ECE470 LAB-2 REPORT

Deniz Akyildiz

1003102224

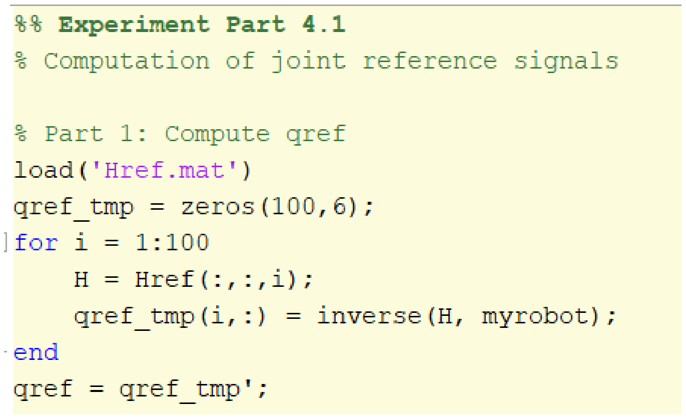
EXPERIMENT

4.1 Computation of joint reference signals

In this part, the reference signals to be used by the robot controller are computed.

4.1.1 Computation of qref

This section shows the computation of qref using the inverse kinematics module from lab 1. Also, the provided Href (reference homogenous transformations) was used as an input to the inverse kinematics function. The Matlab implementation could be seen in Figure 1.



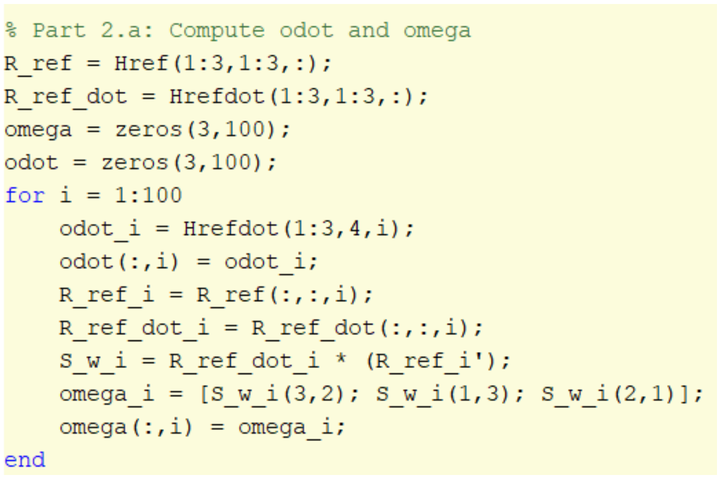
```
%% Experiment Part 4.1
% Computation of joint reference signals

% Part 1: Compute qref
load('Href.mat')
qref_tmp = zeros(100,6);
for i = 1:100
    H = Href(:,:,i);
    qref_tmp(i,:) = inverse(H, myrobot);
end
qref = qref_tmp';
```

Figure 1: Computation of qref

4.1.2 Computation of the derivative of qref

a. Computation of odot and omega

In this part, the linear (odot) and angular (omega) velocities are calculated. To do this, Hrefdot matrix is used which has the derivatives of the homogenous transformations. Odot could directly be extracted from Hrefdot. For omega computation, rotation matrices are extracted from Href and Hrefdot, and omega is found from $S(omega) = \dot{R}_n^o (R_n^o)^T$. The Matlab implementation of this method is shown in Figure 2.

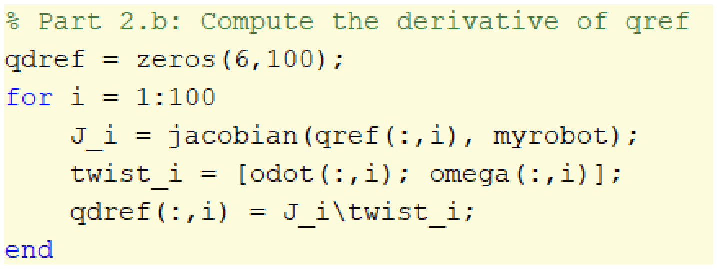


```
% Part 2.a: Compute odot and omega
R_ref = Href(1:3,1:3,:);
R_ref_dot = Hrefdot(1:3,1:3,:);
omega = zeros(3,100);
odot = zeros(3,100);
for i = 1:100
    odot_i = Hrefdot(1:3,4,i);
    odot(:,i) = odot_i;
    R_ref_i = R_ref(:,:,i);
    R_ref_dot_i = R_ref_dot(:,:,i);
    S_w_i = R_ref_dot_i * (R_ref_i');
    omega_i = [S_w_i(3,2); S_w_i(1,3); S_w_i(2,1)];
    omega(:,i) = omega_i;
end
```

Figure 2: Computation of odot and omega

b. Computation of qdref

Using the odot and omega to construct the twist vector, and finding the geometric Jacobian using jacobian.m function that was developed earlier, qdref (joint velocities) were computed using $\dot{q}_{ref} = J^{-1}\, twist$. Implementation in Matlab is seen in Figure 3.

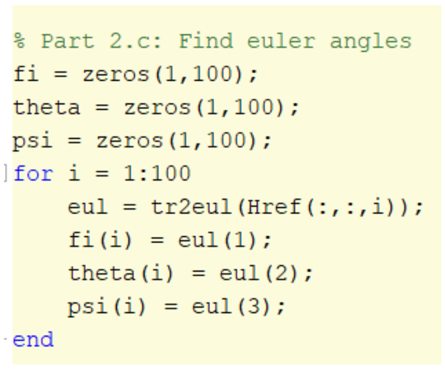


```
% Part 2.b: Compute the derivative of qref
qdref = zeros(6,100);
for i = 1:100
    J_i = jacobian(qref(:,i), myrobot);
    twist_i = [odot(:,i); omega(:,i)];
    qdref(:,i) = J_i\twist_i;
end
```

Figure 3: Computation of qdref

c. Computation of Euler angles

Utilizing the tr2eul() function from Robotics Toolbox, Euler angles were obtained from the Href matrices. The Matlab implementation of this is shown in Figure 4.



```
% Part 2.c: Find euler angles
fi = zeros(1,100);
theta = zeros(1,100);
psi = zeros(1,100);
for i = 1:100
    eul = tr2eul(Href(:,:,i));
    fi(i) = eul(1);
    theta(i) = eul(2);
    psi(i) = eul(3);
end
```

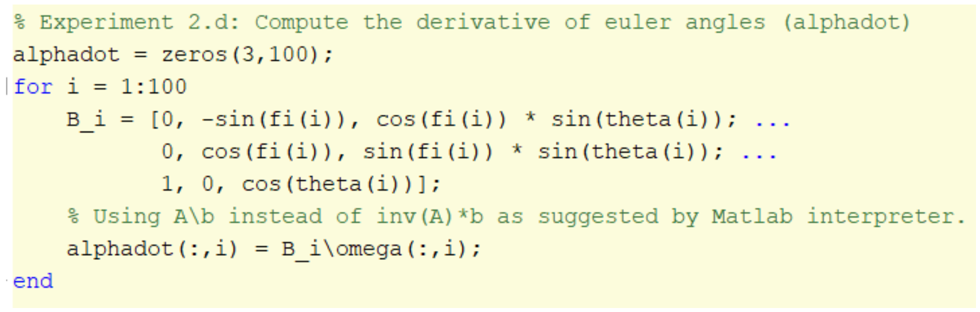
Figure 4: Computation of Euler angles

d. Computation of alphadot

To get the derivatives of the Euler angles (alphadot), the fact that

$$\dot{\alpha} := \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = B(\alpha)^{-1}\omega_n^0, \quad \text{where } B(\alpha) = \begin{bmatrix} 0 & -s_\phi & c_\phi s_\theta \\ 0 & c_\phi & s_\phi s_\theta \\ 1 & 0 & c_\theta \end{bmatrix}$$

was used. All the necessary information (phi, theta, psi, and omega) are already known from previous parts. Implementation of this process in Matlab is shown in Figure 5.



```
% Experiment 2.d: Compute the derivative of euler angles (alphadot)
alphadot = zeros(3,100);
for i = 1:100
    B_i = [0, -sin(fi(i)), cos(fi(i)) * sin(theta(i)); ...
           0, cos(fi(i)), sin(fi(i)) * sin(theta(i)); ...
           1, 0, cos(theta(i))];
    % Using A\b instead of inv(A)*b as suggested by Matlab interpreter.
    alphadot(:,i) = B_i\omega(:,i);
end
```

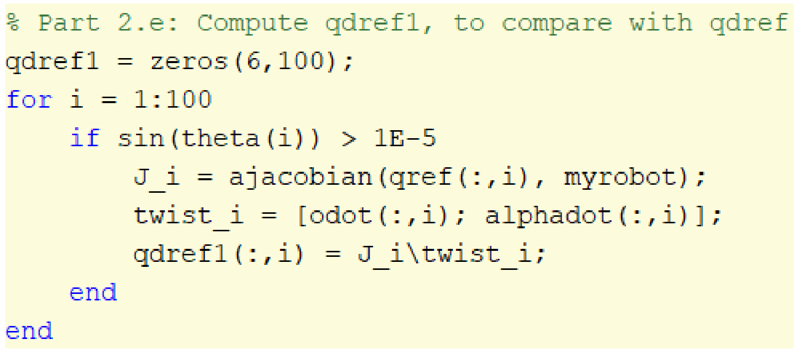
Figure 5: Computation of alphadot

e. Computation of qdref1

To get the derivative of qref (qdref1), the fact that

$$\dot{q}_{\text{ref}} = [J_a(q_{\text{ref}})]^{-1} \begin{bmatrix} \texttt{odot} \\ \texttt{alphadot} \end{bmatrix}$$

was used. All the necessary information (odot, alphadot, analytic Jacobian, qref) are already known from previous parts. One thing to note is that if sin(theta) is close to zero, B matrix is singular. Thus, computation of qdref1 is skipped when sin(theta) is smaller than 10^{-5}. Implementation of this process in Matlab is shown in Figure 6.

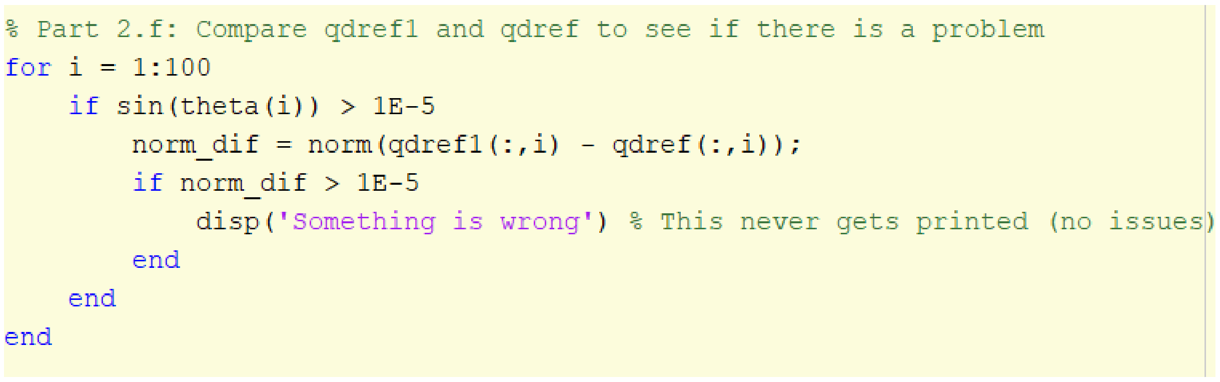


```matlab
% Part 2.e: Compute qdref1, to compare with qdref
qdref1 = zeros(6,100);
for i = 1:100
    if sin(theta(i)) > 1E-5
        J_i = ajacobian(qref(:,i), myrobot);
        twist_i = [odot(:,i); alphadot(:,i)];
        qdref1(:,i) = J_i\twist_i;
    end
end
```

Figure 6: Computation of qdref1

f. Comparison of qdref1 and qdref

In this part qdref1 and qdref that are compared to see if there is any significant difference between them. As expected, results show that there are no issues, and it could be seen by executing the code. The Matlab implementation of comparison is shown in Figure 7.

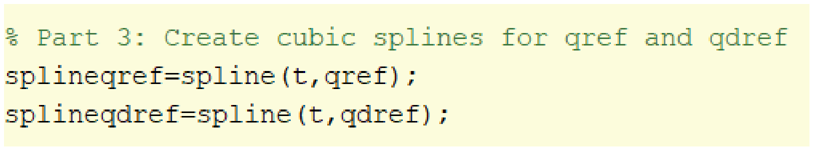


```matlab
% Part 2.f: Compare qdref1 and qdref to see if there is a problem
for i = 1:100
    if sin(theta(i)) > 1E-5
        norm_dif = norm(qdref1(:,i) - qdref(:,i));
        if norm_dif > 1E-5
            disp('Something is wrong') % This never gets printed (no issues)
        end
    end
end
```

Figure 7: Comparison of qdref1 and qdref

4.1.3 Cubic splines for qref and qdref

Now that qref and qdref are known and verified, their corresponding cubic splines could be computed as shown in the handout. The implementation in Matlab could be seen in Figure 8.

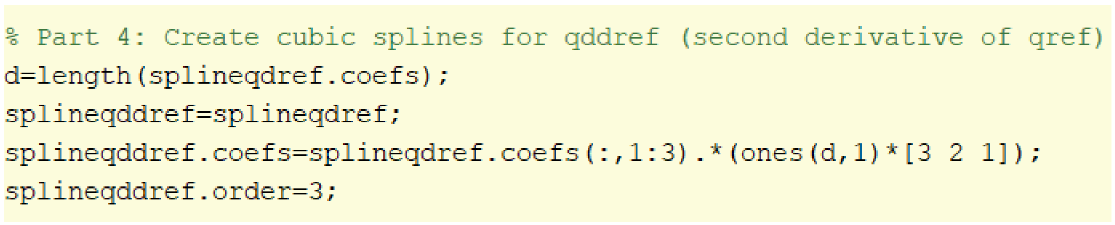


```matlab
% Part 3: Create cubic splines for qref and qdref
splineqref=spline(t,qref);
splineqdref=spline(t,qdref);
```

Figure 8: Cubic splines for qref and qdref

4.1.4 Cubic splines qddref

As described in the handout, the cubic spline for qddref (second derivative of qref) is computed. The implementation in Matlab could be seen in Figure 9.



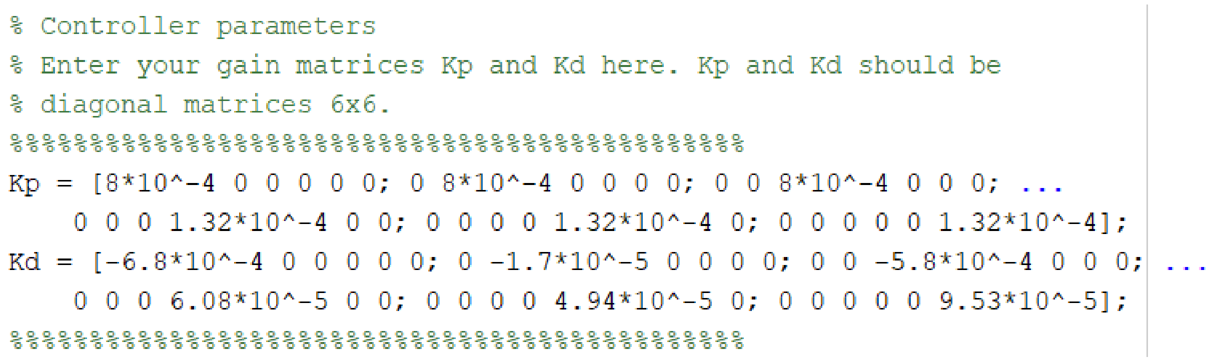
```
% Part 4: Create cubic splines for qddref (second derivative of qref)
d=length(splineqdref.coefs);
splineqddref=splineqdref;
splineqddref.coefs=splineqdref.coefs(:,1:3).*(ones(d,1)*[3 2 1]);
splineqddref.order=3;
```

Figure 9: Cubic splines for qddref

4.2 Testing Independent Joint Controller

4.2.1 Motors.m

Following the results from preparation, following code in Figure 10 was added to the motors.m file to reflect the gain values. Note that these gain values are a result of chosen joint controller design as described in the preparation.

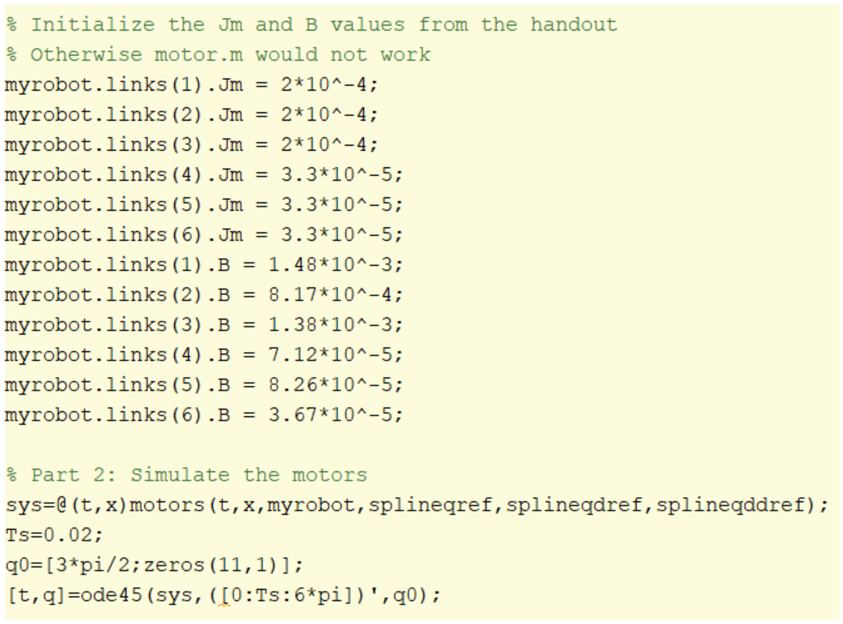


```
% Controller parameters
% Enter your gain matrices Kp and Kd here. Kp and Kd should be
% diagonal matrices 6x6.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Kp = [8*10^-4 0 0 0 0 0; 0 8*10^-4 0 0 0 0; 0 0 8*10^-4 0 0 0; ...
    0 0 0 1.32*10^-4 0 0; 0 0 0 0 1.32*10^-4 0; 0 0 0 0 0 1.32*10^-4];
Kd = [-6.8*10^-4 0 0 0 0 0; 0 -1.7*10^-5 0 0 0 0; 0 0 -5.8*10^-4 0 0 0; ...
    0 0 0 6.08*10^-5 0 0; 0 0 0 0 4.94*10^-5 0; 0 0 0 0 0 9.53*10^-5];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure 10: Gain values in motors.m

4.2.2 Simulation of the motors with joint controller

Using the handout instructions, the motors with the joint controller design was simulated. The implementation in Matlab is shown in Figure 11.

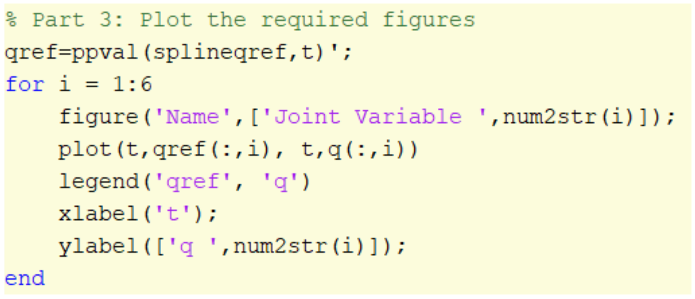


```
% Initialize the Jm and B values from the handout
% Otherwise motor.m would not work
myrobot.links(1).Jm = 2*10^-4;
myrobot.links(2).Jm = 2*10^-4;
myrobot.links(3).Jm = 2*10^-4;
myrobot.links(4).Jm = 3.3*10^-5;
myrobot.links(5).Jm = 3.3*10^-5;
myrobot.links(6).Jm = 3.3*10^-5;
myrobot.links(1).B = 1.48*10^-3;
myrobot.links(2).B = 8.17*10^-4;
myrobot.links(3).B = 1.38*10^-3;
myrobot.links(4).B = 7.12*10^-5;
myrobot.links(5).B = 8.26*10^-5;
myrobot.links(6).B = 3.67*10^-5;

% Part 2: Simulate the motors
sys=@(t,x)motors(t,x,myrobot,splineqref,splineqdref,splineqddref);
Ts=0.02;
q0=[3*pi/2;zeros(11,1)];
[t,q]=ode45(sys,([0:Ts:6*pi])',q0);
```

Figure 11: Simulation of motors

4.2.3 Comparison of joint variables (q) with the reference (qref)

Since the solution for joint variables and a reference trajectory that these variables need to follow is acquired, they could be drawn together. In Figures 13,14,15,16,17, and 18, q and qref elements are compared. It is seen that q values track qref values, meaning that the controller is working. The Matlab code for drawing the plots is shown in Figure 12.



```
% Part 3: Plot the required figures
qref=ppval(splineqref,t)';
for i = 1:6
    figure('Name',['Joint Variable ',num2str(i)]);
    plot(t,qref(:,i), t,q(:,i))
    legend('qref', 'q')
    xlabel('t');
    ylabel(['q ',num2str(i)]);
end
```

Figure 12: Plotting q and qref

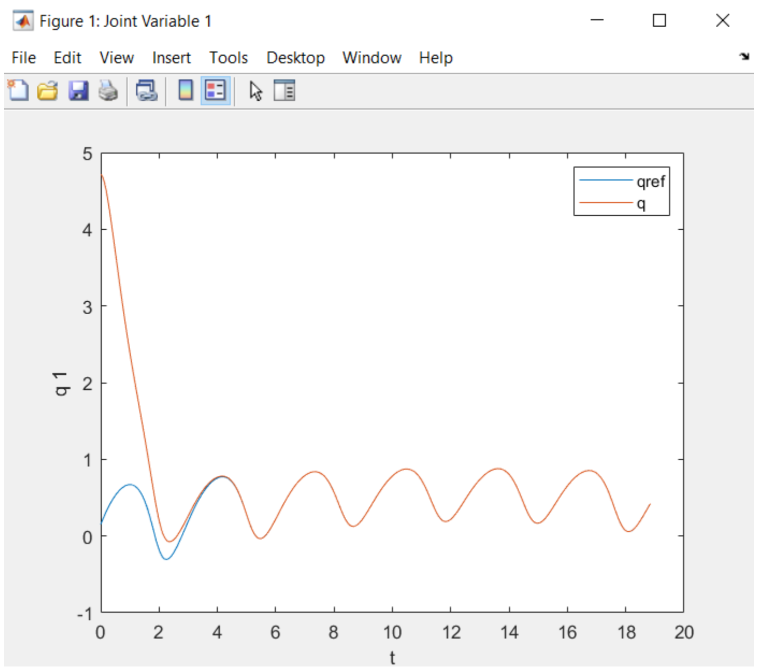


Figure 13: Comparing q(:,1) and qref(:,1)

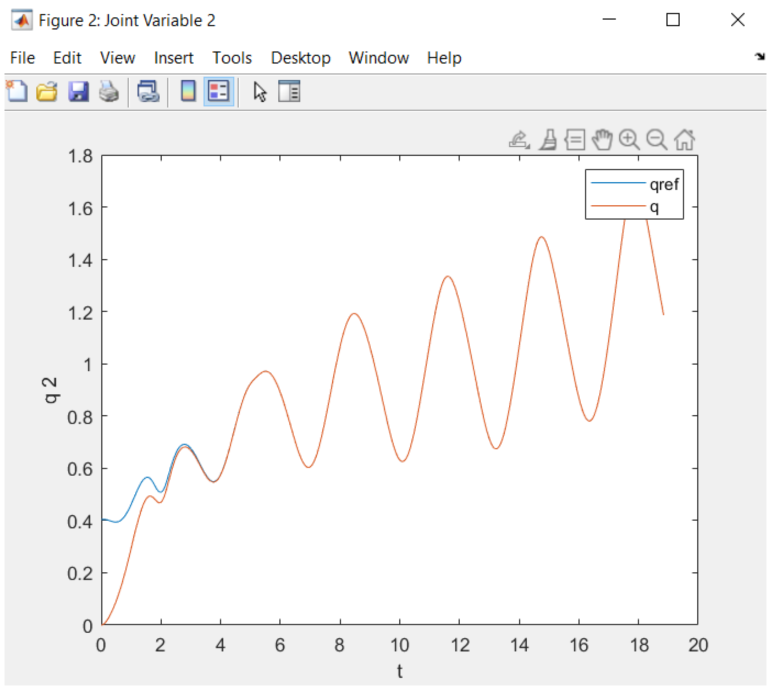


Figure 14: Comparing q(:,2) and qref(:,2)

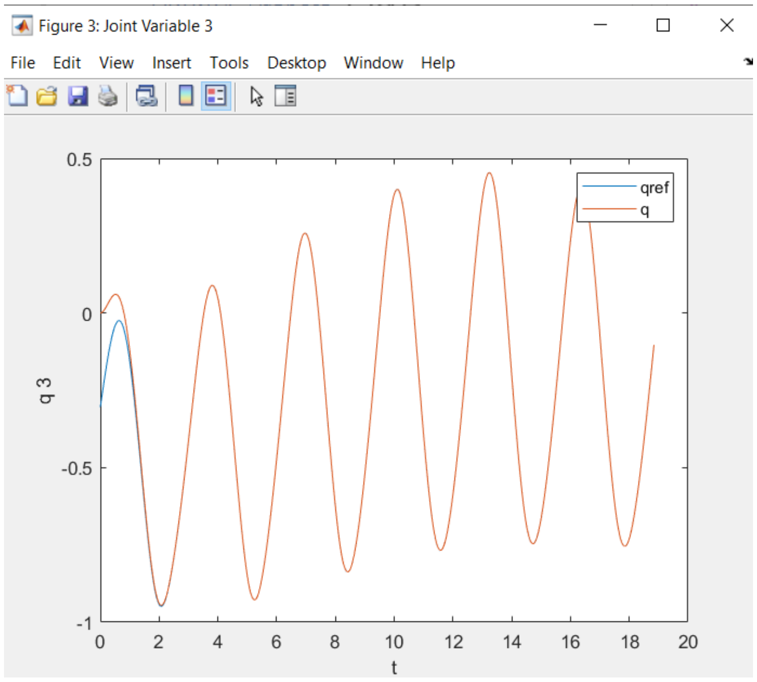


Figure 15: Comparing q(:,3) and qref(:,3)

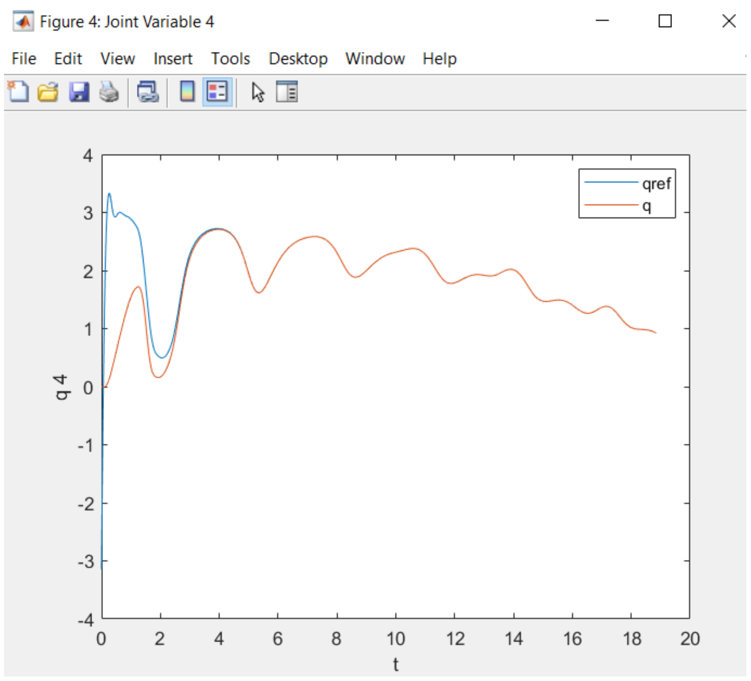


Figure 16: Comparing q(:,4) and qref(:,4)

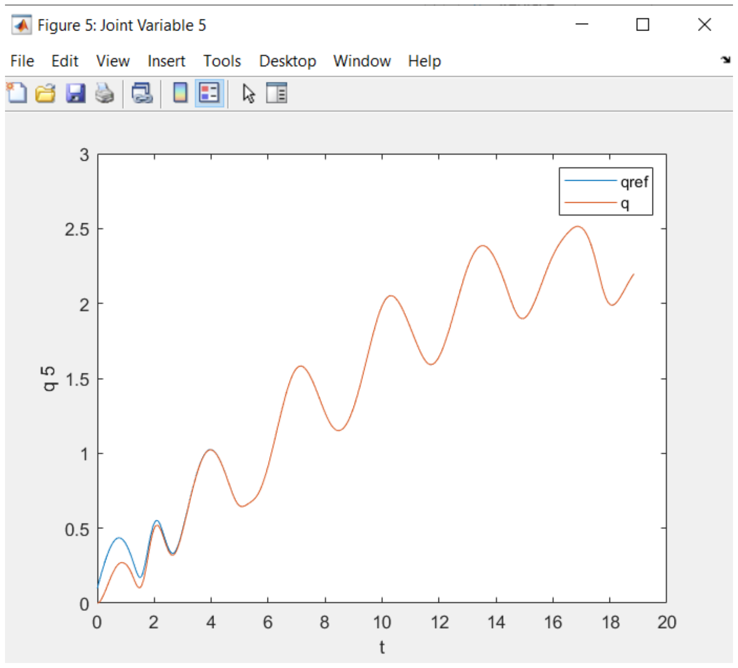


Figure 17: Comparing q(:,5) and qref(:,5)

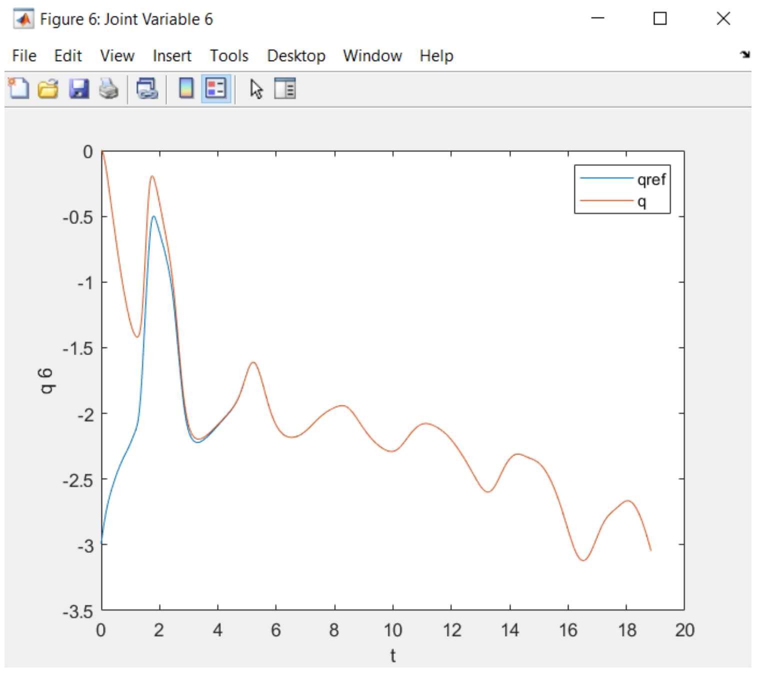


Figure 18: Comparing q(:,6) and qref(:,6)

4.2.4 End effector trajectory

Since all the information about end effector trajectory and desired trajectory is obtained and confirmed in previous parts, the robot motion could be displayed. Using the provided Matlab code in the handout as shown in Figure 19, the reference trajectory was compared with the actual end effector trajectory. It could be observed that the robot follows the reference trajectory in Figure 20. The complete motion of the robot, where the end effector first converges to the start of the desired trajectory and then follows it, could be seen by running the Lab2.m file attached to the submission.

```
% Part 4: Plot the robot movement and the reference trajectory
figure('Name','Robot Movemement and Ref. Trajectory');
hold on
oref=squeeze(Href(1:3,4,:));
plot3(oref(1,:),oref(2,:),oref(3,:),'r')
view(-125,40);
plot(myrobot,q(:,1:6))
hold off
```

Figure 19: Matlab code to plot the end effector trajectory and desired trajectory

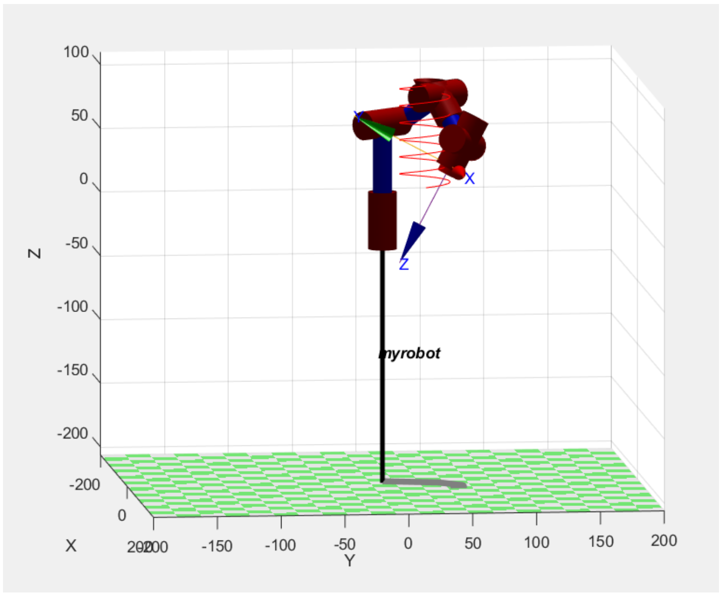


Figure 20: End effector trajectory and desired trajectory