Physics Based Throwing Using Inverse Dynamics Control

Arif Kerem Dayi
Harvard University, Cambridge, United States
arifdayi@mit.edu
keremdayi@college.harvard.edu

Hanqi Su
MIT, Cambridge, United States
hanqisu@mit.edu

Quincy Johnson
MIT, Cambridge, United States
qjohnson@mit.edu

***Abstract*—We construct a method for robots to successfully throw objects between one another. Throwing has the ability to expand a robot's reach and operation space, having many useful applications for solving tasks that would otherwise be complex or impossible to execute without throwing. Many robust methods exist for manipulation throwing problems which leverage techniques such as reinforcement learning. We deviate from methods for throwing that have been proven to be effective in an attempt to explore the feasibility of a physics based method not widely researched. To this end, we designed a robotic manipulator capable of throwing using inverse dynamics control. The system detailed in this paper features two robots that can either be an *idle bot* or a *throwing bot* at any time. We designate *idle bots* to be robots that detect and grasp objects that enter its operation space, and *throwing bots* to be robots that throw grasped objects to a target position. We achieve this through the use of inverse dynamics control, geometric-based antipodal grasping, trajectory planning, and high-level task planning. The result is a closed system of robots that can accurately throw to target positions using purely inverse dynamic control. We show that this is feasible in simple environments, and also investigate the factors that contribute to inaccuracies and the motivation for learning based methods. Demo Link: https://www.youtube.com/watch?v=HvuFJTrRoTU**

***Index Terms*—Robotic Manipulation, Perception, Geometric based grasping, Trajectory Planning, Inverse Dynamics Control**

I. Introduction

Throwing objects is a very important skill to endow robots with, as it is a prerequisite for more complex and dynamic robotic manipulation tasks such as playing badminton, basketball, and juggling. Even in simple daily tasks, humans use throwing skills to move objects around more easily as we throw objects into trash bins, pass an object to someone, or even while sorting bins. Hence, studying throwing in robots can allow us to make more dexterous manipulation systems that behave more similarly to humans. However, as useful as it is, throwing poses important technical challenges such as being a very dynamic task that requires precise control, uncertainty in object variety and grasp reliability, and the requirement of many components of a robotic manipulation system to work smoothly.

As such, throwing has widely been studied in literature. Prior work has explored using analytical, physics based approaches [1]–[4] that exploit the known physics of the throwing task, and learning-based approaches [5], [6]. The advantage of physics based approaches is that they do not require extensive training, but learning-based methods can be more generalizable due to variety in environment conditions such as different objects and different grasps.

We propose a physics based system for throwing to investigate the merits and limitations of physics based controllers. Making such a system work smoothly on a difficult dynamic task like throwing requires implementing many components such as perception, grasp selection, trajectory planning/optimization, and the low level controller. Considering the dynamic nature of the task, we implement an inverse dynamics controller as opposed to position controllers or differential inverse kinematics controllers. This allows the manipulator to track fast and accelerating trajectories with enough precision to succeed on throwing tasks.

Ultimately, we verify that a physics-based controller can be implemented for throwing known objects in simulation. We investigate the performance of the inverse dynamics controller on high-acceleration trajectories and investigate the relation between throwing accuracy and controller performance. We also investigate the limitations of the controller arising from unreliable grasps, disjoint trajectory planning/control or controller error caused by difficult trajectories. This allows us to show which parts of the problem need more investigation and what methods can be used, and provides further motivation for using learning based methods to generalize to more objects or achieve more reliability with different grasps.

II. Related Work

The throwing problem is a kind of dynamic operation which require the specific speed in the specified configurations while adhering to geometric and dynamic constraints. Previous work has looked at dynamic manipulation, which includes throwing [2], and they propose planning based on creating trajectories that end at a particular position with a desired velocity. This velocity is calculated based on the target position for the throw. Other works looked at simplifying the throwing problem into a planar problem with mobile manipulators [1]. The authors propose the concept of the Backward Reachable Tube (BRT), which means that given the flight dynamics and landing speed constraints of a given object and the landing position of the target, there will be a set of valid throwing configurations (throwing position and throwing velocity and flying time) which is defined as BRT. However, using mobile manipulators can make the problem easier since unconstrained

planar motion can be achieved. Our work will be similar to these, but we will be implementing throwing with a 7DOF iiwa as opposed to mobile manipulators and propose a different control strategy. While the methods above use analytical solutions, some methods use sampling based methods [3]. This can improve planning in joint-space, and we will refer to this in our discussion. Some works even looked at optimizing the throwing trajectory with respect to their goals by introducing a general robot trajectory planning method that takes into account the relevant constraints of the robot and minimizes the time for the robot to cycle through a single task [4], which we consider in our throwing trajectory planning. In this work, the authors aim to find the shortest trajectory in terms of time.

In contrast to these physics based approaches, learning based approaches promise to deliver more generalizable methods that can account for the variability in grasps and objects [5]. These methods, however, rely on being able to train the robot for long time and also cannot provide rigorous guarantees about the behavior of the robot. Similarly, other works explored throwing objects to a moving basket [6], also using reinforcement learning methods. While these works are more generalizeable, they require extensive training and it is difficult to reason about cases that the learning algorithm did not encounter.

In our project, we generate a trajectory based on the physics of the throwing problem, and we need to follow a generated trajectory for throwing precisely. This requires a controller that is good for dynamic tasks. In previous works, inverse dynamics controller [7] and model predictive control [8], [9] has been used to achieve precise control of dynamical systems. These controllers make use of the known or estimated dynamics of the system, which make them more powerful in applications where this information can be exploited, and can be more powerful compared to a naive position controller. Since we have access to a accurate multi-body representation of our robot in our simulation, we use the inverse dynamics controller. Even in the simulated environment, we show that certain such as different grasps and variability in trajectory tracking can require reinforcement learning methods, which motivates some of the work described above.

III. Project Setup

More precisely, we can define our project goal as follows. There are two iiwa robots, approximately 2 meters away from each other and two bins in front of them. We want the robots to be able to pass objects from their own bin to the other bin, but since the bins are too far apart, the robots must throw the object. There are perception systems (depth cameras) that allow the robots to perceive the object in their own bin. The robots have to plan their grasp and their throw along with any intermideate step that is involved. Our simulation setup is shown in Figure 1.

IV. System Design

Implementing a full stack throwing system requires selecting good grasps, planning trajectories for throwing, planning

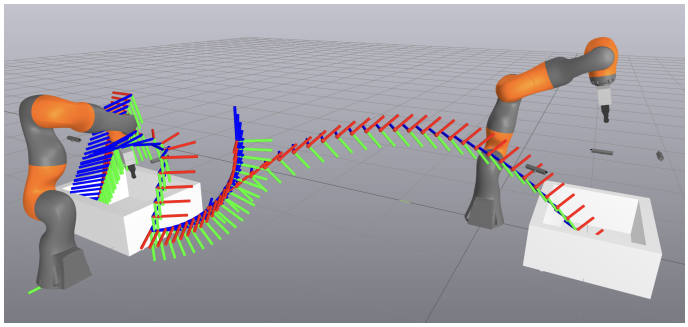

Fig. 1. Project Setup. Each robot uses their perception system to generate candidate grasps, and throws the object into the bin of the other robot. They repeat this process when they can perceive a brick in their bin. The planned trajectory of the robot along with the trajectory after release of the object is shown above.

intermediate trajectories for transitioning from grasping to throwing, and controlling the robot to meet the dynamic constraints of the tasks. This motivates us to build components for (i) perception & grasp selection, (ii) throwing trajectory generation in 3D, and (iii) inverse kinematics to convert 3D poses into joint angles & inverse dynamics to achieve precise trajectory control. Hence, we have a high level planner system, helper methods for sub tasks, and inverse dynamics controllers systems. Our system design is illustrated in Figure 2.

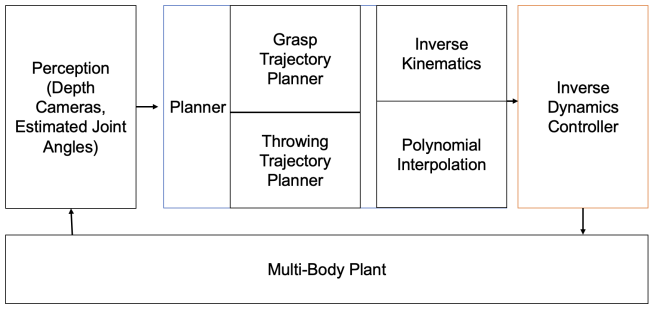

Fig. 2. The planner consists of a grasp trajectory planner and a throwing trajectory planner. These determine the 3D trajectories for their respective tasks. Then, the low level systems in the planner convert these into polynomial interpolated trajectories of joint angles, which is passed to the inverse dynamics controller

Now, we describe how the systems interact more precisely.

1) The perception system uses geometric perception to determine the pose of the object, and decide on a good grasp position. This is passed onto the planner.
2) The planner designs a trajectory to grasp to object, and end in a safe position to not hit any bins or cameras. This zone facilitates transition between different states of the robot (ie. throwing and grasping).
3) An optimization based inverse kinematics system decides on the joint angles for the trajectory, and interpolates between joint positions to generate a continuous (and twice differentiable, for accelerations) trajectory of joint angles. Then, desired velocity and acceleration is extracted from this system.

4) The throwing module generates a trajectory and a release time to achieve accurate throwing. The previous step is repeated and a twice differentiable trajectory is geneerated.
5) The generated joint angle trajectories involving generalized positions, velocities, and accelerations are fed into the inverse dynamics controller.
6) After the iiwa completes the throw, it goes back to a safe position, and checks for any objects in the bin. If there is an object, we start from step 1.

V. Methodology & Implementation

A. Perception & Grasp Selection

The system consists of bins which contain blocks to be grasped by a robot arm for throwing. In order for this to occur, the robot must be able to effectively detect objects in the bin. We achieve this by including three cameras, one for each side of the bin that the robot is not on. For each camera, we take the normals of the point cloud. We then flip the normals to point towards the respective camera which will be important for the grasping candidate step. We then concatenate and downsample all of the cameras' point clouds to form a good point cloud representation of an object in the bin. Next we generate potential grasp candidates. Since the robot arm is expected to be able to grasp objects that were thrown from another arm, object can be of unknown predetermined pose. Therefore, we implement a dynamic grasp candidate selector. Grasp candidates are generated by taking a virtual end point effector (the 2-pronged gripper), picking a random pixel in the point cloud, and aligning the fingers with the normal of the pixel. This is repeated by uniformly sampling many rotations about the normal to generate a grasp candidate. Each candidate represents a potential gripper pose. Each candidate is then scored by creating a bounding box between the gripper prongs and evaluating intersections with the point cloud. The final candidate score is a function of the total normals in the gripper box. Candidates who's prongs intercept the point cloud or make contact with another object in the scene are eliminated. The candidate with the highest score is then returned as the ideal grasp candidate to the high level planner, which then constructs a polynomial trajectory to execute the grasp and return to its start position.

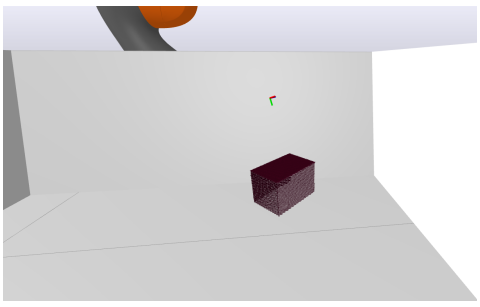

Fig. 3. Constructed point cloud of a brick inside the bin operation space of a robot. The ideal grasp pose has been calculated and can be seen above the point cloud.

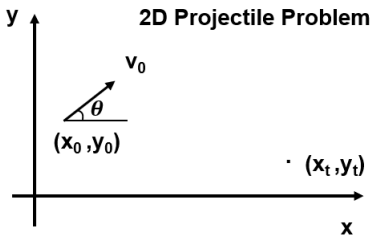

Fig. 4. The 2D Projectile Problem. To generate a trajectory for throwing, we need to determine an release position, angle, and velocity.

B. Projectile Problem

2D Projectile Problem:

Considering the motion of a projectile in a two-dimensional plane, we assume that x is the horizontal and horizontal axis and y is the vertical and vertical axis (along which gravity (g) acts downward). Figure 5 is a schematic diagram for 2D projectile problem.

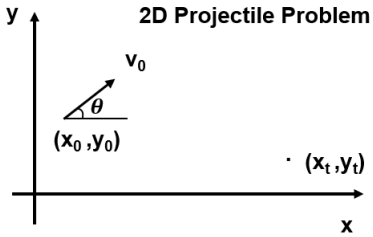

Fig. 5. The 2D Projectile Problem. To generate a trajectory for throwing, we need to determine an release position, angle, and velocity.

For the projectile motion problem, the initial position given is (x_0, y_0), initial velocity v_0, and launched angle θ relative to the horizontal axis, the equation of motion can be written as follows:

$$x(t) = x_0 + tv_0 \cos\theta \tag{1}$$

$$y(t) = y_0 + tv_0 \sin\theta - \frac{1}{2}gt^2 \tag{2}$$

When the object reaches the target position (x_t, y_t) after time T, the distance changed in the x and y directions can be calculated. And the flying time and initial velocity can be calculated from the distance difference.

$$\Delta x = Tv_0 \cos\theta \tag{3}$$

$$\Delta y = Tv_0 \sin\theta - \frac{1}{2}gT^2 \tag{4}$$

With the above formula, we can directly compute the time T and the initial speed v_0:

$$T = \frac{\sqrt{2(\Delta x \tan\theta - \Delta y)}}{g} \tag{5}$$

$$v_0 = \frac{\Delta x}{T \cos\theta} \tag{6}$$

3D Projectile Problem:

Returning to our throwing problem, although our robot is throwing in a 3D world, we can reduce this throwing behavior to a two-dimensional problem. As you can see in Figure 6, when the robot is heading towards the target point, we can set the throwing trajectory in the direction of the target point. At this time, the z axis in the three-dimensional space can be seen as the y axis in the two-dimensional problem, and the robot's orientation is jointly determined by (x, y). The generated angle of the robot orientation is defined as the **orientation angle**, and the robot arm orientation in the three-dimensional problem can be seen as the x axis of direction in the two-dimensional problem.

Consider initial position p_0 = (x_0, y_0, z_0) and target position p_target = (x_t, y_t, z_t) (both in the world frame). Then we can get:

$$\Delta x = \sqrt{(x_t - x_0)^2 + (y_t - y_0)^2} \tag{7}$$

$$\Delta y = z_t - z_0 \tag{8}$$

Orientation angle:

$$\phi = \arctan\left(\frac{y_t - y_0}{x_t - x_0}\right) \tag{9}$$

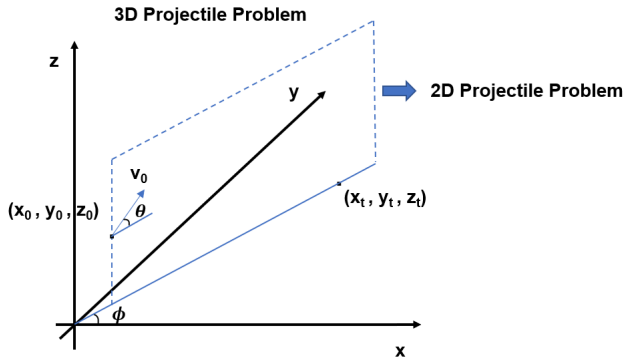

Fig. 6. The 3D projectile problem can be converted to a 2D projectile problem.

C. Planning

The essence of trajectory planning for the iiwa robot arm is to intrinsically calculate the changing process of each joint angle under continuous time series so that the robot can be configured to move from the initial joint angle to the target joint angle, so as to complete specific tasks. The configuration of different joint angles makes the three-dimensional position of the end of the robot arm different. Therefore, the core of this short section is how to construct the pose of the path of the three-dimensional world (including position and orientation).

After the iiwa robot arm successfully grasps the object and moves to the predefined safe area, we will execute the throwing trajectory planning. The complete throwing trajectory consists of the following three parts:

(1) Take the distance between the center of the end gripper of the current iiwa robot arm and the base of the iiwa robot arm as the radius, and rotate the robot arm counterclockwise to make the gripper position move from the safety point to just above the line between the base of the iiwa robot arm and the target position and make the X-axis of the gripper parallel to the direction. The purpose of this is to transform a complex 3D throw problem into a simple 2D throw problem using the concepts we mentioned in Part A, which allows for fewer constraints and a clearer calculation of the trajectory of the object thrown. In fact, when we know the position of the safe point of the robot arm and calculate the corresponding radius and target orientation angle, we can easily calculate the Pose(position and orientation) of each frame by using the $rotate\_to\_orientation\_direction$ method we implemented.

(2) After the first step, move linearly from the current position to the starting position of the throwing trajectory, and keep the orientation consistent. The purpose of this step is to return the robot arm to the starting point of a more appropriate throwing trajectory so that the robot arm can have more activity space, which is conducive to the calculation and optimization of the best throwing trajectory in the last step. It's important to keep the orientation consistent, because only when the orientation of the gripper is the same as that of the iiwa arm base pointing towards the target position, can the trajectory planning of the last step be carried out smoothly.

(3) Third, generate the last segment of the throwing trajectory using our self-designed throwing trajectory method. Since we have the target position and starting position of the throwing trajectory, then if we want to achieve a good throwing task, we need to solve the initial throwing position, the initial launched velocity, and the initial launched angle of the iiwa robot.

Minimum_Energy_Consumption_Trajectory, we still use the circular arc trajectory design idea. Considering Figure 7, since we know the starting position of the throwing trajectory, and we want to use the circular arc trajectory to realize our throwing trajectory planning, in this case, the core of our solution is not to find the throwing position and velocity, on the contrary, We should focus on the radius of the arc and the launched angle. Because these two values combine the starting position of the throwing trajectory, we can directly calculate the throwing position. In addition, we had a new idea. We believe that the robot arm should consume as little energy as possible when moving along the predetermined trajectory, so our cost function is:

$$min \quad E = E_k + E_p = \frac{1}{2} m v_0^2 + mg(y_0 - y_{init}) \tag{10}$$

In addition, we need to limit the radius r and the launched angle θ, because too large a radius will cause the estimated

trajectory to exceed the working space range of the iiwa robot arm, and too small a radius will cause the manipulator to be unable to move sufficiently, resulting in the collapse of the subsequent inverse kinematics solution. In addition, when the launched angle is around $45°$, it can fly the farthest. Therefore, if the launched angle is too large or too small, the launched velocity will become larger, which requires the robot arm to provide more kinetic energy, which is undoubtedly a waste of energy. So we limit the launched angle to between $30°$ and $60°$. Finally, considering that the target position is the center of the box, we need to worry that the height of the boundary may prevent objects in flight from entering the box sequentially, so we impose additional constraints on the boundary, the formula is as follows:

$$T_{fly}^{con} v_0 sin\theta - \frac{1}{2} g (T_{fly}^{con})^2 \geq (y_{tar}^{con} - y_0) \tag{11}$$

where :

$$x_0 = x_{init} + r sin\theta \tag{12}$$
$$y_0 = y_{init} + r - r cos\theta \tag{13}$$
$$\Delta x = x_{tar} - x_0 \tag{14}$$
$$\Delta y = y_{tar} - y_0 \tag{15}$$
$$T_{fly} = \frac{\sqrt{2(\Delta x \tan\theta - \Delta y)}}{g} \tag{16}$$
$$v_0 = \frac{\Delta x}{T_{fly} \cos\theta} \tag{17}$$
$$x_{tar}^{con} = x_{tar} - d \tag{18}$$
$$y_{tar}^{con} = y_{tar} + h \tag{19}$$
$$T_{fly}^{con} = \frac{(x_{tar}^{con} - x_0)}{v_0} \tag{20}$$
$$0.1 \leq r \leq 0.4 \tag{21}$$
$$30° \leq \theta \leq 60° \tag{22}$$

And once we set these constraints and cost function, we can solve for the radius r and the launched angle θ.

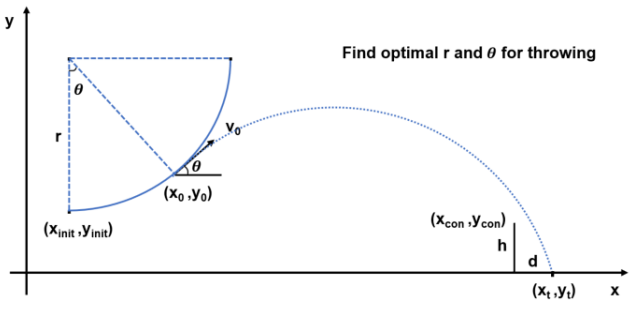

Fig. 7. This figure shows the core problem of the throwing trajectory is to find optimal r and θ by using method(Minimum_Energy_Consumption_Trajectory).

D. Control

Broadly, the control task requires converting the output of the planner which contains of poses in the real world to real-time force. The problems of throwing and catching pose a unique problem from a control aspect: As opposed to standard pick and place or basic manipulation tasks that do not require precise dynamic motion, throwing and catching requires the robot to track throwing and catching trajectories with small tracking error. That is, at a given time t, trajectory $q_d(t)$, and current estimated robot state $\hat{q}(t)$, we want $\|q_d(t) - q(t)\|^2$ to be small. In fact, for the tasks we are concerned about, we want to achieve a desired *pose* and *velocity*. Therefore, we implement (i) solving for joint angles given poses, (ii) converting a list of joint angles to a continuous (and differentiable) trajectory, and (iii) implementing an inverse dynamics controller for trajectory tracking.

First, we use optimization based inverse kinematics to convert real world 3d poses to joint angles. We define a nominal joint angle q_{nom} deviation limits for position d and angle θ. We also have forward kinematics function $f(q)$ that maps joint angles to poses and desired position p_d and orientation R_d. Then, we solve the following optimization program:

$$\min_q \|q - q_{nom}\|^2 \text{ subject to} \tag{23}$$
$$p_d - d \leq f_p(q) \leq p_d + d \tag{24}$$
$$\text{angle}(f_R(q), R_d) \leq \theta \tag{25}$$

This gives us a list of joint angles from poses. Then, we interpolate a continuous and twice differentiable polynomial between these joint angles using **CubicWithContinuousSecondDerivative** under the **PiecewisePolynomial** class in Drake. Twice differentiability is necessary for achieving effective inverse dynamics control.

As mentioned earlier, because of these precise dynamics requirements, we use an *inverse dynamics* controller. Briefly, an inverse dynamics controller makes use of the known robot dynamics to achieve a desired position, velocity, and a feed forward acceleration. We work in joint space for the controller, and we input joint positions, velocities, and feed forward accelerations, and output joint forces. The inputs are estimated and desired positions $\hat{q}$ and q_d, estimated and desired velocity $\hat{v}$ and v_d, and a feedforward acceleration $\alpha_{command}$. Therefore, the **InverseDynamicsController** in Drake works as follows [10]:

$$\text{vd\_command} := k_p(q_d - \hat{q}) + k_d(v_d - \hat{v}) + \tag{26}$$
$$k_i \int (q_d - \hat{q})\, dt + \alpha_{command} \tag{27}$$
$$\text{force} := \text{inverse\_dynamics}(\hat{q}, \hat{v}, vd\_command) \tag{28}$$

Then, force is used as actuation input. The inverse dynamics is obtained from the multi-body plant, which uses a model of the iiwa. As it can be seen above, the inverse dynamics controller is essentially a PID controller stacked up with a dynamics engine. We use $k_p = 100$, $k_i = 1$, and $k_d = 20$ for our project.

VI. Evaluation

A. Experiment Setup

Since throwing is a very dynamic action, to test throwing performance, one proxy we can use is the trajectory tracking error. Ultimately, what determines throwing accuracy is whether the robot has the correct velocity and orientation and close to the release point when the gripper opens. Hence, we measure the trajectory tracking error over time to determine the performance of our controller and gain more insight when high error happens and suggest ways to improve.

In our experiments, we keep track of the desired state $x_d(t)$ and the estimated state $\hat{x}(t)$, which contains the generalized positions q and velocities v, and has size 14. We also keep track of the commanded accelerations $\alpha_d(t)$ at a given time, and establish a relation between these accelerations and the tracking error. To gather these values, we use the logger **LogVectorOutput** in Drake, on the output ports corresponding to these values.

B. Testing Trajectory Tracking Performance

An important aspect that determines throwing and catching performance is trajectory tracking error. More precisely, we look at tracking error over time. Define $e(t) = \|x_d(t) - \hat{x}(t)\|^2$. In Figure 8, we plot $e(t)$ from time 0 to time 23, which is right after the robot completes throwing the object. It can be seen that the tracking error is highest at the end, when the robot is executing the throwing motion. This is expected as the robot has to track the trajectory under large accelerations and velocities. To make more sense of these figures, we also plot the commanded accelerations (α_d, desired accelerations) vs. time in Figure 9 to see the relations between the commanded acceleration and tracking error. We take $\log \alpha_d(t)$ to make the graph more readable as high acceleration values can make the rest of the graph vanish to 0. We also smooth out the graph because of high variation in acceleration over time. In these figures, we observe that the highest desired accelerations and highest tracking errors correspond to the throwing part of the trajectory. High accelerations are expected since the robot needs to achieve a particular desired release velocity in a short amount of time.

We can derive more insight from these figures. First we see that with low commanded accelerations, the tracking error is relatively low. This means that if we want to design robust trajectories that are trackable by an inverse dynamics controller, we need to limit acceleration. Similarly, since the high tracking error at the end corresponds to very high accelerations, limiting the accelerations might reduce the tracking error of the inverse dynamics controller significantly in this region. Hence, using dynamic trajectory optimization to limit the joint accelerations might be a first step to improve the performance of our controller. It is also more realistic in real world uses since high values of acceleration are not very desirable.

VII. Discussion

In this project, we built a full stack physics based manipulation system that can achieve grasping and throwing,

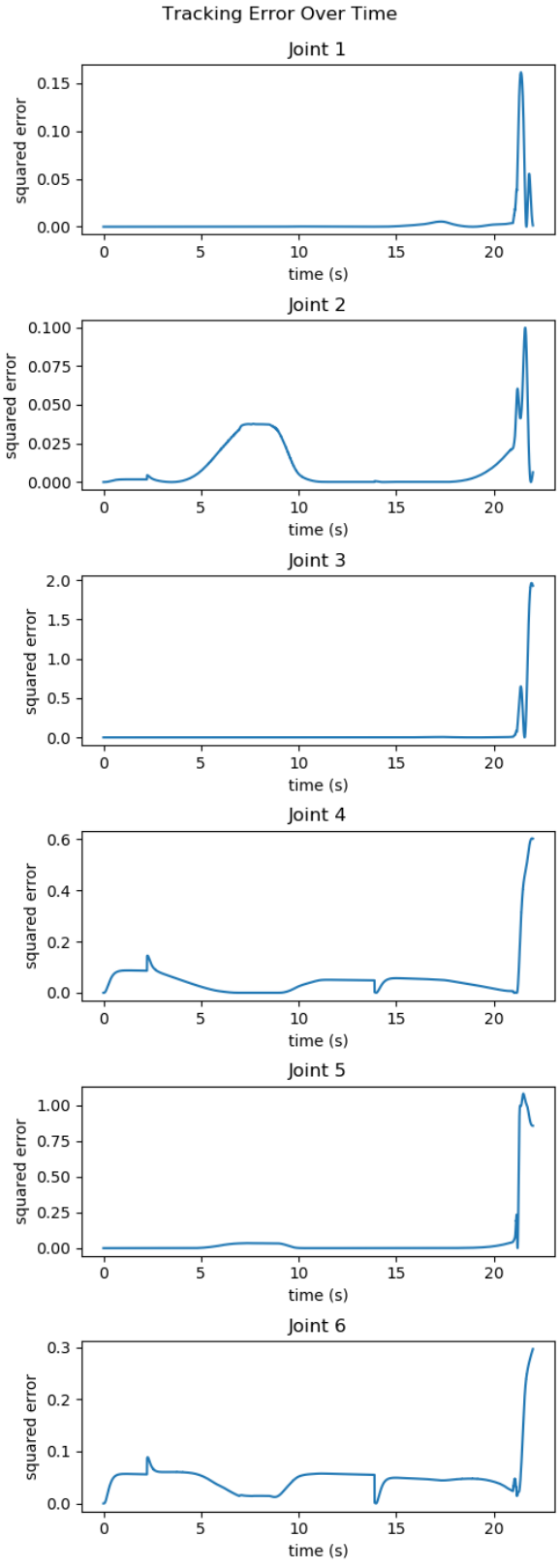

Fig. 8. Trajectory Tracking Error when using the Inverse Dynamics Controller with feedforward accelerations. It can be seen that the error is largest at the end, which is when the robot is executing the throwing trajectory

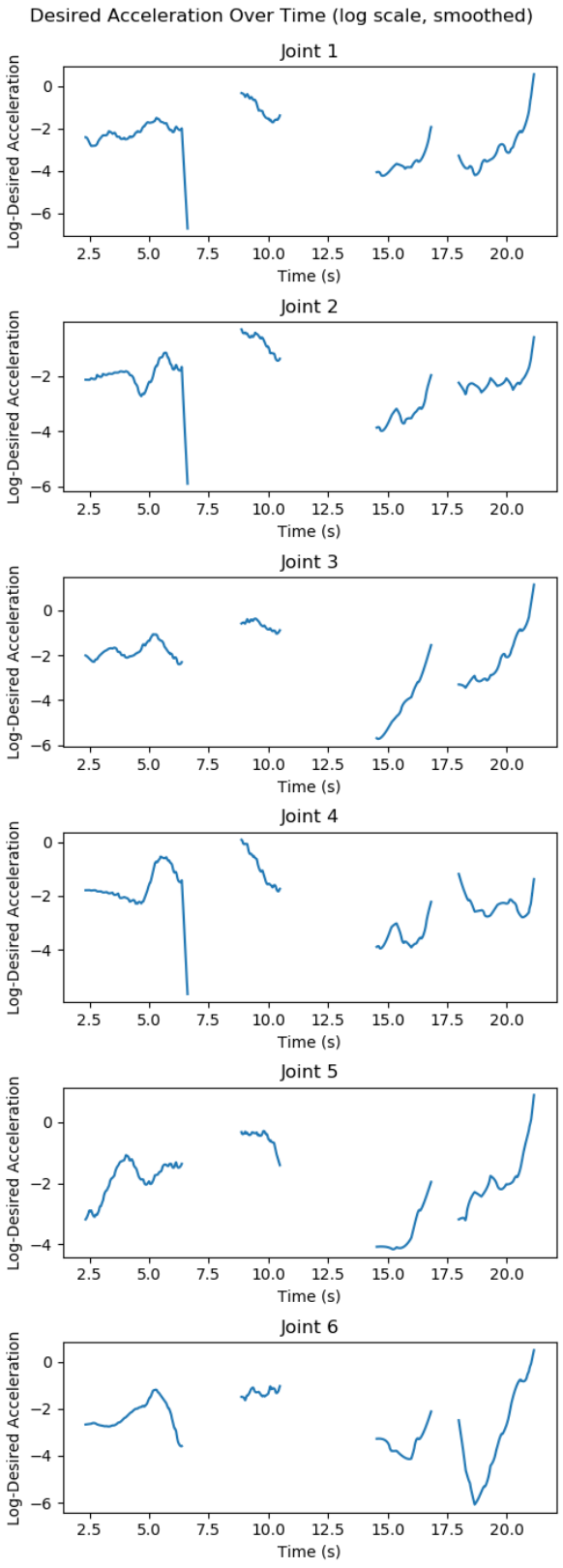

Fig. 9. Smoothed Log-scale Desired Desired Acceleration over time. Parts that do not have a value in the graph correspond to really small tracking error, which goes to $-\infty$ when we take the log. It can be seen that the desired accelerations are highest at the end, which corresponds to the throwing time. It can also be seen that these high accelerations correspond to high tracking error. Hence, lowering accelerations in trajectories might improve tracking performance.

and investigated what components are needed and how they interact. First, we highlight some of successes of our project and some of the limitations we overcame.

1) The inverse dynamics controller is relatively successful compared to other controllers for the task of throwing. The default manipulation system comes with a differential ik controller, which cannot guarantee dynamical performance. Similarly, the position controller does not provide an interface for following highly dynamic trajectories as it can only be commanded position inputs which are not guaranteed to be achieved. However, the inverse dynamics controller can be provided positions, velocities, and accelerations which makes it highly versatile. We saw that even with very fast/large acceleration trajectories, the iiwa was able to follow the trajectory relatively closely.
2) Our centralized planner design allowed us to complete different tasks that might have different natures. For instance, the trajectories generated for grasping and throwing are very different, but we can use the same controller as our planner always sets a trajectory for the iiwa to follow.
3) Our throwing trajectory planning is relatively simple, and it allows us to generate throwing trajectories for a variety of target positions. For close positions, it outputs trajectories with small accelerations.
4) We also establish a relation between trajectory accelerations and tracking error, which motives studyign further optimizing trajectories for tracking error by limiting accelerations. The inverse dynamics controller performs well when the accelerations are relatively low, but can have higher errors with high accelerations, even when used with feed forward acceleration.

However, we also highlight some of the current limitations of this approach and the intricacies of creating this full stack manipulation system. In the proceeding subsections, we talk about each of the components that require improvement and are critical to making a reliable throwing system. These can motivate further study and provide important lessons for control and planning in dynamic robotic manipulation systems.

A. Integrated Trajectory Generation & Inverse Kinematics

There is room for improvement in our inverse kinematics system by using kinematic & dynamic trajectory optimization. This can lead to trajectories that have less acceleration, and therefore are more easily trackable by the inverse dynamics controller and improve the tracking accuracy of the inverse dynamics controller.

B. Throwing Trajectory Generation with Acceleration Limits

In our project, we discovered that a major reason for failed throws is slipping objects. This happens when the accelerations in the throwing trajectory are too large and the friction between the gripper and the object is not too large. Even though we do not formally model this interaction, it required us to slow down our throws to achieve more consistent throws as fast

throws require large accelerations which cause the object to slip. This is one aspect that limits our system from throwing objects to really far locations.

One way we propose, which we believe can tackle this problem, is generating longer trajectories that achieve a high velocity in a longer time. However, this becomes very tricky with the mobility constraints of the iiwa, and requires careful trajectory optimization. Overall, our numerical results show that lower accelerations is better for trajectory tracking, which motivates studying optimization of trajectories for low accelerations.

C. Interactions Between Inverse Dynamics and Interpolated Trajectories & Inverse Kinematics

In our experiments, oftentimes, we encountered subtle interactions between inverse kinematics and inverse dynamics control. Throwing trajectories involve high velocities and accelerations, and feeding the output of a naive inverse kinematics controller to an inverse dynamics controller can lead to problems. For instance, each gripper pose can be achieved via multiple configurations of the robot, and naively choosing one of these configurations can lead to unstable trajectories when combined with a feedforward acceleration controller. This is because in such trajectories, accelerations can be arbitrary, which lead to large forces commanded to the iiwa. Hence, to improve our system and make it more reliable, trajectory optimization is needed to make the joint configurations more compatible with the inverse dynamics controller. This can also help us limit the accelerations of joints in the trajectory design.

D. Heterogeneity in Grasps

In our project, we also observed that with different grasps, we can see different performance in throwing. It is worthwhile to further investigate the effect of grasp selection on throwing trajectories. For instance, grasping an object from near the edge might cause it to swing and go further than expected. The huge variety in possible grasp configurations and the complicated interactions in motivates us to study learning based methods. Modeling all these complex interactions might not always be possible.

Overall, implementing a full stack throwing system is complex and each of the components of the system interact in a variety of ways that are hard to model. This can range from variability in grasps to robustness of the low-level controller to the high level trajectory generated by the planner. Hence, this makes learning based methods highly promising, as they seek to model these complex interactions using machine learning methods. Ideally, learning based methods can exploit the known physics of the system and the rigor of more analytical methods. Even though our system required careful design, learning systems can require long hours of training and might fail to provide rigorous guarantees about the system's behavior. With more optimization, our system can be modified to provide certain guarantees. Hence, this rigor can be combined with learning methods that have limited influence, but can still model the intricacies of the throwing problem as described above.

E. Contributions

In this section, we list the contributions of each team member. Each member worked on general debugging in the project. More specific contributions are as follows:

Arif Kerem Dayi:

- Inverse dynamics controller, and integration with the planner
- Inverse kinematics controller
- Using interpolation to generate continuously differentiable trajectories and trying different interpolations
- Generating numerical results and evaluation of the method.
- Writing the introduction, editing related work, writing system design, project setup, control, and evaluation sections in the report.
- Creating the simulation environment with two robots and managing drake systems

Hanqi Su:

- Throwing trajectory planning
- Projectile problem analysis
- Creating a minimal energy consumption method when constructing circular throwing trajectories
- Creating the simulation environment with two robots and managing drake systems
- Writing the related work, system design, projectile problem, and planning. Edited sections of the paper

Quincy Johnson:

- Simulation Testing and Demo Video creation
- Creating the simulation environment with two robots and managing drake systems
- Created point cloud constructions for object detection
- Implemented Antipodal grasping
- Designed high level task planner/state machine
- Adapted problem for 2+ robots
- Writing the abstract, perception/grasping, and grasping planning. Edited sections of the paper

REFERENCES

[1] Y. Liu, A. Nayak, and A. Billard, "A solution to adaptive mobile manipulator throwing," *arXiv preprint arXiv:2207.10629*, 2022.

[2] M. T. Mason and K. M. Lynch, "Dynamic manipulation," in *Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'93)*, vol. 1. IEEE, 1993, pp. 152–159.

[3] Y. Zhang, J. Luo, and K. Hauser, "Sampling-based motion planning with dynamic intermediate state objectives: Application to throwing," in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 2551–2556.

[4] H. Chen, B. Zhang, and T. Fuhlbrigge, "Robot throwing trajectory planning for solid waste handling," in *2019 IEEE 9th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER)*. IEEE, 2019, pp. 1372–1375.

[5] 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.

[6] H. Kasaei and M. Kasaei, "Throwing objects into a moving basket while avoiding obstacles," *arXiv preprint arXiv:2210.00609*, 2022.

[7] A. Green and J. Z. Sasiadek, "Dynamics and trajectory tracking control of a two-link robot manipulator," *Journal of Vibration and Control*, vol. 10, no. 10, pp. 1415–1440, 2004.
[8] E. F. Camacho and C. B. Alba, *Model predictive control*. Springer science & business media, 2013.
[9] J. B. Rawlings, "Tutorial overview of model predictive control," *IEEE control systems magazine*, vol. 20, no. 3, pp. 38–52, 2000.
[10] R. Tedrake and the Drake Development Team, "Drake: Model-based design and verification for robotics," 2019. [Online]. Available: https://drake.mit.edu