RBE 500 Group Assignment #3

Joshua Gross, Arjan Gupta, Melissa Kelly

Velocity Kinematics

Calculations

Solving for both the forward and inverse velocity kinematics are centered around solving the Jacobian matrix. In order to solve for the Jacobian, we first need to establish the linear and angular velocity components in all directions for each of the joints. These expressions will vary dependent upon whether the joint is revolute or prismatic. Once these determinations are made, a matrix of the form seem in the figure below is created, where, each of the elements is a 3×1 matrix. This matrix comes from the lecture slides.

$$J = \begin{bmatrix} z_0 \times (o_3 - o_0) & z_1 \times (o_3 - o_1) & z_2 \\ z_0 & z_1 & 0 \end{bmatrix}$$

As clearly seen in the above figure, the following elements are needed in order to solve the Jacobian in its entirety: $z_0, z_1, z_2, o_0, o_1, o_3$. These values are found by using elements of the individual transformation matrices found when solving the forward position kinematics. The image below shows what elements of the transformation matrix are extracted in order to solve for Jacobian.

ROS Code

Programmatically, the elements $z_0, z_1, z_2, o_0, o_1, o_3$ mentioned in the Calculations subsection above are extracted and plugged into the Jacobian framework as seen below from the `velocity_kinematics.py` file (which is submitted as part of the `velocity_kin` package).

```
1          # Get z's and o's for Jacobian
2          z0 = np.array([[0], [0], [1]])
3          z1 = A1[:3, 2:3]
4          z2 = A2[:3, 2:3]
5          O0 = np.array([[0], [0], [0]])
6          O1 = A1[:3, 3:4]
7          O3 = A3[:3, 3:4]
8          # Write columns of Jacobian
9          E11 = np.cross(z0, (O3-O0), axis=0)
10         E12 = np.cross(z1, (O3-O1), axis=0)
11         E13 = z2
12         E21 = z0
13         E22 = z1
14         E23 = np.array([[0], [0], [0]])
15         # Build up the Jacobian
16         Jacobian = np.zeros((6,3), dtype=float)
17         Jacobian[:3, :1] = E11
18         Jacobian[:3, 1:2] = E12
19         Jacobian[:3, 2:3] = E13
20         Jacobian[3:6, :1] = E21
21         Jacobian[3:6, 1:2] = E22
22         Jacobian[3:6, 2:3] = E23
```

At this point, we have the ability to solve for the forward or inverse velocity kinematics. By multiplying the Jacobian by the joint's velocities matrix, we can solve for the velocity of the end effector. Likewise, if the inverse of the Jacobian is multiplied by the end effector velocity matrix, the velocity of each of the joints can be found. It should be noted that since this particular Jacobian is not square, the inverse cannot simply be taken and therefore the pseudo-inverse numpy function is used in order to take the inverse. Having the Jacobian solved and regularly available allows us to calculate the forward and inverse velocity kinematics and therefore create a controller for the robot so achieve the desired movement and speed.

Velocity Controller

The process of creating a controller for this application stems from the previous position controller as well as other controllers created throughout the course. The goal velocity of each joint, V_r is is received through a service response. The name of the service is `velocity_inv_kin_service`. Suppose our current velocity is given as V. The error for each joint is then calculated as follows:

$$E = V_r - V$$

When paired with the proportional gain value, we now have one of the terms of the controller. In order to get the other term, the derivative of the error is also needed, and is calculated as shown:

$$\dot{E} = \frac{V - V_{prev}}{\Delta time}$$

This derivative of the error is paired with the derivate gain and now completes the controller. Our controller takers on the form:

$$F = K_p E + K_d \dot{E}$$

Where F is the effort that will be applied in Gazebo and will cause the joints to move.