Problem A: Cubic Polynomial Trajectory:

A Cubic Polynomial Trajectory is generated using the formulation stated below.

$$\begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 \\ 0 & 1 & 2t_0 & 3t_0^2 \\ 1 & tf & t_f^2 & t_f^3 \\ 0 & 1 & 2t_f & 3t_f^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} q_0 \\ \dot{q}_0 \\ q_f \\ \dot{q}_f \end{bmatrix}$$

Which is implemented in the TrajGeneration.m file as shown below:

```
THETA1 = [theta1_initial theta1_dot_initial theta1_final theta1_dot_final]';
THETA2 = [theta2_initial theta2_dot_initial theta2_final theta2_dot_final]';

TIME_Coeff = [1 time_initial time_initial^2 time_initial^3;
                      0 1 2*time_initial 3*(time_initial^2);
              1 time_final time_final^2 time_final^3;
                      0 1 2*time_final 3*(time_final^2)];

Traj_Coeff1 = TIME_Coeff\THETA1;
Traj_Coeff2 = TIME_Coeff\THETA2;
```

The desired trajectory of theta1, theta2, theta1_dot, theta2_dot, theta1_ddot and theta2_ddot is obtained as follows:

:ommand Window

```
>> TrajGeneration
********** Desired Trajectory of Theta1 **********
(pi*t^3)/500 - (3*pi*t^2)/100 - t/18014398509481984 + pi

********** Desired Trajectory of Theta2 **********
(pi*t^3)/1000 - (3*pi*t^2)/200 - t/36028797018963968 + pi/2

********** Desired Trajectory of Theta1_dot **********
(3*pi*t^2)/500 - (3*pi*t)/50 - 1/18014398509481984

********** Desired Trajectory of Theta2_dot **********
(3*pi*t^2)/1000 - (3*pi*t)/100 - 1/36028797018963968

********** Desired Trajectory of Theta1_ddot **********
(3*pi*t)/250 - (3*pi)/50

********** Desired Trajectory of Theta2_ddot **********
(3*pi*t)/500 - (3*pi)/100
```

Problem B: Manipulator Form:

Equation of Motion with **Actual** Parameters:

theta1_ddot*(I1 + I2 + M1*r1^2 + M2*(L1^2 + r2^2) + 2*L1*M2*r2*cos(theta2)) + theta2_ddot*(M2*r2^2 + L1*M2*cos(theta2)*r2 + I2) - M2*g*(r2*sin(theta1 + theta2) + L1*sin(theta1)) - M1*g*r1*sin(theta1) - L1*M2*r2*theta2_dot*sin(theta2)*(theta1_dot + theta2_dot) - L1*M2*r2*theta1_dot*theta2_dot*sin(theta2)

L1*M2*r2*sin(theta2)*theta1_dot^2 + theta1_ddot*(M2*r2^2 + L1*M2*cos(theta2)*r2 + I2) + theta2_ddot*(M2*r2^2 + I2) - M2*g*r2*sin(theta1 + theta2)

Equation of Motion with **Nominal** Parameters:

theta1_ddot*(I1_hat + I2_hat + M1_hat*r1^2 + M2_hat*(L1^2 + r2^2) + 2*L1*M2_hat*r2*cos(theta2)) + theta2_ddot*(M2_hat*r2^2 + L1*M2_hat*cos(theta2)*r2 +

I2_hat) - M2_hat*g*(r2*sin(theta1 + theta2) + L1*sin(theta1)) - M1_hat*g*r1*sin(theta1) - L1*M2_hat*r2*theta2_dot*sin(theta2)*(theta1_dot + theta2_dot) - L1*M2_hat*r2*theta1_dot*theta2_dot*sin(theta2)

L1*M2_hat*r2*sin(theta2)*theta1_dot^2 + theta1_ddot*(M2_hat*r2^2 + L1*M2_hat*cos(theta2)*r2 + I2_hat) + theta2_ddot*(M2_hat*r2^2 + I2_hat) - M2_hat*g*r2*sin(theta1 + theta2)

<u>Problem C: </u> Robust Inverse Dynamics:

Virtual Control Input Design –

```
A = [0 0 1 0; 0 0 0 1; 0 0 0 0; 0 0 0 0];
B = [0 0; 0 0; 1 0; 0 1];
lambda = [-3 -3 -4 -4];
K = place(A,B,lambda);
Kp = K(:,1:2);
Kd = K(:,3:4);
O = [0 0; 0 0];
Acl = [O eye(2); -Kp, -Kd] ;
Q = eye(4)*20;
P = lyap(Acl',Q);
rho = 3.25;
phi = 0.075;
```

```
*******************************************
********** K Gains **********
K =

   12.0000         0    7.0000         0
         0   12.0000         0    7.0000


Kp =

   12.0000         0
         0   12.0000



Kd =

    7.0000         0
         0    7.0000



P =

   24.4048         0    0.8333         0
         0   24.4048         0    0.8333
    0.8333         0    1.5476         0
         0    0.8333         0    1.5476
```

Problem D: ODE update Feedback Linearization Control

Control Input Design:

```
    G = [e; e_dot];
    if boundary_layer
      if norm(B'*P*G) > phi
            Vr = -(rho*(B'*P*G))/norm(B'*P*G);
      Else
            Vr = -(rho*(B'*P*G))/phi;
      End
    Else
      if norm(B'*P*G) ~= 0
            Vr = -(rho*(B'*P*G))/norm(B'*P*G);
      Else
            Vr = [0;0];
      End
  End
V = feed_foward_input - Kp*e - Kd*e_dot + Vr;
U = Mmat_hat*V + Cmat_hat*[y(i,3); y(i,4)] + Gmat_hat;
```

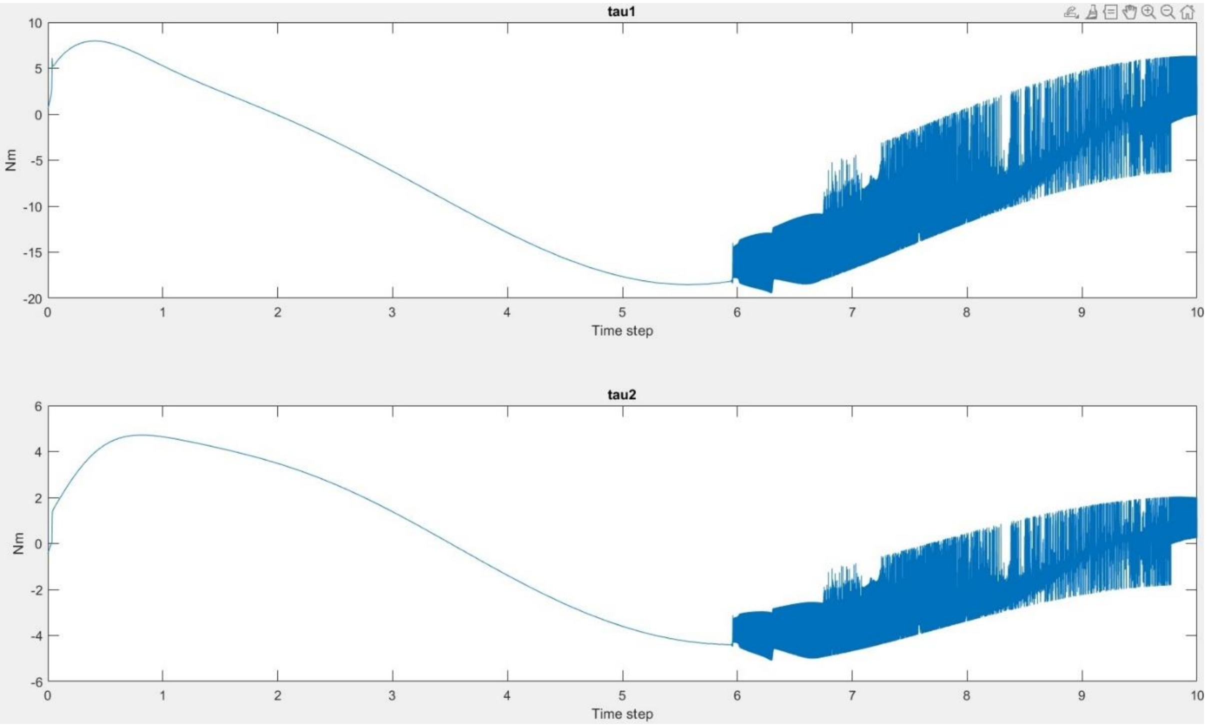
System Design:

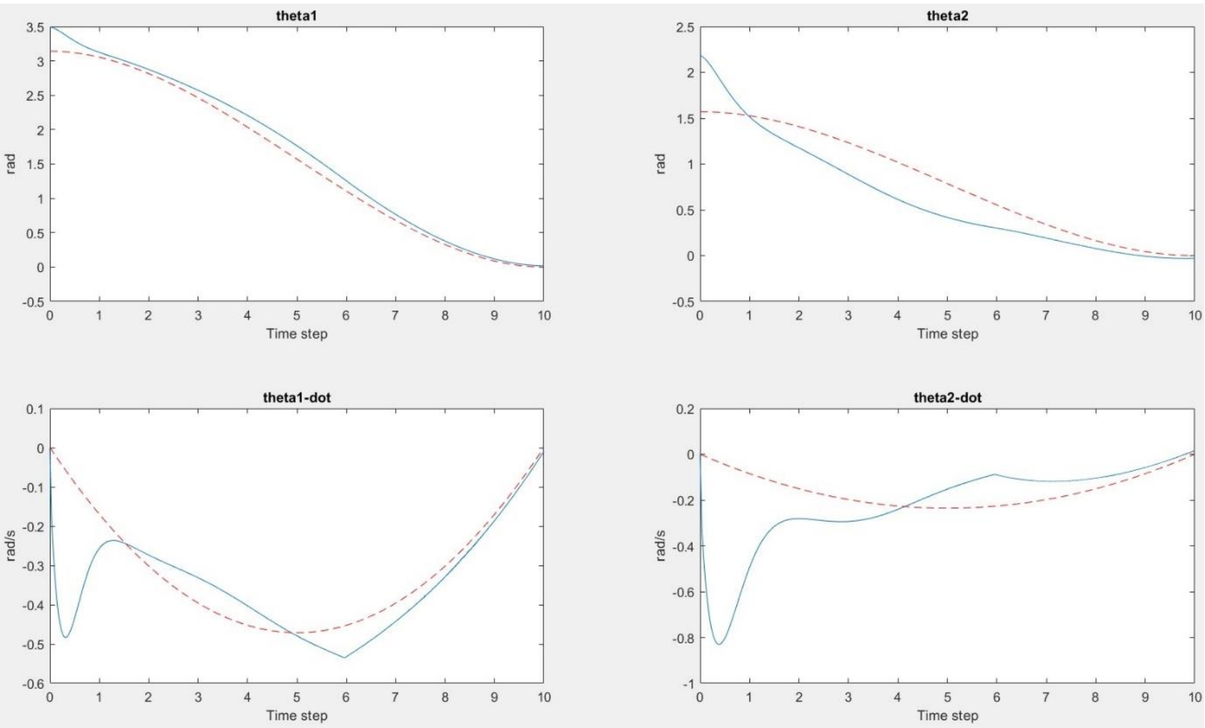
```
dX(1) = theta1_dot;
dX(2) = theta2_dot;
dX(3) = (I2*tau1 - I2*tau2 + M2*r2^2*tau1 - M2*r2^2*tau2 +
L1*M2^2*g*r2^2*sin(theta1) + I2*L1*M2*g*sin(theta1) +
I2*M1*g*r1*sin(theta1) - L1*M2*r2*tau2*cos(theta2) +
L1*M2^2*r2^3*theta1_dot^2*sin(theta2) +
L1*M2^2*r2^3*theta2_dot^2*sin(theta2) +
L1^2*M2^2*r2^2*theta1_dot^2*cos(theta2)*sin(theta2) -
L1*M2^2*g*r2^2*sin(theta1 + theta2)*cos(theta2) +
I2*L1*M2*r2*theta1_dot^2*sin(theta2) +
I2*L1*M2*r2*theta2_dot^2*sin(theta2) + M1*M2*g*r1*r2^2*sin(theta1) +
2*L1*M2^2*r2^3*theta1_dot*theta2_dot*sin(theta2) +
2*I2*L1*M2*r2*theta1_dot*theta2_dot*sin(theta2))/(-
```

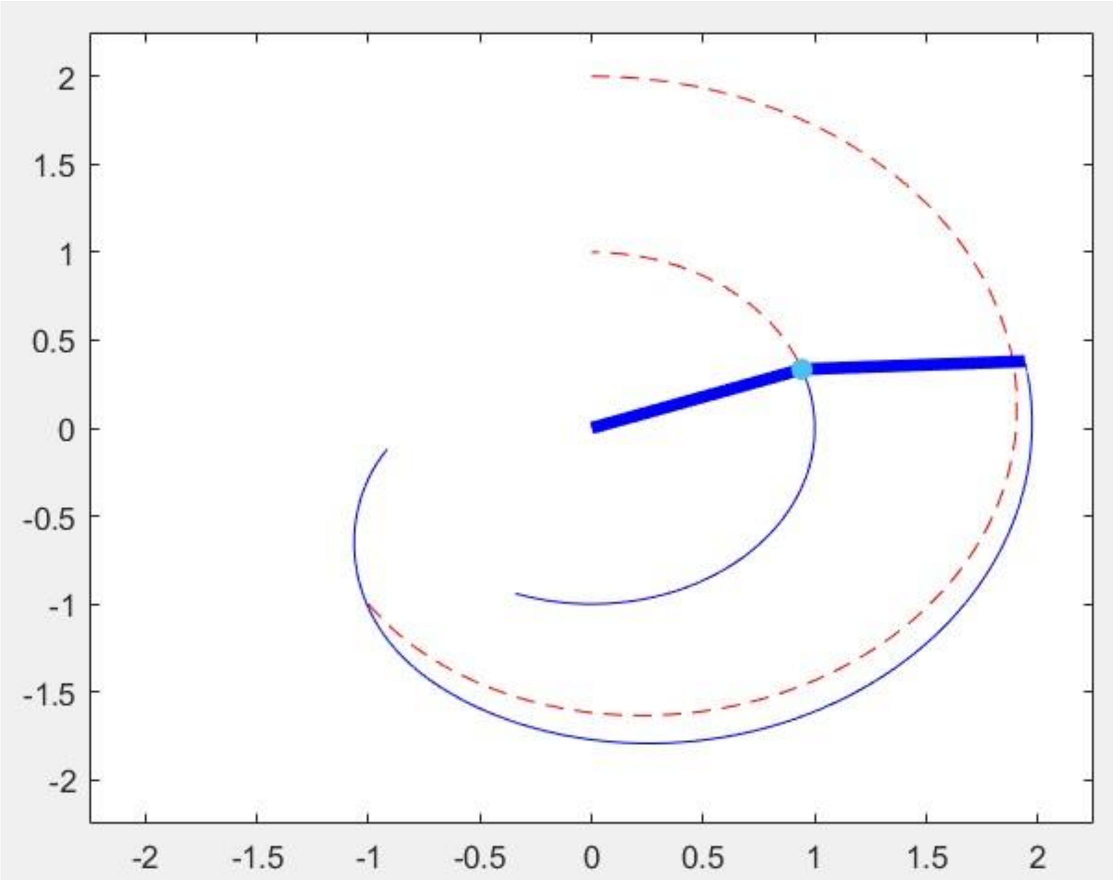
```
L1^2*M2^2*r2^2*cos(theta2)^2 + L1^2*M2^2*r2^2 + I2*L1^2*M2 +
M1*M2*r1^2*r2^2 + I1*M2*r2^2 + I2*M1*r1^2 + I1*I2);

dX(4) = -(I2*tau1 - I1*tau2 - I2*tau2 - L1^2*M2*tau2 - M1*r1^2*tau2
+ M2*r2^2*tau1 - M2*r2^2*tau2 - L1^2*M2^2*g*r2*sin(theta1 + theta2)
+ L1*M2^2*g*r2^2*sin(theta1) - I1*M2*g*r2*sin(theta1 + theta2) +
I2*L1*M2*g*sin(theta1) + I2*M1*g*r1*sin(theta1) +
L1*M2*r2*tau1*cos(theta2) - 2*L1*M2*r2*tau2*cos(theta2) +
L1*M2^2*r2^3*theta1_dot^2*sin(theta2) +
L1^3*M2^2*r2*theta1_dot^2*sin(theta2) +
L1*M2^2*r2^3*theta2_dot^2*sin(theta2) +
2*L1^2*M2^2*r2^2*theta1_dot^2*cos(theta2)*sin(theta2) +
L1^2*M2^2*r2^2*theta2_dot^2*cos(theta2)*sin(theta2) -
L1*M2^2*g*r2^2*sin(theta1 + theta2)*cos(theta2) +
L1^2*M2^2*g*r2*cos(theta2)*sin(theta1) - M1*M2*g*r1^2*r2*sin(theta1
+ theta2) + I1*L1*M2*r2*theta1_dot^2*sin(theta2) +
I2*L1*M2*r2*theta1_dot^2*sin(theta2) +
I2*L1*M2*r2*theta2_dot^2*sin(theta2) + M1*M2*g*r1*r2^2*sin(theta1) +
2*L1*M2^2*r2^3*theta1_dot*theta2_dot*sin(theta2) +
2*L1^2*M2^2*r2^2*theta1_dot*theta2_dot*cos(theta2)*sin(theta2) +
L1*M1*M2*r1^2*r2*theta1_dot^2*sin(theta2) +
2*I2*L1*M2*r2*theta1_dot*theta2_dot*sin(theta2) +
L1*M1*M2*g*r1*r2*cos(theta2)*sin(theta1))/(-
L1^2*M2^2*r2^2*cos(theta2)^2 + L1^2*M2^2*r2^2 + I2*L1^2*M2 +
M1*M2*r1^2*r2^2 + I1*M2*r2^2 + I2*M1*r1^2 + I1*I2);
```

Problem E: **Matlab Simulation- NO BOUNDARY LAYER**

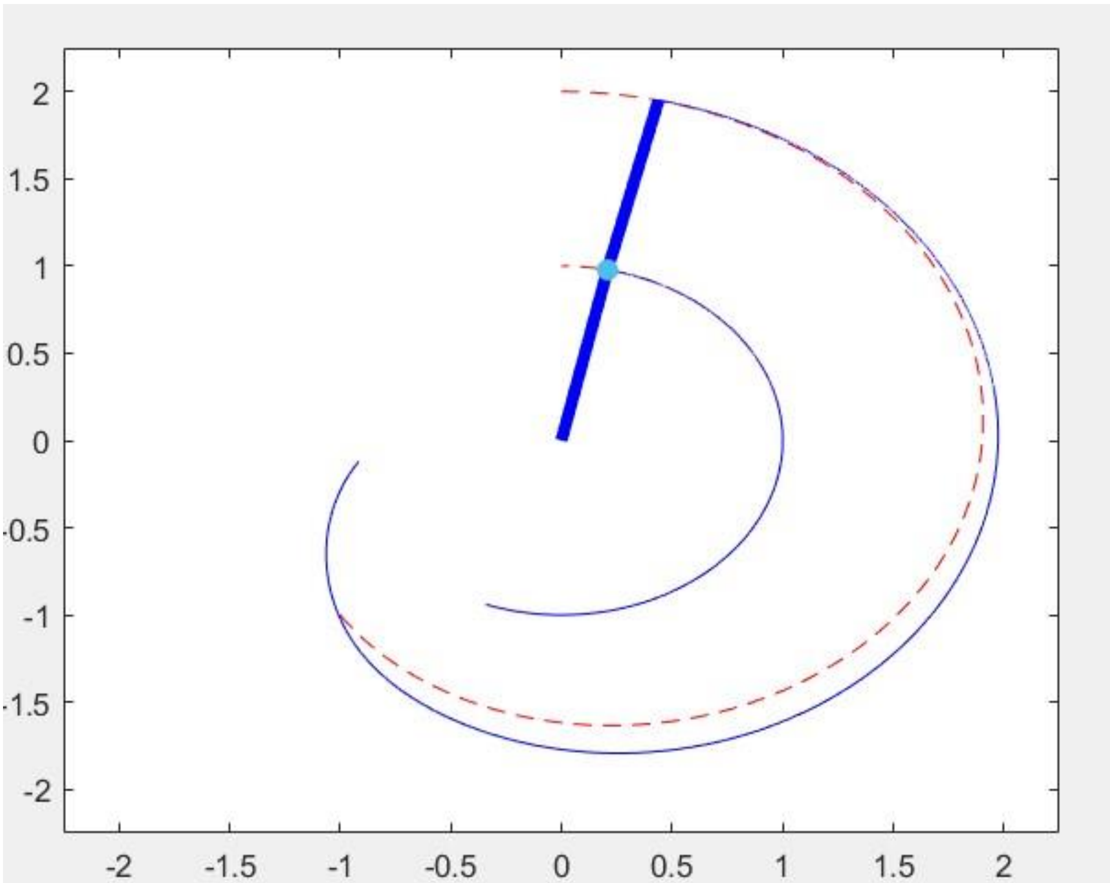
In the below figures, the red dashed line shows the desired state variables and the desired trajectory, and the blue line shows the obtained state variables.





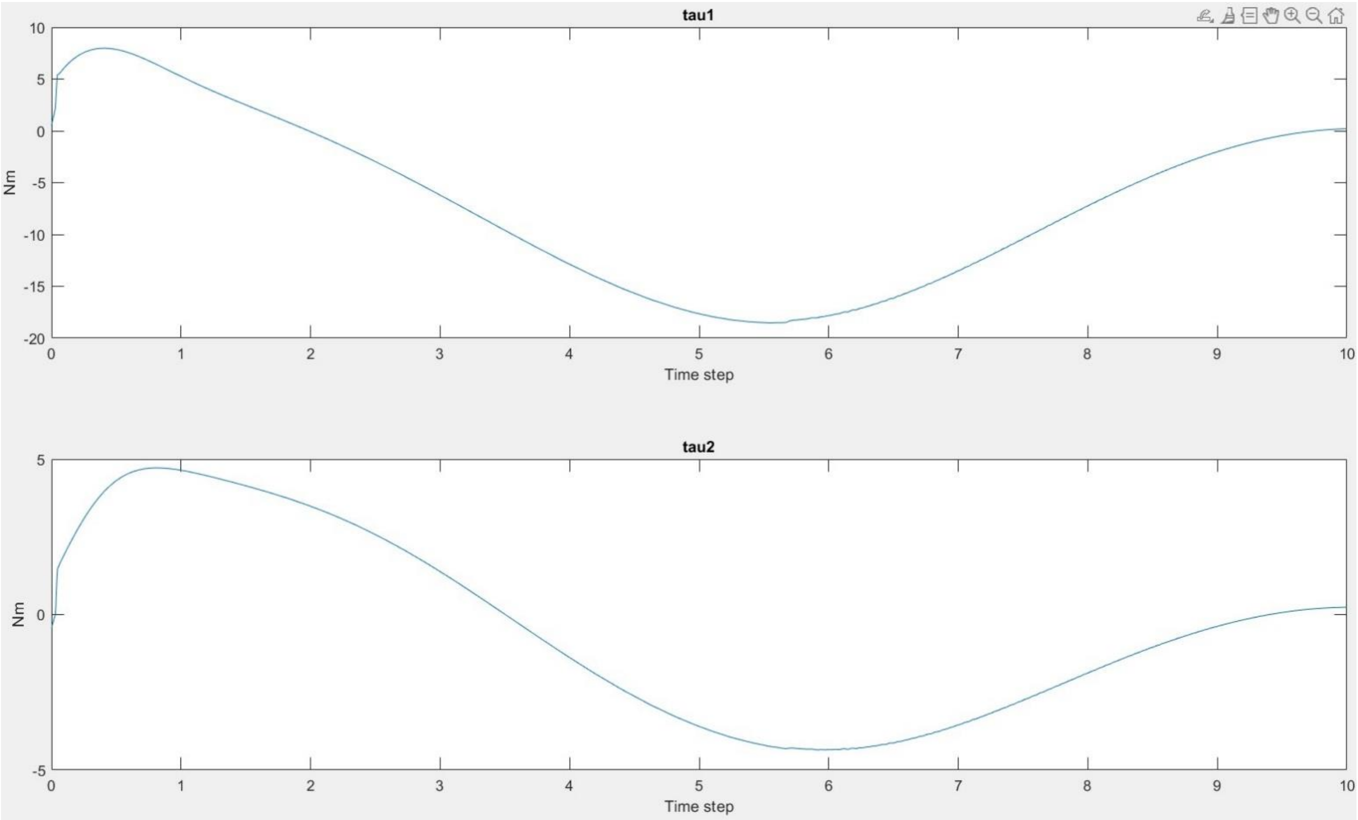


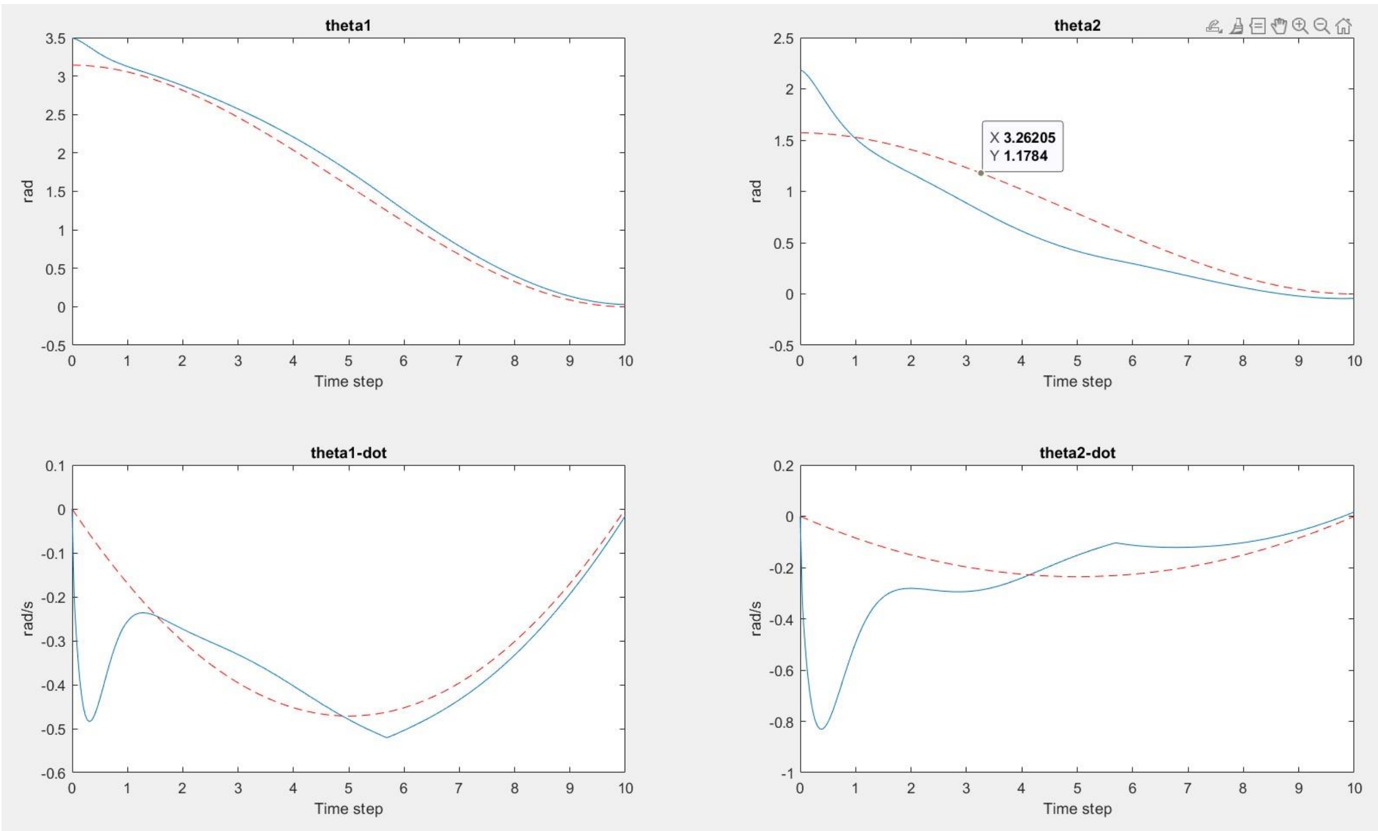
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

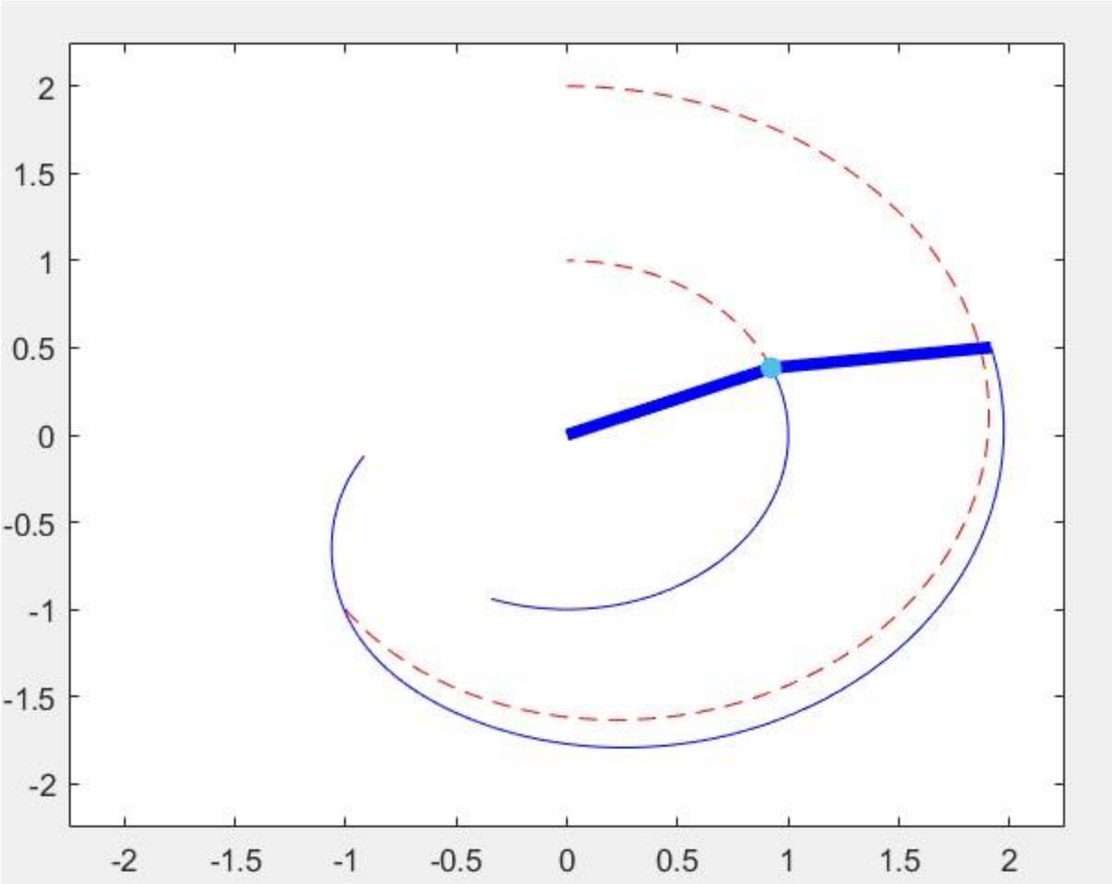


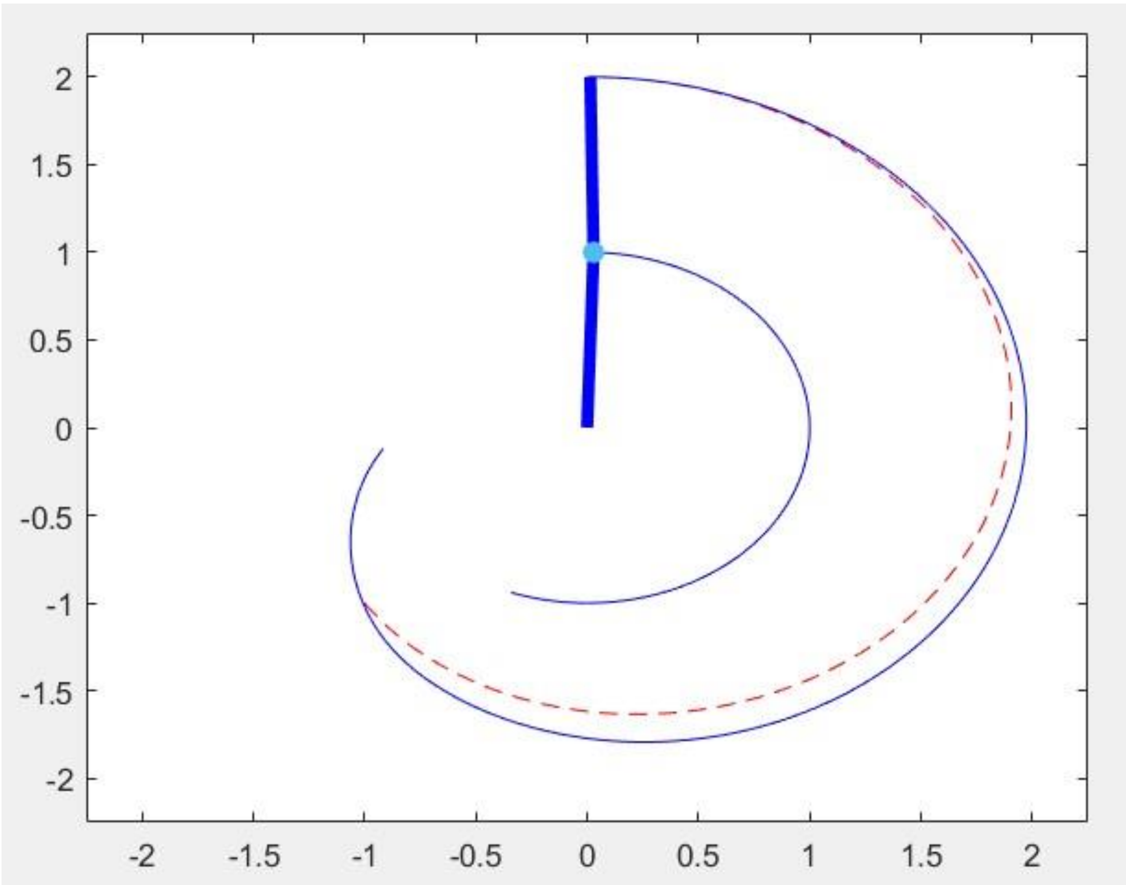
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

Problem F: Matlab Simulation- WITH BOUNDARY LAYER

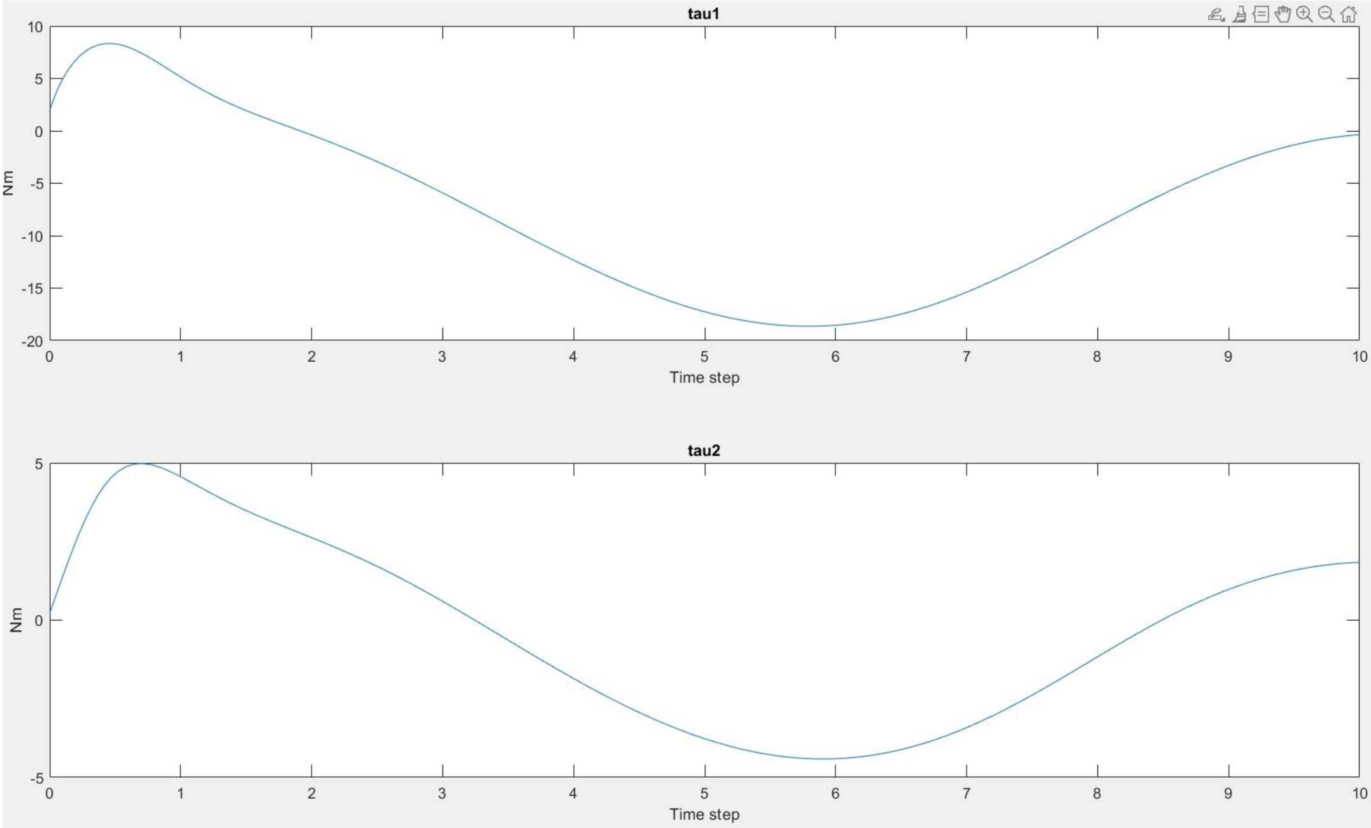


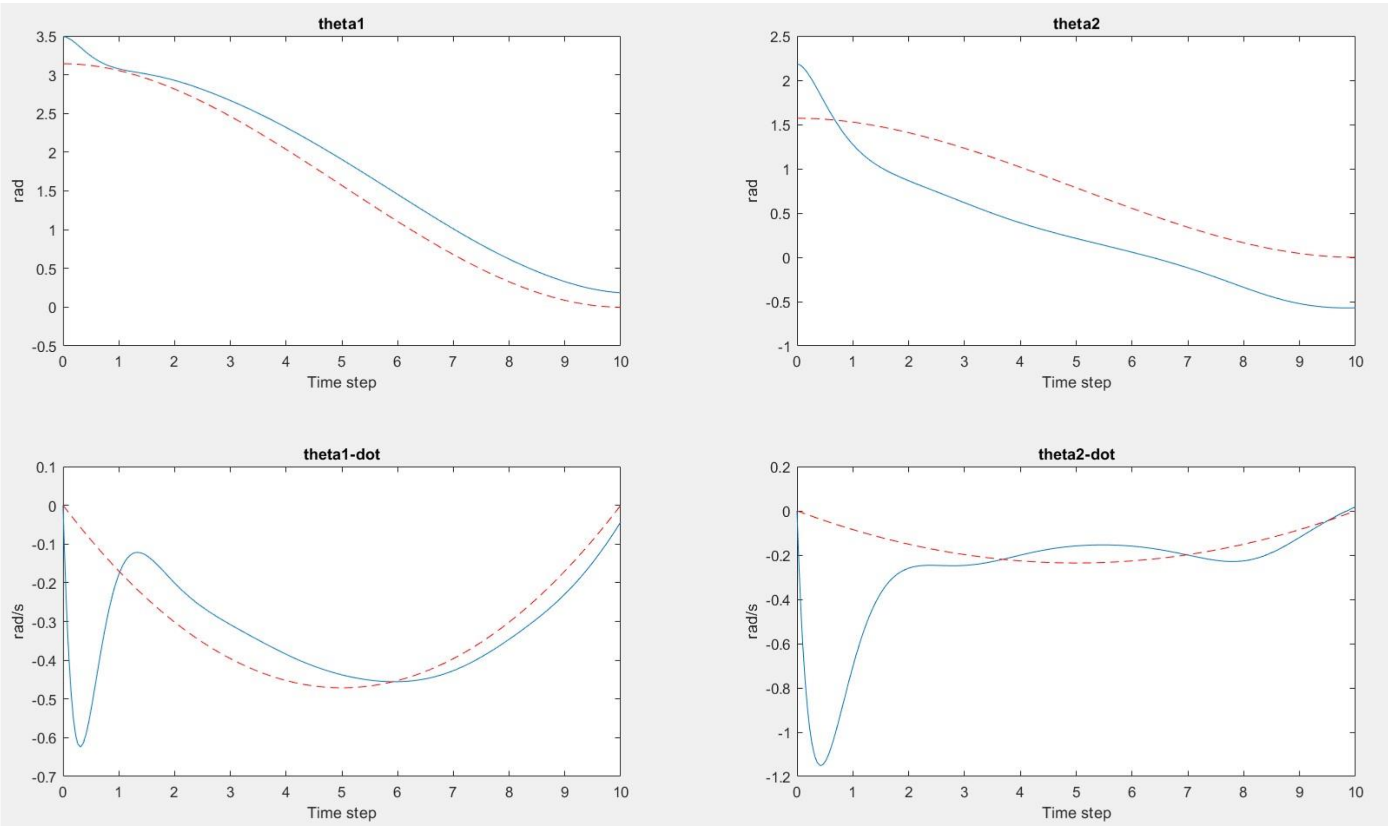


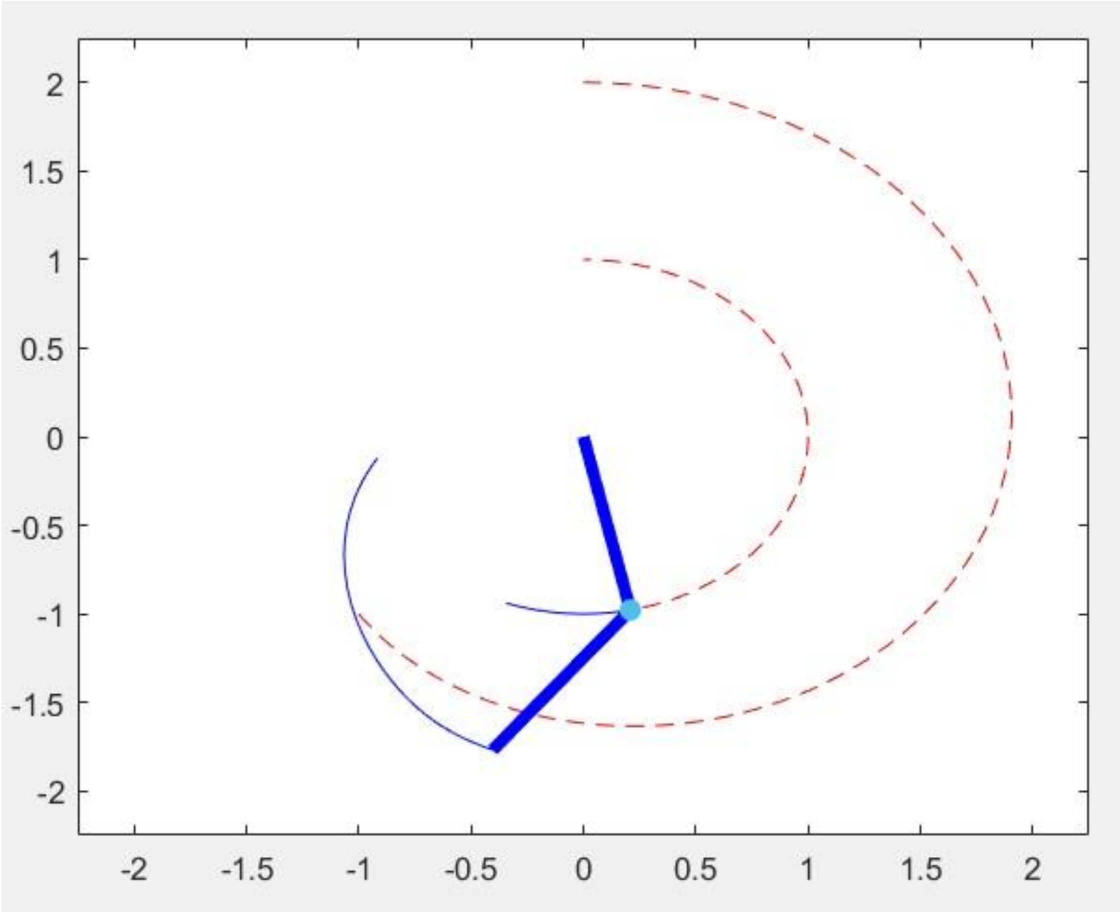

2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

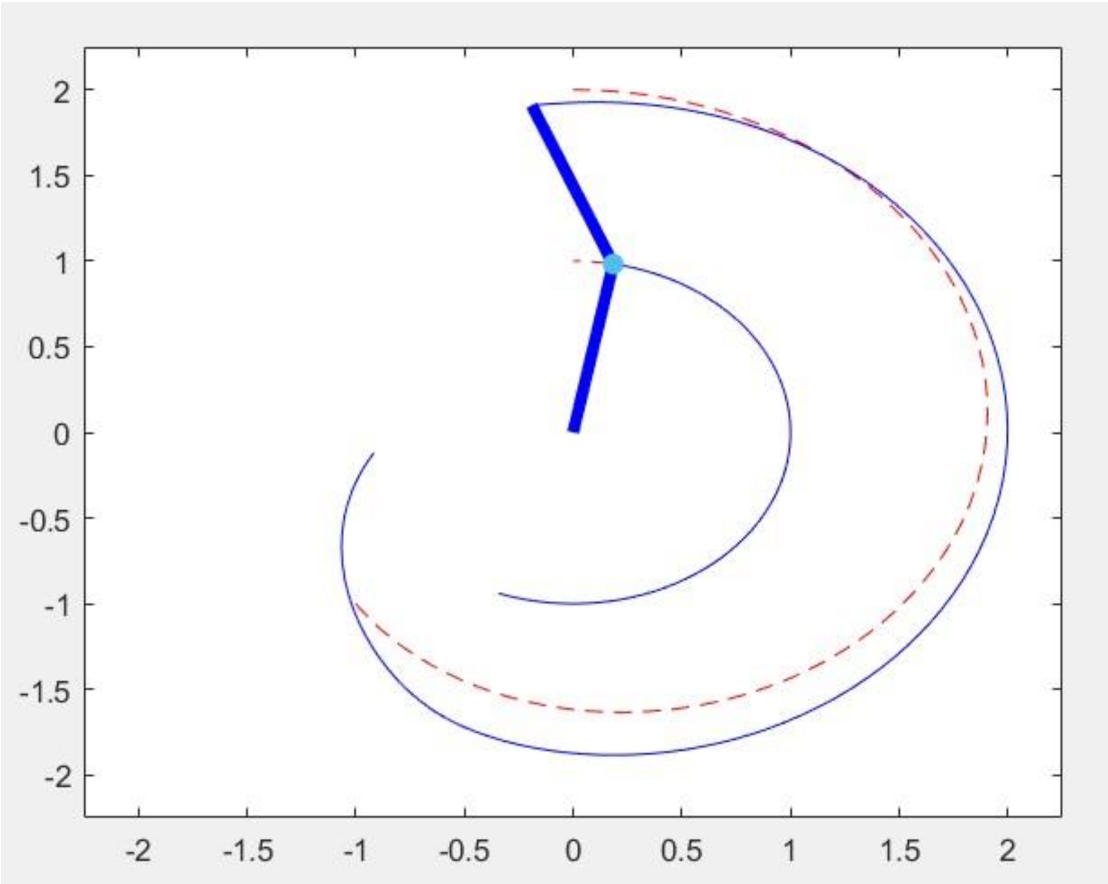

2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

Problem G: Matlab Simulation- VR = 0

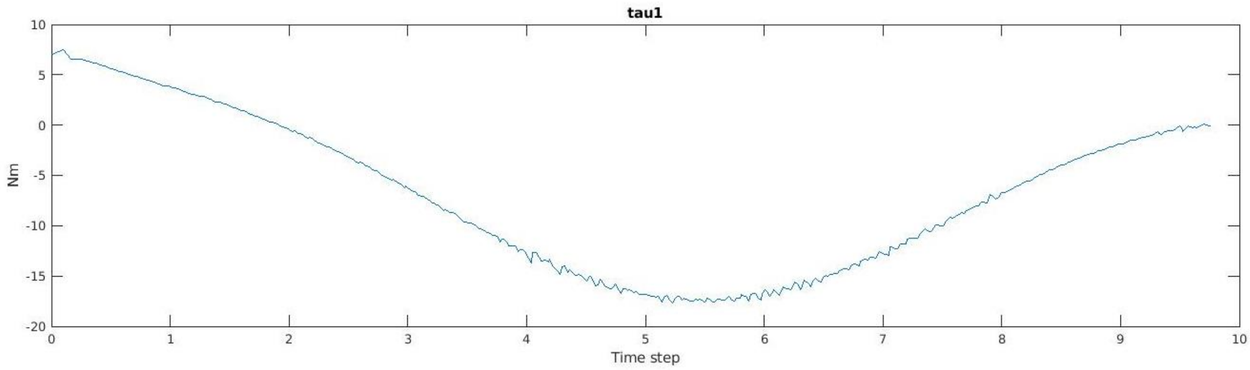


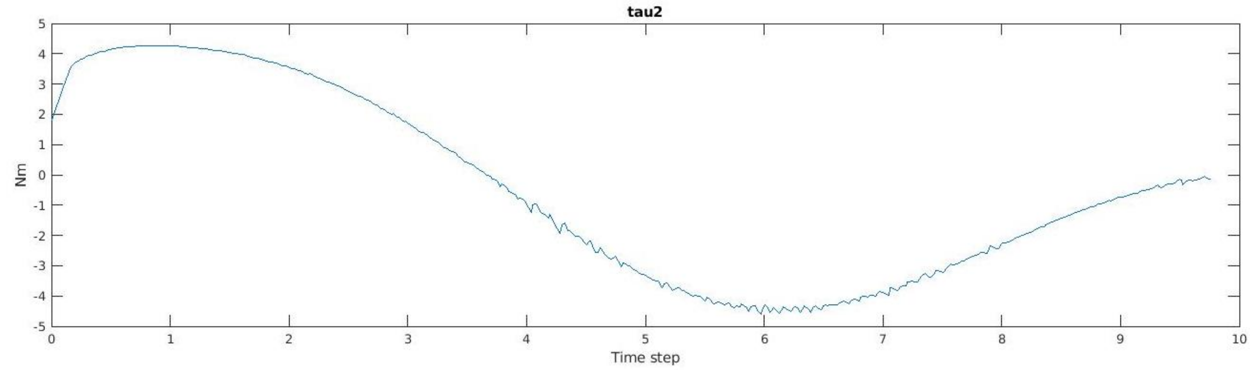


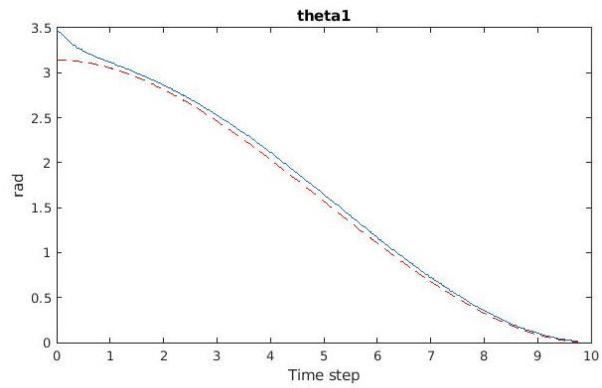

2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

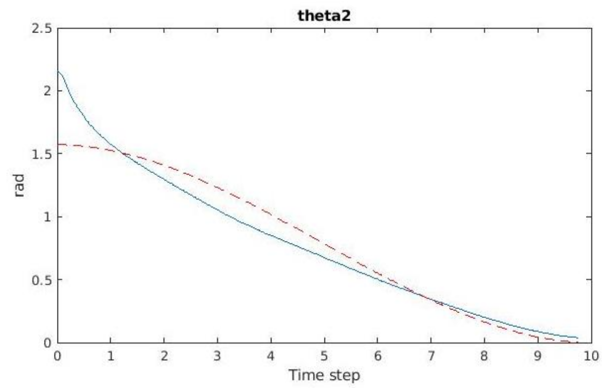

2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

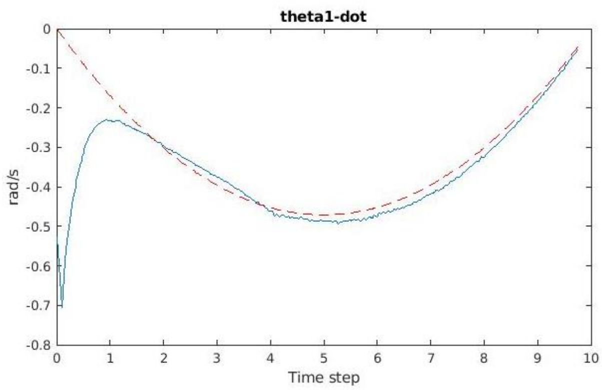
Problem H: ROS-Gazebo Simulation

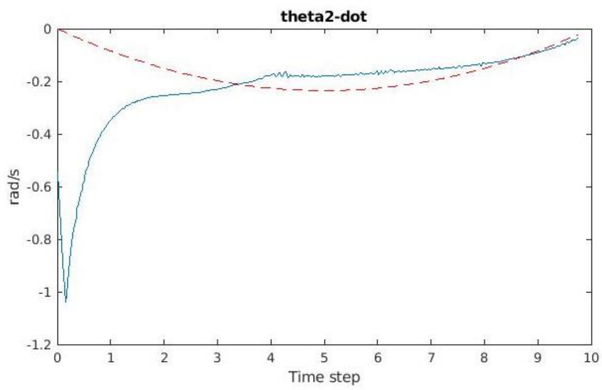


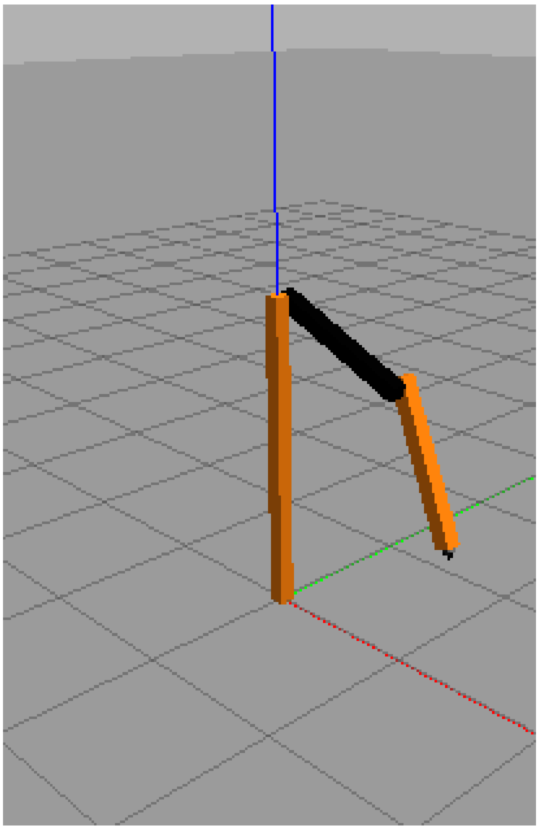


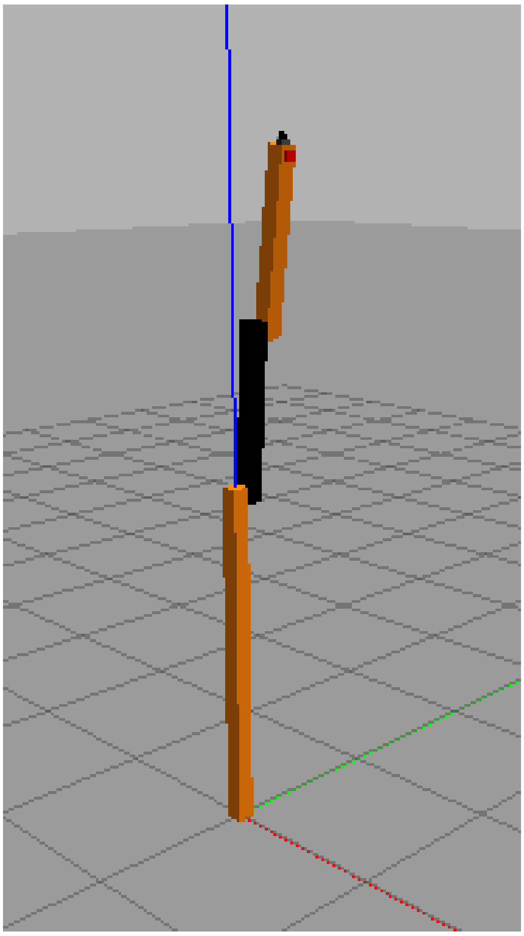












Discussion on ROS results:

The output graphs generated by simulating the RRBot in GAZEBO is different from that of the one found in MATLAB because of multiple reasons which are listed below.

- There is some sort of damping (Friction or Air Drag) that is present in the Gazebo environment which is evident by the fact that the robot comes to rest after oscillating for some time when no control input is applied. This is not considered in our dynamic modelling of the system and we get little higher torques in ROS-GAZEBO implementation as compared to MATLAB implementation.

- We can observe a little gitter in the control inputs in case of the GAZEBO simulation as it is simulated in a real physical environment.

- We can also observe that the initial velocity in the GAZEBO implementation is not Zero. This is because there is a difference between the start time of the GAZEBO simulation and the application of Control input which leads to some gain in velocity due to free-fall under gravity till the time the controller kicks-in. This velocity gain at the beginning can also be the reason for higher initial torques then those found in the MATLAB implementation.

- However, the GAZEBO implementation shows that our controller is applicable and works perfectly on a real-world system.

End Of Assignment