# Soft-Catching a Projectile with a Robotic Arm

1st Fischer Moseley
*Physics, Electrical Engineering and Computer Science*
*Massachusetts Institute of Technology*
Cambridge, USA
fischerm@mit.edu

2nd Daniel Adebi
*Electrical Engineering and Computer Science*
*Massachusetts Institute of Technology*
Cambridge, USA
ikadebi@mit.edu

***Abstract*—Throwing and catching between robots allows for an extension of their workspace without additional hardware, but is difficult to implement with the low-dexterity grippers that are popular in picking robots. A new controller for a ball-catching robot is presented, with the motion planned largely via a trajectory optimization formulated as a direct collocation problem. Although a full implementation of the designed controller was not completed, the optimizer is able to track a projectile as it follows a ballistic motion through the air. Consideration is given to the limitations of the proposed controller, as well as opportunities for further work and integration with throwing controllers for a complete object-passing solution.**

***Index Terms*—robotics, nonlinear control theory, trajectory optimization, robot kinematics, optimal control, robotic control, manipulator dynamics.**

## I. Introduction

Throwing and catching provides robots with a simple mechanism of extending their workspace. A team of robots can pass objects to each other through the air far faster than they could be transported on the ground by a wheeled or legged robot. Throwing is a relatively simple task - it is a conceptually straightforward matter to choose a ballistic path for a projectile to reach a target, and controllers have been devised to impart such a path onto a projectile. Princeton's TossingBot was able to learn to throw via reinforcement learning in 2019, where it picked objects from one bin and threw them into another. [1] This only allows for passing objects in one direction, but if a robot was able to both throw and catch with the same end-effector then it could pass objects through the air faster then they could be transported otherwise.

Catching is hard problem, requiring that estimation of projectile state and the resulting motion planning be done incredibly quickly. Most catching robots accommodate this by choosing their end-effector geometry to make the task easier. Multi-fingered hands are popular, as they stop projectiles instantly as they hit the palm of the hand. This is referred to as 'hard' catching, as the end effector must be in only the right *place* at the right time.

By restricting the end effector to be a two-fingered gripper, we remove the aid of the palm in stopping the object. With the simpler gripper, we must use the robot's motion to decelerate the ball after it has been grasped. This is referred to as 'soft' catching, and it requires that the end effector must be in the right *place* and moving at the right *speed* at the right time. This contact must occur within an exceptionally narrow contact patch on the gripper.

Fig. 1: Princeton's TossingBot, which picks objects from a bin with a 6DoF arm and tosses them into a bin mounted on another arm.

The bulk of the motion is planned via trajectory optimization formulated as a direct collocation problem; with a few additional steps to close in on the object. Although the scope of this project is large, there are considerable simplifications being made. We choose to focus only on the motion planning, grasp selection, and manipulator dynamics of the problem. Accordingly, we ignored the challenges of estimating ball state or planning a trajectory in real-time - we obtained perfect knowledge of the ball state through our simulator, which we could pause for computations at any time. We also restrict our projectile to a spherical ball to simplify grasp generation. Solving the perception and pose estimation problem for arbitrary projectiles would be a considerable undertaking in its own right, but would be necessary for a complete solution of the catching problem.

## II. Related Work

Although difficult to engineer, complete end-to-end catching solutions do exist. Seungsu Kim's thesis provides a controller for hard-catching arbitrary objects using an Allegro hand, that uses human demonstrations for motion planning. A motion-capture setup is used for perception, and machine learning to predict object trajectories. [2] [3] Soft-catching has also been demonstrated on the same hardware by Salehian, Khoramshahi, and Billard using a Linear Parameter Varying

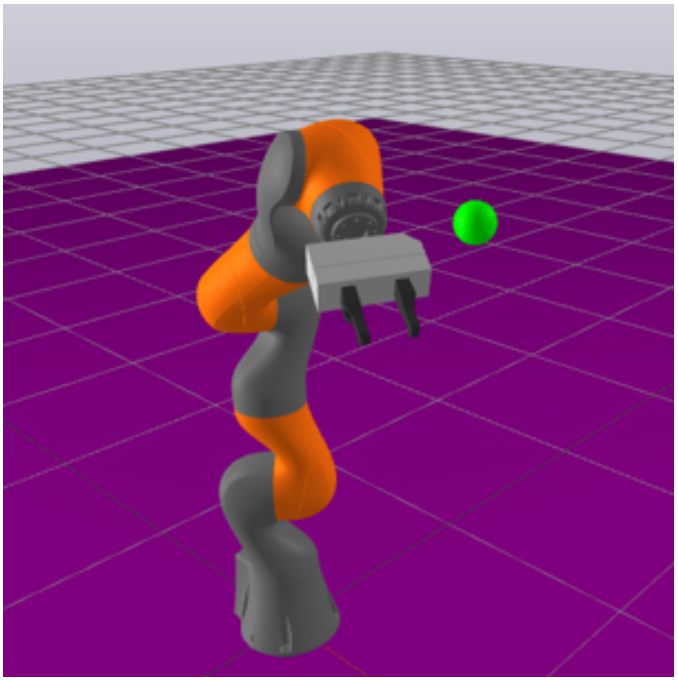
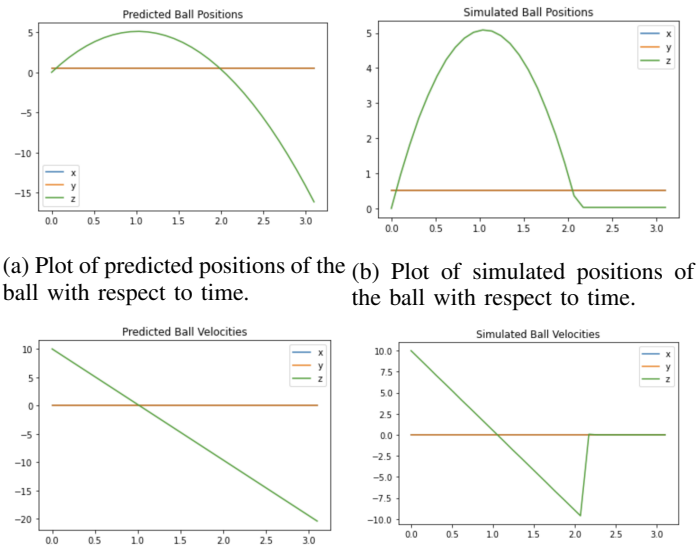


Fig. 2: The full manipulation station setup which includes the KUKA iiwa arm, a planar floor, and a ball to catch.




(a) Plot of predicted positions of the ball with respect to time.

(b) Plot of simulated positions of the ball with respect to time.

(c) Plot of predicted velocities of the ball with respect to time.

(d) Plot of simulated velocities of the ball with respect to time.

Fig. 3: Figures 3a and 3b show that the positions that we produced based off of our prediction match what we expect in the simulation, with the only exception being around the 2 second mark where the ball hits the ground. Likewise, the predicted velocities in Figures 3c and 3d give similarly expected results.

(LPV) system. [4] Bäuml, Wimböck, and Hirzinger implement a semi-soft catching controller for a dextrous hand, using trajectory optimization for motion planning. [5] Linderoth used a box with a small hole in it for an end effector, and as a result was able to perform motion planning faster than real-time. [6]

Controllers that include both catching and throwing have also been demonstrated. Klober, Glisson, and Mistry created a controller that allows a humanoid to play catch or juggle with a human, which uses a depth-camera based perception system. [7] Given that the robot was an animatronic, this used a dextrous hand and so the hard-catching approach they demonstrated was a natural choice.

The controller developed by Tony Wang and Daniel Yang for MIT's 6.881 also provides an excellent reference for the throwing problem. We use the same hardware setup as is in their work, and unfortunately were not able to integrate their controller into our own. [8]

## III. Methods

The hardware setup is shown in Figure 2. Our motion planning approach has two main components: an offline motion planner, and real-time controller. The offline component uses trajectory optimization to find a minimum-time solution to intercepting the ball, and also finds the largest continuous region where the ball is within some distance $\delta_{bg}$ of the gripper. The time at which this 'intercept region' is entered is saved, and used in the real-time controller. A grasp pose is also selected such that the velocity vector of the ball at this time is normal to the plane of the gripper.

After the offline component runs, the simulation is started and the realtime component activated. This feeds the optimizer's joint-space trajectory to a position controller on the iiwa, which is done until the arm enters the intercept region. At this point the ball's position is fed to the position controller, after inverse kinematics is used to obtain setpoints in joint-space. Once the gripper comes within some distance $\epsilon_{bg}$ of the ball, the gripper is commanded closed and the ball is caught.

The trajectory optimization in the offline component is formatted as a Direct Collocation problem in Drake. [9] [10] Bounding-box constraints on the iiwa's joint angles, velocities, and torques are added to the problem, along with a cost function equal to the squared distance between the gripper and the ball:

$$\ell(\mathbf{x_{gripper}}, \mathbf{x_{ball}}, t) = ||\mathbf{x_{gripper}}[t] - \mathbf{x_{ball}}[t]||^2$$

where $\mathbf{x_{ball}}$ is determined by analytically solving for the ballistic motion of the ball given its initial position and velocity. Since the trajectory is analytic, this allows the gradient of the loss function to be computed analytically with auto-differentiation. In our simulation, we added a planar floor for convenience - if we failed to catch a ball, it's a lot easier to see where it went! This wasn't modelled in the analytic solution for the ball trajectory, we wanted to keep the trajectory (and therefore the cost function) as smooth as possible. This was considered a reasonable approximation, as we wanted to consider catches that occur before the ball contacts the ground.

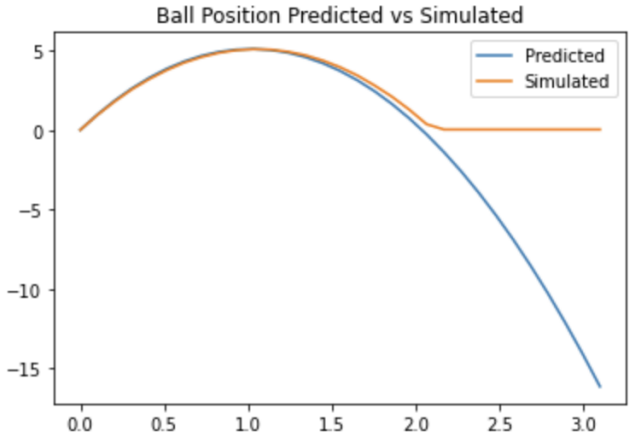



Fig. 4: A graph comparing the predicted and simulated trajectories of the ball's position on the z-axis with respect to time. Note that a little bit after 2 seconds, the ball lands on the ground, and our ball trajectory prediction doesn't take into account what happens when the ball hits an obstacle.

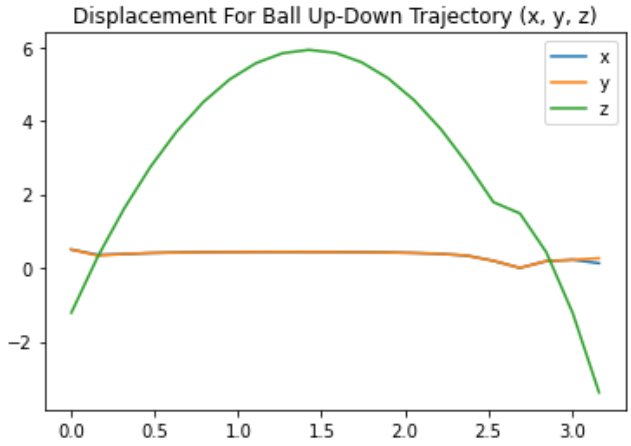



Fig. 5: A graph showing the $x, y$, and $z$ displacements between the end effector of the robot and the ball, given that the ball just goes straight up with a velocity of 10 m/s.

## IV. Results

### A. Predicting Ball Trajectories

In order to make sure that our robot could effectively follow the trajectory of a moving object, we needed to be able to actually predict the motion of the object itself to see if our values match. For the purposes of this project, the trajectories of the ball we used were either straight up, or simple parabolas that can be predicted using simple classical mechanics. The ball is given initial starting position and velocities, and with this information, we can calculate the positions and velocities of the ball at any time $t$, which means that each ball position $\mathbf{x_{ball}}[t]$ and velocity $\mathbf{v_{ball}}[t]$ at a given time $t$w can be represented as follows:

$$\mathbf{x_{ball}}[t] = \mathbf{x_{ball}}[t_0] + \mathbf{v_{ball}}[t_0] * t + \begin{bmatrix} 0 \\ 0 \\ \frac{1}{2}gt^2 \end{bmatrix}$$

$$\mathbf{v_{ball}}[t] = \mathbf{v_{ball}}[t_0] + gt$$

From the graphs in Figure 3, we can see that our predicted results match what we get in our simulated results, except for when the ball hits the ground in the simulation and just ends up rolling. We also have a graph that compares our predicted and simulated trajectories in Figure 4.

### B. Robustness of Method With Different Ball Trajectories

After predicting the ball trajectories, we then performed our trajectory optimization with various initial conditions for our starting ball state to determine how robust our methods were. Placing our robot at the origin, we made sure giving our robot multiple ball trajectories for our robot to follow, varying from the ball going straight up and down, to more parabolic trajectories. Our ultimate goal is to minimize the displacements between our gripper and the ball, even if the gripper would not be in reach of the ball to grab it.

The first trajectory we tested was a ball going straight up and down, where we gave the ball an initial position of $\mathbf{x_{ball}}[t_0] = (1, 1, 1)$, and an initial velocity of 10 m/s up in the $z$ direction. Figure 5 shows the displacement between the end effector of the robot and the ball, and we can notice that for this specific trajectory, the robot can follow the $x$ and $y$ positions of the ball well, and achieves a minimal $z$ distance given that the ball is well out of reach of the arm at this point. When the ball lands again, the $z$ displacement goes to 0 until the ball goes below where the robot can reach. If we managed to get our robot to catch the ball like we originally had planned for this project, we'd want to activate the actuator around the 2.5 second mark, as the ball is nearing the end effector at this point in time.

The next trajectory we tested was for the ball to go in a parabola from the left to the right side of the $x$-axis, and the results for the displacement here can be seen in Figure 6. This example has the ball start off at the point $\mathbf{x_{ball}}[t_0] = (-2, 1, 1)$, and has an initial velocity of $\mathbf{v_{ball}}[t_0] = (2, 0, 8)$. Here, we can notice that the robot can still follow the ball's position relatively well, it struggled a bit more to follow the ball's $x$-position, as we would hope that there would be a flatter line to represent the lack of displacement change along the $x$-axis, but unfortunately, this wasn't the case. This most likely could have been caused by the values we chose when creating our constraints for our torques to restrict the movements of different joints in our robotic arm.

The third and final trajectory we tested was for the ball to go in a parabola over the robot itself across the $x$ and $y$ axes. The graph from Figure 7 makes it initially seem like the tracking did about as well as in the previous trajectory. But looking at the visual results, it seemed like the robot actually performed somewhat better tracking this trajectory than the previous one, and this might have been because moving the arm around to

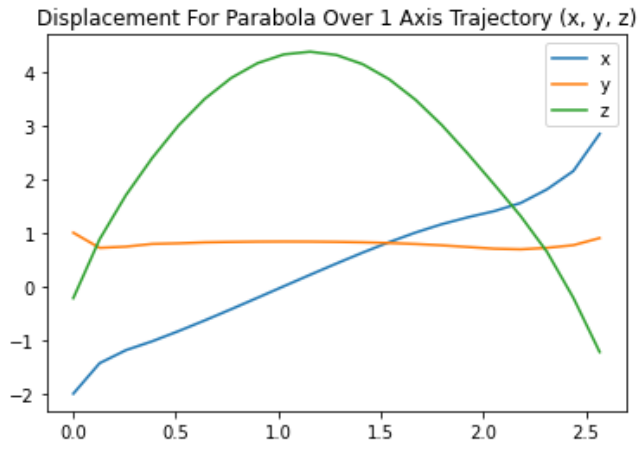



Fig. 6: A graph showing the $x, y$, and $z$ displacements between the end effector of the robot and the ball, given that the ball just goes on a parabolic path from the left side to the right side of the $x$-axis, with a $x$-velocity of 2 m/s and a $z$-velocity of 8 m/s.

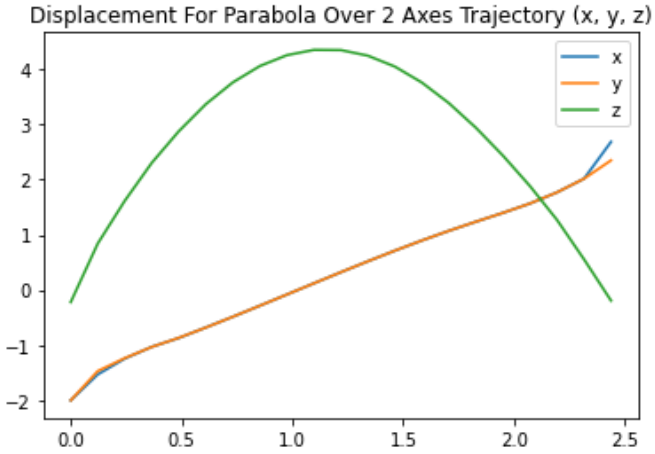



Fig. 7: A graph showing the $x, y$, and $z$ displacements between the end effector of the robot and the ball, given that the ball just goes on a parabolic path from the left side to the right side of the $x$-axis, with a $x$-velocity and $y$-velocity of 2 m/s and a $z$-velocity of 8 m/s.

track this trajectory just might have been easier for the robot to accomplish. It also may have just been because our robot didn't have to worry about as many of its constraints going from one quadrant of the $xy$-plane to the opposite quadrant. Either way, this was an interesting observation to note.

### C. Infeasible Trajectories

Most of the trajectories that we gave our ball were pretty feasible for our robot to track. The only issues arose when the majority of the ball's trajectory was too close to the robot, more specifically, within 0.5 meters of the origin nearby one of the robot's joints, and not above the robot's end effector. In these situations, the robot would often collapse in on itself, and joints would end up going inside each other. Either this occurred, or the problem would be ruled as being infeasible altogether by the Direct Allocation algorithm.

## V. Further Work

Even if fully implemented, our controller would only be a partial solution to the catching problem. The ball state is assumed to be known perfectly, which is impractical. A perception system is required, which may pose additional constraints on the motion planning system to satisfy additional requirements over the gripper's path. Notably, if an optical perception system is used for state estimation, the gripper may occlude the object on approach - right when a good state estimate is most critical. The ball position may have to be predicted over an interval over which it is not visible, and the work presented in Kim's thesis for estimating the states of projectiles with nontrivial geometries is incredibly relevant. [2]

We also would like to add some heuristics to speed up our trajectory optimization. For example, a pre-processing step to 'screen' the ball's trajectory and ensure it will be in the arm's dynamic workspace for long enough to catch. This could be achieved by forward-integrating the ball's trajectory, and determining how long it passes through the robot's workspace.

Another heuristic could be used for steepening the gradient of the optimization problem. When the ball is far from the robot, the straight-line distance to the gripper does not change much for different poses of the arm. This leads to shallow gradients, increasing solve time for intercepting a ball trajectory that may not even pass through the robot's kinematic workspace. If a ball is far from the robot but will eventually pass through its workspace and become interceptable, it would be possible to 'fast-forward' the ball's trajectory to right before it becomes interceptable. This would effectively be incorporating an offset time $t_o$ into the cost function:

$$\ell(\mathbf{x_{gripper}}, \mathbf{x_{ball}}, t) = ||\mathbf{x_{gripper}}[t] - \mathbf{x_{ball}}[t - t_o]||^2$$

Where the offset time would be determined by forward-integrating the ball's trajectory and computing its distance to the edge of the robot's workspace.

## VI. Conclusion

We were able to formulate a Direct Collocation problem to track a ballistic projectile. We managed to get a pretty successful trajectory tracking system too, although the performance was not nearly as great when the ball was at a point where it was too close to the robotic arm's joints. While catching the ball was our initial goal, optimization is used for the bulk of the motion planning onboard.

## VII. Individual Contributions

Fischer formulated the trajectory optimization, wrote all simulations, and composed the final presentation. He also wrote the final report, save the results and conclusions sections. Daniel experimented with different trajectories based off of different initial conditions of the ball and wrote the results and conclusion sections of the paper.

## VIII. Source Code

The code used for this analysis can be found at https://github.com/FischerMoseley/yoink. The associated Deepnote project is linked there, and can be duplicated and run by anyone curious.

## References


[1] Zeng, Andy and Song, Shuran and Lee, Johnny and Rodriguez, Alberto and Funkhouser, Thomas, TossingBot: Learning to Throw Arbitrary Objects with Residual Physics, Proceedings of Robotics: Science and Systems (RSS), 2019.

[2] S. Kim, "Rapid and reactive robot control framework for catching objects in flight," thesis, EPFL, 2014.

[3] S. Kim, A. Shukla, A. Billard, "Catching Objects in Flight," IEEE Transactions On Robotics, Vol. 30, No. 5, October 2014.

[4] S. Salehian, M. Khoramshahi, A. Billard, "A Dynamical System Approach for Catching Softly a Flying Object: Theory and Experiment," EPFL, 2016.

[5] B. Bäuml, T. Wimböck, G. Hirzinger, "Kinematically Optimal Catching a Flying Ball with a Hand-Arm-System," IEEE International Workshop on Intelligent Robots and Systems, 2010.

[6] M. Linderoth, "On Robotic Work-Space Sensing and Control," thesis, Lund University, 2013.

[7] J. Kober, M. Glisson, M. Mistry, "Playing Catch and Juggling with a Humanoid Robot," IEEE-RAS International Conference on Humanoid Robotics, 2012.

[8] Dxyang/manipulation. GitHub. (n.d.). Retrieved December 8, 2021, from https://github.com/dxyang/manipulation.

[9] R. Hargraves and S. W. Paris. "Direct trajectory optimization using nonlinear programming and collocation," Journal of Guidance, Control, and Dynamics, 10(4):338-342, July-August 1987

[10] R. Tedrake. *Underactuated Robotics: Algorithms for Walking, Running, Swimming, Flying, and Manipulation (Course Notes for MIT 6.832)*. Downloaded on Dec 3, 2021 from http://underactuated.mit.edu/