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/RoboticsLab2024/arm_description.git into your `ros2_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 `rviz2` node. Launch the file using `ros2 launch`. **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
2. Add sensors 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 `/robot_description` topic and spawn your robot using the `create` node in the `ros_gz_sim` package.
     Hint: follow the `iiwa.launch.py` example from the package `iiwa_ros2`: https://github.com/ICube-Robotics/iiwa_ros2/tree/main. Launch the `arm_world.launch` file to visualize the robot in Gazebo.
   - (d) Add a `PositionJointInterface` as a hardware interface to your robot using `ros2_control`. Create an `arm_hardware_interface.xacro` file in the `arm_description/urdf` folder, containing a macro that defines the hardware interface for the joint, and include it in your main `arm.urdf.xacro` file using `xacro:include`. Specifically, define the joint using `ros2_control` and specify the hardware interface as `PositionJointInterface`.
     Hint: 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
   - (e) Add inside the `arm.urdf.xacro` the commands to load the joint controller configurations from the `.yaml` file and spawn the controllers using the `controller_manager` package. Then, launch the robot simulation in Gazebo and demonstrate how the hardware interface is correctly loaded and connected.
   - (f) 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.

  (g) Fill the arm `arm_control.yaml` adding a `joint_state_bradcaster` 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`. 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) Create an `arm_camera.xacro` file in the `arm_gazebo/urdf` folder, add the gazebo sensor reference tags and the `gz-sim-sensors-system` plugin to your xacro.
  Hint: define a `xacro:macro` inside your `arm_camera.xacro` file containing the `<gazebo>` tag and import it in `arm.urdf.xacro` using `xacro:include`.
  (c) Launch the Gazebo simulation with using `arm_gazebo.launch`, and check if the image topic is correctly published using `rqt_image_view`.
  Hint: remember to add the `ros_ign_bridge`.
  (d) **Optionally:** You can create a `camera.xacro` file 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) Inside the `arm_controller` package vreate a ROS C++ node named `arm_controller_node`. The dependencies are `rclcpp`, `sensor_msgs` and `std_msgs`. Modify opportunely the `CMakeLists.txt` file to compile your node. **Hint:** use `add_executable` and `ament_target_dependencies` commands
  (b) Create a subscriber to the topic `joint_states` and a callback function that prints the current joint positions. **Note:** the topic contains a `sensor_msgs/JointState`
  (c) Create publishers that write commands onto the `/position_controller /command` topics. **Note:** the command is a `std_msgs/msg/Float64MultiArray`