6.843 - Final Project Report
Catching a Ping Pong Ball with an iiwa

Viraj Parimi
PhD Student
Model-Based Embedded and Robotics Systems
EECS

Cameron W. Pittman
Master's Student
Model-Based Embedded and Robotics Systems
AeroAstro

***Abstract*—For our final project, we programmed the kinematics of a Kuka iiwa using Drake [Tedrake and the Drake Development Team, 2019] to catch a ping pong ball with a ping pong paddle [1]. We identified distinct kinematic regimes in the catching task, which we split into a pre-initial contact state and post-initial contact state. We applied finite-state machine (FSM) techniques to model the paddle's trajectory off equations of motion of projectiles while in a pre-initial-contact mode, while we switched to a PD controller with offset stabilization in the post-initial-contact mode. Both modes of the system relied on passing desired end-effector velocities to a differential inverse kinematics (IK) controller with optimization-based joint limit enforcement. We simulated a perception pipeline by passing ground-truth ball and paddle states to our controller directly. Our design showed success in bringing the ball to a stop from a range of initial positions and velocities. However, keeping the ball at a stable equilibrium on the paddle surface for significant durations proved challenging and would require future work. We evaluated the performance of our system iterations by comparing their performance in stabilizing caught balls with different initial positions and velocities.**

I. Introduction and Related Work

Ping pong is a popular topic for roboticists. In our literature review, we found many authors demonstrating a range of robotics techniques using paddle-wielding robots. [Rapp, 2011] demonstrated perception and kinematics techniques in programming a robot to juggle, or keep the ball in the air indefinitely using successive upwards strikes. [Mülling *et al.*, 2013] applied machine learning techniques in selecting strike movements. More recently, [Gao *et al.*, 2020] used reinforcement learning to train a virtual manipulator to return serves. Others have looked at the grips required for grasping paddles [Liu *et al.*, 2011]. One of the groups from last year's manipulation course created a juggling robot [Beveridge and Shubert, 2020]. While we're initially drawn to the idea of implementing our own juggling robot, we pivoted to focus on catching, which we believed was a unique and challenging task due to the level of precision required to exert continuous control on the movement of a free-moving ball in 3D space without gripping it. In Figure 1, we show how humans perform the task in two distinct phases: (1) bringing the ball to a stop with the paddle, and (2) stabilizing the ball on the paddle.

We distinguish between the act of catching a ball and the task of stabilizing a moving ball on a paddle. We define catching to be the act of bringing a ball into contact with a paddle's surface with no (or minimal) bouncing. We define stabilization as the task of maximizing the time of contact between a free-moving ball and a paddle's surface. A simple intuition for the mechanics of a catch is that a ball and paddle should have the same instantaneous velocity at the initial moment of contact. This allows the normal force between the paddle and the ball to grow smoothly rather than jerk and impart relative momentum to the ball. For stabilization, we sought a controller design that would maximize the likelihood of the ball staying at the center of the paddle.

We mirrored our system design on the same principle. We used a finite-state machine (FSM) approach to switch the mode of the controller based on whether the ball and paddle had come into initial contact. We justified this decision by reasoning about the range of possible trajectories for a ball before and after the initial contact, and we took inspiration from the way humans catch ping pong balls. Note that the iiwa's poses in Figure 2 mirror Cameron's poses in Figure 1. We constrained our simulation environment such that the ball must be falling into the iiwa's workspace. More specifically, we expected the ball's velocity vector to be primarily in the $-z$ direction when it came into initial contact with the paddle. Catching balls served horizontally from the same level as the iiwa is outside the scope of this project. If we assume that the first contact between the ball and the paddle either results in no bounce or a very minimal bounce, we know the ball and paddle must have a significantly smaller relative velocity. As such, our goal then changes to one where we want to manipulate the ball in the direction of a stable position on the paddle. The FSM approach naturally fell out of the distinction between the two kinematic regimes of the ball.

We recognized that the realism of our simulation would depend heavily on the physics of ping pong balls. Namely, ping pong balls are very light objects and their trajectories are significantly influenced by air resistance. Ball spin is also a major factor, producing pronounced curves and deviations from projectile motion. While contact between a ball and paddle is nearly elastic, the frictional forces between rubber surfaces and spinning plastic spheres are complex and strongly influence interactions during the stabilization phase. In the name of expedience and focus on the manipulation tasks at hand, we limited ourselves to simulating ping pong balls as perfect projectiles. This allowed us to use an open-loop design while computing paddle trajectories during the first mode of

[1]https://github.com/cameronwp/ping-pong-catcher

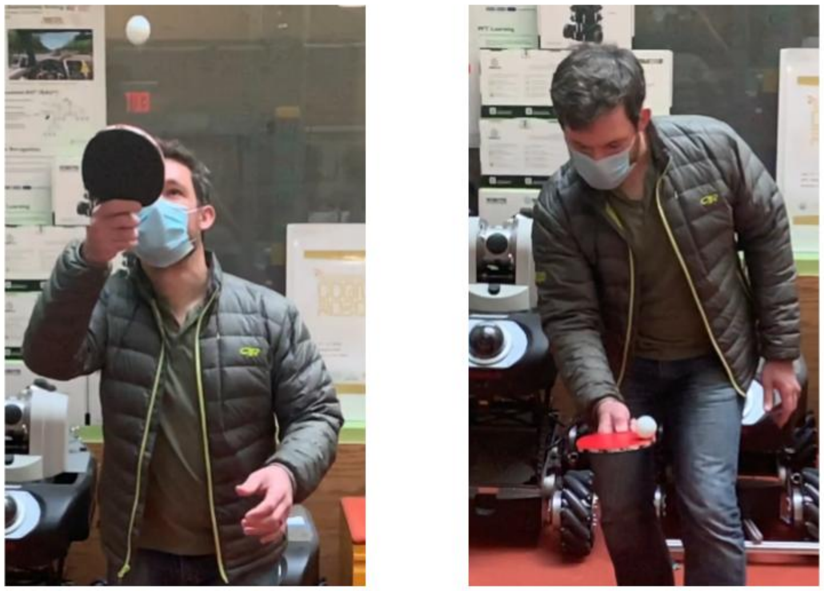

Fig. 1: Cameron demonstrating how to catch a ping pong ball with a paddle in real life. On the left, he tracks the ball and prepares to intercept it at the same velocity as it comes down into his effective workspace. On the right, the ball is in contact with the paddle, however, minute and rapid adjustments in the paddle's position and rotation are required to keep the ball in contact with the paddle surface.

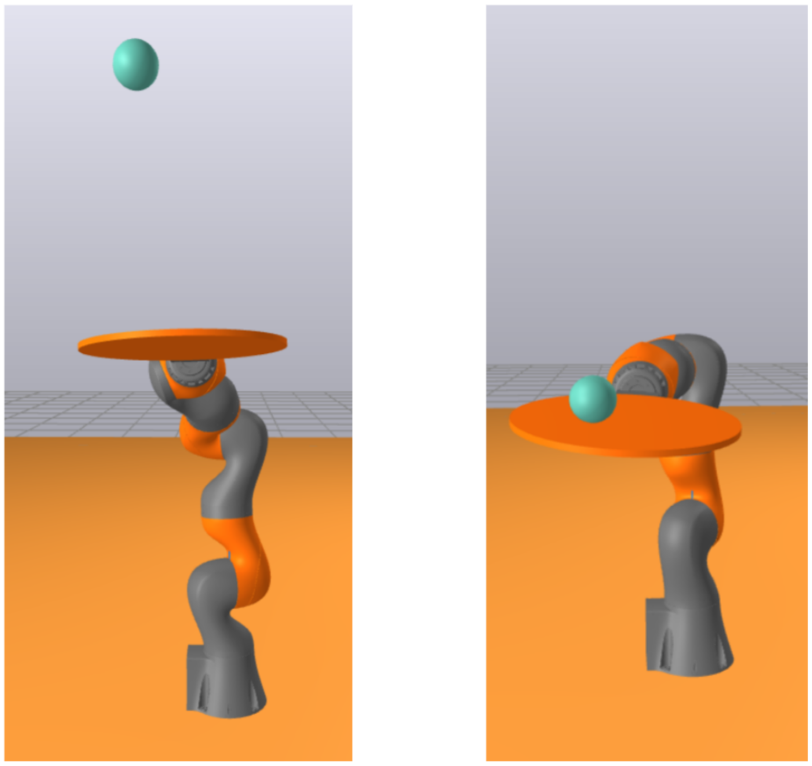

Fig. 2: On the left, the iiwa is in the catching state. Its pose reaching up to the ball resembles how Cameron prepared to make contact. On the right, the iiwa is in the stabilization state. It uses small translational and rotational adjustments to try to work the ball to the center of the paddle.

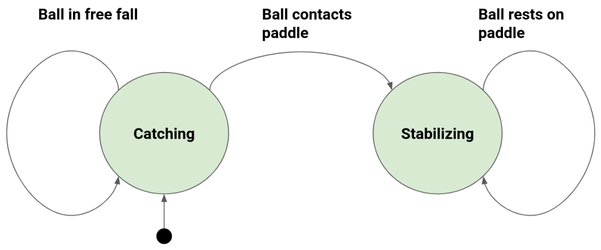

Fig. 3: `Catcher` controller FSM state diagram.

the FSM. For simulating contact dynamics, we chose to use the Hunt-Crossley model that Drake supports with low dissipation values that subjectively appeared to produce ping pong ball-like bounces. For contact forces, we used $\mu = 0$ to simulate a rolling sphere on the paddle surface. As such, the system we simulated is only subjectively similar to the true task of catching a ping pong ball, though we believe the techniques described below in Section II could be extended to successfully perform real-world maneuvers as well.

II. Approach

Our final approach centered around a controller with FSM state transitions, which we called the `Catcher` controller, which would output desired velocities.

Translational and rotational kinematics were calculated separately and all calculations were performed in the world frame. The controller used in our final system transitioned from catching to stabilization states when the ball and paddle made an initial contact. The behavior of both the translational and the rotational kinematics of the controller transitioned between states, as can be seen in Figure 3.

A. Catching State Translational Kinematics

For the first state of the FSM, we needed to compute a trajectory for the paddle whereby it and the ball would meet at some future position with the same velocity, which we visualize in Figure 4. To do so, we modeled the paddle as a second projectile in the scene and computed its initial velocity and acceleration vectors such that the future positions and velocities of the paddle and ball would coincide. We performed all of our calculations in the world frame for simplicity's sake.

Let $p_b[t]$ and $p_p[t]$ represent the position of the ball and paddle respectively at some time t. Likewise, let $\vec{v_b}[t]$, $\vec{v_p}[t]$, $\vec{a_b}$, and $\vec{a_p}$ represent the velocity and acceleration vectors of the ball and paddle (assuming constant acceleration).

To start, we picked an arbitrary z value within the iiwa's workspace to be the height at the time of the catch, $p_{bz}[t_{\text{catch}}]$. Subjectively, it seemed to make sense to set $p_{bz}[t_{\text{catch}}]$ near or slightly above the mid-height of the iiwa's workspace. At higher positions, the ball would be moving more slowly, but the iiwa would lose range of motion. Too low and the ball would be moving too fast and collisions with the floor would be a concern. We calculated t_{catch} by applying the quadratic equation to the ball's parabolic motion in the z direction,

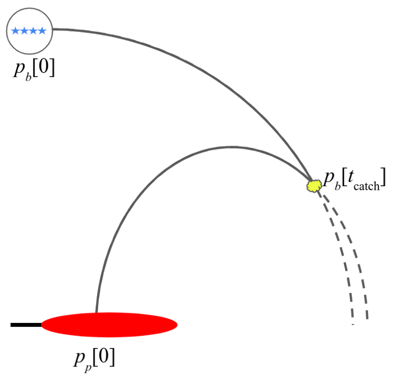

Fig. 4: A not-to-scale representation of the trajectory we computed to catch the ball. While both the ball and paddle have different starting positions, we essentially computed the parabolic motion required to bring the paddle to the ball with the same velocity. While the ball only accelerated in z by g, we allowed the paddle to accelerate in any direction to meet the ball.

much in the same way that [Rapp, 2011] modeled the time of successive bounces.

$$t_{\text{catch}} = \frac{1}{g}\left(v_{bz}[0] + \sqrt{v_{bz}^2[0] + 2a_{bz}(p_{bz}[0] - p_{bz}[t_{\text{catch}}])}\right) \tag{1}$$

Given t_{catch}, we could extrapolate a ball location, $p_b[t_{\text{catch}}]$, and ball velocity, $\vec{v_b}[t_{\text{catch}}]$, that we wanted the paddle to match. We found the initial paddle velocity and paddle acceleration, $\vec{v_p}[0]$ and $\vec{a_p}$, which fell out of the equations of motion for projectiles. Let $p_b[t_{\text{catch}}] = p_b[0] + \vec{v_b}[0]t_{\text{catch}} + \frac{1}{2}\vec{a_b}t_{\text{catch}}^2$. The equalities we needed to satisfy were,

$$p_b[t_{\text{catch}}] = p_p[0] + \vec{v_p}[0]t_{\text{catch}} + \frac{1}{2}\vec{a_p}t_{\text{catch}}^2 \tag{2}$$

$$\vec{v_b}[t_{\text{catch}}] = \vec{v_p}[0] + \vec{a_p}t_{\text{catch}} \tag{3}$$

We have two equations with two unknowns. Simplifying and rearranging, we find

$$\vec{v_p}[0] = \frac{2(p_b[t_{\text{catch}}] - p_p[0])}{t_{\text{catch}}} - \vec{v_b}[t_{\text{catch}}] \tag{4}$$

$$\vec{a_p} = \frac{\vec{v_b}[t_{\text{catch}}] - \vec{v_p}[0]}{t_{\text{catch}}} \tag{5}$$

Thus, at each timestep, t, our controller updated the desired translational velocity of the paddle to

$$\vec{v_p}[t] = \vec{v_p}[0] + \vec{a_p}t \tag{6}$$

We can see example behavior of `Catcher` in the catching state in Figure 5.

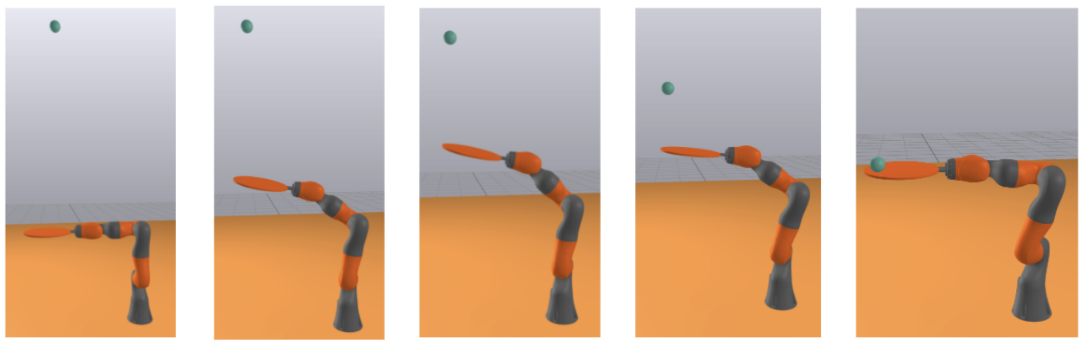

Fig. 5: A timelapse demonstrating `Catcher` in the catching state. Note how the iiwa reaches up towards the ball and then comes to a slightly lower z height after the catch due to momentum. In this example, $p_{bz}[t_{\text{catch}}]$ is at the iiwa's inital z height.

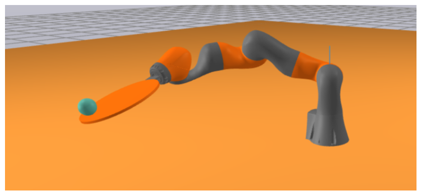

Fig. 6: An example of stabilization PD controller causing the iiwa reach out for a ball post-catch.

B. *Stabilization State Translational Kinematics*

We were no longer concerned with exactly matching the velocities of the ball and paddle in the stabilization state of the FSM. Instead, we aimed to keep the ball as close as possible to the center of the paddle while keeping paddle velocities low. Additionally, we wanted to bias the iiwa to return to its initial position to complete the task.

Let K_p and K_d represent the position and derivative gains respectively. A small K_0 gain, where $K_0 < K_p$, was used to return the iiwa to $p_p[0]$. We applied a modified version of a closed-loop PD controller design with the following formulation

$$\vec{v_p}[t] = K_p(p_b[t] - p_p[t]) + K_d(\vec{v_b}[t] - \vec{v_p}[t]) + K_0(\vec{p_p}[0] - \vec{p_p}[t]) \tag{7}$$

This PD controller is effective at following the ball, an example of which can be seen in Figure 6.

Note that p_p specifically refers to the point at the center of the disk of the paddle. p_b refers to the center of mass of the ball. Assuming the ball and paddle are solids and can not penetrate, p_b can never equal p_p. The minimum offset between p_b and p_p is the radius of the ball and half the thickness of the paddle. We omitted it in Equation 7, but we included this offset in the actual $p_b[t]-p_p[t]$ operations in our controller (including the rotational kinematics controller described in Section II-C.

C. *Rotational Kinematics*

Across both states of the FSM, we used a closed-loop design resembling a PD controller to control the rotational velocity of the paddle to encourage the ball to roll to its center. To do so, we took a naive approach wherein we scaled the rate of roll and pitch based on the offset of the ball from the paddle center, Δp, in the x and y directions. We constrained the rotational velocities in any direction to a maximum of $30\deg/s$, θ, under the assumption that higher angular velocities would lead to the ball being flicked off the paddle. Let $\omega_p[t]$ represent the angular velocity of the paddle at timestep t. Let r_p represent the radius of the paddle. We included an additional correction factor, $c = [1, -1, 0]$, to correct the direction of the pitch and remove the yaw.

$$\Delta p = (p_p[t] - p_b[t])/r_p$$
$$\omega_p[t] = K_p c\,(\theta \Delta p - w_p[t-1])$$

Qualitatively, it appeared that the orientation corrections required for successful catches were significantly smaller than the corrections required for stabilization. As such, we initialized the K_p of the rotational controller to small values while in the catching state, and then transitioned to higher gains when the controller entered stabilization state.

D. *System Design*

We built and simulated `Catcher` with a modified iiwa manipulation station in Drake, architecting the system as seen in Figure 7. Starting from the Perception component, we relied solely on the capability provided by `MultibodyPlant` to get body states directly in lieu of any real perception system. We passed said states directly to the `Catcher` controller described above, which computed paddle trajectories. `Catcher` consisted of a single `LeafSystem` and used conditional logic to transition between states in its `CalcOutput` callback method. A differential inverse kinematics controller computed joint trajectories and enforced joint velocity limits

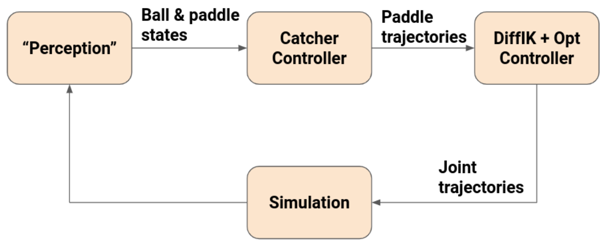

Fig. 7: A simplified view of our overall system design highlighting the components we focused on in this project.

using Drake's `MathematicalProgram` optimization front-end with SNOPT [Gill *et al.*, 2005]. We based our kinematics pipeline on the "Virtual Wall" exercise from Problem Set 3[2].

E. Controller Design Iterations

As we developed `Catcher`, we experimented with different kinematics pipelines. At the outset of the project, we intended to pre-compute piecewise trajectories by interpolating between positions to meet the ball in its projected path. Given the importance of matching the ball's velocity, we found this approach failed to give us enough explicit control to robustly catch. In the second iteration, we developed the kinematics as described in Section II-A. We initially passed the resulting end-effector spatial velocities denoted by V^P to a differential IK controller. The required joint velocities of the iiwa arm, denoted by v, were computed using the standard Moore-Penrose psuedo-inverse J^+ formulation. It is defined as,

$$v = [J^P(q)]^+ V^P$$

where J^P is the Jacobian matrix and q is the joint configuration of the iiwa arm. Upon execution, we realized that these commanded velocities were violating the joint velocity limits of the iiwa, leading to situations where the iiwa would end up colliding with itself. We suspected that this was happening since such controllers do not perform well around singularities i.e, when the minimum singular value of the Jacobian gets small, then some values in the inverse get very large thereby causing the erratic behavior.

To improve the computed joint velocities, we reformulated the problem to leverage a constraint-based approach. Our new version of `Catcher` solved the following problem,

$$\min_v |J^P(q)v - V^P|_2^2$$
$$\text{st. } v_{\text{min}} \leq v \leq v_{\text{max}}$$

where we set $v_{\text{min}} = -3$, $v_{\text{max}} = 3$ (we drew the 3 from the constants given to us in the same controller design in Problem Set 3). We utilized the SNOPT solver provided by Drake to solve this problem. As expected, this approach generated reasonable iiwa joint velocity commands for smooth motion without the erratic behavior we saw previously. We also noted this approach was much more robust in that it could produce qualitatively reasonable motions regardless of ball and paddle initial states.

Our final extension was to incorporate the additional orientation adjustments in `Catcher` as described in the Section II-C and passing desired ω_p to the differential IK system.

[2]https://manipulation.csail.mit.edu/pick.html#virtual_wall

III. Evaluation and Discussion

We wanted to evaluate the catching ability of our system and the choices we made in building it. To do so, we devised an experimental setup to compare the ability for different system designs to catch balls with different initial positions and velocities. We show the initial ball states in Figure 8. To quantify the difference between the performance of different models, we introduce a metric which we term as the stability quotient, φ, which is defined as the amount of the time that the ball was in contact with the paddle over the course of simulation. φ acts a quantitative metric that rewards system designs for the quality of the initial catch (eg. does the ball bounce during the catch? is the ball near the center of the paddle?) and the stability of the ball after the catch. Further, since the different controllers rely on compute intensive procedures, we also present an evaluation of these approaches in terms of how long it took them to complete the simulation. This metric allows us to perform a relative comparison of the performance of the systems in terms of φ.

To compute φ we detected contacts between multiple bodies in the plant and identified timesteps with contact between the paddle and ball. For the purposes of experiments, we ran the simulation for 3 seconds and hence by definition, $\varphi \in [0, 3]$ where higher values means better performance. We queried for contact every 0.05 seconds of wall time elapsed in the simulation. We chose to compute this metric in this fashion to avoid increasing the computation times of the methods. Further, for all the experiments we used a fixed set of gains and did not change them to suit the different scenarios in order to keep the evaluations as fair as possible.

The system designs we tested reflected the evolution of our `Catcher` design. The designs we tested were:

1) **Baseline** - piecewise trajectories in z
2) **DiffIK Translational Projectile Matching without SNOPT** - passing translational kinematics alone to a Pseudo-Inverse controller without optimization
3) **DiffIK Translational Projectile Matching with SNOPT** - passing translational kinematics alone to a Differential IK controller with optimization
4) **DiffIK Spatial Projectile Matching with SNOPT** - passing translational and rotational kinematics alone to a Differential IK controller with optimization

Table I presents the results of different models described in Section II. We can observe that the simple baseline model was able to perform quite well for multiple scenarios which involved initial velocities in at most one dimension. But as soon as we added initial velocity of the ball in more than one dimension, the controller had a hard time keeping up with

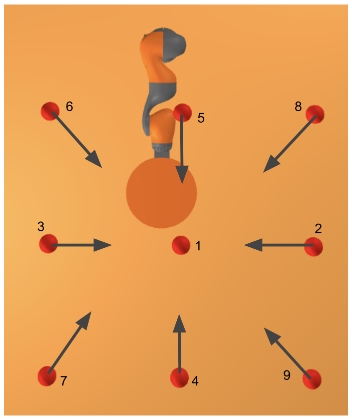

Fig. 8: The initial x, y positions and velocities of the balls we used in testing the systems. All balls started from the same initial z height above the iiwa.

the ball. This behavior is expected since this baseline method simply executes a fixed trajectory and does not account for the current state of the ball.

DiffIK Translational Projectile Matching without SNOPT was the worst performing model amongst all the approaches. We suspect that the reason for its under-performance was due to the controller commanding big changes in successive paddle velocities which meant that the Pseudo-Inverse controller was exporting joint velocities which did not respect the joint velocity limits of the Kuka iiwa arm. This observation motivated us to constrain the joint velocities of the Kuka iiwa arm so that its joint velocity limits are respected while the end-effector is able to keep up with the commanded velocities, which led to our next test case that uses SNOPT.

It can be observed that the first tests with **DiffIK Translational Projectile Matching with SNOPT**, produced significantly improved motions by a huge margin, hence demonstrating the effectiveness of a Differential IK using SNOPT-based approaches compared to the simple Pseudo-Inverse-based approach. Even compared to the baseline approach, this model fared well in most of the scenarios, particularly scenarios 7 and 8 where the baseline struggled. On the other hand, it did under-perform in scenario 2 which we suspect is due to poor position and derivative gains.

As expected, the final approach, **DiffIK Spatial Projectile Matching with SNOPT** performs well when compared to simple translational model without SNOPT. However, we did anticipate even better performance from this approach since we are executing corrective behavior to try to keep the ball at the center of paddle after initial contact is made. For some scenarios, particularly scenarios 1, 3 and 6, this approach did in fact outperform all the other approaches by big margins. Once again, we believe poorly tuned gains made the problem inherently harder for the controller to solve.

Next, we look at computation times, which we present in Table II. We hypothesize that there is a correlation between higher stability and higher computation times. In Table II, we see that the baseline and translational projectile matching without SNOPT approaches to be much faster than other alternatives that depend on SNOPT, which makes sense given the potential compute cost of running optimization. The baseline approach was the fastest since the controller just needed to execute a pre-computed trajectory to complete the task. As expected, the SNOPT based approaches took longer, which, when considered in the context of results shown by Table I, indicates that for better performance we have to give up some computational efficiency.

As can be seen across the scenarios in Table I, this task proved much more challenging than expected. While developing `Catcher` , we learned that we had to fine tune bespoke parameters for each set of ball and paddle initial states to make a successful catch and maximize contact.

For instance, we learned that the math and logic in Section II-A is overly simplified. This fact was apparent in observing that the paddle was sometimes a split-second too slow in reaching $p_b[t_{\text{catch}}]$ (the ball would pass just under the paddle). We attributed its lag to a shortcoming of the translational controller in Section II-A, namely that it relied on generating an instantaneous velocity at the start of its trajectory, $v_p[0]$, which could be quite large. Instantaneous velocities are impossible if we respect joint velocity limits. Further, we suspected that the measured / actual joint velocities of the iiwa would differ from the commanded joint velocities, leading to deviations from the desired path as well. To overcome this issue, we sometimes applied a scaling factor, s, to t_{catch} in Equations 4 and 5. We set $s \in [0, 1]$, and replaced t_{catch} with t'_{catch}, where $t'_{\text{catch}} = st_{\text{catch}}$.

Of course, we also found that the gains needed to be tweaked for each scenario. For instance, we found that higher relative x, y velocities required more aggressive gains on the rotational corrections. Catches with smaller bounces required less aggressive translational gains or we risked accidentally sliding an otherwise stable ball off the paddle.

The final area we tweaked was the catch height, $p_{bz}[t_{\text{catch}}]$. We had no guarantees of optimality with our choice. Through our experimentation, we found that different heights produced qualitatively varying catch performance.

IV. Conclusions

Our aim for this project was to impart similar level of dexterity that humans utilize to catch a ping pong ball on a hard paddle attached to 7-DOF Kuka iiwa manipulator. We iterated from very simple controllers to more complex ones that leverage optimization based routines to generate position and velocity commands on the iiwa arm. We compared system designs and found that a pipeline with Differential IK and optimized joint constraints showed the best performance, though at the expense of compute time. We had much more success in the first state of our FSM, catching the ball, than we did in the second state, stabilizing the ball on the paddle

Model/Scenario	1	2	3	4	5	6	7	8	9
Baseline	2.30	2.25	0.25	2.40	2.40	0.10	0.00	0.15	0.05
Translational Projectile Matching without SNOPT	2.35	0.00	0.00	2.35	0.00	0.00	0.00	0.00	0.00
Translational Projectile Matching with SNOPT	2.20	0.55	0.90	2.45	2.45	0.25	0.90	1.50	0.00
Spatial Projectile Matching with SNOPT	2.60	0.50	2.30	0.40	0.15	1.65	0.05	0.05	0.05

TABLE I: This table showcases how the different approaches performed with respect to φ where each entry is reported in seconds. Note that $\varphi \in [0, 3]$ and higher values means better performance.

Model/Scenario	1	2	3	4	5	6	7	8	9
Baseline	8.96	9.21	9.16	8.98	9.49	9.02	8.87	9.34	9.22
Translational Projectile Matching without SNOPT	13.68	14.37	14.00	13.84	12.95	13.64	14.25	14.19	14.15
Translational Projectile Matching with SNOPT	79.26	82.55	82.28	76.57	83.35	79.95	79.74	80.83	84.96
Spatial Projectile Matching with SNOPT	78.00	82.59	84.90	85.23	83.64	82.33	85.46	85.46	84.65

TABLE II: This table showcases how the different approaches performed in terms of computation time which is reported in seconds. Note that lower values mean better performance.

surface. The challenge of catching a ping pong ball on a paddle turned out to be significantly more difficult than we anticipated. Modeling contact and effecting a ball in contact with a frictionless surface were non-trivial problems that we did not anticipate.

Although the developed models were able to perform well on simpler tasks, we realize that there is still a lot of room for improvement. Namely, we wonder if more aspects of the problem could be performed in optimization. For instance, we could have framed the translational kinematics problem as an optimization problem. Likewise for the stability problem, we wonder if there might be an elegant representation in optimization. We also wonder if stabilizing a ball on a free-floating frictionless surface is a problem that lends itself well to Reinforcement Learning with a simple reward being how close the ball is to the center of the paddle. Or perhaps we could have tried a different approach by modeling the stabilization problem as an inverted pendulum. Lastly, there is a huge range of sampling based motion planners that could have been effective in either state. Another area for improvement is in detecting contact - we used a simple metric based on the z heights of the ball and paddle, but a Signed-Distance Function based approach may be more effective in edge cases, particularly if the ball is bouncing wildly around a paddle being held at an angle. We would have liked to try any and all of these improvements, however, learning how to effectively use Drake and simply run our simulation consumed more time than we expected.

Finally, this project was instrumental in our efforts to learn to be effective with Drake. We appreciate the hands on experience as we found a minor bug [3], spent hours reading Drake's documentation, and got a chance to build and simulate a unique manipulation task.

References

[Beveridge and Shubert, 2020] M. Beveridge and R. Shubert. Robot juggler. https://github.com/mattbev/robot-juggler, 2020.

[Gao *et al.*, 2020] Wenbo Gao, Laura Graesser, Krzysztof Choromanski, Xingyou Song, Nevena Lazic, Pannag Sanketi, Vikas Sindhwani, and Navdeep Jaitly. Robotic table tennis with model-free reinforcement learning, 2020.

[Gill *et al.*, 2005] Philip E. Gill, Walter Murray, and Michael A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Review*, 47(1):99–131, 2005.

[Liu *et al.*, 2011] Yuwang Liu, Yuquan Leng, and Weijia Zhou. Analysis of multi-fingered grasp and manipulation of ping-pong racket. *Advanced Engineering Forum*, 2-3, 12 2011.

[Mülling *et al.*, 2013] Katharina Mülling, Jens Kober, Oliver Kroemer, and Jan Peters. Learning to select and generalize striking movements in robot table tennis. *The International Journal of Robotics Research*, 32(3):263–279, 2013.

[Rapp, 2011] Holger H. Rapp. A ping-pong ball catching and juggling robot: A real-time framework for vision guided acting of an industrial robot arm. In *The 5th International Conference on Automation, Robotics and Applications*, pages 430–435, 2011.

[Tedrake and the Drake Development Team, 2019] Russ Tedrake and the Drake Development Team. Drake: Model-based design and verification for robotics, 2019.

[3]https://github.com/RobotLocomotion/drake/issues/16116