ME 384R ASBR: THA4 - Programming Assignment Report

Daniyal Maroufi and Anas Yousaf

May 1, 2025

1 Introduction

In this assignment we work on the KUKA LBR iiwa 14 ((Fig. 1)) robot and assume that, at every control step, we can measure the current position of its tool tip, p_{tip}, in a fixed world frame. Our objective is two-fold:

1. **Precision:** Keep the tool tip as close as possible to a desired target point, p_{goal}, *without ever allowing* the distance
$$\|p_{\text{tip}} - p_{\text{goal}}\|_2 \leq 3 \text{ mm}.$$

2. **Safety:** Ensure all joint displacements Δq remain within the robot's physical limits, i.e.
$$q^L - q \;\leq\; \Delta q \;\leq\; q^U - q,$$
so as to avoid hardware damage and maintain smooth, collision-free motion.

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

To achieve this we cast the control problem as a constrained optimization: at each time instant the controller solves for the joint increment Δq that minimizes the tracking error $\|p_{\text{tip}} - p_{\text{goal}}\|_2$ (and, in more advanced formulations, also penalizes excessive tip motion), subject to the two constraints above. By doing so in real time—at rates of several hundred hertz—the robot can perform high-precision tasks (e.g., assembly, machining, or delicate manipulation) while guaranteeing both sub-millimeter accuracy around the goal and strict adherence to joint safety limits.

2 Joint Constraints

2.1 Joint Position Limits

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 1 shows the allowable range of each joint with respect to the zero position of the robot seen in Figure 2.

Table 1: 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

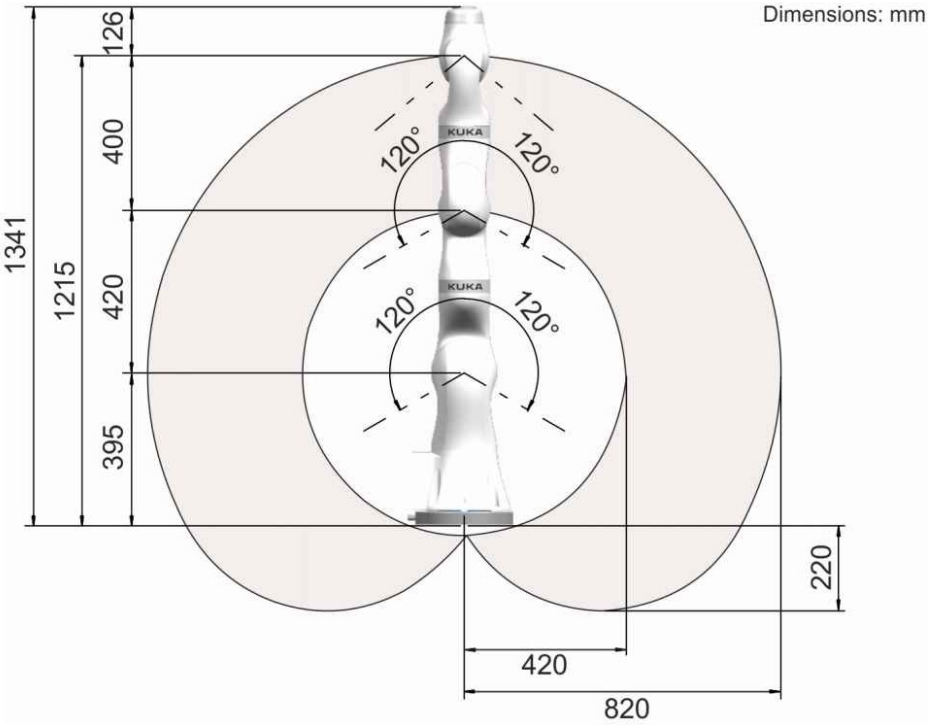

Figure 2: The workspace of the KUKA LBR iiwa 14 robot [1].

2.2 Joint Speed Limits

To ensure safe and smooth motion, each of the seven revolute joints of the KUKA LBR iiwa 14 robot is also subject to a maximum angular velocity. Table 2 summarizes the manufacturer-specified speed limit for each joint.

Table 2: Maximum joint speeds for the KUKA LBR iiwa 14 robot (°/s).

Joint 1	**Joint 2**	**Joint 3**	**Joint 4**	**Joint 5**	**Joint 6**	**Joint 7**
85	85	100	75	130	135	135

3 Method

3.1 Part A: Objective Function

In Part A, the objective function is designed to minimize the distance between the target position p_{goal} and the tip. Numerically, this is written as

$$\Delta q_{\text{des}} = \arg\min_{\Delta q} \left\| \vec{\alpha} \times \vec{t} + \vec{\epsilon} + \vec{t} - p_{\text{goal}} \right\|_2 \tag{1}$$

where Δq denotes the angular displacement for the current iteration of inverse kinematics, and $\vec{\alpha}$ and $\vec{\epsilon}$ represent

$$\vec{\alpha} = J_\omega^s(q)\,\Delta q \tag{2}$$

$$\vec{\epsilon} = J_v^s(q)\,\Delta q \tag{3}$$

Since the tip is fixed with respect to the end effector (i.e. p_{tip} is constant), for each iteration of the inverse kinematics we use the computed transform T_{sb} to compute $\vec{t}$ as

$$\vec{t} = T_{sb}\,p_{\text{tip}} \tag{4}$$

The distance constraint discussed above can be written in terms of the objective function as

$$\left\| \vec{\alpha} \times \vec{t} + \vec{\epsilon} + \vec{t} - p_{\text{goal}} \right\|_2 \leq 3 \text{ mm} \tag{5}$$

For our MATLAB implementation of this constrained optimization, we utilized the built-in function `lsqlin` [2], which solves

$$\min_x \ \|\mathbf{C}\,x - \mathbf{d}\|_2^2 \quad \text{subject to} \quad \mathbf{A}\,x \leq \mathbf{b},\ \mathbf{A}_{\text{eq}}\,x = \mathbf{b}_{\text{eq}},\ \mathbf{lb} \leq x \leq \mathbf{ub}.$$

In our notation, $x = \Delta q$ is the joint-increment vector we seek at each control step. The inputs to `lsqlin` are:

- $\mathbf{C}$ and $\mathbf{d}$, defining the least-squares cost $\|\mathbf{C}\,\Delta q - \mathbf{d}\|_2^2$. For the simple distance objective these are chosen so that $\mathbf{C}\,\Delta q - \mathbf{d} = \vec{\alpha} \times \vec{t} + \vec{\epsilon} + \vec{t} - p_{\text{goal}}$.
- $\mathbf{A}$ and $\mathbf{b}$, collecting all linear inequality constraints. These include:
  1. Joint limits: $q^L - q \leq \Delta q \leq q^U - q$, expressed as two sets of rows in $\mathbf{A}$ and $\mathbf{b}$.
  2. Linearized "stay-within-3 mm" constraints (see next subsection).
- Optionally, $\mathbf{A}_{\text{eq}}, \mathbf{b}_{\text{eq}}$ for any equality constraints (none in Part A), and $\mathbf{lb}, \mathbf{ub}$ to enforce simple lower/upper bounds (same as joint limits).

3.2 Linearization of the 3 mm Distance Constraint (VF1)

We approximate the spherical constraint [3]

$$\left\| \vec{\alpha} \times \vec{t} + \vec{\epsilon} + \vec{t} - p_{\text{goal}} \right\|_2 \ \leq\ 3 \text{ mm}$$

by an inscribed polyhedron. Let $\delta = [\,\delta_p^T,\ 0^T]^T$ with $\delta_p = \vec{t} - p_{\text{goal}}$. We require

$$\|\delta + \Delta x\|_2^2 \ =\ \|\delta_p + \Delta x_p\|_2^2 \ \leq\ \varepsilon^2 \tag{6}$$

where $\varepsilon = 3$ mm and $\Delta x = [\,\Delta x_p^T,\ 0^T]^T$. Introducing spherical angles

$$\alpha_i = \tfrac{2\pi\, i}{n}, \quad \beta_j = \tfrac{2\pi\, j}{m}, \quad i = 0, \ldots, n-1,\ j = 0, \ldots, m-1,$$

we enforce

$$\begin{bmatrix} \cos\alpha_i \cos\beta_j & \cos\alpha_i \sin\beta_j & \sin\alpha_i & 0 & 0 & 0 \end{bmatrix} (\delta + \Delta x) \ \leq\ \varepsilon, \tag{7}$$

for each i, j. Stacking these into $\mathbf{A} \in R^{(nm)\times 7}$ and $\mathbf{b} \in R^{nm}$ gives

$$\mathbf{A} = \begin{bmatrix} \cos\alpha_0 \cos\beta_0 & \cos\alpha_0 \sin\beta_0 & \sin\alpha_0 & 0 & 0 & 0 & 0 \\ \vdots & & & & & & \\ \cos\alpha_{n-1} \cos\beta_{m-1} & \cos\alpha_{n-1} \sin\beta_{m-1} & \sin\alpha_{n-1} & 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{b} = \varepsilon \mathbf{1} - \mathbf{A}\,\delta. \tag{8}$$

By assembling $\mathbf{C}, \mathbf{d}, \mathbf{A}, \mathbf{b}$ as above and passing them to

```
delta_q = lsqlin(C, d, A, b, Aeq, beq, lb, ub);
```

we obtain the joint update Δq that simultaneously minimizes the tip-tracking error and enforces both the 3 mm proximity and joint-limit constraints.

3.3 Part B: Combined Position and Orientation Objective

In Part B, we extend the objective to minimize both the distance to the goal and the change in orientation of the tool tip. Since the tip frame's orientation is fixed relative to the end-effector, the rotation R_{sb} also gives the tip's axes in the fixed frame. In particular, the tip's primary axis in $\{s\}$ is

$$\mathbf{t}_{\text{ips}} = R_{sb}\, z_s \tag{9}$$

where z_s is the unit vector along the z-axis of the fixed frame. The incremental rotation of this axis induced by Δq can be linearized via the skew-symmetric form of $\vec{\alpha}$, so that $\vec{\alpha} \times (R_{sb}\, z_s)$ approximates the change in orientation.

Altogether, the new cost combines position error and orientation error:

$$\Delta q_{\text{des}} = \arg\min_{\Delta q} \left[\zeta \left\| \vec{\alpha} \times \vec{t} \,+\, \vec{\epsilon} \,+\, \vec{t} \,-\, p_{\text{goal}} \right\|_2^2 \,+\, \eta \left\| \vec{\alpha} \times (R_{sb}\, z_s) \right\|_2^2 \right] \tag{10}$$

where ζ and η are scalar weights balancing the two objectives. In the current implementation, the ratio of η to ζ is used for simplicity without loss of generality of the method.

$$r = \frac{\eta}{\zeta} \tag{11}$$

4 Code Details

This section provides detailed explanations of the main functions implemented for this assignment.

4.1 minimum_distance_solver.m

The `minimum_distance_solver` function calculates the joint angle changes required to move a manipulator's end effector towards a desired position while obeying joint limits and minimizing changes in tool tip orientation.

Inputs:

- `t_curr`: A 3-element vector containing the current tool tip position in the space frame
- `p_goal`: A 3-element vector containing the goal tool tip position in the space frame
- `J`: A $6 \times N$ space form Jacobian of the manipulator at the current position
- `q_L`: An N-element vector of the lower limits of manipulator joints (optional)
- `q_U`: An N-element vector of the upper limits of manipulator joints (optional)
- `tool_axis`: A 3-element vector containing the direction of the tool axis (optional)

Output:

- `delta_q`: An N-element vector of joint angle changes

Implementation Details:

1. The function first handles optional inputs, setting default values if parameters are not provided.
2. It then creates skew-symmetric matrix representations of the current tool position (`t_skew`) and the tool axis (`tool_skew`).
3. A linear least squares objective function is defined, where:
   - Matrix `C` combines both position and orientation control components
   - Vector `d` represents the error between current and goal positions
4. The joint angle changes are calculated by solving the optimization problem that minimizes $\|C \cdot \delta q + d\|$ subject to joint limits (`q_L` and `q_U`).
5. The algorithm uses MATLAB's `lsqlin` function with the 'active-set' algorithm to solve this constrained optimization problem.
6. Finally, it checks if the solver exited successfully and provides a warning if not.

4.2 robot_reach_goal.m

The `robot_reach_goal` function calculates a joint space path for a robot to approach a goal point from an initial configuration.

Inputs:

- `robot`: A struct containing robot parameters (number of joints, end-effector transformation matrix, screw axes)
- `p_goal`: A 3-element vector containing the goal position in the space frame
- `p_tip`: A 3-element vector containing the tool tip position in the body frame
- `q_init`: An N-element vector containing the initial joint configuration
- `q_L`: An N-element vector of joint lower limits (optional)
- `q_U`: An N-element vector of joint upper limits (optional)
- `q_speed`: An N-element vector of joint speed limits (optional)
- `orientation`: A weighting factor for orientation control (0 for no orientation control)

Outputs:

- `q`: An $N \times 7$ matrix of joint positions, where each row represents a time step
- `distance_error`: An N-element vector of position errors at each time step

Implementation Details:

1. The function initializes parameters including error threshold and time step.
2. It handles optional inputs, converting joint limits from degrees to radians if provided.
3. The initial tool tip position is calculated using forward kinematics.
4. The algorithm then iteratively:
   - Calculates the space Jacobian for the current configuration
   - Determines joint movement limits based on joint limits and speed constraints
   - Sets the tool axis orientation (multiplied by the orientation weighting factor)

- Calls `minimum_distance_solver` to calculate optimal joint changes
- Updates the joint configuration and tool position
- Recalculates the position error

5. The iteration continues until either the error falls below the threshold or the maximum iterations are reached.
6. The orientation parameter allows toggling between Part A (position-only control when orientation is set to zero) and Part B (position and orientation control when orientation is set to a non-zero value).

4.3 test_robot_goal.m

The `test_robot_goal` script demonstrates and evaluates the robot control algorithm with various test scenarios.

Implementation Details:

1. The script first loads a KUKA LBR iiwa 14 robot model and sets its joint limits and speed constraints.
2. It then runs four separate test scenarios:
   - Part A: Moving from home position to a close goal (position control only)
   - Part A: Moving from home position to a far goal (position control only)
   - Part B: Moving from home position to a close goal with orientation control
   - Part B: Moving from home position to a far goal with orientation control
3. For each test scenario, the script:
   - Defines initial conditions, tool tip position, and goal position
   - Calls `robot_reach_goal` with the appropriate orientation parameter (0 for Part A, 1 for Part B)
   - Visualizes the robot's motion and creates animations
   - Tracks and plots key performance metrics (distance error, isotropy number, condition number)
   - Generates manipulability ellipsoids to analyze the robot's performance
   - Saves outputs as images and GIFs

Table 3: Test scenarios for Parts A and B

Part	Goal Distance	Control Mode
A	Close	Position only
A	Far	Position only
B	Close	Position + Orientation
B	Far	Position + Orientation

The script serves as a comprehensive testing framework that allows for visual and numerical validation of both parts of the assignment by simply toggling the orientation parameter between 0 and 1.

5 Test Function

To test our constrained optimization solver functions, we created two test cases that we used with the position objective and the position and orientation objective. For the first test case, we started the robot at its home position, as seen in Figure 1, and defined a goal point close to the initial tool tip location. For the second test case, we again started the robot at its home position and then defined a goal point that was far away from

the initial tool tip location. These two test cases allowed us to observe the differences in the calculated joint trajectory when varying the goal location as well as the addition of the orientation constraints. To analyze these test cases, we plot the initial and final positions of the robot, the manipulability ellipsoid at the final position, and the distance error, condition number, and isotropy number across the iterations of the robot movement. We also created GIFs of the robot movement throughout all of the joint configurations and the corresponding manipulability ellipsoids which can be found in the 'Outputs' folder in the code base.

5.1 Position Objective

For the first test case, we used inputs of:

- $q_o = [0, 0, 0, \frac{\pi}{2}, 0, -\frac{\pi}{2}, 0]$
- $p_{tip} = [0, 0, 0.1]$ (m)
- $p_{goal} = [-0.5, 0, 0.1]$ (m)

The initial and final position of the robot after iterating through the generated joint trajectory is seen in Figure 3.

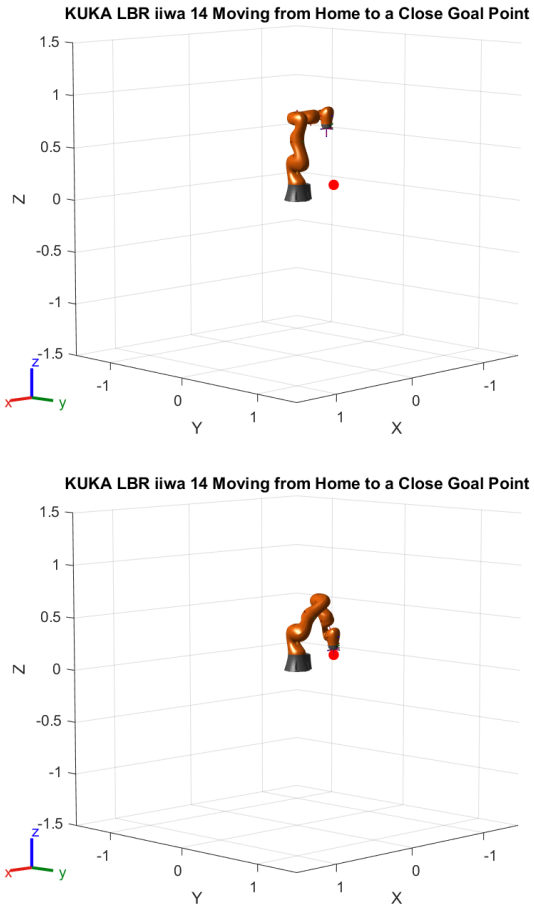

Figure 3: Plots of the KUKA LBR iiwa 14 in its initial position and final position after it has used the position objective function to generate a joint trajectory for a test case with a close goal point.

The angular and linear manipulability ellipsoids of the robot at its final position can be seen in Figure 4.

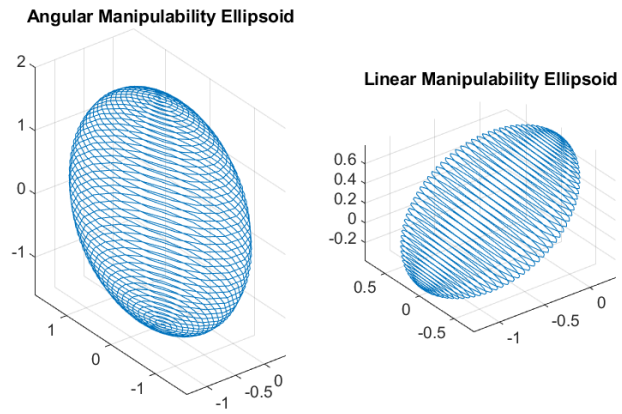

Figure 4: Angular and linear manipulability ellipsoids at the final position of the robot after it has used the position objective function to generate a joint trajectory for a test case with a close goal point.

The distance error, isotropy number, and condition number of the robot as it iterates through each joint configuration can be seen in Figure 5

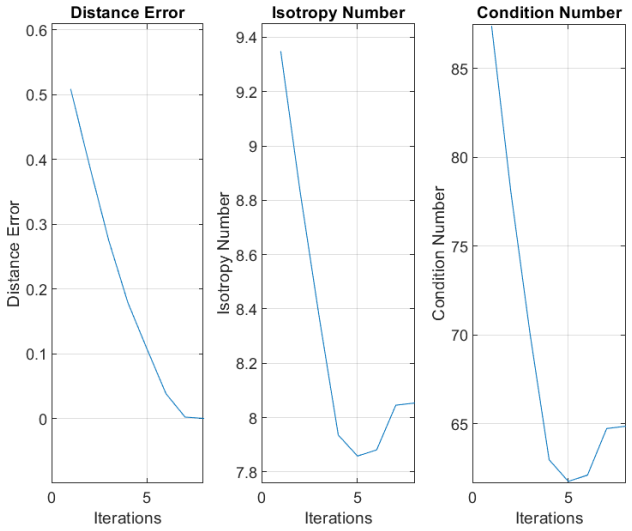

Figure 5: Distance error, isotropy number, and condition number of the robot after it has used the position objective function to generate a joint trajectory for a test case with a close goal point.

For the second test case, we used inputs of:

- $q_o = [0, 0, 0, \frac{\pi}{2}, 0, -\frac{\pi}{2}, 0]$
- $p_{tip} = [0, 0, 0.1]$ (m)
- $p_{goal} = [0.3, 0.5, 0.7]$ (m)

The initial and final position of the robot after iterating through the generated joint trajectory is seen in

Figure 6.

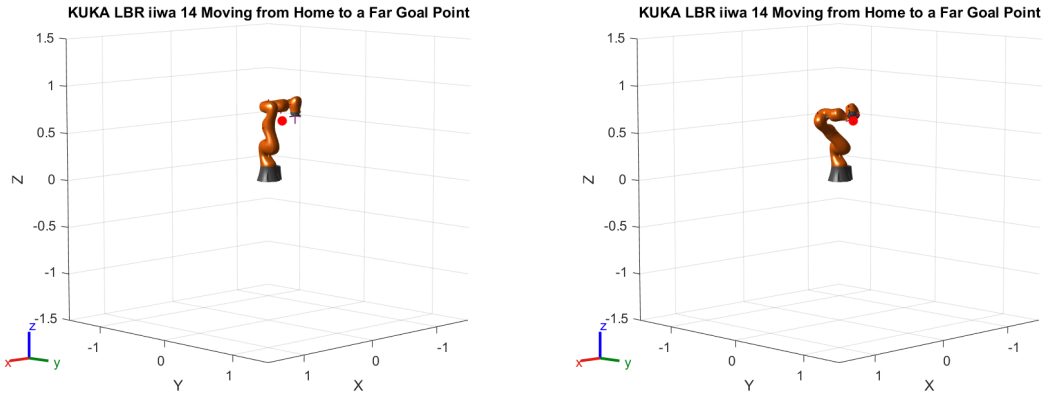

Figure 6: Plots of the KUKA LBR iiwa 14 in its initial position and final position after it has used the position objective function to generate a joint trajectory for a test case with a far goal point.

The angular and linear manipulability ellipsoids of the robot at its final position can be seen in Figure 7.

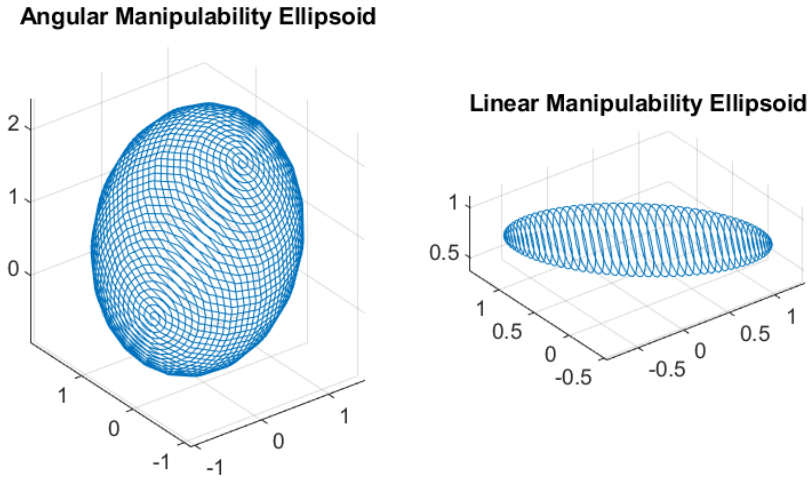

Figure 7: Angular and linear manipulability ellipsoids at the final position of the robot after it has used the position objective function to generate a joint trajectory for a test case with a far goal point.

The distance error, isotropy number, and condition number of the robot as it iterates through each joint configuration can be seen in Figure 8

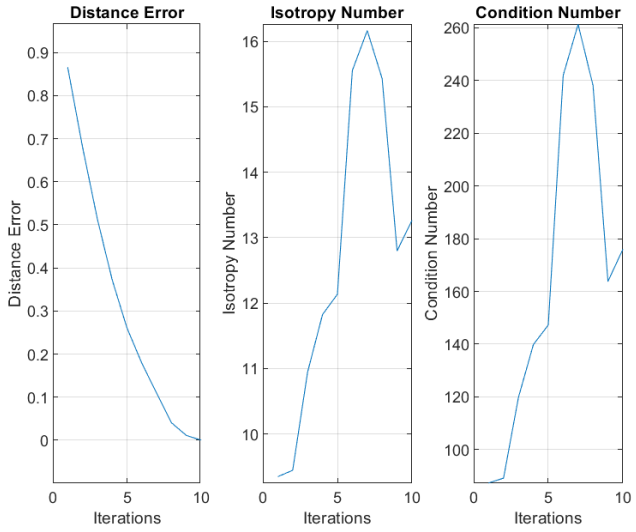

Figure 8: Distance error, isotropy number, and condition number of the robot after it has used the position objective function to generate a joint trajectory for a test case with a far goal point.

5.2 Position and Orientation Objective

For the first test case, we used inputs of:

- $q_o = [0, 0, 0, \frac{\pi}{2}, 0, -\frac{\pi}{2}, 0]$
- $p_{tip} = [0, 0, 0.1]$ (m)
- $p_{goal} = [-0.5, 0, 0.1]$ (m)

The initial and final position of the robot after iterating through the generated joint trajectory is seen in Figure 9.

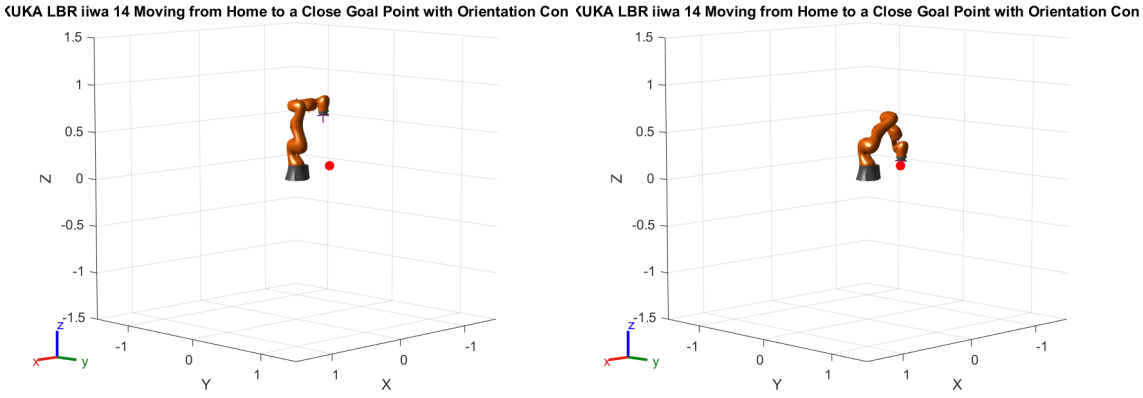

Figure 9: Plots of the KUKA LBR iiwa 14 in its initial position and final position after it has used the position and orientation objective function to generate a joint trajectory for a test case with a close goal point.

The angular and linear manipulability ellipsoids of the robot at its final position can be seen in Figure 10.

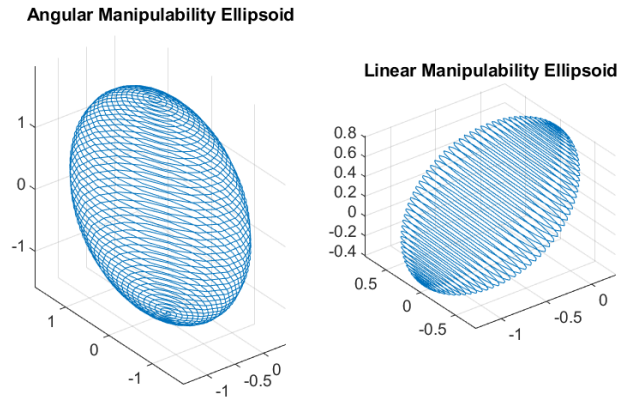

Figure 10: Angular and linear manipulability ellipsoids at the final position of the robot after it has used the position and orientation objective function to generate a joint trajectory for a test case with a close goal point.

The distance error, isotropy number, and condition number of the robot as it iterates through each joint configuration can be seen in Figure 11

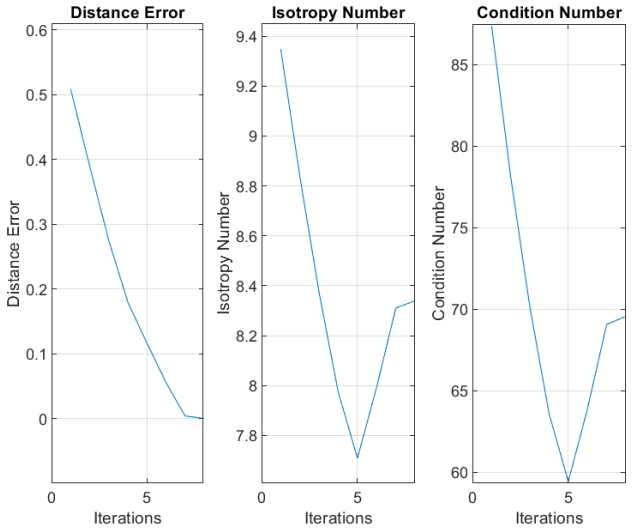

Figure 11: Distance error, isotropy number, and condition number of the robot after it has used the position and orientation objective function to generate a joint trajectory for a test case with a close goal point.

For the second test case, we used inputs of:

- $q_o = [0, 0, 0, \frac{\pi}{2}, 0, -\frac{\pi}{2}, 0]$
- $p_{tip} = [0, 0, 0.1]$ (m)
- $p_{goal} = [0.3, 0.5, 0.7]$ (m)

The initial and final position of the robot after iterating through the generated joint trajectory is seen in Figure 12.

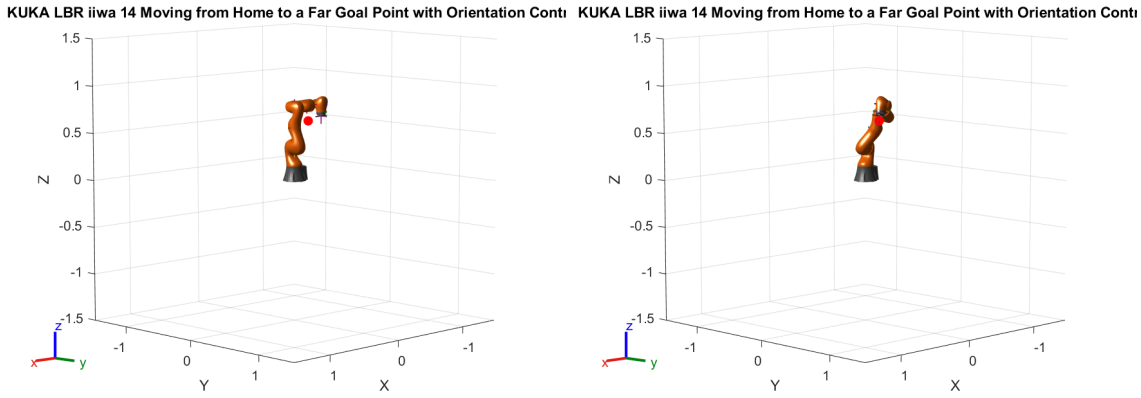

Figure 12: Plots of the KUKA LBR iiwa 14 in its initial position and final position after it has used the position and orientation objective function to generate a joint trajectory for a test case with a far goal point.

The angular and linear manipulability ellipsoids of the robot at its final position can be seen in Figure 13.

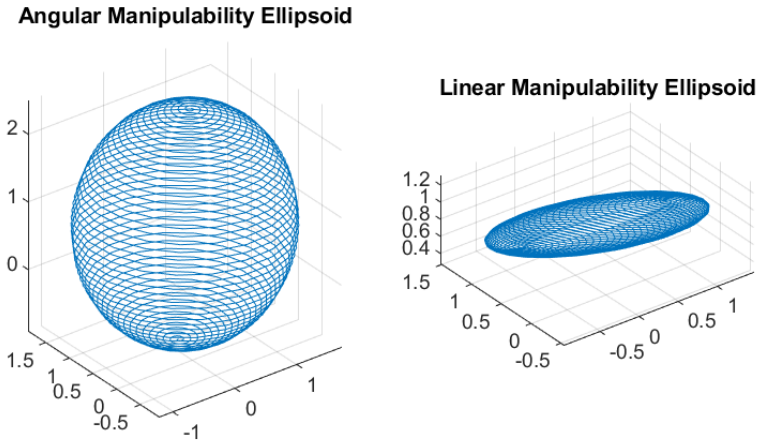

Figure 13: Angular and linear manipulability ellipsoids at the final position of the robot after it has used the position and orientation objective function to generate a joint trajectory for a test case with a far goal point.

The distance error, isotropy number, and condition number of the robot as it iterates through each joint configuration can be seen in Figure 14

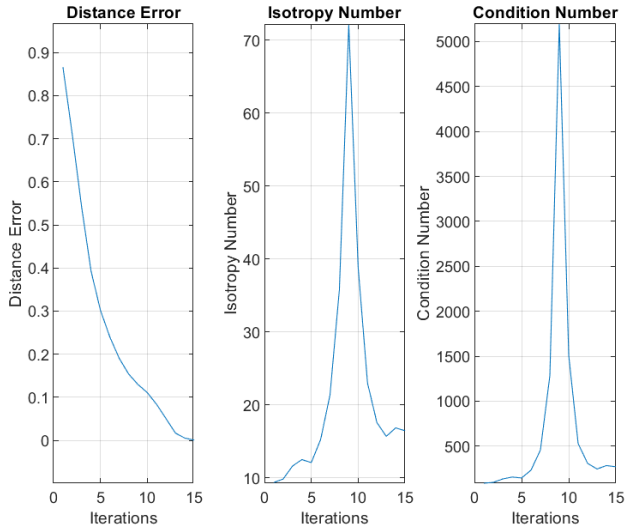

Figure 14: Distance error, isotropy number, and condition number of the robot after it has used the position and orientation objective function to generate a joint trajectory for a test case with a far goal point.

6 Discussion

In our implementation of the minimum distance solver, we encountered significant convergence issues when attempting to incorporate sphere linearization constraints. The theoretical approach involved approximating a sphere using multiple linear constraints as discussed in the methods section. When these constraints were added to our optimization problem, the algorithm consistently failed to converge within the maximum number of iterations. During execution, the robot would move in an oscillatory pattern around the target position without successfully reaching it. This convergence failure can be attributed to several key factors: The sphere linearization approach failed due to multiple compounding issues: the system became over-constrained with 100 additional constraints significantly increasing optimization complexity; numerical precision problems arose from trigonometric functions, especially near the target position; conflicting requirements emerged between sphere constraints and joint limits that couldn't be simultaneously satisfied within tolerance; and the resulting mathematically "stiff" problem meant small joint configuration changes caused large constraint violation fluctuations, preventing the solver from finding viable improvement directions for convergence.

Our alternative implementation, which focuses on minimizing the distance error directly in the objective function without explicitly constraining it within a linearized sphere, proved more effective and reliable. This approach allows the solver to make continuous progress toward the goal without being restricted by potentially conflicting geometric constraints. Since the solver minimizes the distance to the goal point at every time step, the tool tip will still be within the precision constraints defined. Furthermore, since the solver does not have the linearized sphere constraints, the robot could theoretically start at a position further away than 3 millimeters and still converge without running into issues.

Overall, while the sphere linearization method is theoretically sound and would provide explicit guarantees about the maximum distance from the target, its practical implementation requires careful parameter tuning that proved challenging within the scope of this assignment. The direct minimization approach achieved the desired behavior without the convergence issues encountered in the sphere constraint implementation.

In terms of the behavior seen in the joint trajectories produced, we see that the number of iterations naturally increases for a goal point further away from the initial point. However, the increase in iterations is much more drastic when including orientation preservation in the objective function. Intuitively, this makes sense as preserving the tool tip orientation requires to robot to adjust the other joints much more, requiring more

iterations for convergence. Additionally, the distance error decreases across each iteration of each test case. This is the behavior we desired to see, as it shows that the objective function is working properly to minimize the difference in the actual and desired tool tip position. Finally, for the isotropy and condition numbers, we see that in the test cases where the final position is close to the initial position, the isotropy and condition numbers decrease until the robot reaches the final position, meaning that the robot is more stable with each joint configuration iteration. However, for the test case with a far goal point, we see a big jump in the isotropy and condition numbers in the middle of the joint configurations but then see the numbers come back down. This could indicate that a joint configuration in the middle of the trajectory is close to singular, however, the objective function is able to handle this and continue on to reach the goal point. Finally, we see that the condition and isotropy numbers tend to be higher when implementing the orientation control. This again makes sense intuitively, as by fixing the tool tip, the other joints have to be placed in less stable positions.

7 Conclusion

This project involved creating MATLAB functions that implemented a constrained linear least-squares optimization algorithm to solve for a robot trajectory in joint space that moved the robot towards a desired tool tip location. While the functions are created to be modular, we implemented and tested each function with the KUKA LBR iiwa 14 robot. This project allowed us to explore different methods of implementation for constrained optimization problems. Additionally, it demonstrated how to implement things such as precision and safety when developing robotic trajectories like joint position and velocity limits. Finally, we were able to observe the differences in behavior when providing varying goal points as well as the differences in behavior when including tool tip orientation preservation in the objective function.

8 Helper Functions

The helper functions from previous assignments that were utilized include:

- $[adjoint] = adjointMatrix(T)$ - Calculates the adjoint representation of a transformation matrix.
- $J = J\_space(robot, q)$ - Calculates the space Jacobian of a robot using screw theory.
- $T = FK\_space(robot, q)$ - Calculates the forward kinematics of a robot using screw theory in the space frame.
- $[S] = screw\_axis\_to\_se3(s)$ - Converts a screw axis to its SE(3) matrix representation.
- $J = J\_body(robot, q)$ - Calculates the body Jacobian of a robot using screw theory.
- $T = FK\_body(robot, q)$ - Calculates the forward kinematics of a robot using screw theory in the body frame.
- $[dirs, lens] = manipulabilityEllipsoid(J)$ - Calculates the manipulability ellipsoid from a Jacobian matrix.
- $isotropy = J\_isotropy(J)$ - Calculates the isotropy index of a Jacobian matrix.
- $cond = J\_condition(J)$ - Calculates the condition number of a Jacobian matrix.

9 Contributions

Daniyal Maroufi worked on HA 1, PA (a) and PA (b).

Anas Yousaf worked on PA (a) and PA (b).

References

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

[2] T. F. Coleman and Y. Li, "A reflective newton method for minimizing a quadratic function subject to bounds on some of the variables," *SIAM Journal on Optimization*, vol. 6, no. 4, pp. 1040–1058, 1996.

[3] M. Li, A. Kapoor, and R. H. Taylor, "Telerobotic control by virtual fixtures for surgical applications," *Advances in Telerobotics*, pp. 381–401, 2007.