Programming Assignment 4:
Robust Control of the RRBot Robotic Arm

By Chinmayee Prabhakar

The actual dynamic parameters are:

$$m1 = m2 = 1 \text{ (kg)}, l1 = l2 = 1 \text{ (m)}, r1 = r2 = 0.45 \text{ (m)}$$

$$I1 = I2 = 0.084 \text{ (kg} \cdot \text{m2)}, g = 9.81 \text{ (m/s2)}$$

a) The cubic polynomial trajectories for the given time span and the desired initial and final joint angles and velocities :

$$t0 = 0, tf = 10 \text{ sec}$$
$$\theta1(t0) = 180\circ, \theta1(tf) = 0, \theta2(t0) = 90\circ, \theta2(tf) = 0$$
$$\dot{\theta}1\_dot(t0) = \dot{\theta}1\_dot(tf) = \dot{\theta}2\_dot(t0) = \dot{\theta}2\_dot(tf) = 0$$

The trajectories:

```
Theta1 = (pi*t^3)/500 - (3*pi*t^2)/100 + pi
Theta2 = (pi*t^3)/1000 - (3*pi*t^2)/200 + pi/2
Theta1_dot = (3*pi*t^2)/500 - (3*pi*t)/50
Theta2_dot = (3*pi*t^2)/1000 - (3*pi*t)/100]
Theta1_ddot = (3*pi*t)/250 - (3*pi)/50
Theta2_ddot =  (3*pi*t)/500 - (3*pi)/100
```

b) The equations of motion in the Manipulator Equation Form :

```
M = [(m1*r1^2 + m2*r2^2 + 2*m2*cos(theta2)*r2*l1 + m2*l1^2 + I1 + I2), (m2*r2^2 +
l1*m2*cos(theta2)*r2 + I2); (m2*r2^2 + l1*m2*cos(theta2)*r2 + I2), (m2*r2^2 + I2)];

C = [-(2*r2*theta2_dot*l1*m2*sin(theta2)), -(r2*theta2_dot*l1*m2*sin(theta2));
   (r2*l1*m2*sin(theta2)*theta1_dot), 0];

G = [(- sin(theta1)*(r1*g*m1 + g*l1*m2) - r2*g*m2*sin(theta1 + theta2));
   (- r2*g*m2*sin(theta1 + theta2))];

Tau = M*q_ddot+C*q_dot+G;
```

Where ,

```
q = [theta1; theta2];

    q_dot = [theta1_dot; theta2_dot];
    q_ddot = [theta1_ddot; theta2_ddot];
Tau = [Tau1; Tau2];
```

c) The nominal mass and inertia values are given as :

$$\hat{m}1 = \hat{m}2 = 0.75 \text{ (kg)}, \hat{I}1 = \hat{I}2 = 0.063 \text{ (kg} \cdot \text{m2)}$$

Through feedback linearization of the control input term in the manipulator equation form we get,

$$\ddot{q} = v + \eta(q, \dot{q}, v),$$

We take the virtual control input as:

$$v = \ddot{q}_d - K_p e - K_d \dot{e} + v_r$$

We calculate the K gains by taking eigen values to be {−1, −1, −2, −2} using the '*place'* function in MATLAB:

```
k = place(A,B,lambda);
```

We obtain the k values as:

```
k = [ 2.0000    0    3.0000    0; 0    2.0000    0    3.0000];
```

where A and B matrices are computed from the linearized system as :

```
A = [0 0 1 0;    B = [0 0;
     0 0 0 1;         0 0;
     0 0 0 0;         1 0;
     0 0 0 0];        0 1];
```

The closed loop system A_{cl} is calculated as:

```
Acl = (A - B*k);
```

A_{cl} =

Using the equation:

$$A_{cl}^T * P + P * A_{cl} = Q$$

Where Q = [1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1]

We solve for P using *lyap* function:

```
P = lyap(Acl',Q);
```

P = [6.25 0 1.25 0; 0 6.25 0 1.25; 1.25 0 1.25 0; 0 1.25 0 1.25]

We use the positive definite P matrix to define our Lyapunov function candidate:

$$V(x) = x^T P x$$

We initialize the *rho* parameter.

```
rho = [10, 0;0, 10];
```

To ensure the stability of the system, the v_r term in the control input needs to be implemented as

$V_r = - rho * (B^T * P * X) / norm(B^T * P * X)$ if $X \neq 0$

$V_r = 0$ if $X = 0$

Then the overall control law with

$$V = \ddot{q}_d - K_p e - K_d \dot{e} + v_r$$

Where K = [Kp Kd] is a 2 x 4 matrix

We get the overall control law as:

$$Tau = \hat{M}(q)v + \hat{C}(q, \dot{q})\dot{q} + \hat{g}(q)$$

With the nominal dynamic parameters $\hat{M}(q)$, $\hat{C}(q, \dot{q})$, $\hat{g}(q)$.

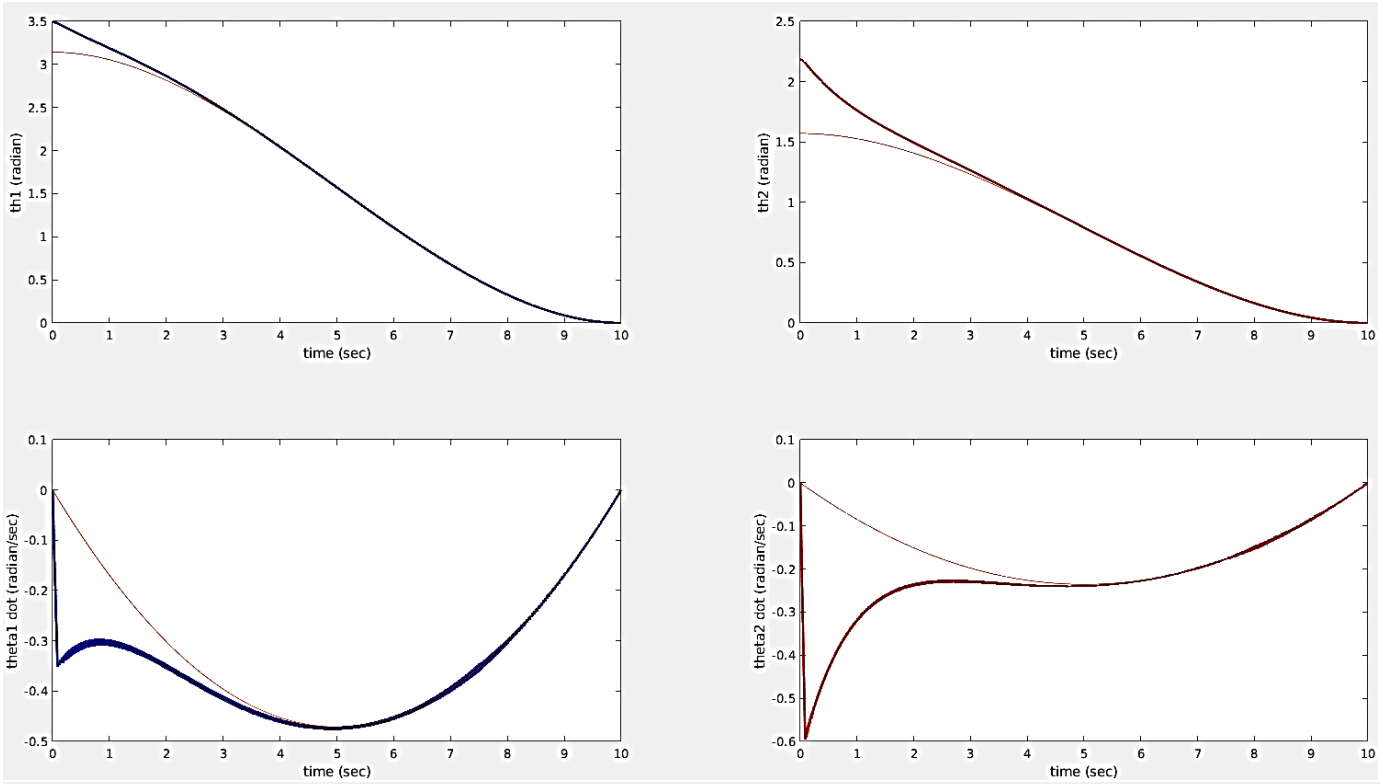
d) The ode function is defined with the cubic polynomial trajectories setting as our desired trajectories, the error matrix and the control law evaluated above for calculating the control inputs to the system. We use the Tau calculated using the robust control law through the nominal values to compute the X_dot state vector using the actual Manipulator Equation Form.

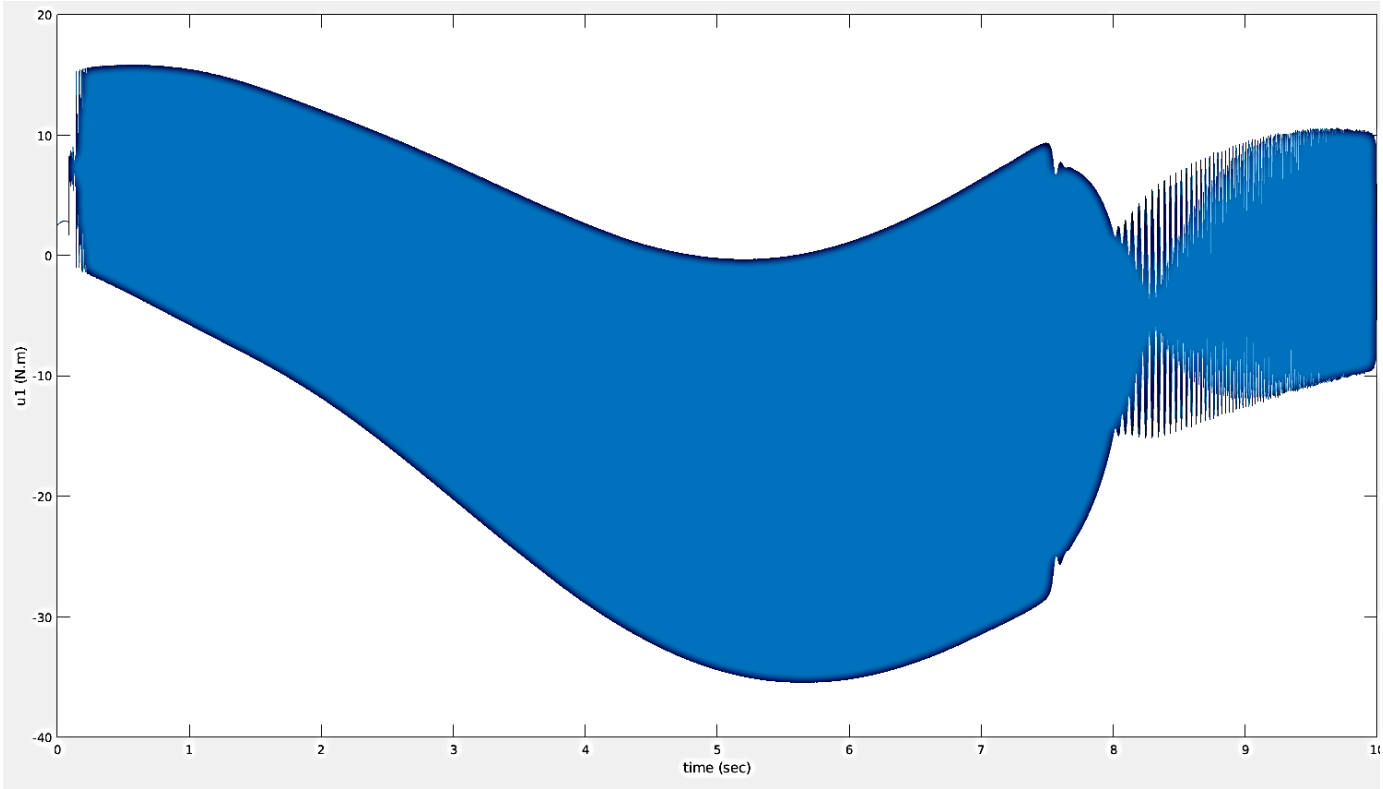
```
dX(1) = theta1_dot;

dX(2) = theta2_dot;
```

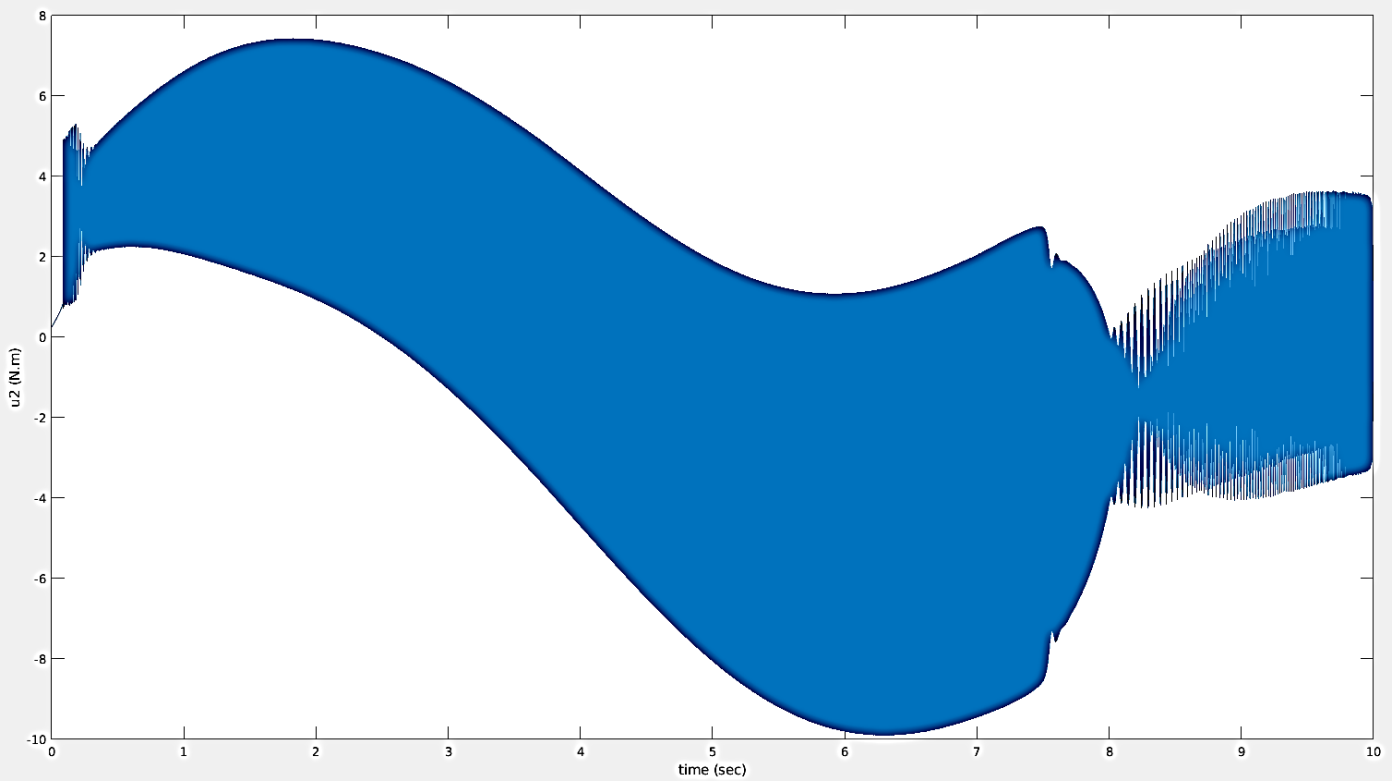
```
dX(3) = (I2*Tau1 - I2*Tau2 + Tau1*r2^2*m2 - Tau2*r2^2*m2 +
r2^3*theta1_dot^2*l1*m2^2*sin(theta2) + r2^3*theta2_dot^2*l1*m2^2*sin(theta2) +
r2^2*g*l1*m2^2*sin(theta1) + I2*r1*g*m1*sin(theta1) - Tau2*r2*l1*m2*cos(theta2) +
I2*g*l1*m2*sin(theta1) + 2*r2^3*theta1_dot*theta2_dot*l1*m2^2*sin(theta2) +
r2^2*theta1_dot^2*l1^2*m2^2*cos(theta2)*sin(theta2) - r2^2*g*l1*m2^2*sin(theta1 +
theta2)*cos(theta2) + I2*r2*theta1_dot^2*l1*m2*sin(theta2) +
I2*r2*theta2_dot^2*l1*m2*sin(theta2) + r1*r2^2*g*m1*m2*sin(theta1) +
2*I2*r2*theta1_dot*theta2_dot*l1*m2*sin(theta2))/(I1*I2 + r2^2*l1^2*m2^2 + I2*r1^2*m1 +
I1*r2^2*m2 + I2*l1^2*m2 + r1^2*r2^2*m1*m2 - r2^2*l1^2*m2^2*cos(theta2)^2);

dX(4) = -(I2*Tau1 - I1*Tau2 - I2*Tau2 - Tau2*r1^2*m1 + Tau1*r2^2*m2 - Tau2*r2^2*m2 -
Tau2*l1^2*m2 + r2*theta1_dot^2*l1^3*m2^2*sin(theta2) +
r2^3*theta1_dot^2*l1*m2^2*sin(theta2) + r2^3*theta2_dot^2*l1*m2^2*sin(theta2) -
r2*g*l1^2*m2^2*sin(theta1 + theta2) - I1*r2*g*m2*sin(theta1 + theta2) +
r2^2*g*l1*m2^2*sin(theta1) + I2*r1*g*m1*sin(theta1) + Tau1*r2*l1*m2*cos(theta2) -
2*Tau2*r2*l1*m2*cos(theta2) + I2*g*l1*m2*sin(theta1) +
2*r2^3*theta1_dot*theta2_dot*l1*m2^2*sin(theta2) +
2*r2^2*theta1_dot^2*l1^2*m2^2*cos(theta2)*sin(theta2) +
r2^2*theta2_dot^2*l1^2*m2^2*cos(theta2)*sin(theta2) - r2^2*g*l1*m2^2*sin(theta1 +
theta2)*cos(theta2) + r2*g*l1^2*m2^2*cos(theta2)*sin(theta1) - r1^2*r2*g*m1*m2*sin(theta1 +
theta2) + I1*r2*theta1_dot^2*l1*m2*sin(theta2) + I2*r2*theta1_dot^2*l1*m2*sin(theta2) +
I2*r2*theta2_dot^2*l1*m2*sin(theta2) + r1*r2^2*g*m1*m2*sin(theta1) +
2*r2^2*theta1_dot*theta2_dot*l1^2*m2^2*cos(theta2)*sin(theta2) +
r1^2*r2*theta1_dot^2*l1*m1*m2*sin(theta2) + 2*I2*r2*theta1_dot*theta2_dot*l1*m2*sin(theta2)
+ r1*r2*g*l1*m1*m2*cos(theta2)*sin(theta1))/(I1*I2 + r2^2*l1^2*m2^2 + I2*r1^2*m1 +
I1*r2^2*m2 + I2*l1^2*m2 + r1^2*r2^2*m1*m2 - r2^2*l1^2*m2^2*cos(theta2)^2);
```

We send the output vector of the ode function as [dX, Tau] which consists of the X_dot state vector and the control input Tau vector.

e) The simulation results under the Robust Control implementation are as follows:

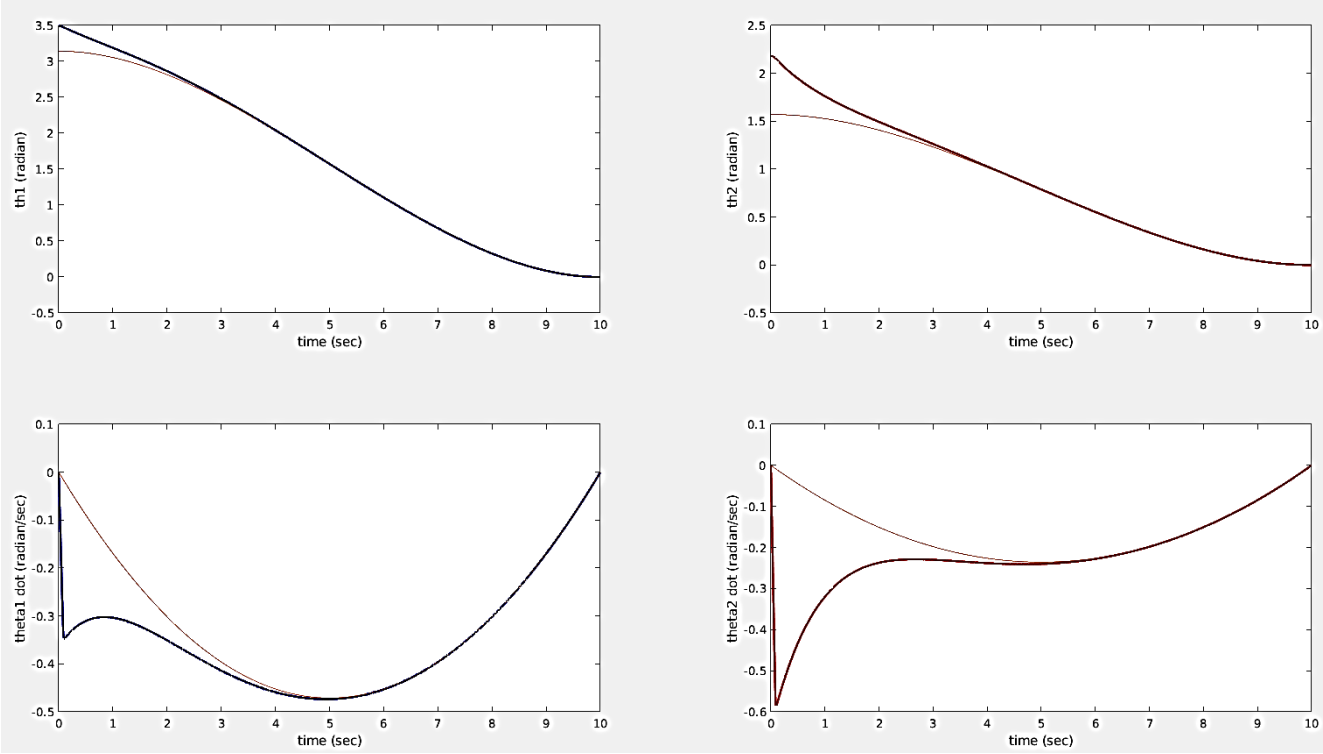


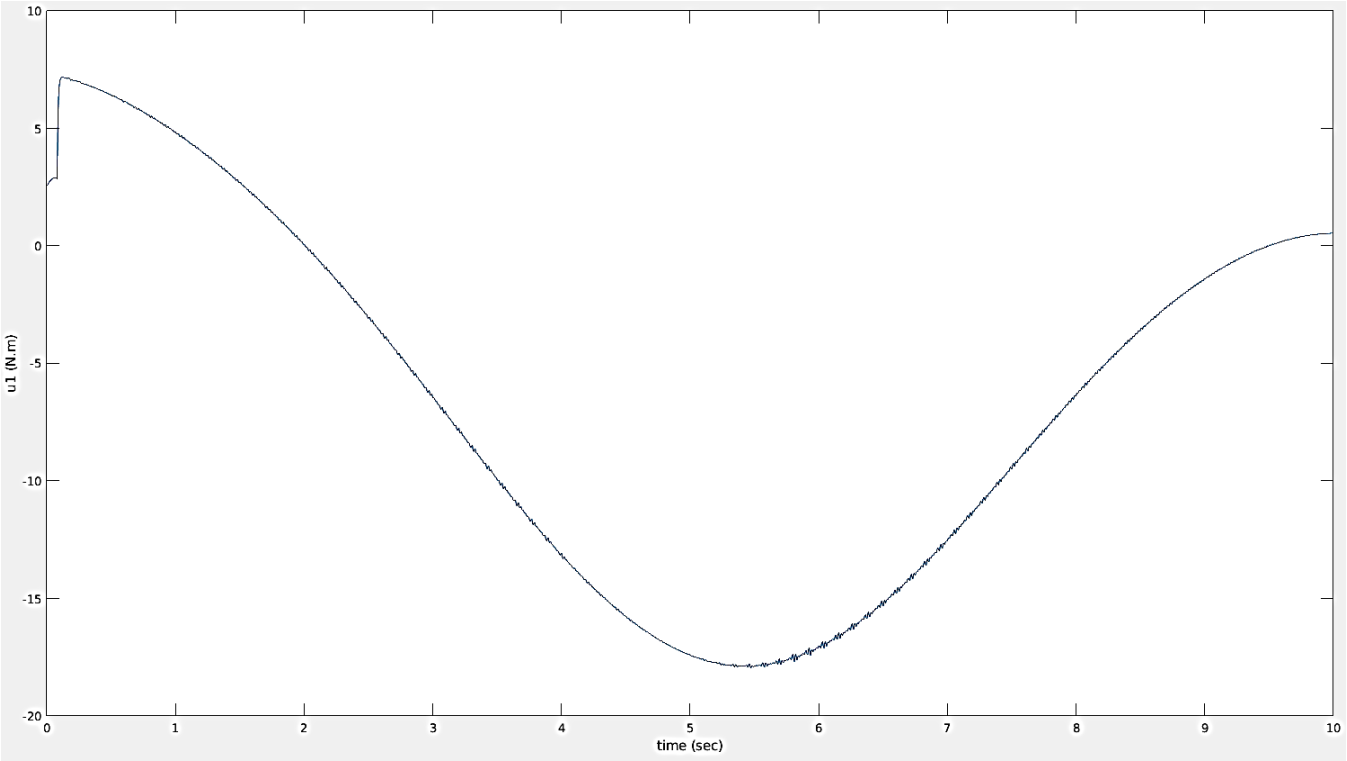


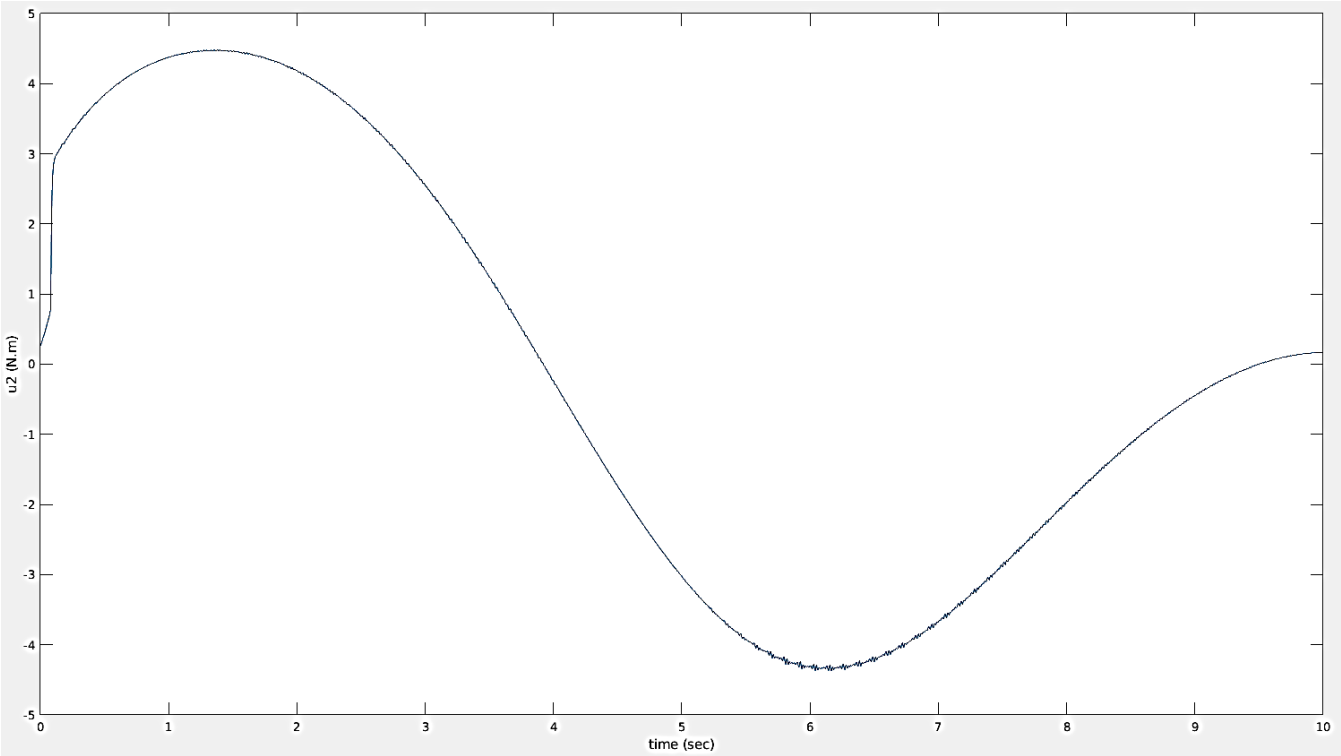


We observe the model trajectories converging to the desired trajectories.
But the control inputs have a lot of chattering due to the 'V_r' term in our controller which tends to fluctuate a lot at the stabilization point.

f) We implement a boundary layer in the robust control term v_r term to reduce chattering.
The *phi* is enforced as follows in the ode function when implementing the control law:
phi = 0.06;
We obtain the following results:

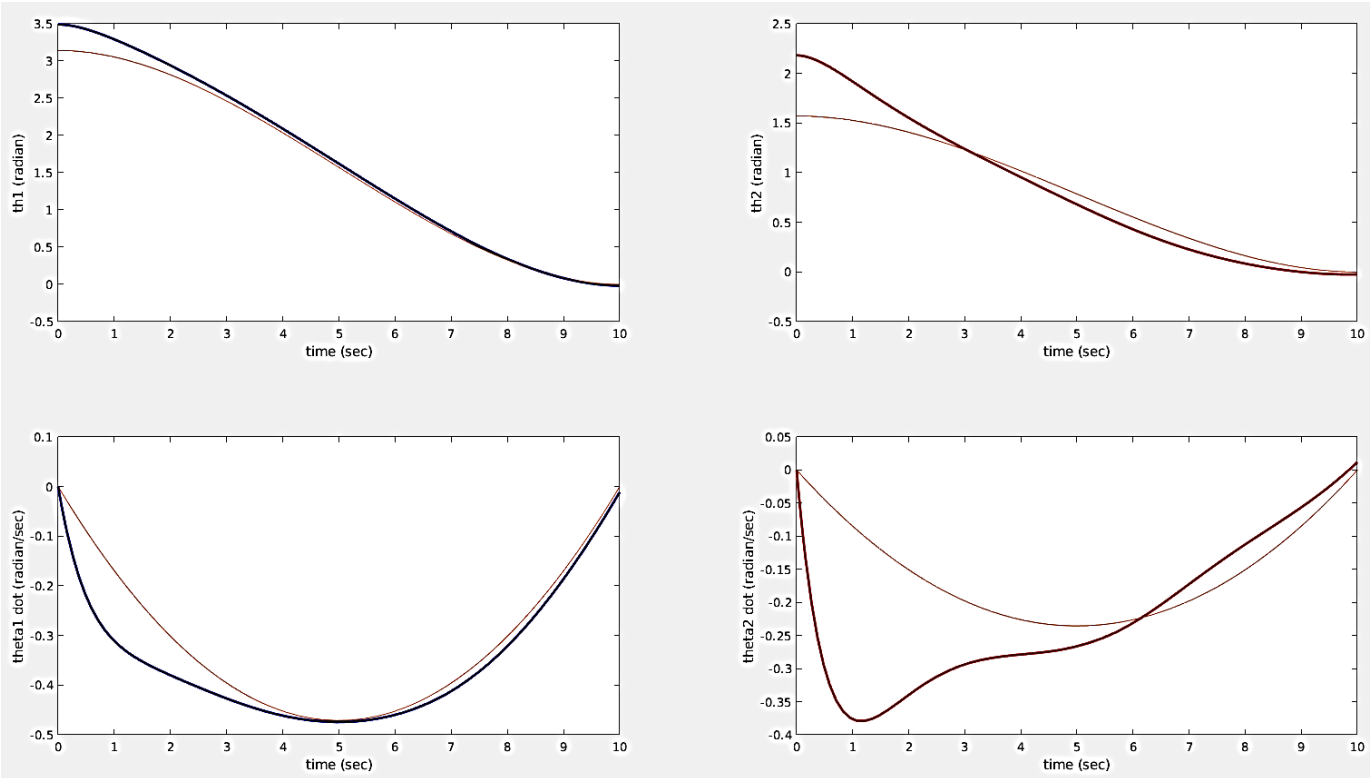

th1 (radian)
time (sec)
th2 (radian)
time (sec)
theta1 dot (radian/sec)
time (sec)
theta2 dot (radian/sec)
time (sec)

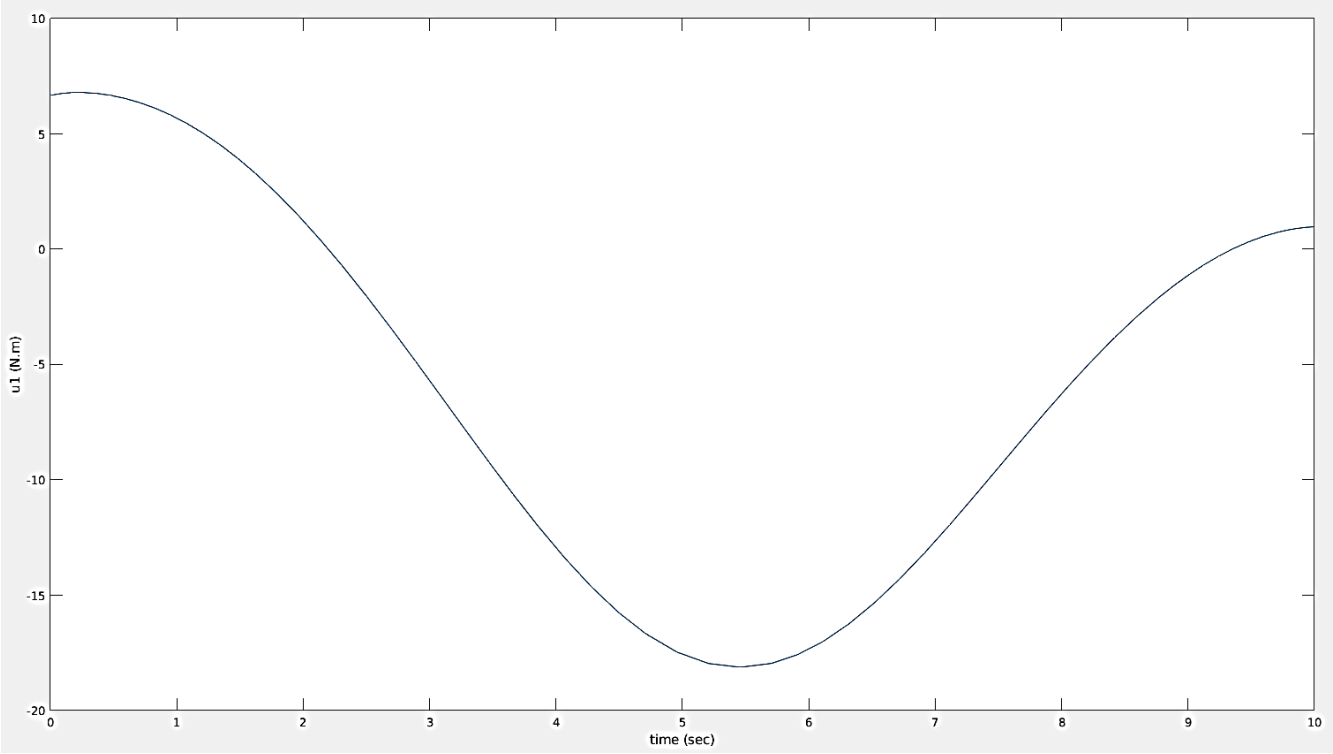

u1 (N.m)
time (sec)

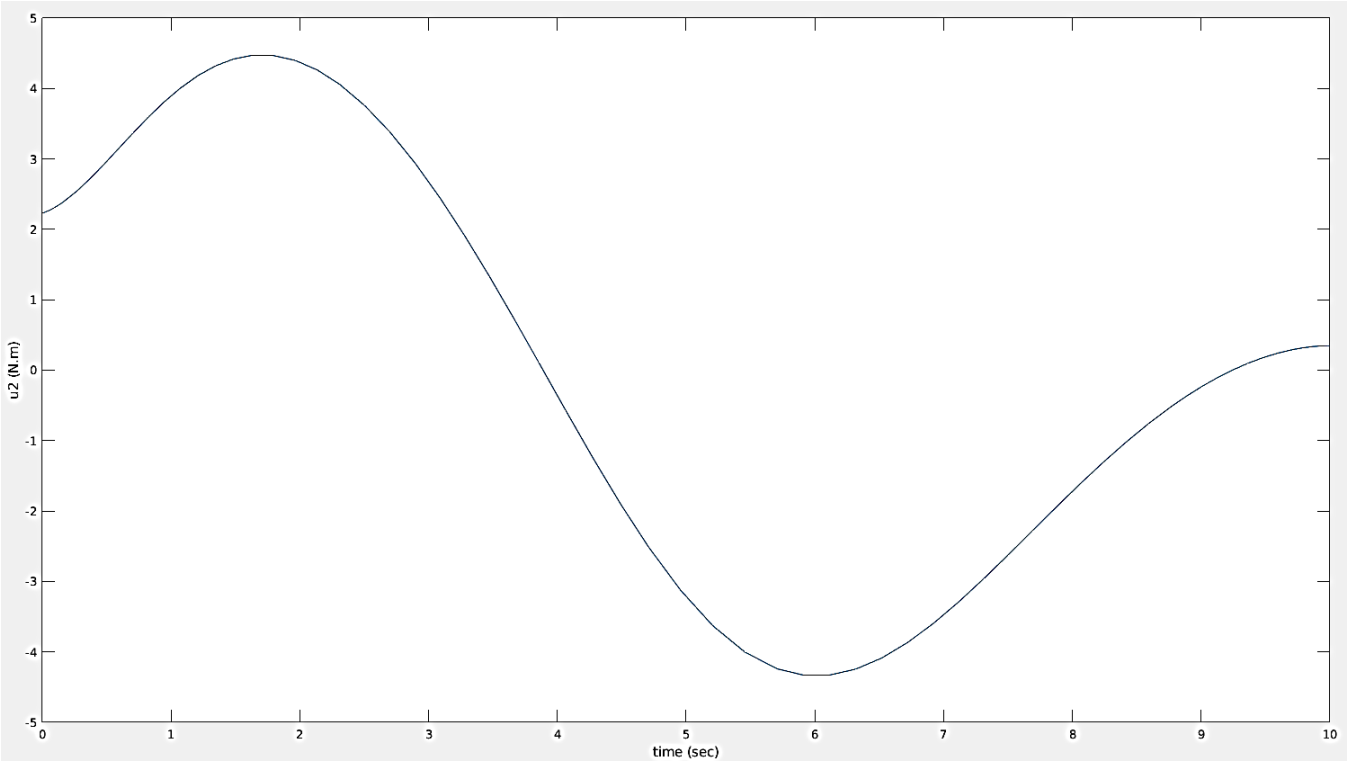


We observe that the trajectories converge as well as the chattering in the control input has reduced to give relatively smoother curves.

g) Without the robust control law implementation :

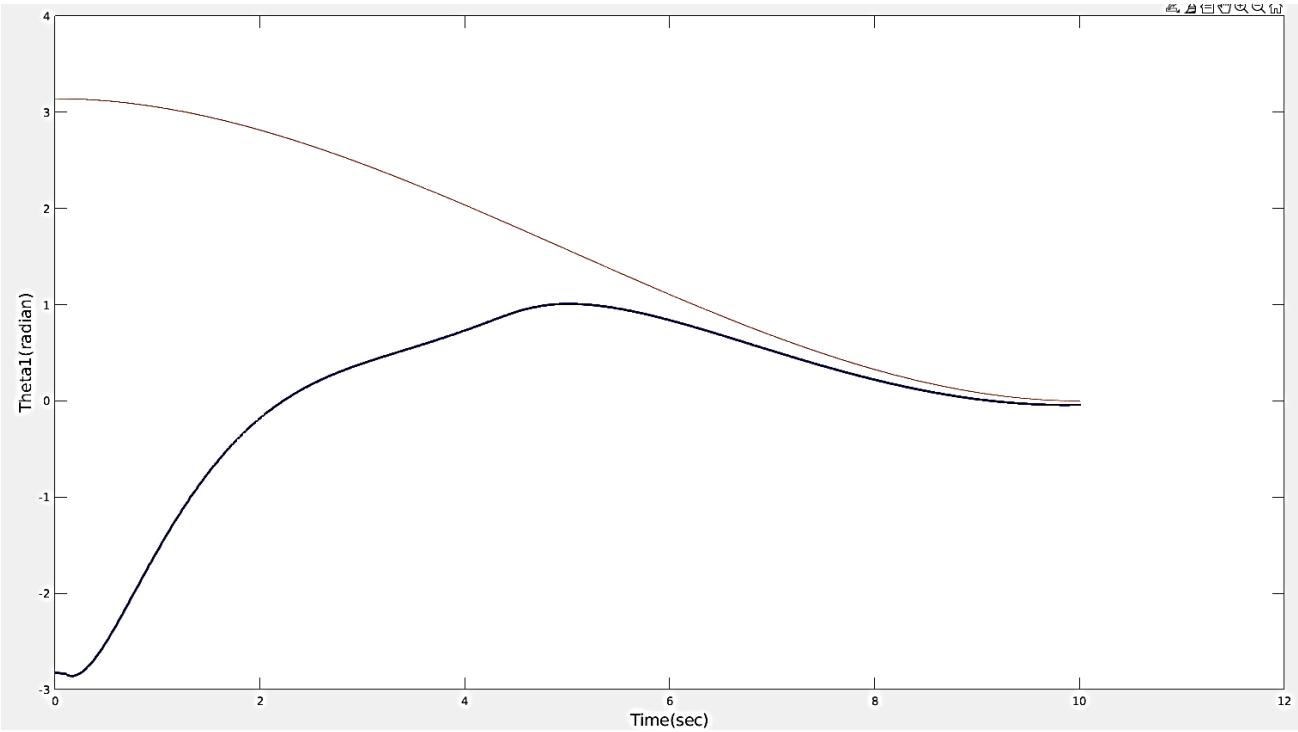


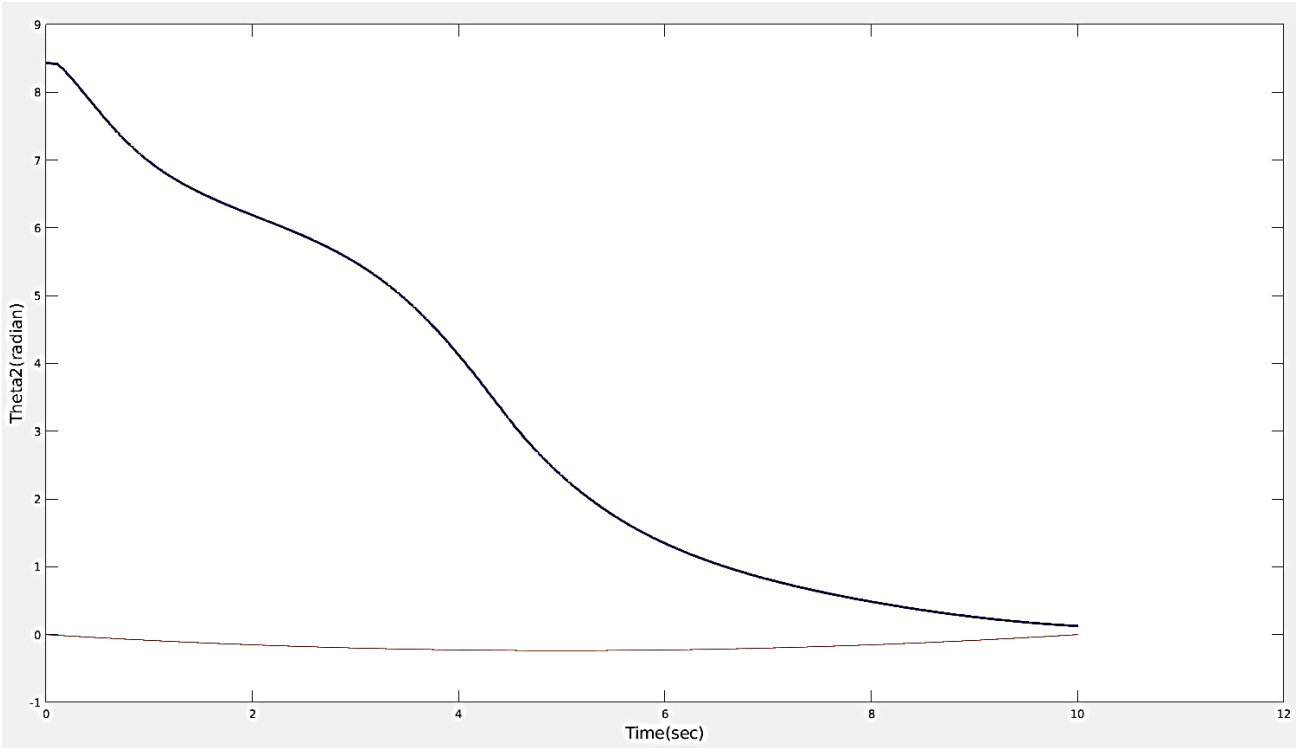


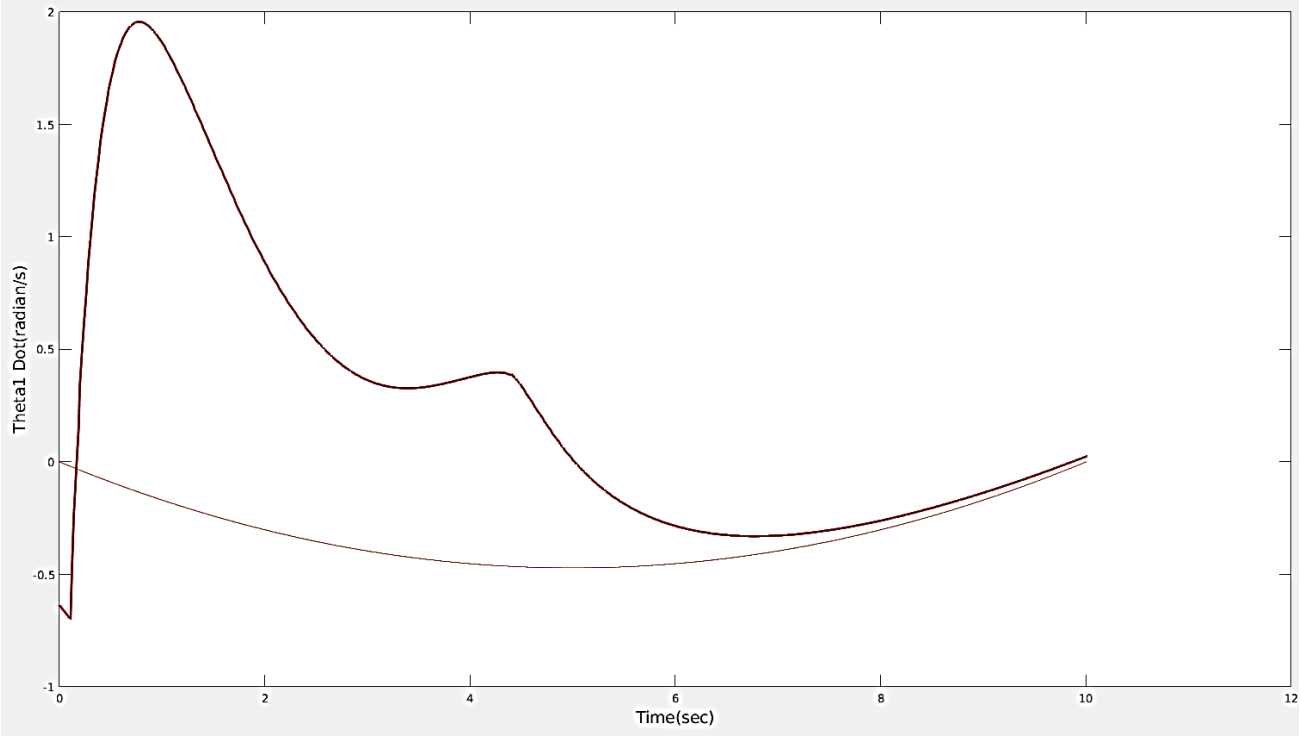


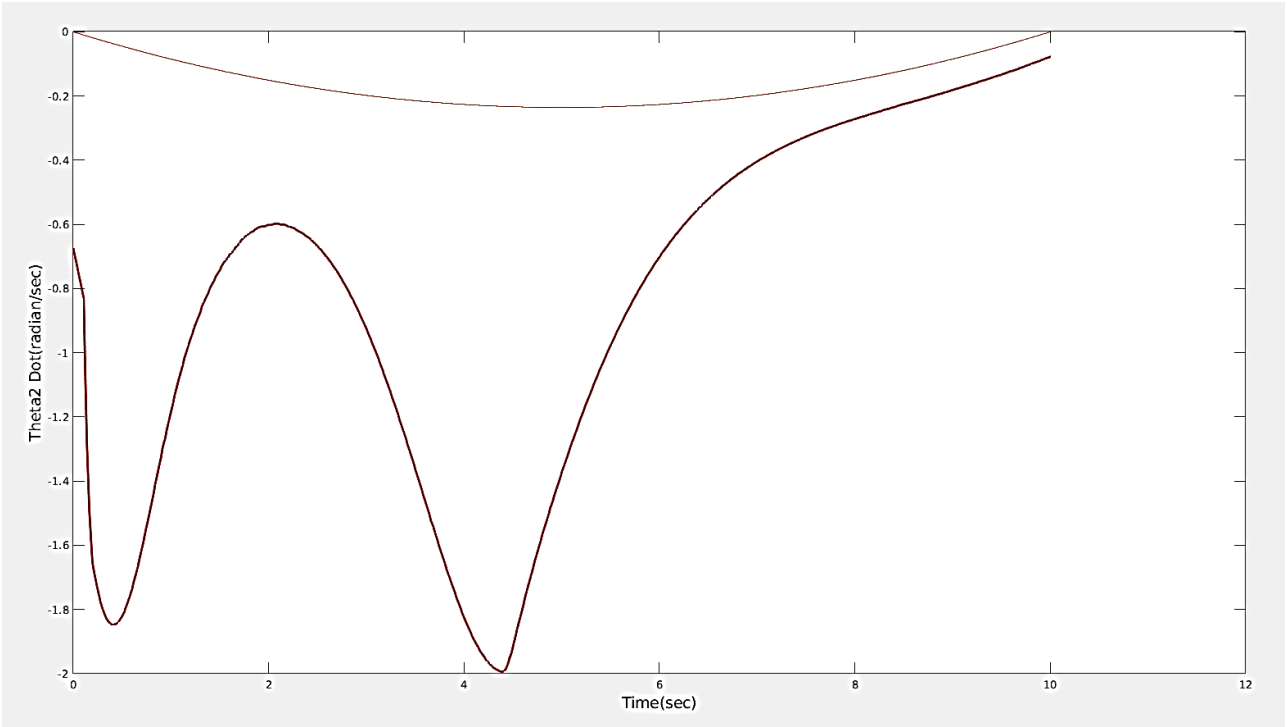
We observe that the trajectories don't converge as well as they do with the robust control law. But the control inputs are smooth.

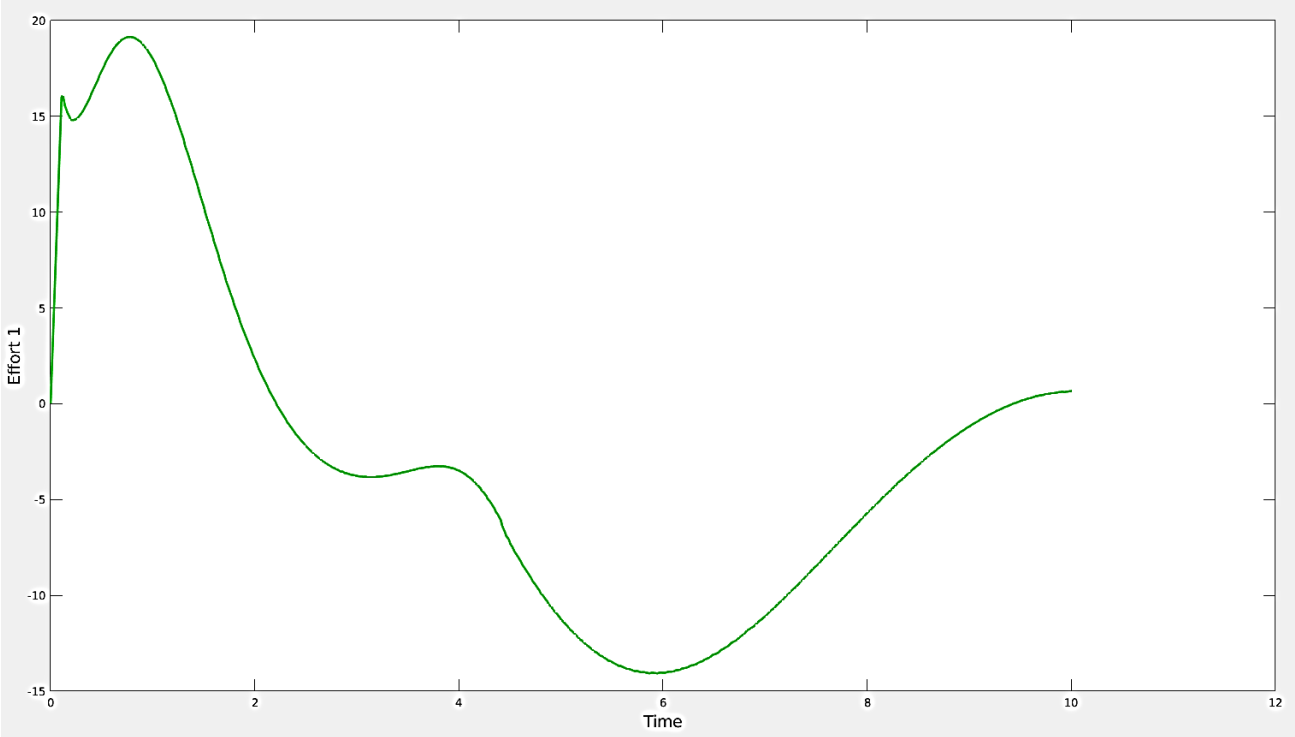
h) We implement the control law for controlling the RRBot in Gazebo to converge it to the upward equilibrium point. By tuning the parameters we obtain satisfactory results as follows:
phi = 0.9;
rho = [10, 0;0, 10];
k = [0.5000 0 2.5000 0; 0 0.5000 0 2.5000];

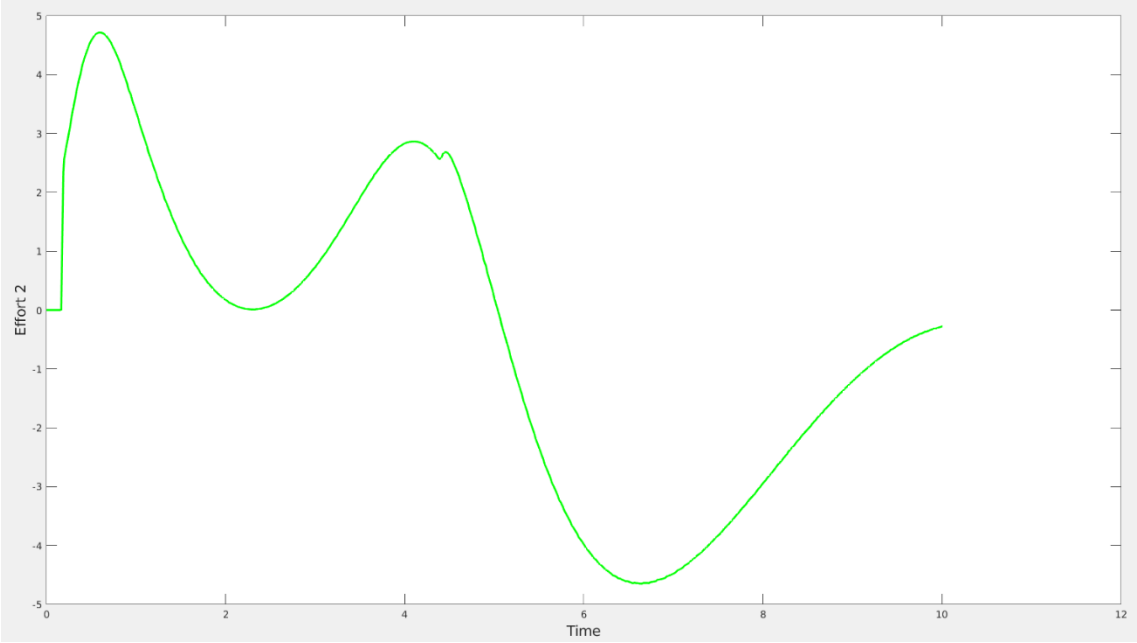

Theta1(radian)
Time(sec)
4
3
2
1
0
-1
-2
-3
0
2
4
6
8
10
12


Theta2(radian)
Time(sec)
9
8
7
6
5
4
3
2
1
0
-1
0
2
4
6
8
10
12


Theta1 Dot(radian/s)
Time(sec)
2
1.5
1
0.5
0
-0.5
-1
0
2
4
6
8
10
12


Theta2 Dot(radian/sec)
Time(sec)
0
-0.2
-0.4
-0.6
-0.8
-1
-1.2
-1.4
-1.6
-1.8
-2
0
2
4
6
8
10
12





We observe that the trajectories converge as well as the control inputs stay within the bounds mentioned in the problem statement.