Lab 4 Report: ROS 2 PX100 Arm Simulation

Tytus Felbor

January 11, 2026

1 Open a simulated robot arm in Gazebo

The lab process began by making sure that all ROS 2 Humble dependencies, such as `ros-humble-ros2-control` and `ros-humble-gazebo-ros2-control`, were installed. The Gazebo Fortress installation was verified using `ign gazebo --versions` to ensure compatibility with the provided robot package.

1.1 Load your robot on Gazebo

The robot description was loaded using the `px100_gazebo.urdf` file. A critical step in this section was updating line 354 of the URDF, which contained a hardcoded path to the controller plugin's parameter file px100_controllers.yaml. I modified this to match our local workspace directory to allow the controller_manager to initialize the joints. The simulation was then launched using `ros2 launch arm_lessons px100_gazebo.launch.py` and verified that the `joint_state_broadcaster` and controllers were "Configured and activated."

1.2 Create an object

Using the Gazebo GUI, a basic cube was inserted into the environment. Then I proceeded to save the world as my_world_square.sdf.

1.2.1 SDF File

The SDF file was manually edited to set the box size to $0.03m \times 0.03m \times 0.03m$. The values in the `<visual>` and `<collision>` tags were ensured to match in order to maintain physical accuracy. To simplify the pick-and-place task, the box was placed directly in front of the arm at `pose 0.28 0.0 0.025`.

2 Control the arm in a node

The goal was to define joint positions and send them to the trajectory controller.

2.1 Analyse the topics available

During simulation, the following topics were analyzed:

- **Topics:** `/arm_controller/joint_trajectory` and `/gripper_controller/commands`.
- **Message Types:** The arm uses `JointTrajectory` (containing joint names and points), while the gripper requires a `Float64MultiArray`.
- **Minimum Value:** It was determined that the minimum velocity to move the gripper is 0.5. Values lower than this cannot overcome the simulated friction and damping in the joint dynamics.

3 Create a program to control the arm

A python script `pick_and_place.py` was built and `CMakeLists.txt` was updated to install it.

3.1 Technical Challenges and Solutions

- **Physics Friction:** The cube initially slid out of the gripper. This was solved this by adding `<gazebo>` tags to the URDF to set high friction (`mu1`, `mu2`) and contact stiffness (`kp`) for the finger links.
- **Non-Blocking Logic:** Instead of using `time.sleep()` a timer-based state machine was incorporated to prevent blocking the ROS 2 executor, allowing continuous communication with Gazebo.

3.2 Code Snippet: State Machine

```
    def execute_step(self):
        if self.step == 1:
            self.send_arm([0.0, 1.65, 0.0, 0.0]) # Reach
            self.step = 2
        elif self.step == 2:
            self.send_gripper([-2, -2]) # Grasp
            self.step = 3
        elif self.step == 3:
            self.send_arm([1.0, 1.5, -0.75, 0.75]) # Move
            self.step = 4
```

4 Results

The following screenshots document the successful execution of the pick-and-place task, showing the transition from the starting position to the successful placement at a new location.

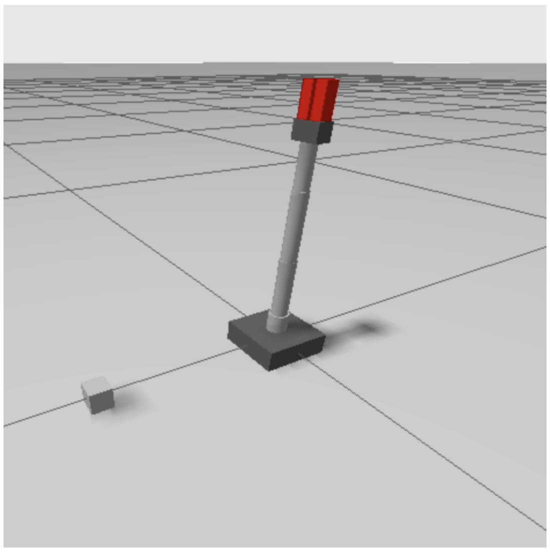

Figure 1: Initial starting position of the PX100 arm.

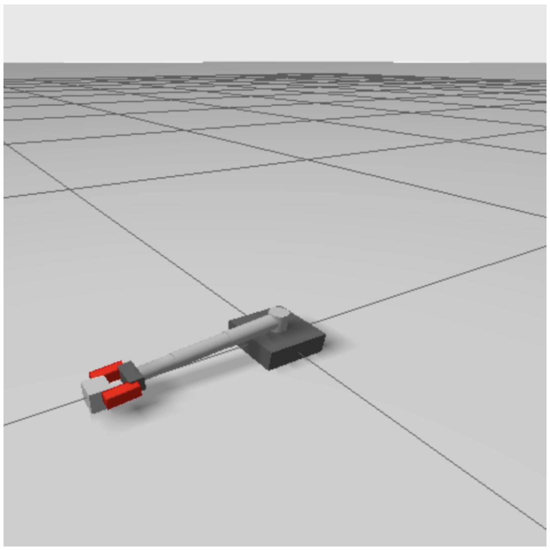

Figure 2: Successful grasp of the 0.03m cube at maximum reach.

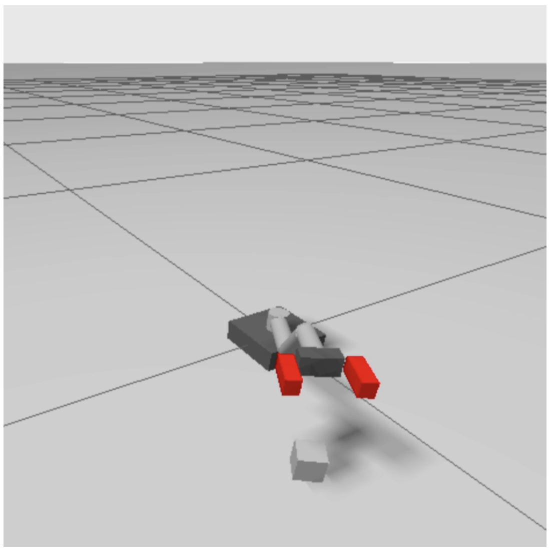

Figure 3: Final state: cube released at the new coordinate.