RBE500: Foundations of Robotics

Group Assignment Part 2

Student 1: Dhruv Agrawal
Student 2: Shreyas Khobragade
Student 3: Pranjali Rangnekar
Student 4: Roman Henry

1 Questions

Taking the DH frames and the DH parameters from the previous assignment, we have

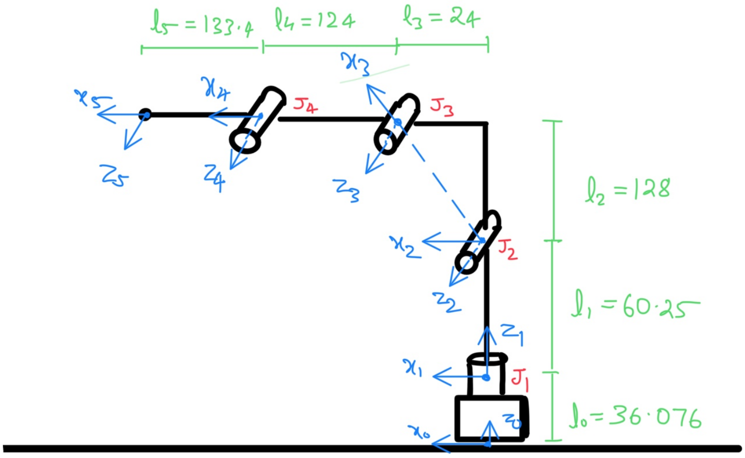

Figure 1: Robot

The DH-parameter table is as follows:

	a	θ	d	α
Link1	0	q_1	l_1	-90
Link2	$\sqrt{l_2^2 + l_3^2}$	$-\arctan\left(\frac{l_2}{l_3}\right) + q_2$	0	0
Link3	l_4	$\arctan\left(\frac{l_2}{l_3}\right) + q_3$	0	0
Link4	l_5	q_4	0	0

Table 1: DH parameters

1. **Velocity Level Kinematics: Implement a node with two services. One takes joint velocities and converts them to end effector velocities, and the second one takes end effector velocities and converts them to joint velocities.**
   Below are the steps taken while implementing the given problem.
   1) Create a new package named grp_assn_2
   2) Create two new files joint_vel_srv.py and end_eff_vel.py

   - **joint_vel_srv.py**
     - Takes in joint velocities and gives back the end effector Twist
     - Created a service message named JointVel.srv in the custom service folder
     - The service message takes Float32Multiarray messgae type as input and provides a Twist message type as an output
     - Created a subscriber that requests Float32Multiarray message type for joint position values

- Created a function that calculates transformation matrix from the given DH parameters for all the frames
- Calculated the origin vectors and z vectors for all the joint frames and the end effector frame
- Calculated the Jacobians for all the joints affecting the end effector and combined them to form the Jacobian matrix
- Calculated the end effector Twist by multiplying the Jacobian Matrix and the joint velocities vector

- **end_eff_vel.py**
  - Takes in end effector Twist and gives back joint velocities
  - Created a service message named Twist.srv the custom service folder
  - The service message takes Twist message type as input and provides a Float32Multiarray message type as an output
  - Created a subscriber that requests Float32Multiarray message type for joint poisiton values
  - Created a function that calculates transformation matrix from the given DH parameters for al frames
  - Calculated the origin vectors and z vectors for all the joint frames and the end effector frame
  - Calculated the Jacobians for all the joints affecting the end effector and combined them to form the Jacobian matrix
  - Calculated the Pseudo-inverse of Jacobian matrix using Numpy
  - Calculated the joint velocities by multiplying the Pseudo-inverse of the Jacobian Matrix and the end effector Twist

Figure 2 shows the terminal output of the Joint Velocities to Twist calculation and Twist to Joint Velocity calculation.

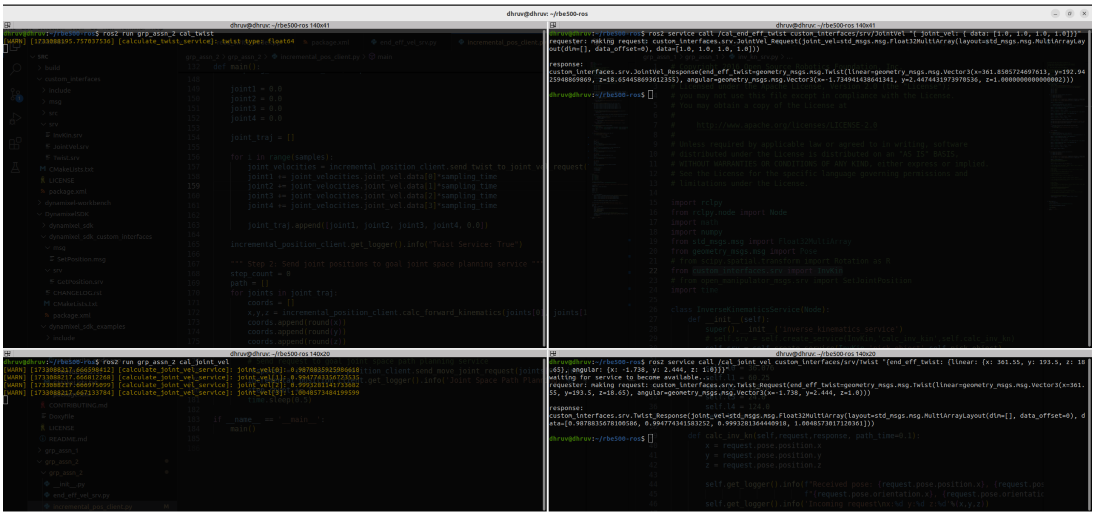

Figure 2: Terminal Output

2. **Write a node that provides incremental position references to the robot joints, i.e.**

$$q\_ref = q\_ref\_old + delta\_q * sampling\_time$$

The node would then send the q_ref to the joint position controllers of the robot as joint goals. You will use this node just like a velocity controller as follows in the next item.

3. **Give a constant velocity reference in the positive 'y' direction of the Cartesian space. Convert this velocity to the joint space velocities using your Jacobian and feed it as a reference (delta_q) to your incremental position reference node. This should make the robot move on a straight line in the +y direction. Record the followed cartesian position from the output of your forward kinematic node, and plot it with respect to time (you can use easy plotting functions of Matlab).**
Record a video of the robot moving on a line and include in the report (please be sure to shrink the video size so that it does not create submission problems).

For the given problem, the steps taken to achive the solution are as follows:

- Created a new node named IncrementalPositionClient
- Created a new client to request the SetJointPosition service
- Created a new client to call the cal_joint_vel service created before which takes in end effector Twist and provides joint velocities
- Developed a function to calculate forward kinematics from the current joint values
- In the main function, hard coded the Twist of the the end effector and calculated the joint velocities and updated the joint position using the given formula

$$q_{ref} = q_{ref\_old} + \Delta q * sampling\_time$$

- Calculated the forward kinematics from the new joint position values and saved the end effector position in an array.
- Plotted the end effector position vs time.

Figure 3 shows the terminal output of the incremental velocity node.
Top right terminal window shows the joint output values from the service cal_joint_vel which takes in end effector twist and gives out joint velocities and the bottom right terminal window shows the output while running the incremental_pos_client node.

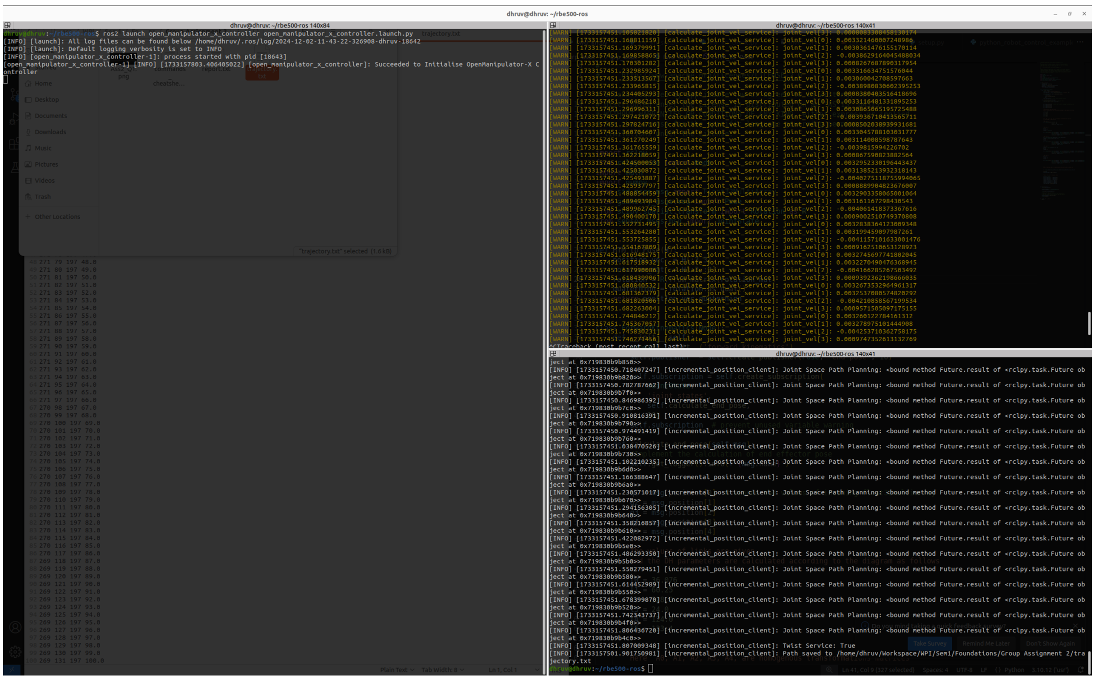

Figure 3: Terminal Output

The trajectory of the end effector of the robot is saved in tractory.txt file and plotted with respect to time.
Figure 4 shows three plots **x vs time**, **y vs time** and **z vs time**.
The number of samples are 100 and sampling time is taken as 1 second.
The end effector velocity in y-direction is taken as 1.0.

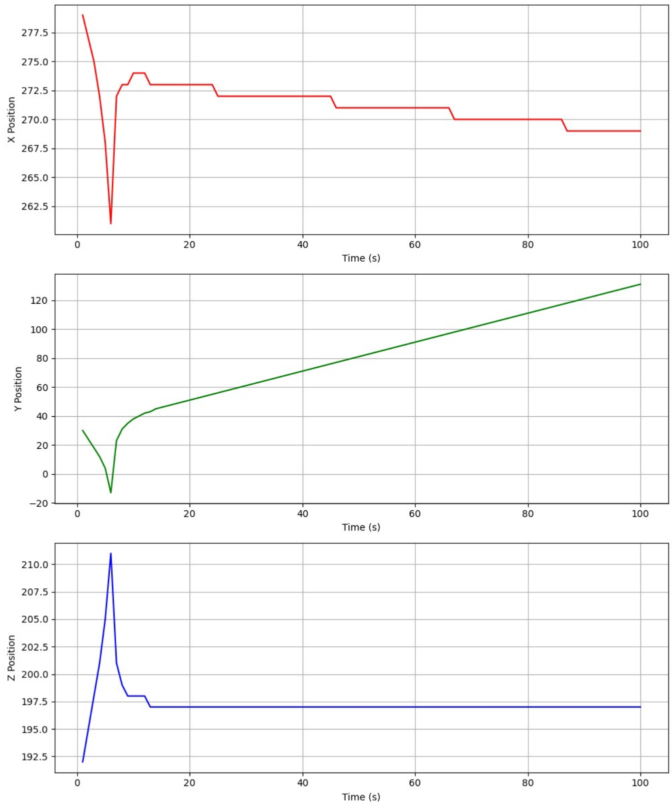

Figure 4: Terminal Output

From the plot it can be seen that the robot is moving in y-direction in linear fashion to time.
The z-direction values are also stable after a while and the x-values are also approximately stable.

The video of the robot moving in the y-direction has been attached in the submssion files along with the report.