Grasping

Ziteng (Ender) Ji   Yuvan Sharma

***Abstract*—In this project, we implemented a multifingered grasping pipeline using an Allegro hand mounted on a Sawyer robot in MuJoCo. We developed a Levenberg–Marquardt inverse kinematics (IK) solver to control all four fingers simultaneously and used a grasp synthesis algorithm based on Q+ and Q− metrics to achieve force closure grasps. The grasping process involved detecting contact, optimizing wrench closure, and lifting the object. For our extension, we demonstrated the ability to pick up a spherical object and place it into a container, and grasping a cube. This project includes dexterous manipulation, integrating theoretical concepts such as grasp matrices, friction cone discretization, and force closure with practical simulation-based control.**

I. Introduction

Grasping remains a fundamental challenge in robotics due to the complexity of perception, contact dynamics, and multi-joint coordination. This project explores multifingered grasping with an Allegro hand mounted on a Sawyer arm using the MuJoCo physics engine. The goal is to simulate dexterous manipulation by combining classical grasping theory with practical algorithmic implementations.

We begin by developing an inverse kinematics (IK) solver using the Levenberg-Marquardt method to control all four fingers of the hand simultaneously. We then implement a grasp synthesis pipeline based on force closure analysis, evaluating grasps using Q+ and Q− metrics to guide contact optimization and ensure object stability. As an extension, we demonstrate a complete pick-and-place pipeline in which the robot grasps a spherical object and places it into a container.

Finally, we provide experimental results that evaluate each component of our system, including IK performance, grasp stability, and extension results.

II. Method

A. IK Solver

In this project, we implemented a multifingered inverse kinematics (IK) solver using the Levenberg–Marquardt (LM) method, a standard numerical optimization technique for solving nonlinear least squares problems. The goal of inverse kinematics is to compute a joint configuration $\mathbf{q}_h \in \mathbb{R}^{16}$ for the Allegro hand such that the end-effector poses—that is, the positions and orientations of the palm and each fingertip—match given target poses in 3D space. Because the Allegro hand contains many joints and multiple independent end-effectors (the palm and four fingers), we require an optimization-based solver capable of handling this complexity. The LM update equation is

$$\mathbf{q}_{t+1} = \mathbf{q}_t + \left(J^\top J + \lambda I\right)^{-1} J^\top (\mathbf{x}_d - \mathbf{x}_t)$$

where $\mathbf{q}_t$ is the joint configuration at iteration t, $\mathbf{x}_d$ is the desired task-space pose (position and orientation), $\mathbf{x}_t$ is the current pose computed via forward kinematics, J is the Jacobian matrix of the hand with respect to the joints, λ is a damping coefficient to handle singularities, and I is the identity matrix.

To adapt this formulation for multiple fingers, we concatenate both the Jacobians and the error vectors for all five bodies (the palm and four fingertips). For each body i, we compute a 6-dimensional error vector consisting of 3 positional and 3 rotational components. The rotational error is derived from the difference between the desired and current orientation quaternions using quaternion multiplication. Specifically, we compute the relative quaternion $q_{\text{err}} = q_d \cdot q_c^{-1}$ and extract its vector part to approximate the orientation error. These per-body errors are stacked to form a global task-space error vector $\mathbf{e} \in \mathbb{R}^{30}$ across all 5 bodies.

Similarly, for each body, we compute a Jacobian matrix $J_i \in \mathbb{R}^{6\times16}$, composed of linear and angular components (from MuJoCo's `mj_jacBody` function). These are vertically stacked to form a global Jacobian $J \in \mathbb{R}^{30\times16}$. Once we have the global error vector and global Jacobian, we apply the LM update formula to compute a joint update $\Delta\mathbf{q}$, which is then added to the current joint configuration. After each update, we clip the joint values to be within valid ranges using MuJoCo's joint limits. This procedure is repeated iteratively until the error $\|\mathbf{x}_d - \mathbf{x}_t\|$ falls below a small threshold, indicating convergence to a solution.

This strategy allows us to simultaneously control multiple end-effectors by treating their constraints as part of a single optimization problem. Rather than solving IK for each fingertip independently—which would lead to inconsistencies and infeasibility due to joint coupling—we construct a unified error vector and Jacobian that considers all fingertip and palm constraints together. This results in smooth and coordinated joint motions across the whole hand. The algorithm terminates when the overall task-space error is sufficiently small or a maximum number of iterations is reached, ensuring stability and efficiency in simulation.

B. Grasping Algorithm

To synthesize a robust, force-closure grasp, we implemented the grasping pipeline described in Section 1.5 of the project document. The core idea is to iteratively optimize the Allegro hand's joint configuration so that the resulting contact wrenches enclose the origin in wrench space, thereby satisfying the conditions for force closure. This is achieved through a two-stage optimization procedure that evaluates both a necessary and a sufficient condition.

The algorithm begins by bringing the hand near the object using a predefined palm pose. From there, the fingers iteratively descend and attempt to make contact. Contact is evaluated by computing the Euclidean distance from each fingertip to the surface of the object; contact is considered established once all distances fall below a small threshold. While contact has not been made, the optimization objective is solely based on the squared distance from fingertips to the surface of the object:

$$D = \sum_{i=1}^{4} d_i^2$$

where d_i is the Euclidean distance from fingertip i to the closest point on the object surface. This term encourages the hand to converge toward the object.

Once contact is established, the algorithm builds a grasp matrix $G \in \mathbb{R}^{6\times kn}$ by discretizing the friction cone at each contact point. The friction cone is approximated using k evenly spaced vectors d_i^j within the cone aperture, as described in Section 1.3.1 of the document. Each contact point c_i generates k wrenches $w_i^j \in \mathbb{R}^6$, defined as:

$$w_i^j = \begin{bmatrix} d_i^j \\ (c_i - o) \times d_i^j \end{bmatrix}$$

Here, c_i is the contact position in world coordinates, and o is the object center. The full grasp matrix G is constructed by horizontally stacking these w_i^j across all contact points.

The first optimization step evaluates a necessary condition for force closure by computing the Q$^+$ score. The goal is to minimize the distance from the origin of wrench space to the set of allowable wrenches generated by nonnegative combinations of G, while also considering the fingertip-object distance:

$$d_{Q^+}^2(0, W) = \min_{\alpha \geq 0} \|G\alpha\|_2^2 + \beta D$$

This is implemented using a quadratic program in `optimize_fc_loss_qp()` where $\alpha \in \mathbb{R}^{kn}$ is constrained to be nonnegative, and β is a hyperparameter balancing grasp quality and proximity. The Q$^+$ score serves as a continuous surrogate for testing whether $0 \in \text{co}(W)$, which is necessary but not sufficient for force closure.

If the Q$^+$ score falls below a small threshold (we found 10^{-6} to work best in our setting), we proceed to evaluate the sufficient condition using Q$^-$, which tests whether the origin lies in the interior of the convex hull of the grasp wrenches. This condition is satisfied if the largest ball around the origin is entirely contained within the feasible wrench set. Mathematically, this is expressed as:

$$d_{Q^-}(k) = \min -r \quad \text{s.t.} \quad rq_k = \sum_{i=1}^{N} \alpha_i g_i$$

$$\sum \alpha_i = 1, \quad \alpha_i \geq 0, \quad r \geq 0$$

where $q_k \in \mathbb{R}^6$ is a sampled unit vector in wrench space and g_i is a column of G. We solve this linear program across multiple directions q_k and define:

$$d_{Q^-}(G) = \max_k d_{Q^-}(k)$$

If $d_{Q^-}(G) < 0$, then the grasp achieves full force closure. The entire optimization is performed in joint space using gradient descent. The objective function is defined piecewise based on whether contact has been made:

$$f(q_h) = \begin{cases} \beta D & \text{if no contact} \\ d_{Q^+}^2(0, W) + \beta D & \text{if Q}^+ > \text{threshold} \\ d_{Q^-}(G) + \beta D & \text{if Q}^+ \leq \text{threshold} \end{cases}$$

To minimize this objective, the joint configuration is updated iteratively:

$$q_h^{\text{new}} = q_h - \lambda \nabla_{q_h} f(q_h)$$

The gradient $\nabla_{q_h} f(q_h)$ is computed using finite differences, as implemented in the function `numeric_gradient()`. Each iteration applies the updated joint angles to the MuJoCo simulation, reconstructs the scene, re-evaluates contact, and recomputes the objective.

The algorithm terminates either when the maximum number of iterations is reached or when the grasp achieves the desired force closure condition.

C. Extensions

1) Grasp Cube: To extend our grasp synthesis pipeline beyond spherical objects, we adapted our simulation environment to support grasping a cube. This required no modification to the grasping algorithm itself, which is designed to be shape-agnostic and generalizes to arbitrary object geometries. The extension primarily involved updating the scene configuration and reusing the same inverse kinematics and grasp synthesis logic described earlier.

We initialized the environment using `AllegroHandEnv(object_type="cube")`, which triggered the placement of a cube into the simulation. Internally, the environment constructs the cube using a MuJoCo primitive with box geometry of dimensions $0.04 \times 0.04 \times 0.04$ m. The cube was positioned near the origin to match the palm's default pregrasp pose.

The grasp synthesis pipeline began with the Allegro hand positioned in a flat configuration above the object. Using the environment's utility functions, we computed fingertip-to-surface distances and projected contact normals at each fingertip. Unlike a sphere, where contact normals always point radially outward, the cube introduces discontinuities at edges and vertices. The surface normal at each contact point c_i was computed based on the face geometry of the cube and used to construct the discrete friction cone.

For each contact point i, we discretized the friction cone into k vectors $\{d_i^j\}_{j=1}^k$ and generated corresponding wrenches:

$$w_i^j = \begin{bmatrix} d_i^j \\ (c_i - o) \times d_i^j \end{bmatrix}$$

where o denotes the cube's center of mass. These wrenches were aggregated into a grasp matrix $G \in \mathbb{R}^{6\times kn}$, which was used to evaluate force closure via the Q^+ and Q^- metrics, as in the original sphere case.

The loss function remained:

$$f(q_h) = \begin{cases} \beta D & \text{if contact not made} \\ \min_{\alpha \geq 0} \|G\alpha\|^2 + \beta D & \text{if contact made } (\mathrm{Q}^+) \\ \max_k d_{Q^-}(k) + \beta D & \text{if } \mathrm{Q}^+ \text{ below threshold } (\mathrm{Q}^-) \end{cases}$$

where $D = \sum_{i=1}^{4} \|c_i - \text{surface}_i\|^2$ is the sum of squared fingertip-to-surface distances, and β is a tunable weight.

Because the optimization loop and wrench computation are generic, the grasp synthesis algorithm required no modification. It operated identically to the sphere case, updating the joint configuration of the hand via:

$$q_h^{\text{new}} = q_h - \lambda \nabla_{q_h} f(q_h)$$

where gradients were computed using finite differences. After convergence, the resulting grasp configuration was executed in simulation using interpolation. The success of the grasp was visually and quantitatively evaluated based on whether the cube was securely lifted without slipping or tipping, despite its flat surfaces and sharp edges.

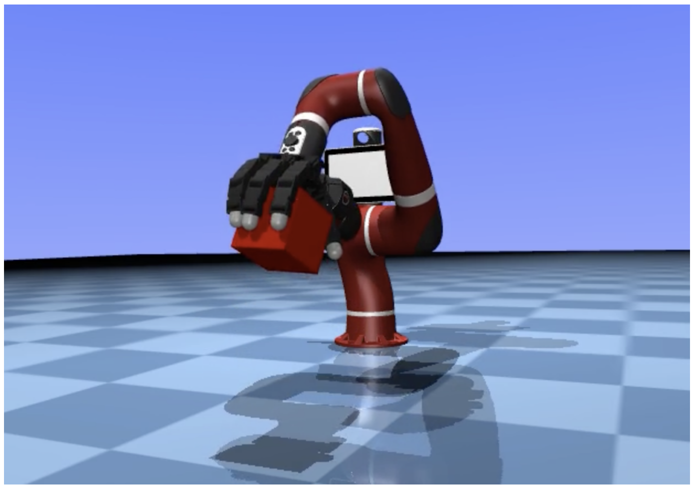

Fig. 1. Grasping cube task in Mujoco

2) Extra Credit: Pick & Place into Container: We also completed another extension task, where the robot picks up the ball, and places it onto a container. The grasp was again found using grasp synthesis as in Task 2, and the robot was then further controlled to move above the container and open the fingers to gently place the ball. The images depicting this task can be found in Figures 20- 23.

III. Results

1) IK Solver Result: In the figure below we show the result of solving IK on the values provided in task 1.

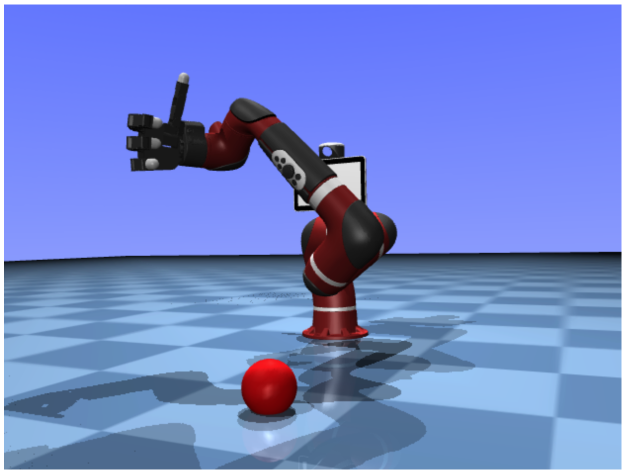

Fig. 2. Result of solving IK on the values provided in task 1

We also generated four equidistant grasp points on the equator of the sphere and used IK to set the fingers to those grasp points.

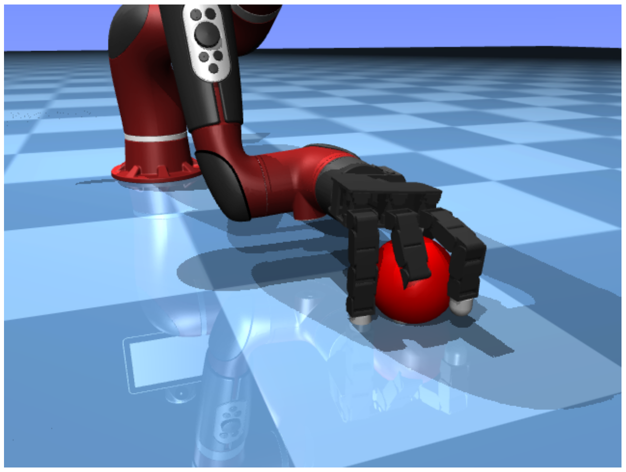

Fig. 3. Four grasp points

This method successfully positioned the fingers to achieve contact, as shown in the image provided. The grasp configuration resulted in a theoretically stable hold on the object, demonstrating that the IK solver is capable of reaching target poses derived from simple geometric reasoning. However, the collisions seen in the image are likely a result of a quirk with the simulator, which leads to uncertainty in whether the grasp is actually stable. While this approach did partially in our case, it is not inherently robust across different object shapes or orientations. It assumes uniform curvature and known geometry, which may not generalize well to objects with complex surfaces or occlusions. Nonetheless, for symmetric objects like a sphere, this method provides a straightforward and effective way to generate initial grasp poses.

2) Grasp Planner Loss Curve: Here we provide the loss curve for the objective over time for grasp planner.

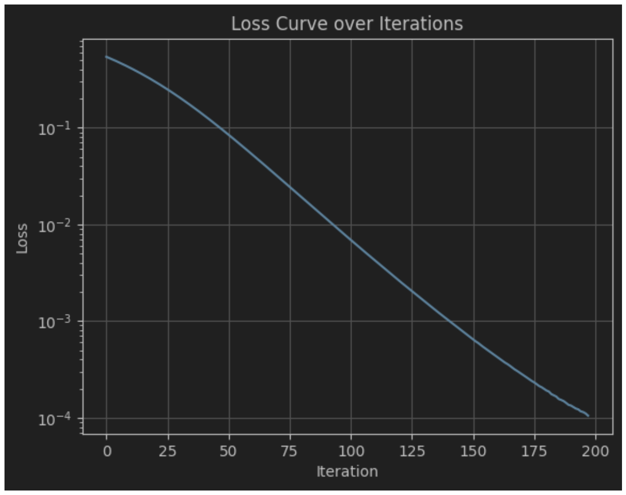

Fig. 4. Loss vs. Iteration Count for the Grasp Planner. The loss here is shown for starting height of 0.14 m.

3) Stages of Grasping: We show four stages of grasping for the ball: before the grasp (Figure 5), after fingers made contact (Figure 6), after adjusting for force closure (Figure 7), and final stage after lifting up the object (Figure 8).

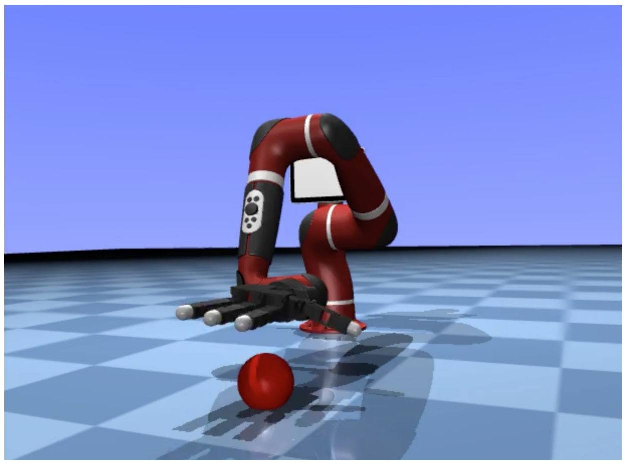

Fig. 5. Before the grasp

4) Q+, Q− Distance Table: The metrics for different grasping heights are presented in Table I. It was observed both from the quantitative results and the qualitative evaluation of different grasps that a larger Q+ distance corresponds to a more stable grasp, and a smaller (more negative) Q- distance similarly corresponds to a more stable grasp. This can also be seen from the results for the starting height 0.19 m: the Q-distance is very close to 0, the Q+ distance is small, and the grasp did not succeed.

5) Grasping Plots: For the grasping plots, we present the Contact Force vs. Time and Number of Contacts vs. Time graphs for two initial palm heights: 0.14 m and 0.19 m. The contact force graphs are presented in Figure 9 and Figure 10, and the number of contacts graphs are presented in Figure 11 and Figure 12.

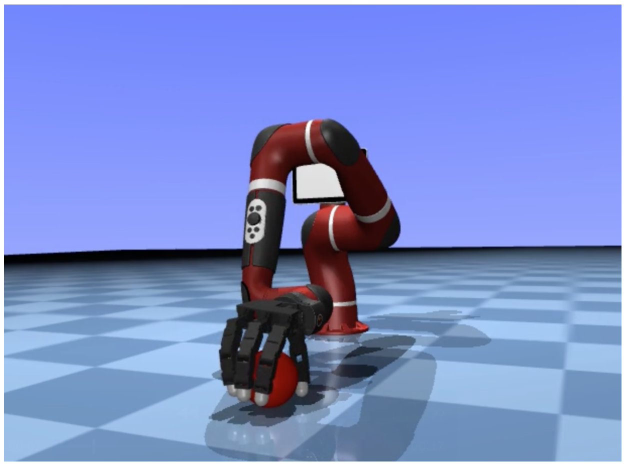

Fig. 6. After fingers made contact

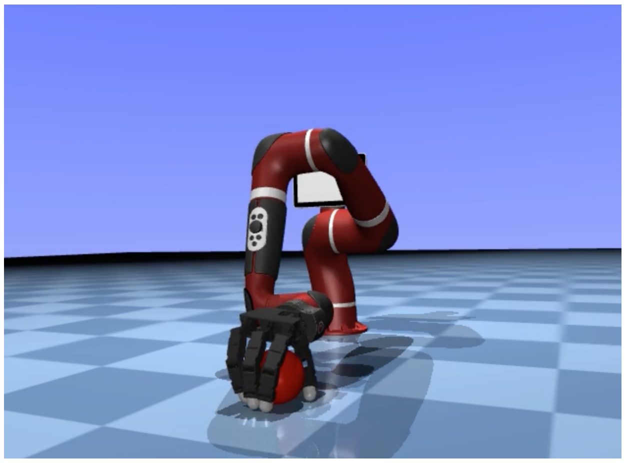

Fig. 7. After adjusting for force closure

6) Extension: For our extension tasks, we recorded the Contact Force vs. Time and Number of Contacts vs. Time graphs. For the container task, these results are presented in Figure 13 and Figure 14 in Appendix. For the cube task, these results are presented in Figure 15 and Figure 16 in the Appendix. We also provide images for the different stages of these extension tasks in the Appendix.

IV. Discussion

Overall, we found that the Levenberg–Marquardt inverse kinematics solver works well. It was observed both while completing the given tasks and general testing that the solver is able to reach target configurations to a good approximation, given enough iterations. However, we found that trying to use inverse kinematics to grasp the ball was not a good strategy. This was because IK is focused more on trying to reach a target configuration and gives no importance to the stability of a grasp. As a result, unless one somehow has the exact

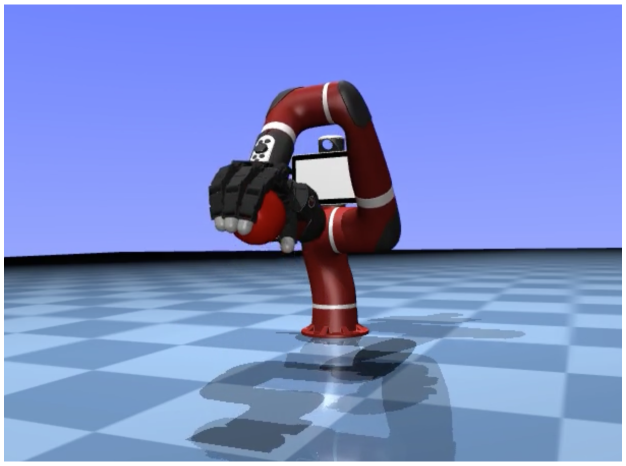

Fig. 8. Final stage

Start Height	Q+ Distance	Q- Distance	Obj. Value	Success
0.09	8.70×10^{-6}	-1.7×10^{-2}	2.9×10^{-3}	✔
0.12	9.04×10^{-6}	-7×10^{-3}	8.87×10^{-5}	✔
0.13	8.80×10^{-6}	-1×10^{-2}	7.14×10^{-5}	✔
0.14	9.83×10^{-6}	-1.1×10^{-2}	1×10^{-4}	✔
0.17	8.58×10^{-6}	-4×10^{-3}	1.6×10^{-4}	✔
0.19	3.97×10^{-6}	-3×10^{-4}	2.6×10^{-3}	✘

TABLE I
COMPARISON OF Q+, Q- AND OBJECTIVE VALUE METRICS FOR DIFFERENT PRE-GRASP HEIGHTS. ALL HEIGHTS ARE GIVEN IN METERS.

target configuration that gives a stable grasp, IK is ineffective since it is difficult to accurately create a stable grasp to give as input. In addition, we found that using IK did not always result in collisions being respected, although this is more of an issue with the MuJoCo simulator than the approach itself.

On the other hand, the grasp synthesis algorithm worked well, generating a stable grasp that we were able to complete a pick and place task with. In addition, the algorithm's performance was robust to changes in height (as seen in Table I), and also transferred over to a different shape in the form of a cube. The main issue we encountered with grasp synthesis was that while trying to execute the grasp with a control loop, the Allegro hand would jerk downwards during the first step of the control loop. Upon debugging this issue, we found that this is because the joint angles of the Sawyer robot change suddenly during the first timestep, likely due to some sort of initialization process in the simulator. To fix this issue, we started the hand at a slightly higher height than what was used during grasp synthesis to compensate for the downward movement. This movement also explains the sudden spikes in all the contact forces graphs presented, as every time we switched from one motion to another, the jerk would occur.

One of the major challenges we encountered during this project arose while implementing the extension task of picking up a ball and placing it into a container. While our grasp synthesis algorithm was able to reliably generate stable grasps around the ball, issues emerged when transitioning to the placement phase. Specifically, after successfully lifting the object, the hand would abruptly "teleport" to the target location above the container rather than moving there through a smooth, interpolated trajectory. This was initially puzzling, as it appeared that our control loop was not executing as intended.

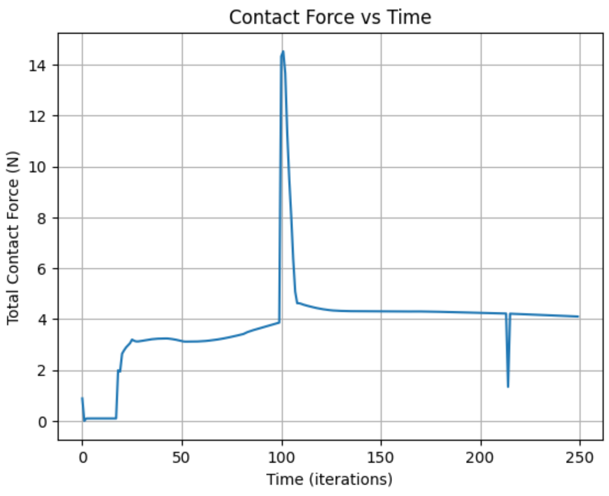

Fig. 9. 0.14 m: Contact Force vs. Time

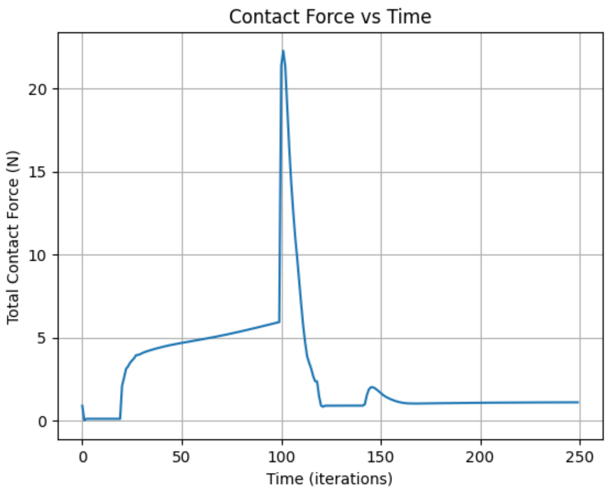

Fig. 10. 0.19 m: Contact Force vs. Time

Upon deeper inspection, we discovered that the issue stemmed from how the inverse kinematics (IK) results were being applied. The problem was that when we called the IK solver to determine the final placement pose, it immediately updated the environment's simulation state to the solution pose. As a result, the control loop intended to animate the transition from the grasp to the target was effectively skipped, because the object was already at its goal before the motion sequence began. Our solution was to first compute all IK results offline, store them as target poses, and then reset the simulation to its pre-motion state. From there, we executed a single, continuous control loop that incrementally interpolated the joint positions to the desired targets. This approach preserved simulation continuity and ensured the arm moved naturally during the placement phase.

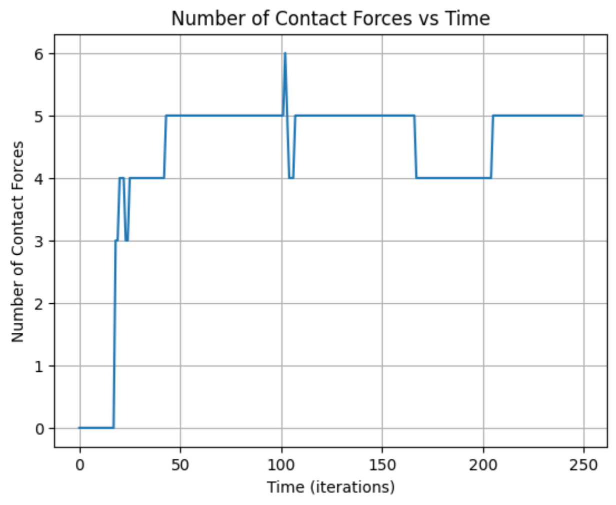

Fig. 11. 0.14 m: Number of Contacts vs. Time

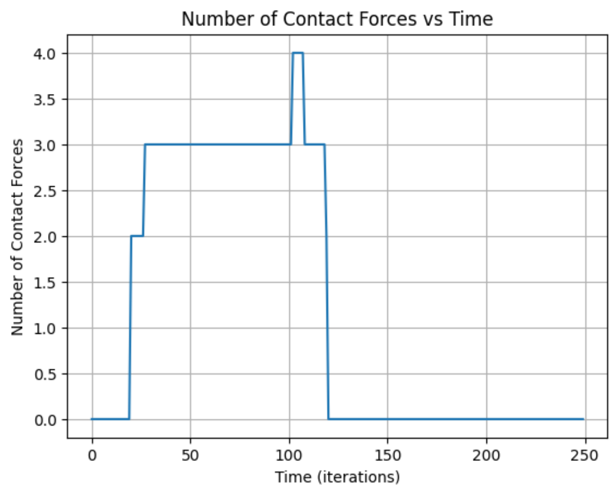

Fig. 12. 0.19 m: Number of Contacts vs. Time

Looking ahead, this project opens several directions for future work. One natural extension is to explore model-less grasping algorithms, which do not rely on known object geometries but instead use real-time sensor data, such as vision or tactile feedback, to infer object shape and plan grasps. This would make the grasping pipeline more robust to unstructured or dynamic environments. Another direction is the integration of learning-based methods, such as reinforcement learning or generative models, to learn grasp strategies directly from experience or data. These methods could generalize across object categories and adapt to new scenarios. Additionally, incorporating dual-arm grasping [1] or soft contact modeling could enable more complex and compliant manipulation tasks. Together, these directions would build on the core techniques implemented in this project and further bridge the gap between theory and real-world robotic dexterity.

V. Bibliography

[1] Shaw, Kenneth, et al. ”Bimanual dexterity for complex tasks.” arXiv preprint arXiv:2411.13677 (2024).

VI. Appendix

Our videos for different tasks can be found here: https://drive.google.com/drive/folders/1JXBwePOtgJWTf3OBbdBdIhhEl5cO7etv?usp=sharing.

Our github repository can be found here: https://github.berkeley.edu/yuvan/project4.

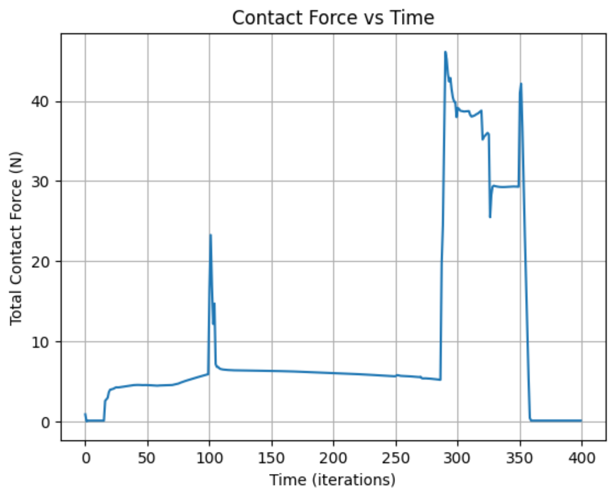

Fig. 13. Container Task: Contact Force vs. Time

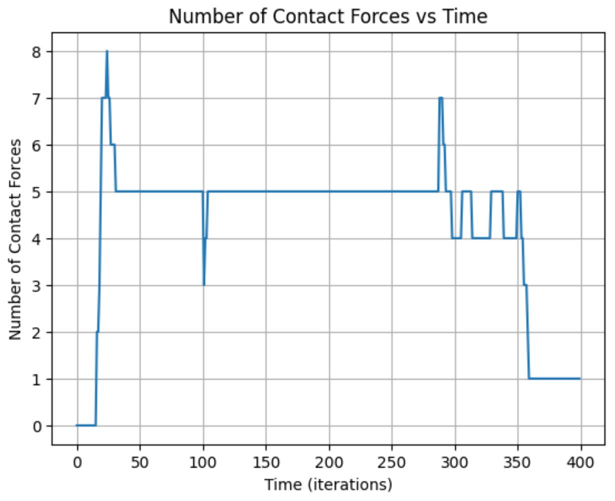

Fig. 14. Container Task: Number of Contacts vs. Time

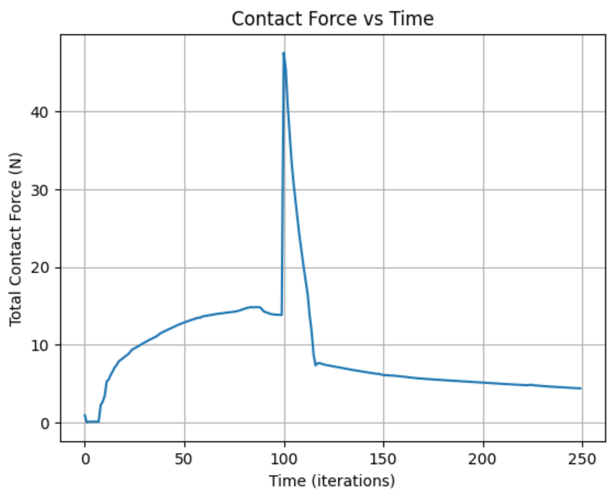

Fig. 15. Cube Task: Contact Force vs. Time

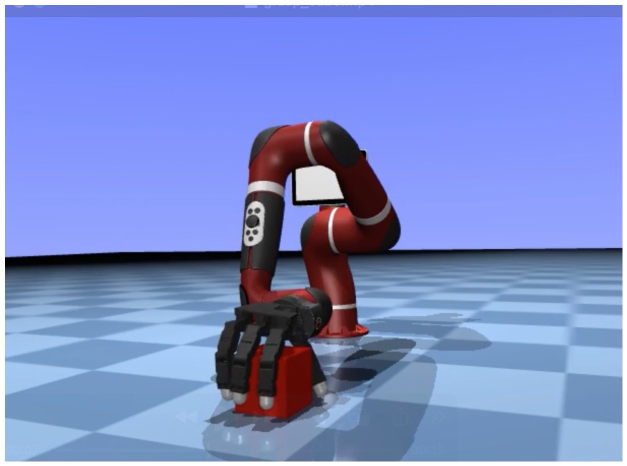

Fig. 18. Cube: after fingers made contact

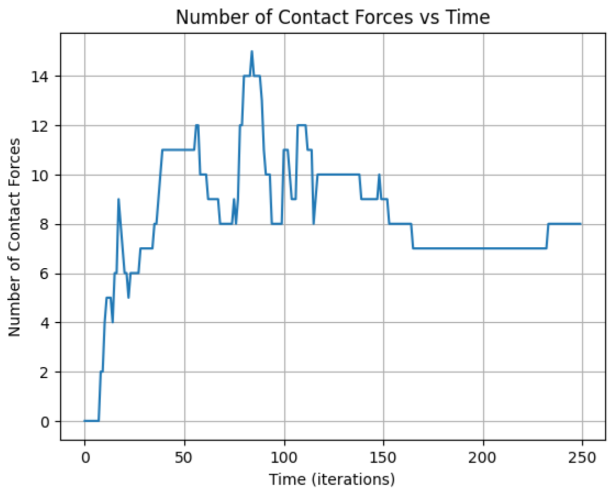

Fig. 16. Cube Task: Number of Contacts vs. Time

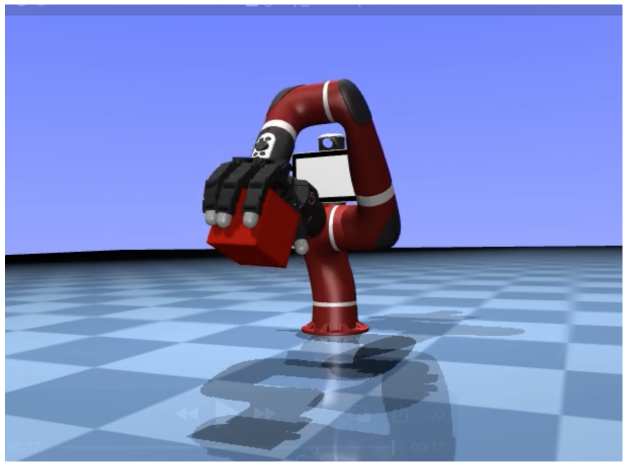

Fig. 19. Cube: lifting up

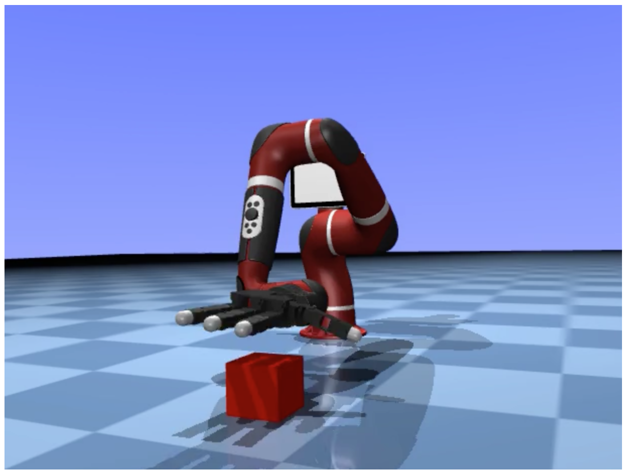

Fig. 17. Cube: before the grasp

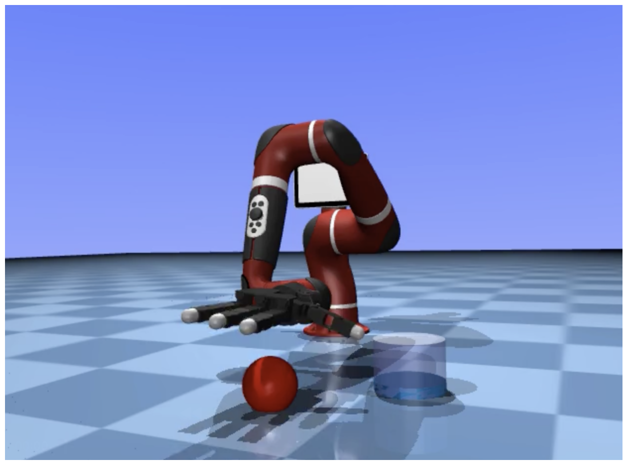

Fig. 20. Container task: before the grasp

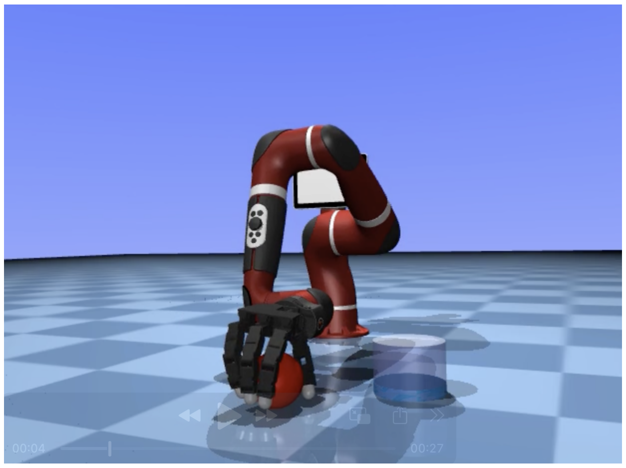

Fig. 21. Container task: after fingers made contact

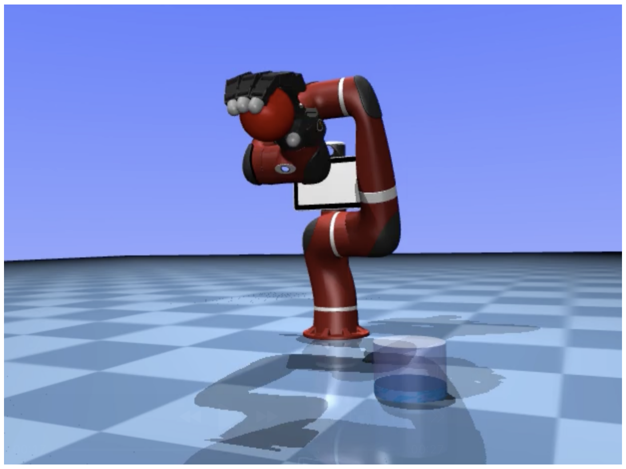

Fig. 22. Container task: lifting up

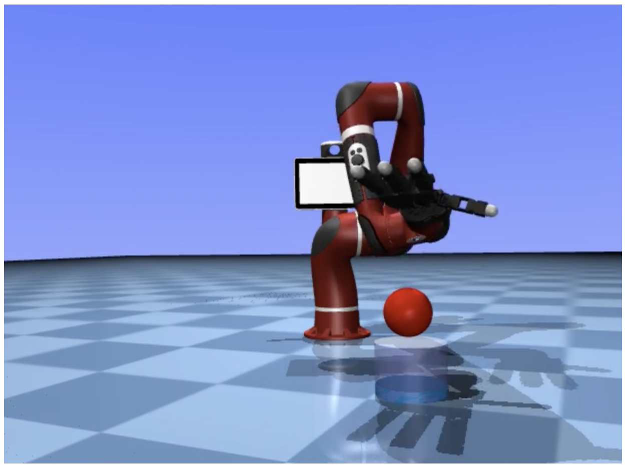

Fig. 23. Container task: Final Stage