ME 384R ASBR: THA2 - Programming Assignment Report

Daniyal Maroufi and Anas Yousaf

March 25, 2025

1 Introduction

For this programming project, we were tasked with developing MATLAB functions that allowed us to study the kinematics of robotic manipulators. Although the developed functions can be applied broadly to any serial open-chain manipulator, we specifically analyzed our functions in relation to the KUKA LBR iiwa 14 robot shown in Figure 1. Some of the specific concepts explored in this project include:

- Forward Kinematics
- Jacobian Calculation
- Singularity Analysis
- Manipulability Analysis
- Inverse Kinematics

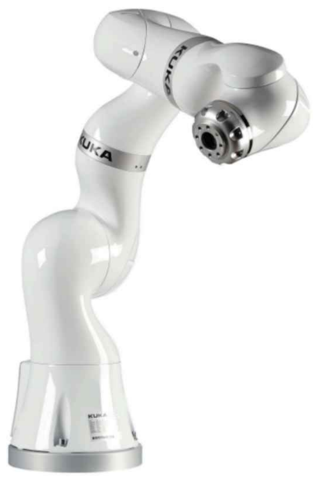

Figure 1: The KUKA LBR iiwa 14 robot in its home configuration [1].

2 Kuka LBR iiwa 14 R820 Specification

This section will detail the technical and geometric specifications of the KUKA LBR iiwa 14 robotic arm. These specifications are repeatedly referenced in the report and used to validate the mathematical model used to solve the forward and inverse kinematics formulations. For brevity, we will refer to the KUKA LBR iiwa 14 R820 as the *iiwa*. The basic details of the iiwa manipulator are shown in Table 1.

Table 1: Basic data of KUKA LBR iiwa 14 R820 manipulator.

Specification	**Value**
Number of Axes	7
Number of Controlled Axes	7
Volume of Working Envelope	1.8 m^3
Pose Repeatability (ISO 9283)	$\pm$ 0.15 mm
Weight	$\approx$ 29.9 kg
Rated Payload	14 kg
Maximum Reach	820 mm

2.1 Joint Configuration

The iiwa is classified as a lightweight, 7-axis jointed arm robot. The iiwa has an RRRRRRR joint configuration, meaning it has 7 revolute joints with no prismatic joints as seen in Figure 2. Each axis contains multiple sensors to allow the iiwa to be controlled and are used as protective functions for the robot. For this project, we operated under the assumption that the robot operates in ideal conditions and within the functions constraints in order to focus on the kinematics of the robot that allow the end-effector to reach a desired position and orientation.

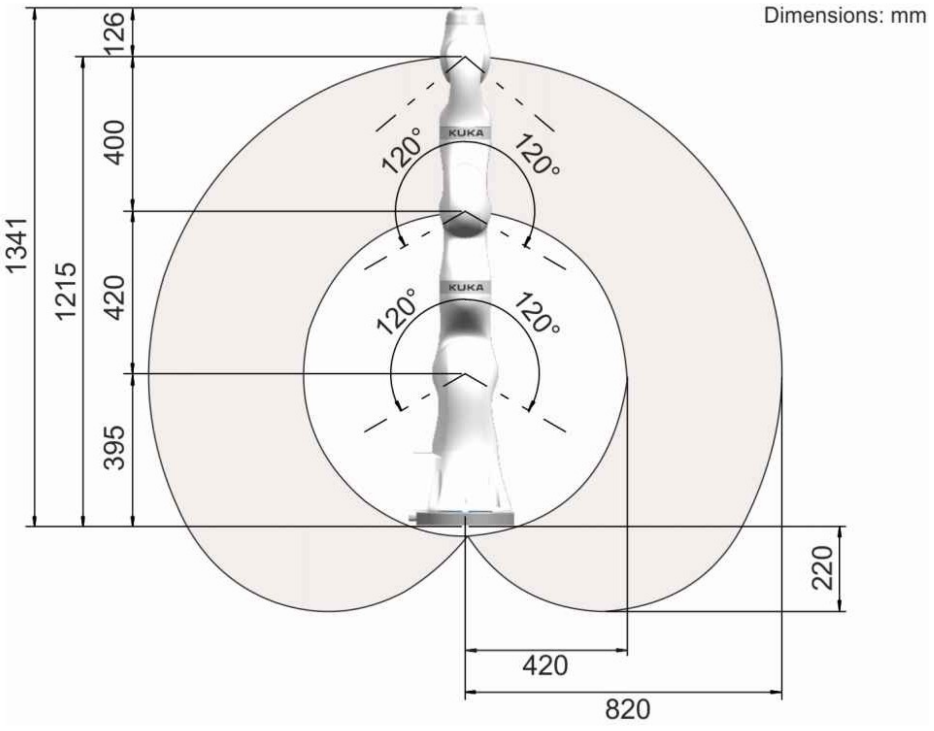

Figure 2: The workspace of the KUKA LBR iiwa 14 robot [1].

2.2 Joint Constraints

The 7 revolute joints of the iiwa are not free to move in 360°, as there are angular constraints imposed on each joint by the manufacturer. Table 2 shows the allowable range of each joint with respect to the zero position of the robot seen in Figure 2.

Table 2: The constrained range of motion for each joint of the KUKA LBR iiwa 14 robot.

Range of motion in degrees (°)						
Joint 1	**Joint 2**	**Joint 3**	**Joint 4**	**Joint 5**	**Joint 6**	**Joint 7**
±170	±120	±170	±120	±170	±120	±175

2.3 Kinematic Configuration

Each of the 7 joints of the KUKA LBR iiwa 14 robot was modeled using screw theory, allowing us to develop the home configuration matrix, *M*, and the screw axes for each joint. The M matrix for the iiwa robot is defined as

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1.261 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

while the screw axes are defined as

$$S_i = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & -0.36 & 0 & 0.78 & 0 & -1.18 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{2}$$

and

$$B_i = [Ad_{M^{-1}}]S_i \tag{3}$$

3 Forward Kinematics

3.1 Method

The **space form** for the product of exponentials formula is given as:

$$T(\theta) = e^{[S_1]\theta_1} \dots e^{[S_{n-1}]\theta_{n-1}} e^{[S_n]\theta_n} M \tag{4}$$

where $M \in SE(3)$ is the end effector configuration, $S_1 \dots S_n$ are the space frame screw axes matrices $\in se(3)$, and $\theta_1 \dots \theta_n$ are the joint angles.

The **body form** formula for the product of exponentials is given as:

$$T(\theta) = M e^{[B_1]\theta_1} \dots e^{[B_{n-1}]\theta_{n-1}} e^{[B_n]\theta_n} \tag{5}$$

where $M \in SE(3)$ is the end effector configuration, B_i, given by $M^{-1}[S_i]M$, are the body frame screw axes matrices $\in se(3)$, and $\theta_1 \dots \theta_n$ are the joint angles.

3.2 Code Details

The modular *FK_space* function is used to calculate the forward kinematics in the space frame. The function inputs a robot structure, the joint configuration q, a show_plot boolean, and a new_fig boolean. The input robot structure contains the number of joints *n_joints*, the end-effector transformation matrix in the home configuration *M*, the space form screw axes *space.screw_axes*, and the optional transformation matrix to display each joint as an axis for the plot *joint_as_end_effector*. The function outputs the end-effector transformation matrix for the space form forward kinematics T as well as a plot of the robot.

The *FK_body* function is used to calculate the forward kinematics in the body frame for any serial open-chain manipulator. The function inputs a robot structure, the joint configuration q, a show_plot boolean, and a new_fig boolean. The input robot structure contains the number of joints *n_joints*, the end-effector transformation matrix in the home configuration M, the body form screw axes *body.screw_axes*, the space form screw axes *space.screw_axes* for plotting purposes, and the optional transformation matrix to display each joint as an axis for the plot *joint_as_end_effector*. The function outputs the end-effector transformation matrix for the body form forward kinematics T as well as a plot of the robot.

3.3 Test Functions

We tested our forward kinematics functions across several test cases. For the first test case, we found a forward kinematics example online that we inputted into our functions [2]. The 3 joint example had an M matrix of

$$M = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 6 \\ 0 & 0 & -1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

The screw axes for the example were defined as

$$S_i = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & -1 \\ 4 & 0 & -6 \\ 0 & 1 & 0 \\ 0 & 0 & -0.1 \end{bmatrix} \tag{7}$$

and

$$B_i = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \\ 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.1 \end{bmatrix} \tag{8}$$

Finally, the input configuration for both cases was equal to

$$q = [\frac{\pi}{2}, 3, \pi] \tag{9}$$

and the desired output for both cases was

$$T = \begin{bmatrix} 0 & 1 & 0 & -5 \\ 1 & 0 & 0 & 4 \\ 0 & 0 & -1 & 1.6858 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10}$$

For our other four test cases, we tested our functions with various input configurations on the iiwa robot. Specifically, we input a configuration of all zeros, the home configuration of the robot as seen in Figure 1, and two randomly generated configurations. To validate, we compared our function outputs to the output generated by MATLAB's Robotics Toolbox *getTransform()* function for the built-in KUKA LBR iiwa 14 model in MATLAB. Furthermore, we used the Robotics Toolbox to visualize each test case for both the native MATLAB iiwa model and our model, as seen in Figures 3 to 6.

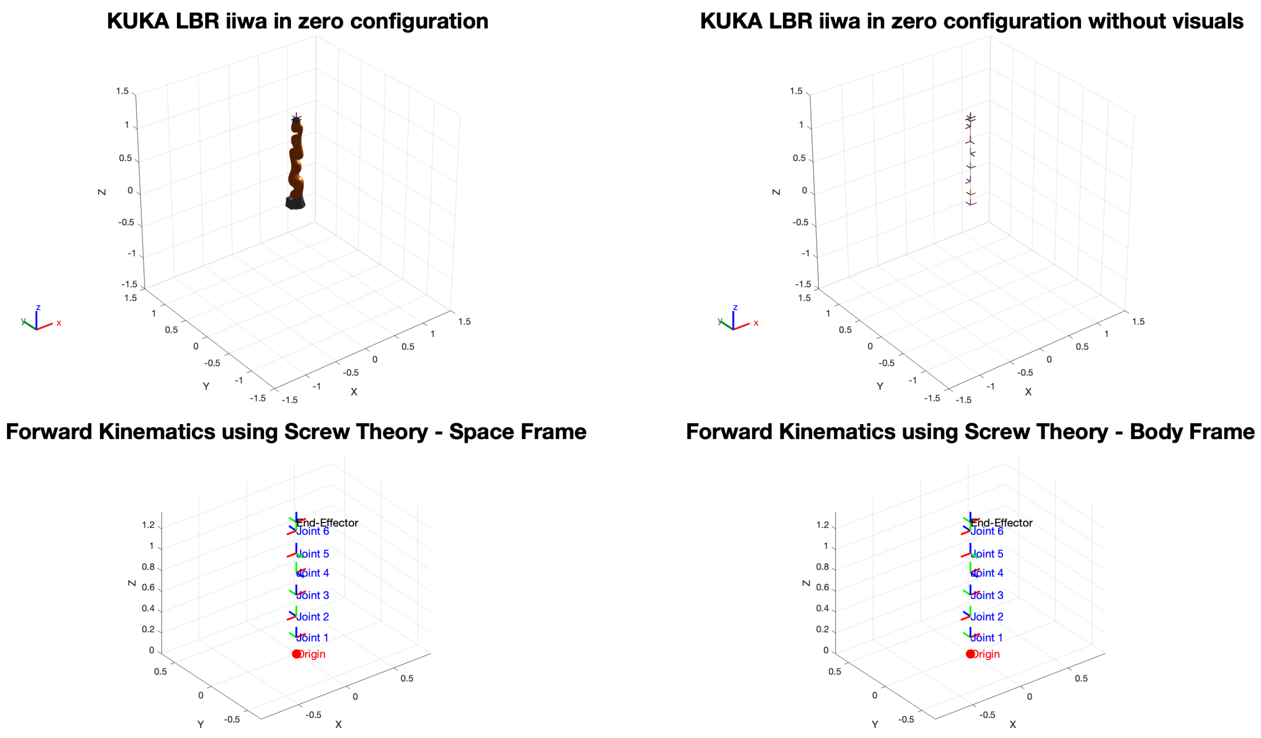
KUKA LBR iiwa in zero configuration
KUKA LBR iiwa in zero configuration without visuals
Forward Kinematics using Screw Theory - Space Frame
Forward Kinematics using Screw Theory - Body Frame
X
Y
Z
End-Effector
Joint 6
Joint 5
Joint 4
Joint 3
Joint 2
Joint 1
Origin

Figure 3: KUKA LBR iiwa 14 forward kinematics model in zero configuration.

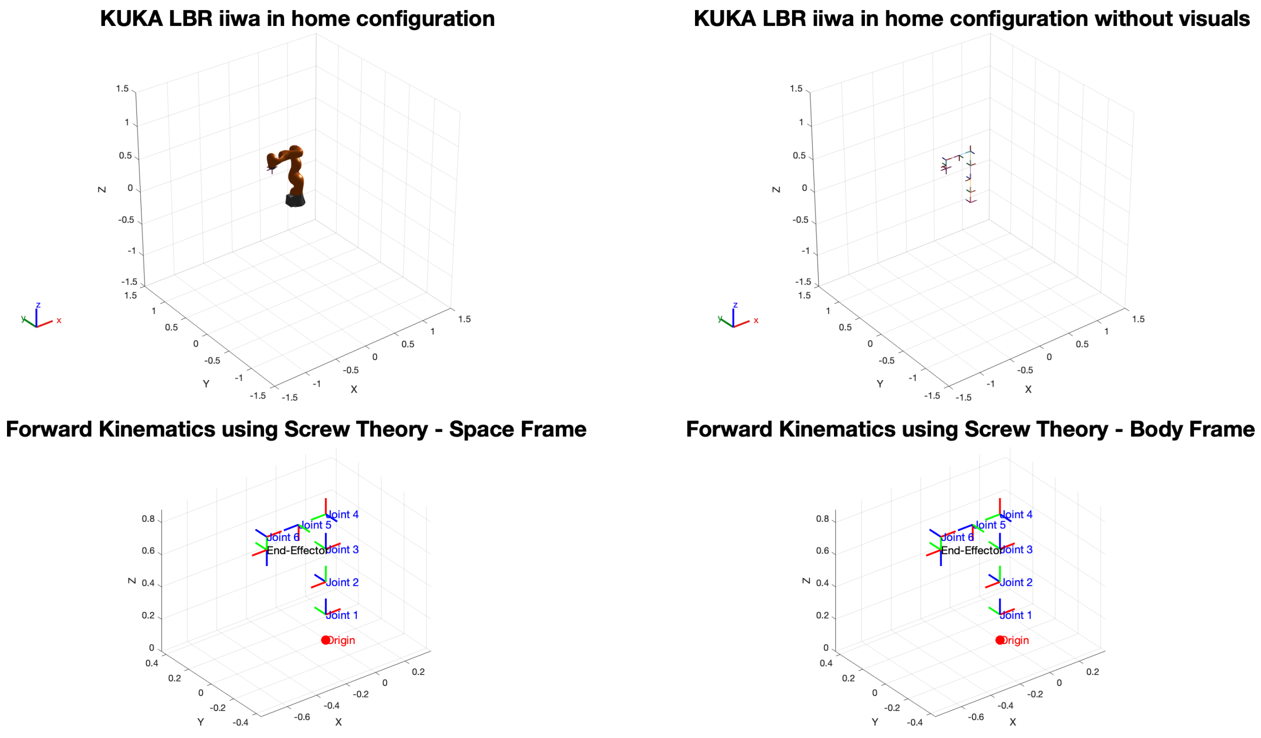
KUKA LBR iiwa in home configuration
KUKA LBR iiwa in home configuration without visuals
Forward Kinematics using Screw Theory - Space Frame
Forward Kinematics using Screw Theory - Body Frame
X
Y
Z
Joint 4
Joint 5
Joint 6
End-Effector
Joint 3
Joint 2
Joint 1
Origin

Figure 4: KUKA LBR iiwa 14 forward kinematics model in home configuration.

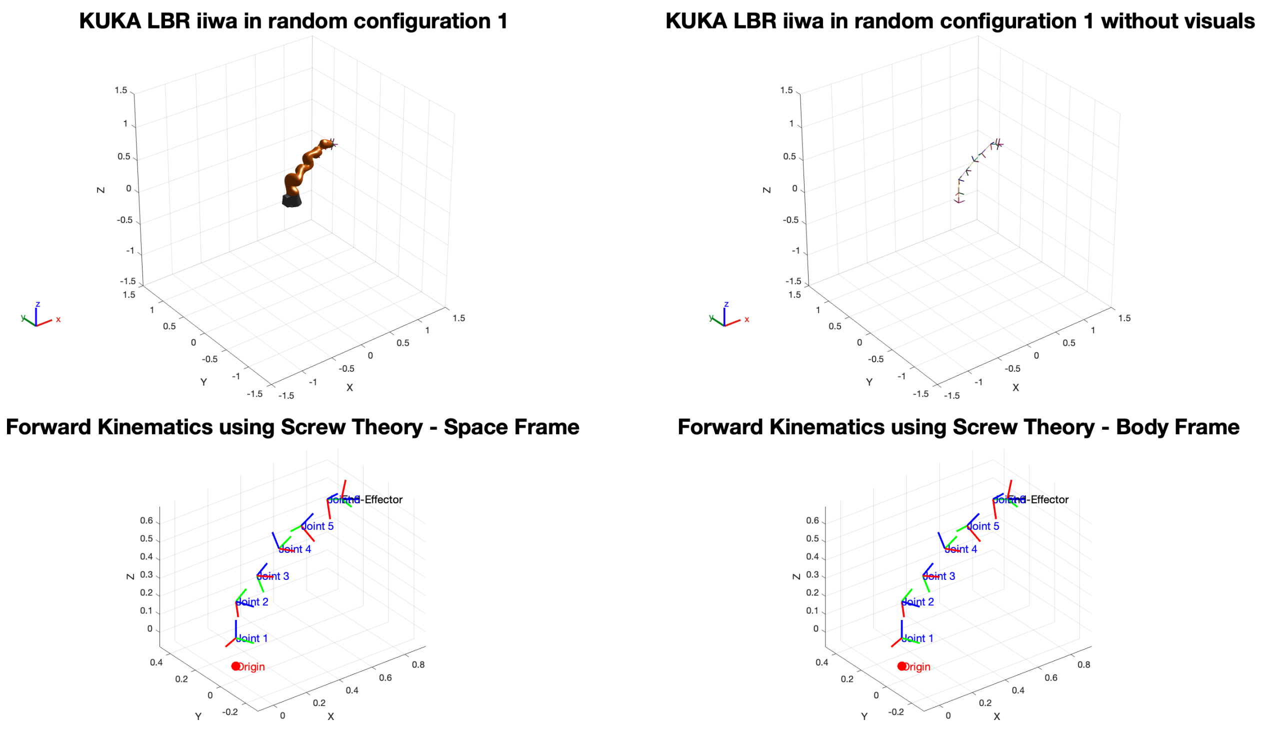

Figure 5: KUKA LBR iiwa 14 forward kinematics model in first random configuration.

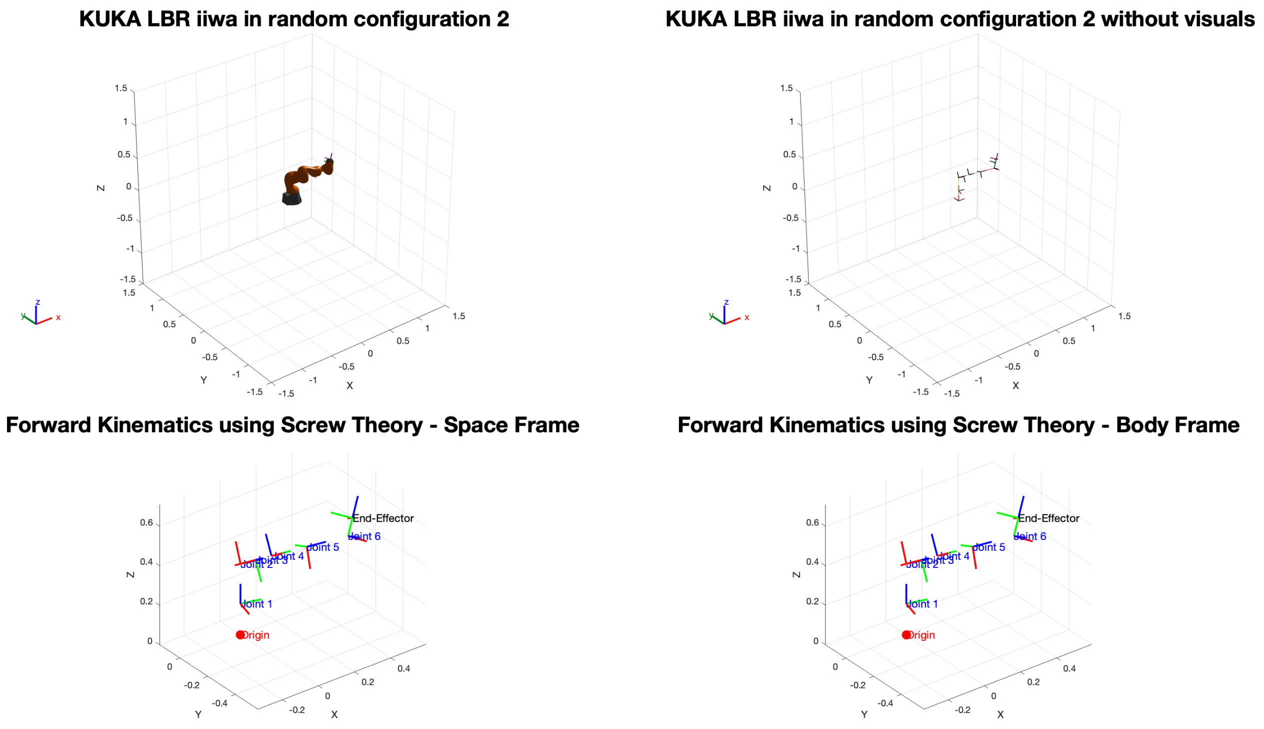

Figure 6: KUKA LBR iiwa 14 forward kinematics model in second random configuration.

4 Jacobian Analysis

4.1 Method

The Jacobian derivation and analysis starts with relating the position of any end-effector or frame, $x(t)$, to the joint angles, $\theta(t)$:

$$x(t) = f(\theta(t)) \tag{11}$$

Differentiating with respect to time, the end-effector velocity, $V_{\text{end-effector}}$, can be written:

$$V_{\text{end-effector}} = \dot{x} = \frac{\partial f(\theta)}{\partial \theta}\dot{\theta} = J(\theta)\dot{\theta} \tag{12}$$

Therefore, the Space Jacobian, $J_s(\theta) \in R^{6m}$, relates the joint rate vector, $\theta \in R^n$, to the twist V_s in the spatial frame via:

$$V_s = J_s(\theta)\dot{\theta} \tag{13}$$

The i-th column of $J_s(\theta)$ is:

$$J_{si}(\theta) = \text{Ad}_{e^{[S_i]\theta} \dots e^{[S_{i-1}]\theta}}(S_i) \tag{14}$$

for $i = 2, \dots, n$, with the first column $J_{s1} = S_1$.

The Body Jacobian, $J_b(\theta) \in R^{6m}$, relates the joint rate vector, $\theta \in R^n$, to the twist V_b in the body frame via:

$$V_b = J_b(\theta)\dot{\theta} \tag{15}$$

The i-th column of $J_b(\theta)$ is:

$$J_{bi}(\theta) = \text{Ad}_{e^{-[B_i]\theta} \dots e^{-[B_n]\theta}}(B_i) \tag{16}$$

for $i = n-1, \dots, 1$, with the first column $J_{bn} = S_n$.

4.2 Code Details

The *J_space* function is used to calculate the space form Jacobian. The function inputs a robot structure and the joint configuration *q*. The input robot structure contains the number of joints *n_joints*, the end-effector transformation matrix in the home configuration *M*, and the space form screw axes *space.screw_axes*. The function then outputs the space form Jacobian *Js*.

The *J_body* function is used to calculate the body form Jacobian. The function inputs a robot structure and the joint configuration *q*. The input robot structure contains the number of joints *n_joints*, the end-effector transformation matrix in the home configuration *M*, and the body form screw axes *body.screw_axes*. The function then outputs the body form Jacobian *Jb*.

4.3 Test Functions

We tested our forward kinematics functions across several test cases. For the first test case, we found a Jacobian calculation example online that we inputted into our functions [2]. The 4 joint example had an input configuration of

$$q = [\frac{\pi}{2}, 3, \pi] \tag{17}$$

The screw axes for the example were defined as

$$S_i = B_i = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0.2 \\ 0.2 & 0 & 2 & 0.3 \\ 0.2 & 3 & -1 & 0.4 \end{bmatrix} \tag{18}$$

The desired output for the space frame was

$$J_s = \begin{bmatrix} 0 & 0.9801 & -0.0901 & 0.9575 \\ 0 & 0.1987 & 0.4446 & 0.2849 \\ 1 & 0 & 0.8912 & -0.0453 \\ 0 & 1.9522 & -2.2164 & -0.5116 \\ 0.2 & 0.4365 & -2.4371 & 2.7754 \\ 0.2 & 2.9603 & 3.2357 & 2.2251 \end{bmatrix} \tag{19}$$

while the desired output for the body frame was

$$J_b = \begin{bmatrix} -0.0453 & 0.995 & 0 & 1 \\ 0.7436 & 0.093 & 0.3624 & 0 \\ -0.6671 & 0.0362 & -0.932 & 0 \\ 2.3259 & 1.6681 & 0.5641 & 0.2 \\ -1.4432 & 2.9456 & 1.4331 & 0.3 \\ -2.0664 & 1.8288 & -1.5887 & 0.4 \end{bmatrix} \tag{20}$$

For our other four test cases, we once again tested our functions using the zero configuration, the home configuration as seen in Figure 1, and two randomly generated configurations. To validate our Jacobian calculations, we used MATLAB's Robotics Toolbox *getTransform()* function to calculate the transformation matrix for the given input and then compared our Jacobian calculations with the following equation from Modern Robotics [3]:

$$J_b(\theta) = [Ad_{T_{bs}}]J_s(\theta) \tag{21}$$

5 Singularity Analysis

5.1 Method

In order to determine if a manipulator is at a singular configuration based on the calculated Jacobian, the rank of the Jacobian has to be checked to ensure that it is full rank.

For the iiwa robot specifically, four configurations were observed that result in a singular configuration [4]:

1. $\theta_4 = 0$
2. $\theta_2 = \theta_6 = 0$
3. $\theta_2 = 0, \theta_3 = \pm\frac{\pi}{2}$
4. $\theta_5 = \pm\frac{\pi}{2}, \theta_6 = 0$

5.2 Code Details

The *singularity* functions is used to determine if the robot is in a singularity configuration. It inputs a Jacobian matrix describing the configuration of the robot at a given point in time. The function then outputs a boolean of *True* if the configuration is singular and *False* if the robot is not in a singularity configuration.

5.3 Test Functions

To test our singularity function, we inputted all four of the singular configurations of the iiwa robot to ensure that the function returned *True* for all of them. Furthermore, we plotted all of the singular configurations using MATLAB's Robotics Toolbox and native KUKA LBR iiwa 14 model, as seen in Figure 7. Finally, we created an input configuration of all random values to ensure that the function was able to recognize non-singular configurations as well.

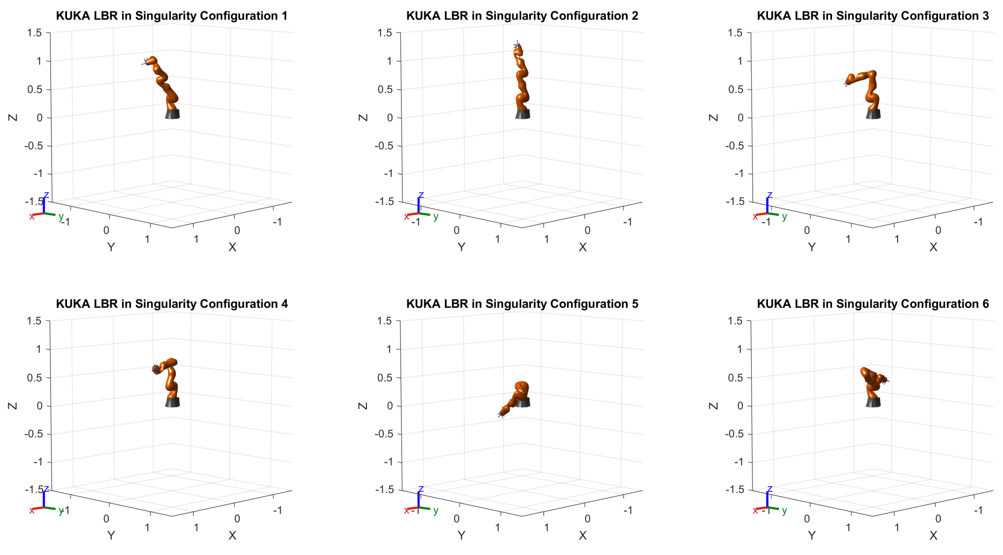
KUKA LBR in Singularity Configuration 1
KUKA LBR in Singularity Configuration 2
KUKA LBR in Singularity Configuration 3
KUKA LBR in Singularity Configuration 4
KUKA LBR in Singularity Configuration 5
KUKA LBR in Singularity Configuration 6
X
Y
Z

Figure 7: Plots of all of the singularity configurations of KUKA LBR iiwa 14.

6 Manipulability Analysis

6.1 Method

To calculate the manipulability ellipsoid, the positive definite matrix containing the Jacobian must first be calculated with

$$A = J * J' \tag{22}$$

The eigenvectors v and eigenvalues λ of this matrix A are then calculated. Finally, the components of the principal semi-axes of the manipulability ellipsoid are calculated by

$$directions_i = \frac{v_i}{norm(v)} \tag{23}$$

$$lengths_i = \sqrt{\lambda_i} \tag{24}$$

The manipulability ellipsoid has several conditions associated with it that describe the ellipsoid. These include the condition number, isotropy number, and ellipsoid volume. To calculate the condition number, the following formula is used

$$condition = \frac{\lambda_{max}}{\lambda_{min}} \tag{25}$$

The isotropy number is calculated with

$$isotropy = \sqrt{\frac{\lambda_{max}}{\lambda_{min}}} \tag{26}$$

Finally, the ellipsoid volume is determined by

$$ellipsoid\ volume = \sqrt{\lambda_1 \dots \lambda_m} \tag{27}$$

6.2 Code Details

The main functions for this section is the *manipulabilityEllipsoid* function. This functions takes in the Jacobian describing the configuration of the robot at a given point in time. It then outputs the directions and lengths of the principal semi-axes of the manipulability ellipsoid.

In order to actually plot the manipulability ellipsoids, the *ellipsoid_plot_angular* and *ellipsoid_plot_linear* functions were created. These functions take in a robot structure and the joint configuration q. The input robot structure contains the number of joints *n_joints*, the end-effector transformation matrix in the home configuration M, and the space form screw axes *space.screw_axes*. Then, the space frame Jacobian and forward kinematics are calculated and the angular or linear portion of the Jacobian is extracted. The manipulability ellipsoid is then calculated for the specific portion of the Jacobian and plotted.

The three functions that check the characteristics of the manipulability ellipsoid, *J_condition, J_isotropy* and *J_ellipsoid_volume*, all take in the Jacobian of a given robot. They then use the *manipulabilityEllipsoid* function to calculate the directions and lengths of the ellipsoid. Finally, the required characteristic is calculated based on the manipulability ellipsoid principal semi-axes.

6.3 Test Functions

To test our manipulability functions, we again used the home configuration and two random configurations with the iiwa robot. To test the condition number and isotropy number of the ellipsoid, we used the *cond* MATLAB function with the $A = JJ'$ matrix calculated separately and ensured that the result of the MATLAB function was the same as our functions. For the ellipsoid volume, we once again calculated $A = JJ'$ separately and calculated the $\sqrt{det(A)}$ to ensure that the output was the same as our created function. Finally, we plotted the angular and linear manipulability ellipsoids on the end-effector of the iiwa robot. We again used MATLAB's Robotics Toolbox and native KUKA LBR iiwa 14 model to visualize the robot, as seen in Figures 8 to 10.

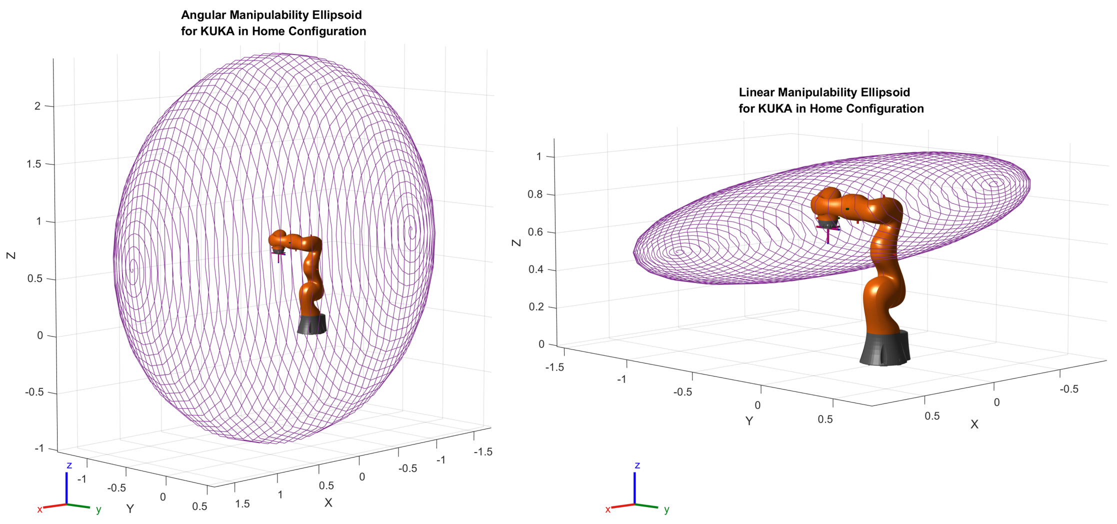

Figure 8: Manipulability ellipsoid of the KUKA LBR iiwa 14 in the home configuration.

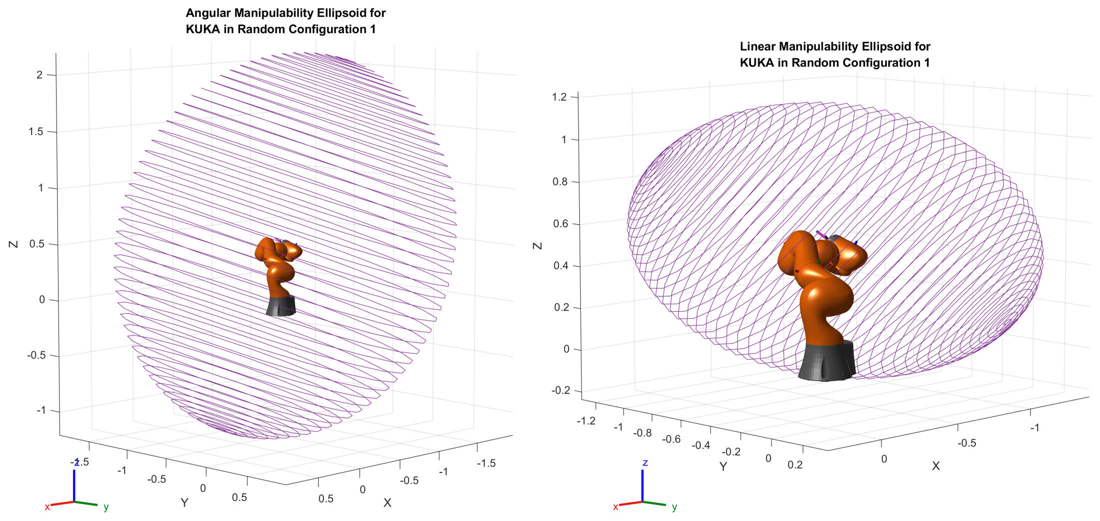

Figure 9: Manipulability ellipsoid of the KUKA LBR iiwa 14 in one random configuration.

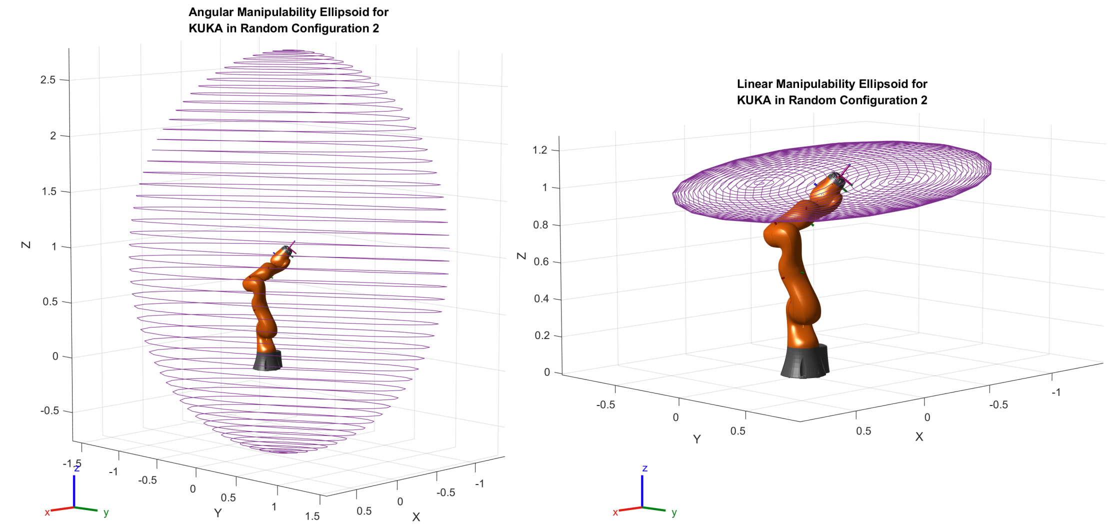

Figure 10: Manipulability ellipsoid of the KUKA LBR iiwa 14 in a second random configuration.

7 Jacobian Inverse Kinematics

7.1 Method

The iterative numerical algorithm to solve for the inverse kinematics using the Jacobian inverse method aims to determine the joint configuration of a robotic manipulator such that its end-effector achieves a desired pose in the workspace. The algorithm iteratively computes the joint configuration $\mathbf{q}$ that minimizes the error between the current end-effector pose and the desired pose.

The forward kinematics function computes the current end-effector pose $\mathbf{T}_b(\mathbf{q})$ in the body frame, given the joint configuration $\mathbf{q}$. The desired pose in the body frame is computed as:

$$\mathbf{T}_{bd} = \mathbf{T}_b(\mathbf{q})^{-1}\mathbf{T}_{sd} \tag{28}$$

where $\mathbf{T}_b(\mathbf{q})$ is the current end-effector pose in the body frame, and $\mathbf{T}_{sd}$ is the desired pose in the space frame.

The error between the current and desired poses is represented as a twist $\mathbf{V}_b$, derived from the matrix logarithm of $\mathbf{T}_{bd}$:

$$\mathbf{V}_b = \text{logm}(\mathbf{T}_{bd}) \tag{29}$$

The twist $\mathbf{V}_b$ is decomposed into angular velocity $\boldsymbol{\omega}$ and linear velocity $\mathbf{v}$:

$$\mathbf{V}_b = \begin{bmatrix} \boldsymbol{\omega} \\ \mathbf{v} \end{bmatrix} \tag{30}$$

The body Jacobian $\mathbf{J}_b(\mathbf{q})$ maps joint velocities to the twist in the body frame. The joint configuration is updated iteratively using the pseudoinverse of the Jacobian:

$$\mathbf{q}_{k+1} = \mathbf{q}_k + \mathbf{J}_b(\mathbf{q}_k)^{\dagger}\mathbf{V}_b \tag{31}$$

The iteration stops when the norm of the twist $\mathbf{V}_b$ falls below a small threshold ϵ (e.g., 10^{-4}) or when the maximum number of iterations $N_{\max}$ is reached.

7.2 Code Details

The *J_inverse_kinematics* function is used to calculate the inverse kinematics of the robot using the iterative numerical Jacobian inverse method. The function takes in a robot structure, the initial transformation matrix $\mathbf{T}_i$, the final transformation matrix $\mathbf{T}_f$, the initial joint configuration $\mathbf{q}_i$, and the maximum number of iterations $N_{\max}$. The input robot structure contains the number of joints *n_joints*, the end-effector transformation matrix in the home configuration *M*, the space form screw axes *space.screw_axes*, and the body form screw axes *body.screw_axes*. The function outputs a vector *q* that contains all of the joint configurations across the iterations, with the last one being the desired joint configuration, along with an error vector *e*.

7.3 Test Functions

To test our Jacobian inverse kinematics function, we created a test function that iterates through 5 configurations with the iiwa robot. The function creates a random initial and final configuration using the *getTransform()* function from the MATLAB Robotics Toolbox. Then, the *J_inverse_kinematics* function is used to conduct the inverse kinematics and calculate the final joint configuration. This final joint configuration is then input into our forward kinematics function to calculate the transformation matrix derived from the inverse kinematics function. Finally, the desired final transformation and calculated transformation are verified.

8 Jacobian Transpose Kinematics

8.1 Method

The iterative numerical algorithm to solve for the inverse kinematics with the Jacobian transpose is similar to the iterative method from the Jacobian inverse calculation. However, instead of calculating $J^{\dagger}$, we define a symmetric positive definite diagonal K matrix with angular and linear error terms such that

$$\mathbf{K} = \begin{bmatrix} a_w & & & & & \\ & a_w & & & & \\ & & a_w & & & \\ & & & a_v & & \\ & & & & a_v & \\ & & & & & a_v \end{bmatrix} \tag{32}$$

This K matrix is used to iteratively update the joint configuration

$$\dot{q} = J_b^T(q_i)Ke \tag{33}$$

leading to

$$q_{i+1} = q_i + J_b^T(q_i)K\upsilon_b \tag{34}$$

$$\mathbf{q}_{k+1} = \mathbf{q}_k + \mathbf{J}_b^T(\mathbf{q}_k)\mathbf{K}\mathbf{V}_b \tag{35}$$

During testing of the function, the inverse kinematics algorithm struggled to converge to a configuration solution for the 1000 iterations with high values of angular and linear error terms in the K matrix. As a result, the angular and linear error terms used for the K matrix currently are 10^{-1}. However, for more thorough use cases, these error terms can and should be adjusted, along with the number of iterations.

8.2 Code Details

The *J_tranpose_kinematics* functions is used to calculate the inverse kinematics of the robot using the Jacobian tranpose method. The functions takes in a robot structure, the final transformation matrix T_f, the initial joint configuration q_i, and the maximum number of iterations. The input robot structure contains the number of joints *n_joints*, the end-effector transformation matrix in the home configuration *M*, the space form screw axes *space.screw_axes*, and the body form screw axes *body.screw_axes*. The function then outputs the desired joint configuration *q_sol*, an error vector *errors*, and a vector *joint_configs* that contains all of the joint configurations across the iterations.

8.3 Test Functions

To test our Jacobian transpose inverse kinematics function, we created a test function that iterates through 5 configurations with the iiwa robot. The function creates a random initial and final configuration using the *getTransform()* function from the MATLAB Robotics Toolbox. Then, the *J_tranpose_kinematics* function is used to conduct the inverse kinematics and calculate the final joint configuration. This final joint configuration is then input into our forward kinematics function to calculate the transformation matrix calculated from the inverse kinematics function. Finally, the desired final transformation and calculated transformation are verified.

9 Redundancy Resolution

9.1 Method

The optimization problem can be mathematically expressed as follows:

$$\min g(q) = \frac{1}{2}q^T W q \tag{36}$$

subject to the constraint:

$$\text{s.t } v_e = J(q)q \tag{37}$$

where q represents the vector of joint angles, v_e is the end-effector velocity, and $J(q)$ is the Jacobian matrix that relates joint velocities to end-effector velocities. The matrix W serves as a weighting matrix that allows for the prioritization of certain joint movements over others, thus enabling fine-tuning of the manipulator's response characteristics.

To solve this constrained optimization problem, we utilize the method of Lagrange multipliers. The first-order necessary conditions yield the following relationship for the optimal joint configuration:

$$q = W^{-1} J^T \lambda \tag{38}$$

where λ is the vector of Lagrange multipliers associated with the constraints imposed by the end-effector velocity.

For redundant manipulators, where the number of degrees of freedom exceeds the number of task constraints, and when W is set to the identity matrix, the general solution can be expressed as:

$$q = q^* + J^\dagger v_e + (I - J^\dagger J)^{-1} \dot{q_0} \tag{39}$$

Here, q^* denotes a particular solution that satisfies the task constraints, P_{q_0} is the projection matrix onto the null space of the Jacobian, and q_0 represents the initial joint configuration. The term $(I - J^T J)^{-1} q_0$ accounts for the contribution of the redundant degrees of freedom, allowing the manipulator to avoid kinematic singularities and maintain operational stability.

To facilitate the avoidance of kinematic singularities, we define our objective function as the manipulability measure:

$$w(q) = \sqrt{\det(J(q)J(q)^T)} \tag{40}$$

This measure quantifies the ability of the manipulator to exert forces in various directions, with higher values indicating greater manipulability. To compute the gradient $\dot{q_0} = k_0 \frac{\partial w(q)}{\partial q}$ numerically, we employ a finite difference approach, calculating the difference between the manipulability measure $w(q)$ at the current iteration and the previous iteration. This gradient is then utilized to inform the optimization process, guiding the manipulator towards configurations that enhance its operational capabilities while adhering to the specified constraints.

9.2 Code Details

The *redundancy_resolution* functions is used to calculate the inverse kinematics of the robot using the redundancy resolution method. The functions takes in a robot structure, the initial transformation matrix T_i, the final transformation matrix T_f, the initial joint configuration q_i, the maximum number of iterations, and the manipulability measure gain K. The input robot structure contains the number of joints *n_joints*, the end-effector transformation matrix in the home configuration *M*, the space form screw axes *space.screw_axes*, and the body form screw axes *body.screw_axes*. The function then outputs a vector q that contains all of the joint configurations across the iterations with the last one being the desired joint configuration along with an error vector e.

9.3 Test Functions

To test our redundancy resolution function, we created a test function that iterates through 5 configurations with the iiwa robot. The function creates a random initial and final configuration using the *getTransform()* function from the MATLAB Robotics Toolbox. Then, the *redundancy_resolution* function is used to conduct the inverse kinematics and calculate the final joint configuration. This final joint configuration is then input into our forward kinematics function to calculate the transformation matrix calculated from the inverse kinematics function. Finally, the desired final transformation and calculated transformation are verified.

10 Damped Least Squares Method

10.1 Method

The Damped Least Squares (DLS) method is a robust approach for solving inverse kinematics problems, particularly in scenarios where the manipulator may encounter kinematic singularities or when the Jacobian matrix is ill-conditioned. This method modifies the standard least squares solution by introducing a damping factor, which helps stabilize the solution and improve convergence. The DLS method aims to minimize the following objective function:

$$\min \|v_e - J(q)\dot{q}\|^2 + \lambda^2\|\dot{q}\|^2 \tag{41}$$

where v_e is the desired end-effector velocity, $J(q)$ is the Jacobian matrix, $\dot{q}$ is the vector of joint velocities, and λ is the damping factor. The term $\lambda^2\|\dot{q}\|^2$ serves to penalize large joint velocities, effectively regularizing the solution and preventing excessive movements that could lead to instability.

To derive the solution, we can rewrite the objective function in matrix form:

$$\min \|v_e - J(q)\dot{q}\|^2 + \lambda^2\|\dot{q}\|^2 \tag{42}$$

The solution to this optimization problem can be obtained by taking the gradient and setting it to zero, leading to the following normal equations:

$$J(q)^T J(q)\dot{q} + \lambda^2\dot{q} = J(q)^T v_e \tag{43}$$

Rearranging this equation gives us:

$$(J(q)^T J(q) + \lambda^2 I)\dot{q} = J(q)^T v_e \tag{44}$$

where I is the identity matrix. The solution for $\dot{q}$ can then be expressed as:

$$\dot{q} = (J(q)^T J(q) + \lambda^2 I)^{-1} J(q)^T v_e \tag{45}$$

$$J^* = (J(q)^T J(q) + \lambda^2 I)^{-1} J(q)^T \tag{46}$$

This formulation ensures that the solution remains well-defined even when the Jacobian $J(q)$ is near singularity, as the damping factor λ provides a means to control the influence of the regularization term.

10.2 Code Details

The *DLS_inverse_kinematics* functions is used to calculate the inverse kinematics of the robot using the Damped Least Squared (DLS) method. The functions takes in a robot structure, the initial transformation matrix T_i, the final transformation matrix T_f, the initial joint configuration q_i, the maximum number of iterations, and the damping factor λ. The input robot structure contains the number of joints *n_joints*, the end-effector transformation matrix in the home configuration *M*, the space form screw axes *space.screw_axes*, and the body form screw axes *body.screw_axes*. The function then outputs a vector q that contains all of the joint configurations across the iterations with the last one being the desired joint configuration along with an error vector e.

10.3 Test Functions

To test our DLS inverse kinematics function, we created a test function that iterates through 5 configurations with the iiwa robot. The function creates a random initial and final configuration using the *getTransform()* function from the MATLAB Robotics Toolbox. Then, the *DLS_inverse_kinematics* function is used to conduct the inverse kinematics and calculate the final joint configuration. This final joint configuration is then input into our forward kinematics function to calculate the transformation matrix calculated from the inverse kinematics function. Finally, the desired final transformation and calculated transformation are verified.

11 Discussion

During the implementation and testing of our kinematic functions, several issues with the native MATLAB Robotic Toolbox KUKA iiwa robot arose. For the Jacobian calculations, we first implemented our test functions use the *geometricJacobian()* function from the Robotic Toolbox. However, we noticed that, while the Jacobian values were the same, some of the signs of the values were different compared to our Jacobian values. We believe that this discrepancy arises from the fact that MATLAB uses DH parameters for the Robotic Toolbox, along with a potential discrepancy in how the axes were defined for each joint on the KUKA iiwa robot. Another issue that we noticed was the scale of the KUKA iiwa robot on the plotting. As seen in Figures 8 - 10, the manipulability ellipsoid calculated seems to be scaled up in comparison to the KUKA iiwa robot plotted by the Robotic Toolbox. While we believe that our manipulability ellipsoid calculations are correct, the discrepancy in scale might arise from the KUKA iiwa robot size being defined differently in the MATLAB Robotic Toolbox as compared to how we defined it.

Since we implemented several different methods to solve for the inverse kinematics, we created an additional test function to compare the solutions of each method across a common test case. While all of the inverse kinematics methods were able to solve for this random test case, there was several noticeable differences between the methods. The Jacobian transpose method reached the maximum number of iterations while the Jacobian inverse method took the least number of iterations as seen in Figure 11. Furthermore, the actual desired configuration and the configurations of each method's final solution can be seen in Figure 12. The Jacobian transpose method can be seen to be the worst performing, both in terms of the final solution and the number of iterations. While the basic Jacobian inverse method took less iterations than the redundancy resolution method, the redundancy resolution method seemed to perform better in terms of the final solution and the number of iterations was not significantly higher. Finally, while the damped least squares method seemed to perform the best in terms of the final solution, it took twice the number of iterations as the Jacobian inverse and redundancy resolution methods. However, the damped least squares method still took significantly less iterations than the Jacobian transpose method and was not close to reaching the maximum number of iterations at all.

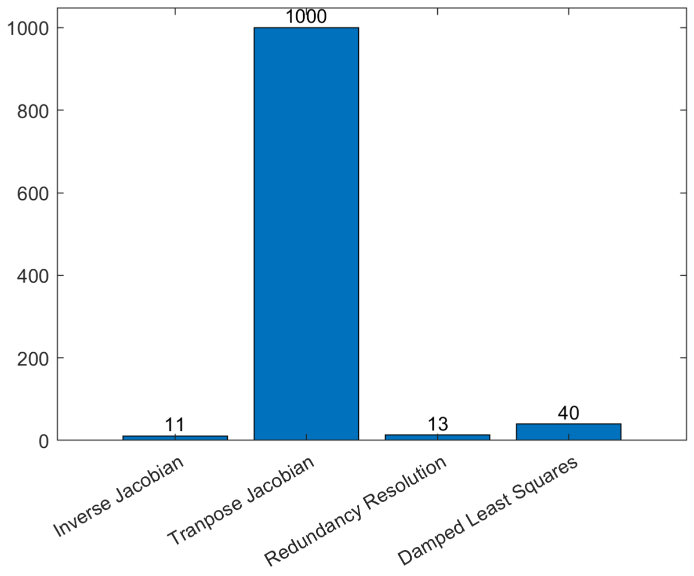

Figure 11: Number of iterations required to reach a solution for each inverse kinematics algorithm on a given random desired configuration.

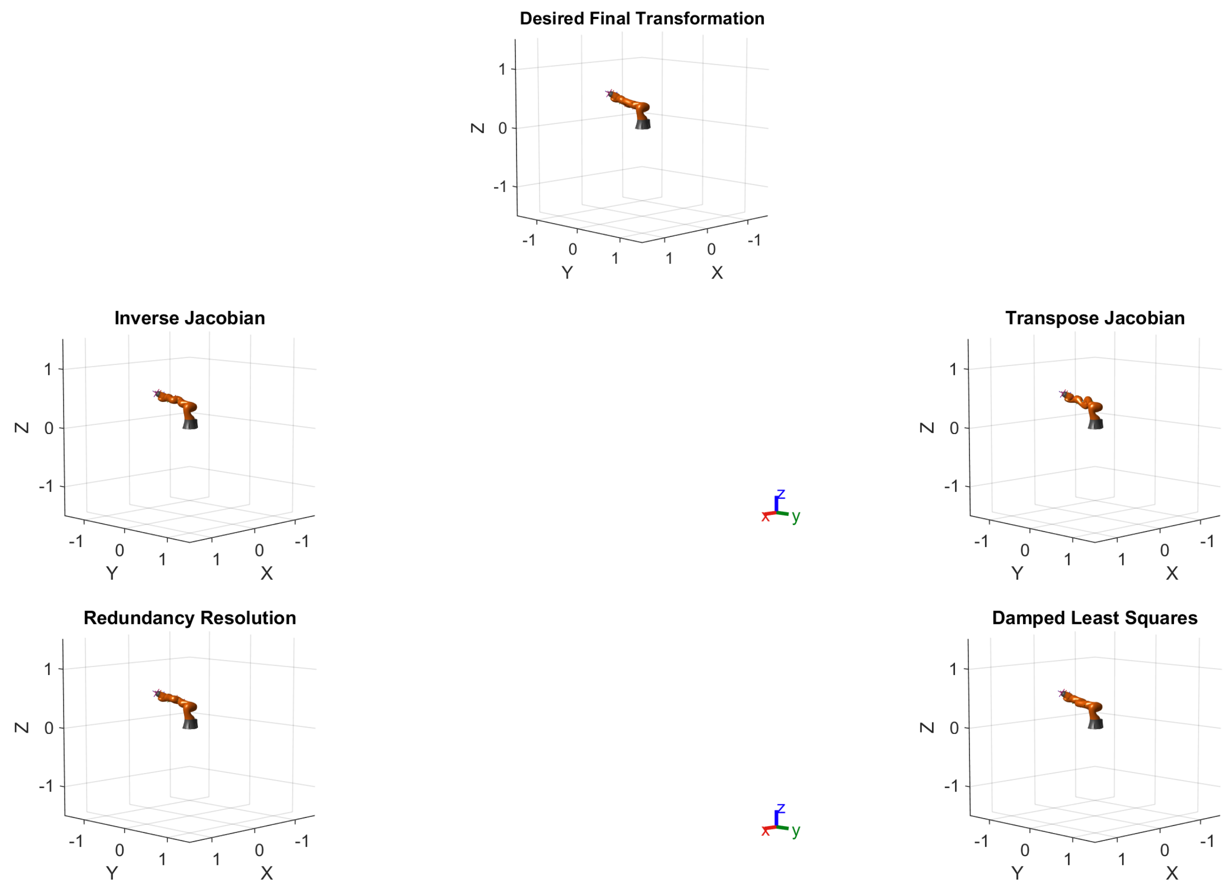

Figure 12: Graphical representation of final solution for each inverse kinematics algorithm on a given random desired configuration.

12 Conclusion

This project involved creating MATLAB functions that solved for the forward and inverse kinematics of a robot, derived the Jacobian of a robot, and analyzed the singularity and manipulability of a robot. While all of these functions are created to be modular, all of these functions were tested with the KUKA LBR iiwa 14 robot. With this project, it was important to designate space and body reference frames, and ensure that we were always operating in the correct frame. Additionally, the graphing capabilities of the MATLAB Robotic Toolbox allowed for more robust understanding of the functions developed, particularly for the kinematic configurations and manipulabilities of the robot. Finally, the implementation of several inverse kinematics methods allowed for a deeper understanding into the advantages and disadvantages of each method.

13 Helper Functions

Several helper functions were created to make the implementation of the core functions easier. The new helper functions for THA2 include:

- $[Adjoint\_T] = adjointMatrix(T)$
- $plotEllipsoid(center, axes)$

- $[twist] = screw2twist(point, axis, pitch)$
- $[matrix] = screw\_axis\_to\_se3(S)$
- $[S] = skew\_to\_screw\_axis(matrix)$
- $[twist] = transfmat2twist(transfmat)$

The old helper functions used from THA1 include:

- $[is\_valid] = is\_valid\_rotation\_matrix(R, check\_identity)$
- $[result] = is\_valid\_transform\_matrix(transfmat)$
- $[isEqual] = isequal\_tol(A, B, tol)$
- $plotScrewAxis(point, axis, pitch)$
- $[axes] = plotTransformAxes(transfmat, color, label)$
- $[axis, angle] = rotmat2axisangle(R)$
- $[point, axis, pitch, angle] = transfmat2screw(transfmat)$

14 Contributions

Daniyal Maroufi worked on HA 1, HA 3, PA a-e, PA h, PA j-m.

Anas Yousaf worked on HA 2, HA 4, PA f-g, PA i, PA l-m.

References

[1] *LBR iiwa*, en-US. [Online]. Available: `https://www.kuka.com/en-us/products/robotics-systems/industrial-robots/lbr-iiwa`.

[2] *ModernRobotics/packages/MATLAB/mr/FKinBody.m at master · NxRLab/ModernRobotics*, en. [Online]. Available: `https://github.com/NxRLab/ModernRobotics/blob/master/packages/MATLAB/mr/FKinBody.m`.

[3] K. M. Lynch and F. C. Park, *Modern Robotics: Mechanics, Planning, and Control*, en, 1st ed. Cambridge University Press, May 2017, ISBN: 978-1-316-66123-9 978-1-107-15630-2 978-1-316-60984-2. DOI: `10.1017/9781316661239`. [Online]. Available: `https://www.cambridge.org/core/product/identifier/9781316661239/type/book`.

[4] "KUKA Sunrise.OS 1.11 KUKA Sunrise.Workbench 1.11," en,