CS 133a **October 20, 2023**

Problem Set 3

Problem 1 (Orientation of 3DOF Pan/Tilt/Roll Gimbal) - 18 points

part (a)

$$^0|R_t = Rot_z(\theta_{pan})Rot_x(\theta_{tilt})Rot_y(\theta_{roll}) = \begin{bmatrix} 0.5543 & -0.4699 & 0.6870 \\ 0.1123 & 0.8601 & 0.4976 \\ -0.8247 & -0.1987 & 0.5295 \end{bmatrix}$$

part (b) Using the output direction we have find θ_{pan} and θ_{tilt}.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = {}^0|R_t \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

First solution, let $R_{y_{ij}}$ be the element in $Rot_y(\theta_{roll})$ that is in the ith row and jth column of $Rot_y(\theta_{roll})$:

$$\theta_{pan} = atan2(-x, y)$$

$$\theta_{tilt} = atan2(z, \sqrt{x^2 + y^2})$$

$$Rot_y(\theta_{roll}) = (Rot_x(\theta_{tilt}))^T (Rot_z(\theta_{pan}))^T \, {}^0|R_t$$

$$\theta_{roll} = atan2(R_{y_{13}}, R_{y_{11}})$$

The second solution is similar:

$$\theta_{pan} = atan2(-x, y) - \pi$$

$$\theta_{tilt} = \pi - atan2(z, \sqrt{x^2 + y^2})$$

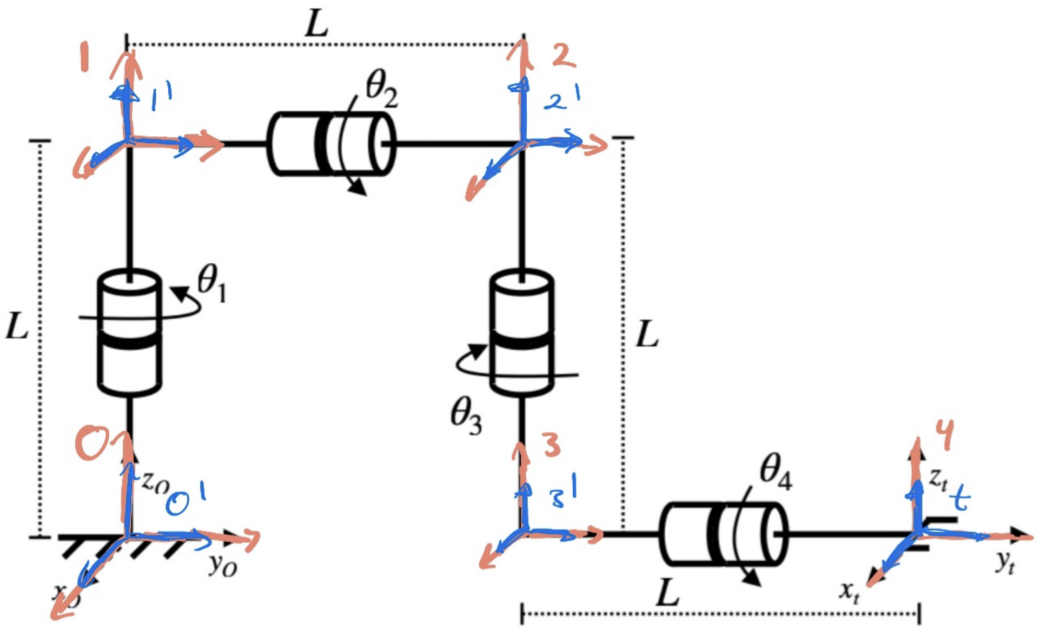
$$Rot_y(\theta_{roll}) = (Rot_x(\theta_{tilt}))^T (Rot_z(\theta_{pan}))^T \, {}^0|R_t$$

$$\theta_{roll} = atan2(R_{y_{13}}, R_{y_{11}})$$

The first solution (in radians) gives $(\theta_{pan}, \theta_{tilt}, \theta_{roll}) = (1.04716, 0.78536, -2.09439)$
The second solution gives $(\theta_{pan}, \theta_{tilt}, \theta_{roll}) = (-2.09443, 2.35623, 1.04720)$

Problem 2 (Forward Kinematics of 4DOF Robot) - 25 points:

part(a)



There is only translation going from the blue to red reference frames. Likewise, there is in only rotation going from the red to blue reference frames (aside from the last one $4 \rightarrow t$). For the exact transformations, refer to part b. Note that for simplicity, all frames above are drawn in the configuration where all angles are 0 ($\theta_1 = \theta_2 = \theta_3 = \theta_4 = 0$).

part (b) Note that $L = 0.4$ as given in the problem.

Col1	Translation	Rotation
$O \to O'$	${}^{O}\|p_{O'} = \begin{bmatrix}0\\0\\0\end{bmatrix}$	${}^{O}\|R_{O'} = Rot_z(\theta_1)$
$O' \to 1$	${}^{O'}\|p_1 = \begin{bmatrix}0\\0\\L\end{bmatrix}$	${}^{O'}\|R_1 = I$
$1 \to 1'$	${}^{1}\|p_{1'} = \begin{bmatrix}0\\0\\0\end{bmatrix}$	${}^{1}\|R_{1'} = Rot_y(\theta_2)$
$1' \to 2$	${}^{1'}\|p_2 = \begin{bmatrix}0\\L\\0\end{bmatrix}$	${}^{1'}\|R_2 = I$
$2 \to 2'$	${}^{2}\|p_{2'} = \begin{bmatrix}0\\0\\0\end{bmatrix}$	${}^{2}\|R_{2'} = Rot_z(-\theta_3)$
$2' \to 3$	${}^{2'}\|p_3 = \begin{bmatrix}0\\0\\-L\end{bmatrix}$	${}^{2'}\|R_3 = I$
$3 \to 3'$	${}^{3}\|p_{3'} = \begin{bmatrix}0\\0\\0\end{bmatrix}$	${}^{3}\|R_{3'} = Rot_y(\theta_4)$
$3' \to 4$	${}^{3'}\|p_4 = \begin{bmatrix}0\\L\\0\end{bmatrix}$	${}^{3'}\|R_4 = I$
$4 \to t$	${}^{4}\|p_t = \begin{bmatrix}0\\0\\0\end{bmatrix}$	${}^{4}\|R_t = I$

part (c)
Using the rotation and translations as described in part B, we get that:
The position of the tip is $\begin{bmatrix} -0.13789374 \\ 0.82779169 \\ 0.44494897 \end{bmatrix}$.
The orientation of the tip relative to the frame world frame O is described by the rotation matrix:

$$\begin{bmatrix} 0.95570527 & -0.25 & -0.15533009 \\ 0.08967987 & 0.75 & -0.65533009 \\ 0.28033009 & 0.61237244 & 0.73919892 \end{bmatrix}$$

Calculation done in python using function shown on the next page.

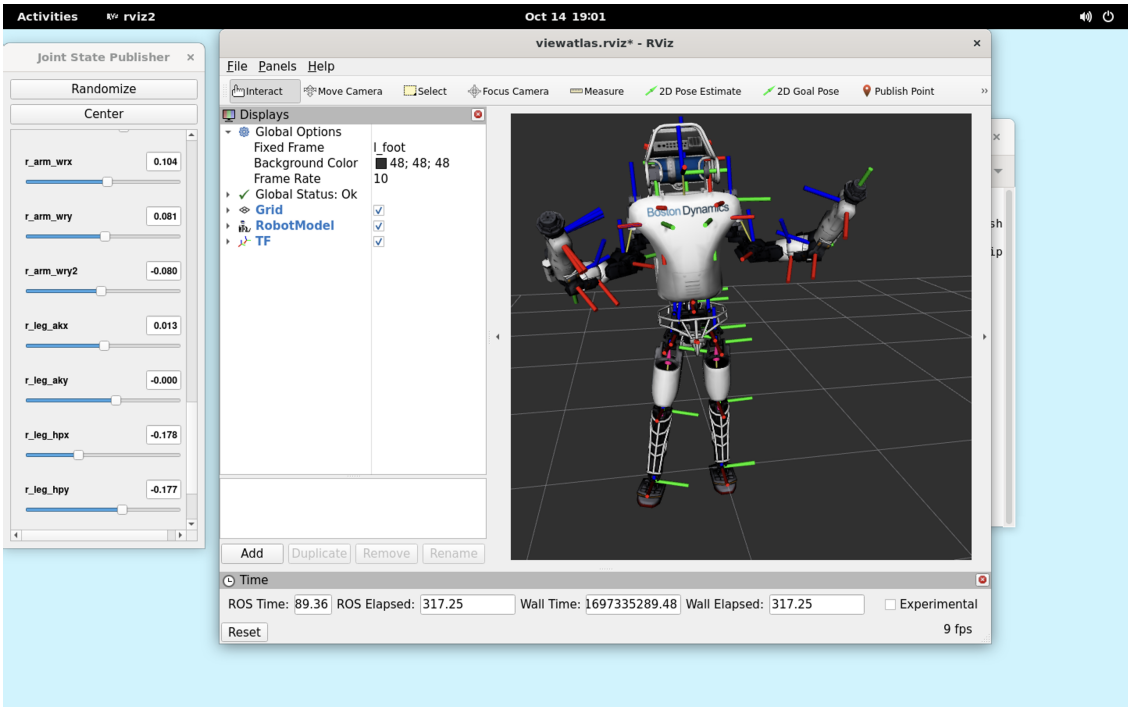
part (d)
Using the rotation matrix of frame t with respect to frame O, we get that the unit vector of the x axis of frame t relative to frame O $\begin{bmatrix} 0.95570527 \\ 0.08967987 \\ 0.28033009 \end{bmatrix}$
Therefore the angle is as calculated:

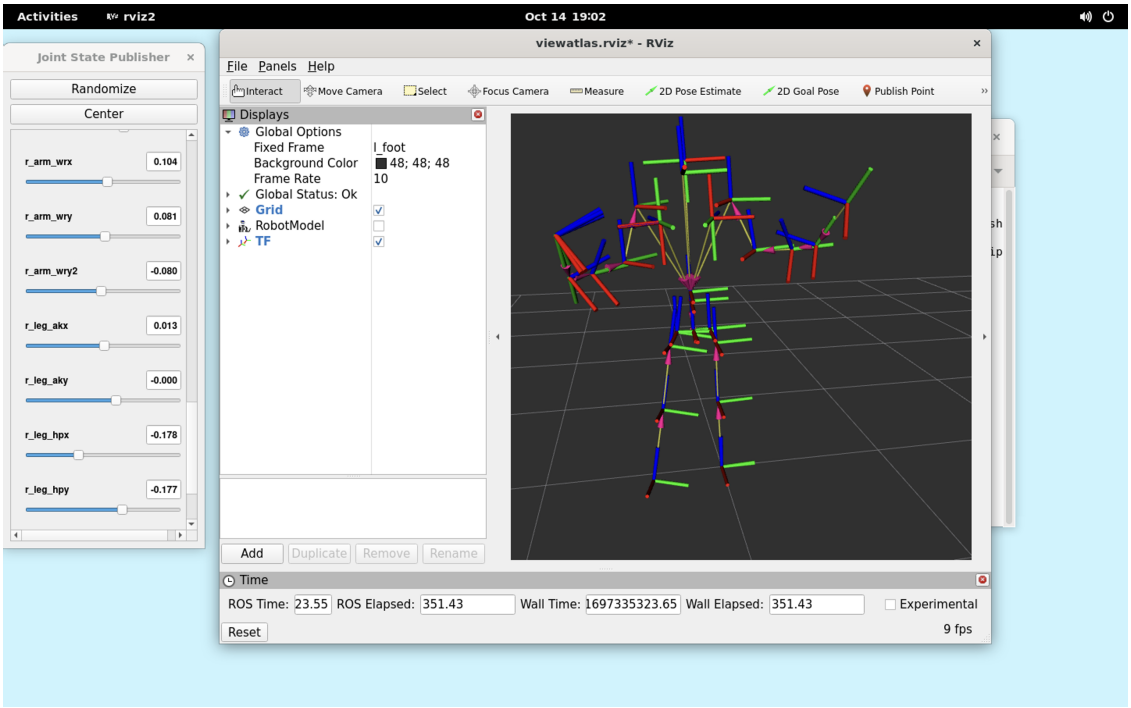
$$angle = atan2(0.28033009, \sqrt{0.95570527^2 + 0.08967987^2}) = 0.28413\ radians = 16.28^\circ$$

Calculation done in python using function shown on the next page.

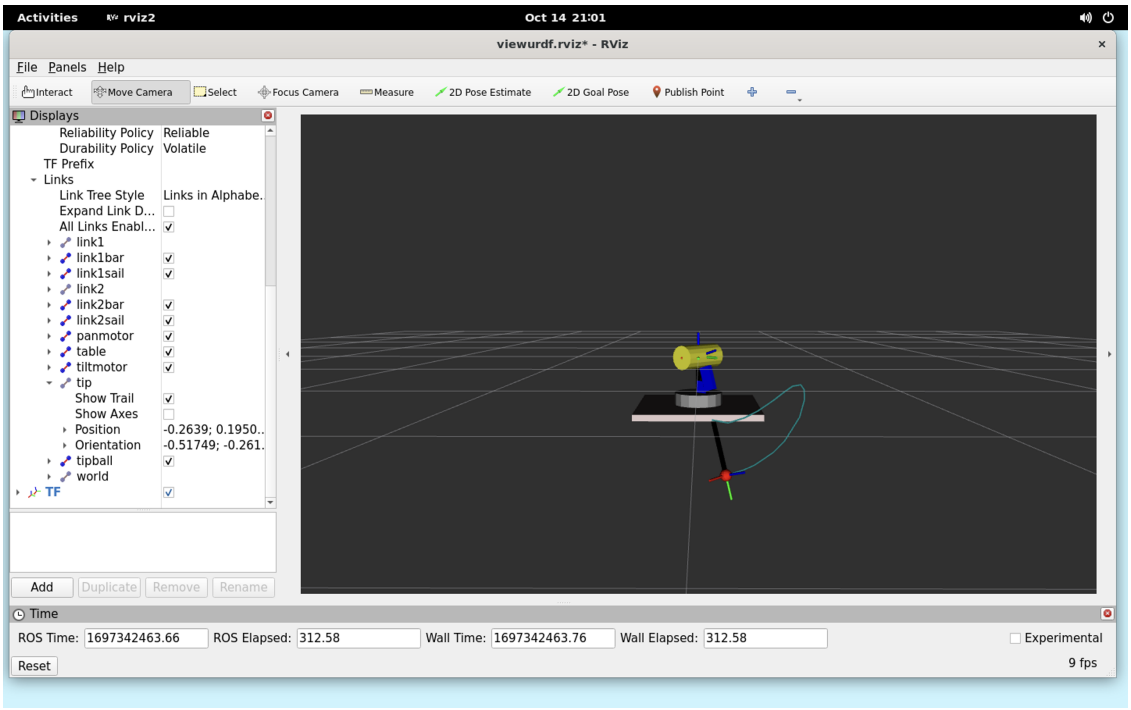
```
def problem_2c(theta_1, theta_2, theta_3, theta_4):
    pos = np.array([0,0,0,1])
    bottom_row = np.array([0, 0, 0, 1])
    L = 0.4

    # O -> O'
    T_0 = np.vstack((np.hstack((rot_z(theta_1), zero)), bottom_row))

    # O' -> 1
    p = np.array([[0], [0], [L]])
    T_1 = np.vstack((np.hstack((I, p)), bottom_row))

    # 1 -> 1'
    T_2 = np.vstack((np.hstack((rot_y(theta_2), zero)), bottom_row))

    # 1' -> 2
    p = np.array([[0], [L], [0]])
    T_3 = np.vstack((np.hstack((I, p)), bottom_row))

    # 2 -> 2'
    T_4 = np.vstack((np.hstack((rot_z(-theta_3), zero)), bottom_row))

    # 2' -> 3
    p = np.array([[0], [0], [-L]])
    T_5 = np.vstack((np.hstack((I, p)), bottom_row))

    # 3 -> 3'
    T_6 = np.vstack((np.hstack((rot_y(theta_4), zero)), bottom_row))

    # 3' -> 4
    p = np.array([[0], [L], [0]])
    T_7 = np.vstack((np.hstack((I, p)), bottom_row))

    # 4 -> t
    T_8 = np.vstack((np.hstack((I, zero)), bottom_row))

    T = T_0 @ T_1 @ T_2 @ T_3 @ T_4 @ T_5 @ T_6 @ T_7 @ T_8
    print(T)

    tip_pos = T @ pos
    print("Print Tip pos: {}".format(tip_pos))
    x_axis = np.array([T[0,0], T[1,0], T[2,0]])
    #print("i hat = {}".format(x_axis))
    angle_above_xy = np.arctan2(x_axis[2], np.sqrt(x_axis[0]**2 + x_axis[1]**2))
    print("Angle above xy plane: {} rads = {} deg".format(angle_above_xy,
    angle_above_xy * 180/np.pi))
```

Problem 3 (Atlas Dance Session with Explicit Commands) - 10 points





Problem 4 (Animate the Pan/Tilt Gimbal) - 18 points:



Relevant Code:

```
class Trajectory():
    # Initialization.
    def __init__(self):
        pass

    # Declare the joint names.
    def jointnames(self):
        # Return a list of joint names MATCHING THE JOINT NAMES IN THE URDF!
        return ['pan', 'tilt']

    # Evaluate at the given time.
    def evaluate(self, t, dt):
        # Compute the joint values.  We could build up numpy arrays as
        # position and velocity vectors, but just compute the values:
        theta_pan  = (np.pi/3) * np.sin(2*t)
        omega_pan  = (2*np.pi/3) * np.cos(2*t)

        theta_tilt = (np.pi/3) * np.sin(t) - (np.pi/9) * np.cos(6*t)
        omega_tilt = (np.pi/3) * np.cos(t) + (6*np.pi/9) * np.sin(6*t)

        # Return the position and velocity as python lists.
        return ([theta_pan, theta_tilt], [omega_pan, omega_tilt])
```

Problem 5 (Trajectory for 3DOF Multiplicity) - 25 points:

part (a)
Inverse kinematics from last set:
Let $\vec{P} = (x, y, z)$ Then we have that (since θ_{pan} is the angle about the yz plane):

$$\theta_{pan} = atan2(-x, y)$$

Note that the distance between P and the z-axis is $\sqrt{x^2 + y^2}$,and the distance between P and the xy plane is simply z. Therefore we have (this follows from the geometry explained in the notes):

$$\theta_2 = \pm acos\left(\frac{(x^2 + y^2 + z^2) - l_1^2 - l_2^2}{2l_1 l_2}\right)$$

By geometry we also have that (this also follows from the notes):

$$\theta_1 = atan2(z, \sqrt{x^2 + y^2}) - atan2(l_2 \cdot sin(\theta_2), l_1 + l_2 \cdot cos(\theta_2))$$

Putting this all together, for $\vec{P} = (x, y, z)$ we have:

$$\theta_{pan} = atan2(-x, y) + 2k_1\pi$$

$$\theta_2 = \pm acos\left(\frac{(x^2 + y^2 + z^2) - l_1^2 - l_2^2}{2l_1 l_2}\right)$$

$$\theta_1 = atan2(z, \sqrt{x^2 + y^2}) - atan2(l_2 \cdot sin(\theta_2), l_1 + l_2 \cdot cos(\theta_2)) + 2k_3\pi$$

where $k_1, k_2, k_3 \in \mathbb{Z}$. Note that the equations above provide us with 2 configurations of the form $(\theta_{pan}, \theta_1, \theta_2)$. For each configuration of the form $(\theta_{pan}, \theta_1, \theta_2)$, there is another solution $(\theta_{pan} \pm \pi, \pi - \theta_1, -\theta_2)$. Thus we have four solutions in total.

From the inverse kinematics of the last set we have the four solutions (in radians):

$$(\theta_{pan}, \theta_1, \theta_2) = (0.6747, 0.5082, 0.6435)$$

$$(\theta_{pan}, \theta_1, \theta_2) = (0.6747, 1.1517, -0.6435)$$

$$(\theta_{pan}, \theta_1, \theta_2) = (-2.4669, 2.6334, -0.6435)$$

$$(\theta_{pan}, \theta_1, \theta_2) = (-2.4669, 1.9899, 0.6435)$$

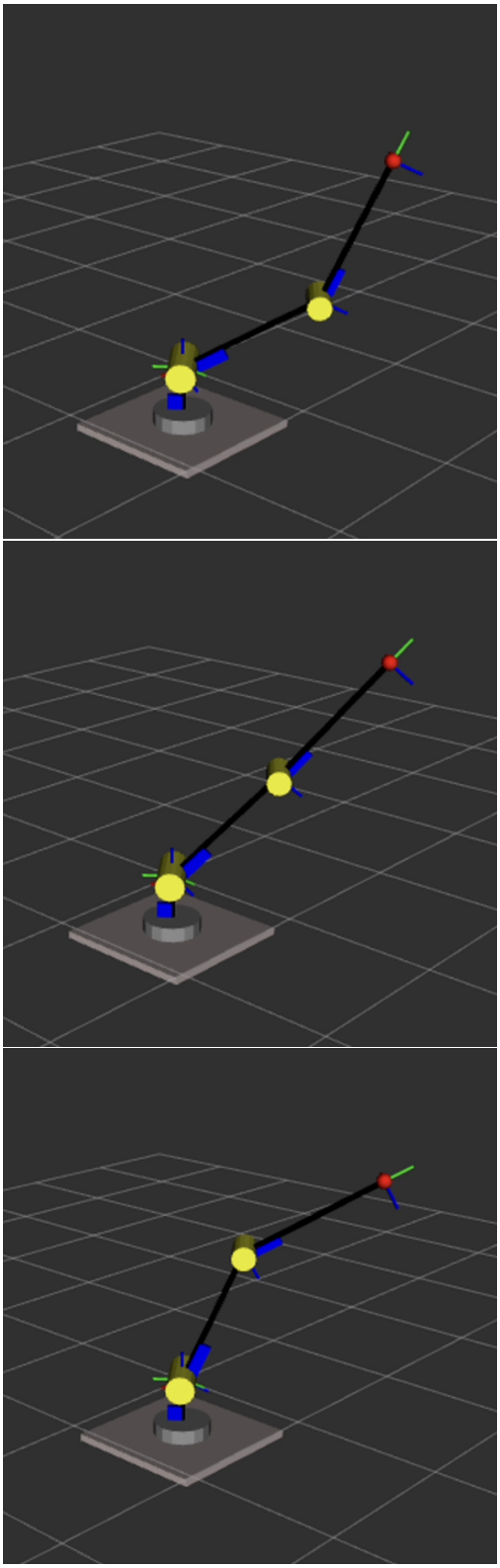
part (b)
Relevant Code:

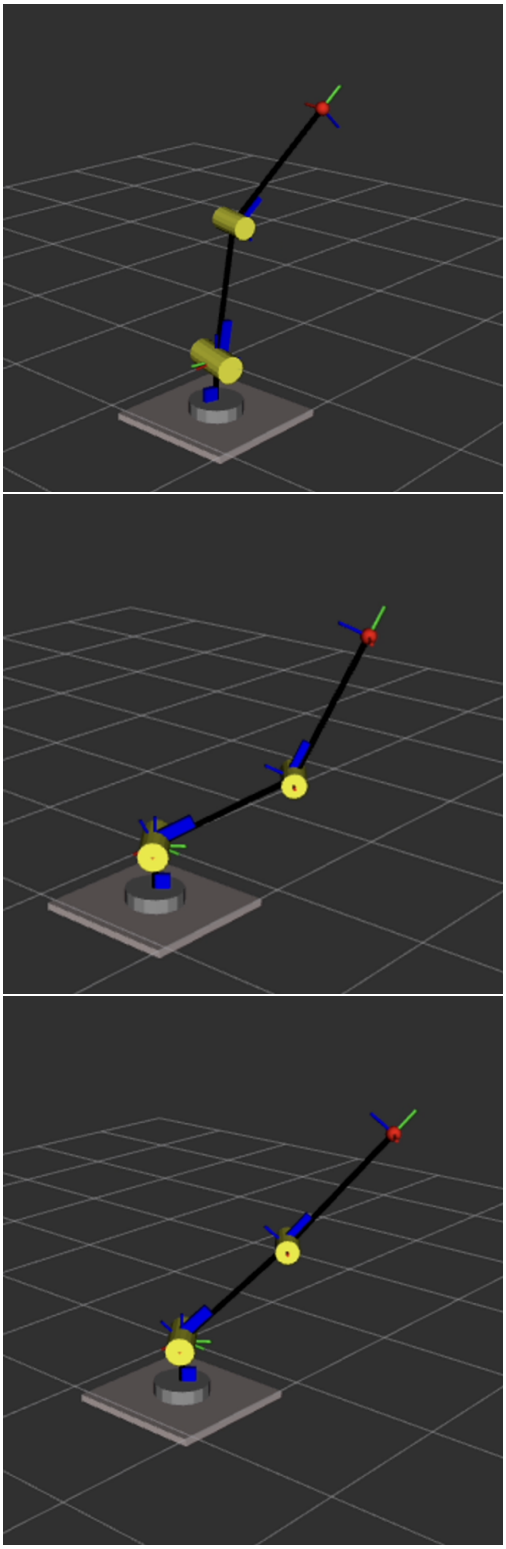
```
class Trajectory():
    # Initialization.
    def __init__(self):
        #### PRECOMPUTE ANY DATA YOU MIGHT NEED.
        self.init_pos = True
        self.timer = 0
        self.flag = 0
        self.wait = False

        # The four solutions
        self.A = [0.6747409422235527, 0.5081507901910796, 0.6435011087932847]
        self.B = [0.6747409422235527, 1.1516518989843643, -0.6435011087932847]
        self.C = [-2.4668517113662403, 2.633441863398714, -0.6435011087932847]
        self.D = [-2.4668517113662403, 1.9899407546054289, 0.6435011087932847]

        self.q_lst = [self.A, self.B, self.C, self.D]

        self.qdot_AB = [self.B[0]-self.A[0], self.B[1]-self.A[1], self.B[2]-self.A[2]]
        self.qdot_BC = [self.C[0]-self.B[0], self.C[1]-self.B[1], self.C[2]-self.B[2]]
        self.qdot_CD = [self.D[0]-self.C[0], self.D[1]-self.C[1], self.D[2]-self.C[2]]
        self.qdot_DA = [self.A[0]-self.D[0], self.A[1]-self.D[1], self.A[2]-self.D[2]]

        self.qdots = [self.qdot_AB, self.qdot_BC, self.qdot_CD, self.qdot_DA]

    # Declare the joint names.
    def jointnames(self):
        # Return a list of joint names
        #### YOU WILL HAVE TO LOOK AT THE URDF TO DETERMINE THESE! ####
        return ['theta1', 'theta2', 'theta3']

    # Evaluate at the given time.
    def evaluate(self, t, dt):
        #### COMPUTE THE POSITION AND VELOCITY VECTORS AS A FUNCTION OF TIME.
        #initialize robot to configuration A
        if self.init_pos:
            #set the position to A
            q = self.A
            qdot = [0.1,0.1,0.1]
            self.init_pos = False
            return (q,qdot)

        if self.wait:
            qdot = [0.1,0.1,0.1]
            q = self.q_lst[(self.flag + 1)%4]
            self.timer += dt
            if self.timer >= 0.50:
                self.wait = False
                self.flag = (self.flag + 1) % 4
                self.timer = 0
```

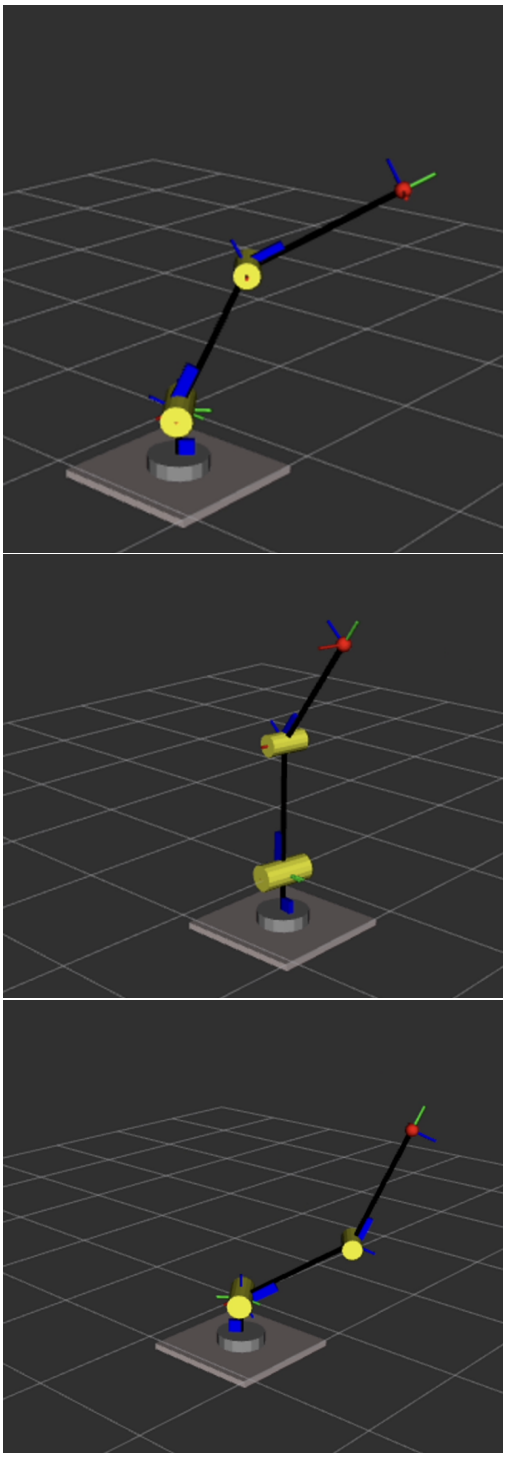
```
            return (q,qdot)

        else:
            self.timer += dt
            qdot = self.qdots[self.flag]
            q = []
            q.append(self.q_lst[self.flag][0] + qdot[0] * self.timer)
            q.append(self.q_lst[self.flag][1] + qdot[1] * self.timer)
            q.append(self.q_lst[self.flag][2] + qdot[2] * self.timer)

            if self.timer >= 1:
                q = self.q_lst[(self.flag + 1)%4]
                qdot = [0.1,0.1,0.1]
                self.wait = True
                self.timer = 0
                return (q,qdot)

            return (q,qdot)
```

Video: Here is a link to a video showing the program running (must sign into your Caltech email to view): bit.ly/3rW9Fyy

Screenshots: Shown on the next three pages, there are shown in series.







Problem 6 (Time Spent) - 4 points:

Time spent was about 6 hours. Did not have any significant bottlenecks.