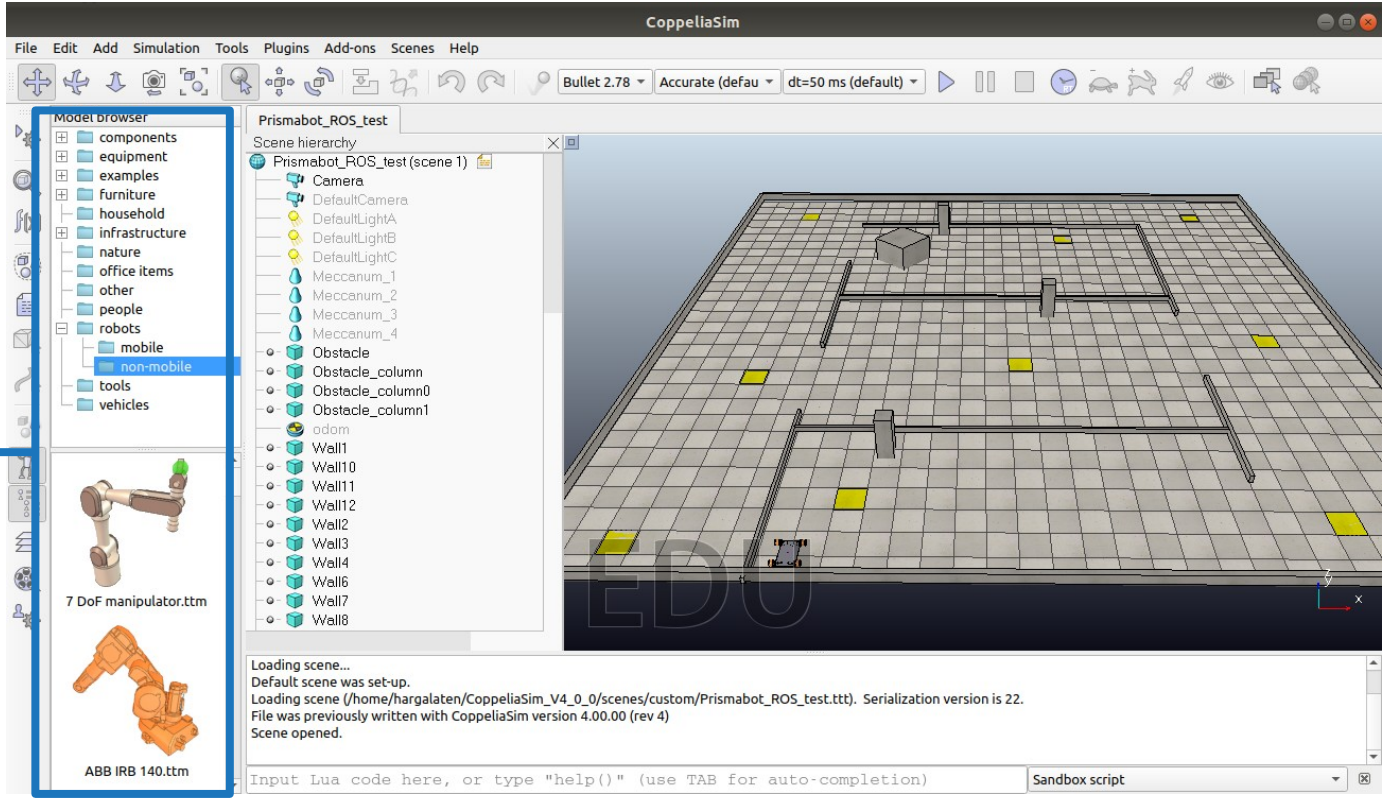
CoppeliaSim is highly configurable from the GUI. This allows to drag-and-drop models into the scene, to set their initial position, to add/remove scripts, etc.



# Introduction to ROS and V-REP
## V-REP: description

### CoppeliaSim GUI

List of built-in models

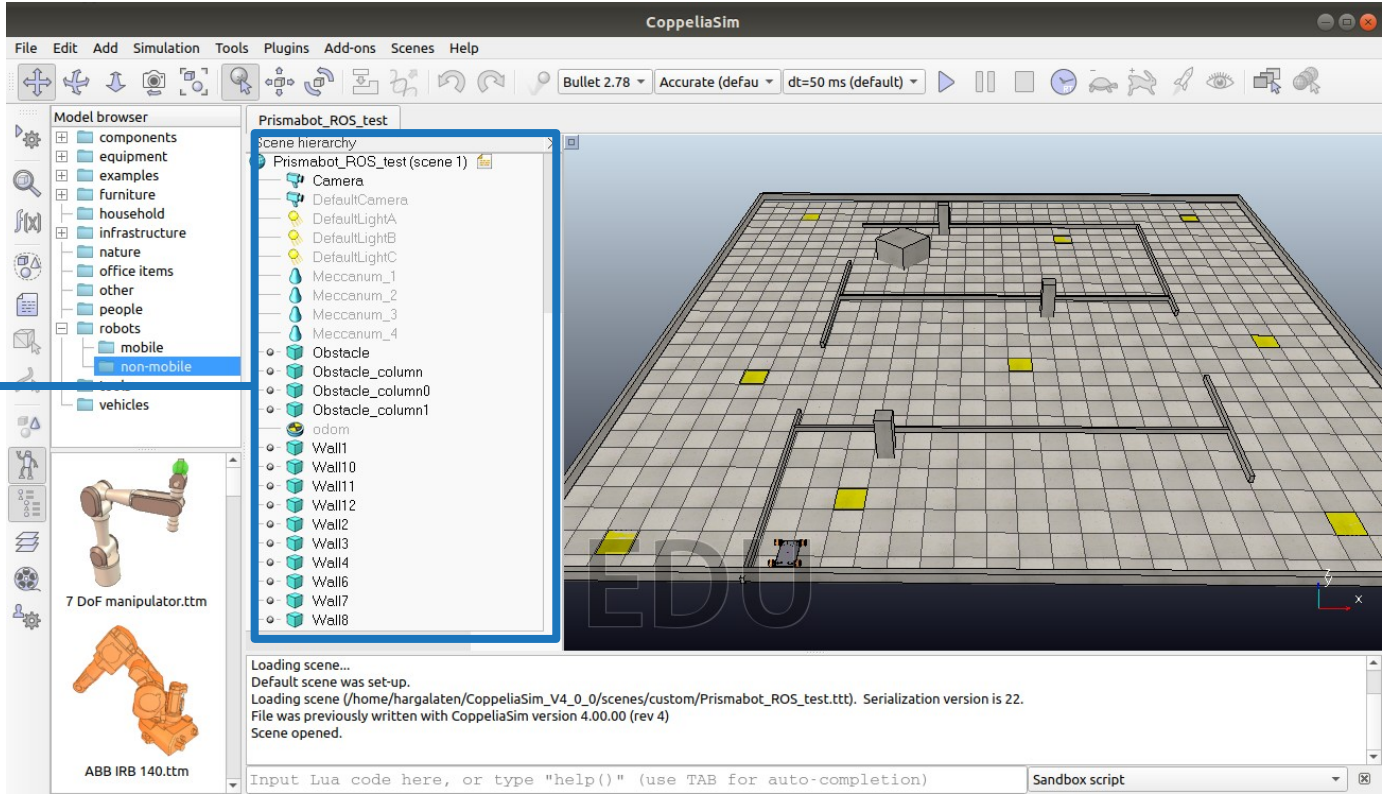


# Introduction to ROS and V-REP
## V-REP: description

### CoppeliaSim GUI

List of built-in models

Tree of models



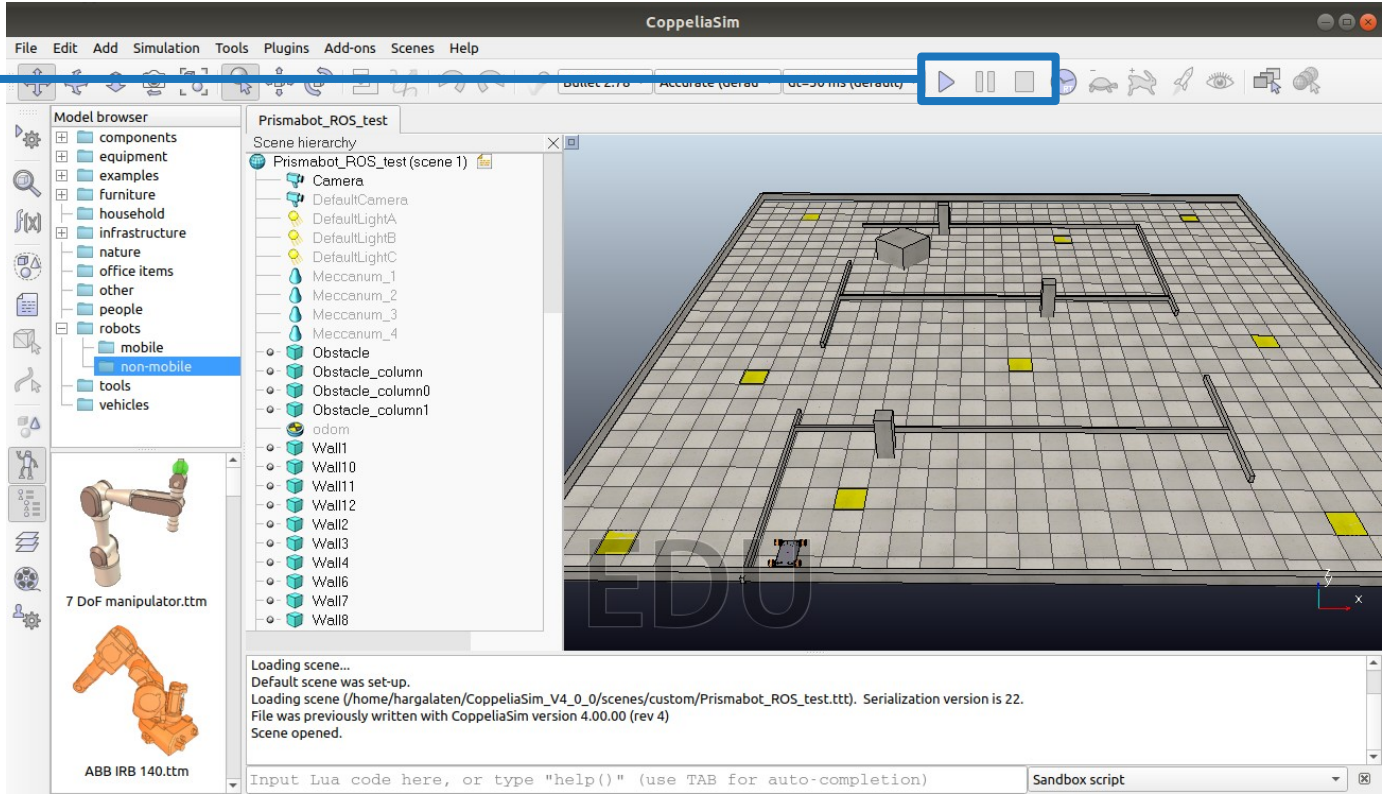
# Introduction to ROS and V-REP
## V-REP: description

### CoppeliaSim GUI

List of built-in models

Tree of models

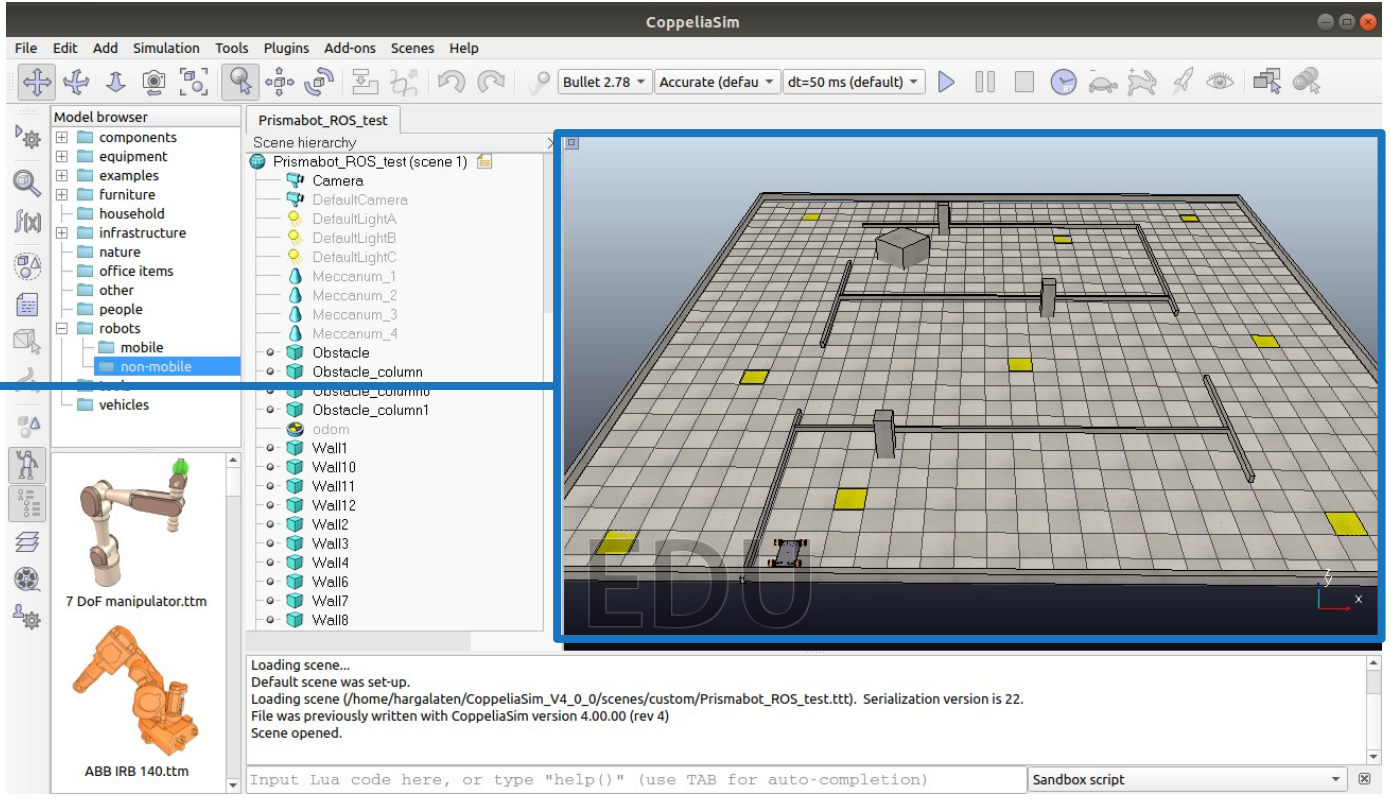
Start/pause/stop simulation



# Introduction to ROS and V-REP
## V-REP: description

### CoppeliaSim GUI

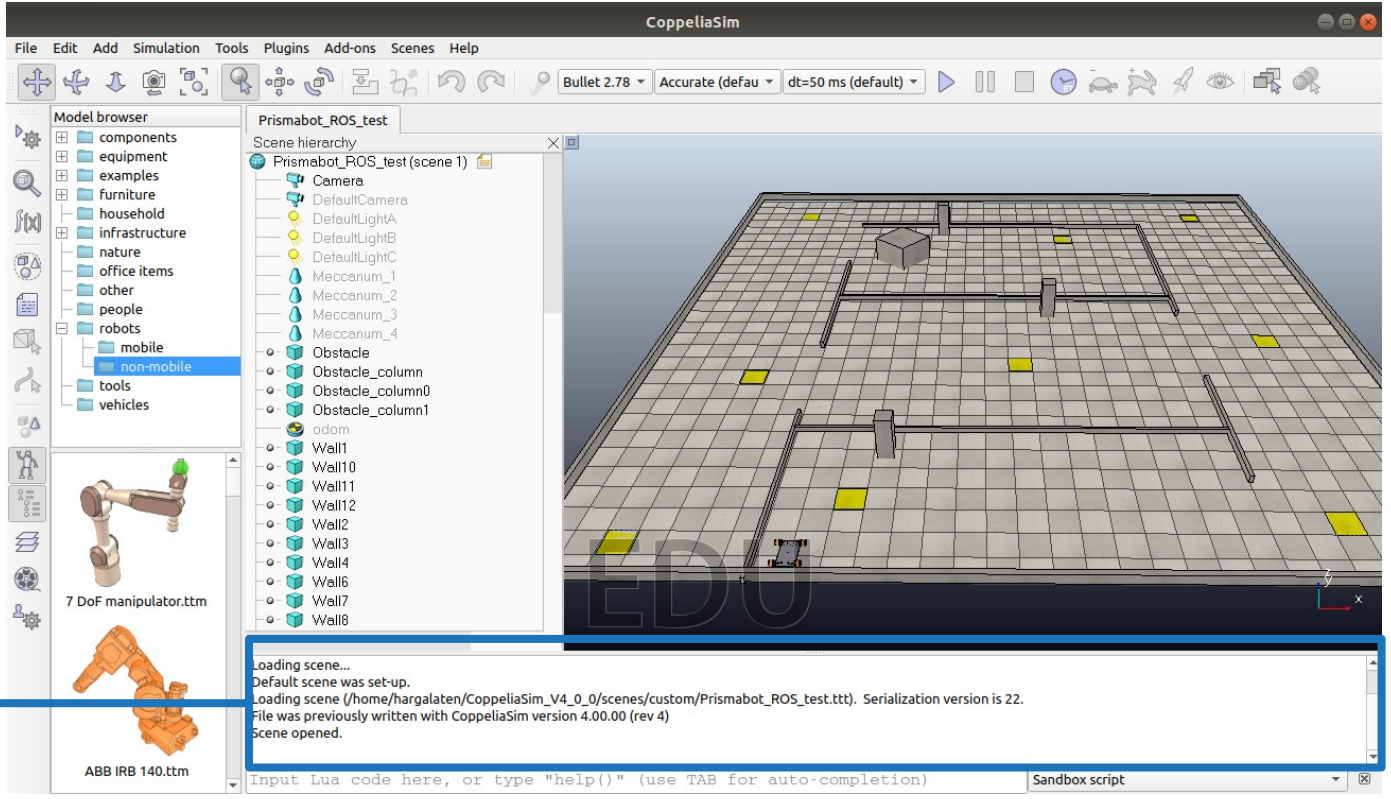
- List of built-in models
- Tree of models
- Start/pause/stop simulation
- Visualized scene



# Introduction to ROS and V-REP
## V-REP: description

### CoppeliaSim GUI

- List of built-in models
- Tree of models
- Start/pause/stop simulation
- Visualized scene
- Output terminal

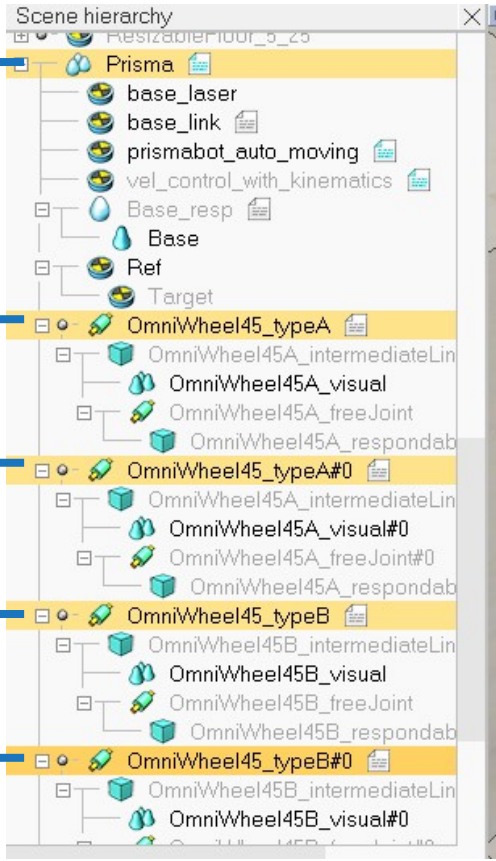


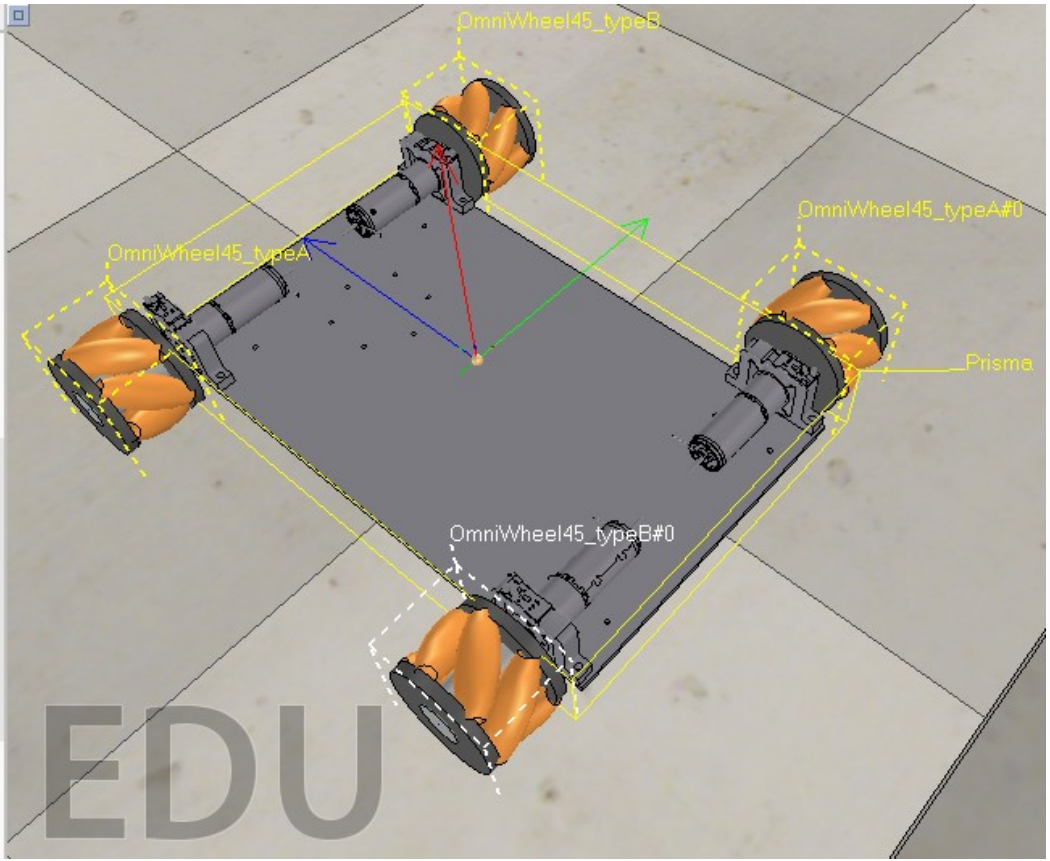
# Introduction to ROS and V-REP
## V-REP: models hierarchy

CoppeliaSim GUI

Root of the robot

Four omni-wheels selected



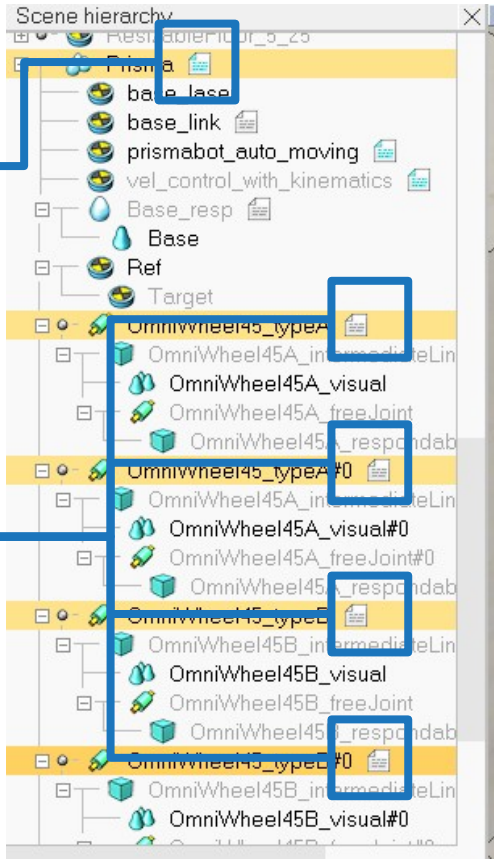



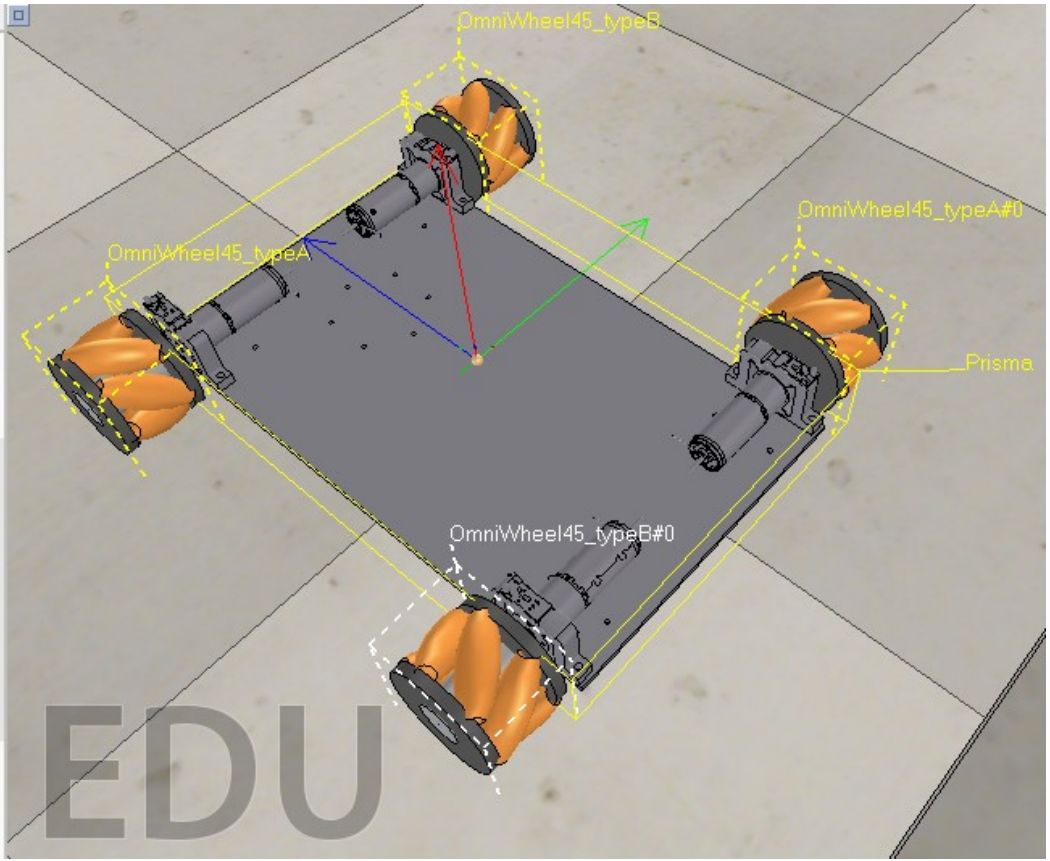
# Introduction to ROS and V-REP
## V-REP: models hierarchy

CoppeliaSim GUI

Robot main threaded script

omni-wheels non-threaded scripts



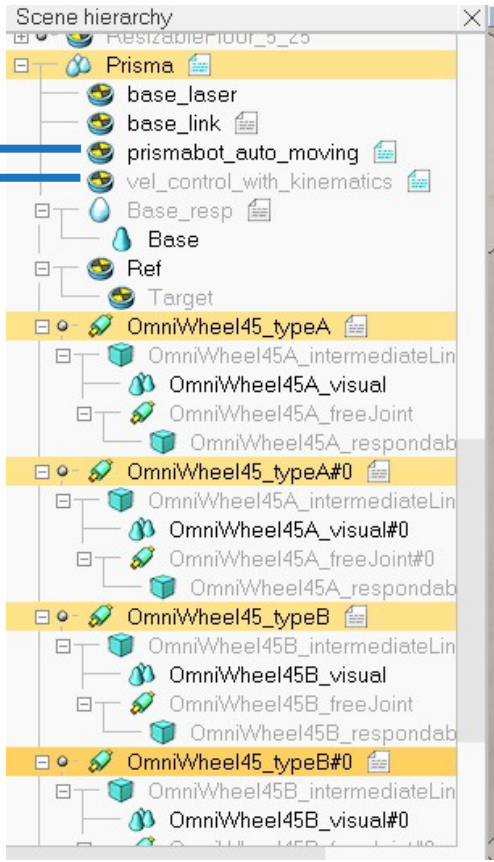



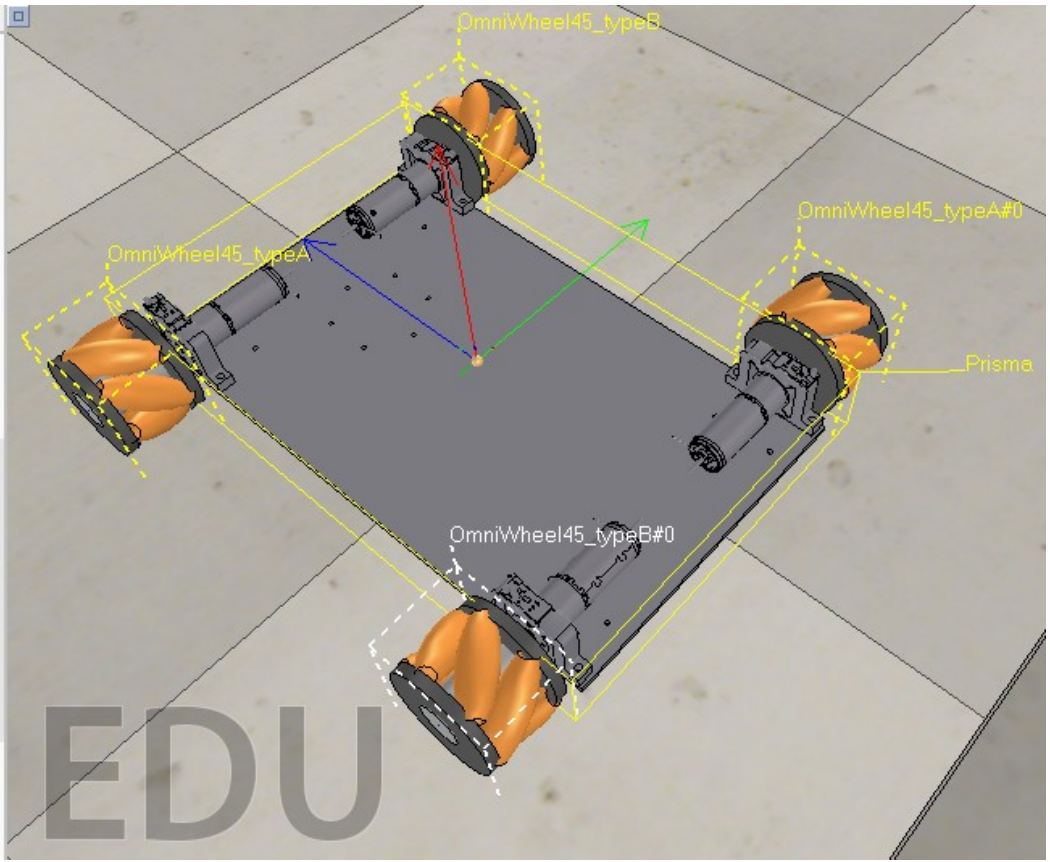
# Introduction to ROS and V-REP
## V-REP: scripts

CoppeliaSim GUI

Autonomous movement script

Velocities from ROS script



# Introduction to ROS and V-REP
## V-REP: scripts

### Script GUI (Lua)

CoppeliaSim provides a built-in editor for scripts, just double-click on a script icon to open it.

When a new script is created with the GUI (right-click -> add child script) the template of a script is given by default

Threaded child script (vel_control_with_kinematics)

```
1
2  function prismabot_fs_callback(msg)
3      sim.setFloatSignal('prismabot_fs_signal',msg.data)
4      sim.addStatusbarMessage('fs_cb: '..msg.data)
5  end
6
7  function prismabot_ls_callback(msg)
8      sim.setFloatSignal('prismabot_ls_signal',msg.data)
9      sim.addStatusbarMessage('ls_cb: '..msg.data)
10 end
11
12 function prismabot_ts_callback(msg)
13     sim.setFloatSignal('prismabot_ts_signal',msg.data)
14     sim.addStatusbarMessage('ts_cb: '..msg.data)
15 end
16
17 -- thread start
18 function sysCall_threadmain()
19 sim.setThreadSwitchTiming(2) -- Default timing for automatic thread switching
20
21 wheelJoints={-1,-1,-1,-1} -- front left, rear left, rear right, front right
22 wheelJoints[1]=sim.getObjectHandle('OmniWheel45_typeA')
23 wheelJoints[2]=sim.getObjectHandle('OmniWheel45_typeB#0')
24 wheelJoints[3]=sim.getObjectHandle('OmniWheel45_typeA#0')
25 wheelJoints[4]=sim.getObjectHandle('OmniWheel45_typeB')
26 --Prismabot dimensions in meters
27 robot_semi_len = 0.2
28 robot_semi_wdt = 0.2
29 wheel_ray      = 0.05
30
31 subscriber1ID=simROS.subscribe('/prismabot/fs', 'std_msgs/Float32', 'prismabot_fs_callback')
32 simROS.subscriberTreatUInt8ArrayAsString(subscriber1ID)
33
```

# Introduction to ROS and V-REP
## V-REP: scripts

Velocities from ROS

3 callbacks

```
function prismabot_fs_callback(msg)
    sim.setFloatSignal('prismabot_fs_signal',msg.data)
    sim.addStatusbarMessage('fs_cb: '..msg.data)
end

function prismabot_ls_callback(msg)
    sim.setFloatSignal('prismabot_ls_signal',msg.data)
    sim.addStatusbarMessage('ls_cb: '..msg.data)
end

function prismabot_ts_callback(msg)
    sim.setFloatSignal('prismabot_ts_signal',msg.data)
    sim.addStatusbarMessage('ts_cb: '..msg.data)
end
```

# Introduction to ROS and V-REP
## V-REP: scripts

Velocities from ROS

3 subscribers

```
17  -- thread start
18  function sysCall_threadmain()
19  sim.setThreadSwitchTiming(2) -- Default timing for automatic thread switching
20
21  wheelJoints={-1,-1,-1,-1} -- front left, rear left, rear right, front right
22  wheelJoints[1]=sim.getObjectHandle('OmniWheel45_typeA')
23  wheelJoints[2]=sim.getObjectHandle('OmniWheel45_typeB#0')
24  wheelJoints[3]=sim.getObjectHandle('OmniWheel45_typeA#0')
25  wheelJoints[4]=sim.getObjectHandle('OmniWheel45_typeB')
26  --Prismabot dimensions in meters
27  robot_semi_len = 0.2
28  robot_semi_wdt = 0.2
29  wheel_ray      = 0.05
30
31  subscriber1ID=simROS.subscribe('/prismabot/fs', 'std_msgs/Float32', 'prismabot_fs_callback')
32  simROS.subscriberTreatUInt8ArrayAsString(subscriber1ID)
33
34  subscriber2ID=simROS.subscribe('/prismabot/ls', 'std_msgs/Float32', 'prismabot_ls_callback')
35  simROS.subscriberTreatUInt8ArrayAsString(subscriber2ID)
36
37  subscriber3ID=simROS.subscribe('/prismabot/ts', 'std_msgs/Float32', 'prismabot_ts_callback')
38  simROS.subscriberTreatUInt8ArrayAsString(subscriber3ID)
```

# Introduction to ROS and V-REP
## V-REP: scripts

Velocities from ROS

Get 3 velocities

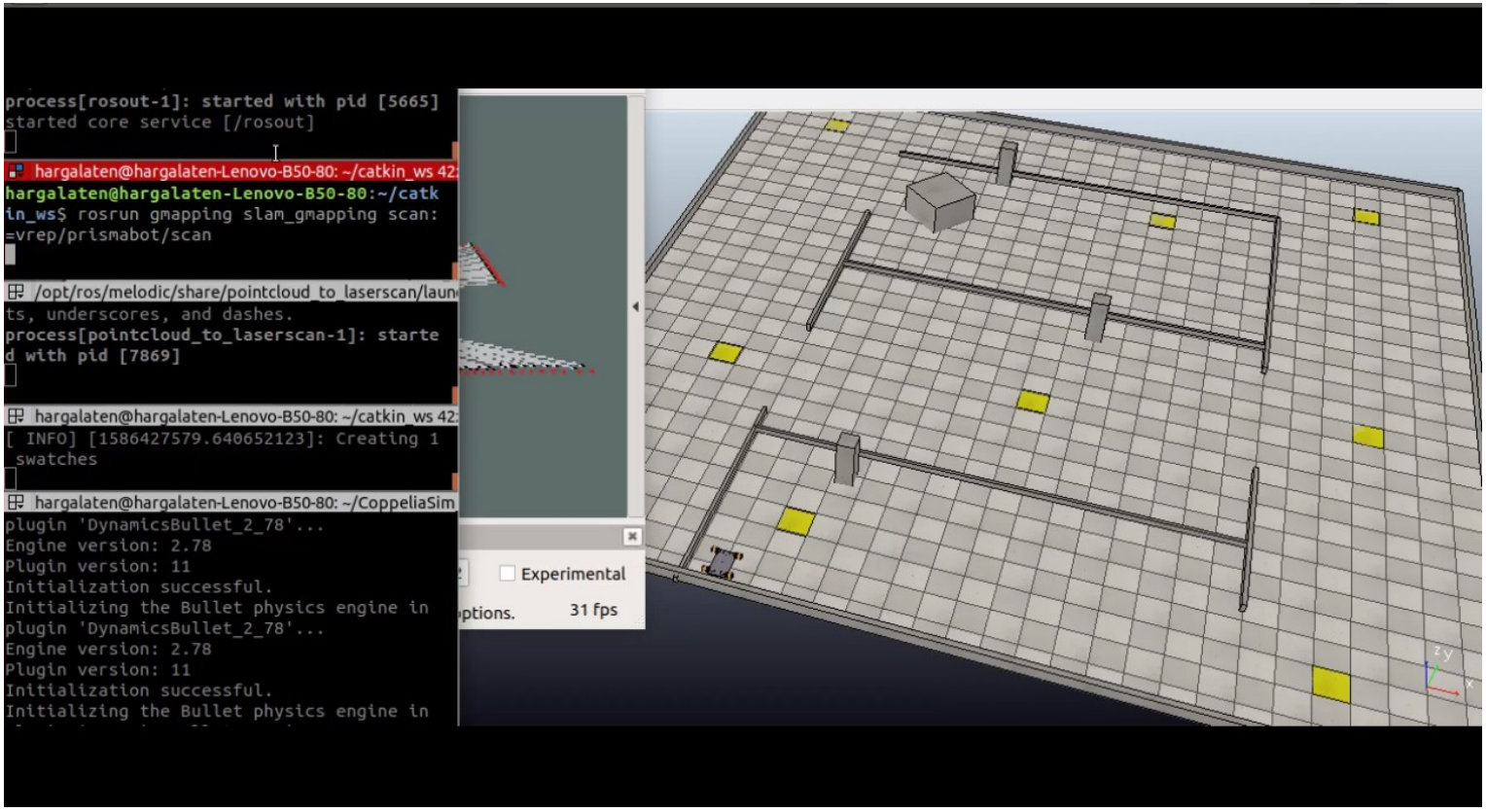
Get 4 wheels velocities

Set to joints

```
52 while(true) do
53     sim.waitForSignal('prismabot_fs_signal')
54     fs=sim.getFloatSignal('prismabot_fs_signal')--*2
55     sim.waitForSignal('prismabot_ls_signal')
56     ls=sim.getFloatSignal('prismabot_ls_signal')--*2
57     sim.waitForSignal('prismabot_ts_signal')
58     ts=sim.getFloatSignal('prismabot_ts_signal')--*2
59
60     --Prismabot-base kinematics
61     fs_motor = fs / wheel_ray
62     ts_motor = ( ( robot_semi_len + robot_semi_wdt ) * ts ) / wheel_ray
63     ls_motor = ls / wheel_ray
64
65     bl_vel = fs_motor - ts_motor + ls_motor
66     br_vel = fs_motor + ts_motor - ls_motor
67     fl_vel = fs_motor - ts_motor - ls_motor
68     fr_vel = fs_motor + ts_motor + ls_motor
69
70     sim.setJointTargetVelocity(wheelJoints[1],fl_vel)
71     sim.setJointTargetVelocity(wheelJoints[2],bl_vel)
72     sim.setJointTargetVelocity(wheelJoints[3],br_vel)
73     sim.setJointTargetVelocity(wheelJoints[4],fr_vel)
74
75
76     sim.clearFloatSignal('prismabot_fs_signal')
77     sim.clearFloatSignal('prismabot_ls_signal')
78     sim.clearFloatSignal('prismabot_ts_signal')
79
80     end
81 end
```

# Introduction to ROS and V-REP
## Running example

- roscore
- gmapping (SLAM)
- pointcloud to laserscan
- rviz
- coppeliaSim



# Introduction to ROS and V-REP
## Running example

Video
[outsourced]