RBE500 Group Assignment – Part 3

Group 10

Team Members: Aniket Patil, Venkatesh Mullur, Febin Fredi

PART 1:

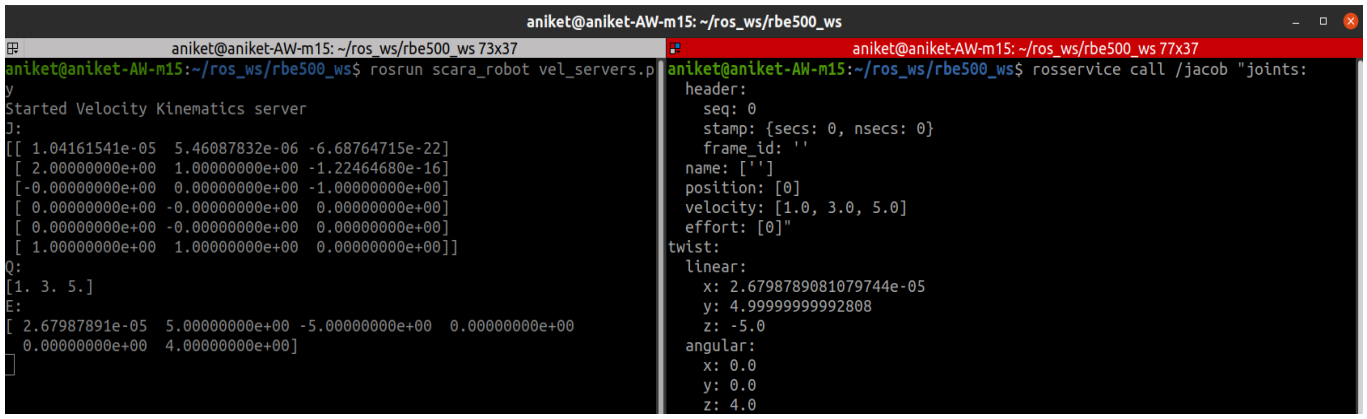
A node is created, which runs two services and a subscriber. For every subscriber call, a callback function get_Jacob runs and reads the joint angle values from the topic "**/scara_robot/joint_states".** The joint values read are used to update the Jacobian.

There are two more functions **jacob_function** and **inv_jacob_function** which respond to rosservice calls.

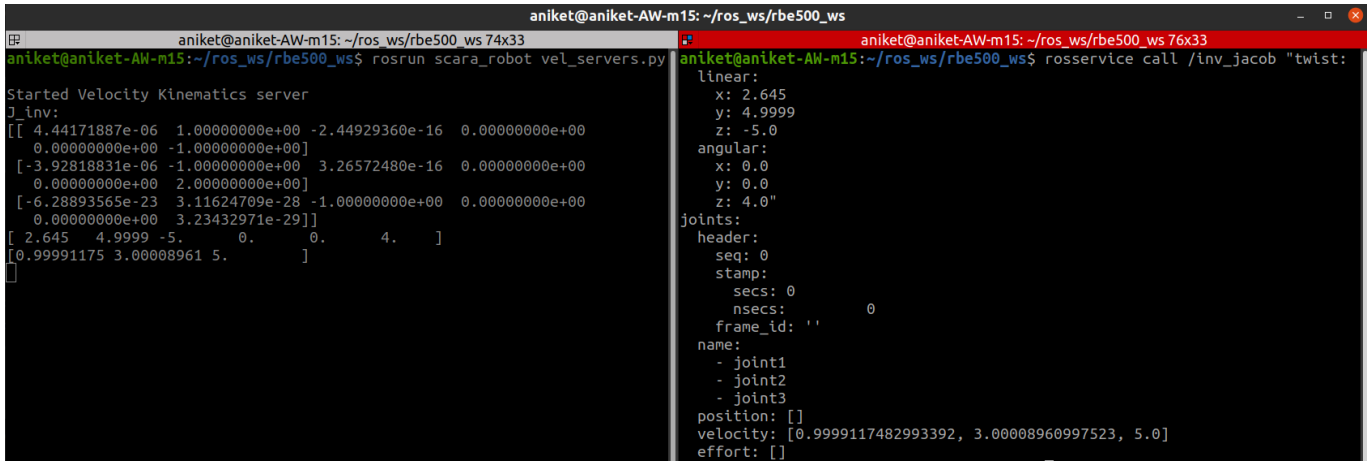
The jacob_function takes request information that is joint velocities and computes Twist values.

The inv_jacob_function takes request information to get end effector Twist values and computes joint velocities.

The two services use the datatypes: **geometry_msgs/Twist** and **sensor_msgs/JointState**



In the picture above, we see that for certain joint velocities, we receive end effector Twist values. Now we will pass these to the second service call to compute the reverse process.

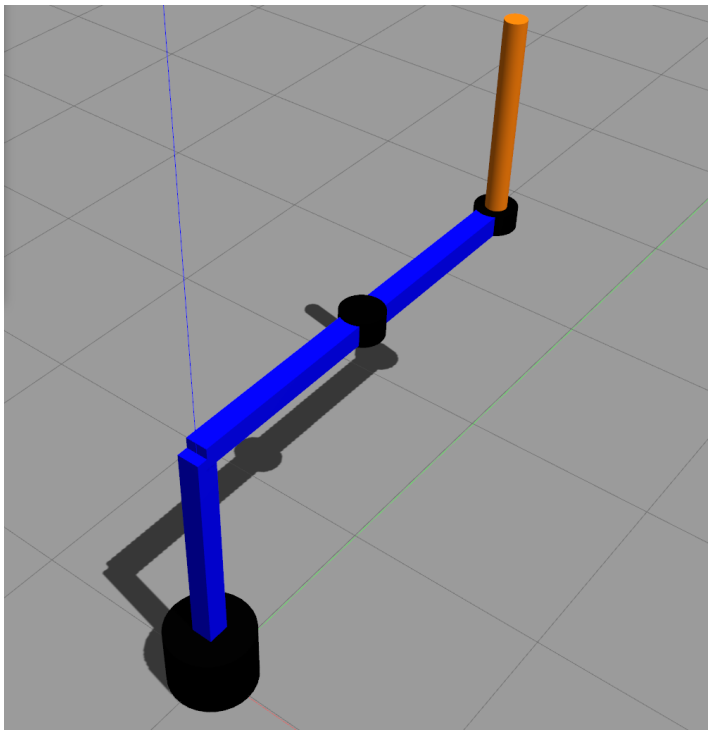


Here, we see that passing the first resultant end effector Twist to the second service call, we get the original Joint Values back.

Results: Service Nodes are working correctly!

PART 2:

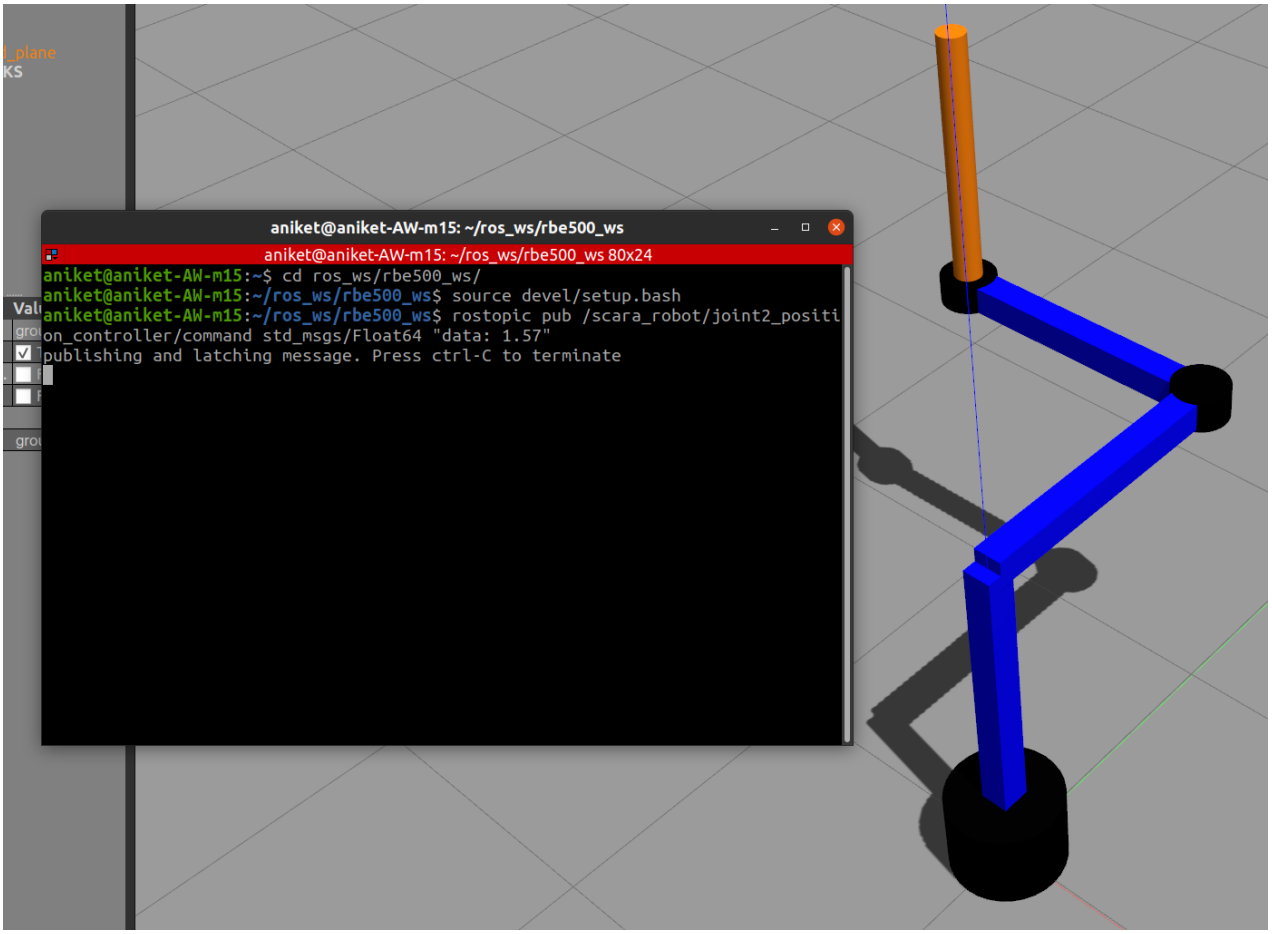
The robot's singular position is at q2 = 0, that is, when the robot is in a straight line. We can also get this from the determinant of the Jacobian.



We have enabled Position Controllers in the last part of the assignment, which means we can use the corresponding topics created by the ros_control package. One such topic is **/scara_robot/joint2_position_controller/command**

We publish the value, 1.57 in order to get the robot away from the singular configuration.

We can see the new robot position below, which is away from the singular position that is q2 = 0:



PART 3:

To implement the controllers, we have used the **ros_control** package.

- We added the **<transmission>** element in our Xacro file which specifies the type of hardware interface between the controller and actuators.
- We used **hardware_interface/EffortJointInterface**.
- We added **gazebo_ros_control** plugin in the robot.gazebo file, which is also included in the Xacro file.
- We created a robot_config.yaml file which specifies three position controllers of type **"effort_controllers/JointVelocityController"** and specifies the P, I and D values for the controller. (We add estimated values for now)

```
# Position Controllers ---------------------------------------
joint1_position_controller:
  type: effort_controllers/JointPositionController
  joint: joint1
  pid: {p: 200.0, i: 0.0, d: 120.0}
joint2_position_controller:
  type: effort_controllers/JointPositionController
  joint: joint2
  pid: {p: 55.0, i: 0.0, d: 30.0}
joint3_position_controller:
  type: effort_controllers/JointPositionController
  joint: joint3
  pid: {p: 50.0, i: 0.0, d: 10.0}

# Velocity Controllers ---------------------------------------
joint1_velocity_controller:
  type: effort_controllers/JointVelocityController
  joint: joint1
  pid: {p: 50.0, i: 0.0, d: 30.0}
joint2_velocity_controller:
  type: effort_controllers/JointVelocityController
  joint: joint2
  pid: {p: 50.0, i: 0.0, d: 30.0}
joint3_velocity_controller:
  type: effort_controllers/JointVelocityController
  joint: joint3
  pid: {p: 50.0, i: 0.0, d: 30.0}
```

- Added the controllers in the .launch file, and initialised the velocity controllers in “stopped” mode using the tag “--stopped”
- This allows us to control the Velocity Controller using the SwitchController service, part of the ros_control package

PART 4:

The python code switch_controller.py was created to run the ROS SwitchController service. The code implements a switcher function which switches from Position Controller to Velocity Controller using a service that is part of the ros_control package.

The code snippet is attached below:

```
rospy.init_node('switcher', anonymous=True)
rate = rospy.Rate(1)  # meaning 1 message published in 1 sec
rospy.sleep(5)
random.seed()

# once the joints have moved from home position,
# the position controller is stopped and velocity controller is started.
# We use ros inbuilt switch_controller service for that.
rospy.wait_for_service('/scara_robot/controller_manager/switch_controller')
try:
    sc_service = rospy.ServiceProxy(
        '/scara_robot/controller_manager/switch_controller', SwitchController)
    start_controllers = ['joint1_velocity_controller',
                         'joint2_velocity_controller']
    stop_controllers = ['joint1_position_controller',
                        'joint2_position_controller']
    strictness = 2
    start_asap = False
    timeout = 0.0
    res = sc_service(start_controllers, stop_controllers,
                     strictness, start_asap, timeout)

except rospy.ServiceException as e:
    print("Service Call Failed")
```

The motion part of this section was very irregular. We were not able to generate plots. This was only partially implemented using the Velocity Controllers after switching. We were not able to generate the necessary plots for our published values.