Initial Setup: 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 A: Linear Parametric form of Manipulator Equations:

Equation of Motion in **Manipulator** Form:

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 **Linear Parametric** Form:

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

L1*M2*r2*(sin(theta2)*theta2_dot^2 + 2*theta1_dot*sin(theta2)*theta2_dot - cos(theta2)*(2*theta1_ddot + theta2_ddot))

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

After comparing term by term the equations in Manipulator Form and Linear Parametric Form, we can conclude that both the equations are indeed equivalent.

Problem B: Adaptive Inverse Dynamics:

Control Law 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).*(0.9);
P = lyap(Acl',Q);
U = Mmat_hat.*V + Cmat_hat + Gmat_hat;
```

Adaptation Law Design:

```
Gamma = eye(5).*50;
Y = [theta1_ddot, ...
```

```
      cos(theta2)*(2*theta1_ddot + theta2_ddot) -
2*sin(theta2)*theta1_dot*theta2_dot - sin(theta2)*theta2_dot^2, ...
      theta2_ddot, ...
      -sin(theta1)*g, ...
      -sin(theta1 + theta2)*g; ...
      0, ...
      sin(theta2)*theta1_dot^2 + cos(theta2)*theta1_ddot, ...
      theta1_ddot + theta2_ddot, ...
      0, ...
      -sin(theta1+theta2)*g];

Phi = Mmat_hat\Y;
alpha_tilda_dot = -(Gamma\(Phi'*B'*P*G));
where G = [e; e_dot];
```

List of all the Parameters used:

```
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
```

```
Acl =

         0         0    1.0000         0
         0         0         0    1.0000
  -12.0000         0   -7.0000         0
         0  -12.0000         0   -7.0000


Q =

    0.9000         0         0         0
         0    0.9000         0         0
         0         0    0.9000         0
         0         0         0    0.9000


P =

    1.0982         0    0.0375         0
         0    1.0982         0    0.0375
    0.0375         0    0.0696         0
         0    0.0375         0    0.0696
```

```
Gamma =

    50     0     0     0     0
     0    50     0     0     0
     0     0    50     0     0
     0     0     0    50     0
     0     0     0     0    50
```

Problem C: ODE update Feedback Linearization Control

Control Input Design:

```
V = feed_foward_input - Kp*e - Kd*e_dot;

Mmat_hat = [a1 + 2*a2*cos(theta2), a3 + a2*cos(theta2);a3 +
a2*cos(theta2),                  a3];

Cmat_hat = [-a2*theta2_dot*sin(theta2)*(2*theta1_dot +
theta2_dot);a2*theta1_dot^2*sin(theta2)];

Gmat_hat = [- a4*g*sin(theta1) - a5*g*sin(theta1 + theta2); -
a5*g*sin(theta1 + theta2)];

U = Mmat_hat.*V + Cmat_hat + Gmat_hat;

tau1 = U(1);

tau2 = U(2);
```

Adaptation Law Design:

```
G = [e; e_dot];

Phi = Mmat_hat\Y;

alpha_tilda = -(Gamma\(Phi'*B'*P*G));
```

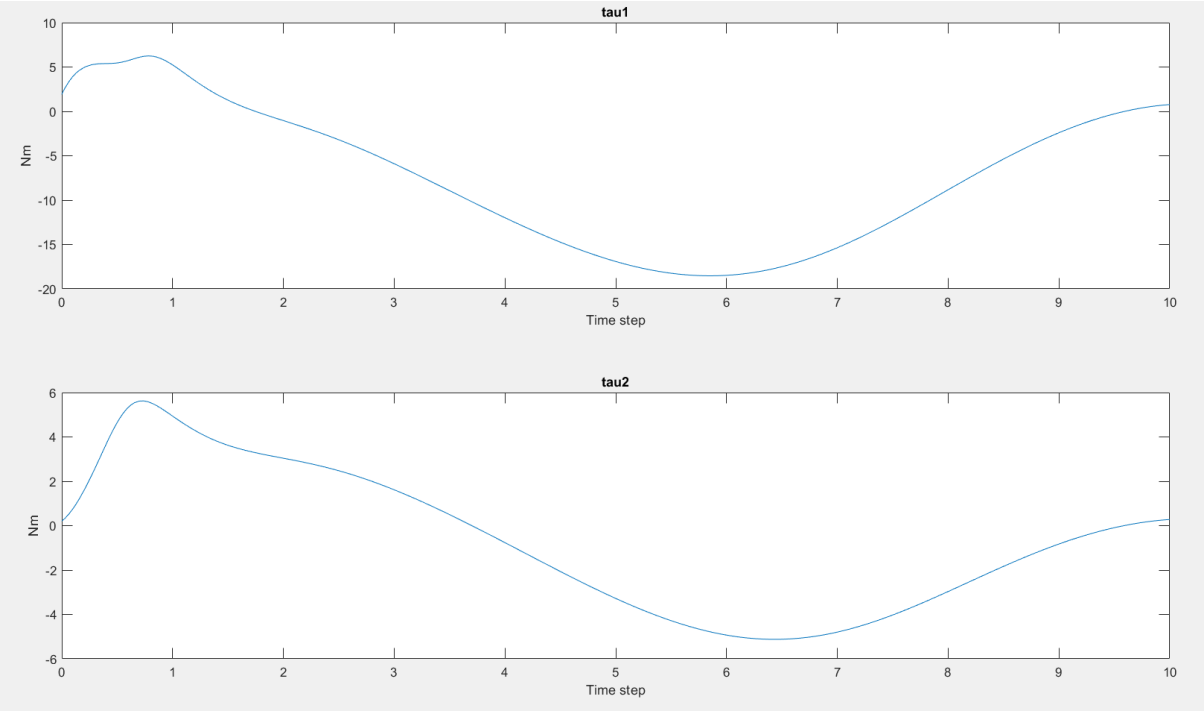
System Design:

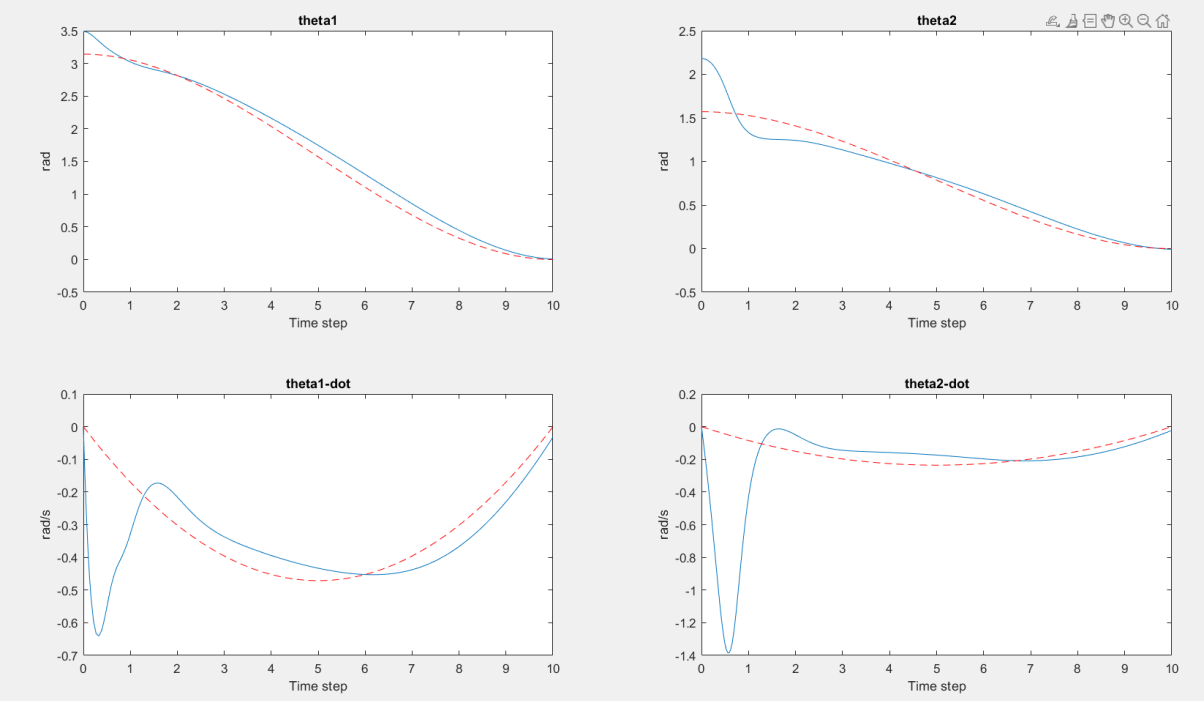
```
dX(1) = theta1_dot;

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);
```

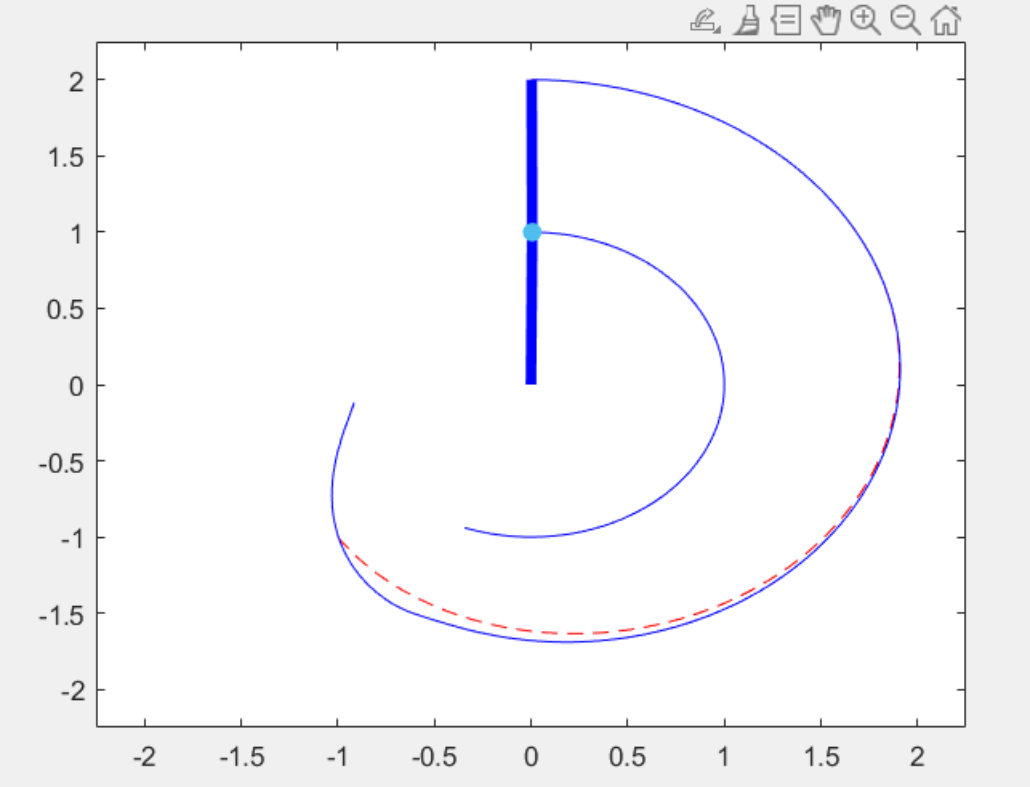
```
dX(4) = -(I2*tau1 - I1*tau2 - I2*tau2 - L1^2*M2*tau2 - M1*r1^2*tau2
+ M2*r2^2*tau1 - M2*r2^2*tau2 - L1^2*M2^2*g*r2*sin(theta1 + theta2)
+ L1*M2^2*g*r2^2*sin(theta1) - I1*M2*g*r2*sin(theta1 + theta2) +
I2*L1*M2*g*sin(theta1) + I2*M1*g*r1*sin(theta1) +
L1*M2*r2*tau1*cos(theta2) - 2*L1*M2*r2*tau2*cos(theta2)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

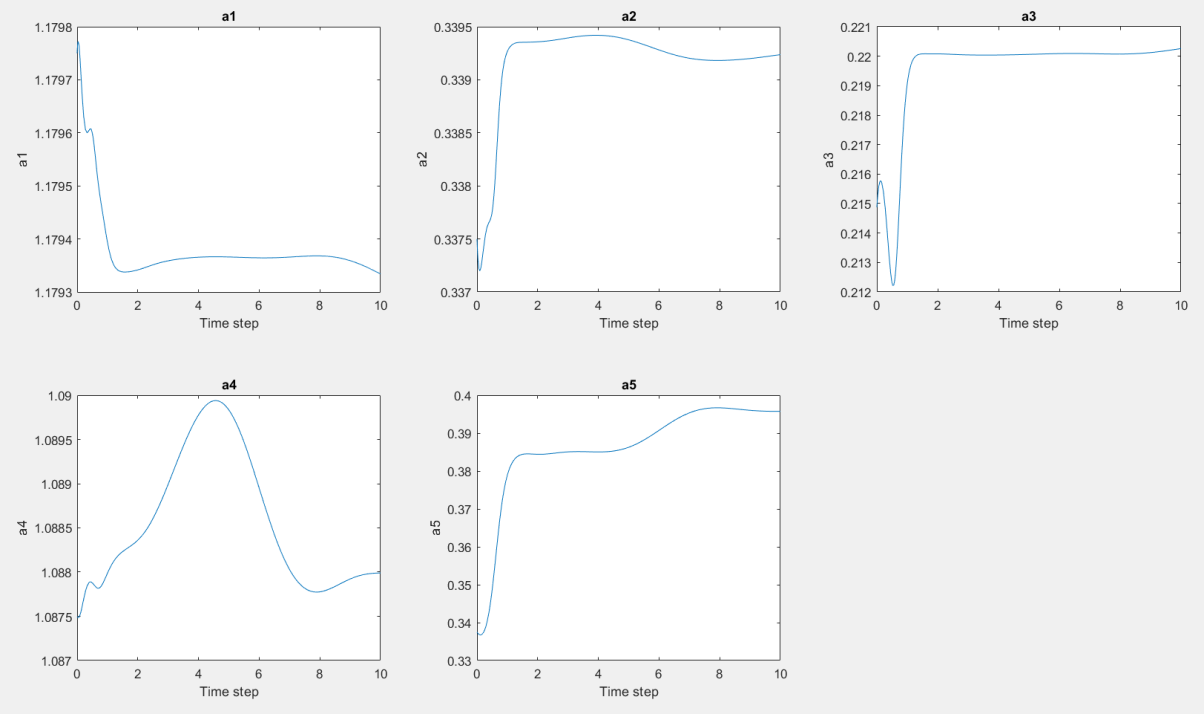
Problem D: Matlab Simulation-Adaptive Control

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

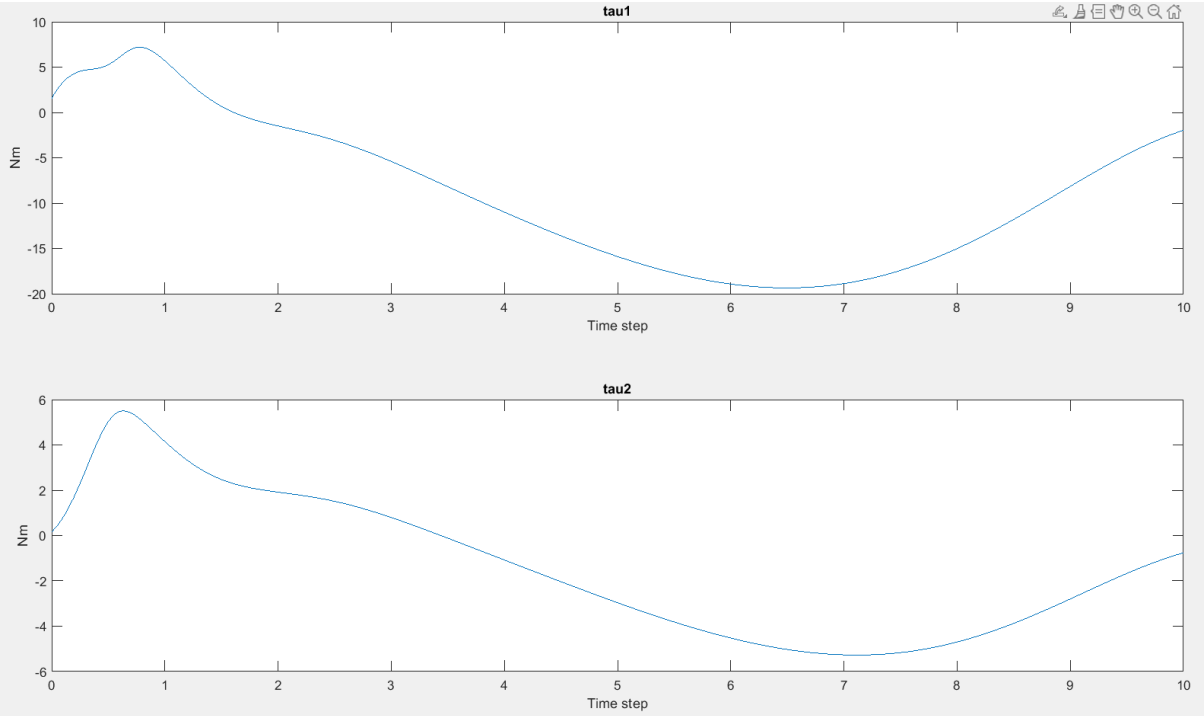

a1
a1
Time step
a2
a2
Time step
a3
a3
Time step
a4
a4
Time step
a5
a5
Time step

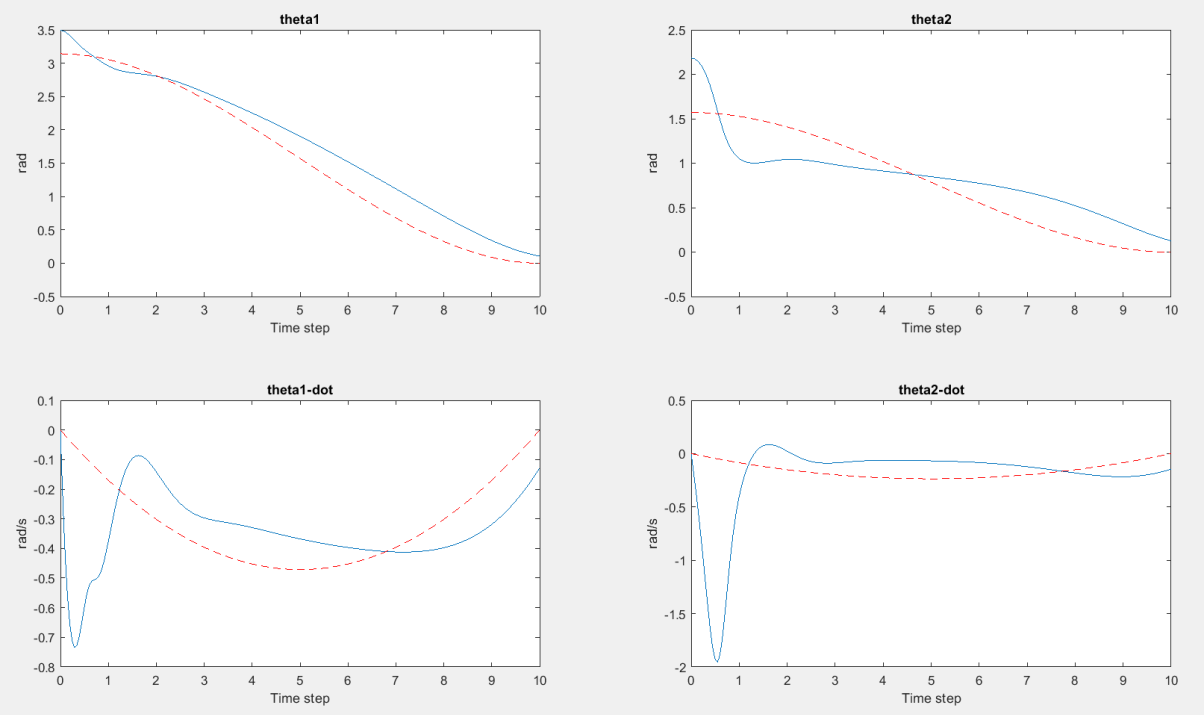
```
aplha_desired =

    1.5730
    0.4500
    0.2865
    1.4500
    0.4500
```

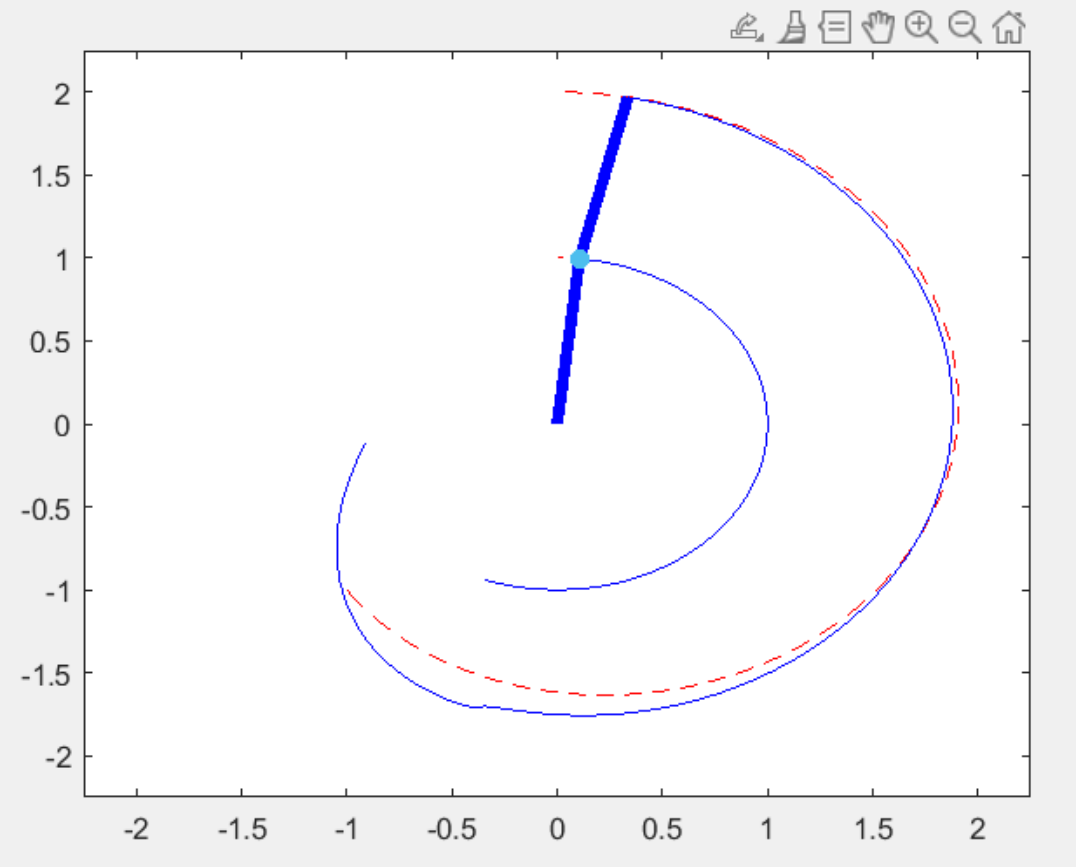
Here we can observe that the desired alpha values are as shown on the left. After studying the graph showing variations in the value of alpha over time we can see that the alpha values converge to some values but however, these values are not the desired values and hence we can conclude that the alpha values do converge but not the desired values.

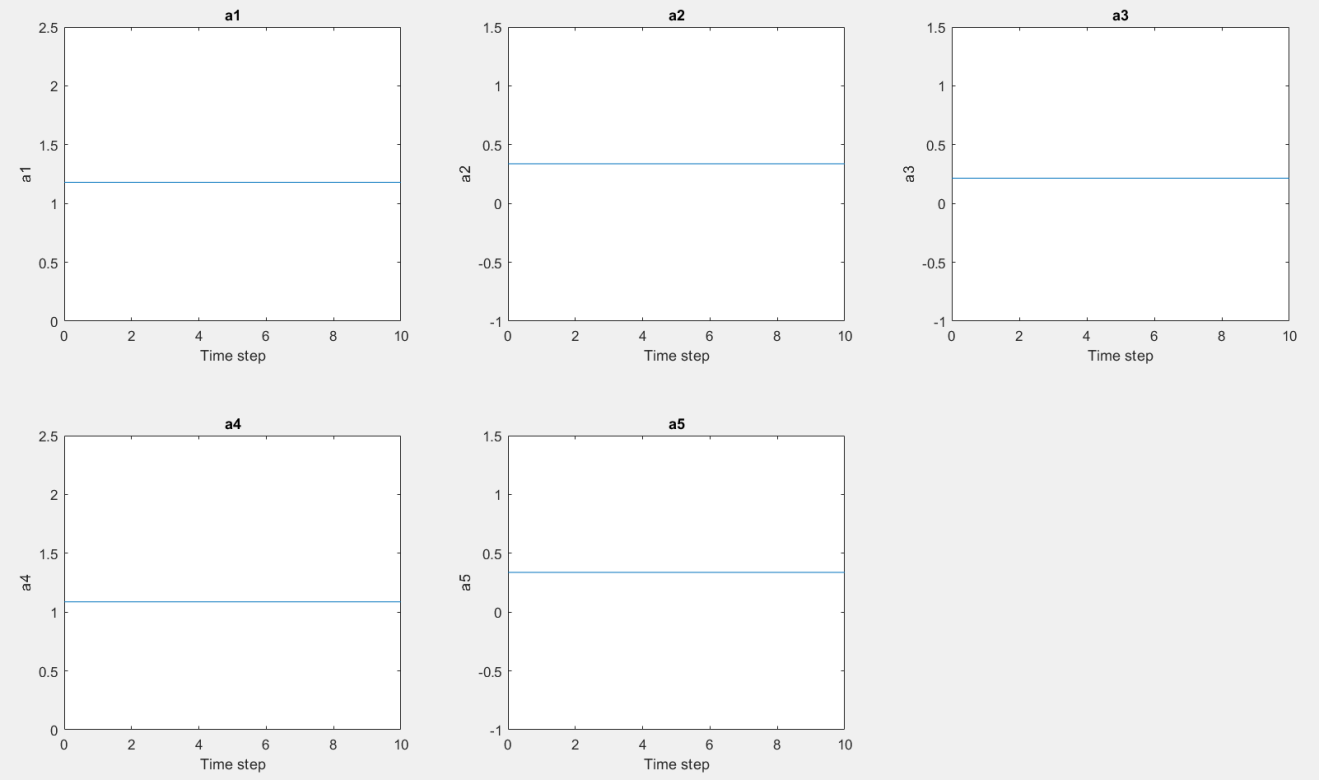
Problem E: Matlab Simulation- Without Adaptive Control

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.



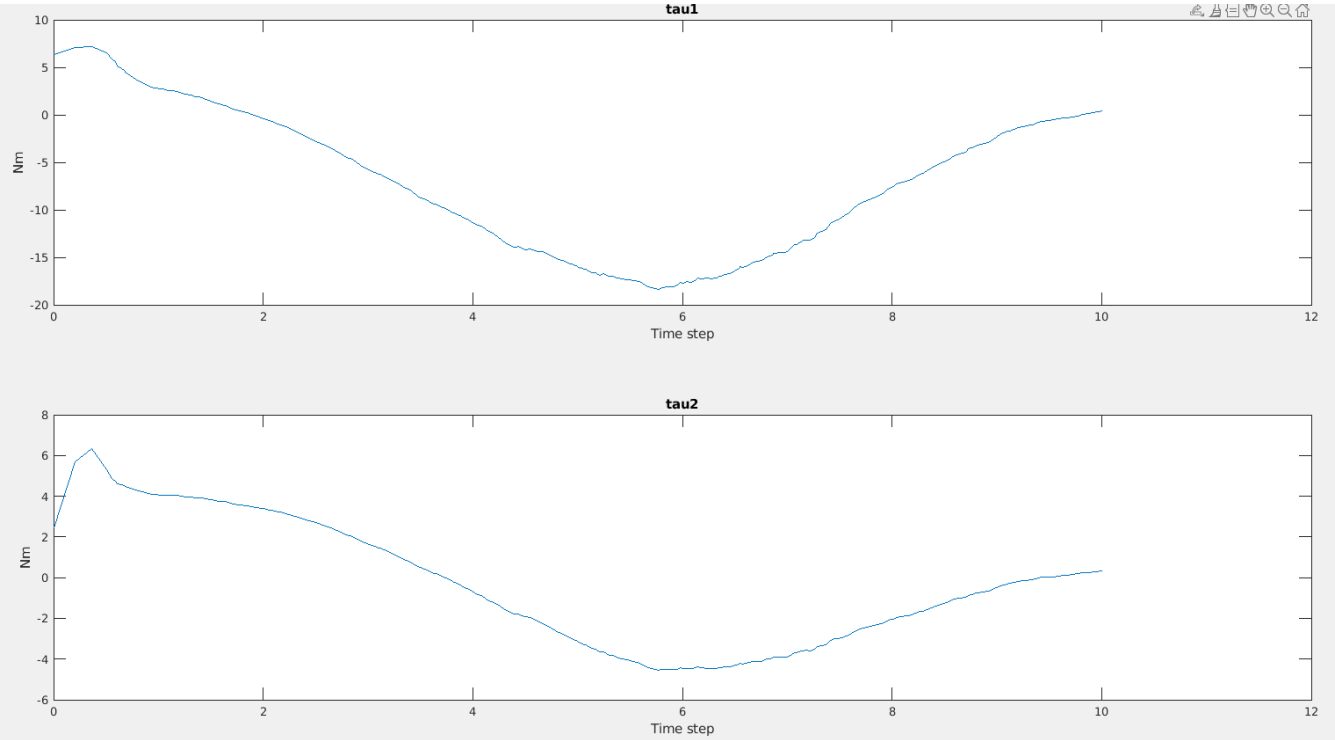


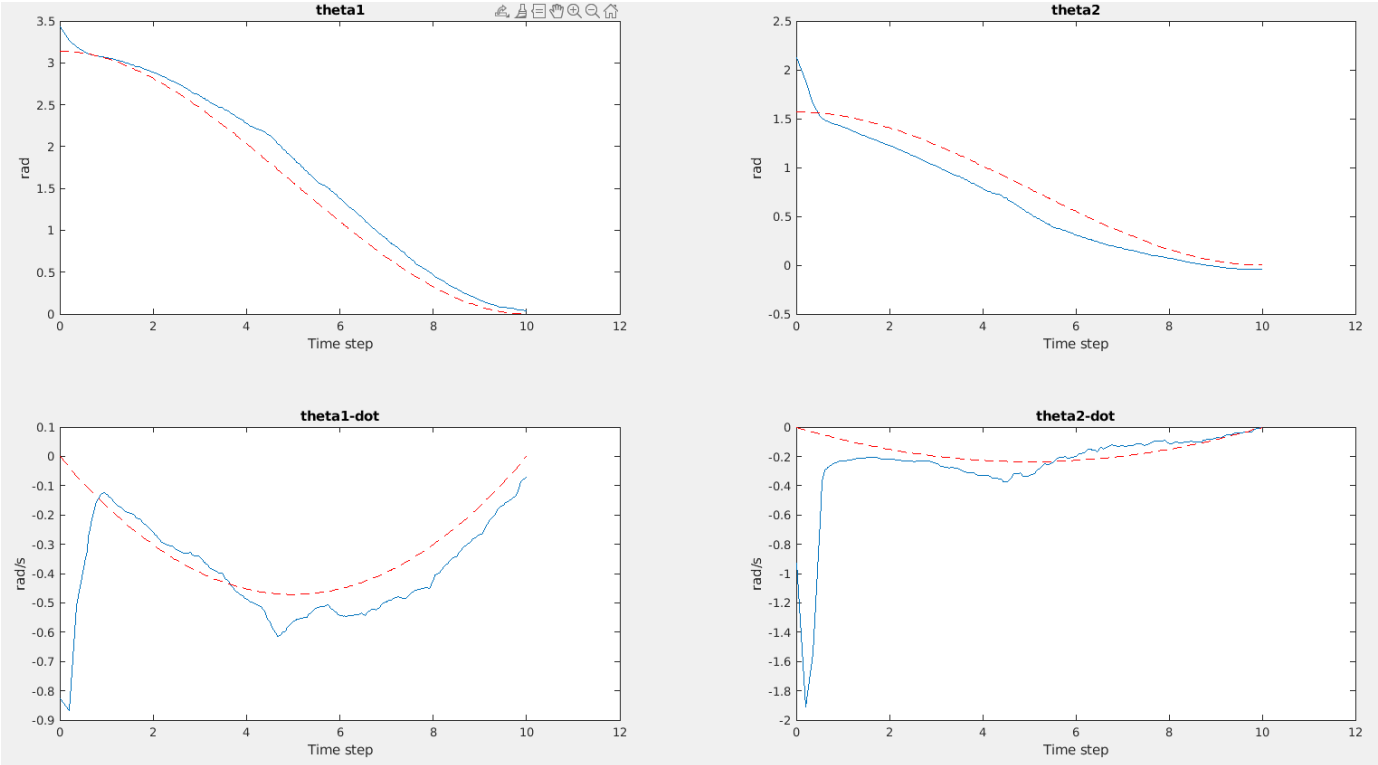

2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2 -1.5 -1 -0.5 0 0.5 1 1.5 2

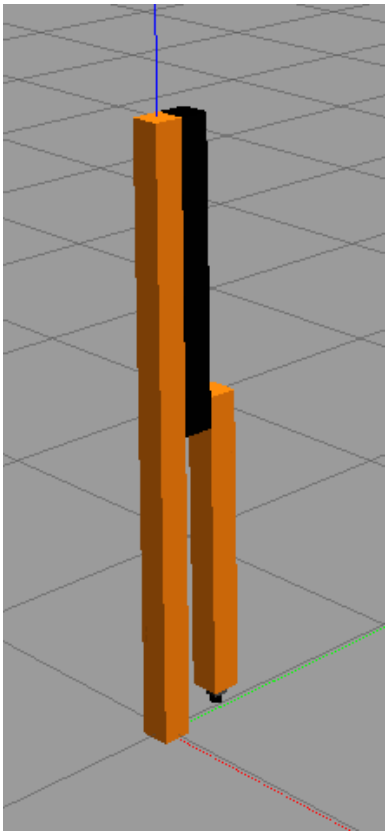

a1
a1
Time step
a2
a2
Time step
a3
a3
Time step
a4
a4
Time step
a5
a5
Time step

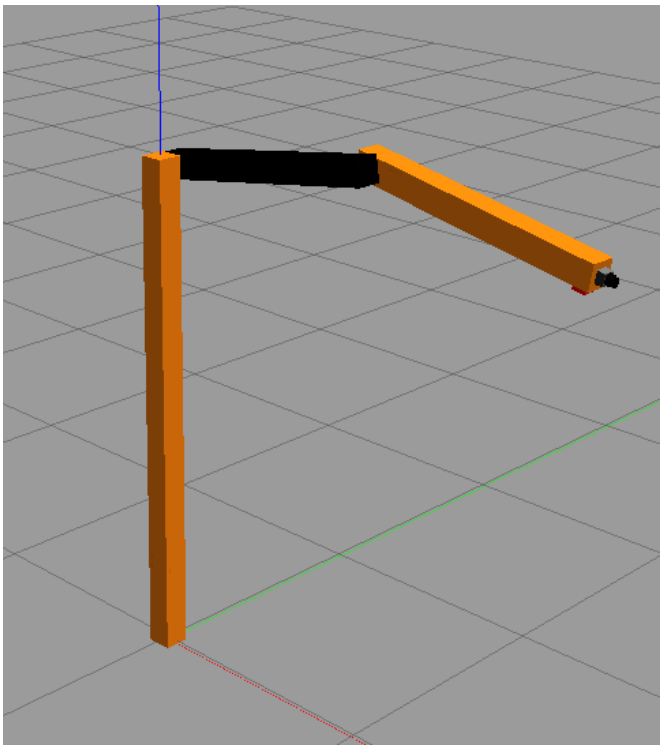
Here we can observe from our findings that the control input in both the cases (i.e, adaptive and non-adaptive) are almost similar however, the one in case of non-adaptive is slightly higher. In case of non-adaptive control the state variables vary differently and does not converge to the trajectory within 10 seconds. We can observe that the trajectory is not properly followed in case of the non-adaptive control, where in case of adaptive control the robot closely follows the trajectory. Since, it is non-adaptive control the alpha values does not change for this case. We can conclude that under such circumstances where disparity is high in the knowledge of the system model, adaptive control performs better.

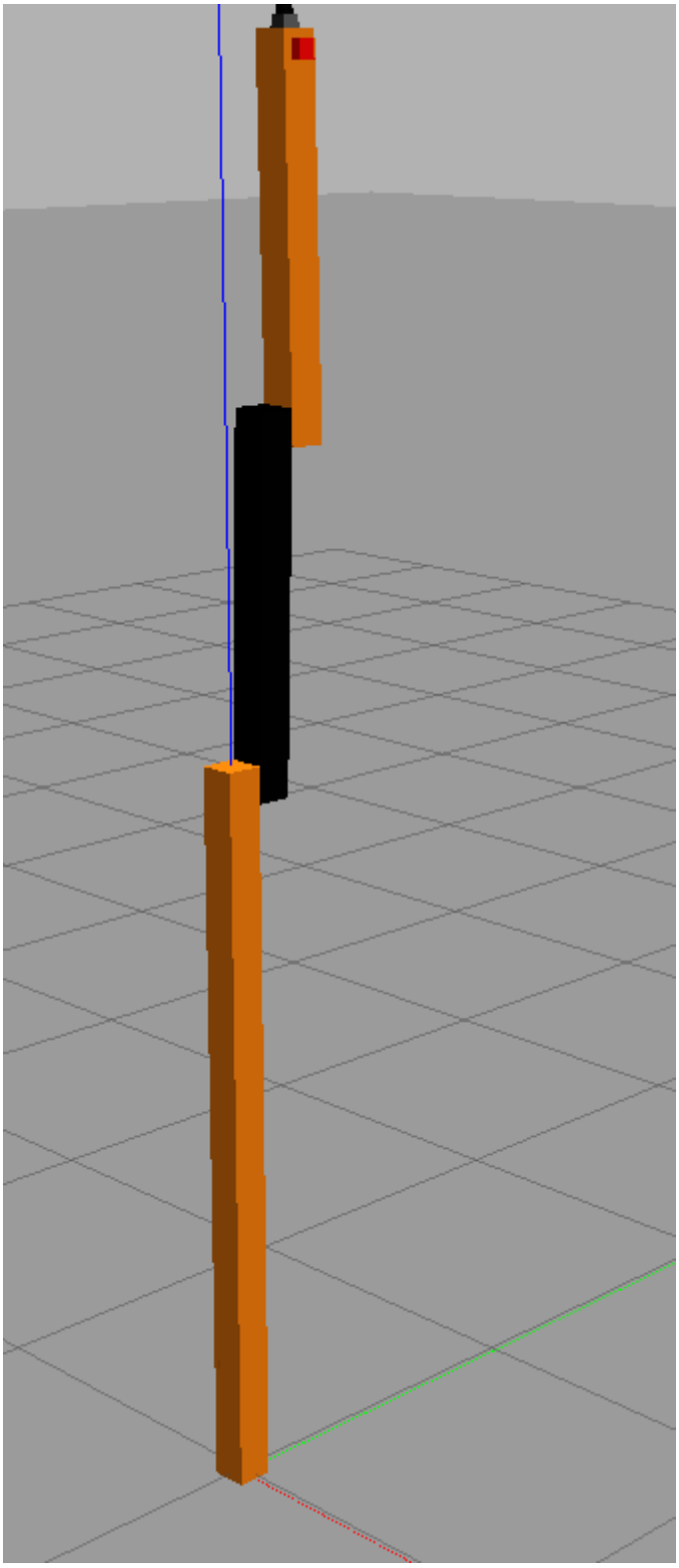
Problem H: ROS-Gazebo Simulation

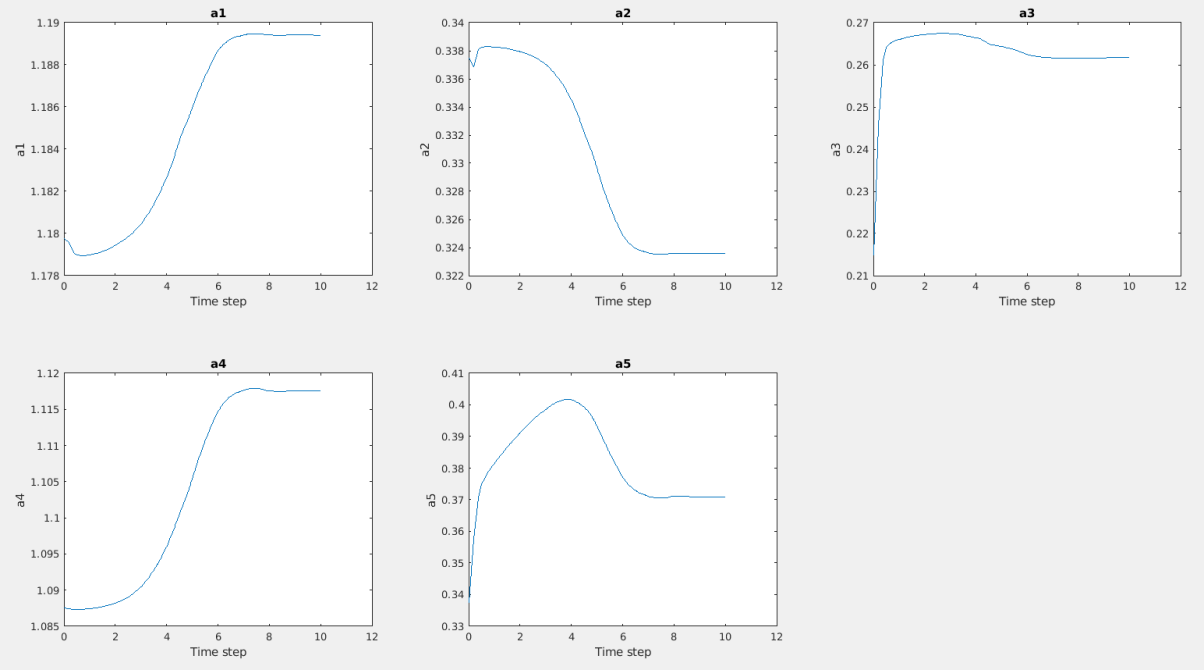












Discussion on ROS results:

Here we can observe that the control inputs in both the simulations (i.e, MATLAB and ROS) have similar profile but the one observed in ROS is a bit jittery. However, the maximum magnitude is almost same, and the profile is very similar for the control input. When studying the state variables in case of GAZEBO simulation we can observe that it is much more jittery as compared to that observed in the MATLAB simulation. But it is still evident that states eventually converge to the desired trajectory. When observing the alpha graphs, we can observe they vary similarly and converge to almost similar values in case of both MATLAB simulation and ROS simulation.

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 jitter 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