Multifingered Grasping using Dexterous Hands in MuJoCo

Tyler Osbey
EECS
College of Engineering
Berkeley, USA
tosbey836@berkeley.edu

Edwin Villalpando
EECS
College of Engineering
Berkeley, USA
edwin23luv@berkeley.edu

Tenzin Norphel
EECS
College of Engineering
Berkeley, USA
tnorphel@berkeley.edu

***Abstract*—In this project, we implemented the Allegro Hand, a four-fingered robotic hand mounted on a Sawyer robot, with the goal of grasping a ball and experimenting with other shapes to observe variations in grasping behavior. To perform inverse kinematics on the multi-fingered hand, we implemented the Levenberg-Marquardt algorithm, also known as the damped least squares method. Unlike the Paden–Kahan subproblem approach commonly used for inverse kinematics in robotic manipulation, the Levenberg–Marquardt algorithm offers a more robust solution in scenarios involving redundancy or near-singularities. Using this algorithm, we are able to accurately move the Allegro Hand and the Sawyer Arm to a desired location for grasping, which are described using the grasp map (G). To test and refine this procedure, we used the MuJoCo physics engine and then implemented the grasping synthesis process using two core components: a grasp synthesis algorithm and a joint-space objective.**

I. Methods

The general aspect and goal of this lab is to move an Allegro Hand (that is attached to a sawyer), over a ball, grasp the ball, and then move the ball using this grasp. To do so we have abstracted the lab into two parts: Multi-fingered Inverse Kinematics, and Grasp Synthesis.

The IK solver we implemented was very similar to the one seen in the lab doc. The algorithm can be generalized to the following equation:

$$q_{t+1} = q_t + \left(\left(J^T J + \lambda I\right)^{-1} J^T \left(x_d - x_t\right)\right) \qquad (1)$$

The Levenberg-Marquardt method equation defined above is then called recursively for each "body part". This consisted of the palm of the hand, three pointing fingers, and a thumb. To each of these parts, we then implemented the pseudo-code seen in Algorihtm 1.

This algorithm, however, calculates the Jacobian only for one body part, thus, it must recursively be implemented for each body part. A faster approach would be to vectorize everything. Instead of computing the next pose, q_{n+1}, for each body part individually, we create a master Jacobian matrix that holds the values of all the smaller ones. Our new algorithm goes as seen in Algorithm 2.

Algorithm 1: Levenberg–Marquardt IK Solver

```
Input: Goal pose $y$, current joint angles $q$, body IDs
Output: Joint configuration $q$ reaching desired pose
$e \leftarrow Goal\_Pose - Curr\_Pose$
while $\|e\| \geq \epsilon$ do
    $J \leftarrow$ Jacobian$(q)$
    $J^T \leftarrow$ transpose$(J)$
    $J_{\text{inv}} \leftarrow (J^T J + \lambda I)^{-1} J^T$
    $\Delta q \leftarrow J_{\text{inv}} \cdot e$
    $q \leftarrow q + \alpha \cdot \Delta q$
    $q \leftarrow$ check_joint_limits$(q)$
    $e \leftarrow Goal\_Pose - Curr\_Pose$
end
return $q$
```

Algorithm 2: Multi-Finger IK

```
Input: Goal pose $y$, current joint angles $q$, body IDs,
       max_steps
Output: Joint configuration $q$ reaching desired pose
$e \leftarrow Goal\_Pose - Curr\_Pose$
while $\|e\| \geq \epsilon$ and $max\_steps > 0$ do
    $error \leftarrow$ empty matrix
    $J\_master \leftarrow$ empty matrix
    foreach body $i$ do
        $J \leftarrow$ Jacobian$(q[i])$
        $J\_master \leftarrow J\_master.append(J)$
        $pos\_error \leftarrow$
          $target\_positions[i] - current\_positions[i]$
        $quat\_error \leftarrow$
          $quat\_error\_calc(t\_quat, c\_quat)$
        $error_i \leftarrow \begin{bmatrix} pos\_error & quat\_error \end{bmatrix}$
        error ← append(error_i, error)
    end
    $J^T \leftarrow$ transpose$(J\_master)$
    $J_{\text{inv}} \leftarrow (J^T J\_master + \lambda I)^{-1} J^T$
    $\Delta q \leftarrow J_{\text{inv}} \cdot e$
    $q \leftarrow q + \alpha \cdot \Delta q$
    $q \leftarrow$ check_joint_limits$(q)$
    max_steps ← max_steps $-1$
end
return $q$
```

After doing so, the algorithm correctly calculates the movement necessary to move the body parts.

The grasp synthesis algorithm is composed of two parts: an optimization problem to minimize the total distance d between the fingertips and the object, and an optimization problem to generate a force closure grasp. The first part is trivial–we use the following gradient descent algorithm to drive d to zero across all fingers.

$$q_h^{\text{new}} = q_h - \lambda \nabla_{q_h} f(q_h) \tag{2}$$

Where $\nabla_{q_h} f(q_h) = \beta d$ if the fingers are not in contact with the object.

Once we have made contact, we can create the grasp matrix. We obtain the normal vectors to the object at the contact points of the fingers to construct friction cones that are then used to generate a feasible grasp matrix.

To measure whether a grasp is force closure, we can use the methods outlined in [1]. We define a loss function f that we want to minimize to ensure that we have a stable grasp:

$$f = \begin{cases} d_Q^+(0, W), & \mathbf{0} \notin co\{W\} \\ d_Q^-(G), & \mathbf{0} \in co\{W\} \end{cases} \tag{3}$$

Where d_Q^+ and d_Q^- represent the necessary condition and sufficient conditions respectively, and $co\{W\}$ represents the convex hull of the wrench applied to the object. The necessary and sufficient conditions are given by the following optimization problems:

$$d_2^+(0, W) = \min_{\alpha \geq 0} \|\mathbf{G}\alpha\|_2 + \beta D \tag{4}$$

$$\begin{aligned} \bar{d}_Q(k) &= \min -r \\ \text{s.t.} \quad r\mathbf{q}_k &= \sum_{i=1}^{N} \alpha_i \mathbf{g}_i \\ \sum_{i=1}^{N} \alpha_i &= 1 \\ \alpha_i \geq 0, &\quad r \geq 0 \end{aligned}$$

To further increase the difficulty of the lab, we decided to extend Grasp Synthesis onto two more objects, a bigger ball of $radius = 0.07$, and a cube with side $length = 0.08$. The grasp algorithm we used is sufficient enough for these shapes; however, increasing the mass and size can lead to slipping, requiring additional modification to the algorithm.

In terms of design decisions, we modified the video player loop so we could use interpolation to lift the ball. This way, we can achieve a smoother trajectory without worrying about changing the orientations of the fingers.

II. Experimental Results

NOTE: All videos will be provided as a link at the end of the paper

After implementing the Levenberg-Marquardt method, we tested its capabilities using three tests: a thumbs-up, a grasp around the ball, and the hand over the ball.

Here we can see how the IK Solver is able to create a thumbs up as seen in Fig. 1.

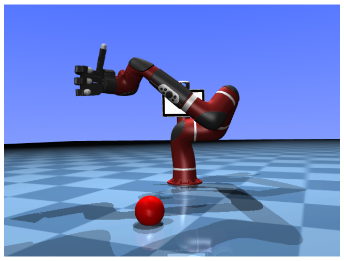

Fig. 1. Thumbs-up

Using our IK solver, we then attempt to grasp the ball using some generated points around the ball, as seen in Fig. 2.

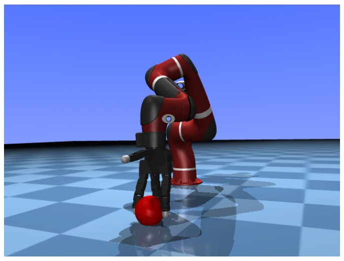

Fig. 2. IK Grasp

We can see that although the fingers are around the ball, this grasp lacks friction and force to move the ball upward. Thus, to generate a more effective grasp, we will implement Grasp Synthesis.

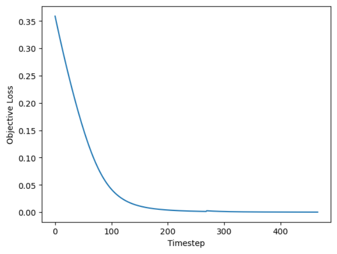

Fig. 3. Loss Curve

Grasping Stages Visualization: To demonstrate the full grasping process, we provide snapshots of the robot hand at three key stages of interaction with the object: before the grasp, after making contact, and after adjusting for force closure. These stages visually verify the progression of our grasp synthesis algorithm. Additionally, above in Fig. 3. we can see the loss curve for the objective over time.

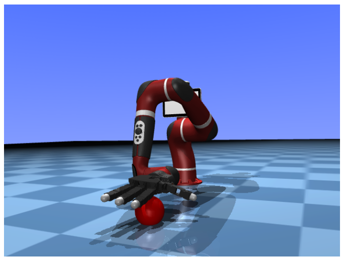
Fig. 4. Before contact

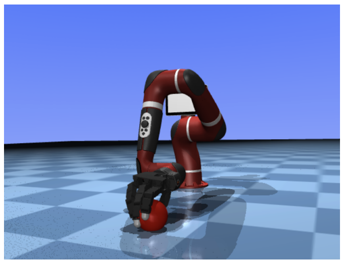
Fig. 5. After contact: touching the ball

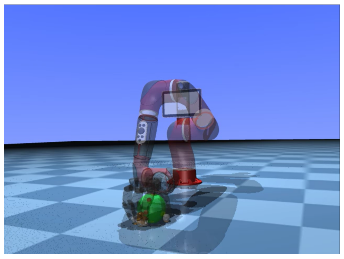
Fig. 6. After force closure - stable grasping

In Fig. 4, it illustrates the initial stage of the grasping process, with the Allegro hand being hovered over the ball in a flat pre-grasp configuration. At this stage, we implemented inverse kinematics through the Levenberg–Marquardt algorithm that helps position the hand to a feasible starting pose without first making contact or touch. In Fig. 5, the multifingers are slowly descending and starting to make contact with the surface of the ball. In Fig. 6, we applied grasping synthesis, which consists of a joint space objective, to minimize the force closure loss while maintaining fingertip contact, ensuring a stable and secure grip. It then closes the fingers to make contact with the object.

Additionally, in Fig. 7, we can see a deeper look at what's happening in a successful grasp. The fingers maintain contact the entire time, first with multiple contact points. As the hand moves up, the number of contact points reduces and stabilizes.

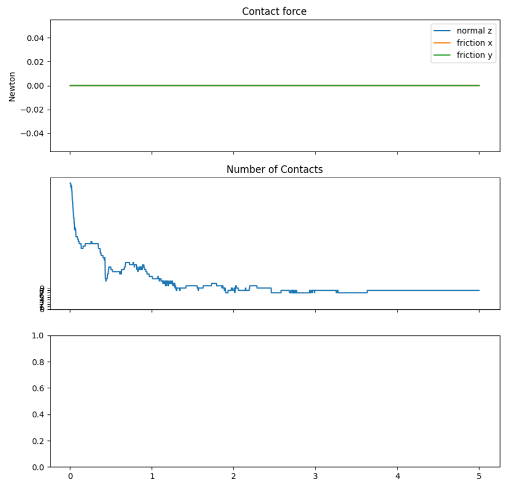

Fig. 7. Contact force vs. Time — Number of Contacts vs. Time — Successful Grasp

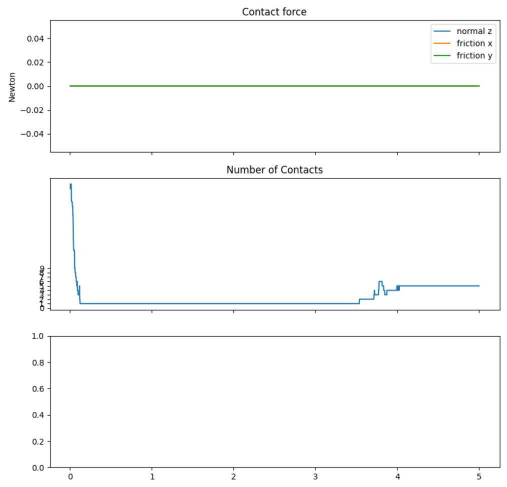

Fig. 8. Contact force vs. Time — Number of Contacts vs. Time — Fail Grasp

For a failed grasp, we can see the number of contact points is, for the most part, at one. When looking at the video you can see one finger touching the ball. The spikes before and after on the graph are difficult to explain even after looking at the video. The best explanation for the rise in the number of contacts registered later in the simulation is best described by the movement observed by the ball. The initial high number of contacts is difficult to source. The video does not show anything, thus, this number is most likely caused by the simulation and the interactions handled by the physics of the simulation.

After successfully grabbing the provided ball, we attempted to grab these additional objects:

Object	**Lifted**	**Slippage**
Big Sphere	Yes	Slightly
Cube	Yes	No

TABLE I
GRASP RESULTS FOR DIFFERENT OBJECTS

III. Discussions

Implementing the IK solver worked well and achieved rapid convergence (given a feasible set of target positions and orientations). Generating the thumbs-up, IK grasp points, and the hand-over-ball position worked. Trying random points above the ball, did not, as this required clipping to a valid state. Additionally, moving the hand from the thumbs-up position to the hand-over-ball introduced some problems. This would cause the sawyer arm to phase through the floor. To fix this, we had to call IK from the initial configuration (which was already close to the ball).

The grasp synthesis algorithm was successful in generating a force closure grasp that was sturdy enough to lift the ball off the ground. The gradient descent algorithm took some time, but we were able to observe the loss function value decrease over time

The extensions were straightforward once the grasp synthesis algorithm was set up. The small was picked up without an issue. The bigger ball was also picked up, however, moving the hand higher caused some slippage. The grasp synthesis would have to be modified to account for a heavier ball as well, and a ball with less friction. The grasp was most effective with the square object, as the fingers were able to nearly wrap around it.

As described previously, unless we are very specific and already know the points for force closure, using IK to generate grasping points does not work well. IK is only able to move the hand to any specified location, but it lacks the ability to locate and achieve force closure. Hence, the grasp synthesis algorithm is necessary. The combination of both of these methods works well, successfully grabbing an object and then move it (given that the hand is close enough initially). IK, however, is only able to avoid phasing and collisions with the ball, the hand, and the sawyer arm. Restricting its motion such that it is unable to move through the ground is necessary since the grasp synthesis algorithm is dependent on the collision that occurs in the arena. Since the grasping algorithm is able to grasp, an innovative way to improve it would be to include RL.

Additionally, in the grasp synthesis algorithm, we minimize a cost function in joint space that balances two goals:

1) bringing the fingertips close to the object surface, and
2) achieving a stable grasp.

If the hand hasn't made contact yet, the cost function is dominated by the distance to the object. Once contact is made, the algorithm evaluates grasp quality using the grasp map G, constructed from the contact normals and friction cones. The force closure loss (f_{c_loss}) quantifies how well the current contact configuration can resist disturbances.

This loss depends heavily on the object's geometry and symmetry — for example, symmetric objects like spheres may offer multiple valid contact combinations, while asymmetric shapes require more precise configurations. The total cost combines stability and proximity, guiding the optimizer to prefer both contact and high-quality grasps.

This project introduced a distinct set of challenges. We first had the issue of getting the thumb to move properly. This was caused by improper joint configuration and control inputs because of the particular kinematics of the thumb. We also very frequently had library issues, i.e., inappropriate dependencies or broken modules, when working with MuJoCo and the Allegro Hand SDK. These issues slowed down our test cycle and took an immense amount of debugging. We were able to resolve most of them by carefully controlling environments and referring to documentation and forums. Some thumb articulation problems, however, were left partially unsolved because of time limitations and hardware feedback limitations.

In the future work, actually using the Sawyer with the hands could lead to many great final projects, as well as implementing double arms and perform more complex and irregular shapes, more like soft contact. Having the robot play rock, paper, scissors; having the robot play tic-tac-toe, and more ideas such as these would be really interesting to see. Incorporating some RL into the grasp would be heavily advanced but could be included as some extra credit.

Acknowledgment

The authors would like to express their sincere gratitude to EE106B Distinguished Professor S. Shankar Sastry for his guidance and support throughout the project. Special thanks to our lab TAs, K. El-Refai and Yarden Goraly, for their invaluable assistance in the development and conceptual understanding of the optimization planner, RRT Planner (Rapid Exploring Random Tree), and sinusoid planner with their implementations on TurtleBot. Additionally, we extend our appreciation to Jaeyun Stella for her support in logistics and policy-related aspects of the project.

References

[1] X. Zhu and J. Wang. "Synthesis of Force-Closure Grasps on 3-D Objects Based on the Q-Distance". IEEE Transactions on Robotics and Automation (2003).

Appendix

- GitHub: Project4 Grasping Code