# Throw Like You Catch: An End-to-end Robotic Throwing and Catching System

David von Wrangel
*MIT 6.843*
Cambridge, MA
wrangel@mit.edu

Ria Sonecha
*MIT 6.800*
Cambridge, MA
rsonecha@mit.edu

Shreya Pandit
*MIT 6.843*
Cambridge, MA
shreyap@mit.edu

***Abstract*— Catching and throwing are ubiquitous capabilities used to manipulate a variety of objects in different locations. Robotic systems programmed with this skill set would be extremely beneficial in a myriad of applications, and we strive to achieve a simple version of this task. We created three different modules in Drake for this project: single arm catching, single arm catching and throwing back, and multiple arms throwing and catching. The final system applies kinematics, trajectory planning, and optimization to the sphere, robot controller, and gripper controller to create a smooth system. The development of this system required a variety of approaches to generate the most fluid and successful arm and gripper motion which could also generalize to a variety of initial conditions. To evaluate the system, we conducted an analysis for both catching and throwing respectively. We measured the system's success rate in catching spheres that landed within the arm's radius of catching. The spheres each had a different initial position and velocity, allowing us to effectively examine the effectiveness of the arm's catching module. Our system is able to catch 42% of spheres that land within the robot arm's radius, failing to catch balls that slipped or violated joint velocity constraints. Our main findings suggest that the robot can successfully catch spheres from a variety of positions with slow to intermediate velocities but struggles to grasp spheres that are travelling at high velocities. We show our robot successfully catching in this representative example video and then catching and throwing back in this video. We finally demonstrate the ability for the robotic arms' to throw back and forth with one another in our last video.**

## I. Introduction

Many menial and laborious day to day tasks can be accomplished using modern day robots. Robots have started to improve efficiency, increase productivity, and enhance human experience. A prominent application of robots is its use in warehouses and manufacturing to move objects around more efficiently and safely than humans. Enabling robots to both throw and catch with one another would expedite the process of obtaining particular packages or materials and moving them to other locations in the warehouse or factory without any bins or human involvement needed. For our project, we want to focus on this improvement by creating a robot that is able to both throw and catch. We believe that this seemingly simple task possesses a plethora of interesting problems and has many other valuable applications. For example, this project could also enter the sphere of entertainment. A robot arm that can throw and catch could play a game of catch with children and families or even play fetch with a dog.

We have built a system that can both throw and catch simple objects. In order to do this, we solved several subproblems using the approaches we have touched upon in class. We applied kinematics, trajectory planning, and optimization to accomplish our task. We created our environment with particular assumptions and constraints in mind, and optimized the subproblems so that the flying objects processing and action can happen fast enough to effectively catch and throw a ball. In the future, these individual tasks can then be combined with perception and target detection to perform the more complex goals described above.

## II. Related Work

Robotic catching and throwing are complicated problems because the dynamics of the object being caught or thrown are dependent on a variety of factors such as the mass distribution of the object, the contact forces between the object and the gripper, and the location at which the object is grasped. These aspects of the catching and throwing problem are often unpredictable and hard to describe with standard projectile physics models. To address such uncertainties some groups have found ways to combine the simple equations of projectile motion with learning-based methods.

For example, TossingBot [3] uses a perception based approach to learn the residual velocity and optimal grasp location for each object. This allows them to compensate for non-linear dynamics, off center grasps, and a variety of object types, and other uncertainties. Similarly, in [2] the authors use learning from demonstrations to train the robot to predict a flying object's trajectory, choose a catching configuration, and plan the robot arm motion to reach that configuration. With this learning based approach the robot was able to catch many different types of objects which had uneven shapes and non-uniform mass distributions.

In addition to learning based methods, other past works have attempted to address uncertainties about object dynamics by estimating mass distribution parameters online. For example,

in [1] the authors use the robot's torque sensors to estimate parameters while following a predefined trajectory. In subsequent work, the same authors followed a similar procedure but made torque measurements immediately before throwing the object. This allowed for a seamless transition between catching, parameter estimation, and accurate throwing.

Unlike these approaches, our project takes a purely physics based approach with no adaptive control. Because we focus on just catching and throwing a simple sphere, there are few uncertainties about the object's mass distribution and dynamics. In the future our work could be augmented with adaptive methods, like those described above, to make catching and throwing objects with non-linear dynamics more feasible. In addition, most of the literature in this area addresses either throwing or catching, and few papers show a fully end-to-end throwing and catching system. In our project we attempt to solve both throwing and catching so that the robots can be chained together and throw and catch between themselves. As discussed in the previous section, this has the potential to make the movement of small objects around warehouses or factories significantly faster.

## III. Method

Our method consists of three main sub-modules: the simulation environment, catching, and throwing. Given our limited time, we made some simplifying assumptions about the types of objects we manipulate and how to predict trajectories. These assumptions and each submodule are described in more depth in the following sections.

### A. Simulation and Environment

Our project was done in simulation using Drake. We used the Kuka LBR iiwa as our robotic arm, which has a payload capacity of seven kilograms and seven degrees of freedom. For our gripper we used the Schunk WSG 50 two finger parallel gripper. Models for both of these devices are defined in Drake. In addition to these we also added a simple sphere with a mass of 56 grams and radius of 3.3 centimeters to the simulated environment.

For catching, we assumed perfect ball trajectory prediction. To simulate this, before doing any arm trajectory planning, we drop the ball with the given initial position and velocity and collect the ball's instantaneous position and velocity at a variety of time steps. These lists of positions and velocities are then passed to the catching environment and used by the arm's controller to plan a trajectory to catch the ball. In the future this step would ideally be performed online as the ball is falling and the position and velocity would be predicted by a perception module.

Additionally, we assume the system has no drag, and as explained earlier, we only throw and catch a sphere with a uniform mass distribution.

### B. Catching

Our initial approach to catching was to solve a mathematical program which found the ideal position for the robot arm to intercept the ball's trajectory. The problem included simple ballistic motion constraints and heuristics for the Kuka arm to reach the intercept location considering reachable space and velocity constraints. We then planned a simple trajectory from the robot's initial position to this target location and waited until the ball reached this position. As soon as the ball was between the gripper, we would close the fingers. Unfortunately, this was not a robust solution because the time it takes to close the gripper is too long, and by the time the gripper was closed, the ball would have slipped through.

In order to solve this problem we moved to a more dynamic approach whereby the gripper matches the ball's trajectory for a period of time before the ball hits the ground. By doing this, we ensure that there is a longer time segment during which the ball is between the gripper's fingers and we have more time to close the hand and catch the ball.

With this more robust solution, there are four key steps to our catching module once the ball's trajectory has been predicted:

1) Ball pose filtering
2) Arm trajectory planning
3) Arm controller
4) Gripper trajectory planning

Putting these aspects together allows us to move the robot from its starting configuration to a reachable position along the ball's predicted trajectory, then continue matching the ball's trajectory such that the ball stays between the fingers of the gripper, and finally close the gripper to catch the ball before it hits the ground.

*1) Ball Pose Filtering:* After obtaining the lists of poses and velocities representing the ball's trajectory, we filter out the poses that are physically out of reach for the Kuka arm either due to joint position or joint velocity constraints. To speed up that process, we remove all poses that are not within the Kuka arm's reachable hemisphere. Then we attempt to solve an inverse kinematics (IK) problem using Drake's built-in IK class for the remaining poses in the ball trajectory. We formulate this IK problem by adding a position constraint which says that the gripper position should match the ball position with an offset in the gripper's y-axis so that the ball does not hit the gray base of the gripper (see figure 1). We also add an angle constraint which forces the z-axis of the gripper frame to be aligned with the ball's velocity vector. This ensures that the gripper's fingers do not interfere with the ball's trajectory while finding the most optimal orientation around the z-axis. And finally, we add a bounding box constraint to ensure that the joint velocities to go from the previous robot configuration to the one IK is solving for does not exceed the Kuka joint velocity limits.

After adding the constraints, we try to solve the IK problem. If the IK solver is not able to find a feasible solution, we conclude that the pose we are solving for is not reachable and remove it from the list. After filtering out unreachable poses, we select the longest contiguous segment of reachable poses and use this as the region of the ball's trajectory where we

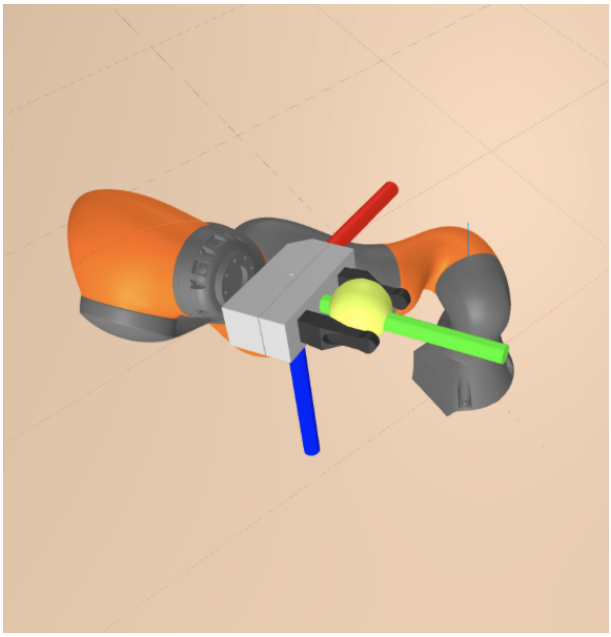

Fig. 1. This figure shows the y-axis offset between the gripper and the ball which ensures that the ball does not hit the gripper and bounce off unexpectedly. The gripper frame in this image has also been oriented such that the z-axis points downwards in the same direction as the ball's velocity.

attempt to catch the ball. Figure 2 shows each step of the filtering process.

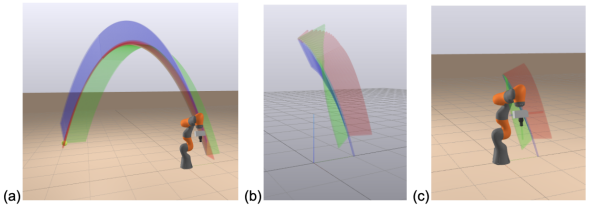


Fig. 2. (a) shows the entire predicted trajectory of the ball (b) shows the re-oriented and reachable pose segments of the original trajectory (c) shows the longest contiguous segment of poses from (b). The set of poses in (c) is the trajectory followed by the arm to catch the ball.

*2) Arm Trajectory Planning:* After using the pose filtering module to obtain a list of reachable poses along the ball's trajectory, we used Drake's 'MakePose' function to create a simple 'PiecewisePose' trajectory. We defined this trajectory such that the gripper would reach each pose in the filtered pose list at the exact same time as the ball. However, for most trajectories the robot has to move in one direction to reach the first pose in the filtered list and then make a significant change in direction to continue accelerating in the same direction as the ball. This proved difficult, even with a closed loop controller, and in the resulting system the gripper lagged significantly behind the ball after their initial interception at the beginning of the filtered pose region.

In an attempt to solve this problem we slightly modified the original trajectory such that the gripper would arrive at the first position on the ball's trajectory long before the ball got there. This gave the robot more time to initiate the change in direction and thus reduced the lag between the gripper and the ball. While this change had better results than the original, the lag was still significant enough to make gripping the ball a challenge.

Ultimately we settled on a trajectory such that the robot arm's timing along the trajectory gradually approached the ball's timing. As we did in our second attempt, we set the pose trajectory such that the gripper would arrive at the first position in the ball's trajectory before the ball arrived there. Specifically, we set this time to be 0.5 seconds after the simulation begins because the Kuka arm can theoretically reach any feasible position in that much time while still respecting joint velocity limits. We also constrained the gripper's trajectory such that it would reach the pose in the middle of the ball's trajectory at the same time as the ball. After this point it continues to match the ball's timing exactly, then grip the ball, and gradually slow the ball and arm to a stop. For all poses between the first pose and middle pose (after which the ball and gripper match) we set the timing such that it would smoothly approach the same timing as the ball. To do this we used the following exponential function where 'start' is 0.5 and 'end' is the time corresponding to the pose where the gripper starts to perfectly match the ball.

```
def approach(start, end, N, exponent = 1/10):
    x = np.linspace(start**(1/exponent),
                    end**(1/exponent),
                    N,
                    endpoint = False)
    return (x**exponent)
```

In order to slow the arm down so that it does not come to a sudden stop and cause the ball to slip out, we added a slow down segment for the last 30% of the trajectory. During this time the arm continues to follow the ball's poses but the timing is slowed down such that the arm reaches the final pose of the trajectory (20 cm above the ground) two seconds later than when the ball would have reached that location in free fall. This causes a more gradual deceleration and avoids the gripper losing the ball.

Figure 3 shows the time when the gripper arrival time and ball arrival time at each of the poses in the filtered list for a representative example. With this method we enforce smooth changes in velocity which minimizes the lag between the gripper and the ball. Ultimately this allowed us to nearly perfectly match the ball for the second half of the planned trajectory and successfully catch the ball.

*3) Kuka Arm Closed Loop Controller:* After the pose trajectory of the ball has been designed, it is sent to the Kuka arm's controller. While we optimized the trajectory to almost eliminate lag between the gripper and the ball, it was still important that we design a controller that would minimize lag as well.

We initially used an open loop inverse kinematics controller because it could solve for the robot's next joint position

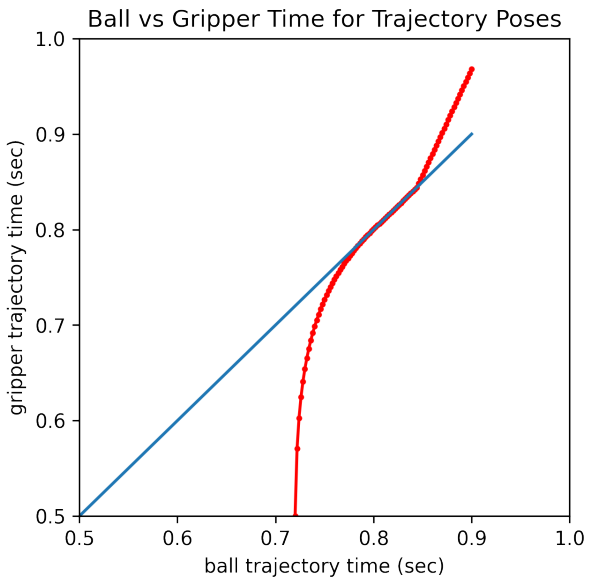


Fig. 3. Each point along the red line represents a position along the selected catching trajectory. The x-axis represents the time at which the ball reaches this position in free-fall and the y-axis represents the time at which the gripper arrives at this position. For the first part of the trajectory the gripper arrives at each position before the ball does, and slowly approaches the same timing as the ball. It then matches the ball's trajectory (the region where the blue and red lines overlap) and after grasping the ball, the gripper decelerates (the region where the red line is above the blue line).

directly from the trajectory input. This controller is simple and straightforward to implement, but it lacks robustness. When tested it produced jerky, sudden movements, and we still noticed a lag between the ball and the gripper.

In order to improve upon this controller setup we switched to a differential inverse kinematics controller with an integrator. As discussed in class, differential IK is generally more robust than IK and will produce smoother results with the Kuka arm. However, this itself was not enough to reduce the lag. In order to do this we decided to enable the Kuka arm's reference velocity port so that we could send both a joint position and acceleration signal to the robot. The position was calculated by integrating the solution of differential IK and the acceleration came from taking the derivative of the differential IK solution. This allowed us to essentially tell the robot not only where to go, but with what acceleration to get there. With these modifications to the first controller we saw significant improvements in the lag. However, the gripper was still slightly lagging behind the ball near the end of the trajectory.

Our final controller used closed-loop differential inverse kinematics. We took the same controller as in the previous section but closed the loop by adding position feedback. This was done by adding an "iiwa_position" input port to the differential IK controller module and connecting it to the output port of the iiwa robot. A schematic of our final closed-loop system can be seen in figure 4. Closing the loop made the system even more robust than the previous differential IK controller and we were able to achieve almost perfect trajectory matching with no lag between the gripper and the ball.

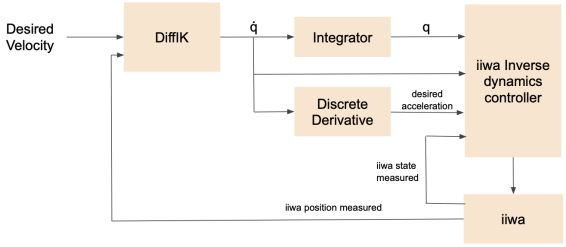


Fig. 4. This figure shows the block diagram of our closed-loop differential inverse kinematics (DiffIK) controller. This is the controller used to minimize lag as much as possible.

*4) Gripper Trajectory Planning and Controller:* The final component to catch the ball is determining the trajectory and controller for the gripper's fingers. Based on our trajectory design for the arm we know that during the whole second half of the arm's trajectory the gripper will be exactly matching the ball. Thus, we command the gripper to begin closing at the time when the ball and gripper first coincide perfectly and to be completely closed before the arm reaches the ground.

With this trajectory plan for the gripper's fingers we first noticed that the gripper closed at the correct time but it seemed to not be applying enough force because by the end of the trajectory the ball would slip out. In order to address this problem we modified the built-in Drake WSG position controller to have a significantly higher kp gain (increased from the default of 200 to the upper limit of 2000). This caused the gripper to clamp down much harder on the ball and reduce the risk of the ball slipping out. This updated controller, combined with the slowing down portion of the arm trajectory eliminates slipping for most catching scenarios.

### C. Throwing

In addition to constructing a system for catching a sphere, we set out to build a module that could throw the ball. Initially, we worked on generating a trajectory where the arm would be able to determine the appropriate release point and velocity based off of an end condition. However, this route proved to be difficult to achieve, such that all joint limitations were respected for the throwing controller. Building off of our catching module enabled us to generate a throwing system in which the robot arm caught a ball mid-flight and threw it back towards the starting position.

In order to implement the throwing environment, we identified a few key components that were necessary to build the system: ball projectile motion, arm trajectory, and gripper trajectory. We have identified key steps for these components and describe how our catching module could be used to capture this.

*1) Ball Projectile Motion:* Given a desired target position for the ball, we wanted to determine the appropriate trajectory

that would ensure that the ball landed at the specified location. In our initial implementation, we created a planar ballistic processor. This component solved a mathematical program to calculate the velocity at the release point using a predetermined release and target point. The method proved to be challenging due to the limitations of the Kuka arm's joint velocities.

Instead, we were able to adapt our catching module and apply the calculations here in order to derive the appropriate release velocity and time. We had collected the ball's instantaneous position and velocity by dropping the ball with the initial conditions specified. The throwback system uses the initial conditions of the catching module as the end condition for throwing, so we already obtained the release position and velocity through our arm trajectory. The ball was intended to retrace the same motion from before it landed in the gripper's fingers.

*2) Arm Trajectory:* To throw the ball to a desired end point, we needed to determine a path for the Kuka arm to follow from its initial position to the release point. For a catching and throwback system, the initial condition for the throwing segment would be the final position from the throwing trajectory. Our catching module generates an arm trajectory that essentially follows the pose of the ball. It only matches the ball's velocity towards the end of the motion, following the exponential function that was previously described. With this approach, we can retrace the trajectory to generate the same throwing motion of the ball. We ended up reversing the trajectory poses from the first system and applied it to the throwing segment in this system.

*3) Gripper Trajectory:* Once we had the release position and time for the gripper to let go of the ball, we could easily implement the gripper trajectory such that it started with applied force while gripping the ball and open after a calculated period of time. Since we already generated the gripper trajectory according to the ball trajectory in the catching environment, we simply reversed the gripper trajectory so that it released the ball at the same position that it initiated catching the ball.

### D. Combined Throwing and Catching

With our two modules, catching and throwback, we wanted to create a final system that combined throwing and catching into a single environment. In this new environment, we initialized two robot arms, both offset equidistant from the center. Our approach involved generating a symmetric trajectory between throwing and catching. We tried combining our systems previously described to the two Kuka arms. We intended to time the trajectories such that it could throw back and forth based off of time since the ball would be following the same path. We discovered a lag due to contact forces between the gripper and the ball that hindered the sphere from reaching the intended final location. Because of this, the arms could not easily throw back and forth without hardcoding specific poses of the ball after it was released from the gripper to inform the next arm on how to catch it.

## IV. Evaluation

Our evaluation focuses on analyzing both the catching and throwing systems respectively. In examining the capabilities of the system, we wanted to understand the robot arm's ability to effectively catch objects within its set radius of catching and throwing a ball back along the same catching trajectory.

### A. Catching

We generated a series of trajectories that would intersect one hemisphere of the robot arm's radius and measured the percentage of positions that the system was able to catch from each of these points. We ensured to specify trajectories that were guaranteed to land within the arm's reach since it would be uninformative to test positions outside of this realm. Our experiment measures the robustness of our system and catching limitations.

*1) Kuka Initial Position:* For each trial, the Kuka arm began in the default orientation as shown in the following figure.

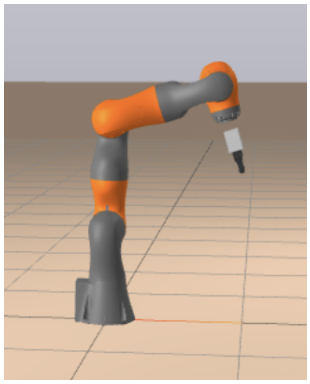

Fig. 5. This shows the default position of the arm before catching.

*2) Initial Conditions:* To ensure that the initial conditions set for each testing trajectory would result in the sphere's path intersecting the hemisphere, we began by initializing 100 ball poses around the radius of the Kuka. Each sphere was given a randomly sampled velocity that ensured that the ball was moving upward and outward so that it would come into contact with the ground outside of the hemisphere at the end of the trajectory.

From these starting poses around the hemisphere with the sampled velocities, the simulator ran for up to 1 second. Sampled spheres were eliminated if they exceeded the Kuka arm's limitations. If the ball hit the ground in less than 0.5 seconds, it was discarded from the experiment. This is because the Kuka needs 0.5 seconds to move to any point.

After this, we were able to identify our initial conditions for the remaining trials. Each ball had the initial position condition of where it landed, and the opposite velocity for when the simulation stopped. When running an experiment with these initial conditions, it ensures that the ball will intersect the arm's radius.

| Initial Position | Final Position | Euclidian Distance |
|---|---|---|
| [1.51, 0.94, 2.85] | [1.74, 0.99, 2.89] | 0.23 |
| [-0.02, 2.34, 0.57] | -0.21, 0.42, 0.03] | 1.99 |
| [-1.23, -2.71, 0.27] | [-1.33, -2.62, 0.33] | 0.14 |

Fig. 6. This figure shows three trajectories in the throwing experiment with the initial catching condition, final throwing position, and the distance between the two.

*3) Catching Conditions:* When running the trials, we implemented a condition to check that the z height of the ball in the world frame was less than 0.1 meters at the end of the simulation. After the simulation ran for the length of the trajectory plus 0.5 seconds, this condition was checked. By doing so, we were able to ensure that the ball was held by the gripper and not on the ground or redirected by bouncing off of the gripper.

*4) Results:* The sphere positions found in step two were used to run the experiment. We found that the gripper successfully caught the ball 33 times out of 78 valid trials, giving an accuracy of 42.3%.

### B. Throwing

The main objective in our evaluation was to measure how close the ball's throwing trajectory matched its motion in the catching trajectory. This would indicate how much our system was affected by other uncertainties such as slipping between the ball and gripper that could result in an asymmetry between the throwing and catching environments.

*1) Initial Conditions:* We used the initial trajectories that were successfully caught in the catching evaluation for the throwing experiments. The ball fell, was caught by the arm, and thrown back out.

*2) Trajectory Comparison:* To measure the distance between two trajectories, we compared the initial position of the ball before it was caught to the final position of the ball at the end of the simulation. This was because we wanted to see how close the catching trajectory matched the throwing trajectory. If they were exactly the same, we would expect the ball to be at the same position at the initial condition after the simulation. The length of the simulation is double the time for the catching trajectory plus the pause time defined between the throwing and catching segments.

*3) Results:* Due to unforeseen limitations imposed by the Deepnote platform, we were unable to run a significant sample of test trials for the throwing system. While we do not have thorough and robust results, we showcase our outcome from a few representative samples.

The average distance for our small sample is about 0.790 meters.

This figure shows three trajectories in the throwing experiment with the initial catching condition, final throwing position, and the distance between the two

## V. Discussion

After conducting the experiments for both throwing and catching, we were able to further analyze our results and find areas for improvement. Though our results are limited, we have intentions of continuing our analysis to understand specific positions and velocities that cause our catching system to fail and examining the nonlinear contact forces between the gripper and ball by measuring its effect on the throwing trajectory with a more thorough set of trials.

### A. Catching

Our results indicate that we were able to catch nearly half of the possible initial conditions of a sphere that would land within the arm's catching radius. Through manual inspection, we identified two cases in which our system faces challenges in successfully grasping the sphere: joint velocity constraints and gripper controller constraints.

*1) Joint Velocity Constraints:* The Kuka arm has set joint velocity constraints that must be respected. Some of the valid test trials that we conducted contained high velocity and close positions that made it impossible for the arm to follow the expected trajectory with the given amount of time.

*2) Gripper Force Constraints:* In the other case where the arm failed to catch the sphere, we noticed that the arm was able to follow the correct trajectory and reach the correct position to catch the ball. However, the ball slipped through the gripper's fingers when it travelled with a high velocity. The gripper's grasp is not refined enough to successfully catch a wide range of trajectories with high velocities. Due to limitations in the controller and force constraints, we were only able to catch balls with limited velocities depending on the position.

### B. Throwing

Though we had a limited sample for our throwing experiment, we noticed a difference in position between the throwing end position and catching initial position. We see this results from a lack of robustness in the catching system and a lag due to contact forces during the throwing trajectory.

*1) Catching System Failures:* As shown in our catching accuracy, the system could use some improvements to improve its overall robustness and success rate to best catch the ball. We found specific cases that show our system's ability to catch the ball but without a perfect grasp. The figure below shows one example of this case, and the initial conditions can be located in the table from the Results.

The gripper catches the ball, but we can see it slightly slip through the gripper's fingers. When the arm attempts to throw the ball, it begins to slip and roll down the gripper. Because of this, the tossing trajectory looks drastically different than the catching trajectory, explaining the wide difference in the z coordinates between the two positions.

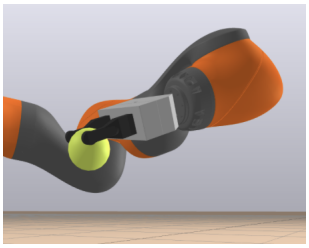

Fig. 7. The ball is slipping from gripper's fingers once it has been caught and just before it is thrown.

*2) Contact Forces:* Due to the contact forces between the sphere and the gripper, the throwing trajectory does not completely match the reverse of the catching trajectory. These forces cause a delay in the positions and may reduce the velocity at which the ball is moving. With this, the final location of the ball is slightly before the intended position.

## VI. Conclusion

In this work, we demonstrated an end-to-end system in Drake that can throw and catch simple, uniform symmetric objects from a variety of initial positions and velocities. In order to achieve this, we experimented with several different trajectory planning methods and Kuka arm controllers. Our final system uses a gradual timing approach to smoothly reach the ball's trajectory, match it, and then catch the ball. Our closed-loop differential inverse kinematics controller minimized lag between the commanded trajectory and the plant's actual trajectory to ensure that the gripper did not lag behind the ball. Accurately tracking the ball allowed us to robustly grip and catch the ball with a lower chance of slipping than with other methods we explored.

Quantitative analysis of our catching system showed that it can successfully catch a sphere 42.3% of the time, assuming the initial conditions of the sphere allowed it to come within reach of the Kuka arm during any point of the ball's trajectory.

Our throwing system leverages all of the findings from the catching system by reversing the catching trajectory to plan a throwing trajectory. The throwing accuracy is slightly lower than that of the catching system due to the non-linear contact forces between the gripper and the ball. These forces, which are difficult to model, often result in the ball slipping away from the center of the gripper which caused changes in the release position and velocity of the ball.

As discussed earlier in the paper, our system could be improved by incorporating adaptive control methods which would allow us to manipulate more complex objects. Further improvements could be made by modeling or learning the contact forces between the gripper and the ball. This would allow for a more robust, closed-loop throwing system that is not affected by slipping. Finally, our system assumes omniscient knowledge of the ball's trajectory at all times. To make this system more realistic and usable, we would need to develop a perception module that can dynamically make predictions about the object's current and future state.


## Acknowledgments

We would like to thank the course staff for 6.843/6.800 for their guidance and support throughout this project.


## Contributions

David von Wrangel: At first I worked on a separate throwing controller and the optimization for the IK based direct catching interception. Then I contributed to the entire stack of trajectory matching via DiffIK, closed loop DiffIK, Sphere trajectory retrieval, IK based trajectory optimization and filtering, continues re sampling of time breaks. After generalizing the catching system, I wanted to work on the throwing controller when I noticed that time reversal would make the entire catching controller reusable. So, I developed the throwback controller, worked on the multi arm throw and catch and attempted building a semi closed loop trajectory re planning algorithm. I helped building the code for the evaluation section of this paper.

Ria Sonecha: I first worked on our intercept/IK based catching system. When we realized that was not a robust solution I worked on the trajectory matching-based catching system (ball pose estimation, arm and gripper trajectory planning, and arm/gripper control loops). I also dedicated a lot of time to the paper which involved getting intermediate feedback from my CI instructor, and creating figures for the paper.

Shreya Pandit: I contributed to figuring out the arm trajectory, gripper trajectory, and evaluation. Additionally, I worked on writing the and creating videos for the final report.

## References


[1] J.-J. E. Slotine, W. Li, "On the adaptive control of robot manipulators," The international journal of robotics research, vol. 6, no. 3, pp. 49–59, 1987.

[2] Kim S, Shukla A, Billard A, "Catching objects in flight," IEEE Transactions on Robotics, vol. 30, no. 5, 1049-65, May 2014.

[3] Zeng A, Song S, Lee J, Rodriguez A, Funkhouser T. "TossingBot: Learning to throw arbitrary objects with residual physics," IEEE Transactions on Robotics, vol. 36, no. 4, 1307-19, Jun 2020.