```
% clear all; close all; clc;
```

ROS Setup

```
rosshutdown;
rosinit;
j1_effort = rospublisher('/rrbot/joint1_effort_controller/command');
j2_effort = rospublisher('/rrbot/joint2_effort_controller/command');
JointStates = rossubscriber('/rrbot/joint_states');
tau1 = rosmessage(j1_effort);
tau2 = rosmessage(j2_effort);
tau1.Data = 0;
tau2.Data = 0;
send(j1_effort,tau1);
send(j2_effort,tau2);
client = rossvcclient('/gazebo/set_model_configuration');
req = rosmessage(client);
req.ModelName = 'rrbot';
req.UrdfParamName = 'robot_description';
req.JointNames = {'joint1','joint2'};
req.JointPositions = [deg2rad(200), deg2rad(125)];
resp = call(client,req,'Timeout',3);
tic;
t = 0;
X_desired_Rec = [];
X_Rec = [];
t_Rec = [];
u_Rec = [];

% Feedback Linearized Controller - Manipulator eq
global K
global a
M = [m2*l1^2 + 2*m2*cos(q2)*l1*r2 + m1*r1^2 + m2*r2^2 + I1 + I2,
 m2*r2^2 + l1*m2*cos(q2)*r2 + I2
     m2*r2^2 + l1*m2*cos(q2)*r2 + I2,
 m2*r2^2 + I2];
EOM_Coriolis_term = [-l1*m2*q2d*r2*sin(q2)*(2*q1d + q2d)
                      l1*m2*q1d^2*r2*sin(q2)];
EOM_gravity_term = [-g*(l1*m2*sin(q1) + m1*r1*sin(q1) + m2*r2*sin(q1 + q2))
                    -g*m2*r2*sin(q1 + q2)];
while(t < 9.9)
    t = toc;
    % read the joint states
    jointData = receive(JointStates);

    % Construct state vector
    X = [wrapTo2Pi(jointData.Position(1));
         wrapTo2Pi(jointData.Position(2));
         jointData.Velocity(1);
         jointData.Velocity(2)];
```

```
    % design your linearized feedback controller in the following
    % [a] Desired trajectory at this given time instant
    % Finding joint1 trajectory (q1 and q1d) at given time using cubic
 polynomial eq
    a_j1 = a(:,1);
    a0=a_j1(1); a1=a_j1(2); a2=a_j1(3); a3=a_j1(4);
    q1_desired = a0 + a1*t + a2*t^2 + a3*t^3;
    q1d_desired = a1 + 2*a2*t + 3*a3*t^2;
    q1dd_desired = 2*a2 + 6*a3*t;

    % Finding joint2 trajectory (q2 and q2d) at given time using cubic
 polynomial eq
    a_j2 = a(:,2);
    a0=a_j2(1); a1=a_j2(2); a2=a_j2(3); a3=a_j2(4);
    q2_desired = a0 + a1*t + a2*t^2 + a3*t^3;
    q2d_desired = a1 + 2*a2*t + 3*a3*t^2;
    q2dd_desired = 2*a2 + 6*a3*t;

    X_desired = [q1_desired;
                 q2_desired;
                 q1d_desired;
                 q2d_desired];

    U_desired = [q1dd_desired
                 q2dd_desired];

    % [b] Virtual Control Input
    v = -K*(X - X_desired) + U_desired;

    % [c] Final feedback linearized control law
    q1 = X(1);    q1d = X(3);
    q2 = X(2);    q2d = X(4);
    u = M*v + EOM_Coriolis_term + EOM_gravity_term;

    tau1.Data = u(1);
    tau2.Data = u(2);
    send(j1_effort,tau1);
    send(j2_effort,tau2);

    % you can sample data here to be plotted at the end
    X_Rec = [X_Rec X];
    t_Rec = [t_Rec t];
    u_Rec = [u_Rec u];
    X_desired_Rec = [X_desired_Rec X_desired];
end
tau1.Data = 0;
tau2.Data = 0;
send(j1_effort,tau1);
send(j2_effort,tau2);
% disconnect from roscore
rosshutdown;
```

Initializing global node /matlab_global_node_94537 with NodeURI http://msi:36509/

Shutting down global node /matlab_global_node_94537 with NodeURI http://msi:36509/

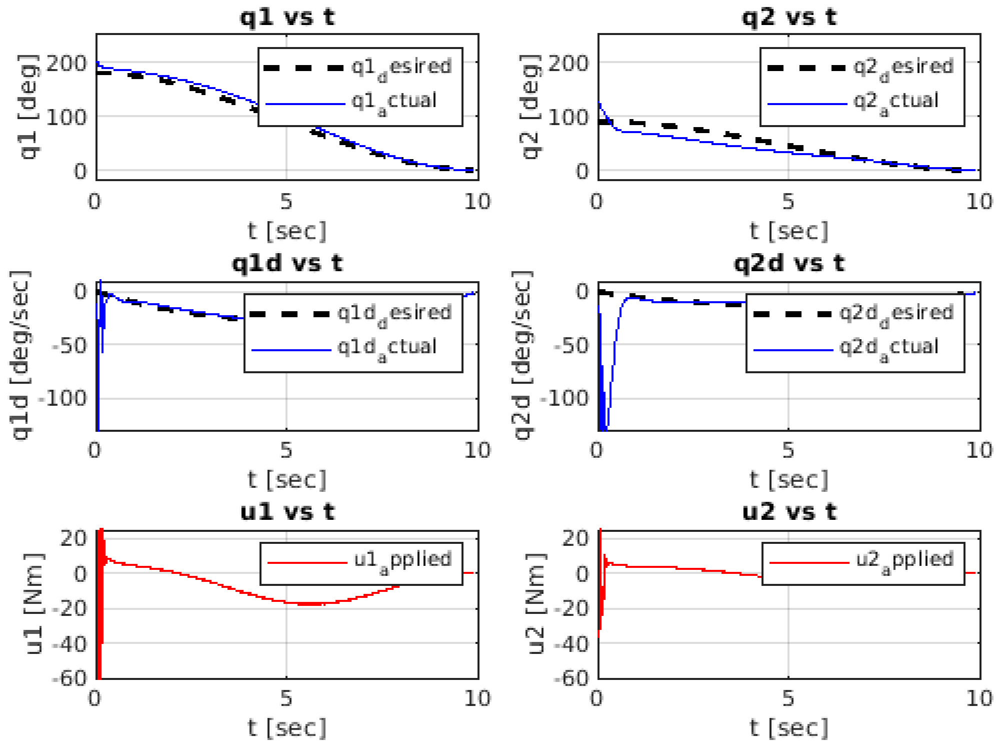
Plotting the results

Plot the q1, q1d, q2, q2d

```
figure;

% [1] Trajectory tracking of q1
subplot(3,2,1);
% Plotting q1_desired trajectory
wrapTo2Pi(X_desired_Rec(1,:));
plot(t_Rec,rad2deg(X_desired_Rec(1,:)),'k--', 'linewidth',2);  % ref line in
 black
% Plotting q1_actual trajectory from ODE solver
hold on
wrapTo2Pi(X_Rec(1,:));
plot(t_Rec,rad2deg(X_Rec(1,:)),'b');
% Graph Styling
title('q1 vs t');
xlabel('t [sec]');
ylabel('q1 [deg]');
axis([0 10 -20 250]);  % Setting limits on the output for pretty view
grid on;
legend('q1_desired', 'q1_actual');


% [2] Trajectory tracking of q2
subplot(3,2,2);
% Plotting q2_desired trajectory
wrapTo2Pi(X_desired_Rec(2,:));
plot(t_Rec,rad2deg(X_desired_Rec(2,:)),'k--', 'linewidth',2);  % ref line in
 black
% Plotting q2_actual trajectory from ODE solver
hold on
wrapTo2Pi(X_Rec(2,:));
plot(t_Rec,rad2deg(X_Rec(2,:)),'b');
% Graph Styling
title('q2 vs t');
xlabel('t [sec]');
ylabel('q2 [deg]');
axis([0 10 -20 250]);  % Setting limits on the output
grid on;
legend('q2_desired', 'q2_actual');



% [3] Trajectory tracking of q1d
subplot(3,2,3);
% Plotting q1d_desired trajectory
wrapTo2Pi(X_desired_Rec(3,:));
```

```
plot(t_Rec,rad2deg(X_desired_Rec(3,:)),'k--', 'linewidth',2);  % ref line in
 black
% Plotting q1d_actual trajectory from ODE solver
hold on
wrapTo2Pi(X_Rec(3,:));
plot(t_Rec,rad2deg(X_Rec(3,:)),'b');
% Graph Styling
title('q1d vs t');
xlabel('t [sec]');
ylabel('q1d [deg/sec]');
axis([0 10 -130 10]);  % Setting limits on the output for better view
grid on;
legend('q1d_desired', 'q1d_actual');



% [4] Trajectory tracking of q2d
subplot(3,2,4);
% Plotting q2d_desired trajectory
wrapTo2Pi(X_desired_Rec(4,:));
plot(t_Rec,rad2deg(X_desired_Rec(4,:)),'k--', 'linewidth',2);  % ref line in
 black
% Plotting q2d_actual trajectory from ODE solver
hold on
wrapTo2Pi(X_Rec(4,:));
plot(t_Rec,rad2deg(X_Rec(4,:)),'b');
% Graph Styling
title('q2d vs t');
xlabel('t [sec]');
ylabel('q2d [deg/sec]');
axis([0 10 -130 10]);  % Setting limits on the output for better view
grid on;
legend('q2d_desired', 'q2d_actual');



% [5] Plotting u1 generated and applied to joints with our control law
subplot(3,2,5);
plot(t_Rec,u_Rec(1,:),'r');
% Graph Styling
title('u1 vs t');
xlabel('t [sec]');
ylabel('u1 [Nm]');
axis([0 10 -60 25]);  % Setting limits on the output
grid on;
legend('u1_applied');



% [6] Plotting u2 generated and applied to joints with our control law
subplot(3,2,6);
plot(t_Rec,u_Rec(2,:),'r');
% Graph Styling
title('u2 vs t');
```

```
xlabel('t [sec]');
ylabel('u2 [Nm]');
axis([0 10 -60 25]);  % Setting limits on the output
grid on;
legend('u2_applied');
```



Published with MATLAB® R2021b