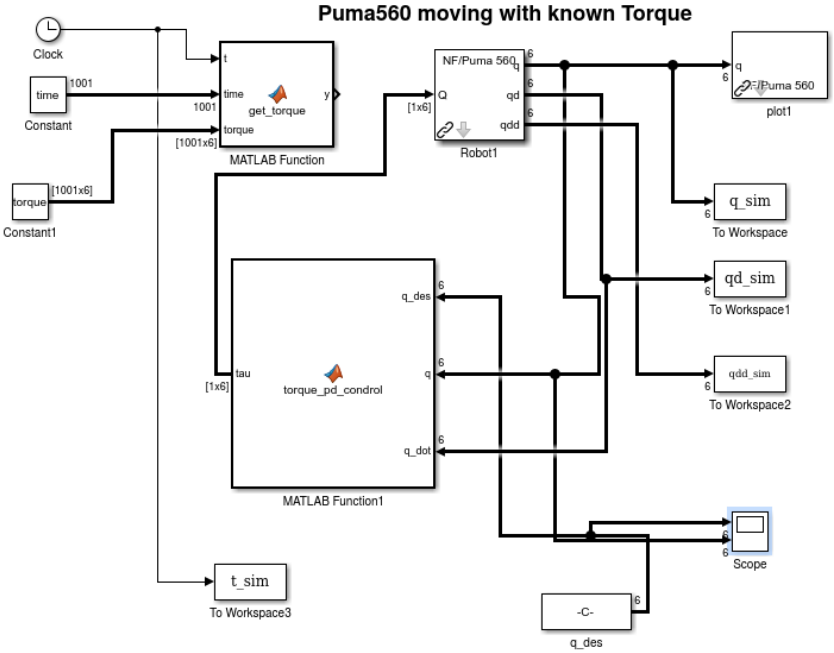
Jesse Wynn
MeEn 537 Robotics
Homework 7
December 13, 2017

Problem 1

PD control code:

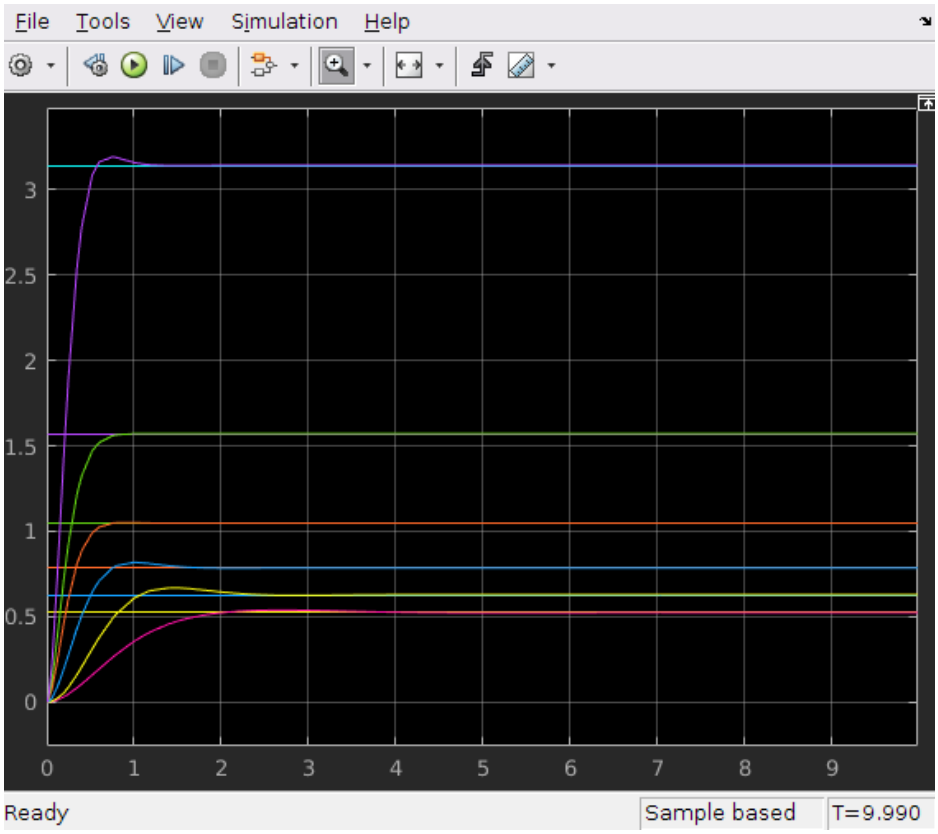
```
function tau = torque_pd_condrol(q_des, q, q_dot)

% q_desired_1 = [pi, pi/2, pi/3, pi/4, pi/5, pi/6];
% q_desired_2 = [pi/6, pi/5, pi/4, pi/3, pi/2, pi];

% kp = 10;
% kd = 2;

kp = [10, 30, 20, 10, 10, 10];
kd = [2, 5, 2, 2, 2, 2];

tau1 = kp(1)*(q_des(1) - q(1)) - kd(1)*(q_dot(1));
tau2 = kp(2)*(q_des(2) - q(2)) - kd(2)*(q_dot(2));
tau3 = kp(3)*(q_des(3) - q(3)) - kd(3)*(q_dot(3));
tau4 = kp(4)*(q_des(4) - q(4)) - kd(4)*(q_dot(4));
tau5 = kp(5)*(q_des(5) - q(5)) - kd(5)*(q_dot(5));
tau6 = kp(6)*(q_des(6) - q(6)) - kd(6)*(q_dot(6));

tau = [tau1, tau2, tau3, tau4, tau5, tau6];
end
```

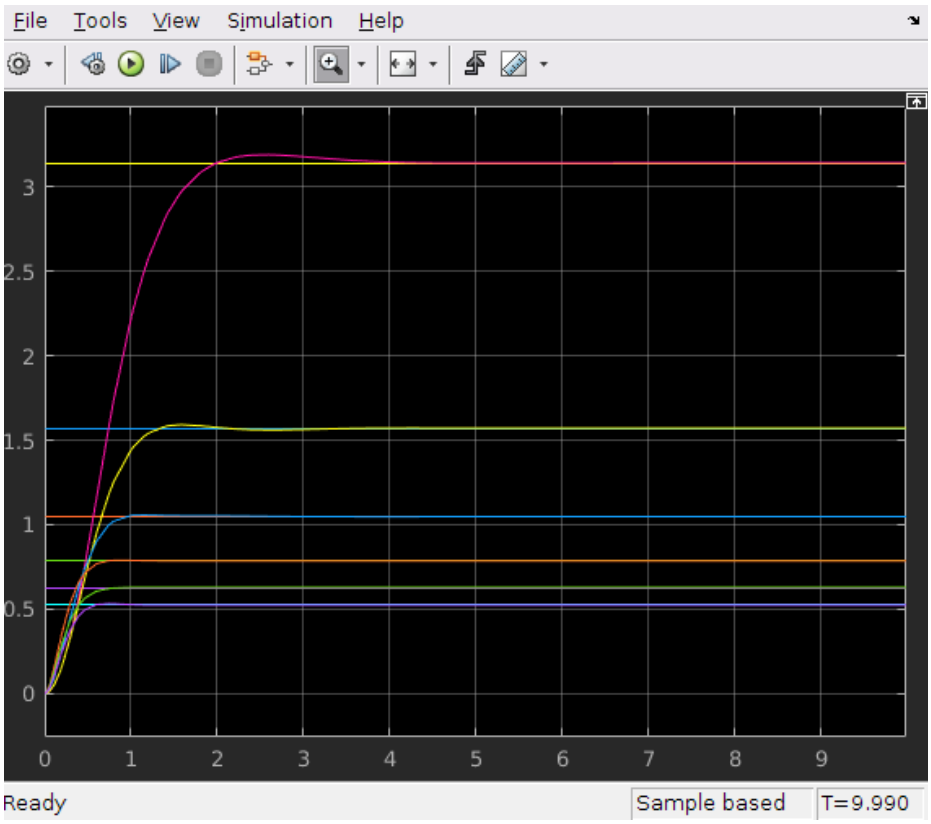
Simulink Block Diagram:



Problem 1 cont.

Plots q-desired and q-actual vs time:



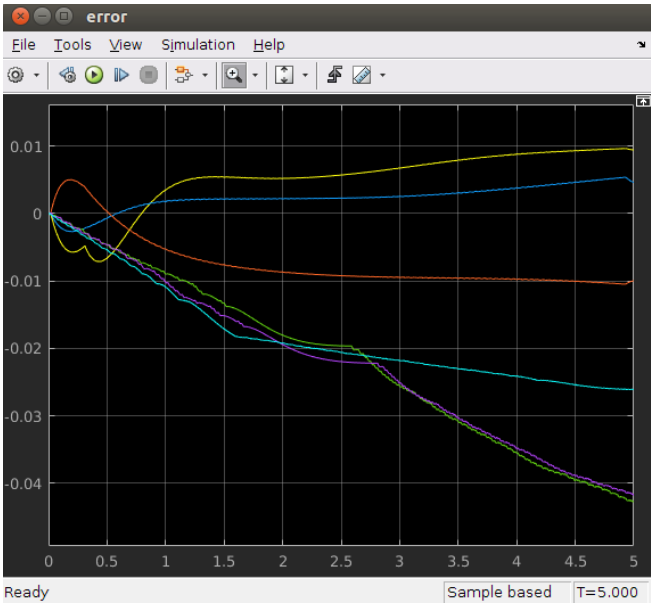


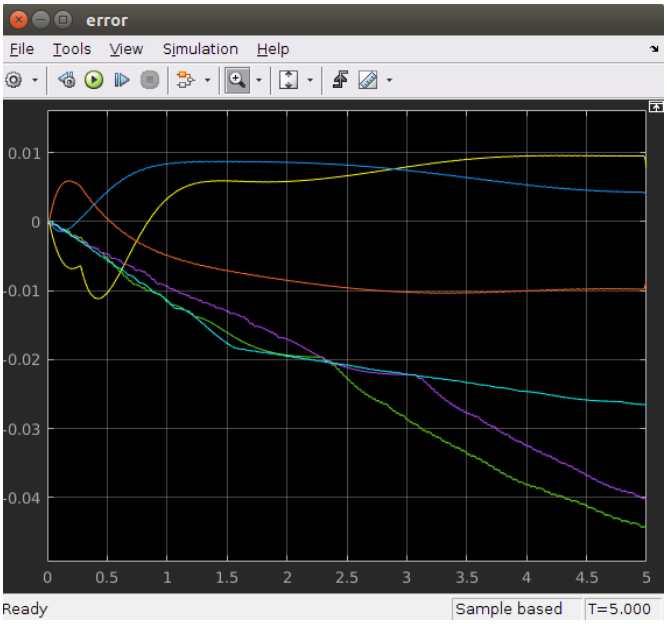
Problem 2

(a) Done.

(b) See plots below

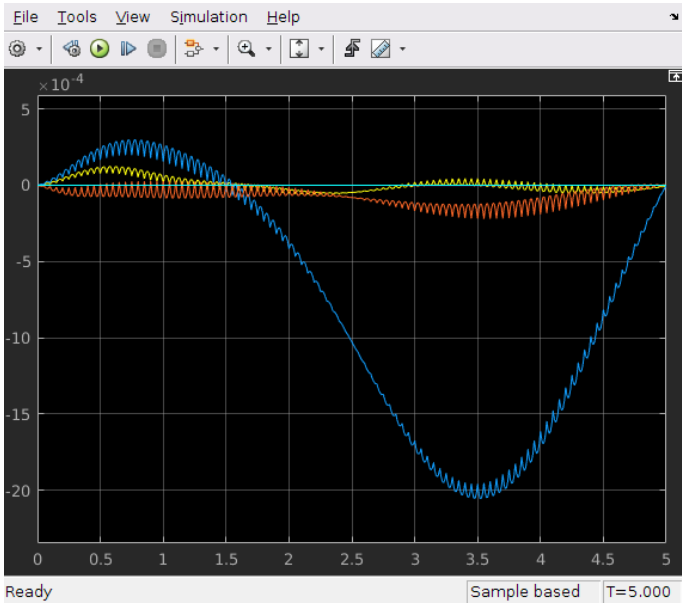
sl ctorque plots (correct params and then perturbed params):

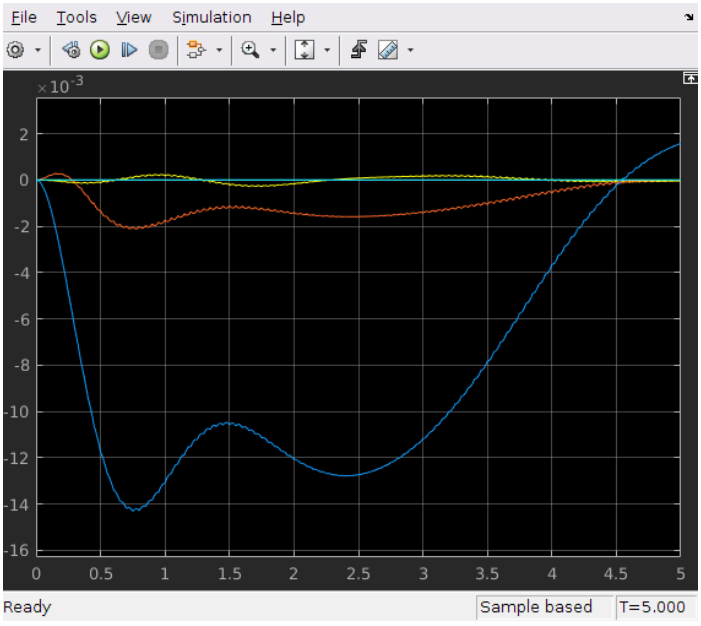




For the sl_ctorque controller simulation, the results were pretty similar using the real robot parameters and the perturbed parameters. This tells us that this particular controller does a pretty good job.

sl fforward plots (correct params and then perturbed params):





In the case of the feed-forward controller, the error with the perturbed parameters was an order of magnitude larger than it was with perfect parameters but the overall performance is still very good. In both cases the error is very small.

Problem 3

Toolbox code:

```
% hw_7 problem 3

% load camera
cam = CentralCamera('default');

% load the data
load('hw7_prob3.mat');

% estrimate the transformation between the world points and the camera
T_est_1 = cam.estpose(P1, p1)
T_est_2 = cam.estpose(P2, p2)
T_est_3 = cam.estpose(P3, p3)
```

Output:

```
T_est_1 =

    0.9801         0    0.1987         0
   -0.0198    0.9950    0.0978         0
   -0.1977   -0.0998    0.9752       0.5
         0         0         0         1


T_est_2 =

         1         0         0         0
         0    1.0000         0       0.1
         0         0         1         0
         0         0         0         1


T_est_3 =

    0.9801   -0.1987         0       0.1
    0.1977    0.9752    0.0998         0
   -0.0198   -0.0978    0.9950         0
         0         0         0         1
```

Problem 4

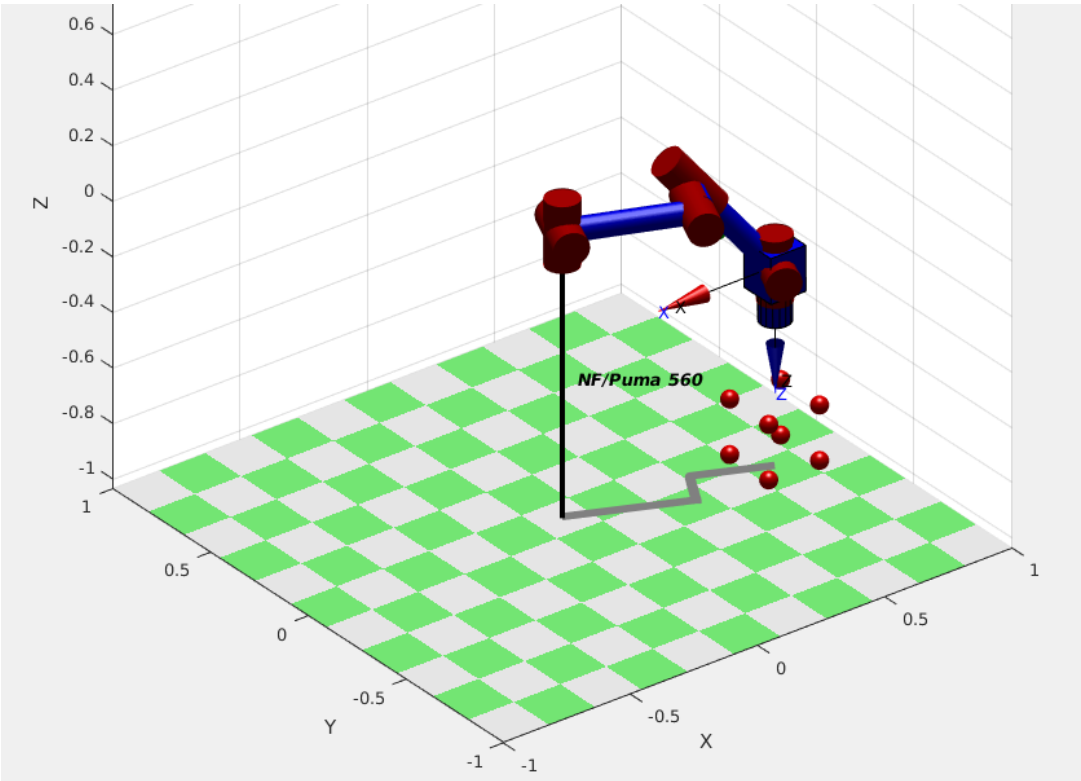
Matlab Code:

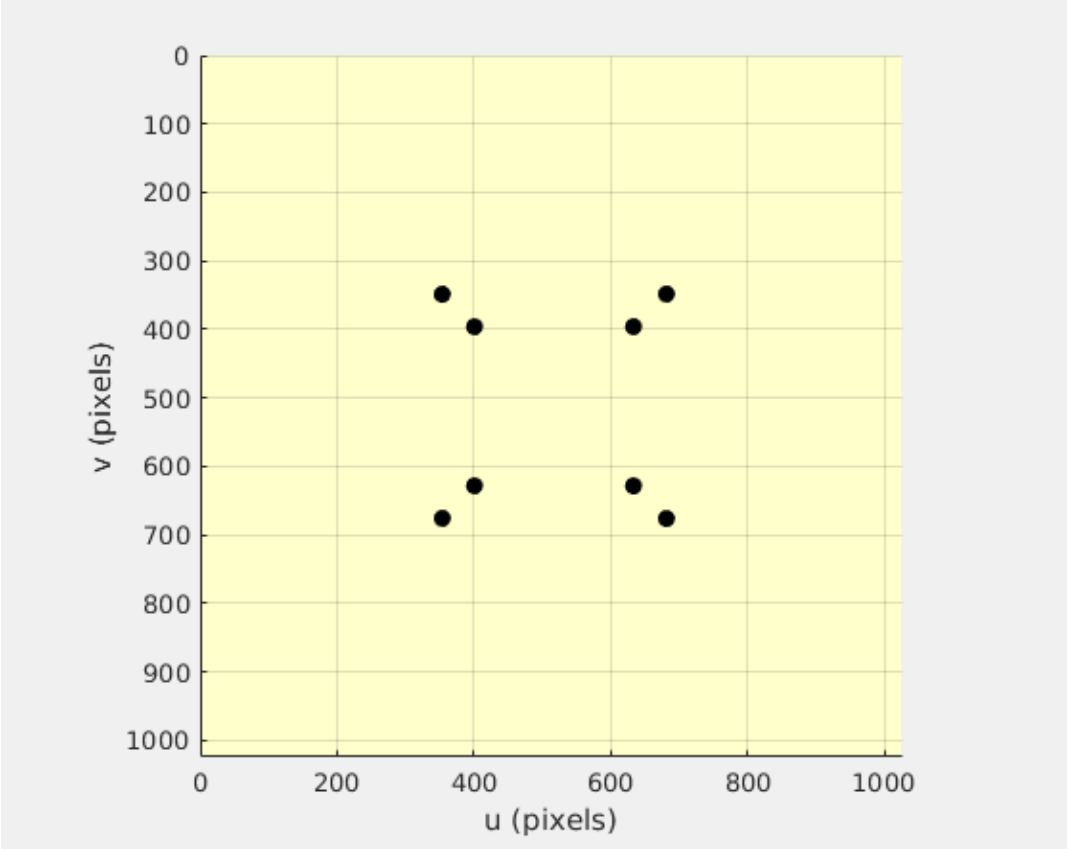
```
clc
clear
close all

% load puma robot
mdl_puma560

p560 = p560.nofriction();

% load camera
cam = CentralCamera('default');
% cam = CentralCamera('focal', 0.015, 'pixel', 10e-6, ...
% 'resolution', [1280 1024], 'centre', [640 512], 'name', 'mycamera');

P = mkcube(0.2, 'pose', transl([0.72, -0.15, -0.9]) );    % make a cube of points

% initial joint angles
q0 = [degtorad(0), degtorad(70), degtorad(-160), 0, degtorad(-70), 0];

% final joint angles
qfinal = [degtorad(0), degtorad(0), degtorad(-135), 0, degtorad(-45), 0];

figure(1), clf
p560.plot(q0);

% get end effector poses
T_init = p560.fkine(q0).T;
T_final = p560.fkine(qfinal).T;

% stick the camera at the end effector
cam.T = T_init;%*trotz(-pi/2);

cam.plot_camera;

% plot the points in the image plane
cam.plot(P)
pause;

% see how the points look in the final configuration
p560.plot(qfinal);
cam.T = T_final;%*trotz(-pi/2);
cam.plot_camera;
cam.plot(P)

% looks good! now let's do this

p = cam.project(P);
T_est = cam.estpose(P, p).T;
T_est = inv(T_est); % invert to put it in the right frame

% reset the robot and camera to q0
T = p560.fkine(q0).T;
plot_sphere(P, 0.03, 'r');  % display the cube
```

```
p560.plot(q0);
cam.T = T;

% lots of error to begin with
error = norm(tr2delta(T_init, T_final));

% implement PBVS in a loop
while error > 0.01
    pause(0.2);

    % get pixel projections
    p = cam.project(P);

    % get estimated transformation
    T_est = cam.estpose(P, p).T;
    T_est = inv(T_est); % invert to put it in the right frame

    % get transformation to step towards the goal based on the estimate
    T_step = trinterp(T_est, T_final, 0.1);

    % get the appropriate joint angles to step to using IK
    q = p560.ikine6s(T_step);

    % Update the robot and cameras positions
    T = p560.fkine(q).T;
    cam.T = T;

    % update the error
    error = norm(tr2delta(T, T_final))

    % update the plots
    p560.plot(q);
    cam.plot_camera;
    cam.plot(P)
end

disp("done!")
```

The PBVS method seemed to work pretty well (above I'm not simulating dynamics, just moving joint angles) as long as the the estimated transformation was close to correct. I noticed that using a cube (more points) instead of a square(fewer points) resulted in a better estimate. This makes sense.

Problem 4 Cont.





Problem 5 & 6

I didn't complete these, but I do understand what I would have had to do to complete them. I fought with the toolbox and Simulink models for a while but seemed to run into the same snags as others did.