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

Bring up your robot

The goal of this homework is to build ROS packages to simulate a 4-degrees-of-freedom robotic manipulator arm (Armando) into the Gazebo environment. The student is requested to address the following problems and provide a detailed point-to-point solution report. A personal github repo containing all the code and a README file detailing how to build and run it must be shared with the instructor (including the URL in the report). The report is due in one week from the homework release.

As a start point, download the `armando_description` package from this repo into your `ros2_ws`. Then,

1. Modify the URDF description of your robot and visualize it in Rviz.

   (a) Create a `launch` folder within the `armando_description` package containing a launch file named `armando_display.launch` that loads the URDF as a `robot_description` ROS param, 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 change the Fixed Frame in the lateral bar and add the `RobotModel` display

   (b) Create a config folder and save a `.rviz` configuration file within, that automatically loads the `RobotModel` display, and pass it as an argument to your node in the `armando_display.launch` file

   (c) Substitute the collision meshes of your URDF with primitive shapes. Use `<box>` geometries to approximate the bounding box of the links. **Hint:** Enable collision visualization in rviz (go to the lateral bar > Robot model > Collision Enabled) to adjust the collision meshes size to match (approximately) to the bounding box of the visual meshes

2. Add sensors and controllers to your robot and spawn it in Gazebo

   (a) Create a package named `armando_gazebo` using `ros2` CLI. Within this package, create a `launch` folder containing a `armando_world.launch` file and fill it 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 `gazebo.launch.py` from the `ros2_urdf` tutorial package here. Launch the `armando_world.launch` file to start the simulation of your robot in Gazebo

   (b) Add a `PositionJointInterface` as a hardware interface to your robot using the `ros2_control` framework. Create an `armando_hardware_interface.xacro` file in the `armando_description/urdf` folder, containing a macro that defines the hardware interface for the joints of your robot, and include it in your main `armando.urdf.xacro` file using `xacro:include`. **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 as shown in here

   (c) Add inside the `armando.urdf.xacro` the commands to enable the Gazebo ROS2 control plugin and load the joint position controllers from a `.yaml` file. Then, spawn the joint state broadcaster and the position controllers using the `controller_manager` package from the `armando_world.launch`. Launch the Gazebo robot simulation and demonstrate how the hardware interface is correctly loaded and connected. **Hint:** use the RegisterEventHandler function to load controllers after gazebo is started

3. Add a camera sensor to your robot

(a) Go into your `armando.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 at the base of your robot

(b) Create an `armando_camera.xacro` file in the `armando_gazebo/urdf` folder, add the sensor specifications within a `xacro:macro` and the `gz-sim-sensors-system` plugin. Import it in `armando.urdf.xacro` using the `xacro:include` command

(c) Launch the Gazebo simulation using `armando_gazebo.launch`, and check if the image topic is correctly published using `rqt_image_view`. **Hint:** remember to add the correct `ros_ign_bridge` commands into the launch file

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

(a) Create the `armando_controller` package with a ROS C++ node named `arm_controller_node`. The dependencies are `rclcpp`, `sensor_msgs` and `std_msgs`. Modify opportunely the `CMakeLists.txt` and the `package.xml` files to compile your node. **Hint:** adjust the `add_executable` and `ament_target_dependencies` commands

(b) Within the node, create a subscriber to the topic `joint_states` and a callback function that prints the current joint positions of the robot. **Note:** the topic contains a `sensor_msgs/JointState`

(c) Create a publisher that writes sequentially at least four different position commands onto the `/position_controller/command` topics. **Note:** the command is a `std_msgs/msg/Float64MultiArray`

(d) Create a joint trajectory publisher that sends the same positions commands and make it work by loading the corresponding controllers. Allow the user to select which publisher/controllers to use (position or trajectory) by adding ROS arguments to your node and to your launch files