Precise Robotic Throwing of an Adversarial Object

Isaiah Brand and Christopher Bradley
CSAIL. Massachusetts Institute of Technology
Cambridge, MA
{ibrand, cbrad} @ mit.edu

***Abstract*—Modern robot arms are capable of highly precise, powerful, and dynamic motion. Numerous applications leverage the available precision and power: robot arms are frequently used both to weld and to lift car chassis in assembly plants. There are comparatively fewer robot arms performing highly dynamic tasks. We would like to close this gap. In this project, we address the highly precise and dynamic task of throwing a water bottle such that it flips and lands upright. Our approach consists a multi-step optimization, which considers the arm dynamics as well as the complex contact dynamics of the bottle on landing. We validate our system in the PyDrake simulation environment.**

I. Introduction

Humans are unrivaled in the animal kingdom in their ability to throw objects precisely. This skill is thought to have given early humans an edge in hunting [1] and other tasks (no other animal species has made the MLB as of this printing). Today, humans regularly use throwing to save energy and time, to put objects into otherwise unreachable positions, or simply for fun.

We would like our robots to have the same manipulation abilities as humans. While there has been extensive work in relatively static manipulation domains, there is significantly less work that addresses dynamic tasks like throwing. Throwing in the context of robotics has the potential to increase the efficiency of maniupulation platforms in pick and place tasks, as well as extend the effective working area of manipulators [2].

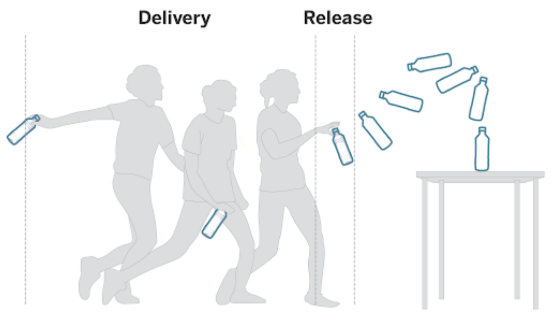


Fig. 1. A human performing bottle flipping.

In this work we investigate throwing with a robotic arm. We restrict our attention to the trendy task called "bottle flipping," in which one aims to throw a partially filled water bottle such that it performs one flip in the air and lands upright (Figure 1). We use the Drake simulation environment to attempt this task with a simulated Kuka Iiwa 7 robot arm, and a rigid-body cylinder representing the water bottle.

II. Related Work

There is a scarcity of work that focuses on robotic throwing, which makes this an exciting domain for future exploration.

One of the most recent example of prior work in the area of robotic tossing is TossingBot [2]. In their approach, the authors use an RL framework to learn grasps which lead to successful tosses across arbitrary objects. To do this, they conditions their grasps on the outcome {success, failure} of the subsequent toss, which improves overall performance. Additionally, they learn residuals on top of a physics engine for release parameters, meaning they learn the δ on top of the optimal velocity given by their simulator. In this way, they can account for areas in which their physics are imperfectly modeled (e.g., as it relates to the contact dynamics between gripper and object upon release). In our own investigation, we found that it would have been quite difficult to compensate for these factors in a purely analytical approach such as ours.

In a much earlier paper, Ishikawa et. al. develop a "kinetic chain" approach to throw a baseball at high speeds [3]. The kinetic chain refers to the idea that joint torques create kinetic energy which propagates from the driven joint out to the end effector. They formalize this notion, and propose an optimization problem (solved with SQP) that seeks to move the peak of the velocity waveform from the robot base to the end effector throughout the throwing trajectory. They control the arm along this trajectory using a gravity-compensated PD controller, which is a similar to the controller used in our approach. With the kinetic chain objective, they are able to drive the end effector at up to 6m/s. In our work, our focus is primarily on achieving throwing accuracy, rather than maximum velocity.

III. Methods

Our approach is comprised of three main stages. First, we perform an optimization to compute a release pose and velocity which will flip and land the bottle on the target platform. Next, the perception system is used to localize the bottle in the environment and a simple position controller is used to grasp the bottle. Once we have established a rigid grasp with the bottle, we compute a configuration space trajectory so that the robot arm drives the bottle to the release pose and velocity obtained in the the first part.

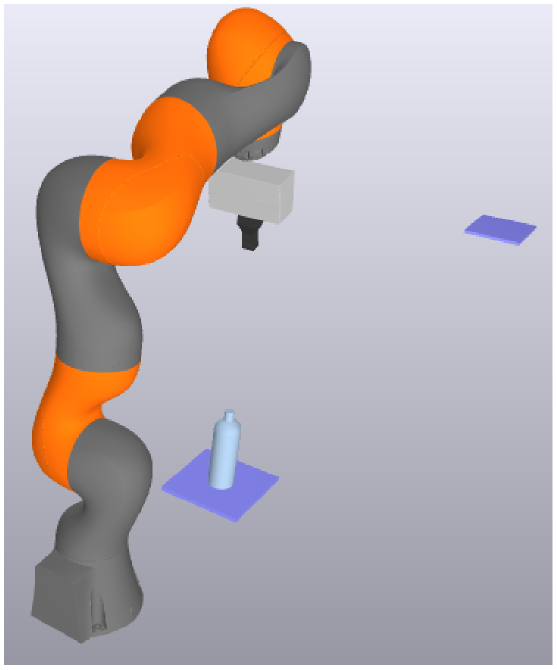


Fig. 2. Meshcat visualization of the initial setup for our problem. Notice the bottle mounted below the arm on a platform, and a second ‘target’ platform in the distance where our robot will attempt to flip it.

Although we could jointly optimize the configuration space trajectory and release state, we chose to separate the problem into two tasks to reduce the number of optimization variables and increase the speed of the computation. We also assume that the grippers are capable of releasing the bottle instantaneously, as modelling the dynamics of the release is beyond the scope of this project.

A. *Perception*

In order to make our simulation more complete, we do not assume that we have access to the initial pose of the bottle. Therefore, before we can even consider manipulating the bottle, we must be able to reliably estimate the pose of where our gripper must be placed to grasp it. To solve this perception problem, we mount an RGBD camera offset from the seventh link in our iiwa robot arm. From this sensor, we receive a point cloud, as well as both a color and depth image.

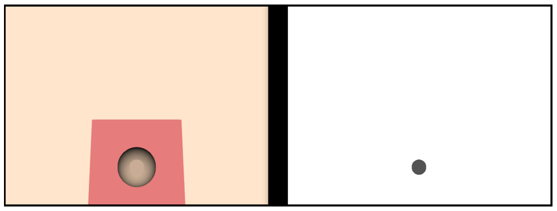


Fig. 3. Side by side color and masked depth image captured from a camera attached just above the gripper. The depth image on the right is masked to show only the cap of the bottle in order to plan a grasp.

We primarily rely on the depth image to estimate our grasp pose, with our goal being to grasp the bottle from above, with the cap in the center of the gripper facing down. To do this, we first identify the closest point on the bottle, then mask all parts of the image that are further than a small offset away from this point. We assume this masked image corresponds to the cap of the bottle. From there, we determine which pixels correspond to the centroid of the cap, and find the depth at that point. After extracting our camera’s intrinsics, we use this information to project our centroid and depth in pixel space into the frame of the camera, which we then use to in conjunction with the forward kinematics of the arm to determine the world frame of the bottle cap.

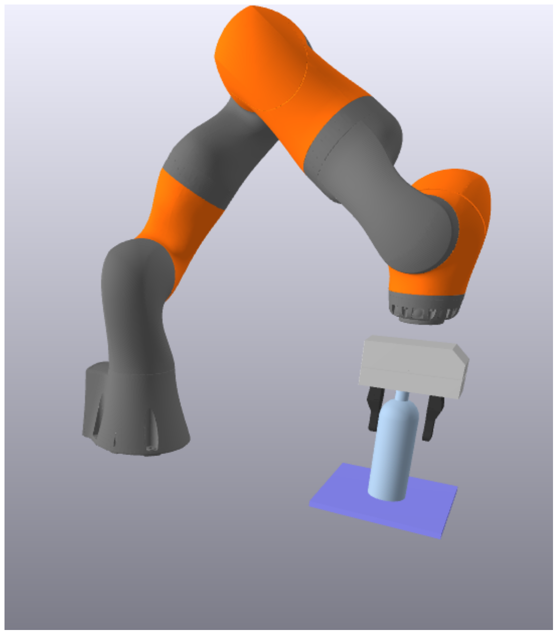


Fig. 4. Grasp in progress

This approach relies on a few assumptions for ease of implementation. First, we are assuming that the bottle is within view of the camera at the outset. We could have instructed our arm to sweep over the scene if the camera were not in position, but we decided to focus more on other parts of the project. Similarly, we assume the closest point on the bottle to the camera is on the cap. Once again, we could could play some other tricks (like plane fitting, ICP, or color threshold) to account for this if it were the focus of our project.

B. *Initial Grasp*

Once the grasp pose has been identified by our perception system, the next step is to move the gripper to that pose, securely grasp the bottle, and pick it up off the platform. First, we identify 5 steps in the process of picking up the bottle: 1) The initial state of the arm. 2) A ‘pre-grasp’ state with the gripper open above the bottle. 3) The arm in the grasping position in contact with the bottle with the gripper open. 4) The arm in the grasping position with the gripper closed. 5) In a ‘post-grasp’ state with the gripper closed around the bottle above the platform.

Next, we use the inverse kinematics of the arm to determine the joint angles for the different steps in this plan. With those q-values in hand, we use 'FirstOrderHold' to interpolate a trajectory in joint space for the arm to complete the grasp. Once this plan has been executed by the simulator, the bottle in in hand and the robot is ready to toss.

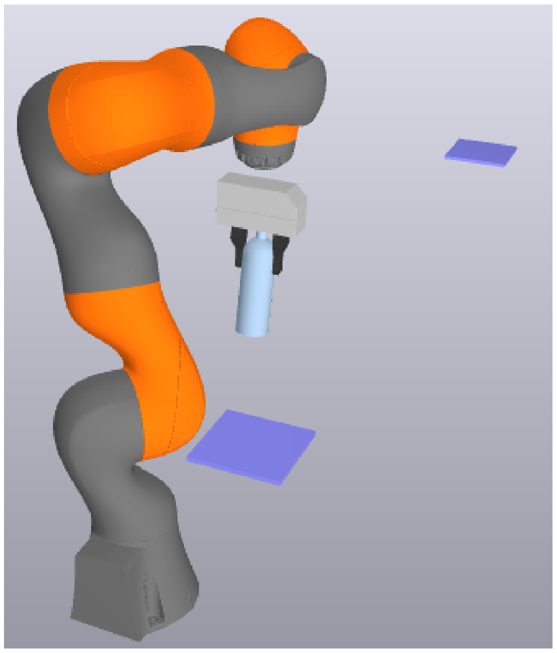


Fig. 5. Completed grasp, robot is ready to toss the bottle.

Our answer to the grasping problem is not a general solution by any means. By simply interpolating in joint space we open ourselves up to potential collisions with obstacles, or even the bottle itself. Though it worked for this problem setup, to make our approach more robust, we might consider planning in configuration space, using RRT* to determine the optimal trajectory for our grasp.

C. *Release State Optimization*

Since there are no aerodynamics forces in the simulator, it is relatively simple to compute the trajectory of the bottle after it is thrown. This means that given a target pose where we would like the bottle to land, and an initial pose from which we release the bottle, we can simply solve for an initial spatial velocity such that when the bottle follows free-body dynamics from that state it reaches the target pose. Note that this assumption may not transfer well to the real world, as a real bottle containing water has complex dynamics, and would be subject to air resistance.

1) Naive Release State Optimization: We initially formulate this problem as a constraint satisfaction problem so that we can leverage PyDrake's `MathematicalProgram` framework.

$$\begin{aligned}
\text{find} \quad & X, V, h \\
\text{s.t.} \quad & X_0 = X^{\text{initial}} \\
& X_T = X^{\text{target}} \\
& X_{t+1} = X_t + hV_t \\
& h \le h_{\text{max}} \\
& -h \le -h_{\text{min}}
\end{aligned} \tag{1}$$

where the optimization variables are pose trajectories $X = \{X_0, \ldots, X_T\}$ and spatial velocity trajectories $V = \{V_0, \ldots, V_{T-1}\}$ of the bottle in flight, and h is a timestep. X^{initial} specifies the release pose of the bottle and X^{target} specifies the target pose of the bottle. Note that we encode the rotations using a roll, pitch, yaw scheme, so that the angular velocities simply add with the current rotation to specify the free-body dynamics with forward Euler in constraint (5). The roll, pitch, yaw encoding also allows us to specify that the bottle completes a rotation in the air, by imposing that $\text{pitch}_0 = \text{pitch}_T - 2\pi$.

When the optimization has completed, we are only interested in V_0, as this is the release spatial velocity of the bottle. In practice we perform this optimization using $T = 500$ timesteps, and we find that it takes less than one second to compute a satisfying initial velocity.

We validated the results of this optimization by setting the initial condition of a simulation of the bottle to X_0, V_0 and running it forward to observe the outcome. This revealed that although although the bottle always achieves the target pose at the end of its flight phase, it almost always falls over after landing in that position. We believe that this occurs for two reasons. First, the bottle has momentum from its flight phase, which is not addressed in our optimization. Second, because there is a discrepancy between the timestep and simulation method encoded in our optimization, and the full PyDrake simulator, the bottle may not land exactly flat, and therefore either makes contact with its leading edge (farther away from the thrower) or trailing edge (closer to the thrower). These contacts impart a torque which can cause the bottle to fall over.

2) Rimless Wheel Constraint: We attempt to address both of these issue by introducing an additional constraint to our optimization that deliberately lands the bottle on the trailing edge, and uses the resulting contact torque to cancel out the angular momentum from the flight phase. Imposing a constraint that angular momentum be exactly cancelled by the impact torque is difficult, because it would require intimate knowledge of the contact dynamics of the simulator, as well as the simulation method. Instead, inspired by the rimless wheel model of walking robots, we approximate the dynamics of the bottle landing on the trailing edge and find criteria under which it will not fall over. In this approximation we make three additional assumptions

1) All collisions are inelastic.

2) The bottle is a point mass.
3) The first point of contact lies in the vertical plane spanned by the z-axis and the bottle's velocity vector just before impact.

With these assumptions we can derive the conditions on the angle at impact, and the velocity at impact under which the bottle will not fall over.

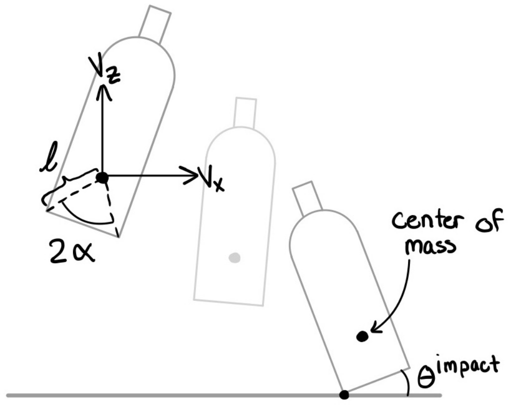


Fig. 6. A diagram of the bottle impacting the platform.

Without loss of generality, we assume the bottle is traveling in the (x, z) plane, with positive x velocity and negative z velocity. The bottle impacts the surface at angle θ^{impact}. The mass of the bottle is m, and α is the angle away from the edge of the bottle of a line between the center of mass and the corner of the bottle, and ℓ is the distance from a corner to the center of mass (see Figure 6 for details).

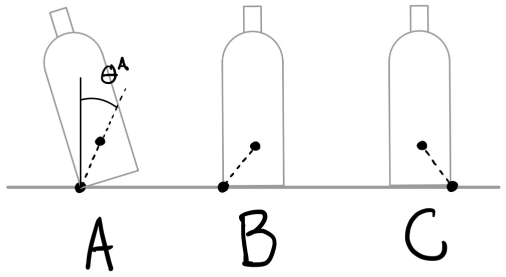


Fig. 7. A diagram of the bottle as it transfers from one pivot to the other.

We define a series of times, A, B, C, as shown in Figure 7. We can see that $\theta^A = \alpha - \theta^{impact}$.

Just after impact, the angular velocity of the bottle about the point of impact is (see Appendix Figure 8 for the derivation)

$$\dot{\theta}^A = \frac{v_x \cos(\theta^A) + v_z \sin(\theta^A)}{\ell} \tag{2}$$

the bottle then rotates onto the other edge starting at angle at angle θ^A with angular velocity $\dot{\theta}^A$. At the moment just before the other edge touches the ground it has angular velocity $\dot{\theta}^B$ (see Appendix Figure 9 for the derivation)

$$\dot{\theta}^B = \sqrt{(\dot{\theta}^A)^2 + \frac{2g}{\ell^2}\cos(\theta^A)} \tag{3}$$

just after the second edge touches the ground, the angular velocity about the new pivot is [4]

$$\dot{\theta}^C = \dot{\theta}^B \cos(2\alpha) \tag{4}$$

and the bottle will come to rest if

$$\dot{\theta}^C \leq \sqrt{2\frac{g}{l}\left(1 - \cos\left(-\alpha\right)\right)}. \tag{5}$$

By chaining these relations, we obtain a constraint on the bottle's velocity (v_x, v_z) and angle of impact θ^{impact} that will prevent the bottle from falling over away from the thrower. Also note that we do not want the bottle to immediately fall back toward the thrower in impact, so we constraint $\dot{\theta}^A \geq 0$. To add these constraints, to our original optimization, we observe that $\theta^{impact} = -X_T.\text{pitch} + 2\pi$, and we remove the equality constraint on X_T.pitch, instead constraining $0 \leq \theta^{impact} \leq \frac{\pi}{6}$. We also introduce a squared release velocity cost, to make the release velocity more attainable by the robot arm.

$$\begin{aligned}
\underset{X,V,h}{\text{argmin}} \quad & V_0^T V_0 \\
\\
\text{s.t.} \qquad X_0 &= X^{\text{initial}} \\
X_T.\text{position} &= X^{\text{target}}.\text{position} \\
X_T.\text{roll} &= X^{\text{target}}.\text{roll} \\
X_T.\text{yaw} &= X^{\text{target}}.\text{yaw} \\
-\theta^{impact} &\leq 0 \\
\theta^{impact} &\leq \frac{\pi}{6} \\
X_{t+1} &= X_t + hV_t \\
h &\leq h_{\text{max}} \\
-h &\leq -h_{\text{min}} \\
-\dot{\theta}^A &\leq 0 \\
\dot{\theta}^C &\leq \sqrt{2\frac{g}{l}\left(1 - \cos\left(-\alpha\right)\right)}
\end{aligned} \tag{6}$$

This optimization proved significantly more reliable in identifying release states that resulted in a stable landing ($> 50\%$). We were able to land on platforms between 1 and 4 meters away, with target-pose translations in all three axes. This method is imperfect however, as a result of the discrepancy between our forward Euler dynamics constraint and Drake's dynamics, as well as the assumptions we made in the rimless wheel approximation.

D. Trajectory Optimization

In Section III.*B* we grasped the bottle, which specified a transform from the gripper frame to the bottle X_G^{bottle}. In

Section III.*C* we determined a release pose X_W^{release} and spatial velocity V_W^{release} that will carry our bottle to the target landing pose. In this section we address the challenge of accelerating the bottle to the desired spatial velocity and pose without dropping it.

The PID-based `InverseDynamicsController` supplied by PyDrake allows us to command joint poses q_{desired} and velocities $\dot{q}_{\text{desired}}$. It might seem that we could simply use inverse kinematics to determine q_{desired}, $\dot{q}_{\text{desired}}$ given X_W^{release}, V_W^{release} and pass the results to `InverseDynamicsController`. This will fail, as the PID controller is a local controller which does not consider the dynamics of the system. It may move the arm closer to the release pose and velocity, but it is not guaranteed to achieve the release velocity at the same time as the release pose.

Instead we propose a joint-space trajectory optimization approach, to ensure that the release pose and velocity are achieved simultaneously. Our optimization variables are joint angle trajectories $q = \{q_0, \ldots, q_T\}$, joint angular velocity trajectories $\dot{q} = \{\dot{q}_0, \ldots, \dot{q}_{T-1}\}$ and a timestep h.

$$\underset{q,\dot{q}}{\text{argmin}} \sum_{t=0}^{t=T-1} \left(\frac{\dot{q}_t - \dot{q}_{t+1}}{h} \right)^2$$

$$\begin{aligned} \text{s.t.} \quad q_0 &= q_{\text{initial}} \\ f_{\text{kin}}^{\text{bottle}}(q_T) &= X_W^{\text{release}} \\ J^{\text{bottle}}(q_T)\dot{q}_T &= V_W^{\text{release}} \\ q_{t+1} &= q_t + h\dot{q}_{t+1} \\ h &\leq h_{\max} \\ -h &\leq -h_{\min} \end{aligned} \tag{7}$$

q_{initial} is the initial pose of the arm after grasping the bottle $f_{\text{kin}}^{\text{bottle}}(q_T)$ is the pose of the pose of the bottle as a function of the joint angles $J^{\text{bottle}}(q_T)$ is the Jacobian of that pose with respect to the joint angles.

Note that in this optimization we are minimizing the joint acceleration, as estimated by $\frac{\dot{q}_t - \dot{q}_{t+1}}{h}$, to ensure smooth, feasible trajectories.

It is important to note that this optimization does not constrain collisions between the bottle, the robot, and the environment. In practice we found that the resulting trajectories were infeasible due to self collisions.

IV. Discussion

In this project, we were able to: identify a valid grasp from RGBD input, plan and execute a trajectory to pick up the bottle, run an optimization to determine a target velocity and position such that the bottle will successfully flip onto our target platform, and plan (and execute) a path that results in the robot reaching these specifications. Unfortunately, we were not able to tie these components together into one system that performed our task as desired for a few different reasons. First and foremost, we could not model the bottle slipping in our gripper as it reached speeds needed to reach the target practice. This is a problem solved (in part) by the authors of the TossingBot paper via learning, which we did not have the time to implement. In principle, we could have solved this through parameter tuning however. In addition to slip, we also did not account for self-collision during our throwing trajectory, which led to the agent planning unattainable paths. These issues are certainly solvable, and we are confident that we can accomplish the bottle flip task with more time investment.

Code for this project is available on GitHub at https://github.com/IzzyBrand/robotBottleFlip

References

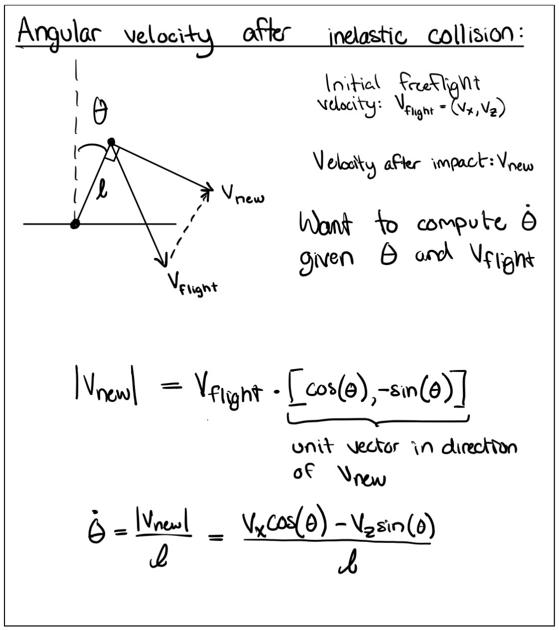
[1] J. G. Goldman, "Can humans throw better than animals?" *BBC Future*, 2014.

[2] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser, "Tossingbot: Learning to throw arbitrary objects with residual physics," *IEEE Transactions on Robotics*, 2020.

[3] T. Senoo, A. Namiki, and M. Ishikawa, "High-speed throwing motion based on kinetic chain approach," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008, pp. 3206–3211.

[4] R. Tedrake, "Underactuated robotics: Algorithms for walking, running, swimming, flying, and manipulation (course notes for mit 6.832)," Downloaded on December 11, 2020. [Online]. Available: http://underactuated.mit.edu/

V. Appendix



Angular velocity after inelastic collision:

Initial freeflight velocity: $V_{flight} = (V_x, V_z)$

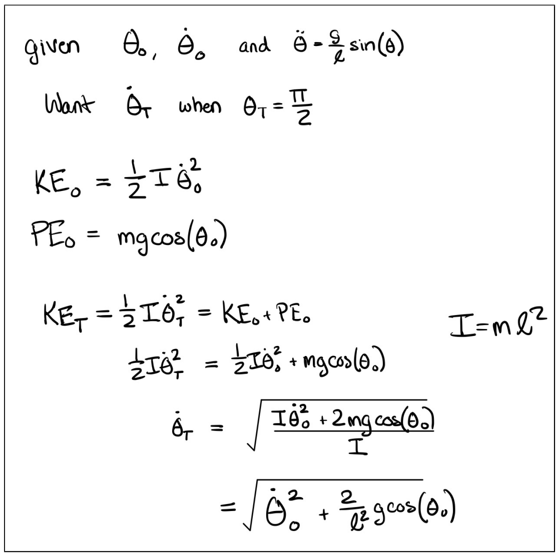
Velocity after impact: V_{new}

Want to compute $\dot{\theta}$ given θ and V_{flight}

$$|V_{new}| = V_{flight} \cdot \underbrace{[\cos(\theta), -\sin(\theta)]}_{\text{unit vector in direction of } V_{new}}$$

$$\dot{\theta} = \frac{|V_{new}|}{l} = \frac{V_x\cos(\theta) - V_z\sin(\theta)}{l}$$

Fig. 8. Derivation of the angular velocity after the bottle first makes impact with the surface.



given θ_0, $\dot{\theta}_0$ and $\ddot{\theta} = \frac{g}{\ell}\sin(\theta)$

Want $\dot{\theta}_T$ when $\theta_T = \frac{\pi}{2}$

$$KE_0 = \frac{1}{2} I \dot{\theta}_0^2$$

$$PE_0 = mg\cos(\theta_0)$$

$$KE_T = \frac{1}{2} I \dot{\theta}_T^2 = KE_0 + PE_0 \qquad I = m\ell^2$$

$$\frac{1}{2} I \dot{\theta}_T^2 = \frac{1}{2} I \dot{\theta}_0^2 + mg\cos(\theta_0)$$

$$\dot{\theta}_T = \sqrt{\frac{I\dot{\theta}_0^2 + 2mg\cos(\theta_0)}{I}}$$

$$= \sqrt{\dot{\theta}_0^2 + \frac{2}{\ell^2} g\cos(\theta_0)}$$

Fig. 9. Derivation of the angular velocity after the bottle falls to level.