Real-Time Model Predictive Control for Shot Aiming in a Physical Pinball Machine

Zachariah E. Fuchs Pavan Saranguhewa Michael Ikuru

***Abstract*—We examine the shot aiming problem in the context of a physical pinball machine. A parametric switched mode system model is constructed and parameter tuning techniques are presented. A model predictive controller is developed that leverages the tuned model, and experimental results using a physical pinball testbed are examined. The developed controller successfully aimed shots with sufficient precision to satisfy most shot constraints within real-world, commercial pinball machines and provides a good baseline level of performance to use as a point of comparison for future aiming controllers.**

I. Introduction

Playing pinball machines is a remarkably complex task which requires a great deal of skill and finesse despite the deceptively simple input of just pressing a right or left button. The physical pinball system is governed by highly complex and nonlinear dynamics as the ball is constantly ricocheting off of different targets, ramps, or bumpers (some of which are active elements). Furthermore, the scoring often has complex rules and many different possible states. Therefore, the points a particular shot produces are highly variable and depend on the sequence of previous shots. However, the actual control input is very simple and just represents a left or right flipper activation.

Despite this complexity, there are several fundamental flipper skills that are universally useful, such as the dead flip, live catch, drop catch, and post pass. Also, the general method of aiming for particular shots on the playfield is similar across all machines. However, the exact way you perform the different flipper techniques is slightly different for each machine because the geometry of the flippers is not standard and can vary considerably from game to game. Additionally, every game has a different strategy for which sequence of shots and modes to activate in order to maximize your expected score. These characteristics make pinball a great platform to investigate the use of machine learning and control theory techniques to generate adaptable artificial intelligence strategies to play games.

There has been previous work exploring the development of automated pinball playing strategies. Much of the previous work was done in simulation using reinforcement learning techniques to learn black box flipper strategies [1]. Although these techniques demonstrate the power of reinforcement learning methods, like many other ML based game playing AI [2], [3], [4], they often require extensive amounts of training data or test runs. This can become problematic when attempting to train a controller on a real machine where each training iteration may take several seconds to several minutes. Even after the controllers are trained, it can be difficult to gain insight into the underlying system or control behaviors. There has been work on training a controller on a real machines as well [5]. These experiments focused primarily on how long the ball was kept from draining. Although this is a good heuristic metric for successful pinball playing, it doesn't capture the controller's ability to play strategically or make particular shots. This work did contain some preliminary analysis of shot aiming and achieved approximately a 50% success rate. However, these were for fixed shots on a specific playfield and only considered cradled initial conditions.

Skilled pinball play involves several layers of strategy, planning, and control. The highest level of control uses a shot map that may either be predefined or learned while playing in conjunction with a rule-state estimator to plan a sequence of shots to maximize the expected score. The expected score is calculated using the probability of successfully making a sequence of shots and how those shots progress through the rule-states for that particular machine. In order to successfully hit the desired shots, a flipper mode control selects a sequence of pinball control techniques to gain control of the pinball and move it to the desired flipper for the desired shot.

Pinball shot planning is similar to the development of shot strategies in the Angry Birds video game [6], [7] in that the selection of a launch trajectory triggers a sequence of interactions that are dependent upon the nonlinear interactions of different elements within the game environment. However, the primary difference is that a player cannot directly choose a shot angle and power when playing pinball. Instead the player must choose the correct moment to activate flipper to launch the ball along the desired launch trajectory as the pinball approaches the flipper. The resulting launch trajectory is dependent on a number of factors that are not directly controlled by the player such as the incoming ball trajectory and flipper geometry. Therefore, the player (or an automated controller) must determine the optimal activation time based on the given ball state and the desired launch angle. It is therefore necessary to develop a shot aiming controller before for any other higher-level strategic optimization can be achieved. If a controller is unable to aim at desired shots, it will be very difficult to efficiently explore the shot space and rule-set of the game.

This paper specifically focuses on the development of a Shot Aiming Controller (SAC), which represents a low-level flipper control mode that would be activated by a flipper-mode

Zachariah Fuchs, Pavan Saranguhewa, and Michael Ikuru are with the Department of Electrical Engineering and Computer Science, University of Cincinnati, OH

978-1-6654-3886-5/21/ $31.00 ©2021 IEEE

selection controller. The SAC is developed using traditional model estimation and model predictive control methods. In this framework, the motion of the pinball is modeled as a switch mode system using three different dynamic models corresponding to three different phases of the shooting scenario. The controller aims the ball towards a target by selecting when to transition between the phases of the game. Three parametric models are developed for each mode and the parameters are tuned using data collected by playing the machine. Using the tuned models, the controller can calculate the optimal moment to flip the pinball by projecting the current state estimate forward through the system models. By utilizing traditional system modeling and control techniques, the controller can exploit system and dynamic information and greatly reduce the amount of required training data or test runs. Additionally, the trained controller and developed system models can be further analyzed to gain insight into the general behavior of the system and control strategies.

We begin by describing the rolling shot scenario in Section II. The experimental setup and parameter estimation methods are described in Section III. The shot aiming controller is developed in Section IV, and its performance is evaluated in Section V. Concluding remarks and a brief discussion of future work is provided in Section VI.

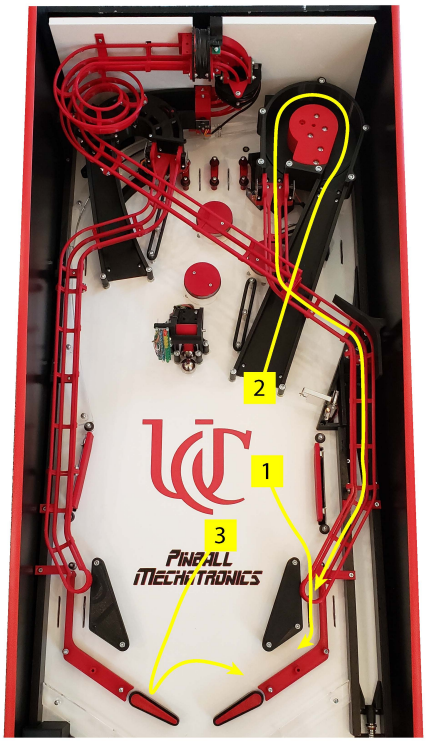


Fig. 1: Prelude Trajectories for the Rolling Shot Scenario [8]

II. Rolling Shot Scenario (RSS) System Model

This paper focuses on the shot aiming controller for the specific scenario in which the ball is rolling down the ball guide and flipper. We will refer to this as the *rolling shot scenario (RSS)*. Although only considering the RSS may seem like a restrictive assumption, the RSS is a very common scenario while playing real-world pinball games, and represents the ideal shooting situation for controlled play. Rarely are accurate shots made while the ball is rolling freely around the playfield. Instead, other flipper techniques are used to gain control of the ball and move the ball into the RSS. These other techniques are modeled using separate control modes within the overall pinball control framework and will not be discussed in this paper. Generally, there are three primary events that lead to RSS scenario, which are illustrated in Figure 1:

1) The ball enters an inlane and rolls down the lane guide to the flipper.
2) The ball returns along a rail after making a ramp shot, is dropped into the inlane, and then rolls down to the flipper.
3) Other flipper techniques are utilized to capture and hold the ball with an activated flipper. When the flipper is released, the ball rolls down the flipper.

The RSS can be divided into three distinct phases as shown in Figure 2. In the first phase, referred to as the *rolling phase*, the pinball rolls down the ball guide and onto the flipper. The second phase, referred to as the *flip phase*, begins when the player activates the flipper causing it to rotate and accelerate the ball towards the top of the playfield. The second phase ends and the third phase, referred to as the *launch phase*, begins when the flipper reaches the end of it's stroke, at which point the flipper stops and the ball rolls across the pinball playfield until striking a target or being diverted by other playfield elements such as ball guides or ramps.

This three stage model can be represented as a switched dynamical system, in which a distinct set of differential equations governs the behavior of the ball within each phase. We define the moment when the system switches from the rolling phase to the flip phase as the flip time t_F, and we define the moment the system switches from the flip phase to the launch phase as the launch time t_L. The player controls the transition from the rolling phase to flip phase by activating the flipper. The state of the ball at t_F provides the initial condition for the flip and eventual launch phase, which uniquely determines the path of the ball. Therefore, the SAC aims the ball by activating the flipper at the appropriate moment (corresponding to the state of the ball) to strategically transition between phases.

If we have accurate dynamic models for all three stages, model predictive control methods can be utilized to calculate the optimal moment to activate the flipper to shoot the ball towards a desired target. In practice, these models vary from machine to machine, which causes every pinball machine to play slightly differently. There is a wide variety of lower playfield configurations with different ball guides and flipper geometries. Additionally, there are different types of flipper mechanisms as well as coil strengths that can change the launch characteristics. Pinball machines can also be setup with different playfield pitches. Lastly, even if two machines were identical when first configured, mechanical wear over the life of the machine and even coil heating during a single game can cause the machines' dynamics to diverge. Therefore, it is necessary to develop methods that can adaptively tune the underlying system models using data collected while playing

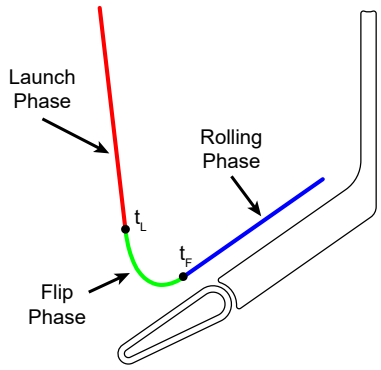


Fig. 2: Rolling Shot Scenario (RSS) Trajectory Phases

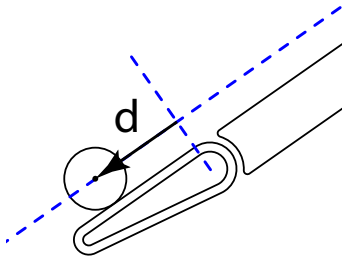


Fig. 3: Rolling Phase Coordinate System

the particular game of interest, and then use these tuned models within the control framework to aim shots.

In the following subsections, we develop parametric dynamic models for each of the RSS phases, which will be later tuned to match a particular machine using collected data.

A. Rolling Phase

In the Rolling Phase, we utilize a relative coordinate system to describe the position of the ball as it rolls down the guide and flipper. Although the ball is moving about a two-dimensional playfield, the ball guide restricts its motion along a line. Therefore, the state of the ball is represented using a signed distance coordinate, d, and speed coordinate, $\dot{d}$. The complete state of the ball is then defined as $\mathbf{d} := (d, \dot{d})^T$. The distance is measured along the rolling path and the origin is placed at the intersection of the rolling path and a line through the flipper pivot point which is perpendicular to the top edge of the flipper. An illustration of the rolling coordinate system is shown in Figure 3. With this coordinate system, we can represent the scenario as a ball rolling down a frictionless inclined ramp whose motion is modeled by the dynamics: $\ddot{d} = \gamma_d$, where γ_d is an acceleration term caused by gravity. The precise value of γ_d is a function of playfield pitch (inclination), ball guide geometry relative to the playfield, and the mass and rotational inertia of the ball. As mentioned before, the pitch and ball guide geometry are machine specific. However, they are constant, which allows the combination of these unknown system characteristics into the single unknown parameter γ_d.

Given an initial state $\mathbf{d}_0 := (d_0, \dot{d}_0)$ and defining the initial time $t_0 = 0$, the dynamics can be integrated forward in time to develop time dependent functions for both position and speed:

$$d(t; \mathbf{d}_0, \gamma) = \tfrac{\gamma}{2}t^2 + \dot{d}_0 t + d_{10} \quad (1)$$

$$\dot{d}(t; \mathbf{d}_0, \gamma) = \gamma t + \dot{d}_0. \quad (2)$$

We define the moment the flippers are activated as t_F and the corresponding state of the ball as $\mathbf{d}(t_F) = (d_F, \dot{d}_F)$.

B. Launch Phase

We will momentarily skip over discussion of the flip phase, and first develop the motion model for the launch phase. In this phase, the ball is able to roll freely about the playfield until it strikes an object. Therefore, the ball moves in both the x and y directions and we use a Cartesian coordinate system to describe the position and velocity of the ball $\mathbf{x} = (x, \dot{x}, y, \dot{y})$. The dynamics are modeled as

$$\ddot{x} = \gamma_x \quad \text{and} \quad \ddot{y} = \gamma_y, \quad (3)$$

where γ_x and γ_y are the gravitational acceleration terms in the x and y directions, which are dependent on the playfield pitch and ball characteristics. Given initial conditions $\mathbf{x}_L := (x_L, \dot{x}_L, y_L, \dot{y}_L)$, the dynamics are integrated forward in time to generate time dependent trajectories:

$$x(t) = \tfrac{\gamma_x}{2}t^2 + \dot{x}_L t + x_L \quad (4)$$

$$\dot{x}(t) = \gamma_x t + \dot{x}_L \quad (5)$$

$$y(t) = \tfrac{\gamma_y}{2}t^2 + \dot{y}_L t + y_L \quad (6)$$

$$\dot{y}(t) = \gamma_y t + \dot{y}_L. \quad (7)$$

When a pinball machine is correctly setup, the playfield should be leveled from left to right. Therefore, there should be minimal to no acceleration in the x-direction: $\gamma_x \approx 0$. Also, the ball is typically launched at high speed in the mostly y-direction, $\dot{y}_L \gg \dot{x}_L$, because there is limited range of motion in the x-direction due to the geometry of the playfield. Since the majority of the initial speed is in the y-direction and the resulting trajectory is relatively short in duration before it hits an object on the playfield, the initial velocity typically overshadows the acceleration term within the y-component of the state equation models: $|\dot{y}_L t| \gg |\frac{\gamma_y}{2}t^2|$. Under these assumptions, the system dynamics can be approximated using a constant velocity model:

$$x(t) \approx \dot{x}_L t + x_L \quad \text{and} \quad \dot{x}(t) \approx \dot{x}_L \quad (8)$$

$$y(t) \approx \dot{y}_L t + y_L \quad \text{and} \quad \dot{y}(t) \approx \dot{y}_L. \quad (9)$$

Using the constant velocity models, we create a linear function relating the x and y components of state:

$$\begin{aligned} g(x, y) &= x - \tfrac{\dot{x}_L}{\dot{y}_L}y + \dot{x}_L\tfrac{y_L}{\dot{y}_L} - x_L = 0 \\ &= x - \beta_1 y - \beta_0, \end{aligned} \quad (10)$$

where $\beta_1 := \frac{\dot{x}_L}{\dot{y}_L}$ and $\beta_0 := -\dot{x}_L\frac{y_L}{\dot{y}_L} + x_L$ are the shot trajectory characteristics that define the path of the ball after launch.

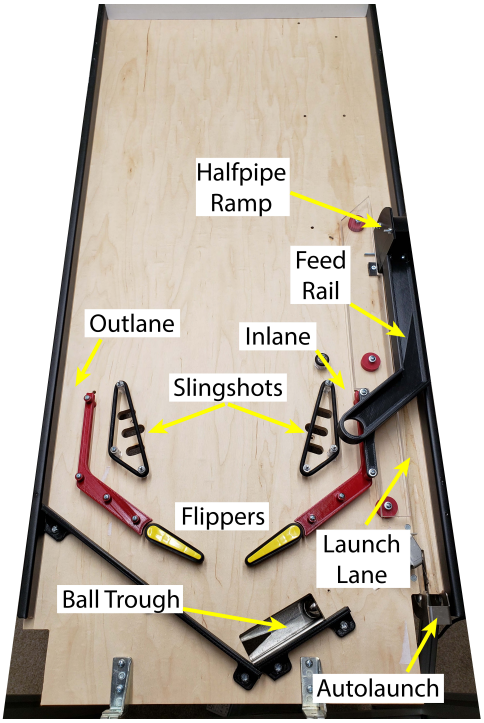


Fig. 4: Pinball Testbed

C. Flip Phase

The launch position (x_L, y_L) and launch velocity $(\dot{x}_L, \dot{y}_L)$ are determined by the nonlinear flip dynamics between t_F and t_L. These dynamics are highly nonlinear, but they are deterministic given a flip state $\mathbf{d}_F = (d_F, \dot{d}_F)$. It is not necessary to develop an explicit dynamic model for this phase because the details of the transitional trajectory bectween t_F to t_L are inconsequential for aiming the shot. Instead, it is sufficient to develop a mapping function that provides the β_0 and β_1 that describe the launch trajectory when the flipper is activated at $\mathbf{d}_F$. We define these functions as $\beta_0(\mathbf{d}_F) = \beta_0(d_F, \dot{d}_F)$ and $\beta_1(\mathbf{d}_F) = \beta_1(d_F, \dot{d}_F)$ and refer to them as the *flip functions*. Note that these mapping functions are dependent on both the position, d_F, as well as the speed, $\dot{d}_F$, at the moment of flipper activation. This is because the flipper actuation is not instantaneous, and the ball continues to roll along the flipper during the actuation. If the ball has a higher speed at the beginning of the flip, it will travel further towards the tip of the flipper before leaving, which creates a very different trajectory than if it is stationary or rolling slowly.

III. Experimental Setup and Parameter Estimation

The rolling dynamics described by (1) and (2), the flip mapping functions $\beta_0(\mathbf{d}_F)$ and $\beta_1(\mathbf{d}_F)$, and the launch trajectory (10), fully define the behavior of the RSS. However, the rolling acceleration parameter γ_d and flip functions $\beta_0(\mathbf{d}_F)$ and $\beta_1(\mathbf{d}_F)$ are unknown a priori for a given pinball machine. Additionally, the initial state at the beginning of the rolling phase $\mathbf{d}_0$ may not be known either. Therefore, these system parameters must be estimated online while playing the game. There are variety of techniques that could be implemented to estimate each of the these system components, but we discuss the specific estimation techniques used for this paper in the following sections.

A. Experimental Setup

In order to develop and test our pinball control architecture, we designed and constructed the pinball testbed shown in Figure 4. This playfield has a standard lower playfield consisting of two flippers, two inlanes, two outlanes, and two slingshots. The playfield also has a ball trough that captures drained balls and feeds them to a launch lane on the right side. This launch lane uses a solenoid controlled autolauncher to launch the ball onto the playfield. For the experiments in this paper, a halfpipe ramp was installed in the launch lane to load the ball into a feed rail, which drops the ball into the right inlane and initializes the RSS. Several different feed rails were designed so that the ball can be initialized at different positions and velocities in order to test the SAC under different conditions. The flippers, ball trough solenoid, and autolaunch solenoid are controlled by a custom designed microcontroller board that was developed for a pinball mechatronics course taught at the University of Cincinnati [8]. This microcontroller communicates via a USB interface with a control PC, which performs the image processing and SAC calculations. No physical targets or ramps were installed in the upper playfield so that virtual targets could be created within the software to test a variety of possible target positions. The ball trough, autolauncher, and feed rail allow the system to automatically and repeatably test the SAC in controlled scenarios in order to evaluate its performance. Although we plan to implement and evaluate the controller in real-world, commercial pinball machines in the future, it would be difficult to accurately control the shot conditions in a repeatable manner without extensive modifications to the machine. Also, the use of virtual targets allows for wider variety of shot configurations while a traditional pinball machine would have a fixed shot layout.

A camera was mounted above the playfield to track the ball in real time. For this paper, we utilized a FLIR Blackfly S BFS-UE-32S4C color camera which has a resolution of 2048x1536 at 120 frames per second. The aspect ratio of the playfield does not perfectly match the resulting image format, and as a result, the playfield does not fully occupy to the full image. For the lens used in this configuration, the playfield occupies a 1700x800 region of the camera image, which results in a physical resolution of approximately .015”/pixel or 67 pixels/inch. The ball is detected in each frame using a series of image processing techniques such as lens correction, background estimation and subtraction, and circle and contour detection. The detection algorithm runs at the full camera frame rate and produces a ball position measurement at each frame. Although the image processing steps are critical to the performance of the overall system, the details do not fit within the page limitations of this paper.

B. Rolling Phase State Estimation

After processing, each camera frame provides a position measurement of the ball. This measurement contains positional

measurement errors caused by relative motion between the camera and playfield, finite resolution of the camera, and inaccuracies within the image processing pipeline. The ball state within the rolling phase, $\mathbf{d}$, is described using both the ball position and speed, where a single frame only provides positional information. Applying direct difference methods between ball positions in subsequent frames could provide an estimate of speed, but difference methods are prone to amplifying measurement noise. Therefore, it is necessary to implement an estimation algorithm that simultaneously estimates both the ball position and speed. Ideally, the estimator should utilize the system information provided by the parametric dynamic model (1)-(2) to improve the state estimate if possible.

Kalman filtering techniques are commonly used in tracking scenarios [9], [10], and they provide a powerful toolset for combining information from state measurements (in this case position) with dynamic model information. In order to initialize the Kalman filter, a state estimate must be used as a starting point. For this application, an initial state estimate can be generated by performing a least square error fitting between a series of state measurements and the dynamic model (1). Once an initial state estimate is generated, the tracker can then switch to the Kalman filtering mode for the remainder of the rolling phase until the flipper is activated. The Kalman filter would then need to be reinitialized each time the ball entered the rolling phase.

Additionally, Kalman filtering methods require reasonably accurate noise models for the dynamics and sensor measurements in order to generate estimates with high confidence. Unfortunately, the noise generated by errors within the image processing pipeline is highly nonlinear with heavy tailed distributions, which correspond to occasional glitching errors. These glitching artifacts are rare, but they produce large measurement errors which can drive large changes in the Kalman filter estimate. If we were tracking the ball over a long duration, special preprocessing methods could be used to identify these glitches and minimize their effects on state estimation. However, we are only tracking the ball for a short duration before the flipper is activated and the state transistions to the next phase. Through experimentation it was found that simply updating the least squares dynamic model fit provided accurate estimations which were also robust to outlier measurements caused by the occasional image glitching. Since the rolling phase has a relatively short duration, the number of samples remains small and the least squares fit could be computed within the 120 FPS requirement. Therefore, we did not utilize a Kalman filter tracker for the results in this paper. However, we are looking at modifications for future work.

To estimate the state of the ball, we fit the polynomial parametric rolling phase dynamics (1) by performing a least squares error minimization. We define the ball trajectory estimate, $\hat{d}(t;\mathbf{z}_i)$, and the corresponding estimated speed, $\hat{\dot{d}}(t_i;\mathbf{z}_i)$, at time t_i corresponding to the ith image sample as

$$\hat{d}(t;\mathbf{z}_i) = z_2 t^2 + z_1 t + z_0 \tag{11}$$

$$\hat{\dot{d}}(t;\mathbf{z}_i) = 2z_2 t + z_1, \tag{12}$$

where the parameters $\mathbf{z}_i := \{z_2, z_1, z_0\}$ are found by minimizing the error function:

$$\mathbf{z}_i := \arg\min_{\mathbf{z}} \sum_{j=1}^{i} (d_j - \hat{d}_i(t_j;\mathbf{z}))^2. \tag{13}$$

If needed, the polynomial coefficients could be used to estimate the initial state and system parameter γ_d within the parameterized system model (1)-(2). However, the controller only needs an estimate of the state value, which is found by evaluating the trajectory estimate at the current sample time t_i. This state estimate will then be used within the SAC to determine when to activate the flipper.

C. Flip Transition Function Estimation

In order for the SAC to predict the launch trajectory when activating the flipper at state $\mathbf{d}$, it must utilize accurate mapping functions for $\beta_0(\mathbf{d})$ and $\beta_1(\mathbf{d})$. These functions could be estimated using a variety of model estimation techniques. After preliminary data analysis, it was found that using a third-order, two-variable polynomial model provided a sufficiently accurate fit for this application. In this model each of the flip functions is parameterized using coefficients $\mathbf{b}_j = \{b_{j0}, b_{j1}, \ldots, b_{j9}\}$, and the estimated flip function $\hat{\beta}_j(\mathbf{d}_F)$ is defined as

$$\begin{aligned}\hat{\beta}_j(\mathbf{d}_F) = {} & b_{19} d_F^3 + b_{18} \dot{d}_F^2 + b_{17} d_F^2 \dot{d}_F + b_{16} d_F \dot{d}_F^2 + b_{15} d_F^2 \\ & + b_{14} \dot{d}_F^2 + b_{13} d_F \dot{d}_F + b_{12} d_F + b_{11} \dot{d}_F + b_{10}.\end{aligned} \tag{14}$$

The flip model is trained using a collection of M test flips, where each test flip $\mathbf{f}_i := \left(d_F^{(i)}, \beta_0^{(i)}, \beta_1^{(i)}\right)$ consists of the flip state and the resulting measured launch parameters. For each flip, the resulting launch trajectory is measured by sampling the position along the trajectory then fitting the linear trajectory model (10) using Deming regression. Deming regression is used because the x and y measurement errors in the image processing are assumed to be independent. Three examples of a Deming regression fits are shown in Figure 9. The measured ball positions are indicated by the black markers and the Deming fit is indicated by the red line. We define the collection of training data as $\mathbf{S} := \{\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_M\}$. The coefficients are found by minimizing the error function

$$\mathbf{b}_j = \arg\min_{\mathbf{b}_j} \sum_{\mathbf{f}\in\mathbf{S}} \beta_j - \hat{\beta}_j(d_F;\mathbf{b}_j) \tag{15}$$

for each $j \in \{0, 1\}$. The collection of test flips can be collected online and updated while actively playing the game, or they could be collected in a preliminary training phase by flipping at several distances and speeds in order to sufficiently sample the state. The experiments discussed in this paper utilizes an initial training set of twenty-seven sample flips which consists of nine distances for each of the three entry scenarios described in Figure 1. Two example flip function estimates are shown in Figure 5 and Figure 6. These flip function estimates were generated using eighty-one test flips (three flips x nine distances x three initial positions). It can be seen that the third order models fit the collected data very closely and the beta values are very consistent.

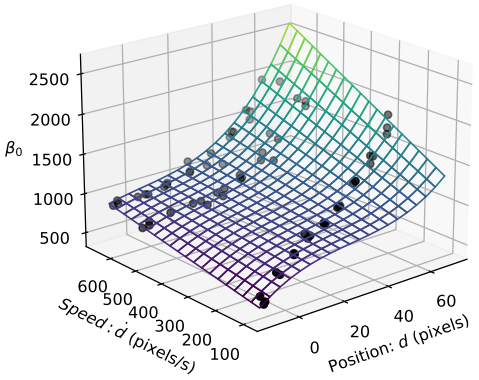


Fig. 5: Flip Function Parameter Fit: β_0

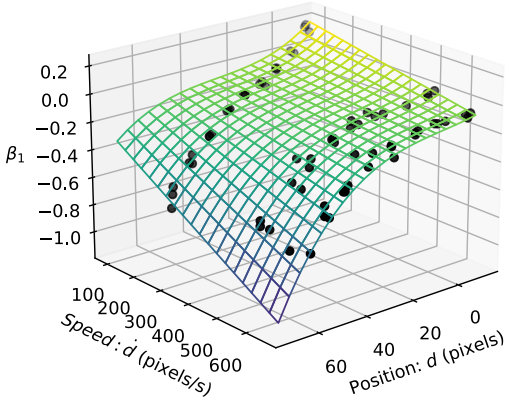


Fig. 6: Flip Function Parameter Fit: β_1

IV. The Shot Aiming Controller (SAC)

The estimated flip functions provide a mapping from ball state to launch trajectory, which can be used to predict the ball path when the flipper is activated. In order for a flip controller to successfully hit a targeted shot, it must generate a launch trajectory that passes through the target position $\mathbf{x}_T = (x_T, y_T)$:

$$g(x_T, y_T) = x_T - \beta_1 y_T - \beta_0 = 0. \tag{16}$$

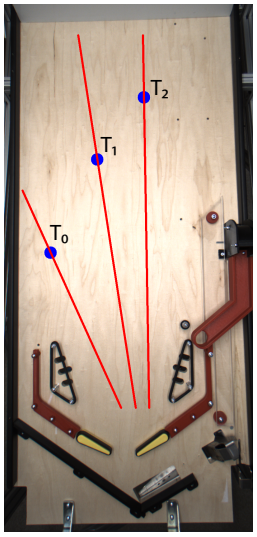
Using the third degree polynomial estimates for the launch functions (14) provides a two-dimensional cubic constraint:

$$\begin{aligned} g(\mathbf{d}_F; \hat{\beta}_0, \hat{\beta}_1, x_T, y_T) &= x_T - \hat{\beta}_1(\mathbf{d}_F) y_T - \hat{\beta}_0(\mathbf{d}_F) \\ &= k_9 d_F^3 + k_8 \dot{d}_F^3 + k_7 d_F^2 \dot{d}_F + k_6 d_F \dot{d}_F^2 + k_5 d_F^2 \\ &\quad + k_4 \dot{d}_F^2 + k_3 d_F \dot{d}_F + k_2 d_F + k_1 \dot{d}_F + k_0 \\ &= 0, \end{aligned} \tag{17}$$

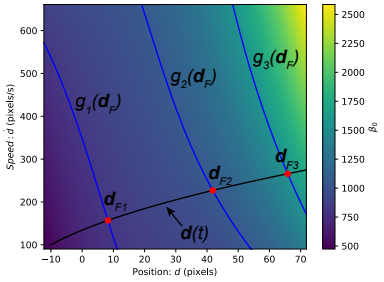
where

$$k_j = \begin{cases} -b_{10} y_T - b_{00} + x_T & j = 0 \\ -b_{1j} y_T - b_{0j} & j > 0 \end{cases}. \tag{18}$$

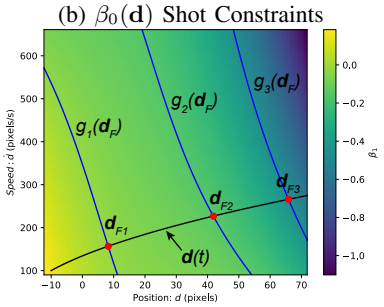
The shot constraint function (17) creates a curve within the flip space composed of admissible combinations of d_F and $\dot{d}_F$ that generate launch trajectories passing through the target position. These curves are not constant value contours of the $\hat{\beta}_0$ or $\hat{\beta}_1$ functions. As the ball moves down the flipper (change in d-component), the shot will start with a different offset (β_0 term) from the target and need to follow a different angle (β_1 term) towards the target position. Therefore, both the β_0 and β_1 coefficients within the launch trajectory will need to be adjusted. Three example targets are shown in Figure 7a. The targets' locations on the playfield are indicated by blue markers in Figure 7a and the example shot trajectories that hit the targets are indicated by red lines. Figure 7b and Figure 7c show the estimated flip functions, $\hat{\beta}_0(\mathbf{d})$ and $\hat{\beta}_1(\mathbf{d})$. In these figures, the collection of states that satisfy the target constraint functions, $g_i(d_F)$, corresponding to Target T_i are represented by the blue curves. The black curve labeled $\mathbf{d}(t)$ represents an example trajectory of the ball rolling down the flipper and satisfies the rolling phase dynamics defined by (1).



(a) Playfield Shots



(b) $\beta_0(\mathbf{d})$ Shot Constraints



(c) $\beta_1(\mathbf{d})$ Shot Constraints

Fig. 7: Shot Examples

The ball state will follow this curve moving from left to right as the ball rolls down the guide and flipper. Unfortunately, we cannot control d or $\dot{d}$ directly during the rolling phase of the RSS because the behavior of the system is governed by the state equations (1)-(2), which have no control input. Instead, the controller must continually monitor the ball state and activate the flipper when the state satisfies the shot constraint (17) for the desired target location. This occurs when the rolling phase trajectory intersects the shot constraint curve (17) within the flip space. These points are indicated by the red markers labeled $\mathbf{d}_i$ and correspond to the optimal flip state to hit Target T_i. It's interesting to note that the Target positions correspond to points in the playfield space and curves in the flip space, while launch trajectories correspond to lines in the playfield space and points in the flip space.

A particular rolling phase trajectory is dependent on the initial position, d_0, and initial velocity, $\dot{d}_0$, of the ball when it begins rolling down the ramp. If we were able to perfectly measure these initial conditions (as well as perfectly estimate all the system parameters and assume that there was no system noise), we could immediately calculate the optimal flip time t_F^*

corresponding to the intersection state $\mathbf{d}_F^* := \mathbf{d}(t_F^*; \mathbf{d}_0, \mathbf{x}_T)$ s.t. $g(\mathbf{d}_F^*; \hat{\beta}_0, \hat{\beta}_1, x_T, y_T)) = 0$. Unfortunately, our state measurements are not perfect due to noise and precision limitations of the camera and image processing algorithms. Additionally, we do not have infinite precision in timing the flipper activation because we are sampling the state at discrete times, which means the optimal flip time would occur between samples. Although we are currently investigating the use of external triggers to synchronize the camera sample rate and microcontroller timing, there is not currently a way for precise intersample timing control. Therefore, our current hardware configuration only allows for flipper activation when a frame is sampled. The discretization of time introduces a potential aiming error which we measure as the perpendicular signed distance between the launch trajectory and the target point. The expected error for a launch trajectory resulting from a flip at $\mathbf{d}_F$ is calculated as

$$e(\mathbf{d}_F) = \frac{x_T - \hat{\beta}_1(\mathbf{d}_F) y_T - \hat{\beta}_0(\mathbf{d}_F)}{\sqrt{\hat{\beta}_1^2(\mathbf{d}_F) + 1}}. \tag{19}$$

When $\mathbf{d}_F = \mathbf{d}_F^*$, the numerator goes to zero resulting in zero error.

Using the expected error function (19), we implement a model predictive control strategy that extrapolates the current state estimate one sample step ahead to predict shot constraint crossings. Let t_i be the current sample time, t_{i-1} be the previous sample time, and $t_{i+1} = t_i + \Delta t$ be the expected future sample time, where Δt is the sample period. The current state estimate, $\hat{\mathbf{d}}_i$, is found by evaluating the current polynomial trajectory approximation and derivative at t_i:

$$\hat{d}_i = \hat{d}(t_i, \mathbf{z}_i) = z_2 t_i^2 + z_1 t_i + z_0$$
$$\hat{\dot{d}}_i = \hat{\dot{d}}(t_i, \mathbf{z}_i) = 2 z_2 t_i + z_1$$

Similarly, the previous and extrapolated state estimates, $\hat{\mathbf{d}}_{i-1}$ and $\hat{\mathbf{d}}_{i+1}$, are found by evaluating the current polynomial fit at the corresponding times:

$$\hat{\mathbf{d}}_{i+1} = \hat{\mathbf{d}}(t_{i+1}, \mathbf{z}_i)$$
$$\hat{\mathbf{d}}_{i-1} = \hat{\mathbf{d}}(t_{i-1}, \mathbf{z}_i).$$

The estimated error at each time is found by evaluating the error function using the corresponding state estimates:

$$\hat{e}_j = e(\hat{\mathbf{d}}_j) \quad \forall j \in \{i-1, i, i+1\}. \tag{20}$$

Although it is difficult to prove analytically without closed form expressions for the flip functions, the error function is monotonic as the ball rolls down the flipper. Detecting a crossing of the flip constraint is equivalent to detecting a change in sign of the error function. Using the previous error, current error, and future error values, there are three possible scenarios that dictate whether or not the flipper should be activated. The flipper activation rules are outlined below.

1) $\text{sign}(e_i) \neq \text{sign}(e_{i-1})$: Flip constraint crossing in the previous time interval. Activate flipper.
2) $\text{sign}(e_{i+1}) \neq \text{sign}(e_i)$: Anticipated flip constraint crossing in the upcoming sample interval.
   a) $|e_{i+1}| >= |e_i|$: Future error is larger than current error. Activate flipper.
   b) $|e_{i+1}| < |e_i|$: Future error is smaller than current error. Do not flip.
3) $\text{sign}(e_{i+1}) = \text{sign}(e_i) = \text{sign}(e_{i-1})$: No crossing detected or anticipated. Do not flip.

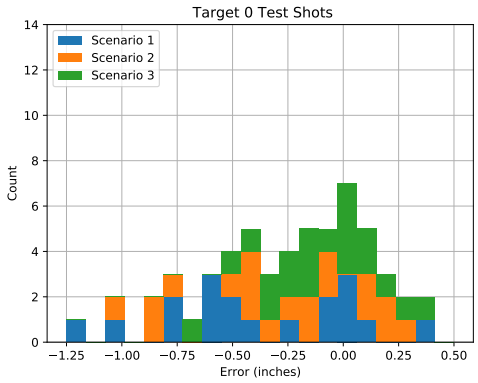
V. Experimental Results

The estimation techniques and SAC were tested using the pinball testbed described in Section III-A. To evaluate the performance of the system, the SAC was aimed at the three targets in Figure 7a multiple times. The resulting trajectory was recorded and the perpendicular error between the launch trajectory and virtual target position where measured. Three different initial conditions were used during these tests to emulate the three scenarios that lead to the rolling shot scenarios depicted in Figure 1. Each scenario was run twenty times for each Target resulting in sixty trials per target. Three examples of measured trajectories are shown in Figure 9. In this figure, the virtual target locations are indicated by the cyan circles. The expected launch trajectory is indicated with a green line. The measured ball positions are indicated by the black markers, and the resulting fitted launch trajectory is indicated by the red line.

A histogram of the measured error for each collection of the Target trials is shown in Figure 8. Although the error is nonzero, the performance is adequate for playing a real-world pinball machine. Targets within a real pinball machine are not singular points. Instead they have nominal widths which provides a margin of error. Most openings for ramp and orbit shots are between 2" and 2.5" wide. A pinball is 1.0625" wide and must pass inside this opening without striking the sides, which means that a ramp shot has a margin of error of approximately $\pm$.5 to $\pm$.75 inch. Standup targets usually range from .5 to 1.5 inches wide. Since the pinball can strike the target with a glancing blow, this provides a margin of error of $\pm$.75" to $\pm$1.25". Targets on the far right and left sides of the machines (similar to the position of Target 0) are either large standup targets or collections of multiple smaller targets arranged in a line. Typically these shots require far less precision to strike and can have margins of errors from $\pm$.75" all the way up to $\pm$2.5" for the largest targets like the side targets in Figure 1.

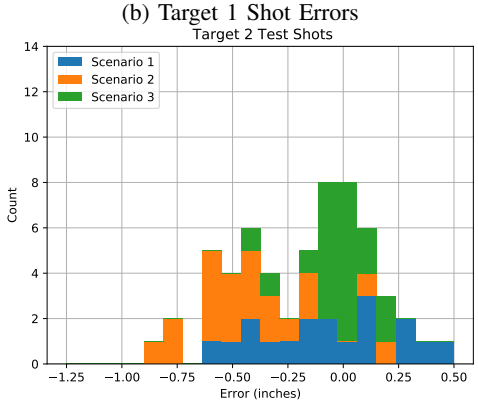
There is a clear difference in accuracy between the different test shot results in Figure 8 as well as the different initial condition Scenarios. Targets 1 and 2 represent common locations for ramp type shots. From Figure 8b, the SAC could make a 2" ramp located at Target position 1 shot consistently except for one outlier with an error of $\approx$1.25". It is believed that this bad shot was a result of OS induced delay in sending the fire command via the USB port. A ramp or target located at position 2 would also be made with a high probability.

The SAC had a much harder time hitting Target 0 with low error values as seen by the wider distribution of error values in Figure 8a. There are two contributing factors for this larger error. First, the SAC must wait until the ball is closer to the tip of the flipper to make these shots. This means the ball is moving faster, and there is larger increase in both the d and $\dot{d}$ state components between state samples. Therefore, there



(a) Target 0 Shot Errors

(b) Target 1 Shot Errors



(c) Target 2 Shot Errors

Fig. 8: Target Test Results

is a higher likelihood the flip constraint crossing will occur further away from samples which results in less fidelity in control. Second, any delay in transmitting the USB command to flip will result in a greater error because the ball is moving at a higher speed. Unfortunately, these are limitations of the current hardware system. We are currently developing a synchronization technique which allow much more precise interframe timing, which will greatly improve the shot timing precision. Despite the larger error, most types of shots in Target location 0, would generally be larger standup targets or more open orbit shots, which generally have larger margins of error. Therefore, even the current level of performance would be sufficient to play a real-world machine.

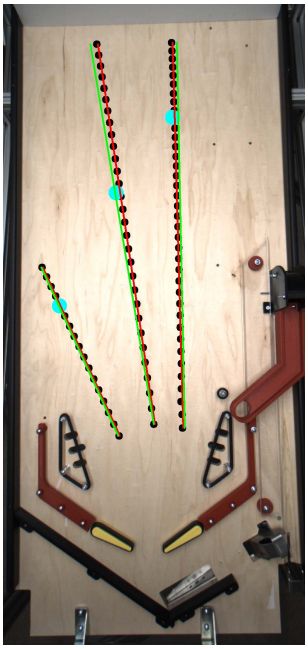


Fig. 9: Test Shot Examples

VI. Conclusion and Future Work

Overall, the SAC performed well and provides a good baseline level of performance to use as a point of comparison for future aiming controllers. The use of traditional estimation and control techniques provides valuable insight into the underlying system behavior and the key limitations on controller performance. Future work will explore other flip function estimation methods such as neural networks or support vector machines.

References

[1] N. Winstead, "Some explorations in reinforcement learning techniques applied to the problem of learning to play pinball," in *Proceedings of the AAAI-03 Workshop on Entertainment and AI/A-Life*, 1996, pp. 1–5.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, and G. Ostrovski, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[3] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[4] A. Kumar, K. Jani, and N. K. Sahu, "A comparative study of various artificial intelligence based agents for the game of angry birds with and without splitting," in *Journal of Physics: Conference Series*, vol. 1694, no. 1. IOP Publishing, 2020, p. 012001.

[5] A. Metcalf, "Pinball: High-speed real-time tracking and playing," 2011.

[6] J. Renz, X. Y. Ge, M. Stephenson, and P. Zhang, "AI meets Angry Birds," *Nature Machine Intelligence*, vol. 1, no. 7, p. 328, 2019.

[7] M. Stephenson, J. Renz, X. Ge, and P. Zhang, "The 2017 AIBIRDS Competition," pp. 1–11, 2018. [Online]. Available: http://arxiv.org/abs/1803.05156

[8] Z. Fuchs, "Pinball applications for engineering education," in *American Society of Engineering Education Annual Conference*. ASEE, 2021.

[9] S.-K. Weng, C.-M. Kuo, and S.-K. Tu, "Video object tracking using adaptive kalman filter," *Journal of Visual Communication and Image Representation*, vol. 17, no. 6, pp. 1190–1208, 2006.

[10] H. A. Patel and D. G. Thakore, "Moving object tracking using kalman filter," *International Journal of Computer Science and Mobile Computing*, vol. 2, no. 4, pp. 326–332, 2013.