Robotics Lab: Homework 1

Building your robot manipulator

Mario Selvaggio

This document contains the homwework 1 of the Robotics Lab class.

Building your robot manipulator

The goal of this homework is to build ROS packages to simulate a 4-degrees-of-freedom robotic manipulator arm into the Gazebo environment. The student is requested to address the following points and provide a detailed report of the employed methods. In addition, a personal github repo with all the developed code must be shared with the instuctor. The report is due in one week from the homewerk release.

1. Create the description of your robot and visualize it in Rviz
   (a) Download the `arm_description` package from the repo https://github.com/RoboticsLab2023/arm_description.git into your `catkin_ws` using `git` commands
   (b) Within the package create a `launch` folder containing a launch file named `display.launch` that loads the URDF as a `robot_description` ROS param and starts the `robot_state_publisher` node, the `joint_state_publisher` node, and the `rviz` node. Launch the file using `roslaunch`. **Note:** To visualize your robot in rviz you have to changhe the Fixed Frame in the lateral bar and add the `RobotModel` plugin interface. **Optional:** save a `.rviz` configuration file, thad automatically loads the `RobotModel` plugin by default, and give it as an argument to your node in the `display.launch` file
   (c) Substitute the collision meshes of your URDF with primitive shapes. Use `<box>` geometries of reasonabe size approximating the links. **Hint:** Enable collision visualization in rviz (go to the lateral bar > Robot model > Collision Enabled) to adjust the collision meshes size
   (d) Create a file named `arm.gazebo.xacro` within your package, define a `xacro:macro` inside your file containing all the `<gazebo>` tags you find within your `arm.urdf` and import it in your URDF using `xacro:include`. Remember to rename your URDF file to `arm.urdf.xacro`, add the string `xmlns:xacro="http://www.ros.org/wiki/xacro"` within the `<robot>` tag, and load the URDF in your launch file using the `xacro` routine
2. Add transmission and controllers to your robot and spawn it in Gazebo
   (a) Create a package named `arm_gazebo`
   (b) Within this package create a `launch` folder containing a `arm_world.launch` file
   (c) Fill this launch file with commands that load the URDF into the ROS Parameter Server and spawn your robot using the `spawn_model` node. **Hint:** follow the `iiwa_world.launch` example from the package `iiwa_stack`: https://github.com/IFL-CAMP/iiwa_stack/tree/master. Launch the `arm_world.launch` file to visualize the robot in Gazebo
   (d) Now add a `PositionJointInterface` as hardware interface to your robot: create a `arm.transmission.xacro` file into your `arm_description/urdf` folder containing a `xacro:macro` with the hardware interface and load it into your `arm.urdf.xacro` file using `xacro:include`. Launch the file
   (e) Add joint position controllers to your robot: create a `arm_control` package with a `arm_control.launch` file inside its `launch` folder and a `arm_control.yaml` file within its `config` folder
   (f) Fill the `arm_control.launch` file with commands that load the joint controller configurations from the `.yaml` file to the parameter server and spawn the controllers using the `controller_manager` package. **Hint:** follow the `iiwa_control.launch` example from corresponding package

(g) Fill the arm `arm_control.yaml` adding a `joint_state_controller` and a `JointPositionController` to all the joints

(h) Create an `arm_gazebo.launch` file into the `launch` folder of the `arm_gazebo` package loading the Gazebo world with `arm_world.launch` and spawning the controllers within `arm_control.launch`. Go to the `arm_description` package and add the `gazebo_ros_control` plugin to your main URDF into the `arm.gazebo.xacro` file. Launch the simulation and check if your controllers are correctly loaded

3. Add a camera sensor to your robot

   (a) Go into your `arm.urdf.xacro` file and add a `camera_link` and a fixed `camera_joint` with `base_link` as a parent link. Size and position the camera link opportunely

   (b) In the `arm.gazebo.xacro` add the gazebo sensor reference tags and the `libgazebo_ros_camera` plugin to your xacro (slide 74-75)

   (c) Launch the Gazebo simulation with using `arm_gazebo.launch` and check if the image topic is correctly published using `rqt_image_view`

   (d) **Optionally:** You can create a `camera.xacro` file (or download one from https://github.com/CentroEPiaggio/irobotcreate2ros/blob/master/model/camera.urdf.xacro) and add it to your robot URDF using `<xacro:include>`

4. Create a ROS publisher node that reads the joint state and sends joint position commands to your robot

   (a) Create an `arm_controller` package with a ROS C++ node named `arm_controller_node`. The dependencies are `roscpp`, `sensor_msgs` and `std_msgs`. Modify opportunely the `CMakeLists.txt` file to compile your node. **Hint:** uncomment `add_executable` and `target_link_libraries` lines

   (b) Create a subscriber to the topic `joint_states` and a callback function that prints the current joint positions (see Slide 45). **Note:** the topic contains a `sensor_msgs/JointState`

   (c) Create publishers that write commands onto the controllers' `/command` topics (see Slide 46). **Note:** the command is a `std_msgs/Float64`