National University of Singapore
Facility of Engineering
Department of Mechanical Engineering

Project of Advanced Robotics

ME5402 CA I

Student Name:	Sun Zhanhong A0225282J
	Cao Yuhong A0219126J
	Pang Yatian A0225452L
Supervisor:	Prof. Chui Chee Kong

1. Problem 1

In this section, two sub-questions about basic rotation as well as change of representation are gave. The first problem is to derive the rotation matrix based on given angles about different axis respectively. Based on the problem given, we extract it and wrote a general function that one can get the rotation matrix not only about axis y and x, but about all three axis, and can define the sequence. For the second problem, we simply use the formula derived in class.

1.1. Rotation matrix derivation

In this section of code we have the main code and a function. The main code is used to input the basic parameters, and the function is used to compute and output the resulted rotation matrix.

Algorithm 1: Main code of problem 1 (a).

```
%% Problem 1 sub-problem 1a
% about y by alpha
% about x by beta
% define a function to give the rotation matrix
beta   = pi/4;     % about x axis
alpha  = pi/6;     % about y axis
gamma  = 0;  % about z axis
order = [2,1,3];  % order of rotation(sequence:beta,alpha,gamma) x-y-z
rot_matrix = y_x_rot(beta, alpha, gamma, order)  % in order x-y-zx
```

For this section, the rotating angle about three axis are defined. Also there is a order matrix to define the order of rotation because we want to write a general function. The input of function *y_x_rotation* is the three angles and the order. The formula is:

$$\mathcal{R}_x(\theta_x) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x \\ 0 & \sin\theta_x & \cos\theta_x \end{bmatrix}$$

$$\mathcal{R}_y(\theta_y) = \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{bmatrix}$$

$$\mathcal{R}_z(\theta_z) = \begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 \\ \sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

And based on the formula of three angles we writing the following function:

Algorithm 2: Function code of problem 1 (a).

```
  function rot_mat = y_x_rot(beta,alpha,gamma,order)
  rot(:,:,1) = [    1     ,      0      ,      0     ;...
                    0     ,  cos(beta) ,  -sin(beta);...
                    0     ,  sin(beta) ,  cos(beta)];
```

```
rot(:,:,2) = [cos(alpha) ,      0      ,  sin(alpha);...
                   0     ,      1      ,      0     ;...
             -sin(alpha),      0      , cos(alpha)];
rot(:,:,3) = [ cos(gamma), -sin(gamma) ,      0     ;...
             sin(gamma), cos(gamma)  ,      0     ;...
                   0     ,      0      ,      1    ];
rot_mat = eye(3,3);
for i = 1:3
rotation = rot(:,:,find(order==i));
rot_mat = rotation*rot_mat;
end
```

By plugging in the data given in the problem, we get the resulted rotation matrix:

$$\begin{bmatrix} 0.8660 & 0 & 0.5000 \\ 0.3536 & 0.7071 & -0.6124 \\ -0.3536 & 0.7071 & 0.6124 \end{bmatrix}$$

1.2. Representation transformation

We have two tasks: 1) rotation matrix to RPY angles and 2) RPY angles to rotation matrix representation.

Algorithm 3: Main code of problem 1 (b).

```
 %% Problem 1 sub-problem 1b
 % rotation matrix to XY
r11 = 0.866; r12 = -0.5; r13 = 0;
r21 = 0.5; r22 = 0.866; r23 = 0;
r31 = 0; r32 = 0; r33 = 1;
R = [r11,r12,r13;r21,r22,r23;r31,r32,r33]; % easy to check and change
XYZ = rot_to_XYZ(R);
XYZ1 = XYZ(1,:)  % solution 1 - sequence x-y-z
XYZ2 = XYZ(2,:)  % solution 2 - sequence x-y-z
 %% Problem 1 sub-problem 1b
 % XYZ to rotation matrix
gamma  = 0   ; % x
beta   = 0   ;  % y
alpha  = pi/6   ; % z
rot = xyz_to_rot(alpha,beta,gamma)
```

Based on the formula given in class:

$$\beta = A\tan 2\left(-r_{31}, \pm\sqrt{r_{11}^2 + r_{21}^2}\right) \quad \alpha = A\tan 2\left(\frac{r_{21}}{c\beta}, \frac{r_{11}}{c\beta}\right) \quad \gamma = A\tan 2\left(\frac{r_{32}}{c\beta}, \frac{r_{33}}{c\beta}\right)$$

, we write the following function:

Algorithm 4: Function code of problem 1 (b).

```
function XYZ = rot_to_XYZ(R)
r11 = R(1,1); r12 = R(1,2); r13 =R(1,3);
```

```
r21 = R(2,1); r22 = R(2,2); r23 = R(2,3);
r31 = R(3,1); r32 = R(3,2); r33 = R(3,3);
% since there are 2 solns in general,
% we seperate 2 betas into beta1 and beta2
beta1 = atan2(-r31, sqrt(r11^2+r21^2));
beta2 = atan2(-r31,-sqrt(r11^2+r21^2));
alpha1 = atan2(r21/cos(beta1),r11/cos(beta1));
alpha2 = atan2(r21/cos(beta2),r11/cos(beta2));
gamma1 = atan2(r32/cos(beta1),r33/cos(beta1));
gamma2 = atan2(r32/cos(beta2),r33/cos(beta2));
XYZ = round([gamma1,beta1,alpha1;...
gamma2,beta2,alpha2]./pi.*180);
```

Finally we get our two solutions:

$$\begin{bmatrix} 0 & 0 & 30 \end{bmatrix}$$

and

$$\begin{bmatrix} -180 & -180 & -150 \end{bmatrix}$$

For another question in 1(b), it simply requires a formula derived in class:

$$\begin{aligned} {}^{A}_{B}R_{XYZ}(\gamma,\beta,\alpha) &= R_Z(\alpha)R_Y(\beta)R_X(\gamma) \\ &= \begin{bmatrix} c\alpha & -s\alpha & 0 \\ s\alpha & c\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c\beta & 0 & s\beta \\ 0 & 1 & 0 \\ -s\beta & 0 & c\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\gamma & -s\gamma \\ 0 & s\gamma & c\gamma \end{bmatrix} \\ &= \begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{bmatrix} \end{aligned}$$

Then we write the following main code and a function to derive the resulted rotation matrix:

Algorithm 5: Main code and function of problem 1 (b) section 2.

```
% XYZ to rotation matrix
gamma  = pi/4   ; % x
beta   = pi/6   ; % y
alpha  = 0   ; % z
rot = xyz_to_rot(alpha,beta,gamma)

function rot = xyz_to_rot(alpha,beta,gamma)
rot = [cos(alpha)*cos(beta) , cos(alpha)*sin(beta)*sin(gamma)-sin(alpha)*cos(gamma),
cos(alpha)*sin(beta)*cos(gamma)+sin(alpha)*sin(gamma);sin(alpha)*cos(beta),
sin(alpha)*sin(beta)*sin(gamma)+cos(alpha)*cos(gamma),
sin(alpha)*sin(beta)*cos(gamma)-cos(alpha)*sin(gamma);
-sin(beta),cos(beta)*sin(gamma),cos(beta)*cos(gamma)];
```

We set the angles of x, y, and z as $\frac{\pi}{4}$, $\frac{\pi}{6}$ and 0 respectively. The result is:

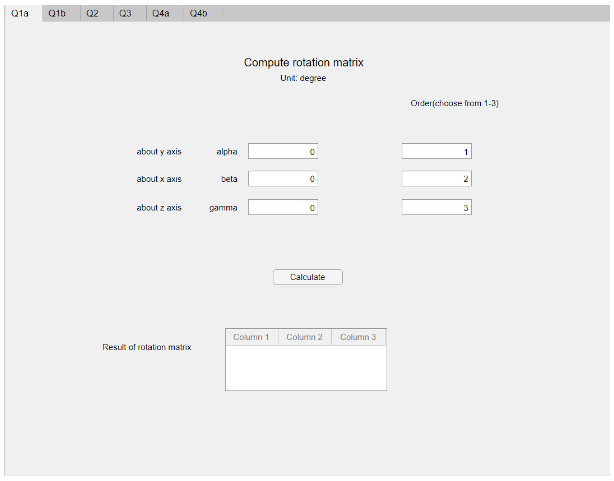
$$\begin{bmatrix} 0.8660 & 0.3536 & 0.3536 \\ 0 & 0.7071 & -0.7071 \\ -0.5000 & 0.6124 & 0.6124 \end{bmatrix}$$

This result is different from that of problem 1(a) because of the difference of rotation sequence.

1.3. GUI for problem 1

Based on the work above, we design and realize a graphical user interface (GUI) program for problem 1.

Below is GUI for problem 1.a. To compute rotation matrix, we input the angles of gamma, beta and alpha. Then, use 1-3 to represent the sequence of rotation. Press the button to get result of rotation matrix.



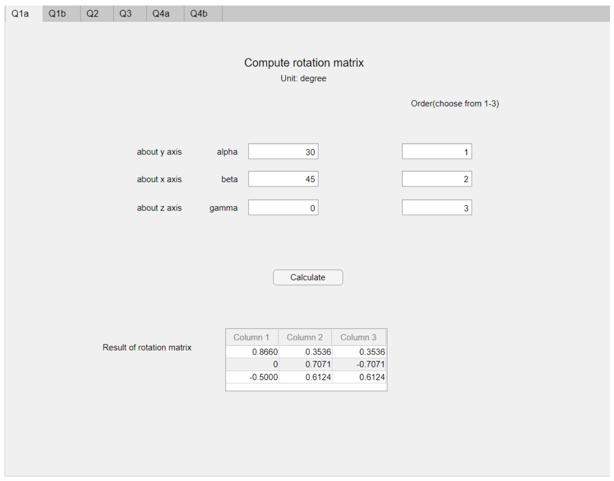
Compute rotation matrix
Unit: degree

			Order(choose from 1-3)
about y axis	alpha	0	1
about x axis	beta	0	2
about z axis	gamma	0	3

Result of rotation matrix

Column 1	Column 2	Column 3

(a) GUI display



Compute rotation matrix
Unit: degree

			Order(choose from 1-3)
about y axis	alpha	30	1
about x axis	beta	45	2
about z axis	gamma	0	3

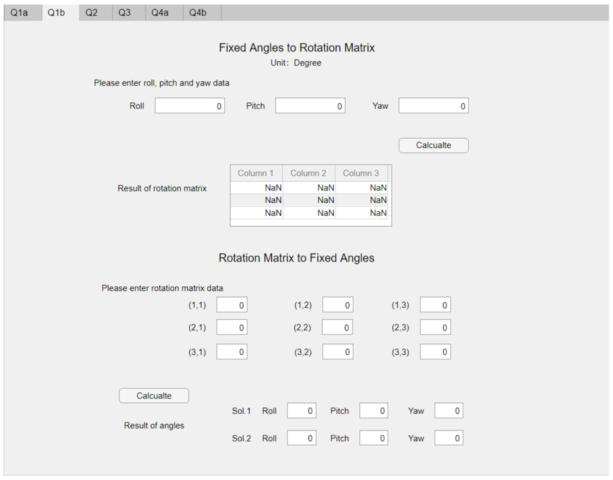
Result of rotation matrix

Column 1	Column 2	Column 3
0.8660	0.3536	0.3536
0	0.7071	-0.7071
-0.5000	0.6124	0.6124

(b) Compute rotation matrix

Figure 1: GUI for problem 1.a

Below is GUI for problem 1.a.We input fixed angles (pitch, roll and yaw) and press calculate button to get corresponding rotation matrix. To get fixed angles from rotation matrix, we input the rotation matrix and press calculate button to get two results of angles.



Fixed Angles to Rotation Matrix

Unit: Degree

Please enter roll, pitch and yaw data

Roll 0 Pitch 0 Yaw 0

Result of rotation matrix

Column 1	Column 2	Column 3
NaN	NaN	NaN
NaN	NaN	NaN
NaN	NaN	NaN

Rotation Matrix to Fixed Angles

Please enter rotation matrix data

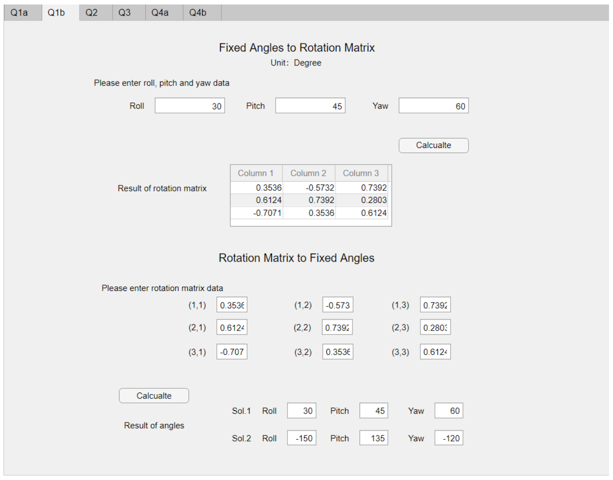
(1,1)	0	(1,2)	0	(1,3)	0
(2,1)	0	(2,2)	0	(2,3)	0
(3,1)	0	(3,2)	0	(3,3)	0

Result of angles

Sol.1 Roll 0 Pitch 0 Yaw 0

Sol.2 Roll 0 Pitch 0 Yaw 0

(a) GUI display



Fixed Angles to Rotation Matrix

Unit: Degree

Please enter roll, pitch and yaw data

Roll 30 Pitch 45 Yaw 60

Result of rotation matrix

Column 1	Column 2	Column 3
0.3536	-0.5732	0.7392
0.6124	0.7392	0.2803
-0.7071	0.3536	0.6124

Rotation Matrix to Fixed Angles

Please enter rotation matrix data

(1,1)	0.353	(1,2)	-0.573	(1,3)	0.739
(2,1)	0.612	(2,2)	0.739	(2,3)	0.280
(3,1)	-0.707	(3,2)	0.353	(3,3)	0.612

Result of angles

Sol.1 Roll 30 Pitch 45 Yaw 60

Sol.2 Roll -150 Pitch 135 Yaw -120

(b) Transformation between fixed angle and rotation matrix

Figure 2: GUI for problem 1.b

2. Problem 2

2.1. D-H Table Derivation

We can define the parameters in D-H table through imaging the rotation and translation motion respectively. From frame $i-1$ to frame i, we can operate by four steps: rotation about z_{i-1} by θ_i, translation along z_{i-1} by d_i, rotation about x_{i-1} by α_i, and finally translation along x_{i-1} by a_i, then we can get to frame i. These four operations can all be define by a single-parameter transformation. By multiplying the four transformation matrices together, we can get the transformation matrix from frame 0 to any frame we want.

Tabla 1: D-H table of PUMA 600

i	α_i	a_i	d_i	θ_i
1	$\frac{\pi}{2}$	0	l_0	θ_1
2	0	l_1	$-l_{offset}$	θ_2
3	$\frac{\pi}{2}$	0	0	θ_3
4	$-\frac{\pi}{2}$	0	l_2	θ_4
5	$\frac{\pi}{2}$	0	0	θ_5
6	0	0	l_3	θ_6

Given this D-H table, we can derive the transformation matrices with the formula given in class:

$$ {}^{i-1}_{i}A = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Here we write a function to get the transformation matrix from frame $i-1$ to frame i by simply use the above formula:

Algorithm 6: Function to get transformation matrix from D-H parameters.

```
function rot = DH_rot(joint)
alpha = joint.DH(1);
a = joint.DH(2);
d = joint.DH(3);
theta = joint.DH(4);
rot = [cos(theta)  , -sin(theta)*cos(alpha)  , sin(theta)*sin(alpha)  , a*cos(theta);
       sin(theta)  ,  cos(theta)*cos(alpha)  , -cos(theta)*sin(alpha)  , a*sin(theta);
           0       ,        sin(alpha)       ,         cos(alpha)     ,     d      ;
           0       ,            0            ,             0          ,     1      ;];
```

And if we input the D-H table into the program, we can get the kinematic equation, which is also the transformation matrix between the world frame(frame 0) and the end-effector frame.

Algorithm 7: Main code to get transformation matrix from D-H parameters.

```
1  %% Problem 2 sub-problem 2 PUMA 600
2  % DH table of PUMA600
3  syms theta1 theta2 theta3 theta4 theta5 theta6 % joint variables
4  syms l0 l1 l2 l3 l_offset
5  l0 = 500;
6  l1 = 500;
7  l2 = 400;
8  l3 = 150;
9  l_offset = 200;
10 % joint 1
11 joint(1).DH = [pi/2, 0, l0, theta1];
12 joint(1).type = 2; % trans:1 rot:2
13 % joint 2
14 joint(2).DH = [0, l1, -l_offset, theta2];
15 joint(2).type = 2; % trans:1 rot:2
16 % joint 3
17 joint(3).DH = [pi/2, 0, 0, theta3];
18 joint(3).type = 2; % trans:1 rot:2
19 % joint 4
20 joint(4).DH = [-pi/2, 0, l2, theta4];
21 joint(4).type = 1; % trans:1 rot:2
22 % joint 5
23 joint(5).DH = [pi/2, 0, 0, theta5];
24 joint(5).type = 2; % trans:1 rot:2
25 % joint 6
26 joint(6).DH = [0, 0, l3, theta6];
27 joint(6).type = 2; % trans:1 rot:2
28 for i = 1:6
29 joint(i).rot = DH_rot(joint(i));
30 end
31 % till now we have defined the DH table and all transformation matrices
32 % we can get the kinematical function easily by multiplying them together
33 Kinem = eye(4,4);
34 for i = 1:6
35 Kinem = Kinem*joint(i).rot;
36 end
```

In line 6-10 of the belowed source code, we defined the PUMA600 dimensions on our own. By doing so, we can get the final result. Since the result related to all joint variables is too complex, we do not show it here. If interested in the result, one can run the code of this part.

Here we plot a example of the manipulators of PUMA600.

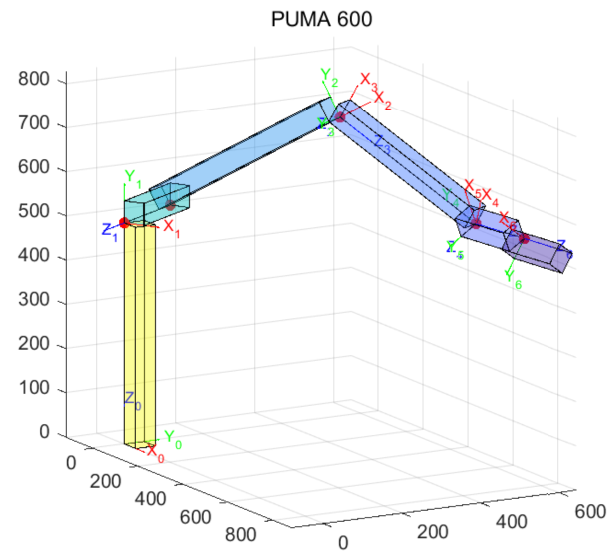


Figure 3: A example of the manipulators of PUMA600

2.2. Inverse Kinematics of the PUMA 600

To make the result easier to observe, we define all rotational joint variables equals to $\frac{\pi}{6}$. First we plug all the joint variables into the forward kinematic equation

$$ {}^0_6T = {}^0_1A \cdot {}^1_2A \cdot {}^2_3A \cdot {}^3_4A \cdot {}^4_5A \cdot {}^5_6A $$

and get the resulted end-effector frame:

$$ T = \begin{bmatrix} 0.2522 & 0.1044 & 0.9620 & 719.3029 \\ -0.7874 & -0.5558 & 0.2667 & 602.9285 \\ 0.5625 & -0.8248 & -0.0580 & 541.2981 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix} $$

Here we propose two kinds of solutions to solve inverse kinematics. The first solution is to calculate the concrete value of joint variables in the process and The second solution is to calculate the general expressions of the joint variables and substitute the concrete values at the end. For the first solution,firstly we move ${}^1_0A$ to the left hand side of equation. Solve the equation to get the joint variables.

$$ {}^1_0A^{-1} \cdot {}^0_6T = {}^1_6T $$

Solve the equation above in Matlab to get the joint 1 variable θ_1.

```
 % first move the inverse of T1 to the LHS of equation
lhs1 = simplify(inv(joint(1).rot))*T;
Kinem = eye(4,4);
for i = 2:6
Kinem = Kinem*joint(i).rot;
end
```

```
Kinem = simplify(Kinem)
```

We find that the entry $(3,3)$ and $(3,4)$ of the equation can actually be simplified into a single-variable equation which can be used to solve θ_1 easily.

```
eq = subs(Kinem(3,4),sin(theta4),lhs1(3,3)/sin(theta5));
ans = solve(lhs1(3,4)==eq);
jointv1 = real(double(ans(2)));
```

Here we can actually get two answers of θ_1: -2.1156 and 0.5236. This is the multi-answer of inverse kinematic. We leave the discussion to the third part of this problem. Here we choose the more reasonable answer 0.5236 and continue our inverse solution.

After this, we move ${}^{2}_{1}A$ and ${}^{3}_{2}A$ to the LHS of equation

$$ {}^{3}_{2}A^{-1} \cdot {}^{2}_{1}A^{-1} \cdot {}^{1}_{6}T = {}^{3}_{6}T $$

And solve other variables in Matlab after plugging in the variables we have solved.

```
lhs2 = simplify(subs(inv(joint(1).rot*joint(2).rot*joint(3).rot)*T,theta1,jointv1))
Kinem = eye(4,4);
for i = 4:6
Kinem = Kinem*joint(i).rot;
end
```

By observing, we find that we can solve some joint variables by solving entries: $(1,3)$, $(1,4)$, $(2,3)$, and $(2,4)$. Therefore, we use **fsolve** to solve the quaternary nonlinear equations.

```
fun = @root;
x0 = [pi/6,pi/6,pi/6,pi/6];
x = fsolve(fun,x0);
jointv2 = x(1);
jointv3 = x(2);
jointv4 = x(3);
jointv5 = x(4);
```

Here the root function is obtained by copy and palse into a function code. After this operation, we can easily get the last joint variable θ_6 by solving equation ${}^{5}_{6}A^{-1} \cdot {}^{5}_{0}T = {}^{6}_{5}T$

```
Kinem = subs(Kinem,[theta2,theta3,theta4,theta5],[jointv2,jointv3,jointv4,jointv5]);
lhs = subs(lhs2,[theta2,theta3],[jointv2,jointv3]);
ans = solve(lhs(1,1)==Kinem(1,1));
jointv6 = double(ans(1));
```

The result of inverse is:

$$ \begin{bmatrix} 0.5236 \\ 0.5237 \\ 0.52343 \\ 0.52362 \\ 0.52366 \\ 0.52361 \end{bmatrix} $$

which is the same as the variables we set when doing forward kinematics in a reasonable tolerance. Now we propose a general solution for the inverse kinematics of PUMA 600, which gives the expressions of the joint variables.

For this PUMA 600 manipulator, the last three joints are all rotational joints, and the joint axis intersects at one point. Here we call the intersection point wrist point W.

It is easy to find a relation: Given an end-effector matrix, the wrist point W is also fixed.

Assume the end-effector matrix is represented by a translation matrix p and a rotation matrix R, and

$$R = \begin{bmatrix} n & o & a \end{bmatrix},$$

which represents the orientation of frame 6.

Then, it is obvious that the position of wrist is:

$$W = p - d_6 a$$

This represents three-dimensional equations only related to angle $\theta_1, \theta_2, \theta_3$. We can easily solve the equations and get the values of θ_1, θ_2, and θ_3.

```
Kinem = eye(4,4);
for i = 1:4
   Kinem = Kinem*joint(i).rot;
end
T04 = Kinem; % wrist

Pw04 = T04(1:3,4);
Pw64 = T(1:3,4)-l3*T(1:3,3);

ans = solve(-sin(theta1)*Pw64(1)+cos(theta1)*Pw64(2)==l_offset, theta1);
theta1_1 = real(double(ans(1)));
theta1_2 = real(double(ans(2)));
if theta1_1>=0 && theta1_1<=2*pi
   thetav1 = theta1_1;
else
   thetav1 = theta1_2;
end
A=cos(thetav1)*Pw64(1)+sin(thetav1)*Pw64(2);
B=(A^2+l1^2+(l0-Pw64(3))^2-l2^2)/(2*l1);
theta2 = atan(A/(l0-Pw64(3)))-atan(B/(A^2+(l0-Pw64(3))^2-B^2)^0.5);
if theta2 <= 0
   thetav2 = theta2+pi;
else
   thetav2 = theta2;
end
theta3=atan((A-l1*cos(thetav2))/(l0+l1*sin(thetav2)-Pw64(3)))-thetav2;
if theta3 <= 0
   thetav3 = theta3+pi;
else
   thetav3 = theta3;
end
```

Then, in order to solve $\theta_4, \theta_5, \theta_6$, we write a equation corresponding to R:

$$R = {}^0_3R\,(q_1, q_2, q_3)\,{}^3_6R\,(q_4, q_5, q_6)$$

Since we have solved q_1, q_2, q_3 now, we move all constant values to the left hand side of equation:

$$\Rightarrow {}^3_6R\,(q_4, q_5, q_6) = {}^0_3R^T R$$

With this step, we can solve all joint variables. Note that for this equation, the orientation equation can be represented by Euler ZYZ or ZXZ rotation matrix with only q_4, q_5, q_6, which gives two solutions.

```
    theta4=atan((T(1,3)*sin(thetav1)-T(2,3)*cos(thetav1))/(T(1,3)*cos(thetav1)*cos(thetav1+
      ↪ thetav2)+T(2,3)*sin(thetav1)*cos(thetav1+thetav2)+T(3,3)*sin(thetav1+thetav2)));
if theta4 <= 0
    thetav4 = theta4+pi;
else
    thetav4 = theta4;
end
theta5=acos(T(1,3)*cos(thetav1)*sin(thetav1+thetav2)+T(2,3)*sin(thetav1)*sin(thetav1+thetav2)-T
      ↪ (3,3)*cos(thetav1+thetav2));
if theta5 <= 0
    thetav5 = theta5+pi;
else
    thetav5 = theta5;
end
theta6=atan((T(1,2)*cos(thetav1)*sin(thetav1+thetav2)+T(2,2)*sin(thetav1)*sin(thetav1+thetav2)-
      ↪ T(3,2)*cos(thetav1+thetav2))/-(T(1,1)*cos(thetav1)*sin(thetav1+thetav2)+T(2,1)*sin(
      ↪ thetav1)*sin(thetav1+thetav2)-T(3,1)*cos(thetav1+thetav2)));
if theta6 <= 0
    thetav6 = theta6+pi;
else
    thetav6 = theta6;
end
```

By now we get all joint variables, which is more precise comparing to the result from the first solution. The result is as below:

$$\begin{bmatrix} 0.5236 \\ 0.5236 \\ 0.5236 \\ 0.5236 \\ 0.5236 \\ 0.5236 \end{bmatrix}$$

In the following section, we will discuss the number of solutions when doing the inverse kinematics given the position of end-effector.

2.3. Solutions of Inverse Kinematics

Above in section 2.2 is a general statement of the wrist point. Now we go into detail about the number of solutions of inverse kinematics of PUMA 600 using the wrist point.

In this section, we assume the end-effector is in its working space.

Given the transformation matrix of end-effector:

$$ {}_6^0T = \begin{bmatrix} n_x & a_x & o_x & p_x \\ n_y & a_y & o_y & p_y \\ n_z & a_z & o_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}, $$

The inverse kinematics needs to solve for all joint variables: $\theta_1 - \theta_6$. First, find solutions for $\theta_1 - \theta_3$. According to the frame, the center of wrist is at the intersection of the three terminal revolute axes, which is the origin of frame 4. Hence, the position of the wrist can be determined by:

$$ {}_4^0T = \begin{bmatrix} n_{wx} & a_{wx} & o_{wx} & p_{wx} \\ n_{wy} & a_{wy} & o_{wy} & p_{wy} \\ n_{wz} & a_{wz} & o_{wz} & p_{wz} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} n_{wx} & a_{wx} & o_{wx} & l_2c_1s_{23} - l_{offset}s_1 + l_1c_1c_2 \\ n_{wy} & a_{wy} & o_{wy} & l_2s_1s_{23} + l_{offset}c_1 + l_1s_1c_2 \\ n_{wz} & a_{wz} & o_{wz} & l_0 - l_2c_{23} + l_1s_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Besides, based on geometry, the position of the wrist also equals to:

$$ \begin{bmatrix} p_{wx} \\ p_{wy} \\ p_{wz} \end{bmatrix} = \begin{bmatrix} p_x - l_3a_x \\ p_y - l_3a_y \\ p_z - l_3a_z \end{bmatrix} $$

There is a potential relation:

$$ -s_1p_{wx} + c_1p_{wy} = l_{offset} $$

This is an equation with only one variable θ_1. Generally, since the left hand side of this equation is a sin-type periodic function with period of 2π, and the right hand side of equation is a constant function in the working space, θ_1 has **two solutions**.

Similarly,

$$ c_1p_{wx} + s_1p_{wy} = l_2s_{23} + l_1c_2 $$

This is a equation of $\theta_2 + \theta_3$ and θ_2. We need another equation to solve these two variables:

$$ p_{wz} = l_0 - l_2c_{23} + l_1s_2 $$

Let $c_1p_{wx} + s_1p_{wy} = A$, and $l_0 - p_{wz} = B$

$$ s_{23}^2 + c_{23}^2 = \left(\frac{A - l_1c_2}{l_2}\right)^2 + \left(\frac{l_1s_2 + B}{l_2}\right)^2 = 1 $$

Then,

$$ 2Bl_1s_2 - 2Al_1c_2 - l_2^2 = l_2^2 - l_1^2 - A^2 - B^2 $$

Similarly, since it is in working space, this equation should have **two solutions**.

Then, in order to solve θ_3 with θ_2 and θ_1,

$$ \tan(\theta_2 + \theta_3) = \frac{s_{23}}{c_{23}} = \frac{A - l_1c_2}{B + l_1s_2} $$

Since θ_2 has already fixed, this gives only **one solution** for θ_3.

Then the task is to solve for $\theta_4, \theta_5, \theta_6$. With the variables that has been solved.

$$ {}^3_6R = \left({}^0_3R\right)^{-1} {}^0_6R = \left({}^0_3R\right)^T {}^0_6R $$

With analytical calculation,

$$ {}^3_6R = \begin{bmatrix} c_4c_5c_6 - s_4s_6 & -c_6s_4 - c_4c_5s_6 & c_4s_5 \\ c_4s_6 + c_5c_6s_4 & c_4c_6 - c_5s_4s_6 & s_4s_5 \\ -c_6s_5 & s_5s_6 & c_5 \end{bmatrix} $$

$$ \left({}^0_3R\right)^T {}^0_6R $$

$$ = \begin{bmatrix} c_1c_{23} & s_1c_{23} & s_{23} \\ s_1 & -c_1 & 0 \\ c_1s_{23} & s_1s_{23} & -c_{23} \end{bmatrix} \begin{bmatrix} n_{wx} & o_{wx} & a_{wx} \\ n_{wy} & o_{wy} & a_{wy} \\ n_{wz} & o_{wz} & a_{wz} \end{bmatrix} $$

$$ = \begin{bmatrix} n_{wx}c_1c_{23} + n_{wy}s_1c_{23} + n_{wz}s_{23} & o_{wx}c_1c_{23} + o_{wy}s_1c_{23} + o_{wz}s_{23} & a_{wx}c_1c_{23} + a_{wy}s_1c_{23} + a_{wz}s_{23} \\ n_{wx}s_1 - n_{wy}c_1 & o_{wx}s_1 - o_{wy}c_1 & a_{wx}s_1 - a_{wy}c_1 \\ n_{wx}c_1s_{23} + n_{wy}s_1s_{23} - n_{wz}c_{23} & o_{wx}c_1s_{23} + o_{wy}s_1s_{23} - o_{wz}c_{23} & a_{wx}c_1s_{23} + a_{wy}s_1s_{23} - a_{wz}c_{23} \end{bmatrix} $$

Solve the equation,

$$ \tan\theta_4 = \frac{s_4s_5}{c_4s_5} = \frac{a_{wx}s_1 - a_{wy}c_1}{a_{wx}c_1c_{23} + a_{wy}s_1c_{23} + a_{wz}s_{23}} $$

then

$$ \theta_4 = \arctan\left(\frac{a_{wx}s_1 - a_{wy}c_1}{a_{wx}c_1c_{23} + a_{wy}s_1c_{23} + a_{wz}s_{23}}\right) $$

This gives **two solutions** for θ_4.

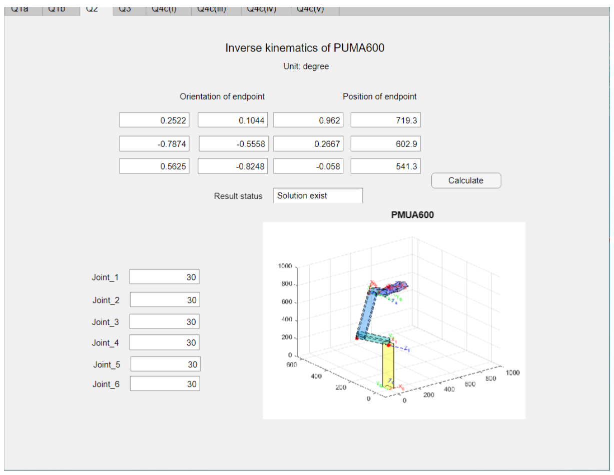
For now, as illustrated earlier, $\theta_4, \theta_5, \theta_6$ have co-relation of intersecting the axis at one point. Therefore, two solutions of θ_4 means two solution sets of $\theta_4, \theta_5, \theta_6$.

Finally, in total, there are **8 solutions**.

2.4. GUI for problem 2

Based on the work above, we design and realize a graphical user interface (GUI) program for problem 2.

To compute inverse kinematics of PUMA600, we input the orientation and position of endpoint and press calculate button. If there is a solution, result status will show solution exist, with corresponding joint parameters and overall figure being shown. If the input data of endpoint is not in the workspace, result status will show out of workspace.



Inverse kinematics of PUMA600

Unit: degree

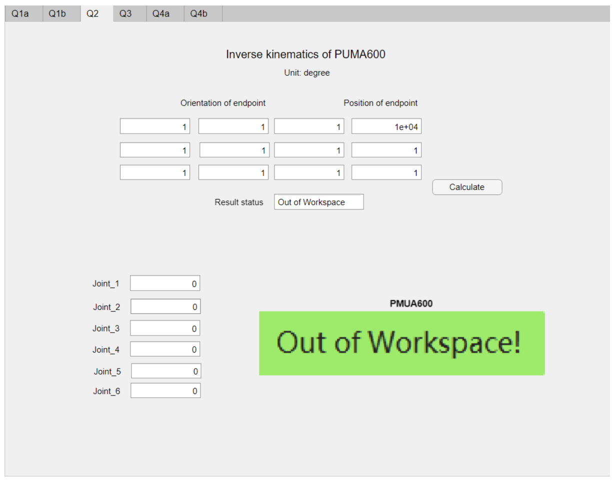
Orientation of endpoint			Position of endpoint
0.2522	0.1044	0.962	719.3
-0.7874	-0.5558	0.2667	602.9
0.5625	-0.8248	-0.058	541.3

Result status: Solution exist

PMUA600

Joint_1	30
Joint_2	30
Joint_3	30
Joint_4	30
Joint_5	30
Joint_6	30

(a) Compute inverse kinematics of PUMA600



Inverse kinematics of PUMA600

Unit: degree

Orientation of endpoint			Position of endpoint
1	1	1	1e+04
1	1	1	1
1	1	1	1

Result status: Out of Workspace

Joint_1	0
Joint_2	0
Joint_3	0
Joint_4	0
Joint_5	0
Joint_6	0

PMUA600

Out of Workspace!

(b) Out of workspace

Figure 4: GUI for problem 2

3. Problem 3

3.1. Jacobian Finding

Given the D-H table of 3-DOF manipulator,

Tabla 2: D-H table of 3-DOF manipulator

i	α_i	a_i	d_i	θ_i
1	0	0	l_1	θ_1
2	$\frac{\pi}{2}$	0	d_2	$\frac{\pi}{2}$
3	0	0	d_3	0

, we can first get the transformation matrices of all frames(links), further we have all parameters needed for Jacobian(unit vector of z-axis b_i and r_i if rotational joint). To be more specific, if the joint is translational,

$$\mathbf{J}_i = \begin{bmatrix} \mathbf{J}_{Li} \\ \mathbf{J}_{Ai} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_{i-1} \\ \mathbf{0} \end{bmatrix},$$

and if it is rotational,

$$\mathbf{J}_i = \begin{bmatrix} \mathbf{J}_{Li} \\ \mathbf{J}_{Ai} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_{i-1} \times \mathbf{r}_{i-1,e} \\ \mathbf{b}_{i-1} \end{bmatrix}.$$

So the code here also follows the step: 1) Based on the D-H table, calculate the transformation matrices; 2) Based on the transformational matrices, calculate the frame representations of all joints; 3) Based on the transformational matrices, calculate the positional vectors of joints $r_{i-1,e}$; 4) Based on frame representations, calculate b_{i-1}s. There steps are in one function:

```
function Ji = cal_Ji(joint,k,j)
  % k is the Jacobian column we wanna calculate
  % j is the total number of joint
  Kinem = eye(4,4);
  if k == 1
  b_i_1 = [0;0;1];
  else
  for i = 1:k-1
  Kinem = Kinem*joint(i).rot;
  end
  b_i_1 = Kinem(1:3,3);
  end
  % b:3x1

  if joint(k).type1 == 1 % if translation
  Ji = [b_i_1;0;0;0]; % 6x1
  end

  if joint(k).type1 == 2 % if rotation
  r_i_1 = Kinem(1:3,4); % r:3x1
  Kinem1 = eye(4,4);
```

```
P_end_effector = eye(4,4);
for i = 1:j
P_end_effector = P_end_effector * joint(i).rot;
end
P_end_effector = P_end_effector(1:3,4);

r = P_end_effector - r_i_1;
Ji = [b_i_1(2)*r(3)-b_i_1(3)*r(2);-b_i_1(1)*r(3)+b_i_1(3)*r(1);b_i_1(1)*r(2)-b_i_1(2)*r(1);
  ↪ b_i_1];
end
```

For representation convinence, we write the main code as follows:

```
%% 3
syms theta1 l1 d2 d3
% joint 1
% joint(1).DH = [theta1 l1 0 0];
joint(1).DH = [0 0 l1 theta1];
joint(1).type1 = 2; % 1:trans 2:rot
% joint 2
% joint(2).DH = [pi/2 d2 0 pi/2];
joint(2).DH = [pi/2 0 d2 pi/2];
joint(2).type1 = 1;
% joint 3
% joint(3).DH = [0 d3 0 0];
joint(3).DH = [0 0 d3 0];
joint(3).type1 = 1;

J = zeros(6,1);
for i = 1:3
joint(i).rot = DH_rot(joint(i));
J = [J,cal_Ji(joint,i,3)];
end
J = J(:,2:4)
```

and we get the Jacobian of this configuration:

$$J = \begin{bmatrix} -d_3 sin\theta_1 & 0 & cos\theta_1 \\ d_3 cos\theta_1 & 0 & sin\theta_1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

3.2. Joint Forces and Torques

Once getting the Jacobian of this manipulator configuration, we also have the relation between the end-effector force and joint forces and torques by simply applying the formula derived in class:

$$\tau = \mathbf{J}^{\mathrm{T}}\mathbf{F}.$$

By assuming the end-effector force:

$$\mathbf{F} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ n_1 \\ n_2 \\ n_3 \end{bmatrix},$$

We can calculate the corresponding joint forces and torques:

```
syms f1 f2 f3 n1 n2 n3
F = [f1;f2;f3;n1;n2;n3];
joint_force = J.'*F
```

and the result is:

$$\tau = \begin{bmatrix} n_3 + d_3 f_2 cos\theta_1 - d_3 f_1 sin\theta_1 \\ f_3 \\ f_1 cos\theta_1 + f_2 sin\theta_1 \end{bmatrix}$$

Then, we can simply plug in the parameters given in problem,

```
  J3 = subs(J,[theta1,d2,d3],[0,1,1]);
  F3 = subs(F,[f1,f2,f3,n1,n2,n3],[1,2,3,0,0,0]);
  joint_force3 = J3.'*F3
```

The result is:

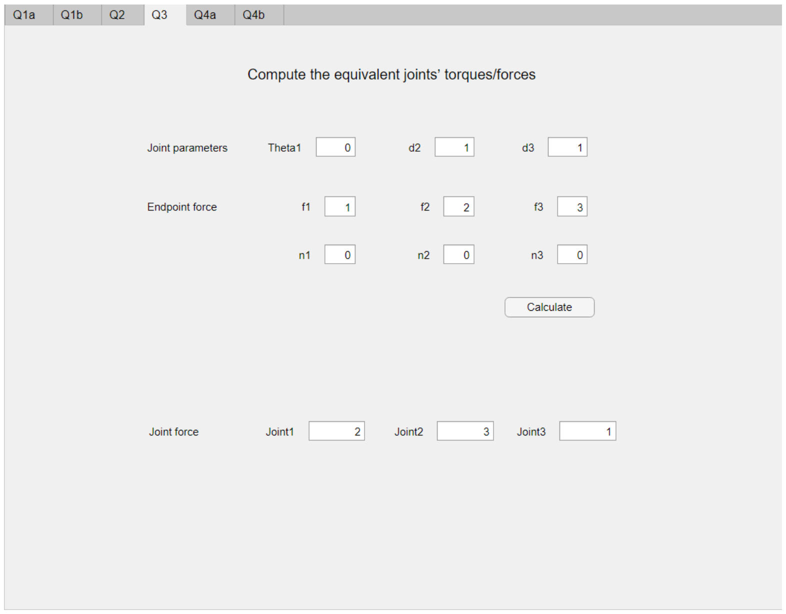
$$\tau = \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$$

This result is the same as doing force analysis directly.

3.3. GUI for problem 3

Based on the work above, we design and realize a graphical user interface (GUI) program for problem 3.

To compute the equivalent joints' forces based on endpoint force, we input the joint parameters and endpoint force. After pressing calculate button, the corresponding joints' forces are shown respectively.



Compute the equivalent joints' torques/forces

Joint parameters	Theta1	0	d2	1	d3	1
Endpoint force	f1	1	f2	2	f3	3
	n1	0	n2	0	n3	0
Joint force	Joint1	2	Joint2	3	Joint3	1

Figure 5: Compute the equivalent joints' torques/forces

4. Problem 4

In this problem, the last three links of PUMA 600 are taken out. For sub-problem a, we are required to derive the Jacobian matrix; in sub-problem b, the equivalent forces and torques are to derive given forces and torques at the tool; and in sub-problem c, given the trajectory of tool, we are required to derive some parameters of the manipulator.

4.1. Jacobian Derivation

In this problem, we follow the steps of deriving a Jacobian in class. First, we derive the D-H table. Although it is already given in problem, frame 3 is inconvenient when solving the problem. So, we add one row under the given D-H table. This is convenient to use our existing code to derive the transformation matrix from frame 3 to point A, and further derive the other quantities.

Tabla 3: D-H table of wrist joints of PUMA 600

i	α_i	a_i	d_i	θ_i
1	$-90°$	0	400	θ_1
2	$+90°$	0	0	θ_2
3	0	0	100	θ_3
(4)	0	100	50	0

Then from the D-H table, we derive the transformation matrix with:

$$ {}_{i}^{i-1}A = {}_{\text{int}}^{i-1}A_{i}^{\text{int}}A = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

. With the transformation matrices, we can get the origin position and also the orientation of such frame. After this we can get the parameters required in the derivation of Jacobian: For translational joints,

$$ \begin{bmatrix} \mathbf{J}_{Li} \\ \mathbf{J}_{Ai} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_{i-1} \\ \mathbf{0} \end{bmatrix} $$

and for rotational joints,

$$ \begin{bmatrix} \mathbf{J}_{Li} \\ \mathbf{J}_{Ai} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_{i-1} \times \mathbf{r}_{i-1,e} \\ \mathbf{b}_{i-1} \end{bmatrix} $$

, where b_{i-1} is the third column of ${}_{i-1}^{0}R$ and $\mathbf{r}_{i-1,e}$ represents the vector pointed from O_{i-1} to the end-effector frame.

Finally, we combine each column and forms the full Jacobian.

The code is realized as follows:

```
%% 4
% 1 & 2
syms theta1 theta2 theta3 real
syms f1 f2 f3 n1 n2 n3
```

```
% joint 1
joint(1).DH = [-pi/2 0 400 theta1];
joint(1).type1 = 2; % 1:trans 2:rot
% joint 2
joint(2).DH = [pi/2 0 0 theta2];
joint(2).type1 = 2;
% joint 3
joint(3).DH = [0 0 100 theta3];
joint(3).type1 = 2;
% tool working point A
joint(4).DH = [0 100 50 0];
% calculate rotation representations
for i = 1:4
joint(i).rot = DH_rot(joint(i));
end
```

Then, we follow the steps of deriving a Jacobian in class and write a function. For this function, the input is the struct joint, the number of Jacobian columns, and the total number of joints, and the output is the corresponding Jacobian column.

```
function Ji = cal_Ji(joint,k,j)
% k is the Jacobian column we wanna calculate
% j is the total number of joint
Kinem = eye(4,4);
if k == 1
b_i_1 = [0;0;1];
else
for i = 1:k-1
Kinem = Kinem*joint(i).rot;
end
b_i_1 = Kinem(1:3,3);
end
% b:3x1

if joint(k).type1 == 1 % if translation
Ji = [b_i_1;0;0;0]; % 6x1
end

if joint(k).type1 == 2 % if rotation
r_i_1 = Kinem(1:3,4); % r:3x1
Kinem1 = eye(4,4);
P_end_effector = eye(4,4);
for i = 1:j
P_end_effector = P_end_effector * joint(i).rot;
end
P_end_effector = P_end_effector(1:3,4);

r = P_end_effector - r_i_1;
Ji = [b_i_1(2)*r(3)-b_i_1(3)*r(2);-b_i_1(1)*r(3)+b_i_1(3)*r(1);b_i_1(1)*r(2)-b_i_1(2)*r(1);
    ↪ b_i_1];
end
```

```
% In main code
J = zeros(6,1);
for i = 1:3
J = [J,cal_Ji(joint,i,3)];
end
J = simplify(J(:,2:4))
```

For this function, we discussed different kinds of conditions including the type of corresponding joint (translational or rotational). For each type of joint we follow the step of derivation. Since this calculation is after the calculation of translational matrices, the z axis can be easily got from part of the rotation matrix. Also, we can easily get the position vectors of the origins of all frames, therefore getting the vector from specific frame to the end-effector.

The result is:

$$\begin{bmatrix} -100\sin\theta_1\sin\theta_2 & 100\cos\theta_1\cos\theta_2 & 0 \\ 100\cos\theta_1\sin\theta_2 & 100\sin\theta_1 cos\theta_2 & 0 \\ 0 & -100\sin\theta_2 & 0 \\ 0 & -\sin\theta_1 & \cos\theta_1\sin\theta_2 \\ 0 & \cos\theta_1 & sin\theta_1\sin\theta_2 \\ 1 & 0 & \cos\theta_2 \end{bmatrix}$$

4.2. Joint Torques Derivation And Trajectory Planning

For joint torques derivation, we can simply use the formula

$$\tau = \mathbf{J}^{\mathrm{T}}\mathbf{F}.$$

Now the thing is to derive the generalized force vector. The Jacobian of this configuration is also determined with the assumption. Then we can set a pair of forces: f_t, f_N, N_{x_3}. The direction of N_{x_3} is along the direction of x_3. F_t is in the opposite direction with velocity. F_N is perpendicular to paper plane, along $-z_0$. The only undetermined direction is f_t. So we calculate the direction of velocity first.

Two things are important for this section:

- Given the force on tool end, how to derive it to the end-effector frame 3?
- Given the magnitude of local forces only, how to derive the generalized force vector?

$$\begin{bmatrix} {}^{\mathrm{A}}\mathrm{f}_{\mathrm{B}} \\ \mathrm{A}_{\mathrm{B}} \end{bmatrix} = \begin{bmatrix} \mathrm{I} & 0 \\ \lfloor {}^{\mathrm{A}}\mathrm{R}_{\mathrm{B}}{}^{\mathrm{B}}\mathrm{p}_{\mathrm{C}}\mathbf{x} \rfloor & \mathrm{I} \end{bmatrix} \begin{bmatrix} \mathrm{Af}_{\mathrm{C}} \\ \mathrm{A}_{\mathrm{C}} \end{bmatrix}$$

Then we use both the direction and magnitude of forces and torques to calculate the genralized force vector F_t, and use the Jacobian to get the joint forces. Here we assume $|f_t| = 10$ and $|f_n| = 10$.

```
fn = [0;0;-1];
ft_mag = 10;
fn_mag = 10;
f = ft_mag*ft + fn_mag*fn;
% torque:
nx3 = 0.04;
```

```
x3 = R_0_3(1:3,1);
n = nx3*x3;
Ft = double(subs([f;n],[theta1,theta2,theta3],initial_thetas)); % on tool
% now apply force transformation
lower_vec = double(subs(R_0_3*joint(4).rot(1:3,4),[theta1,theta2,theta3],initial_thetas));
upper_mat = cross_dot(lower_vec);
F3 = [eye(3,3),zeros(3,3);
upper_mat,eye(3,3)]*Ft;
joint_force = vpa(double(subs(J.'*F3,[theta1,theta2,theta3],initial_thetas)/1000),5)
% here divide by 1000 because the unit of length is mm
```

And input the specific magnitudes in, the result is:

$$\tau = \begin{bmatrix} 0.081598 \\ -0.081563 \\ -0.74319 \end{bmatrix}$$

Then our goal is to confirm the exact configuration of manipulator so that we can set the relation between task space and joint space. Here we make assumption that the angle between link 2 and vertical line is 45^o. Then $\alpha = \tan^{-1}(\frac{100}{150})$, and $\beta = 45^o - \alpha$. Also from the triangle relation, $c = \sqrt{100^2 + 150^2}$. Then $h = c\cos(\beta)$ and the radius of trajectory is $c\sin(\beta)$.

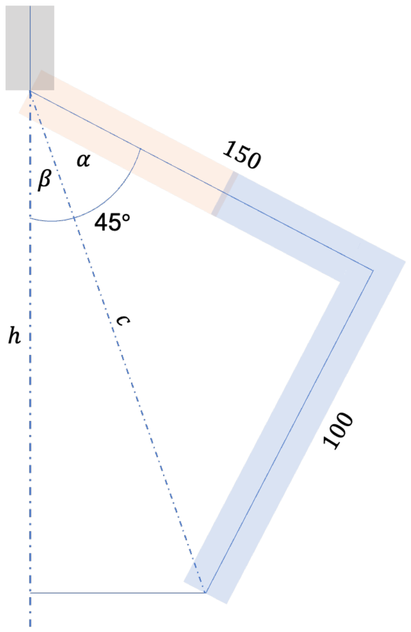


Figure 6: Configuration assumption.

```
% initializing joint variables
alpha = atan(100/150);
c = sqrt(150^2+100^2);
beta = pi/4-alpha;
a = c*sin(beta); % the rotating radius of tool end
h = c*cos(beta);
```

```
zz = 400+h; % this is the distance between frame 0 and the working surface
```

Once we get the true configuration, we can calculate the forward kinematic equations of the manipulator and sequencely get the position of tool end.

```
Kinem = eye(4,4);
for i = 1:4
Kinem = Kinem*joint(i).rot;
end
R_0_t = Kinem(1:3,1:3);
P_0_t = simplify(Kinem(1:3,4))
x0 = [0 0 0];
 % double(subs(P_0_t,[theta1,theta2,theta3],x0))
options = optimoptions(@fsolve,'OptimalityTolerance',1e-8)
ans = fsolve(@buzhi,x0,options);
initial_thetas = double(ans)   % set the inital point at (0,100,250) and get the theta values
```

We inversely prove the result is correct.

```
Ptsol = double(subs(P_0_t,[theta1,theta2,theta3],ans)) % inverse proof
double(Ptsol-[0; a; zz])
```

This will give us a result which is very close to zero. Thus we get a true configuration.

Actually, we can prove that the intersection line of tool plain and paper plain are always perpendicular to z_0 axis and along z_1 axis. About the initial configuration, we make an assumption that the link surface S_1 is perpendicular to the tool surface S_2.

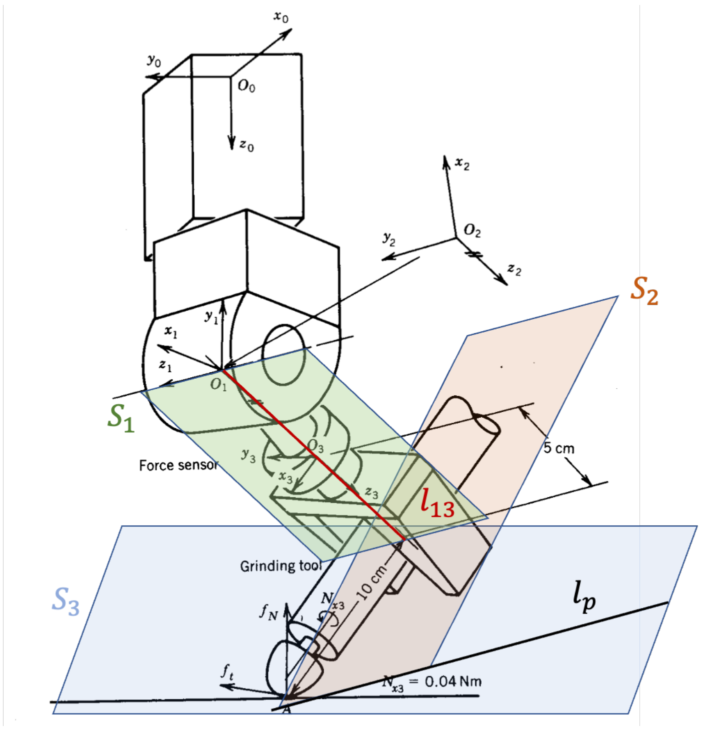


Figure 7: Surface assumption.

Based on the assumption, we can prove the direction of velocity. Obviously the direction of velocity is along l_p. Since $z_1 \mathbin{/\!/} x_3o_3y_3$. $l_{13} \perp S_2$ and also $l_{13} \perp x_3o_3y_3$. Therefore $x_3o_3y_3 \mathbin{/\!/} S_2$ and thus $z_1 \mathbin{/\!/} S_2$. Since l_p is the intersection line of S_2 and S_3, $z_1 \mathbin{/\!/} l_p$, which means that the direction of velocity is always parallel to z_1 axis. Therefore we can get the direction of f_t by simply using z_1:

```matlab
% by observation, we notice that ft is parallel to z1
% force:
z1 = joint(1).rot(1:3,3);
ft = -z1;
fn = [0;0;-1];
ft_mag = 10;
fn_mag = 10;
f = ft_mag*ft + fn_mag*fn;
% torque:
nx3 = 0.04;
x3 = R_0_3(1:3,1);
n = nx3*x3;
```

This also means that the direction of tool end velocity is parallel to the rotationaxis z_1. We assume the position and velocity of tool end as parametric equation.

$$X = [-a\sin t, a\cos t, h]^T$$
$$V = [-a\cos t, -a\sin t, 0]^T$$

Since this is the tool end velocity and actually we want to use the velocity of V_3. Actually we have used the assumption of tool end velocity to derive the position and orientation of each frame. Here we directly use P_3 and do a differential to get the velocity vector of the origin of frame 3. Since the joint variable $\theta_1, \theta_2, \theta_3$ are determined to be related to our assumption, θ_1 should be related to t, and θ_2, θ_3 are fixed.

```matlab
PO3 = subs(P_0_3,[theta2,theta3],[initial_thetas(2),initial_thetas(3)]);
xy3 = PO3;
xy3m = subs(xy3,theta1,theta_1);
v3 = diff(xy3m,t);
```

Now we have already get the velocity v_3, we can pre-multiply the Jacobian matrix to it directly and get the result of joint velocity control, thus other parameters.

```matlab
J33 = simplify(subs(J,[theta1,theta2,theta3],[theta_1,initial_thetas(2),initial_thetas(3)]));
% if using J.'*inv(J*J.'), the result is with entries of inf. this is the
% singular point (guess), therefore using pinv
q_dot = simplify(subs(simplify(pinv(J33)*[v3;0;0;1]),theta1,theta_1)) % here use pinv to calculate
    ↪ the pseudo inverse of J33
```

Till now, we have done most of the work, mainly replacing all variables with only parameter t. Here t actually means the true arc angle of trajectory circle, which is very convenient for simulation.

4.3. Motion simulation

From the result in the last section, we solve the initial angles and rates of each joint. According to this We can easily write the joint angles as the function of time.

$$\begin{cases} \theta_1 = -0.5008 + t \\ \theta_2 = -0.6606 \\ \theta_3 = 0.3153 \end{cases}$$

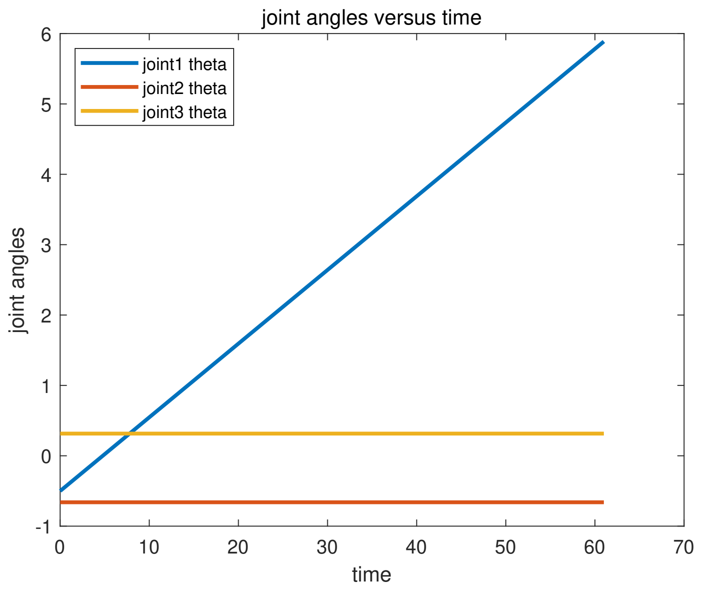
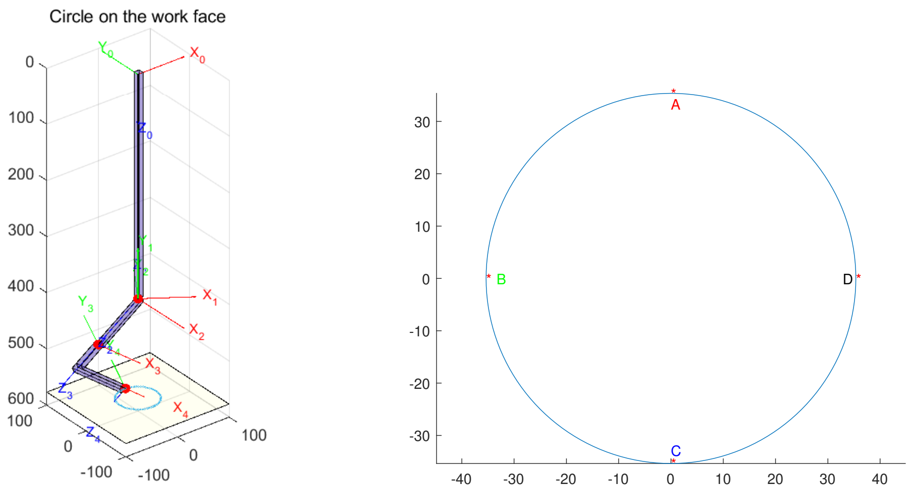


Figure 8: Joint angles versus time

The function clearly illustrates the kinematics of PUMA 600. We firstly plot the circle on the work surface you we drawn with the tool.

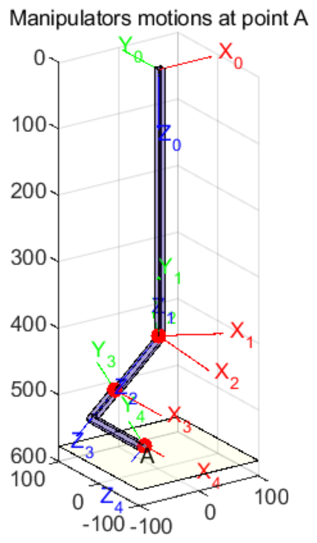


(a) 3D

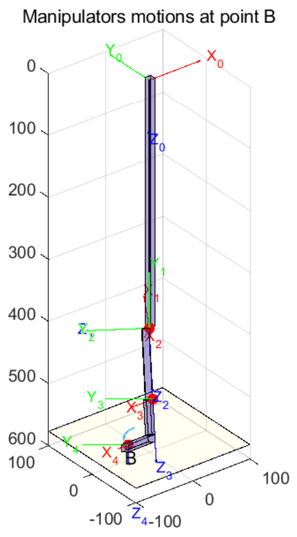
(b) 2D

Figure 9: Circle on the work surface

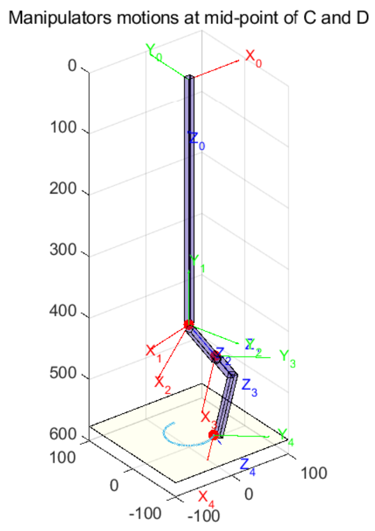
Then, we plot the joints and links at points A, B and the mid-point between C and D respectively on plane(s) that clearly illustrates the manipulator's motion.



(a) Point A



(b) Point B



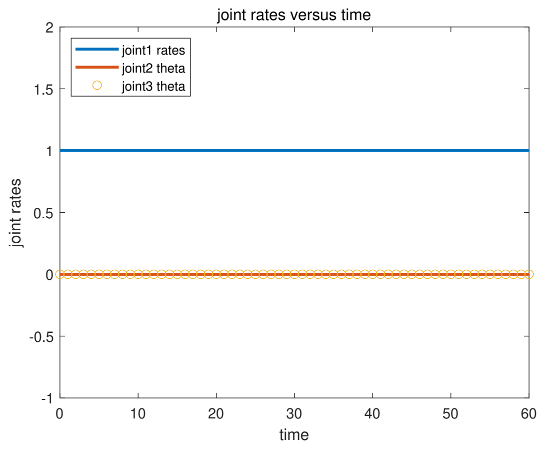
(c) Mid-point between C&D

Figure 10: Manipulators motion at A, B and mid-point of C&D

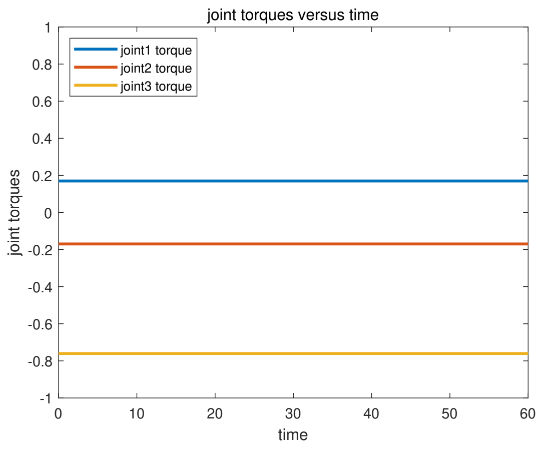
Also, in last section we get the joint rates and torques which are as below:

$$\begin{cases} \dot{\theta}_1 = 1 \\ \dot{\theta}_2 = 0 \\ \dot{\theta}_3 = 0 \end{cases}$$

$$\begin{cases} M_1 = 0.16976 \\ M_2 = -0.16972 \\ M_3 = -0.76038 \end{cases}$$



(a) Joint rates versus time



(b) Joint torques versus time

Figure 11: Joint rates & torques versus time

4.4. GUI for problem 4

Based on the work above, we design and realize a graphical user interface (GUI) program for problem 4.

The first function of GUI is to illustrate the process about how the PUMA600 draws the circle on the work surface as we programmed.

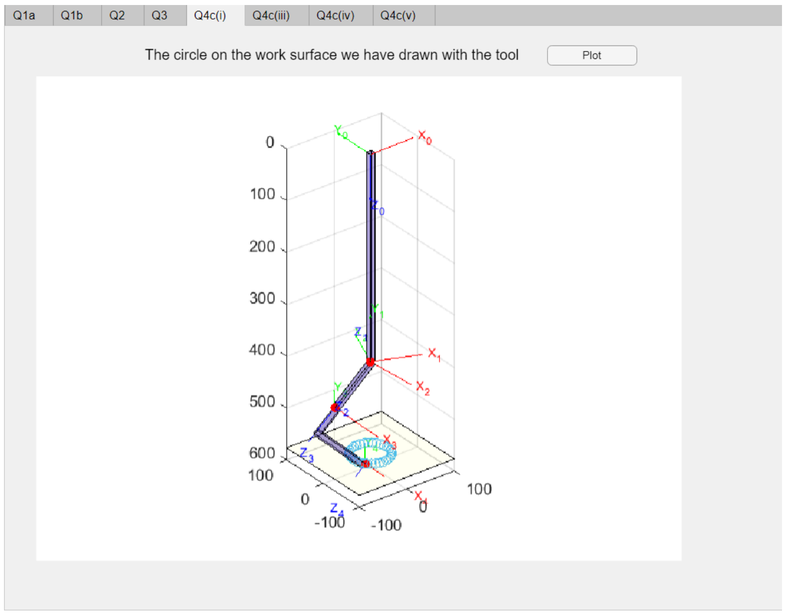
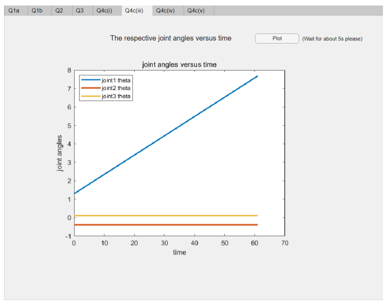
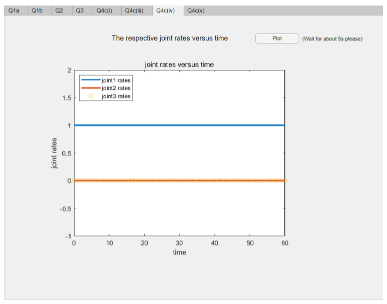


Figure 12: Simulation on the PUMA600 drawing circle process

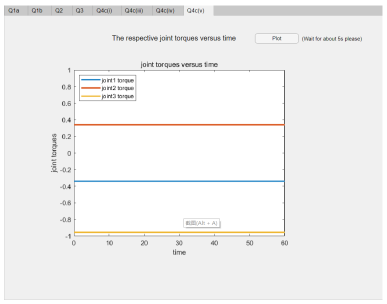
The second function of GUI is to plot the joint angles,rates and torques versus time, which we have showed in last section.



(a) Joint angles versus time



(b) Joint rates versus time



(c) Joint torques versus time

Figure 13: GUI for plot Joint angles,rates and torques versus time

5. Conclusion of Project

Now we review the methods and procedure of solving the problems in this project.

For the first problem, the basic representation transformations are used. Basically we used the formulas derived in class. Except for the basic requirements of problem itself, we also extend it such as changing the sequence ans so on. With this problem, we reviewed the section of representation transformation in class.

The second problem gives a scenario of PUMA 600. The forward and inverse kinematics are used in the subproblems. The basic and fundamental part is the D-H table. This is also the very basic part of all sections in robotics]. With the correct D-H table and joint variables, we can do forward kinematics by deriving the transformation matrices between two joints, multiplying them and derive the position and orientation of end-effector. Then we assume a configuration and apply the inverse kinematics by doing inverse to the transformation matrices and find the solvable equations. Finally we analyse the number of solutions by analytically solve the special "wrist"manipulators.

For the third problem, all things are begin with the Jacobian matrix. Since Jacobian gives the corresponding relationship between joint space and work space, it is usually used in space transformation related to force, velocity and so on. For this problem, we applied the procedure of deriving a Jacobian matrix and relate the end-effector forces to joint forces. After this, all subproblems can be solved easily.

The forth problem should be the most comprehensive problem among these four problems. It first requires the derivation of jacobian matrix. Then apply it to the derivation of joint forces. Finally a trajectory planning problem is asked. For the trajectory problem, we first do some assumptions related to geometry, planes, angles and so on. Then we assume the trajectory as a paramestic equation, and based on the assumptions we can derive the corresponding joint variables(angles, velocities, torques) by inverse derivation. Finally we plot all parameters needed.

During the period of this project, we gained a lot of coding experience related to robotics area especially on Matlab. Expect for the required coding and plotting sections, we also did 3-D solid plotting of the manipulators, made animations and made an UI interface. With these, users can easily visualize the abstract problem as a movable concrete figure.

Finally we would like to send our great thanks to Prof. Chui Chee Kong. Thanks to his detailed lectures, we can gain so much about robotics kinematics, statics and trajectory planning, with which we can successfully finish this complex project. Hope we can all continue studying in this area and do contribute to robotics.