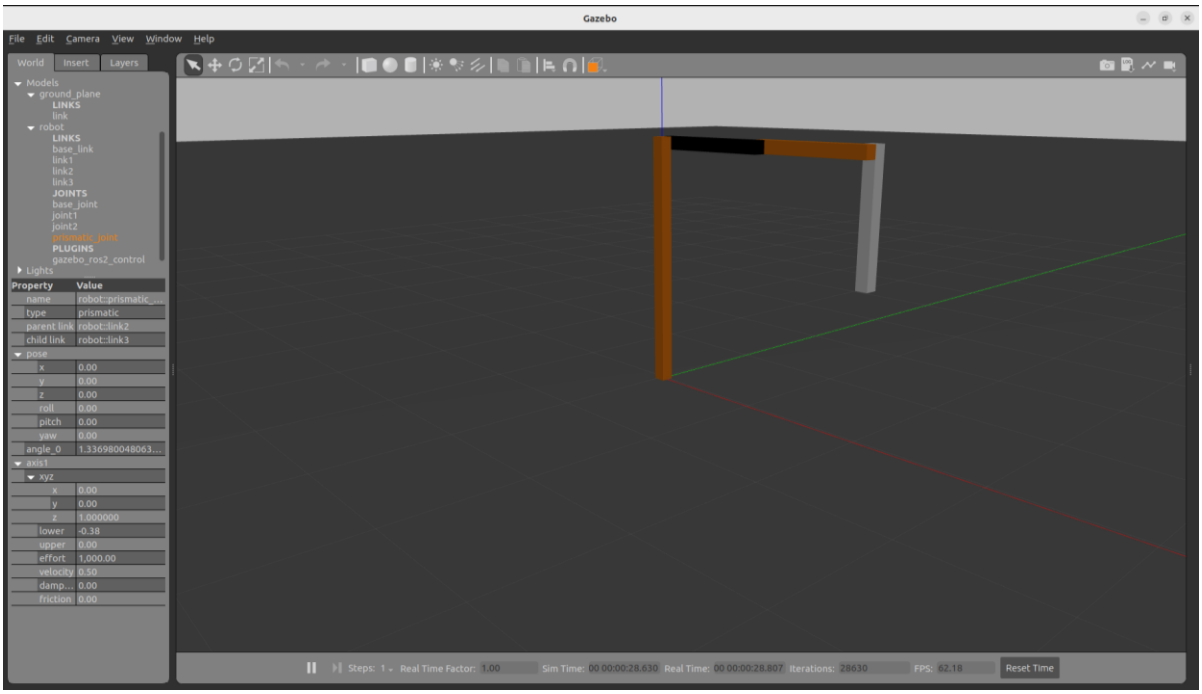
Report for Foundations of Robotics (RBE500)_Group Assignment Part 1:

Report By:
Swapneel Dhananjay Wagholikar (WPI ID: 257598983)
Shounak Sheshadri Naik (WPI ID: 728556739)
Nikunj Reddy Polasani (WPI ID: 901004192)

Part1: Creating a robot in Gazebo

To install gazebo and spawn the robot instructions from the ‘gazebo.md file’ were followed. After this was done the robot spawned in gazebo was modified. This was achieved by making changes to one urdf file, ‘rrbot.ros2_control’ file and two .yml extensions. For the urdf line one prismatic chain and prismatic joint types were added. The joint types for the rest of the joints were changed from continuous type to revolute based on the diagram. For ‘rrbot.ros2_control’ and .yml extension files the newly added prismatic joint was included in the config. Following these steps would yield the required robot configuration as shown in Figure 1.

Modified robot in gazebo:



Part 2: Forward Kinematics

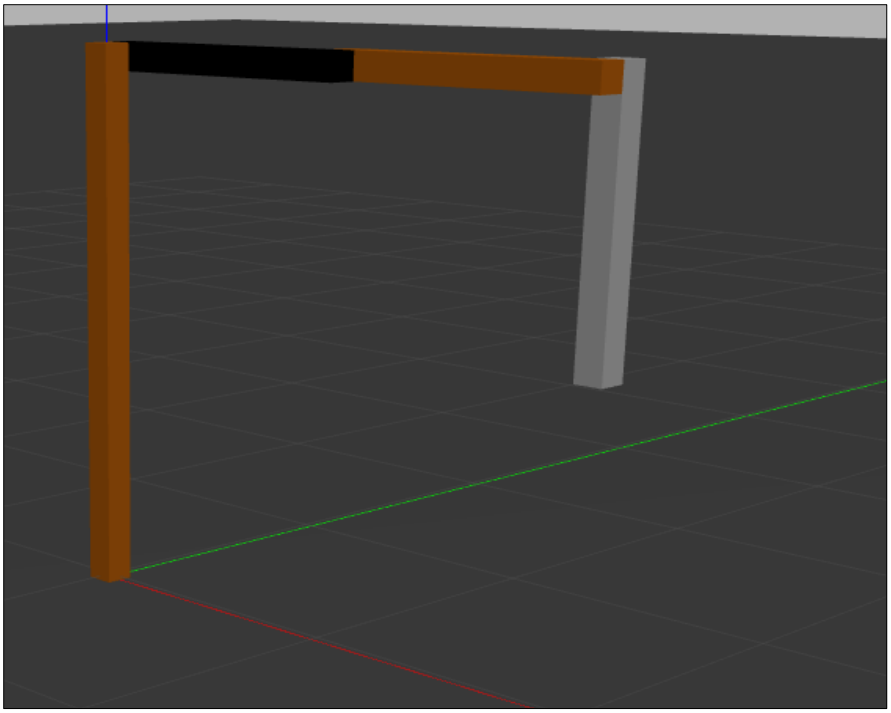
In this part of the assignment, only subscriber is created and publisher inputs are taken from gazebo simulator. As the task was to perform the forward kinematics on the SCARA robot, joint variables q1, q2, q3 are taken as input in listener callback function (first by using JointState in minimal subscriber) and homogeneous matrix is printed by computing for the same in subscriber node.

In the code, subscriber begins with calling dependencies. rclpy, node, numpy, math and JointState from sensor msgs are used here. In this node, it subscribes to the message received from gazebo simulator and act upon it in /joint_states function.

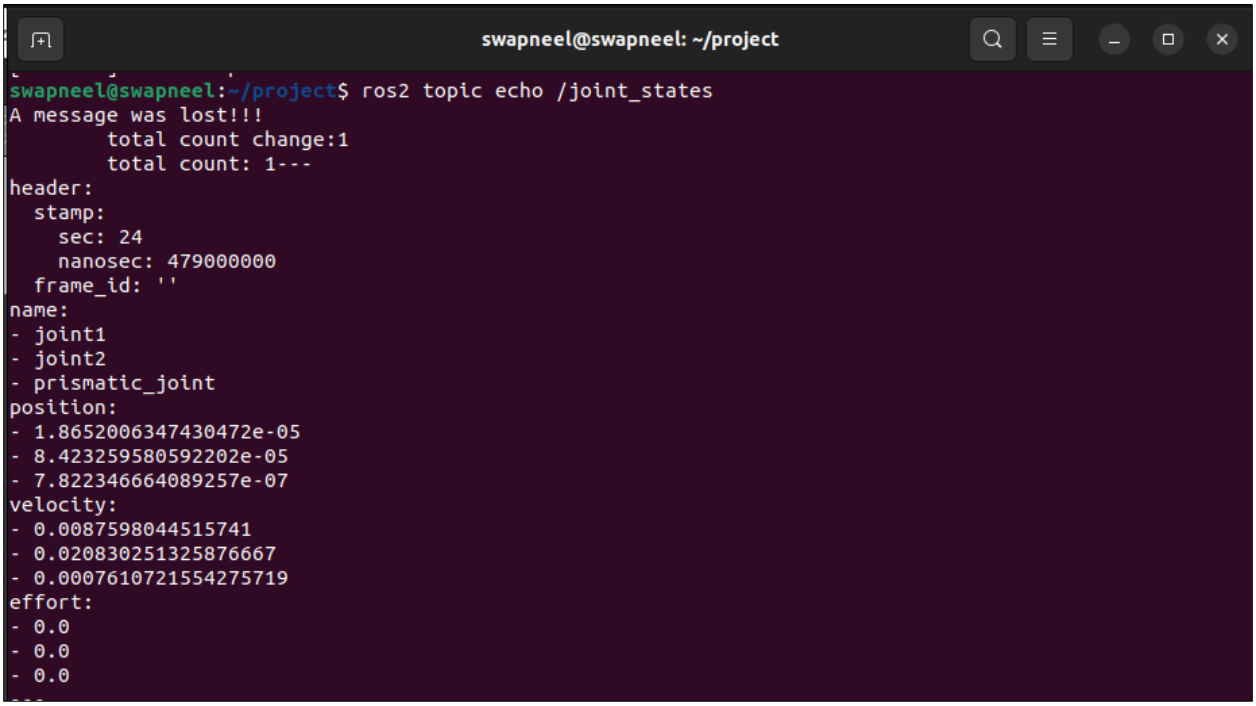
In the listener callback function, variables (q1, q2, q3) are used to store the subscribed data. Also, this function involves the computation for computing the joint transformation matrices (A1, A2, A3) and ultimately the homogeneous matrix by multiplying these 3 matrices. Finally, the homogeneous matrix is computed and printed (which is pose of the end effector).

Using the logic explained above, publisher-subscriber implementation is done.

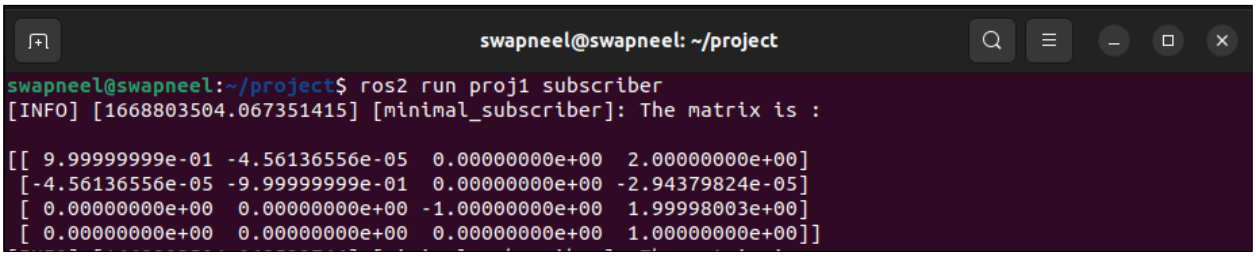
For position 1:
Screenshot of the robot:



Resulting pose in terminal (using "ros2 topic echo…" command):

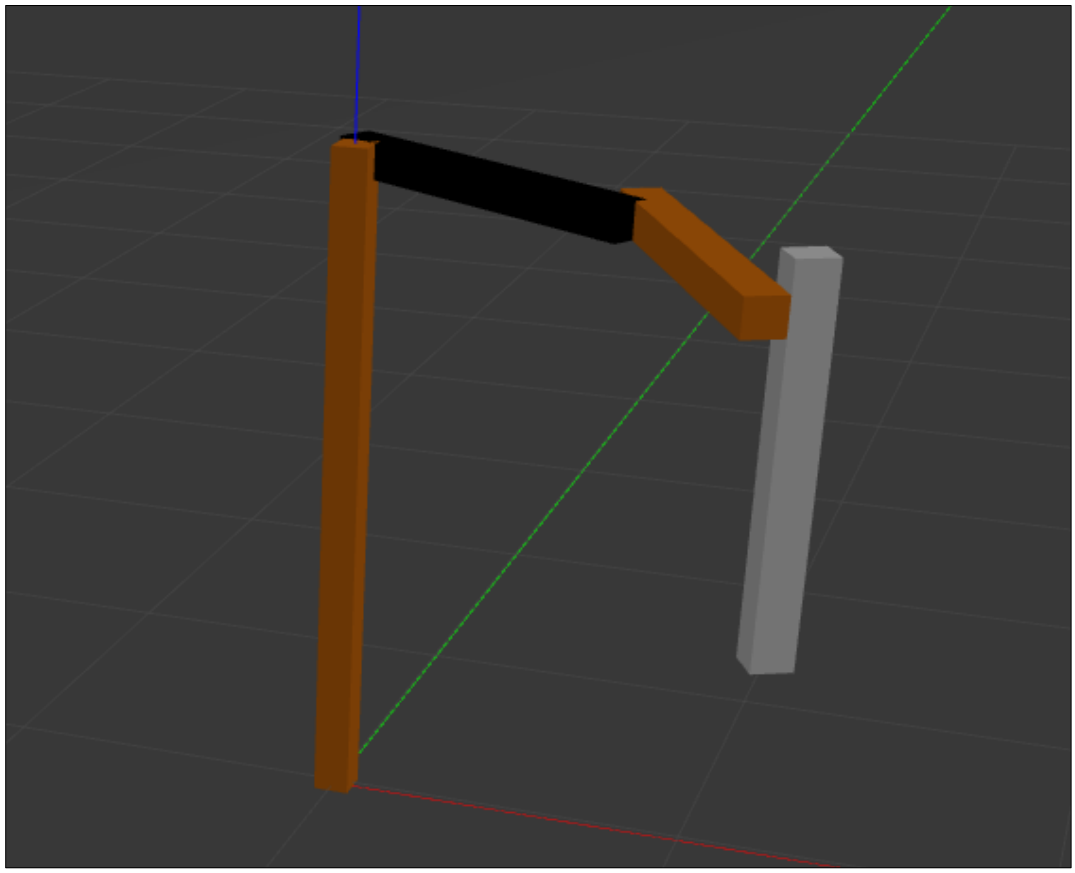


Output of Forward Kinematic Node:

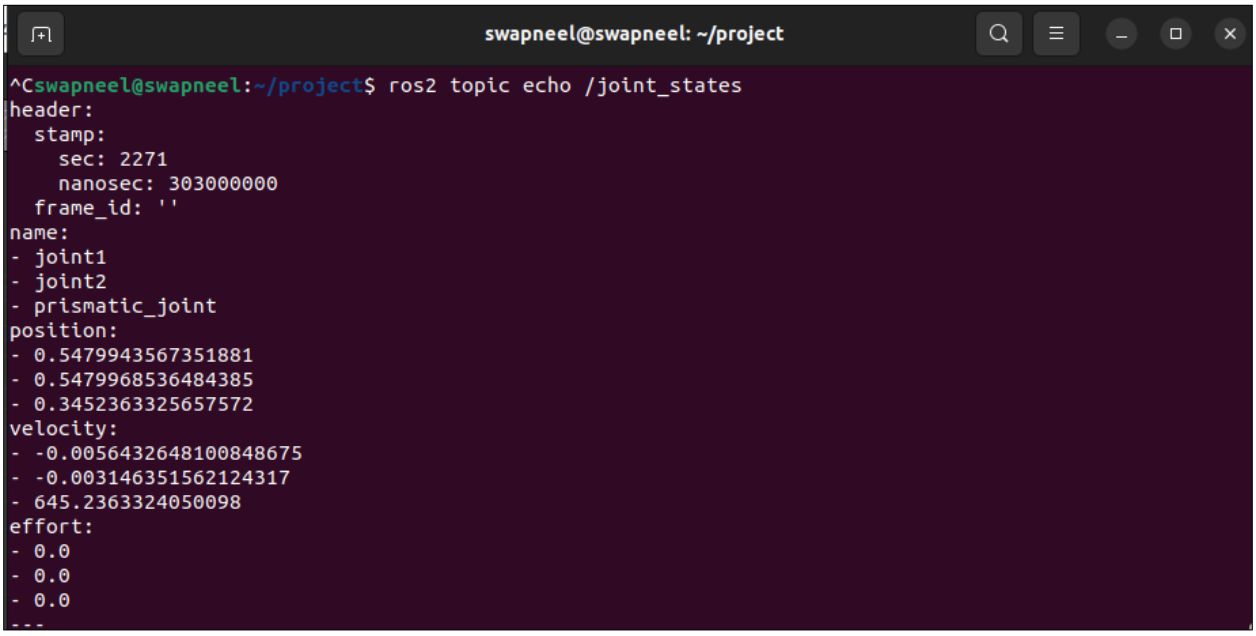


For Position 2:

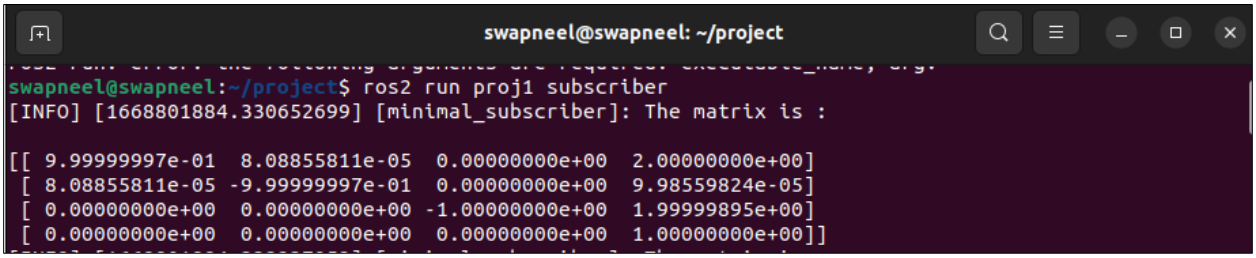
Screenshot of the robot:



Resulting pose in terminal (using "ros2 topic echo…" command):

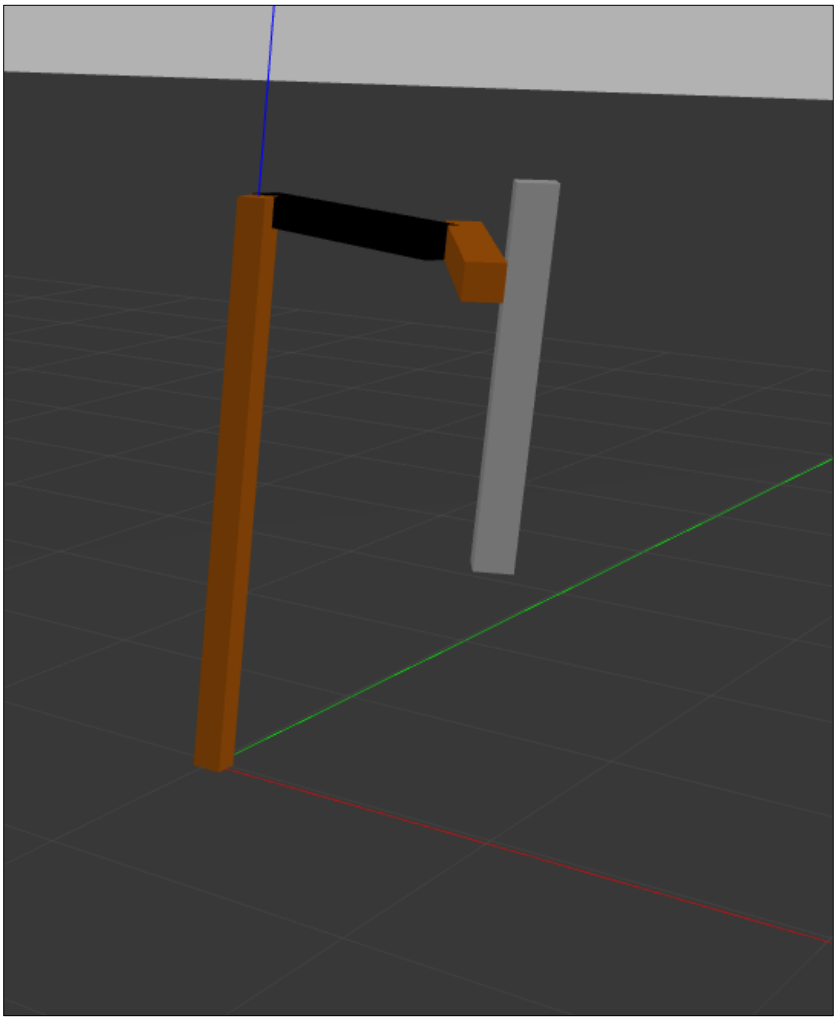


Output of Forward Kinematic Node:

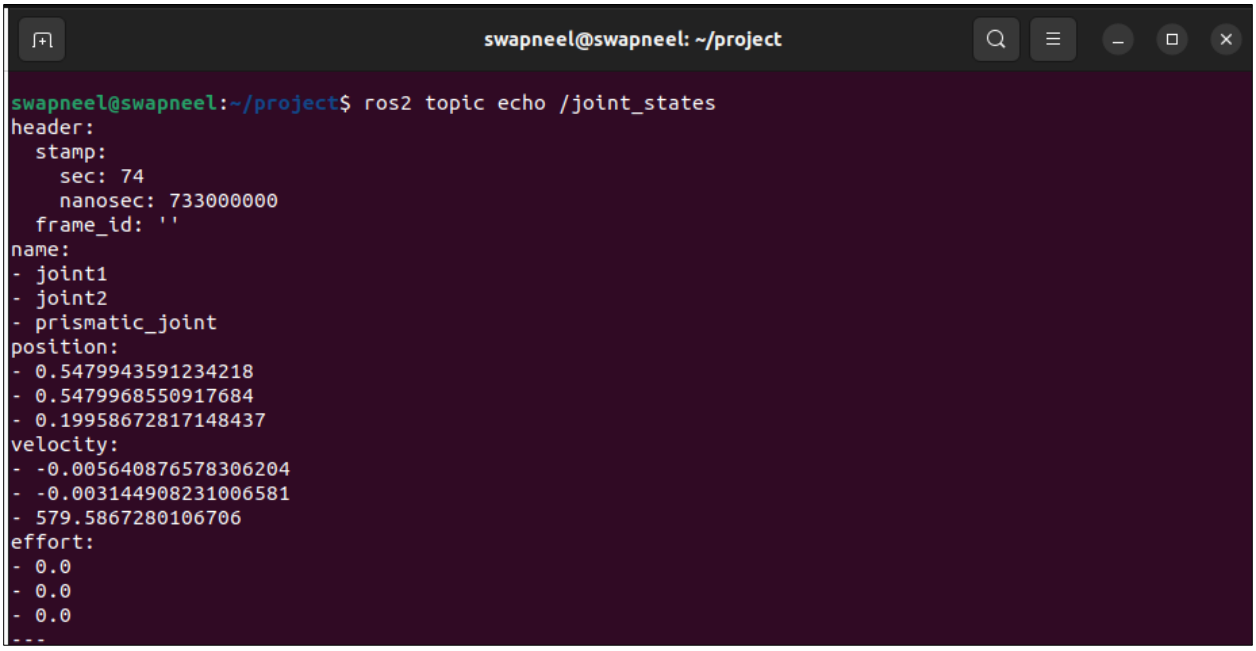


For position 3:

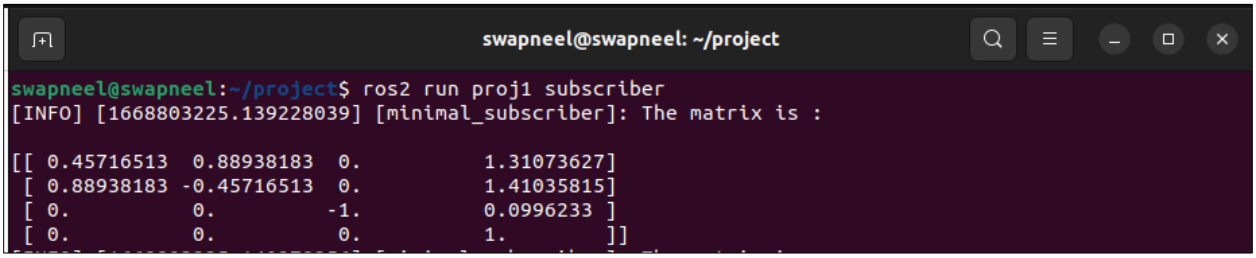
Screenshot of the robot:



Resulting pose in terminal (using "ros2 topic echo…" command):



Output of Forward Kinematic Node:



Part 3: Inverse Kinematics

In this part of the assignment, separate inverse kinematics node is implemented. The implementation is done in service server-service client format. As the task was to perform the inverse kinematics on the SCARA robot, position of the end effector i.e. x, y, z is taken as input in service callback function and joint variables q1, q2, q3 are printed by computing for the same.

Firstly, a different package had to be made for the service message. The service message takes x,y and z in the request part and in the response it gives out q1,q2 and q3. This package is in C++.

The Service Node is written in a different package named inverse kinematics. The service message datatype is called in the inverse kinematics package. This Service Node waits for a request on the 'inverse' service channel and once it gets a request, it computes the q1,q2 and q3 and then outputs these q1,q2 and q3 as a response.

In the service callback function, variables (x, y, z) are used to store the subscribing requests. Also, this function involves the computation for calculating joint variables for the required link lengths. Finally, the values for joint variables are computed and printed.

Using the logic explained above, service server-service client implementation is done.

The created node is tested with "ros2 service call" command for three separate end effector positions.

Please find the below screenshot of the terminal:

```
dell@dell-G3-3579:~/RBE_500/grp1$ ros2 service call /inverse tutorial_interfaces/srv/FirstService "{x: 1.31,y: 1.41,z: 1.2}"
waiting for service to become available...
requester: making request: tutorial_interfaces.srv.FirstService_Request(x=1.31, y=1.41, z=1.2)

response:
tutorial_interfaces.srv.FirstService_Response(q1=0.5467404127120972, q2=0.5508118867874146, q3=0.7999999523162842)

dell@dell-G3-3579:~/RBE_500/grp1$ source install/setup.bash
dell@dell-G3-3579:~/RBE_500/grp1$ ros2 service call /inverse tutorial_interfaces/srv/FirstService "{x: 1.31,y: 1.41,z: 1.2}"
requester: making request: tutorial_interfaces.srv.FirstService_Request(x=1.31, y=1.41, z=1.2)

response:
tutorial_interfaces.srv.FirstService_Response(q1=0.5467404127120972, q2=0.5508118867874146, q3=0.7999999523162842)

dell@dell-G3-3579:~/RBE_500/grp1$ ros2 service call /inverse tutorial_interfaces/srv/FirstService "{x: 1.2,y: 1.3,z: 1.4}"
requester: making request: tutorial_interfaces.srv.FirstService_Request(x=1.2, y=1.3, z=1.4)

response:
tutorial_interfaces.srv.FirstService_Response(q1=0.34019526839256287, q2=0.9703630805015564, q3=0.6000000238418579)

dell@dell-G3-3579:~/RBE_500/grp1$ ros2 service call /inverse tutorial_interfaces/srv/FirstService "{x: 1.25,y: 1.23,z: 1.5}"
requester: making request: tutorial_interfaces.srv.FirstService_Request(x=1.25, y=1.23, z=1.5)

response:
tutorial_interfaces.srv.FirstService_Response(q1=0.2757890820503235, q2=1.003089427947998, q3=0.5)

dell@dell-G3-3579:~/RBE_500/grp1$
```