Robotics Library

Xu Duan

May 1, 2025

Chapter 1

Robot Kinematics

1.1 Introduction to Franka Emika Panda

Panda is a 7-DoF robot arm. Here is the picture GmbH, 2017.

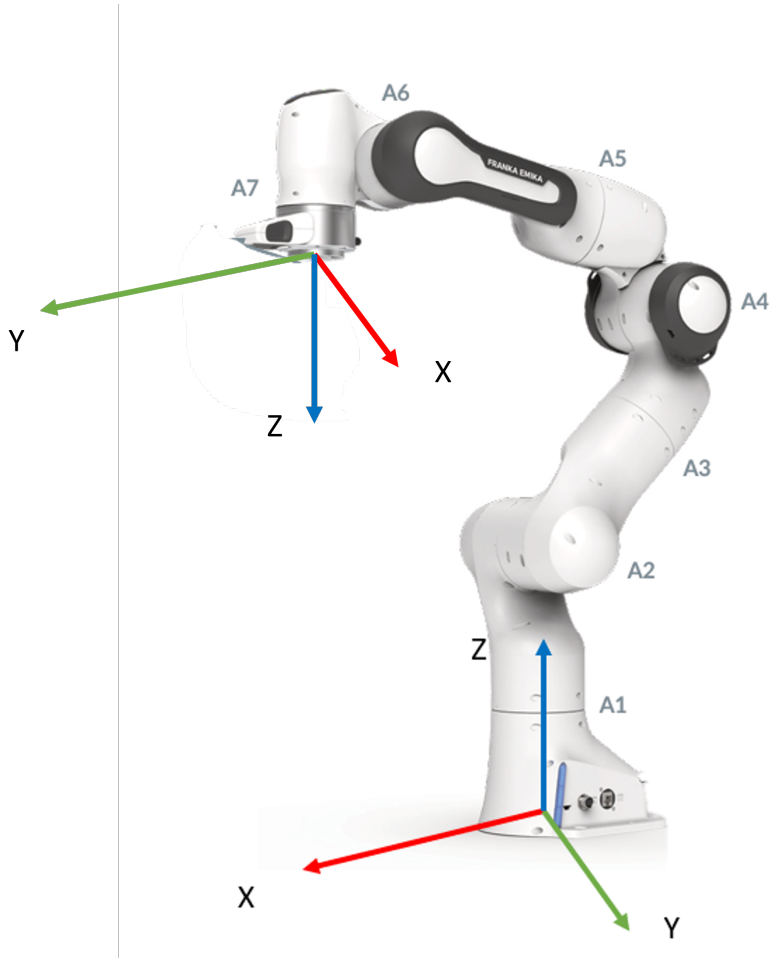


Figure 1.1: The arms and joints of the Panda robot

Here are the dimensions of the robot.

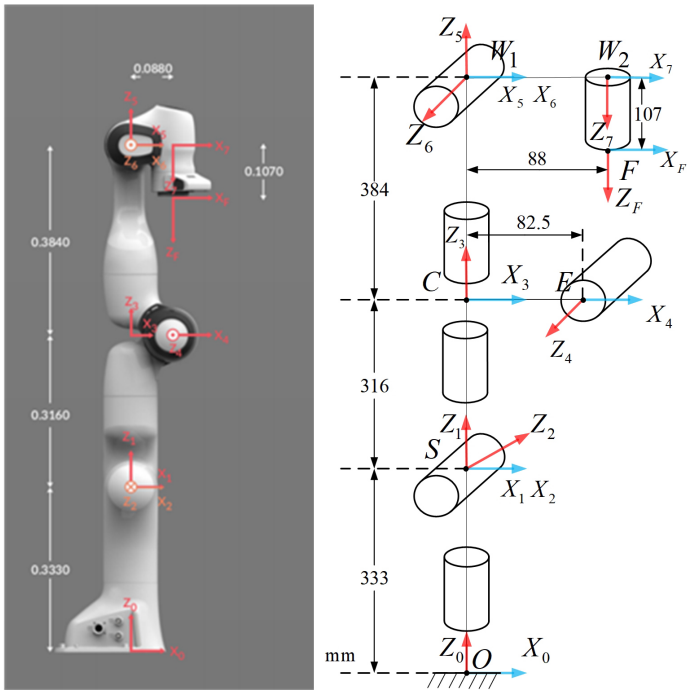


Figure 1.2: The parameters of Panda

The initial configuration is as follows:

$$M = \begin{bmatrix} 0 & 1 & 0 & 0.088 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0.926 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Here is the twist for each joints.

	ω_i	v_i
A1	$[0,0,1]^T$	$[0,0,0]^T$
A1	$[0,1,0]^T$	$[-0.333,0,0]^T$
A3	$[0,0,1]^T$	$[0,0,0]^T$
A4	$[0,-1,0]^T$	$[0.649,0,-0.0825]^T$
A5	$[0,0,1]^T$	$[0,0,0]^T$
A6	$[0,-1,0]^T$	$[1.033,0,0]^T$
A7	$[0,0,-1]^T$	$[0,-0.088,0]^T$

Rotation angle restrictions

	min/°	max/°
A1	-166	166
A1	-101	101
A3	-166	166
A4	-176	-4
A5	-166	166
A6	-1	215
A7	-166	166

Joint velocity limits

	limits $/(°/s)$
A1	150
A1	150
A3	150
A4	150
A5	180
A6	180
A7	180

1.2 Programming Assignments

a) Find the FK of Panda using the space form of the exponential products

The forward kinematics using the Product of Exponentials formula in the space frame is given by:

$$T(\theta) = e^{[S_1]\theta_1} e^{[S_2]\theta_2} \cdots e^{[S_n]\theta_n} M \tag{1.1}$$

This is implemented in **FK_space.m**. This function takes the initial configuration of the end-effector, the screw axes and the joint angles as pararmeters and output the coordinates of the end-effector. The declaration of the function is as follows:

```
function [T] = FK_space(M, S, theta)
% FK_space calculates the configuration of the end-effector
% Inputs:
%     M is a matrix of 4x4 of initial configuration of end-effector
%     S is a matrix of 6xn of screw axes
%     theta is a joint angle vector nx1
% Outputs:
%   T   - 4x4 the configuration of the end-effector
```

The test program is called **FK_space_test.m**. In this program, we tested several cases of the Panda robot and compare it with the results from matlab robotics toolbox. (Note that the Panda in MATLAB robotics toolbox uses a different initial config of joint 7 with our config. Specifically, when we set the angle of joint 7 to be $-\pi/4$ and all other angles to be zero, we recover our initial configuration.)

This is the screw axes for each joint of Panda

	S
A1	$[0, 0, 1, 0, 0, 0]^T$
A1	$[0, 1, 0, -0.333, 0, 0]^T$
A3	$[0, 0, 1, 0, 0, 0]^T$
A4	$[0, -1, 0, 0.649, 0, -0.0825]^T$
A5	$[0, 0, 1, 0, 0, 0]^T$
A6	$[0, -1, 0, 1.033, 0, 0]^T$
A7	$[0, 0, -1, 0, 0.088, 0]^T$

Case 1: Forward Reach

$$T_d = \begin{bmatrix} 0 & 1 & 0 & 0.7 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0.3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Case 2: Side Reach

$$T_d = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0.7 \\ 0 & 0 & -1 & 0.3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Case 3: Backward Reach

$$T_d = \begin{bmatrix} 0 & 1 & 0 & -0.6 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0.3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Case 4: Near singularity

$$T_d = \begin{bmatrix} 0 & 1 & 0 & 0.088 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0.926 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Case 1: Benchmark (Alough it is not feasible due to the angle constraint of joint 4)

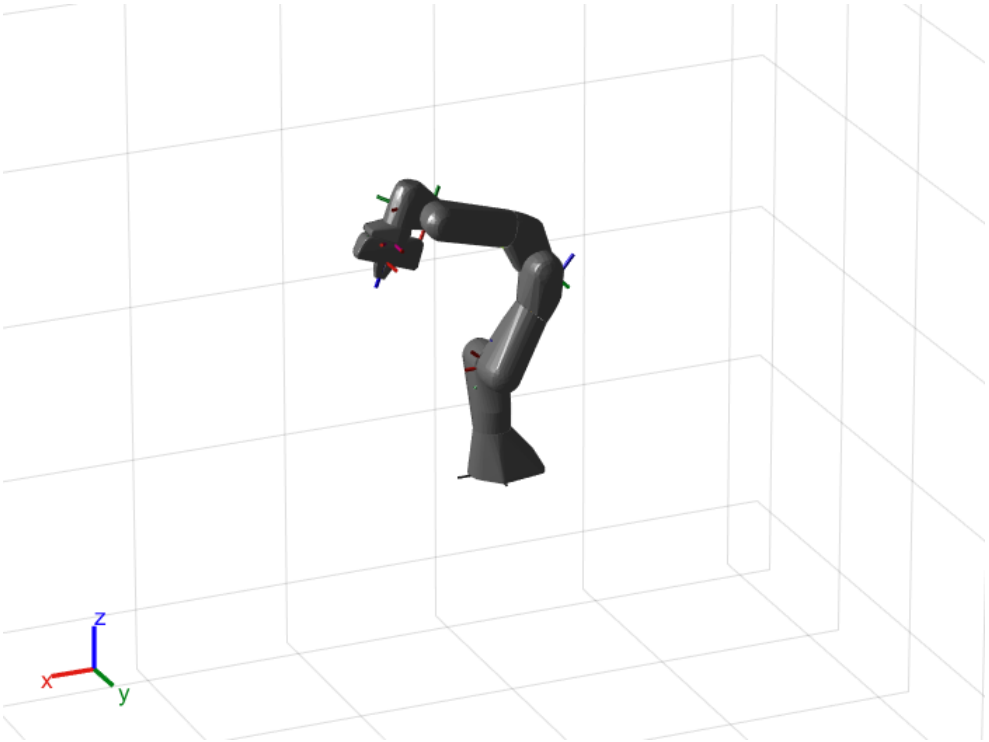
We set $\theta = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$. In this case, the configuration should be the initial configuration and it is.

Case 2

We set $\theta = \begin{bmatrix} 0 & -40^o & 0 & -110^o & 0 & 90^o & 0 \end{bmatrix}^T$. In this case, we try to recover the case in 1.1. Again, we got the same configuration with the MATLAB robotics toolbox.

$$T = \begin{bmatrix} 0 & 0.9397 & 0.3420 & 0.312 \\ 1 & 0 & 0 & 0 \\ 0 & 0.3420 & -0.9397 & 0.7665 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This is a visualization using MATLAB robotics toolbox.

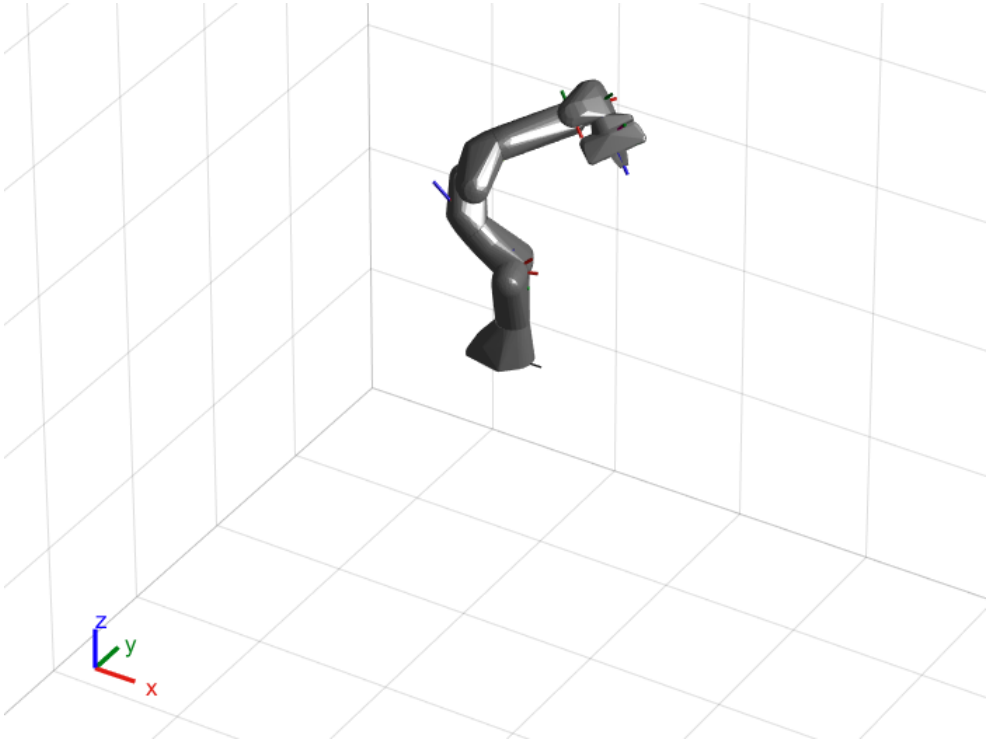


Case 3

We set $\theta = \begin{bmatrix} 20^o & -40^o & 30^o & -110^o & -25^o & 90^o & 10^o \end{bmatrix}^T$. This is just a random case to test the rotation along z axis (odd number joints). Again, we got the same configuration with the MATLAB robotics toolbox.

$$T = \begin{bmatrix} -0.3041 & 0.6951 & 0.6515 & 0.1803 \\ 0.7230 & 0.6137 & -0.3173 & 0.3012 \\ -0.6204 & 0.3745 & -0.6891 & 0.7456 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix}$$

This is a visualization using MATLAB robotics toolbox.



b) Calculate the space form FK of Panda and represent the defined frames and screw axis graphically

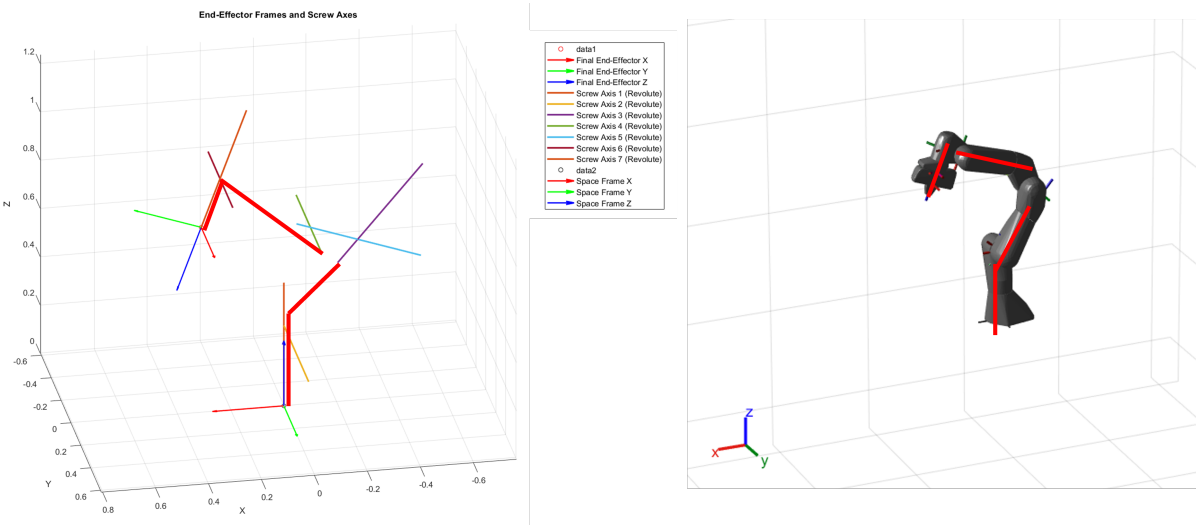
This is implemented in **visualize_FK_space.m** (instead of **FK_space.m**, because **FK_space.m** will be used later for many times to calculate the configuration of the end-effector and we think it is annoyting to plot the graph every time we call the function to get the end-effector).

This function takes the initial configuration of the end-effector, the screw axes, the initial rotation directions, the initial joint positions and the joint angles. The declaration of the function is as follows:

```
function visualize_FK_space(M, S, omega, r, theta)
% FK_space calculates the configuration of the end-effector
% Inputs:
%     M is a matrix of 4x4 of initial configuration of end-effector
%     S is a matrix of 6xn of screw axes
%     omega is a matrix of 3xn of initial rotation directions
%     r is a matrix of 3xn of initial joint positions
%     theta is a joint angle vector nx1
% Outputs:
%   T   - 4x4 the configuration of the end-effector
```

This function does pretty much the same thing with **FK_space.m** except it plots the screw axes of each joint at configuration specified by parameter **theta**.

In the test program **visualize_FK_space_test.m**. We tested **Case 2** where $\theta = \begin{bmatrix} 0 & -40^o & 0 & -110^o & 0 & 90^o & 0 \end{bmatrix}^T$. Here is the graph generated by it compared with the MATLAB robotics toolbox.



c) Repeat (a) and (b) for body form FK

This is implemented in **FK_body.m**. This function takes the initial configuration of the end-effector, the screw axes in body form and the joint angles as parameters and outputs the coordinates of the end-effector.

This is the screw axes in body form for each joint of Panda

	B
A1	$[0, 0, -1, 0.088, 0, 0]^T$
A1	$[1, 0, 0, 0, 0.593, 0.088]^T$
A3	$[0, 0, -1, 0.088, 0, 0]^T$
A4	$[-1, 0, 0, 0, -0.277, -0.0055]^T$
A5	$[0, 0, -1, 0.088, 0, 0]^T$
A6	$[-1, 0, 0, 0, 0.1070, -0.088]^T$
A7	$[0, 0, 1, 0, 0, 0]^T$

The test program is called **FK_body_test.m** where the three cases in **FK_space_test.m** were used and validated against the results here.

Similarly, we have the **visualize_FK_body.m** to plot the defined frames and screw axes graphically and the corresponding test program **visualize_FK_body_test.m** and we get the same results.

d) Find the space and body form Jacobian of Panda

We use formulae

$$\mathcal{V}_s = \begin{bmatrix} J_{s1} & J_{s2} & \cdots & J_{sn} \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \vdots \\ \dot{\theta}_n \end{bmatrix}$$

where

$$\begin{aligned} J_{s1} &= \mathcal{S}_1 \\ J_{s2} &= \mathrm{Ad}_{e^{[\mathcal{S}_1]\theta_1}} \mathcal{S}_2 \\ &\cdots \end{aligned}$$

Similarly, for body form Jacobian:

$$\mathcal{V}_b = \begin{bmatrix} J_{b1} & J_{b2} & \cdots & J_{bn} \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \vdots \\ \dot{\theta}_n \end{bmatrix}$$

where

$$\begin{aligned} J_{bn} &= \mathcal{B}_n \\ J_{b,n-1} &= \mathrm{Ad}_{e^{-[B_n]\theta_n}} \mathcal{B}_{n-1} \\ &\dots \end{aligned}$$

e) Write functions to calculate the space and body form Jacobians of the robot

These are implemented in **J_space.m** and **J_body.m** for space and body form Jacobians, respectively. For the space-form Jacobian, the function takes the screw axes in body form and the joint angles as parameters and outputs the space-form Jacobian of the end-effector. The declaration of the function is as follows:

```
function [J] = J_space(S, theta)
% J_SPACE Calculate the space Jacobian matrix for a serial robot
%
% Inputs:
%   S     - 6xn matrix of normalized twists, where each column is a 6x1 twist
%           vector [omega; v] representing the joint axis in space frame
%   theta - nx1 vector of joint angles/positions
%
% Output:
%   J     - 6xn space Jacobian matrix
```

The body form of the Jacobian is similar.

Case 1-3

The test program is called **J_space_body_test.m** to test the two functions. Again, we used the three cases mentioned earlier. The test is two-fold. Firstly, we use cross-validation of the space-form Jacobian and body-form Jacobian with the following formulae:

$$J_s = \mathrm{Ad}_{T_{sb}} J_b$$

Secondly, we validate our results with the results from MATLAB robotics toolbox. Note that the Jacobian calculated by **geometricJacobian** is actually analytic form Jacobian, as stated in Lynch and Park, 2017. And it can be transformed to body-form Jacobian using the following formulae:

$$J_a = \begin{bmatrix} R_{sb} & 0 \\ 0 & R_{sb} \end{bmatrix} J_b$$

Case 4

We further tested our program using the definition of Jacobian matrix.

$$J(\theta)\Delta\theta = \mathrm{Ad}(\Delta T) \tag{1.2}$$

or

$$J(\theta)\Delta\theta = \mathrm{Ad}_{T(\theta+\Delta\theta)/T(\theta)} \tag{1.3}$$

For example, when $\theta = \begin{bmatrix} 0 & -40^o & 0 & -110^o & 0 & 90^o & 0 \end{bmatrix}^T$ and $\Delta\theta$ is chosen to be $\theta = \pi/200 \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}^T$. We get

$$\mathrm{Ad}_{T(\theta+\Delta\theta)/T(\theta)} = \begin{bmatrix} 0.0106 & -0.0154 & 0.0185 & 0.0175 & 0.0172 & -0.0009 \end{bmatrix}^T \tag{1.4}$$

And

$$J(\theta)\Delta\theta = \begin{bmatrix} 0.0100 & -0.0157 & 0.0184 & 0.0178 & 0.0167 & -0.0008 \end{bmatrix}^T \tag{1.5}$$

which are very close, as expected.

f) Write a function singularity.m to calculate the singularity configurations of the robot

According to He and Liu, 2021 and Tittel, 2021, Panda's singularity exists when joints 1, 3 and 5 aligned and the robot would lose two DoF, leaving it with just 5-DoF and incapable of certain motion. It is acchieved when $\theta = \begin{bmatrix}0 & 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}^T$, as shown in the following picture.

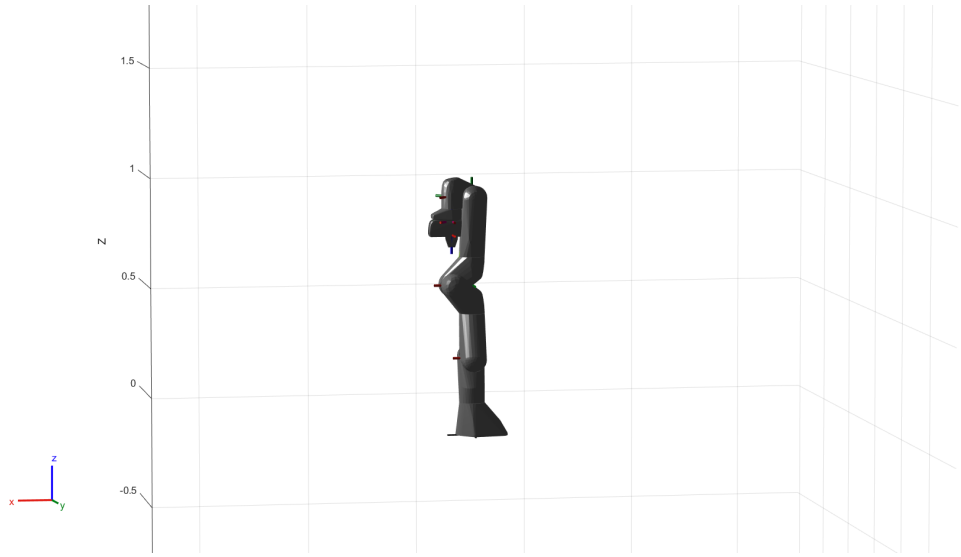


Figure 1.3: Singularity case

This could be validated using the rank of Jacobian matrix. Furthermore, we can use function **jsingu** to show which columns are linearly dependent.

```
          5

2 linearly dependent joints:
  q3 depends on: q1
  q5 depends on: q1
```

g) Write functions that

a) show/plot the manipulability ellipsoids for the angular and linear velocities and their axes

The two functions are implemented in **ellipsoid_plot_angular.m** and **ellipsoid_plot_linear.m**, respectively. And their test program with sufix **_test**.

When $\theta = \begin{bmatrix}0 & 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}^T$, we know it is at singularity and the ellipsoid should be a plane for angular velocities. (Because we lost the freedom in angular motions instead of linear motions) And so did our program got.

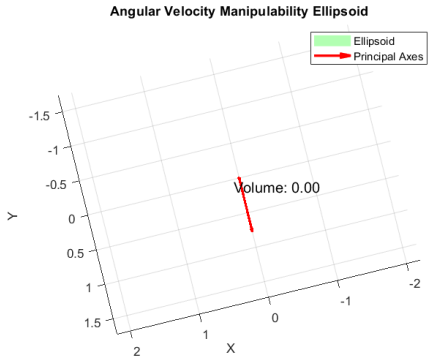


Figure 1.4: Angular Velocity Manipulability Ellipsoid

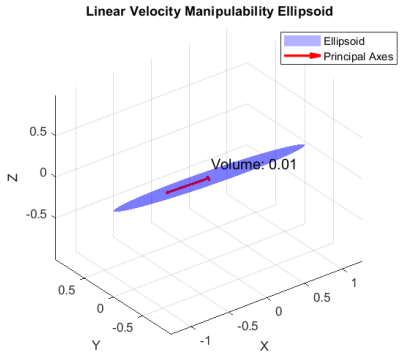


Figure 1.5: Linear Velocity Manipulability Ellipsoid

The following are the results when $\theta = \begin{bmatrix} 0 & -40^o & 0 & -110^o & 0 & 90^o & 0 \end{bmatrix}^T$ which is not at singularity.

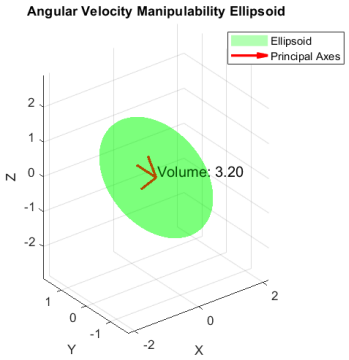


Figure 1.6: Angular Velocity Manipulability Ellipsoid

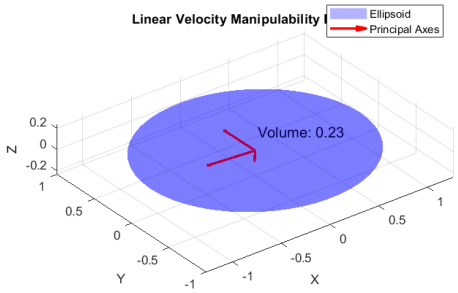


Figure 1.7: Linear Velocity Manipulability Ellipsoid

b) Calculate the isotropy, condition number and volume of the ellipsoids

	Case 1 $\theta = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$		Case 2 $\theta = \begin{bmatrix} 0 & -40^o & 0 & -110^o & 0 & 90^o & 0 \end{bmatrix}^T$	
	space-form Jacobian	body-form Jacobian	space-form Jacobian	body-form Jacobian
isotropy	Inf	Inf	10.90	9.75
condition number	Inf	Inf	118	95
volume of the ellipsoid	0	0	0.0929	0.0929

Table 1.1: test results for isotropy, condition number and volume of the ellipsoids

From the test results, we observe that

1. when $\theta = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$, the robot is at singularity. Therefore, the isotropy and condition number is infinity which is as expected.

2. when $\theta = \begin{bmatrix} 0 & -40^o & 0 & -110^o & 0 & 90^o & 0 \end{bmatrix}^T$, the isotropy and condition number are greater than 1. The volume by calculated by space-form Jacobian and body-form Jacobian are the same.

h) Use the derived forward kinematics and Jacobians, write a function that uses the iterative numerical inverse kinematics algorithm to control the robot from arbitrary configuration a to configuration b

Here, we used the space-form Jacobian to conduct inverse kinematics. The update formulae is

Algorithm 1 Iteration Framework for All Inverse Kinematics Methods

Input:
  M: 4×4 home configuration matrix
  S: 6×n matrix of normalized twists in space frame
  T_d: 4×4 desired end-effector configuration matrix
  θ_0: $n \times 1$ initial joint angles
  ϵ: Error tolerance (default: 10^{-4})
Output:
  θ^*: $n \times 1$ joint angles achieving T_d (or approximation)

Initialization:
$\theta \leftarrow \theta_0$ ▷ Set initial joint angles
$\mathbf{e} \leftarrow \mathbf{1}_{n\times 1}$ ▷ Initialize error vector
while $\|\mathbf{e}\| > \epsilon$ **do** ▷ Check convergence
  $T_c \leftarrow FK_{space}(M, S, \theta)$ ▷ Compute current pose via forward kinematics
  $\mathbf{e} \leftarrow Ad(T_c) \cdot \log(T_c^{-1} T_d)$ ▷ Compute twist error in space frame
  $J \leftarrow J_{space}(S, \theta)$ ▷ Compute space Jacobian
  $\Delta\theta \leftarrow f(\mathbf{e})$ ▷ Different update formulaes
  $\theta \leftarrow \theta + \Delta\theta$ ▷ Update joint angles
end while
return θ ▷ Final joint configuration

$$\begin{aligned} e &= \mathrm{Ad}_{T_{sb}}[T_{sb}^{-1} T_{sd}] \\ \Delta\theta &= J^{\dagger} e \\ \theta &\leftarrow \theta + \Delta\theta \end{aligned} \tag{1.6}$$

The declaration of the function is as follows:

```
function theta = J_inverse_kinematics(M, S, T_desired, theta_init, epsilon)
% J_inverse_kinematics Solve inverse kinematics using iterative Newton-Raphson
    method
%
% Inputs:
%   B           - 6xn matrix of normalized twists in body frame
%   M           - 4x4 home configuration matrix
%   T_desired   - 4x4 desired end-effector configuration matrix
%   theta_init  - nx1 initial guess for joint angles
%   epsilon     - Error tolerance (default: 1e-4)
%
% Outputs:
%   theta   - nx1 vector of joint angles that achieve T_desired
%   success - Boolean indicating whether algorithm converged
%
% Note:
%   Uses Newton-Raphson iteration with space Jacobian.
```

The test program is called **J_inverse_kinematics_test.m** to test the function.

Case 1

In this case, we set T_{sd} (which is random)

$$T_{sd} = \begin{bmatrix} 0.3862 & -0.2690 & -0.8823 & 0.4225 \\ 0.8917 & 0.3535 & 0.2826 & 0.3776 \\ 0.2359 & -0.8959 & 0.3764 & -0.0874 \\ 0 & 0 & 0 & 1.00 \end{bmatrix} \tag{1.7}$$

And we use random initial $\theta = \begin{bmatrix} 0 & -\pi/2 & 0 & -\pi/2 & 0 & 0 & 0 \end{bmatrix}^T$. (Note to keep to within the joint angle constraints). The followings pictures shows the initial and desired configuration of the robot.

Figure 1.8: Initial Configuration

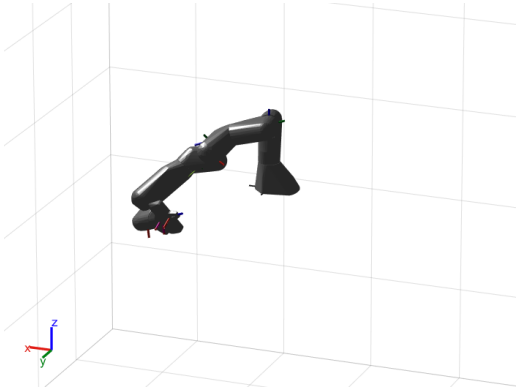


Figure 1.9: Desired Configuration

This program output $\theta = \begin{bmatrix}0.5642 & 1.8297 & -0.1648 & -0.6123 & 1.3935 & 0.7434 & 0.4404\end{bmatrix}^T$ and exactly the same final configuration of the end-effector.

Case 2

In this case, we test the robustness of the program by setting the desired configuration to be very close to singularity. ($\theta = \begin{bmatrix}0.01 & 0.02 & 0.03 & -0.1 & 0.05 & 0.06 & 0.07\end{bmatrix}^T$) The desired configuration is

$$T_{sd} = \begin{bmatrix} -0.0198 & 0.9980 & -0.0601 & 0.1339 \\ 0.9998 & 0.0199 & 0.0012 & 0.0097 \\ 0.0024 & -0.0601 & -0.9982 & 0.9263 \\ 0 & 0 & 0 & 1.00 \end{bmatrix} \tag{1.8}$$

And the condition number at this place is 1621.7.

And we use initial guess $\theta = \begin{bmatrix}0 & 1 & 0 & -0.5 & 0 & 0 & 0\end{bmatrix}^T$. The followings pictures shows the initial and desired configuration of the robot.

Figure 1.10: Initial Configuration

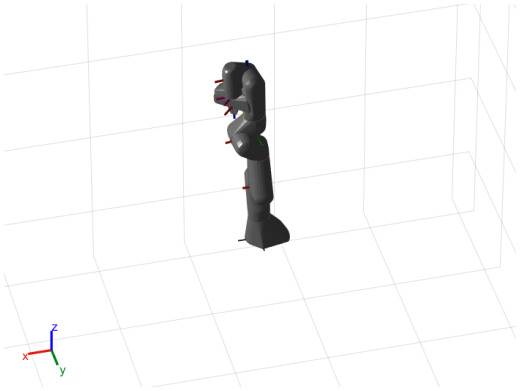


Figure 1.11: Desired Configuration

This program output exactly the same final configuration of the end-effector and $\theta = \begin{bmatrix}-3.2855 & 12.1961 & 5.6213 & 18.3704 & 2.4528 & -6.5735 & 4.8143\end{bmatrix}^T$. The θ exceeds the joint angles limits and needed further modification.

i) Use Jacobian Transpose algorithm

Again, we used the space-form Jacobian to conduct transpose kinematics. According to Buss, 2004, the update formulae is

$$e = \mathrm{Ad}_{T_{sb}}[T_{sb}\ T_{sd}] \tag{1.9}$$

$$\Delta\theta = \alpha J^T e$$

$$\alpha = \frac{< e, JJ^T e >}{< JJ^T e, JJ^T e >} \tag{1.10}$$

We used the two cases introduced in **J_inverse_kinematics_test.m** test the function in **J_transpose_kinematics_test.m**

Case 1

In this case, we set T_{sd}

$$T_{sd} = \begin{bmatrix} 0.3862 & -0.2690 & -0.8823 & 0.4225 \\ 0.8917 & 0.3535 & 0.2826 & 0.3776 \\ 0.2359 & -0.8959 & 0.3764 & -0.0874 \\ 0 & 0 & 0 & 1.00 \end{bmatrix} \tag{1.11}$$

This program outputs $\theta = \begin{bmatrix} 0.3642 & 1.8267 & 0.3159 & -0.6573 & 0.8888 & 0.8956 & 0.4970 \end{bmatrix}^T$ and exactly the same final configuration of the end-effector.

Case 2

In this case, the desired configuration is

$$T_{sd} = \begin{bmatrix} -0.0198 & 0.9980 & -0.0601 & 0.1339 \\ 0.9998 & 0.0199 & 0.0012 & 0.0097 \\ 0.0024 & -0.0601 & -0.9982 & 0.9263 \\ 0 & 0 & 0 & 1.00 \end{bmatrix} \tag{1.12}$$

This program output exactly the same final configuration of the end-effector and $\theta = \begin{bmatrix} 0.0046 & 0.0208 & 0.0369 & -0.0985 & 0.0491 & 0.0593 & 0.0707 \end{bmatrix}^T$, which is a much better result than **J_inverse_kinematics.m**

j) Use the redundancy resolution approach to maximize the manipulability and exploit redundancy to move away from singularities

We use volume of the manipulability ellipsoids as the second objective function.

$$\Delta\theta = J^{\dagger}e + \alpha N \nabla V \tag{1.13}$$

$$V = \sqrt{\det(JJ^T)}$$

$$N = I - J^{\dagger}J \tag{1.14}$$

Here, we choose $\alpha = 0.1$

Case 1

In this case, we set T_{sd}

$$T_{sd} = \begin{bmatrix} 0.3862 & -0.2690 & -0.8823 & 0.4225 \\ 0.8917 & 0.3535 & 0.2826 & 0.3776 \\ 0.2359 & -0.8959 & 0.3764 & -0.0874 \\ 0 & 0 & 0 & 1.00 \end{bmatrix} \tag{1.15}$$

This program outputs $\theta = \begin{bmatrix} 0.5631 & 1.8295 & -0.1622 & -0.6124 & 1.3906 & 0.7442 & 0.4405 \end{bmatrix}^T$ and exactly the same final configuration of the end-effector.

Case 2

In this case, the desired configuration is

$$T_{sd} = \begin{bmatrix} -0.0198 & 0.9980 & -0.0601 & 0.1339 \\ 0.9998 & 0.0199 & 0.0012 & 0.0097 \\ 0.0024 & -0.0601 & -0.9982 & 0.9263 \\ 0 & 0 & 0 & 1.00 \end{bmatrix} \tag{1.16}$$

This program output exactly the same final configuration of the end-effector and $\theta = \begin{bmatrix} -3.3297 & 12.1911 & 5.6309 & 18.3437 & 2.4948 & -6.5875 & 4.8244 \end{bmatrix}^T$, which is a very close to the result from **J_inverse_kinematics.m** and not a reasonable result.

h) Extend the inverse kinematics utilizing the Damped Least Square Approach

We use the volume of the manipulability ellipsoids as judging criterion, when the volume is large (larger than 0.01 in our program) we stick to the method in **J_inverse_kinematics.m**. Otherwise, the update formulae is switched to

$$\Delta\theta = J^T(JJ^T + k^2 I)e \tag{1.17}$$

Case 1

In this case, we set T_{sd}

$$T_{sd} = \begin{bmatrix} 0.3862 & -0.2690 & -0.8823 & 0.4225 \\ 0.8917 & 0.3535 & 0.2826 & 0.3776 \\ 0.2359 & -0.8959 & 0.3764 & -0.0874 \\ 0 & 0 & 0 & 1.00 \end{bmatrix} \tag{1.18}$$

This program outputs $\theta = \begin{bmatrix} 0.5642 & 1.8297 & -0.1648 & -0.6123 & 1.3935 & 0.7434 & 0.4404 \end{bmatrix}^T$ which is exactly the same with the result from **J_inverse_kinematics.m**. This is because in this case, the robot is far from the singularity. Therefore, it works exactly the same with **J_inverse_kinematics.m**.

Case 2

In this case, the desired configuration is

$$T_{sd} = \begin{bmatrix} -0.0198 & 0.9980 & -0.0601 & 0.1339 \\ 0.9998 & 0.0199 & 0.0012 & 0.0097 \\ 0.0024 & -0.0601 & -0.9982 & 0.9263 \\ 0 & 0 & 0 & 1.00 \end{bmatrix} \tag{1.19}$$

This program output exactly the same final configuration of the end-effector and $\theta = \begin{bmatrix} 0.0195 & 0.0206 & 0.0182 & -0.0989 & 0.0519 & 0.0595 & 0.0696 \end{bmatrix}^T$, which is a much better result than **J_inverse_kinematics.m**

Chapter 2

Sensor Data Integration

2.1 PA1

2.1.1 3D Point Set Registration Algorithm

We developed three algorithms (SVD approach, Eigenvalue Decomposition, and Quarternion method) and a simple test program **test_correspondance.m** to compare and validate them.

Problem Formulation

Given corresponding points $\mathbf{p}_i \leftrightarrow \mathbf{q}_i$, the goal is to find $\mathbf{R}$ and $\mathbf{t}$ that minimize the least-squares error:

$$E(\mathbf{R}, \mathbf{t}) = \sum_{i=1}^{N} \|\mathbf{q}_i - (\mathbf{R}\mathbf{p}_i + \mathbf{t})\|^2. \tag{2.1}$$

Singular Value Decomposition (SVD) for Rotation Estimation

SVD is used to estimate the optimal rotation matrix $\mathbf{R}$ that aligns the centered point clouds.

Steps:

1. Compute centroids:
$$\bar{\mathbf{p}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{p}_i, \quad \bar{\mathbf{q}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{q}_i$$

2. Center the points:
$$\mathbf{p}'_i = \mathbf{p}_i - \bar{\mathbf{p}}, \quad \mathbf{q}'_i = \mathbf{q}_i - \bar{\mathbf{q}}$$

3. Compute the cross-covariance matrix:
$$\mathbf{H} = \sum_{i=1}^{N} \mathbf{p}'_i (\mathbf{q}'_i)^T$$

4. Perform SVD:
$$\mathbf{H} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

5. Compute rotation:
$$\mathbf{R} = \mathbf{V}\mathbf{U}^T$$

6. Ensure $\mathbf{R} \in SO(3)$ by checking $\det(\mathbf{R}) = 1$. If $\det(\mathbf{R}) < 0$, adjust $\mathbf{V}$:
$$\mathbf{V}' = \mathbf{V} \cdot \mathrm{diag}(1, 1, -1), \quad \mathbf{R} = \mathbf{V}'\mathbf{U}^T$$

Quaternion-Based Rotation Estimation

According to Horn, 1987

Steps:

1. Compute the centroids of the point sets:

$$\bar{\mathbf{a}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{a}_i, \quad \bar{\mathbf{b}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{b}_i$$

2. Center the point sets:

$$\mathbf{a}'_i = \mathbf{a}_i - \bar{\mathbf{a}}, \quad \mathbf{b}'_i = \mathbf{b}_i - \bar{\mathbf{b}}$$

3. Construct the 4×4 matrix $\mathbf{M}$ by summing contributions from each point pair:

$$\mathbf{M} = \sum_{i=1}^{N}\mathbf{M}_i^T\mathbf{M}_i,$$

where for each point pair $\mathbf{a}'_i$, $\mathbf{b}'_i$:

$$\mathbf{b}'_i - \mathbf{a}'_i = \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix}, \quad \mathbf{b}'_i + \mathbf{a}'_i = \begin{bmatrix} u_i \\ v_i \\ w_i \end{bmatrix},$$

$$\mathbf{M}_i = \begin{bmatrix} 0 & x_i & y_i & z_i \\ x_i & 0 & -w_i & v_i \\ y_i & w_i & 0 & -u_i \\ z_i & -v_i & u_i & 0 \end{bmatrix},$$

and $\mathbf{M}_i$ is derived from the difference and sum of centered points, with vecToso3($\mathbf{b}'_i + \mathbf{a}'_i$) representing the skew-symmetric matrix:

$$\text{vecToso3}\left(\begin{bmatrix} u_i \\ v_i \\ w_i \end{bmatrix}\right) = \begin{bmatrix} 0 & -w_i & v_i \\ w_i & 0 & -u_i \\ -v_i & u_i & 0 \end{bmatrix}.$$

4. Perform eigenvalue decomposition on $\mathbf{M}$:

$$\mathbf{M}\mathbf{q} = \lambda\mathbf{q},$$

where $\mathbf{q}$ is an eigenvector, and λ is the corresponding eigenvalue.

5. Select the eigenvector $\mathbf{q} = [q_0, q_1, q_2, q_3]^T$ corresponding to the *smallest* eigenvalue as the unit quaternion.

6. Convert the quaternion to a rotation matrix $\mathbf{R}$:

$$\mathbf{R} = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 - q_0q_3) & 2(q_1q_3 + q_0q_2) \\ 2(q_1q_2 + q_0q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_2q_3 + q_0q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}$$

7. Compute the translation vector:

$$\mathbf{p} = \bar{\mathbf{b}} - \mathbf{R}\bar{\mathbf{a}}$$

Eigenvalue Decomposition for Rotation Estimation

Eigenvalue decomposition analyzes the covariance of the aligned points to assess uncertainty or stability.

Steps:

1. Compute the centroids of the point sets:

$$\bar{\mathbf{a}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{a}_i, \quad \bar{\mathbf{b}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{b}_i$$

2. Compute the cross-covariance matrix $\mathbf{H}$:

$$\mathbf{H} = \sum_{i=1}^{N} (\mathbf{a}_i - \bar{\mathbf{a}})(\mathbf{b}_i - \bar{\mathbf{b}})^T$$

3. Construct the 4×4 matrix $\mathbf{G}$:

$$\text{trace}(\mathbf{H}) = h_{11} + h_{22} + h_{33},$$

$$\Delta = \begin{bmatrix} h_{23} - h_{32} \\ h_{31} - h_{13} \\ h_{12} - h_{21} \end{bmatrix},$$

$$\mathbf{G} = \begin{bmatrix} \text{trace}(\mathbf{H}) & \Delta^T \\ \Delta & \mathbf{H} + \mathbf{H}^T - \text{trace}(\mathbf{H})\mathbf{I}_3 \end{bmatrix},$$

where h_{ij} are elements of $\mathbf{H}$, and $\mathbf{I}_3$ is the 3×3 identity matrix.

4. Perform eigenvalue decomposition on $\mathbf{G}$:

$$\mathbf{G}\mathbf{q} = \lambda \mathbf{q},$$

where $\mathbf{q}$ is an eigenvector, and λ is the corresponding eigenvalue.

5. Select the eigenvector $\mathbf{q} = [q_0, q_1, q_2, q_3]^T$ corresponding to the largest eigenvalue as the unit quaternion.

6. Convert the quaternion to a rotation matrix $\mathbf{R}$:

$$\mathbf{R} = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1 q_2 - q_0 q_3) & 2(q_1 q_3 + q_0 q_2) \\ 2(q_1 q_2 + q_0 q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2 q_3 - q_0 q_1) \\ 2(q_1 q_3 - q_0 q_2) & 2(q_2 q_3 + q_0 q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}$$

7. Compute the translation vector:

$$\mathbf{p} = \bar{\mathbf{b}} - \mathbf{R}\bar{\mathbf{a}}$$

Translation Estimation

The translation vector is computed after rotation estimation using the following formulae:

$$\mathbf{t} = \bar{\mathbf{q}} - \mathbf{R}\bar{\mathbf{p}}$$

2.1.2 Pivot Calibration Method

For each measurement k, we have known $\mathbf{R}_k$ and $\mathbf{p}_k$ by correspondence method, then we can write:

$$\mathbf{b}_{\text{post}} = \mathbf{R}_k \mathbf{b}_{\text{tip}} + \mathbf{p}_k$$

We can rewrite this equation as:

$$\mathbf{R}_k \mathbf{b}_{\text{tip}} - \mathbf{b}_{\text{post}} = -\mathbf{p}_k$$

So, we can set up a least squares problem in the form of $\mathbf{A}\mathbf{x} = \mathbf{b}$ as follows and find the unknowns $\mathbf{b}_{\text{tip}}$ and $\mathbf{b}_{\text{post}}$: Lynch and Park, 2017

$$\begin{bmatrix} \vdots & \vdots \\ \mathbf{R}_k & -\mathbf{I} \\ \vdots & \vdots \end{bmatrix} \begin{bmatrix} \mathbf{b}_{\text{tip}} \\ \mathbf{b}_{\text{post}} \end{bmatrix} = \begin{bmatrix} \vdots \\ -\mathbf{p}_k \\ \vdots \end{bmatrix}$$

2.1.3 Results and Discussion

The main script is called **main.m** (which can run all the cases by simply modify the **filebase** variable to specify the datasets).

Here are the algorithm for Distortion Calibration and EM Tracker and Optical Tracker Pivot Calibration.

Algorithm 2 Distortion Calibration

Input:
- Calibration object data file (e.g., `pa1-debug-c-calbody.txt`) containing:
  - Coordinates: $\mathbf{d}_j$, $\mathbf{a}_j$, $\mathbf{c}_i$ of the calibration object.
- Sensor readings file (e.g., `pa1-debug-c-calreadings.txt`) containing:
  - Sensor data frames: D_j, A_j, C_i for each frame

Output:
- $C_i^{(\text{expected})}$: Expected EM tracker coordinates for each calibration point $\mathbf{c}_i$ in each frame

Step 1: Load Calibration Object Data
Open the calibration object file (e.g., `pa1-debug-c-calbody.txt`).
Read the coordinate data into matrices:
- $\mathbf{d}$: $N_d \times 3$ matrix of EM tracker points $\mathbf{d}_j$.
- $\mathbf{a}$: $N_a \times 3$ matrix of optical tracker points $\mathbf{a}_j$.
- $\mathbf{c}$: $N_c \times 3$ matrix of calibration object points $\mathbf{c}_i$.

Step 2: Load Sensor Data
Open the sensor readings file (e.g., `pa1-debug-c-calreadings.txt`).
Split the frame data into:
- D: $N_d \times 3$ matrix of EM tracker sensor readings D_j.
- A: $N_a \times 3$ matrix of optical tracker sensor readings A_j.
- C: $N_c \times 3$ matrix of calibration sensor readings C_i.

Step 3: Compute Transformation F_D (EM Tracker to Sensor Frame)
Construct the homogeneous transformation matrix F_D using correspondence method for each frame:

Step 4: Compute Transformation F_A (Optical Tracker to Calibration Object)
Construct the homogeneous transformation matrix F_A using correspondence method for each frame:

Step 5: Compute Expected Coordinates $C_i^{(\text{expected})}$
Compute the transformation:

$$C_i^{(\text{expected})} = F_D^{-1} \cdot F_A \cdot \mathbf{c}_i$$

Step 6: Output Results
Output $C_i^{(\text{expected})}$ for all i_c and all frames in the specified file format.

Algorithm 3 EM Tracker and Optical Tracker Pivot Calibration

Input: Observed points $\mathbf{G}_i$, $i = 1, \ldots, N_G$ (For optical data, they should be transformed)
Output: $\mathbf{b}_{post}$ and $\mathbf{b}_{tip}$
Step 1: Compute the midpoint $\mathbf{g}_0$ of the first frame

$$\mathbf{G}_0 = \frac{1}{N_G} \sum_{i=1}^{N_G} \mathbf{g}_i$$

Step 2: Translate observations relative to the midpoint

$$\mathbf{g}_i' = \mathbf{G}_i - \mathbf{G}_0 \quad \text{for} \quad i = 1, \ldots, N_G$$

Step 3: Compute the transformation for each frame k

$$\mathbf{G}_k = \mathbf{F}_k[\mathbf{g}] + \mathbf{t}_G$$

Step 4: Compute the $\mathbf{b}_{post}$ and $\mathbf{b}_{tip}$ using algorithm in 2.1.2

Debug datasets (a-g)

For known datasets (a-g). We write the results into **NAME-output2.txt** which exactly following the specified file format.

datasets	EM dstr fct	EM noise	Opt trk jggl fct	RMS of C	RMS of EM dmpl	RMS of Opt dmpl
a	0	0	0	0.0052 mm	0.0014 mm	0.0040 mm
b	0	0.5	0	0.4919 mm	0.0024 mm	0.0035 mm
c	0.01	0	0	0.4273 mm	0.0016 mm	0.0008 mm
d	0	0	0.01	0.0131 mm	0.0016 mm	0.0032 mm
e	0.02	0	0.01	1.7313 mm	0.0058 mm	0.0023 mm
f	0.02	0.2	0.01	1.9566 mm	0.0110 mm	0.0006 mm
g	0.02	0.2	0.01	2.0596 mm	0.0070 mm	0.0032 mm

Remarks: One interesting to note is that the RMS of C can be linearly attributed to the EM noise.

Unknown datasets (h-k)

And for unknown datasets (h-k). We write the results into **NAME-output1.txt** which exactly following the specified file format.

datasets	$\mathbf{b}_{post}$ in EM tracker	$\mathbf{b}_{post}$ in optical tracker	1st expected C
h	$(181.05, 182.96, 199.87)$	$(393.53, 400.35, 194.06)$	$(209.35, 209.69, 211.01)$
i	$(201.39, 207.12, 196.52)$	$(405.90, 407.60, 209.66)$	$(211.03, 209.43, 208.25)$
j	$(207.74, 205.29, 188.11)$	$(396.35, 402.93, 190.79)$	$(209.38, 209.93, 209.55)$
k	$(198.40, 201.69, 205.08)$	$(391.34, 397.05, 196.42)$	$(209.76, 208.20, 210.64)$

2.2 PA2

1)

Given rotation in the quaternion form and a translation vector for different robot and sensor configurations, solve for transformation X.

We first calculate the E_i and S_i based on the quaternions and translation vectors. The we construct transformation As and Bs based on consecutive E_i and S_i, respectively. After transforming the rotation matrices of As and Bs into quaternion form and construct matrix M, we can calculate the rotation R_X in quaternion form

$$\min \|\mathbf{M}\hat{\mathbf{q}}_X\| \quad \text{subject to} \quad \|\hat{\mathbf{q}}_X\| = 1 \tag{2.2}$$

where

$$\mathbf{M} = \begin{bmatrix} \mathbf{M}(q_{A,1}, q_{B,1}) \\ \vdots \\ \mathbf{M}(q_{A,n}, q_{B,n}) \end{bmatrix} \tag{2.3}$$

$$\mathbf{M}(q_A, q_B) = \begin{bmatrix} (\mathbf{s}_A - \mathbf{s}_B)(\mathbf{v}_A - \mathbf{v}_B)^\top \\ (\mathbf{v}_A - \mathbf{v}_B)(\mathbf{s}_A - \mathbf{s}_B)_3^\top + \mathrm{sk}(\mathbf{v}_A + \mathbf{v}_B) \end{bmatrix} \tag{2.4}$$

The analytical solution is

$$\hat{\mathbf{q}}_X = V_4 \tag{2.5}$$

where V_4 is the last eigen vector of the SVD decomposition of matrix $\mathbf{M}(q_A, q_B)$.

And then solve

$$(\mathbf{R}_{A,k} - \mathbf{I})\hat{\mathbf{p}}_X \approx \mathbf{R}_X\hat{\mathbf{p}}_{B,k} - \hat{\mathbf{p}}_{A,k} \tag{2.6}$$

to get $\hat{\mathbf{p}}_X$ using least square method (**pinv** in MATLAB).

The above algorithm is implemented in **hand_eye_calibration.m** which takes rotations in the quaternion form and translation vectors as parameters.

After calling the function from script **main.m**, we get

$$T = \begin{bmatrix} -0.0032 & 1.0000 & -0.0001 & 0.1355 \\ -0.0008 & 0.0001 & 1.0000 & -0.9677 \\ 1.0000 & 0.0032 & 0.0008 & -0.3144 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix} \tag{2.7}$$

2)

Here we deal with noisy data

a) Repeat part 1

After calling the function from script **main.m**, we get

$$T = \begin{bmatrix} -0.0034 & 1.0000 & 0.0000 & 0.1358 \\ -0.0009 & -0.0000 & 1.0000 & -0.9673 \\ 1.0000 & 0.0034 & 0.0009 & -0.3141 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix} \tag{2.8}$$

which is very close to noise free data. The translation error is 6.28×10^{-4}. The rotation error is 0.0135

b) Use half of the data sets to get the result

Here we randomly choose 5 configurations

```
rng(2025)
X = randperm(10, 5);
q_Robot_config_half = q_Robot_config(X, :);
q_camera_config_half = q_camera_config(X, :);
t_Robot_config_half = t_Robot_config(X, :);
```

This yields the following transformation matrix:

$$T = \begin{bmatrix} -0.0038 & 1.0000 & 0.0008 & 0.2321 \\ 0.0011 & -0.0008 & 1.0000 & -0.5425 \\ 1.0000 & 0.0038 & -0.0011 & 0.0391 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix} \tag{2.9}$$

The rotation matrix is very similar to previous results; however, the translation vector deviates significantly from them. The translation error is 0.5606. The rotation error is 0.1238 Therefore, obtaining an accurate translation vector requires more data.

Chapter 3

Constrained Control

3.1 Introduction to Franka Emika Panda

Panda is a 7-DoF robot arm. Here is the picture GmbH, 2017.

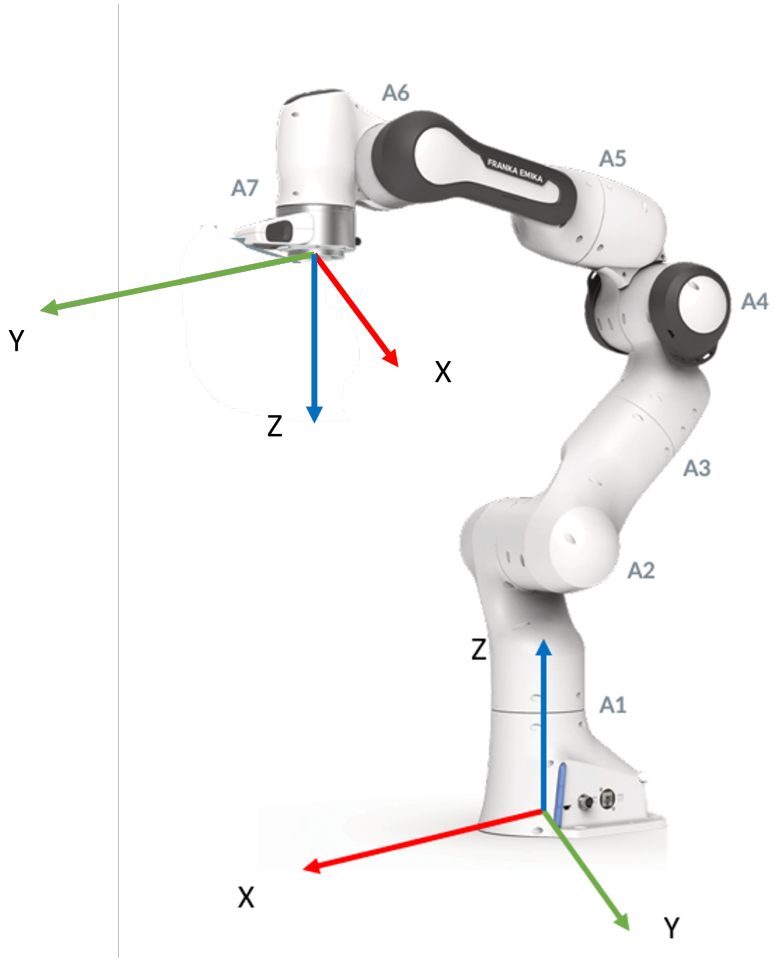


Figure 3.1: The arms and joints of the Panda robot

Here are the dimensions of the robot.

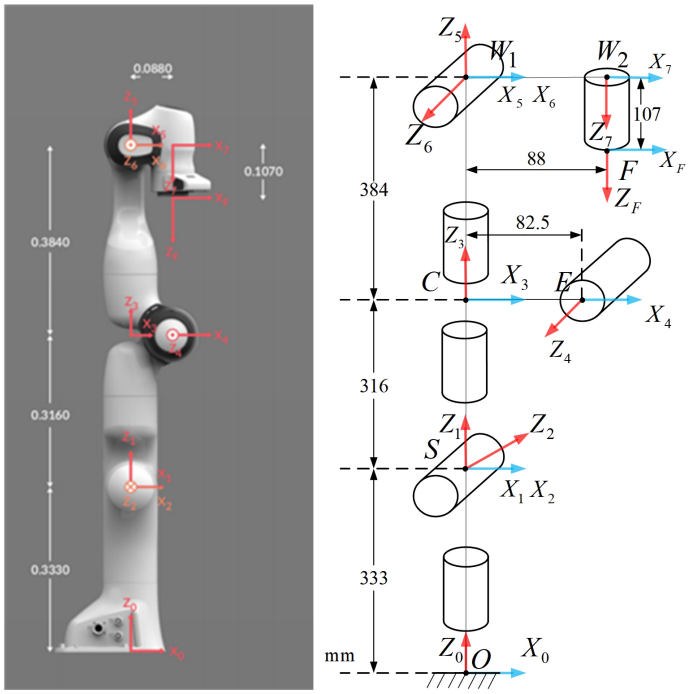


Figure 3.2: The parameters of Panda

The initial configuration of the end-effector is as follows:

$$M = \begin{bmatrix} 0.7071 & 0.7071 & 0 & 0.088 \\ 0.7071 & -0.7071 & 0 & 0 \\ 0 & 0 & -1 & 0.926 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

But after attaching the cylindrical tool, the end-effector become the tip of the tool.

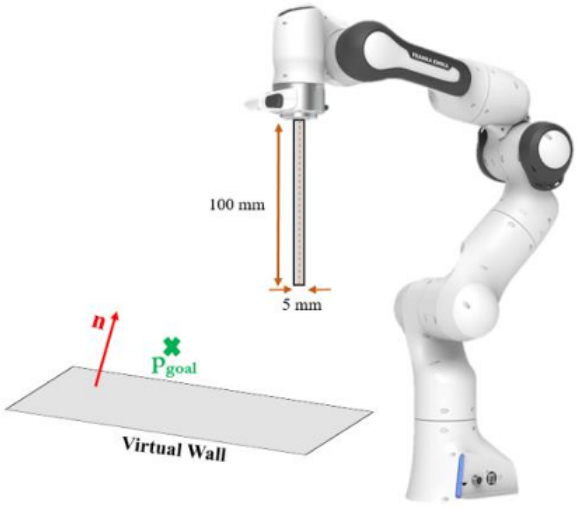


Figure 3.3: Panda with Cylindrical Tool

Therefore, the initial configuration of the cylindrical tool is

$$M = \begin{bmatrix} 0.7071 & 0.7071 & 0 & 0.088 \\ 0.7071 & -0.7071 & 0 & 0 \\ 0 & 0 & -1 & 0.826 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

Here are the twists for every joint.

Table 3.1: Normalized Twists

	ω_i	v_i
A1	$[0,0,1]^T$	$[0,0,0]^T$
A1	$[0,1,0]^T$	$[-0.333,0,0]^T$
A3	$[0,0,1]^T$	$[0,0,0]^T$
A4	$[0,-1,0]^T$	$[0.649,0,-0.0825]^T$
A5	$[0,0,1]^T$	$[0,0,0]^T$
A6	$[0,-1,0]^T$	$[1.033,0,0]^T$
A7	$[0,0,-1]^T$	$[0,-0.088,0]^T$

Joint angle restrictions

Table 3.2: Joint Angle Limits

	min/°	max/°
A1	-166	166
A1	-101	101
A3	-166	166
A4	-176	-4
A5	-166	166
A6	-1	215
A7	-166	166

3.2 Test Cases

2.0 Initial Guess

$$q = \begin{bmatrix} 0 & 0 & 0 & 0 & -\frac{\pi}{2} & 0 & \frac{\pi}{2} & 0 \end{bmatrix} \tag{3.2}$$

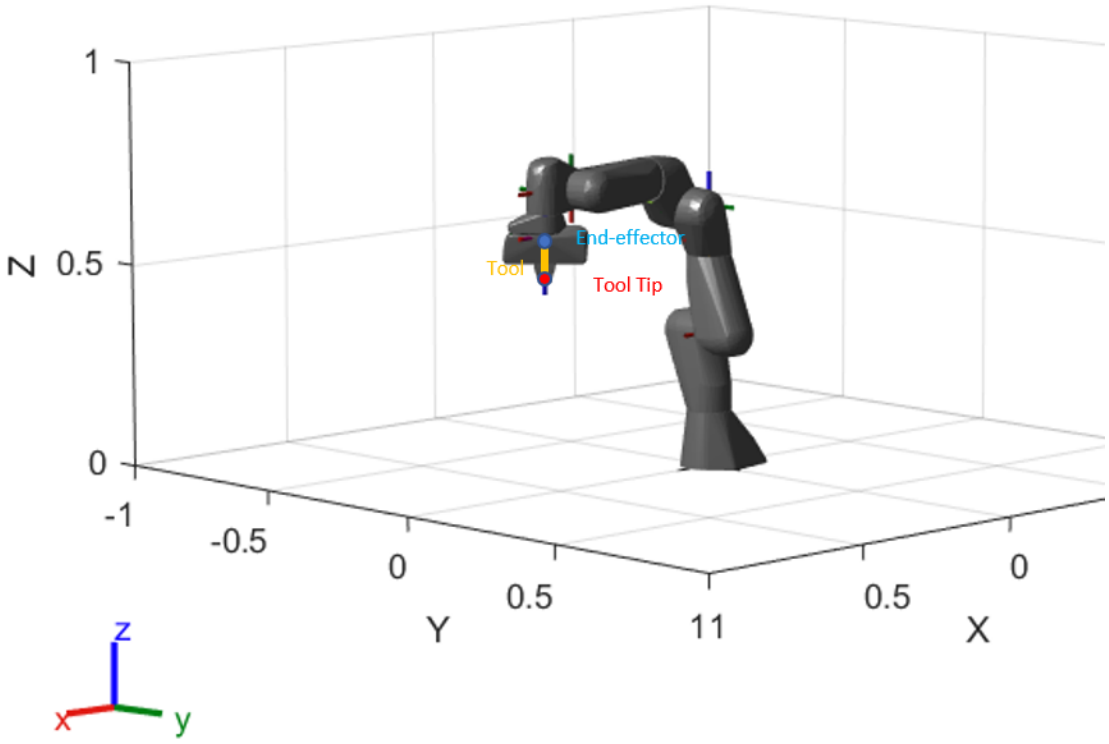


Figure 3.4: Initial Configuration

And the axis of the tool is $(0,0,-1)$. Therefore, when we want to minimize the change of the direction of the shaft, the axis of the tool at final configuration should also be $(0,0,-1)$.

3.2.1 Forward Reach

$$p_{goal} = \begin{bmatrix} 0.75 & 0 & 0.3 \end{bmatrix} \tag{3.3}$$

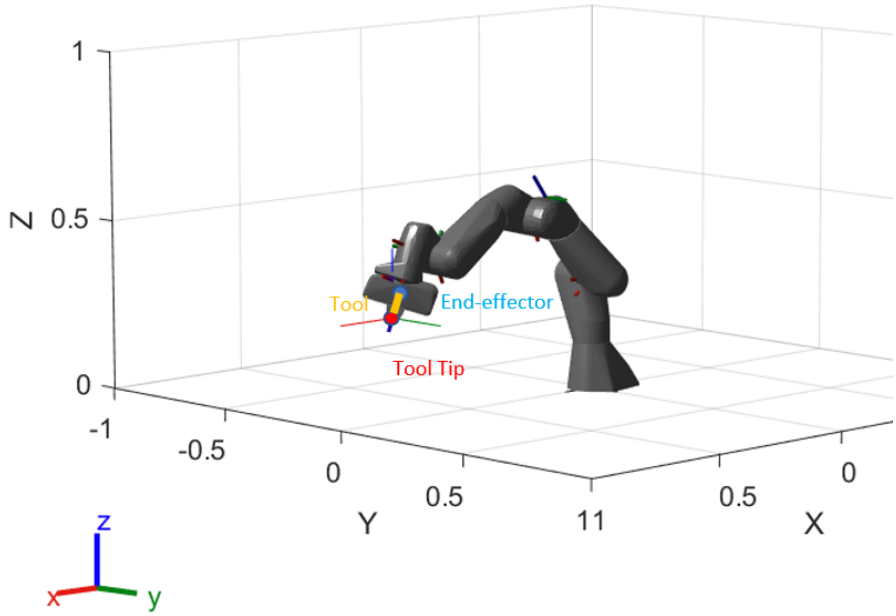


Figure 3.5: Forward Reach

The equation of the virtual wall is

$$z = 0.29$$

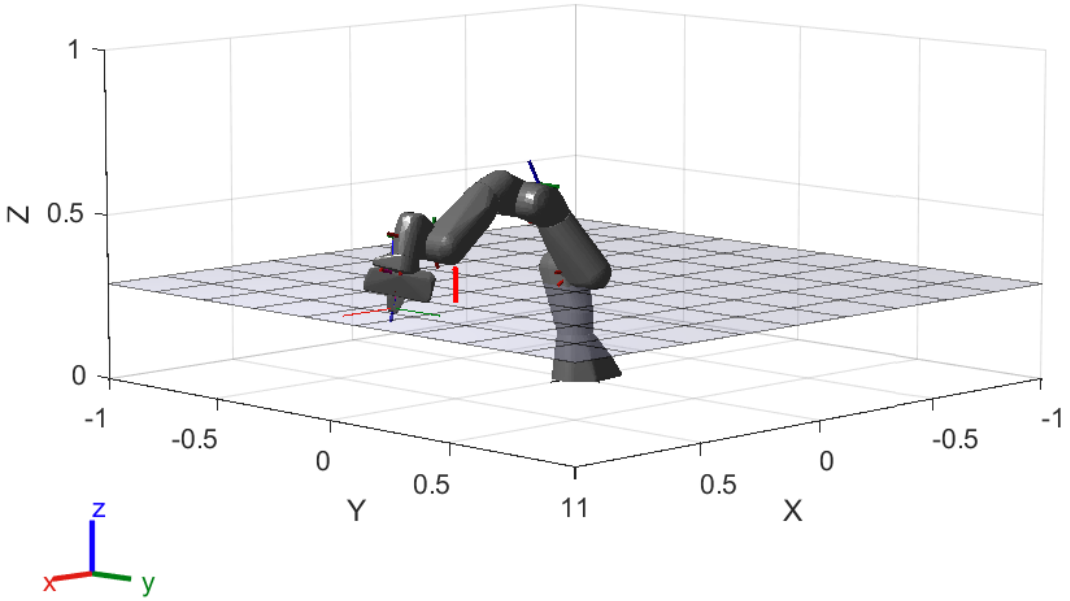


Figure 3.6: Forward Reach with Wall

3.2.2 Side Reach

$$p_{goal} = \begin{bmatrix} 0 & 0.5 & 0.3 \end{bmatrix} \tag{3.4}$$

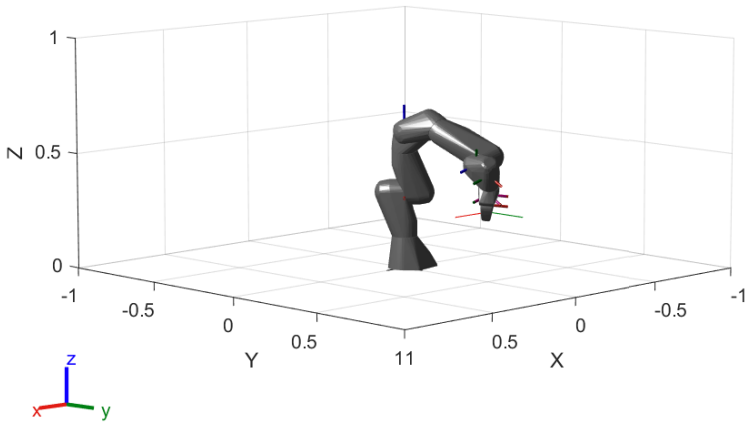


Figure 3.7: Side Reach

The equation of the virtual wall is

$$x - z = -0.5$$

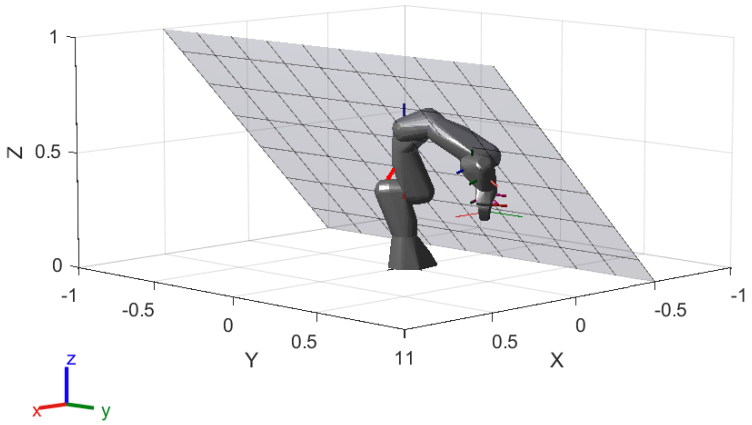


Figure 3.8: Side Reach with Wall

3.2.3 Backward Reach

$$p_{goal} = \begin{bmatrix} -0.4 & 0.3 & 0.6 \end{bmatrix} \tag{3.5}$$

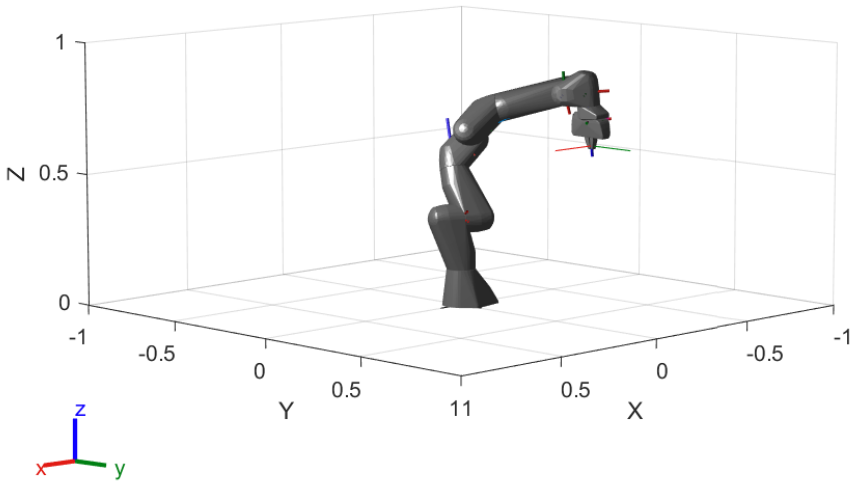


Figure 3.9: Backward Reach Configuration

The equation of the virtual wall is

$$\frac{1}{2}x + \frac{\sqrt{3}}{2}y = -0.1$$

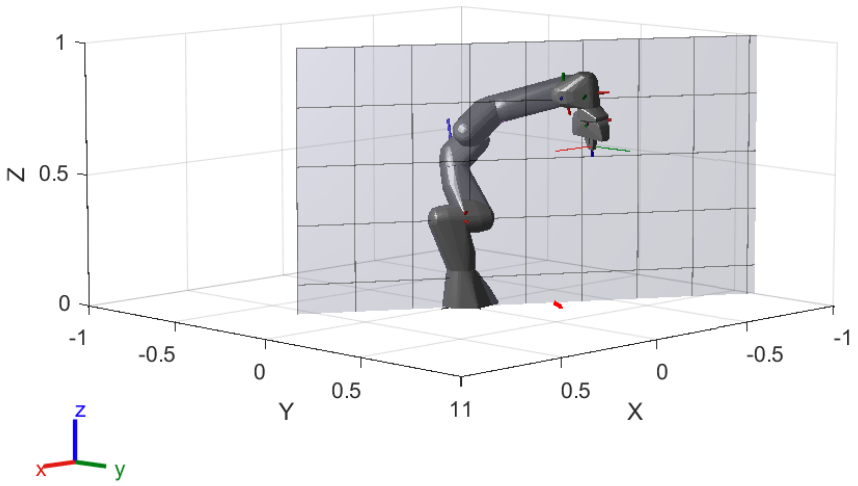


Figure 3.10: Backward Reach with Wall

3.3 Programming Assignments a)

Implement an algorithm that moves the tool tip toward an arbitrary goal point while

1. Keeps tool tip as close as possible to P_{goal} while never going beyond 3 mm from the goal
2. and obeying joint limits.

3.3.1 Algorithm

This is the optimization-based control problem that need to solve Lynch and Park, 2017

$$\begin{aligned} \Delta\mathbf{q}_{\text{des}} &= \arg\min_{\Delta\mathbf{q}} \|\mathbf{D}(\Delta\mathbf{x})\|^2 = \|\alpha \times \mathbf{t} + \epsilon + \mathbf{t} - \mathbf{p}_{\text{goal}}\|^2 \\ \alpha &= \mathbf{J}_\alpha(\mathbf{q})\Delta\mathbf{q} \\ \epsilon &= \mathbf{J}_\epsilon(\mathbf{q})\Delta\mathbf{q} \\ \|\alpha\times\mathbf{t} + \epsilon + \mathbf{t} - \mathbf{p}_{\text{goal}}\| &\leq 3 \\ \mathbf{q}_L - \mathbf{q} \leq \Delta\mathbf{q} &\leq \mathbf{q}_U - \mathbf{q} \end{aligned} \tag{3.6}$$

This algorithms is implemented in `opti_ik_space.m`

Algorithm 4 `opti_ik_space` Optimization-based Inverse Kinematics Algorithm in Space Frame

Inputs: M (Initial Configuration), S (Normalized Twists), p_{goal} (Destination), q_{init} (Initial Guess), qL (Joint Lower Limits), qU (Joint Upper Limits)
Initialize: Set $q \leftarrow q_{\text{init}}$, $\epsilon \leftarrow 0.003$, error $\leftarrow 1$.
while error $> \epsilon$ **do**
    Compute forward kinematics: $T_{\text{sb}} \leftarrow$ FK_space(M, S, q).
    Extract current position: $t \leftarrow T_{\text{sb}}(1:3, 4)$.
    Compute position error: error $\leftarrow \|t - p_{\text{des}}\|$.
    Compute space Jacobian: $J \leftarrow$ J_space(S, q).
    Extract submatrices: $J_\alpha \leftarrow J(1:3, :)$, $J_p \leftarrow J(4:6, :)$.
    Compute coefficient matrix: $A \leftarrow J_p - \text{vecToso3}(t) \cdot J_\alpha$.
    Compute residual: $b \leftarrow p_{\text{des}} - t$.
    Compute bounds: lb $\leftarrow qL - q$, ub $\leftarrow qU - q$.
    Solve constrained least-squares problem: $\Delta q \leftarrow \text{lsqlin}(A, b, \text{lb}, \text{ub})$.
    Update joint angles: $q \leftarrow q + \Delta q$.
end while
Return: q.

Note that:

1. The calculation of forward kinematics and Jacobian matrix are in space frame.

2. We used the `lsqlin` function in MATLAB to implement the optimization process.

3. Requirement $\|\alpha \times \mathbf{t} + \epsilon + \mathbf{t} - \mathbf{p}_{\text{goal}}\| \leq 3$ is fulfilled implicitly in this program by setting the threshold to be small enough (Here we set it to be 0.003 or 3 mm).

3.3.2 Test

Case 1: Forward Reach

In this case, five iterations are needed to reach the target

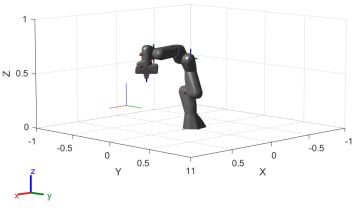


Figure 3.11: Initial Configuration

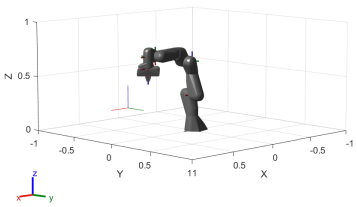


Figure 3.12: Iteration 1

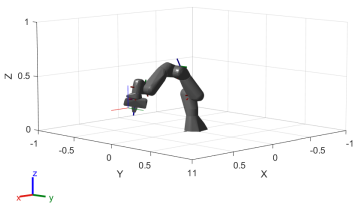


Figure 3.13: Iteration 2

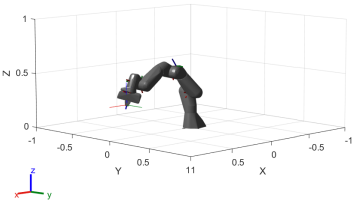


Figure 3.14: Iteration 3

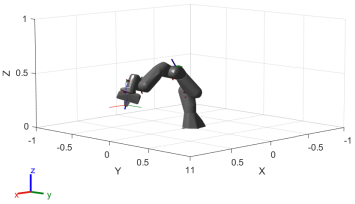


Figure 3.15: Iteration 4

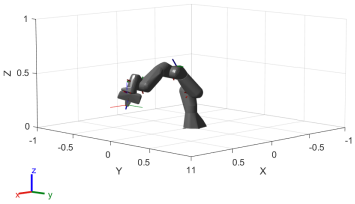


Figure 3.16: Iteration 5

The final position of the tool tip is $(0.750, 0, 0.300)$. And the final direction of the tool axis is $(0.262, 0.000, -0.965)$

Case 2: Side Reach

Here we omit the iteration process, just give the final result. Ten iterations are need to converge. The final position of the tool tip is $(0.000, 0.500, 0.300)$. And the final direction of the tool axis is $(-0.395, -0.398, -0.828)$

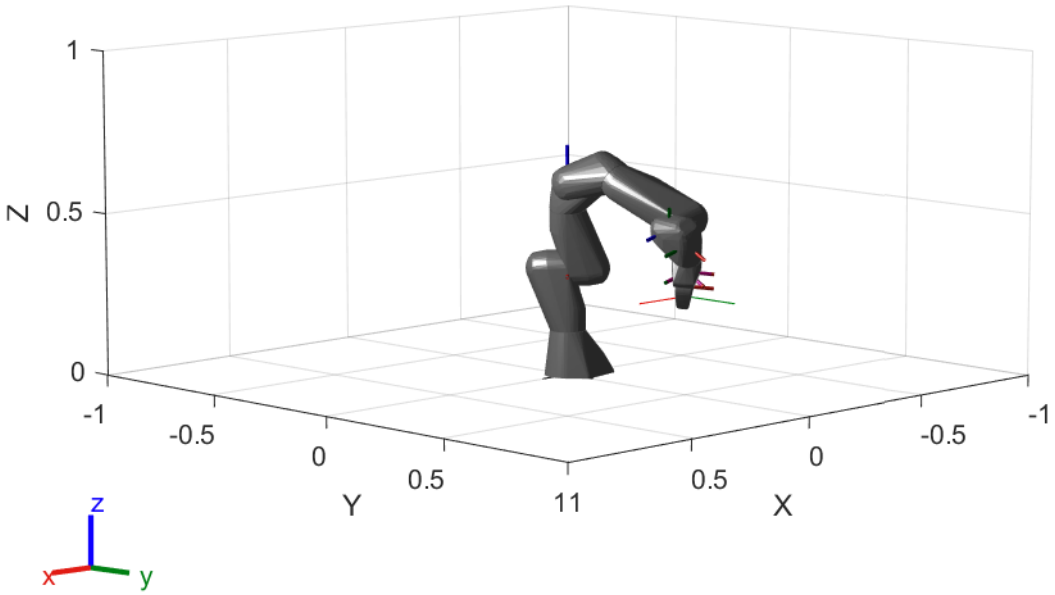


Figure 3.17: Final Configuration in the Side Reach Case

Case 3: Backward Reach

14 Iterations are needed to converge. The final position of the tool tip is $(-0.400, 0.300, 0.600)$. And the final direction of the tool axis is $(-0.068, 0.038, -0.997)$

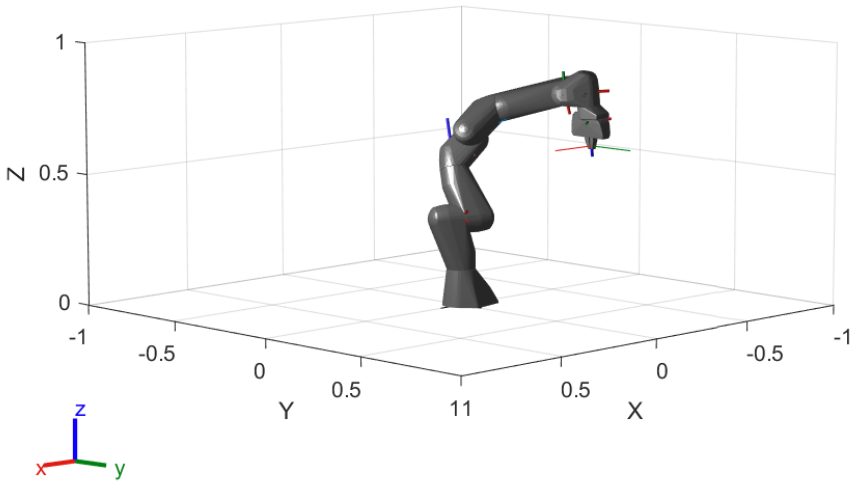


Figure 3.18: Final Configuration in the Backward Reach Case

3.4 Programming Assignments b)

Similar to a), implement an algorithm that moves the tool tip toward an arbitrary goal point while

1. Keeps tool tip as close as possible to P_{goal} while never going beyond 3 mm from the goal

2. Obeying joint limits.

3. Minimizes the change in direction of the tool shaft during motion

3.4.1 Algorithm

This is the optimization-based control problem that to be solved

$$\begin{aligned} \Delta\mathbf{q}_{\text{des}} &= \arg\min_{\Delta\mathbf{q}} \|\mathbf{D}(\Delta\mathbf{x})\|^2 = \|\alpha \times \mathbf{t} + \epsilon + \mathbf{t} - \mathbf{p}_{\text{goal}}\|^2 + \eta\|\alpha \times \mathbf{R}\cdot\mathbf{z}\|^2 \\ \alpha &= \mathbf{J}_\alpha(\mathbf{q})\Delta\mathbf{q} \\ \epsilon &= \mathbf{J}_\epsilon(\mathbf{q})\Delta\mathbf{q} \\ \|\alpha\times\mathbf{t} + \epsilon + \mathbf{t} - \mathbf{p}_{\text{goal}}\| &\leq 3 \\ \mathbf{q}_L - \mathbf{q} \leq \Delta\mathbf{q} &\leq \mathbf{q}_U - \mathbf{q} \end{aligned} \tag{3.7}$$

This algorithms is implemented in `opti_ik_angle_space.m`

Algorithm 5 `opti_ik_angle_space` Optimization-based Inverse Kinematics Algorithm while Minimizing Axis Angle Change in Space Frame

function OPTI_IK_ANGLE_SPACE($\mathbf{M}, \mathbf{S}, \mathbf{p}_{\text{goal}}, \mathbf{q}_{\text{init}}, \mathbf{q}_L, \mathbf{q}_U$)
  Inputs: M (Initial Configuration), S (Normalized Twists), p_{goal} (Destination), q_{init} (Initial Guess), qL (Joint Lower Limits), qU (Joint Upper Limits)
  Set constants: $\epsilon_1 = 0.003$, $\epsilon_2 = 0.01$, $\eta = 0.1$, $\mathbf{z}_s = [0, 0, 1]^T$.
  Initialize: $\mathbf{q} = \mathbf{q}_{\text{init}}$.
  Initialize errors: $\text{error}_1 = 1$, $\text{error}_2 = 1$.
  while $\text{error}_1 > \epsilon_1$ or $\text{error}_2 > \epsilon_2$ **do**
    Compute forward kinematics: $\mathbf{T}_{sb} = \text{FK\_space}(\mathbf{M}, \mathbf{S}, \mathbf{q})$.
    Extract current orientation: $\mathbf{R}_z = \mathbf{T}_{sb}(1:3, 1:3) \cdot \mathbf{z}_s$.
    Extract current position: $\mathbf{t} = \mathbf{T}_{sb}(1:3, 4)$.
    Compute errors:
      $\text{error}_1 = \|\mathbf{t} - \mathbf{p}_{\text{goal}}\|$.
      $\text{error}_2 = \|\mathbf{R}_z - \mathbf{R}_{\text{init}}\|$.
    Compute space Jacobian: $\mathbf{J} = \text{J\_space}(\mathbf{S}, \mathbf{q})$.
    Extract sub-Jacobians: $\mathbf{J}_\alpha = \mathbf{J}(1:3, :), \mathbf{J}_p = \mathbf{J}(4:6, :)$.
    Formulate least-squares system:
      $\mathbf{A}_1 = \mathbf{J}_p - \text{vecToso3}(\mathbf{t}) \cdot \mathbf{J}_\alpha$.
      $\mathbf{b}_1 = \mathbf{p}_{\text{goal}} - \mathbf{t}$.
      $\mathbf{A}_2 = \sqrt{\eta} \cdot \text{vecToso3}(\mathbf{R}_z) \cdot \mathbf{J}_\alpha$.
      $\mathbf{b}_2 = -\sqrt{\eta} \cdot (\mathbf{R}_{\text{init}} - \mathbf{R}_z)$.
    Define joint bounds: $\mathbf{lb} = \mathbf{q}_L - \mathbf{q}$, $\mathbf{ub} = \mathbf{q}_U - \mathbf{q}$.
    Solve constrained least-squares: $\Delta\mathbf{q}_{\text{dir}} = \text{lsqlin}([\mathbf{A}_1; \mathbf{A}_2], [\mathbf{b}_1; \mathbf{b}_2], [], [], [], [], \mathbf{lb}, \mathbf{ub})$.
    Line search to get best step length α
    for $i = 1$ to max_iter **do**
      Compute $q_{\text{candidate}} \leftarrow q + \alpha \cdot \delta q_{\text{dir}}$
      Compute $T_{\text{sb}}^{\text{candidate}} \leftarrow \text{FK\_space}(M, S, q_{\text{candidate}})$
      Extract $t_{\text{candidate}} \leftarrow T_{\text{sb}}^{\text{candidate}}(1:3, 4)$
      if $\|t_{\text{candidate}} - p_{\text{goal}}\|_2 > \|t - p_{\text{goal}}\|_2$ **then**
        Update $\alpha \leftarrow \alpha \cdot \beta$, where $\beta \in (0, 1)$
        Continue
      end if
    end for
    Update joint angles: $\mathbf{q} = \mathbf{q} + \alpha\Delta\mathbf{q}$.
  end while
  return q
end function

Note that:

1. The calculation of forward kinematics and Jacobian matrix are in space frame.
2. There are two weights, one for the distance to the target and one for the change in direction of the tool axis. But since this is an parameter optimization problem. What really matters is relative weight. Therefore, we just use one weight η (which is set to be 0.1) for the change in direction of the tool axis.
3. We used the `lsqlin` function in MATLAB to implement the optimization process.
4. Requirement $\|\alpha \times \mathbf{t} + \epsilon + \mathbf{t} - \mathbf{p}_{\text{goal}}\| \leq 3$ is fulfilled implicitly in this program by setting the threshold to be small enough.
5. We used two criterion (the distance to target and axis change) the judge the convergence.
6. We used line search method to get appropriate step length α. This could improve the program's robustness, because of the linearization and small angle approximation of the method Nocedal and Wright, 2006.

3.4.2 Test

Case 1: Forward Reach

In this case, five iterations are needed to reach the target

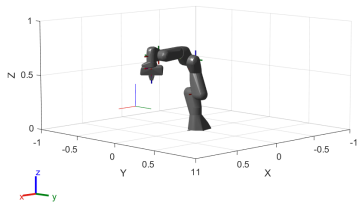


Figure 3.19: Initial Configuration

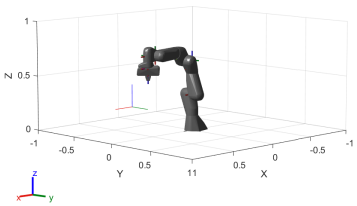


Figure 3.20: Iteration 1

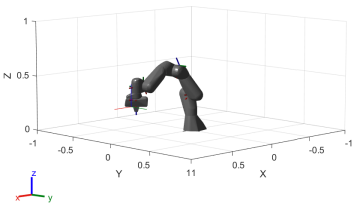


Figure 3.21: Iteration 2

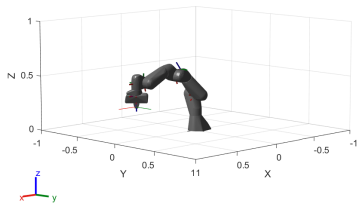


Figure 3.22: Iteration 3

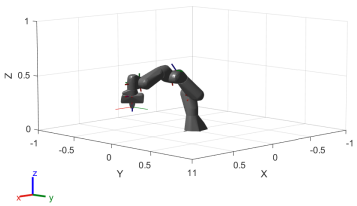


Figure 3.23: Iteration 4

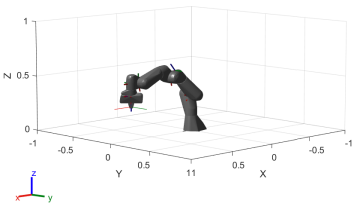


Figure 3.24: Iteration 5

The final position of the tool tip is $(0.750, 0.000, 0.300)$. And the final direction of the tool axis is $(-0.00, -0.000, -1.00)$.

Case 2: Side Reach

8 Iterations are need to converge. The final position of the tool tip is $(0.000, 0.500, 0.300)$. And the final direction of the tool axis is $(-0.000, -0.000, -1.000)$

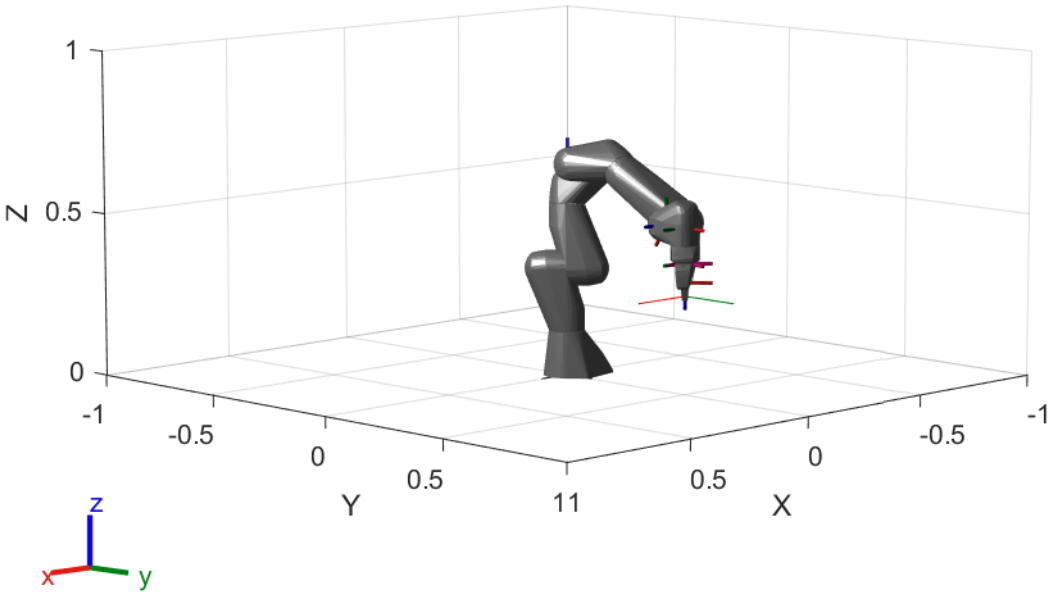


Figure 3.25: Final Configuration in the Side Reach Case

Case 3: Backward Reach

8 Iterations are need to converge. The final position of the tool tip is $(-0.400, 0.300, 0.600)$. And the final direction of the tool axis is $(-0.000, 0.000, -1.000)$

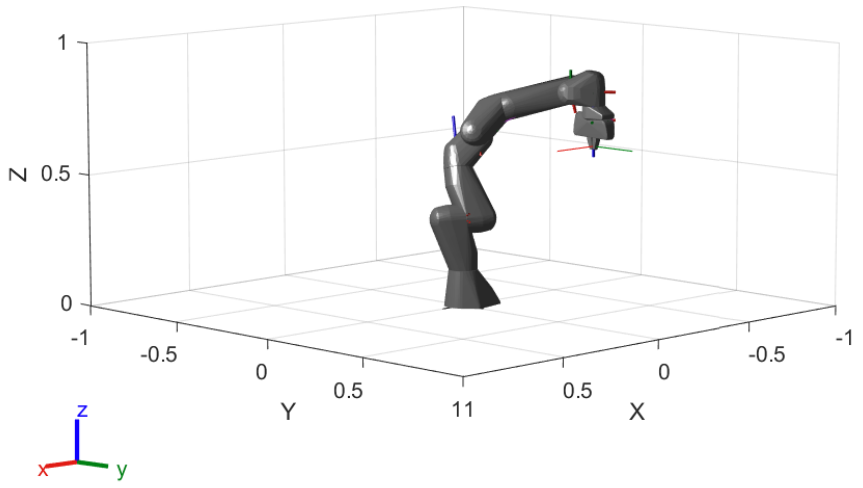


Figure 3.26: Final Configuration in the Backward Reach Case

3.5 Programming Assignments c)

Repeat part (a) and (b) while having a **virtual wall** defined close to the desired goal point.

3.5.1 Algorithm

This is the optimization-based control problem that need to be solved with a virtual wall but without minimizing the change of tool shaft.

$$\begin{aligned}
\Delta\mathbf{q}_{\text{des}} &= \arg\min_{\Delta\mathbf{q}} \|\mathbf{D}(\Delta\mathbf{x})\|^2 = \|\alpha \times \mathbf{t} + \epsilon + \mathbf{t} - \mathbf{p}_{\text{goal}}\|^2 \\
\alpha &= \mathbf{J}_\alpha(\mathbf{q})\Delta\mathbf{q} \\
\epsilon &= \mathbf{J}_\epsilon(\mathbf{q})\Delta\mathbf{q} \\
\mathbf{n}_{\text{wall}} \cdot (\alpha \times \mathbf{t} + \epsilon + \mathbf{t} - \mathbf{p}_{\text{wall}}) &\geq 0 \\
\|\alpha \times \mathbf{t} + \epsilon + \mathbf{t} - \mathbf{p}_{\text{goal}}\| &\leq 3 \\
\mathbf{q}_L - \mathbf{q} \leq \Delta\mathbf{q} &\leq \mathbf{q}_U - \mathbf{q}
\end{aligned} \tag{3.8}$$

And this is the optimization-based control problem with a virtual wall while minimizing the change of tool shaft.

$$\begin{aligned}
\Delta\mathbf{q}_{\text{des}} &= \arg\min_{\Delta\mathbf{q}} \|\mathbf{D}(\Delta\mathbf{x})\|^2 = \|\alpha \times \mathbf{t} + \epsilon + \mathbf{t} - \mathbf{p}_{\text{goal}}\|^2 + \eta\|\alpha \times \mathbf{R} \cdot \mathbf{z}\|^2 \\
\alpha &= \mathbf{J}_\alpha(\mathbf{q})\Delta\mathbf{q} \\
\epsilon &= \mathbf{J}_\epsilon(\mathbf{q})\Delta\mathbf{q} \\
\mathbf{n}_{\text{wall}} \cdot (\alpha \times \mathbf{t} + \epsilon + \mathbf{t} - \mathbf{p}_{\text{wall}}) &\geq 0 \\
\|\alpha \times \mathbf{t} + \epsilon + \mathbf{t} - \mathbf{p}_{\text{goal}}\| &\leq 3 \\
\mathbf{q}_L - \mathbf{q} \leq \Delta\mathbf{q} &\leq \mathbf{q}_U - \mathbf{q}
\end{aligned} \tag{3.9}$$

where $\mathbf{n}_{\text{wall}}$ is the normal vector of the virtual wall, while $\mathbf{p}_{\text{goal}}$ is a point on the virtual wall.

The above two algorithms only differs from their corresponding part (a) and part (b) in adding an additional inequality constraint. So, we omit their algorithms here.

3.5.2 Test of 3.8 without Minimizing the Change in Direction of the Tool Shaft

Case 1: Forward Reach

In this case, five iterations are needed to reach the target

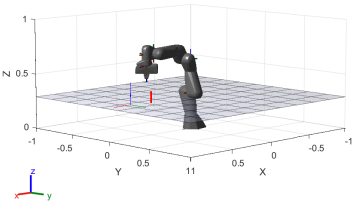


Figure 3.27: Initial Configuration

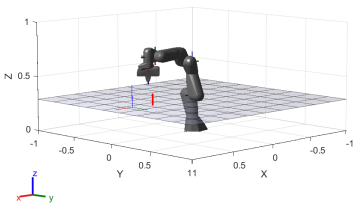


Figure 3.28: Iteration 1

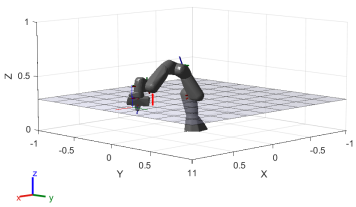


Figure 3.29: Iteration 2

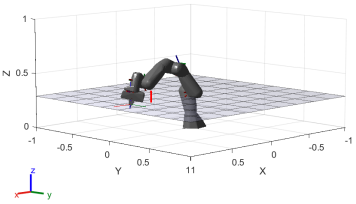


Figure 3.30: Iteration 3

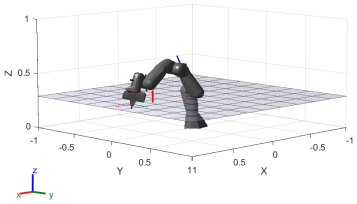


Figure 3.31: Iteration 4

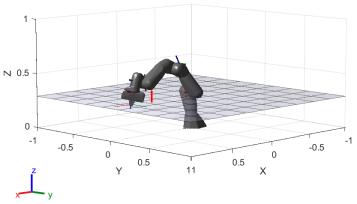


Figure 3.32: Iteration 5

The final position of the tool tip is $(0.750, 0.000, 0.300)$. And the final direction of the tool axis is $(0.287, -0.000, -0.958)$.

Case 2: Side Reach

8 iterations are needed to reach convergence. The final position of the tool tip is $(0.000, 0.500, 0.300)$. And the final direction of the tool axis is $(-0.452, -0.345, -0.823)$

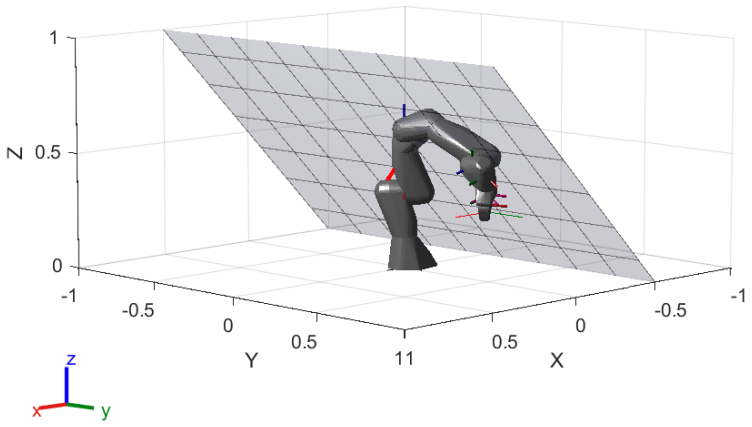


Figure 3.33: Final Configuration in the Side Reach Case

Case 3: Backward Reach

11 iterations are needed to reach convergence. The final position of the tool tip is $(-0.400, 0.300, 0.600)$. And the final direction of the tool axis is $(-0.129, -0.0155, -0.991)$

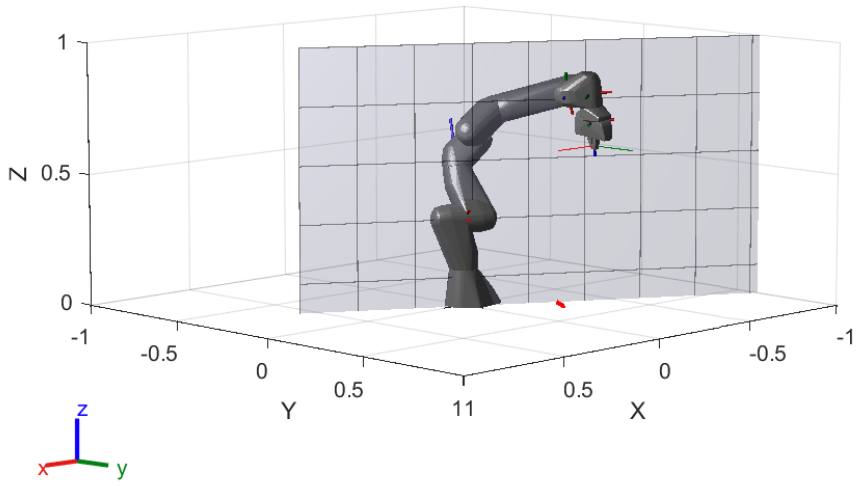


Figure 3.34: Final Configuration in the Backward Reach Case

3.5.3 Test of 3.9 while Minimizing the Change in Direction of the Tool Shaft

Case 1: Forward Reach

In this case, six iterations are needed to reach the target (first five iterations are plotted here)

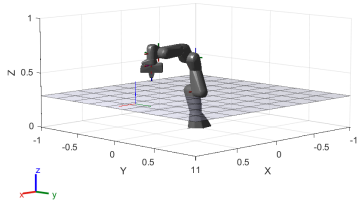


Figure 3.35: Initial Configuration

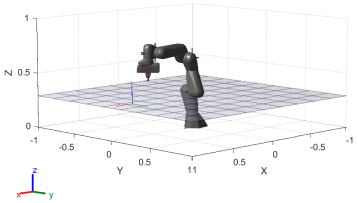


Figure 3.36: Iteration 1

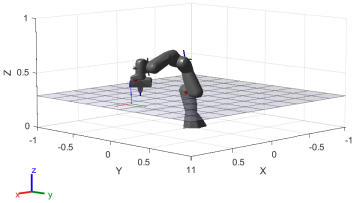


Figure 3.37: Iteration 2

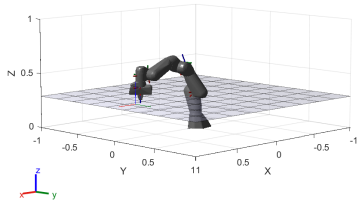


Figure 3.38: Iteration 3

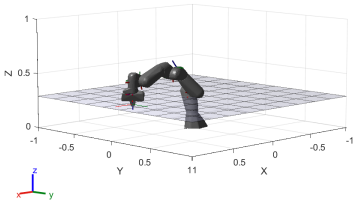


Figure 3.39: Iteration 4

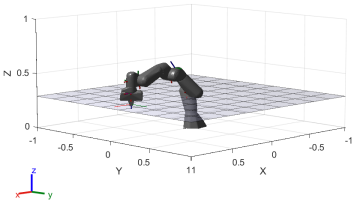


Figure 3.40: Iteration 5

Case 2: Side Reach

7 iterations are needed to reach convergence. The final position of the tool tip is $(0.000, 0.500, 0.300)$. And the final direction of the tool axis is $(-0.000, -0.000, -1.000)$

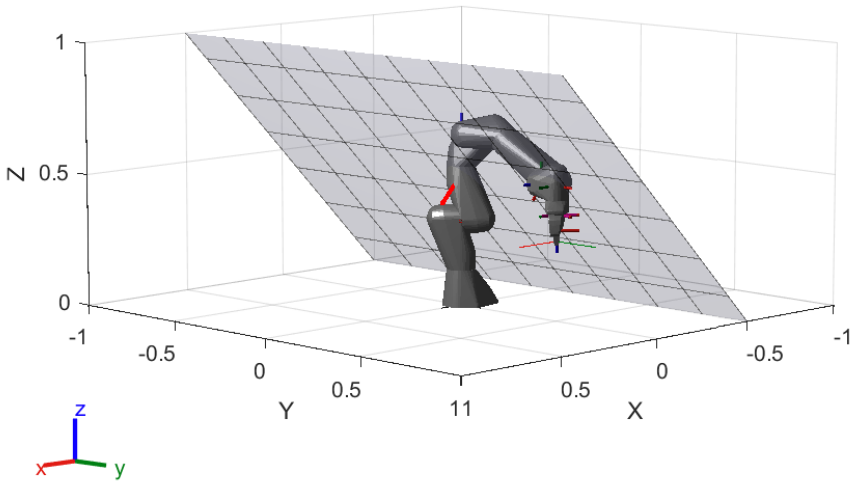


Figure 3.41: Final Configuration in the Side Reach Case

Case 3: Backward Reach

10 iterations are needed to reach convergence. The final position of the tool tip is $(-0.400, 0.300, 0.600)$. And the final direction of the tool axis is $(-0.000, -0.000, -1.000)$

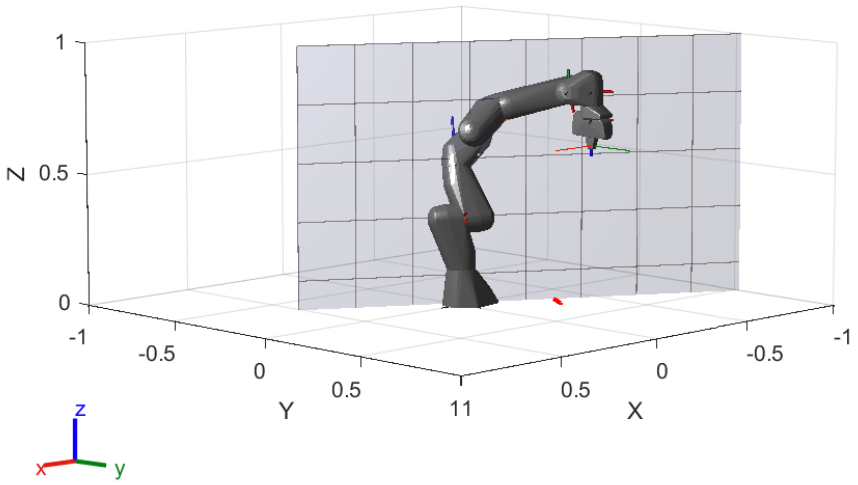


Figure 3.42: Final Configuration in the Backward Reach Case

3.6 Comparison

The comparison is implemented in `compare.m`. For the visualization, we separately created corresponding files `xxx_visualize.m` for all the four algorithms mentioned earlier, which basically completed the same task as their original version but output additional three variables: distance to the goal, change of tool shaft direction, and distance to the wall.

And we adopt the following notations:

Table 3.3: Notations

notation	meaning
ik	approximating a point in part (a)
angle	approximating a point while minimizing axis angle change in part (b)
wall	approximating a point with a wall in part (c)
angle_wall	approximating a point with a wall while minimizing axis angle change in part (c)

3.6.1 Case 1: Forward Reach

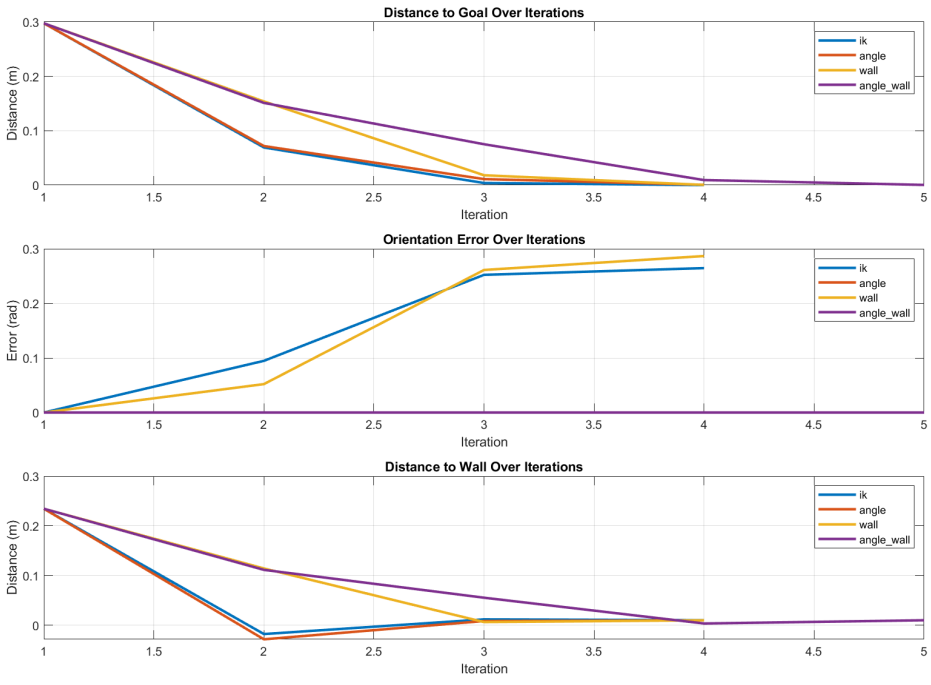


Figure 3.43: Comparison of Methods in the Forward Reach Case

Case 2: Side Reach

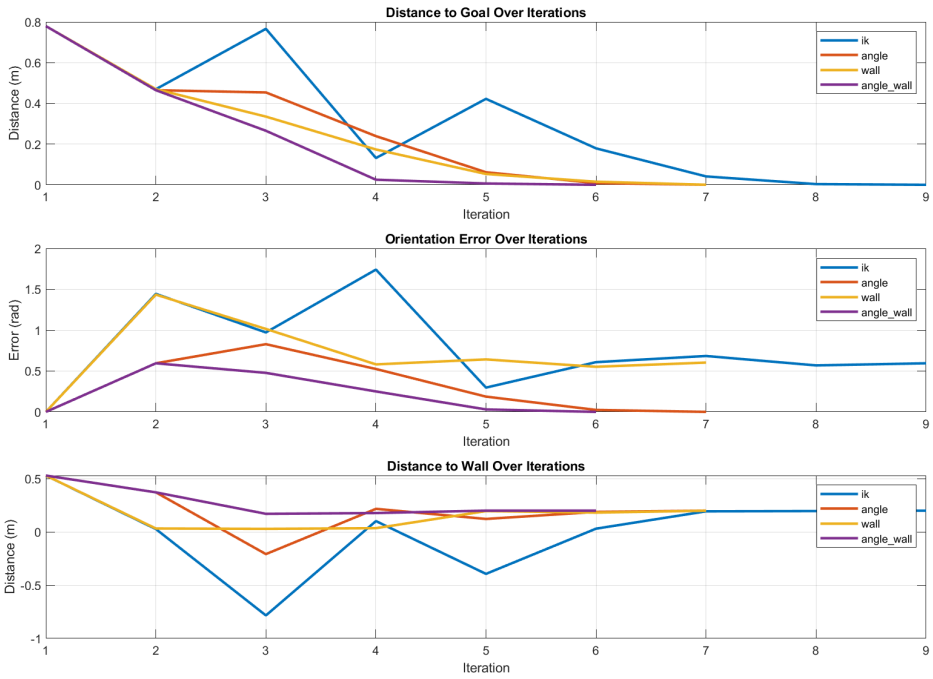


Figure 3.44: Final Configuration in the Side Reach Case

Case 3: Backward Reach

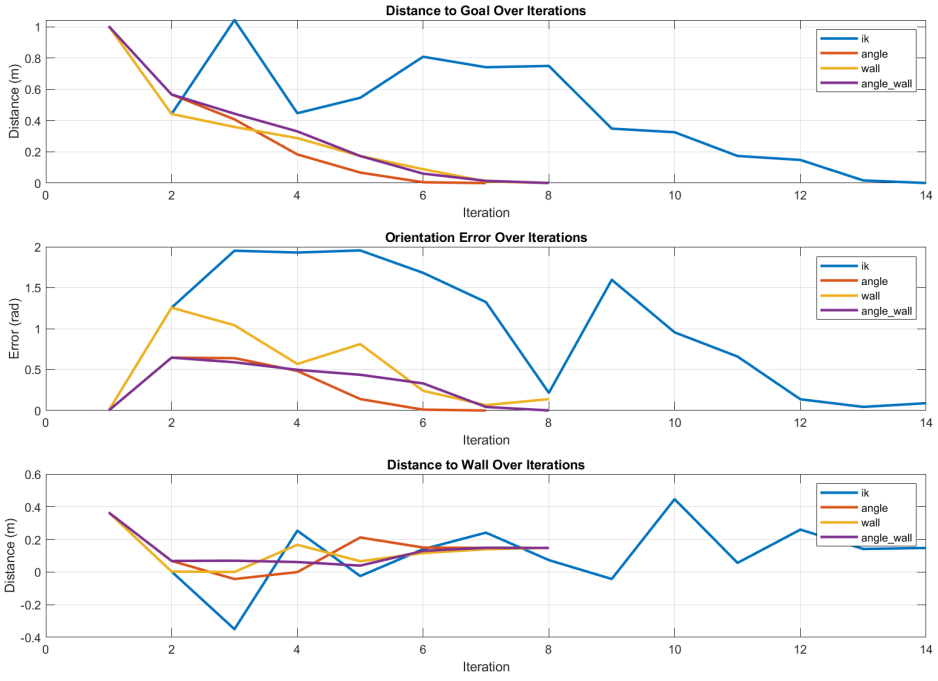


Figure 3.45: Comparison of Methods in the Backward Reach Case

3.7 Summary

1. All the four methods successfully reached the goal points in all the three cases (shown in the upper subplots).

2. For the three cases, the third case (Backward Reach) requires the most number of iterations because it is the hardest to reach.

3. Of all the four methods, the vanilla inverse kinematics method (denoted by "ik") requires the most number of iterations because it does use line search method. Since we are using Jacobian matrix and small angle rotation transformation which are only applicable for small joint angle change, it is not doing well for large transformation, especially for side reach and backward reach cases.

4. The methods involving minimizing the change of tool axis direction (denoted by "angle" and "angle_wall") successfully reached their mission, reaching 0 in the change of tool shaft (or orientation error shown in the middle subplots).

5. The methods virtual wall constraints successfully kept the distance to the wall greater than 0 (shown in the lower subplots).

Bibliography

Buss, S. (2004). Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. *IEEE Transactions in Robotics and Automation*, *17*.

GmbH, F. E. (2017). Robot and interface specifications: Franka Control Interface (FCI) documentation [Accessed: Mar. 23, 2025].

He, Y., & Liu, S. (2021). Analytical inverse kinematics for franka emika panda – a geometrical solver for 7-dof manipulators with unconventional design. *2021 9th International Conference on Control, Mechatronics and Automation (ICCMA)*, 194–199. https://doi.org/10.1109/ICCMA54375.2021.9646185

Horn, B. (1987). Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, *4*, 629–642. https://doi.org/10.1364/JOSAA.4.000629

Lynch, K. M., & Park, F. C. (2017). *Modern robotics: Mechanics, planning, and control*. Cambridge University Press.

Nocedal, J., & Wright, S. J. (2006). *Numerical optimization* (2nd). Springer.

Tittel, S. (2021). Analytical solution for the inverse kinematics problem of the franka emika panda seven-dof light-weight robot arm. *2021 20th International Conference on Advanced Robotics (ICAR)*, 1042–1047. https://doi.org/10.1109/ICAR53236.2021.9659393