Project 2: inverse kinematics and resolved rate control

Name: Puda Zhao
studentID: N13625214
netID: pz2078

Before Questions:

I saved the functions written in the previous homework in the "calcurobots.py" and import them in the form of modules. (import calcurobots as cr)

I save the functions written in the previous homework in the calcurobots.py file and import them in the form of modules.

In the same code block, I calculated and saved some basic data. In addition, the optimal solution of Q1 so far is included. I also store functions: forward kinematics and Jacobian here.

Question 1: inverse kinematics

The way to write down function compute_IK_position:

Step1. Get desired positions and divide them into way points

$$p_i^{des} = 0.1 * i * p^{des}$$

Step2. Set proper initial value for start $\boldsymbol{\theta_0}$ and step size $\boldsymbol{\varepsilon}$

$$\theta_0 \; is \; a \; 7 \times 1 \; vector, every \; element \in (-5,5)$$
$$\varepsilon \; is \; a \; constant, \varepsilon \in (0.2, 0.6)$$

Step3. We do an iterative algorithm for the ten waypoints in turn. in the beginning, $\boldsymbol{\theta = \theta_0}$.

(Here i represents the currently processed waypoint, j represents the number of iterations of the waypoint)

Set the θ_{result} of the last way point as a new start, $\theta = \theta_{result}$

When moving away from the starting point, lower the value of e to prevent overshoot

$$\varepsilon_i = \varepsilon_{i-1} * 0.95$$
$$p(\theta) = forward\_kinematics(\theta)$$
$$error = p_i^{des} - p(\theta)$$
$$\theta_{j+1} = \theta_j + \varepsilon_i * Jacobian^{+} * error$$

When the convergence value appears, save it.

The way to write down function compute_IK_position_nullspace:

Modify this formula, set $\bar{\theta} = [1,1,-1,-1,1,1,1]$

And $\theta_{j+1} = \theta_j + \varepsilon_i * (Jacobian^{+} * error + (I - Jacobian^{+} * Jacobian) * \bar{\theta}$

So far, the best solutions for the 10 given positions are: (Keep two decimal places)

Use compute_IK_position:

$$\theta = \begin{bmatrix} -13.7 & 1.23 & -3.35 & -12.1 & -1.28 & 30.3 & 12.1 & -2.37 & -0.29 & -3.41 \\ 13.6 & -5.9 & -6.17 & 12.6 & -6.15 & 75.2 & 30.7 & -5.46 & 6.15 & -0.61 \\ -10.6 & -3.44 & -0.49 & 13.6 & 0.58 & 10.1 & -23.8 & 3.79 & 3.35 & -0.06 \\ 12.6 & -5.28 & 4.81 & -12 & -1.33 & -88.1 & -18.7 & 0.5 & 1.4 & -5.47 \\ -12.9 & -2.73 & 5.85 & -42.2 & -0.77 & 42.7 & 43.1 & -0.2 & 2.53 & -2.8 \\ 31.4 & -5.62 & 1.62 & 7.9 & -4.83 & -24.9 & 150.7 & -0.68 & 15.45 & -5.42 \\ 24.1 & 3.26 & 7.22 & -18.4 & 2.51 & 51.8 & 10 & 2.88 & -3.94 & -0.03 \end{bmatrix}$$

$$\|error\|_2 = [0.286 \quad 0 \quad 0 \quad 0.016 \quad 0 \quad 0.077 \quad 0.380 \quad 0.11 \quad 0.18 \quad 0]$$

Use compute_IK_position_nullspace:

$$\theta = \begin{bmatrix} -13.7 & 1.23 & -3.35 & -12.1 & -1.28 & 30.3 & 12.1 & -2.37 & -0.29 & -3.41 \\ 13.6 & -5.9 & -6.17 & 12.6 & -6.15 & 75.2 & 30.7 & -5.46 & 6.15 & -0.61 \\ -10.6 & -3.44 & -0.49 & 13.6 & 0.58 & 10.1 & -23.8 & 3.79 & 3.35 & -0.06 \\ 12.6 & -5.28 & 4.81 & -12 & -1.33 & -88.1 & -18.7 & 0.5 & 1.4 & -5.47 \\ -12.9 & -2.73 & 5.85 & -42.2 & -0.77 & 42.7 & 43.1 & -0.2 & 2.53 & -2.8 \\ 31.4 & -5.62 & 1.62 & 7.9 & -4.83 & -24.9 & 150.7 & -0.68 & 15.45 & -5.42 \\ 24.1 & 3.26 & 7.22 & -18.4 & 2.51 & 51.8 & 10 & 2.88 & -3.94 & -0.03 \end{bmatrix}$$

$$\|error\|_2 = [0.286 \quad 0 \quad 0 \quad 0.016 \quad 0 \quad 0.077 \quad 0.380 \quad 0.11 \quad 0.18 \quad 0]$$

All the required functions and Exact solution can be found in codes.

Question 2: Joint control and joint trajectories generation

Step1. From given positions

$$p_{goal1} = [0.7 \quad 0.2 \quad 0.7]$$
$$p_{goal2} = [0.3 \quad 0.5 \quad 0.9]$$

Do inverse kinematics, we can get

$$\theta_{goal1} = \begin{bmatrix} 3.47929649 \\ -0.6415276 \\ 3.19288371 \\ 5.5332155 \\ 2.47376351 \\ 5.29469703 \\ 3.30724482 \end{bmatrix}$$

$$\theta_{goal2} = \begin{bmatrix} -4.85905387 \\ 0.37347647 \\ -0.4585807 \\ 5.5372262 \\ -3.80869246 \\ -7.35879002 \\ -2.9276608 \end{bmatrix}$$

Set start point

$$theta_0 = [0\ 0\ 0\ 0\ 0\ 0\ 0]$$

The way to write down function get_point_to_point_motion:

$$\theta_{des}(t) = \theta_{init} + (\frac{10}{T^3}t^3 + \frac{-15}{T^4}t^4 + \frac{6}{T^5}t^5)(\theta_{goal} - \theta_{init})$$

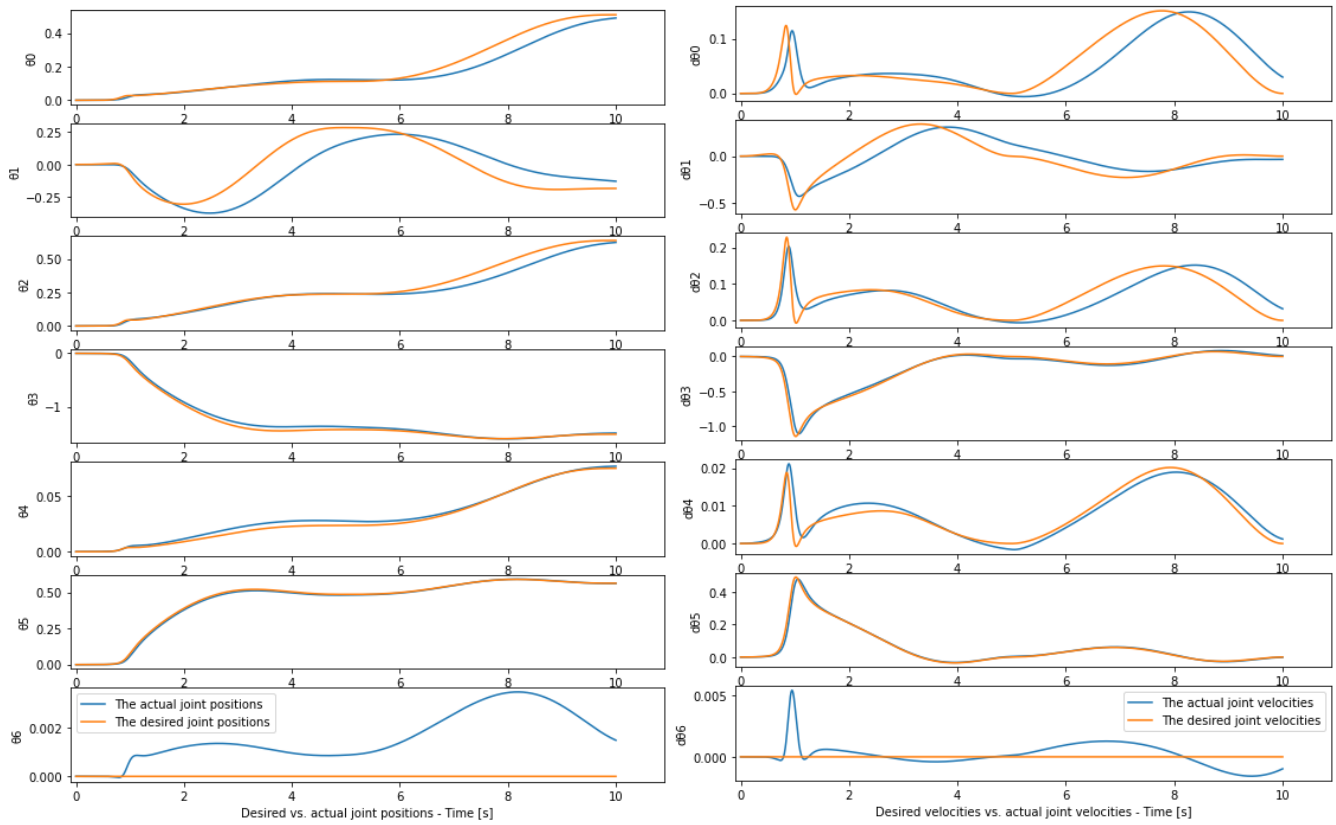
$$\dot{\theta}_{des}(t) = \theta_{init} + (\frac{30}{T^3}t^2 + \frac{-60}{T^4}t^3 + \frac{30}{T^5}t^4)(\theta_{goal} - \theta_{init})$$

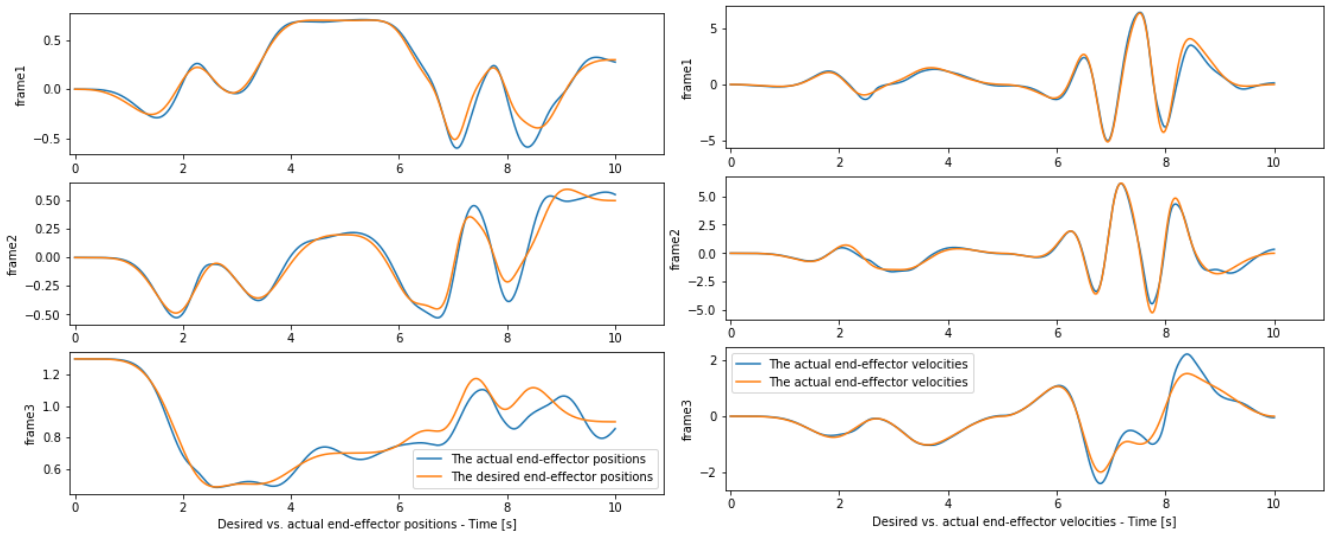
The core task of function robot_controller:

$$\tau_{des} = P\big(\theta_{des}(t) - \theta(t)\big) + D\left(\dot{\theta}_{des}(t) - \dot{\theta}(t)\right)$$

the required function has been posted in codes.

Plotting results:





Question 3: End-effector control

Here is the formula I use to write down the function:

Consider a time horizon with discrete time steps such that the interval between two consecutive steps is Δt. The following sequence of steps are repeated for as long as the robot's end-effector is required to move at the specified Cartesian velocity v:

Before start, we need to know $v(t)$. We can do this by function go_point_to_point:

$$p_0 = [0 \quad 0 \quad 1.301]$$
$$p_{goal1} = [0.7 \quad 0.2 \quad 0.7]$$
$$p_{goal2} = [0.3 \quad 0.5 \quad 0.9]$$

1. At each time step k, compute the kinematic Jacobian matrix $J(q_k)$ using the current values of the joint angles q_k.
2. Calculate the joint velocity vector $\dot{q}$ that must be achieved in the current time step using the equation
$$\dot{q} = J(q_k)^{-1} v$$
3. The joint angle displacement is then calculated as, here we set $\Delta t = 0.001$
$$q_{dist} = \dot{q}\Delta t$$

This quantity signifies how much we want to move each of the joints in the given time step based on the value of joint velocity.

4. The next joint configuration is calculated by
$$q_{k+1} = q_k + q_{dist}$$
5. The robot hardware is commanded to move the next joint configuration q_{k+1}.

The steps above are repeated for as long as necessary.

To add a nullspace:

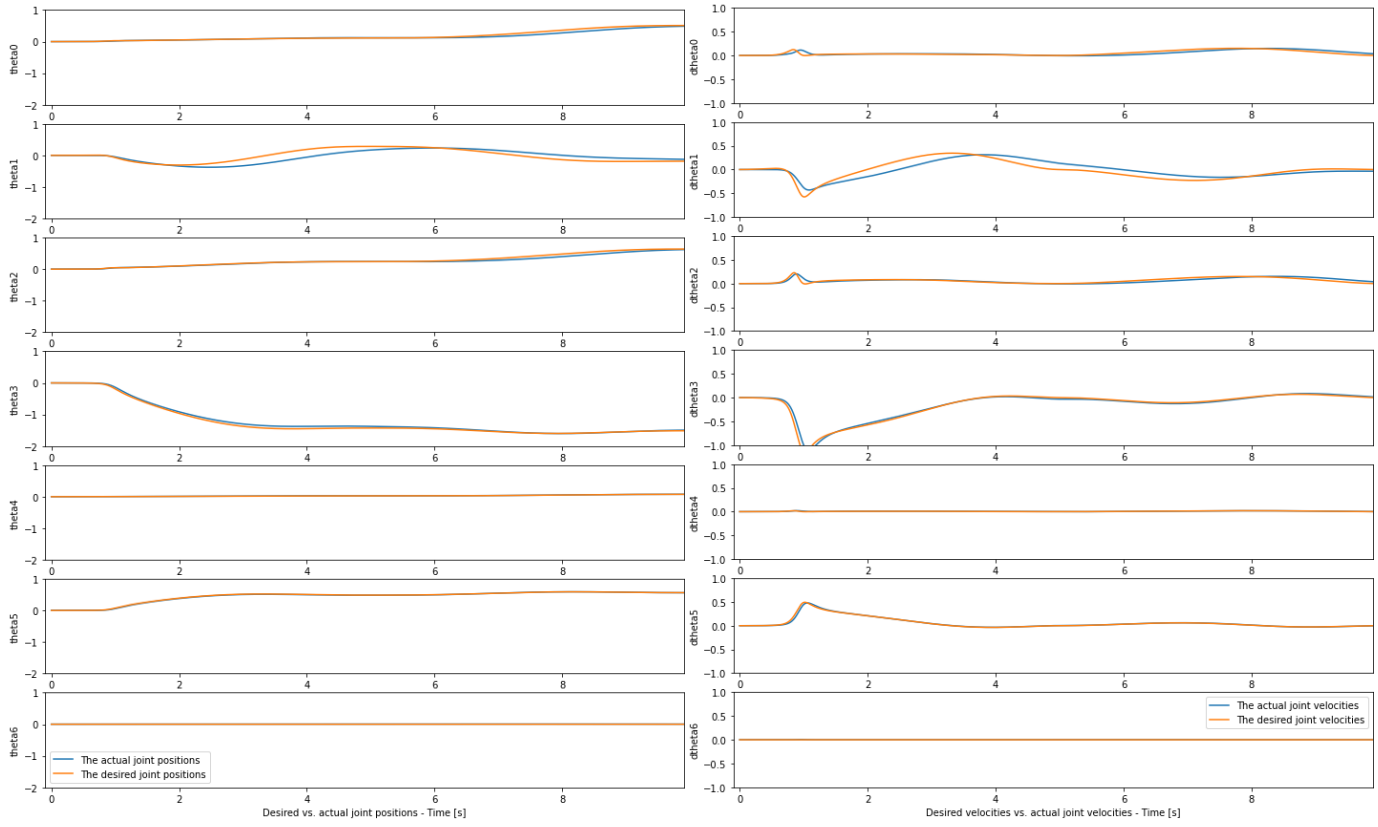
Modify here,

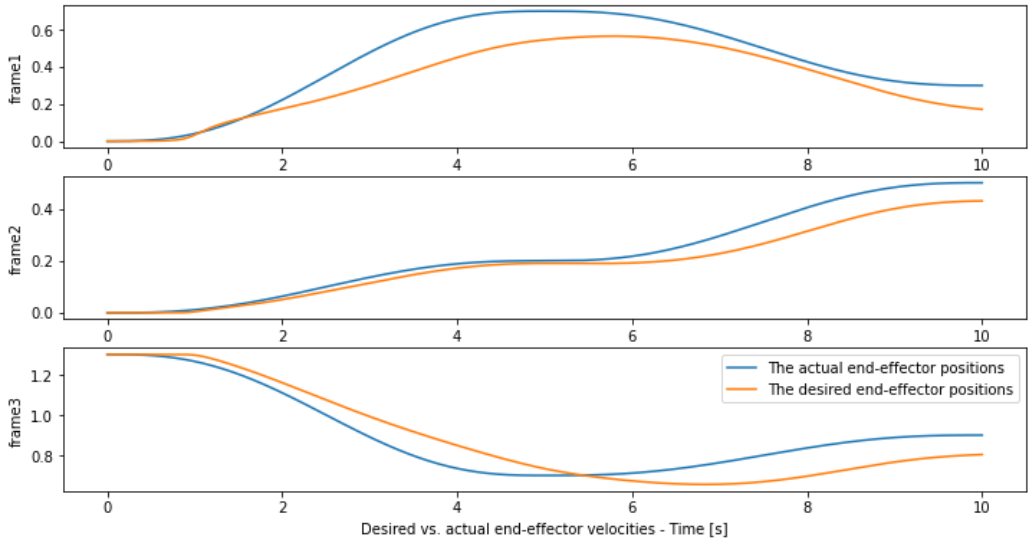
$$\dot{q} = J(q_k)^{-1} v + \left(I - J(q_k)^{-1} J(q_k)\right)\bar{\theta}$$

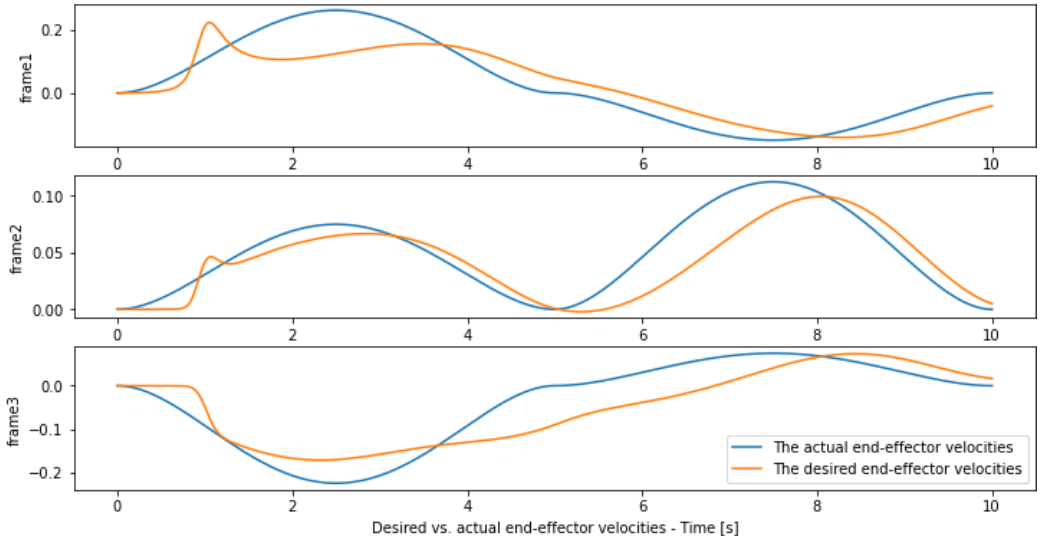
Set different $\bar{\theta}$ and try to find the best one, here is an example:

$$\bar{\theta} = [1,1,-1,-1,1,1,1]$$

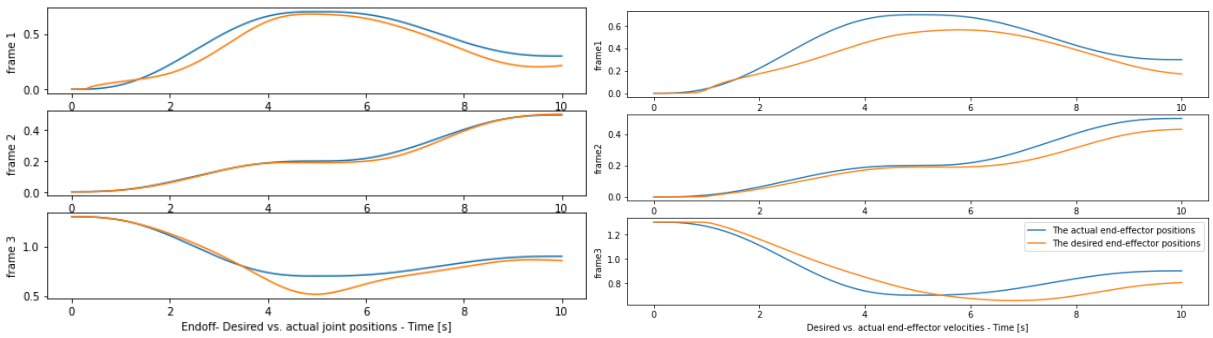
Plotting results:







By the way, after adding some P control in Q3, I found a better result (Compared to only add D control):



You see, in the left picture, the result after PD control is better that the right one.