PROJECT ASSIGNMENT PART – 2

Shreyas Chigurupati
Tripth Sharma

Problem:

This assignment is about controlling the robot joints.

1) First fix the all the joints except the last joint by changing the joint type of field of the corresponding joints to “fixed” in your robot description file.

2) Write a position controller node.

a. This node will get the joint positions from Gazebo and will be able to send joint efforts by “/gazebo/apply_joint_effort” topic.

b. Design a PD controller for the last joint (tune the parameters; you do not need to calculate).

c. Implement a service that gets a reference position for the last joint, and makes it go there.

d. Record the reference position and current position of the joint in a text file and plot it via Matlab.

Write a report about your implementation. The report does not have to be long, but it should explain all the steps of the implementation. Copy-pasting the code and the results is not enough. Submit your report together your node.

Solution:

First, we set all the joint except the prismatic joint (last joint) types from **continuous** to **fixed** in the urdf file (rrp.xacro). Then we commented out the respective joint controllers, if not done, it will cause an error as we are defining a controller for a joint that is fixed. So, for this question we neglect the controllers of joint 1 and 2.

Now, we have implemented a service client environment for designing the position controller for our last joint (prismatic).

Server code – **rrp_controller_server.py**

First we created a service node called **“pose_controller_rrp”.** This node will basically gets the joint states from the gazebo using the ROS topic “**/rrp/joint_states**” and then it receives a position value from the client and pass it on to the prismatic joint. When this is excepted the joint moves to the position specified during the client call.

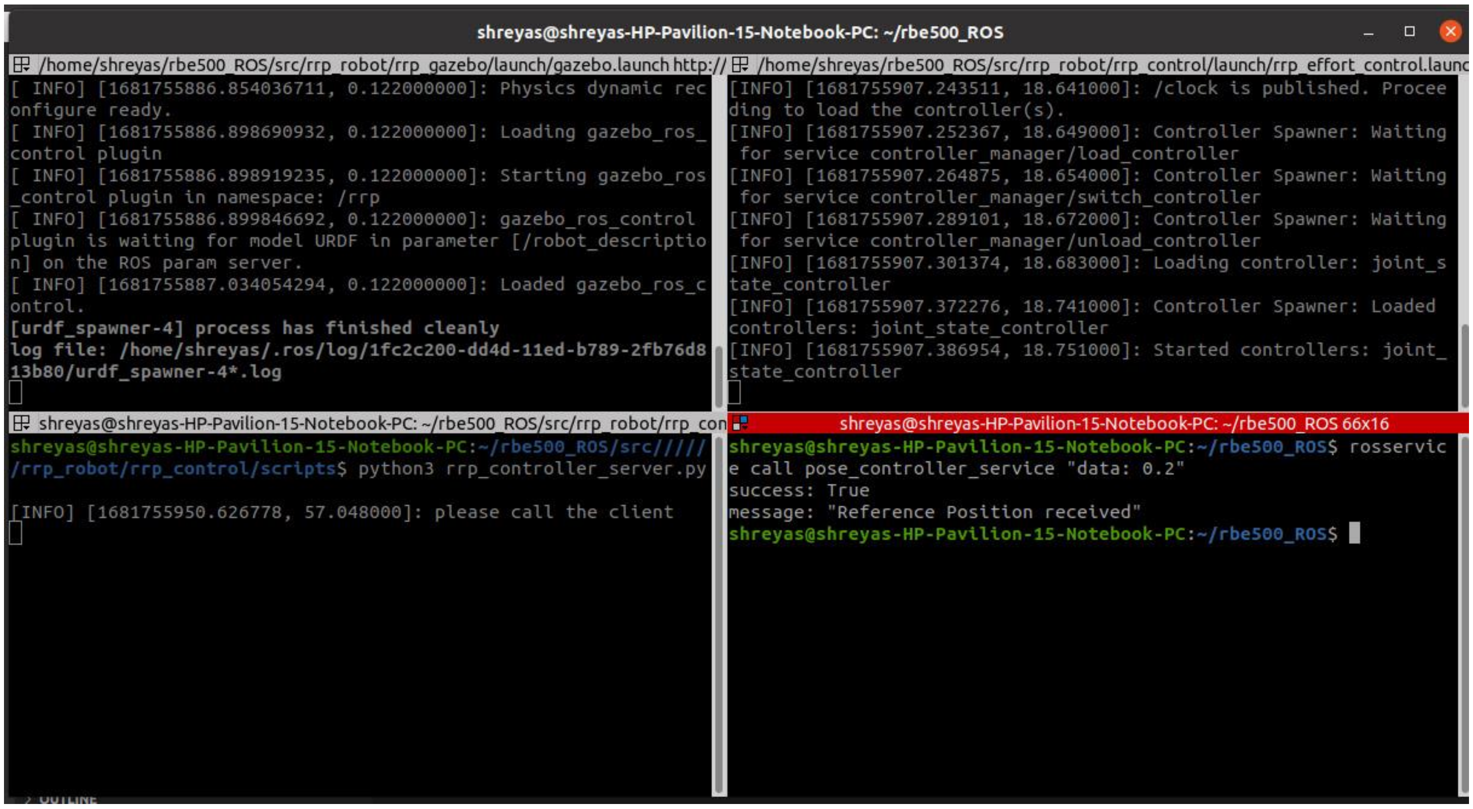
In this python script we have implemented out PD controller using proportional and derivative gains. The gains after tuning procedure are : Kp = 0.03, Kd = 3.
Then we formed the control law as :
Effort = Kp * error + Kd * derivative (error)

Where, derivative (error) here is defined as (error – previous error)

This control law tells the prismatic joint to track the desired value specified by the client. We also found that by **increasing** the control gains the controller converges faster but with much fluctuations, which is not ideal for the real time robot, as it will damage our controller.

So, after trying numerous different values for control gains, we found that these values are good enough for the controller to converge to specified position **smoothly**.

The terminal commands are as shown below:



```
shreyas@shreyas-HP-Pavilion-15-Notebook-PC: ~/rbe500_ROS
```

1

```
/home/shreyas/rbe500_ROS/src/rrp_robot/rrp_gazebo/launch/gazebo.launch http://
[ INFO] [1681755886.854036711, 0.122000000]: Physics dynamic rec
onfigure ready.
[ INFO] [1681755886.898690932, 0.122000000]: Loading gazebo_ros_
control plugin
[ INFO] [1681755886.898919235, 0.122000000]: Starting gazebo_ros
_control plugin in namespace: /rrp
[ INFO] [1681755886.899846692, 0.122000000]: gazebo_ros_control
plugin is waiting for model URDF in parameter [/robot_descriptio
n] on the ROS param server.
[ INFO] [1681755887.034054294, 0.122000000]: Loaded gazebo_ros_c
ontrol.
[urdf_spawner-4] process has finished cleanly
log file: /home/shreyas/.ros/log/1fc2c200-dd4d-11ed-b789-2fb76d8
13b80/urdf_spawner-4*.log
```

2

```
/home/shreyas/rbe500_ROS/src/rrp_robot/rrp_control/launch/rrp_effort_control.launc
[INFO] [1681755907.243511, 18.641000]: /clock is published. Procee
ding to load the controller(s).
[INFO] [1681755907.252367, 18.649000]: Controller Spawner: Waiting
 for service controller_manager/load_controller
[INFO] [1681755907.264875, 18.654000]: Controller Spawner: Waiting
 for service controller_manager/switch_controller
[INFO] [1681755907.289101, 18.672000]: Controller Spawner: Waiting
 for service controller_manager/unload_controller
[INFO] [1681755907.301374, 18.683000]: Loading controller: joint_s
tate_controller
[INFO] [1681755907.372276, 18.741000]: Controller Spawner: Loaded
controllers: joint_state_controller
[INFO] [1681755907.386954, 18.751000]: Started controllers: joint_
state_controller
```

3

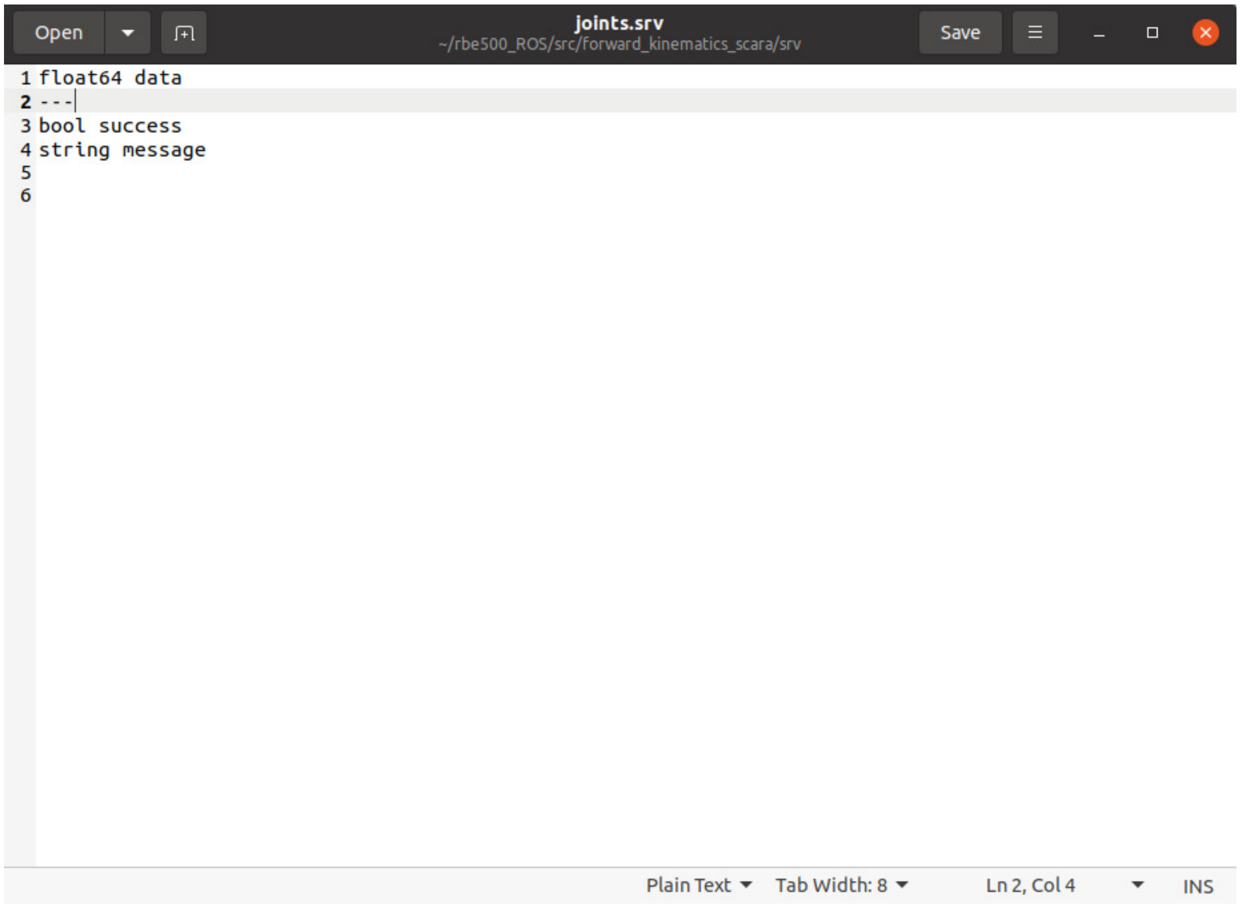
```
shreyas@shreyas-HP-Pavilion-15-Notebook-PC: ~/rbe500_ROS/src/rrp_robot/rrp_con
shreyas@shreyas-HP-Pavilion-15-Notebook-PC:~/rbe500_ROS/src/////
/rrp_robot/rrp_control/scripts$ python3 rrp_controller_server.py

[INFO] [1681755950.626778, 57.048000]: please call the client
```

4

```
shreyas@shreyas-HP-Pavilion-15-Notebook-PC: ~/rbe500_ROS 66x16
shreyas@shreyas-HP-Pavilion-15-Notebook-PC:~/rbe500_ROS$ rosservic
e call pose_controller_service "data: 0.2"
success: True
message: "Reference Position received"
shreyas@shreyas-HP-Pavilion-15-Notebook-PC:~/rbe500_ROS$
```

Here, the terminal 1 is used to spawn our robot in gazebo.
Terminal 2 is used to launch our controller for the joint 3 (prismatic joint)
Terminal 3 is used to call the server script
Terminal 4 is used to pass the arguments for the service

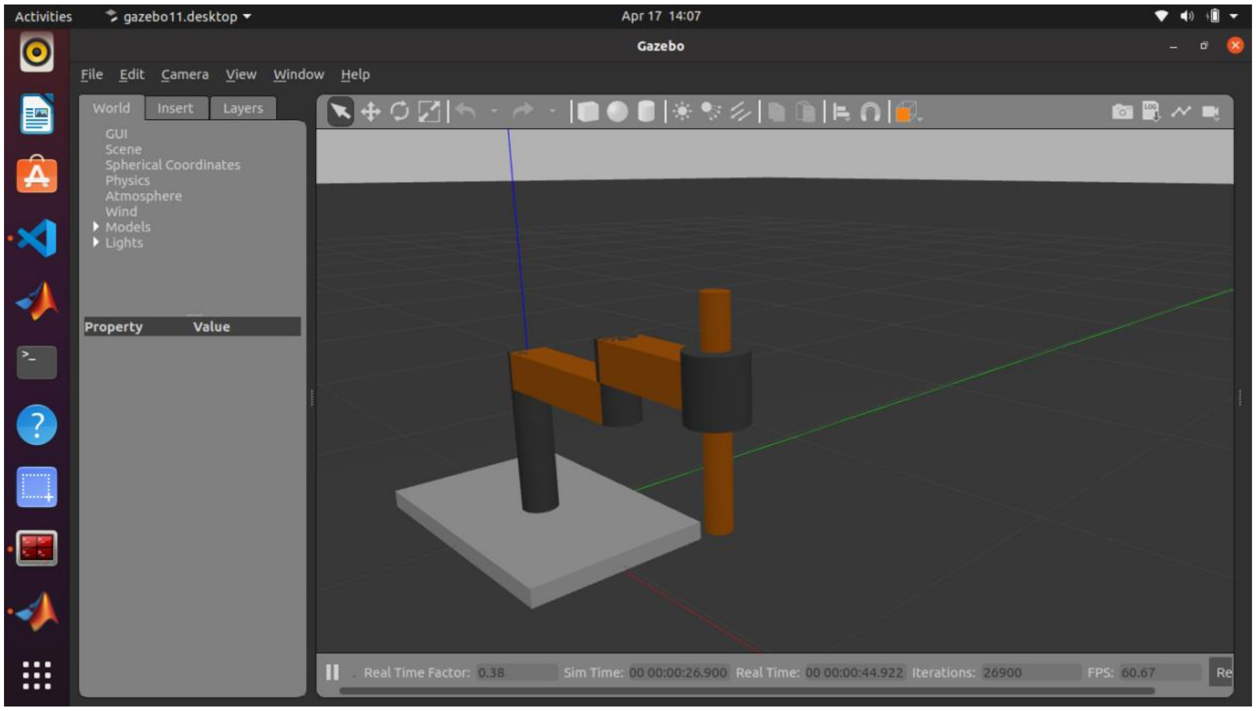
So, now we use the command "**rosservice call pose_controller_server" .** Through this command we can directly pass the **float** value (here we passed 0.2) which is taken as a request from our custom service called **"joint.srv".** This service is constructed as shown:



```
float64 data
---
bool success
string message


```

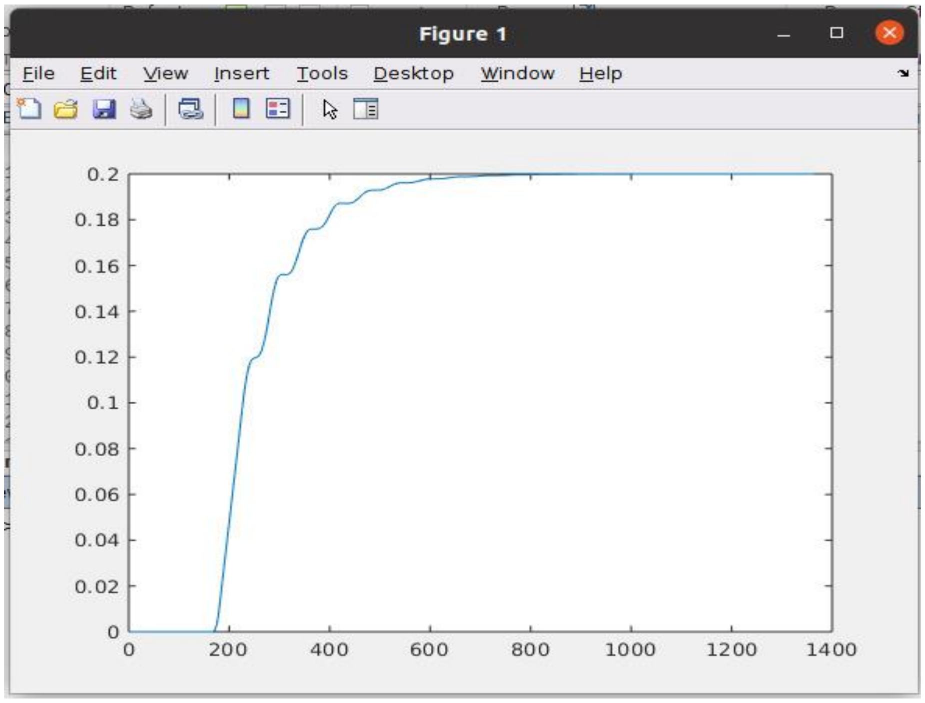
The service takes a **float** value as **request** and gives a **string** message as **response**.

Now, when we execute the code and pass the desired position as 0.2 in the client terminal (4). We can see the prismatic joint move down linearly to 0.2 position. The robot after reaching the desired position is shown below:



Now, we need to plot extract the movements and plot then on a graph in order to analyze the performance of the robot. For this we have implemented a function in our script called **“export_joint_position_to_csv”** , this function writes the positions taken by the robot to reach the desired position in a text file called “**currentPositionvsdesiredPostion.csv**”. This csv file contains two columns which gives the current position and desired positions of the robot until it reach the specified position. This data can then be used to plot in **Matlab**.

For plotting we created the Matlab script “**graphs.m”.** Now, we import the matrix table form the text file. Then we created a variable for storing the current position separately. Then we plot this variable using **plot** command, the result is observed as:



As we can see that the joint has reached to the specified position 0.2 quite smoothly. Hence, our position controller is efficient and works well.