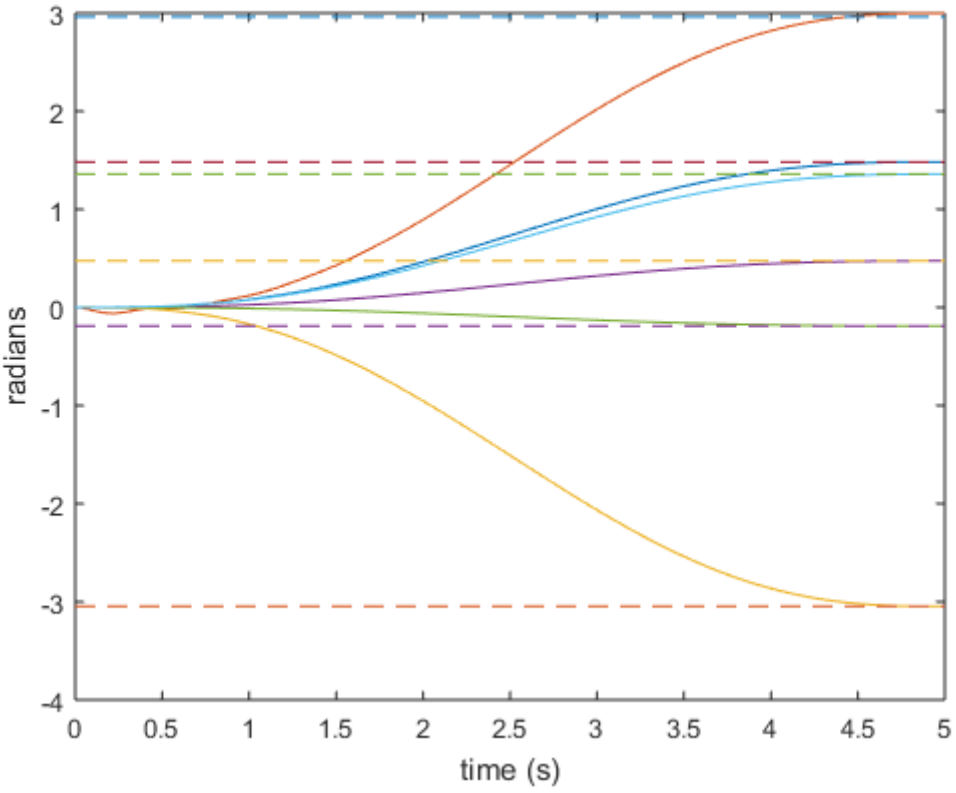
MeEn 537 Homework #7 Solution

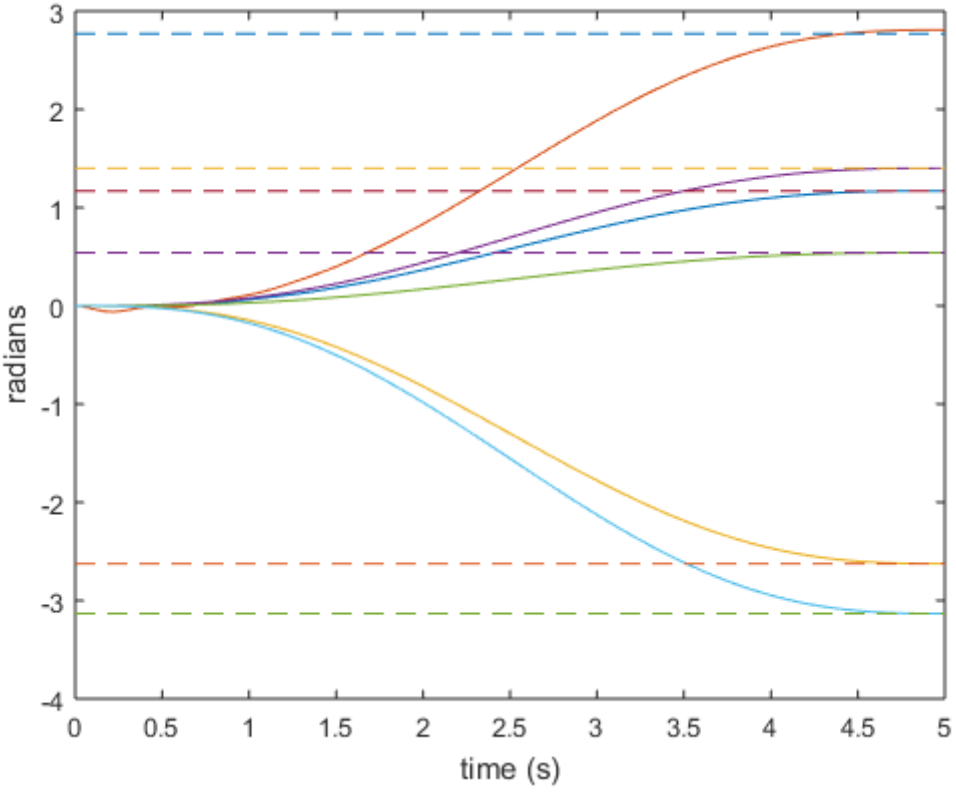
Problem 1:

K_p and K_d were diagonal matrices with values of 1000 and 10 respectively along the diagonal. See code and plots below.

```
clear all;
close all;
clc;

%define the robot and PD gains
mdl_puma560;
p560 = p560.nofriction();
Kp = 1000*eye(p560.n);
Kd = 10*eye(p560.n);

%make a first set of simulated joint angles that result from PD
 control

%randomly generate the command from within the robot joint limits
q_cmd = random('unif', p560.qlim(:,1), p560.qlim(:,2))

%simulate the control
out = sim('sl_puma_PD_control');

%plot the steady state desired position and the actual joint angles
 over time
figure()
q_cmd_vec = [q_cmd, q_cmd]
t_cmd = [0, 5];
q = out.get('q_sim');
t = out.get('t_sim');
plot(t, q, t_cmd, q_cmd_vec, '--');
xlabel('time (s)')
ylabel('radians')


%repeat for another set of joint angles
q_cmd = random('unif', p560.qlim(:,1), p560.qlim(:,2))
out = sim('sl_puma_PD_control');

figure()
q_cmd_vec = [q_cmd, q_cmd]
t_cmd = [0, 5];
q = out.get('q_sim');
t = out.get('t_sim');
plot(t, q, t_cmd, q_cmd_vec, '--');
xlabel('time (s)')
ylabel('radians')


q_cmd =

    1.4829
    2.9619
   -3.0464
    0.4736
   -0.1899
    1.3585
```

```
Warning: Output port 3 of 'sl_puma_PD_control/Robot' is not
 connected.
Warning: Output port 3 of 'sl_puma_PD_control/jtraj' is not
 connected.
Fast RNE: (c) Peter Corke 2002-2012

q_cmd_vec =

    1.4829    1.4829
    2.9619    2.9619
   -3.0464   -3.0464
    0.4736    0.4736
   -0.1899   -0.1899
    1.3585    1.3585


q_cmd =

    1.1693
    2.7710
   -2.6263
    1.4018
    0.5414
   -3.1327

Warning: Output port 3 of 'sl_puma_PD_control/Robot' is not
 connected.
Warning: Output port 3 of 'sl_puma_PD_control/jtraj' is not
 connected.

q_cmd_vec =

    1.1693    1.1693
    2.7710    2.7710
   -2.6263   -2.6263
    1.4018    1.4018
    0.5414    0.5414
   -3.1327   -3.1327
```


radians
time (s)
3
2
1
0
-1
-2
-3
-4
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5


radians
time (s)
3
2
1
0
-1
-2
-3
-4
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

Published with MATLAB® R2015a

Problem 2:

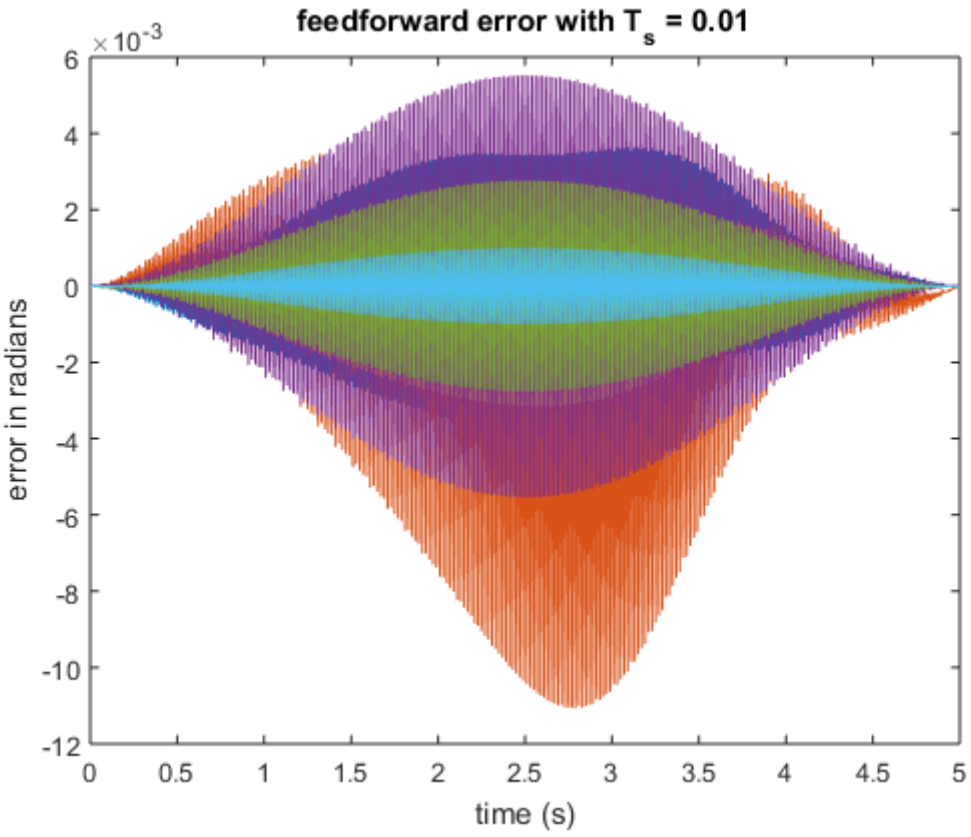
See code and first four plots below for varying the control/sampling time for part a. The next two plots show what happens when we model the robot dynamic model as being off by as much as 15% of the nominal amount.

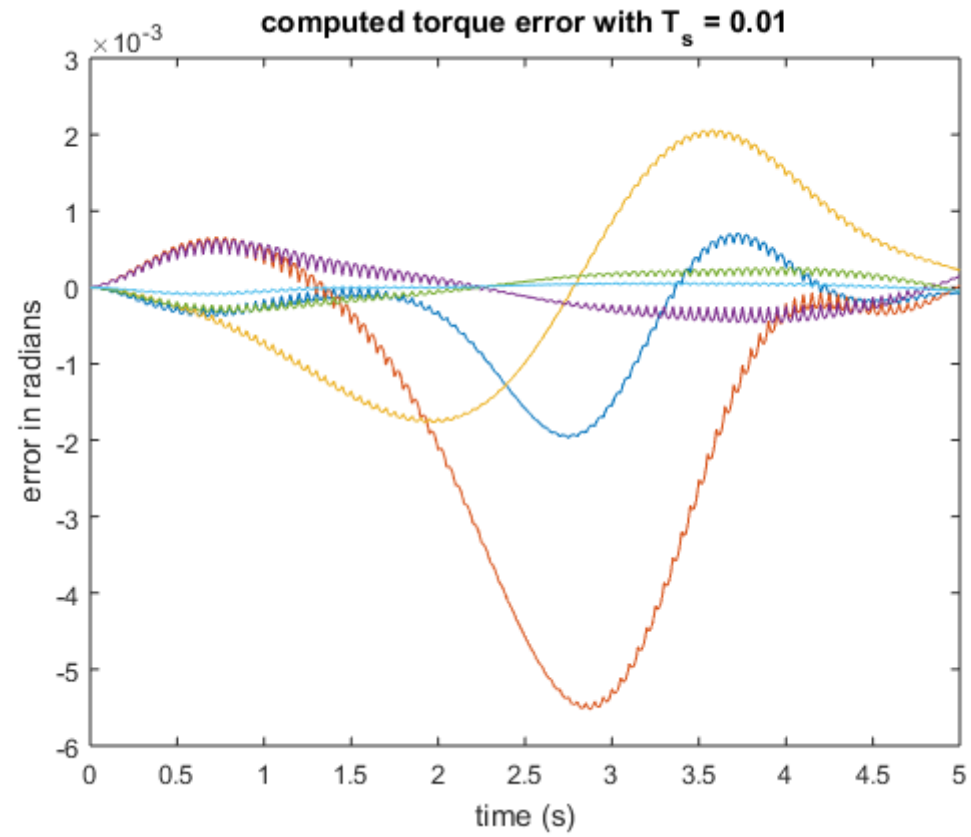
```
run D:\Dropbox\vision-3.4\rvctools\startup_rvc.m
run D:\Dropbox\rvctools_new\startup_rvc.m

clear all;
close all;
clc;

%define the robot and PD gains
mdl_puma560;
p560 = p560.nofriction();

%randomly generate the command from within the robot joint limits
q_cmd = random('unif', p560.qlim(:,1), p560.qlim(:,2))
Ts = 0.01;

%simulate the control
out = sim('sl_ctorque');
out2 = sim('sl_fforward');

ctorque_error = out.get('q_error');
ctorque_time = out.get('time');

ff_error = q_error;
ff_t = time;

figure()
plot(ff_t, ff_error);
title('feedforward error with T_s = 0.01')
xlabel('time (s)');
ylabel('error in radians');

figure()
plot(ctorque_time, ctorque_error);
title('computed torque error with T_s = 0.01')
xlabel('time (s)');
ylabel('error in radians');

Ts = 0.001;

%simulate the control
out = sim('sl_ctorque');
out2 = sim('sl_fforward');

ctorque_error = out.get('q_error');
ctorque_time = out.get('time');

ff_error = q_error;
ff_t = time;

figure()
plot(ff_t, ff_error);
title('feedforward error with T_s = 0.001')
```

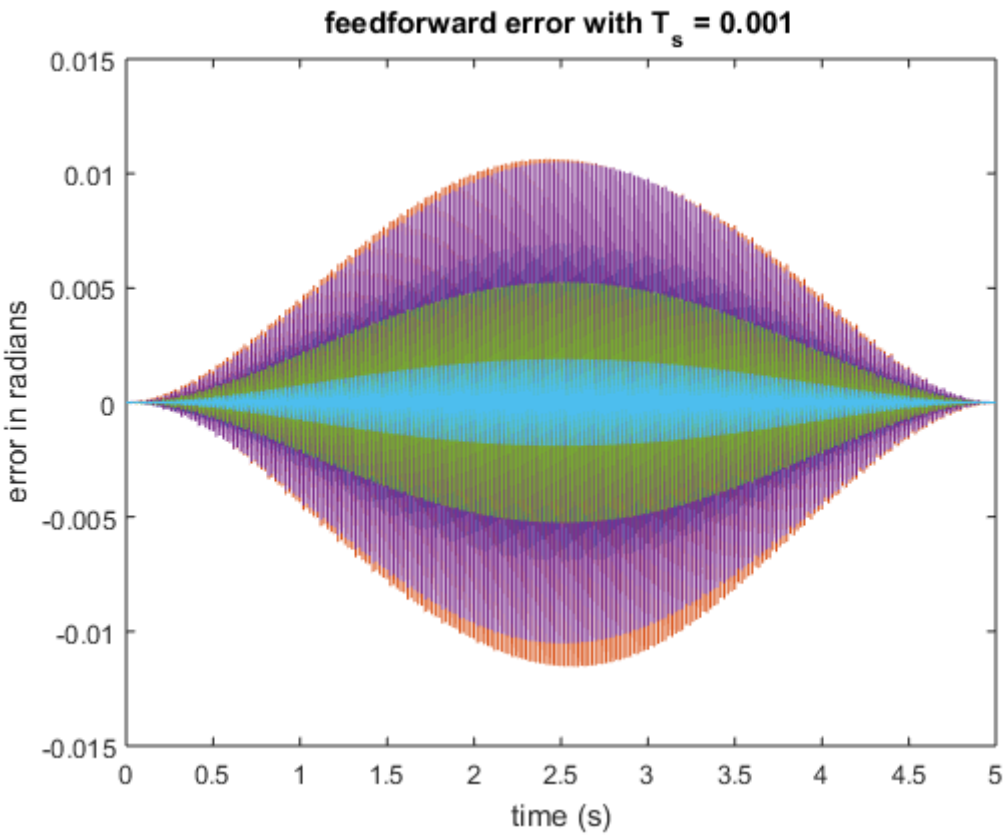
```
xlabel('time (s)');
ylabel('error in radians');

figure()
plot(ctorque_time, ctorque_error);
title('computed torque error with T_s = 0.001')
xlabel('time (s)');
ylabel('error in radians');



%simulate the control for the perturbed models
p560_model=p560.perturb(0.15);
out = sim('sl_ctorque_modified');
out2 = sim('sl_fforward_modified');

ctorque_error = out.get('q_error');
ctorque_time = out.get('time');

ff_error = q_error;
ff_t = time;

figure()
plot(ff_t, ff_error);
title('feedforward error with perturbed model')
xlabel('time (s)');
ylabel('error in radians');

figure()
plot(ctorque_time, ctorque_error);
title('computed torque error with perturbed model')
xlabel('time (s)');
ylabel('error in radians');
```

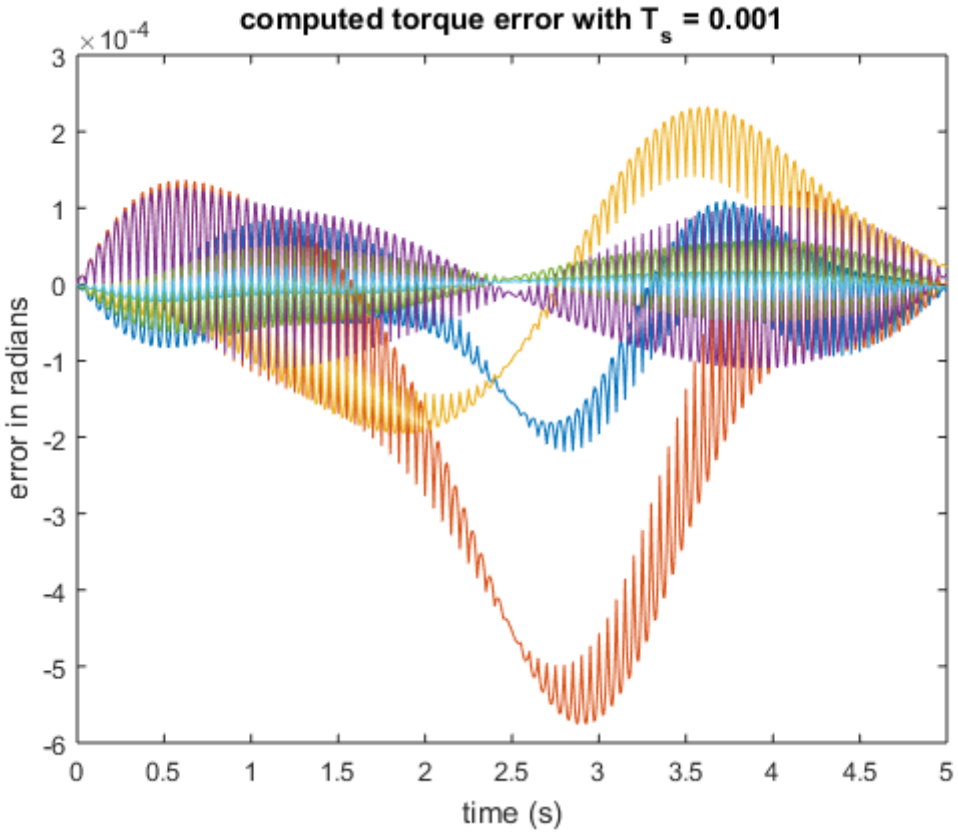
```
Robotics, Vision & Control: (c) Peter Corke 1992-2011 http://
www.petercorke.com
- Machine Vision Toolbox for Matlab (release 3.4)
Robotics, Vision & Control: (c) Peter Corke 1992-2011 http://
www.petercorke.com
- Robotics Toolbox for Matlab (release 9.10)
 - pHRIWARE (release 1.1): pHRIWARE is Copyrighted by Bryan Moutrie
 (2013-2016) (c)
Run rtbdemo to explore the toolbox

q_cmd =

   -1.9415
    3.1062
   -1.3901
    2.9482
   -1.4724
   -0.5322
```

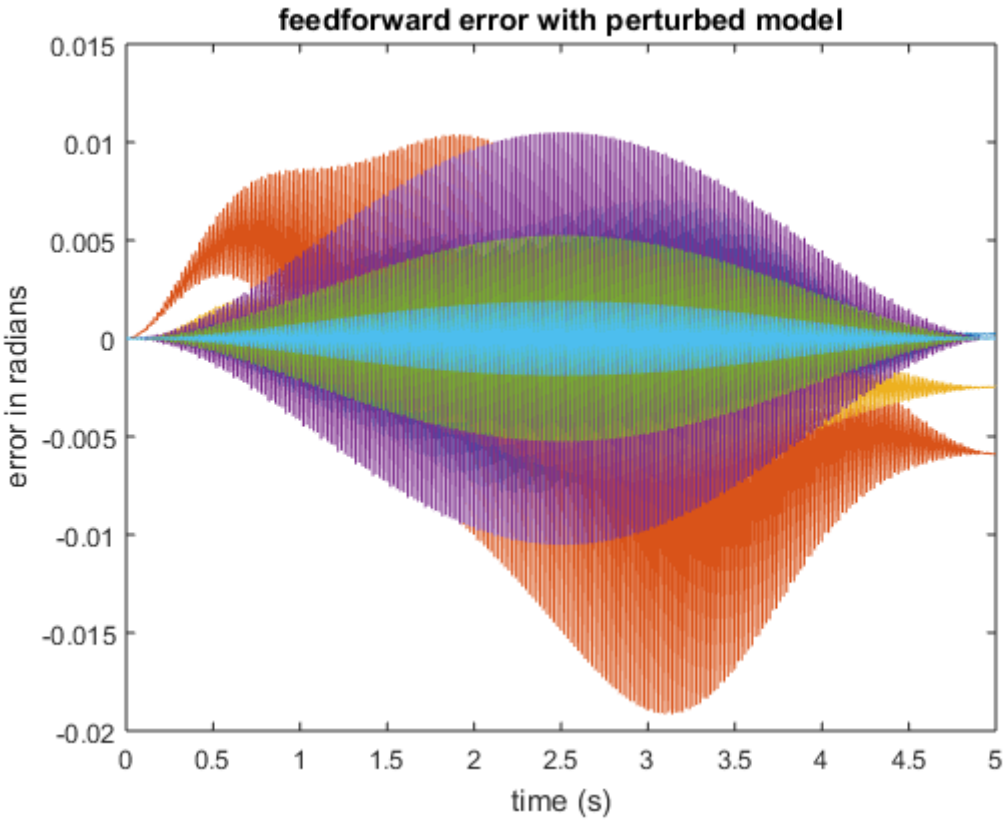
```
n =

     6

Fast RNE: (c) Peter Corke 2002-2012

n =

     6


n =

     6


n =

     6


n =

     6


n =

     6
```

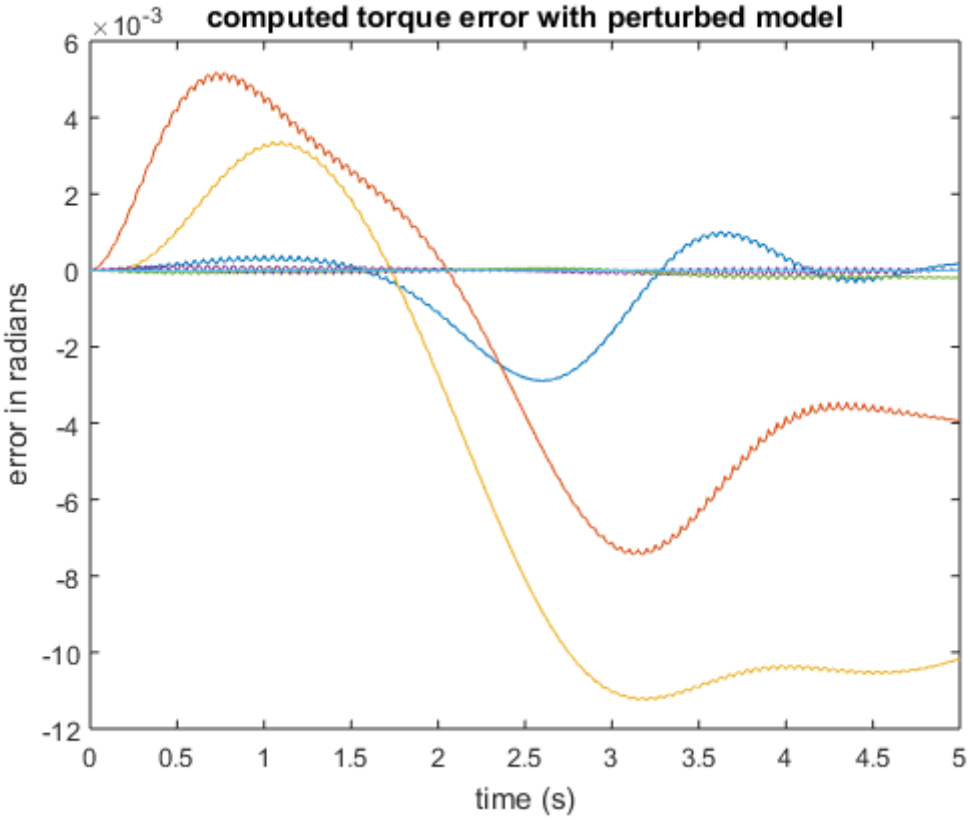

feedforward error with T_s = 0.01
$\times 10^{-3}$
6
4
2
0
-2
-4
-6
-8
-10
-12
error in radians
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
time (s)


computed torque error with T_s = 0.01
$\times 10^{-3}$
3
2
1
0
-1
-2
-3
-4
-5
-6
error in radians
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
time (s)


feedforward error with T_s = 0.001
error in radians
time (s)
0.015
0.01
0.005
0
-0.005
-0.01
-0.015
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5


computed torque error with T_s = 0.001
×10^-4
error in radians
time (s)
3
2
1
0
-1
-2
-3
-4
-5
-6
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5


feedforward error with perturbed model
error in radians
time (s)
0.015
0.01
0.005
0
-0.005
-0.01
-0.015
-0.02
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5


computed torque error with perturbed model
×10^-3
error in radians
time (s)
6
4
2
0
-2
-4
-6
-8
-10
-12
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

Published with MATLAB® R2015a

Problem 3:

```
%HW 7 problem 3
run D:\Dropbox\vision-3.4\rvctools\startup_rvc.m
run D:\Dropbox\rvctools_new\startup_rvc.m

close all;
clear all;
clc;

load ./hw7_prob3.mat

cam = CentralCamera('default');

T_est1 = cam.estpose(P1, p1)
T_est2 = cam.estpose(P2, p2)
T_est3 = cam.estpose(P3, p3)
```

```
Robotics, Vision & Control: (c) Peter Corke 1992-2011 http://
www.petercorke.com
- Machine Vision Toolbox for Matlab (release 3.4)
Robotics, Vision & Control: (c) Peter Corke 1992-2011 http://
www.petercorke.com
- Robotics Toolbox for Matlab (release 9.10)
 - pHRIWARE (release 1.1): pHRIWARE is Copyrighted by Bryan Moutrie
 (2013-2016) (c)
Run rtbdemo to explore the toolbox
delete camera object
Warning: The following error was caught while executing 'Camera' class
destructor:
Invalid or deleted object.
principal point not specified, setting it to centre of image plane

T_est1 =

    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


T_est2 =

    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


T_est3 =

    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
```

Published with MATLAB® R2015a

Problem 4:

See code below for PBVS implementation.

```
clear all;
close all;

%make a camera
cam = CentralCamera('default');

%instantiate robot model, initial position, and vectors for plotting
mdl_puma560;
q = [0, -pi/4, 0, 0, pi/4, 0];
rpy_pbvs_plot = [];
cam_pbvs_pos_plot = [];
q_pbvs_plot = [];
t_pbvs_plot = [];

%get the end effector position and set the camera at that position
T = p560.fkine(q);
cam.T = T;

%this transform defines where the object starts relative to my initial
 robo
%configuration
%this moves the object -0.2 meters in the x and 1 meter in z away from
 my
%camera/end effector position
move_obj_position = [-0.2; 0; 1;];
T_points = T*[rotz(pi/2), move_obj_position;
              0, 0, 0, 1];

%this is the definition of the nominal vertices for our object in
%homogenous coordinates
P_nom = [-0.1   0.1 -0.1   0.1;
         0.1    0.1 -0.1  -0.1;
         0      0    0      0;
         1      1    1      1];

%this is transforming the nominal object model out to its actual
 location
P = T_points*P_nom;
P = P(1:3,:);

%get the pixel coordinates of the object in the camera now
p = cam.project(P);
cam.plot(P)

%from that we can estimate where the object is relative to the camera
 at
%least
T_est_cam = cam.estpose(P_nom, p);

%then we can define a desired position for the camera relative to the
%object, which in this case is just 0.5 meters back in the z-direction
 and
```

```
%aligned with the object
T_des = T*T_est_cam*[eye(3), [0; 0; -0.5];
                     0 0 0 1];

%plot everything for the initial conditions
figure()
p560.plot(q)
cam.plot_camera()
plot_sphere(P, 0.05, 'r')

%this is just a starting condition for the while loop
error = 1;

time = 0;
while error > 0.001
    %find how we can take small step towards our desired camera
 position
    T_cmd = trinterp(T, T_des, 0.1);
    time = time + 0.1; %assume it takes 0.1 seconds to get there

    %do inverse kinematics to find the new commanded joint angles. The
    %assumption is that we immediately are able to attain those
 angles.
    %Could also do this in terms of joint velocities if preferred.
    q = p560.ikine(T_cmd, q);

    %find how the end effector and camera have moved based on our
 robot
    %kinematic  model
    T = p560.fkine(q);
    cam.T = T;

    %update all plots
    %cam.plot(P)
    %p560.plot(q)
    %cam.plot_camera()
    %plot_sphere(P, 0.05, 'r')

    %project the object into the camera image plane now that we've
 moved
    p = cam.project(P);

    %solve for the relative pose of the object with respect to the
 camera
    %agian
    T_est_cam = cam.estpose(P_nom, p);

    %and define a desired camera pose relative to that position for
 the
    %camera
    T_des = T*T_est_cam*[eye(3), [0; 0; -0.5];
        0 0 0 1];
```

```
    %calculate our error in terms of roll, pitch, yaw and then
 position too
    rpy_des = tr2rpy(T_des);
    rpy = tr2rpy(T);
    error = norm(T(1:3,4)-T_des(1:3,4))+norm(rpy_des-rpy);

    rpy_pbvs_plot = [rpy_pbvs_plot; rpy];
    cam_pbvs_pos_plot = [cam_pbvs_pos_plot; T(1:3,4)'];
    q_pbvs_plot = [q_pbvs_plot; q];
    t_pbvs_plot = [t_pbvs_plot; time];

end
```

Published with MATLAB® R2015a

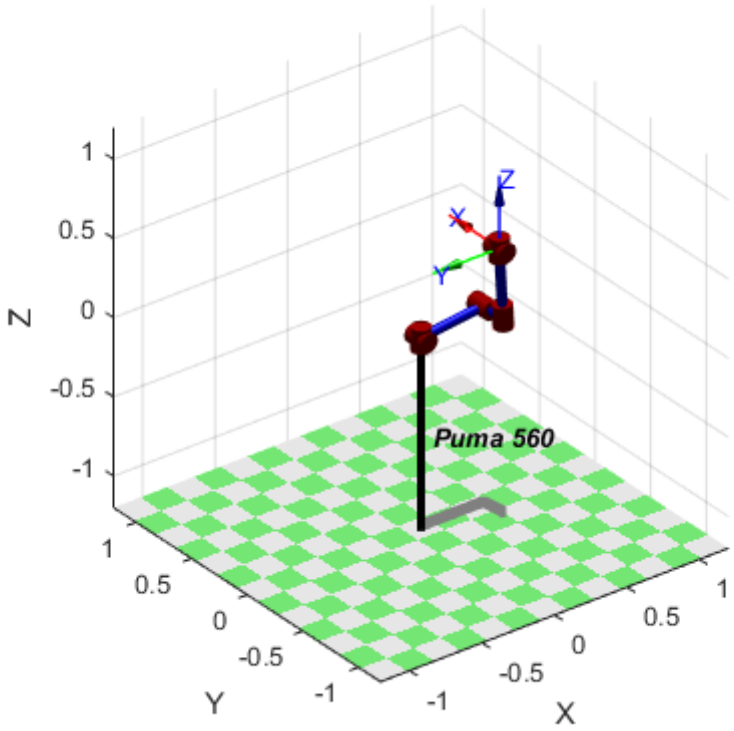
Problem 5:

See code below and modified 'sl_arm_ibvs_hw_7' file on learning suite to see the comparison between the two methods. The timing can't be compared super well since we aren't checking torque or velocity limits on actual robot arms. However, it is clear from the graphs that the PBVS method reduces Cartesian error more quickly/directly than the IBVS method.

```
clear all;
clc;

%these are just the time, position and roll/pitch/yaw vectors for the
 pbvs
%solution from problem 4
load results_hw7_prob4.mat;

%making a puma robot and default camera
mdl_puma560;
camera = CentralCamera('default');

%these are the same object vertices that we defined in problem 4
P = [0.3250    0.3250    0.5250    0.5250;
   -0.2500   -0.0501   -0.2500   -0.0501;
    0.9856    0.9856    0.9856    0.9856];

%these are the same initial joint angles
q0 = [0, -pi/4, 0, 0, pi/4, 0]';

%simulating the ibvs algorithm
out = sim('sl_arm_ibvs_hw_7');

%getting all the data out and reformatting it
Ts = out.get('T_ibvs');
q_ibvs = out.get('q_ibvs');
t_ibvs = out.get('t_ibvs');

q_ibvs = reshape(q_ibvs, [length(t_ibvs), p560.n]);

pos_ibvs = [];
rpy_ibvs = [];

for i = 1:1:length(t_ibvs)
    pos_ibvs = [pos_ibvs; reshape(Ts(1:3,4,i), [1,3])];
    rpy = tr2rpy(Ts(:,:,i));
    rpy_ibvs = [rpy_ibvs; rpy];
end


%comparing roll/pitch/yaw and position response for each method
figure()
plot(t_pbvs_plot, rpy_pbvs_plot(:,1), t_ibvs, rpy_ibvs(:,1), 'r--');
legend('pbvs', 'ibvs');
xlabel('time (s)');
ylabel('roll (radians)');
title('PBVS vs IBVS for roll');

figure()
plot(t_pbvs_plot, rpy_pbvs_plot(:,2), t_ibvs, rpy_ibvs(:,2), 'r--');
legend('pbvs', 'ibvs');
xlabel('time (s)');
```

```
ylabel('pitch (radians)');
title('PBVS vs IBVS for pitch');

figure()
plot(t_pbvs_plot, rpy_pbvs_plot(:,3), t_ibvs, rpy_ibvs(:,3), 'r--');
legend('pbvs', 'ibvs');
xlabel('time (s)');
ylabel('yaw (radians)');
title('PBVS vs IBVS for yaw');

figure()
plot(t_pbvs_plot, cam_pbvs_pos_plot(:,1), t_ibvs,
 pos_ibvs(:,1), 'r--');
legend('pbvs', 'ibvs');
xlabel('time (s)');
ylabel('x position (m)');
title('PBVS vs IBVS for x position');

figure()
plot(t_pbvs_plot, cam_pbvs_pos_plot(:,2), t_ibvs,
 pos_ibvs(:,2), 'r--');
legend('pbvs', 'ibvs');
xlabel('time (s)');
ylabel('y position (m)');
title('PBVS vs IBVS for y position');

figure()
plot(t_pbvs_plot, cam_pbvs_pos_plot(:,3), t_ibvs,
 pos_ibvs(:,3), 'r--');
legend('pbvs', 'ibvs');
xlabel('time (s)');
ylabel('z position (m)');
title('PBVS vs IBVS for z position');

%cam_ibvs_pos = T

principal point not specified, setting it to centre of image plane
delete camera object
Warning: The following error was caught while executing 'Camera'
class destructor:
Invalid or deleted object.
Warning: Output port 2 of 'sl_arm_ibvs_hw_7/invJac' is not
connected.
Warning: Output port 2 of 'sl_arm_ibvs_hw_7/invJac1' is not
connected.
creating new figure for camera

h =

  Axes with properties:

            XLim: [0 1]
            YLim: [0 1]
          XScale: 'linear'
```

```
            YScale: 'linear'
     GridLineStyle: '-'
          Position: [0.1300 0.1100 0.7750 0.8150]
             Units: 'normalized'

  Use GET to show all properties

make axes
```



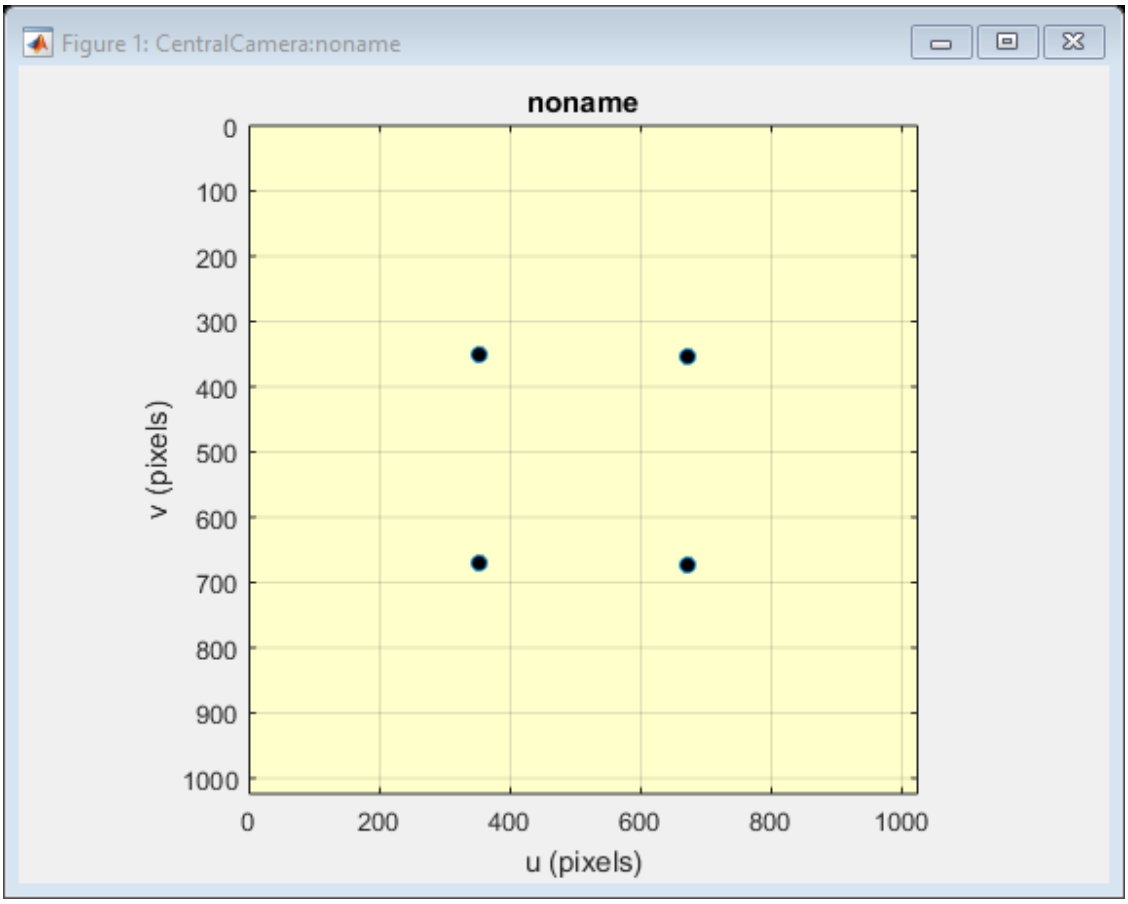
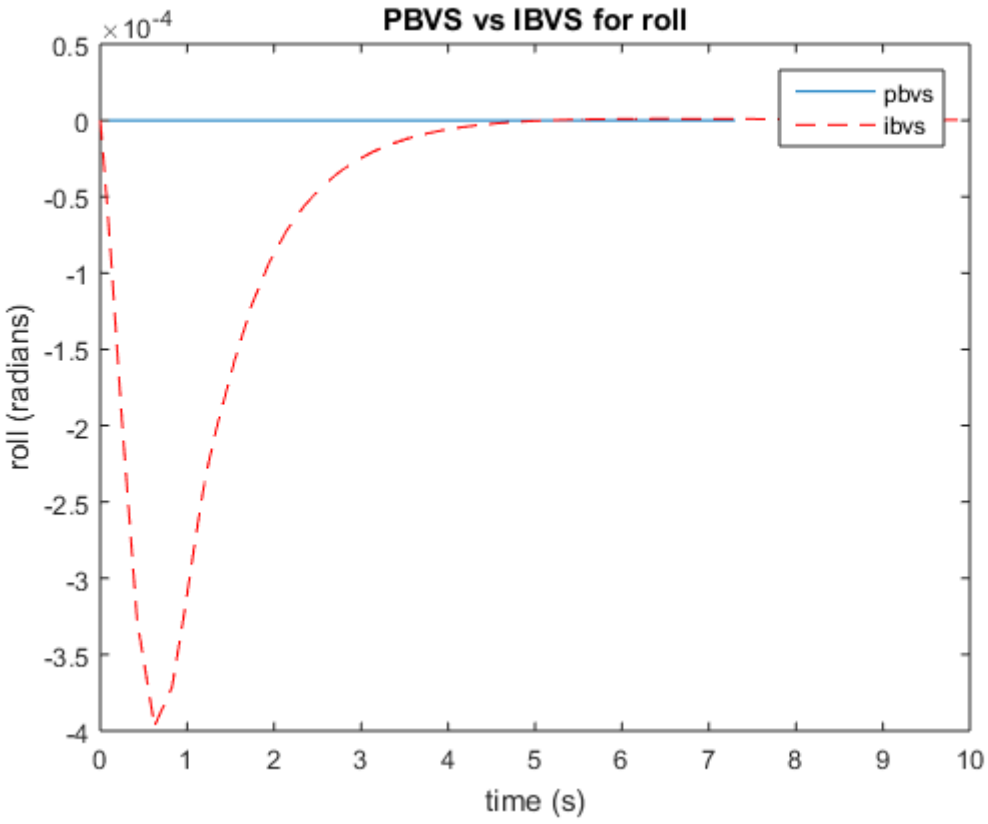
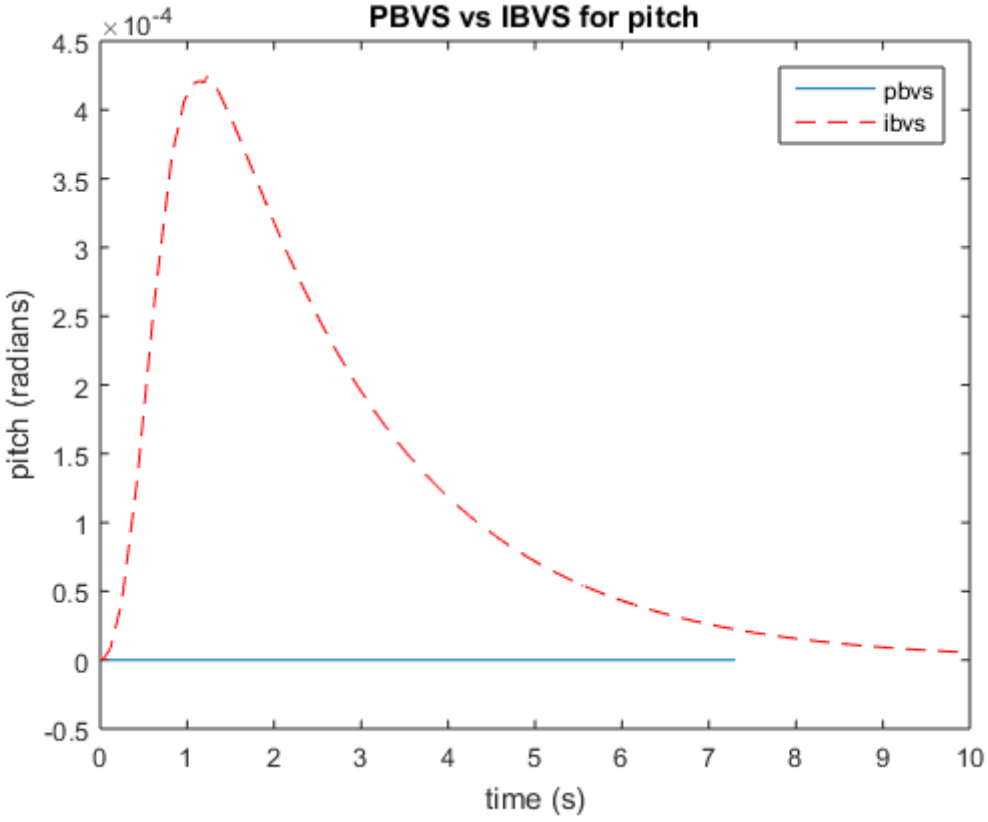
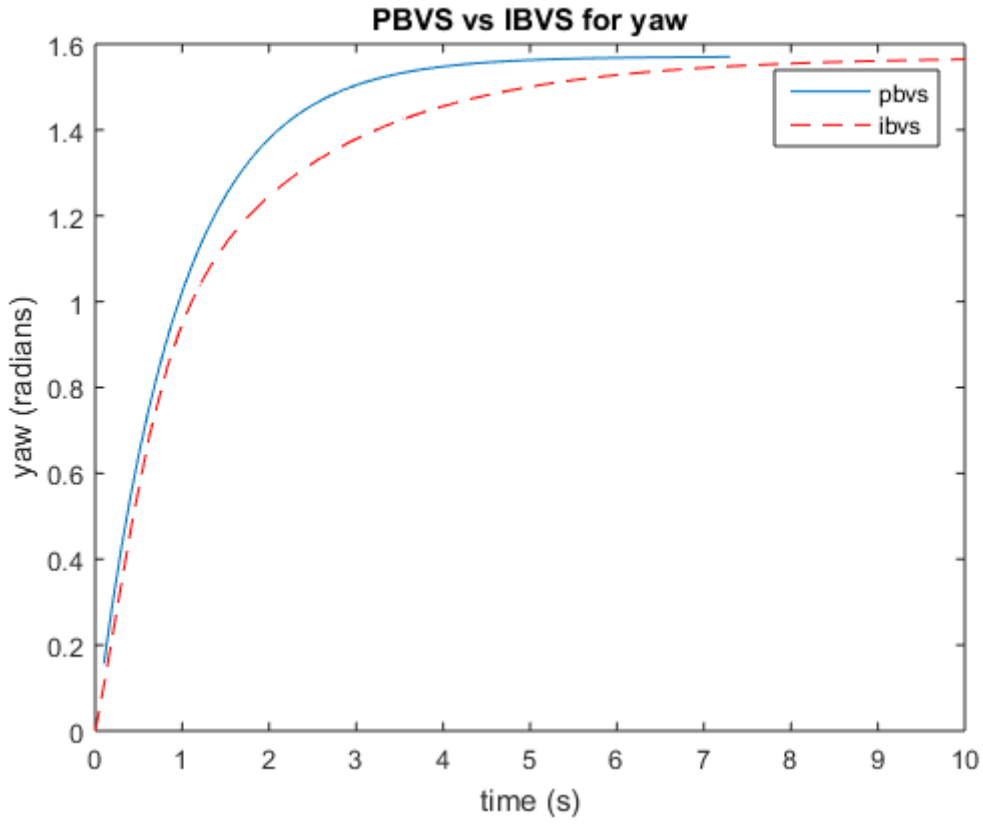
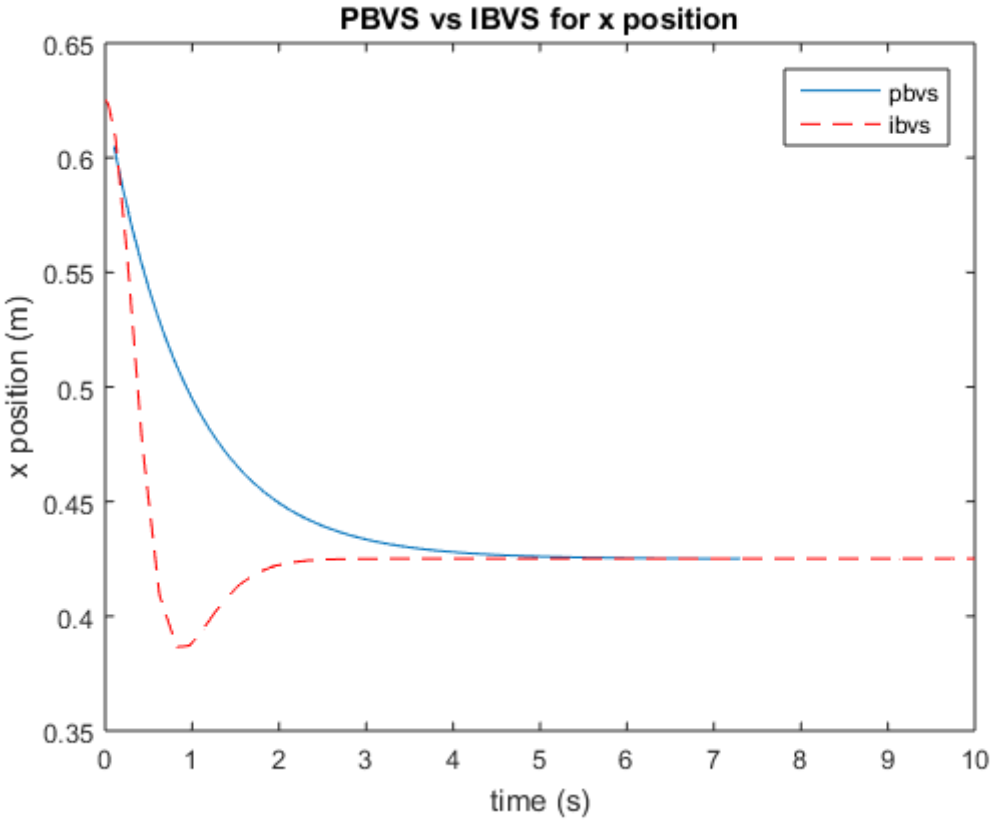


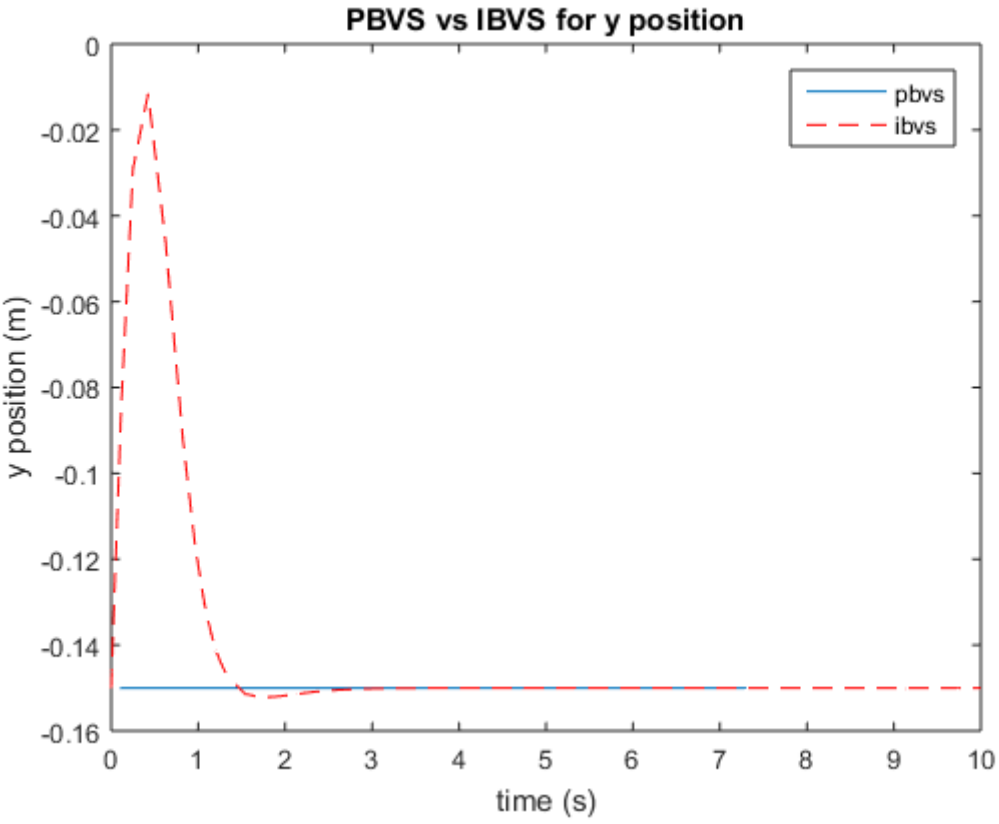
Figure 1: CentralCamera:noname
noname
v (pixels)
u (pixels)
0
100
200
300
400
500
600
700
800
900
1000
0
200
400
600
800
1000

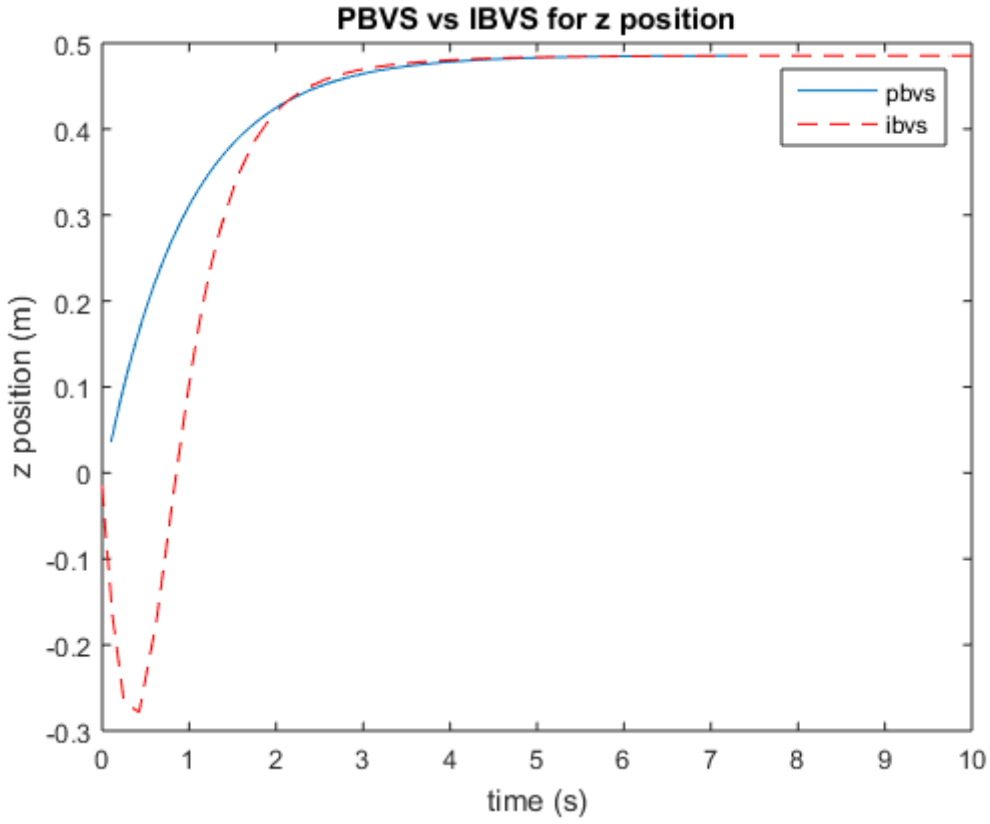

PBVS vs IBVS for roll
×10^-4
0.5
0
-0.5
-1
-1.5
-2
-2.5
-3
-3.5
-4
roll (radians)
0
1
2
3
4
5
6
7
8
9
10
time (s)
pbvs
ibvs


PBVS vs IBVS for pitch
×10^-4
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
-0.5
pitch (radians)
0
1
2
3
4
5
6
7
8
9
10
time (s)
pbvs
ibvs


PBVS vs IBVS for yaw
yaw (radians)
time (s)
pbvs
ibvs
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
0
1
2
3
4
5
6
7
8
9
10


PBVS vs IBVS for x position
x position (m)
time (s)
pbvs
ibvs
0.35
0.4
0.45
0.5
0.55
0.6
0.65
0
1
2
3
4
5
6
7
8
9
10


PBVS vs IBVS for y position
pbvs
ibvs
y position (m)
time (s)
0
-0.02
-0.04
-0.06
-0.08
-0.1
-0.12
-0.14
-0.16
0
1
2
3
4
5
6
7
8
9
10


PBVS vs IBVS for z position
pbvs
ibvs
z position (m)
time (s)
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
0
1
2
3
4
5
6
7
8
9
10

Published with MATLAB® R2015a

Problem 6:

See code below for my implementation of PBVS with the computed torque controller. The most important take-away is the form of the computed torque controller. I tested the computed torque controller in isolation and it worked very well. However, when I included the PBVS in the loop, it would drive the object's points out of the field of view of the camera. This is a common error with PBVS. However, I varied both the step size in PBVS and the controller as well as attempting to include a better desired velocity using trajecotory generation. None of this seemed to help. Finally, I took the commanded joint angles directly from problem 4 and just sent them to the computed torque controller. That worked very well and kept the object in the camera's view the entire time. So there is a weird interaction here with numerical IK that I am using for PBVS and the computed torque controller.

```
run D:\Dropbox\vision-3.4\rvctools\startup_rvc.m
run D:\Dropbox\rvctools_new\startup_rvc.m
clear all;
close all;

%make a camera
cam = CentralCamera('default');

%instantiate robot model, initial position, and vectors for plotting
mdl_puma560;
p560 = p560.nofriction();
q = [0, -pi/4, 0, 0, pi/4, 0];
qd = zeros(1, 6);
rpy_pbvs_cntrl = [];
pos_pbvs_cntrl = [];
q_pbvs_cntrl = [];
t_pbvs_cntrl = [];

%get the end effector position and set the camera at that position
T = p560.fkine(q);
cam.T = T;

%this transform defines where the object starts relative to my initial
 robo
%configuration
%this moves the object -0.2 meters in the x and 1 meter in z away from
 my
%camera/end effector position
move_obj_position = [-0.2; 0; 1;]
%T_points = T*[rotz(pi/2), move_obj_position;
T_points = T*[eye(3), move_obj_position;
              0, 0, 0, 1];

%this is the definition of the nominal vertices for our object in
%homogenous coordinates
P_nom = [-0.1   0.1 -0.1   0.1;
         0.1    0.1 -0.1  -0.1;
         0      0    0      0;
         1      1    1      1]

%this is transforming the nominal object model out to its actual
 location
P = T_points*P_nom
P = P(1:3,:);

%get the pixel coordinates of the object in the camera now
p = cam.project(P);
cam.plot(P);

%from that we can estimate where the object is relative to the camera
 at
%least
```

```
T_est_cam = cam.estpose(P_nom, p);

%then we can define a desired position for the camera relative to the
%object, which in this case is just 1.0 meter back in the z-direction
 and
%aligned with the object
T_des = T*T_est_cam*[eye(3), [0; 0; -0.5];
                     0 0 0 1];

%plot everything for the initial conditions
figure();
p560.plot(q);
cam.plot_camera();
plot_sphere(P, 0.05, 'r');

%this is just a starting condition for the while loop
error = 1;

time = 0;
while error > 0.001
    %find how we can take small step towards our desired camera
 position
    T_cmd = trnorm(trinterp(T, T_des, 0.1));

    %do inverse kinematics to find the new commanded joint angles.
    %Weird interactions here between the output of this function and
    %simulated computed torque controller below are causing problems
    q_goal = p560.ikine(T_cmd, q);

    % I tried using trajectory generation, but it didn't immediately
 help
    %    [q_goals,qd_goals,] = jtraj(q, q_goal, [0:0.01:0.1]);
    qd_goal = zeros(1, p560.n); %this is a hack because I'm not
 generating trajectories

    %forward simulate using the torque controller in "computed_torque"
    [t, q_model, qd_model] = p560.fdyn(0.1, @computed_torque, ...
                         q, qd, q_goal, qd_goal)

    time = time + t(end);
    q = q_model(end,:);
    qd = qd_model(end,:);

    %find how the end effector and camera have moved based on our
 robot
    %kinematic  model
    T = p560.fkine(q);
    cam.T = T;

    %update all plots
    cam.plot(P)
    p560.plot(q)
    cam.plot_camera()
    plot_sphere(P, 0.05, 'r')
```

```
    %project the object into the camera image plane now that we've
 moved
    p = cam.project(P)

    %solve for the relative pose of the object with respect to the
 camera
    %agian
    T_est_cam = cam.estpose(P_nom, p);

    %and define a desired camera pose relative to that position for
 the
    %camera
    T_des = T*T_est_cam*[eye(3), [0; 0; -0.5];
        0 0 0 1];

    %calculate our error in terms of roll, pitch, yaw and then
 position too
    rpy_des = tr2rpy(T_des);
    rpy = tr2rpy(T);
    error = norm(T(1:3,4)-T_des(1:3,4))+norm(rpy_des-rpy);

    rpy_pbvs_cntrl = [rpy_pbvs_cntrl; rpy]
    pos_pbvs_cntrl = [pos_pbvs_cntrl; T(1:3,4)'];
    q_pbvs_cntrl = [q_pbvs_cntrl; q];
    t_pbvs_cntrl = [t_pbvs_cntrl; time];

end
```

Published with MATLAB® R2015a