RBE502 HW3 By Ajay Balasubramanian

Table of Contents

In this homework, I implement controllers for a 2-link arm whose dynamics are already provided. Let's initialize some symbols for the joint angles, and angular velocities and accelerations. Let's also use a time interval of 10 seconds with a sampling time of 0.1 s.

```
syms q1 q1_dot q1_ddot q2 q2_dot q2_ddot
sampling_time = 0.1;
tspan = 0:sampling_time:10; %time steps, 101 steps of equal size from
 0 to 10
```

Part 1: Feedback Controller

For this part, we need a feedback controller. The dynamic equation for the 2 link manipulator is: M*q_ddot + C*q_dot + G = Tau, which is non-linear. We can use feedback linearization and decouple the joints to get the following equation for joint i: qi_ddot = acci

Here qi_ddot and acci are the angular acceleration for joint i and the control input respectively.

If x is the state vector: [qi;qi_dot], then we can represent this equation in state space form as: x_dot = [0 1; 0 0]*x + [0; 1]*acci. That is: x_dot = A*x + B*acci, where A = [0 1; 0 0] and B = [0; 1].

Since we want a feedback controller, let's put the controller acci = -K*x for some row vector K. That is, x_dot = (A - BK)*x. We find a K such that the matrix (A - BK) is stable (eigenvalues have -ve real parts). I do this using the place function and with eigenvalues -3 and -5.

Finally I need to model the whole system dynamics in state space form. Let z = [q1; q2; q1_dot; q2_dot] be the state vector. We can represent the dynamics equation: M*q_ddot + C*q_dot + G = Tau in state space form: z_dot = [z(3); z(4); invM(Tau - C*[z(3);z(4)] - G)(1); invM(Tau - C*[z(3);z(4)] - G)(2)]*z.

I implement a feedback controller along with the above operations inside a function called ode_feedback_2link.m. The full function is explained with comments and shown below.

```
function dzdt = ode_feedback_2link(t,z)
%Defining the A,B and K matrices in the equation: x_dot = Ax + Bacc
```

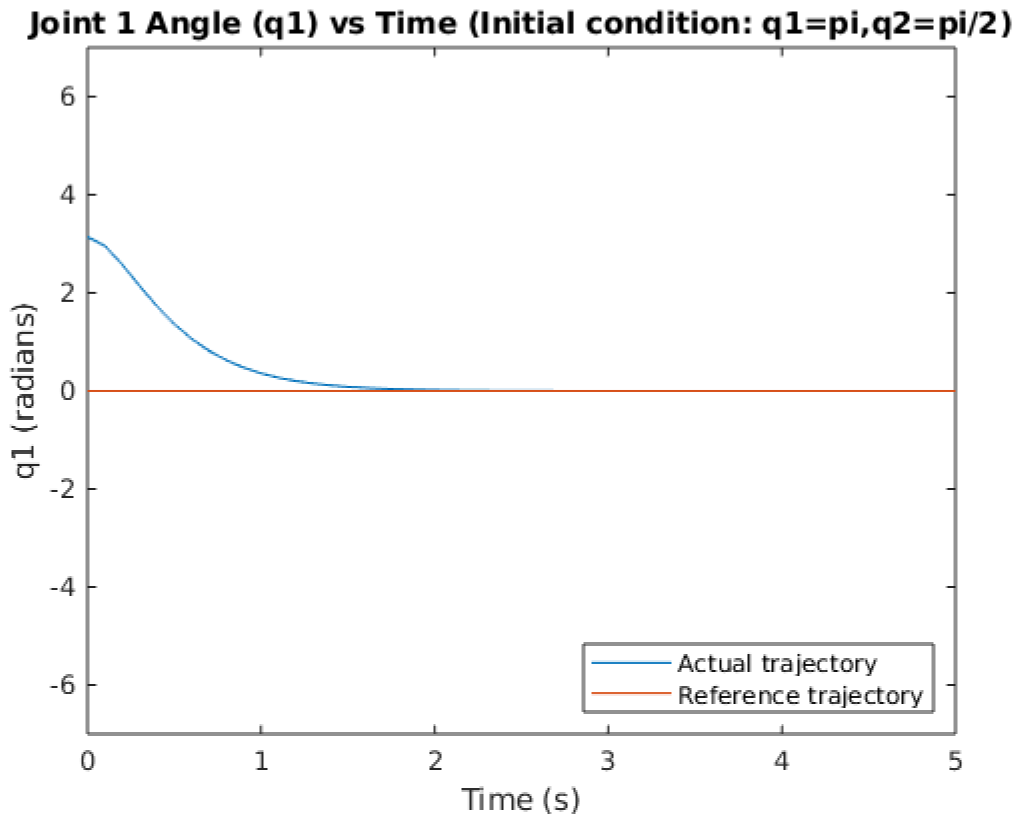
```
%And: acc = -Kx
A = [0 1; 0 0];
B = [0; 1];
K = place(A,B,[-3 -5]); %eigenvalues -3, -5
%Defining the output time derivative of state: z_dot
dzdt = zeros(4,1);
z = num2cell(z);
[q1, q2, q1_dot, q2_dot] = deal(z{:});
%Checking bounds of joint angles
if abs(q1) > 2*pi
q1 = mod(q1, 2*pi);
end
if abs(q2) > 2*pi
q2 = mod(q2, 2*pi);
end
%ai = -Kxi
acc1 = -K*[q1; q1_dot];
acc2 = -K*[q2; q2_dot];
acc = [acc1;acc2];
%2-link Arm parameters, (given)
I1=10;  I2 = 10; m1=5; r1=.5; m2=5; r2=.5; l1=1; l2=1;
g=9.8;
a = I1+I2+m1*r1^2+ m2*(l1^2+ r2^2);
b = m2*l1*r2;
d = I2+ m2*r2^2;
%
M = [a+2*b*cos(z{2}), d+b*cos(z{2});
    d+b*cos(z{2}), d];
C = [-b*sin(z{2})*z{4}, -b*sin(z{2})*(z{3}+z{4}); b*sin(z{2})*z{3},0];
G = [m1*g*r1*cos(z{1})+m2*g*(l1*cos(z{1})+r2*cos(z{1}+z{2}));
    m2*g*r2*cos(z{1}+z{2})];
%
invM = inv(M);
q_dot_vec = [z{3};z{4}];
%Calculating torque from dynamic equation: M*acc + C*q_dot + G = Tau
Tau = (M*acc) + (C*q_dot_vec) + G;
% Expression for qi_ddot/acc
qi_ddot = invM*(Tau - (C*q_dot_vec) - G);
%Defining the dynamics with first order ODEs
dzdt(1) = z{3};
dzdt(2) = z{4};
dzdt(3) = qi_ddot(1);
dzdt(4) = qi_ddot(2);
end
```

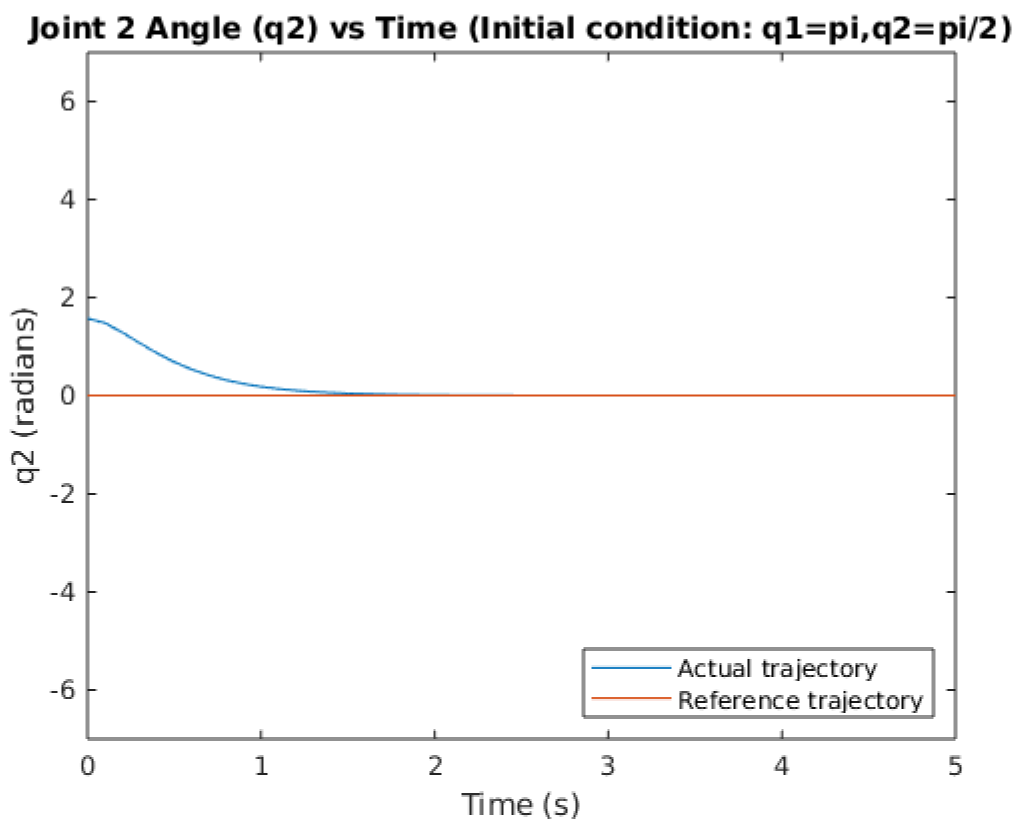
Now, that I've written the code for the controller in the ode function, it is time to feed it to the ode45 function and plot the results! Since we need to drive the system from [pi;pi/2] to the origin let's put the origin as reference trajectory and [pi;pi/2] as the initial position. We then feed the above ode function into ode45 and provide the time interval and initial configuration.

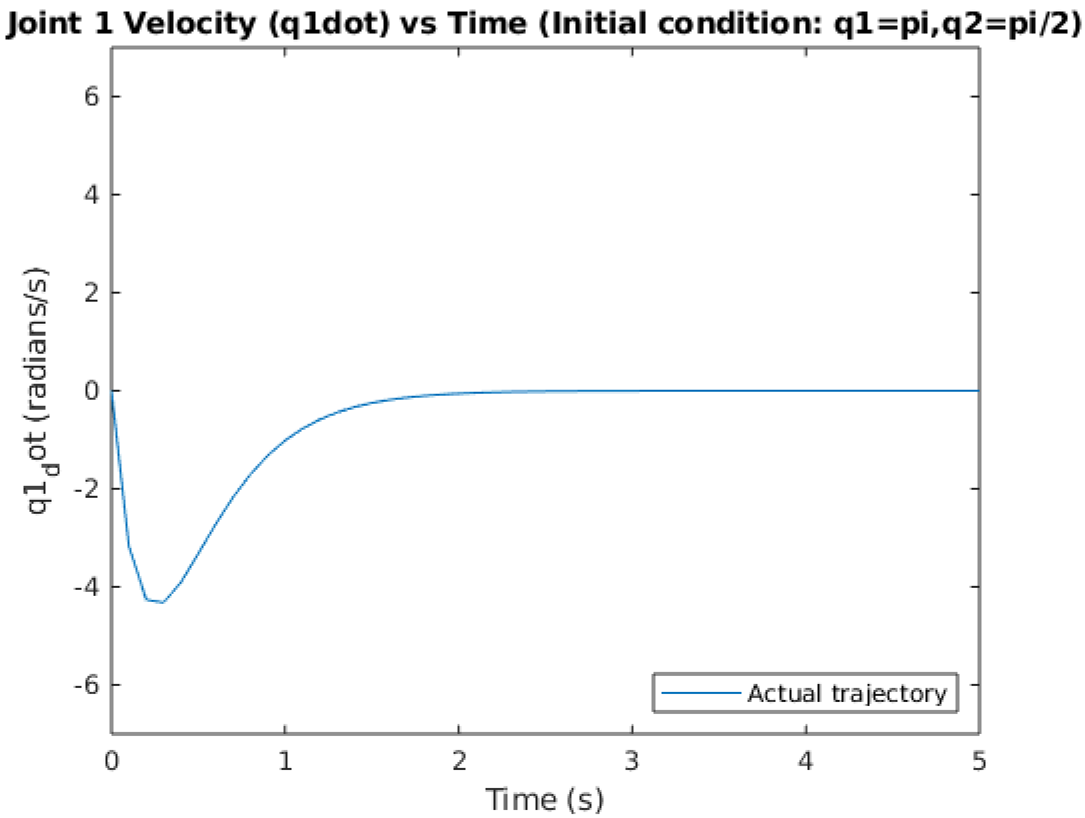
```
qi_ref = zeros(size(tspan));
xinit = [pi,pi/2,0,0]; %Initial configuration
[t,y] = ode45(@(t,z) ode_feedback_2link(t,z), tspan, xinit);
```

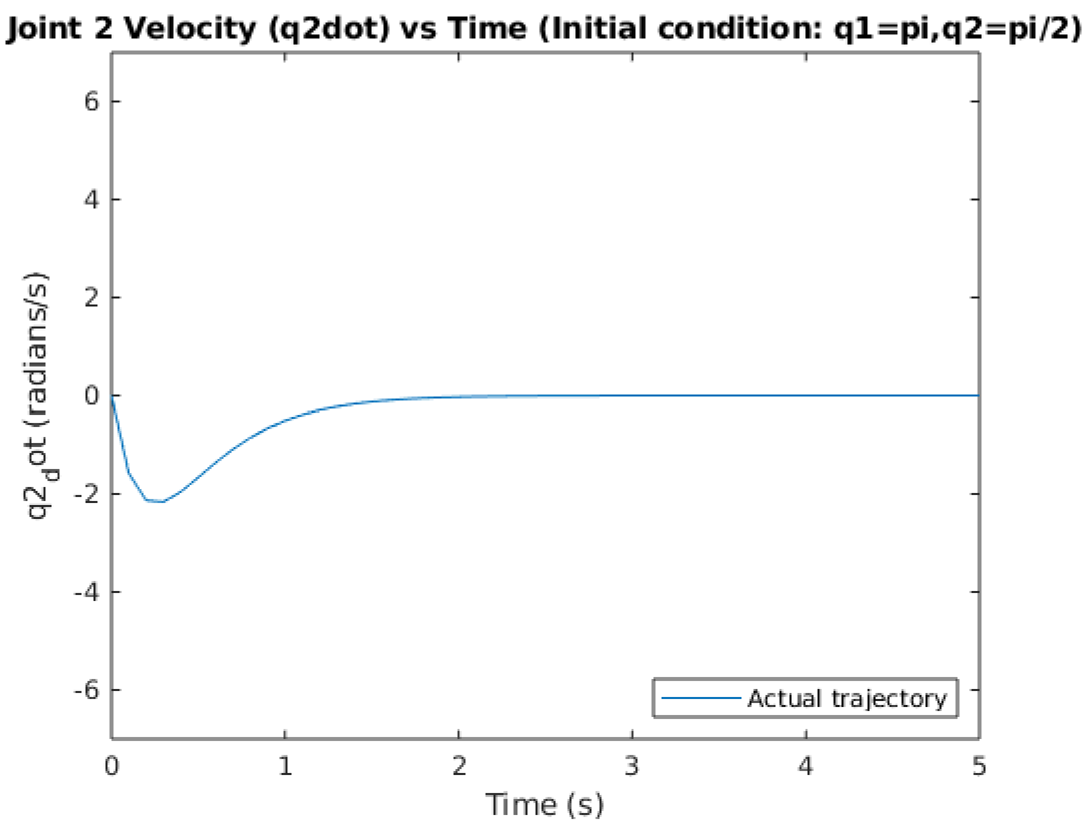
Let's plot the state trajectories!

```
figure(1);
h(1) = plot(t, y(:,1));
xlim([0 5]);
ylim([-7 7]);
hold on;
title(['Joint 1 Angle (q1) vs Time (Initial condition:
 q1=pi,q2=pi/2)']);
ylabel('q1 (radians)')
xlabel('Time (s)')
h(2) = plot(t, qi_ref);
legend(h,'Actual trajectory','Reference
 trajectory','Location','southeast');
%
figure(2);
h(1) = plot(t, y(:,2));
xlim([0 5]);
ylim([-7 7]);
hold on;
title(['Joint 2 Angle (q2) vs Time (Initial condition:
 q1=pi,q2=pi/2)']);
ylabel('q2 (radians)')
xlabel('Time (s)')
h(2) = plot(t, qi_ref);
legend(h,'Actual trajectory','Reference
 trajectory','Location','southeast');
%
figure(3);
h = plot(t, y(:,3));
xlim([0 5]);
ylim([-7 7]);
hold on;
title(['Joint 1 Velocity (q1dot) vs Time (Initial condition:
 q1=pi,q2=pi/2)']);
ylabel('q1_dot (radians/s)')
xlabel('Time (s)')
legend(h,'Actual trajectory','Location','southeast');
%
figure(4);
h = plot(t, y(:,4));
xlim([0 5]);
ylim([-7 7]);
hold on;
title(['Joint 2 Velocity (q2dot) vs Time (Initial condition:
 q1=pi,q2=pi/2)']);
ylabel('q2_dot (radians/s)')
xlabel('Time (s)')
legend(h,'Actual trajectory','Location','southeast');
```


Joint 1 Angle (q1) vs Time (Initial condition: q1=pi,q2=pi/2)
q1 (radians)
Time (s)
Actual trajectory
Reference trajectory
6
4
2
0
-2
-4
-6
0
1
2
3
4
5


Joint 2 Angle (q2) vs Time (Initial condition: q1=pi,q2=pi/2)
q2 (radians)
Time (s)
Actual trajectory
Reference trajectory
6
4
2
0
-2
-4
-6
0
1
2
3
4
5


Joint 1 Velocity (q1dot) vs Time (Initial condition: q1=pi,q2=pi/2)
q1dot (radians/s)
Time (s)
Actual trajectory


Joint 2 Velocity (q2dot) vs Time (Initial condition: q1=pi,q2=pi/2)
q2dot (radians/s)
Time (s)
Actual trajectory

As we can see, from the joint angle trajectories, they converge to 0 in finite time, as desired.

Part 2: Trajectory Generation

For this part, I generate cubic polynomial trajectories separately for each joint angle. The state space form for qi_ddot = acci is: x_dot = [0 1; 0 0]*x + [0;1]*acci. Clearly, this is in control canonical form and the trajectory will be dynamically feasible.

A cubic polynomial in time is of the form: a0 + a1*t + a2*t^2 + a3*t^3. Let this be equal to d_q1(t) (desired joint angle trajectory). Similarly, let d_q2(t) = b0 + b1*t + b2*t^2 + b3*t^3. Let us plug in the initial and final values to get some equations.

We are given: d_q1(0) = 0, d_q2(0) = 0, d_q1(10) = pi/3, d_q2(10) = pi/4. If we also take the initial and final joint angular velocities to be zero, we get 4 more equations. d_q1_dot(0) = 0, d_q2_dot(0) = 0, d_q1_dot(10) = 0, d_q2_dot(10) = 0.

Now we have 8 equations in 8 variables. Let's solve it separately for the two joints. We can model the equations for joint 1 as the matrix equation: A1*p1 = B1, where A1 = [1 0 0 0; 0 1 0 0; 1 10 100 1000; 0 1 20 300] B1 = [0; 0; pi/3; 0] and p1 = inv(A1)*B1 = [a0;a1;a2;a3]. Similarly, for joint 2, A2*p2 = B2, where A2 = [1 0 0 0; 0 1 0 0; 1 10 100 1000; 0 1 20 300], B2 = [0; 0; pi/4; 0]; p2 = inv(A2)*B2 = [b0;b1;b2;b3]. Let's code this.

```
%trajectory parameters for joint 1
A1 = [1 0 0 0; 0 1 0 0; 1 10 100 1000; 0 1 20 300];
B1 = [0; 0; pi/3; 0];
p1 = inv(A1)*B1;
%
%trajectory parameters for joint 2
A2 = [1 0 0 0; 0 1 0 0; 1 10 100 1000; 0 1 20 300];
B2 = [0; 0; pi/4; 0];
p2 = inv(A2)*B2;
fprintf('a0, a1, a2, a3 is: \n');
disp(p1);
fprintf('b0, b1, b2, b3 is: \n');
disp(p2);
```

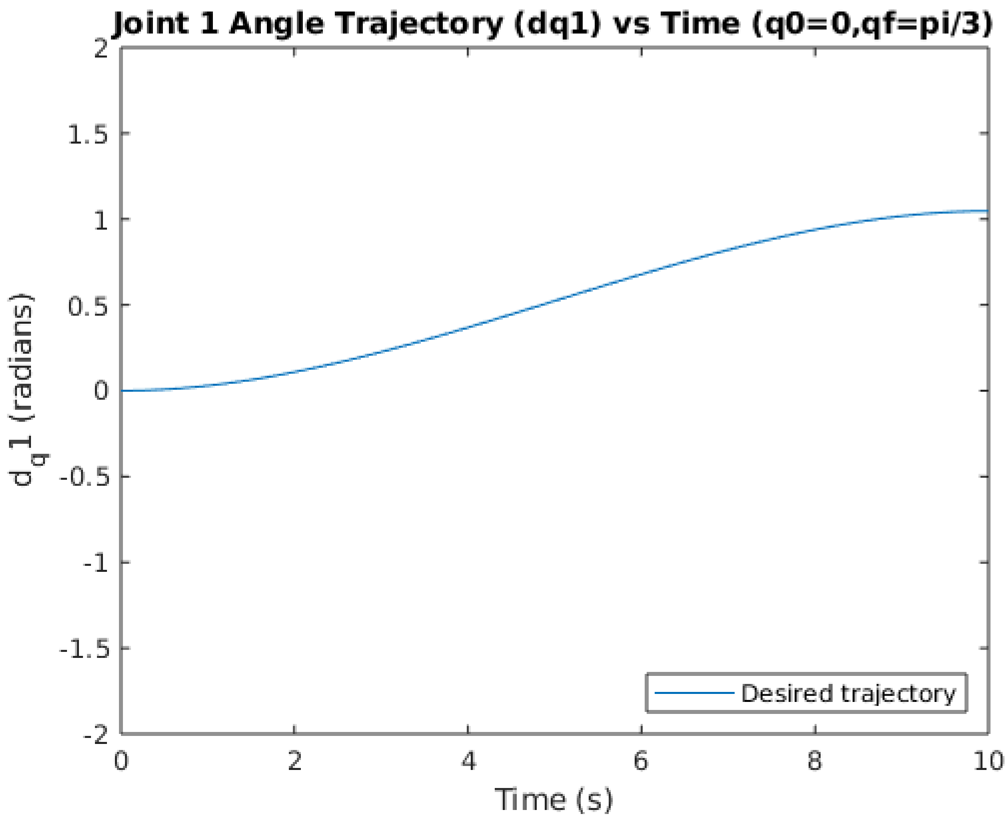
```
a0, a1, a2, a3 is:
         0
         0
    0.0314
   -0.0021

b0, b1, b2, b3 is:
         0
         0
    0.0236
   -0.0016
```

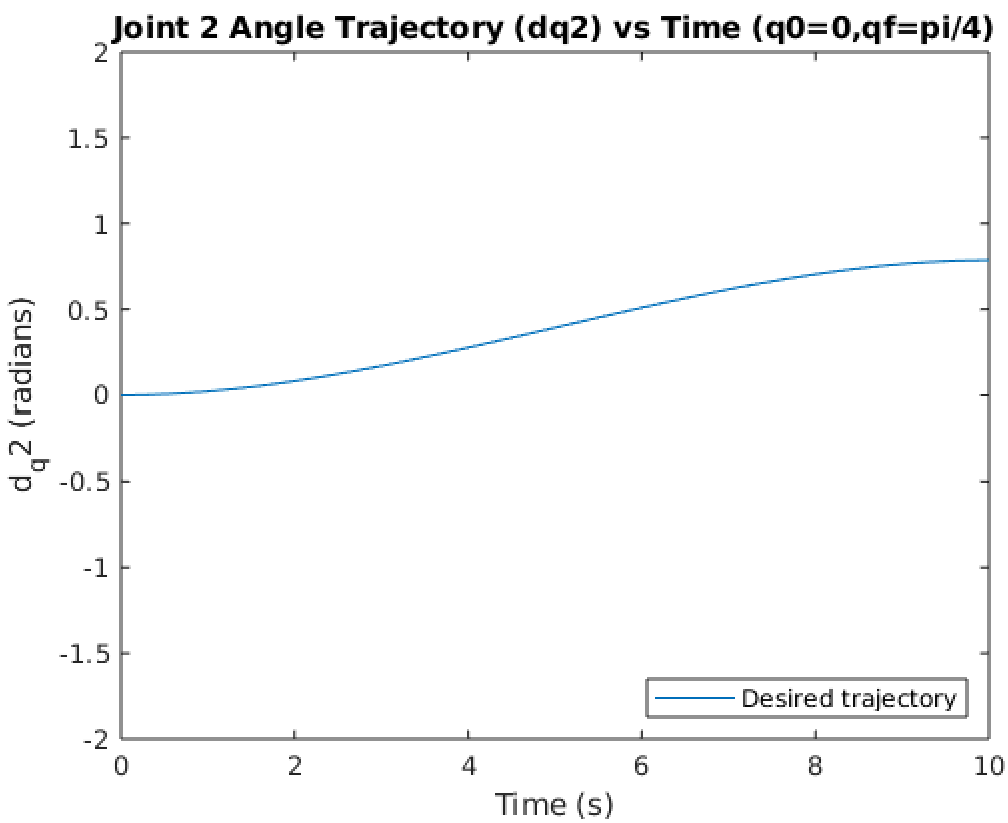
```
syms t
%trajectory equations for joint position and velocity
traj1 = p1(1) + (p1(2)*t) + (p1(3)*t^2) + (p1(4)*t^3);
traj2 = p2(1) + (p2(2)*t) + (p2(3)*t^2) + (p2(4)*t^3);
traj1_dot = (p1(2)) + (2*p1(3)*t) + (3*p1(4)*t^2);
```

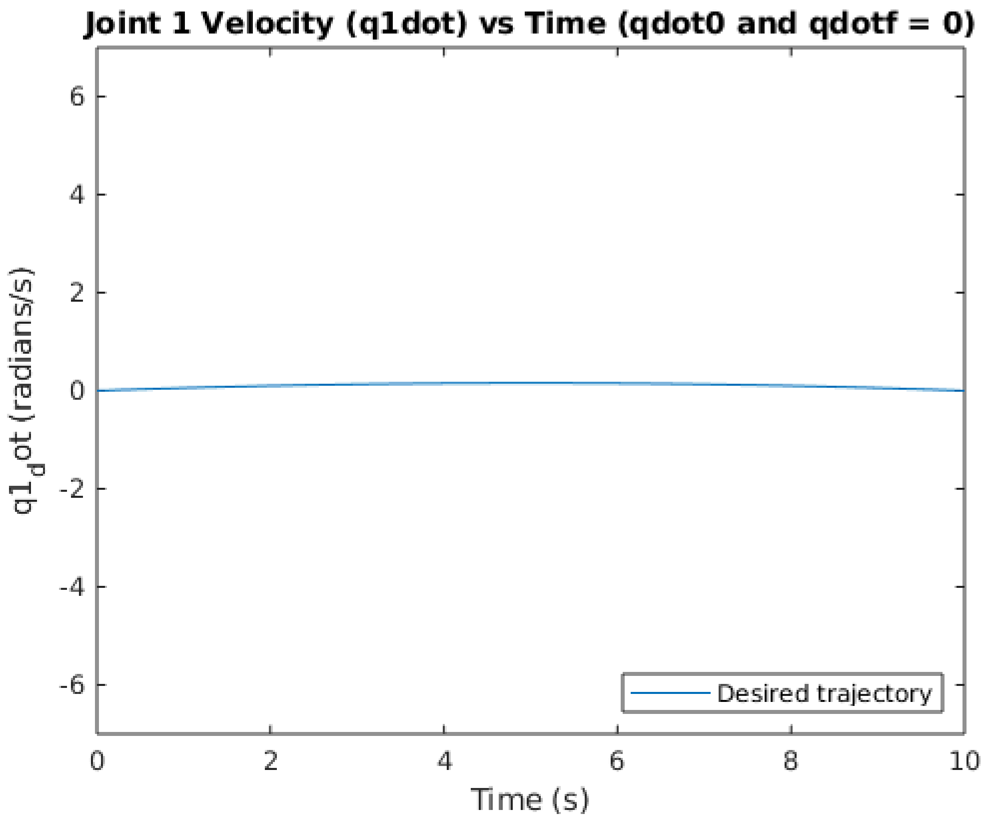
```
traj2_dot = (p2(2)) + (2*p2(3)*t) + (3*p2(4)*t^2);
%trajectory values at each timestep
traj1_num = zeros(size(tspan));
traj2_num = zeros(size(tspan));
traj1_dot_num = zeros(size(tspan));
traj2_dot_num = zeros(size(tspan));
for i=1:101
    traj1_num(i) = subs(traj1, t, tspan(i));
    traj2_num(i) = subs(traj2, t, tspan(i));
    traj1_dot_num(i) = subs(traj1_dot, t, tspan(i));
    traj2_dot_num(i) = subs(traj2_dot, t, tspan(i));
end
```

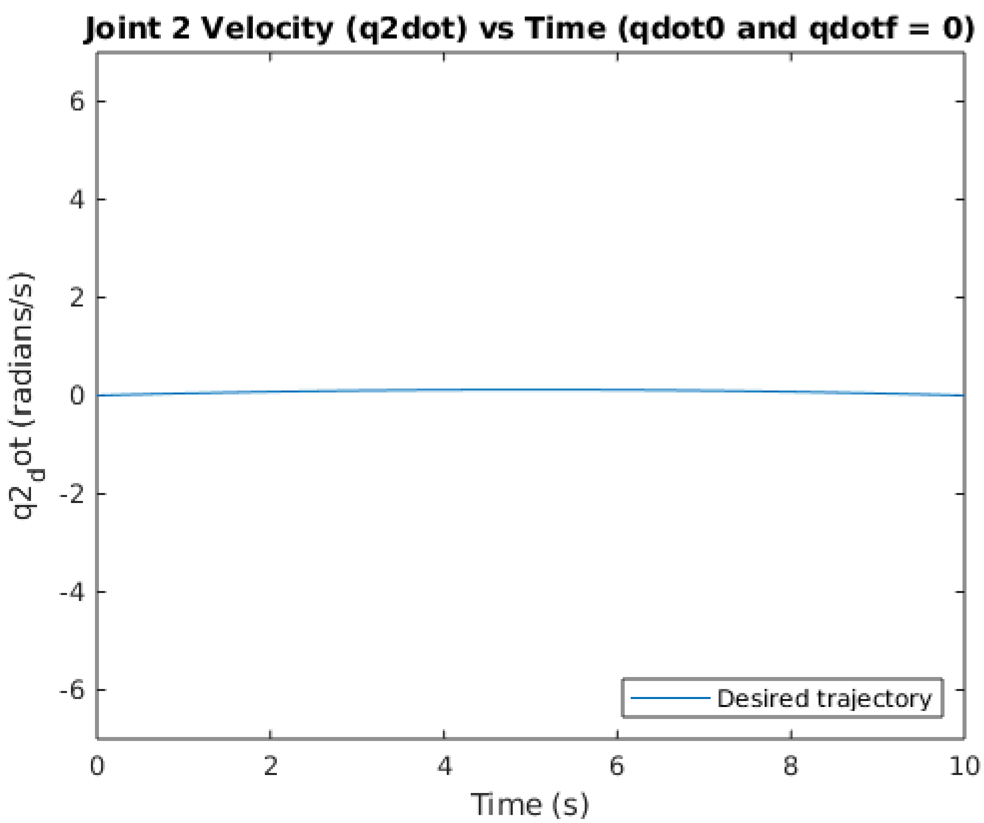
Let us plot the trajectories!

```
figure(5);
h = plot(tspan, traj1_num);
xlim([0 10]);
ylim([-2 2]);
hold on;
title(['Joint 1 Angle Trajectory (dq1) vs Time (q0=0,qf=pi/3)']);
ylabel('d_q1 (radians)')
xlabel('Time (s)')
legend(h,'Desired trajectory','Location','southeast');
%
figure(6);
h = plot(tspan, traj2_num);
xlim([0 10]);
ylim([-2 2]);
hold on;
title(['Joint 2 Angle Trajectory (dq2) vs Time (q0=0,qf=pi/4)']);
ylabel('d_q2 (radians)')
xlabel('Time (s)')
legend(h,'Desired trajectory','Location','southeast');
%
figure(7);
h = plot(tspan, traj1_dot_num);
xlim([0 10]);
ylim([-7 7]);
hold on;
title(['Joint 1 Velocity (q1dot) vs Time (qdot0 and qdotf = 0)']);
ylabel('q1_dot (radians/s)')
xlabel('Time (s)')
legend(h,'Desired trajectory','Location','southeast');
%
figure(8);
h = plot(tspan, traj2_dot_num);
xlim([0 10]);
ylim([-7 7]);
hold on;
title(['Joint 2 Velocity (q2dot) vs Time (qdot0 and qdotf = 0)']);
ylabel('q2_dot (radians/s)')
xlabel('Time (s)')
legend(h,'Desired trajectory','Location','southeast');
```


Joint 1 Angle Trajectory (dq1) vs Time (q0=0,qf=pi/3)
d_q1 (radians)
Time (s)
Desired trajectory
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
2
4
6
8
10


Joint 2 Angle Trajectory (dq2) vs Time (q0=0,qf=pi/4)
d_q2 (radians)
Time (s)
Desired trajectory
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
2
4
6
8
10


Joint 1 Velocity (q1dot) vs Time (qdot0 and qdotf = 0)
q1_dot (radians/s)
6
4
2
0
-2
-4
-6
Desired trajectory
0
2
4
6
8
10
Time (s)


Joint 2 Velocity (q2dot) vs Time (qdot0 and qdotf = 0)
q2_dot (radians/s)
6
4
2
0
-2
-4
-6
Desired trajectory
0
2
4
6
8
10
Time (s)

Part 3: Trajectory Tracking Controller

Let us again consider the system equation: qi_ddot = acci. We have the state space form as: x_dot = A*x + B*acci (from Part 1), where x = [qi; qi_dot]. Similarly, for the desired trajectories, d_x_dot = A*d_x + B*d_acci.

Let us model the error dynamics, e = x - d_x = [qi - d_qi; qi_dot - d_qi_dot]. Therefore, e_dot = (A*x + B*acci) - (A*d_x + B*d_acci) = A(x - d_x) + B(acci - d_acci) = A*e + B*v, where v = acci - d_acci.

Therefore, e_dot = A*e + B*v. This looks just like the state space form equation of a linear system. To drive this error to zero, let's put v = -K*e such that (A - B*K) is stable. But v = acci - d_acci. Therefore, acci = -K*e + d_acci. This is the desired control input. Everything else is the same as part 1.

I implement a trajectory tracking controller along with the above operations inside a function called ode_trajtracking_2link.m. It is very similar to the ode in the first part except for the trajectory parameters(a function input) and control. The full function is explained with comments and shown below.

```
function dzdt = ode_trajtracking_2link(t,z,params)
%Params is a 8x1 vector with the constants of the cubic polynomial
%trajectories of first and second joints.
A = [0 1; 0 0];
B = [0; 1];
K = place(A,B,[-3 -5]);
%
dzdt = zeros(4,1);
z = num2cell(z);
[q1, q2, q1_dot, q2_dot] = deal(z{:});
if abs(q1) > 2*pi
q1 = mod(q1, 2*pi);
end
if abs(q2) > 2*pi
q2 = mod(q2, 2*pi);
end
%defining desired trajectories
d_q1 = params(1) + (params(2)*t) + (params(3)*t^2) + (params(4)*t^3);
d_q2 = params(5) + (params(6)*t) + (params(7)*t^2) + (params(8)*t^3);
d_q1_dot = (params(2)) + (2*params(3)*t) + (3*params(4)*t^2); %first
 time derivative
d_q2_dot = (params(6)) + (2*params(7)*t) + (3*params(8)*t^2); %first
 time derivative
d_acc1 = (2*params(3)) + (6*params(4)*t); %second time derivative
d_acc2 = (2*params(7)) + (6*params(8)*t); %second time derivative
%control inputs for the joints
acc1 = (-K*[q1 - d_q1; q1_dot - d_q1_dot]) + d_acc1;
acc2 = (-K*[q2 - d_q2; q2_dot - d_q2_dot]) + d_acc2;
acc = [acc1;acc2];
%2-link Arm parameters, (given)
I1=10;  I2 = 10; m1=5; r1=.5; m2=5; r2=.5; l1=1; l2=1;
g=9.8;
a = I1+I2+m1*r1^2+ m2*(l1^2+ r2^2);
b = m2*l1*r2;
d = I2+ m2*r2^2;
%
```

```
M = [a+2*b*cos(z{2}), d+b*cos(z{2});
    d+b*cos(z{2}), d];
C = [-b*sin(z{2})*z{4}, -b*sin(z{2})*(z{3}+z{4}); b*sin(z{2})*z{3},0];
G = [m1*g*r1*cos(z{1})+m2*g*(l1*cos(z{1})+r2*cos(z{1}+z{2}));
    m2*g*r2*cos(z{1}+z{2})];
%
invM = inv(M);
q_dot_vec = [z{3};z{4}];
%Calculating torque
Tau = (M*acc) + (C*q_dot_vec) + G;
% Expression for qi_ddot
qi_ddot = invM*(Tau - (C*q_dot_vec) - G);
%Defining the dynamics with first order ODEs
dzdt(1) = z{3};
dzdt(2) = z{4};
dzdt(3) = qi_ddot(1);
dzdt(4) = qi_ddot(2);
end
```

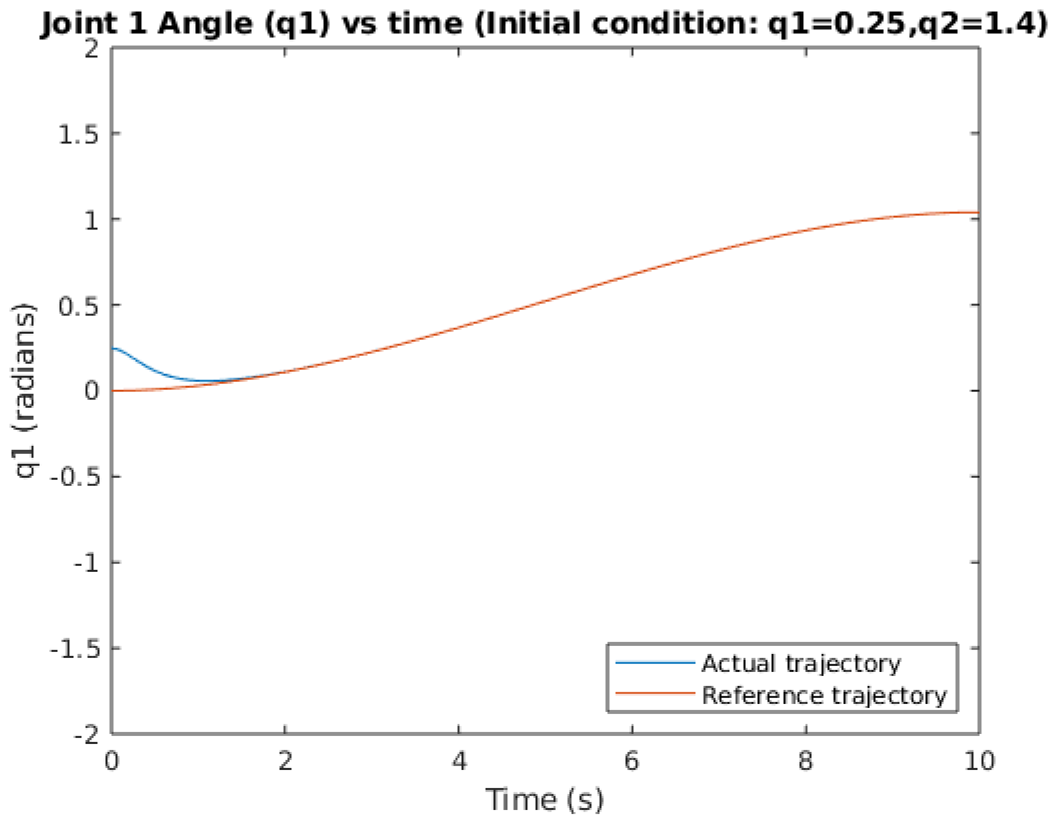
Now let's feed this ode to the ode45 function along with the initial joint positions: 0.25 and 1.4 for joints 1 and 2 respectively.

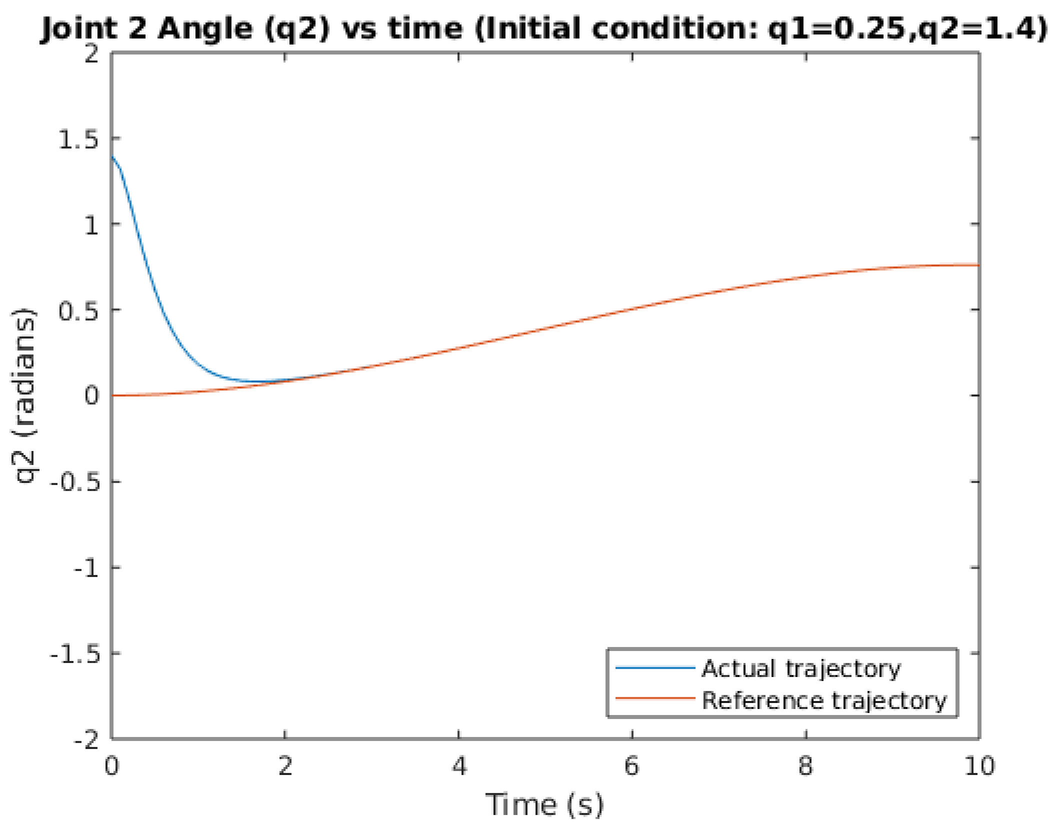
```
params_num = [0;0;0.0314;-0.0021;0;0;0.0236;-0.0016]; %The 8
 trajectory parameters obtained from Part 2
traj1 = params_num(1) + (params_num(2)*t) + (params_num(3)*t^2) +
 (params_num(4)*t^3);
traj2 = params_num(5) + (params_num(6)*t) + (params_num(7)*t^2) +
 (params_num(8)*t^3);
traj1_dot = (params_num(2)) + (2*params_num(3)*t) +
 (3*params_num(4)*t^2);
traj2_dot = (params_num(6)) + (2*params_num(7)*t) +
 (3*params_num(8)*t^2);
%
traj1_ref = zeros(size(tspan));
traj2_ref = zeros(size(tspan));
traj1_dot_ref = zeros(size(tspan));
traj2_dot_ref = zeros(size(tspan));
for i=1:101
    traj1_ref(i) = subs(traj1, t, tspan(i));
    traj2_ref(i) = subs(traj2, t, tspan(i));
    traj1_dot_ref(i) = subs(traj1_dot, t, tspan(i));
    traj2_dot_ref(i) = subs(traj2_dot, t, tspan(i));
end
xinit = [0.25,1.4,0,0]; %Initial configuration
[t,y] = ode45(@(t,z) ode_trajtracking_2link(t,z,params_num), tspan,
 xinit);
```

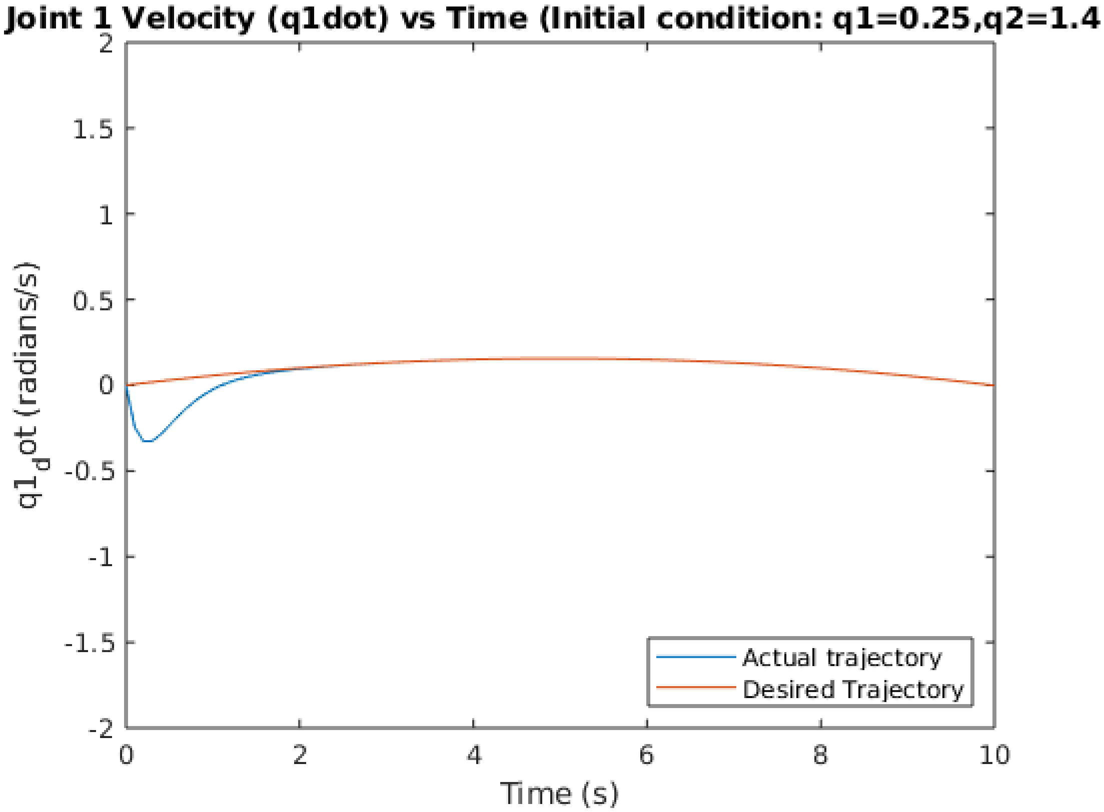
Let's plot the results of the trajectory tracking controller!

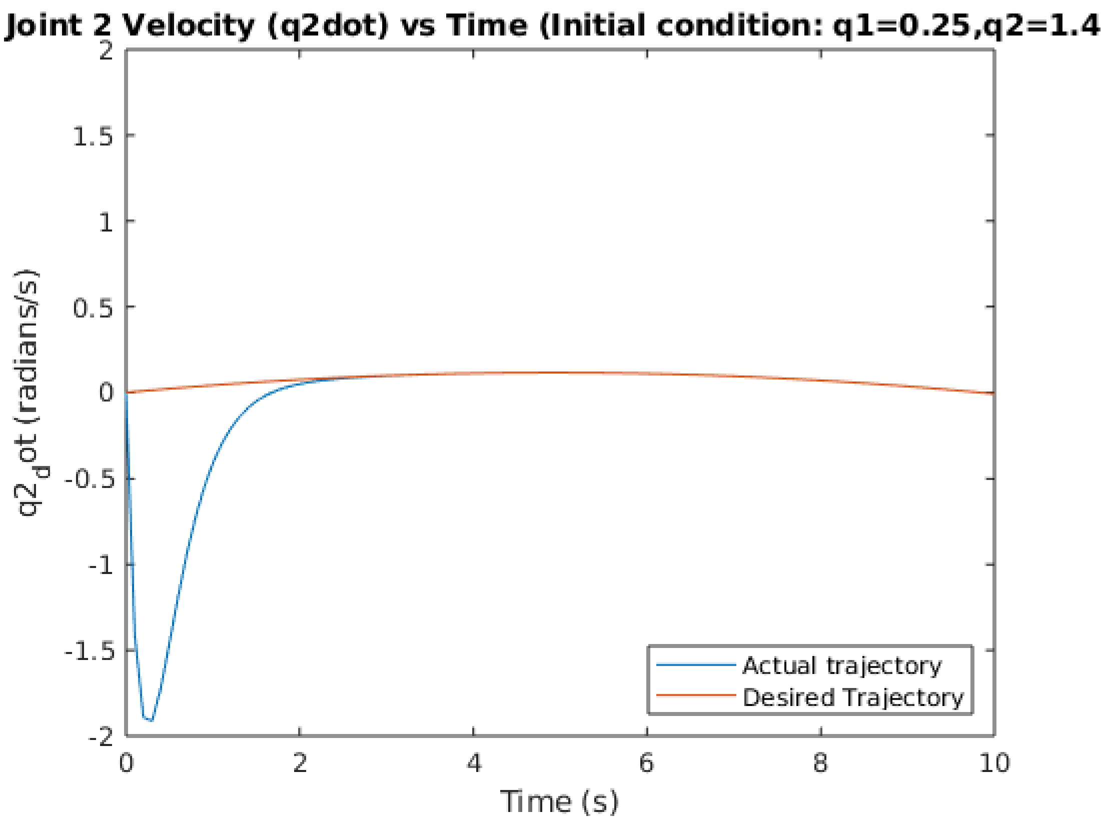
```
figure(9);
h(1) = plot(t, y(:,1));
xlim([0 10]);
ylim([-2 2]);
hold on;
```

```
title(['Joint 1 Angle (q1) vs time (Initial condition:
 q1=0.25,q2=1.4)']);
ylabel('q1 (radians)')
xlabel('Time (s)')
h(2) = plot(t, traj1_ref);
legend(h,'Actual trajectory','Reference
 trajectory','Location','southeast');
%
figure(10);
h(1) = plot(t, y(:,2));
xlim([0 10]);
ylim([-2 2]);
hold on;
title(['Joint 2 Angle (q2) vs time (Initial condition:
 q1=0.25,q2=1.4)']);
ylabel('q2 (radians)')
xlabel('Time (s)')
h(2) = plot(t, traj2_ref);
legend(h,'Actual trajectory','Reference
 trajectory','Location','southeast');
%
figure(11);
h(1) = plot(t, y(:,3));
xlim([0 10]);
ylim([-2 2]);
hold on;
title(['Joint 1 Velocity (q1dot) vs Time (Initial condition:
 q1=0.25,q2=1.4)']);
ylabel('q1_dot (radians/s)')
xlabel('Time (s)')
h(2) = plot(t, traj1_dot_ref);
legend(h,'Actual trajectory','Desired
 Trajectory','Location','southeast');
%
figure(12);
h(1) = plot(t, y(:,4));
xlim([0 10]);
ylim([-2 2]);
hold on;
title(['Joint 2 Velocity (q2dot) vs Time (Initial condition:
 q1=0.25,q2=1.4)']);
ylabel('q2_dot (radians/s)')
xlabel('Time (s)')
h(2) = plot(t, traj2_dot_ref);
legend(h,'Actual trajectory','Desired
 Trajectory','Location','southeast');
%
```


Joint 1 Angle (q1) vs time (Initial condition: q1=0.25,q2=1.4)
q1 (radians)
Time (s)
Actual trajectory
Reference trajectory
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
2
4
6
8
10


Joint 2 Angle (q2) vs time (Initial condition: q1=0.25,q2=1.4)
q2 (radians)
Time (s)
Actual trajectory
Reference trajectory
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
2
4
6
8
10


Joint 1 Velocity (q1dot) vs Time (Initial condition: q1=0.25,q2=1.4
q1$_d$ot (radians/s)
Time (s)
Actual trajectory
Desired Trajectory


Joint 2 Velocity (q2dot) vs Time (Initial condition: q1=0.25,q2=1.4
q2$_d$ot (radians/s)
Time (s)
Actual trajectory
Desired Trajectory

Part 4: Bonus (Trajectory Tracking with Integral Action)

For this part only a few changes/additions have to be made from Part 3. Instead of passing a vector of [q1,q2,q1_dot,q2_dot] to ode45, I pass the vector of errors that also includes the integral, so the new vector is [e1_i,e2_i,e1,e2,e1_dot,e2_dot], where e1_i = q1_i - d_q1_i is the integral of the error in joint 1 positions, and so on.

Like in Part 3 we model the error dynamics for the equation: qi_ddot = acci, but this time I also include the integral error. Therefore e = [e_i;e;e_dot] and the state space form becomes: A*e + B*v, where A = [0 1 0;0 0 1;0 0 0], B = [0;0;1] and v is the error in control input (acc - d_acc). Let v = -Ke like before to drive the error to zero.

This time K is a 1x3 vector and I find the poles with eigenvalues -3, -5, -7. The new control becomes acc = -K*e + d_acc.

I implement the new controller in a function called ode_integral_2link.m as shown below.

```
function dedt = ode_integral_2link(t,e,params)
%Params is a 8x1 vector with the constants of the cubic polynomial
%trajectories of first and second joints.
A = [0 1 0; 0 0 1; 0 0 0];
B = [0; 0; 1];
K = place(A,B,[-3 -5 -7]);
%
dedt = zeros(6,1);
e = num2cell(e);
[e1_i, e2_i, e1, e2, e1_dot, e2_dot] = deal(e{:});
if abs(e1) > 2*pi
e1 = mod(e1, 2*pi);
end
if abs(e2) > 2*pi
e2 = mod(e2, 2*pi);
end
%defining desired trajectories at time t
d_q1 = params(1) + (params(2)*t) + (params(3)*t^2) + (params(4)*t^3);
d_q2 = params(5) + (params(6)*t) + (params(7)*t^2) + (params(8)*t^3);
d_q1_dot = (params(2)) + (2*params(3)*t) + (3*params(4)*t^2); %first
 time derivative
d_q2_dot = (params(6)) + (2*params(7)*t) + (3*params(8)*t^2); %first
 time derivative
d_acc1 = (2*params(3)) + (6*params(4)*t); %second time derivative
d_acc2 = (2*params(7)) + (6*params(8)*t); %second time derivative
%control input
acc1 = (-K*[e1_i; e1; e1_dot]) + d_acc1;
acc2 = (-K*[e2_i; e2; e2_dot]) + d_acc2;
acc = [acc1;acc2];
%2-link Arm parameters, (given)
I1=10;  I2 = 10; m1=5; r1=.5; m2=5; r2=.5; l1=1; l2=1;
g=9.8;
a = I1+I2+m1*r1^2+ m2*(l1^2+ r2^2);
b = m2*l1*r2;
```

```
d = I2+ m2*r2^2;
%
M = [a+2*b*cos(e{4}+d_q2), d+b*cos(e{4}+d_q2);
    d+b*cos(e{4}+d_q2), d];
C = [-b*sin(e{4}+d_q2)*(e{6}+d_q2_dot), -
b*sin(e{4}+d_q2)*(e{5}+d_q1_dot+e{6}+d_q2_dot);
 b*sin(e{4}+d_q2)*(e{5}+d_q1_dot),0];
G =
 [m1*g*r1*cos(e{3}+d_q1)+m2*g*(l1*cos(e{3}+d_q1)+r2*cos(e{3}+d_q1+e{4}+d_q2));
    m2*g*r2*cos(e{3}+d_q1+e{4}+d_q2)];
%
invM = inv(M);
q_dot_vec = [e{5}+d_q1_dot; e{6}+d_q2_dot];
%Calculating torque
Tau = (M*acc) + (C*q_dot_vec) + G;
% Expression for qi_ddot
qi_ddot = invM*(Tau - (C*q_dot_vec) - G);
%Defining the dynamics with first order ODEs
dedt(1) = e{3};
dedt(2) = e{4};
dedt(3) = e{5};
dedt(4) = e{6};
dedt(5) = qi_ddot(1) - d_acc1;
dedt(6) = qi_ddot(2) - d_acc2;
end
```

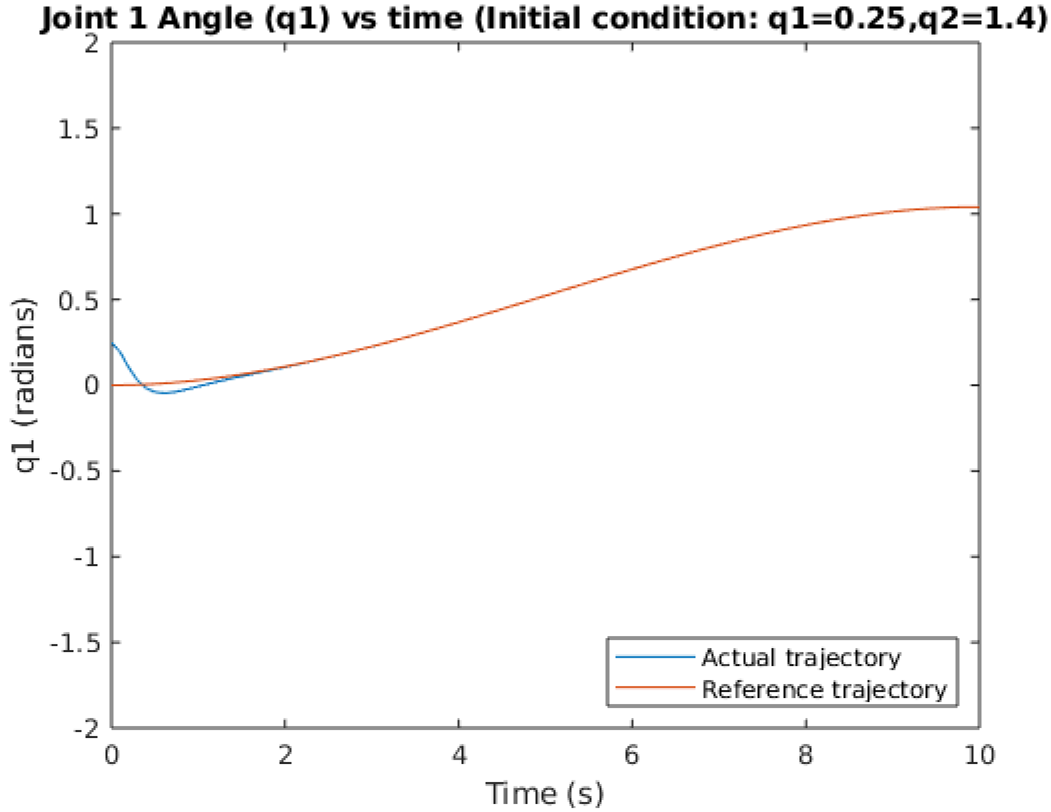
Let us plot the trajectories after using ode45 with the same initial errors as the controller without integral action.

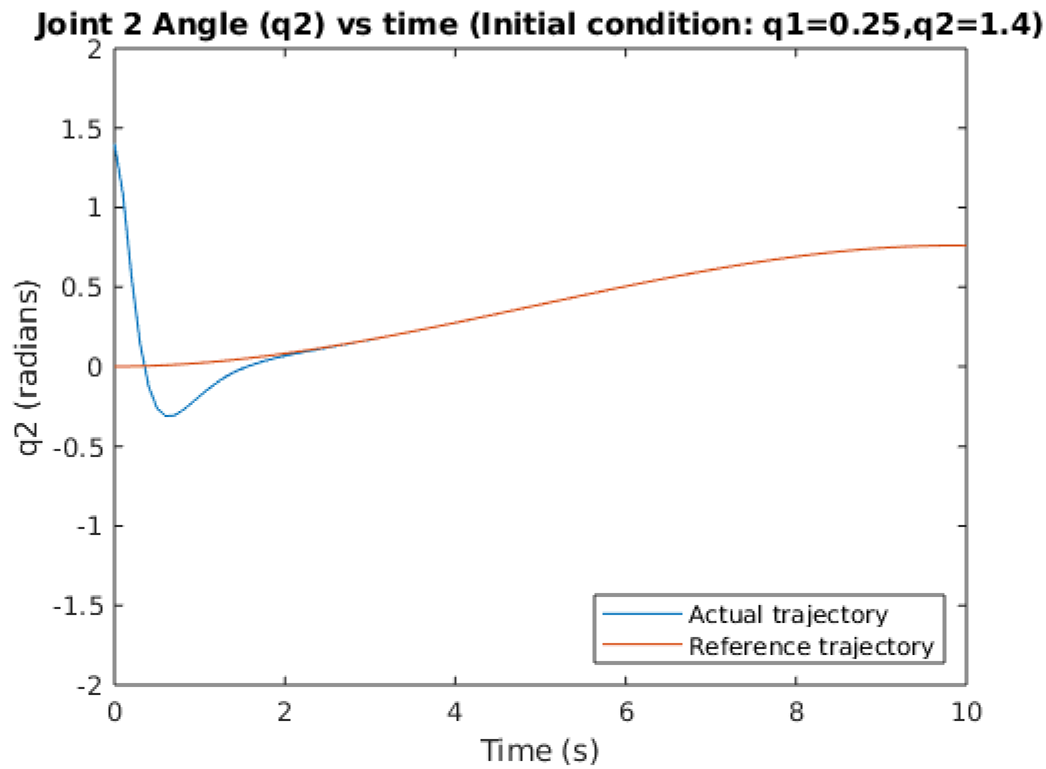
```
einit = [0,0,0.25, 1.4,0,0]; %Initial error in position
[t,y] = ode45(@(t,z) ode_integral_2link(t,z,params_num), tspan,
 einit);
plotfun1 = y(:,3)+transpose(traj1_ref);
plotfun2 = y(:,4)+transpose(traj2_ref);
plotfun3 = y(:,5)+transpose(traj1_dot_ref);
plotfun4 = y(:,6)+transpose(traj2_dot_ref);
```

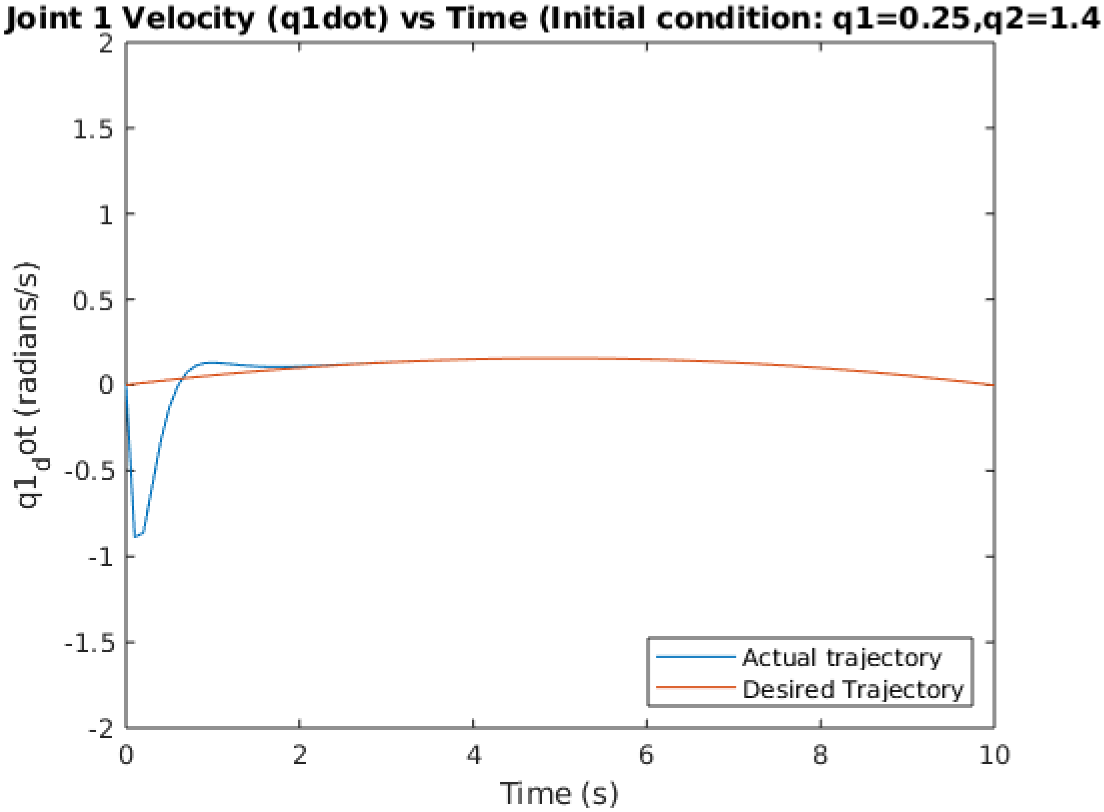
Plotting results!

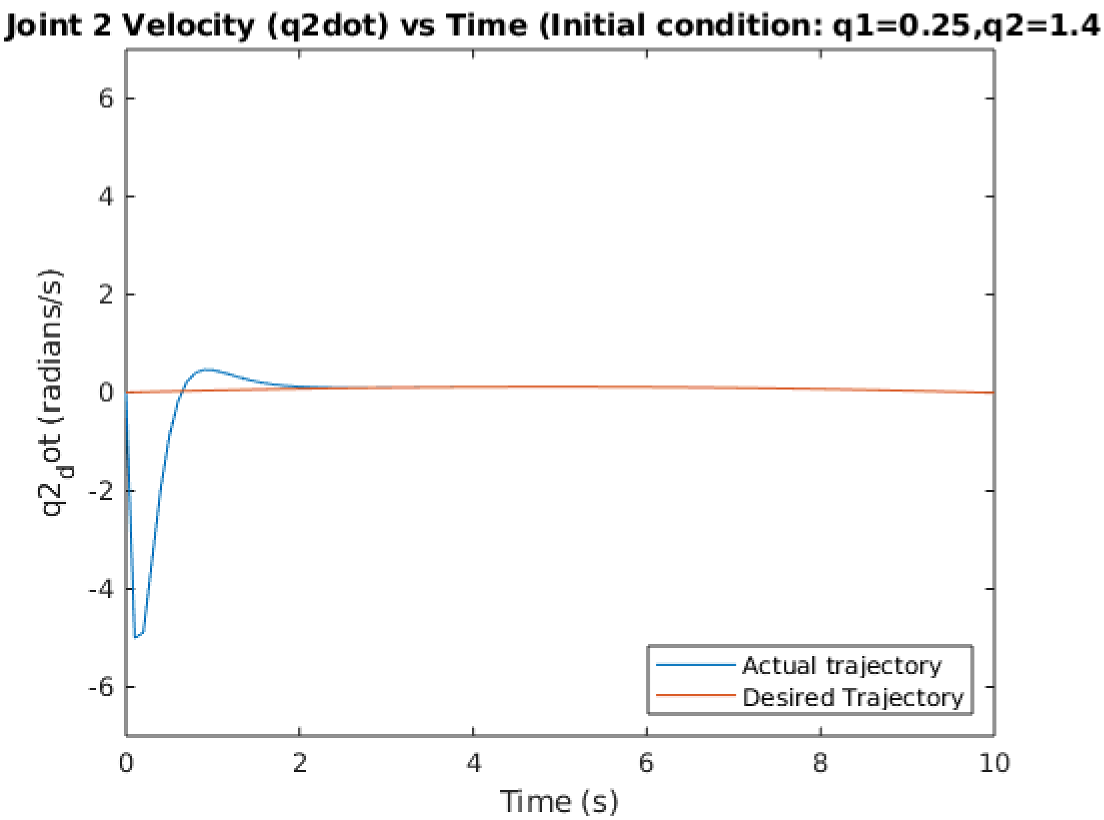
```
figure(13);
h(1) = plot(t, plotfun1);
xlim([0 10]);
ylim([-2 2]);
hold on;
title(['Joint 1 Angle (q1) vs time (Initial condition:
 q1=0.25,q2=1.4)']);
ylabel('q1 (radians)')
xlabel('Time (s)')
h(2) = plot(t, traj1_ref);
legend(h,'Actual trajectory','Reference
 trajectory','Location','southeast');
%
figure(14);
h(1) = plot(t, plotfun2);
```

```
xlim([0 10]);
ylim([-2 2]);
hold on;
title(['Joint 2 Angle (q2) vs time (Initial condition:
 q1=0.25,q2=1.4)']);
ylabel('q2 (radians)')
xlabel('Time (s)')
h(2) = plot(t, traj2_ref);
legend(h,'Actual trajectory','Reference
 trajectory','Location','southeast');
%
figure(15);
h(1) = plot(t, plotfun3);
xlim([0 10]);
ylim([-2 2]);
hold on;
title(['Joint 1 Velocity (q1dot) vs Time (Initial condition:
 q1=0.25,q2=1.4)']);
ylabel('q1_dot (radians/s)')
xlabel('Time (s)')
h(2) = plot(t, traj1_dot_ref);
legend(h,'Actual trajectory','Desired
 Trajectory','Location','southeast');
%
figure(16);
h(1) = plot(t, plotfun4);
xlim([0 10]);
ylim([-7 7]);
hold on;
title(['Joint 2 Velocity (q2dot) vs Time (Initial condition:
 q1=0.25,q2=1.4)']);
ylabel('q2_dot (radians/s)')
xlabel('Time (s)')
h(2) = plot(t, traj2_dot_ref);
legend(h,'Actual trajectory','Desired
 Trajectory','Location','southeast');
%
```


Joint 1 Angle (q1) vs time (Initial condition: q1=0.25,q2=1.4)
q1 (radians)
Time (s)
Actual trajectory
Reference trajectory
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
2
4
6
8
10


Joint 2 Angle (q2) vs time (Initial condition: q1=0.25,q2=1.4)
q2 (radians)
Time (s)
Actual trajectory
Reference trajectory
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
2
4
6
8
10


Joint 1 Velocity (q1dot) vs Time (Initial condition: q1=0.25,q2=1.4
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
2
4
6
8
10
Time (s)
q1_dot (radians/s)
Actual trajectory
Desired Trajectory


Joint 2 Velocity (q2dot) vs Time (Initial condition: q1=0.25,q2=1.4
6
4
2
0
-2
-4
-6
0
2
4
6
8
10
Time (s)
q2_dot (radians/s)
Actual trajectory
Desired Trajectory

We can see that the actual trajectories for the joints converge to the desired trajectories as desired. We can notice some differences in the performance of the controller compared to the case without integral action. I make the following observations:

1. The controller with integral action has large overshoots initially before converging, unlike the controller without integral action. 2. The controller with integral action reaches the desired trajectory sooner but takes time to stabilise. The controller without integral action takes more time to reach the desired trajectory but it is more stable.

Published with MATLAB® R2019b