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**

**Step2. Set proper initial value for start and step size**

**Step3. We do an iterative algorithm for the ten waypoints in turn. in the beginning, .**

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

Set the of the last way point as a new start,

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

When the convergence value appears, save it.

The way to write down function compute\_IK\_position\_nullspace:

Modify this formula, set

And

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

Use compute\_IK\_position:

Use compute\_IK\_position\_nullspace:

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

Question 2: Joint control and joint trajectories generation

Step1. From given positions

Do inverse kinematics, we can get

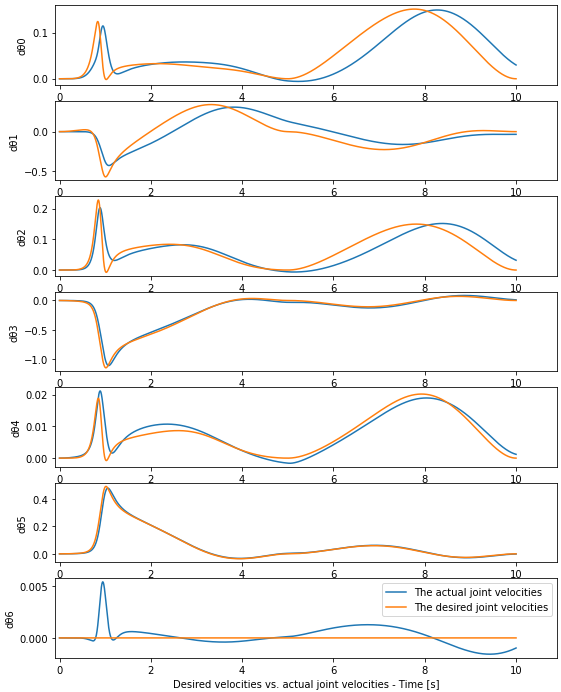
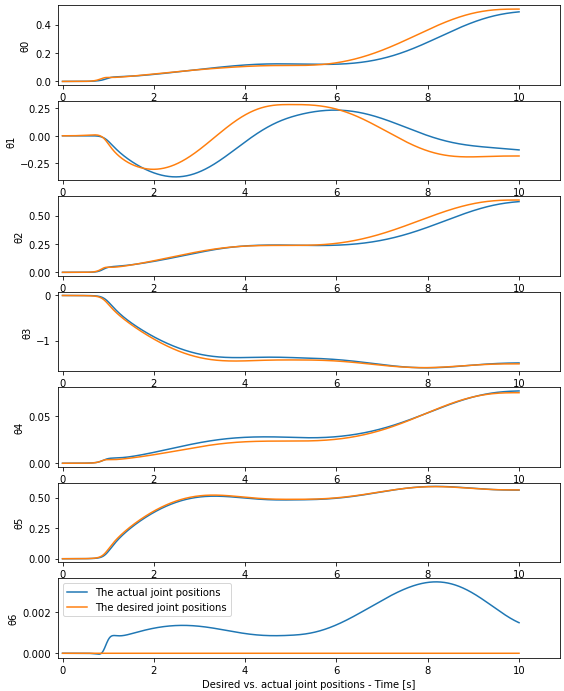
Set start point

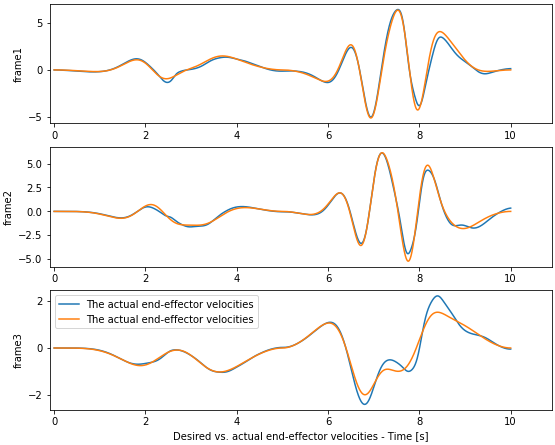
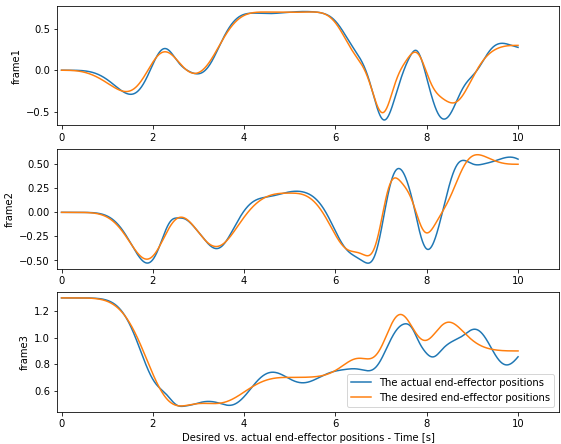
The way to write down function get\_point\_to\_point\_motion:

The core task of function robot\_controller:

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 . 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 :

Before start, we need to know . We can do this by function go\_point\_to\_point:

1. At each time step , compute the kinematic Jacobian matrix  using the current values of the joint angles .
2. Calculate the joint velocity vector  that must be achieved in the current time step using the equation
3. The joint angle displacement is then calculated as, here we set

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.

1. The next joint configuration is calculated by
2. The robot hardware is commanded to move the next joint configuration .

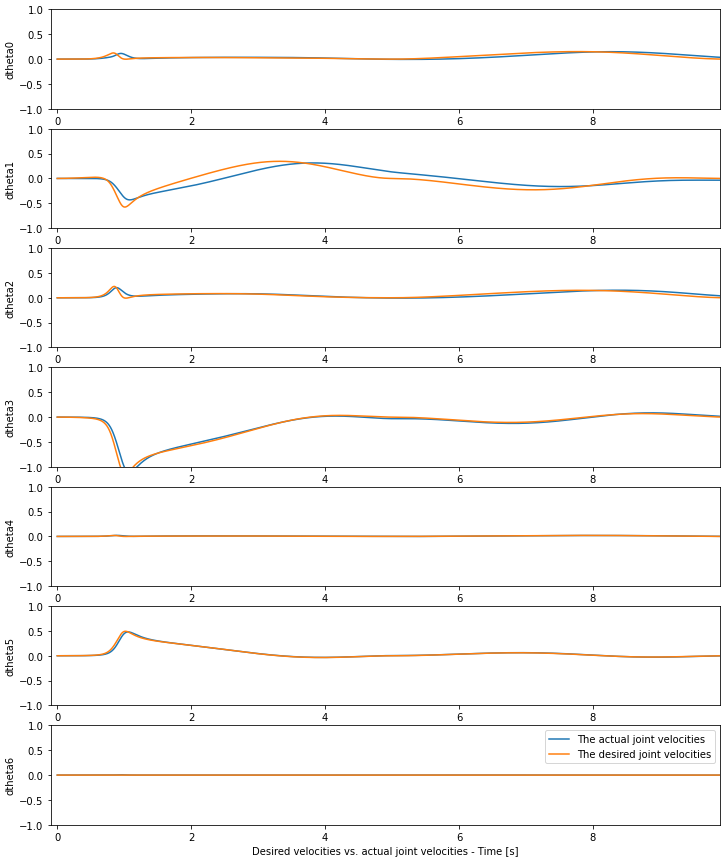
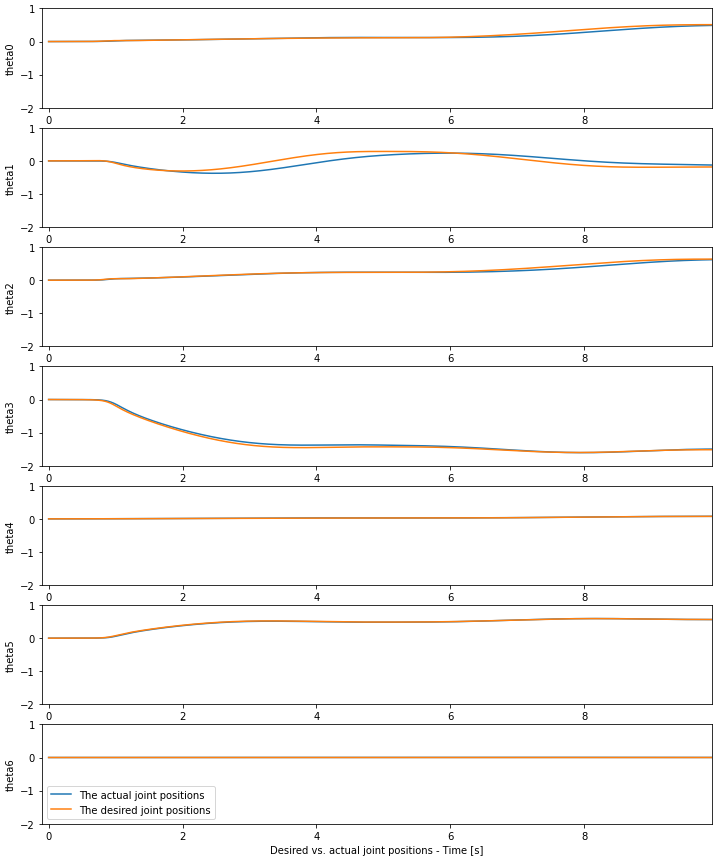
The steps above are repeated for as long as necessary.

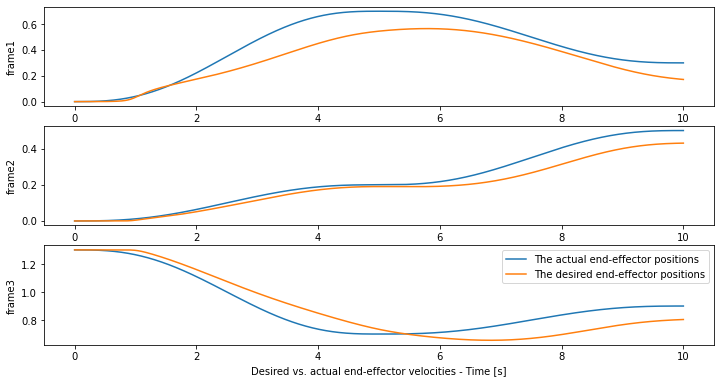
To add a nullspace:

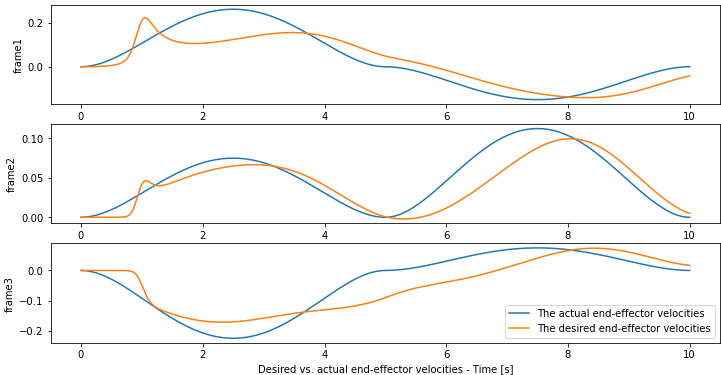
Modify here,

Set different and try to find the best one, here is an example:

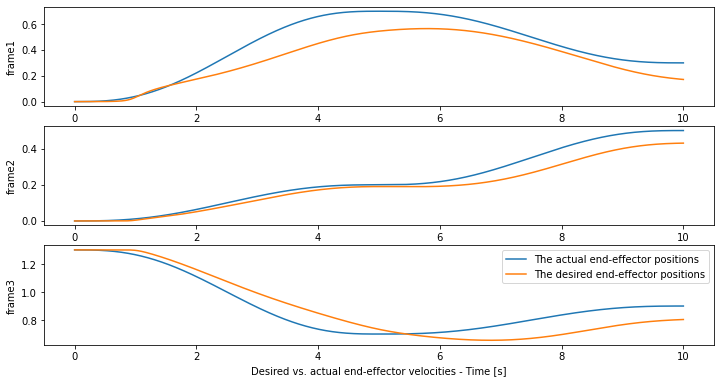
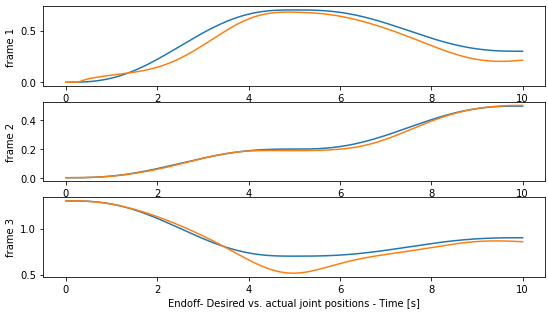
Plotting results:







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.