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:

```
Command 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:

The symbolic and numerical form of the 2x2 Mass matrix obtained original equation of motion is as follows:

```
*******************************************************************************************
********** Symbolic Mass Matrix M_q_symbolic **********
[M2*L1^2 + 2*M2*cos(theta2)*L1*r2 + M1*r1^2 + M2*r2^2 + I1 + I2, M2*r2^2 + L1*M2*cos(theta2)*r2 + I2]
[                         M2*r2^2 + L1*M2*cos(theta2)*r2 + I2,                         M2*r2^2 + I2]

********** Numerical Mass Matrix M_q_numerical **********
[(9*cos(theta2))/10 + 1573/1000, (9*cos(theta2))/20 + 573/2000]
[ (9*cos(theta2))/20 + 573/2000,                      573/2000]
```

The symbolic and numerical form of the 2x1 Gravity Vector obtained original equation of motion is as follows:

```
**************************************************************************
********** Symbolic Gravity Vector g_q_symbolic **********
-g*(M1*r1*sin(theta1) + M2*r2*sin(theta1 + theta2) + L1*M2*sin(theta1))
                                          -M2*g*r2*sin(theta1 + theta2)

********** Numerical Gravity Vector g_q_numerical **********
- (8829*sin(theta1 + theta2))/2000 - (28449*sin(theta1))/2000
                            -(8829*sin(theta1 + theta2))/2000
```

The symbolic and numerical form of the 2x1 Coriolis Term (2x2 coriolis metrix times the vector [theta1_dot, theta2_dot]) obtained original equation of motion is as follows:

```
**************************************************************
********** Symbolic Coriolis term C_q_qd_symbolic **********
-L1*M2*r2*theta2_dot*sin(theta2)*(2*theta1_dot + theta2_dot)
                           L1*M2*r2*theta1_dot^2*sin(theta2)

********** Numerical Coriolis term C_q_qd_numerical **********
-(9*theta2_dot*sin(theta2)*(2*theta1_dot + theta2_dot))/20
                           (9*theta1_dot^2*sin(theta2))/20
```

Equation of motion obtained by State Feedback Linearization

(Symbolic)

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

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)

(Numerical)

theta1_ddot*((9*cos(theta2))/10 + 1573/1000) - (28449*sin(theta1))/2000 - (9*theta2_dot^2*sin(theta2))/20 - (8829*sin(theta1 + theta2))/2000 + theta2_ddot*((9*cos(theta2))/20 + 573/2000) - (9*theta1_dot*theta2_dot*sin(theta2))/10

(9*sin(theta2)*theta1_dot^2)/20 + (573*theta2_ddot)/2000 - (8829*sin(theta1 + theta2))/2000 + theta1_ddot*((9*cos(theta2))/20 + 573/2000)

Problem C: Feedback Linearization Control:

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];
```

Eigne Values:

```
lambda = [-2 -4 -6 -8];
```

Eigne Value Placement:

```
K = place(A,B,lambda)
```

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

   48.0000         0   14.0000         0
         0    8.0000         0    6.0000
```

Control Input for Trajectory Tracking:

```
U = M_q*(-K*(X-X_desired) + feed_foward_input) + C_q_qd + g_q
```

Problem D: ODE **update Feedback Linearization Control**

Instead of continuously calculating the desired trajectory at each time step in the ODE function, I calculated a parametric desired trajectory for theta1, theta2, theta1_dot, theta2_dot, theta1_ddot, theta2_ddot.

```
theta1_desired = (pi*t^3)/500 - (3*pi*t^2)/100 - t/18014398509481984 + pi;
theta2_desired = (pi*t^3)/1000 - (3*pi*t^2)/200 - t/36028797018963968 +
pi/2;
theta1_dot_desired = (3*pi*t^2)/500 - (3*pi*t)/50 - 1/18014398509481984
theta2_dot_desired = (3*pi*t^2)/1000 - (3*pi*t)/100 - 1/36028797018963968;
theta1_ddot_desired = (3*pi*t)/250 - (3*pi)/50;
theta2_ddot_desired = (3*pi*t)/500 - (3*pi)/100;
```

And substituted these equations in the main equation to generate the control input.

```
U = M_q*(-K*(X-X_desired) + feed_foward_input) + C_q_qd + g_q
```

Here, X = [theta1, theta2, theta1_dot, theta2_dot]

X_desired = [theta1_desired, theta2_desired, theta1_dot_desired, theta2_dot_desired]

Now the obtained overall control intputs (law) tau1 and tau2 are as follows:

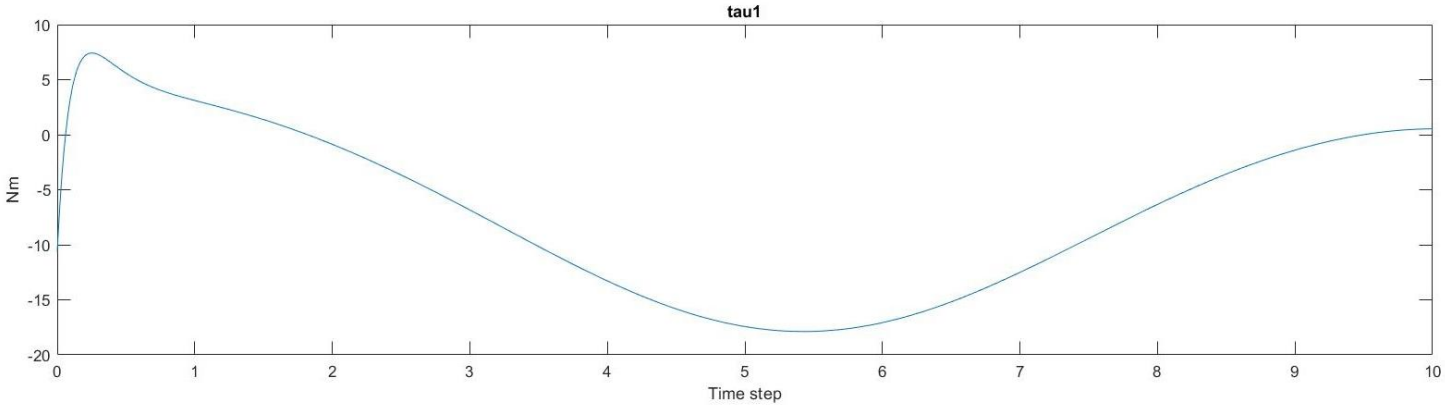
tau1 = - (8829*sin(theta1 + theta2))/2000 - (28449*sin(theta1))/2000 - ((9*cos(theta2))/20 + 573/2000)*(t/4503599627370496 + 8*theta2 + 6*theta2_dot - (397*pi)/100 + (87*pi*t)/500 + (51*pi*t^2)/500 - (pi*t^3)/125 + 3/18014398509481984) - ((9*cos(theta2))/10 + 1573/1000)*((3*t)/1125899906842624 + 48*theta1 + 14*theta1_dot - (2397*pi)/50 + (207*pi*t)/250 + (339*pi*t^2)/250 - (12*pi*t^3)/125 + 7/9007199254740992) - (9*theta2_dot*sin(theta2)*(2*theta1_dot + theta2_dot))/20

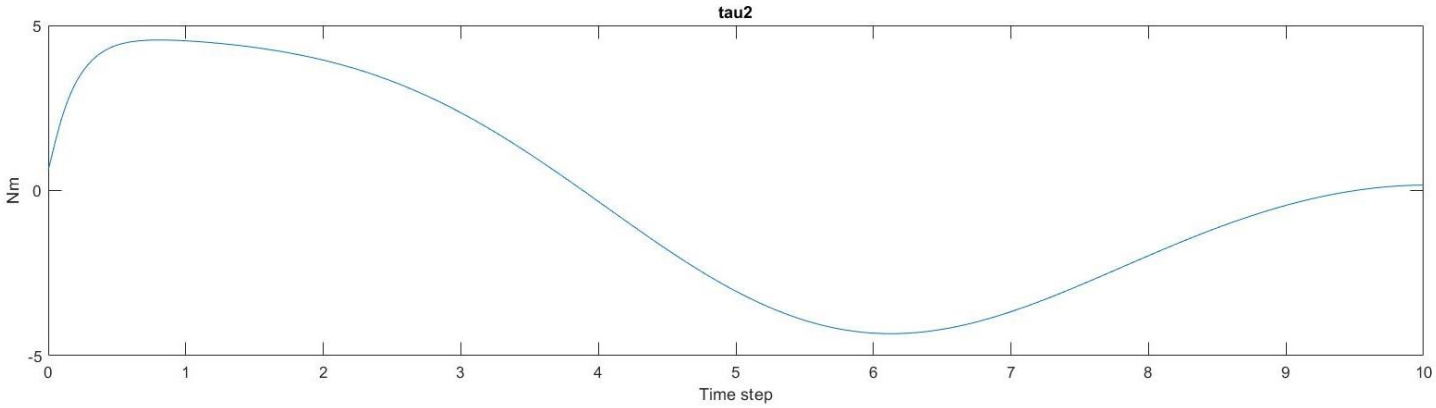
tau2 = (227481*pi)/200000 - (573*theta2)/250 - (1719*theta2_dot)/1000 - (573*t)/9007199254740992000 - (8829*sin(theta1 + theta2))/2000 + (9*theta1_dot^2*sin(theta2))/20 - (49851*pi*t)/1000000 - (29223*pi*t^2)/1000000 + (573*pi*t^3)/250000 - ((9*cos(theta2))/20 + 573/2000)*((3*t)/1125899906842624 + 48*theta1 + 14*theta1_dot - (2397*pi)/50 + (207*pi*t)/250 + (339*pi*t^2)/250 - (12*pi*t^3)/125 + 7/9007199254740992) - 1719/36028797018963968000

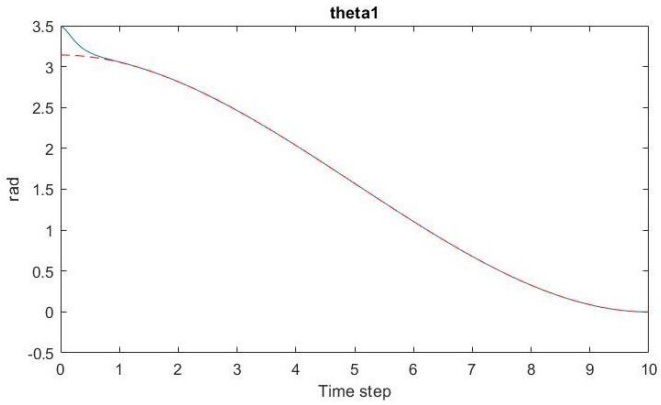
Here we can observe that tau1 and tau2 has only four variables theta1, theta2, theta1_dot and theta2_dot. Hence the ODE function can only update theta1 and theta2 at each time step to find the control inputs.

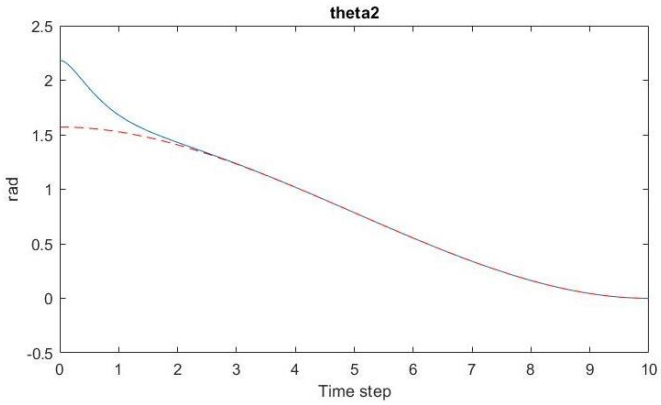
Problem E: Matlab Simulation

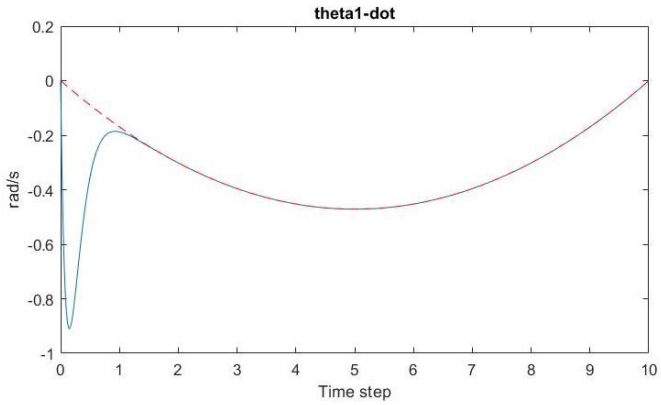
In the below figures, the red dashed line shows the desired state variables and the desired trajectory, and the blue line shows the obtained state variables.

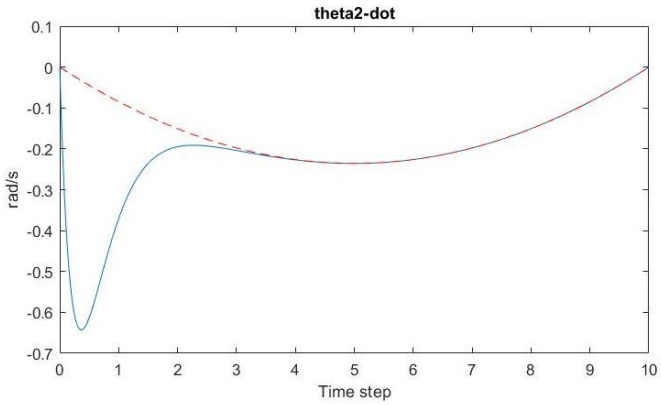






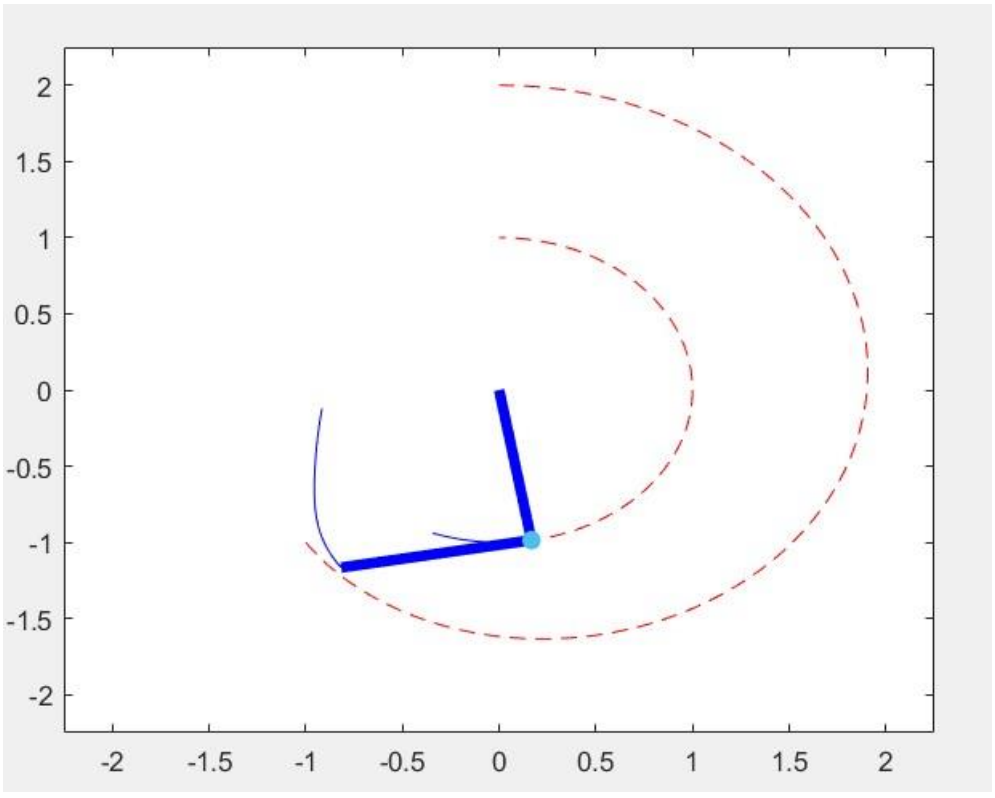




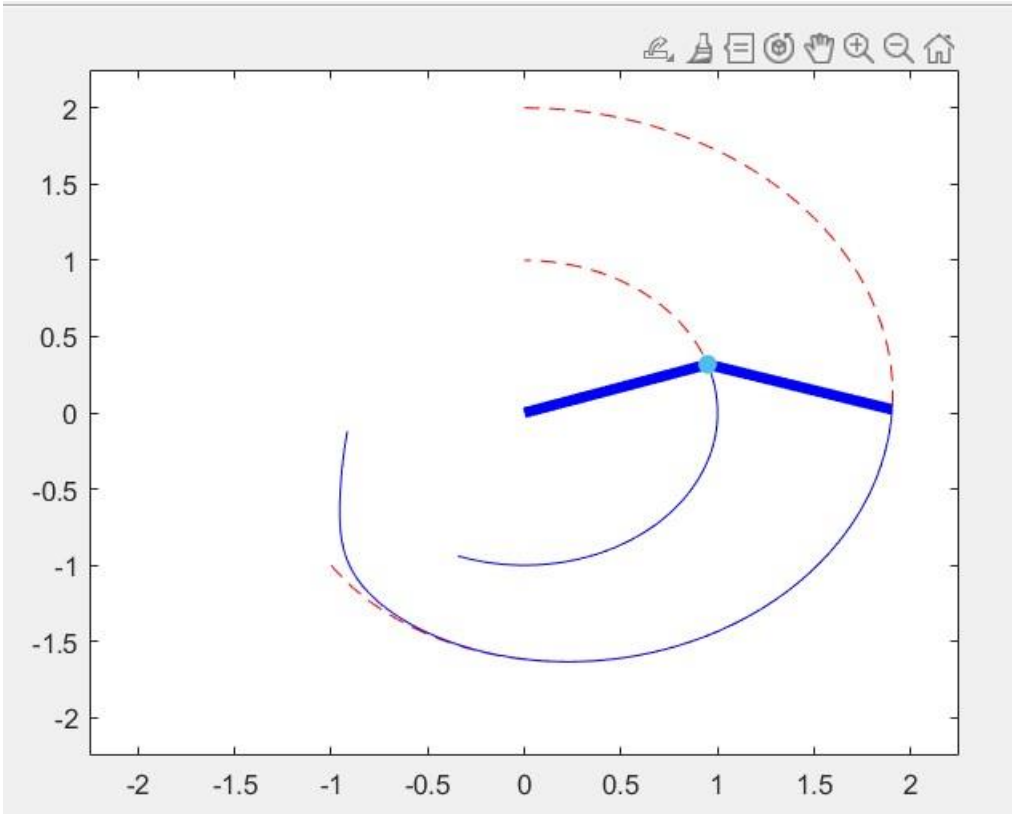


MATLAB SIMULATION:

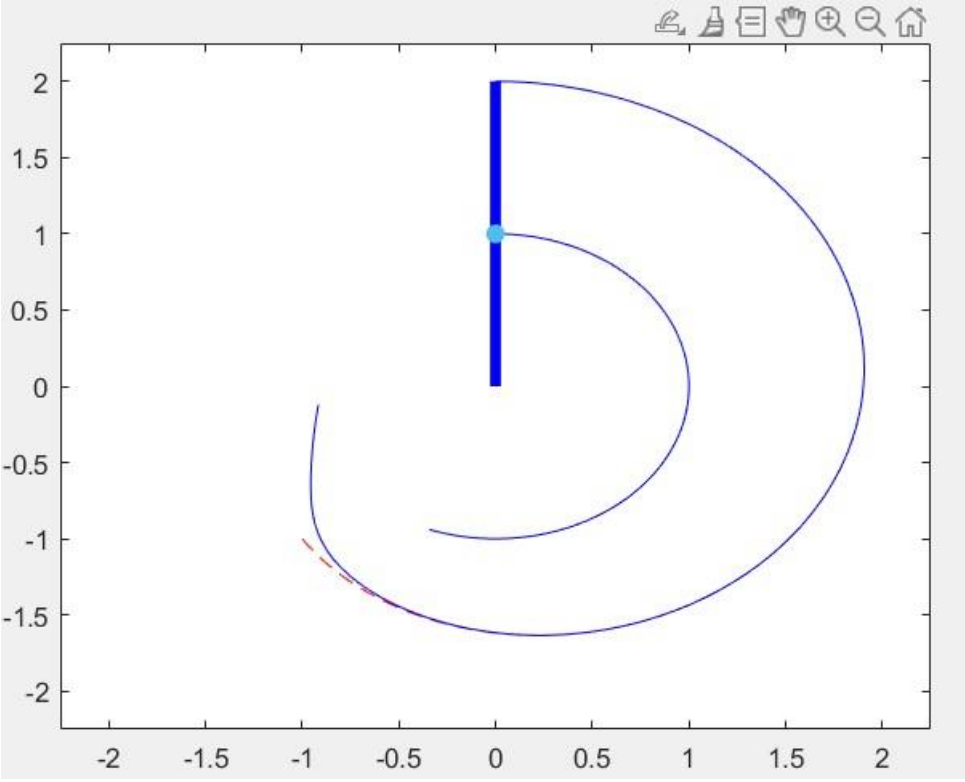
1



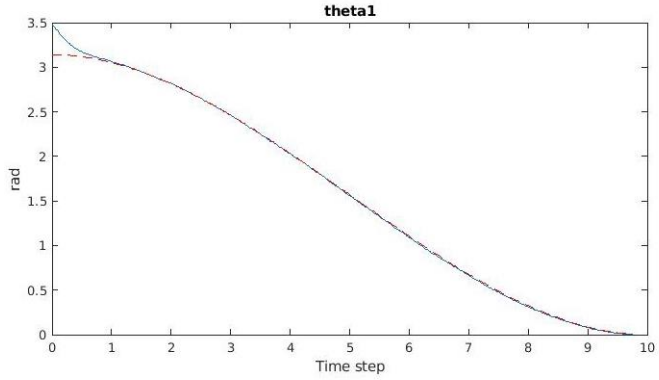
2

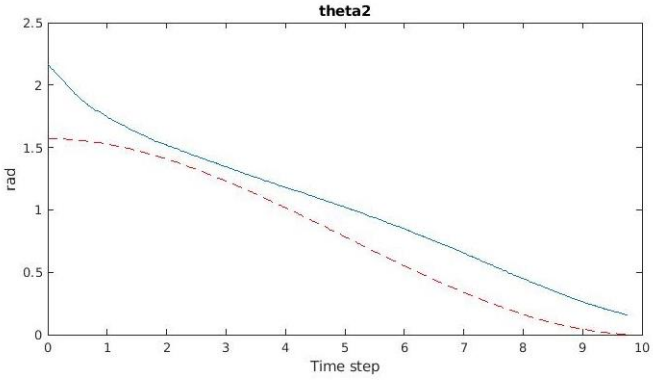

2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2 -1.5 -1 -0.5 0 0.5 1 1.5 2

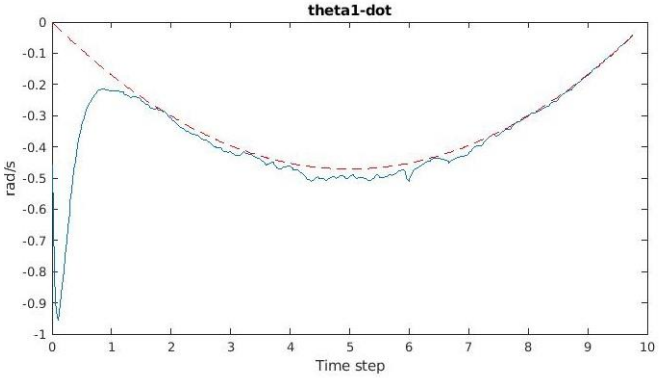
3

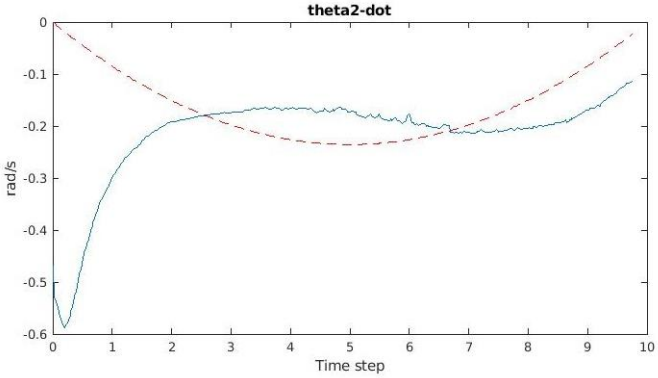

2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2 -1.5 -1 -0.5 0 0.5 1 1.5 2

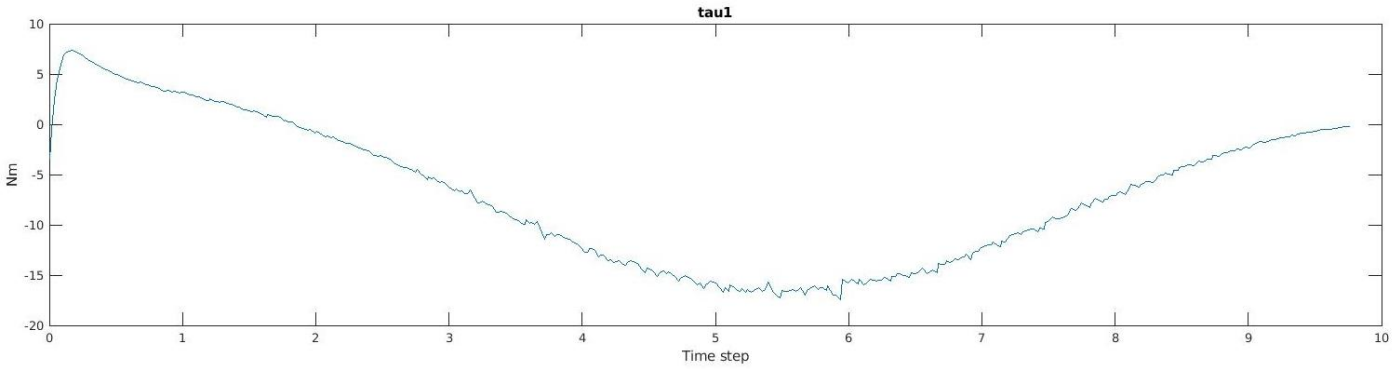
Problem F: ROS-GAZEBO Simulation

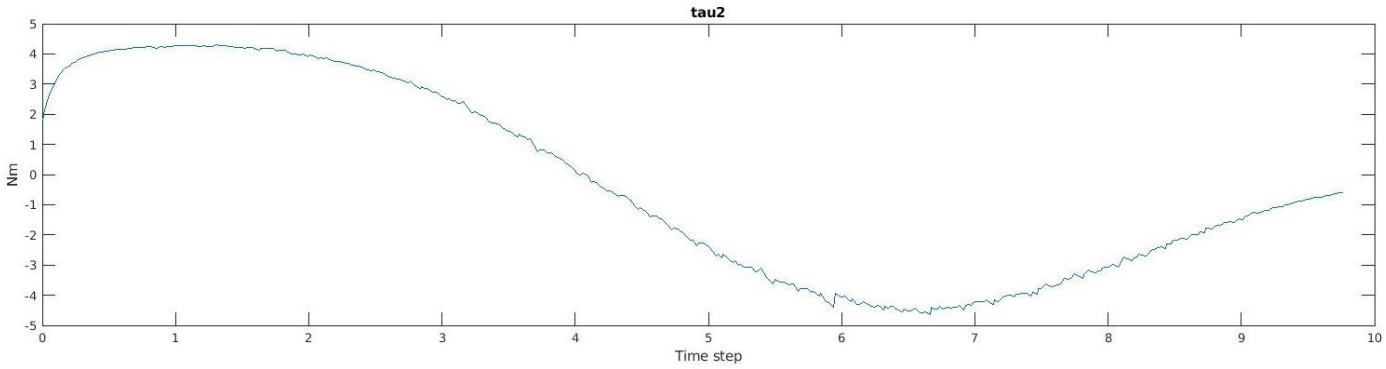




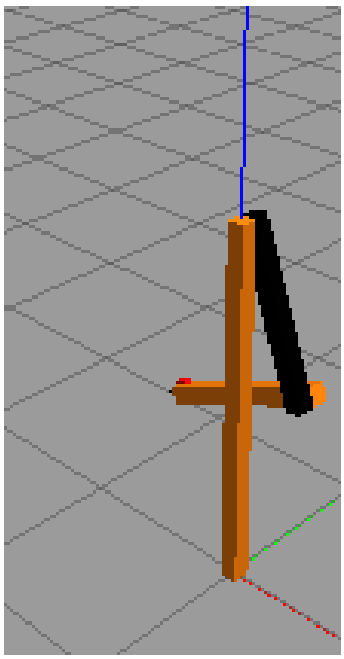


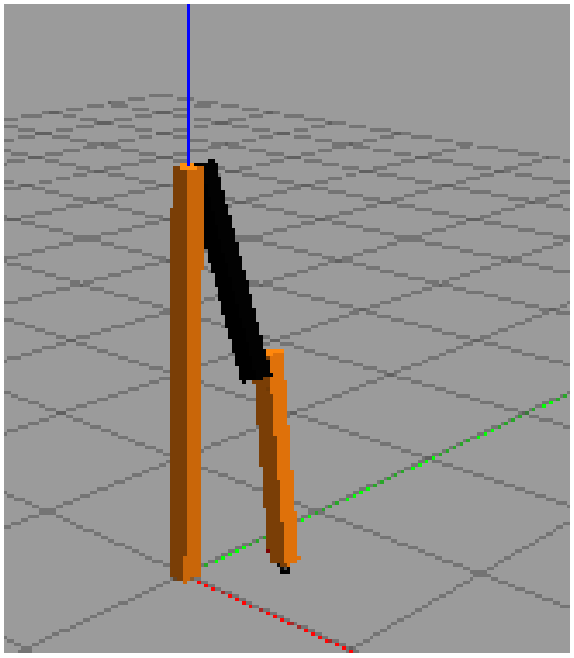


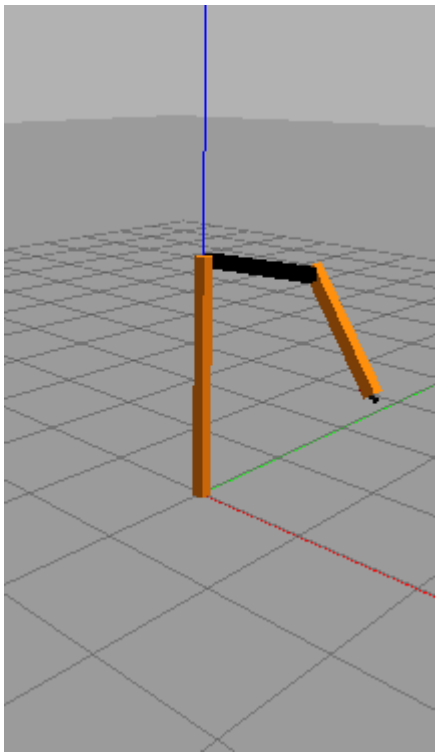


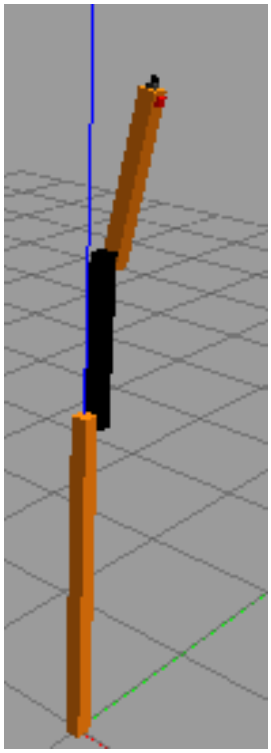


ROS SIMULATION:









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