BatterBot: A Case Study in Trajectory Optimization for Nonprehensile Manipulation

1st Yajvan Ravan
EECS
Massachusetts Institute of Technology
Cambridge, USA
yravan@mit.edu

2nd Felix Huang
EECS
Massachusetts Institute of Technology
Cambridge, USA
fhuang25@mit.edu

3rd Stephen Hong
EECS
Massachusetts Institute of Technology
Cambridge, USA
sshong@mit.edu

***Abstract*—Baseball is a highly dynamic and complex sport that is difficult to play, even for humans. This project aims to isolate and investigate the actions of pitching and hitting into a robotic setting, utilizing the lessons and material covered in 6.4210 Robotic Manipulation. Specifically, this project implements a system consisting of two robotic arms, one that tosses baseballs and one that swings a bat, to perform the pitch and hit actions in simulation. Through various trajectory optimization methods, we evaluate the nuances and tradeoffs involved in throwing and hitting a ball with a robot. We implement inverse kinematics motion planning as well as two different types of kinematic trajectory optimization. We demonstrate an iterative method to use initial guesses to solve these optimizations in the complex joint space of this problem. We find that BatterBot can pitch with 60% accuracy within the strike zone from 8 m away, and has less than 5mm of batting error at bat speeds less than 4m/s. Both the batter and pitcher use the same optimization frameworks and parameters, indicating the promise of such techniques for nonprehensile manipulation.**

I. Introduction

Robotic systems that are reactive to objects in motion have become increasingly relevant as we attempt to implement robots into scenarios that are dynamic, uncertain, and as robust as possible. Two such scenarios that fall into this category are hitting an aerial projectile in motion and predictably tossing an aerial projectile. The robotic system must react to the trajectory of the incoming projectile to estimate where the projectile will end up to accurately hit it. Both tasks involve significant challenges in inverse kinematics, as in addition to controlling position, we also must control the velocity of the projectile for both tasks. Currently, there is limited research done in the field of hitting aerial objects in motion. In this project, we hope to explore this field and gain insight into the tasks of dynamic object perception and hitting aerial projectiles.

Our system, BatterBot, simulates a simplified game of baseball in which one robot arm pitches the ball and another robot bats the ball. To pitch the baseball, the pitcher robot chooses a randomly desired position within the "strike zone" and calculates and executes a trajectory that can be generated, within robot constraints, to aim for that position. To bat the ball, the batter robot uses knowledge of the initial position and velocity of the ball and determines a trajectory to meet the ball at a desired position with the desired velocity.

Thus, we hypothesize that by using just position and velocity, we can determine the optimal trajectory to throw a ball and also determine the optimal point of contact for the bat to collide with the ball. The goals of our design are as follows. First, we want to minimize the error on any given throw by measuring the magnitude of the distance between the ball's position before the bat swing to the desired strike zone area. Second, we want to minimize the error on any given bat swing by calculating the magnitude of distance between the simulated bat swing and the desired bat swing position. This allows us to assess the performance of our Batterbot as we can analyze the performance with binary results (thrown ball lands in or out of the strike zone; bat makes contact or does not make contact with the ball) as well as more advanced metrics that can quantify the percent error based on the magnitude of distance between real-time and desired positions.

There are several useful applications of a robotic system capable of hitting moving objects in the air. An obvious application of BatterBot is a robotic system that can assist baseball players in practicing without a partner. The pitcher robot can throw baseballs at a human batter to practice their swings and the the batter robot can hit baseballs to human players that want to practice their catching. Furthermore, the work done to expand our observations of dynamic object manipulation and the kinematic trajectory optimization needed to make it predictable applies to any robotic system that needs to react to external stimuli with high velocity and predictability. Finally, our work shows the transference of these techniques between batting and pitching, demonstrating the robustness of our methods.

II. Related Work

Several prior works discuss approaches for tossing and hitting an object (e.g. ball) with a robotic arm. One of the papers referenced for this project was the experiment conducted by Senoo et al. [1] which analyzes the performance of their most recently developed baseball robots. The paper dives into how the integration of the system's components (high-speed hand-arm, a high-speed bipedal mechanism, and high-speed vision) allows the robot to perform feedback-based reactive motion in real time such as catching, batting, throwing, and running. While the feedback-based reactive motion allows the robot to move at high speeds, hardware, and control limitations arose from this study due to the robot experiencing a greater impact of force at the point of contact, thus having poor shock absorption. However, aspects of this study such as the high-speed active vision for tracking dynamic, aerial objects, and the collision model to determine arm (bat) swing trajectory were influential in how we designed our robot. It was very useful to gain an intuitive understanding of the parameters we must consider when creating a baseball robot and the drawbacks of certain technical approaches.

Another work referenced for this project was the paper published by Hsiao et al. [2] which dives deep into the hand-eye coordination techniques for robots, specifically in the case of ball-batting. The main focus of this paper centers around not only making clean contact with the incoming ball but also directing the rebounding ball to a specific target location once it leaves the bat. This adds a level to the design complexity as the robot must calculate the outward trajectory of its projectile as well. This paper is extremely useful for our project as the author's results show that their perception and deep learning models were capable of having 0% swing-and-miss error. In our project, we follow the same framework of a constrained optimization problem where we also aim to minimize our swing-and-miss error through vision tracking through perception and decision-making through deep learning models.

The third paper referenced for this project was authored by Gardner et al. [3] which expands on a simulated robot arm that tosses a ball at certain target distances. The paper dives into the challenges that people without arms face and how robotic manipulation coupled with deep learning networks can help alleviate these challenges. By constraining their simulated robotic arm to have the same properties as a normal human arm, the researchers made the deep learning algorithm as realistic as possible to account for possible implementation in a real-world setting. Through the use of an LSTM, or a recurrent neural network (RNN) that can learn trends and long-term dependencies in information over an extended number of time steps, the researchers were able to achieve a training accuracy of 97.9% within a circle with a 5cm radius and 58.7% within a circle with a 1 cm radius when tossing a ball. While our project focuses on making contact after landing within our target strike zone, the intuition about tossing accuracy and trajectory optimization gained from this paper proved useful for our work.

III. Methods

In this section, we introduce our methods and iteration process for each of the components of this system. Detailed results of intermediate steps are included in the next section.

A. Simulation Environment

To begin the project, we set up our simulation using PyDrake [4] and used Deepnote as the development pipeline. We chose these tools as we have become familiar with them throughout the course. Our simulation includes two Kuka LBR

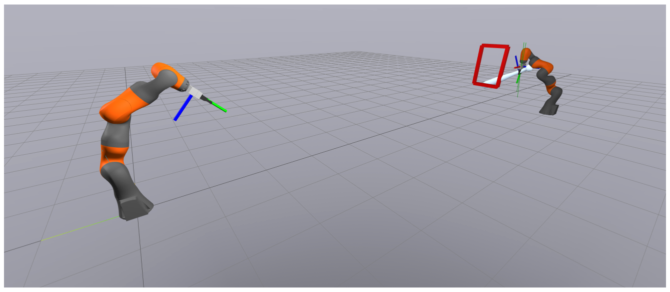

Fig. 1. Initialization of our simulation setup with the iiwa "pitching" arm (left) as well as the iiwa "batting" arm (right). The red strike zone has no physical properties and is shown purely for visualization.

iiwa robot arms, each equipped with a Schunk WSG gripper. The SDF and OBJ files of the iiwas, grippers, and ball are sourced from Drake. We believe that the joint-torque sensing and control of the iiwa arm and the simplicity of the Schunk gripper are suitable for our manipulation task of pitching and batting a baseball. In our simulation, we generated a baseball bat using an OBJ file sourced from TurboSquid. The inertial and geometric properties of the SDFs for these objects had to be adjusted to fit appropriately into our simulation by scaling down the size and mass so that it could be handled predictably by the simple 2-fingered Gripper. In addition, the SDF of the bat was modified to add an extra link at the handle to allow for ease of picking up from the ground. We also wrote a custom SDF for the ground that the ball and the bat rest on and a strike zone box for visualization. Visualization can be seen in Fig. 1.

For the custom ground, the strike zone box, and the two iiwa arms, we fixed their locations in the simulation. These objects are intended to be stationary so we keep their location fixed. The ball and the bat are allowed to move freely as they are to be manipulated in the simulation. Note that the strike zone box has no physical or collision properties since it is used for visualization purposes only.

As we mention below, we found it necessary to increase the friction coefficient on the gripper to prevent slipping of the ball and bat and make their manipulation more predictable. Although this doesn't translate exactly to reality, we can get similar results through methods such as increasing the torque on the gripper, changing the gripper material, or changing the bat/ball material. Thus, increasing the friction coefficient serves as a quick and approximate heuristic.

B. Pitching

1) Picking: We split the pitching motion into a pick and then throw. We assume that the arm knows exactly where the ball is placed. Given this, we define 5 key frames necessary to execute the pick action, namely the initial pose, a pose above the ball, a pose around the ball, a post-pick pose above the ball, and then a goal pose at roughly 0.3 m height. A visualization can be seen in Fig. 2.

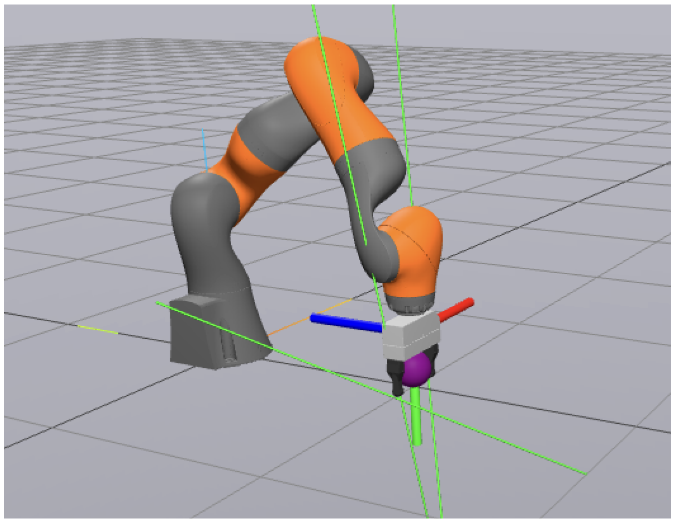
Fig. 2. Depicted here is the gripper picking up the ball. This antipodal grasp maximizes the contact area between the ball and gripper which allows for a more secure hold.

Our first method of picking involved generating a trajectory of poses using linear interpolation between the key frames. Taking the derivative of this trajectory gives us a trajectory of spatial velocities. We then used a differential inverse kinematic (DIK) controller with the equation below:

$$V_G^W = J(q) * \dot{q} \tag{1}$$

where q represents the joint angles and V_G^W is the spatial velocity. We then convert $\dot{q}$ to q using integration. This technique was effective in generating smooth trajectories. However, after the addition of a second iiwa and more free bodies into our simulation, computing the Jacobian, i.e. the time derivative of the forward kinematic function, became computationally expensive, slowing down the simulation.

Our second approach used optimization-based inverse kinematics. In this iteration, we used the 5 key gripper frames and performed inverse kinematics (IK) for all the poses. Given that the iiwa has 7 joints, this was a highly unconstrained environment, with multiple joint angles mapping to the same pose. However, we wanted to have a relatively clean and smooth movement, so we chose to penalize the cost between consecutive points. Furthermore, setting a good initial guess for the optimization program was important. The first key pose corresponded to the initial joint angles (q_0). To perform IK for the next pose, we use the joint angles from the previous computation as the initial guess and repeat until we have all 5. Finally, we linearly interpolate between these 5 points in joint space to define the trajectory. An example optimization is depicted below:

$$\begin{aligned} \min_{q_{PrePick}} \quad & |q_{PrePick} - q_{Initial}|_2^2 \\ \text{subject to} \quad & X_{PrePick} = f_{kin}(q_{PrePick}) \end{aligned} \tag{2}$$

2) Throwing: After executing the pick action, we focus on throwing the ball at a defined velocity and position in the world frame. A visualization can be seen in Fig. 3. To simplify our calculation, we assume that the ball is rigidly fixed to the gripper. Therefore, we can simply transform the velocity of the ball into a desired velocity (V_d) and a desired position (p_d) of the gripper. As we talk about below, however, this assumption is not always true. Using V_d and p_d we determine a trajectory for the gripper as before. However, a key difference here is the additional velocity constraint compared to picking. Let us note that such a trajectory has a large number of degrees of freedom, particularly time and joint angles. Furthermore, this trajectory must lie within the entire set of positions available to the robot and must respect hardware velocity and acceleration constraints.

In our first approach, we imitated the key frame approach from above. To ensure the velocity of the ball, we first defined a line of fixed distance, aligned with the direction of V_d, and positioned the midpoint of the line at p_d. We then this line as a "launchpad" for our gripper, i.e. the gripper starts with velocity 0 at the start of the line and proceeds with constant acceleration to achieve velocity V_d at p_d, and then decelerates, ending with 0 velocity at the end of the line. See Fig. 3 for an example. Finally, we convert these poses into a trajectory in joint space using both the DIK controller and the IK approaches as before. However, this method had numerous issues. Notably, we had to manually constrain many of the degrees of freedom. In particular, defining the length of the launchpad, the total time of the trajectory, and the number of waypoints was necessary before we could compute the trajectory. In practice, this is suboptimal, as depending on V_d and joint constraints, certain times or launchpads were not possible. For example, we found that for $V_d > 20$, only launchpad distances of 0.05 m or smaller were feasible, and in such situations the robot's jerk was too large for the ball's trajectory to be predictable, often causing the ball to slip out of the grasp.

Given the constraints that had to be manually defined for the previous approach, we decided to use kinematic trajectory optimization, modeling the trajectory as a B-spline with 10 control points. The choice of this method was both to minimize the manual optimization required from above (i.e. choosing appropriate time and acceleration) and to make an optimization method that was more generally applicable and minimally constrained. In order to do this, we began by constraining the initial pose of the gripper to where the gripper would be after picking up the ball. Then, we constrain the final pose to be at p_d with an orientation perpendicular to V_d. The latter constraint is done by aligning the $-z$-asix of the gripper with V_d. Finally, to implement the velocity constraint, we constrain a "pre-throw" pose. In order to define this constraint, we needed to fix the duration of the trajectory, T, and then we defined the pre-throw pose to be $T/100$ seconds before the throw pose. The pre-throw pose is constrained so that the average velocity between that and the throw pose is equal to V_d, necessitating that the position is constrained at $p_d - T/100 * V_d$, and that the rotation be the same. We can

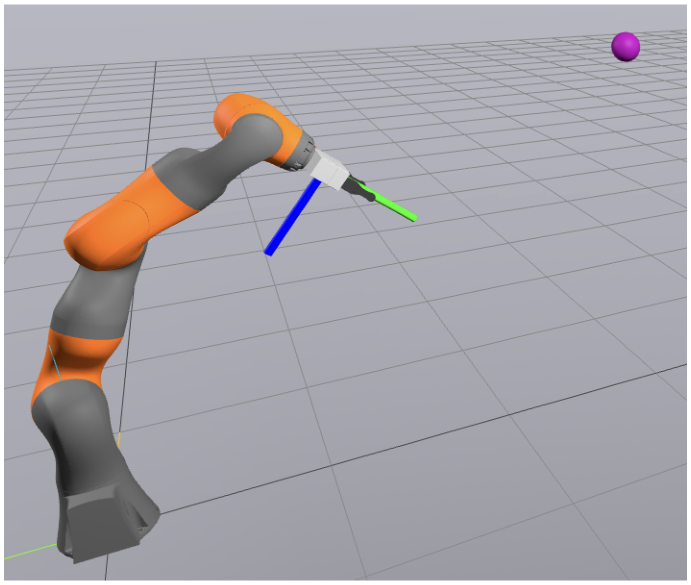

Fig. 3. Visual demonstration of the throwing mechanics of the iiwa "pitching" arm throwing the ball. The gripper releases its grasp once the arm has reached its terminal pose. Also an example of the launchpad method, where the blue line is the "launchpad."

write these constraints (in the world frame) in spatial algebra notation as follows:

$$X^{Gripper}_{initial} = X^{Gripper}_{PickDone}$$

$$p^{Gripper}_{Throw} = p_d$$

$$R^{Gripper}_{Throw}(\theta) = \arccos\left(\frac{-\hat{k} * \vec{V_d}}{|V_d|}\right)$$

$$R^{Gripper}_{Throw}(axis) = \left(\frac{-\hat{k} \times \vec{V_d}}{|V_d|}\right)$$

$$p^{Gripper}_{PreThrow} = p_d - T/100 * V_d$$

$$R^{Gripper}_{PreThrow} = R^{Gripper}_{Throw}$$

Finally, we bound the maximum and minimum velocities and positions of the joints along the entire trajectory with the iiwa hardware limits. The first objective we chose to minimize was simply the path length cost. Such optimization led to convergence less than 1% of the time, due to the highly non-convex nature of the joint space, as well as discontinuities due to hardware constraints.

To improve on this, we decided to combine a portion of the previous approach. Using IK, we can determine the joint positions at both the initial pose, $q_{initial}$, and the throw pose, q_{final}. Ultimately we desired a smooth throw, rather than sharp changes in the joint angles, so we set the initial guess of our optimization problem to be a linear interpolation between $q_{initial}$ and q_{final}. Furthermore, we penalize the distance between the first control point and $q_{initial}$ and the distance between the second control point and q_{final}. Thus, our objective is

$$min_{c_1,...c_{10}} |c_1 - q_{initial}|^2_2 + |c_{10} - q_{final}|^2_2$$

Note that the initial guess is a global minimum, however does not satisfy the constraints, particularly the pre-throw pose. As a result, the SNOPT algorithm will not be stalled at the initial guess. This approach was a massive improvement over the previous one, however, it suffered from significant errors in the projectile motion of the ball.

Our final version involved a rearrangement of the kinematic trajectory optimization problem. Rather than constraining poses only, we added a velocity constraint, constraining the gripper to move at speed V_d at the end of the trajectory. This replaced the pre-throw position constraint, with the one depicted below:

$$v^W_{Gripper} = v_d$$

To improve our results further, we added multiple velocity constraints at times $\frac{95}{100}T$, $\frac{96}{100}T$, $\frac{97}{100}T$, $\frac{98}{100}T$, $\frac{99}{100}T$. This method gave us the least error in the landing position of the ball and was also the easiest to optimize.

C. Batting Dynamics

For the batting robot, we use the same optimization methods as for pitching. Namely, to pick up the bat, we assume a known position, and use the key frame approach with IK. To swing the bat, we define a desired swing position and velocity. Notably, there is no difference between this problem and the problem of throwing, since we assume that the bat, just like the ball, is rigidly fixed to the gripper once picked up. In fact, we can simply write the constraints in terms of the pose of the center of the bat itself and solve in the same fashion.

The batter takes in a pitch position and velocity and computes the trajectory of the path. It then picks the position where the trajectory intersects its "strike zone" (as defined by the rules of baseball) and computes a desired velocity in a direction opposite of the ball's velocity and twice the magnitude. It then uses the above to compute a trajectory. A visualization can be seen in Fig. 4.

D. Trajectory Optimization

Although formulating the trajectory optimizations is relatively simple, the actual mechanics of obtaining a convergent solution are far more complex. The forward kinematics function of the iiwa is complex and many of the constraints defined above, particularly the velocities, add another level of complexity. This, in addition, to the high number of decision variables and hardware constraints makes the landscape of this kinematic trajectory optimization extraordinarily nonconvex and difficult to solve. However, the relative simplicity and freedom of the optimization, compared to defining a trajectory with acceleration and time, makes this method incredibly powerful. We begin our system by optimizing for a pitch position and velocity that is within reach of the pitcher robot, choosing to minimize the velocity to limit the strain on the joints. All of our optimizations are solved with SNOPT [5].

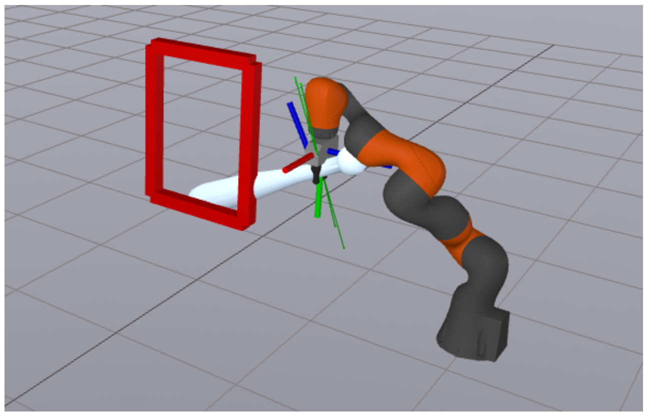

Fig. 4. Visual representation of the iiwa ”batting” arm swinging a baseball bat. The highlighted red rectangle represents the viable pitch strike zone and has no physical properties (for visual purposes only).

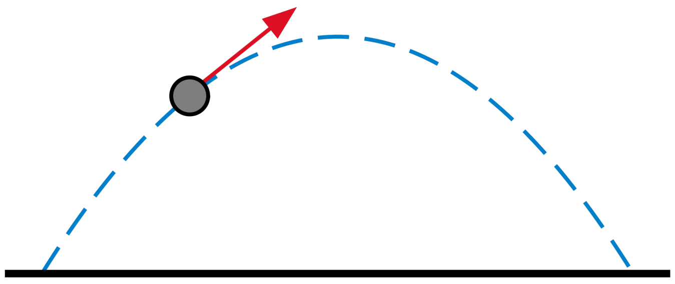

Fig. 5. Diagram of the trajectory path of the ball leaving the iiwa ”pitching” arm at the terminal pose.

Our first attempt at solving this trajectory optimization was to simply start with random initial guesses and repeat until we obtained a trajectory that converged. Ultimately, this led to incredibly poor performance, because of the complexity of the trajectory space, rarely converging before 100 iterations. This is not feasible in reality.

We attempted to simplify this by locking a number of the joints, trying combinations of 2-5 joints. However, this often led to unforeseen restrictions in the joint space that did not allow us to reach the desired positions. This method is also not particularly generalizable to other problems or robot models.

Our next approach was to use intelligent initial guesses. We began by setting the first and last control points to $q_{initial}$ and q_{final} and then turned to an initial trajectory that was a linear interpolation between those two. Although this worked somewhat well, the velocity constraints were not able to be satisfied most of the time, even after adding random noise.

The final approach combined both of the previous ones along with tolerance and iterative convergence on an optimal solution. We begin by allowing a large tolerance on the positions (0.5 m), orientations (2.5 deg), and velocities (0.05 m/s). Note that the latter is restricted more since small variations in velocity can lead to large variations in position that may be outside of joint constraints. We then set an initial guess using the linear interpolation and noise described above.

We then half the tolerances and resolve the problem with the previous solution as an initial guess. We repeat this until the tolerance is 1 mm, 5e-4 degrees, and 0.1 mm/s. In practice, this successfully found a solution almost every time, since it allows for slow convergence to the optimal trajectory.

Pitch Distance	1 m	2 m	4 m	8 m
Position Constraints Only	0.12m	0.20m	0.33 m	0.76m
Velocity Constraints	0.074m	0.16m	0.25 m	0.57m
% Strikes	100	100	90	60

TABLE I
AVERAGE ERROR FOR 10 TOSSES (5 M/S USING TWO DIFFERENT METHODS OF KINEMATIC TRAJECTORY OPTIMIZATION

Bat Swing Velocity	0.5 m/s	1 m/s	2 m/s	4 m/s
Position Constraints Only	0.031m	0.077m	0.11 m	0.19m
Velocity Constraints	0.001m	0.0042m	0.013 m	0.022m
% Hits (1 m Pitch)	30	20	40	40

TABLE II
AVERAGE ERROR FOR 10 BAT SWINGS USING TWO DIFFERENT METHODS OF KINEMATIC TRAJECTORY OPTIMIZATION

IV. RESULTS AND DISCUSSION

A. *Throwing/Batting Results*

Results from testing our system are shown in Tables 1 and 2. We compare the two different versions of kinematic trajectory optimization outlined in the methods, namely one only using position constraints and the other with velocity constraints. An example of a bat swing is shown in Fig. 6.

We evaluated the average error between the location of the desired throw in the strike zone and the location of the actual throw in the strike zone for pitches from distances of 1m, 2m, 4m, and 8m. Testing the average error allows us to quantify how close the commanded behavior is to the desired robot behavior. The results show that the pitches are pretty accurate.

Pitch Distance	1 m	2 m	4 m
Velocity Constraints Normal Friction	0.074m	0.16m	0.25 m
Velocity Constraints High Friction	0.022m	0.045m	0.10 m

TABLE III
AVERAGE ERROR FOR 10 PITCHES WITH DIFFERENT FRICTION VALUES

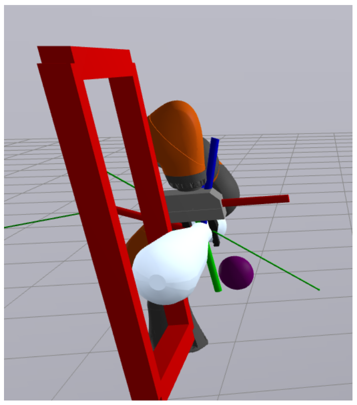

Fig. 6. Example of a bat swing

We model the strike zone with the same dimensions as in official baseball, and the results indicate 60% strike rate at 8m. For reference, a regular pitch in baseball is thrown from roughly 19m. Thus, although our system can't compete with humans yet, it is substantially accurate.

Similar results are shown for batting in Table 2. Rather than varying the bat position, as the batting position will be roughly the same for each swing, we vary the velocity. Increasing velocity results in increasing acceleration and jerk which is what causes higher error. Yet, we note that the variation in the bat position is significantly smaller than the pitch position. The difference most likely comes from a small error in orientation. For the task of batting, such orientation errors are unimportant since the bat is spherical, but for pitching, using the same motion planning framework results in significantly higher errors. Such error can likely be partially mitigated through continued iterations of low-tolerance motion planning.

Another key source of error comes from friction. We observed that near the end of a pitch, i.e. during moments of high acceleration, the ball slipped from the gripper, violating our assumption that the two were rigidly fixed during the throw. This was a significant source of error, as shown in Table 3. We increased the coefficients of friction on the ball and the gripper by a factor of 10, drastically decreasing the error.

A third source of error comes from gripper release dynamics. We constrained the gripper to open just before the throw at time $\frac{99}{100}T$, where T was the length of the trajectory. Due to large contact forces between the ball and the gripper, the effect of friction on throws was unpredictable.

A key implicit result in our work is the use of the same motion planning framework for both tasks. It is not readily obvious that batting and throwing a ball would be similar tasks for the robot to solve, however, our results showed that we could optimize a batting trajectory in the same way as the pitching. Thus our optimization framework and iterated convergence show promise to transfer to other manipulation tasks as well.

B. Use Cases

Our project has several useful applications, one of which is a robotic system that helps baseball players practice without a partner. A player working on their batting can practice with the pitcher arm and a player working on their catching can practice with the batter arm. This system can also be applied to other bat-ball sports with minor adjustments.

A key capability of our system is accurately tossing objects, which can be applied to many use cases. For example, the pitching robot can be used for sorting objects in factories and waste processing plants. It can also be used to increase pick-and-place efficiency, such as in the agriculture setting to toss seeds to increase sowing efficiency.

C. Limitations

One key limitation in our system is the use of privileged state information not available in the real world. Our robots are aware of the locations of their respective objects. Furthermore, the batter is aware of the pitch trajectory. In the real world, such information is not available, and estimation of the pitch trajectory would have to be visual. One possible algorithm could be using an RGBD camera and generating a 3D point cloud. With one frame, we would likely be able to estimate the position of the ball's center, and with two, we would likely be able to estimate its velocity. This would also likely improve the hit rate of bat swings in simulation. Currently, error in the pitch trajectory causes missed swings, but real-time pose estimation would likely overcome that issue.

Another key limitation is the physics modeling. As mentioned above, we increased the friction coefficients to improve consistency, but in real life, this would have to be done by increasing the strength of the grip or changing the material of the gripper. However, the physics of that are still significantly complex, and not entirely predictable in our current framework.

D. Future Work

There are many avenues of future work to extend our current framework. One such path involves the implementation of the visual pose estimation mentioned above. One can even imagine a system with 2 RGB-D cameras for increased accuracy. Another avenue involves exploring the dynamics of friction. Although we were primarily focused on trajectory optimization, modeling the gripper friction and release dynamics is a complex problem in itself, and advancements there would drastically improve our accuracy. A third future avenue we see is testing this trajectory optimization framework for other nonprehensile manipulation tasks, such as pushing a chair or rolling a ball. Although in this situation the framework transferred readily between tasks, it is not obvious that it will be as effective for other nonprehensile manipulation scenarios.

V. Conclusion

BatterBot is a robotic system that has the capability of pitching and hitting baseballs in simulation. We implement both of these capabilities through the use of kinematic trajectory optimization. We use an optimization framework that makes smart initial guesses and iteratively converges on a solution to successfully navigate a highly unconstrained non-convex joint space. We evaluate the error of our system by pitching and swinging from different distances and velocities. Compared to other motion planning methods, our framework demonstrates high guarantees of finding a solution, albeit at the cost of increased computation time. Our framework readily transfers between the tasks both for picking up objects and moving at desired velocities and positions. However, our system makes use of privileged state information to preplan its motion, making it difficult to transfer to the real world currently. However, future work would involve adding visual pitch estimation and potentially more accurate models of gripper dynamics to decrease the sim2real gap.

VI. Team Acknowledgements

Thank you to Professor Tedrake, Elena Kalodner-Martin, and the 6.4210 Teaching Staff for working with us this

semester. For this project, Yajvan worked on kinematic trajectory optimization, Felix worked on kinematic trajectory optimization and grasping mechanics, and Stephen worked on IK trajectory optimization and simulation.

References

[1] T. Senoo and I. Ishii, "Baseball robots based on sensory-motor integration," in *2021 21st International Conference on Control, Automation and Systems (ICCAS)*, 2021, pp. 1772–1777.

[2] T. Hsiao and S. Wu, "Decision making based on physical and neural network models for precision ball-batting robots," in *2021 American Control Conference, ACC 2021*, ser. Proceedings of the American Control Conference. United States: Institute of Electrical and Electronics Engineers Inc., May 2021, pp. 3787–3792, publisher Copyright: © 2021 American Automatic Control Council.; 2021 American Control Conference, ACC 2021 ; Conference date: 25-05-2021 Through 28-05-2021.

[3] A. Gardner and K. Rahnamai, "Robot trajectory target delivery using machine learning," 08 2020, pp. 93–96.

[4] R. Tedrake and the Drake Development Team, "Drake: Model-based design and verification for robotics," 2019. [Online]. Available: https://drake.mit.edu

[5] P. E. Gill, W. Murray, and M. A. Saunders, "Snopt: An sqp algorithm for large-scale constrained optimization." *SIAM J. Optimization*, vol. 12, no. 4, pp. 979–1006, 2002. [Online]. Available: http://dblp.uni-trier.de/db/journals/siamjo/siamjo12.html#GillMS02