# RoboFlipper: Precisely Perceiving and Flipping Objects

Aarush Gupta
*Massachusetts Institute of Technology*
Cambridge, MA, USA
aarushg@mit.edu

Rayna Arora
*Massachusetts Institute of Technology*
Cambridge, MA, USA
raynaaro@mit.edu

***Abstract*—Flipping objects requires precision in perception, grasping, and throwing, and is thus an interesting task to explore for robotic manipulation. This paper presents RoboFlipper, a system that allows an IIWA arm to flip and land a box at arbitrary poses. We break down the full manipulation pipeline, including physical analysis, perception, key pose construction, trajectory construction, and simulation. Through this approach, we are able to flip a box at a random pose with 60% accuracy, which is better than a human benchmark of 44%. Considering limited work relating to object flipping exists, this result is substantial and provides a baseline for further research into related manipulation problems. To facilitate expansion on our work, we elaborate on limitations of the project, including a lack of generalizability to flipping more complex objects, the use of naive perception techniques, and grasp instability.**

## I. Introduction

The goal of our project is to construct a full pipeline that allows a camera setup to detect the pose of a box, and then relay it to an IIWA arm which will grasp and flip the object so that it lands upright.

Flipping an object is an interesting task as it involves a complex manipulation pipeline: perceiving the object, producing an optimal trajectory, and simulating and executing this trajectory. These steps span a large number of techniques discussed in Robotic Manipulation, which is why we chose to pursue this task as our final project. Additionally, flipping objects is particularly well-suited for robotic arms as it requires a high level of precision. A sufficiently robust system is potentially capable of outperforming most humans at this task (most humans are unable to consistently flip some object upright). This is not the case for many manipulation tasks: for example, tasks like pick-and-place, perception, etc. seem very intuitive and easy for humans.

Another reason we are interested in pursuing this project is that it lends well to kinematic analysis. As will be discussed throughout this paper, all of our steps are motivated by calculations and intuition from Newtonian mechanics. However, this problem is also non-trivial in the sense that the exact mechanics of the gripper release play a very important role in how the object lands: we still need to do empirical work in order to figure out how to precisely generate optimal flip trajectories.

There is limited work in the space of flipping objects — in fact, this is a key reason we want to pursue this project. Working on this project opens opportunities in a number of related manipulation tasks that involve launching or throwing objects in a precise way.

To measure the effectiveness of our algorithm, we use consistency over different object positions. As a rough threshold, we tried flipping a similar box ourselves 25 times: we had 11 successful flips, yielding an accuracy of roughly $44\%$ (we would even say this is above average for humans!). To outdo an average human, we aim to achieve at least $50\%$ accuracy over randomly initialized object positions. We expect that prioritizing consistency over different starting poses may compromise our ability to flip multiple objects. Thus, for this project, we focus our efforts on flipping a rectangular box, which can be seen in Fig. 7.

## II. Related Work

As mentioned earlier, limited work has been done in flipping objects. However, we drew inspiration from research relating to throwing objects. While the specific goals of throwing and flipping (sending an object to a certain location versus landing an object in a certain orientation) differ, both are tasks that can be modeled through kinematics.

We were particularly drawn to the approach of TossingBot [1], which uses a residual physics-based approach to throw objects. In essence, a physics-based controller performs baseline calculations based on simple mechanical principles. Then, a model is trained to perform residual empirical corrections to the output of the physics-based controller.

While we want to avoid the complexity of using a machine learning model in our manipulation pipeline, we adopt the general approach of planning physics-driven trajectories and then adapting them accordingly. This approach has a key benefit over a machine learning approach in that we do not require a significant amount of training data but rather use heuristics informed by physical intuition, which are more interpretable. The tradeoff is that machine learning techniques are usually be more sophisticated and accurate.

We further investigated the physics behind throwing object with [2]. This paper describes the physics of throwing an object with a singular hinge joint at a specific angle and angular speed, in an attempt to make the object land at a specified

location. It comes up with constraints, such as the position and the direction the object must be moving in when it reaches the top of the bin. It then finds equations describing solutions for both overhand and underhand throws, and analyzes the variability of accuracy in different scenarios such as how faster throws lead to a sharper decay in accuracy as parameters are changed than slower throws. We take inspiration from the way this paper has calculated the constraints of the position of the object when it hits the target. We also find it very useful to see that accuracy can vary more when rotational velocity is higher; we intend to use this to make our flips more reliable by trying to minimize angular velocity. However, for our specific use case, we must specify not the position but the orientation of the object when it lands, so our calculations will differ.

In the domain of grasp selection, we feel [3] is relevant to our work. Given the point cloud of an object and geometrical information about a robot's hand, this paper uses machine learning to generate potential grasp poses. We take inspiration from this paper's use of the point cloud to gain a strong sense of an object's geometry. We feel this work and similar research on generating grasp poses will be relevant to extending our work to flip objects with more complicated geometries, although we feel that a simpler grasp selection algorithm is sufficient for simply flipping a rectangular box. An additional consideration needed is that when the robot grasps the object, its arm should be in an optimal position to pick up and quickly flip the object.

## III. Dynamics of Flipping

To understand the mechanics of flipping objects, we first do some analysis using classical Newtonian mechanics. Assume an object of height $H$ and width $W$ (diameter for a cylinder) with uniform density is launched with translational and rotational velocities $v_i$ and $\omega_i$ at angle $\theta$ (from the vertical) and height $h_i$. We want to compute a relation between these variables that will ensure stable landing.

We also expect there to be some error, since the gripper opens during the last 0.01 seconds of the arm's rotation. During this time, we expect the object to slip in the gripper and move slightly slower due to friction between the gripper and the object. Thus, we will leave room for error in our calculations by using an angular velocity comfortably within the appropriate landing condition.

From simulations, we have determined that when the object is perfectly upright when it hits the table, its existing rotational velocity causes it to continue turning and fall over. Thus, it should impact the table before being fully upright. We also ensure that the object's center of mass is above its bottom face so that it does not fall backward onto its side. Fig. 1 demonstrates this concept. Let $\theta_0$ be the angle between the bottom of the object and the ground upon the first impact; the object will not fall to the right as long as $0 \leq \theta_0 \leq \tan^{-1}\left(\frac{W}{H}\right)$. We will take the highest possible value of $\theta_0$, $\tan^{-1}\left(\frac{W}{H}\right)$, to ensure that the rotational velocity does not cause the box to fall onto its left side. This equation applies whether the object is a rectangular prism or a cylinder.

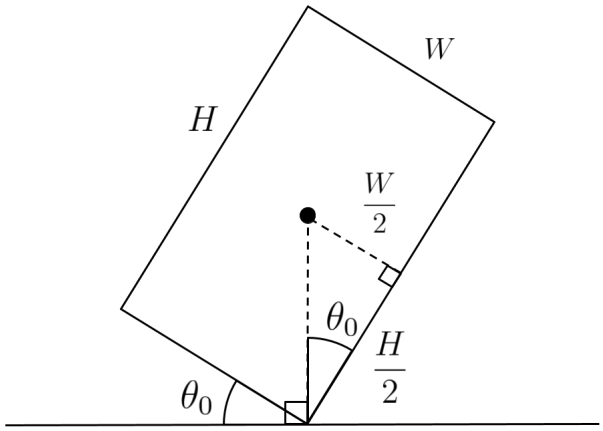


Fig. 1. Example of an object as it hits the ground

We will have the robot flip the object at an angle of $\theta = \frac{\pi}{2}$ for simplicity. For a single flip, we want

$$\Delta\theta = 2\pi - \theta - \theta_0 = \frac{3\pi}{2} - \theta_0$$

so

$$\Delta\theta = \frac{3\pi}{2} - \tan^{-1}\left(\frac{W}{H}\right)$$

Since $\frac{\Delta\theta}{\omega_i} = t_{\text{fall}}$

$$\boxed{t_{\text{fall}} = \left(\frac{3\pi}{2} - \tan^{-1}\left(\frac{W}{H}\right)\right)/\omega_i}$$

Also, let $\Delta h$ be the distance that the center of mass of the object falls. We are measuring with the center of mass rather than the bottom of the object because the object will be rotation about its center of mass. Let $h_i$ be the initial height between the bottom of the box and the ground right before it is released. Then, according to Fig. 2,

$$\Delta h = h_i + \frac{H}{2} - \frac{\sqrt{W^2 + H^2}}{2}$$

From kinematic equations, we know that $\Delta h = \frac{1}{2} g t_{\text{fall}}^2$ since $\theta = \pi/2$.

Combining these results yields

$$\boxed{t_{\text{fall}} = \sqrt{\frac{2}{g}\left(h_i + \frac{H}{2} - \frac{\sqrt{W^2 + H^2}}{2}\right)}}$$

From setting our two expressions for $t_{\text{fall}}$ equal, we calculate the necessary initial angular velocity based on the object's dimensions.

## IV. Methods

Our manipulation pipeline consists of three main components: perceiving the object and determining its pose, constructing key end-effector poses for the flip trajectory, and constructing optimal joint trajectories to achieve all of these

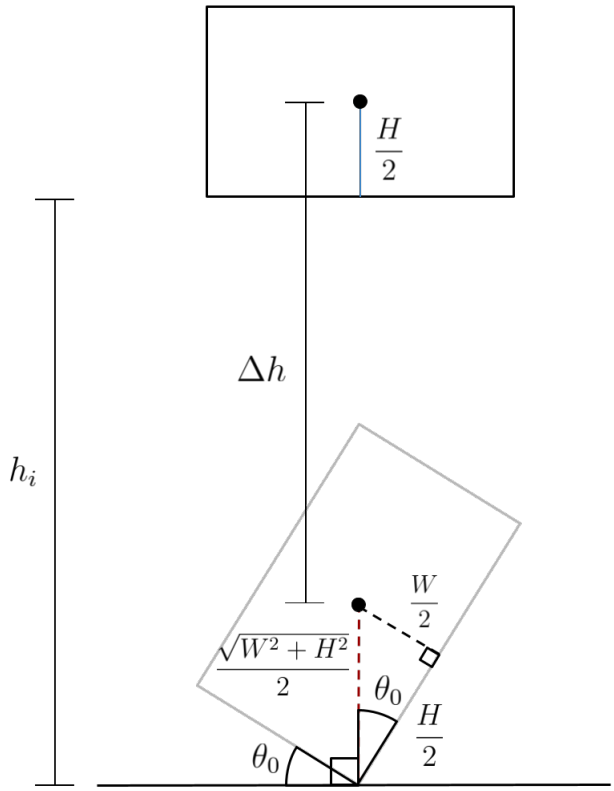


Fig. 2. Demonstration of the height that the center of mass must fall

key poses. This is illustrated in Fig. 3. This sections describes the methods we use in our pipeline. The last component of the pipeline is actually simulating and executing the trajectory, but this will be discussed in a later section.

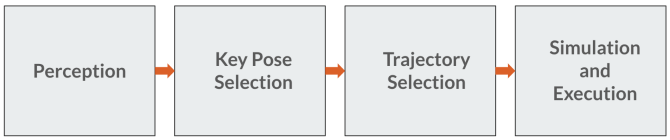


Fig. 3. The full pipeline used to flip an object.

### A. Perception

An important part of our problem is determining the location of the object to be flipped. Additionally, we also want to determine the orientation of the object so that we can grip the surfaces of the object with maximal surface area to ensure grip stability.

To do this, we use cameras to first estimate the pointcloud of the object. The camera is placed high above the scene, capturing the iiwa arm and the object, and is oriented downward. After obtaining the pointcloud, we filter out the points corresponding to the iiwa arm and table, which is done through finding all points that are upper bounded by a certain $z$ coordinate or are close to the $x-y$ origin.

Now that we have the object pointcloud, we find the position of the object by computing the mean of the $x$ and $y$ coordinates of the corner points. The four corner points are the points with minimum and maximum $x$ and $y$ coordinates, as can be seen from Fig. 4. The grasp height is determined through the maximum $z$ coordinate of the object points.

In order to find the orientation of the object for optimal grasping, we make a simplifying assumption that the object we are considering has a rectangular shape. This assumption also works fine for roughly rectangular objects as well as circular objects, which are the main other class of objects that are relevant.

To find the direction of the longer side of the rectangle, we first project all the points in the $x-y$ plane, ignoring the $z$ coordinate. Then, we find the two relevant corners that define the longer side by computing pairwise distances between the corners. Once we have found the right corners, we compute the direction of the line running through them. We can convert the slope $m$ of this line to an orientation angle $\theta_x = \tan^{-1} m$. Then, we orient the length of the gap in the gripper along this direction.

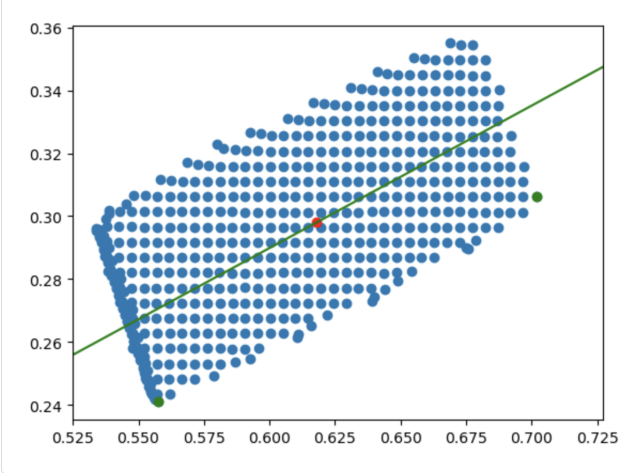


Fig. 4. Example of calculated orientation (green line) and center (red dot) in $x-y$ plane for arbitrary box orientation. The green dots are the corner points used to calculate the line of orientation.

This perception scheme is simple, fast, and very accurate for our purposes.

### B. Key Pose Construction

Now, we can construct four key poses of the end-effector trajectory for the flipping action.

First, we start off with the picking action. Through perception, we know ${}^{W}X^{G}_{\text{pick}}$, which is the combination of the calculated $\theta_x$ and the center $x-y$ coordinates as well as the top $z$ coordinate. We also define ${}^{W}X^{G}_{\text{above object}}$, which is the same as ${}^{W}X^{G}_{\text{pick}}$ except we slightly increase the $z$ coordinate. This is the pose that the gripper will move to right before picking up the object, so that it can move directly downward to pick up the object, avoiding any unwanted collisions.

We then close the gripper and move to ${}^{W}X^{G}_{\text{above object}}$ once more. This is because we have set the throw frames in a specific spot in the air in front of the iiwa arm to allow for an easy, simple throwing motion. If we do not add this frame, the robot moves in a diagonal upwards motion that drags the box on the ground, leading to instability in its position in the robot's grip.

Then, we have poses ${}^{W}X^{G}_{\text{before throw}}$ and ${}^{W}X^{G}_{\text{throw}}$ for the throwing action. The former is a setup pose close to throwing position. Then, ${}^{W}X^{G}_{\text{throw}}$ is the pose where we wish to launch the object, which is ${}^{W}X^{G}_{\text{throw}}$ rotated by some angle $\theta$. Assuming that the arm starts with $\omega_i = 0$ and uniform acceleration $\alpha$, $\alpha = \frac{\omega}{\Delta t}$ and $\Delta\theta = \frac{1}{2}\alpha\Delta t^2$, so

$$\boxed{\Delta t = \frac{2\theta}{\omega}}$$

### C. Trajectory Selection

A large part of the work we did was designing a reliable system to choose trajectories for the flipping process. These trajectories start at some arbitrary location, move next to and grasp the object to be flipped, and then quickly launch it at a specified position and velocity. The setup of this problem is that given a set of position and velocity constraints along the path, we aim to calculate the corresponding optimal joint trajectories. We attempt three approaches for trajectory selection.

*1) Differential Inverse Kinematics:* We initially set up a controller based on differential IK. To do this, we manually create an end-effector trajectory that satisfies our desired position and velocity constraints. The trajectory is designed by an interpolation algorithm based on the key poses. Then, the differential IK system process the trajectory in sequential order, calculates the corresponding Jacobian to convert from end-effector to joint phase space, and then computes the differentials in the joint angles and iteratively constructs the joint angle trajectory.

However, this approach has a number of flaws which caused it to fail. Differential IK only calculates local changes in joint angles, so it fails to consider what joint angles would be best suited for actions farther in the future. For example, consider a certain trajectory. At some time $t$, we might have multiple options for how to change joint trajectories to satisfy the local end-effector trajectory. However, one of these options may twist the joints in a way that makes it much more difficult to execute a future action; differential IK fails to take this into account.

*2) Kinematic Trajectory Optimization:* A more general approach that solves a number of the aforementioned problems is kinematic trajectory optimization. Instead of setting up a differential system, we create an optimization problem with a number of constraints. The costs in the optimization include path length costs, duration costs, costs for deviation from a "simple" path, etc.. The constraints can be pose and velocity constraints at certain times along the path. Additionally, this method is very convenient to implement through Drake's `KinematicTrajectoryOptimization` class.

For the flipping problem, we have key poses ${}^{W}X^{G}_{\text{above object}}$ and ${}^{W}X^{G}_{\text{pick}}$ as well as some desired launch velocity $\mathbf{v}$ (we don't even need the throwing poses as the launch velocity constraint will take care of that for us). Note that we do not need the throw poses since our velocity constraint will automatically take care of that. Our constraints on our trajectory $\mathbf{q}(s)$ are then

$$||\mathbf{q}(s_{\text{above object}}) - {}^{W}X^{G}_{\text{above object}}|| \leq \epsilon$$
$$||\mathbf{q}(s_{\text{pick}}) - {}^{W}X^{G}_{\text{pick}}|| \leq \epsilon$$
$$||\dot{\mathbf{q}}(1) - \mathbf{q}|| \leq \delta$$

To solve this problem, we used the built-in SNOPT solver [4]. However, we found that SNOPT was often unable to find solutions, even with just a few simple constraints. Despite increasing tolerances significantly, this approach did not work. Thus, while being the most general approach, kinematic trajectory optimization does not work well for our use case.

*3) Inverse Kinematics with Trajectory Interpolation:* The approach we finally settled on was using an inverse kinematics system with trajectory interpolation. Unlike differential IK, we do not actually construct a trajectory from the key poses. Instead, we directly solve an IK problem for each of these poses, getting the joint angles that correspond to each of the poses. Then, we interpolate the joint trajectories from a number of *joint angle poses*. We choose to use piecewise linear interpolation as this is simple and efficient.

This method worked very well for us in a reliable and consistent manner for varying trajectories. However, we make one crucial change to the key poses for this approach to work.

Upon plotting the rotational velocity of the robot's final joint (the only one moving while flipping the object) with respect to time, we find that the robot slows down at the very end, as shown in Fig. 5. This is likely because we use a cubic shape preserving piecewise polynomial for pose interpolation, rather than linear interpolation.

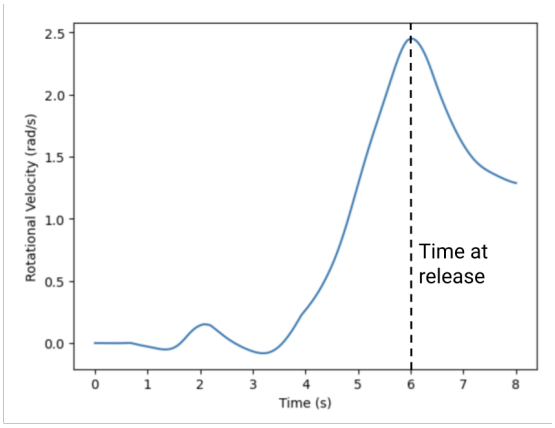


Fig. 5. $\omega(t)$ before implementing follow-through key pose

This is a significant issue because the object's launch angular velocity is very dependent on the gripper angular velocity right around the time of release: we want the desired

velocity to be achieved for longer near the peak. To fix this, we cause the robot to "follow through," finishing its trajectory past the target end angle. In mathematical terms, say $t, \omega_i$, and $\Delta\theta$ are the original calculated values for time to flip, final rotational velocity, and original angle change ($\frac{\pi}{2}$). We still want the rotational velocity at time = $t$ to be $\omega_i$. This means that we instead ask the robot to flip the object for time $t + \epsilon$ and assume uniform acceleration, it will end up at rotational velocity $\omega_i + \epsilon\omega_i/t$, meaning $\Delta\theta_{\text{new}} = \frac{1}{2}(t + \epsilon)(\omega + \epsilon\omega_i/t)$. Through experimentation, we found a value of $\epsilon$ high enough so that the rotational velocity at the time of release is nearly $\omega_i$, but low enough so that the robot arm is able to move out of the way so that the box does not collide with it.

## V. Experiments

### A. System Setup

Our system is composed of three main components: the IIWA arm and gripper, the object to be flipped, and a camera setup, as shown in Fig. 6. The camera is placed high up to provide a wide view of the surroundings of the arm.

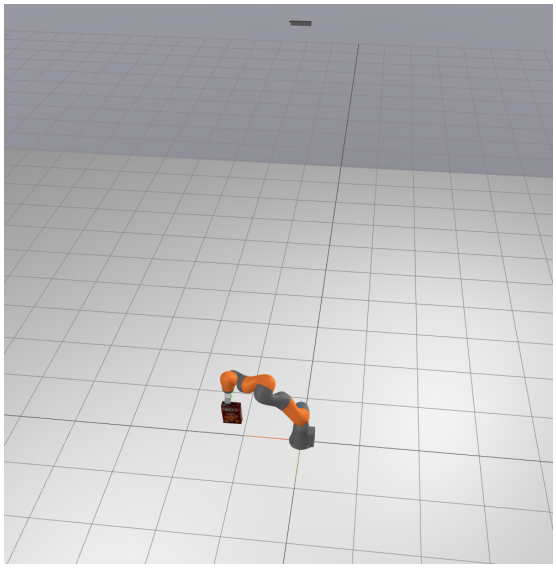

Fig. 6. Camera, arm, and object setup.

We set up a `Diagram` in Drake that contains all of these components through creating the relevant scenario data. The camera sends its perceived pointcloud through its output port, which is then processed to compute the desired grasp and pregrasp poses. After the optimal joint trajectory is computed using the methods described above, `TrajectorySources` for the IIWA joints and gripper are initialized and connected to the respective input ports. We then run our simulation and record it in Meshcat for viewing and analysis. Sample frames of a flip are illustrated in Fig. 7.

### B. Results

Our original goal was to be able to flip a box at an arbitrary location and orientation. Thus, to test the robustness of our framework, we run numerous simulations with the arm attempting to grasp and flip the box placed at a random $x - y$ position and oriented at an arbitrary angle $\theta_x$ from the $x$-axis. This is a reasonable evaluation protocol as we want to see how accurate the robot is at flipping the box irrespective of its specific pose. The table of the initial $x, y, \theta_x$ is shown in Table 1 while Fig 8 and Fig 9 display the distribution of successes and fails in the $x - y$ plane and $\theta_x$ numberline.

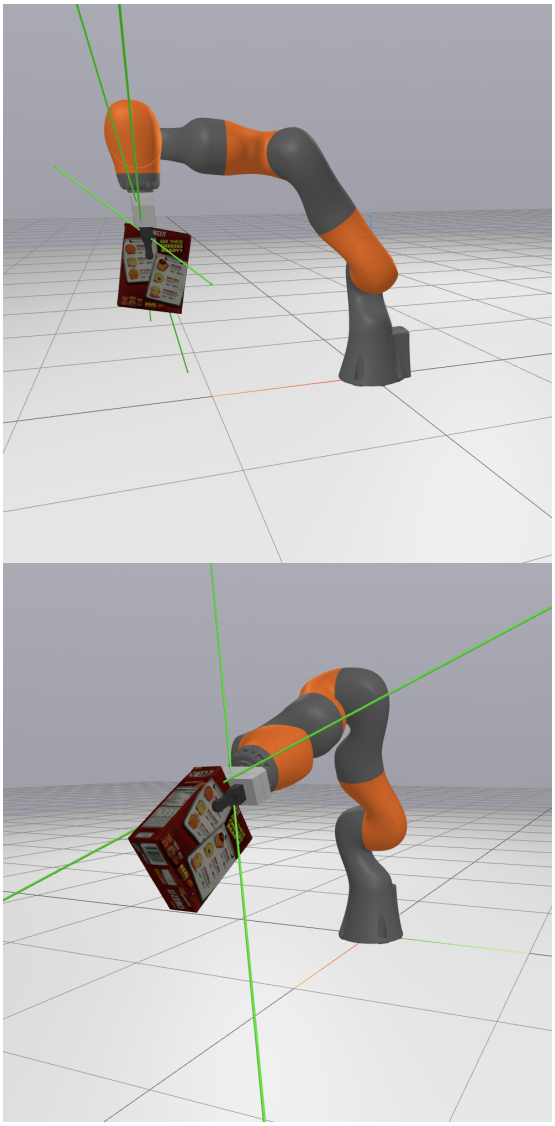

Fig. 7. Two frames of arm while rotating to flip box.

## VI. Discussion and Conclusion

### A. Analysis of Experimental Results

Overall, the arm does a good job of flipping the box, achieving an accuracy of 60%. This is better than the threshold benchmark we set earlier in the paper!

In general, the successes and fails had some randomness, fails often being caused by slight deviations in perception and grasp. A clear trend is that more extreme angles lead to higher failure rates, which makes sense.

TABLE I
FLIP RESULTS

| $x$ (m) | $y$ (m) | $\theta_x$ (rad) | Success |
|---|---|---|---|
| 0.5879 | 0.08532 | 0.0306 | ✓ |
| 0.62717 | -0.194997 | 0.3027 | × |
| 0.573408 | -0.007412 | 0.2509 | × |
| 0.2939 | 0.48198 | 0.195 | ✓ |
| 0.20757 | 0.3643 | 0.396 | × |
| 0.3216 | -0.434 | 0.192 | ✓ |
| 0.216 | 0.443 | 0..064 | ✓ |
| 0.473 | -0.471 | 0.404 | ✓ |
| 0.407 | -0.437 | -0.974 | ✓ |
| 0.567 | -0.003 | -1.077 | × |

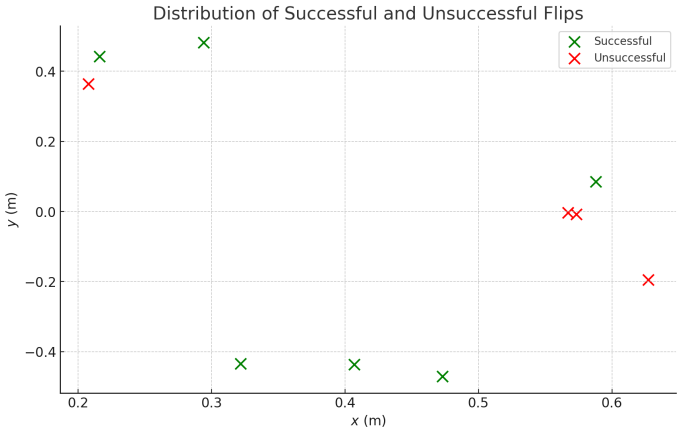


Fig. 8. Distribution of successful and unsuccessful flips in $x - y$ plane

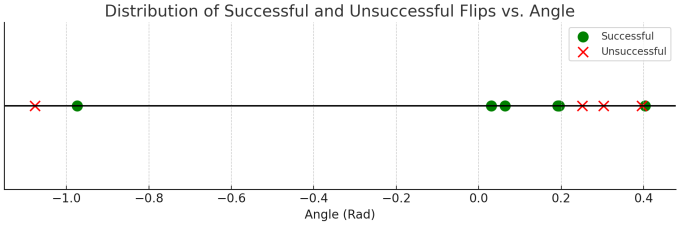


Fig. 9. Distribution of successful and unsuccessful flips in terms of $\theta_x$

However, one interesting trend we noticed through observing the simulation was that the box being aligned with either the $x$ or $y$ axis increased error in the perception algorithm. This is because we determine box corners through finding minimal and maximal $x$ and $y$ coordinates. For a box oriented along the $x$-axis for example, all the points along the long side will have the same $y$ coordinate: then, random error will determine which point is chosen as the corner which leads to flawed perception. We also noticed that the flipping task tended to fail sometimes when the box wobbled in the robot's grasp as it was being picked up. When the robot flipped an object from an unstable grasp, there was understandably some deviation in the box's trajectory based on our calculations. This could be combatted in the future by using a robotic gripper with rubber or another material that is able to grasp more firmly.

### B. Significance of Work

This work is significant as we took built a functional pipeline to accurately flip objects, which is a task for which limited prior work has been done. As mentioned earlier, our work combines many of the topics discussed in Robotic Manipulation, making it an instructive and comprehensive final project. Even without using any machine learning techniques, we achieved a reasonable accuracy of 60% which beats the average human's accuracy. Additionally, *no arbitrary empirical corrections were used*. All of our equations and poses were concretely derived from first principles and *empirical corrections were motivated by physical intuition*. We are confident that with more advanced techniques, object flipping could become a task easily doable through an automated pipeline. Below, we provide some suggestions relating to the areas of expansion in our project.

### C. Limitations and Further Work

The primary limitation is the type of objects the arm is able to flip. We worked with a box which has substantial flat surface area on two sides, which allows it to be easily grasped by the WSG gripper. The robot was able to flip a cylindrical can but only if it grasped it precisely in the middle; otherwise, the can would fall out of its grasp. The robot was not able to grasp more complex objects like a mustard bottle. Humans can flip such objects as we can curve and deform our hands to grasp them effectively. For robotic flipping, we could investigate using different hardware for grasping (e.g hand with dexterous fingers). Additionally, a more complicated perception system, potentially incorporating CNNs or some other machine learning models, could be used to calculate an optimal grasp for these objects. Aside from different shapes, there are also objects with non-uniform and even dynamic densities, such as water bottles! In fact, flipping water bottles was our initial project idea — we decided against this because the internal fluid dynamics would make this nearly impossible to analyze non-empirically. A basic search could be run to flip these sorts of objects to manually figure out what angular velocity would result in a successful flip. Additionally, a neural network could be trained to predict this launch velocity. [5] provides physical intuition about the dynamics of flipping a water bottle. This paper provides useful physics insight into how the moment of intertia, the center of mass, and initial rotational velocity impact an object's path while in the air. This paper demonstrates an intriguing method to simulate fluid dynamics of a water bottle through placing a few solid objects inside it, rather than countless particles.

Another limitation is that the flipping is very sensitive to factors like the exact grasp position. Further work should aim to develop a system that is more resistant and fault-tolerant, which would allow it to be used in real-world settings where external factors are almost always present. Increased robustness may also naturally follow from using more advanced

approaches (e.g. neural networks) to make predictions and perform certain actions.

## VII. Contributions

Both of us split most of the tasks evenly, as we worked jointly to brainstorm and debug a variety of approaches. Specifically, we collaborated equally on all the following tasks.

- Analyzing of the dynamics of flipping.
- Setting up the simulation environment (e.g. setting up the `Diagram` and all relevant system components).
- Experimenting with and debugging different trajectory construction methods.
- Running tests to analyze the efficacy of our methods.
- Debugging issues faced throughout the project.
- Writing the final report.

Individually, Rayna also worked on fine-grained key pose selection based on dynamics analysis as well as empirical corrections. Aarush worked on setting up the camera and perception system and algorithm to determine the pose of the object.

## References


[1] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser, "Tossingbot: Learning to throw arbitrary objects with residual physics," *IEEE Transactions on Robotics*, vol. 36, no. 4, pp. 1307–1319, 2020.

[2] M. Venkadesan and L. Mahadevan, "Optimal strategies for throwing accurately," *Royal Society Open Science*, vol. 4, no. 4, p. 170136, 2017.

[3] A. ten Pas and R. Platt, "Using geometry to detect grasps in 3d point clouds," 2015.

[4] P. E. Gill, W. Murray, and M. A. Saunders, "Snopt: An sqp algorithm for large-scale constrained optimization," *SIAM Review*, vol. 47, no. 1, pp. 99–131, 2005. [Online]. Available: https://doi.org/10.1137/S0036144504446096

[5] P. J. Dekker, L. A. G. Eek, M. M. Flapper, H. J. C. Horstink, A. R. Meulenkamp, J. van der Meulen, E. S. Kooij, J. H. Snoeijer, and A. Marin, "Water bottle flipping physics," *American Journal of Physics*, vol. 86, no. 10, p. 733–739, Oct. 2018. [Online]. Available: http://dx.doi.org/10.1119/1.5052441