Report for Foundations of Robotics (RBE500)_Group Assignment Part 3:

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

Part 1: Velocity Level Kinematics

This assignment uses the last project as its base/starter code and utilises urdf files from last group assignment. To execute the requirements in this assignment, a node is created with 2 services. One service takes the joint velocities and convert it into the end effector velocity. Other one takes the end effector velocities and convert it into the joint velocities.

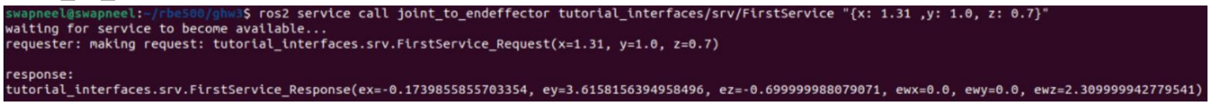
In service_callback_1 function, joint values are requested from terminal first and then end effector velocity is returned from this function as response. To perform this task, we have used already built jacobian function in listener_callback.

In service_callback_2 function, end effector velocity components are requested from the terminal and joint velocities are returned from this function as response. To perform this task, we have used the inverse of the jacobian calculated (with the help of pinv function).

In listener_callback function, execution is done to read the joint states i.e. q1,q2,q3 and link lengths are given manually. Using these values, we have calculated the jacobian.

The below terminal responses are received: (snaps attached)

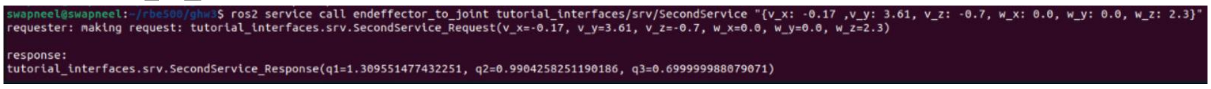
Joint_to_endeffector:



```
swapneel@swapneel:~/rbe500/ghw3$ ros2 service call joint_to_endeffector tutorial_interfaces/srv/FirstService "{x: 1.31 ,y: 1.0, z: 0.7}"
waiting for service to become available...
requester: making request: tutorial_interfaces.srv.FirstService_Request(x=1.31, y=1.0, z=0.7)

response:
tutorial_interfaces.srv.FirstService_Response(ex=-0.1739855855703354, ey=3.6158156394958496, ez=-0.699999988079071, ewx=0.0, ewy=0.0, ewz=2.309999942779541)
```

endeffector_to_joint:



```
swapneel@swapneel:~/rbe500/ghw3$ ros2 service call endeffector_to_joint tutorial_interfaces/srv/SecondService "{v_x: -0.17 ,v_y: 3.61, v_z: -0.7, w_x: 0.0, w_y: 0.0, w_z: 2.3}"
requester: making request: tutorial_interfaces.srv.SecondService_Request(v_x=-0.17, v_y=3.61, v_z=-0.7, w_x=0.0, w_y=0.0, w_z=2.3)

response:
tutorial_interfaces.srv.SecondService_Response(q1=1.309551477432251, q2=0.9904258251190186, q3=0.699999988079071)
```

As we can see in the screenshots above, the jacobian function is correct as the results are getting validated by running both the services.

Part 2: Velocity Controllers

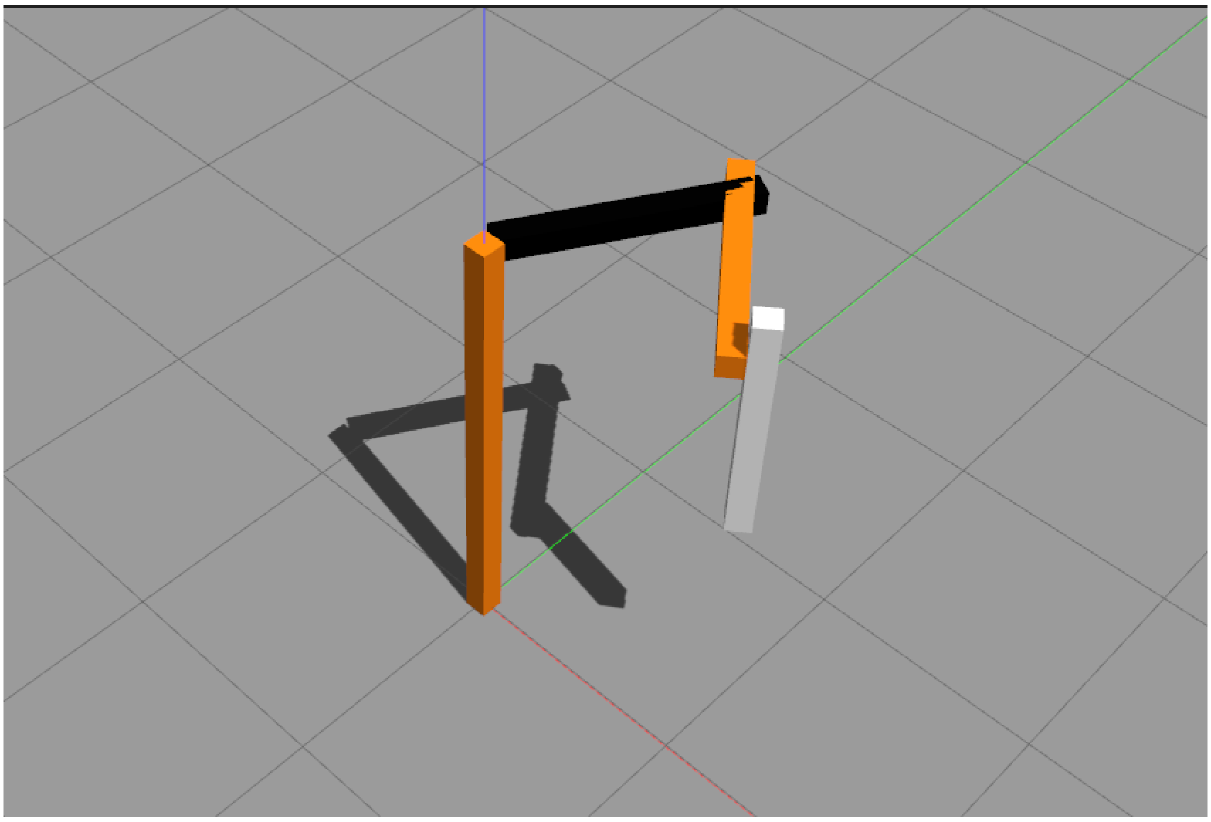
In this part of the assignment, the velocity controller is implemented. The node created in this part subscribes to joint velocities from the gazebo, get reference velocity values from service and calculate the efforts.

In service_callback function, reference velocities for joints are requested from terminal first and then these values are returned as response from here.

In listener_callback function, execution is done to read the joint velocities from gazebo and efforts for each joint are calculated. K_p and K_d values are also assigned in this function. [effort=K_p*error + K_d*error_dot]. The effort is published in the same function to the topic '/forward_effort_controller/commands'. Gazebo listenes to these commands and moves the bot accordingly.
Some changes were made in the urdf, where revolute joint and velocity limits were increased such that the robot would be able to move freely.

It is observed that the joints are moving with the same velocity when we run the code. It finally stabilizes with the reference velocity values.

The robot is spawned while running the code. (screenshot attached)



The robot was giving jerk in gazebo before tuning the K_p and K_d values. For tuning, we have tried different sets of K_p and K_d values by trial and error method.
Our final K_p and K_d values for all 3 joints are:
K_p = [4.2, 4, 10]
K_d = [0.4, 0.5, 1]

Part 3:

For part three of the group assignment our task was to make the gazebo simulation follow a straight line, we do this by making use of our second service which converts end effectors to joint velocities. End effector values were taken such that the cartesian Y axis would have a positive value while X and Z remain zero. After the step was done, we use the end effector values as our reference values for the gazebo simulation.

We observed that the robot was following a close enough straight path for some instances iteratively, however some jerk was still seen. But we couldn’t manage to get the consistent results.

The video is attached in the submission. Though it is having jerk in between, there was an issue with native video recording software.