

Lab 3: Optimization Inverse Kinematics

MEAM 5200

Bruke Baraki, Hongyi Lai

Methods

Inverse Kinematics with Optimization

The goal of using optimization with inverse kinematics is to perform both the primary task (i.e moving the current end effector's pose to that of the target end effector) and the secondary task (i.e maintain joint angles close to the center of its range). The analytic representation of these two tasks are shown in *Equation 1*, which are calculated numerically by a python script, specifically in its “inverse” function.

$$\dot{\vec{q}} = \underbrace{J^+ \vec{\xi}}_{\text{Primary task}} + \underbrace{(I - J^+ J) \vec{b}}_{\text{Secondary task}} \quad \{\dot{q} \mid \mathbf{J}(q)\dot{q} = \xi\} = \{\dot{q}^* + z \mid z \in \text{null}(\mathbf{J}(q))\}$$

Eq. 1: Inverse Kinematics with Optimization

Eq. 2: Nullspace of the Jacobian

Primary Task

The primary task is calculated by the “end_effector_task” function, which is responsible for outputting the instantaneous change of joint velocity that would result in moving the end effector closer to the target pose. The current end effector pose was calculated using the current joint angles by means of the forward kinematics approach from the CalculateFK.py script. Running the current and target poses through the “displacement_and_axis” function provides the displacement vector and axis of rotation between the poses. The displacement vector and axis of rotation are used to get $\vec{\xi}$, which describes the linear and angular velocity vector the current end effector would need to attain the pose of the target. Thus, $\vec{\xi}$ is proportional to the linear and rotational displacement between the current and target pose.

Multiplying the the pseudoinverse inverse of the Jacobian by the velocity vector $\vec{\xi}$ provides $\dot{\vec{q}}$, the primary task. The “inverse” function is responsible for combining the primary task and the secondary and adjusting the joint angle accordingly with “dq”. This is done repeatedly until the success condition is met (i.e the joints are within the acceptable range and the distance/rotation between poses are negligible).

Secondary Task

Using the “joint_centering_task” function, there is a preference for joint angles closer to the center of its range. This is represented in \vec{b} , the cost function, as shown in *Equation 1*. Multiplying $(I - J^+ J)$ by \vec{b} ensures that the secondary task is within the nullspace of the Jacobian, as shown in *Equation 2*. As a result, when $\dot{\vec{q}}$ is adjusted by the secondary task, the end effector's position does not change.

Code Structure

Bruke Baraki's code was used in the experimentation. The visualization.py code was utilized not only for software testing but also hardware experiments. It is responsible for looping through various configurations, providing miscellaneous information such as time eclipsed, number of iterations, EE pose. Finding the optimized paths to these configurations were calculated by the solveIK.py script, which contains the class IK. In the “inverse” function, an initial guess and a target are its input. A while loop moves the joint angles by a “dq” until the success condition is reached, where the loop would break if the

success condition is met. The success conditions are tested in the “is_valid_solution” function where a boolean would be returned.

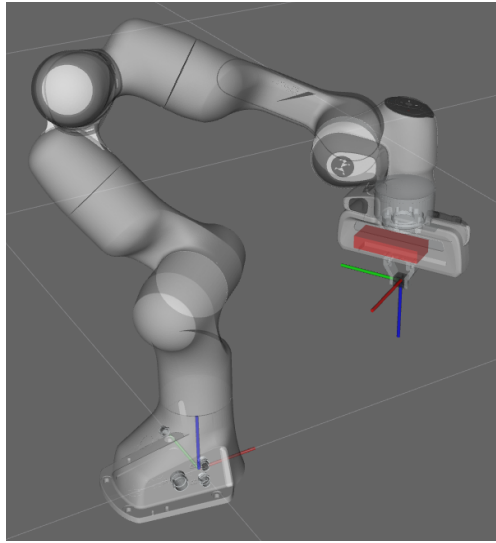
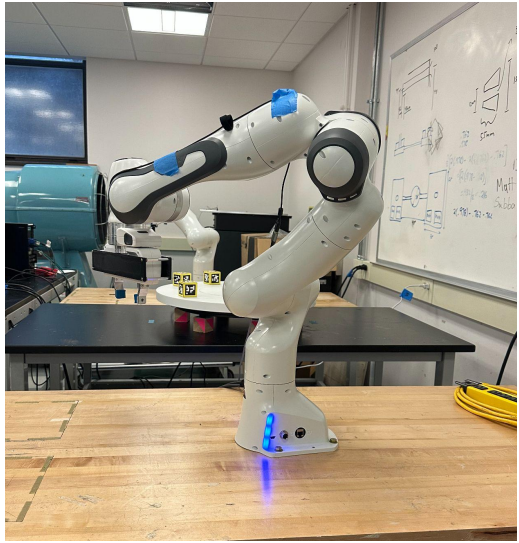
Evaluation

We followed a systematic approach to implementing and testing the analytical and numerical algorithms for inverse kinematics. To ensure that our solutions are free of bugs and that we can be confident in their correctness, we performed a thorough testing process and documented the relevant data. The end effectors were measured using a measuring tape along its principle axes, as indicated by the base.

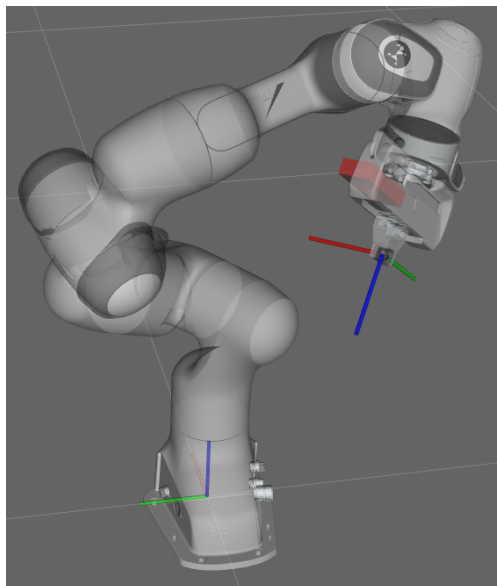
Software/ Hardware Comparison

EE Pose 1:	EE Pose 2:	EE Pose 3:
$\begin{bmatrix} -0.866 & -0.5 & 0 & 0.3 \\ -0.5 & 0.866 & 0 & -0.1 \\ 0 & 0 & -1 & 0.3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 0.75 & 0.433 & -0.5 & -0.2 \\ 0.650 & -0.625 & 0.433 & -0.15 \\ -0.125 & -0.650 & -0.75 & 0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 & 0.5 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0.7 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
EE Pose 4:	EE Pose 5:	
$\begin{bmatrix} 1 & 0 & 0 & 0.2 \\ 0 & -1 & 0 & 0.3 \\ 0 & 0 & -1 & 0.4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 0.75 & 0.433 & 0.5 & 0.15 \\ 0.65 & 0.63 & -0.4 & -0.3 \\ 0.125 & 0.65 & -0.75 & 0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	

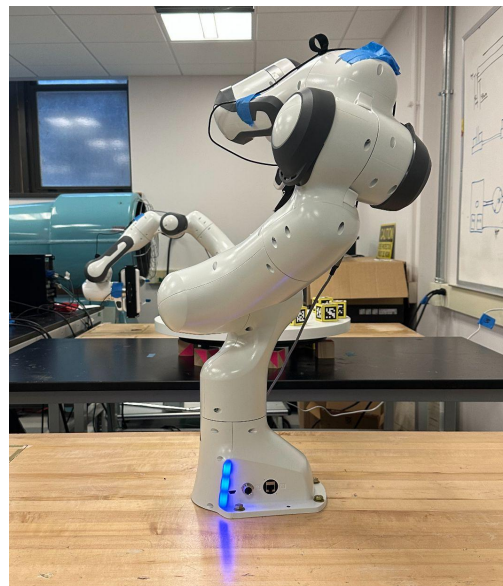
Table 1: Test Poses for the Inverse Kinematics Solver

	Software	Experiment
Pose 1	 <p>Calculated EE Translation: [0.3, -0.1, 0.3]</p>	 <p>Measured EE Translation: [0.33, -0.13, 0.30]</p>

Pose 2

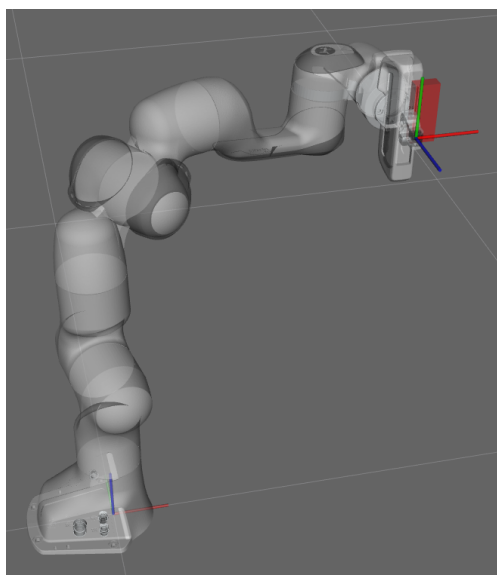


Calculated EE Translation: $[-0.2, -0.15, 0.5]$

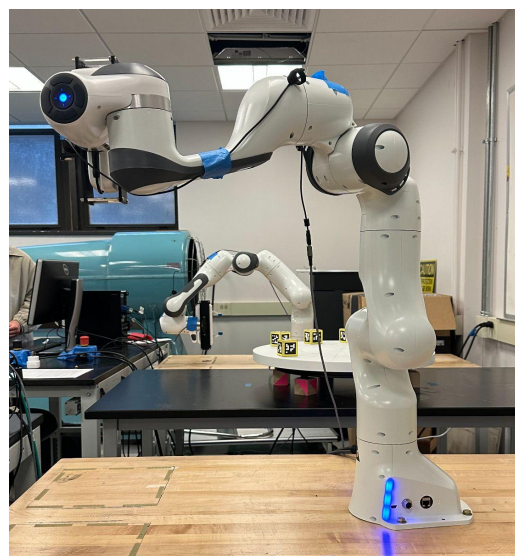


Measured EE Translation: $[-0.13, -0.17, 0.53]$

Pose 3



Calculated EE Translation: $[0.5, 0, 0.7]$



Measured EE Translation: $[0.46, 0.06, 0.69]$

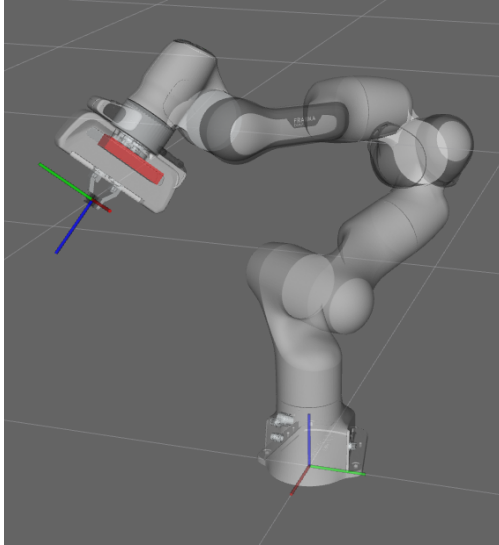
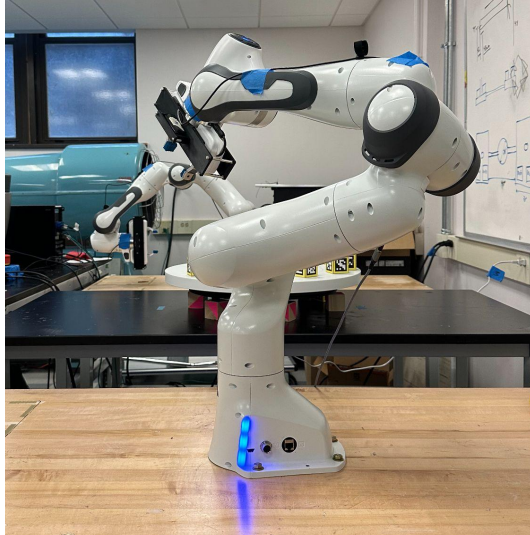
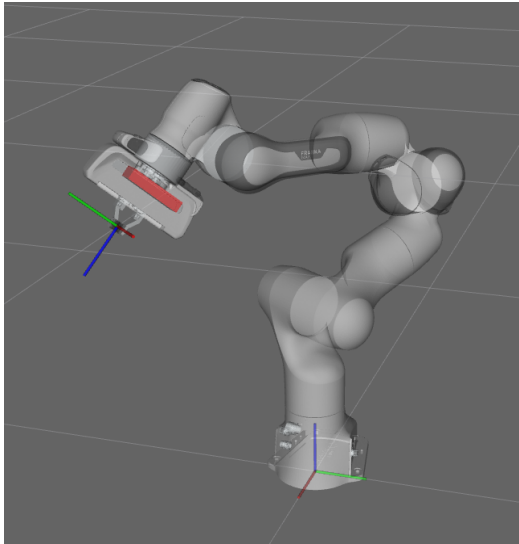
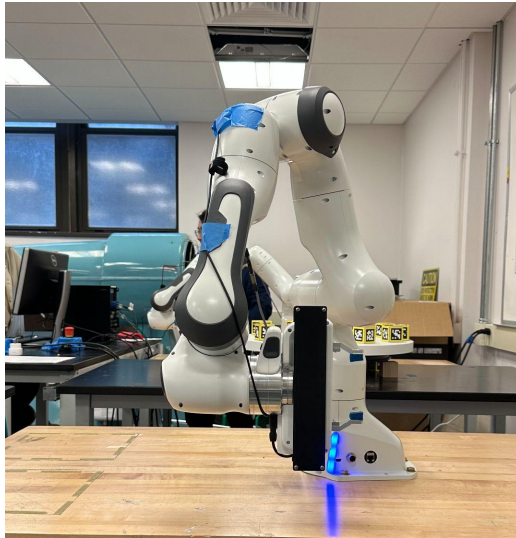
Pose 4	 <p>Calculated EE Translation: [0.2, 0.3, 0.4]</p>	 <p>Measured EE Translation: [0.22, 0.32, 0.43]</p>
Pose 5	 <p>Calculated EE Translation: [0.15, -0.3, 0.5]</p>	 <p>Measured EE Translation: [0.18, -0.29, 0.51]</p>

Table 3: Comparison of Simulation and Hardware Results with End Effector Displacement

Inverse Kinematics Solver Performance Evaluation

	Time Elapsed [sec.]	Iterations	Success Status
Pose 1	0.14	43	True
Pose 2	2.20	500	True
Pose 3	0.28	51	True
Pose 4	0.17	34	True

Pose 5	2.32	500	True
Pose 6			False
Pose 7	2.68	500	True
Pose 8			False
Pose 9			False
Pose 10			False

Table 4: Testing 10 Different Poses, Measuring Time Elapsed, Iterations, Success Status

	Mean	Median	Maximum	STD
Time Elapsed [sec.]	1.298	1.24	2.68	1.218

Table 5: Mean, Median, Maximum, and Standard Deviation for Time Elapsed to Get IK Solution

	Mean	Median	Maximum	STD
Iterations	271.33	275.5	500	250.55

Table 6: Mean, Median, Maximum, and Standard Deviation for Number of Iterations to Get IK Solution

Success Rate	60%
--------------	-----

Table 7: Success Rate for getting IK Solution.

Analysis

The primary task is computed using the *end_effector_task* function, which calculates the instantaneous change in joint velocity needed to move the end effector closer to the target pose. The secondary task is computed using the *joint_centering_task* function, which prioritizes joint angles closer to the center of their range. The inverse function combines the two tasks to determine the joint velocities.

The code that we use is a Python implementation of the IK solver, including helper functions, task functions, and the main inverse kinematics solver. The IK solver uses gradient descent to solve the full inverse kinematics of the robot manipulator. It takes a target pose and an initial guess of joint angles as input and outputs the solution, a success flag, and a rollout of the guess for each algorithm iteration.

Seed 1: [0,0,0,-pi/2,0,pi/2,pi/4]

Solution 1: [-0.53303 -0.86059 0.3201 -2.32023 0.24176 1.48797 2.05643]

Seed 2: [pi/6,0,0,-pi/2,0,pi/2,4*pi/3]

Solution 2: [-0.39268 -1.10067 -0. -1.72171 0. 0.62103 0.39272]

The two seeds provided here are the initial guesses for the inverse kinematics solver, and the corresponding solutions are the joint angles calculated by the IK solver for the desired end-effector position and orientation.

As the seeds above, the solutions are different for each seed. This discrepancy could be due to lots of reasons. For example, it might come from the robotic system, the presence of multiple solutions or it just comes from the IK solver's algorithm. In order to analyze the uniqueness of the solutions, we can test the IK solver with multiple seeds or initial guesses and see the resulting joint angle. If the IK solver consistently yields the exact solution for different initial guesses, it indicates a unique solution for the given end-effector position and orientation. However, if the solver provides different solutions for various initial guesses, there might be multiple valid joint configurations for the given end-effector pose, or the algorithm might converge to different local minima.

Algorithmic completeness is a property that describes whether an algorithm can find a solution for every possible input within a problem domain. In the case of inverse kinematics (IK) solver, algorithmic completeness means the solver can find a valid joint configuration for any given end effector pose, assuming such a configuration exists. If the algorithm fails to find a solution for some target end-effector poses, it does not necessarily mean that the robot cannot achieve that pose, nor does it imply that the algorithm was coded incorrectly. There could be several reasons for this: The target pose is outside the robot's workspace. In this case, the robot physically cannot reach the desired end effector pose due to its mechanical constraints. This is not a limitation of the algorithm but of the robot itself. Or the algorithm's convergence properties. The algorithm may have poor convergence properties, meaning it might take a very long time to find a solution or get stuck in a local minimum. This can happen if the algorithm is not well-suited for solving the problem or if the initial joint configuration guess is too far from the real solution.

In this case, we can take advantage of the target poses clustered closely in the reachable workspace with similar orientations. To decrease the total runtime/iterations necessary to compute IK solutions for all these poses, we can use a "warm start" approach, which leverages the solution of a previous pose to initialize the optimization process for the next pose. This can significantly reduce the number of iterations required to converge to a solution, as the initial guess is already close to the desired solution.

Here is the reasoning behind this approach:

1. Similar poses imply similar joint configurations: Since the target poses are close and have similar orientations, the joint configurations required to achieve these poses are likely to be similar. Thus, the IK solution for one pose can be an excellent initial guess for the IK solution of a nearby pose.
2. Faster convergence: Numerical optimization algorithms typically converge faster if the initial guess is closer to the solution. By initializing the optimization process with the IK solution from a previous pose, we provide a better starting point for the algorithm, resulting in faster convergence and fewer iterations.
3. Reduced computational effort: As the number of iterations required for convergence decreases, the overall computational effort and runtime decrease. This makes the warm start approach more efficient when solving IK solutions in similar poses and orientations scenarios.

4. Adaptive step size: Besides using a warm start, we can employ an adaptive step size in our numerical optimization algorithm. This means the step size changes depending on the error between the current guess and the desired pose. If the error is significant, the step size can be increased to make more giant steps toward the solution. Conversely, if the error is small, the step size can be reduced to make smaller steps and refine the solution. This approach further helps to speed up the convergence and reduce the total runtime.

By using a warm start approach and leveraging the IK solutions from previous poses, we can decrease the total runtime and the number of iterations necessary to compute IK solutions for multiple poses clustered close together in the reachable workspace with similar orientations. This method takes advantage of the similarity between the poses and the fact that the corresponding joint configurations are likely to be similar.

In conclusion, this lab report has presented a detailed analysis of the inverse kinematics problem for a robotic manipulator. The exploration of analytical and numerical approaches provided valuable insights into the strengths and limitations of each method. Additionally, investigating algorithmic completeness and the warm start approach has deepened our understanding of the challenges and potential solutions to improve the performance of numerical optimization algorithms for IK. Throughout the lab, it became evident that the analytical approach, while faster and more straightforward, is limited by its dependence on specific robot geometries and the potential for multiple solutions or singularities. In contrast, although slower and more computationally intensive, the numerical optimization approach offers greater flexibility and applicability to a wide range of robotic systems.

The discussion of algorithmic completeness has highlighted the importance of carefully considering the limitations of the IK algorithm and the robot's physical constraints. It is crucial to differentiate between the algorithm's inability to find a solution and the robot's inability to achieve a specific end effector pose. This awareness helps to prevent incorrect conclusions about the robot's capabilities or the correctness of the implemented algorithm. Lastly, exploring the warm start approach demonstrated how the total runtime and iterations required to compute IK solutions for multiple similar poses can be decreased. By taking advantage of the similarity between poses, we can improve the efficiency of the algorithm without compromising the quality of the solutions.