Contents


```
% Jacob Sayono
% 505368811

% MAE C163B
% Project 2
```

Part 1:

Forward Kinematics Inverse Kinematics Matlab Robotics Toolbox

```
close; clear; clc;

% lengths (m)
a1 = 0.325;
a2 = 0.225;
d1 = 0.416;
d4 = 0.093;

% key positions
home = [0, -1, 0, .325;
        1, 0, 0, .225;
        0, 0, 1, .203;
        0, 0, 0, 1];
feeder = [0, -1, 0, .325;
          1, 0, 0, .225;
          0, 0, 1, .180;
          0, 0, 0, 1];
goal1 = [1, 0, 0, .280;
         0, 1, 0, .240;
         0, 0, 1, .180;
         0, 0, 0, 1];
goal2 = [0, -1, 0, .280;
         1, 0, 0, .330;
         0, 0, 1, .180;
         0, 0, 0, 1];
goal3 = [-1, 0, 0, .370;
         -1, 0, 0, .240;
          0, 0, 1, .180;
          0, 0, 0, 1];
goal4 = [0, 1, 0, .370;
        -1, 0, 0, .240;
         0, 0, 1, .180;
         0, 0, 0, 1];

% via points
a = [0, -1, 0, .325;
     1, 0, 0, .225;
     0, 0, 1, .190;
     0, 0, 0, 1];
b = [1, 0, 0, .280;
     0, 1, 0, .240;
     0, 0, 1, .190;
     0, 0, 0, 1];
c = [0, -1, 0, .280;
     1, 0, 0, .330;
     0, 0, 1, .190;
     0, 0, 0, 1];
d = [-1, 0, 0, .370;
     0, -1, 0, .330;
     0, 0, 1, .190;
     0, 0, 0, 1];
e = [0, 1, 0, .370;
     -1, 0, 0, .240;
     0, 0, 1, .190;
     0, 0, 0, 1];
```

```
% test forward kinematics
T = fk(0, 0, .100, 0);

% test inverse kinematics
[~, t2, d3, t4] = ik(a);

% visualize robot
L1 = Revolute('alpha', 0, 'a', 0, 'd', d1, 'qlim', [-170 170]*pi/180);
L2 = Revolute('alpha', 0, 'a', a1, 'd', 0, 'qlim', [-145 145]*pi/180);
L3 = Prismatic('alpha', pi, 'a', a2, 'theta', 0, 'qlim', [0 .15]);
L4 = Revolute('alpha', 0, 'a', 0, 'd', d4, 'qlim', [-360 360]*pi/180);
tool = transpose([0, 0, 0]);
PCB_Robot = SerialLink([L1 L2 L3 L4], 'name', 'PCB_Robot', 'tool', tool);

% now convert waypoints into joint space values and interpolate
trajectory_sequence = [a', a, b, b', b, a, a', a, c, c', c, a, a', a, d, d', d, a, a', a, e, e', e];
time_between_via = [4 0.2 0.2 4 0.2 0.2 4 0.2 0.2 4 0.2 0.2 4 0.2 0.2 4 0.2 0.2 4 0.2 0.2 4 0.2 0.2 4 0.2 0.2 4 0.2 0.2 4];
```

Part 2:

Joint Space Trajectory Generation (Linear w/ Parabolic Blend) Joing Angle, Velocity, Acceleration Simulate Robot and Animate Trajectory

```
% convert via points to joint space with elbow up
vp_js = zeros(4,4,6);
for i = 1:6
    vp_js(:,i+1) = IK(zeros(4,4,i));
end

% initial position and final position of end effector
vp_js(:,1) = [0 pi/2 0 -140];  % initial
vp_js(:,26) = [0 pi/2 0 -140];  % final

% time component between via points
t1 = 0;
for i = 1:6
    [q,qd,qdd,t]=traj_linear_w_parabolic_blend_vector(t1, t1+time_between_via(i), JS_via(:,i), JS_via(:,i+1), des_qdd, K*time_between_via(i)) ;
     if(i<25)
         q(:,end)=[];
         qd(:,end)=[];
         qdd(:,end)=[];
         t(end)=[];
     end
    theta1=[q_all q];
    theta2=[qd_all qd];
    theta3=[qdd_all qdd];
    if i>1
           sum = sum + time_between_via(i-1);
    end
    t = t + sum;
end

% joint acceleration, velocity, position vs time
q_all(1:3,:) = (180/pi)*q_all(1:3,:);
qd_all(1:3,:) = (180/pi)*qd_all(1:3,:);
qdd_all(1:3,:) = (180/pi)*qdd_all(1:3,:);

n = 4 * 9 * 0.2 * 16 * K - 24;

% plot graphs for theta1, theta2, theta4, and d3
titles = {'Joint Angle vs Time', 'Joint Speed vs Time', 'Joint Acceleration vs Time', ...
          'Joint Position vs Time', 'Joint Speed vs Time', 'Joint Acceleration vs Time'};
ylabels = {'Joint Angle [degree]', 'Joint Speed [degree/s]', 'Joint Acceleration [degree/s^2]', ...
           'Joint Position [mm]', 'Joint Speed [mm/s]', 'Joint Acceleration [mm/s^2]'};
data = {q_all, qd_all, qdd_all, q_all, qd_all, qdd_all};
indices = {[1, 2, 3], [1, 2, 3], [1, 2, 3], 4, 4, 4};

for i = 1:6
    figure()
    plot(t_all(1:N), data{i}(indices{i},1:N));
    legend('q1', 'q2', 'q3','d4','location', 'best');
    title(titles{i});
    xlabel('Time (s)'); ylabel(ylabels{i});
    xlim([0 t_all(N)]);
    grid on;
    hold off
```

```
end

% extract via points into simulation
viapoints=zeros(3,26);
q1=[0, pi/2, 0, -140];
T_initial_final = RH3FRH55.fkine(q1);
viapoints(:,1)=T_initial_final.t;
viapoints(:,26)=T_initial_final.t;

for i=1:24
    viapoints(:,i+1)=T_O_all_v(1:3,4,i);
end

% Extract xyz positions of each intermediary point
xyz_jointTraj = zeros(3,length(q_all));
for i = 1:length(q_all)
    T_all_points = RH3FRH55.fkine(q_all(:,i));
    xyz_jointTraj(:,i) = T_all_points.t;
end

% Plot robot with viapoints and xyz trajectories
figure()
PCB_Robot.plot(q1, 'jointdiam', 1.5,'workspace',[-1000,1000,-1000,1000,0,1000]);
hold on;
plot3(viapoints(1,:), viapoints(2,:), viapoints(3,:), 'ro', 'LineWidth', 2); hold on;
plot3(xyz_jointTraj(1,:), xyz_jointTraj(2,:), xyz_jointTraj(3,:), 'b.-');
grid on;
```

Function Definitions

```
% forward kinematics function
function T = fk(theta1, theta2, d3, theta4)
    a1 = 0.325;
    a2 = 0.225;
    d1 = 0.416;
    d4 = 0.093;

    T01 = mat_from_DH(0, 0, d1, theta1);
    T12 = mat_from_DH(0, a1, 0, theta2 + 90);
    T23 = mat_from_DH(0, a2, -d3, 0);
    T34 = mat_from_DH(0, 0, -d4, theta4);

    T = T01*T12*T23*T34;
end

% matrix function given DH parameters
function matrix = mat_from_DH(alpha_iminus1, a_iminus1, d_i, theta_i)
    matrix = [cosd(theta_i) -sind(theta_i) 0 a_iminus1;
              sind(theta_i)*cosd(alpha_iminus1) cosd(theta_i)*cosd(alpha_iminus1) -sind(alpha_iminus1) -sind(alpha_iminus1)*d_i;
              sind(theta_i)*sind(alpha_iminus1) cosd(theta_i)*sind(alpha_iminus1) cosd(alpha_iminus1) cosd(alpha_iminus1)*d_i;
              0 0 0 1];
end

% inverse kinematics function
function [theta1, theta2, d3, theta4] = ik(T)
    [x, y, z, theta] = matrix_to_position(T);

    a1 = 0.325;
    a2 = 0.225;
    d3_val = .323-z;

    c_theta_2 = (x^2+y^2-a1^2-a2^2)/(2*a1*a2);
    s_theta_2_pos = sqrt(1-c_theta_2^2);
    s_theta_2_neg = -sqrt(1-c_theta_2^2);

    theta_2_val1 = atan2(s_theta_2_pos,c_theta_2);
    theta_2_val2 = atan2(s_theta_2_neg,c_theta_2);

    L3_1 = a1+cos(theta_2_val1)*a2;
    L3_2 = a1+cos(theta_2_val2)*a2;
    L4_1 = sin(theta_2_val1)*a2;
    L4_2 = sin(theta_2_val2)*a2;

    theta_1_val1 = atan2(y,x) - atan2(L4_1,L3_1);
    theta_1_val2 = atan2(y,x) - atan2(L4_2,L3_2);
```

```matlab
    theta_4_val1 = theta - pi/4 - theta_1_val1 - theta_2_val1;
    theta_4_val2 = theta - pi/4 - theta_1_val2 - theta_2_val2;

    if (d3_val < 0 || d3_val > 0.150)
        d3 = -1;
    else
        d3 = d3_val;
    end

    if (check_angles(theta_1_val1, theta_2_val1, theta_4_val1) == 1)
        theta1 = theta_1_val1;
        theta2 = theta_2_val1;
        theta4 = theta_4_val1;
        return
    elseif (check_angles(theta_1_val2, theta_2_val2, theta_4_val2) == 1)
        theta1 = theta_1_val2;
        theta2 = theta_2_val2;
        theta4 = theta_4_val2;
        return
    else
        theta1 = -1;
        theta2 = -1;
        theta4 = -1;
        return
    end
end

% check angles function
function check = check_angles(t1, t2, t4)
    check = -1;
    if (t1 < -170 || t1 > 170)
        return
    elseif (t2 < -145 || t2 > 145)
        return
    elseif (t4 < -360 || t4 > 360)
        return
    else
        check = 1;
        return
    end
end

% euler angles function
function [x, y, z, theta] = matrix_to_position(T)
    x = T(1,4);
    y = T(2,4);
    z = T(3,4);
    euler_angles = rotm2eul(T(1:3,1:3));
    theta = euler_angles(1);
end

% trajectory generation (provided by TA)
function [q, qd, qdd, t] = traj_linear_w_parabolic_blend_vector(t1, t2, q1, q2, des_qdd, n_intervals)
    q = zeros(length(q1), n_intervals); qd = q; qdd = q; t = q;
    for i = 1:length(q1)
        [q(i,:), qd(i,:), qdd(i,:), t] = traj_linear_w_parabolic_blend_scalar(t1, t2, q1(i), q2(i), des_qdd(i), n_intervals);
    end
end

function [q, qd, qdd, t] = traj_linear_w_parabolic_blend_scalar(t1, t2, q1, q2, ~, n_intervals)
    t = linspace(t1, t2, n_intervals);
    des_qdd = 4*(abs(q1-q2))/(t2-t1)^2+0.0001;
    tb = 0.5*(t2-t1) - 0.5*sqrt(des_qdd^2*(t2-t1)^2 - 4*abs(des_qdd)*abs(q2-q1))/abs(des_qdd);

    if q2 > q1
        % First parabolic region
        constraints = [q1; 0; des_qdd];
        relationships = [1, t1, t1^2;
                         0,  1, 2*t1;
                         0,  0,    2];
        ab1 = (relationships\constraints)';

        % Second parabolic region
        constraints = [q2; 0; -des_qdd];
        relationships = [1, t2, t2^2;
```

```
                        0,  1, 2*t2;
                        0,  0,    2];
        ab2 = (relationships\constraints)';
    else
        % First parabolic region
        constraints = [q1; 0; -des_qdd];
        relationships = [1, t1, t1^2;
                        0,  1, 2*t1;
                        0,  0,    2];
        ab1 = (relationships\constraints)';

        % Second parabolic region
        constraints = [q2; 0;  des_qdd];
        relationships = [1, t2, t2^2;
                        0,  1, 2*t2;
                        0,  0,    2];
        ab2 = (relationships\constraints)';
    end

    % Linear region
    q1b = ab1(1) + ab1(2)*(t1+tb) + ab1(3)*(t1+tb)^2;
    q2b = ab2(1) + ab2(2)*(t2-tb) + ab2(3)*(t2-tb)^2;
    constraints =   [q1b; q2b];
    relationships = [1 tb+t1; 1 t2-tb];
    al = (relationships\constraints)';

    % Outputs
    t11 = t((t1<=t) & (t<=t1+tb)); % first parabolic region
        a0 = ab1(1); a1 = ab1(2); a2 = ab1(3);
        q   = a0 + a1*t11 + a2*t11.^2;
        qd  =      a1   + 2*a2*t11;
        qdd =             2*a2*ones(size(t11));

    t22 = t((t1+tb<t) & (t<t2-tb)); % linear region
        a0 = al(1); a1 = al(2);
        q   = [q, a0 + a1*t22];
        qd  = [qd, a1.*ones(size(t22))];
        qdd = [qdd, zeros(size(t22))];

    t33 = t((t2-tb<=t) & (t<=t2)); % second parabolic region
        a0 = ab2(1); a1 = ab2(2); a2 = ab2(3);
        q   = [q, a0 + a1*t33 + a2*t33.^2];
        qd  = [qd,     a1   + 2*a2*t33];
        qdd = [qdd,           2*a2*ones(size(t33))];
end
```

Published with MATLAB® R2022b