Throwing in Drake

6.881 Fall 2020 Final Project

Tony Wang*
twang6@mit.edu

Daniel Yang*
dxyang@mit.edu

***Abstract*—Throwing is a useful strategy that can be leveraged by robotic arms to manipulate objects to poses that are not kinematically feasible for the robot arm as well as increase the throughput of robotic manipulation systems. In this work, we build a robotic system in Drake that is capable of tossing objects to reach a range of desired positions, approaching the targets with desired approach angles as well. Our final system consists of three key components - an optimization-based projectile motion planner, a robot motion controller, and a gripper release controller. We document and discuss the iterative development of these components in addition to evaluating the performance of our system at hitting targets across a grid of locations. Our system hits targets within 3 meters of the robot with an average error of 0.187 m and targets within 5 meters of the robot with an average error of 0.497 m and always within 10 degrees of the desired approach angle and in the heading direction originally desired. Representative videos of our robot hitting targets at 7.07 m, 4.24 m, and 1.41 m away from the base are shown here. We also show our robot successfully tossing an object into a 4m high goal as well as a 4m low goal. We further show our robot successfully making a three point basketball shot, 7.25 m far and 3 m high as defined by the NBA, in this video.**

I. Introduction

Recent advances in robot manipulation have seen remarkable success at grasping [1]–[3] and pick and place [4], [5] in real unstructured environments using data-driven methods. However, these methods focus on manipulation where the object is restricted to be touching the gripper during the whole motion. Inspired by [6], our work focuses on building a robot capable of tossing objects to a desired target location. By tossing an object, a robot expands the range of poses to which it can manipulate an object. This is particularly useful for stationary robots that do not have a mobile platform on which to navigate through space. Furthermore, [6] note that by tossing objects their robot is able to achieve significantly more pick and places per hour than other state of the art systems, highlighting the ability of tossing to save time and increase robot efficiency.

In this work, we explore the components necessary to effectively build a tossing robot in the Drake robotics toolbox [7]. First, we derive the analytical equations necessary to plan a successful toss. We then discuss how to control our robot to achieve such a toss. Finally, we present experimental results showing the accuracy of our method as well as possible areas for improvement.

Project code at https://github.com/dxyang/manipulation.

II. Related Work

TossingBot [6] is a recent success from which our project draws inspiration. TossingBot jointly learns object grasps and tossing parameters from RGB-D input of a bin of objects, with grasp success scores and velocity residuals learned on a dense per-pixel level. TossingBot approximates objects as point particles only affected by gravity and assumes gripper velocity approximates object velocity. However, these assumptions can be inaccurate for grasps off the center of mass of the object. To combat this, TossingBot uses a learned residual physics module that makes corrections for phenomena that simple projectile motion does not account for. Overall, TossingBot shows that jointly learning grasps and tasks provides mutually beneficial training signal, as grasps that are successfully tossed are likely to be more physically stable.

In contrast to TossingBot, our work consists of a baseline physics-based model and focuses on tossing a single object successfully from a predefined grasp. We take a similar approach in constraining the initial pose of the toss, but allow for more freedom in some of our experiments, for example allowing the release angle to vary between tosses. We also assume that our object center of mass is within the gripper at some known translational offset from the gripper body frame.

Past work in adaptive control provide other methods to deal with imperfect understandings (e.g., unknown center of mass, unknown moment of inertia) of the objects being tossed. One work utilizes adaptive control to estimate both manipulator parameters and payload parameters online via data from torque sensors collected during a sufficiently rich trajectory. It solves a least squares problem to estimate the unknown parameters [8]. An unpublished work by the same authors referenced in the course materials, lecture 19, show such a procedure occurring before an object is tossed [9].

While our work does not incorporate more complex object geometries where online estimation of the exact center of the mass would be helpful, one could envision incorporating such an algorithm to augment the baseline physical controllers utilized by [4] or our work so that the release velocity is estimated at a more precisely. Similarly, knowing the mass would also allow us to incorporate slower or faster motion to minimize the likelihood of slip occurring at the interface between the gripper fingers and the thrown object.

Finally, other work deals with catching and tossing of more complex objects, specifically paper airplanes whose trajectories are non-trivially affected by aerodynamic forces [10].

III. Method

We utilize the Kuka LBR iiwa 7-DOF robot arm with the Schunk WSG 50 parallel gripper as defined in Drake. We utilize the clutter clearing version of the `ManipulationStation` that creates two bins with the robot as well as setting up useful input and output ports.

Our system consists of three components to help achieve our tossing task — a projectile motion planner, the robot kinematic controller, and the gripper controller. We envision these parts to be extensible and compatible with sensing modalities to toss and adjust plans for any object. However, our system as is focuses on tossing geometric primitives, specifically a sphere, with omniscient knowledge of the pose of the sphere. While grasping and the location of the grasp relative to the object center of mass is critical to successful tossing, we do not focus on grasping within our task. Instead we predefine grasps that will lead to successfully grasping the object such that the center of mass is between our grippers. As noted in Section II, we could extend our work with adaptive control techniques to estimate the center of mass of the object and adjust our planning accordingly.

A. Projectile Motion

For motion constrained to a plane where x is the horizontal axis and y is the vertical axis (upon which gravity acts downwards), the basic equations of motion for a projectile with initial position (x_0, y_0), initial velocity v_0, and launch angle θ relative to the horizontal plane, are

$$x(t) = x_0 + t\, v_0 \cos\theta, \tag{1}$$

$$y(t) = y_0 + t\, v_0 \sin\theta - \frac{1}{2} g t^2. \tag{2}$$

If we would like the projectile to hit an target at $(x_0 + \Delta x, y_0 + \Delta y)$ after a time duration T of flight, we must have

$$\Delta x = T\, v_0 \cos\theta, \tag{3}$$

$$\Delta y = T\, v_0 \sin\theta - \frac{1}{2} g\, T^2 = \Delta x \tan\theta - \frac{1}{2} g\, T^2, \tag{4}$$

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

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

Though our robot operates in a 3D world, we can reduce 3D projectile motion to this 2D case by restricting our toss to occur within in a vertical plane that contains both the initial and target position. For an initial position $p_0 = (x_0, y_0, z_0)$ and target position $p_{\text{target}} = (x_t, y_t, z_t)$ (both in the world frame), we can calculate the horizontal displacement (looking down upon the XY plane) and vertical displacement to use with our 2D equations:

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

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

Similarly, we can define a heading, ϕ, which along with the Z-axis defines the plane along which our projectile motion lies:

$$\phi = \operatorname{atan2}(y_t - y_0, x_t - x_0). \tag{9}$$

Our baseline projectile motion planner takes an initial position, a target position, and a launch angle, and uses the above equations to generate a desired release velocity. The motion planner assumes that the initial position, target position, and the origin are coplanar (i.e., the projectile motion plane is the XZ plane rotated by ϕ around the Z-axis).

Later in Section III-D, we do not assume a set initial position for our throw but instead allow an optimizer to choose between various positions that are along the throwing motion arc of our robot.

B. Robot Motion Control

While simple physics tells us the dynamics we must reach at release time, commanding the robot to actually achieve such a configuration is nontrivial. We tried a couple approaches for motion command before finding one that worked well.

1) End effector space: Our second approach drew inspiration from the door opening exercise where we generated a trajectory of gripper poses, $X_1^G, \ldots, X_n^G$, and defined associated timestamps at which each pose should be reached [9]. Each of these poses was then converted into joint space via solving an inverse kinematics optimization problem. Continuous motion was promoted by solving each optimization problem with an initial guess equal to the previous solution, and a joint centering cost was also present.

We initially defined a circular arc of poses as shown in Figure 1. With a perfectly circular arc, we could analytically calculate the velocity vector at each point along the arc and use our projectile motion planner to determine a required launch velocity. We ended up fixing the release to be the halfway point of the arc which results in a launch angle of $45°$. Finally we compare the required launch velocity to the base launch velocity vector and scale time over which the throw motion occurs to get our desired speed.

Unfortunately, this method was prone to errors. The end effector pose waypoints we defined were not guaranteed to be always reached and motion between waypoints may have brought the end effector off the trajectory as multiple joints of the robot actuate. This resulted in throws that deviated significantly off of our desired trajectory and intended plane of motion as shown in this video.

2) Joint angle space: With the insights from the poor performance of our first method, we decided that we need to better constrain the motion actuation of all the joints during our throwing motion. For our seven joint robot, we noticed that with all other joints set to 0, moving joints 4 and 6 led to clean arc motions along the same vertical plane. Furthermore, rotating joint 0 would change the heading of the throw to fit into our parameterization Section III-A.

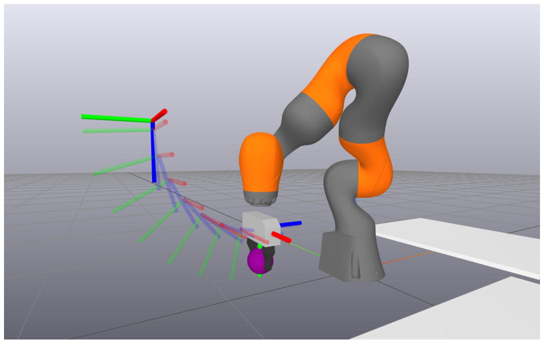

Fig. 1. Our first attempt at controlling the robot to reach a desired velocity at release involved defining a series of waypoints in end effector space, $X_1^G, \ldots, X_n^G$, that traced a perfect quarter circle. This was ineffective as the robot did not perfectly follow the trajectory. Different joints would accelerate at different rates, which caused the actual motion to deviate off of the plane of this arc, leading to throws in the wrong direction. A video of this throw can be found here.

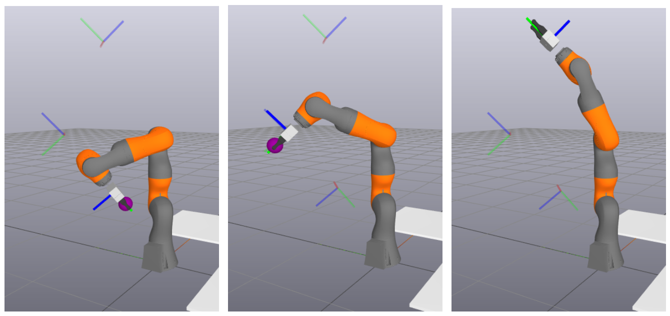

Fig. 2. The nominal prethrow and postthrow joint states with an intermediary halfway state. The movement along this arc results solely from actuation of joints 4 and 6. The values could be adjusted if one were designing a throwing robot for a specific application (for example, a paper tossing robot that could throw at far targets or a basketball shooting robot that could throw at high targets).

Thus we define pre-throw, post-throw, and release joint configurations as follows:

$$q^{\text{prethrow}} = \left(\phi, 0, 0, q_4^{\text{initial}}, 0, q_6^{\text{initial}}, 0\right), \tag{10}$$

$$q^{\text{postthrow}} = \left(\phi, 0, 0, q_4^{\text{final}}, 0, q_6^{\text{final}}, 0\right), \tag{11}$$

$$q^{\text{release}}(\lambda) = q^{\text{prethrow}} + \lambda\left(q^{\text{postthrow}} - q^{\text{prethrow}}\right), \tag{12}$$

with $\lambda \in [0,1]$ In practice, we set $q_4^{\text{initial}} = 1.9$, $q_4^{\text{final}} = 0.4$, $q_6^{\text{initial}} = -1.9$, and $q_6^{\text{final}} = -0.4$. This leads to a prethrow and postthrow configuration as shown in Figure 2.

We control our robot by linearly interpolating between prethrow and postthrow joint angles. In other words, if we start the motion at time $t = 0$, at time t our commanded joint angles are

$$q(t) = q^{\text{prethrow}} + \frac{t}{\tau}\left(q^{\text{postthrow}} - q^{\text{prethrow}}\right), \tag{13}$$

where τ is the time over which the entire commanded motion occurs.

We can utilize the Jacobian of the forward kinematics of the robot, to determine the launch angle at $q_{release}$. Once we have the launch direction, we can calculate the required launch speed using the equations in Section III-A. . Using the following relation, we can solve for dq, the velocity at which the joints should be moving, for the object to achieve the desired velocity:

$$^{W}v_{\text{desired}}^{O} = J_{\text{spatial}}^{O}\left(q^{\text{release}}\right) dq. \tag{14}$$

This can be done through least squares or determining the ratio between the target velocity and a reference velocity for a reference change in joint angles.

Note that O refers to the coordinate frame at the object center of mass, not the frame centered at the gripper pose. We assume that the transformation representing the object center of mass frame relative to the gripper body frame is pure translation in the y-direction, which is not always true in practice especially with imperfect grasping and unusually shaped objects.

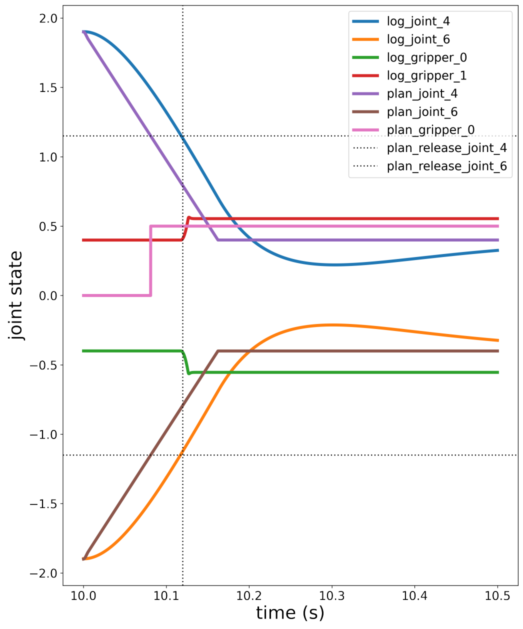

Fig. 3. Planned and logged-from-simulation state for joint 4, joint 6, and the gripper during the throwing motion, occurring here starting from 10.0 seconds. The horizontal dashed lines show the joint state at release that we expect. We can see that the sim states deviate notably from the planned states as the joints are not able to instantaneously move from 0 to constant velocity, instead following a gradual ramping up period. This leads to a delay between the planned gripper release time (shown by the discontinuity in the light pink line) and the actual release time (the vertical dashed line) that is accounted for by our closed-loop gripper controller.

In practice, we also found that our assumption of constant velocity to be imperfect. The robot cannot instantaneous output non-trivial joint velocities, and instead ramps up to them as shown in Figure 3. This is especially true when we are commanding the robot to move fast, which occurs when we throw to far away targets. This ramp up means that we cannot naively release our object at some predetermined time step like in door opening exercise [9]. We discuss our solution to this problem in the next section.

C. Release Control

We implement a closed loop gripper controller that opens the gripper to release the object. Specifically, we iterated through three versions of this controller with increasing complexity. Each controller shares a basic finite state machine structure that ensures we don't open the gripper prematurely during the motion in which the robot is picking up the object for maneuvering to the prethrow pose. As shown in Figure 3, a closed-loop gripper controller can help us account for the differences between our plans and reality, in this case resulting from a faulty assumption of being able to instantaneously change joint velocities.

1) Target Release Pose: Command release when a predefined pose in world space is reached. This only depends on current joint state from `iiwa_position`. This faced numerical issues occasionally when the robot would be moving quite fast and our closeness conditions for the world space translation would never be satisfied. Thus, the object would never be released. This is alleviated with a finer simulation timescale at the cost of iteration speed or by making the closeness threshold larger (order of magnitude of cms).

2) Projected Projectile Pose: Command release when its projected you would hit the target if released. This depends on the current joint state from `iiwa_position_measured` for Jacobian calculation and current joint velocity from `iiwa_velocity_estimated` for object translational velocity estimation. This faces similar issues with the first method in that the closeness constraints may not ever be satisfied and thus we need a finer simulation or a looser threshold. During our throwing trajectory, it is also possible under certain configurations (e.g., trying to hit a very high or a close target) that there you could reach the target position on the rising or falling portions of projectile motion which occasionally caused our controller to mis-open.

3) Target Launch Angle: Command release when the current launch angle passes a predefined launch angle. This only depends on current joint state from `iiwa_position`. Since this operates based on a "greater than" inequality as opposed to a "closeness" inequality as the past two methods, this guarantees the gripper will always open. We also included an angle adjustment threshold that makes the robot release slightly before or after the predefined launch angle. This was used to tune the robot if it was slightly under or overthrowing.

D. Optimization-based projectile motion planning

With the above components implemented, we wanted to further iterate on our tossing system such that it could throw an object not just to a target position but also reach that target position with a desired steepness. This became apparent as we were tossing objects into the bin and would occasionally become blocked by the sides of the bin upon approach of our object. See here for a video.

One could also imagine applications where this capability is desirable. For example, with a basketball tossing robot, we might want a steeper angle for the satisfying "swish" as the ball enters the hoop. Another example would be for a narrower peg in hole tossing task, where the cone of acceptable approach angles may be more limited.

To enable controlling the landing angle, we move past the analytical equations defined in Section III-A and take an optimization based approach.[1] Our optimization problem is as follows:

$$\min_{\lambda,\tau} \quad \left(\Delta z - \widehat{\Delta z}\right)^2 + \max\left(\widehat{\phi} + 45^\circ, 0\right)^2 \tag{15}$$

$$\text{where} \quad v_0 = J^O(q^{\text{release}}(\lambda)) \cdot \frac{q^{\text{postthrow}} - q^{\text{prethrow}}}{\tau} \tag{16}$$

$$^W X^{G_{\text{release}}} = f_{\text{fk}}(q^{\text{release}}(\lambda)) \tag{17}$$

$$t_{\text{land}} = \sqrt{(\Delta x)^2 + (\Delta y)^2} / \sqrt{v_{0,x}^2 + v_{0,y}^2} \tag{18}$$

$$\widehat{\Delta z} = v_{0,z} \cdot t_{\text{land}} - \frac{1}{2} g\, t_{\text{land}}^2 \tag{19}$$

$$\widehat{\phi} = \arctan\left(\frac{v_{0,z} - g\, t_{\text{land}}}{\sqrt{v_{0,x}^2 + v_{0,y}^2}}\right), \tag{20}$$

where λ is the fraction into our trajectory we release in Equation 12, τ is the total throw time, and Δ values are determined by the difference between the x, y, and z coordinates of the translation components of the target and release poses.

In a nutshell, we would like a throw configuration (defined through λ into the trajectory and t determining the joint velocities) where the projected change in height is close to the desired change in height, the approach angle (note that for a descending object this is negative) is at most $-45°$, and the joint velocity is as low as possible.

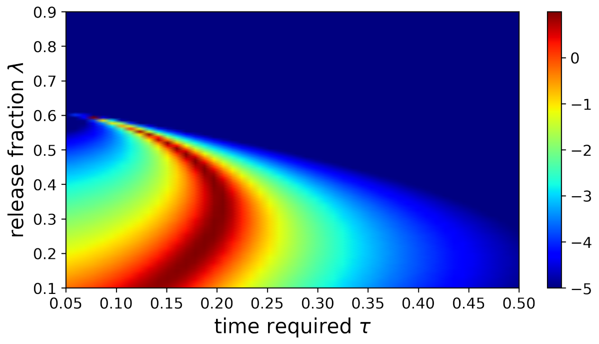

Fig. 4. Plot of the cost landscape for our optimization problem for a throw targeting (-3, 3, 0) meters. Plotted values are transformed by $f(x) = 1 - \min(6, \log(1 + x))$ to make the visualization nicer. Thus the red regions correspond to low objective values in Equation (15).

We implement this optimization problem using the differential evolution method as implemented in SciPy [11], [12]. We show a visualization of the cost landscape for one throw targeting (-3, 3, 0) meters in Figure 4.

With our new method, we can use same the gripper controllers as defined in Section III-C and control the robot motion

[1]An optimization-based approach is needed here because the end-effector trajectory does not have a nice closed form. However, if we make some simplifying assumptions, we actually get some very pretty results. See Appendix A for details.

similarly in joint space as defined in Section III-B. All that has changed is we are no longer assuming a specific release point for all throws.

IV. Experiments

We have representative videos showing the performance of our robot tossing at bins set at z = 0 m and a radial distance from the robot base at 7.07 m, 4.24 m, and 1.41 m. We also have videos of our robot successfully tossing an object into a high goal at position (-1, 1, 4) as well as a low goal at (-3, -3, -4), where the coordinates are in meters.

A. Quantitative error analysis

Say we throw to a target at (x_t, y_t, z_t) from an initial position (x_0, y_0, z_0). How can quantify this error?

Let $(\hat{x}_t, \hat{y}_t)$ denote the XY coordinates of the projectile at the latest moment the projectile has a z value of z_t (this is well defined for all throws in our experiments). We quantify throw error in the following ways:

XY-error: the Euclidean distance between $(\hat{x}_t, \hat{y}_t)$ and (x_t, y_t).

Angle-error: the signed distance between the actual approach angle and the planned approach angle when the projectile is at $(\hat{x}_t, \hat{y}_t)$.

ϕ-error: $\text{atan2}(\hat{y}_t - y_0, \hat{x}_t - x_0) - \text{atan2}(y_t - y_0, x_t - x_0)$.

For our experiments, we used a joint-angle based throw motion, optimization based projectile motion planning, and a target-launch-angle based gripper controller. We also tried a few different values of the launch threshold parameter from our gripper controller discussed in Section III-C3. Results are given in Figure 5.

Our experimental results indicate that our robot has essentially zero ϕ-error (indicating that the robot always throws true to our intended heading), mild land angle error ($< 12°$ in all situations and $< 5°$ in most), and a decent amount of XY-error. However, we note that XY-error is greatly reduced for closer throws compared to farther throws (0.187 m error for the best parameter for throws within 3 m of the robot and 0.475 m error for the best parameter for throws within 5 m of the robot). We give some hypotheses for the causes of these errors in the following section.

B. Sources of Error

In IV-A, we noted that our method often misses its targets by up to half a meter. We hypothesize that one source of this error is gripper-object dynamics. In particular, we observed that during the throw, the object would slightly slip in the gripper's grasp. Since we plan the throw based on the initial position grasp position, a slip in grip results in the actual throw trajectory from the planned trajectory. See Figure 6 for a visualization of this slipping.

We think the slippage may be caused by our gripper not having enough friction. However turning up the `mu` values in the gripper `.sdf` files did not seem to have much of an effect.

thresh	mxdist	rms-XY-e.	rms-angle-e.	rms-ϕ-e.
$-3°$	∞ m	.670 m	4.88°	.001°
$0°$	∞ m	.497 m	2.37°	.001°
$3°$	∞ m	.475 m	5.33°	.002°
$6°$	∞ m	.624 m	11.5°	.002°
$-3°$	3 m	.357 m	2.53°	.002°
$0°$	3 m	.250 m	3.85°	.002°
$3°$	3 m	.187 m	7.29°	.002°
$6°$	3 m	.292 m	10.6°	.003°

Fig. 5. Root mean square throw errors of our robot to targets with integer-coordinates in the region $[-5, -1] \times [1, 5] \times [-3, 3] \subset \mathbb{R}^3$ in the world frame. We report XY-error, angle-error, and ϕ-error. "thresh" means the release angle threshold for the gripper controller as discussed in Section III-C3, and "mxdist" indicates which subset of the points in the region we evaluate on.

Overall, we don't quite understand how contact physics works in drake, and this is something we would like to study more carefully if we had more time.

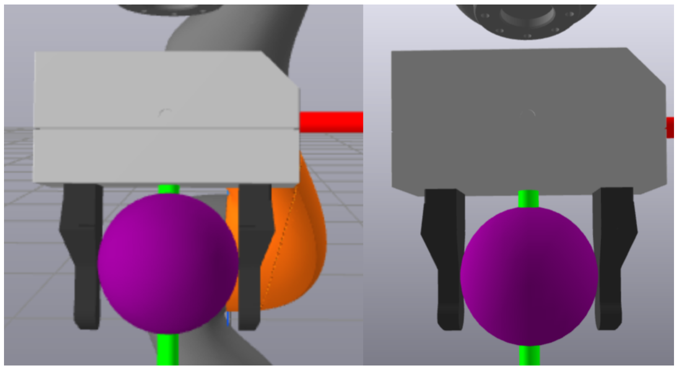
Fig. 6. A visualization of the ball slipping in our grasp. The left hand side is the grasp position of the ball, the the right hand side the grasp position right before release. Note the ball sits slightly lower in the gripper on the right.

C. Basketball Three Point Shot

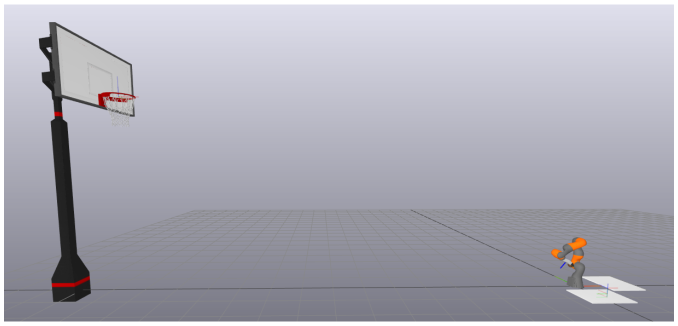
Fig. 7. We add a basketball hoop as our target and set it according to NBA regulations for a three point shot relative to our robot [13]. Specifically, the hoop is 7.25 meters from the base of the robot and 3 meters above the ground. A video of our robot making this shot can be found here.

Here we attempt to get our robot to successfully score a three point shot in basketball. At the furthest point along the three point shot line on an NBA basketball court, a player will be 23 feet and 9 inches away from the hoop which will be 10 feet above the ground [13]. We approximate this by setting our target to be a position 7.25 meters horizontally away from our robot and 3 meters vertically above the ground as shown in Figure 7. We adjust the range of our prethrow and postthrow joint configurations to be $q_4^{\text{initial}} = 2.1$, $q_4^{\text{final}} = 0.6$, $q_6^{\text{initial}} = -1.95$, and $q_6^{\text{final}} = -0.3$, hypothesizing that such a far throw might require some more "windup" of our joints. We utilize the optimization based planning method described in Section III-D as well as the target launch angle release gripper controller described in Section III-C3. A video of our robot successfully making this throw can be found here.

V. Conclusion

In this work, we have successfully built a robotic system in Drake that is capable of tossing objects to reach a range of desired positions, approaching the targets with desired approach angles as well. We iteratively experimented with multiple approaches to plan our motion as well as control the robot to successfully execute the motion. Our final system consists of an optimization-based projectile motion planner along with closed-loop gripper release control based on release angle. The throw trajectory of the robot is planned in joint space instead of end-effector space to constraint the throw along a single plane of motion.

Our quantitative analysis shows that our system hits targets within 3 meters of the robot with an average error of 0.187 m and targets within 5 meters of the robot with an average error of 0.497 m and always within 10 degrees of the desired approach angle and in the heading direction originally desired. Videos of the robot show qualitatively satisfying behavior as well.

Tossing was an interesting problem to work on given the fast dynamics at play as well as the clean analytical models we have for projectile motion and robot kinematic motion. Pushing our robot to move quite rapidly and achieve success at tasks like making a three point shot suggest to us that perhaps we should not be as conservative in limiting the motion of our real world robots.

For our toy problem of tossing a small sphere, this model works quite well. However, future work can be done to increase the robustness of our system and allow it to perform with a wider range of daily household objects. For example, incorporating adaptive control would allows us to estimate parameters of our system, such as the center of mass as well as the mass of the object, given a sufficiently rich motion trajectory and torque sensors. Knowing the exact center of mass would be useful in launch velocity estimation for example as could adjust our plans to better account for the center of mass not being exactly within our gripper.

We tried experimenting with rods as shown in this video and found difficulty in obtaining a robust throw. Similarly, knowing the mass of the object would allow us to better anticipate when slip could happen within our gripper-object interface as this was a source of error when we tried experimenting with heavier objects like the YCB mustard bottle as shown in this video. We would also be curious to incorporate learning into the planning and control portions of our project, for example in controlling the gripper release. Similarly, our project presupposes omniscient information about the pose of the object of interest. In any real world scenario, we would be estimating the pose of the object through RGB-D input. RGBD input and related 3D geometrical representations (point clouds, meshes, etc.) could provide a rich source of information from which to learn grasp poses or grasp quality metrics.

References

[1] A. Mousavian, C. Eppner, and D. Fox, "6-dof graspnet: Variational grasp generation for object manipulation," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 2901–2910.

[2] J. Mahler, M. Matl, X. Liu, A. Li, D. Gealy, and K. Goldberg, "Dex-net 3.0: Computing robust vacuum suction grasp targets in point clouds using a new analytic model and deep learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1–8.

[3] P. R. Florence, L. Manuelli, and R. Tedrake, "Dense object nets: Learning dense visual object descriptors by and for robotic manipulation," in *Conference on Robot Learning*, 2018, pp. 373–385.

[4] A. Zeng, S. Song, K.-T. Yu, E. Donlon, F. R. Hogan, M. Bauza, D. Ma, O. Taylor, M. Liu, E. Romo *et al.*, "Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 1–8.

[5] W. Gao and R. Tedrake, "kpam-sc: Generalizable manipulation planning using keypoint affordance and shape completion," *arXiv preprint arXiv:1909.06980*, 2019.

[6] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser, "Tossingbot: Learning to throw arbitrary objects with residual physics," *arXiv preprint arXiv:1903.11239*, 2019.

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

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

[9] R. Tedrake, "Robot manipulation: Perception, planning, and control (course notes for mit 6.881)." [Online]. Available: http://manipulation.csail.mit.edu/

[10] W. Hong and J.-J. E. Slotine, "Experiments in hand-eye coordination using active vision," in *Experimental Robotics IV*. Springer, 1997, pp. 130–139.

[11] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[12] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[13] N. B. Association, "Rule no. 1: Court dimensions – equipment." [Online]. Available: https://official.nba.com/rule-no-1-court-dimensions-equipment/

Appendix A
Hitting a desired target at a given angle, analytic version

Say we want to launch a projectile from $(0, 0)$ such that it hits (x, y) at an angle θ_{land} (i.e. the angle the land velocity makes counterclockwise from $(1, 0)$). What angle θ_{launch} should we launch at? If one works through the math (thanks Mathematica), one gets the following:

$$\theta_{\text{launch}} = \arctan\left(\frac{2y}{x} - \tan\theta_{\text{land}}\right). \tag{21}$$

Quite pretty!

A few interesting observations about this equation. It is symmetric in θ_{launch} and θ_{land}, meaning the inverse relation just swaps the two θs. Additionally, it does not depend on gravitational acceleration – in fact it cannot by unit analysis, since no part of the problem specification can cancel out the inverse second units in g.