A Physics Based Full Stack System for Throwing and Catching

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 both throw and catch rigid objects with 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 without. The ability to detect and catch objects in motion has similar usefulness by allowing for a robot to intercept and later interact with objects not initially in it's operation space. In this paper we create a powerful system that combines both throwing and catching. The system features two robots that can either be a *throwing bot* or a *catching bot* at any time. We achieve this through the use of geometric based antipodal grasping, object motion tracking, trajectory planning, and high level planning [potentially edit]. The result is a [Fill in]**

***Index Terms*—robotic manipulation, perception, trajectory planning, grasp planning**

I. Introduction

Throwing and catching are very important topics in the field of robotic manipulation, as they are prerequisites for more complex robotic manipulations such as badminton, basketball, juggling, table tennis, and so on. Even in simple daily tasks, humans use throwing skills when putting something in trash, passing an object to a friend, or putting objects in boxes that are not in reach. Similarly, humans use catching skills while receiving an object from someone else or when a glass or plate falls from the side of a table. Hence, it is critical to implement throwing and catching skills on robotic systems for better robot performance and realistic human-robot interaction.

Here, we will also refer to some prior work and how we will incorporate it in our project.

We also will at a high level formulate what we do in our project.

Finally, we will foreshadow how we will be testing our robot.

II. Related Work

Note: We will add proper citations here, using the IEEE format.

More precisely, we can describe the problem as follows. First, there is the throwing problem. Questions such as planning the throw trajectory, being able to implement effective control for differently shaped objects, and how to move to and plan a reasonable position to maximize the success of the throw are some of the major questions that are present in just the problem of throwing. Then of course there is the catching problem. For this operation, there is the challenging problem of completing real-time trajectory planning of perceived aerial objects in a very short time. At the same time, achieving a good prediction for object trajectories, and optimizing the capture mechanism of the end-effector are both interesting problems that will prove challenging, and have many applications outside of direct throwing and catching, such as safe human-robot interaction under uncertainty.

In "A Solution to Adaptive Mobile Manipulator Throwing" (Yang Liu, et al.), they introduce an adaptive method for throwing for mobile manipulators. The paper proposes 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.

In "Revisiting Ball Catching with a Mobile Manipulator: A Discrete Trajectory Planning Approach" (Ke Dong, et al.), this paper addresses this problem by constantly interweaving predictions and replanning to successfully catch the ball in the air and implementing a new and different approach to the problem than Sequential Quadratic Programming (SQP). Another important paper that we feel will be critical to our work is "TossingBot: Learning to Throw Arbitrary Objects with Residual Physics" (Andy Zeng, et al.) which will help us in understanding the challenges and methods of throwing an object [1].

"Coordinated multi-arm motion planning: Reaching for moving objects in the face of uncertainty" (Mirrazavi Sina, et al.) since this paper might give us some idea about catching an object in motion and under uncertainty since the paper involves intercepting a moving object.

Compared to other work, our work falls under the category of physics based controllers. **We will elaborate on this**

III. System Design

This section is under continuous change, but this should serve as a good starting point! The tasks of throwing and catching require numerous components to work effectively such as a perception unit, pick/place planner, a trajectory planner, state estimators for object in motion, and a low level controller. Furthermore, a lot of tasks require real time

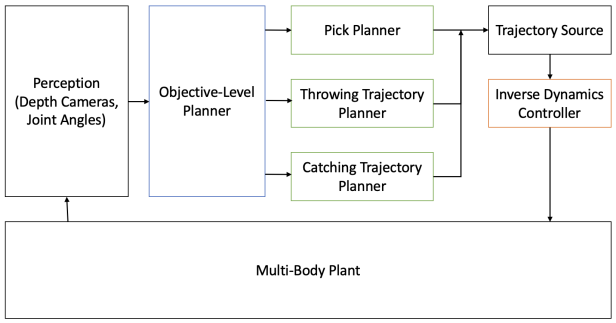

Fig. 1. A high level planner (blue) decides on which task to execute next. Then, the low level planners (in green) decide on what the trajectory in the next time steps should be, and sets a trajectory source. This is then fed to the inverse dynamics controller to decide on the right actuation inputs for the robot.

planning. Therefore, we carefully design our system to ensure that all of these components work correctly.

Since our robot needs to achieve multiple tasks, we have a central objective-level planner which decides on which task to complete next. Then, the right lower level planner (pick planner, throwing trajectory planner, catching trajectory planner) is activated and sets the desired trajectory to track. A trajectory source is used to constantly tell the inverse dynamics controller where the robot should be, and the inverse dynamics controller decides on the right actuation input. Then, the actuation input is fed into the plant and the robot moves, generating the sensor inputs for the next step and the cycle continues. This is illustrated in Figure 1.

IV. Throwing an Object to a Fixed Location

In this section, we will give a precise problem definition throwing and how we approach it. We are likely to first use physics to determine which locations and velocities are feasible to release the object, and then find a trajectory that closely matches the flight trajectory near the release.

A. 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).

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:

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

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

will add an image there to help people to understand the problem

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. 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 is the y axis in the two-dimensional problem, and the robot's orientation is jointly determined by (x, y). The generated angle is defined as the orientation angle, and the ray where the orientation angle lies is the x axis in the two-dimensional problem.

Consider initial position $p0$ = (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:

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

will add two images there to help people to understand the problem

B. Planning

Since we have the target position, then if we want to achieve a good throwing task, we need to solve the initial position, initial speed, and initial angle of the iiwa robot.

solve orientation interpolation problem

a trajectory for quaternions that are interpolated using piecewise slerp (spherical linear interpolation)

def $interpolate_orientation(Rotation_A, Rotation_B)$

linear interpolation

def $interpolate_pose_linear(X_A, X_B, t)$

circular interpolation

def $interpolate_pose_circular(X_A, X_B, t)$

I'll go into more detail about our approach to trajectory design, the considerations, and what needs to be talked about.

C. 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:

$$\min_q \|q - q_{nom}\|^2 \text{ subject to} \tag{10}$$

$$p_d - d \leq f_p(q) \leq p_d + d \tag{11}$$

$$\text{angle}(f_R(q), R_d) \leq \theta \tag{12}$$

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 [2]:

$$\text{vd\_command} := k_p(q_d - \hat{q}) + k_d(v_d - \hat{v}) + \tag{13}$$

$$k_i \int (q_d - \hat{q})\, dt + \alpha_{command} \tag{14}$$

$$\text{force} := \text{inverse\_dynamics}(\hat{q}, \hat{v}, vd\_command) \tag{15}$$

Then, force is used as actuation input. 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.

D. Methodology

Our overall experimental process is as follows:

1. The throwing robot locates and grabs the target object according to the depth camera information

2. After successful capture, the throwing robot will first move to the safe position above the captured object (to avoid collision with the box), and then keep the height of Z-axis unchanged according to the direction of the target position. Taking the distance from this point to the base of the throwing robot as the radius, the throwing robot arm will rotate counterclockwise to the direction of the target, and then move linearly to the initial point of the throwing trajectory.

3. When the end claw of the throwing robot hand reaches the initial point, the robot can choose linear trajectory or circular trajectory for trajectory planning of the throwing path and perform the throwing task.

4. When the throwing robot is ready to throw the object, the grasping robot also starts to prepare. The grasping robot will make a plan according to the throwing trajectory and move to the predicted position in time after the object is thrown. Here we simplify the fetching. We have the grasping robot grab a smaller box to hold the object, which is much more likely than the grasping robot claw. This also requires that our forecast trajectory needs to be more precise.

V. Catching an Object

Here, we will refer to previous work on estimating the trajectory of a moving object and the dynamical systems approach use to generate a trajectory for the catching robot. There is some nice prior work we can reference here.

A. Perception

[Explain perception in catching]. If an object ends up on the ground within the robot's reachable operation space as a result of problem initialization or simply failing to catch a previously thrown object, the cameras surrounding the robot will form a point cloud of the object. As long as the robot is a *catching bot*, the catching planner will transition to the *grasping_state*.

VI. Evaluation

A. Simulation Setup

We will describe our simulation setup, and what we are aiming to test. Some metrics we can use is throw accuracy, catch accuracy in randomly generated environments. Since we will use the simulation, there won't be much uncertainty if we run the same setting over and over, but we can randomly generate scenes and test the accuracy.

We will also add a screenshot of our simulation here

B. Testing Throwing Accuracy

Variables we can change are: throwing distance

C. Test Catching accuracy

We can test different velocities for the incoming object.

D. Testing Controller Performance on tracking Trajectory

An important aspect that determines throwing and catching performance is trajectory tracking performance. More precisely, we look at the mean squared error (MSE) in trajectory tracking as follows. For time steps $t \in \{0, 1, \ldots, T\}$ we are given observed joint angles and velocities $\hat{q}(t)$ and $\hat{v}(t)$ and have desired angles and velocities $q_d(t)$ and $v_d(t)$. We combine these into vectors $\hat{x}(t) = [\hat{q}(t)\ \hat{v}(t)]^T$ and $x_d(t) = [q_d(t)\ v_d(t)]^T$. Hence, we define total error Δ_{track} as

$$\Delta_{track} = \frac{1}{T} \sum_{t=0}^{T} \|x_d(t) - \hat{x}(t)\|^2 \tag{16}$$

To make more sense for throwing and catching, we might also look at Δ_{track} restricted to only the last few time instants near throwing and catching.

In addition to this metric, we look at error over time. Define $e(t) = \|x_d(t) - \hat{x}(t)\|^2$. We will plot $e(t)$ over time to see controller deviation and recovery. We might compare the inverse dynamics controller to the baseline position controller (and even ik maybe).

VII. Conclusion

In this part, depending on how good our methods work, we will elaborate on improvements. For instance, if our method does not work as we wanted, we can suggest why this is and how it can be improved. If it works well, we can explain why, and give directions for further research.

References

References

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

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