

Faculty of Power and Aeronautical Engineering
Warsaw University of Technology



Group Project

Created by:

Wenjie Yi [330149]

Alaa Ahmed [330294]

Jehad Alostaz [330491]

Supervised by:

Dr.Maciąg Paweł

WARSZAWA 4 June 2023

Contents

Table of Figures.....	3
1. Project Description.....	4
2. Kinematic Structure.....	5
2.1 Denavit-Hartenberg parameters	6
2.1.1 Assignment of coordinate frames.....	6
2.1.2 Determine the link and joint parameters	6
2.1.3 Denavit-Hartenberg Table.....	7
2.2 Validation.....	8
3. Direct Kinematics Problem	9
3.1 Homogeneous transformations.....	9
3.1.1 Joint/Link description matrices	9
3.1.2 The final direct kinematic analysis matrices	10
3.2 Implementation	10
4. Inverse Kinematic Problem.....	12
4.1 Solution for angle Joint 1 (θ_1).....	12
4.2 Solution for angle Joint 1 (θ_2).....	12
4.3 Solution for angle Joint 1 (θ_3).....	13
4.4 Solution for angle Joint 1 (θ_5).....	14
4.5 Solution for angle Joint 1 (θ_4).....	14
4.6 Solution for angle Joint 1 (θ_6).....	15
4.7 Implantation	15
5. Trajectory Generation.....	17
5.1 Task Definition.....	17
5.2 Approach and depart trajectory generation	19
5.3 Cubic Trajectory generation.....	19
5.4 LSPB trajectory generation	20
5.5 Trajectory Testing	20
6. The Whole Processing.....	22
6.1 Input Parameters.....	23
6.2 For the Joint Space	23
6.3 For the Task Space	23
6.4 For the Velocity.....	24
7. Summary.....	26

Table of Figures

Figure 1 Kinematic structure diagram	5
Figure 2 Screenshot of a visualization of the home position of the manipulator	8
Figure 3 Trajectory steps diagram.....	18
Figure 4 Trajectory steps- Visualization.....	19
Figure 5 Trajectory Testing in MATLAB	21
Figure 6 The whole processing.....	22
Figure 7 The connection of two coordinates.....	22
Figure 8 The codes for getting input parameters	23
Figure 9 The codes for trajectory ①.....	23
Figure 10 The codes for trajectory ②③	24
Figure 11 The codes for trajectory in task space	24
Figure 12 The part codes for LSPB trajectory.....	25
Figure 13 The codes for Cubic trajectory	25

1. Project Description

The primary objective of this project is to address the necessary procedures for implementing a software that facilitates the programming of a pick and place task in a 6 DOF anthropomorphic manipulator. The project will focus on the following tasks:

1. Illustrating and explaining the kinematic structure of the manipulator.
2. Describing and solving the direct kinematic problem.
3. Explaining and resolving the inverse kinematic problem.
4. Creating a visualization of the manipulator.
5. Describing and solving the trajectory generation for the pick and place problem.
6. Finalizing the software to enable easy programming of pick and place tasks on a simulated manipulator.

2. Kinematic Structure

Kinematics refers to the description of motion without considering the forces responsible for it. In the case of a robot manipulator, its kinematic solutions can be categorized into two types: forward kinematics and inverse kinematics.

Forward kinematics involves determining the position of the robot's manipulator hand when all joint values are known. It provides information on where the end-effector of the manipulator will be located in space.

On the other hand, inverse kinematics involves calculating the joint variables required to achieve a desired position and orientation of the end-effector. It focuses on determining the appropriate joint values based on a given Cartesian position.

In summary, forward kinematics entails transforming joint space to Cartesian space, while inverse kinematics involves transforming Cartesian space to joint space. **Figure 1** shows the **Kinematic structure diagram**.

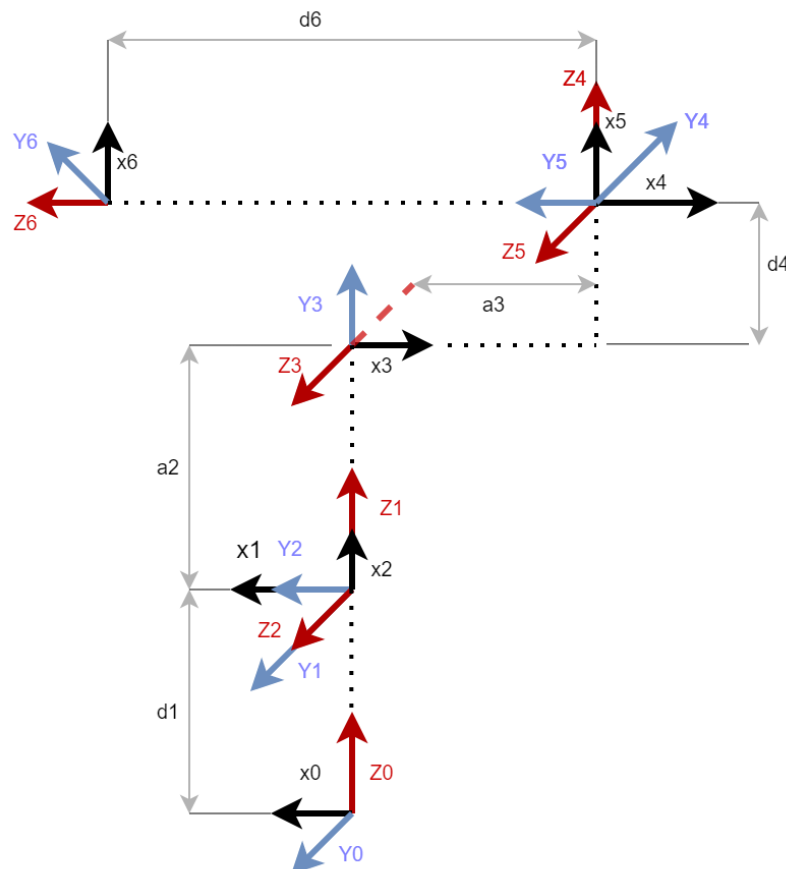


Figure 1 Kinematic structure diagram

2.1 Denavit-Hartenberg parameters

Frames on links can be arbitrarily assigned, but it is beneficial to establish a standard convention for frame placement, as it simplifies the analysis and creates a common language. In this paper, the modern Denavit-Hartenberg (DH) parameter convention is utilized. The DH convention, introduced by Denavit and Hartenberg in 1955, provides a systematic notation for assigning right-handed orthonormal coordinate frames to each link in an open kinematic chain.

By assigning a reference frame (x_i, y_i, z_i) to each link, a set of DH parameters is defined to describe the spatial relationships between a joint axis and its neighboring joint axes. Once these link-attached coordinate frames are established, the transformation between adjacent coordinate frames can be represented using a standard 4×4 homogeneous coordinate transformation matrix.

2.1.1 Assignment of coordinate frames

The z_i axes:

- $i = 1$ z_1 pointing up along axis 1;
- $i = 2$ z_1 pointing into the screen along axis 2;
- $i = 3$ z_2 pointing into the screen along axis 3;
- $i = 4$ z_3 in the plane of the screen along axis 4;
- $i = 5$ z_5 pointing into the screen along axis 5;
- $i = 6$ z_6 in the plane of the screen along axis 6.

The x_i axes:

- $i = 1$ $x_1 : -z_1 \times z_2$
- $i = 2$ $x_2 : z_2 \rightarrow z_3$ along common perpendicular
- $i = 3$ $x_3 : -z_3 \times z_4$
- $i = 4$ $x_4 : z_4 \times z_5$
- $i = 5$ $x_5 : -z_5 \times z_6$
- $i = 6$ $x_6 : \text{perpendicular to } z_6$

2.1.2 Determine the link and joint parameters

Link lengths:

- a_0 : the distance between axes z_0 and z_1 measured along x_0 ;
- a_1 : the distance between axes z_1 and z_2 measured along x_1 ;
- a_2 : the distance between axes z_2 and z_3 measured along x_2 ;
- a_3 : the distance between axes z_3 and z_4 measured along x_3 ;

a_4 : the distance between axes z_4 and z_5 measured along x_4 ;

a_5 : the distance between axes z_5 and z_6 measured along x_5 .

Link twists:

α_0 : the angle between axes z_0 and z_1 measured along x_0 ;

α_1 : the angle between axes z_1 and z_2 measured along x_1 ;

α_2 : the angle between axes z_2 and z_3 measured along x_2 ;

α_3 : the angle between axes z_3 and z_4 measured along x_3 ;

α_4 : the angle between axes z_4 and z_5 measured along x_4 ;

α_5 : the angle between axes z_5 and z_6 measured along x_5 .

Joint offsets:

d_1 : the distance between axes x_0 and x_1 measured along z_1 ;

d_2 : the distance between axes x_1 and x_2 measured along z_2 ;

d_3 : the distance between axes x_2 and x_3 measured along z_3 ;

d_4 : the distance between axes x_3 and x_4 measured along z_4 ;

d_5 : the distance between axes x_4 and x_5 measured along z_5 ;

d_6 : the distance between axes x_5 and x_6 measured along z_6 ;

Joint angles:

θ_1 : the angle between axes x_0 and x_1 measured along z_1 ;

θ_2 : the angle between axes x_1 and x_2 measured along z_2 ;

θ_3 : the angle between axes x_2 and x_3 measured along z_3 ;

θ_4 : the angle between axes x_3 and x_4 measured along z_4 ;

θ_5 : the angle between axes x_4 and x_5 measured along z_5 ;

θ_6 : the angle between axes x_5 and x_6 measured along z_6 ;

2.1.3 Denavit-Hartenberg Table

i	$\alpha_{i-1}[\text{rad}]$	$a_{i-1}[\text{m}]$	$d_i[\text{m}]$	$\theta_i[\text{rad}]$
1	0	0	60	θ_1
2	$-\pi/2$	0	0	$\theta_2 - \pi/2$
3	0	150	0	$\theta_3 - \pi/2$
4	$-\pi/2$	70	130	θ_4
5	$\pi/2$	0	0	$\theta_5 + \pi/2$
6	$-\pi/2$	0	90	θ_6

2.2 Validation

This kinematic structure and the choice of Denavit-Hartenberg parameters has been validated by means of a visualization of the simulated direct kinematics.

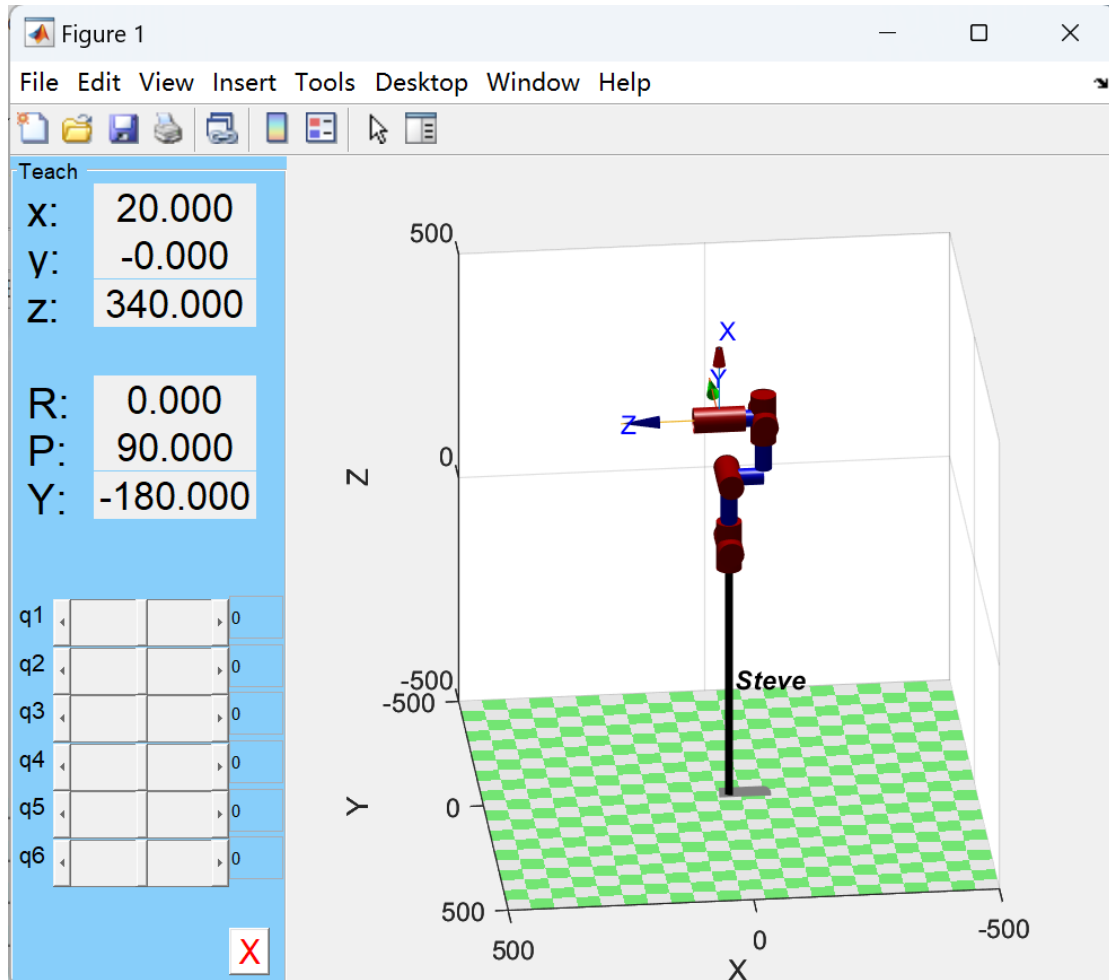


Figure 2 Screenshot of a visualization of the home position of the manipulator

3. Direct Kinematics Problem

The forward kinematics, also known as direct kinematics, of a robot is employed to determine the position and orientation of the end-effector when the joint angles and link parameters are known. Since a manipulator comprises multiple parts, the positions can be calculated with respect to different reference frames. An analytical examination of the links in various positions is systematically carried out to obtain these calculations.

3.1 Homogeneous transformations

$${}^{i-1}_iT = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & a_{i-1} \\ \sin\theta_i \cos\alpha_{i-1} & \cos\theta_i \cos\alpha_{i-1} & -\sin\alpha_{i-1} & -d_i \sin\alpha_{i-1} \\ \sin\theta_i \sin\alpha_{i-1} & \cos\theta_i \sin\alpha_{i-1} & \cos\alpha_{i-1} & d_i \cos\alpha_{i-1} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3.1.1 Joint/Link description matrices

$${}^0_1T = \begin{bmatrix} c\theta_1 & -s\theta_1 & 0 & 0 \\ s\theta_1 & c\theta_1 & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$${}^1_2T = \begin{bmatrix} s\theta_2 & c\theta_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ c\theta_2 & -s\theta_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$${}^2_3T = \begin{bmatrix} c\theta_3 & -s\theta_3 & 0 & a_2 \\ s\theta_3 & c\theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$${}^3_4T = \begin{bmatrix} c\theta_4 & -s\theta_4 & 0 & a_3 \\ 0 & 0 & 1 & d_4 \\ -s\theta_4 & -c\theta_4 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$${}^4_5T = \begin{bmatrix} c\theta_5 & -s\theta_5 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ s\theta_5 & c\theta_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$${}^5_6T = \begin{bmatrix} c\theta_6 & -s\theta_6 & 0 & 0 \\ 0 & 0 & -1 & 90 \\ -s\theta_6 & -c\theta_6 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3.1.2 The final direct kinematic analysis matrices

$${}^0_6T = {}^0_1T {}^1_2T {}^2_3T {}^3_4T {}^4_5T {}^5_6T$$

$${}^0_6T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{aligned} r_{11} &= c_1[(c_2c_3 - s_2s_3)(c_4c_5c_6 - s_4s_6) + (-c_2s_3 - s_2c_3)s_5c_6] + (-s_1)(-s_4c_5c_6 - c_4s_6) \\ r_{21} &= s_1[(c_2c_3 - s_2s_3)(c_4c_5c_6 - s_4s_6) + (-c_2s_3 - s_2c_3)s_5c_6] + c_1(-s_4c_5c_6 - c_4s_6) \\ r_{31} &= (-s_2c_3 - c_2s_3)(c_4c_5c_6 - s_4s_6) + (s_2s_3 - c_2c_3)s_5c_6 \\ r_{12} &= c_1[(c_2c_3 - s_2s_3)(-c_4c_5s_6 - s_4c_6) + (-c_2s_3 - s_2c_3)(-s_5s_6)] + (-s_1)(s_4c_5s_6 - c_4c_6) \\ r_{22} &= s_1[(c_2c_3 - s_2s_3)(c_4c_5c_6 - s_4s_6) + (-c_2s_3 - s_2c_3)s_5c_6] + c_1(s_4c_5s_6 - c_4c_6) \\ r_{32} &= (-s_2c_3 - c_2s_3)(-c_4c_5s_6 - s_4c_6) + (s_2s_3 - c_2c_3)(-s_5s_6) \\ r_{13} &= c_1[-(c_2c_3 - s_2s_3)c_4s_5 + (-c_2s_3 - s_2c_3)c_5] + (-s_1)(s_4s_5) \\ r_{23} &= s_1[-(c_2c_3 - s_2s_3)c_4s_5 + (-c_2s_3 - s_2c_3)c_5] + c_1s_4s_5 \\ r_{33} &= -(c_2s_3 - s_2c_3)c_4s_5 + (s_2s_3 - c_2c_3)c_5 \\ p_x &= c_1((s_2c_3 + c_2s_3)(-c_4s_5d_6 + a_3) + (-s_2s_3 + c_2c_3)(d_6c_5 + d_4) + (s_2a_2 + a_1)) - s_1(s_4s_5d_6) \\ p_y &= s_1((s_2c_3 + c_2s_3)(-c_4s_5d_6 + a_3) + (-s_2s_3 + c_2c_3)(d_6c_5 + d_4) + (s_2a_2 + a_1)) + c_1(s_4s_5d_6) \\ p_z &= (c_2c_3 - s_2s_3)(-c_4s_5d_6 + a_3) + (-c_2s_3 - s_2c_3)(d_6c_5 + d_4) + (c_2a_2) + d_1 \end{aligned}$$

3.2 Implementation

The direct kinematic matrices should be tested, the test results should be compared with the results from Robotics Toolbox.

Two groups of the input parameters are used to test. The results is shown below:

q1	0.1	0.1	0.0193	0.1193	0	0
----	-----	-----	--------	--------	---	---

robot.fkine(q)	$\text{robot.fkine}(q) = \begin{bmatrix} 0.1184 & 0.2167 & 0.9690 & 48.35 \\ 0.0119 & -0.9761 & 0.2168 & 15.62 \\ 0.9929 & -0.0142 & -0.1182 & 336 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
myfkine(q)	$\text{myfkine}(q)$ $= \begin{bmatrix} 0.11842262 & 0.21670471 & 0.96902794 & 48.352424 \\ 0.11881894 & -0.97613442 & 0.21684188 & 15.616755 \\ 0.99289219 & -0.01416509 & 0.11817126 & 336.022401 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

q2	1	-0.5	0.5	0.66	-0.9	1.5
robot.fkine(q)	$\text{robot.fkine}(q) = \begin{bmatrix} 0.9985 & 0.0008 & -0.0554 & -81.66 \\ 0.0337 & 0.7846 & 0.6191 & -63.69 \\ 0.0440 & -0.6201 & 0.7833 & 392.1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$					
myfkine(q)	$\text{myfkine}(q)$ $= \begin{bmatrix} 0.99846519 & 0.00084775 & -0.05537638 & -81.660244 \\ 0.03367221 & 0.78455960 & 0.61913844 & -63.693911 \\ 0.04397094 & -0.62005282 & 0.78332690 & 392.136806 \\ 0 & 0 & 0 & 1 \end{bmatrix}$					

From the above results, the direct kinematic matrix can get the same results from The Robotics Toolbox.

4. Inverse Kinematic Problem

In a robotic manipulator, the joint variables serve as the independent variables, which vary depending on different reference frames. In an inverse kinematic problem, if we have knowledge of the end-effector's position, we can determine the corresponding joint variables. Various methods exist to solve the inverse kinematic problem, including analytical/geometrical approaches, iterative approaches, pseudo-inverse methods, and matrix algebraic approaches.

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

4.1 Solution for angle Joint 1 (θ_1)

According to this matrix:

$${}^1T = {}^0T_i {}^0T_d = {}^1T_d$$

We know:

$${}^1T(2,3) = {}^1T_d(2,3)$$

$${}^1T(2,3) = -\sin(\theta_4)\cos(\theta_5)$$

$${}^1T_d(2,3) = p_x \sin(\theta_1) - p_y \cos(\theta_1)$$

$${}^1T(2,4) = {}^1T_d(2,4)$$

$${}^1T(2,3) = {}^1T_d$$

$${}^1T(2,4) = -90\sin(\theta_4)\cos(\theta_5)$$

$${}^1T_d(2,4) = p_x \sin(\theta_1) - p_y \cos(\theta_1)$$

Then

$$\frac{{}^1T(2,3)}{{}^1T(2,4)} = \frac{{}^1T_d(2,3)}{{}^1T_d(2,4)}$$

We can get:

$$\theta_1 = \arctan \frac{p_y - 90a_y}{p_x - 90a_x}$$

For the computation of θ_1 , two solutions will be produced.

4.2 Solution for angle Joint 1 (θ_2)

According to this matrix:

$${}^0_4T = ({}^0_6Td_6^5Ti) {}^4_5Ti = {}^0_4Td$$

We know:

$${}^0_4T(1,4) = {}^0_4Td(1,4)$$

$${}^0_4T(2,4) = {}^0_4Td(2,4)$$

$${}^0_4T(3,4) = {}^0_4T(3,4)$$

$${}^0_4T(1,4) = px - 90a_x$$

$${}^0_4T(2,4) = py - 90a_y$$

$${}^0_4T(3,4) = pz - 90a_z$$

$${}^0_4Td(1,4) = -70c_1c(\theta_2 + \theta_3) + 130c_1s(\theta_2 + \theta_3) + 150c_1s_2$$

$${}^0_4Td(2,4) = -70s_1c(\theta_2 + \theta_3) + 130s_1s(\theta_2 + \theta_3) + 150s_1s_2$$

$${}^0_4T(3,4) = 130c(\theta_2 + \theta_3) + 70s(\theta_2 + \theta_3) + 150c_2 + 60$$

Simplified:

$$a_3 \sin(\theta_2 + \theta_3) + d_4 \cos(\theta_2 + \theta_3) = -d_6 (c_1 r_{13} + s_1 r_{23}) + c_1 p_x + s_1 p_y - a_1 - s_2 a_2$$

$$a_3 \cos(\theta_2 + \theta_3) - d_4 \sin(\theta_2 + \theta_3) = -d_6 r_{33} + p_z - d_1 - c_2 a_2$$

Defining k_1 and k_2 :

$$a_3 \sin(\theta_2 + \theta_3) + d_4 \cos(\theta_2 + \theta_3) = k_1 - s_2 a_2$$

$$a_3 \cos(\theta_2 + \theta_3) - d_4 \sin(\theta_2 + \theta_3) = k_2 - c_2 a_2$$

$$k_1 = -d_6 (c_1 r_{13} + s_1 r_{23}) + c_1 p_x + s_1 p_y - a_1$$

$$k_2 = -d_6 r_{33} + p_z - d_1$$

$$\text{Setting } \sin(\theta_2 + \phi) = k_1 s_2 + k_2 c_2 = \frac{k_1^2 + k_2^2 + a_2^2 - a_3^2 - d_4^2}{2a_2}$$

Getting the angle of θ_2 :

$$\theta_2 = \arcsin\left(\frac{k_1^2 + k_2^2 + a_2^2 - a_3^2 - d_4^2}{2a_2\sqrt{k_1^2 + k_2^2}}\right) - \phi$$

$$\sin(\phi) = \frac{k_2}{\sqrt{k_1^2 + k_2^2}}$$

$$\cos(\phi) = \frac{k_1}{\sqrt{k_1^2 + k_2^2}}$$

For θ_2 , here are two solutions.

4.3 Solution for angle Joint 1 (θ_3)

θ_1 and θ_2 are obtained, we define:

$$X = \sin(\theta_2 + \theta_3) = \frac{a_3 A + d_4 B}{a_3^2 + d_4^2}$$

$$Y = \cos(\theta_2 + \theta_3) = \frac{a_3 B - d_4 A}{a_3^2 + d_4^2}$$

$$A = k_1 - a_2 \sin \theta_2$$

$$B = k_2 - a_2 \cos \theta_2$$

We can get θ_3 :

$$\theta_3 = \arcsin \left(\frac{a_3(k_1 - s_2 a_2) - d_4(k_2 - c_2 a_2)}{a_3^2 + d_4^2} \right) - \theta_2$$

$$\theta_3 = \arccos \left(\frac{d_4(k_1 - s_2 a_2) + a_3(k_2 - c_2 a_2)}{a_3^2 + d_4^2} \right) - \theta_2$$

$$k_1 = -d_6 (c_1 r_{13} + s_1 r_{23}) + c_1 p_x + s_1 p_y - a_1$$

$$k_2 = -d_6 r_{33} + p_z - d_1$$

4.4 Solution for angle Joint 1 (θ_5)

According to:

$${}^3_6T = {}^2_3Ti({}^1_2Ti({}^0_1Ti({}^0_6Td))) = {}^3_6Td$$

We know that:

$${}^3_6T(2,3) = {}^3_6Td(2,3)$$

$${}^3_6T = -\sin(\theta_5)$$

$${}^3_6Td(2,3) = -a_z \cos(\theta_2 + \theta_3) + a_x(c_1 s_2 + c_1 c_2) + a_y(s_1 s_2 + s_1 c_2)$$

Then we can get θ_5 :

$$\sin(\theta_5) = -a_z \cos(\theta_2 + \theta_3) + a_x(c_1 s_2 + c_1 c_2) + a_y(s_1 s_2 + s_1 c_2)$$

$$\theta_5 = \arcsin(-a_z \cos(\theta_2 + \theta_3) + a_x(c_1 s_2 + c_1 c_2) + a_y(s_1 s_2 + s_1 c_2))$$

For θ_5 , here are two solutions.

4.5 Solution for angle Joint 1 (θ_4)

According to:

$${}^3_6T(3,3) = {}^3_6Td(3,3)$$

We know:

$$\begin{aligned} {}^3_6T(3,3) &= -\sin(\theta_4)\cos(\theta_5) \\ {}^3_6Td(3,3) &= a_y\cos(\theta_1) - a_x\sin(\theta_1) \end{aligned}$$

Then we can get θ_4 :

$$\cos(\theta_4) = -(a_z\sin(\theta_2 + \theta_3) - a_x\cos(\theta_1)\cos(\theta_2 + \theta_3) - a_y\sin(\theta_1)\cos(\theta_2 + \theta_3))/\cos(\theta_5)$$

$$\theta_4 = \arccos(-(a_z\sin(\theta_2 + \theta_3) - a_x\cos(\theta_1)\cos(\theta_2 + \theta_3) - a_y\sin(\theta_1)\cos(\theta_2 + \theta_3))/\cos(\theta_5))$$

4.6 Solution for angle Joint 1 (θ_6)

According to:

$${}^3_6T(2,2) = {}^3_6Td(2,2)$$

We know that:

$$\begin{aligned} {}^3_6T(2,2) &= -\sin(\theta_6)\cos(\theta_5) \\ {}^3_6Td(2,2) &= o_z\cos(\theta_2 + \theta_3) + o_x\cos(\theta_1)\sin(\theta_2 + \theta_3) + o_y\sin(\theta_1)\sin(\theta_2 + \theta_3) \end{aligned}$$

Then we can get θ_6 :

$$\sin(\theta_6) = -(o_z\cos(\theta_2 + \theta_3) + o_x\cos(\theta_1)\sin(\theta_2 + \theta_3) + o_y\sin(\theta_1)\sin(\theta_2 + \theta_3))/\cos(\theta_5)$$

$$\theta_6 = \arcsin(-(o_z\cos(\theta_2 + \theta_3) + o_x\cos(\theta_1)\sin(\theta_2 + \theta_3) + o_y\sin(\theta_1)\sin(\theta_2 + \theta_3))/\cos(\theta_5))$$

4.7 Implantation

the following table shows the results of the tests performed for four possible configurations.

		θ_1	θ_2	θ_3	θ_4	θ_5	θ_6
Test1	Input	-2.3939	-0.0103	-1.0029	0.5358	-1.7353	2.8887
	Myinv	-2.3939	-0.0103	-1.0029	0.5358	-1.7353	2.8887

Test2	Input	1.2946	-2.9416	-1.4016	-2.8515	-2.5313	2.0323
	Myinv	1.2946	-2.9416	-1.4016	-2.8515	-2.5313	2.0323
Test3	Input	1.2241	-1.1492	2.8288	-2.9252	-0.3849	-0.7442
	Myinv	1.2241	-1.1492	2.8288	-2.9252	-0.3849	-0.7442
Test4	Input	1.1230	1.6194	1.5276	-0.6772	0.9769	-2.0660
	Myinv	1.1230	1.6194	1.5276	-0.6772	0.9769	-2.0660

5. Trajectory Generation

This section aims to explore the trajectory generation procedure for the pick-and-place task performed by a 6DOF robot manipulator. Two different techniques will be employed, namely cubic trajectories in both task space and joint space, as well as LSPB (Linear Segments with Parabolic Blends) trapezoidal trajectories in both task space and joint space. By utilizing these techniques, we can effectively plan and execute the desired pick-and-place motions, considering the specific characteristics of each trajectory generation method.

5.1 Task Definition

The process of picking up an object using a manipulator can be divided into various stages, considering both the task space and joint space. Each space offers unique characteristics:

- Task space:
 - Control in the task space is more intuitive since the position of the end effector in Cartesian space is always known. This allows for precise control over the manipulator.
 - Straight lines in Cartesian space translate to straight lines in the task space, simplifying trajectory planning.
 - However, computing the trajectory in the task space requires solving the Inverse Kinematics Problem (IKP) for each point, which can be computationally demanding for long trajectories.
- Joint space:
 - In the joint space, only the starting and ending points in Cartesian space need to be considered, reducing the IKP computation to only those points.
 - Trajectories in the joint space automatically avoid singular positions, enhancing stability and safety.
 - Straight lines in Cartesian space are represented as curves in the joint space, requiring careful consideration for motion planning.
 - The position and orientation of the end effector are not directly known in joint space and require solving the Direct Kinematics Problem (DKP) to obtain them.

From these considerations, it is advantageous to use the task space for short and precise movements, such as approaching and departing from the object. The task space allows for accurate control and avoids singularities. On the other hand, the joint space is more suitable

for longer movements between the main starting and ending points. It eliminates the need to know the precise position at each intermediate point and ensures stability along the trajectory.

Moreover, the specific stages for the pick-and-place task, can be outlined as follows:

1. Move from the current position to the of *Grasp-up position*, which is strategically positioned at a distance from the TCP (Tool Control Point) to prevent potential collisions.
2. Approach the object in the task space by following a straight line from the *Grasp-up position* to *Initial position*.
3. (Note: This is not part of the trajectory description) Close the gripper and securely grasp the object.
4. Depart from the object in the task space by following a straight line from the *Grasp-up position* to *Goal-up position*.
5. Transition from the *Grasp-up position* to *Goal-up position* in the joint space.
6. Approach the target location in the task space by following a straight line from the *Goal-up position* to *Goal position*.
7. (Note: This is not part of the trajectory description) Open the gripper and release the object at the designated location.
8. Depart from the target location in the task space by following a straight line from the *Goal-up position* to *Goal position*.

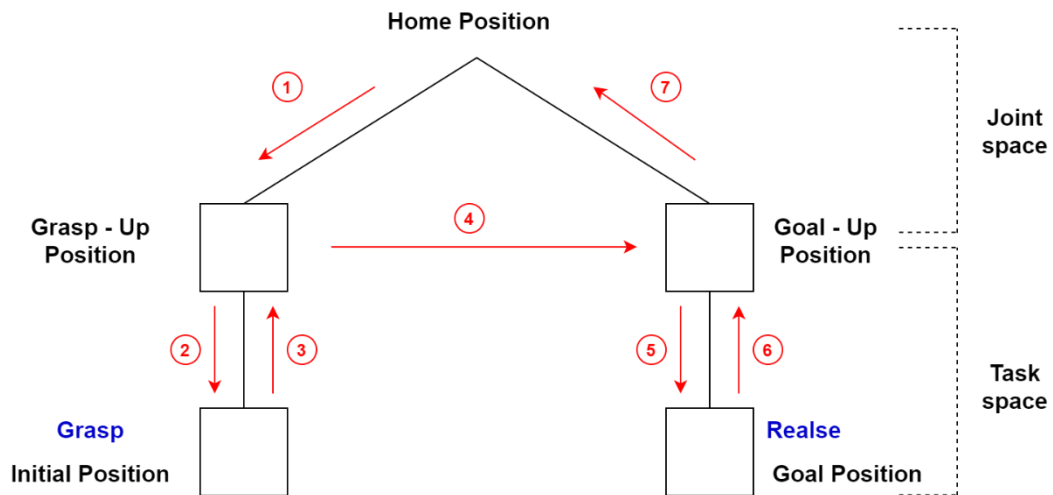


Figure 3 Trajectory steps diagram

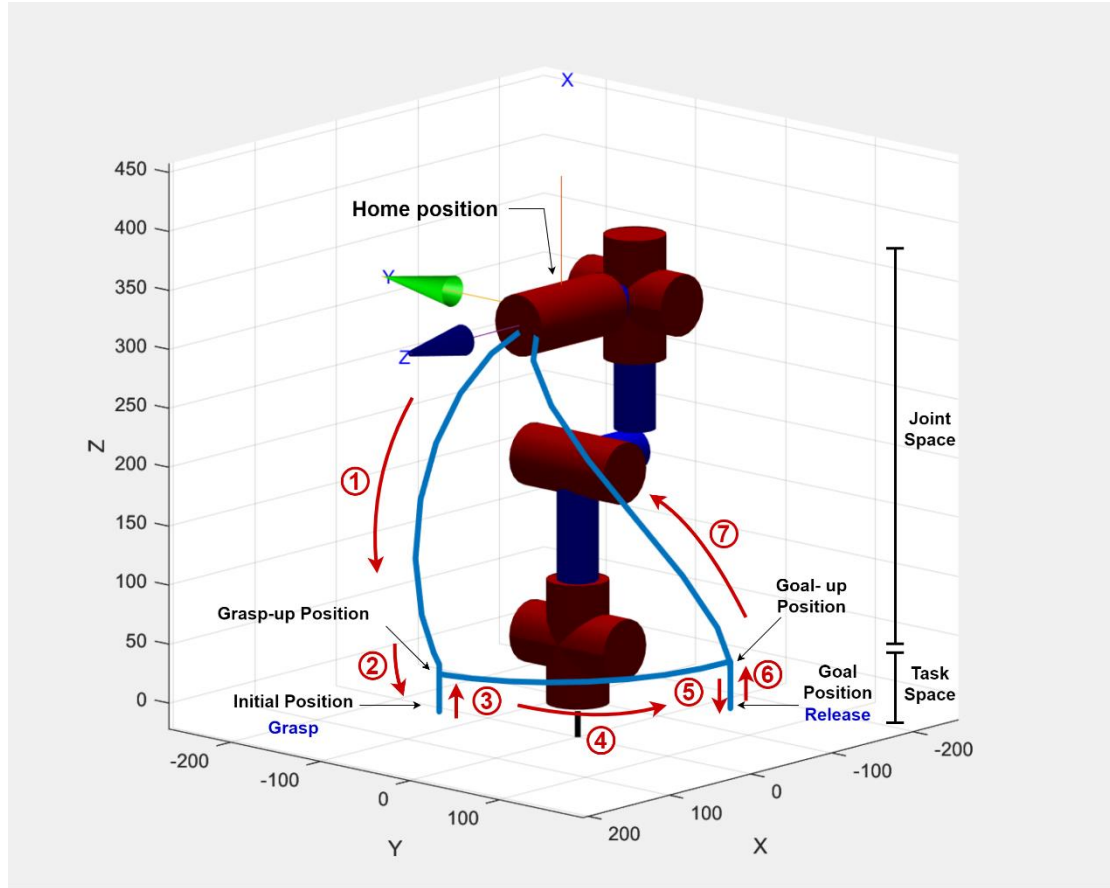


Figure 4 Trajectory steps- Visualization

5.2 Approach and depart trajectory generation

To ensure precise control over the position of the end effector, the approach and depart movements between two points are solved in the task space. In this project, a generic approach was taken for the offset retraction. The process involved obtaining the unit vector representing the z-axis of the end effector frame using the Euler rotation matrix. This unit vector was then multiplied by the offset distance and added to the end effector's position, resulting in the offsite position.

The orientation of the end effector is represented by the angles $[\gamma, \beta, \alpha]$, and the z-axis unit vector can be derived from the third column of the Euler rotation matrix.

5.3 Cubic Trajectory generation

The cubic trajectory is a commonly used method that involves fitting a cubic polynomial between two points. This approach results in a trajectory with a smooth velocity profile and linear acceleration.

The cubic trajectory is defined by four parameters, which represent the position and speed of the initial and final points. These parameters determine the shape and characteristics of the trajectory, allowing for precise control over the motion between the two points.

Cubic polynomial form

$$q(t) = a_0 + a_1 t^2 + a_2 t^3 \quad (1)$$

By imposing the endpoint constraints $q(t_0) = q_0$, $q(t_f) = q_f$, $\dot{q}(t_0) = \dot{q}_0$, $\dot{q}(t_f) = \dot{q}_f$, we can determine the values of the polynomial coefficients. This is achieved by forming a system of four equations that incorporate these constraints along with Equation (2) .

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 \\ 0 & 1 & 2 * t_0 & 3 * t_0^2 \\ 1 & t_f & t_f^2 & t_f^3 \\ 0 & 1 & 2 * t_f & 3 * t_f^2 \end{bmatrix}^{-1} \begin{bmatrix} q_0 \\ \dot{q}_0 \\ q_1 \\ \dot{q}_1 \end{bmatrix} \quad (2)$$

5.4 LSPB trajectory generation

Another approach to trajectory generation is the Linear Segments with Parabolic Blends (LSPB) trajectory. This trajectory is divided into three sections, with the starting and ending sections being parabolic and dependent on a specified blending time t_b (considered as 0.2s for testing purposes). The intermediate section is linear, with zero acceleration during this period. Consequently, the velocity varies linearly at the beginning and end of the trajectory while remaining constant between these points.

Equation 3 represents these stages. Depending on the time step, the trajectory will be fitted either in the linear segment or the parabolic sections.

$$\begin{cases} q_0 + \frac{\alpha}{2} t^2 & t_0 \leq t \leq t_0 + t_b \\ \frac{q_0 + q_1 - v t_1}{2} + v t & t_0 + t_b \leq t \leq t_1 \\ q_1 - \frac{\alpha}{2} t_1^2 - \frac{\alpha}{2} t^2 + \alpha t_1 & t_f - t_b \leq t \leq t_f \end{cases} \quad (3)$$

5.5 Trajectory Testing

Initial position is: (170, 0, 0)(x, y, z), the goal position is (0, 170, 0)(x, y, z). The whole trajectory is shown below:

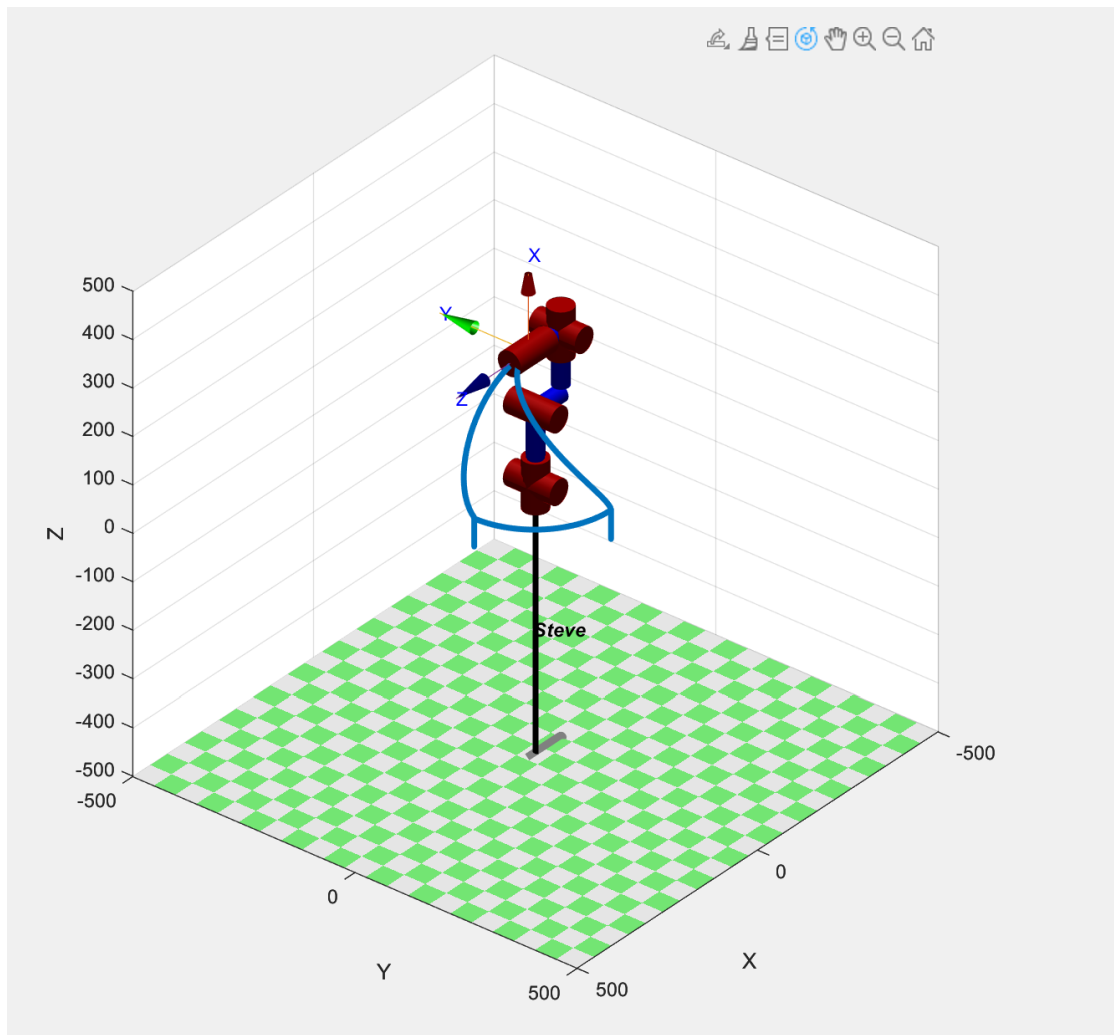


Figure 5 Trajectory Testing in MATLAB

6. The Whole Processing

The whole processing is shown below:

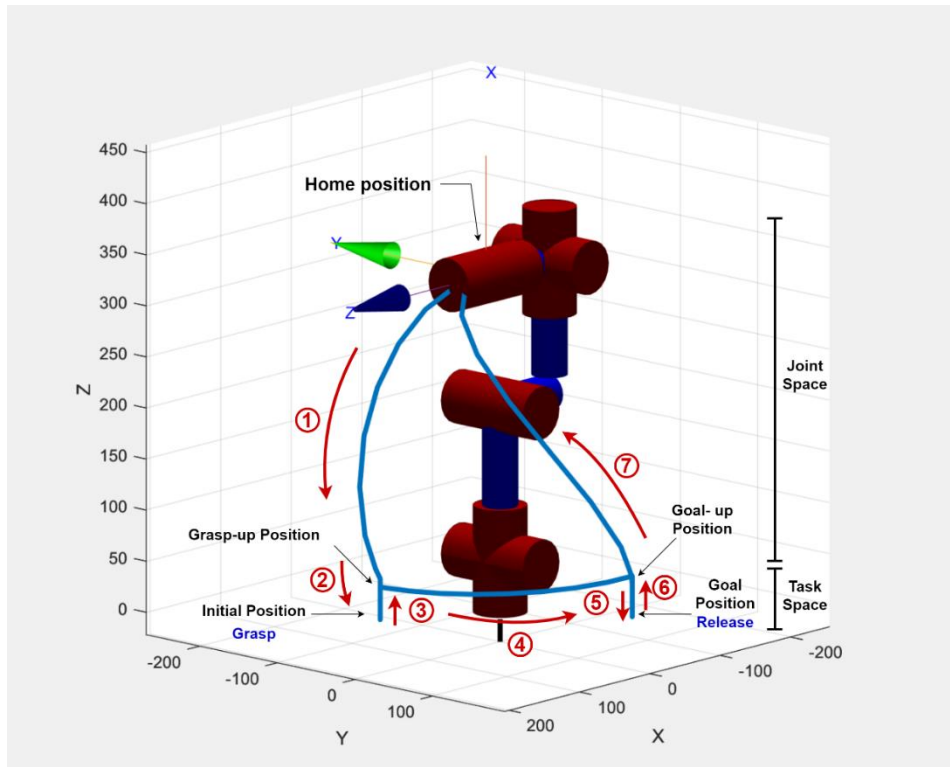


Figure 6 The whole processing

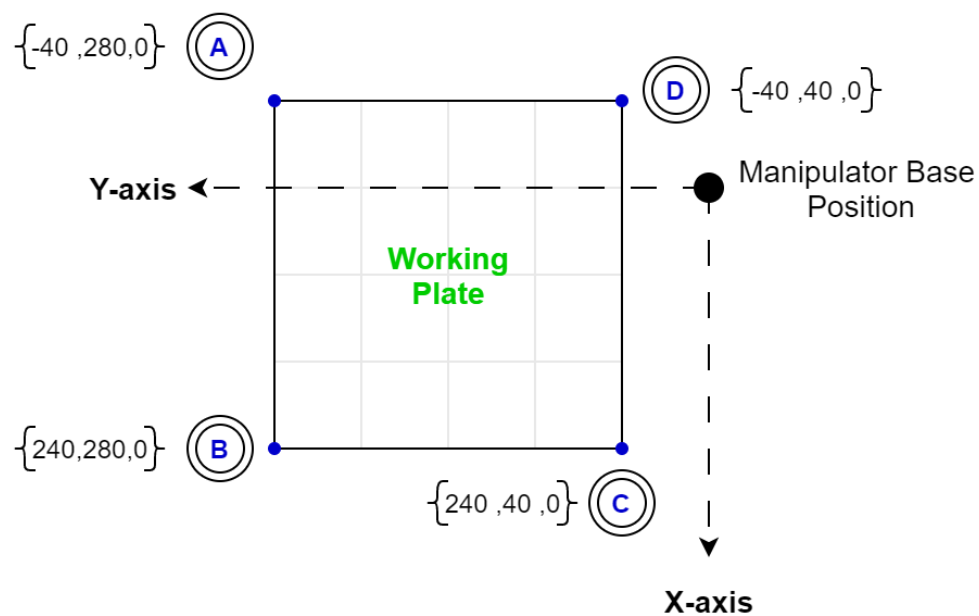
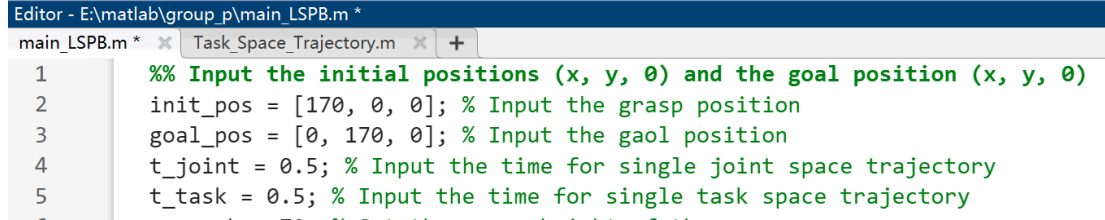


Figure 7 The connection of two coordinates

6.1 Input Parameters

For the simulation, we should input some parameters below at the beginning of the codes.



```
Editor - E:\matlab\group_p\main_LSPB.m *
main_LSPB.m * Task_Space_Trajectory.m *
1      %% Input the initial positions (x, y, 0) and the goal position (x, y, 0)
2      init_pos = [170, 0, 0]; % Input the grasp position
3      goal_pos = [0, 170, 0]; % Input the goal position
4      t_joint = 0.5; % Input the time for single joint space trajectory
5      t_task = 0.5; % Input the time for single task space trajectory
```

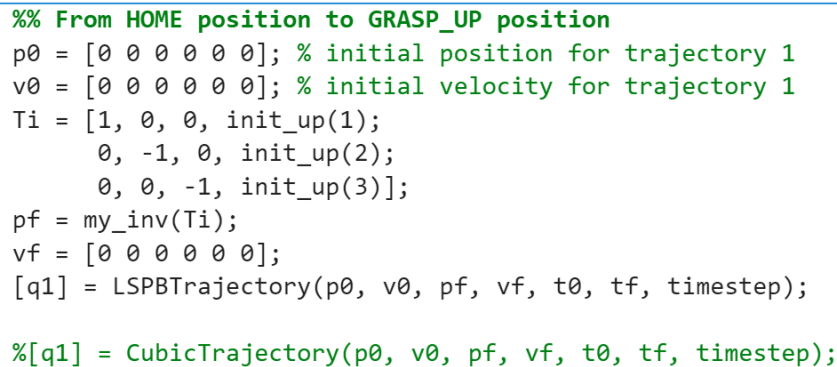
Figure 8 The codes for getting input parameters

According to the Figure 6 (The whole processing), the execution processing of trajectory is from ① to ⑥.

6.2 For the Joint Space

For the joint space trajectory ①④⑦:

We use the function 'LSPBTrajectory' or 'CUBICTrajectory' to generate the trajectory in the joint space. The orientation of three axes can be changed.



```
%% From HOME position to GRASP_UP position
p0 = [0 0 0 0 0 0]; % initial position for trajectory 1
v0 = [0 0 0 0 0 0]; % initial velocity for trajectory 1
Ti = [1, 0, 0, init_up(1);
      0, -1, 0, init_up(2);
      0, 0, -1, init_up(3)];
pf = my_inv(Ti);
vf = [0 0 0 0 0 0];
[q1] = LSPBTrajectory(p0, v0, pf, vf, t0, tf, timestep);

%[q1] = CubicTrajectory(p0, v0, pf, vf, t0, tf, timestep);
```

Figure 9 The codes for trajectory ①

For trajectory ④⑦, We use same trajectory function, but getting the initial position from the previous trajectory.

6.3 For the Task Space

For the task space trajectory ②③⑤⑥:

We use the function ‘Task_Space_Trajectory’ to generate the trajectory in the task space. The orientation of three axes can’t be changed.

```

%% From GRASP_UP position to GRASP position 2
p_up = init_up; % getting the initial position
take_or_place = 0; % if this value is 0, moving down
[q2] = Task_Space_Trajectory(p_up, t0, tff, timesteps, movestep, take_or_place);
q = [q1; q2];

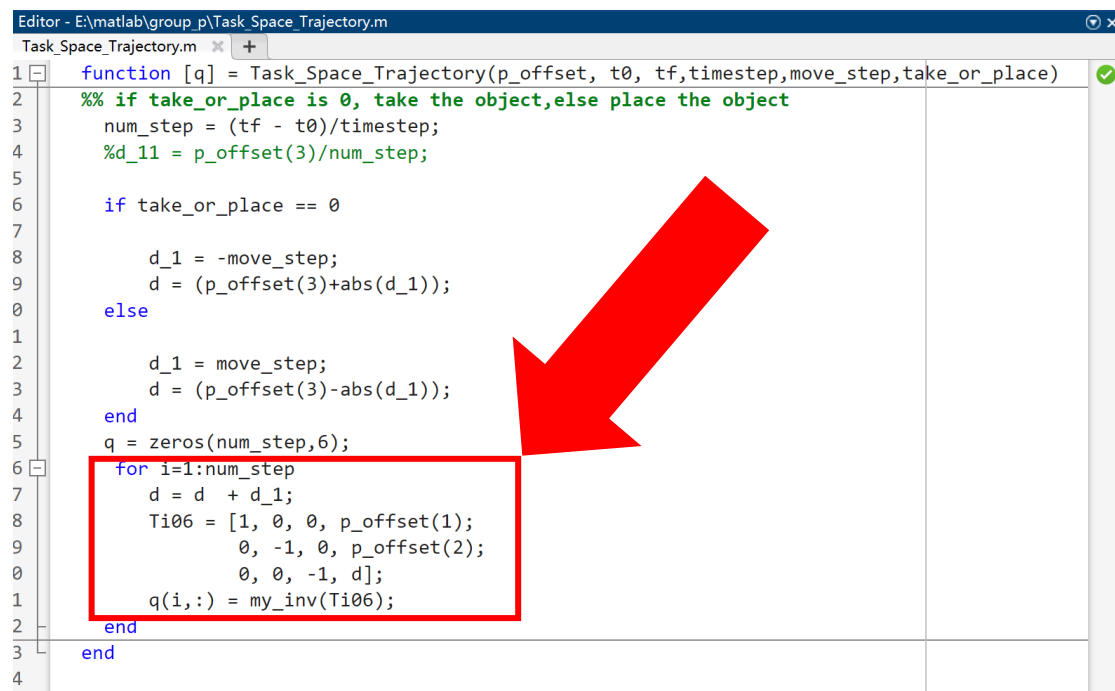
%% From GRASP position to GRASP_UP position 3
p_offset = init_grasp_pos;
take_or_place = 1; % if this value is 1, moving up
[q3] = Task_Space_Trajectory(p_offset, t0, tff, timesteps, movestep, take_or_place);
q = [q; q3];

```

Figure 10 The codes for trajectory ②③

In the codes, We use the variable ‘take_or_place’ to control the moving directions (up or down). If the value of ‘take_or_place’ is 0, moving down, If the value of ‘take_or_place’ is 1, moving up.

From Figure 11 below, we use the for loop to make the orientation of three axes constant.



```

Editor - E:\matlab\group_p\Task_Space_Trajectory.m
Task_Space_Trajectory.m x +
1 function [q] = Task_Space_Trajectory(p_offset, t0, tf, timestep, move_step, take_or_place)
2 % if take_or_place is 0, take the object, else place the object
3 num_step = (tf - t0)/timestep;
4 %d_11 = p_offset(3)/num_step;
5
6 if take_or_place == 0
7     d_1 = -move_step;
8     d = (p_offset(3)+abs(d_1));
9 else
10    d_1 = move_step;
11    d = (p_offset(3)-abs(d_1));
12 end
13 q = zeros(num_step,6);
14 for i=1:num_step
15     d = d + d_1;
16     Ti06 = [1, 0, 0, p_offset(1);
17             0, -1, 0, p_offset(2);
18             0, 0, -1, d];
19     q(i,:) = my_inv(Ti06);
20 end
21 end

```

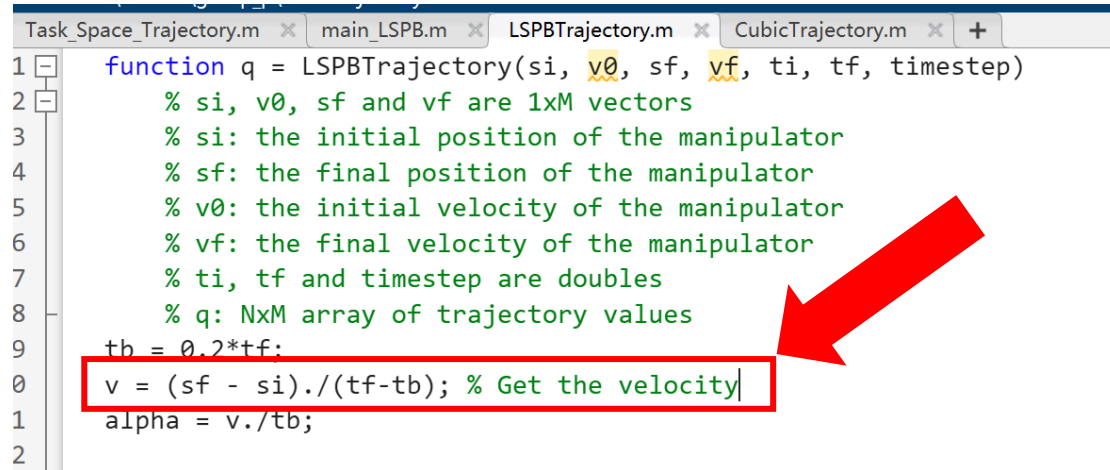
Figure 11 The codes for trajectory in task space

6.4 For the Velocity

We build the velocity control inside the trajectory function.

For LSPB trajectory:

The initial position and final position by initial time and final time can get the velocity of the LSPB trajectory.

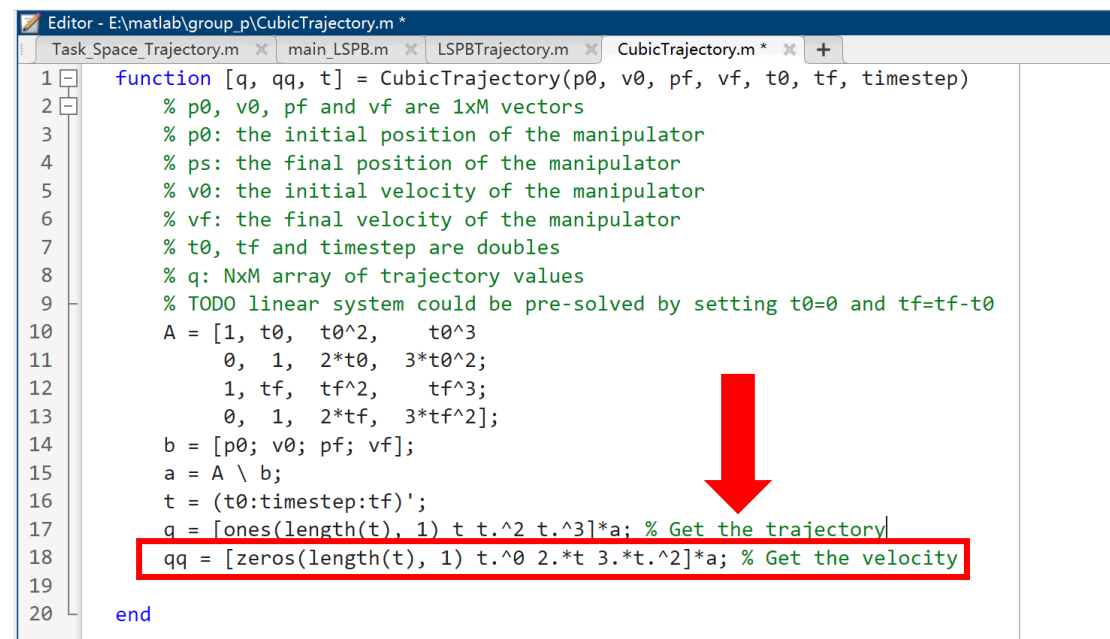


```
1 function q = LSPBTrajectory(si, v0, sf, vf, ti, tf, timestep)
2     % si, v0, sf and vf are 1xM vectors
3     % si: the initial position of the manipulator
4     % sf: the final position of the manipulator
5     % v0: the initial velocity of the manipulator
6     % vf: the final velocity of the manipulator
7     % ti, tf and timestep are doubles
8     % q: NxM array of trajectory values
9     tb = 0.2*tf;
10    v = (sf - si)./(tf-tb); % Get the velocity
11    alpha = v./tb;
```

Figure 12 The part codes for LSPB trajectory

For Cubic trajectory:

We can get the velocity from the variable of 'qq'.



```
1 function [q, qq, t] = CubicTrajectory(p0, v0, pf, vf, t0, tf, timestep)
2     % p0, v0, pf and vf are 1xM vectors
3     % p0: the initial position of the manipulator
4     % pf: the final position of the manipulator
5     % v0: the initial velocity of the manipulator
6     % vf: the final velocity of the manipulator
7     % t0, tf and timestep are doubles
8     % q: NxM array of trajectory values
9     % TODO linear system could be pre-solved by setting t0=0 and tf=tf-t0
10    A = [1, t0, t0^2, t0^3;
11         0, 1, 2*t0, 3*t0^2;
12         1, tf, tf^2, tf^3;
13         0, 1, 2*tf, 3*tf^2];
14    b = [p0; v0; pf; vf];
15    a = A \ b;
16    t = (t0:timestep:tf)';
17    q = [ones(length(t), 1) t t.^2 t.^3]*a; % Get the trajectory
18    qq = [zeros(length(t), 1) t.^0 2.*t 3.*t.^2]*a; % Get the velocity
19
20 end
```

Figure 13 The codes for Cubic trajectory

7. Summary

This project focused on the comprehensive development and implementation of a pick and place task for a 6DOF manipulator. The study involved analyzing the kinematic structure and solving both the inverse and direct kinematic problems. Various tests were conducted to verify the accuracy of the approach. However, a major challenge encountered was effectively handling singularities when solving the IKP for specific positions.

The next phase involved creating a visualization of the manipulator's motion using MATLAB and an additional library. This visualization demonstrated the pick and place task, as well as allowing for user-controlled movements.

Subsequently, trajectory generation techniques were explored, combining the task space and joint space to overcome the issue of unreachable points. Two types of trajectories, cubic and LSPB, were studied. One particular challenge resolved during this stage was maintaining a consistent end-effector orientation during the approach and depart motions.

Overall, this project encompassed a comprehensive and systematic exploration of the pick and place task in a 6DOF manipulator. It involved various stages, including kinematic analysis, visualization development, trajectory generation, and performance validation.