ROBT 611. Industrial Robotics

Final Project

Students:

Soibkhon Khajikhanov, Yermakhan Kassym

Introduction

Our project was about implementing a “Moveit!2” on Franka robot with Robotiq 2F-85 gripper attached as an end effector to perform pick-and-place operations. The setup can be seen in the Figure 1 below.

Figure 1. The setup in Tactile Lab used for the final project

The goal was first to implement the setup in a simulation and then convert to the real hardware, which was located in the “Tactile Lab”. The gripper was attached to the manipulator via a custom connector, thus we needed to modify the original URDF files of the Franka to properly attach the gripper. Moreover, we had to make our own “Moveit” package, since the available package was only for a Franka with a Hand gripper. Our system was already working on ROS1 before but we wanted it to work with ROS2.

Methodology/Implementation

Initially we thought that problems may occur with the Robotiq 2F-85 gripper and the Franka FER since the last official package for it was released for ROS Melodic and Noetic respectively. However, there were unofficial packages for it for later ROS.

In order to connect the Robotiq 2F-85 gripper with the custom mount, the settings in the Franka FCI had to be changed as follows:

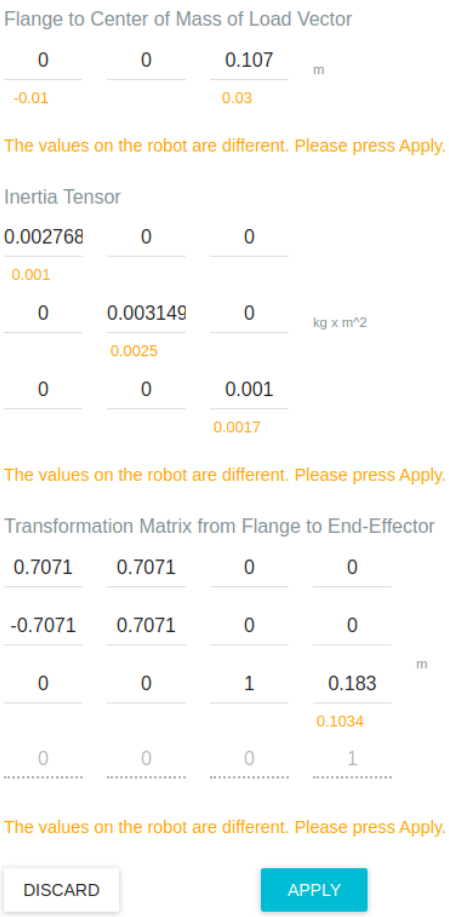

Figure 2. Franka FCI updated parameters

The base numbers were taken from the gripper manual and the measurements of custom mount were added; the weight was calculated manually. Inertia matrix was taken from the manual.

The mount is 0.05m in length and it was added to Z-axis of the Center of mass vector. Also, the distance between the mounting point of end-effector and the gripping point center was measured and added to the translational matrix. The end-effector is rotated about -45 degrees, which was also taken into account in the translational matrix.


Figure 3. Custom connector mount measurements

We needed to have a real-time control in our system for fast and responsive communication between the computer and the robot, as the name suggests. For that we ran the following commands inside our catkin workspace:

```
robot@robot:~/catkin_ws$ sudo usermod -aG realtime robot
robot@robot:~/catkin_ws$ sudo groupadd realtime
```

Furthermore, we updated the "90-proc.conf" file – a configuration file typically used to fine-tune the behavior of real-time processes managed by the robot's control system. The following changes were introduced to the file:

```
robot soft rtprio 99
robot hard rtprio 99
robot soft memlock unlimited
robot hard memlock unlimited
```

Figure 4. Changes made in "90-proc.conf" file

From the Figure 2 we can observe that real-time priorities (*rtprio*) for scheduling were given the highest values (*99*), which ensure that tasks related to the robot have precedence over other processes in the operating system. This is critical for maintaining the low-latency control loops necessary for real-time robot operation. Furthermore, setting memory locking (*memlock*) to *unlimited* allows the process to lock all the memory it requires without restriction. It prevents critical robot processes from experiencing delays due to memory swapping (paging). Swapping can cause significant latency, which is unacceptable for real-time control. This ensures stability and predictable performance for the robot's control system.

There were 3 packages of the franka_ros available online for the Franka FER on ROS2. First, we tried one from the following link: https://github.com/mcbed/franka_ros2/tree/humble. However, even "move_to_start" script did not work. Then, we tried this one: https://github.com/yilmazabdurrah/multi_franka_arm_ros2. This package also was not good for our system. At last, we tried one from: https://github.com/LCAS/franka_arm_ros2/tree/humble.

The last package had not been tested with Franka FCI 4.1.2, which is the version we were using, and the 0.8.0 version of the libfranka. Additionally, we had to modify the files to align with our specific Franka setup. The original launch files and URDF included configurations for a camera, which was not part of our system, requiring further adjustments. Furthermore, we incorporated a MoveIt! package we had previously generated for Franka on ROS Noetic, using its files along with the downloaded package to achieve functionality.

We used following ROS controller that was written for Noetic system but can work with Humble too:

```
# Simulation settings for using moveit_sim_controllers
moveit_sim_hw_interface:
  joint_model_group: panda_arm
  joint_model_group_pose: start_pose
# Settings for ros_control_boilerplate control loop
generic_hw_control_loop:
  loop_hz: 300
  cycle_time_error_threshold: 0.01
# Settings for ros_control hardware interface
hardware_interface:
  joints:
    - panda_joint1
    - panda_joint2
    - panda_joint3
```

```
    - panda_joint4
    - panda_joint5
    - panda_joint6
    - panda_joint7
    - finger_joint
  sim_control_mode: 1  # 0: position, 1: velocity
# Publish all joint states
# Creates the /joint_states topic necessary in ROS
joint_state_controller:
  type: joint_state_controller/JointStateController
  publish_rate: 50
controller_list:
  []
arm_position_controller:
  type: position_controllers/JointPositionController
  joints:
    - panda_joint1
    - panda_joint2
    - panda_joint3
    - panda_joint4
    - panda_joint5
    - panda_joint6
    - panda_joint7
  gains:
    panda_joint1:
      p: 100
      d: 1
      i: 1
      i_clamp: 1
    panda_joint2:
      p: 100
      d: 1
      i: 1
      i_clamp: 1
    panda_joint3:
      p: 100
      d: 1
      i: 1
      i_clamp: 1
    panda_joint4:
      p: 100
      d: 1
      i: 1
      i_clamp: 1
    panda_joint5:
      p: 100
      d: 1
      i: 1
      i_clamp: 1
    panda_joint6:
      p: 100
      d: 1
      i: 1
      i_clamp: 1
    panda_joint7:
      p: 100
      d: 1
      i: 1
      i_clamp: 1
```

Also this package had the camera in its setup so we modified to launch file by deleting the camera setup:

```
return LaunchDescription(
    [robot_arg,
     use_fake_hardware_arg,
     fake_sensor_commands_arg,
     load_gripper_arg,
     planner_arg,
     db_arg,
     rviz_node,
     run_move_group_node,
     robot_state_publisher,
     ros2_control_node,
     mongodb_server_node,
     joint_state_publisher,
     gripper_launch_file,
     *load_controllers])
```

Figure 5. Modifications made into the launch file of the package

We also included the URDF of the gripper for the RViz and SRDF for the MoveIt! so that robot could plan its way properly and account for external the gripper:

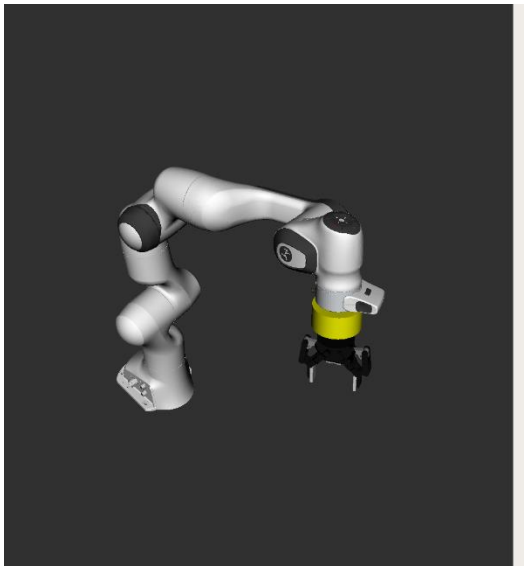

Figure 6. Additions to the robot model for planning and simulation

As for the gripper, it is not core ROS2 supported, thus we implemented the following package: https://github.com/PickNikRobotics/ros2_robotiq_gripper

However we faced error from CMake about serial, and could be installed with apt-get ros-humble-serial, because it was not available for the humble version of the ROS. We installed it using following lines:
git clone -b humble https://github.com/PickNikRobotics/ros2_robotiq_gripper.git src/<name-of-the-folder>
vcs import src --skip-existing --input src/<name-of-the-folder>/ros2_robotiq_gripper-not-released.humble.repos

Now gripper worked and this was our simple code to control it:

```
#!/usr/bin/env python3
```

```
import rclpy
from rclpy.action import ActionClient
from rclpy.node import Node
from control_msgs.action import GripperCommand
from time import sleep

class GripperControl(Node):
def __init__(self):
super().__init__('gripper_control_node')
# Connect to the gripper action server on '/robotiq_gripper_controller/gripper_cmd'
self._action_client = ActionClient(self, GripperCommand, '/robotiq_gripper_controller/gripper_cmd')
self.get_logger().info('Gripper control node started')

def send_gripper_command(self, position: float):
# Wait for the action server to be available
while not self._action_client.wait_for_server(timeout_sec=1.0):
self.get_logger().info('Waiting for action server to be available...')
# Prepare the goal message with the gripper's position command
goal_msg = GripperCommand.Goal()
goal_msg.command.position = position # 0.0 for open, 1.0 for closed
goal_msg.command.max_effort = 50.0 # Adjust based on your gripper's need

# Send the goal to the action server
self.get_logger().info(f'Sending gripper command: {position}')
self._action_client.send_goal_async(goal_msg)

def main(args=None):
rclpy.init(args=args)

gripper_control = GripperControl()

try:
# Close the gripper
gripper_control.send_gripper_command(0.0) # 0.0 for open, 1.0 for closed
sleep(3) # Wait for 3 seconds
# Open the gripper
gripper_control.send_gripper_command(1.0)
except KeyboardInterrupt:
pass

gripper_control.destroy_node()
rclpy.shutdown()

if __name__ == '__main__':
main()
```

Results & Conclusion

Moveit2 package works fine with our setup and can be used to control the robot, and the gripper also works fine with ROS2 humble. While both packages for Franka and the Robotiq gripper work fine separately, when we tried to combine them together, an incompatibility error arose. Despite this, the goal of making the Franka FER robot and Robotiq 2F-85 gripper work on ROS2 humble with the setup was achieved successfully.

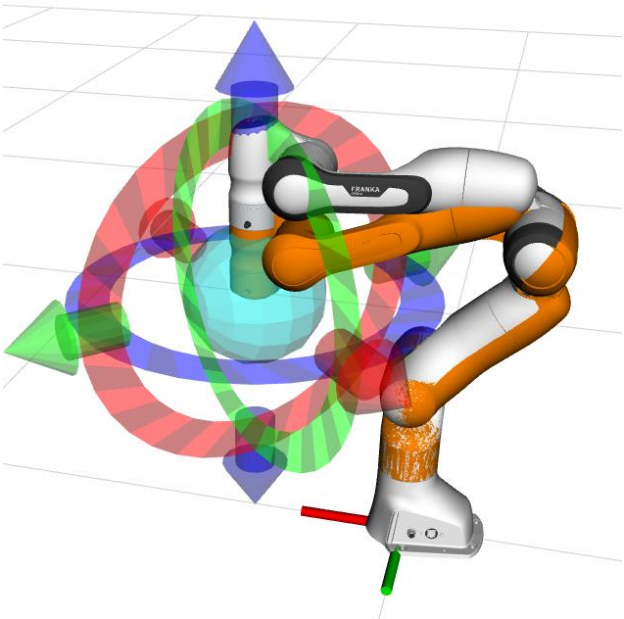

Figure 7. Franka setup moving in MoveIt

The additional video attachment shows the movement of the robot in a real world. Below are the screenshots of the manipulator moving to the desired position and also the gripper handling the object on the table.

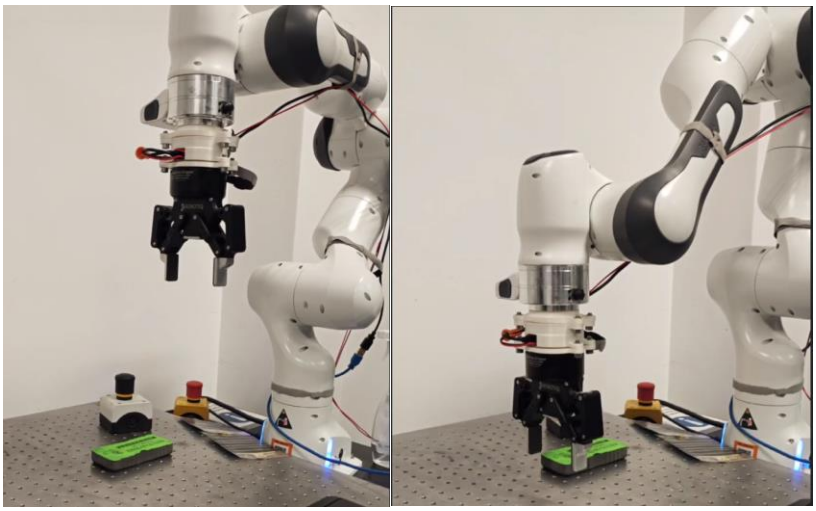

Figure 8. Manipulator moving to the desired position

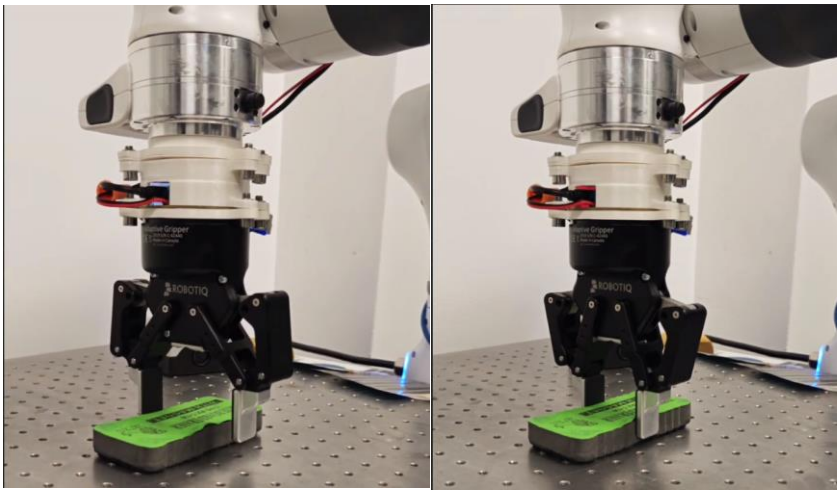

Figure 9. The end-effector gripping onto the object on the table

To conclude, although working separately, the setup components work as desired. We could successfully integrate the system with ROS2 and transfer the simulated setup into the real hardware.

Project files: https://github.com/soibkhon/panda_ros2_humbe_robotiq2f