Development of a Baseline Agent in Robot Air Hockey

Entwicklung eines Baselineagenten für Roboter-Airhockey
Master thesis by Verena Sieburger
Date of submission: May 12, 2022

1. Review: Puze Liu, M. Sc.
2. Review: Dr. Davide Tateo
3. Review: Prof. Jan Peters
Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Erklärung zur Abschlussarbeit gemäß §22 Abs. 7 und §23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Verena Sieburger, die vorliegende Masterarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß §23 Abs. 7 APB überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 12. Mai 2022

Verena Sieburger

Contents

Abstract

In this work, we introduce the development of a baseline agent in robot air hockey. The aim is to build up a system of two general-purpose robot arms to play air hockey against each other. In order to the high velocities of the dynamical system, the robot has to reach fast and reactive movements in the end effector. This task is even more challenging due to the precise limitation constraints in the joint and the task space of the robot. To solve the problem the state information of the puck is needed in advance. We mainly focus on precise state estimation and prediction of the puck. Using a composite model for the puck's movement and a Kalman filter for the state estimation, we manage to predict the movement precisely. We learn the physical parameter of the model with Bayesian optimization, where we vary the prediction horizon as well as the objective function. Furthermore, we estimate the variance of the noise of the components of the Kalman filter: the measurement and system model in order to get the filter to work optimally. This allows us to predict the position with an accuracy of less than one cm one second in advance. Based on this prediction, we evaluate different hitting movements of the robot using predefined metrics and implement an automatic testing routine. These results serve as a baseline and automatic evaluation of future work.

Zusammenfassung

In dieser Arbeit stellen wir die Entwicklung eines Basisagenten für Roboter Airhockey vor. Ziel ist es, ein System aus zwei universell einsetzbaren Roboterarmen aufzubauen, um Airhockey gegeneinander zu spielen. Um die hohen Geschwindigkeiten des dynamischen Systems zu erreichen, muss der Roboter schnelle und reaktive Bewegungen im Endeffektor ausführen. Diese Aufgabe ist aufgrund der präzisen Begrenzungsbedingungen sowohl im Gelenk als auch im Aufgabenraum des Roboters noch anspruchsvoller. Um das Problem zu lösen, wird die Zustandsinformation des Pucks im Voraus benötigt. Wir konzentrieren uns hauptsächlich auf die präzise Zustandsschätzung und -vorhersage des Pucks. Mit Hilfe von einem zusammengesetzten Bewegungsmodell des Pucks und einem Kalman Filter für die Zustandsschätzung gelingt es uns, die Bewegung präzise vorherzusagen. Wir lernen die physikalischen Parameter des Modells mit Bayes'scher Optimierung, wobei wir sowohl den Prognosehorizont als auch die Zielfunktion variieren. Darüber hinaus schätzen wir die Varianz des Fehlers der verschiedenen Komponenten des Kalman Filters: die Beobachtung und das Systemmodell, damit der Filter optimal arbeiten kann. Dadurch können wir die Position mit einer Genauigkeit von weniger als einem Zentimeter eine Sekunde im Voraus vorhersagen. Auf der Grundlage dieser Vorhersage bewerten wir verschiedene Schlagbewegungen des Roboters anhand vordefinierter Metriken und implementieren eine automatische Testroutine. Diese Ergebnisse dienen als Grundlage für die automatische Bewertung zukünftiger Arbeiten.

1 Introduction

Implementing robots for competitive games is a popular topic not only for entertainment yet for research. Tactics and planning define the interaction of the robots and enable a wide field for learning and optimization. The dynamical change of the game state of such systems and high uncertainties caused by the opponent or the system itself challenge the robot control. Furthermore, the dynamic gameplay requires fast responses and movements by the robot in which it has to satisfy constraints in the task space, like not complying with the game's rules.

We study the problem of implementing robot air hockey, which is a comparative table game. Two robots try to strike the cylindrical puck into the opponent's goal using a paddle. The table has small holes with flowing air to reduce the friction and allow high-velocity movements of up to 15 m/s in professional games by humans. Additionally, the spin of the puck makes the dynamical system more uncertain and enables the player to perform tricks for an exciting game. Based on these characteristics, it is crucial to have a fast prediction and response by the robot for a stable dynamical game. We divide the movement of the puck into three different parts. Firstly the linear part, where the puck flows on the table without any distraction. Secondarily the impact with the wall, where the direction of the puck's velocity changes. Thirdly the moment when the mallet transmits with a hit the momentum to the puck. These discontinuous movements depend on many different physical parameters like the coefficient of restitution of the puck and the table or friction and the preimpact state of the puck, i.e., the velocities. To hit the puck precisely is significant to understand all parts of the movement and predict it in advance in order to have enough time for the robot to plan the desired control response trajectory and perform it. The correct prediction of the puck is the essential precondition for optimal control. When the state prediction is wrong, even the best control strategy fails. Furthermore, the earlier we get an accurate state prediction the more time we have for the planning and strategy selection.

While playing air hockey, the robot has two aims: scoring goals and defending its own. In a complete game are more than two skills needed, like a safe status or different tactics to score or defend. The robot's control selection is often found on a two-layer system. One layer determines the game status based on observations and predictions, and another layer plans the trajectory and applies the control selected based on the game status.

In this work, we want to expand previous work on the robot air hockey [1]. The aim is to adapt the simulation system to the real-world robot system. Therefore, we perform two steps. First, we adjust the physical model and parameters to the real world using recorded trajectories of the puck. Second, we show an evaluation of the tactics in the simulation to ensure they are stable and performing correctly before applying them to the real-world system.

The first part mainly focuses on the puck prediction, as, especially to prevent goals, it is necessary to have a precise state estimation of the puck movement. Therefore, we use a Kalman filter and implement a complex dynamical system model capable of predicting the collisions of the puck. To determine the physical parameters values of the model, we use a black box system identification, Bayesian optimization, to get a proper parameter calibration for the system. The loss is based on recorded puck position data from the real-world air hockey table by an OptiTrack system. To deal with the recorded data of the camera, including lost frames and irregular time intervals, we also need to implement some preprocessing of the data. To get the best out of the Kalman filter, we evaluate the effect of different improvement strategies like noise estimation for the system and measurement noise. While adjusting the movement model improves the prediction precision significantly, the variance estimation of the system and the measurement model do not improve against the preassumptions we have of the variance.

The second part uses the new model with the optimized configurations to evaluate the different skills of the robot: Smash, Cut, Prepare and Repel. Therefore, we define success criteria for each skill and develop an automatic test setup. It enables us to reevaluate different approaches rapidly to accelerate the research process. We require the results for two reasons. First, we want to verify the safety of the robot's movement to apply it to the real-world robot. Second, we want to set a baseline for the robot, when we perform controls based on observation only without learning, to refer to in future work.

2 Related Work

A lot of different work has been done on playing air hockey and predicting the puck movement. Each focuses on another problem we already introduced: the tactics selection, the system identification, and the movement prediction.

In 1999, Bishop et al. developed an integrated system for robot air hockey [2]. This approach included state estimation with additional uncertainty of the state and prediction. The state prediction in the 2D system relies on translational position data and a simple bounce model using the reflection laws. Using this results in a high uncertainty for the wall impacts and forces the robot to play very defensively. On the other hand, the state prediction of the linear parts of the trajectory is very accurate and precise. To allow more aggressive playing the authors show the need to include the rotation information of the puck. In later work, the same group developed a detailed physical model for the collisions of an air hockey puck [3]. It is based on the Routh impact model derived from linear and angular momentum laws and based on the work of Wang and Mason [4]. This model is described in more detail in section 3 as we use it to improve our prediction. In contrast to the model of the complex dynamics, further investigations have been made by Park using a neuronal network for the prediction of the puck [5]. They use position and velocity as the input of the network, ignoring the rotation. For the training, they use data from linear and one bounce trajectories. The results once more show the high impact of the bounce and prone the influence of the puck's rotation on the prediction's accuracy. The group of Ghanzvini [6] showed good results using the physical model of Partridge et al. for a prediction, even though its input only is based on the less precise human hand movement before hitting the puck with the mallet. For this approach, they work with an off-the-shelf camera that could not capture the fast movement of the puck after the impulse of the mallet. Nevertheless, the model needs adjustment to the specific environment, as it depends on the material and other properties of the components involved. To derive the physical parameter of the system of Partridge, Alizadeh implemented an automatic calibration of the air hockey environment parameter like damping and restitution [7]. They derive the damping from a regression procedure that adds noise for each point on the

table to describe the difference in the airflow, as the amount of flow is not uniformly over the whole table distributed. In a second step, they determine the restitution using a binary search procedure. For this approach, they only use a few observations. The resulting noise of the parameter is tiny, so we will assume the parameters to be invariant for our work. Differently from this approach, we implement a Bayesian optimization routine for the automatic calibration of the air hockey robot environment.

The two-layer approach [8] is a study focusing more on the tactics of a robot than on the prediction of the puck. This work describes the development of an entertainment robot air hockey that performs tactics, which are several strikes together, to play successfully against humans. The skill layer selects a specific movement like smash or defend by two criteria: the tactics layer and the current game situation. For this, they use a horizontal two-link robotic arm. For the baseline of our robot, we rely on the skills and the proposed layer architecture but set the tactics layer to a fixed state machine. Updating the weights of the state machine by a second tactical layer is future work.

Another work that considers the whole robot air hockey system is by Wang who developed a system playing air hockey [9]. For the prediction, they only view the linear part of the puck, so they begin the state prediction after the last collision with the table and apply a uniform acceleration and a constant velocity model. The robot's skills are: attack, defend, and stand-by, and are selected by fuzzy control based on the distance of the puck's last impact with the wall to the defend line and the puck's velocity. This simple approach shows how to build up a visual-based control system like ours. Beyond, we develop a more complex system including a longer prediction horizon to perform more skills.

In contrast to the discussed works, which mainly use robots acting in the 2D space of the table, we use a general-propose robot to perform the action. On the one hand, this enables us to utilize more complex movements. On the other hand, our - thus bigger - task space requires additional restrictions. Beyond, we want to build a closed system including the puck prediction as well as system identification for a dynamic robot vs. robot air hockey play.

3 Foundations

3.1 Puck Movement Model

The model of the puck's movement on the table is a combination of two second order system models with a constant acceleration model. One is for the linear movement of the puck on a table without any external force disturbing the trajectory. The other one specifies the collision behavior of the puck with the rim of the table or the mallet, respectively. The puck state s of the system has six dimensions: two for the translation x and y, one for the rotation θ and for each the corresponding velocity value $\dot{x}, \dot{y}, \dot{\theta}$.

$$s = (x, y, \dot{x}, \dot{y}, \theta, \dot{\theta})$$

For the linear part (see eq. 3.1), the changes in position p (consisting of x, y) and orientation θ of the puck are defined by the current velocity. The velocity of the translation part is reduced by a constant friction μ_{surface} between the puck and the table surface and the damping d of the system. For the rotation velocity, the only reduction comes through the model's noise assumption, which is denoted with ϵ.

$$\begin{aligned} p_{t+1} &= p_t + \Delta T \dot{p}_t + \epsilon \\ \dot{p}_{t+1} &= \dot{p}_t - \Delta T \dot{p}_t \mu_{\text{surface}} d + \epsilon \\ \theta_{t+1} &= \theta_t + \Delta T \dot{\theta}_t + \epsilon \\ \dot{\theta}_{t+1} &= \dot{\theta}_t + \epsilon \end{aligned} \tag{3.1}$$

As a basis of the collision model, we rely on the work of Partridge [3], who derived a physical model for the contact of two rigid bodies in robot air hockey. The method is based on the Routh impact model and on the work of Wang [4] about rigid body collisions

with friction to get a mapping from preimpact velocity to post impact velocity of the puck. This allows a precise puck prediction and trajectory planning for the hitting. One of the main challenges is to include the effect of friction, restitution, and rotation of the puck.

The Routh method divides the normal impulse P into a compression impulse P_c and a restitution impulse P_r. The compression impulse is measured from the time the two objects contact until their relative velocity is zero. The restitution impulse is measured from the time the relative velocity is zero until the two objects do no longer contact. The restitution coefficient e of the rigid body can be computed as the ratio of the restitution and compression impulse and mainly depends on the material properties.

The impulse during the collision $P = (P_b, P_n)^T$ is modeled by the linear and angular momentum laws, which can be applied and reformulated to the puck-wall collision. The main simplification comes from the fixed table as one part of the collision. The coordinate system is transformed from the *base frame* (x, y) to the *impact frame* (b, n) such that the coordinate system's origin is the point of the first contact, the b-axis is tangent to the objects, and the n-axis is normal to the point of contact. That way, the impact forces align with the system. When the collision is not perfectly straight, there is friction μ_{rim} between the objects. If the friction is high or the velocity low enough, it prevents sliding in the collision. For each of the two scenarios, the impact's impulse can be computed like shown in equation 3.3 and 3.2 for the impact with sliding and without sliding in the collision, respectively. Here, r is the radius of the puck, m is the mass of the puck, e is the restitution of the table, μ_{rim} is the friction between the puck and the table's rim and s is the sign of the relative sliding velocity before the impact.

$$\begin{aligned} P_b &= -\frac{m}{3}(\dot{b}_0 + \dot{\theta}_0 \cdot r) \\ P_n &= -(1+e)m \cdot \dot{n}_0 \end{aligned} \tag{3.2}$$

$$\begin{aligned} P_b &= \mu_{\text{rim}} s(1+e)m \cdot \dot{n}_0 \\ P_n &= -(1+e)m \cdot \dot{n}_0 \end{aligned} \tag{3.3}$$

The resulting velocity $(\dot{b}, \dot{n}, \dot{\theta})$ in the *impact frame* is based on the impact in the collision and the velocity just before it, as shown in equation 3.4. Finally, to derive the final velocity the results need to be transformed into the $basefrane$.

$$
\begin{aligned}
\dot{b} &= \dot{b}_0 + \frac{P_b}{m} \\
\dot{n} &= \dot{n}_0 + \frac{P_n}{m} \\
\dot{\theta} &= \dot{\theta}_0 + 2\frac{P_b}{m \cdot r}
\end{aligned} \tag{3.4}
$$

3.2 Kalman Filter

The Kalman filter is a widely used filter for state estimation in uncertain environments [10]. Unlike other filters, it is only based on the last and the current measurement and can be applied simultaneously with the observation. For each time step, the filter holds two variables: The current state belief $\hat{x}$ and the covariance matrix P of the certainty of the state. The notation of $\hat{x}$ denotes that the state is an estimation of the real state x, which is unknown. The subscript $k|k-1$ consists of two parts, where the first part k defines the state that we are estimating, and the second part $k-1$ defines the state which is used as a basis for the estimation.

The Kalman filter consists of two phases, namely the prediction phase and the update phase. In the prediction phase, the state is propagated forward using a dynamic linear system model F and an additional control input u with the control matrix B to predict the next state (eq. 3.5). Besides, the certainty P of the predicted state is computed based on the certainty of the last step and the system model (eq. 3.6). Additionally, Gaussian white noise with covariance Q can be considered. As the current time step information is not included at this point, the state estimate is also called the a priori state estimate.

In the update phase, the current observation z of the state will be included in the estimation, so the state is also called the posteriori state estimate. Based on the difference of the estimation and the observation, mapped into the state space using an observation model H, the innovation y (eq. 3.7), as well as the innovation covariance S (eq. 3.8) is computed. The innovation depends on the covariance of the state and the measurement noise R, so the greater it is, the less we trust in the current state estimation. Based on the innovation and the covariance, the Kalman gain K is computed (eq. 3.9), which defines how much the current observation influences the state estimation $\hat{x}$ (eq. 3.10) and the belief P (eq. 3.11) in the updated state.

Prediction:

$$\hat{x}_{k|k-1} = F\hat{x}_{k-1|k-1} + B_k u_k \tag{3.5}$$

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k \tag{3.6}$$

Update:

$$y_k = z_k - H_k \hat{x}_{k|k-1} \tag{3.7}$$

$$S_K = H_k P_{k|k-1} H_k^T + R_k \tag{3.8}$$

$$K_k = P_{k|k-1} H_k^T S_k^{-1} \tag{3.9}$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k y_k \tag{3.10}$$

$$P_{k|k} = (I - K_k H_k) P_{k|k-1} \tag{3.11}$$

3.2.1 Extended Kalman Filter

As already mentioned, the forward model F in the Kalman filter is a linear system model. In many applications, the dynamics of the system are not linear. In the extended Kalman filter, the state transition model as well as the observation model H do not have to be linear but differentiable functions anymore. These functions $f(\cdot)$ and $h(\cdot)$ still have the same functionality as in the non extended version: $f(\cdot)$ is now the nonlinear state transformation and $h(\cdot)$ a nonlinear transformation of the predicted state to the measurement. As before, the system and observation model can contain noise which we do not include in the equations here.

For the state prediction the function can be applied directly: $\hat{x}_{k|k-1} = f(\hat{x}_{k-1|k-1}, u_k)$, but for the covariance prediction we need to use the partial derivative, i.e. the Jacobian:

$$F_k = \left.\frac{\partial f}{\partial x}\right|_{\hat{x}_{k-1|k-1}, u_k}$$

The Jabobian is updated every step to get a linearization about the current mean and covariance.

Respectively, for the observation model: $z_k = h(x_k)$ and

$$H_k = \left.\frac{\partial h}{\partial x}\right|_{\hat{x}_{k|k-1}}$$

3.2.2 Noise Estimation

As the Kalman filter balances between the system model belief and the measured observation, the performance highly depends on the correctness of the model and the correct assumptions about the noise of the system model and the measurement noise. If the measurement noise is too high, we do not trust the measurements and the Kalman Filter needs more time to adapt to changes. Setting the noise too small will take outliers into account so the estimation of the state will include the outliers as well. On the opposite, miss-guessing the system noise will make the system adapt very fast or slow to new observations. To improve the performance of the filter, we use variance estimation to approximate the noise better. Therefore, we start with noise assumptions inferred from the camera and model precision. Following, we describe the variance estimation for the two covariance matrices.

Measurment Noise

To calculate the noise of the measurement, we compute the covariance between the actual state x and the measured state z like shown in equation 3.12. In an uncertain world, we are not able to observe the actual state without error, so we have to estimate it. This results in using the estimation of the state $\hat{x}_{k|k}$ of the Kalman filter and assuming it to be the actual state. Based on this estimation, we also get an estimated covariance for the measurement. We get a mutual dependency from the estimated state to the Kalman filter, which we are trying to solve by using multiple iterations for convergence.

$$R = \frac{1}{n-1} \sum_{k=0}^{n} (h(z_k) - x_k)(h(z_k) - x_k)^T \tag{3.12}$$

System Nosie

For the system noise we compute the covariance between the state evolution by the system model $f(x_n)$ and the next (true) state x_{n+1}, see equation 3.13. As for the measurement noise we use the estimated state to approximate it.

$$Q = \frac{1}{n-1} \sum_{k=1}^{n} (f(x_n) - x_{n+1})(f(x_n) - x_{n+1})^T \tag{3.13}$$

3.3 Bayesian Optimization

Bayesian Optimization (BO) is a common black-box optimization algorithm [11]. The function to be optimized, called objective function, is unknown and modeled by a Gaussian Process over the function. The optimization is sample based, so we observe some samples of the function and have a Gaussian distribution over the unknown parts of the objective function. The uncertainty of the Gaussian Process is defined by the kernel function, which depends on how fast the function value changes with the function argument.

To optimize the function, initially some start samples are evaluated, that can be chosen randomly or based on assumptions of the objective function. Given the observed samples, the probabilities of the Gaussian Process can be computed and used to generate new samples to find the optimum. Therefore the following two policies are used:

- Exploitation: Select a point with a high probability to be the optimum
- Exploration: Select a point with a high uncertainty

The trade-off between the policies and the next sample point is defined by the acquisition function. Frequently used acquisition functions are Probability of Improvement and Expected Improvement.

Probability of Improvement

$$\text{MPI}(x) := \text{cdf}\left(\frac{(\mu - y^*) - \alpha}{\sigma(x)}\right)$$

Expected improvement

$$\text{EI}(x) := (\mu - y^* - \alpha) * \text{cdf}((\mu - y^* - \alpha)) * \text{pdf}((\mu - y^* - \alpha))$$

Algorithm 1 shows the pseudo code of BO, where f is the objective function, and g is the

acquisition function. In each iteration, multiple samples can be selected and evaluated.

Result: $x^* = arg\max f(x)$
observe f on initial values $x_{1..i}$;
$p(x)$ = Gaussian process of $GP(f, \mu, \Sigma)$;
while $n < N$ **do**
    $x_n = \max g(P, samples_n)$;
    $y_n = f(x_n)$;
    add (x_n, y_n) to samples;
    update P with all samples;
end

Algorithm 1: Bayesian Optimization

The Gaussian process model is based on two assumptions. First, that the function can be modeled by a multivariate Gaussian distribution with a fixed variance, and second, that if two function arguments x and x' are close to each other, then also the difference between the corresponding function values y and y' is small.

To get good results, it is necessary to have a good setup with the correct kernel function and hyperparameter as shown by the group of Snoek [12].

For the Bayesian optimization, we use HEBO [13], Heteroscedastic Evolutionary Bayesian Optimization. In comparison to standard BO algorithms, HEBO includes two adaptions:

- For the model, non-linear input and output transformations are used, so complex noise can be handled, as many problems are not homoscedasticity (so the variance of the noise is not equal on the whole function). Further, an additional likelihood is added to the Gaussian process mean so the exploration is amplified.
- Multi-objective functions are defined for the acquisition function, so the optimal point is found across all the acquisition functions and not by only one single function.

We assume our problem to be heteroscedastic, i.e., the noise of the states varies over the function, as the data we have is just a mini-batch of the problem. Furthermore, the loss function changes radically w.r.t. different values of the parameter.

4 Methodology

The key to a successful game of dynamical robot air hockey is the puck prediction, as the control has to react to future states. To predict the puck's movement precisely, we define a Kalman filter. To get optimal results of the state estimation, we need a good approximation model for the state process as well as noise variance assumptions.

In this section, we will describe the specific process we followed to get an improvement of the state prediction, as well as the testing routine of the robot hitting strategies based on the state prediction.

We will start with an overall introduction to the system architecture, to understand how the different components work together. Afterward, we will focus first on the prediction of the puck, describing the implementation of the puck's state process model and the corresponding optimizations based on the already introduced Bayesian optimization (see section 3.3) and noise covariance estimation (see section 3.2.2). These optimization algorithms are based on data sets described in the final section of the prediction.

In the last section of the chapter, we will describe the hitting strategies of the robots and define success characteristics to identify the baseline performance of the robot.

4.1 Build Architecture

To build our robot application, we use ROS[1] for managing the components as well as message passing. Each of the robots has its own node, including two main components. First, the puck tracker, the observer of the game state, which contains the Kalman filter implementation for state prediction and estimation. Second, the control, a two layer system, one for the selection of the hitting strategy (Tactics Processor) and one for the

[1] https://www.ros.org/

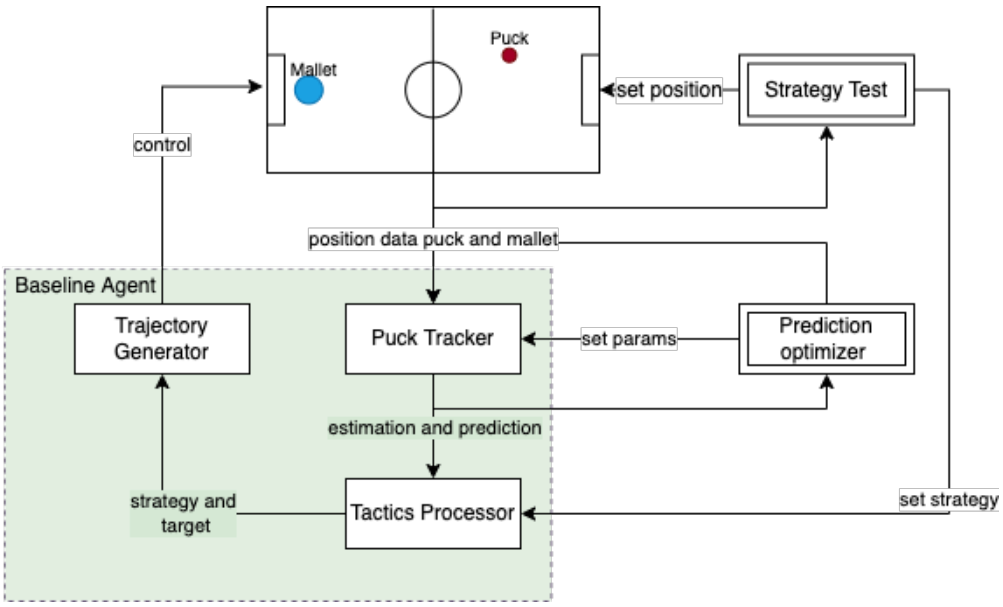


Figure 4.1: Simplified architecture of the baseline agent

control trajectory planning (Trajectory Generator). The strategy selection is based on a state machine depending on the puck's current and prediction state. Figure 4.1 shows a simplified version of one baseline agent, as well as the relations between the components. The entities on the right are the extensions we implemented for the prediction optimization or strategy testing, respectively, as described in the next section.

Additionally, one node is implemented as the referee, also observing the game status to detect goals as well as illegal states, i.e., when the puck is not on the table or the end effector moves into the table. Furthermore, the referee interacts with the system by resetting the puck after a goal or when the puck is not on the table anymore. It can also stop the game, e.g., if the robot gets into an illegal state.

This basic architecture is shared for simulation and real-world setup. While the puck position in the real-world setup is observed by the camera system, in the simulation the position information is provided by the Gazebo[2] simulator.

[2]http://gazebosim.org/

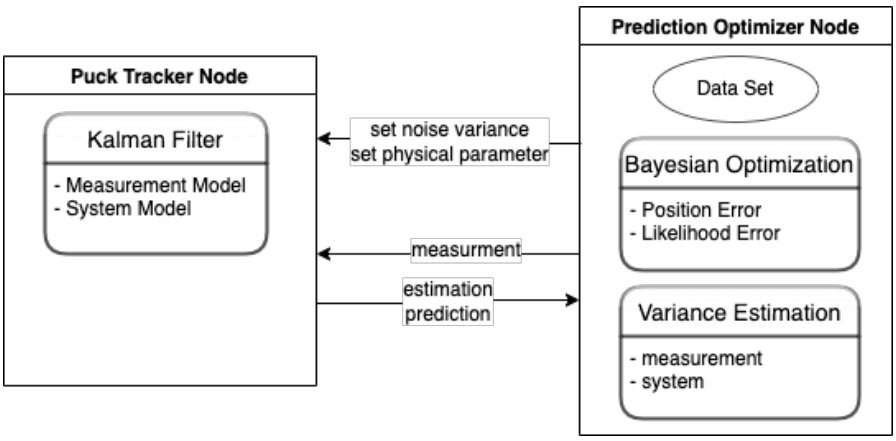


Figure 4.2: Shows the components of the nodes involved in the prediction optimization

4.1.1 Architecture Extension

For the optimization, we extended the system as already shown in figure 4.1 with two additional, temporal ROS nodes, one to perform different optimization and evaluation tools for the Kalman filter (Prediction Optimizer) and one for the strategy testing (Strategy Test). Figure 4.2 shows the components of the prediction optimizer and the puck tracker in more detail. Through a Server-Client structure, the prediction optimizer node replaces the observation of the real-word or simulation with the prerecorded data.

To reduce the complexity, we are not launching the whole baseline agent, but only the puck tracker used for the state estimation. Furthermore, this makes the new system independent of the actual time, as the filter only processes a step on request. Note that this forces us to make some of the transformations and preprocess structures, which are normally handled by the additional baseline agent code. The puck tracker node provides several services:

- update physical parameter of the system model
- update noise covariance of the system or measurement model
- perform Kalman filter step
- reset Kalman filter

The update services are needed to change the behavior of the Kalman filter. To evaluate the changes, the puck tracker provides the perform Kalman filter step service, which takes the measurement data and returns the estimation and covariance of the state as well as the prediction. If not specified, it returns the one-step-prediction.

The reset service is needed every time we are resetting the puck state, e.g., at the beginning of a new trajectory or for the tactics test.

In the following sections, we will focus on the puck tracker node components first and then on the prediction optimizer node components from top to bottom.

4.2 Puck Tracker Process Model

The system model of the Kalman filter is the model for the linear state transition. For the prediction and the filtering, this step is very important as it defines the expected behavior of the system.

The extended Kalman filter described in 3.2.1 additionally allows non linear updates based on the partial derivative in the current point. For linear functions, the extended and non-extended version do not differ. The main part of the puck's movement is linear except when colliding with the wall.

A simple concept is to use the Kalman filter without extension and just update the velocities when a collision is detected. This predicts $\hat{x}_{k|k-1}$ properly but the covariance prediction of $P_{k|k-1}$ is still based on the linear model F and might result in a wrong belief about the state estimation. This is especially a problem, as the collision is the critical part of the trajectory for the prediction and the most uncertain state.

To overcome this problem, we use the extended Kalman filter with the collision model 3.1 f and the partial derivative S of it. As the model differs whether sliding occurs in the impact or not, different Jacobians are needed. Moreover, the transition A into the collision model space orthogonal to the impact is needed, e.g., a rotation along the z-axis. The system model of the Kalman filter depends on the state, and the selection can be formulated as follow:

$$J \leftarrow \begin{cases} A_i^{-1} S_j A_i \quad i \in [0 \dots 3]\, j \in [0, 1], & \text{if } collision(x) \\ F & \text{otherwise} \end{cases}$$

For the states in the linear movement of the puck, the state transition is still the linear model F, but if the puck borders on the table's rim, the Jacobian is $A^{-1}SA$, where A_i is the transformation matrix for the affected wall i and S_j is the partial derivative of the collision model with, $j = 0$, or without, $j = 1$, sliding.

4.3 System Identification

To determine the physical parameters, we perform 50 Bayesian optimization iterations. The method is described in section 3.3. In each iteration the acquisition function selects 5 suggestions for the parameters configuration to evaluate. In total we specify four parameters: $\mu_{\text{surface}}, d, e$ and μ_{rim}. From the linear transition model (eq. 3.1), we need to identify μ_{surface} which is the friction between the table surface and the puck as well as the damping d of the system. For the collision transition model, (eq. 3.2 and eq. 3.3), we need to identify the coefficient of restitution e for the collision and the friction between the puck and the table's rim μ_{rim}.

In contrast to the system identification of the previous work [1] where the aim was to adjust the simulation environment, we want to adjust the forward system model and include the new parameter of the more complex collision model. To obtain this, we compare the prediction $\hat{x}_{k|t}$ we make at time t for the time k with $t < k$ and a prediction horizon of $k - t$ with the measured state z_k at time k. Nevertheless, we assume the values of the parameters to be similar to the hand tuned used before and use them as initial values and to derive the upper and lower bounds of the parameter as shown in table 4.1.

For the robot, we need at least a prediction horizon of 0.3 sec to respond to the state and plan and run the control trajectory. So the prediction should be precise for a prediction horizon between 36 and 180 steps, as this is the upper bound for the prediction horizon.

Notice that changing the prediction steps in the learning process can affect the outcomes, as for a small prediction horizon the prediction does not include impacts with the rim, and therefore, the parameters of the collision do not affect the result. On the other hand, having a high prediction horizon increases the distance of the compared data points and fewer comparisons. We therefore learn with prediction horizons of 60, 120, and 180 steps and validate the learned parameters against the two others.

	lower bound	upper bound
friction surface	0	0.5
friction rim	0	0.6
restitution	0.5	1
damping	0.2	0.8

Table 4.1: Lower and upper bound for the physical parameters

4.3.1 Objective Function

For the performance of the Bayesian optimization, the choice of the objective function has a huge impact. Following, we describe the two different functions we used to optimize the physical parameters of the Kalman filter system model.

- The first metric is the mean squared error of the position. The difference between the predicted and measured state is added up for each point. We begin with the first predicted step $\hat{x}_0$ with a prediction step size of j which is compared to the actual value x_j, so we lose $j-1$ points for the comparison.

$$L(x, \hat{x}) = \frac{1}{N} \sum_{i}^{N} \hat{x}_i - x_i$$

- The second metric is a probabilistic-based loss function. In contrast to the method before, not every discrepancy is weighted the same. For this method we sum up the probability of the actual value x_{i+j} given by the normal distribution $\mathcal{N}(\mu, \Sigma)$ with the prediction $\hat{x}_i$ as mean and the covariance of the prediction P_i as variance.

$$L(x, \hat{x}) = \sum_{i}^{N} \ln(p(x|\mathcal{N}(\hat{x}, P)))$$

4.4 Noise Estimation

For the optimal state estimation of the Kalman filter, the variance of the system Q and the measurement R needs to be set accordingly. We implement the formula 3.12 and

3.13 without correlation terms, so the resulting noise matrix is diagonal. Therefore, the data set as well as the system architecture is shared with the system identification routine described in the section before.

Unknown values for R and Q will result in suboptimal $\hat{x}_{k|k}$. Changing the covariance matrix affects the state estimation which again affects the noise estimation. Therefore, we need an initial guess about the noise and perform 5 iterations for convergence.

As we already have information about the precision of the camera system which should reflect the measurement noise in position data, we can roughly assume the noise and set it initially to

$$R = diag(2.5 \times 10^{-6}, 2.5 \times 10^{-6}, 2.5 \times 10^{-6}),$$

as we assume the precision of the camera to be $1 \times 10^{-6}m$. The initial guess for the system variance is more challenging, as it is not attached to a specific value. We assume the model to be very accurate. So we start with a variance that is very certain in the position, and a little more uncertain in the rotation.

$$Q = diag(1 \times 10^{-10}, 1 \times 10^{-10}, 2 \times 10^{-7}, 2 \times 10^{-7}, 1 \times 10^{-6}, 1 \times 10^{-6}).$$

Note that the filter expects an invariant noise over the whole system model which includes both the collision and the linear part of the puck's movement.

Moreover, we can configure the algorithm to estimate only a subset of the state variance while assuming fixed pre-defined noise for the remaining ones.

4.5 Data Set

To generate training and validation data we recorded the puck's movement on the real air hockey table. We used an OptiTrack system with six OptiTrack Flex 13 cameras with (1920 × 1680) pixels resolution. Each camera has a frame rate of 120 Hz and is placed above the air hockey table. The table has a length of 1.956m and a width of 1.042m. The goal is 0.25m wide and the puck has a radius of 31.65cm. The world coordinate origin is placed outside of the table like shown in 4.3 in the bottom left corner of the mount. Therefore, the table's center is located at the coordinate of $(1.504, 0.0.07)$ with the same orientation as the world coordinate frame. The puck and the table are tagged with multiple markers to allow us to observe spin information and a position with a precision of $1mm$. The data recorded by the camera system is the center of the disc, which is calibrated before. In

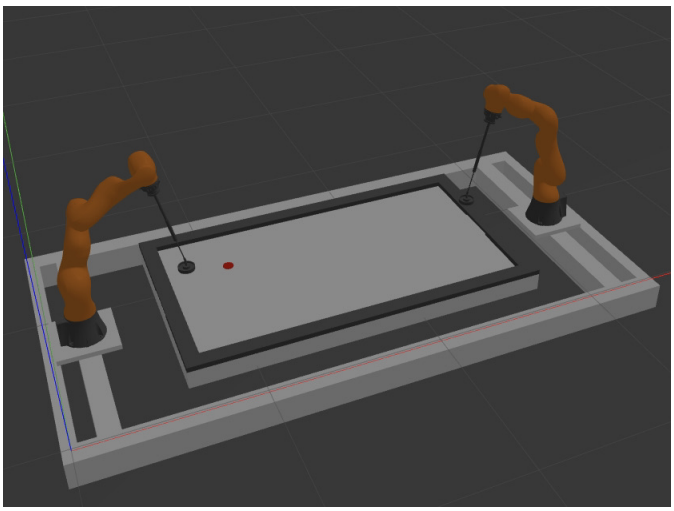


Figure 4.3: Air Hockey Table in simulation

total, we recorded 104 trajectories with an average length of 845 data points and with at least one impact in each trajectory. The puck is hit by the mallet from a rest position, the average of the initial velocity is $3.86m/s$. To collect as many different positions of the table as possible, we are varying the direction, trying not to score a goal to get a long trajectory also containing the process of slowing down. The aim is to get as many points of the table and different impacts with the wall as possible.

The resulting data set is split up into a training data set with 94 trajectories and a validation set with 10 trajectories with a total of 8400 data points.

4.5.1 Preprocessing

In the Kalman filter, the time step size dt for the forward model is fixed. In the recorded data, the time step between two observations is defined by the camera rate of $120Hz$ but can vary $dt + \epsilon$ due to technical noise. Furthermore, some frames can be lost and the records can contain consecutive frames with the same value, which causes outliers in the velocity calculations. To limit the effects of the noisy data, we implement preprocessing procedures to clean the data. Besides this, we also cut the data at the beginning and the end to exclude the influences of the humans, like the hitting or the adjustment of the puck in the end for the next trajectory.

Lost frames

To detect lost frames and to adjust the real-world data to the model, we compare the timestamp of the data with the model timestamp of the Kalman filter. This method ensures that for each model timestamp, we have the corresponding timestamp in the data set. Note that a tiny difference is still possible.

As shown in algorithm 2, we are initializing the model time stamps by the fixed frequency of the model from 0 to the end time of the recorded data. After this, we are iterating over each of the model timestamps to find the closest data point in the recorded data. If the next data point we get is closer to the next model time stamp, we insert a blank data point with the right timestamp, which helps for the Kalman Filter update described in 3.2. In case there would be two data points referring to the same model time stamp, we delete one to get an exact mapping.

$data \leftarrow t_0 \dots t_n$
$time_{model} \leftarrow [0 \dots k_n, stepsize = 1/120]$
for k_i *in model* **do**
    if t_j *is closer to model* k_{i+1} **then**
        append(blank data with model time k_i)
    else
        while k_i *closer to* t_{j+1} **do**
            remove t_j from dataset

Algorithm 2: Add lost data frames

Outlier

In our definition, an outlier is a point where the value of the position does not change at all, so the resulting velocity in this point would be zero, which is unexpected for the dynamical system. To avoid these jumps in the velocity, we apply interpolation for the data points, where the distance of the position is smaller than a threshold value of 10^{-6}.

4.6 Hitting Strategies

In this section, we describe the test setup and the evaluation criteria of the different skills of the robot. Many of the criteria are based on specific positions on the table which will be described first.

The air hockey table with length l and width w is divided into two sides for each robot: the own side and the opponent's side, delimited by the centerline. As $l >> w$, we divide the rims into two groups: long rim, for the rims along the length, and short rim, for the rims along the width. Each side contains a defending line, which has a distance of 20cm to the short rim and is used as a reference line for the robot to plan an action. Furthermore, each side contains a smash area with a distance to each rim, so the robot can hit the puck from every side and find a feasible solution for a trajectory towards the opponent's goal.

The robot is able to perform four different actions: Smash, Cut, Prepare and Repel. Smash is the strategy of the robot to score goals and has three sub strategies: scoring directly without wall contact or playing across one of the long rims to the goal. Cut is a defense mode with the aim to be able to Smash afterward, so the puck should stay in the smash area after a Cut. Sometimes the puck gets into areas at the edge of the table, where it is impossible to apply Smash directly, so we also have a strategy Prepare to hit the puck back into the smash area. The Repel strategy is applied when the puck is approaching the goal directly without hitting the walls. This strategy is to defend and use the velocity of the puck directly for a counter strike. To evaluate the strategies we define a success characteristic for each one:

Smash Success is defined by the score ratio and the distance to the goal center.

Cut Success is defined by the ratio the puck stays in the smash area after hitting and partial success the hitting rate of the puck.

Prepare Success is defined by the ratio of the puck stopping in the smash area or partial success for stopping in the own area.

Repel Success is defined by the hitting rate of the puck and the velocity of the repelled puck on the opponent's defending line.

For the testing, we are setting the puck's state manually and forcing the robot to use the desired strategy, while the opponent robot is disabled. Figure 4.4 shows the table with the smash area in blue and the defending line of the own side and the opponent's side in cyan. The back lines indicate the border of the table and the red line the goal. The

orange line defines the task space reachable for the robot on the table, without running into singularities.

For the initialization, we have a static position for Smash and Prepare and a dynamic position for Cut and Repel. The initial position of the puck for the Smash action is inside the smash area. To force the robot to use Prepare, the puck is set outside the smash area but inside the task space, reachable for the robot.

Initializing the position and velocity for Cut and Repel includes challenges about the timing of setting the puck and the strategy. To simulate a hit from the opponent, we are setting the puck randomly on the defend line of the opponent and set the magnitude of velocity to the maximal possible velocity reached by the robot, which is $2m/s$. To perform the action correctly, we need the prediction of the puck on the defend line in advance. So, for determining the time to execute the action the robot is waiting until the prediction of the puck satisfies all restrictions for performing, e.g., reaching the defend line straight to perform Repel.

The direction of the puck's velocity is set for the Repel action directly towards the goal's center, and for the Cut action across one long rim, varying the collision point in the own side.

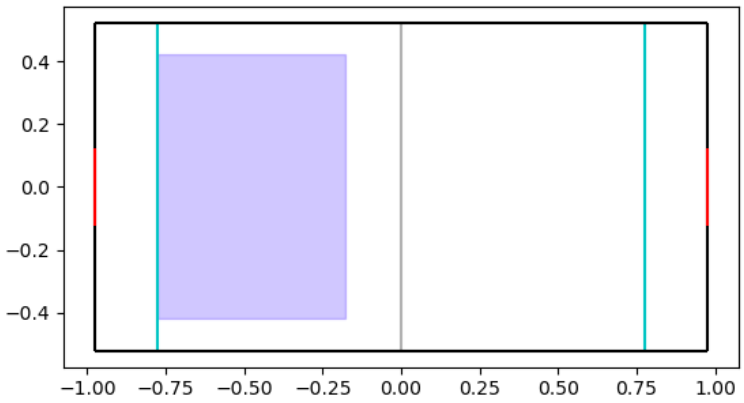


Figure 4.4: Air Hockey Table visualization with the smash area in blue and defending line in cyan

5 Results

In this chapter, we will provide and discuss our results. For visualizations of the puck movement, we choose a challenging trajectory from the validation set, which is hard to predict. That enables us to point out the improvements, as the non-optimized model is already capable of dealing with simple trajectories with a low number of collisions and a moderate initial velocity and spin.

We start discussing the results from the changed dynamical model of the puck movement and continue with the physical parameter identification.

5.1 System Model

For the process model of the Kalman filter we implement two improvements:

1. using a more differentiated collision model (see section 3.1)
2. using Jacobians for covariance update (see section 4.2)

To validate the updated system model, we compare the velocity of the estimated and the measured state of the puck. We choose the velocity, as the new system model redefines the calculation of the velocity after an impact with the wall. Beyond, the noise of the state is higher in the derived velocity than in the position and therefore the filtering is more visible.

Figure 5.1 shows the derived velocity of the measurement and the estimation of the velocity by the Kalman filter. The line colored in cyan represents the estimation and the blue line the measurement for each direction $(\dot{x}, \dot{y})$ and the rotational velocity $\dot{\theta}$. We can see a lot of oscillations in the measurement due to the noise of the measured data. Differences between the new and old system come from slightly different preprocessing methods. The changes of the sign in the velocity, in x or y or both directions, indicate a

collision with the wall. The more consistent parts indicate the linear movement of the puck, where the velocity is only reduced by friction and damping. At the top of figure 5.1, we see in the first collision a high discrepancy of the expected velocity, displayed by the light blue line, and the resulting velocity in the measurement. This error slightly reduces with time as the filter adapts to the changes. Nevertheless, the error is propagated and reinforced when the velocity estimation is wrong before the collision.

After about 3 seconds, we can observe a spike in $\dot{y}$ of the old model, which is caused by the collision model. The old model only models the case where the sliding terminates in the collision. As before, the filter adapts to the measurements quickly but will provide a wrong belief because of the wrong collision estimation.

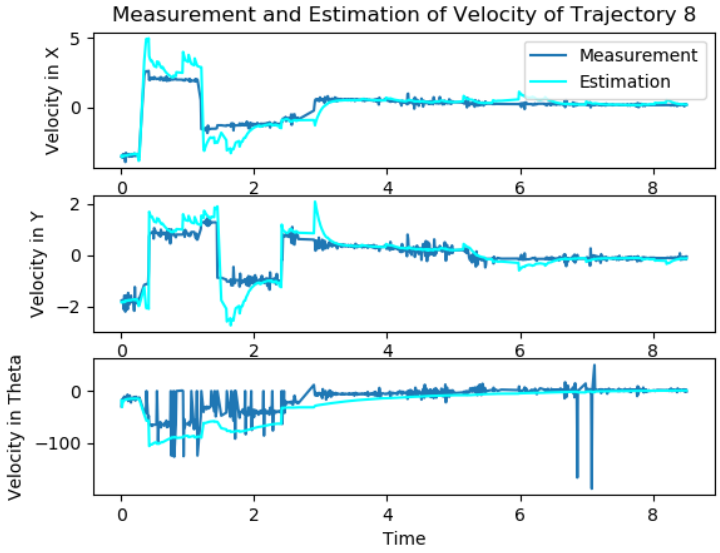
With the new system model in the lower part of figure 5.1, the velocity is estimated before and after an impact very precisely and only shows little differences. While the estimation of the translation velocity is very smooth, the rotational velocity estimation still fluctuates a bit, because of suboptimal noise configuration. Note, the new system model uses the suited physical parameters.

5.1.1 Execution Time

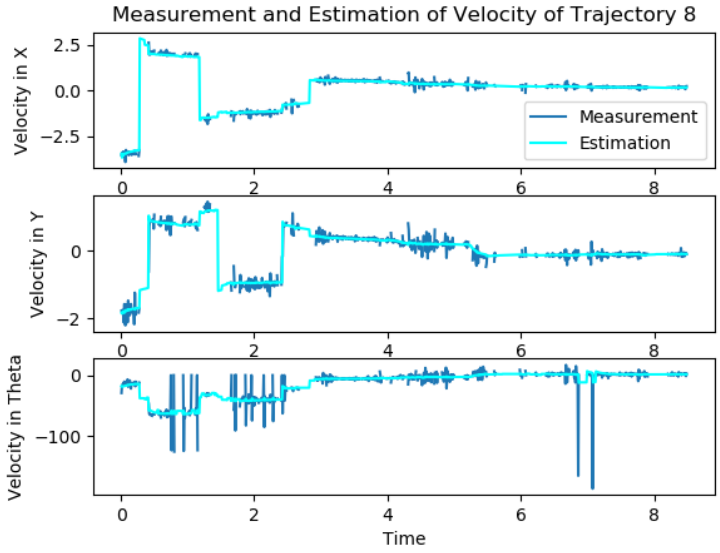
In the last section, we discussed the improvements of the modifications to the system model. However, the modified model uses an extended version of the collision model, which is also used for the system transition model. Therefore, the model requires additional matrix transformations and computations which increase the execution time of a prediction. To ensure the prediction can still be performed in one step of the Kalman filter, we measured the execution time of 2000 predictions.

The update rate of the system is at 120Hz, i.e., $0.00833sec \widehat{=} 8.33ms$, defined by the observation rate of the camera system, and is the maximum time which is available for the prediction. To evaluate the execution time, we therefore use the worst case of performing 180 steps, which is the prediction horizon maximum of the baseline agent.

Table 5.1 shows the execution time of one prediction for each model type. We assume the time to be Gaussian distributed, resulting in 95% of the prediction computations can be performed in under $93259ns \widehat{=} 0.093259ms$, which is still fast enough to perform the prediction in one time step.



(a) Old model



(b) New system model

Figure 5.1: Estimated vs. measured velocity of Kalman filter

	mean	std
simple Model	21837	3578
modified Model	75741	8759

Table 5.1: Mean and standard deviation of prediction execution Time in ns

t_{pred}	μ_{surface}	μ_{rim}	damping	e_{table}	loss	min	median
0.5	1.01×10^{-6}	0.13167	0.50001	0.90038	0.00706	0.00110	0.00405
1	7.96×10^{-8}	0.13523	0.31874	0.88078	0.01174	0.00542	0.01548
1.5	3.13×10^{-7}	0.11105	0.50015	0.99825	0.04356	0.01982	0.03969

Table 5.2: Loss and physical parameter values for learning with position error for different prediction horizons

5.2 System Identification

In section 4.3, we described the system identification with the two different objective functions. Table 5.2 and 5.3 show the resulting parameter values and the respective position error for different prediction horizons. Each of the tables is based on a different loss function.

The tables allow conclusions about the comparison of the different loss functions, as well as the loss evolution over the prediction horizon. Further, we can derive the final physical parameters. In the following subsections, we will discuss each of the conclusions in more detail, starting with the objective function, then the loss function, and finally the physical parameters.

5.2.1 Objective function

The loss values in the table are the average position distance in meter between the prediction and the measurement for each point. For the position based optimization, we, therefore, get an average divergence of $7mm$ per point and for the log likelihood based

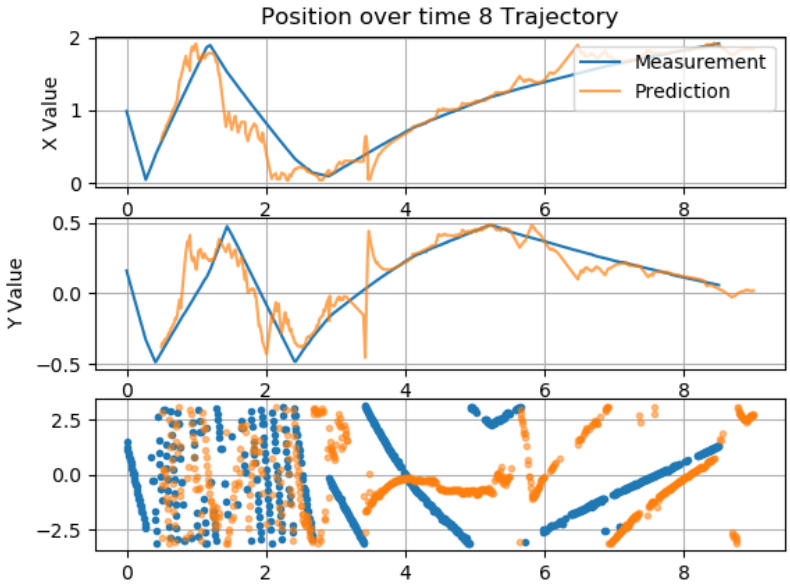


Figure 5.2: Comparison of the real data and the prediction data using hand tuned physical parameter

optimization $3mm$. These values are only based on x and y, the rotation is not included in the loss.

Even though the differences do not seem very big, Figure 5.2 to 5.4 show the effects on the position values (x, y, θ) over time. Therefore, we use a prediction horizon of half a second and visualize the example trajectory. As already mentioned, the trajectories in the validation set differ a lot regarding the loss. To emphasize the improvement of the different methods, we selected a trajectory with a comparatively high loss. In the example trajectory, the measurement data is indicated by the blue line and the prediction data is indicated in orange. Figure 5.2 shows the trajectory with hand tuned physical parameter, figure 5.3 with learned physical parameter using the position based error function and figure 5.4 with the log likelihood based error function. While the position data improve for both of the optimization routines, the big difference is in the improvement of predicting the rotation, pictured at the bottom of the figure and is particularly noticeable after 3 seconds.

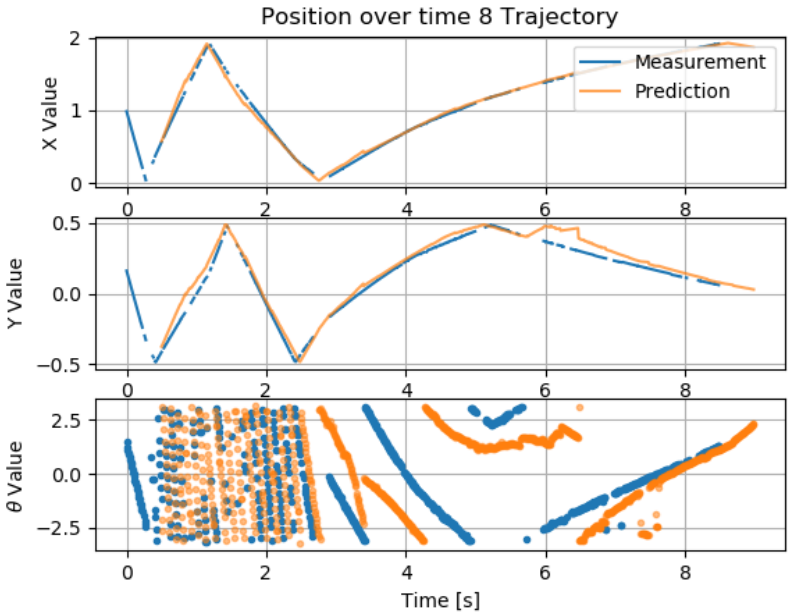


Figure 5.3: Comparison of the real data and the prediction data where the physical parameter are learned using position error

The highest derivations are the predictions with no new observation (due to lost frames) when the state prediction only is based on the system model without any new data information. Even though we use an extensive model, it is impossible to get a perfect state prediction, so small differences between the state prediction and the measurement are unavoidable.

Notice that the prediction of the orientations gets much better with increasing time, that is, when the puck movement slows down. The validation trajectory 8 visualized in figure 5.4 has an initial velocity of $3.9m/s$ and reaches the maximal hitting velocity of the robot, i.e., $2m/s$, at second 1.25. Hence, we are especially interested in the predictions after 1.75 seconds. Of cause, it is desirable to predict higher velocities to make the robot able to play against a human, but as the aim is to develop a baseline for the robot-against-robot air hockey, this prediction is accurate enough.

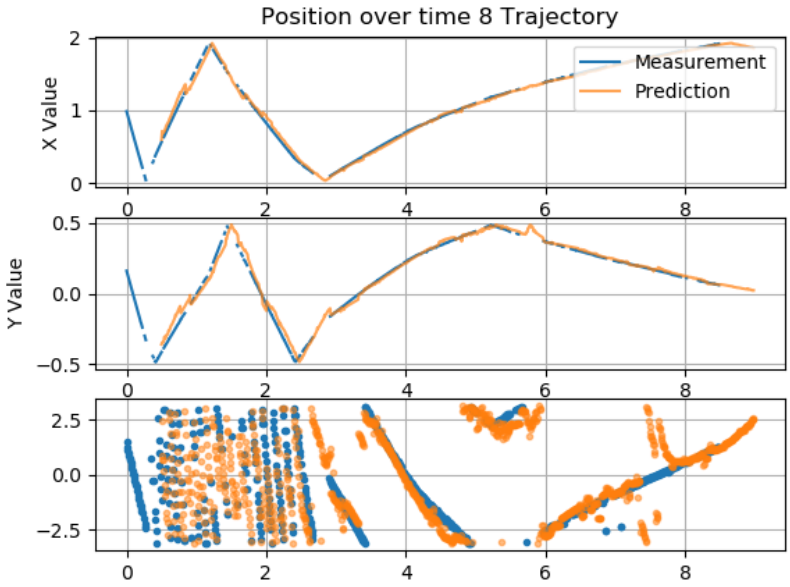


Figure 5.4: Comparison of the real data and the prediction data where the physical parameter are learned using log likelihood error

5.2.2 Loss Trend

The loss is described by three values. *Loss* indicates the average position distance for each data point, i.e., each data point is weight the same. *Median* indicates the average position distance in the trajectory which is in the stored list midst. Finally, *min* is the average position distance of the trajectory which is fitted best.

In the parameter table 5.3, we only show the error based on the position rather than the error based on log likelihood for comparability and interpretability. We rate the results with a lower median than loss higher, as we assume the loss to be more affected by outlier trajectories, than the median. As the prediction horizon increases, so does the error. Figure 5.5 shows the evolution of the error with increasing time, using the same physical parameters.

The log likelihood based error function outperforms the position based error function for all prediction horizons. We attribute this behavior to the fact that the error in the position

t_{pred}	μ_{surface}	μ_{rim}	damping	e_{table}	loss	min	median
0.5	2.79×10^{-6}	0.15213	0.49032	0.87490	0.00371	0.00068	0.00321
1	6.31×10^{-6}	0.15340	0.51433	0.92426	0.00855	0.00283	0.01164
1.5	2.28×10^{-4}	0.09848	0.50128	0.94717	0.02229	0.00931	0.02441

Table 5.3: Loss and physical parameter values for learning with log likelihood error for different prediction horizons

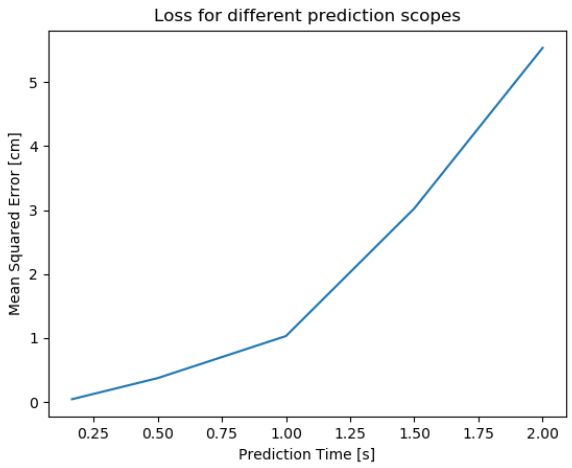


Figure 5.5: Position error of prediction with increasing prediction horizon

based method is weighted the same, regardless of the certainty, i.e., a difference in the collision is weighted the same as in the linear part.

5.2.3 Physical Parameters

Due to the sample based optimization routine, we get different results for each run. The estimated value for the surface friction μ_{surface} varies a lot. However, this should not affect the model a lot, as all the values are close to zero. The friction of the rim μ_{rim} is significantly smaller for the small prediction horizons than for the long prediction horizon. We expect the long prediction horizon to be more affected by correlations of the parameters.

t_{pred}	learn t_{pred}	loss	min	median
1	0.5	0.00943	0.00304	0.01311
1	1.0	0.00855	0.00283	0.01164
1	1.5	0.00852	0.00299	0.01001

Table 5.4: Validation of the physical parameters learned with different prediction horizons for a prediction horizon of 1

In section 4.3 we described the advantages and disadvantages of using the different prediction horizons for learning. To set the final physical parameters we evaluated the learned parameters against each other.

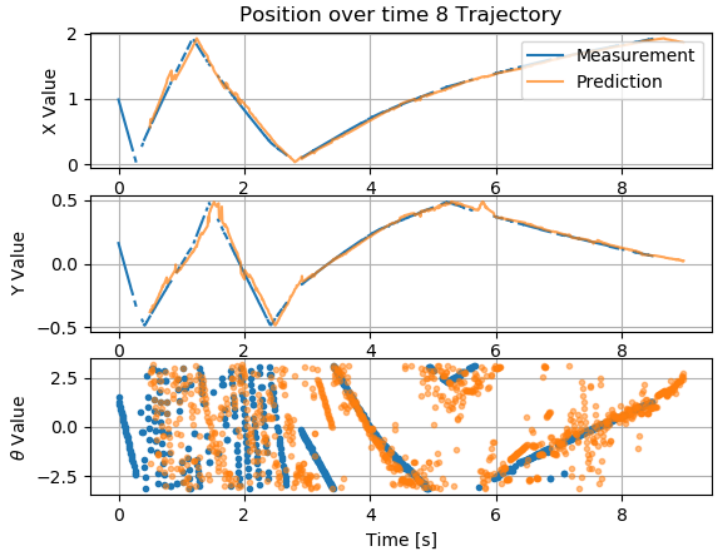
In order to choose the final parameter configuration of our system, we first define the desired prediction horizon to be $1sec$. This provides enough time for the control and, as the position error of the prediction increases exponentially, it is still small enough to get a good state prediction. So, we performed the validation for the $1sec$ prediction horizon using the physical parameter of the $0.5sec$ and $1.5sec$ prediction horizon, respectively. Table 5.4 shows the results where the first column is the prediction horizon for the validation and the second column is the prediction horizon of the system for the learning part. We finally select the parameters learned with a prediction horizon of $1.5sec$, as this improves the performance in the average and median loss.

5.3 Noise Estimation

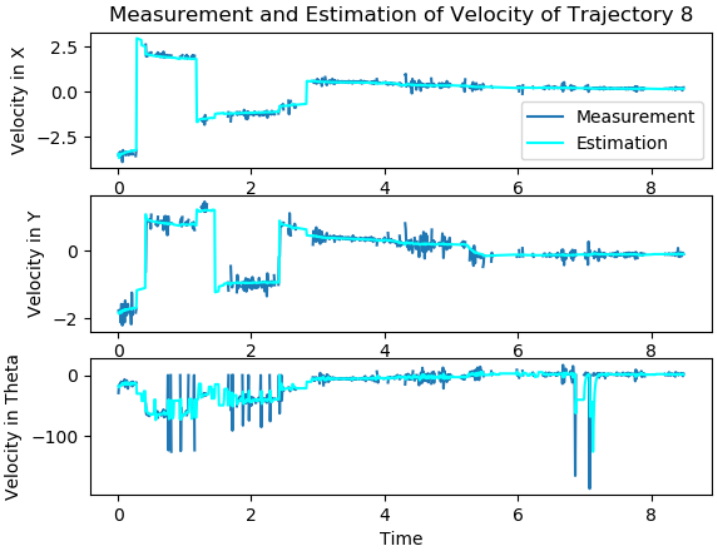
To improve further of the Kalman filter and obtain smoother trajectories we apply noise variance estimation. The measurement variance estimation, described in section 3.2.2 with eq. 3.12 results in a variance of:

$$R = \text{diag}(1.036 \times 10^{-7}, 1.361 \times 10^{-7}, 9.050 \times 10^{-5})$$

This reflects the technical details of the Opti Track Flex 13 camera, which indicates a precision of $0.1mm \widehat{=} 1 \times 10^{-7}m$ for most systems. As the variance estimation for the rotation seems reasonable, we are setting the measurement variance to the estimated values.



(a) Position prediction



(b) Velocity estimation

Figure 5.6: Example trajectory with prediction and estimated noise

Estimating the system noise variance with eq. 3.13 results in:

$$Q = \text{diag}(7.028 \times 10^{-6}, 7.625 \times 10^{-6}, 5.450 \times 10^{-4}, 7.015 \times 10^{-4}, 0.014, 42.43).$$

In contrast to the initial covariance, which we are using for the data in figure 5.2 to 5.4, the estimated noise is a lot higher for all parts. This high noise leads to a lot of trusts in the measurement data, especially in the rotational part, as the covariance estimation rates this part very uncertain. A possible reason could be noise in the velocity data. It is well known that computing derivatives of noisy measurement data are critical. Through the expected high noise in the system, the outliers of the measurements and their derivatives influence the state estimation. Moreover, we model the decreasing velocity of the rotation only by the noise, so this may also lead to a higher estimate.

In figure 5.6, we show the same trajectory as in figure 5.4 using the same physical parameters. The only difference is Q and R, where we use the values estimated before in contrast to the initial covariance. The miss estimation of the noise is particularly visible in the rotational data, but also in the translation data. This effect is mostly caused by the updated system noise, as the measurement noise does not change much. We already discussed possible reasons in the paragraph before. In order to set the final noise covariance for the filter, we evaluated different configurations:

1. Change the measurement covariance and keep the system covariance to the initial assumption
2. Change the measurement noise covariance and the translational part of the system noise covariance

While the prediction accuracy improves a little ($\approx -2 \times 10^{-6}$) when we adapt R, changing Q to the estimated covariance worsens the prediction. We therefore change R but keep Q to the initial values.

5.4 Hitting Strategy Test

Figure 5.7 shows the success rate for each of the hitting strategies. For Cut and Prepare it also shows the defined partial success in more transparent blue. Smash has a score of 82,2% for the straight hit, 73,6% across the right, and 73,4% across the left rim. After a Prepare hit, the puck stops in the smash area in 26,15% and in the own side in 44.62% of the cases. The success score of the Cut strategy is 34.3%, i.e., the puck stops in the smash

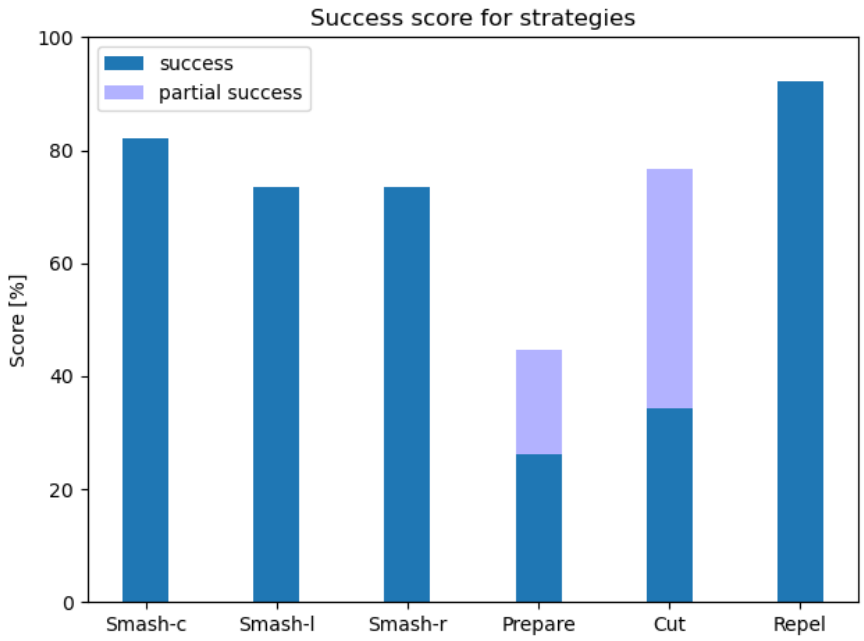


Figure 5.7: Success and partial success score for each strategy

area after the Cut, but as it's also a defend mode, 76.66 % of the incoming pucks can be stopped. The robot repels the puck after a straight attack in 92% of the cases.

5.4.1 Smash

For each of the smash strategies, we aim at the center of the goal. Figure 5.8 shows the average distance of the puck to the goal's center after applying one of the three Smash strategies. The red lines indicate the goal's boundaries, so a sample hitting one of these lines would not score a goal.

While playing directly has the lowest bias to one side, it is noticeable playing against the right wall results in hitting the goal rather in the left and vice versa. This can be explained by the model we use to plan the hitting, which does not take rotation into account, but the reflection law. The robot usually gives a spin to the puck through the cylindrical shape, which affects the impact and outcome. Nevertheless, the mean and standard derivation of the positions are still inside the goal.

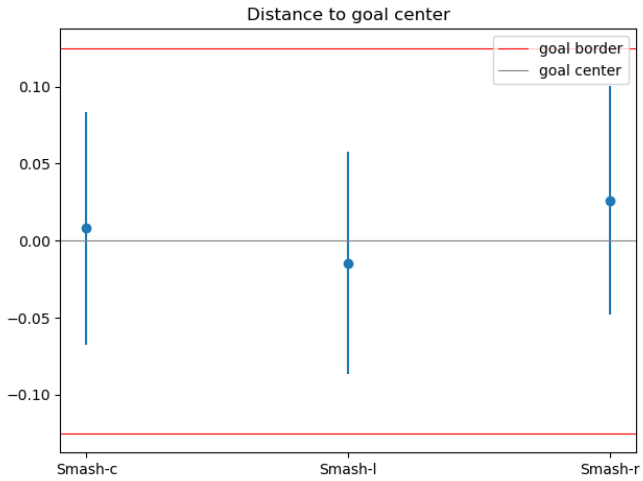


Figure 5.8: Distance and standard derivation to goal center of different smash strategies. The red line indicates the goal limits

5.4.2 Prepare

The aim of the Prepare strategy is to put the puck into a better position to be able to Smash it. Therefore, the puck has to be pushed against the long rim with a regulated velocity. In our Prepare implementation, we always hit a little in the x-direction, i.e., in the direction of the opponent's goal. This leads, especially for the initial samples near the centerline, to cross the centerline into the opponent's side. When the puck crosses that line, it's automatically counted as a fail. On the other hand, when the impact mainly pushes the puck in the y-direction to the closest rim, the mallet might still be unintendedly placed in the trajectory of the puck bouncing back from the wall, which can lead to high spin movements. The task of positioning the puck accordingly is not trivial, as it would also need back movements by avoiding own goals strictly. Nevertheless, the score is still upgradable, even in the limitations of the baseline agent, i.e., the strategy needs to be based on the puck's state only.

5.4.3 Cut

By generating the test data, we did not control the puck's movement to aim at the goal, making it hard to evaluate the strategy by the ratio of defended incoming pucks. Still, 1%

of the test attacks could score a goal. On the other hand, using the Cut strategy, the robot manages to stop the puck in the smash area which is, as described before, a challenging task. The translation velocities of the pucks reaching the defend line on the own side are $0.63 \pm 0.2m/s$, as the initial velocity is $2m/s$ and the puck's trajectory includes at least one collision. The partial success of over 75% can be rated as very good. For success, the momentum of the puck is often still too high to stop in the smash area.

5.4.4 Repel

The Repel strategy's goal is to return the puck as fast as possible to the opponent's goal. The almost optimal success score is caused by the test setup, as the puck is approaching straight without wall impact, which is very accurately predictable. Indeed, the strategy performs very well, reaching velocities of over $2m/s$ on the opponent's defending line, and an average velocity of $1.57 \pm 0.1m/s$. For future work, the hitting direction could be improved as only 13% of the pucks score a goal. Currently, scoring is no part of the trajectory planning in Repel.

6 Conclusion

In this work, we showed routines and algorithms to get a precise prediction of a puck's movement in robot air hockey. For slower movements, we are even able to predict the orientation of the puck, which will allow more complex hitting strategies in the future.

To be able to improve the prediction, we used a combination of a linear and differential system model, modeling the straight movement on the table without impacts and the collisions with the wall. These models are based on the physical parameters which we learned using a black-box optimization algorithm. Therefore, we use large data sets of recorded puck trajectories and two different loss functions. There, we showed that using a likelihood loss performs better and results in a more precise prediction than using a loss function based on the position's distance.

Using this, we were able to improve the prediction accuracy significantly by over 80% for the $1sec$ prediction horizon even though the results still suffer from the noisy measurement data.

To deal with the noisy data, we implement an estimation for the variance of the noise for the process model and for the measurement model. This routine is affected by outliers and cannot compete with the results we get setting the variance to our assumptions. When we estimate the variance of the model system, we approximate the variance for the whole system and assume a homoscedastic noise. This is problematic for the composite model due to the different characteristics of the model components.

In order to improve the puck's movement prediction further, one research direction could be to use different noise covariances for the different model parts, i.e., the linear and the collision model. Another promising approach could be to adapt the dynamics to a higher order system with constant acceleration.

Even though we do not have an optimal noise variance set up, using the optimized prediction, we showed and introduced evaluation methods for the robot air hockey baseline agent and evaluated the different strategies of the robot. These results can be used for

further work on the tactics, on the one hand, e.g., for using a complex strategy selection based on machine learning, or on the other hand to see if improving the puck prediction further will lead to a better playing performance. Furthermore, the tactics themselves can be improved, e.g., by combining different consecutive strategies. The robots enable us to implement new stand of art algorithms regarding selecting and planning complex tactics. The baseline agent then serves as the opponent and validates the improvement of the learning. For the robot baseline agent in the simulation, we showed that the tactics perform as expected by following the safety constraints, like keeping the mallet on the table's surface. Hence, the next step for our system is to transfer the strategy test to the real-world robot to get the robots to play air hockey against each other.

Bibliography

[1] P. Liu, D. Tateo, H. Bou-Ammar, and J. Peters, "Efficient and reactive planning for high speed robot air hockey," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 586–593, 2021.

[2] B. E. Bishop and M. W. Spong, "Vision-based control of an air hockey playing robot," *IEEE Control Systems Magazine*, vol. 19, no. 3, pp. 23–32, 1999.

[3] C. B. Partridge and M. W. Spong, "Control of planar rigid body sliding with impacts and friction," *The International Journal of Robotics Research*, vol. 19, no. 4, pp. 336–348, 2000.

[4] Y. Wang and M. T. Mason, "Two-dimensional rigid-body collisions with friction," *J. Appl. Mech.*, 1992.

[5] J. I. Park, C. B. Partridge, and M. W. Spong, "Neural network-based state prediction for strategy planning of an air hockey robot," *Journal of Robotic Systems*, vol. 18, no. 4, pp. 187–196, 2001.

[6] S. Ghazvini, H. Moradi, M. N. Ahmadabadi, and B. Araabi, "Fast outcome prediction based on slow cause estimation: A human inspired approach in air hockey game," in *2011 IEEE International Conference on Robotics and Biomimetics*, pp. 2810–2815, IEEE, 2011.

[7] H. Alizadeh, H. Moradi, and M. N. Ahmadabadi, "Automatic calibration of an air hockey robot," in *2013 First RSI/ISM International Conference on Robotics and Mechatronics (ICRoM)*, pp. 107–112, 2013.

[8] H. Shimada, Y. Kutsuna, S. Kudoh, and T. Suehiro, "A two-layer tactical system for an air-hockey-playing robot," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6985–6990, 2017.

[9] W.-J. Wang, I.-D. Tsai, Z.-D. Chen, and G.-H. Wang, "A vision based air hockey system with fuzzy control," in *Proceedings of the International Conference on Control Applications*, vol. 2, pp. 754–759, IEEE, 2002.

[10] G. Welch, G. Bishop, *et al.*, "An introduction to the kalman filter," 1995.

[11] P. I. Frazier, "A tutorial on bayesian optimization," *arXiv preprint arXiv:1807.02811*, 2018.

[12] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *Advances in neural information processing systems*, vol. 25, 2012.

[13] A. I. Cowen-Rivers, W. Lyu, R. Tutunov, Z. Wang, A. Grosnit, R. R. Griffiths, H. Jianye, J. Wang, and H. B. Ammar, "An empirical study of assumptions in bayesian optimisation," *arXiv preprint arXiv:2012.03826*, 2020.