Tytus Felbor
Andrea Infanti
ROS & Gazebo : Lab 3 Report

Introduction

The objective of this lab session was to introduce the Interbotix PX100 robotic arm and its corresponding ROS 2 packages along with their functionality. We were taught how to control the arm using both high-level interfaces - *MoveIt*, as well as, low-level ROS 2 nodes. The lab's ultimate goal was to develop a C++ node capable of operating the arm to pick and place an object by controlling the joint groups and the gripper using ROS topics.

Software Setup and MoveIt Testing

Before implementing the control nodes, our software environment was verified using the lab specified demo packages:

- MoveIt Interface:
    - Called *interbotix_xsarm_moveit_interface*, which launched to visualize the robot in RViz.
- Functionality Check:
    - We have managed to successfully plan and execute a trajectory to the "upright" goal state using the *MoveIt* Planning tab.

Joint State Analysis

To program the arm effectively, We first needed to understand how joint positions are encoded.

- Procedure:
    - We launched the *interbotix_xsarm_ros_control* package and disabled the torque on the motors using the */px100/torque_enable* service. This allowed for manual manipulation of the arm.
- Observation:
    - By subscribing to the topic */px100/joint_states*, we managed to observe the real-time angular positions (in radians) of the arm's four joints: *waist*, *shoulder*, *elbow*, and *wrist_angle*.

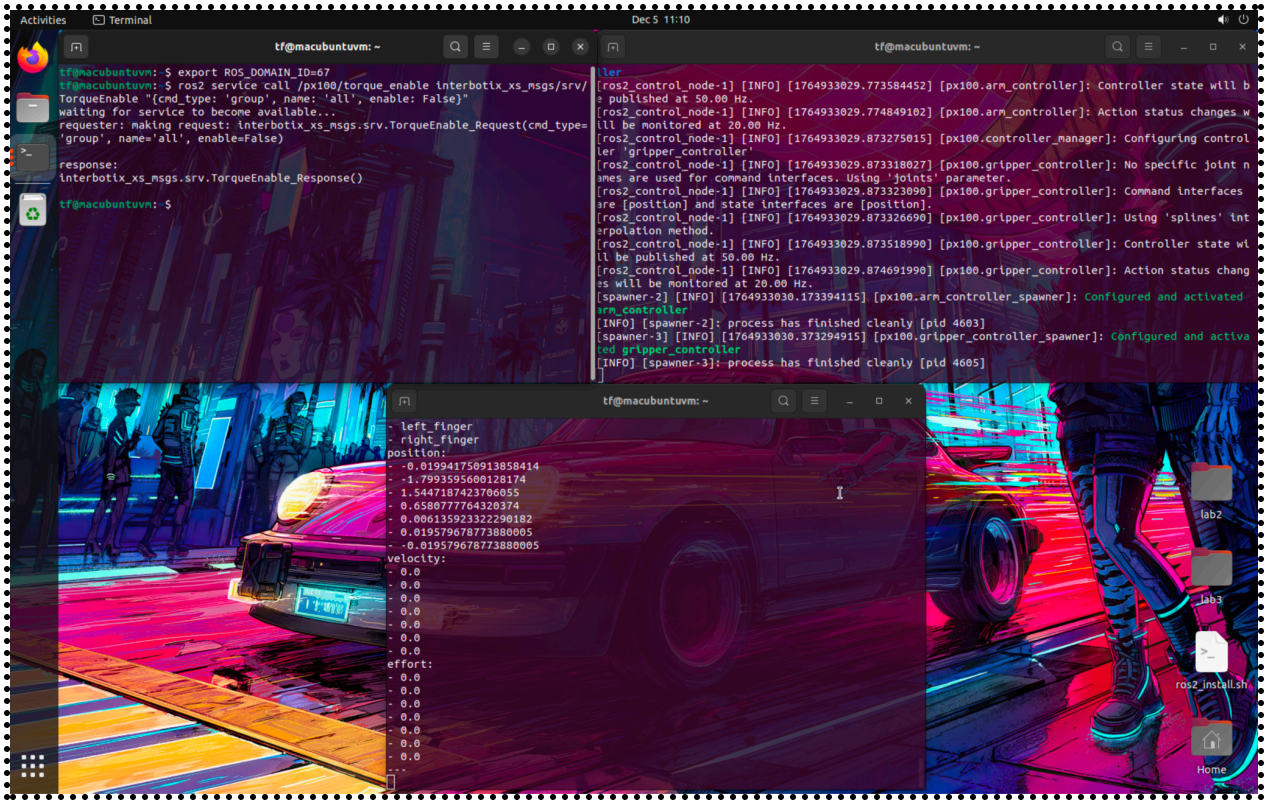

Figure 1: Real-time joint state values observed via ros2 topic echo.

Programming the Control Node

The main task was to create a ROS 2 package, which we called *my_arm_cmd*, to programmatically control the arm and gripper.

Gripper Configuration (PWM Mode)

A critical step was configuring the gripper to operate in PWM (Pulse Width Modulation) mode rather than the default Position mode, which allowed the gripper to apply force to grab objects.

- Modification:
    - We edited the *modes.yaml* file in the *interbotix_xsarm_ros_control* package to *operating_mode: pwm*.
- *Deployment*:
    - The control package was rebuilt (using *colcon build*) to ensure the robot driver loaded and acknowledged the new configuration.

C++ Implementation

A C++ node called *move_arm_node* was built to publish commands to two specific topics:

1. Arm Joints:
    a. */px100/commands/joint_group* which was of type *JointGroupCommand*.
2. Gripper:
    a. */px100/commands/joint_single* which was of type *JointSingleCommand*.

The code implements a timer loop that alternates the robot between two specified states:

- Position 1 - Pick:
    - The arm rotates to the left and leans forward; then the gripper was supposed to apply closing torque with *PWM* set to -300.
- Position 2 - Place:
    - The arm rotates to the right and ascends; the gripper was meant to apply opening torque with *PWM* set +300.
- Side Note:
    - Even though the gripper command and *PWM* were invoked, the gripper's functionality failed. This might have been due to a manufacturing error or broken gripper.

Code Snippet (Main Logic):

C++

```
 void timer_callback()
 {
   auto arm_msg = interbotix_xs_msgs::msg::JointGroupCommand();
   auto gripper_msg = interbotix_xs_msgs::msg::JointSingleCommand();

   arm_msg.name = "arm";
   gripper_msg.name = "gripper";

   if (step_ % 2 == 0) {
     // --- POSITION 1: Left & Extended ---
     // Waist: 1.0 rad (Turn Left)
     // Shoulder: -0.5 rad (Lean forward)
     // Elbow: 0.5 rad (Flex up)
```

```
      // Wrist: 0.0 rad (Neutral)
      arm_msg.cmd = {1.0, -0.5, 0.5, 0.0};

      // Close Gripper (PWM Mode required!)
      gripper_msg.cmd = -300.0;

      RCLCPP_INFO(this->get_logger(), "Pos 1: Left + Grip Close");
    } else {
      // --- POSITION 2: Right & Retracted ---
      // Waist: -1.0 rad (Turn Right)
      // Shoulder: 0.0 rad (Upright)
      // Elbow: 0.0 rad (Straight)
      // Wrist: 0.0 rad (Neutral)
      arm_msg.cmd = {-1.0, 0.0, 0.0, 0.0};

      // Open Gripper
      gripper_msg.cmd = 300.0;

      RCLCPP_INFO(this->get_logger(), "Pos 2: Right + Grip Open");
    }

    publisher_arm_->publish(arm_msg);
    publisher_gripper_->publish(gripper_msg);

    step_++;
  }
```

Build Configuration

The CMakeLists.txt file was modified in order to generate the executable and link the required dependencies - *rclcpp*, *interbotix_xs_msgs*.

Results

The implementation presented several challenges that were resolved during the lab:

1. Compilation Errors: Initial builds failed due to a few syntax errors as well as missing executable definitions in CMakeLists.txt. These were resolved by refining the code and adding the missing *add_executable* instructions.

2. Gripper Responsiveness: The gripper did not respond to *PWM* commands. I've tried rebuilding the *interbotix_xsarm_ros_control* package and restarting the launch file which did not bring any improvement which reinforced us in the belief that hardware was the core issue.

Conclusion

The *move_arm_node* successfully controlled the PX100 robot's joints, alternating between the orchestrated pick and place locations. This lab emphasized the importance of correct message types and driver configurations when using ROS 2 packages.