

RBE 500

PROJECT ASSIGNMENT

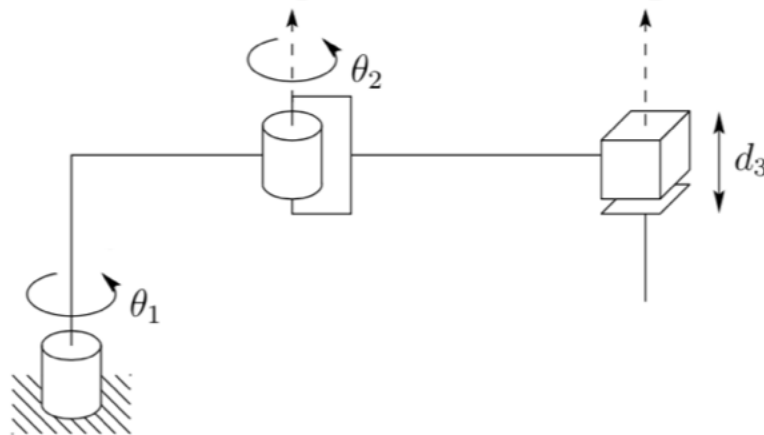
PART - 1

SHREYAS CHIGURUPATI

TRIPTH SHARMA

Problem Statement

1) (4 pts) **Create the robot:** Create the following SCARA robot in ROS and Gazebo.



You can utilize ROS Session 5 and the Gazebo tutorials about how to create the robot and how to integrate it with ROS. Please note that only the configuration of the robot is important; it does not need to visually look like it is in the figure. You can represent the links with cylinders, for example.

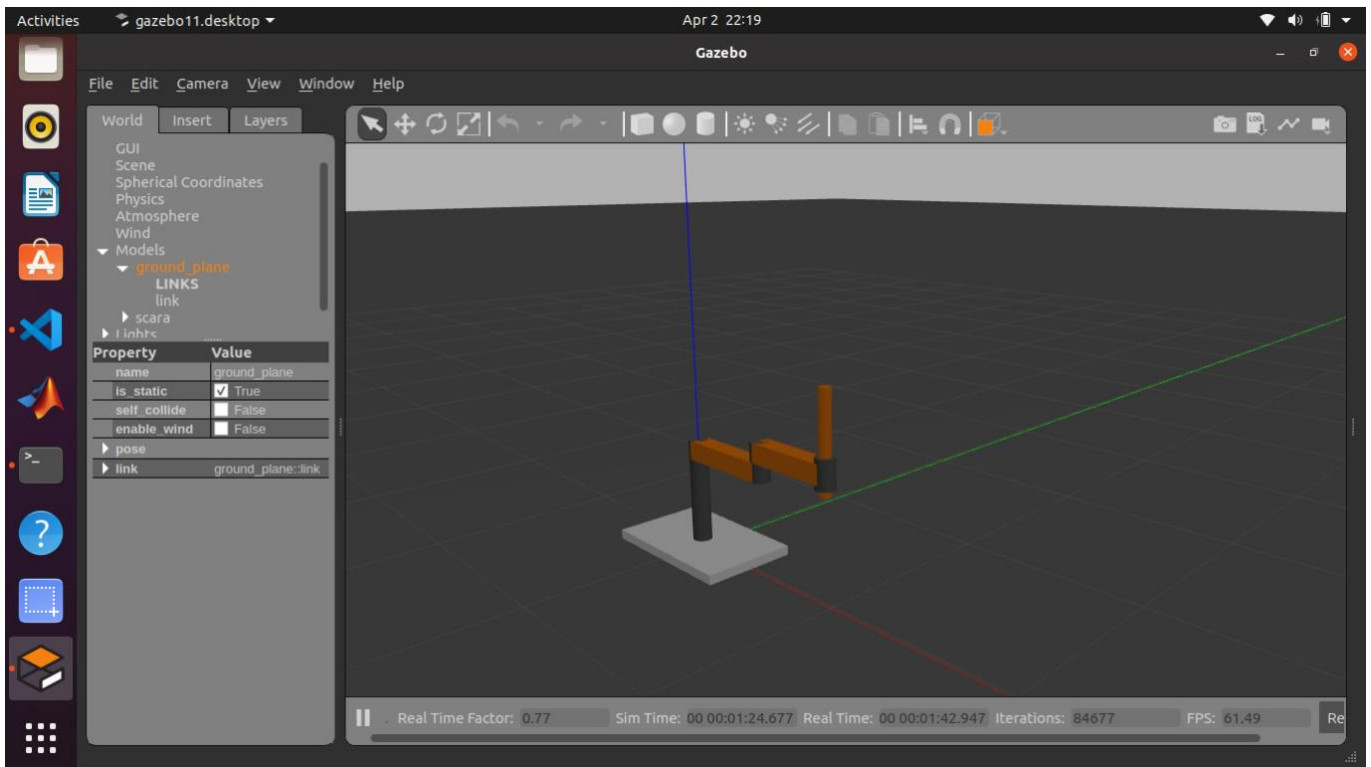
Please note that you can read the joint value of the robot via topics published by Gazebo to the ROS environment.

Spawn your robot in your ROS-Gazebo environment and take an image of your robot. Include the image in your report together with your robot definition file.

Solution:

To create the SCARA robot in ROS – GAZEBO environment, we have created our ROS workspace named **rbe500_ROS**. In this workspace we have created two packages one for the scripts and another for the robot description and control. The **urdf** file is used to spawn the robot in Gazebo world. We have created the robot in VS code using the ROS extensions. The file urdf contains the physical parameters of the robot like link lengths, mass of links, inertial properties, dimensions of the robot, etc.

The spawned robot in Gazebo is shown below.



2) (3 pts) **Forward Kinematics:** Implement a forward kinematics node that

- Subscribes to the joint values topic and reads them from the gazebo simulator
- Calculate the end effector pose
- publishes the pose as a ROS topic (inside the callback function that reads the joint values)

This is a publisher-subscriber implementation. Print the resulting pose to the terminal by using “rostopic echo” command, and include the screenshot in your report.

Solution:

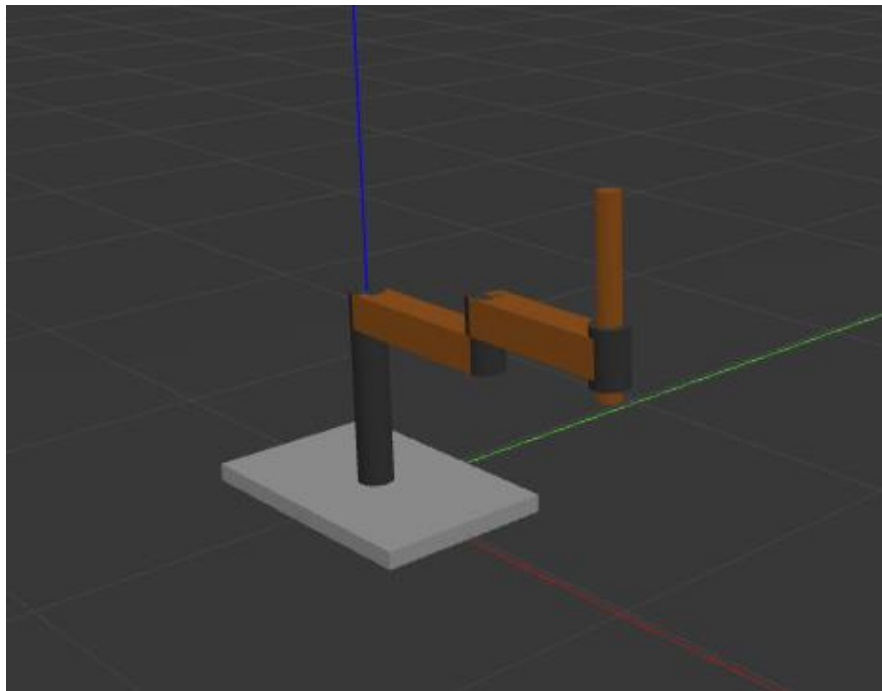
To implement the forward kinematics of the robot we created a **Publisher – Subscriber** node that communicate with each other using the topic “**rrp/joints_states**”. The python code for the subscriber is developed in such a way that it grabs the joint variables from the **joint state msg** that contains the position and orientation values of the robot in real time. That is, as the robot is spawned in gazebo world, it returns the joint variables related to the robot in it’s environment to the joint state msg.

Now that we have the joint variables for our robot, we use the call back function in our subscriber code to calculate the forward kinematics for the robot. Note that the **DH parameters** are designed to calculate the homogeneous transformation matrices. For calculating the transformation matrix we have created a function that takes the dh parameters as arguments and return a **4x4** homogeneous transformation matrix. The formula used to calculate it is:

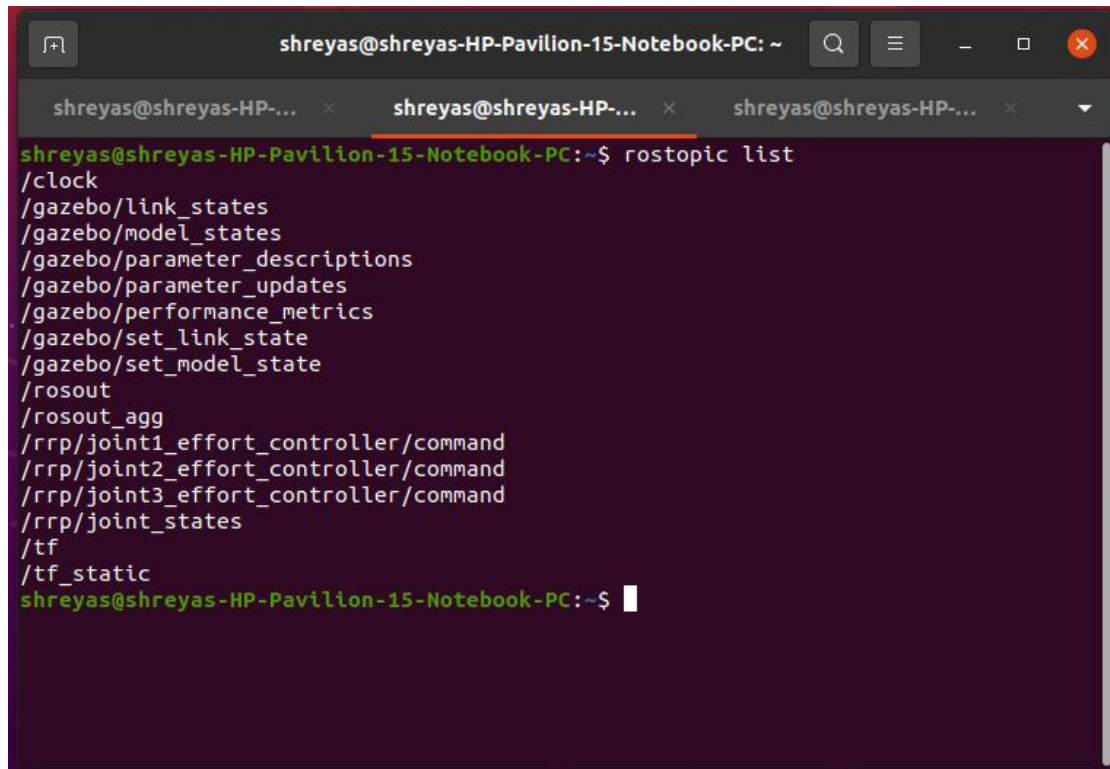
$$T_{i+1}^i = \begin{bmatrix} \cos \theta_i & -\sin \theta_i \cos \alpha_i & \sin \theta_i \sin \alpha_i & a_i \cos \theta_i \\ \sin \theta_i & \cos \theta_i \cos \alpha_i & -\cos \theta_i \sin \alpha_i & a_i \sin \theta_i \\ 0 & \sin \alpha_i & \cos \alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

And the DH parameters for the same is given by:

	T_{i+1}^i	θ	d	a	α
Link 1	T_2^1	θ_1	0	l_2	0
Link 2	T_3^2	θ_2	0	l_3	0
Link 3	T_4^3	0	$-d_3$	0	$+180$

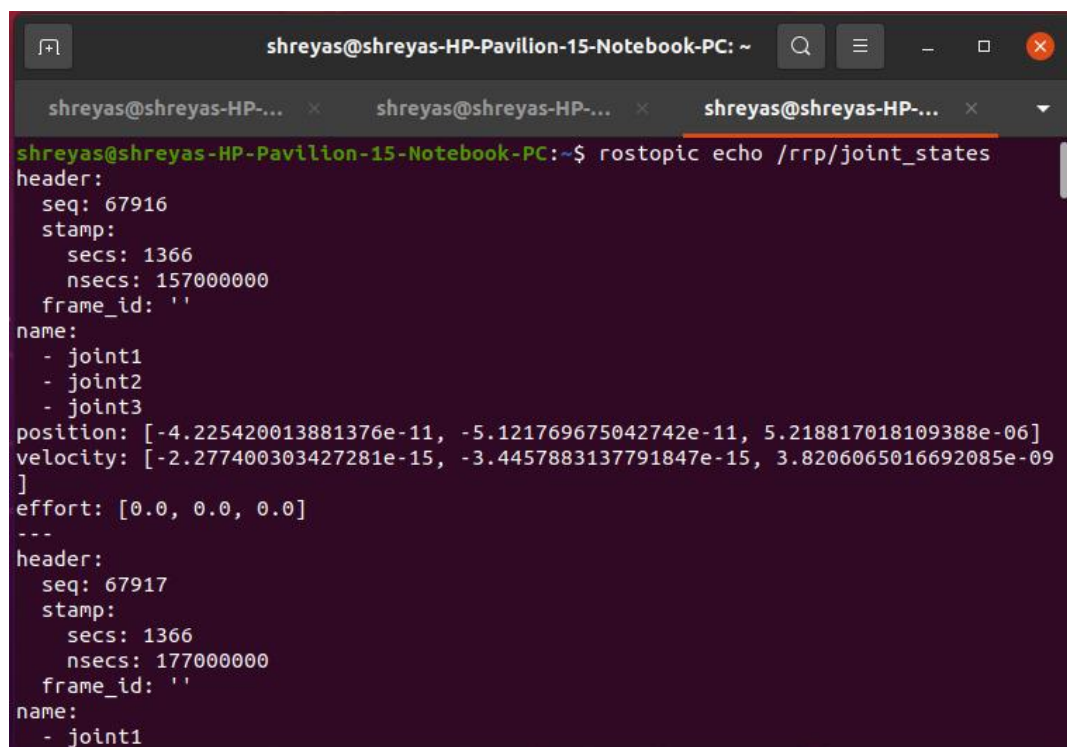


We can see the topics that are being published while the gazebo simulation is working as below, this can be seen using the “**rostopic list**” command.



```
shreyas@shreyas-HP-Pavilion-15-Notebook-PC: ~$ rostopic list
/clock
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/performance_metrics
/gazebo/set_link_state
/gazebo/set_model_state
/rosout
/rosout_agg
/rrp/joint1_effort_controller/command
/rrp/joint2_effort_controller/command
/rrp/joint3_effort_controller/command
/rrp/joint_states
/tf
/tf_static
shreyas@shreyas-HP-Pavilion-15-Notebook-PC:~$
```

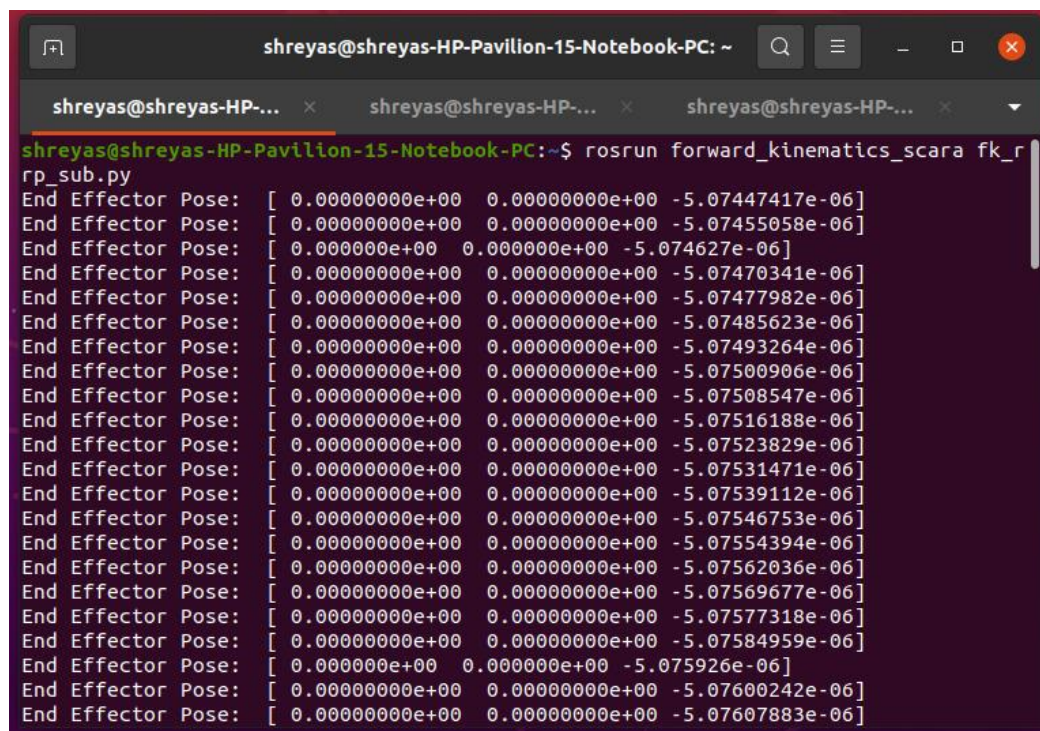
Now, when we are interested in extracting the joint states of the robot. It is done by using the command “**rostopic echo /rrp/joint_states**” , it gives the joint states as shown below:



```
shreyas@shreyas-HP-Pavilion-15-Notebook-PC:~$ rostopic echo /rrp/joint_states
header:
  seq: 67916
  stamp:
    secs: 1366
    nsecs: 157000000
  frame_id: ''
name:
- joint1
- joint2
- joint3
position: [-4.225420013881376e-11, -5.121769675042742e-11, 5.218817018109388e-06]
velocity: [-2.277400303427281e-15, -3.4457883137791847e-15, 3.8206065016692085e-09]
effort: [0.0, 0.0, 0.0]
---
header:
  seq: 67917
  stamp:
    secs: 1366
    nsecs: 177000000
  frame_id: ''
name:
- joint1
```

The joint variables here, θ_1 , θ_2 , d_3 are defined in the urdf file. So, the joint state msg publishes these values and our subscriber node subscribes to this msg and calls our callback function to perform the forward kinematics of our SCARA robot.

The results(end effector pose) of this are then published on the terminal, as shown below:



```
shreyas@shreyas-HP-Pavilion-15-Notebook-PC: ~  
shreyas@shreyas-HP-Pavilion-15-Notebook-PC:~$ rosrn forward_kinematics_scara fk_r  
rp_sub.py  
End Effector Pose: [ 0.00000000e+00 0.00000000e+00 -5.07447417e-06]  
End Effector Pose: [ 0.00000000e+00 0.00000000e+00 -5.07455058e-06]  
End Effector Pose: [ 0.000000e+00 0.000000e+00 -5.074627e-06]  
End Effector Pose: [ 0.00000000e+00 0.00000000e+00 -5.07470341e-06]  
End Effector Pose: [ 0.00000000e+00 0.00000000e+00 -5.07477982e-06]  
End Effector Pose: [ 0.00000000e+00 0.00000000e+00 -5.07485623e-06]  
End Effector Pose: [ 0.00000000e+00 0.00000000e+00 -5.07493264e-06]  
End Effector Pose: [ 0.00000000e+00 0.00000000e+00 -5.07500906e-06]  
End Effector Pose: [ 0.00000000e+00 0.00000000e+00 -5.07508547e-06]  
End Effector Pose: [ 0.00000000e+00 0.00000000e+00 -5.07516188e-06]  
End Effector Pose: [ 0.00000000e+00 0.00000000e+00 -5.07523829e-06]  
End Effector Pose: [ 0.00000000e+00 0.00000000e+00 -5.07531471e-06]  
End Effector Pose: [ 0.00000000e+00 0.00000000e+00 -5.07539112e-06]  
End Effector Pose: [ 0.00000000e+00 0.00000000e+00 -5.07546753e-06]  
End Effector Pose: [ 0.00000000e+00 0.00000000e+00 -5.07554394e-06]  
End Effector Pose: [ 0.00000000e+00 0.00000000e+00 -5.07562036e-06]  
End Effector Pose: [ 0.00000000e+00 0.00000000e+00 -5.07569677e-06]  
End Effector Pose: [ 0.00000000e+00 0.00000000e+00 -5.07577318e-06]  
End Effector Pose: [ 0.00000000e+00 0.00000000e+00 -5.07584959e-06]  
End Effector Pose: [ 0.000000e+00 0.000000e+00 -5.075926e-06]  
End Effector Pose: [ 0.00000000e+00 0.00000000e+00 -5.07600242e-06]  
End Effector Pose: [ 0.00000000e+00 0.00000000e+00 -5.07607883e-06]
```

3) (3 pts) Inverse Kinematics: Implement an inverse kinematics node (a separate node) that has a service client that takes a (desired) pose of the end effector and returns joint positions as a response.

This is a service server-service client implementation.

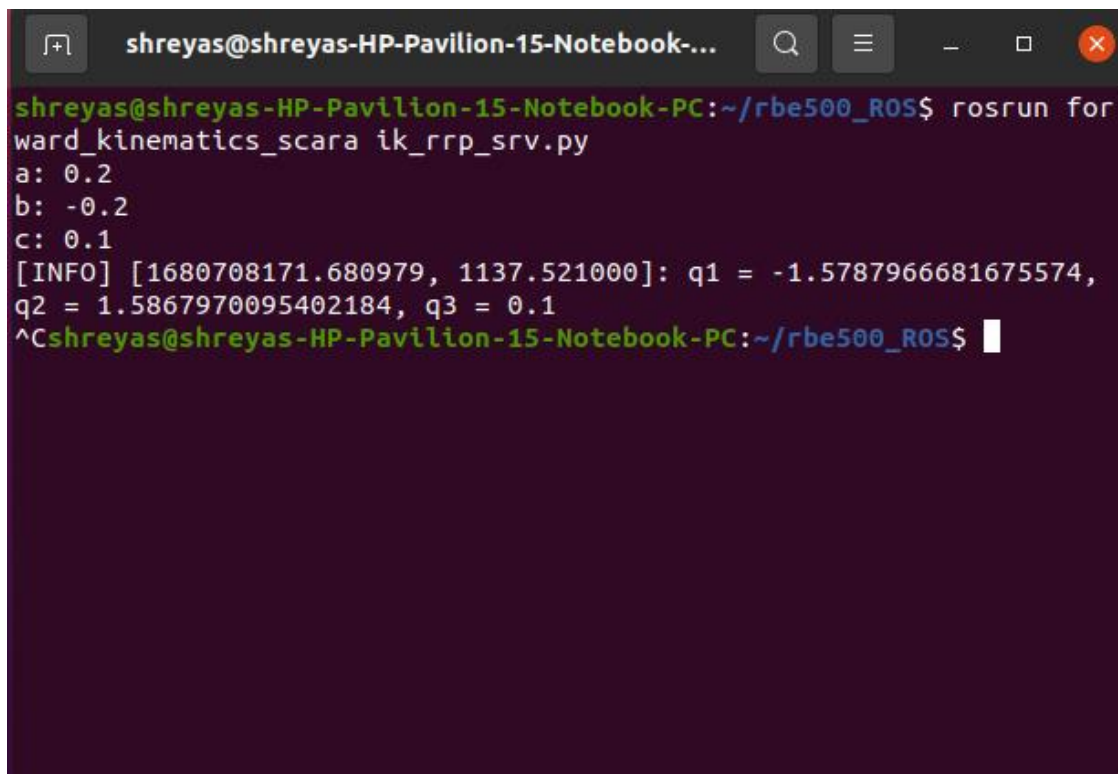
Test your node with the “rosservice call” command. Take the screen shot together and include it to your report

Solution:

To implement the inverse kinematics of the robot, we used the **server – client** approach. In this method we have created a node “**rrp_invkine_server**” that helps the server and client code to communicate with each other in order to perform the inverse kinematics of the robot. So, we have created a server and client python script that takes the input the desired pose of the robot and returns the joint variables on the terminal.

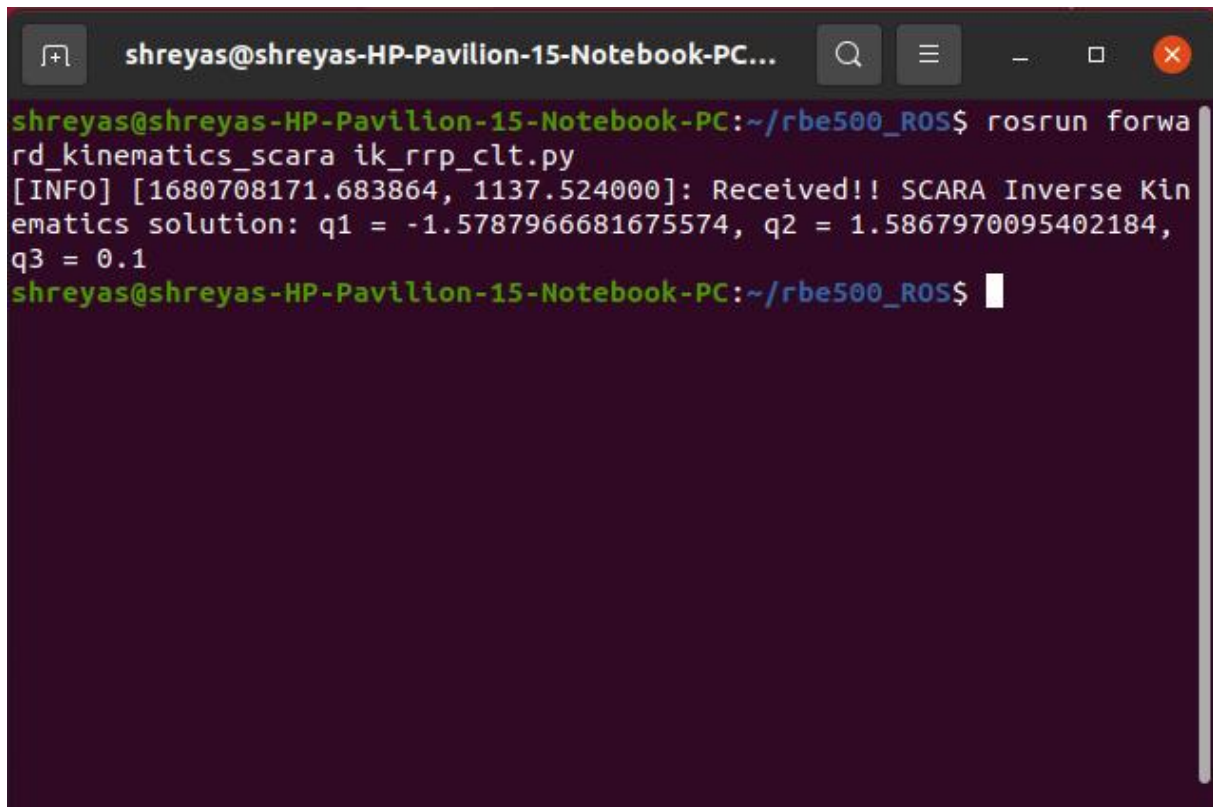
This is tested by running the scripts in terminal and the output is as shown below:

Server:

A terminal window with a dark background and light-colored text. The window title is "shreyas@shreyas-HP-Pavillion-15-Notebook-...". The prompt is "shreyas@shreyas-HP-Pavillion-15-Notebook-PC:~/rbe500_ROS\$". The command entered is "roslaunch forward_kinematics_scara ik_rrp_srv.py". The output shows three joint positions: "a: 0.2", "b: -0.2", and "c: 0.1". Below these, an INFO message is displayed: "[INFO] [1680708171.680979, 1137.521000]: q1 = -1.5787966681675574, q2 = 1.5867970095402184, q3 = 0.1". The prompt returns to "shreyas@shreyas-HP-Pavillion-15-Notebook-PC:~/rbe500_ROS\$".

```
shreyas@shreyas-HP-Pavillion-15-Notebook-PC:~/rbe500_ROS$ roslaunch forward_kinematics_scara ik_rrp_srv.py
a: 0.2
b: -0.2
c: 0.1
[INFO] [1680708171.680979, 1137.521000]: q1 = -1.5787966681675574, q2 = 1.5867970095402184, q3 = 0.1
^Cshreyas@shreyas-HP-Pavillion-15-Notebook-PC:~/rbe500_ROS$
```


Client:



```
shreyas@shreyas-HP-Pavilion-15-Notebook-PC...  
shreyas@shreyas-HP-Pavilion-15-Notebook-PC:~/rbe500_ROS$ rosrn forwa  
rd_kinematics_scara ik_rrp_clt.py  
[INFO] [1680708171.683864, 1137.524000]: Received!! SCARA Inverse Kin  
ematics solution: q1 = -1.5787966681675574, q2 = 1.5867970095402184,  
q3 = 0.1  
shreyas@shreyas-HP-Pavilion-15-Notebook-PC:~/rbe500_ROS$
```