Optimizing Numerical Inverse Kinematics of the Thumb in Python

Engineering 021 - Final Project

Dec 16, 2024

Paolo Bosques-Paulet, Dylan Jacobs, Ryan Bollimpalli

Objectives

Objective 1:	Simulate natural thumb movement
Objective 2:	Given an initial position and desired final position, move thumb iteratively toward the final position via the most efficient path.
Objective 3:	Apply optimization to forward and inverse kinematics of the thumb (and robotic joints) to solve this problem.

Overview

Step 0: Make the Thumb.

Define Position of thumb based on joint angles

Define joint constraints

Step 1: Go Forward.

Approximate a step forward by first order taylor series

Ensure correct direction using the damped Jacobian pseudoinverse

Step 2: Find a Better Way.

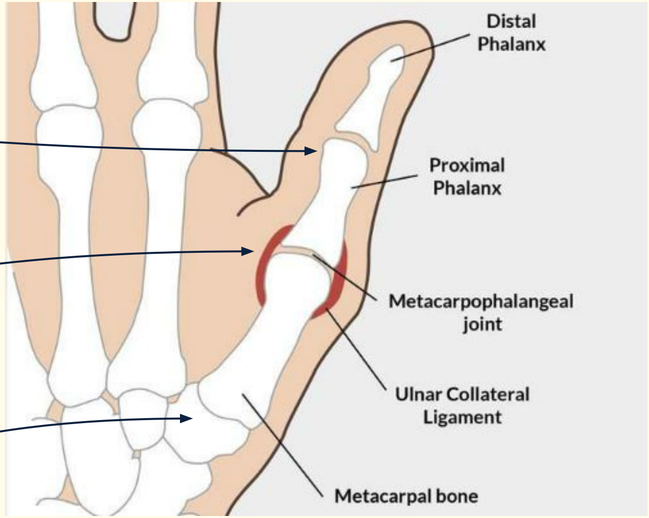
Search the null space of the jacobian for a more optimal set of theta values

Travel along the determined endpoints using the optimal theta values

Step 0 - Thumb design

Joints & Links

- **Interphalangeal joint**
  - 88° flexion, 12° extension
- **Metacarpophalangeal joint**
  - ~0° adduction, ~10° abduction
  - 60° flexion, 8.1° extension
- **Carpometacarpal joint**
  - 10.2° adduction, 62.9° abduction
  - 31.2° flexion, 61.2° extension



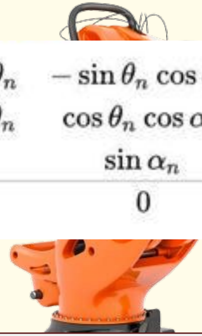
3 Joints with 5 Degrees of Freedom

Step 0 - Forward Kinematics

Joints & Links

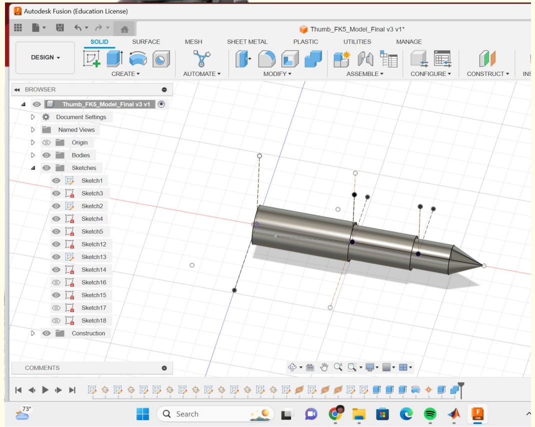
- **Denavit-Hartenberg Parameters**

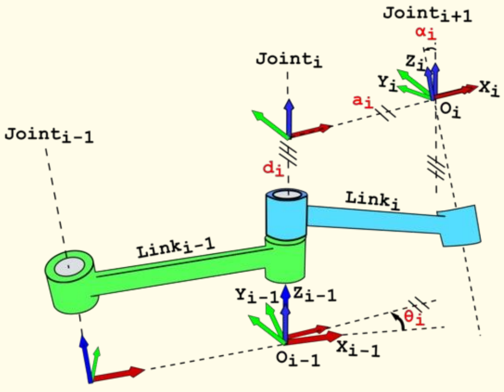
allow easy/standardized endpoint computation



$$\begin{bmatrix} \cos\theta_n & -\sin\theta_n\cos\alpha_n & \sin\theta_n\sin\alpha_n & r_n\cos\theta_n \\ \sin\theta_n & \cos\theta_n\cos\alpha_n & -\cos\theta_n\sin\alpha_n & r_n\sin\theta_n \\ 0 & \sin\alpha_n & \cos\alpha_n & d_n \\ \hline 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\left[\begin{array}{ccc|c} & R & & T \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$





Step 1: Towards Endpoint Via Iteration

Goal: $q_i \rightarrow q_f$

Characterize angle changes to reach a 3D endpoint $\mathbf{q_f}$ from an initial position $\mathbf{q_i}$ using a series of small linear steps.

Method:

$$f(\theta_n + \Delta\theta) \approx f(\theta_n) + J(\theta_n)\Delta\theta$$

- **Find guess for ΔΘ**
  Which leads to a final position $\mathbf{f(\Theta_n+\Delta\Theta)}$ that is closer to $\mathbf{q_f}$ using the above approximation

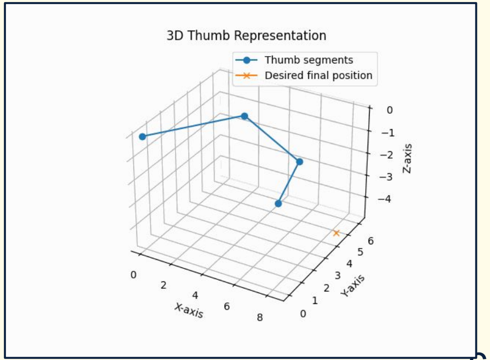
$$l(\Delta\theta) = ||q_f - (f(\theta_n) + J(\theta_n)\Delta\theta)|| + \lambda||\Delta\theta||^2$$

- **Minimize the Loss**
  Set derivative of loss function = 0 → rearrange to achieve final solution for **ΔΘ.**
- λ penalizes large steps

Final Solution:

$$\Delta\theta = (J^T J + \lambda I)^{-1} J^T \Delta q$$

- **Damped Jacobian Pseudoinverse** allows for computation of **ΔΘ**



Step 2: Optimizing ΔΘ

Goal: optimize ΔΘ

Given some $\Delta\Theta_n$ that achieves $(q_n \rightarrow q_{n+1})$,

find smallest angle change $\Delta\Theta_f$ that achieves same $\mathbf{q_{n+1}}$

First, find all ΔΘ that result in no positional change
$f(\Theta_{n+1} + \Delta\Theta) - f(\Theta_{n+1}) = 0$

$$\mathbf{f}(\theta_n + \boldsymbol{\Delta}\theta) \approx \mathbf{f}(\theta_n) + \mathbf{f}'(\theta_n)\boldsymbol{\Delta}\theta$$

$$\mathbf{0} = \mathbf{f}(\theta_n + \boldsymbol{\Delta}\theta) - \mathbf{f}(\theta_n) \approx \mathbf{f}'(\theta_n)\boldsymbol{\Delta}\theta$$

$$\mathbf{f}'(\theta_n)\boldsymbol{\Delta}\theta \approx \mathbf{0}$$

$$J_{\mathbf{f}}(\theta_n)\boldsymbol{\Delta}\theta \approx \mathbf{0}$$

$$J(\boldsymbol{\theta})\Delta\boldsymbol{\theta} = \begin{bmatrix} \frac{\partial f_1}{\partial \theta_1} & \frac{\partial f_1}{\partial \theta_2} & \frac{\partial f_1}{\partial \theta_3} & \frac{\partial f_1}{\partial \theta_4} & \frac{\partial f_1}{\partial \theta_5} \\ \frac{\partial f_2}{\partial \theta_1} & \frac{\partial f_2}{\partial \theta_2} & \frac{\partial f_2}{\partial \theta_3} & \frac{\partial f_2}{\partial \theta_4} & \frac{\partial f_2}{\partial \theta_5} \\ \frac{\partial f_3}{\partial \theta_1} & \frac{\partial f_3}{\partial \theta_2} & \frac{\partial f_3}{\partial \theta_3} & \frac{\partial f_3}{\partial \theta_4} & \frac{\partial f_3}{\partial \theta_5} \end{bmatrix} \begin{bmatrix} \Delta\theta_1 \\ \Delta\theta_2 \\ \Delta\theta_3 \\ \Delta\theta_4 \\ \Delta\theta_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\boldsymbol{\Delta}\theta \in null\{\mathbf{J}(\theta_\mathbf{n})\}$$

Step 2 Optimizing ΔΘ

Goal: Given set of **ΔΘ** that achieve same position $\mathbf{q}_{n+1}$

Find smallest **ΔΘ** so that at each iteration, we update **ΔΘ** as little as possible → most optimal path for the thumb to take from $\mathbf{q}_n \rightarrow \mathbf{q}_{n+1}$.

Minimize **ΔΘ** in null space and penalize objective function if new **ΔΘ** violates biological thumb constraints.

$$\Delta\theta \in null\{J(\theta)\}$$

Find **ΔΘ** that minimizes f^*

$$f^*(\theta_n, \Delta\theta, \lambda) = ||\Delta\theta||^2 + \lambda||g(\theta_n, \Delta\theta)||^2$$

$$g(\theta_n, \Delta\theta) = \min(0, \theta_{min} - (\theta_n + \Delta\theta)) + \max((\theta_n + \Delta\theta) - \theta_{max}, 0)$$

$g(\mathbf{\Theta}_n, \mathbf{\Delta\Theta})$ = constraint function

Nullspace of the Jacobian

$$J(\theta) = \begin{bmatrix} \frac{\partial x}{\partial \theta_1} & \frac{\partial x}{\partial \theta_2} & \frac{\partial x}{\partial \theta_3} & \frac{\partial x}{\partial \theta_4} & \frac{\partial x}{\partial \theta_5} \\ \frac{\partial y}{\partial \theta_1} & \frac{\partial y}{\partial \theta_2} & \frac{\partial y}{\partial \theta_3} & \frac{\partial y}{\partial \theta_4} & \frac{\partial y}{\partial \theta_5} \\ \frac{\partial z}{\partial \theta_1} & \frac{\partial z}{\partial \theta_2} & \frac{\partial z}{\partial \theta_3} & \frac{\partial z}{\partial \theta_4} & \frac{\partial z}{\partial \theta_5} \end{bmatrix}$$

Procedure to find null space of jacobian:

- Row reduce jacobian
- Form vectors based on RREF of Jacobian
- Find nullspace of current angle by substituting in theta values

$$\mathrm{RREF}(J(\theta)) = \begin{bmatrix} 1 & 0 & 0 & a(\theta) & b(\theta) \\ 0 & 1 & 0 & c(\theta) & d(\theta) \\ 0 & 0 & 1 & e(\theta) & f(\theta) \end{bmatrix}$$

$$\mathrm{null}(J(\theta)) = \mathrm{span}\left\{ \begin{bmatrix} -a(\theta) \\ -c(\theta) \\ -e(\theta) \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} -b(\theta) \\ -d(\theta) \\ -f(\theta) \\ 0 \\ 1 \end{bmatrix} \right\}$$

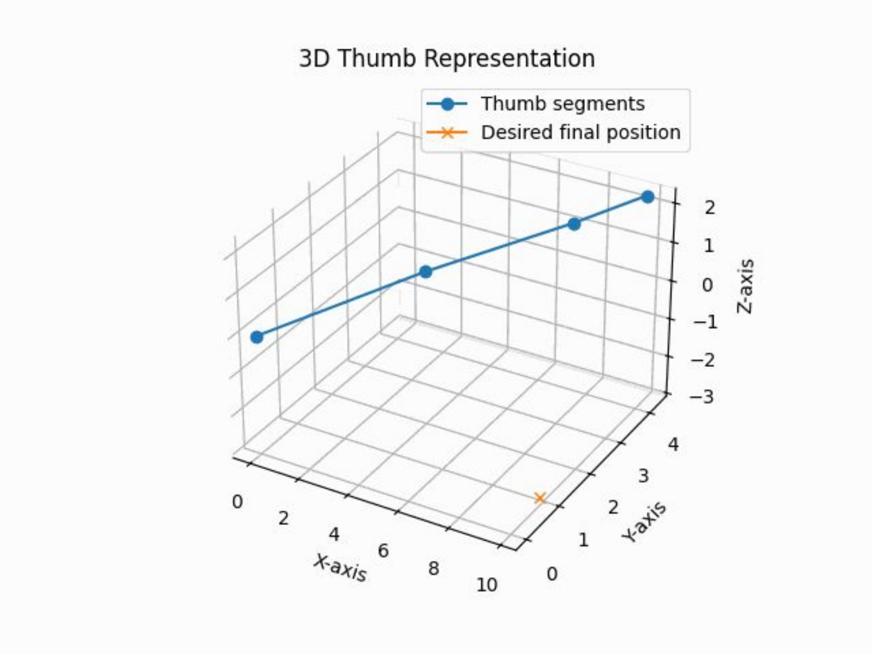
Searching the Nullspace

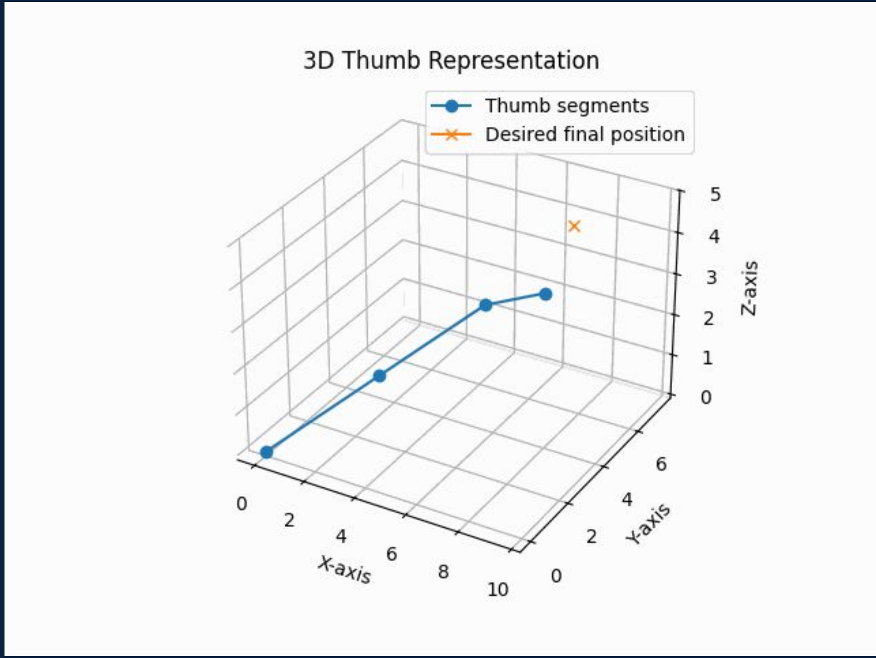
```python
def optimize_theta(theta_vals,theta_past):
    # Using span of vectors in nullspace retroactively determine a set of solution thets that would minimize magnitude delta theta move
    # MINI OPTIMIZATION PROBLEM
    # MINIMIZE THE FUNCTION OF 2 VARIABLES WHICH IS THE SPAN OF THE NULLSPACE MINUS THE PAST THETA VALS -> BEST
    # N1 and N2 are numeric not symbolic
    # mag_delta_theta_move = (theta_vals-theta_past).norm()

    def penalize_illegal_angles(updated_theta): ···

    J = numerical_jacobian(theta_vals)
    null = scipy.linalg.null_space(J)

    # Objective function to minimize
    def objective(c):
        c1, c2 = c  # Coefficients
        delta_theta = (c1 * null[:, 0]) + (c2 * null[:, 1])
        updated_theta = theta_vals + delta_theta

        # Core objective: minimize the magnitude of delta_theta
        core_objective = np.dot(updated_theta - theta_past, updated_theta - theta_past)

        # Add penalty terms for constraints
        constraint_penalty = 100  # Scaling factor for penalties

        # Total objective: core objective + penalties
        return core_objective + (constraint_penalty*penalize_illegal_angles(updated_theta))

    # Initial guess for coefficients
    initial_guess = [0, 0]

    # Minimize the objective function
    optimal_nullspace_parameters = scipy.optimize.minimize(objective, initial_guess, method='L-BFGS-B', jac='2-point', options=dict(maxfu
    s_opt, t_opt = optimal_nullspace_parameters.x

    # linear combination of nullspace vectors that minimizes delta_theta
    theta_opt = theta_vals + (s_opt * null[:, 0]) + (t_opt * null[:, 1])

    return theta_opt
```

Objective Snippet

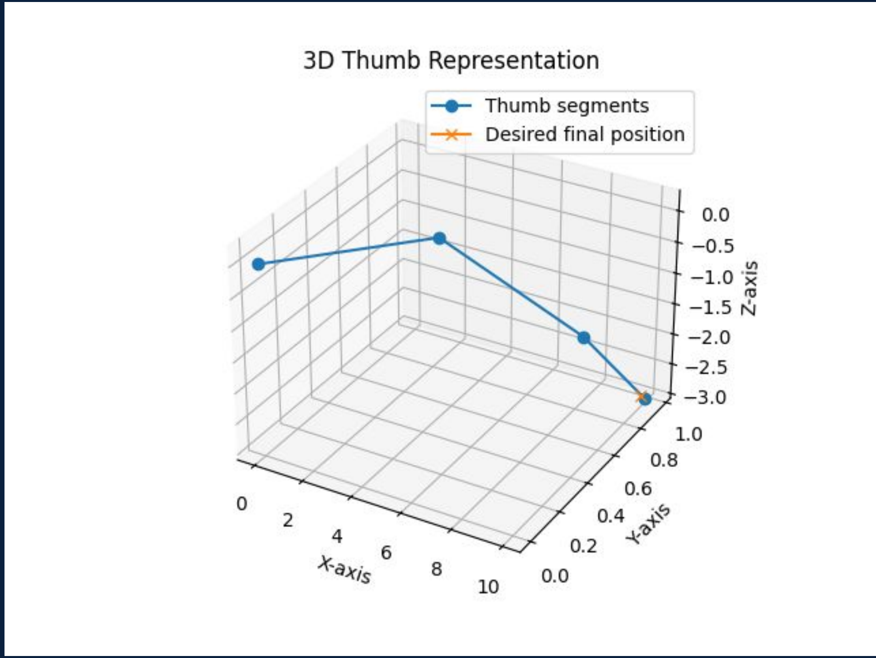
- Define span of the nullspace of the jacobian
- Subtracting the theta values (θ_{n+1}) that correspond to the thumb at $\mathbf{q_n}$.
- Enforce joint constraints using lagrangian penalties
    - `penalize_illegal_angles` (θ_{n+1})

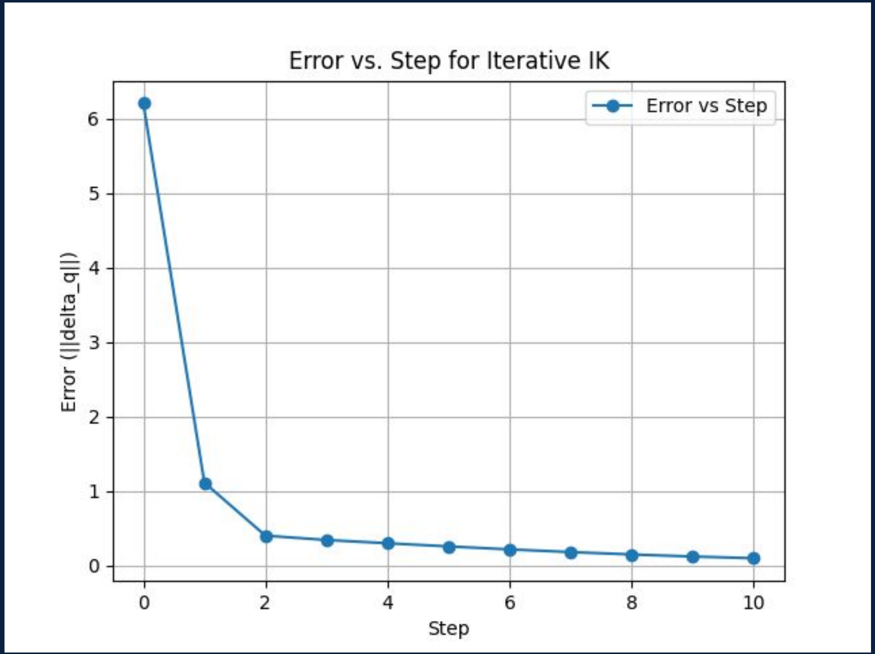
Results


3D Thumb Representation
Thumb segments
Desired final position
X-axis
Y-axis
Z-axis
0
2
4
6
8
10
0
1
2
3
4
2
1
0
−1
−2
−3

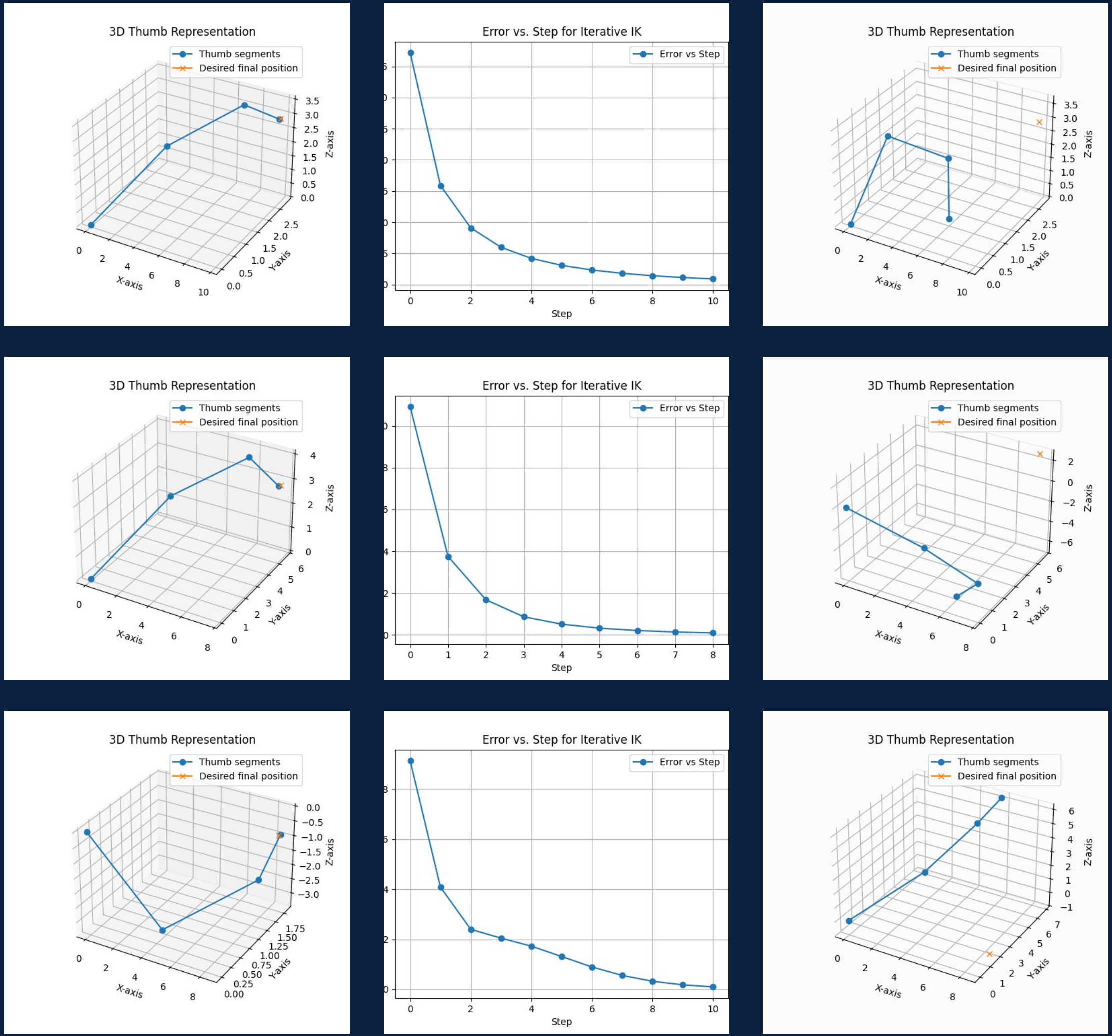

3D Thumb Representation
Thumb segments
Desired final position
X-axis
Y-axis
Z-axis
0
2
4
6
8
10
0
2
4
6
0
1
2
3
4
5

Results

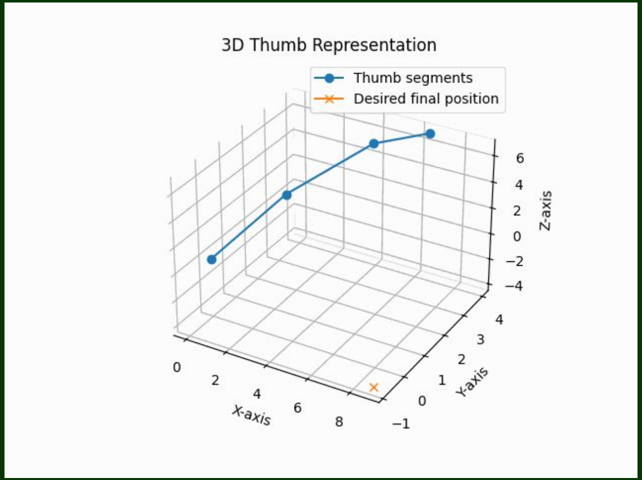

3D Thumb Representation
Thumb segments
Desired final position
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
−3.0
Z-axis
1.0
0.8
0.6
0.4
0.2
0.0
Y-axis
0
2
4
6
8
10
X-axis


Error vs. Step for Iterative IK
Error vs Step
6
5
4
3
2
1
0
Error (||delta_q||)
0
2
4
6
8
10
Step

Results


3D Thumb Representation
Thumb segments
Desired final position
X-axis
Y-axis
Z-axis
Error vs. Step for Iterative IK
Error vs Step
Step
3D Thumb Representation
Thumb segments
Desired final position
X-axis
Y-axis
Z-axis
3D Thumb Representation
Thumb segments
Desired final position
X-axis
Y-axis
Z-axis
Error vs. Step for Iterative IK
Error vs Step
Step
3D Thumb Representation
Thumb segments
Desired final position
X-axis
Y-axis
Z-axis
3D Thumb Representation
Thumb segments
Desired final position
X-axis
Y-axis
Z-axis
Error vs. Step for Iterative IK
Error vs Step
Step
3D Thumb Representation
Thumb segments
Desired final position
X-axis
Y-axis
Z-axis

Results vs. Initial Objectives



- ✅ Simulate natural thumb movement
  - Thumb moves smoothly towards endpoints while adhering to restraints
- ✅ Use forward and iterative inverse kinematics
- ✅ Optimize naive iterative kinematic step to achieve optimal movement
  - Null space traversal correctly adjusts to optimal theta for a given q

Limitations

- Using the nullspace to determine the optimal $\Delta\Theta$ is based on a linearized, first-order approximation → could achieve better accuracy!
- Theoretically, the model could simulate impossible thumb movements due to Lagrangian multiplier.
- Computationally expensive

Future Improvements

- Implement pathfinding procedure into real robotic thumb
- Add/remove degrees of freedom for other arm types
- Implement higher order approximations (utilize Hessian)
- Add functionality to simulate natural human thumb movement
- Implement other minimization methods
- Determine jacobian/nullspace analytically for more exact solution

Acknowledgements

- The thumb's range of motion
  https://pmc.ncbi.nlm.nih.gov/articles/PMC3653006/#:~:text=The%20Metacarpophalangeal%20Joint,range%200%E2%80%9315%20%C2%B0

Thank you to Professor Zucker, Professor Masroor, and Professor Towles for their help and guidance throughout this project.