Imperial College of London

DE3-ROB1 ROBOTICS 1

Tutorial 1: Kinematics of a 3-DOF Robot

Marcin Laskowski

supervised by
Dr Petar Kormushev

October 16, 2017

1 Abstract

During the first Tutorial classes it was given a 3-degree-of-freedom parallel robot. In Fig.1 is illustrated the 3D structure

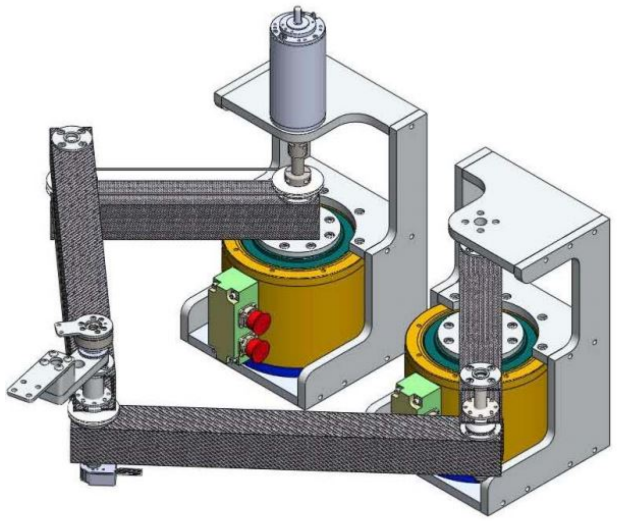
Figure 1: Picture of the 3-dof parallel robot

In the Fig.2 it is shown the Diagram of the robot. On the basis of that structure it was important to calculate the Forward Kinematics and Redundancy resolution. All graphs and more complicated calculations were perform in the MatLab Software.

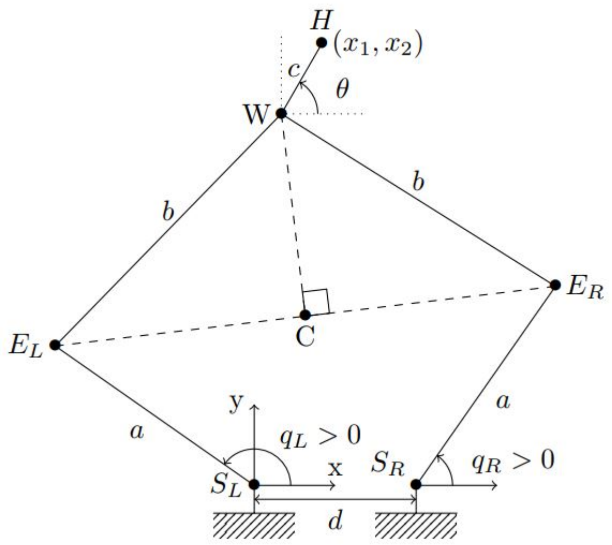

Figure 2: Diagram of the 3-dof parallel robot

2 Direct kinematics

The first task concerned determination of the Direct Kinematics. On the basis of the structure it was necessary to compute the position of H point as a function of the two motors angles q_L, q_R, the angle , and the parameters a, b, c, d. S_L was treated as the coordinate origin.

The calculations were as follows.

Listing 1: Matlab code for the task 1

```
% ROBTOICS - tutorial 1

clc
clear all

% a = 20;
% b = 30;
% c = 5;
% d = 10;
% q1 = deg2rad(150);
% q2 = deg2rad(45);
% q3 = deg2rad(45);

syms q1 q2 q3 a b c d
%% Q1 // Forward Kinematics

SL = [0;0];      % point SL
SR = [d;0];
EL = [a*cos(q1); a*sin(q1)]; % point EL
ER = [d + a*cos(q2); a*sin(q2)]; % point ER

EL_ER = [(-1*EL(1)+ER(1)); (-1*EL(2)+ER(2))]; % vector EL_ER
mag_EL_ER = sqrt((EL_ER(1))^2+(EL_ER(2))^2); % magnitude of the vector EL_ER
unit_EL_ER = EL_ER / mag_EL_ER; % unit vector EL_ER

EL_C = EL_ER / 2; % vector EL_C
mag_EL_C = sqrt((EL_C(1))^2+(EL_C(2))^2); % magnitude of the vector EL_C
C = SL + EL + EL_C;      % point C

mag_C_W = sqrt(b^2 - mag_EL_C^2); % magnitude of the vector C_w
unit_C_W = [-1 * unit_EL_ER(2); unit_EL_ER(1)]; % unit vector C_W
C_W = mag_C_W * unit_C_W; % vector C_W
W = C + C_W; % point W

W_H = [c*cos(q3); c*sin(q3)]; % vector W_H

H = W + W_H;
% simplify(H);
% pretty(H) % point H

%%%% P L O T S %%%%%
% figure(1)
% plot([SL(1),EL(1)],[SL(2),EL(2)]); hold on;
% plot([SL(1),SR(1)],[SL(2),SR(2)]); hold on;
% plot([SR(1),ER(1)],[SR(2),ER(2)]); hold on;
% plot([ER(1), W(1)], [ER(2), W(2)]); hold on;
% plot([EL(1), W(1)], [EL(2), W(2)]); hold on;
% plot([W(1), H(1)], [W(2), H(2)]); hold on;
% xlabel('x axis [m]')
% ylabel('y axis [m]')
% h = figure(1);
% saveas(h, 'plot1.png')
```

In order to check the equations, the following assumptions were made. In the Fig.3 it can be seen the output structure of the robot created in Matlab software on the basis of previously calculated equations.

$$a = 20,\ b = 30,\ c = 5,\ d = 10,\ q_1 = 150^\circ,\ q_2 = 45^\circ,\ q_3 = 45^\circ \tag{1}$$

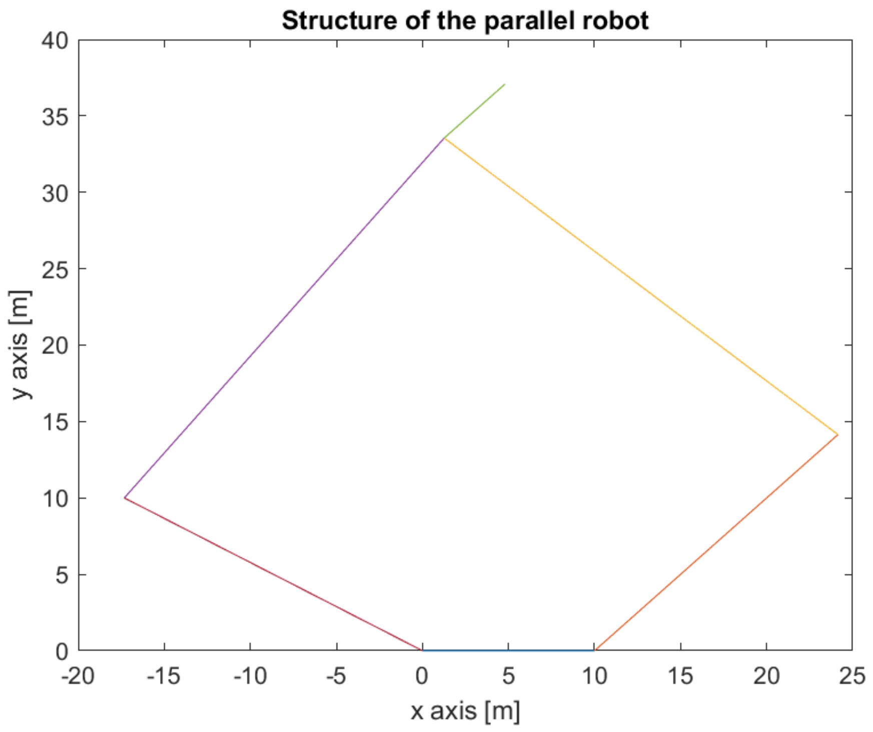

Figure 3: Plot of the robot structure created in Matlab Software

3 Redundancy resolution

3.1 Jacobian

In the second task structure of the robot is different than in the previous one. Link a has the same value as link b, while $d = 0$. In this case forward kinematics is more simple and it looks as follows:

$$X = a \cdot cos(q_L) + \ a \cdot cos(q_R) + \ c \cdot cos(\theta)$$

$$Y = a \cdot sin(q_L) + \ a \cdot sin(q_R) + \ c \cdot sin(\theta)$$

Position of the H is as follows:

$$H = [X, \ Y]$$

$$H = \begin{bmatrix} a \cdot cos(q_L) + \ a \cdot cos(q_R) + \ c \cdot cos(\theta) \\ a \cdot sin(q_L) + \ a \cdot sin(q_R) + \ c \cdot sin(\theta) \end{bmatrix}$$

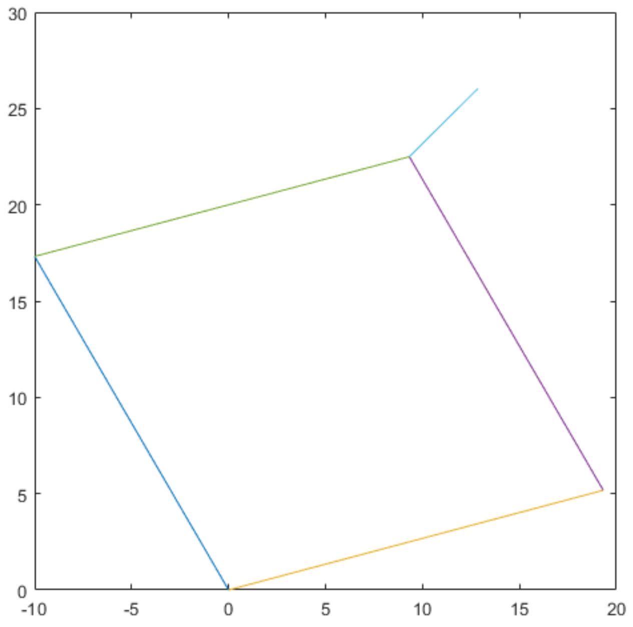

Figure 4: Plot of the velocity profile

Listing 2: Matlab code for the task 1

```
%% Q2_A Jacobian

syms a c q1 q2 q3

X = a*cos(q1) + a*cos(q2) + c*cos(q3);
Y = a*sin(q1) + a*sin(q2) + c*sin(q3);
H = [X; Y];

```

```
J11 = jacobian(H(1),q1);
J12 = jacobian(H(1),q2);
J13 = jacobian(H(1),q3);

J21 = jacobian(H(2),q1);
J22 = jacobian(H(2),q2);
J23 = jacobian(H(2),q3);

J = [J11 J12 J13; J21 J22 J23];

% output:
% J =
% [ -a*sin(q1), -a*sin(q2), -c*sin(q3)]
% [  a*cos(q1),  a*cos(q2),  c*cos(q3)]
```

Jacobian matrix:

$$J = \begin{bmatrix} -a \cdot sin(q_L) + & -a \cdot sin(q_R) + & -c \cdot sin(\theta) \\ a \cdot cos(q_L) + & a \cdot cos(q_R) + & c \cdot cos(\theta) \end{bmatrix}$$

3.2 Movement integration

Using Matlab software hand velocity, hand position profile and hand trajectory have been calculated.

Listing 3: Matlab code for the task 1

```
%% Q2_B Movement integration

% PLOT OF THE HAND VELOCITY
T = 2;
H1 = [0.141, 0.441]';
H2 = [0.241, 0.641]';
X = H2(1) - H1(1);
Y = H2(2) - H1(2);
trajectory = sqrt((X)^2 + (Y)^2);

t = linspace(0,T,100);
tau = t/T;
vel = (30.*tau.^2).*(tau.^2-2*tau+1);
vel = vel'*trajectory;

figure(1);
plot(t, vel);
xlabel('time [s]');
ylabel('velocity [m/s]');
fig1 = figure(1);
% saveas(fig1, 'hand_velocity.png')


% PLOT OF THE HAND POSITION PROFILE
hand = 6.*tau.^5-15.*tau.^4+10.*tau.^3;
hand_pos = hand * trajectory;
figure(2);
plot(t, hand_pos);
xlabel('time [s]');
ylabel('velocity [m/s^2]');
fig2 = figure(2);
```

```
% saveas(fig2, 'hand_position.png')


% PLOT OF THE HAND TRAJCETORY

hand_pos_X = H1(1) + (hand_pos) * X/trajectory
hand_pos_Y = H1(2) + (hand_pos) * Y/trajectory

figure(3)
plot(hand_pos_X, hand_pos_Y);
xlabel('x axis [m]');
ylabel('y axis [m]');
hand_trajectory = figure(3);
% saveas(hand_trajectory, 'hand_trajectory.png');
```

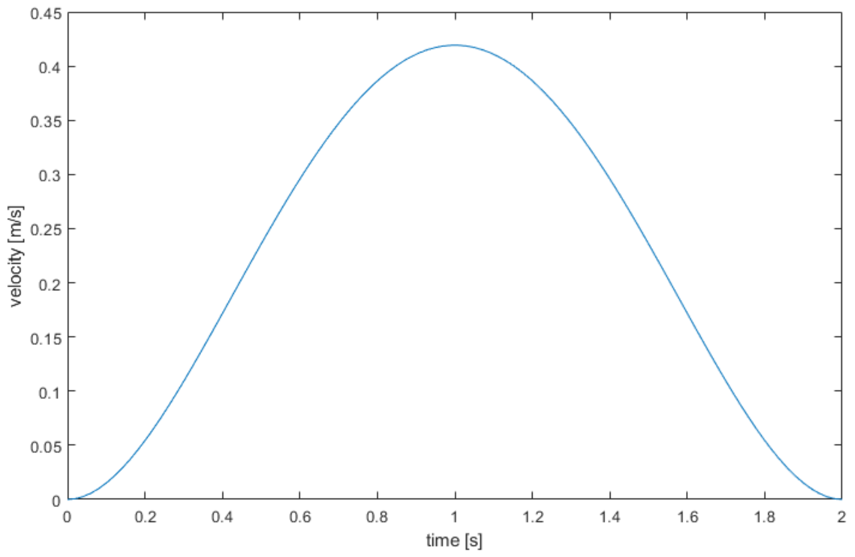

Figure 5: Plot of the velocity profile

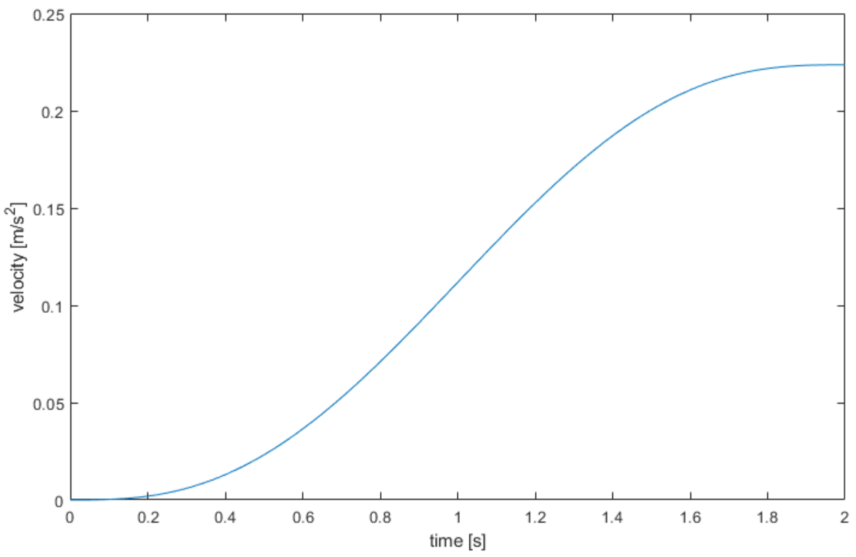

Figure 6: Plot of the hand position

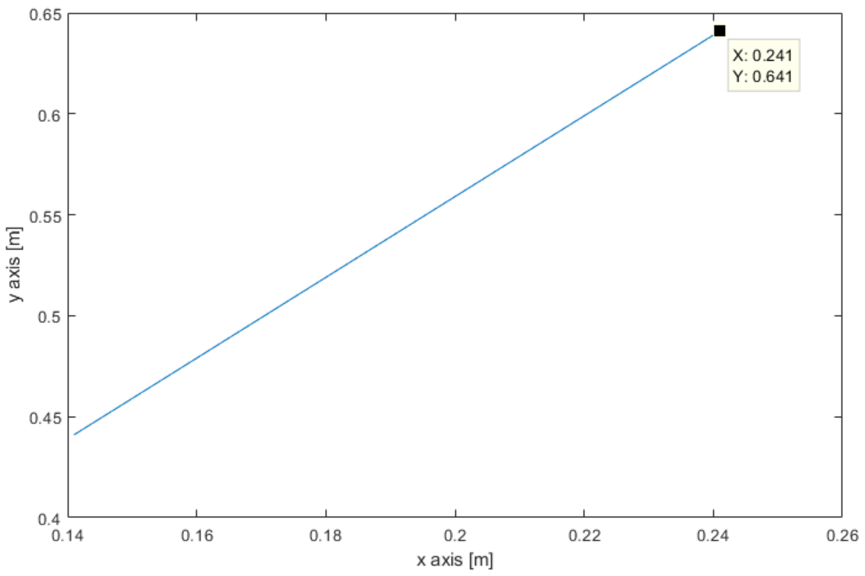

Figure 7: Plot of the hand trajectory

3.3 Movement integration

Last task concerns calculation of the joint variables using Inverse Kinematics. Having desired position and orientation of the end effector it was calculated possible sets of joint angles. All calculations have been performed in Matlab software. Main goal was to obtain the profiles of the q_L, q_R and θ angles. The code looks as follows:

Listing 4: Matlab code for the task 1

```
%% Movement integration

a = 0.3;
c = 0.2;
T = 2;
t = linspace(0,T,100);
tau = t/T;

H1 = [0.141, 0.441];
H2 = [0.241, 0.641];
X = H2(1) - H1(1);
Y = H2(2) - H1(2);
trajectory = sqrt((X)^2+(Y)^2);
sigma = (30.*tau.^2).*(tau.^2-2*tau+1);

velocity_x = (sigma*trajectory) * (X/trajectory);
velocity_y = (sigma*trajectory) * (Y/trajectory);
velocity = sigma * trajectory;

dX = [velocity_x; velocity_y];
B = dX';

q1 = deg2rad(150);
q2 = deg2rad(30);
q3 = deg2rad(45);
q1_0 = deg2rad(150);
q2_0 = deg2rad(30);
q3_0 = deg2rad(45);

syms dq1 dq2 dq3
dQ = [];

%%

for i = 2:100
    J = [-a*sin(q1), -a*sin(q2), -c*sin(q3); a*cos(q1), a*cos(q2), c*cos(q3)];
    dQ(:,i) = pinv(J) * B(i,:)';

    time = linspace(0,1,100);
    Q1(i) = trapz(time(1:i), dQ(1, 1:i)) + q1_0;
    Q2(i) = trapz(time(1:i), dQ(2, 1:i)) + q2_0;
    Q3(i) = trapz(time(1:i), dQ(3, 1:i)) + q3_0;

    q1 = Q1(i);
    q2 = Q2(i);
    q3 = Q3(i);

end


figure(1)
plot(t, rad2deg(Q1))
% title('qL profile')
xlabel('time [s]')
ylabel('position [\circ]')

figure(2)
plot(t, rad2deg(Q2))
% title('qR profile')
xlabel('time [s]')
ylabel('position [\circ]')

figure(3)
```

```
plot(t, rad2deg(Q3))
% title('theta profile')
xlabel('time [s]')
ylabel('position [\circ]')
```

The output results are presented below:

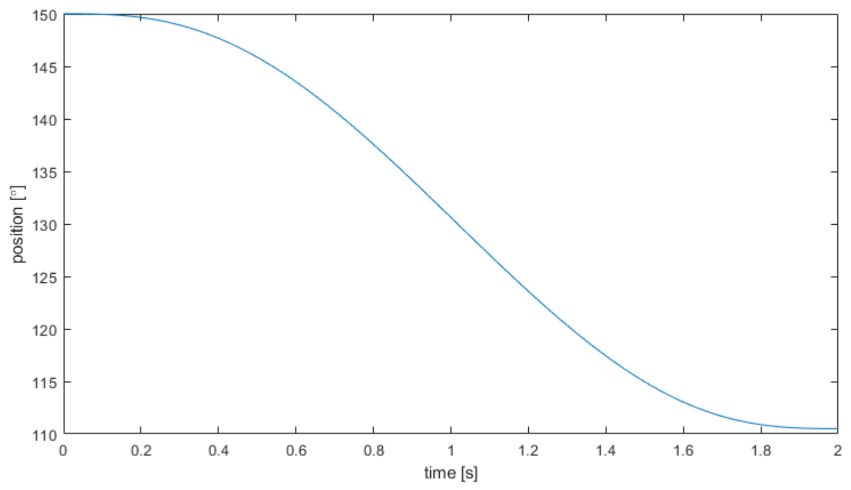

Figure 8: Plot of the Q_L profile

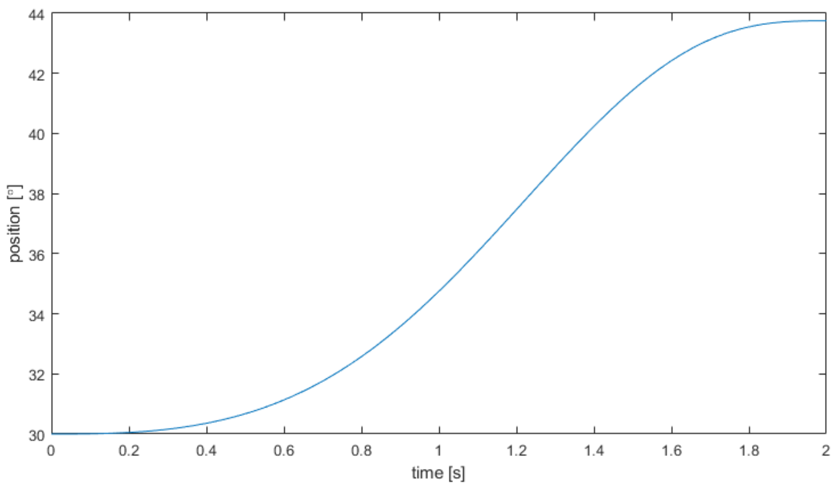

Figure 9: Plot of the Q_R profile

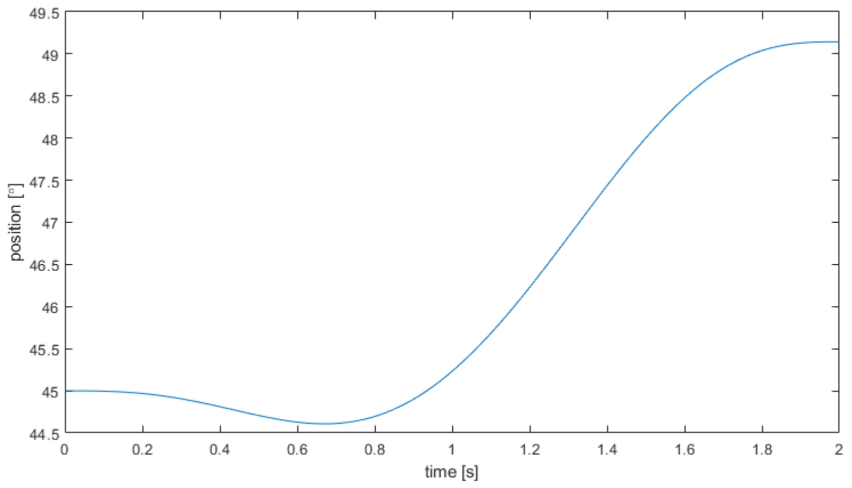

Figure 10: Plot of the θ profile

4 Conclusion

Tutorial 1 classes allows to understand the basic knowledge of the Forward kinematics of the 3-dof parallel robot. On the basis of that it was performed some desire calculations leading to obtain the Inverse Kinematics and Redundancy Reducion.