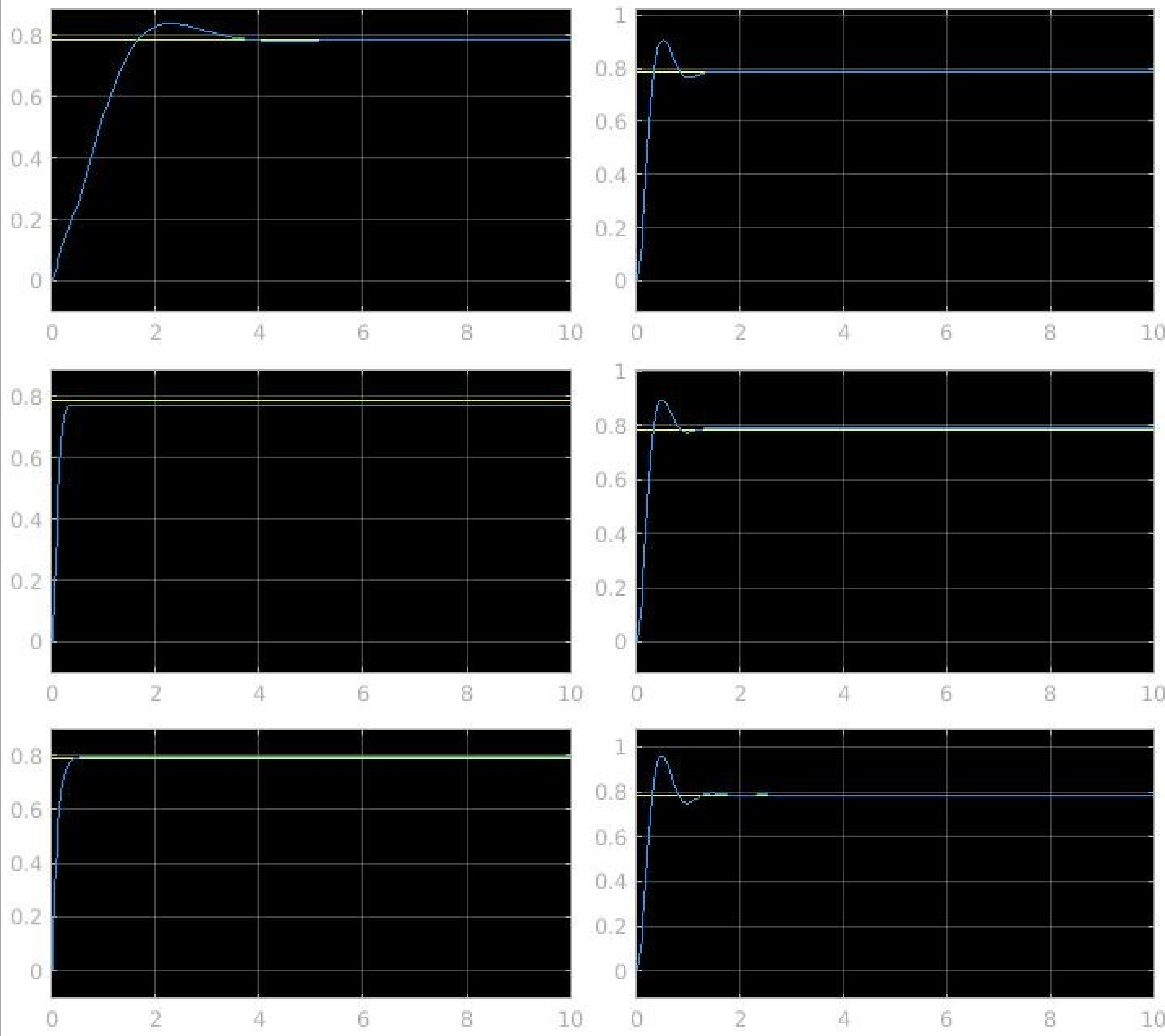
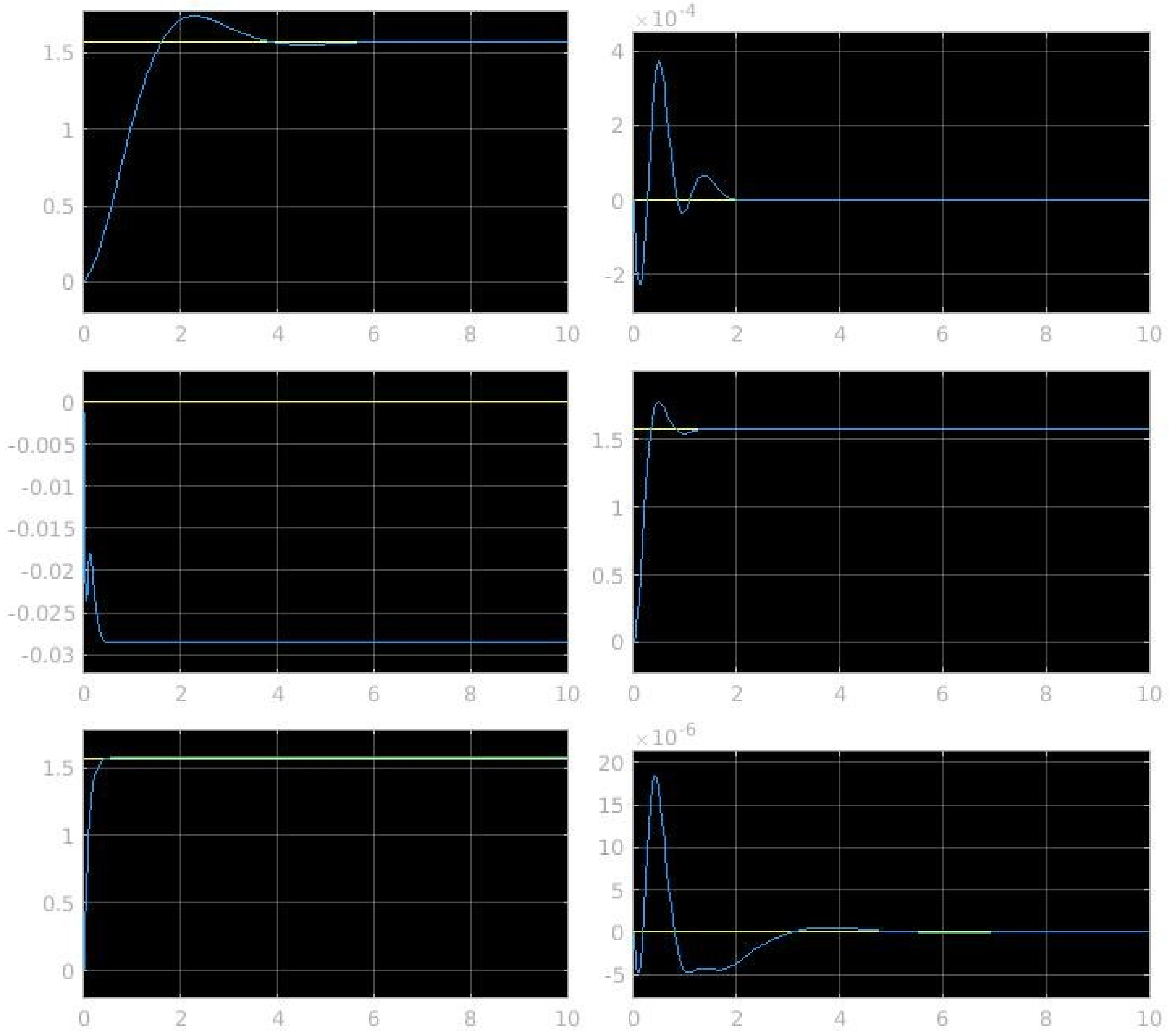
MEEn 537
HW #7
Jae Lee

1.

The plots show the tracking performance for each joint using the gain matrices below.
Desired joint angles = [pi/4 pi/4 pi/4 pi/4 pi/4 pi/4]



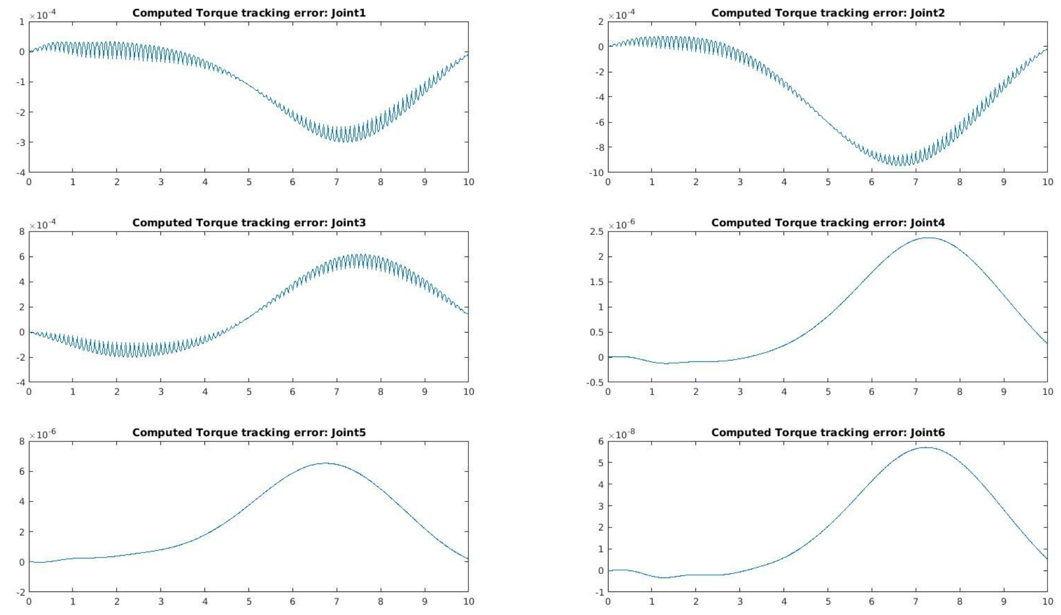
Desired joint angles = [pi/2 0 pi/2 0 pi/2 0]



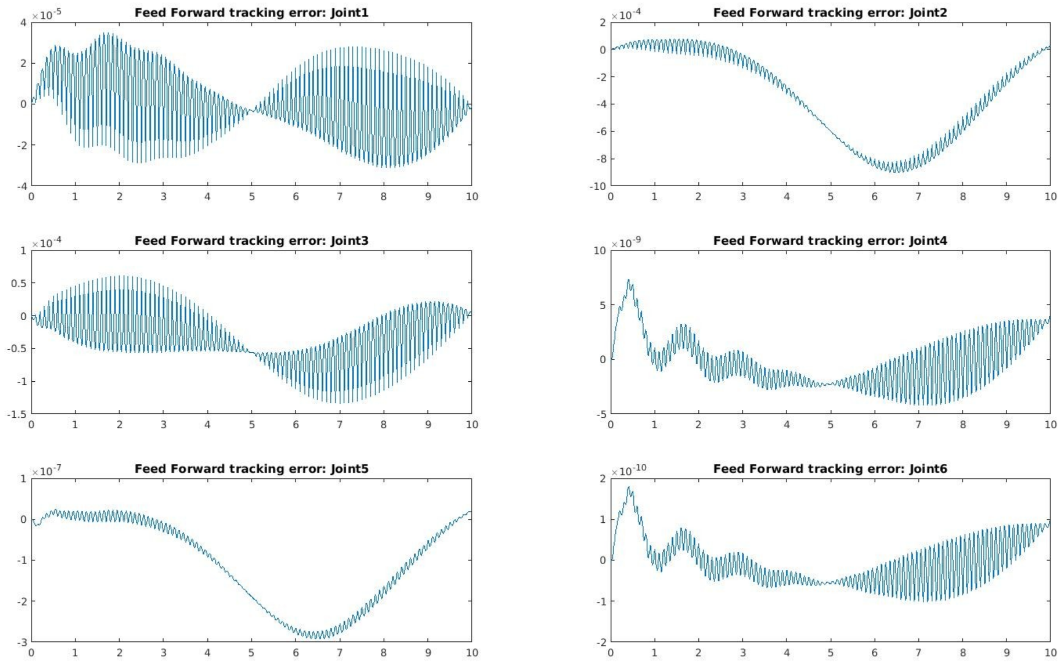
```
Kp = [10 0 0 0 0 0;...
      0 1000 0 0 0 0;...
      0 0 1000 0 0 0;...
      0 0 0 10 0 0;...
      0 0 0 0 10 0;...
      0 0 0 0 0 10];

Kd = [1 0 0 0 0 0;...
      0 100 0 0 0 0;...
      0 0 100 0 0 0;...
      0 0 0 1 0 0;...
      0 0 0 0 1 0;...
      0 0 0 0 0 1];
```

2. (a)

Desired joint angles = [pi/4 pi/2 -pi/2 0 0 0]
Computed torque tracking error for each joint



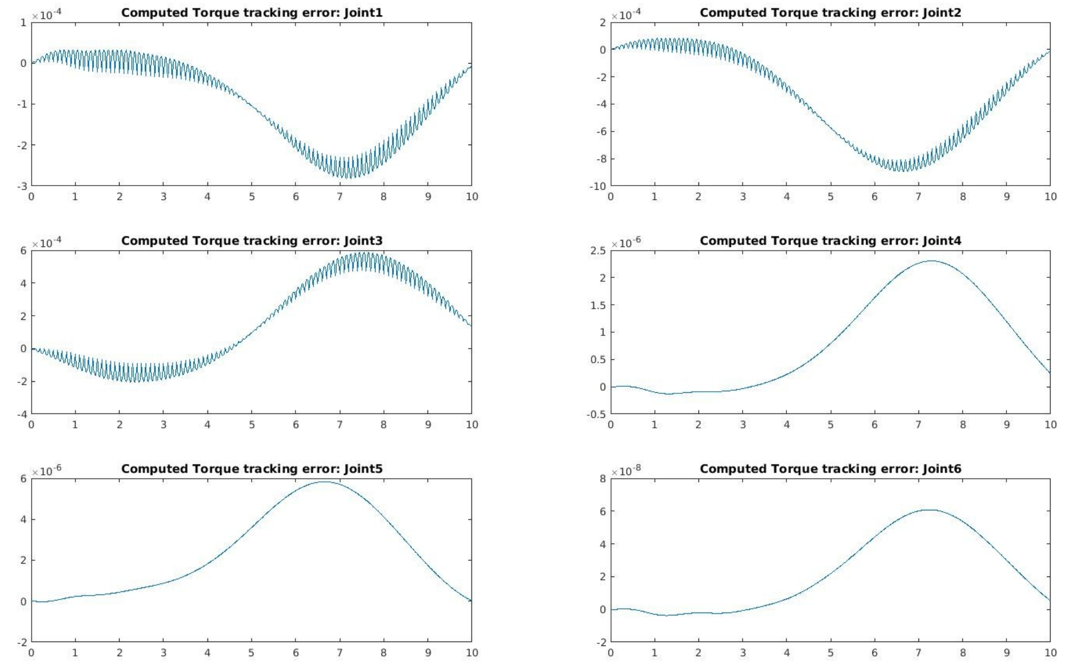
Feed forward tracking error for each joint



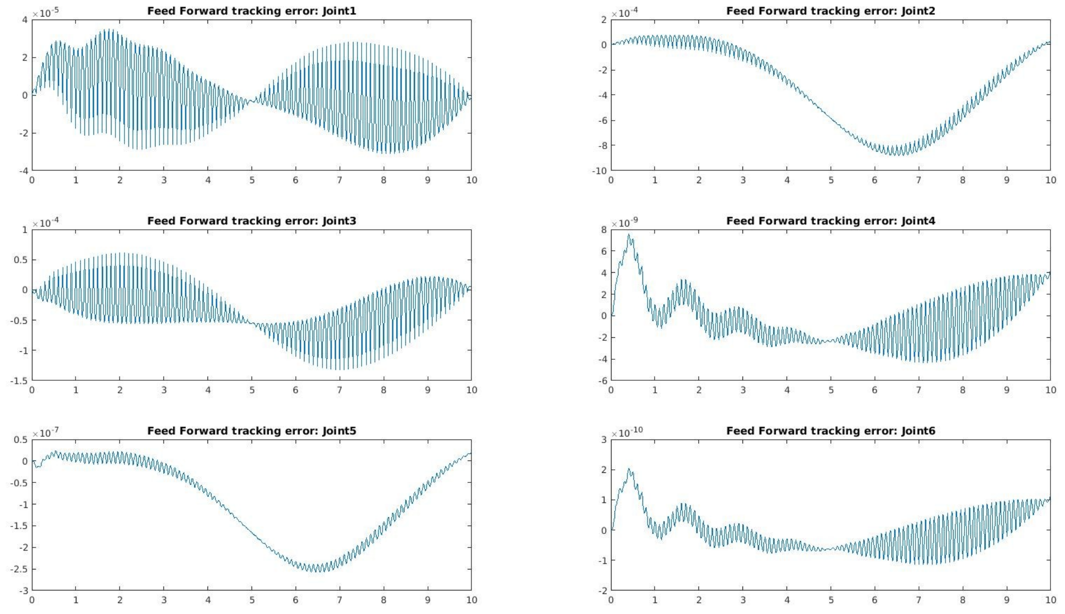
Both Feed forward and Computed torque methods worked quite well in zeroing the tracking errors as long as there is no perturbation. The scales of the errors are so small that it almost means nothing.

2. (b)

Desired joint angles = [pi/4 pi/2 -pi/2 0 0 0]
Computed torque tracking error for each joint with perturbation



Feed forward tracking error for each joint with perturbation



I expected some difference between no perturbation and perturbation for either computed torque or feed forward method, but I don’t really see any differences.

3.

(a)

```
T1 =
    0.9801    0.0000    0.1987    0.0000
   -0.0198    0.9950    0.0978    0.0000
   -0.1977   -0.0998    0.9752    0.5000
         0         0         0    1.0000
```

(b)

```
T2 =
    1.0000    0.0000    0.0000   -0.0000
   -0.0000    1.0000    0.0000    0.1000
   -0.0000   -0.0000    1.0000    0.0000
         0         0         0    1.0000
```

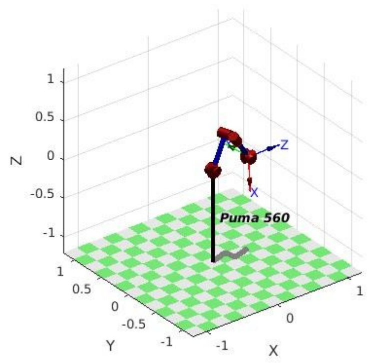
(c)

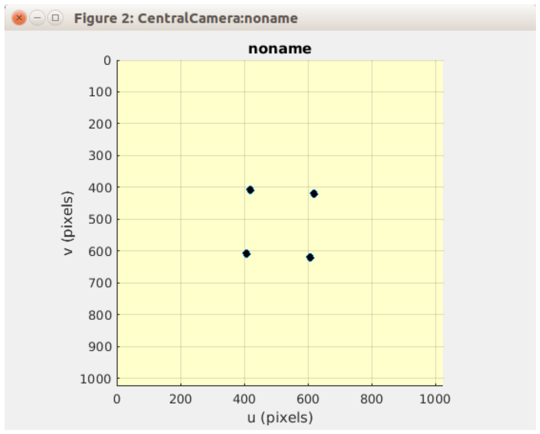
```
T3 =
    0.9801   -0.1987    0.0000    0.1000
    0.1977    0.9752    0.0998   -0.0000
   -0.0198   -0.0978    0.9950   -0.0000
         0         0         0    1.0000
```

4.

I picked the transformation from camera desired position to object position as specified below.

```
T_cdes_to_p = [0 0 -1 0;...
               0 1 0 0;...
               1 0 0 -0.5;...
               0 0 0 1];
```

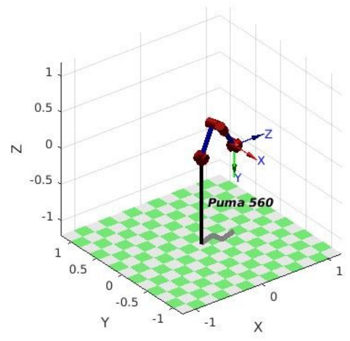


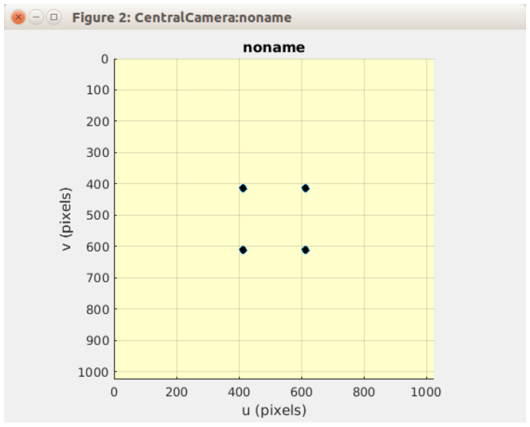


However, my simulation was not perfect because
    T_c_to_cdes = lambda*(u*inv(T_cdes_to_p)) kept ending up in error due to SVD containing NaN or Inf. (I spent about 2 hours on this problem)

5.

It was a lot easier to use IBVS especially when the object of interest started in the camera FOV. The most difficult thing for PBVS was to come up with a good enough transformation between camera desired pose and object pose. IBVS results in a lot more smooth path and better resulting image.





6.

Since I couldn’t get problem 4 working perfectly even if I spent about 2 hours, I couldn’t finish this problem.