Robotics Library: Forward and Inverse Kinematics

Xu Duan & Benito Ribadeneira

March 23, 2025

1 Introduction to Franka Emika Panda

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

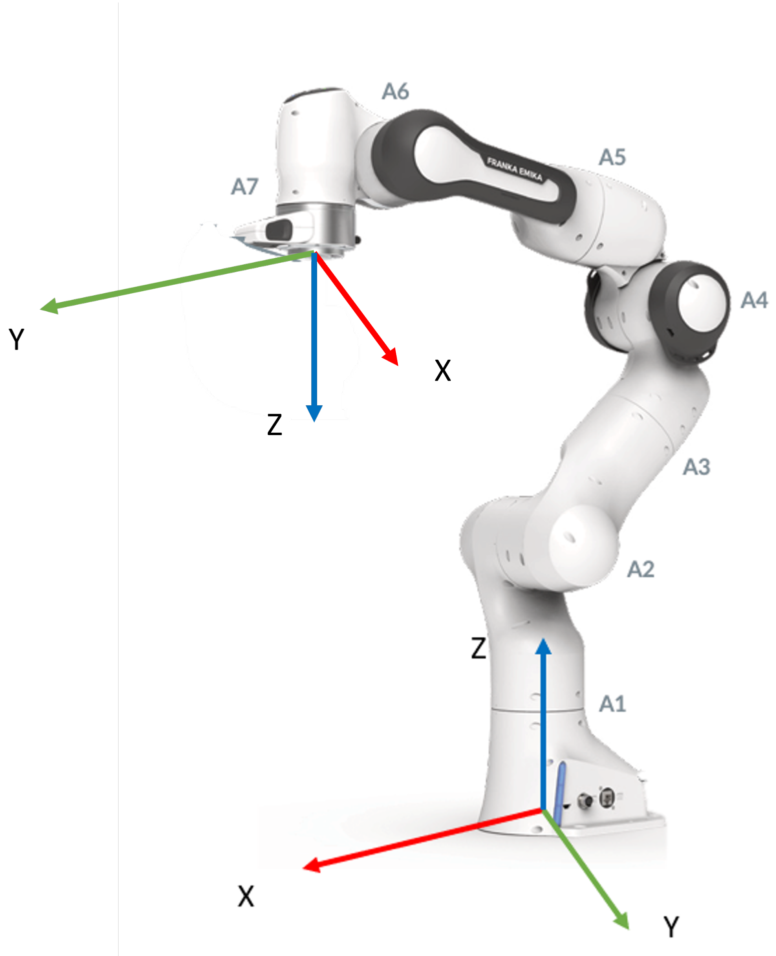


Figure 1: The arms and joints of the Panda robot

Here are the dimensions of the robot.

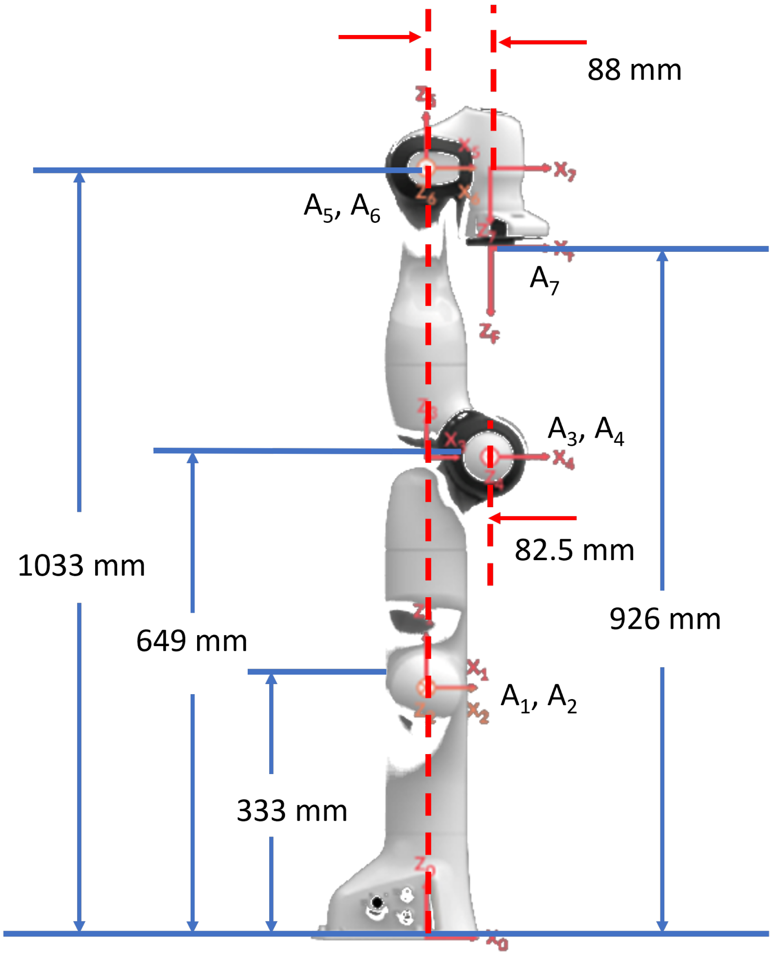


Figure 2: The parameters of Panda

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

2 Programming Assignments

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

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

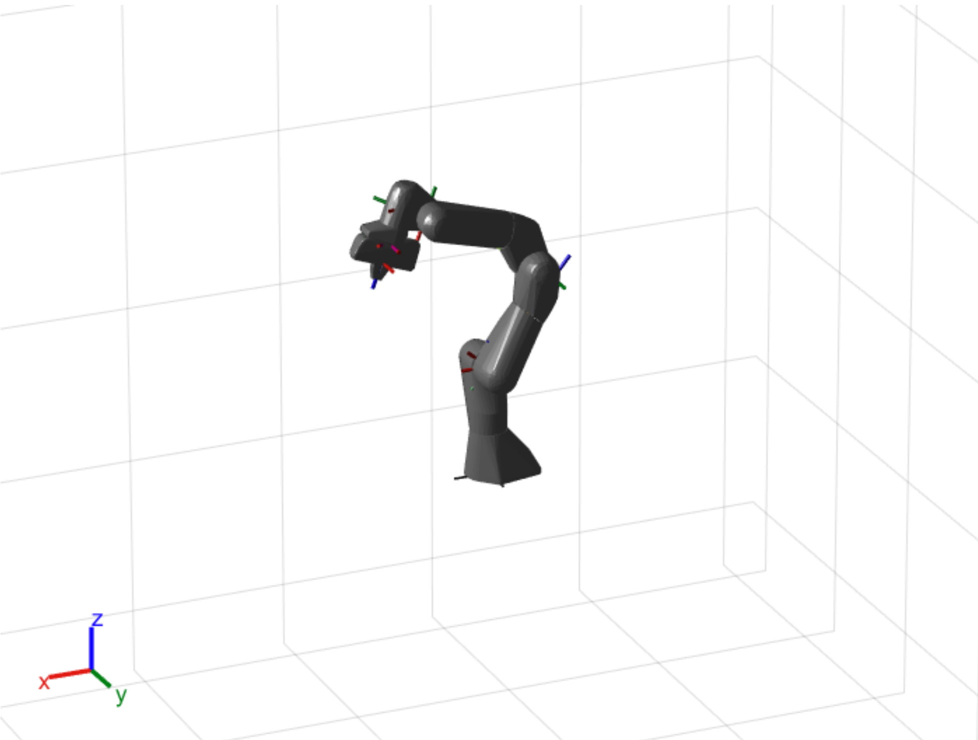
We set $\theta = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$. In this case, the configuration should be the initial configuration and it is.

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. 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.

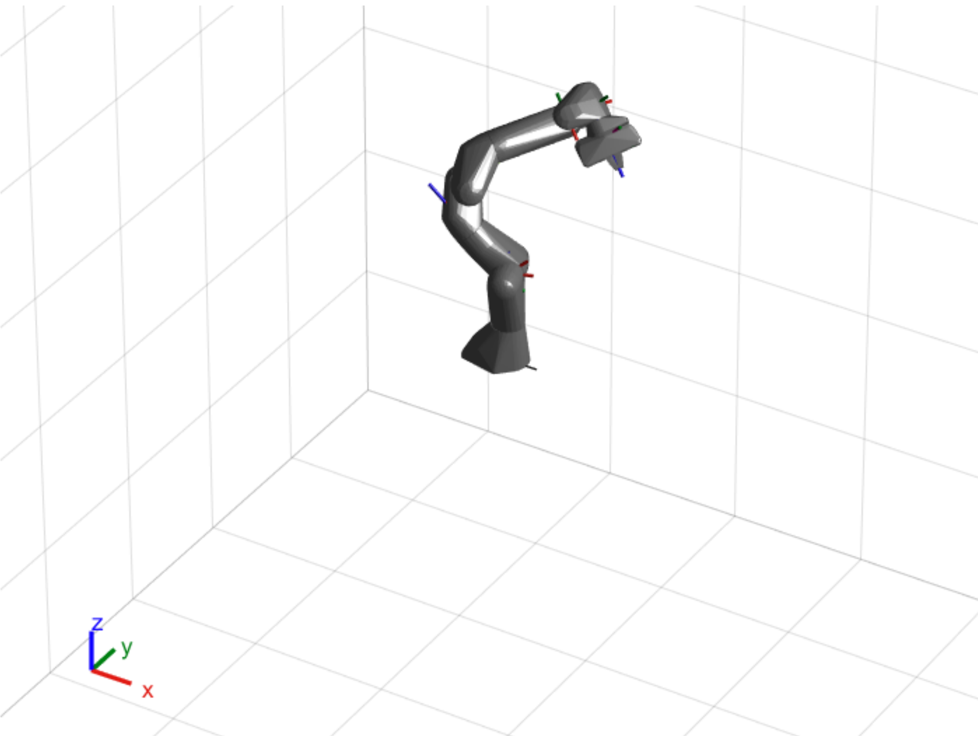


Case 3

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

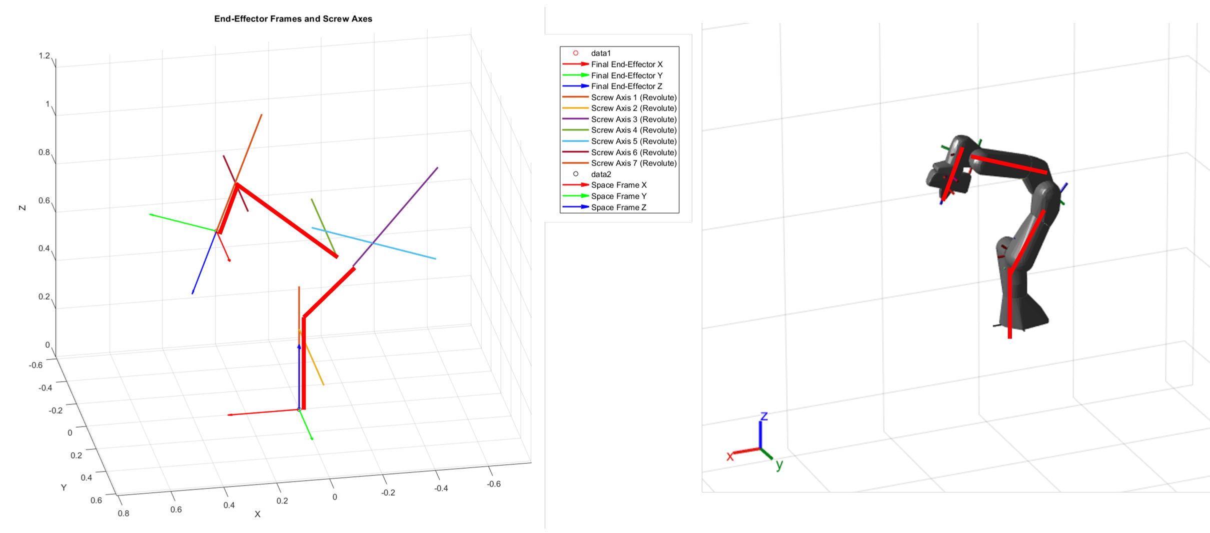
This is implemented in **visualize_FK_space.m** (instead of **FK_space.m**, because **FK_space.m** will be used later for many times to calculate the configuration of the end-effector and we think it is annoyting to plot the graph every time we call the function to get the end-effector).

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

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

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

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

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.

$$J(\theta)\Delta\theta = \mathrm{Ad}(\Delta T) \tag{1}$$

or

$$J(\theta)\Delta\theta = \mathrm{Ad}_{T(\theta+\Delta\theta)/T(\theta)} \tag{2}$$

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

$$\mathrm{Ad}_{T(\theta+\Delta\theta)/T(\theta)} = \begin{bmatrix}0.0106 & -0.0154 & 0.0185 & 0.0175 & 0.0172 & -0.0009\end{bmatrix}^T \tag{3}$$

And

$$J(\theta)\Delta\theta = \begin{bmatrix}0.0100 & -0.0157 & 0.0184 & 0.0178 & 0.0167 & -0.0008\end{bmatrix}^T \tag{4}$$

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.

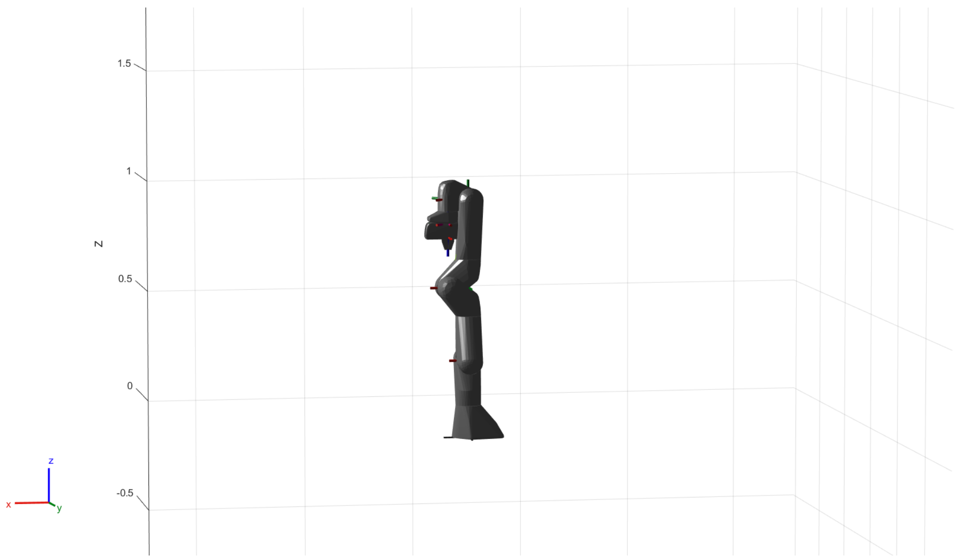


Figure 3: Singularity case

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.

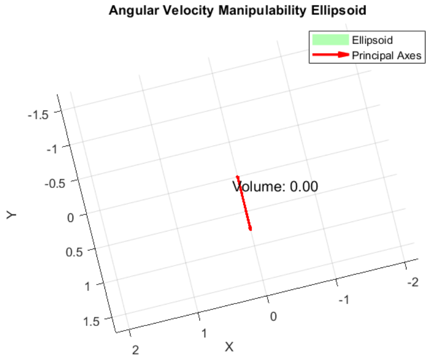


Figure 4: Angular Velocity Manipulability Ellipsoid

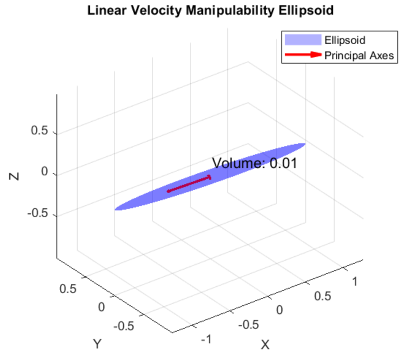


Figure 5: Linear Velocity Manipulability Ellipsoid

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.

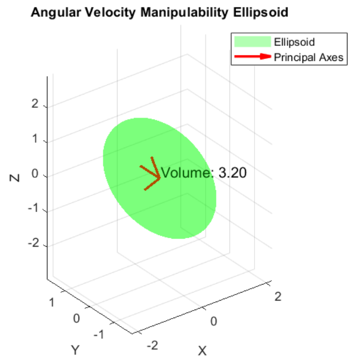


Figure 6: Angular Velocity Manipulability Ellipsoid

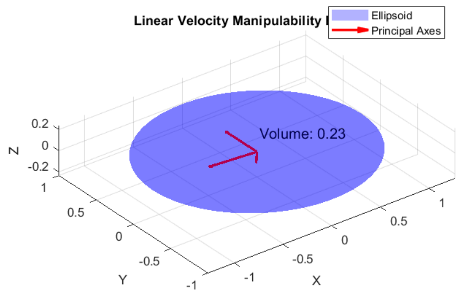


Figure 7: Linear Velocity Manipulability Ellipsoid

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: 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

$$\begin{aligned} error &= \mathrm{Ad}_{T_{sb}}[T_{sb}\, T_{sd}] \\ \theta &= \theta + J^{\dagger} error \end{aligned} \tag{5}$$

The declaration of the function is as follows:

```matlab
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)

$$T_{sd} = \begin{bmatrix} 0.3862 & -0.2690 & -0.8823 & 0.4225 \\ 0.8917 & 0.3535 & 0.2826 & 0.3776 \\ 0.2359 & -0.8959 & 0.3764 & -0.0874 \\ 0 & 0 & 0 & 1.00 \end{bmatrix} \tag{6}$$

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.

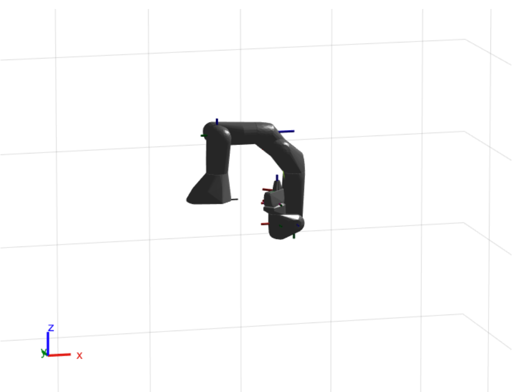


Figure 8: Initial Configuration

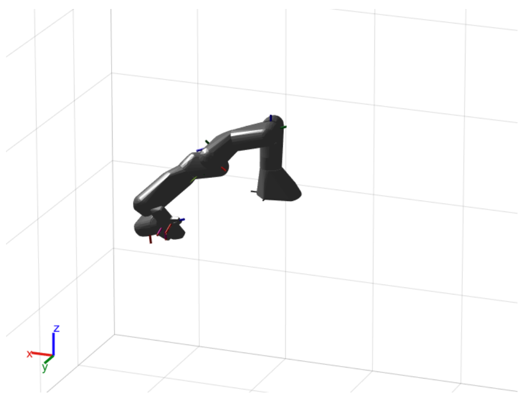


Figure 9: Desired Configuration

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

$$T_{sd} = \begin{bmatrix} -0.0198 & 0.9980 & -0.0601 & 0.1339 \\ 0.9998 & 0.0199 & 0.0012 & 0.0097 \\ 0.0024 & -0.0601 & -0.9982 & 0.9263 \\ 0 & 0 & 0 & 1.00 \end{bmatrix} \tag{7}$$

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 10: Initial Configuration

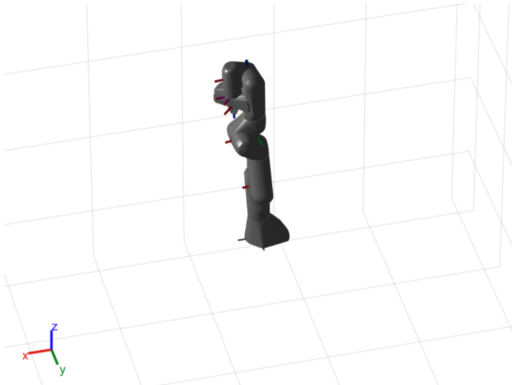


Figure 11: Desired Configuration

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

$$e = \mathrm{Ad}_{T_{sb}}[T_{sb}\ T_{sd}] \tag{8}$$

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

$$\alpha = \frac{< e, JJ^T e >}{< JJ^T e, JJ^T e >} \tag{9}$$

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}

$$T_{sd} = \begin{bmatrix} 0.3862 & -0.2690 & -0.8823 & 0.4225 \\ 0.8917 & 0.3535 & 0.2826 & 0.3776 \\ 0.2359 & -0.8959 & 0.3764 & -0.0874 \\ 0 & 0 & 0 & 1.00 \end{bmatrix} \tag{10}$$

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

$$T_{sd} = \begin{bmatrix} -0.0198 & 0.9980 & -0.0601 & 0.1339 \\ 0.9998 & 0.0199 & 0.0012 & 0.0097 \\ 0.0024 & -0.0601 & -0.9982 & 0.9263 \\ 0 & 0 & 0 & 1.00 \end{bmatrix} \tag{11}$$

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.

$$\theta = \theta + J^{\dagger}e + \alpha N \nabla V \tag{12}$$

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

$$N = I - J^{\dagger}J \tag{13}$$

Here, we choose $\alpha = 0.1$

Case 1

In this case, we set T_{sd}

$$T_{sd} = \begin{bmatrix} 0.3862 & -0.2690 & -0.8823 & 0.4225 \\ 0.8917 & 0.3535 & 0.2826 & 0.3776 \\ 0.2359 & -0.8959 & 0.3764 & -0.0874 \\ 0 & 0 & 0 & 1.00 \end{bmatrix} \tag{14}$$

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

$$T_{sd} = \begin{bmatrix} -0.0198 & 0.9980 & -0.0601 & 0.1339 \\ 0.9998 & 0.0199 & 0.0012 & 0.0097 \\ 0.0024 & -0.0601 & -0.9982 & 0.9263 \\ 0 & 0 & 0 & 1.00 \end{bmatrix} \tag{15}$$

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

$$\theta = \theta + J^T(JJ^T + k^2 I)e \tag{16}$$

Case 1

In this case, we set T_{sd}

$$T_{sd} = \begin{bmatrix} 0.3862 & -0.2690 & -0.8823 & 0.4225 \\ 0.8917 & 0.3535 & 0.2826 & 0.3776 \\ 0.2359 & -0.8959 & 0.3764 & -0.0874 \\ 0 & 0 & 0 & 1.00 \end{bmatrix} \tag{17}$$

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

$$T_{sd} = \begin{bmatrix} -0.0198 & 0.9980 & -0.0601 & 0.1339 \\ 0.9998 & 0.0199 & 0.0012 & 0.0097 \\ 0.0024 & -0.0601 & -0.9982 & 0.9263 \\ 0 & 0 & 0 & 1.00 \end{bmatrix} \tag{18}$$

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**

References

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

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

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