MSc Human and Biological Robotics

Author:
Edward McLaughlin

Lecturer:
Dr. Petar *Kormushev*

Robotics: Tutorial 2 - 2D Dynamics and Control

Imperial College London

Department of Bioengineering

May 5, 2018

1 Question 1: Dynamics

This tutorial concerns the dynamics of a parallel 4-bar system with an end-effector whose desired trajectory is described by $x_d(\omega)$.

$$x_d(\omega) = \begin{bmatrix} 0.273 - 0.2(6\omega^5 - 15\omega^4 + 10\omega^3) \\ 0.273 - 0.1(6\omega^5 - 15\omega^4 + 10\omega^3 \end{bmatrix} \tag{1}$$

In order to determine the required torque to apply at the joints S_L and S_R, the dynamics of the system must be considered. This can be done using Newtonian physics or Lagrange's equation (energy based) - in this instance the latter will be used. Lagrange's equation is given in eqn 2.

$$\tau = \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}}\right) - \frac{\partial L}{\partial q} \tag{2}$$

$$= H\ddot{q} + V(\dot{q}, q) + G(q) \tag{3}$$

where $q = \begin{bmatrix} q_L \\ q_R \end{bmatrix}$ and $L = T - U$; T and U and the parametrised kinetic and potential energy of the system respectively. In this case, as the motion is in the horizontal plane, the effects of gravity can be ignored and hence $U \to 0$, $G(q) \to 0$. Therefore, the lagrangian equation is reduced to its form in eqn 4.

$$\tau = \frac{d}{dt}\left(\frac{\partial T}{\partial \dot{q}}\right) - \frac{\partial T}{\partial q} \tag{4}$$

The kinetic energy of the system is given by:

$$T = \frac{1}{2}\dot{q}^T H \dot{q} \tag{5}$$

$$= \frac{1}{2}\left(\alpha \dot{q_R}^2 + 2\beta \dot{q_R}\dot{q_L} + \alpha \dot{q_L}^2\right) \tag{6}$$

$$H = \begin{bmatrix} \alpha & \beta \\ \beta & \alpha \end{bmatrix} \tag{7}$$

$$\alpha = 2ml_m^2 + ml^2 + 2I \tag{8}$$

$$\beta = 2mll_m \cos(q_R - q_L) \tag{9}$$

Therefore, determining the individual terms in eqn. 4 for q_L and q_R:

$$\frac{\partial T}{\partial \dot{q_L}} = \alpha \dot{q_L} + \beta \dot{q_R} \tag{10}$$

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{q_L}}\right) = \alpha \ddot{q_L} + \beta \ddot{q_R} - 4ml_m^2 \sin(q_R - q_R)(\dot{q_R} - \dot{q_L})\dot{q_R} \tag{11}$$

$$\frac{\partial T}{\partial q_L} = 4\dot{q_L}\dot{q_R}ml_m^2 \sin(q_R - q_L) \tag{12}$$

$$\tau_L = \alpha \ddot{q_L} + \beta \ddot{q_R} - 4ml_m^2 \sin(q_R - q_L)\dot{q_R}^2 \tag{13}$$

similarly:

$$\tau_R = \alpha \ddot{q_R} + \beta \ddot{q_L} + 4ml_m^2 \sin(q_R - q_L)\dot{q_L}^2 \tag{14}$$

Combining eqns. 13 and 14:

$$\tau = \begin{bmatrix} \tau_L \\ \tau_R \end{bmatrix} = \begin{bmatrix} \alpha & \beta \\ \beta & \alpha \end{bmatrix} \begin{bmatrix} \ddot{q_L} \\ \ddot{q_R} \end{bmatrix} + 4ml_m^2 \sin(q_R - q_L) \begin{bmatrix} -\dot{q_R}^2 \\ \dot{q_L}^2 \end{bmatrix} \tag{15}$$

Comparing eqns. 3 and 15:

$$H\ddot{q} = \begin{bmatrix} \alpha & \beta \\ \beta & \alpha \end{bmatrix} \begin{bmatrix} \ddot{q_L} \\ \ddot{q_R} \end{bmatrix} \tag{16}$$

$$V(\dot{q}, q) = 4ml_m^2 \sin(q_R - q_L) \begin{bmatrix} -\dot{q_R}^2 \\ \dot{q_L}^2 \end{bmatrix} \tag{17}$$

$$G(q) = 0 \tag{18}$$

2 Question 2: Control

Initial conditions

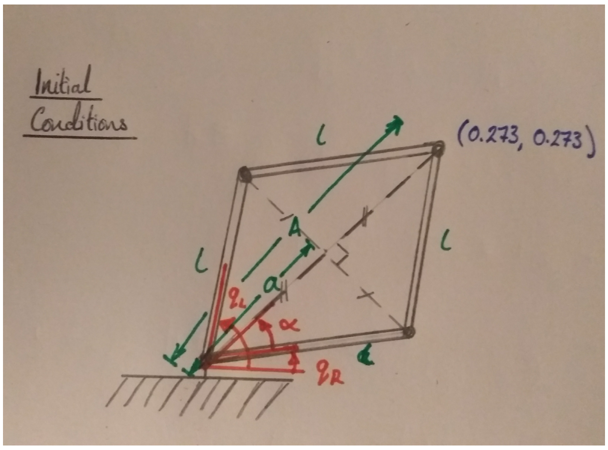

Figure 2.1

Since the initial position has the same x- and y- coordinate, it can be shown that $\frac{q_L+q_R}{2} = \frac{\pi}{4}$.

$$\begin{aligned}\alpha &= \frac{q_L - q_R}{2} \\ a &= l\cos\alpha \\ A &= 2a \\ &= 2l\cos\alpha\end{aligned}$$

From simple trigonometry:

$$\begin{aligned}0.273 &= A\cos(\alpha + q_R) \\ &= 2l\cos(\frac{q_L - q_R}{2})\cos(\frac{q_L + q_R}{2})\end{aligned}$$

Given $l = 0.2m$:

$$\begin{aligned}q_L - q_R &= 2\arccos(\frac{0.273}{0.4\cos(\frac{\pi}{4})}) \\ &= \frac{\pi}{6}\end{aligned}$$

We now know that $\frac{q_L+q_R}{2} = \frac{\pi}{4}$ and $q_L - q_R = \frac{\pi}{6}$. Hence, initially, $q_L = \frac{\pi}{3}$ and $q_R = \frac{\pi}{6}$.

Part A: Feedback controller

Figure 2.2 shows the actual and desired end-effector trajectory in the x-y plane as well as the x- and y- positions over time for the feedback controller. Figure 2.3 shows the actual and desired angles (q_L and q_R) over time. The feedback controller tracks the desired trajectory and angles with reasonable accuracy. The exact results can be seen in table 1 in the Appendix. From these results it can be seen that the final x- and y- positions are 2.2% and 2.1% off the desired final position while the final q_L and q_R angles are 0.5% and 13.4% off the desired final angles respectively. This level of accuracy may be acceptable in low precision tasks, however, high precision tasks such as surgery tools or car assembly lines would require a more accurate performance.

Feedback flow chart

Figure 2.4, shows the implementation of the feedback controller alongside the corresponding lines of Matlab code. Initially, the system's angles are set as calculated in section 2 'Initinal conditions' and the angular velocity and angular acceleration of the system are zero. The loop is initialised from $i = 2$ and the new angle position and velocity are calculated. These new system angles are then fed into the feedback controller and compared with the desired system angles. The feedback torque required is calculated based on proportional and differential gains, where proportional gain is given by $K_P = K$ and $K_D = K\kappa$. The mass matrix is updated with the current actual angles. This torque induces angular acceleration which is computed using the dynamics in Question 1.

Part B: Feed-forward and feed-back controller

Figure 2.5 shows the actual and desired end effector trajectory in the x-y plane as well as the x- and y- positions over time for the feedforward-feedback controller. Figure 2.6 shows the actual and desired angles (q_L and q_R) over time. Again, The exact results can be seen in table 1 in the Appendix. From these results it can be seen that the final x- and y- positions are identical to the desired final position (to 4 s.f.) while the final q_L and q_R angles are 0.005% and 0.2% off the desired final angles respectively. This level of accuracy to within 0.1 degrees and tenths of millimetres would be much more suitable for high precision tasks.

Feedforward-feedback flow chart

Figure 2.7 shows the flow chart of the feedforward and feedback controller. Again, the initial angles are as calculated previously and initial angular velocity and angular acceleration are zero. The iteration loop begins at $i = 2$ and the actual angles and angular velocity are calculated based on the previous angular velocity and angular acceleration. The new values of β, J and $\dot{J}$ are calculated and used to determine the desired feedforward angles in the feedforward controller. These desired feedforward angles are then fed into the feedback controller (as seen in figure 2.4) along with the actual angles in order to determine the angular velocity due to the torque exerted on the system. The new actual angles and angular velocities are once again calculated. This loop continues for the duration of the simulation.

Changing controller gains

$$\begin{aligned} \tau &= K(e + \kappa \dot{e}) && (19) \\ &= Ke + K\kappa\dot{e} && (20) \\ &\equiv K_P e + K_D \dot{e} && (21) \end{aligned}$$

Changing K

Changing the value of K changes both the proportional and derivative gains of the controller since $K_P = K$ and $K_D = K\kappa$. To look at the effect of changing the value of K, the performance of the feedback controller and feedforward and feedback controller for K values of 0.01Nm and 0.001Nm were compared. The results are shown numerically in table 1, and the relative performance is reflected in the % error compared with the desired final position and final angles.

For the feedback controller, this reduction in the controller gain resulted in a much slower and weaker response. This can be seen clearly in figure 2.8 when comparing the actual angles (dashed red) and the desired angles (black) and by examining the error (blue) which both increases in magnitude and shifts to the right. Moreover, the magnitude of the angular velocity to compensate for the error (yellow) decreases in amplitude.

For the feedforward and feedback controller, the change in performance is negligible as can be seen in figure 2.9.

Changing κ

Likewise, the effects of reducing κ to a tenth of its original value were assessed on the two controllers. The results are shown numerically in table 1, and the relative performance is reflected in the % error compared with the desired final position and final angles.

As previously seen for the reduction in K, reducing the value of κ negatively affects the feedback controller's performance. However, in this instance, the increase in error is slightly less and consequently the reduction in angular velocity to compensate for this is also less.

Once more the feedforward and feedback controller manages remarkably well with these changes in controller gains. This shows that the feedforward part of the controller enables it to be very robust, while the feedback part can allow for fine tuning of the motion of the end-effector for more exact precision. The consideration of the system's dynamics in the feedforward part of the controller means that the response isn't completely blind to the current state of the system. This is the case with the feedback controller alone as the only information which it has to act on is the previous state of the controller. As a result, the performance of the feedback controller is heavily reliant on the discrete time-step used in the controller, or more generally, the sampling frequency with which it monitors the state of the system.

Appendix

Graphical Results

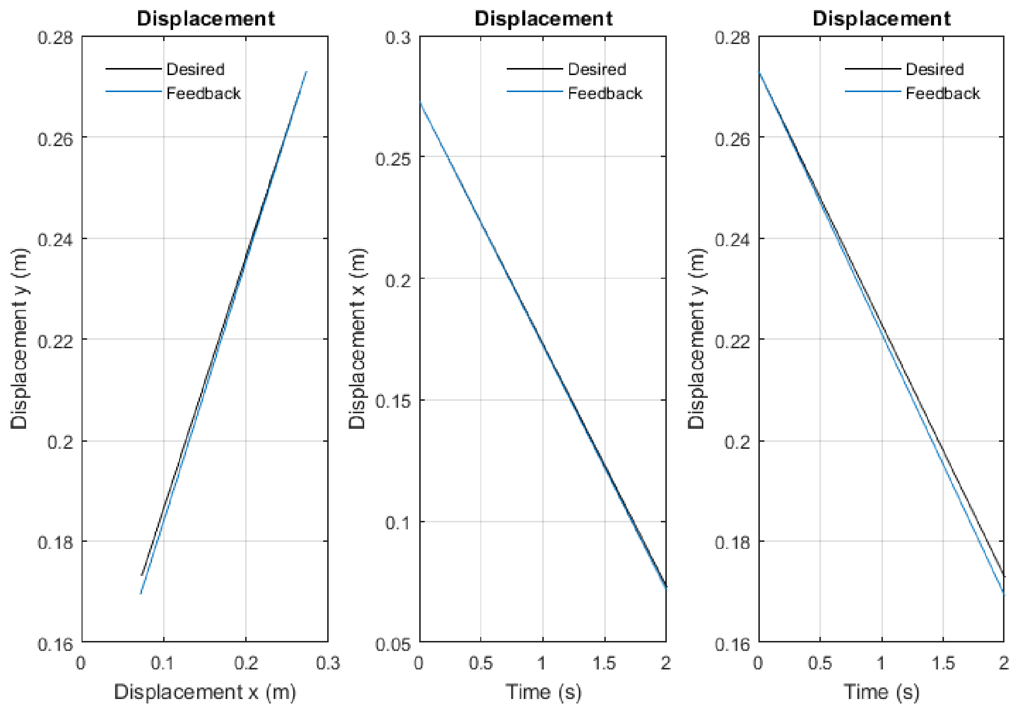

Figure 2.2: Feedback controller angles, errors and $\dot{q}$: $\mathrm{K} = 0.01\mathrm{Nm}$, $\kappa = 100\mathrm{s}$. Left: x-y trajectory, centre: x trajectory, right: y trajectory.

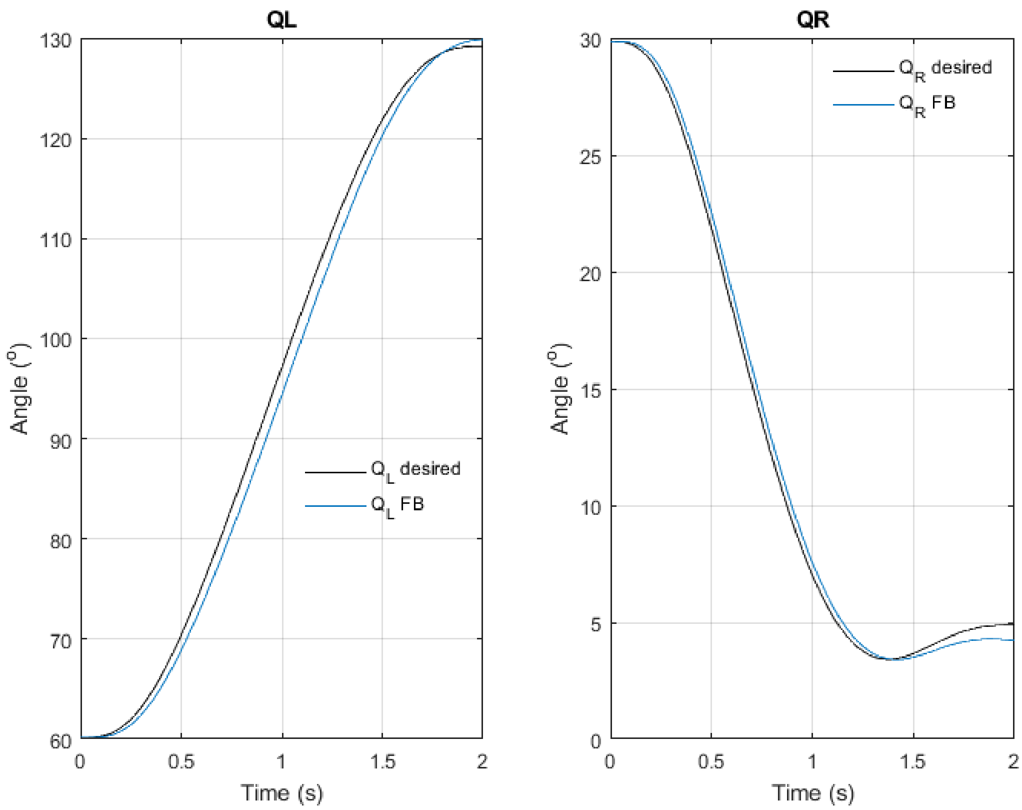

Figure 2.3: Feedback controller angles: K = 0.01Nm, $\kappa = 100$s

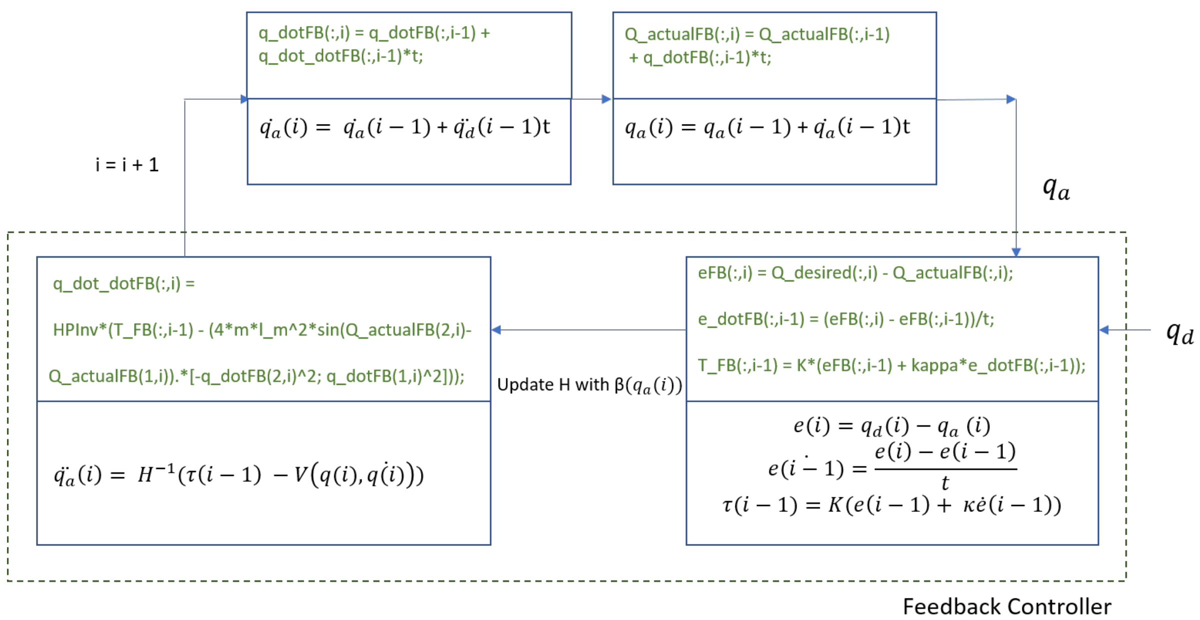

Figure 2.4: Flow chart of the feedback loop with Matlab code

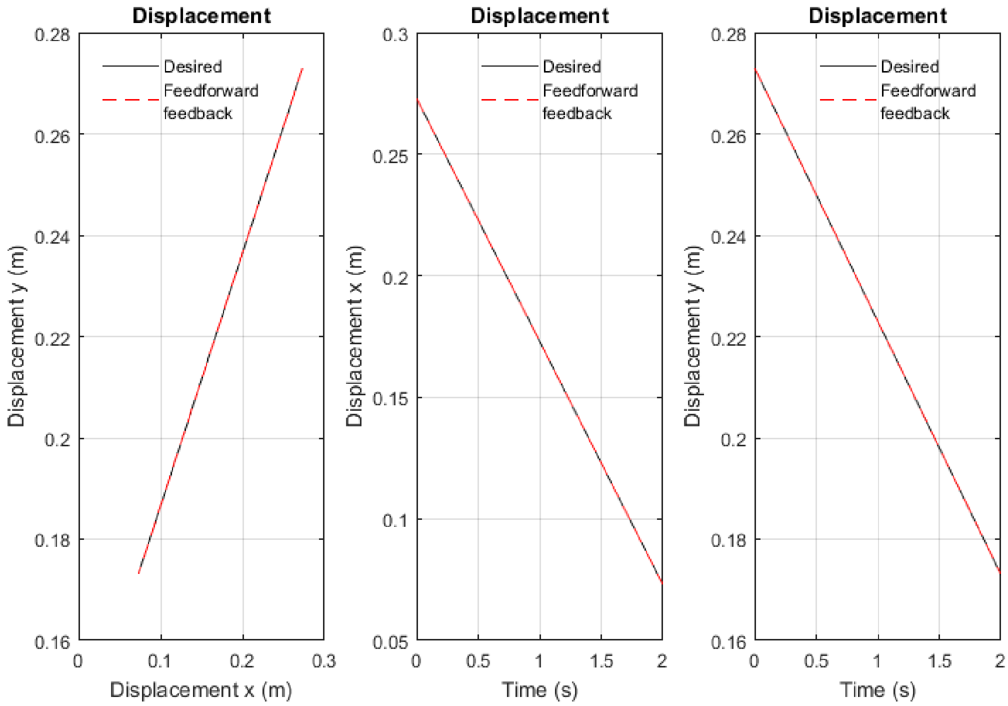

Figure 2.5: Feedforward-feedback controller trajectories: K = 0.01Nm, $\kappa = 100$s. Left: x-y trajectory, centre: x trajectory, right: y trajectory.

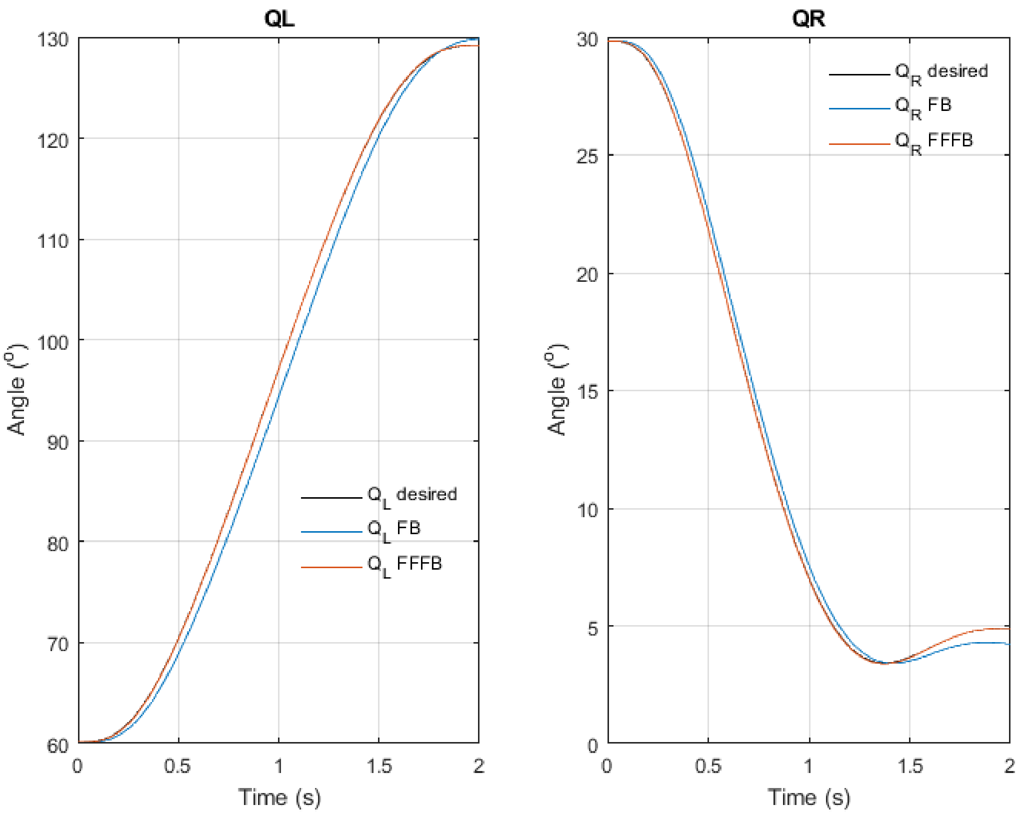

Figure 2.6: Feedforward-feedback controller angles: K = 0.01Nm, $\kappa = 100$s

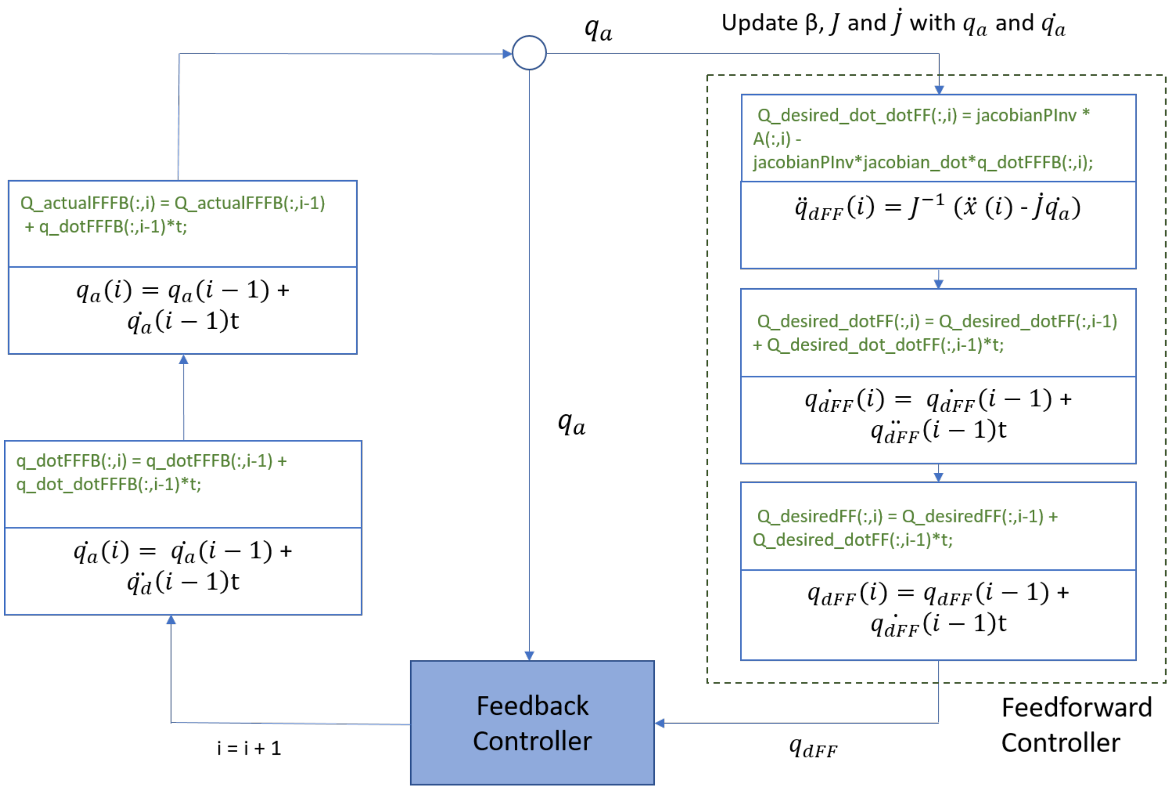

Figure 2.7: Flow chart of the feedforward loop with Matlab code

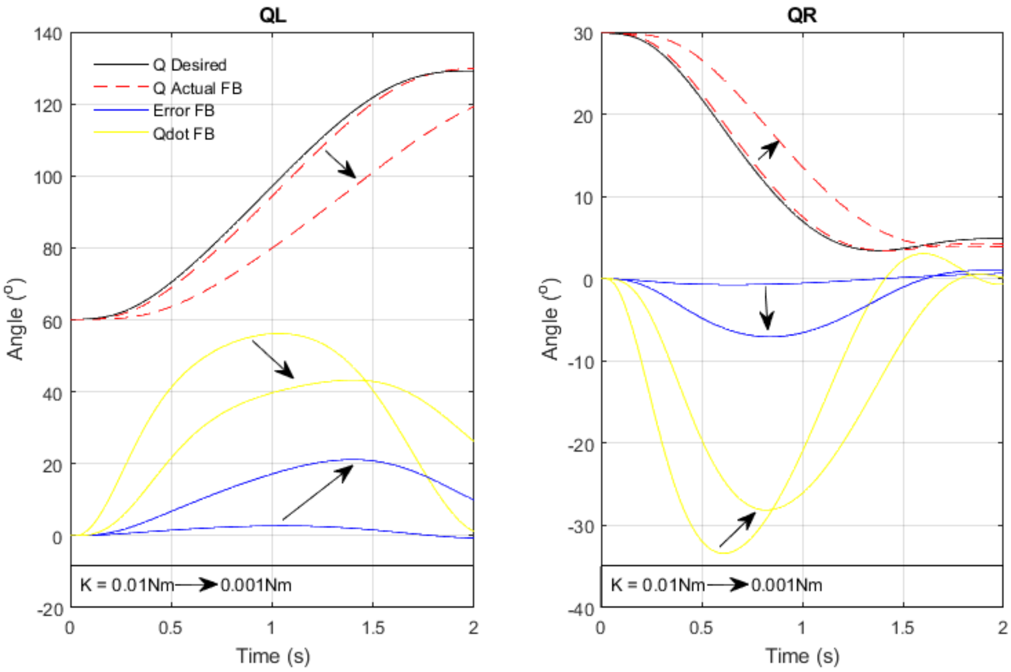

Figure 2.8: Feedback controller angles, errors and $\dot{q}$: K = 0.01Nm $\rightarrow$ 0.001Nm, $\kappa = 100$s

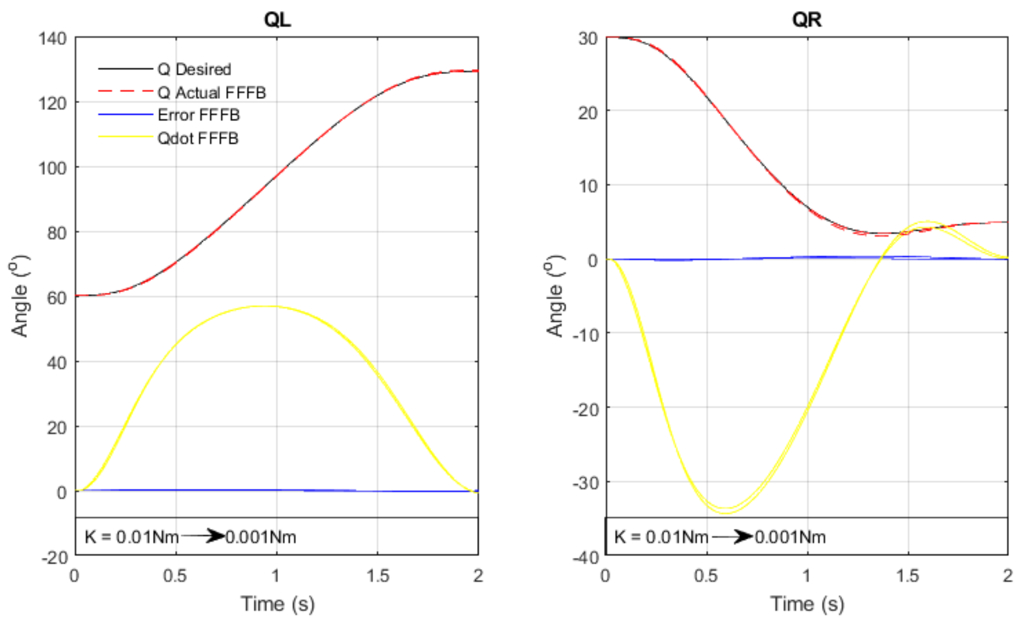

Figure 2.9: Feedforward feedback controller angles, errors and $\dot{q}$: K = 0.01Nm $\rightarrow$ 0.001Nm, $\kappa = 100$s

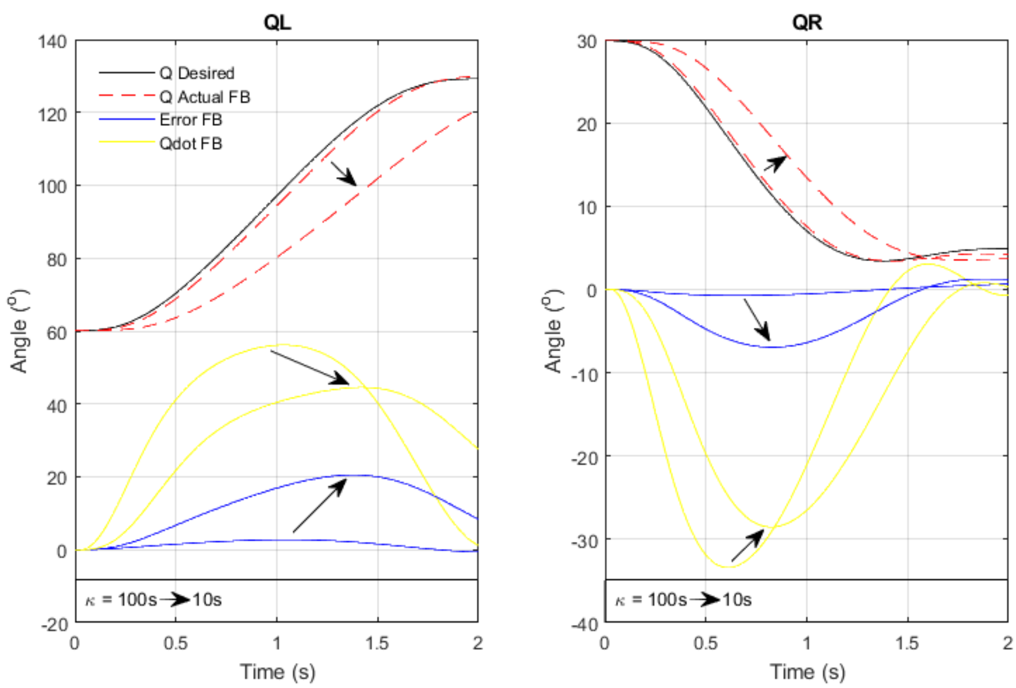

Figure 2.10: Feedback controller angles, errors and $\dot{q}$: K = 0.01Nm, $\kappa = 100$s $\rightarrow$ $\kappa = 10$s

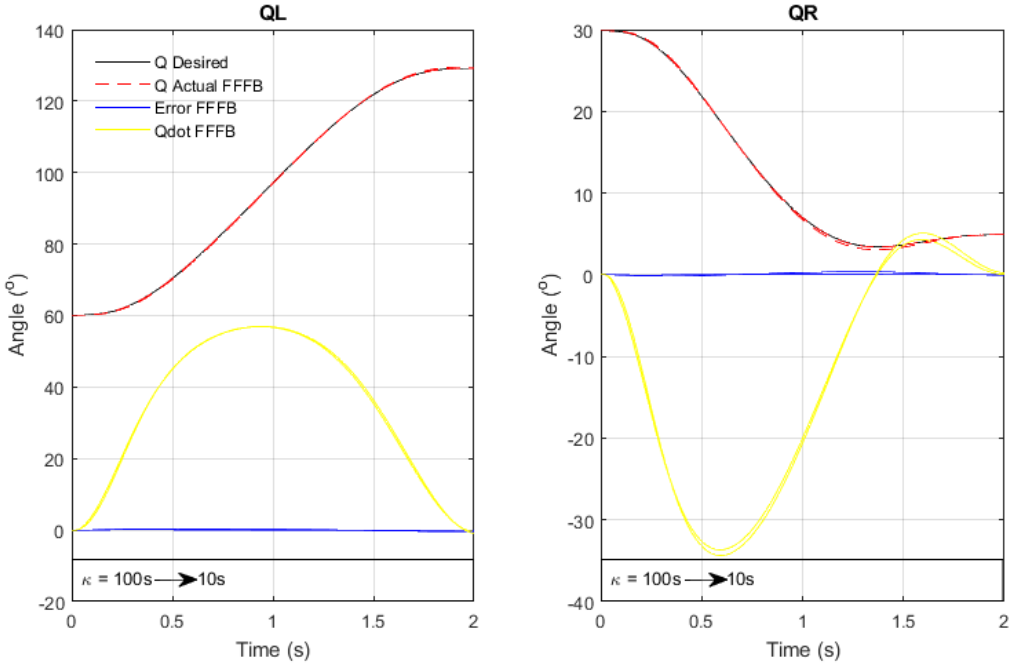

Figure 2.11: Feedforward feedback controller angles, errors and $\dot{q}$: K = 0.01Nm $\rightarrow$ 0.001Nm, $\kappa = 100$s

Table of results

Table 1: Table to show the initial and final positions and angles of the end effector.

	Start X-position (m)	Start Y-position (m)	Final X-position [%error] (m)	Final Y-position [%error] (m)	Final q_L [%error] (^{o})	Final [%error] q_R (^{o})
Desired	0.2730	0.2730	0.0730	0.1730	129.200	4.885
Feedback controller	0.2730	0.2730	0.0714 [2.2]	0.1693 [2.1]	129.852 [0.5]	4.231 [13.4]
K = 0.001Nm	0.2730	0.2730	0.1018 [39.5]	0.1887 [9.1]	119.293 [7.7]	3.898 [20.2]
κ = 10s	0.2730	0.2730	0.0972 [33.2]	0.1853 [7.1]	120.838 [6.5]	3.673 [24.8]
Feedforward + feedback controller	0.2730	0.2730	0.0730 [0]	0.1730 [0]	129.194 [$\simeq$0]	4.894 [0.2]
K = 0.001Nm	0.2730	0.2730	0.0721 [1.2]	0.1724 [0.4]	129.524 [0.3]	4.913 [0.6]
κ = 10s	0.2730	0.2730	0.0721 [1.2]	0.1724 [0.4]	129.536 [0.3]	4.937 [1.1]

MatLab code

```
clc
clear all

%% Preamble and System Spec

%make qL and qR symbolic variables
syms qL qR qL_dot qR_dot

%Assign time variables
T = 2;
t = 0.01;
omega = 0:t/T:1;
time = 0:t:T;

%System Properties
l = 0.2;
l_m = 0.1;
m = 1;
I = 0.01;
K = 0.01;
kappa = 100;
alpha = 2*m*l_m^2 + m*l^2;

%Intitial angle
QR_Initial = 1/2 * asin(0.273^2/l^2 - 1);
QL_Initial = pi/2 - QR_Initial;
Q = [QL_Initial; QR_Initial];
Q_actualFB = [Q, Q, Q];
Q_actualFFFB = Q;
Q_desiredFF = Q;

%Initialising matrices
Q_desired_dotFF = zeros(2,T/t);
Q_desired_dot_dotFF = zeros(2,T/t);
q_dotFB = zeros(2,T/t);
q_dot_dotFB = zeros(2,T/t);
q_dotFFFB = zeros(2,T/t);
q_dot_dotFFFB = zeros(2,T/t);
T_FB = zeros(2,T/t);
T_FFFB = zeros(2,T/t);
```

```
T_FB_FFFB = zeros(2,T/t);
V_actualFB = zeros(2,T/t);
V_actualFFFB = zeros(2,T/t);
eFB = zeros(2,T/t);
eFFFB = zeros(2,T/t);
e_dotFB = zeros(2,T/t);
e_dotFFFB = zeros(2,T/t);

%% Question 2a: Desired Angles

%Calculate position for x, y and overall coordinates
%Plot the position profiles
[X, XTotal] = PositionCalc(omega);

%Find length, x- and y- step in each time interval
X_actualFB = [X(:,1), X(:,1), X(:,1)];
X_actualFFFB = X(:,1);
posInitial(:) = X(:,1);
posFinal(:) = X(:,end);

%calculate velocity profile through integration of position
    profile
[V, VTotal] = VelocityCalc(omega, T);

%calculate acceleration profile through integration of
    velocity profile
[A] = AccCalc(omega, T);

%calculate angle profiles to give the desired velocity and
    position profiles
[Q_desired, Q_desired_dot] = AnglesCalc(Q, V, l, t, T);

%% Question 2a: Feedback Control

for i=2:1:(T/t+1)

    % current angles = old angles + rate of change of angles x
        t
    Q_actualFB(:,i) = Q_actualFB(:,i-1) + q_dotFB(:,i-1)*t;
    q_dotFB(:,i) = q_dotFB(:,i-1) + q_dot_dotFB(:,i-1)*t;

    % current error = current desired - current actual
```

```
    eFB(:,i) = Q_desired(:,i) - Q_actualFB(:,i);
    % old rate of change in error = (current error - old error
        )/t
    e_dotFB(:,i-1) = (eFB(:,i) - eFB(:,i-1))/t;
    % old tau = k x (old error + kappa x old rate of change of
        error
    T_FB(:,i-1) = K*(eFB(:,i-1) + kappa*e_dotFB(:,i-1));

    %update beta with actual FB angles
    beta = updateBeta(Q_actualFB, l, l_m, m, i);

    %update HPInv with actual FB angles
    H = [alpha, beta; beta, alpha];
    HPInv = pinv(H);

    q_dot_dotFB(:,i) = HPInv*(T_FB(:,i-1) - (4*m*l_m^2*sin(
        Q_actualFB(2,i)-Q_actualFB(1,i)).*[-q_dotFB(2,i)^2;
        q_dotFB(1,i)^2]));
    % Update Jacobian with actual angles, calculate the speed
        and position
    % of end effector in x and y
    jacobian_FB = jacobianUpdate(Q_actualFB, l, i);
    V_actualFB(:,i) = jacobian_FB*q_dotFB(:,i);
    X_actualFB(:,i) = X_actualFB(:,i-1) + V_actualFB(:,i-1)*t;

end

%% Question 2b Feedforward + Feedback control

for i=2:1:(T/t+1)

    Q_actualFFFB(:,i) = Q_actualFFFB(:,i-1) + q_dotFFFB(:,i-1)
        *t;
    q_dotFFFB(:,i) = q_dotFFFB(:,i-1) + q_dot_dotFFFB(:,i-1)*t
        ;

    %update inverse Jacobian and jacobian_dot with Q_desired
    jacobian_dot = jacobianDOTUpdate(Q_actualFFFB, q_dotFFFB,
        l, i);
    jacobian = jacobianUpdate(Q_actualFFFB, l, i);
    jacobianPInv = pinv(jacobian);

```

```
    %update beta with desired FF angles
    beta = updateBeta(Q_actualFFFB, l, l_m, m, i);

    %calculate Q desired dot dot for feedforward control
    Q_desired_dot_dotFF(:,i) = jacobianPInv * A(:,i) - jacobianPInv*jacobian_dot*q_dotFFFB(:,i);

    %calculate Q desired dot for feedforward control
    Q_desired_dotFF(:,i) = Q_desired_dotFF(:,i-1) + Q_desired_dot_dotFF(:,i-1)*t;

    %calculate desired Q for feedforward control
    Q_desiredFF(:,i) = Q_desiredFF(:,i-1) + Q_desired_dotFF(:,i-1)*t;

    %calculate T_FF
    tau_L = alpha*Q_desired_dot_dotFF(1,:) + beta*Q_desired_dot_dotFF(2,:) + 4*m*l_m^2*sin(Q_desiredFF(2,i)-Q_desiredFF(1,i)).*-Q_desired_dotFF(2,i)^2;
    tau_R = alpha*Q_desired_dot_dotFF(2,:) + beta*Q_desired_dot_dotFF(1,:) + 4*m*l_m^2*sin(Q_desiredFF(2,i)-Q_desiredFF(1,i)).*(Q_desired_dotFF(1,i)^2);
    T_FF = [tau_L;tau_R];

    %calcutlate new feedback control and combine with feedforward control
    %values
    eFFFB(:,i) = Q_desired(:,i) - Q_actualFFFB(:,i);
    e_dotFFFB(:,i-1) = (eFFFB(:,i) - eFFFB(:,i-1))/t;
    T_FB_FFFB(:,i-1) = K*(eFFFB(:,i-1) + kappa*e_dotFFFB(:,i-1));
    T_FFFB(:,i-1) = T_FB_FFFB(:,i-1) + T_FF(:,i-1);

    %update HPInv with actual FB angles
    H = [alpha, beta; beta, alpha];
    HPInv = pinv(H);

    q_dot_dotFFFB(:,i) = HPInv*(T_FFFB(:,i-1) - (4*m*l_m^2*sin(Q_actualFFFB(2,i)-Q_actualFFFB(1,i)).*[-q_dotFFFB(2,i)^2; q_dotFFFB(1,i)^2]));

```

```
    % Update Jacobian with actual angles, calculate the speed and position
    % of end effector in x and y
    jacobian_FFFB = jacobianUpdate(Q_actualFFFB, 1, i);
    V_actualFFFB(:,1)=[0;0];
    V_actualFFFB(:,i) = jacobian_FFFB*q_dotFFFB(:,i);
    X_actualFFFB(:,i) = X_actualFFFB(:,i-1) + V_actualFFFB(:,i-1)*t;

end

%% Plot Graphs
plotGraphs(X, X_actualFB, X_actualFFFB, Q_desired, Q_actualFB, Q_actualFFFB, time, eFB, q_dotFB, eFFFB, q_dotFFFB, t, T)

%% Functions

% Calculate position trajectory
function [X, XTotal] = PositionCalc(omega)

    X = [0.273 - 0.2*(6*omega(1,:).^5 - 15*omega(1,:).^4 + 10*omega(1,:).^3);
         0.273 - 0.1*(6*omega(1,:).^5 - 15*omega(1,:).^4 + 10*omega(1,:).^3)];
    XTotal = sqrt(X(1,:).^2 + X(2,:).^2);

end

% Calculate velocity trajectory
function [V, VTotal] = VelocityCalc(omega, T)

    V = [-0.2*(30*omega(1,:).^4 - 60*omega(1,:).^3 + 30*omega(1,:).^2)/T;
         -0.1*(30*omega(1,:).^4 - 60*omega(1,:).^3 + 30*omega(1,:).^2)/T];
    VTotal = sqrt(V(1,:).^2 + V(2,:).^2);

end

% Calculate acceleration trajectory
function [A] = AccCalc(omega, T)

```

```
    A = [-0.2*(120*omega(1,:).^3 - 180*omega(1,:).^2 + 60*
        omega(1,:))/T^2;
         -0.1*(120*omega(1,:).^3 - 180*omega(1,:).^2 + 60*
            omega(1,:))/T^2];

end

% Calculate angles to achieve required position trajectory
function [Q, Q_dot] = AnglesCalc (Q, V, l, t, T)

    for i=1:1:(T/t)

        %update inverse Jacobian with new angle
        jacobian = jacobianUpdate(Q, l, i);
        jacobianPInv = pinv(jacobian);

        %calculate Q dot
        Q_dot(:,i) = jacobianPInv * V(:,i);

        %calculate new angles
        Q(:,i+1) = Q(:,i) + Q_dot(:,i)*t;
    end

end

% Update jacobian matrix with current angles
function [jacobian] = jacobianUpdate(Q, l, i)

    jacobian = [-l*sin(Q(1,i)), -l*sin(Q(2,i));
                 l*cos(Q(1,i)),  l*cos(Q(2,i))];

end

% Update jacobian dot with current angles and angular
    velocities
function [jacobian_dot] = jacobianDOTUpdate(Q, Q_dot, l, i)

    jacobian_dot = [-l*cos(Q(1,i))*Q_dot(1,i), -l*cos(Q(2,i))*
        Q_dot(2,i);
                    -l*sin(Q(1,i))*Q_dot(1,i), -l*sin(Q(2,i))*
                        Q_dot(2,i)];

```

```
end

% Update beta with current angles
function [beta] = updateBeta(Q, l, l_m, m, i)

    beta = 2*m*l*l_m*cos(Q(2,i) - Q(1,i));
end
```