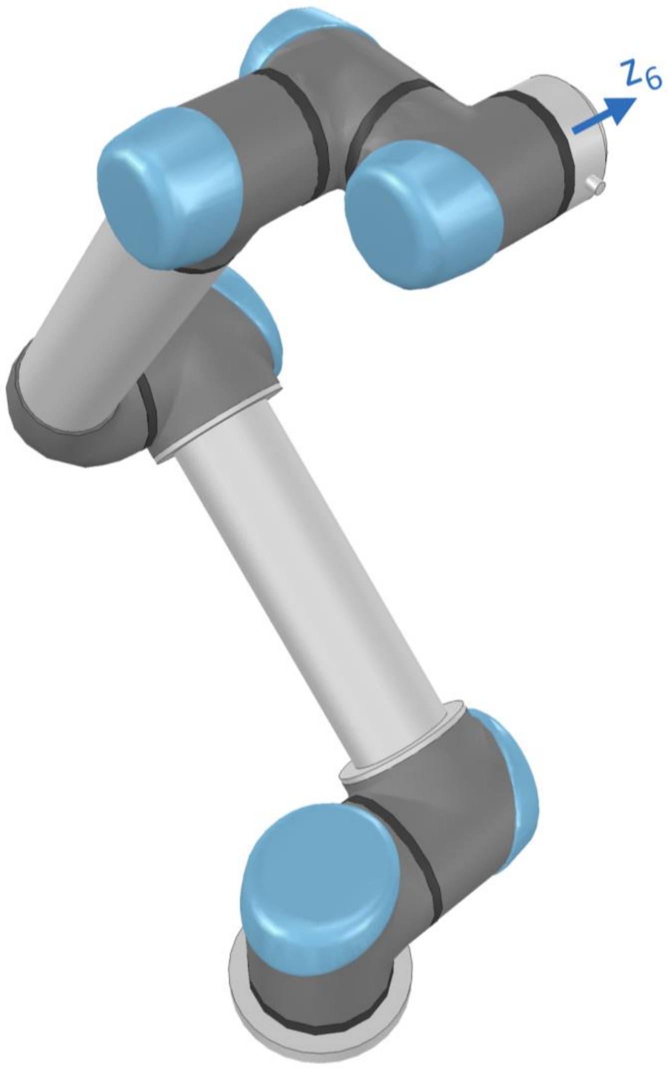
ME5402/EE5106 ADVANCED ROBOTICS

CA1: UR5e Modelling and Simulation



Guo Shiping	**A0260014Y**	**e0973856@u.nus.edu**
Qin Ziyue	**A0267500L**	**e1099950@u.nus.edu**
Wang yukai	**A0263721J**	**e1011078@u.nus.edu**

Contents

Chapter 1 Introduction

Robots are being using to increase production and improve product quality nowadays due to its high efficiency compared with human. As a result, Industry 4.0 employs robotic manipulators and their digital twins to develop cyber-physical systems, which are advanced supervisory control systems that improve the overall performance of physical systems. This project is aiming to study, analysis and design a simulation model for a 6-degree of freedom robotics arm. Therefore, some relevant techniques for robotics arm design and control will be discussed, including forward kinematics derivation, inverse kinematics derivation and dynamic controlling methods.

This report focuses on kinematic modelling only and aim to build a simulation where the robotic arm can write the desired English letters on a 3-dimensional workspace with specified motion of speed. Robotic arm used in this project is a universal Robot 5e (UR5e), as shown in Figure 1. It enables 6 degrees of freedom motion in space, with six revolute joints and one end-effector. In the remaining sections, classical Denavit-Hartenberg (D-H) representation, forward kinematics analysis, inverse kinematics analysis, Jacobian matrix analysis and trajectory planning are discussed in detail.

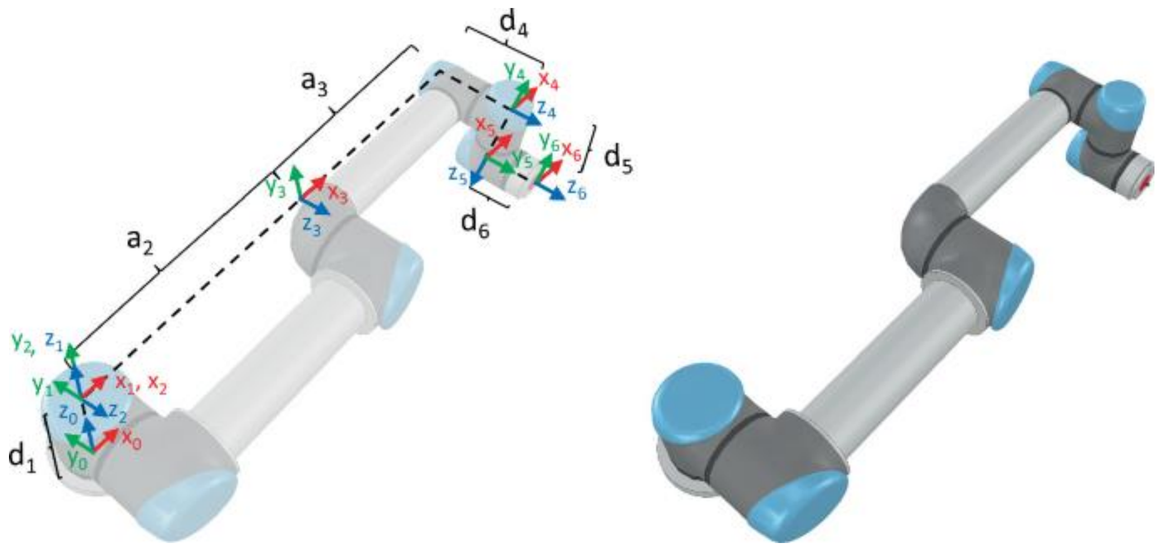


Figure 1. Configuration of UR5e and its dimensions.

Chapter 2 Modelling

2.1 Forward Kinematics analysis

Modelling of the robotic arm is essential for controlling it to achieve expected tasks. Additionally, numerical modelling can be easily transformed to computer-based simulation which is critical for this project. The first step to establish robotic model is forward kinematics. It is a function that takes the angle of each joint as input and outputs the position and pose of end effector in space.

In order to achieve this function, the relevant coordinate of each joint and dimension of the robotic arm ought to be discovered first. In Figure 1, the cartesian coordinates and joint displacement of each joint are represented in $x_i, y_i, z_i$and θ_i , where i means the i-th joint and i could take 1 to 6 separately. The dimension of each link of robotic arm are listed in Table 1.

Table 1. The dimension of UR5e robotic arm

Dimension	Value (mm)
d_1	89
a_2	425
a_3	392
d_4	109
d_5	95
d_6	82

Denavit-Hartenberg (D-H) representation is a systematic method to representing kinematic relationship between two adjacent links of robotic arm. The D-H representation contains four parameters associated with joint i, d_i, a_i, α_i, and θ_i. And each of the D-H parameters are associated with the link relationship between link i and link i-1.

Parameter d denotes the distance from x_i axis to x_i axis along the z_{i-1} direction, a means distance between common normal axis, ɑ represents link twist angle difference between z_i axis and z_{i+1}axis, while the θ is the joint angle difference between x_{i-1} axis and x_i axis. Therefore, the D-H representation can be listed in Table 2 with 6 revolute joints, where value of d and α are defined in Table 1 and value of θ is subject to change during robotic manipulation. Once the D-H table is established, the transformation matrix between any pair of adjacent joints,

Table 2. D-H representation table.

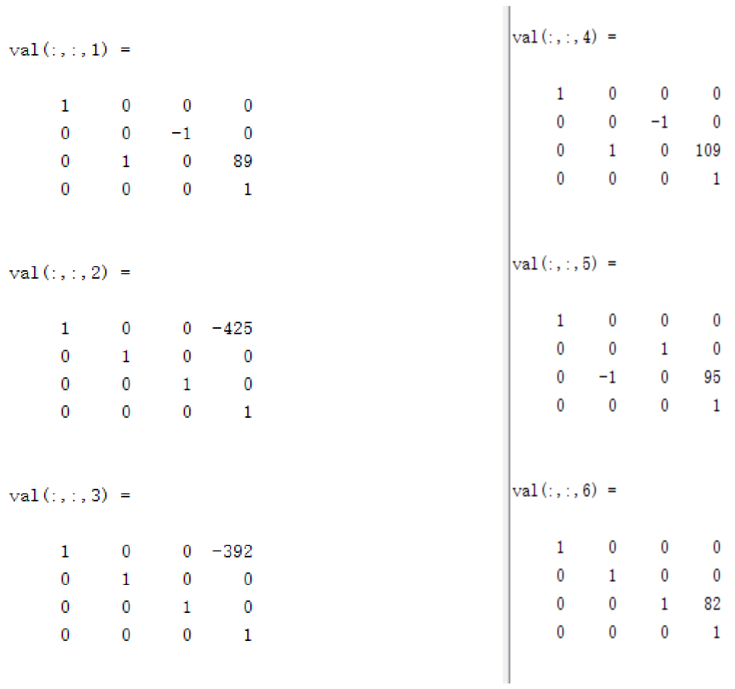
i	d_i	a_i	α$_i$	θ_i
1	d_1	0	$\pi/2$	θ_1
2	0	a_2	0	θ_2
3	0	a_3	0	θ_3
4	d_4	0	$\pi/2$	θ_4
5	d_5	0	-$\pi/2$	θ_5
6	d_5	0	0	θ_6

A_i^{i-1} can be found based on Eq.1. Meanwhile, by multiplying transformation matrix of each pair links all together (Eq.2), the total transformation matrix T_i^0 between base coordinate and end-effector can be found which is the kinematic equation of the robotic arm. In another word, the spatial position and rotational angle of the end-effector now can be represented as a function of joint displacements θ_i and i takes value of 1 to 6.

$$A_i^{i-1} = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i)\cos(\alpha_i) & \sin(\theta_i)\sin(\alpha_i) & \alpha_i\cos(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i)\cos(\alpha_i) & -\cos(\theta_i)\sin(\alpha_i) & \alpha_i\sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{(Eq.1)}$$

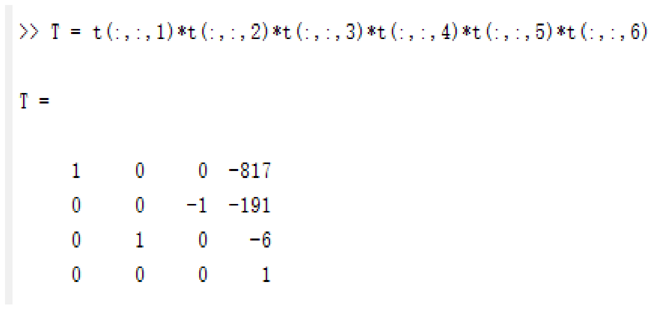
$$T_i^0 = A_1^0(\theta_1)A_2^1(\theta_2)\ldots\ldots_i^{i-1}A(\theta_i) \qquad \text{(Eq.2)}$$

In MATLAB, (Figure 2) six transformation matrixes A_i^{i-1} (i takes value of 1 to 6) can be calculated when θ_i are all set to zero for demonstration purpose. Furthermore, the kinematic equation result of the robotic arm can also be illustrated (Figure 3).



```
val(:,:,1) =

     1     0     0     0
     0     0    -1     0
     0     1     0    89
     0     0     0     1


val(:,:,2) =

     1     0     0  -425
     0     1     0     0
     0     0     1     0
     0     0     0     1


val(:,:,3) =

     1     0     0  -392
     0     1     0     0
     0     0     1     0
     0     0     0     1


val(:,:,4) =

     1     0     0     0
     0     0    -1     0
     0     1     0   109
     0     0     0     1


val(:,:,5) =

     1     0     0     0
     0     0     1     0
     0    -1     0    95
     0     0     0     1


val(:,:,6) =

     1     0     0     0
     0     1     0     0
     0     0     1    82
     0     0     0     1
```

Figure 2. Six transformation matrices of adjacent links

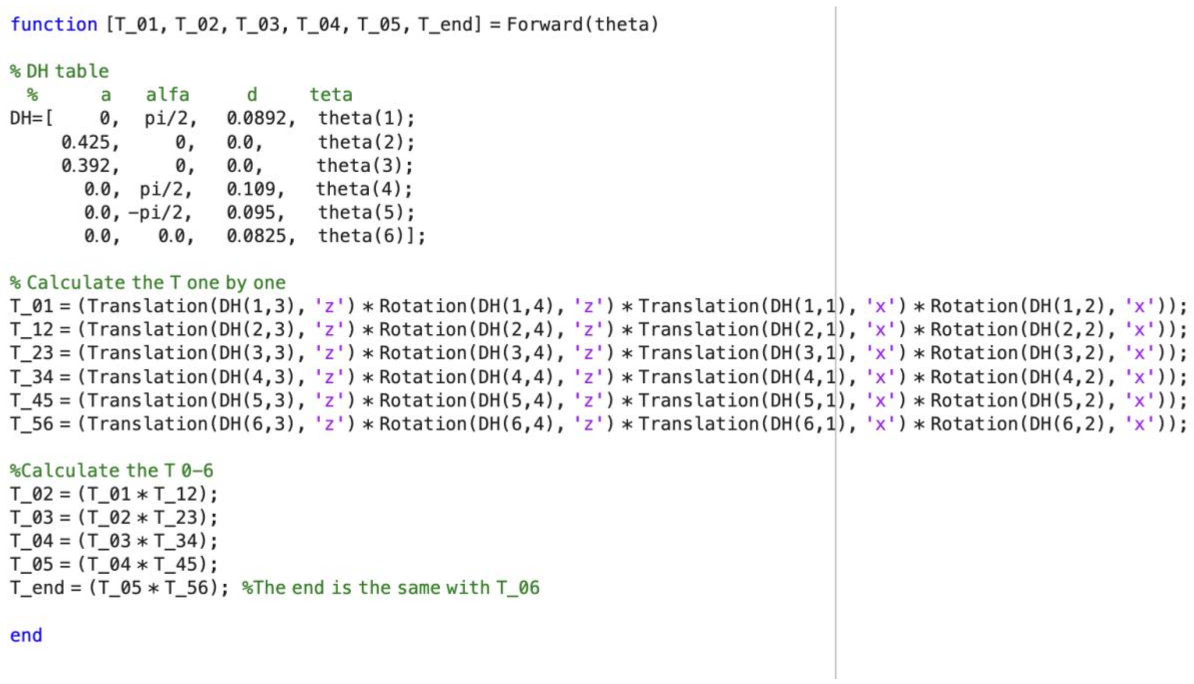


```
>> T = t(:,:,1)*t(:,:,2)*t(:,:,3)*t(:,:,4)*t(:,:,5)*t(:,:,6)

T =

     1     0     0  -817
     0     0    -1  -191
     0     1     0    -6
     0     0     0     1
```

Figure 3. Transformation matrix between base coordinate and end-effector

2.2 Forward Kinematics MATLAB simulation

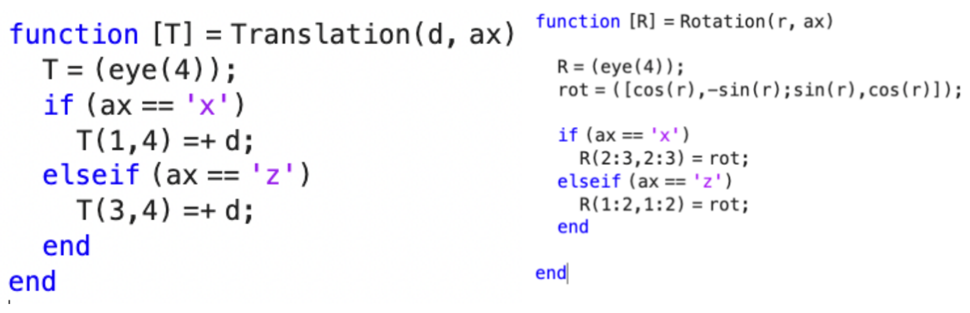
This part mainly shows the simulation process in MATLAB coding. The figures below show computational process of implementing a forward kinematic function in MATLAB.



```
function [T_01, T_02, T_03, T_04, T_05, T_end] = Forward(theta)

% DH table
  %      a    alfa      d      teta
DH=[     0,   pi/2,   0.0892,  theta(1);
     0.425,      0,   0.0,     theta(2);
     0.392,      0,   0.0,     theta(3);
       0.0,   pi/2,   0.109,   theta(4);
       0.0,  -pi/2,   0.095,   theta(5);
       0.0,    0.0,   0.0825,  theta(6)];

% Calculate the T one by one
T_01 = (Translation(DH(1,3), 'z') * Rotation(DH(1,4), 'z') * Translation(DH(1,1), 'x') * Rotation(DH(1,2), 'x'));
T_12 = (Translation(DH(2,3), 'z') * Rotation(DH(2,4), 'z') * Translation(DH(2,1), 'x') * Rotation(DH(2,2), 'x'));
T_23 = (Translation(DH(3,3), 'z') * Rotation(DH(3,4), 'z') * Translation(DH(3,1), 'x') * Rotation(DH(3,2), 'x'));
T_34 = (Translation(DH(4,3), 'z') * Rotation(DH(4,4), 'z') * Translation(DH(4,1), 'x') * Rotation(DH(4,2), 'x'));
T_45 = (Translation(DH(5,3), 'z') * Rotation(DH(5,4), 'z') * Translation(DH(5,1), 'x') * Rotation(DH(5,2), 'x'));
T_56 = (Translation(DH(6,3), 'z') * Rotation(DH(6,4), 'z') * Translation(DH(6,1), 'x') * Rotation(DH(6,2), 'x'));

%Calculate the T 0-6
T_02 = (T_01 * T_12);
T_03 = (T_02 * T_23);
T_04 = (T_03 * T_34);
T_05 = (T_04 * T_45);
T_end = (T_05 * T_56);  %The end is the same with T_06

end
```

Figure 4. Forward Kinematics MATLAB simulation

The function called Translation and Rotation is derived by the following computing process.



```
function [T] = Translation(d, ax)
  T = (eye(4));
  if (ax == 'x')
    T(1,4) =+ d;
  elseif (ax == 'z')
    T(3,4) =+ d;
  end
end
```

```
function [R] = Rotation(r, ax)

  R = (eye(4));
  rot = ([cos(r),-sin(r);sin(r),cos(r)]);

  if (ax == 'x')
    R(2:3,2:3) = rot;
  elseif (ax == 'z')
    R(1:2,1:2) = rot;
  end

end
```

Figure 5. Translation and Rotation function

After computing each transformation matrix, the corresponding T_i^{i-1} can be derived. Then the position information of each joint on robotic arm can be found in their corresponding T_i^{i-1} matrix.

2.3 Inverse Kinematics analysis

To solve the inverse kinematics problem, the initial step is to determine the value of θ_1. This requires us to identify the position of the 5-th coordinate frame in relation to the base frame, which is represented by the point P_{05}. In order to achieve this, we need to move the 6-th frame in the opposite direction of the z-axis by a distance of d_6, as depicted in Figure below.

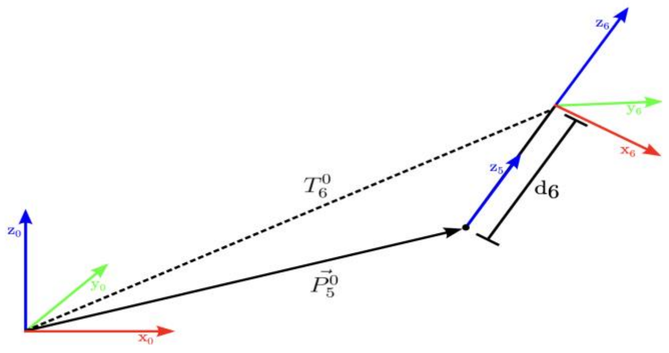


Figure 6. Finding the Origin of the 5th Frame.

This is equivalent to:

$$\overrightarrow{P_5^0} = T_6^0 \begin{bmatrix} 0 \\ 0 \\ -d_6 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad \text{(Eq.3)}$$

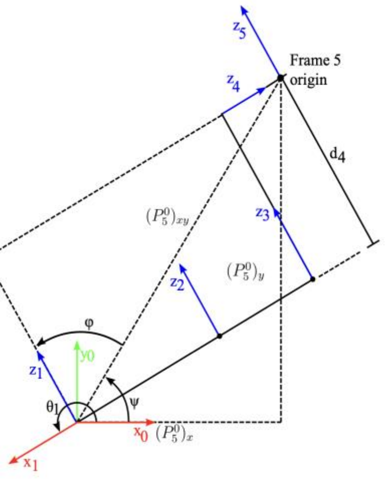


Figure 7. Finding θ_1

Note that the rotations around the axes z_1, z_2, z_3, and z_4 do not change the fact that the origin of Frame 5 is in the plane that is parallel to the x_1 axis, offset by d_4 as shown in Figure 7.

The crucial point to note in this observation is that T_{60} is already known, as it is the intended transformation. As a result, we can compute the displacement vector between the origin of Frame 0 and the origin of Frame 5 using T_{60}. Once we have located the position of the 5-th frame, we can depict an aerial view of the robot, as demonstrated in Figure 3. From this depiction, we can deduce that θ_1 equals ψ, then plus φ, and finally plus π/2, where ψ and φ are given by follows:

$$\psi = \text{atan2}((P_5^0)_y, (P_5^0)_x) \qquad \text{(Eq.4)}$$

$$\varphi = \pm \arccos\left(\frac{d_4}{(P_5^0)_{xy}}\right) = \pm\arccos\left(\frac{d_4}{\sqrt{{(P_5^0)_x}^2 + {(P_5^0)_y}^2}}\right) \qquad \text{(Eq.5)}$$

The two possible values of θ_1 mentioned earlier correspond to the position of the shoulder, which can either be on the left or right side. It's important to note that Equation 5 has a valid solution in all scenarios, except when d_4 exceeds the $(P_{50})xy$ value. This can be observed in Figure 7, where the origin of the 3-rd frame is situated near the z-axis of frame 0. This creates an unattainable cylinder within the spherical workspace of the UR5 (as indicated in Figure 8, which is taken from the UR5 manual)

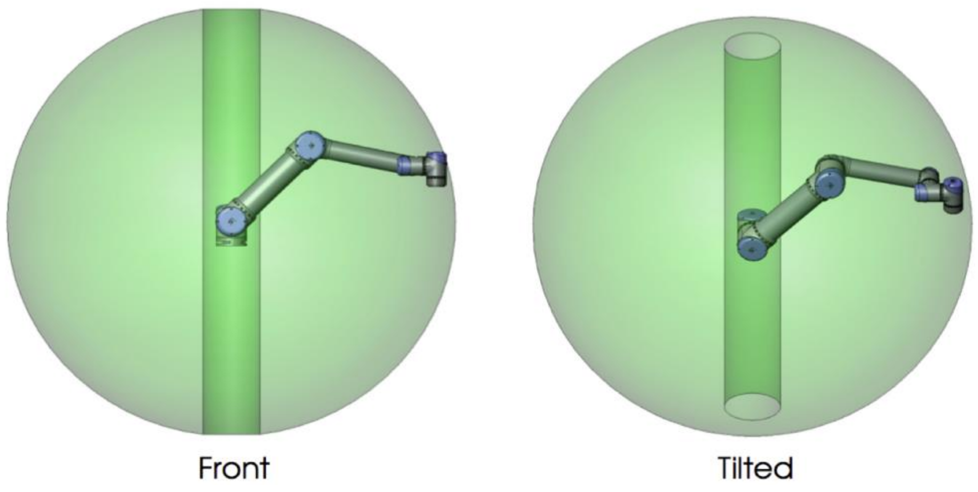


Figure 8: The Workspace of the UR5

With the value of θ_1 determined, we can proceed to calculate θ_5. To do so, we need to examine the location of the 6-th frame in relation to the 1-st frame by creating an aerial view of the robot, as illustrated in Figure 5. It's apparent that $(T_6^1)_z = d_6\cos(\theta_5) + d_4$. Furthermore, $(T_6^1)_z = (T_6^1)_x \sin(\theta_1) - (T_6^0)_y \cos(\theta_1)$. By solving this equation for θ_5, we can obtain its value.

$$\theta_5 = \pm \arccos\left(\frac{(P_6^1)_z - d_4}{d_6}\right) \qquad \text{(Eq.6)}$$

Similar processing as before, there is two possible solutions for θ5, which indicate the orientation of the wrist, either "up" or "down." Figure 6 showcases that, if we disregard translations between frames, z1 can be defined concerning frame 6 as a unit vector with spherical coordinates. We can then obtain the x and y components of this vector by projecting it onto the x-y plane, followed by projecting it onto either the x or y-axis.

Thus, the transformation from frame 6 to frame 1 need to be determined.

$$T_1^6 = ((T_1^0)^{-1} T_6^0)^{-1} \qquad \text{(Eq.7)}$$

Recalling the structure of the first three columns of the homogeneous transformation T_1^6, we can establish the following equations.

$$-\sin(\theta_6)\sin(\theta_5) = z_y \qquad \text{(Eq.8)}$$

$$\cos(\theta_6)\sin(\theta_5) = z_x \qquad \text{(Eq.9)}$$

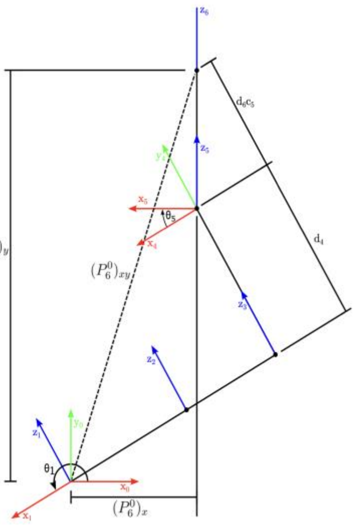


Figure 9: Finding θ_5

First of all, θ_6 can be calculated by equation

$$\theta_6 = atan2\left(\frac{-z_y}{\sin(\theta_5)}, \frac{-z_x}{\sin(\theta_5)}\right) \qquad \text{(Eq.10)}$$

Equation 10 reveals that θ_6 cannot be determined precisely when either sin (θ_5) equals zero or when zx and zy both equal zero. Figure 6 demonstrates that these conditions are, in fact, identical. In this setup, joints 2, 3, 4, and 6 are parallel, resulting in four degrees of freedom for determining the position and rotation of the end-effector in the plane, leading to an infinite number of solutions. In this scenario, a value for $q_6$can be chosen to limit the degrees of freedom to three.

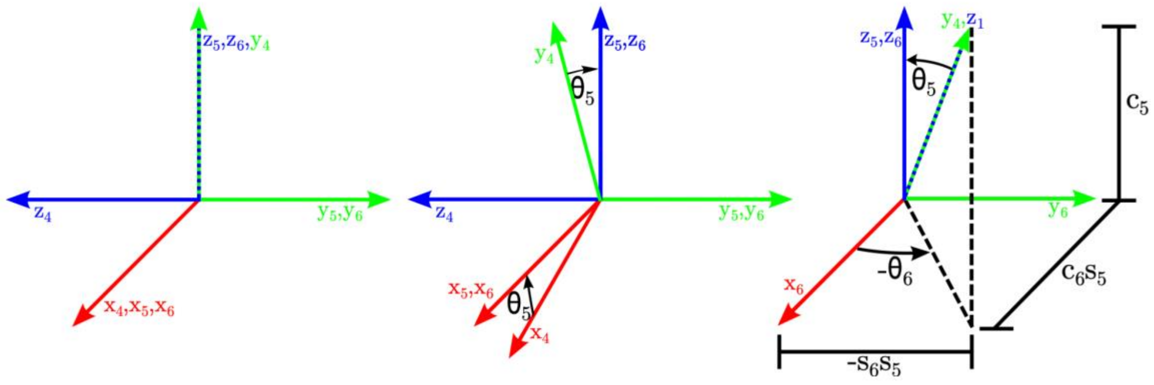


Figure 10: Finding θ_6

$$T_4^1 = T_6^1 T_4^6 = T_6^1 (T_5^4 T_6^5)^{-1} \qquad \text{(Eq.11)}$$

$$\overrightarrow{P_3^1} = T_4^1 \begin{bmatrix} 0 \\ -d_4 \\ 0 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad \text{(Eq.12)}$$

At this point, we can draw the plane that includes frames 1-3, as illustrated in Figure 7. It's evident that:

$$\cos(\xi) = \frac{\left\|\overrightarrow{P_3^1}\right\|^2 - {a_2}^2 - {a_3}^2}{2a_2a_3} \qquad \text{(Eq.13)}$$

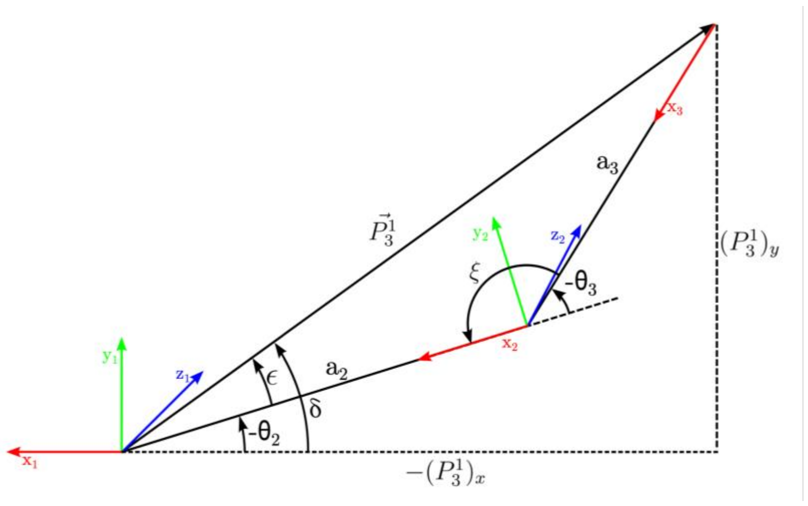


Figure 11: Finding θ_2 and θ_3

with use of the law of cosines.

$$\cos(\xi) = -\cos(\pi - \xi) = -\cos(-\theta_3) = \cos(\theta_3) \qquad \text{(Eq.14)}$$

Combining equation 13 and equation14

$$\theta_3 = \pm\arccos \frac{(\|\overrightarrow{P_3^1}\|)^2 - {a_2}^2 - {a_3}^2}{2a_2a_3} \qquad \text{(Eq.15)}$$

Equation 15 has a solution as long as the argument to arccos is ∈[-1, 1]. We can see that large value of $\left\|\overrightarrow{P_3^1}\right\|$ will cause this the argument to exceed 1. The physical interpretation here is that robot can only reach so far out in any direction.

$$\theta_2 = -(\delta - \epsilon) \qquad \text{(Eq.16)}$$

$$\frac{\sin(\xi)}{\left\|\overrightarrow{P_3^1}\right\|} = \frac{\sin(\epsilon)}{a_3} \qquad \text{(Eq.17)}$$

Combine Eq.16 and Eq.17:

$$\theta_2 = -atan2\big((P_3^1)_y, -(P_3^1)_x\big) + \arcsin\left(\frac{a_3 sin(\theta_3)}{\left\|\overrightarrow{P_3^1}\right\|}\right) \qquad \text{(Eq.18)}$$

There are two solutions for $\theta_2$and θ_3. These solutions are known as “elbow up” and “elbow down.”

The final step to solving the inverse kinematics is to solve for θ_4. First, we want to find T_4^3:

$$T_4^3 = T_1^3 T_4^1 = (T_2^1 T_3^2)^{-1} T_4^1 \qquad \text{(Eq.19)}$$

Using the first column of T_4^3,

$$\theta_4 = atan2(x_y, x_x) \qquad \text{(Eq.20)}$$

$$T_i{}^{i-1} = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i)\cdot\cos(a_i) & \sin(\theta_i)\cdot\sin(a_i) & a_i\cdot\cos(\theta_i) \\ \sin(\theta_i) & cos(\theta_i)\cdot\cos(a_i) & \cos(\theta_i)\cdot\sin(a_i) & a_i\cdot\sin(\theta_i) \\ 0 & \sin(a_i) & \cos(a_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{(Eq.21)}$$

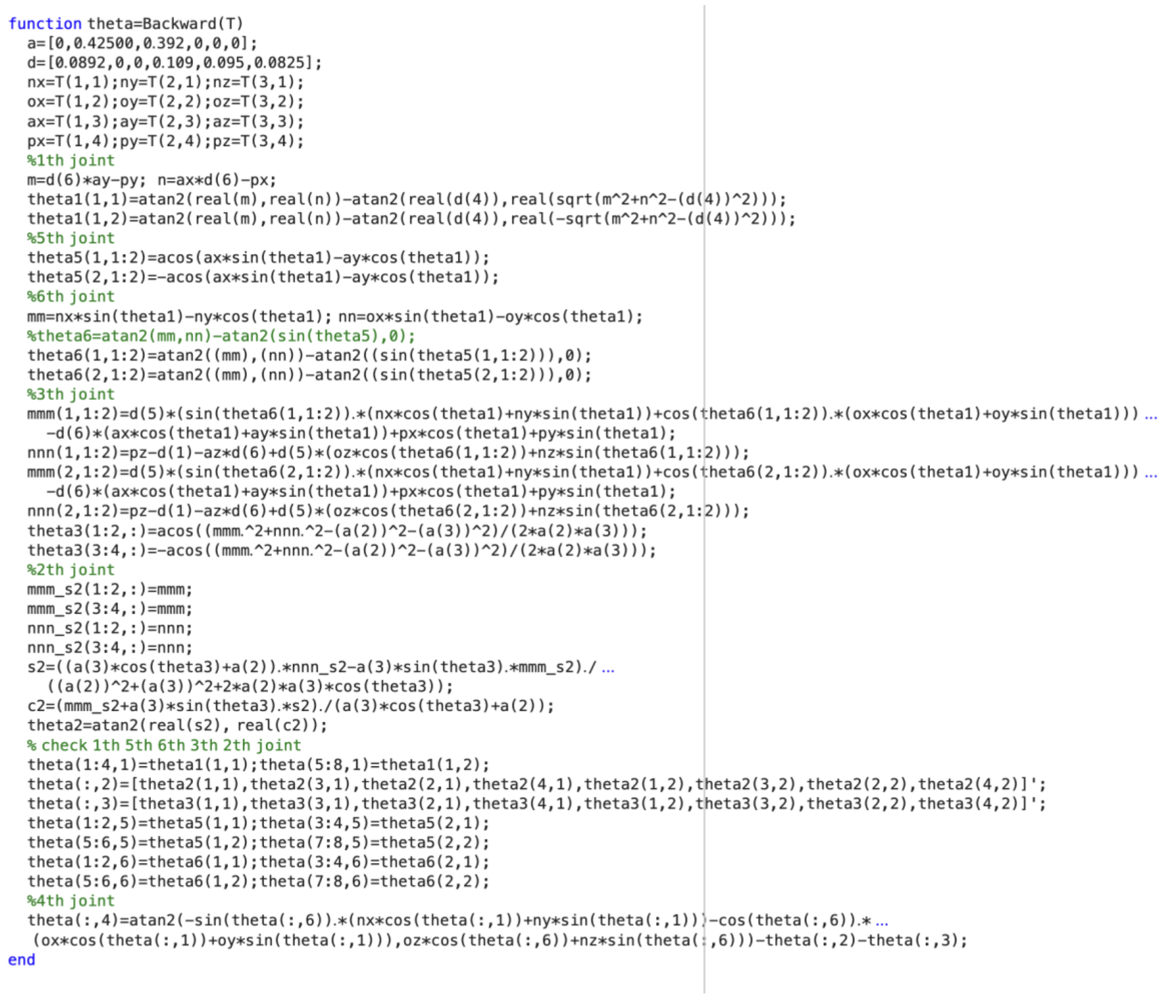
$$T_6^0 = T_1^0 \cdot T_2^1 \cdot T_3^2 \cdot T_4^3 \cdot T_5^4 \cdot T_6^5 \qquad \text{(Eq.22)}$$

Inverse kinematics is used for calculating each angle of joints based on a given terminal position matrix. Position matrix of the end-effector is consisting of rotation matrix and translation matrix. In this simulation, the rotation matrix of the end-effector is fixed, so that the end-effector enables the trajectory to be traced in a fixed plane.
From the Figure 3., the transformation matrix between base coordinate and end-effector can be decomposed into 3x3 rotational matrix and 3x1 position matrix, in

$$T_6^0 = \begin{bmatrix} R & P \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & P_x \\ r_{21} & r_{22} & r_{23} & P_y \\ r_{31} & r_{32} & r_{33} & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{(Eq.23)}$$

2.4 Inverse Kinematics MATLAB simulation

This part mainly shows the simulation process in MATLAB coding. The Figure 12 shows computational process of implementing Inverse kinematic function in MATLAB. In this part, we just follow 2.3 Inverse Kinematics analysis equations from (Eq.3) – (Eq.23) step by step. After getting the inverse Kinematics function, we tried to verify it by giving a target position to replace the position in T and then calculate the joint angle θ_{1-6}. And the input the θ_{1-6} in forwards Kinematics function to get the target position. By testing hundreds of points, we found the error is very small (Figure. 13) which mainly cause by omission of decimal places in data which also cater for the reality because there is almost no noise in this system.



```
function theta=Backward(T)
  a=[0,0.42500,0.392,0,0,0];
  d=[0.0892,0,0,0.109,0.095,0.0825];
  nx=T(1,1);ny=T(2,1);nz=T(3,1);
  ox=T(1,2);oy=T(2,2);oz=T(3,2);
  ax=T(1,3);ay=T(2,3);az=T(3,3);
  px=T(1,4);py=T(2,4);pz=T(3,4);
  %1th joint
  m=d(6)*ay-py;  n=ax*d(6)-px;
  theta1(1,1)=atan2(real(m),real(n))-atan2(real(d(4)),real(sqrt(m^2+n^2-(d(4))^2)));
  theta1(1,2)=atan2(real(m),real(n))-atan2(real(d(4)),real(-sqrt(m^2+n^2-(d(4))^2)));
  %5th joint
  theta5(1,1:2)=acos(ax*sin(theta1)-ay*cos(theta1));
  theta5(2,1:2)=-acos(ax*sin(theta1)-ay*cos(theta1));
  %6th joint
  mm=nx*sin(theta1)-ny*cos(theta1); nn=ox*sin(theta1)-oy*cos(theta1);
  %theta6=atan2(mm,nn)-atan2(sin(theta5),0);
  theta6(1,1:2)=atan2((mm),(nn))-atan2((sin(theta5(1,1:2))),0);
  theta6(2,1:2)=atan2((mm),(nn))-atan2((sin(theta5(2,1:2))),0);
  %3th joint
  mmm(1,1:2)=d(5)*(sin(theta6(1,1:2)).*(nx*cos(theta1)+ny*sin(theta1))+cos(theta6(1,1:2)).*(ox*cos(theta1)+oy*sin(theta1))) ...
    -d(6)*(ax*cos(theta1)+ay*sin(theta1))+px*cos(theta1)+py*sin(theta1);
  nnn(1,1:2)=pz-d(1)-az*d(6)+d(5)*(oz*cos(theta6(1,1:2))+nz*sin(theta6(1,1:2)));
  mmm(2,1:2)=d(5)*(sin(theta6(2,1:2)).*(nx*cos(theta1)+ny*sin(theta1))+cos(theta6(2,1:2)).*(ox*cos(theta1)+oy*sin(theta1))) ...
    -d(6)*(ax*cos(theta1)+ay*sin(theta1))+px*cos(theta1)+py*sin(theta1);
  nnn(2,1:2)=pz-d(1)-az*d(6)+d(5)*(oz*cos(theta6(2,1:2))+nz*sin(theta6(2,1:2)));
  theta3(1:2,:)=acos((mmm.^2+nnn.^2-(a(2))^2-(a(3))^2)/(2*a(2)*a(3)));
  theta3(3:4,:)=-acos((mmm.^2+nnn.^2-(a(2))^2-(a(3))^2)/(2*a(2)*a(3)));
  %2th joint
  mmm_s2(1:2,:)=mmm;
  mmm_s2(3:4,:)=mmm;
  nnn_s2(1:2,:)=nnn;
  nnn_s2(3:4,:)=nnn;
  s2=((a(3)*cos(theta3)+a(2)).*nnn_s2-a(3)*sin(theta3).*mmm_s2)./ ...
    ((a(2))^2+(a(3))^2+2*a(2)*a(3)*cos(theta3));
  c2=(mmm_s2+a(3)*sin(theta3).*s2)./(a(3)*cos(theta3)+a(2));
  theta2=atan2(real(s2), real(c2));
  % check 1th 5th 6th 3th 2th joint
  theta(1:4,1)=theta1(1,1);theta(5:8,1)=theta1(1,2);
  theta(:,2)=[theta2(1,1),theta2(3,1),theta2(2,1),theta2(4,1),theta2(1,2),theta2(3,2),theta2(2,2),theta2(4,2)]';
  theta(:,3)=[theta3(1,1),theta3(3,1),theta3(2,1),theta3(4,1),theta3(1,2),theta3(3,2),theta3(2,2),theta3(4,2)]';
  theta(1:2,5)=theta5(1,1);theta(3:4,5)=theta5(2,1);
  theta(5:6,5)=theta5(1,2);theta(7:8,5)=theta5(2,2);
  theta(1:2,6)=theta6(1,1);theta(3:4,6)=theta6(2,1);
  theta(5:6,6)=theta6(1,2);theta(7:8,6)=theta6(2,2);
  %4th joint
  theta(:,4)=atan2(-sin(theta(:,6)).*(nx*cos(theta(:,1))+ny*sin(theta(:,1)))-cos(theta(:,6)).* ...
   (ox*cos(theta(:,1))+oy*sin(theta(:,1))),oz*cos(theta(:,6))+nz*sin(theta(:,6)))-theta(:,2)-theta(:,3);
end
```

Figure 12. Inverse Kinematics MATLAB simulation

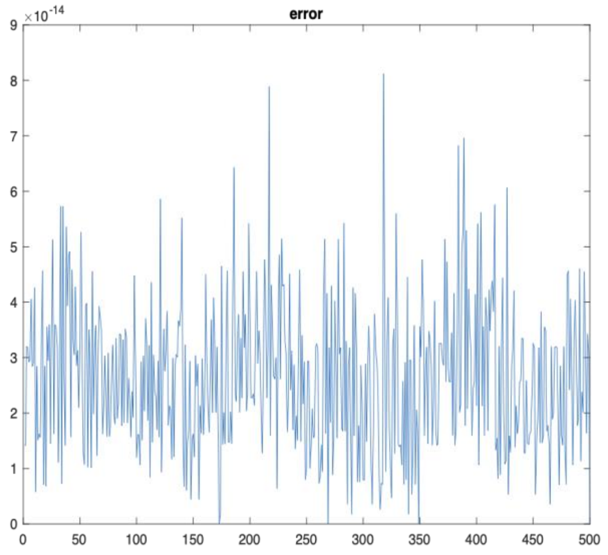


Figure 13. Inverse calculate error

2.5 Jacobian Matrix

In order to achieve the task that assign particular speed for robotic arm to draw trajectory, the Jacobian Matrix is required to be analysed. If the speed of end-effector is known, the velocity vector of each joints can be calculated based on the formular given below.

$$\dot{x} = J\dot{q} \qquad \text{(Eq.23)}$$

where x is position and orientation of end-effector in Cartesian coordinate, while q is the position of each joint. Be taking derivation of these x and q, a linear relationship, or the Jacobian Matrix J between rotational rate of joints and speed of end-effector can be defined. Jacobian matrix can be used to describe the relationship between velocity and acceleration of the end-effector in the robot arm and of individuals joints. Thus, it can control the movement of the robot arm. Jacobian matrix is consisting of vectors representing translational and rotational movement of each joint of robotic arm, as shown below.

$$J = \begin{bmatrix} J_{L_1} & \cdots & J_{L_i} \\ J_{A_1} & \dots & J_{A_i} \end{bmatrix} \qquad \text{(Eq.24)}$$

where J_L is linear velocity component of each joint and J_A is angular velocity component of each joint, while i is the number of joints that robotic arm has. For this project, the robotic arm has six revolute joints. The Jacobian matrix can be further written as:

$$J_i = \begin{bmatrix} J_{L_I} \\ J_{A_1} \end{bmatrix} = \begin{bmatrix} b_{i-1} \times r_{i-1,e} \\ b_{i-1} \end{bmatrix} \qquad \text{(Eq.25)}$$

where b_{i-1} is the unit vector along z-axis of frame i-1, and $r_{i-1,e}$ is the position vector from fram i-1 to end-effector.

$$J_A = \begin{bmatrix} 0 & s_1 & s_1 & s & c_1 s & r_{13} \\ 0 & -c_1 & -c_1 & -c_1 & s_1 s_{234} & r_{23} \\ 1 & 0 & 0 & 0 & -c_{234} & r_{33} \end{bmatrix} \qquad \text{(Eq.26)}$$

In Equation 26, the Jacobian matrix for revolute joints can be easily found by identifying b_{i-1} the unit vector representing the z-axis of joint i-1 relative to base frame. And for the purpose of convenience, s_1 means cousin of joint angle 1, s_{234} means sine of sum of joint angle 2, 3, and 4. Furthermore, the Jacobian maxtrix for prismatic joints can be found by identifying $r_{i-1,e}$, the relative position with respect to frame i-1 and multiply them with relative b_{i-1} :

$$r_{0,e} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \qquad \text{(Eq.27)}$$

$$r_{1,e} = \begin{bmatrix} p_x \\ p_y \\ p_z - d_1 \end{bmatrix} \qquad \text{(Eq.28)}$$

$$r_{2,e} = \begin{bmatrix} p_x - c_1 c_2 a_1 \\ p_y - c_2 s_1 a_2 \\ p_z - s_2 a_2 - d_1 \end{bmatrix} \qquad \text{(Eq.29)}$$

$$r_{3,e} = \begin{bmatrix} p_x - c_1 c_{23} a_3 - c_1 c_2 a_2 \\ p_y - c_{23} s_1 a_3 - c_2 s_1 a_2 \\ p_z - s_{23} a_3 - s_2 a_2 - d_1 \end{bmatrix} \qquad \text{(Eq.30)}$$

$$r_{4,e} = \begin{bmatrix} r_{13} d_6 + c_1 s_{234} d_5 \\ r_{23} d_6 + s_1 s_{234} d_5 \\ r_{33} d_6 - c_{234} d_5 \end{bmatrix} \qquad \text{(Eq.31)}$$

$$r_{5,e} = \begin{bmatrix} r_{13} d_6 \\ r_{23} d_6 \\ r_{33} d_6 \end{bmatrix} \qquad \text{(Eq.32)}$$

$$J_L = \begin{bmatrix} -p_y & -c_1 p_z + c_1 d_1 & c_1(s_{234} s_5 d_6 + c_{234} d_5 - s_{23} a_3) & c_1(s_{234} s_5 d_6 + c_{234} d_5) & -d_6(s_1 s_5 + c_1 c_{234} c_5) & 0 \\ p_x & -s_1 p_z + s_1 d_1 & s_1(s_{234} s_5 d_6 + c_{234} d_5 - s_{23} a_3) & s_1(s_{234} s_5 d_6 + c_{234} d_5) & d_6(c_1 s_5 - c_{234} c_5 s_1 & 0 \\ 0 & s_1 p_y + c_1 p_x & -c_{234} s_5 d_6 + c_{234} d_5 + c_{23} a_3 & -c_{234} s_5 d_6 + s_{234} d_5 & -c_5 s_{234} d_6 & 0 \end{bmatrix}$$

(Eq.33)

Besides, the numerical method also can be applied to calculate Jacobian matrix. In this simulation, both methods are tried. Hence, a numerical solution method called finite differences is introduced. For finite differences method, through adding small increments, the translation change and rotation change of the robot arm end effector can be calculated in each individual joints.

$$Increment = 1e - 6 \qquad \text{(Eq.34)}$$

$$T_{new} = Forwardkinematics(a, alpha, d, theta + Increment) \qquad \text{(Eq.35)}$$

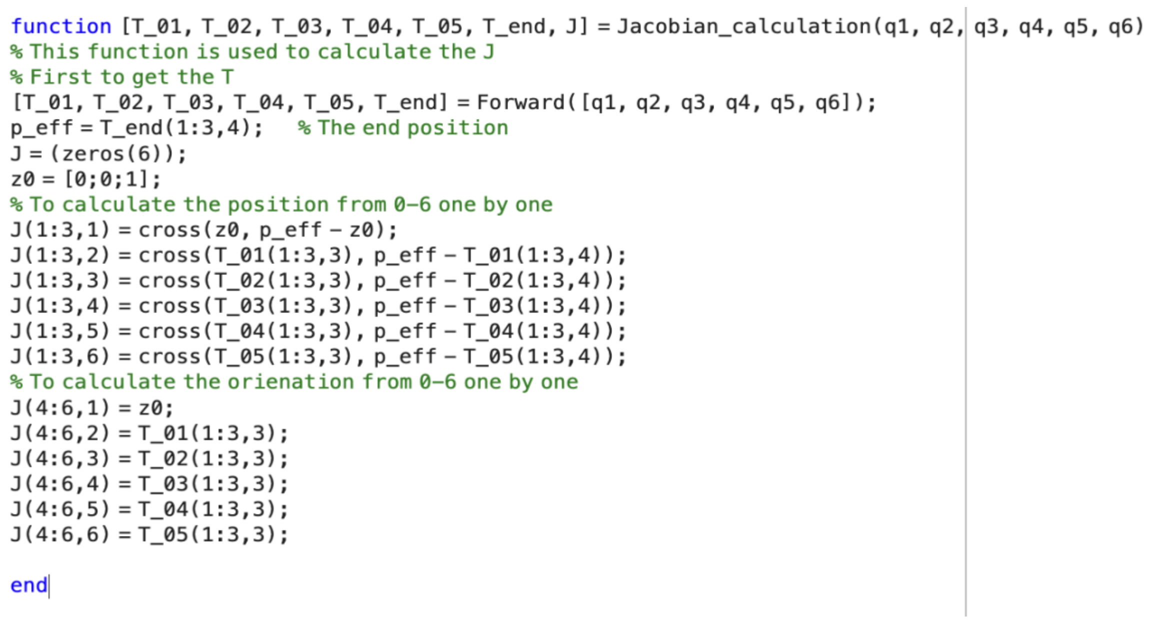
$$J_{L(:,i)} = [T_{new(1:3,4)} - T_{(1:3,4)}]/Increment \qquad \text{(Eq.36)}$$

$$J_{A(:,i)} = \left[T_{new(1:3,1:3)} - T_{(1:3,1:3)}\right]/Increment \qquad \text{(Eq.37)}$$

Finally, the difference is divided by the small increment to obtain the partial differential values of position and attitude so that the Jacobian matrix can be calculated.

2.6 Jacobian Matrix MATLAB Simulation

This part mainly shows the simulation process in MATLAB coding. The figures 14 show computational process of implementing Jacobian calculation function in MATLAB which just follow the instruction in “2.5 Jacobian Matrix”.



```
function [T_01, T_02, T_03, T_04, T_05, T_end, J] = Jacobian_calculation(q1, q2, q3, q4, q5, q6)
% This function is used to calculate the J
% First to get the T
[T_01, T_02, T_03, T_04, T_05, T_end] = Forward([q1, q2, q3, q4, q5, q6]);
p_eff = T_end(1:3,4);    % The end position
J = (zeros(6));
z0 = [0;0;1];
% To calculate the position from 0-6 one by one
J(1:3,1) = cross(z0, p_eff - z0);
J(1:3,2) = cross(T_01(1:3,3), p_eff - T_01(1:3,4));
J(1:3,3) = cross(T_02(1:3,3), p_eff - T_02(1:3,4));
J(1:3,4) = cross(T_03(1:3,3), p_eff - T_03(1:3,4));
J(1:3,5) = cross(T_04(1:3,3), p_eff - T_04(1:3,4));
J(1:3,6) = cross(T_05(1:3,3), p_eff - T_05(1:3,4));
% To calculate the orienation from 0-6 one by one
J(4:6,1) = z0;
J(4:6,2) = T_01(1:3,3);
J(4:6,3) = T_02(1:3,3);
J(4:6,4) = T_03(1:3,3);
J(4:6,5) = T_04(1:3,3);
J(4:6,6) = T_05(1:3,3);

end
```

Figure 14. Jacobian Simulation

Later we calculate the inverse Jacobian in two ways, the first way is directly use pinv function to calculate inverse Jacobian.

Inverse Jacobian = pinv(J_curr)

The second way is:

```
Winv = zeros(6);
for i = 1:6
    Winv(i,i) = i/6 + 0.1;
end
Inverse Jacobian  = Winv * J_curr' * inv(J_curr * Winv * J_curr' + inv(C));
```

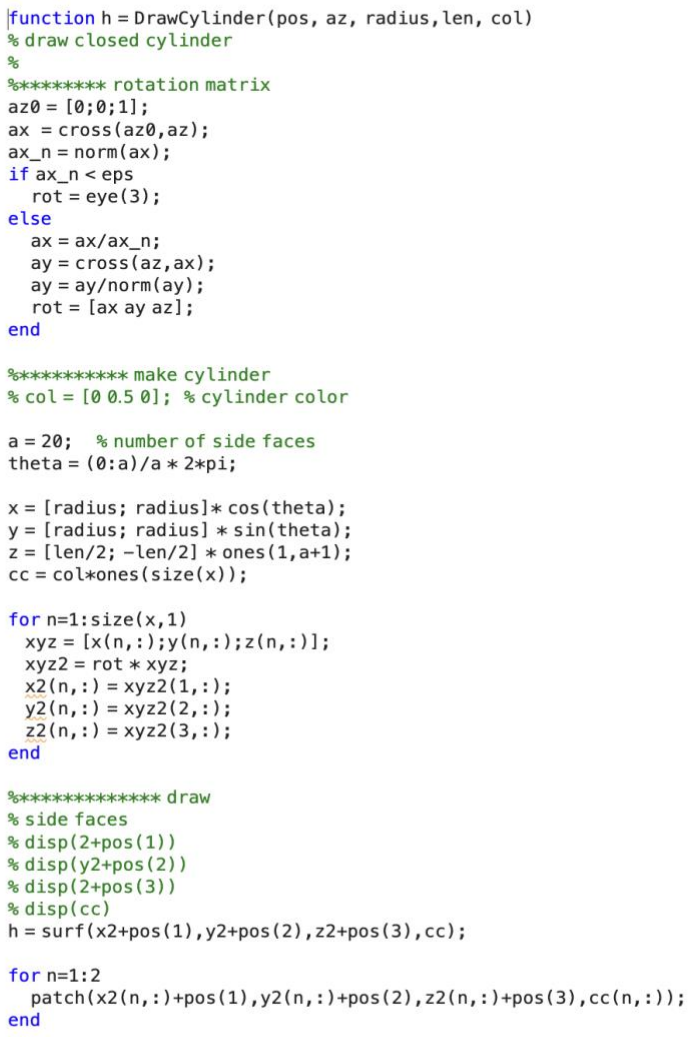
By comparation, the results are almost the same.

Chapter 3 UR5 UI design

In order to have a better outlook, we design the UI by ourself. Here are some key functions.

3.1 Joint drawing function

The figure below shows a function to draw cylinder for each joint to simulate the arm of UR5e.

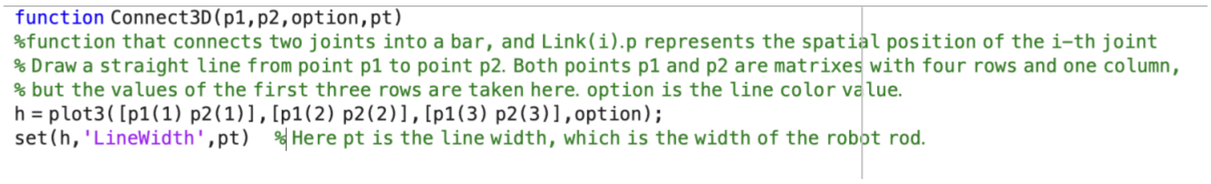


```
function h = DrawCylinder(pos, az, radius,len, col)
% draw closed cylinder
%
%******** rotation matrix
az0 = [0;0;1];
ax  = cross(az0,az);
ax_n = norm(ax);
if ax_n < eps
   rot = eye(3);
else
   ax = ax/ax_n;
   ay = cross(az,ax);
   ay = ay/norm(ay);
   rot = [ax ay az];
end

%********** make cylinder
% col = [0 0.5 0];  % cylinder color

a = 20;   % number of side faces
theta = (0:a)/a * 2*pi;

x = [radius; radius]* cos(theta);
y = [radius; radius] * sin(theta);
z = [len/2; -len/2] * ones(1,a+1);
cc = col*ones(size(x));

for n=1:size(x,1)
  xyz = [x(n,:);y(n,:);z(n,:)];
  xyz2 = rot * xyz;
  x2(n,:) = xyz2(1,:);
  y2(n,:) = xyz2(2,:);
  z2(n,:) = xyz2(3,:);
end

%************* draw
% side faces
% disp(2+pos(1))
% disp(y2+pos(2))
% disp(2+pos(3))
% disp(cc)
h = surf(x2+pos(1),y2+pos(2),z2+pos(3),cc);

for n=1:2
   patch(x2(n,:)+pos(1),y2(n,:)+pos(2),z2(n,:)+pos(3),cc(n,:));
end
```

Figure 14. Joint drawing function

3.2 Joint connection drawing function

The figure below shows a function to connect each cylinder for each joint to simulate UR5e.



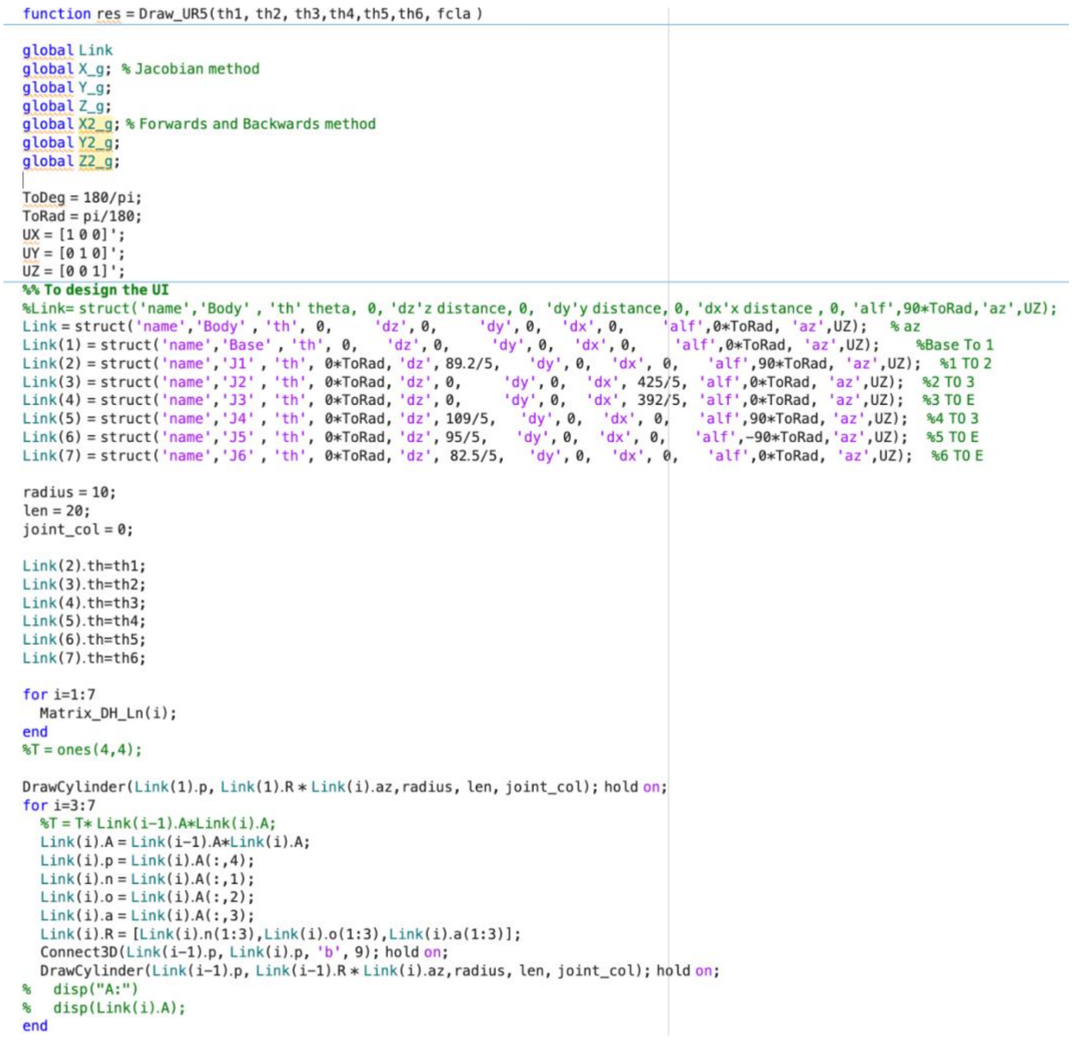
```
function Connect3D(p1,p2,option,pt)
%function that connects two joints into a bar, and Link(i).p represents the spatial position of the i-th joint
% Draw a straight line from point p1 to point p2. Both points p1 and p2 are matrixes with four rows and one column,
% but the values of the first three rows are taken here. option is the line color value.
h = plot3([p1(1) p2(1)],[p1(2) p2(2)],[p1(3) p2(3)],option);
set(h,'LineWidth',pt)   % Here pt is the line width, which is the width of the robot rod.
```

Figure 15. Joint connection function

3.3 UR5 drawing function

It should be noted that the data in the DH table is in meters, and it needs to be in millimetres when drawing the robot arm. Here we also made a five-fold proportional reduction.

For example, the first dz = 0.892m * 1000 / 5



```
function res = Draw_UR5(th1, th2, th3,th4,th5,th6, fcla )

global Link
global X_g; % Jacobian method
global Y_g;
global Z_g;
global X2_g; % Forwards and Backwards method
global Y2_g;
global Z2_g;

ToDeg = 180/pi;
ToRad = pi/180;
UX = [1 0 0]';
UY = [0 1 0]';
UZ = [0 0 1]';
%% To design the UI
%Link= struct('name','Body' , 'th' theta, 0, 'dz'z distance, 0, 'dy'y distance, 0, 'dx'x distance , 0, 'alf',90*ToRad,'az',UZ);
Link = struct('name','Body' , 'th', 0,     'dz', 0,     'dy', 0,  'dx', 0,    'alf',0*ToRad, 'az',UZ);   % az
Link(1) = struct('name','Base' , 'th', 0,   'dz', 0,     'dy', 0,  'dx', 0,    'alf',0*ToRad, 'az',UZ);    %Base To 1
Link(2) = struct('name','J1' , 'th', 0*ToRad, 'dz', 89.2/5,   'dy', 0,  'dx', 0,   'alf',90*ToRad, 'az',UZ);  %1 TO 2
Link(3) = struct('name','J2' , 'th', 0*ToRad, 'dz', 0,      'dy', 0,  'dx', 425/5, 'alf',0*ToRad, 'az',UZ);  %2 TO 3
Link(4) = struct('name','J3' , 'th', 0*ToRad, 'dz', 0,      'dy', 0,  'dx', 392/5, 'alf',0*ToRad, 'az',UZ);  %3 TO E
Link(5) = struct('name','J4' , 'th', 0*ToRad, 'dz', 109/5,   'dy', 0,  'dx', 0,   'alf',90*ToRad, 'az',UZ);  %4 TO 3
Link(6) = struct('name','J5' , 'th', 0*ToRad, 'dz', 95/5,   'dy', 0,  'dx', 0,   'alf',-90*ToRad,'az',UZ);  %5 TO E
Link(7) = struct('name','J6' , 'th', 0*ToRad, 'dz', 82.5/5,   'dy', 0,  'dx', 0,   'alf',0*ToRad, 'az',UZ);  %6 TO E

radius = 10;
len = 20;
joint_col = 0;

Link(2).th=th1;
Link(3).th=th2;
Link(4).th=th3;
Link(5).th=th4;
Link(6).th=th5;
Link(7).th=th6;

for i=1:7
  Matrix_DH_Ln(i);
end
%T = ones(4,4);

DrawCylinder(Link(1).p, Link(1).R * Link(i).az,radius, len, joint_col); hold on;
for i=3:7
  %T = T* Link(i-1).A*Link(i).A;
  Link(i).A = Link(i-1).A*Link(i).A;
  Link(i).p = Link(i).A(:,4);
  Link(i).n = Link(i).A(:,1);
  Link(i).o = Link(i).A(:,2);
  Link(i).a = Link(i).A(:,3);
  Link(i).R = [Link(i).n(1:3),Link(i).o(1:3),Link(i).a(1:3)];
  Connect3D(Link(i-1).p, Link(i).p, 'b', 9); hold on;
  DrawCylinder(Link(i-1).p, Link(i-1).R * Link(i).az,radius, len, joint_col); hold on;
%   disp("A:")
%   disp(Link(i).A);
end
```

Figure 16. Joint connection function

The figure shows below is the original position of UR5e modelling obtained by our UI design function. However, for better display purposes, this model is a scaled-down version of the real UR5E robot arm model.

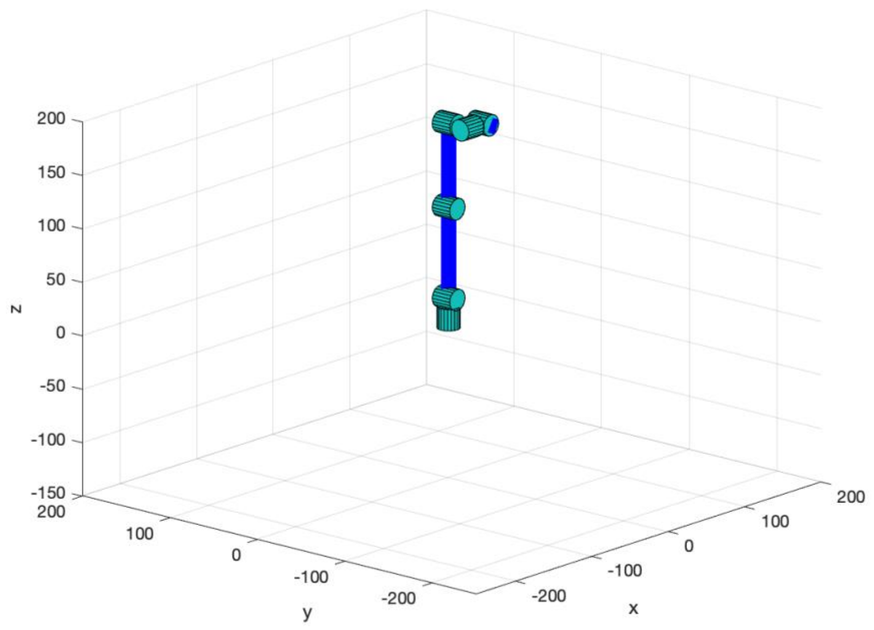


Figure17. UR5 UI design

So far, we have completed the forward and reverse operations and graphic design of the robotic arm. We first test the correctness of the forward and reverse operations and the correctness of the Jacobian calculation on the robotic arm.

Chapter 4 Movement design

4.1 Canvas

First of all, as the base of robotic arm is located at (0,0,0) position in 3-dimensional work space, the canvas is set to be perpendicular to x-y plane to make the drawing easier. The origin of the canvas is located at (100,0,0) and the dimension of the canvas is a 200x200 units square shape as shown in figure below. The canvas is evenly divided into 4 subsections, where each letter will be written on each subsection. As we predefines the region where the letter will be written, the control of end-effector's trajectory is more organized and convenient.

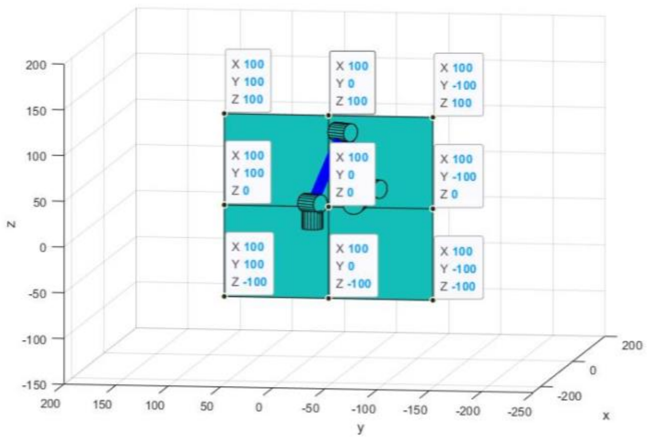
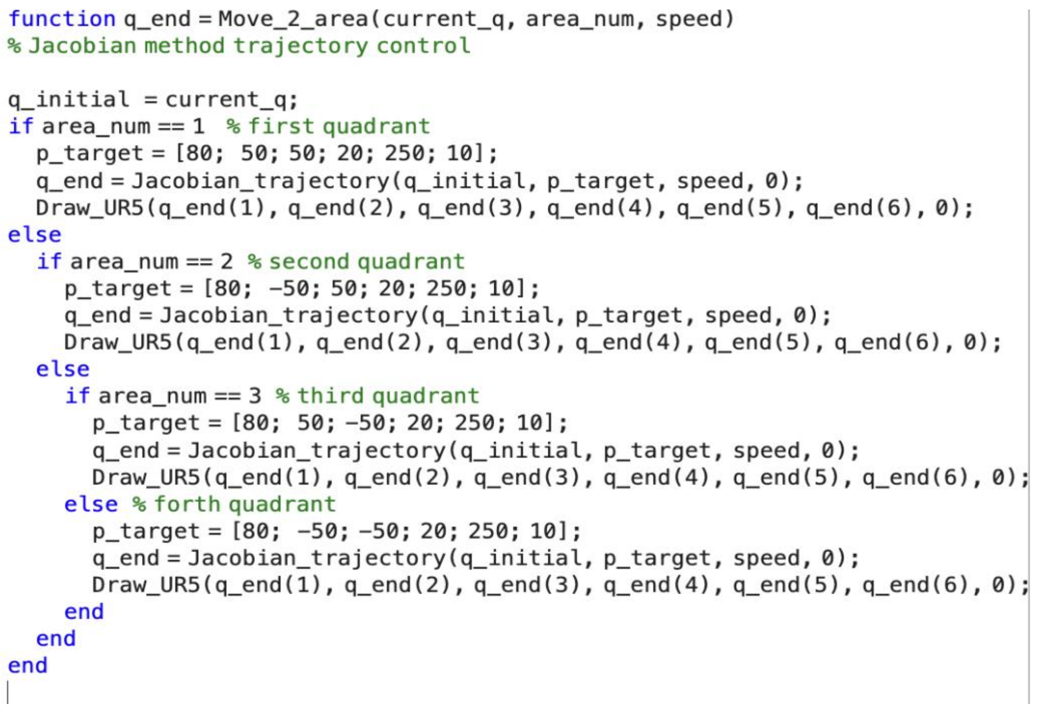


Figure 18. Demonstration of the working plane

4.2 Trajectory design

Before writing, the arm end actuator need arrive at the front of specific quadrant and then become drawing.We design the function to move to specific quadrant first.

Move_2_area(current_q, area_num, speed) area_num represent the quadrant number (1-4).



```
function q_end = Move_2_area(current_q, area_num, speed)
% Jacobian method trajectory control

q_initial = current_q;
if area_num == 1  % first quadrant
  p_target = [80;  50; 50; 20; 250; 10];
  q_end = Jacobian_trajectory(q_initial, p_target, speed, 0);
  Draw_UR5(q_end(1), q_end(2), q_end(3), q_end(4), q_end(5), q_end(6), 0);
else
  if area_num == 2 % second quadrant
    p_target = [80;  -50; 50; 20; 250; 10];
    q_end = Jacobian_trajectory(q_initial, p_target, speed, 0);
    Draw_UR5(q_end(1), q_end(2), q_end(3), q_end(4), q_end(5), q_end(6), 0);
  else
    if area_num == 3 % third quadrant
      p_target = [80;  50; -50; 20; 250; 10];
      q_end = Jacobian_trajectory(q_initial, p_target, speed, 0);
      Draw_UR5(q_end(1), q_end(2), q_end(3), q_end(4), q_end(5), q_end(6), 0);
    else % forth quadrant
      p_target = [80;  -50; -50; 20; 250; 10];
      q_end = Jacobian_trajectory(q_initial, p_target, speed, 0);
      Draw_UR5(q_end(1), q_end(2), q_end(3), q_end(4), q_end(5), q_end(6), 0);
    end
  end
end
```

Figure 19. Demonstration of function that move the robot to specific quadrant.

4.3 Characters design

To write letters properly, first it needs to plan the trajectory of each letter in advance and set some path points. The following diagram shows some of the selected trajectory points for each letter of the alphabet, through which the robot arm will pass in order to complete the task of writing the letters

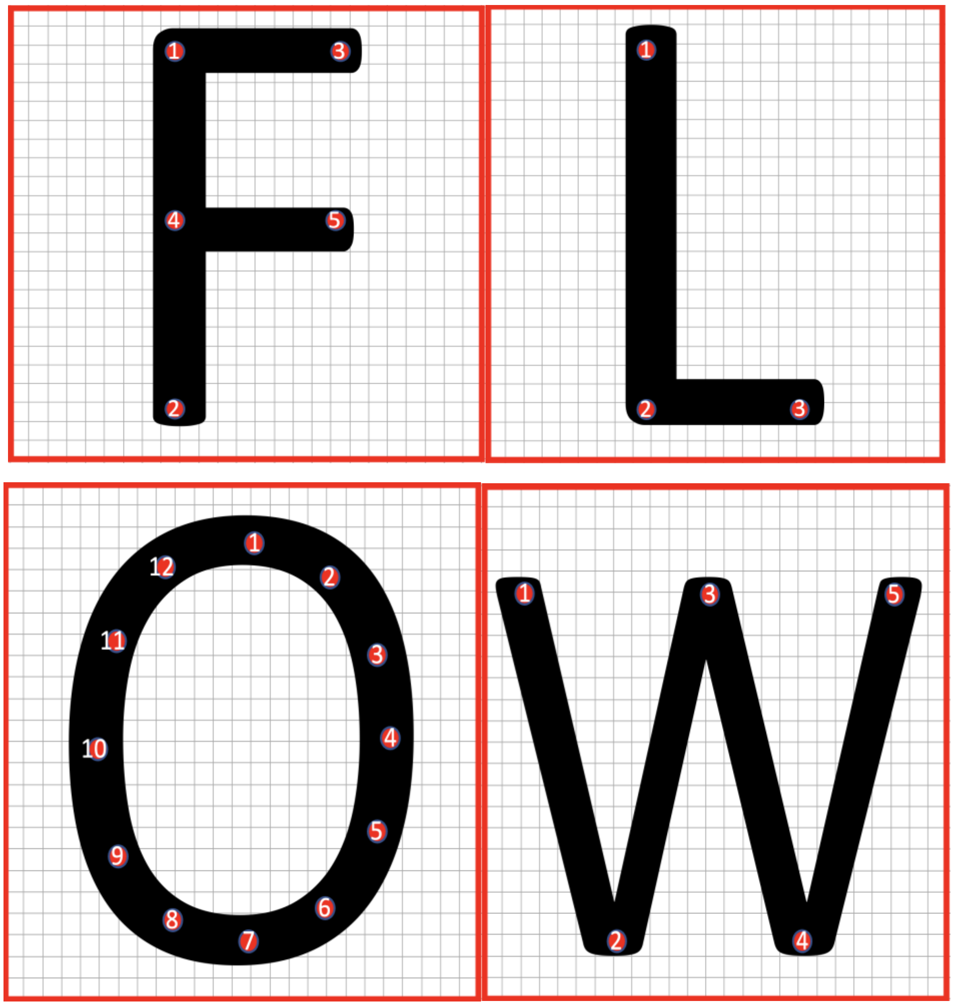
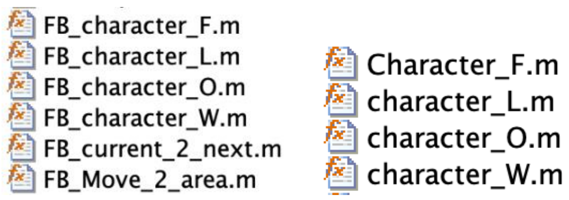


Figure 20. “FLOW” characters design

For “O” character, we also tried to use the circle function to write.

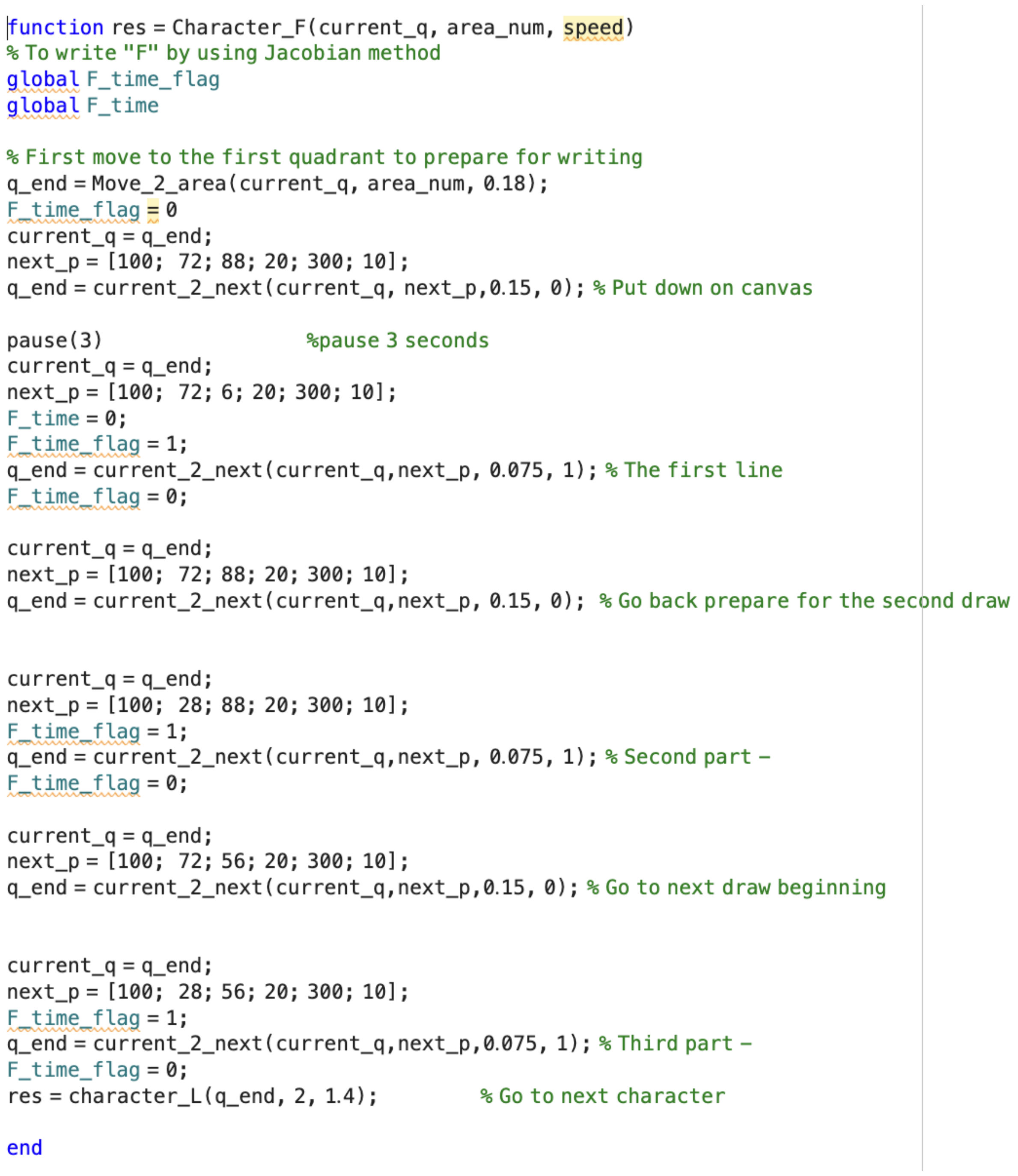
Then we wrap each word into a function. In each function we plan each step and write. It is showed in figure 21.



FB_character_F.m
FB_character_L.m
FB_character_O.m
FB_character_W.m
FB_current_2_next.m
FB_Move_2_area.m

Character_F.m
character_L.m
character_O.m
character_W.m

Figure 21. Forwards and backwards & Jacobian method “FLOW” functions

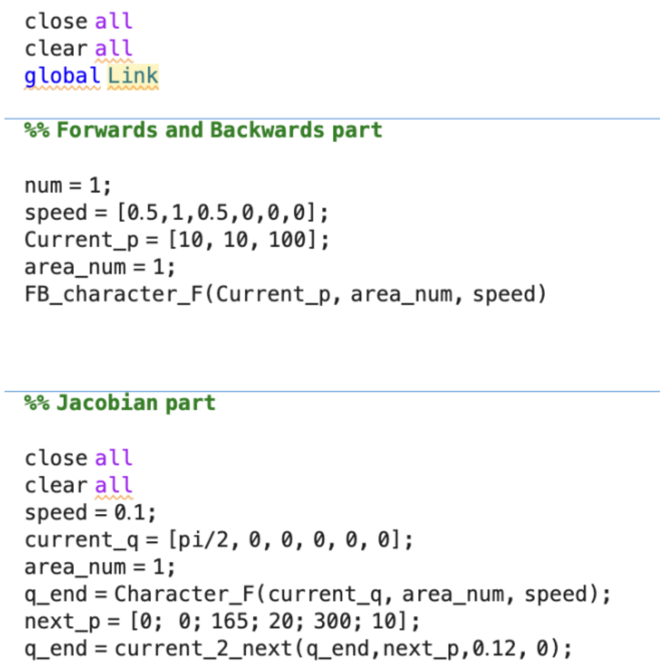


```
function res = Character_F(current_q, area_num, speed)
% To write "F" by using Jacobian method
global F_time_flag
global F_time

% First move to the first quadrant to prepare for writing
q_end = Move_2_area(current_q, area_num, 0.18);
F_time_flag = 0
current_q = q_end;
next_p = [100;  72; 88; 20; 300; 10];
q_end = current_2_next(current_q, next_p,0.15, 0); % Put down on canvas

pause(3)                    %pause 3 seconds
current_q = q_end;
next_p = [100;  72; 6; 20; 300; 10];
F_time = 0;
F_time_flag = 1;
q_end = current_2_next(current_q,next_p, 0.075, 1); % The first line
F_time_flag = 0;

current_q = q_end;
next_p = [100;  72; 88; 20; 300; 10];
q_end = current_2_next(current_q,next_p, 0.15, 0);  % Go back prepare for the second draw


current_q = q_end;
next_p = [100;  28; 88; 20; 300; 10];
F_time_flag = 1;
q_end = current_2_next(current_q,next_p, 0.075, 1); % Second part -
F_time_flag = 0;

current_q = q_end;
next_p = [100;  72; 56; 20; 300; 10];
q_end = current_2_next(current_q,next_p,0.15, 0); % Go to next draw beginning


current_q = q_end;
next_p = [100;  28; 56; 20; 300; 10];
F_time_flag = 1;
q_end = current_2_next(current_q,next_p,0.075, 1); % Third part -
F_time_flag = 0;
res = character_L(q_end, 2, 1.4);           % Go to next character

end
```

Figure 22. Character “F” packaged function

Chapter 5 Writing simulation

Here is the main.m function. We tried to use pure forwards and backwards method and Jacobian method to implement the simulation. Both results are reasonable and ideal.



```
close all
clear all
global Link

%% Forwards and Backwards part

num = 1;
speed = [0.5,1,0.5,0,0,0];
Current_p = [10, 10, 100];
area_num = 1;
FB_character_F(Current_p, area_num, speed)


%% Jacobian part

close all
clear all
speed = 0.1;
current_q = [pi/2, 0, 0, 0, 0, 0];
area_num = 1;
q_end = Character_F(current_q, area_num, speed);
next_p = [0;  0; 165; 20; 300; 10];
q_end = current_2_next(q_end,next_p,0.12, 0);
```

Figure 23: The main function

5.1 Speed control

We tried two methods. For both methods we used similar way to control the speed. The jacobian method can directly operate the speed of joint. We calculate the speed by using the distance and desired time. For Jacobinan method, we have two methods to write. The first way to write is to write at a decreasing speed which showed in Figure 24.

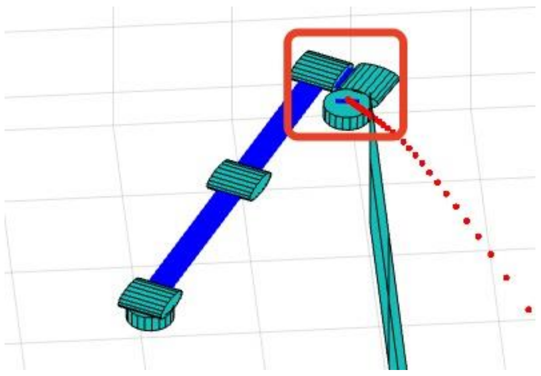


Figure 24. Write in decreasing speed

The advantage is this method is relatively stable, but is hard to control the writing time.

According to the requirement, we must ensure that the time to write each word is equal. We edit the codes and change the writing speed to a constant speed, which is easier to control. But this way has a very obvious disadvantage, we can't write in a very high speed, which will causing the arm to become unstable which can be find in Figure 24 and Figure 25. That's mainly because of the singularities.

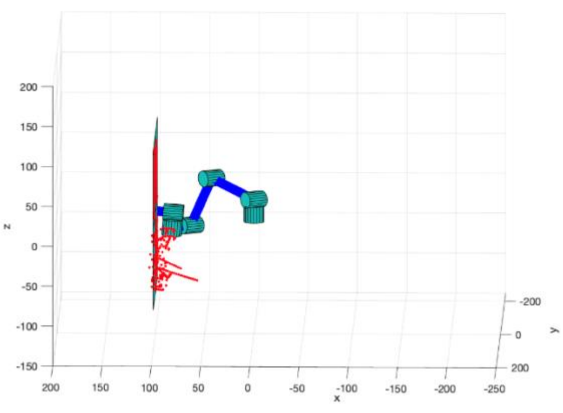


Figure 24. Unstable writing left view

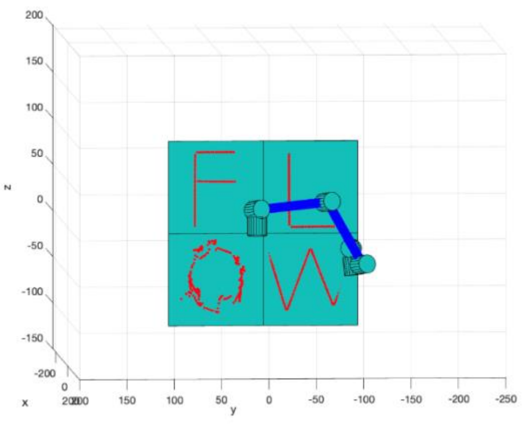


Figure 25. Unstable writing first view

So, choosing a suitable speed to control the speed of very character. We get the result below showed in figure 27. Every character need almost 18 seconds. The result is not stable because the computer calculation. The error floats within one second.

Similar with Jacobian method, the forwards and Backwards method just based on the number of points to get the writing time, which means we can insert different points to control the time consumption.

5.2 Forwards and Backwards method

Backward kinematics is the relationship established between end-effector and every joint of the robotic arm. With known end-effector position, the revolute angle of each joint can be derived using homogeneous transformational matrix. This method is relative straight forward and require fewer computation.

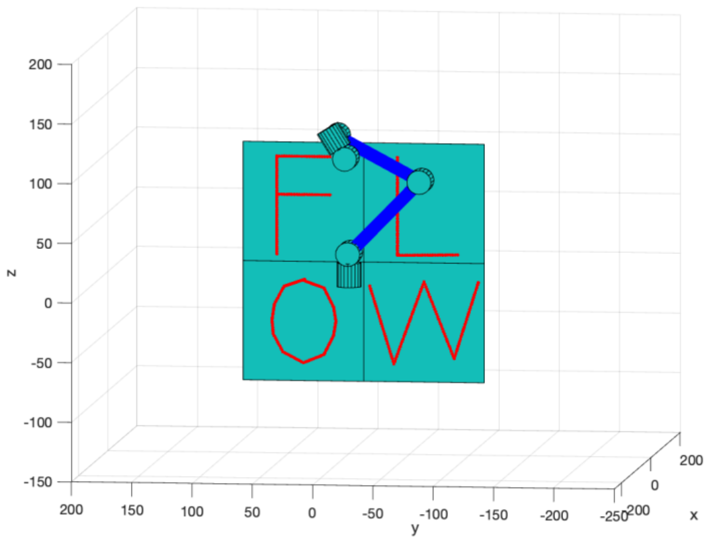


Figure 26. The result of forwards and backwards method writing

On the other hand, the precise controlling of the end-effector in terms of its speed are unlikely to achieved. Moreover, although the end-effector ought to move to the location where it has been assigned, the discrepancies are still existing due to unavoidable internal noise and error in MATLAB. The figure below is the error tracking between the assigned position and the calculated end-effector position. In general, the error is small enough and it approves that the method is reliable.

5.3 Jacobian method

The Jacobian method enable us to control the speed of the end-effector. By first calculating the Jacobian matrix based on angle of each joint, the inverse Jacobian matrix will be utilized to derive amount of angle changed for each joint. Then the new updated angle of each joint will be fed to forward kinematic again to find position of each joint. Here is result of simulation using Jacobian method (Figure 26).

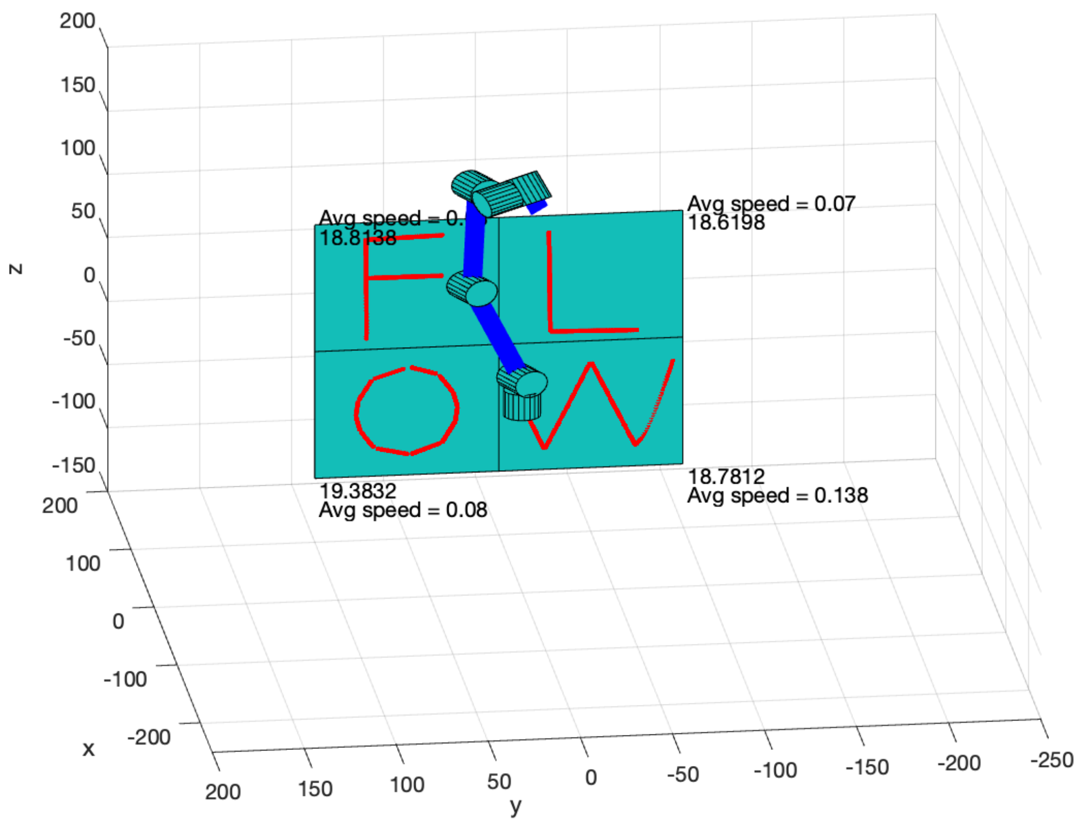


Figure 27. The time need to write every character

The letter ‘F’, ‘L’, ‘W’ can be written precisely and stalely, except letter ‘O’. From the result, it is clear that the fluctuations of the end-effector in x, y, and z direction are large and consistent when the robot is writing letter ‘O’. The reason may cause such unstableness is the way how letter ‘O’ is constructed. As discussed before, the reference points are marked on the path where the letter ‘O’ will be written on. The number of reference points are higher than it of other letters. Additionally, the path of letter ‘O’ is not consistent in either y-axis nor z-axis direction which make the calculation of trajectory complex. Another reason that leads to such chaos is the present of singularities when the Jacobian matrix is calculated. Singularity of robotic arm is a particular point where the robotic arm loses certain degree of freedom and can not move in certain direction. Therefore, the robotic arm may be immovable and turn itself to another direction when singularity is reached.

5.4 Comparisons of Jacobian and F&B method.

Although both methods can achieve writing letters and complete the task, the two methods are very different. The utilization of inverse kinematic enable the robotic arm to derive the exact theoretical angle of motion after the position of end-effector is assigned. During implementation, even though the trajectory errors are still existing, the overall performance of inverse kinematic is wonderful for both linear motion and curve motion. However, one of challenge that inverse kinematic faces is the nonlinear mapping between configuration space and workspace. In another word, the straight line in robot's joint space will be mapped to a curve in its workspace which cause continuous changing in workspace. Therefore, nonlinear calculations are conducted and lead to nonunique solutions. For example, there are multiple solutions for the angle of motion for each joint. Another disadvantages of inverse kinematic is that this method requires active planning of the entire motion path, which is feasible for simple tasks, but not advisable for complex motions.

For the Jacobian method, we can directly control the speed of the joint without caring about the trajectory between the starting point and the target point, which is more in line with actual needs. However, from the simulation, the Jacobian method tends to result in more error when compares with inverse kinematic method, especially when drawing the circular trajectory. Moreover, the more the curvatures a path have, more error will be generated during calculation [4]. Indeed, the error can be decreased as the step time used in Jacobina calculation increased, but the calculation time will correspondingly increase.

Chapter 6. Discussion and Improvement

This project basically implements the task of writing out the FLOW letters in MATLAB simulation of a robotic arm. However, as you can see from the diagram, the trajectory of the letter "F" "L" "W" can be achieved perfectly, but the letter "O" performed worse by using Jacobina method which mainly caused by singularities. This problem may lead to inaccurate control of the robot arm and result in scattered points. Then, it may also have had problems due to trajectory planning. So if we want to have a better result we need find other ways to deal with singularities problem. We can find some solution in article [3].

To improve the robust of this control system, we need to optimise the control algorithm, which means try to optimise the algorithm parameters, improve the controller response speed and ensure the stability and accuracy of the control algorithm. It also needs to optimise the motion planning, checking the algorithm and parameter settings for motion planning to ensure that the planned trajectory is smooth and of the right interval distance. This will improve the control accuracy and motion stability of the robot arm. Moreover, the Proportional-Integral-Derivative (PID) controller can be applied to minimize the error between actual and desired position of robotic end-effector. By consider the position error as feedback information, the PID controller enable the end-effector manipulate towards desired position progressively. But the integration of PID control and inverse kinematic/Jacobian control method ought to require more study in future.

Reference

[1] R. Keating and N. J. Cowan, "M.E. 530.646 UR5 Inverse Kinematics," Johns Hopkins University, Baltimore, MD, USA, 2016. [Online]. Available: https://hml-robotics.jhu.edu/assets/pdfs/ur5-ik.pdf. [Accessed: Apr. 15, 2023].

[2] F. Arenas-Rosales, F. Martell-Chavez, I. Y. Sanchez-Chavez, and C. A. Paredes-Orta, "Virtual UR5 Robot for Online Learning of Inverse Kinematics and Independent Joint Control Validated with FSM Position Control," 2021 IEEE International Conference on Mechatronics, Electronics and Automotive Engineering (ICMEAE), Morelos, Mexico, 2021, pp. 81-86. doi: 10.1109/ICMEAE51454.2021.9522605

[3] L. H. Nguyen, N. N. Le, N. T. T. Vu, and H. P. H. Pham, "Singularity Analysis and Complete Methods to Compute the Inverse Kinematics for a 6-DOF UR/TM-Type Robot," 2020 IEEE 6th International Conference on Control, Automation and Robotics (ICCAR), Singapore, Singapore, 2020, pp. 165-170. doi: 10.1109/ICCAR49630.2020.9096229.

[4] A. A. Hayat, "Robot manipulation through inverse kinematics," 2015 International Conference on Robotics and Artificial Intelligence (ICRAI), Dubai, United Arab Emirates, 2015, pp. 1-5. doi: 10.1109/ICRAI.2015.7414664