

Air hockey Robot

sw503e19

Autumn 2019



AALBORG UNIVERSITY
STUDENT REPORT

Air Hockey Robot
An Autonomous Sparring Partner

Group Name:

sw503e19

Supervisor:

Peter Gjør Jensen

Group Members:

Simon Vinberg Andersen
Mathias Knøsgaard Kristensen
Sebastian Hjorth Hyberts
Simon Svendsgaard Nielsen
Felix Cho Petersen
Shahab Shajarat

Project Page Count:

68

Total Page Count:

74

Denne rapport udforsker muligheder og problemstillinger forbundet med at lave et indlejret system. Omdrejningspunktet for projektet, er en undren vedrørende hvorvidt det er muligt at udvikle en robot som på autonom kan simulere dele af air hockey processen, ved at beskytte en målzone. Dette gøres via en analyse, som belyser anvendelige sensoriske enheder, kontrolenheder, samt motorer. Endvidere danner disse teknologier basis for en simuleret model, som opstilles i et hybrid tids-automat afprøvningssoftware med henblik på at verificere modellen inden implementation. Denne proces beskrives i omfattende stil, og designvalg taget i løbet af processen forklares – både i hardwarevalg under designprocessen, men også valg relateret til den simulerede model, som skulle tilpasses virkeligheden. Yderligere, på baggrund af en omfattende verifikationsproces, bliver der konkluderet hvorvidt det indlejrede system reelt lykkedes i at håndtere den påkrævede spilproces. Der konkluderes at air hockey robotten er et færdigt system, forstået på sådan vis at det kan opfylde de meste basale krav for at spille air hockey. Ydermere konkluderes der også, at kamera input har stor indflydelse på en lineært ekstrapolerende kontrolenhed, og at fejlbart hardware har en stor indflydelse på ydeevnen. Ligeledes konkluderes der også at det kan være svært at inkludere virkelighedens brister i simulerede modeller.

The contents of this report are publicly available, but publication (with bibliography) has to be in agreement with the authors.

Preface

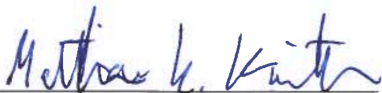
This report is written by sw503e19 during the autumn of 2019. The purpose of the project is to understand the process of creating an embedded system. All of this is done as an exercise to help understand real-time systems and their underlying concepts.



Simon Vinberg Andersen
svan17@student.aau.dk



Sebastian Hjorth Hyberts
shyber17@student.aau.dk



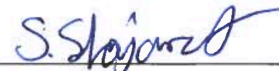
Mathias Knøsgaard Kristensen
matkri15@student.aau.dk



Simon Svendsgaard Nielsen
ssni17@student.aau.dk



Felix Cho Petersen
fcpe17@student.aau.dk



Shahab Shajarat
sshaja17@student.aau.dk

Contents

1	Introduction	6
1.1	Training	6
1.2	Air hockey	7
1.2.1	Related Work	8
2	Problem Analysis	9
2.1	Sensing	9
2.1.1	Sensing Light	9
2.1.2	Sensing Distance	10
2.1.3	Processing	11
2.2	Prediction	12
2.2.1	Naive Linear Extrapolated Prediction	13
2.2.2	Path Problem	13
2.3	Response	15
2.3.1	Legal Shot	15
2.3.2	Moving the Mallet	16
2.3.3	Platform	19
2.4	Problem Formulation	21

2.4.1	Problem Scope Limitation	21
3	Design	22
3.1	System Requirements	22
3.1.1	Performance Criteria	22
3.1.2	Dependability Requirements	23
3.2	Physical Design of the Table	26
3.2.1	Physical Design Decisions	26
3.2.2	Overview of the Table	27
3.3	Modelling the System	29
3.3.1	The Camera	29
3.3.2	The Puck	30
3.3.3	The Mallet	34
3.4	Controllers	36
3.4.1	Patrolling Controller	37
3.4.2	Two-step Controller	38
3.4.3	Three-Step Controller	40
3.4.4	Naive Linear Extrapolation Controller	41
3.4.5	Rebounding Linear Extrapolation Controller	43
3.4.6	Bi Rebounding Linear Extrapolation Controller	44
3.4.7	Combining Controllers	47
3.4.8	Picking a Controller	48
3.5	Robustness Analysis	49
3.5.1	Modifying the Camera Rate	49
3.5.2	Modifying the Camera Faultiness	51
3.5.3	Modifying the Mallet Speed	52

3.5.4	Modifying the Puck Speed	54
4	Implementation	56
4.1	Controlling the Motor	56
4.2	Controlling the Camera	56
4.3	Scheduling	57
5	Evaluation	61
5.1	Verification	61
5.1.1	Camera	61
5.1.2	Puck	63
5.1.3	Mallet	64
5.2	Discussion	65
5.3	Conclusion	66
5.4	Future Work	67
5.4.1	Utilising a Better Camera	67
5.4.2	Improving Upon the Model	68
5.4.3	Implementing Other Controllers	68
	Appendix A Restitution Influence	73

Chapter 1

Introduction

1.1 Training

Training in different sports has been commonplace throughout time. Often, a trainer helps a trainee, for example personal trainers for professional athletes, or club trainers in the case of sports clubs. A human trainer is versatile and can help with training not only physically, but also mentally, by performance reasoning and evaluation.

However, human trainers are not always available, and practising alone can in some cases be a difficult or, in some sports, impossible task. Therefore, for some of the simpler types of training that focuses on physical repetition, like a football goalkeeper practising catches, or a tennis player practising return shots, a robot can potentially be a solution. This type of automated training technology already exists today, in the form of, as examples, tracking a ball via computer vision or utilising ball cannons [1].

Simply shooting the ball at one or more pre-planned locations is not always enough. In some sports, like air hockey, training individual skills might best be exemplified during training. This is done by playing the game, as it naturally focuses on continually improving reaction times, precision shooting, and trajectory calculation. This type of practice through playing against a robot forms the basis of the initial problem:

What are the necessary components of a reactive autonomous air hockey robot?

1.2 Air hockey

In order to become familiar with the different components and their interactions within the game, the concept of air hockey and its rules are explored. As such, this section should give a basic understanding of the rules of air hockey and how the game progresses.



Figure 1.1: An air hockey board.

Air hockey is a game designed to be played by two players, each utilising a mallet, to interact with a puck. The puck travels across the board on a low-friction surface, aided by a constant air current, which allows it to reach high velocities. The two mallets and the puck are the only objects which are allowed to touch the surface of the board. The different components can be seen in Figure 1.1.

The objective of air hockey is to shoot the puck into the opponent's goal by using only the mallet. This leads to a game where both players attack and defend against each other until one player reaches a specified number of goals, in which case they are declared the winner.

1.2.1 Related Work

The goal of the initial problem statement is to find the necessary components for an air hockey robot. This report is not the first instance of such a robot being developed. Projects of this kind have been done before, one other instance of this being an air hockey robot that allows people to play together over a distance [2]. This robot is equipped with four puck cannons, that are hidden from the player. These are meant to emulate an incoming shot made from another table, while shots the player makes are collected and reloaded by hand. This process is reciprocated on the other table, effectively allowing to players to play out a match over great distances.

The robot created for air hockey over a distance gives an insight into some of the components that may be necessary to create an autonomous robot. One of these is some sensory equipment needed to detect the puck and its trajectory. However, as we wish to create an autonomous robot on a single table, some components can also definitively be excluded. The puck cannons, while a sufficient approach to reflecting a puck, would not be applicable for our robot, as we want to simulate a real player playing against the training player.

Chapter 2

Problem Analysis

From a human perspective playing air hockey can be considered relatively simple, especially if more complex matters like strategy are omitted. For instance, based on the information human eyes gather about the position of the puck, a response occurs via some sort of prediction as to the end position of the puck. Finally, with respect to said prediction, a reaction occurs by moving the mallet to intercept the puck. This simplified process of hitting the puck can be broken into three processes, namely, sensing, processing, and responding to information.

This chapter will explore said aspects, in order to consider how they can be utilised to fulfil the initiating problem.

2.1 Sensing

Before the robot even begins to physically interact with the game, it first needs to be able to gather information about the game and its current state. This can be done in multiple ways. We discuss a number of information gathering methods, and weigh the pros and cons of each technology, in order to find one that fits the problem domain. Additionally, this section also delves into how to possibly process the information the sensory equipment collects.

2.1.1 Sensing Light

Utilising cameras as digital substitutes, can be an adequate way to mimic human sensory input. There are multiple types of cameras, that each process information differently, as can be seen in Figure 2.1.

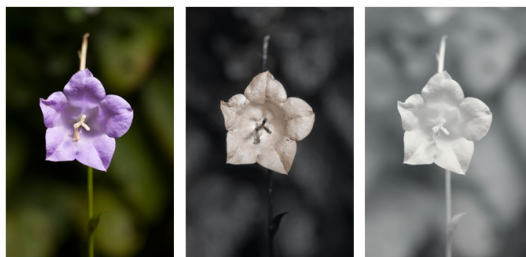


Figure 2.1: Telham flower shown in respectively visible light, ultraviolet light and infrared light. (Image by Dave Kennard [3])

Visible-Light Camera is an approach to simulating sight that, through the use of common cameras, captures light in the visible spectrum. Seeing visible light can be a great way to capture a specific target, such as a puck. This is because objects in sports are often painted a distinct colour to be visible and eye-catching to the human eye. A camera sensor is usually made to emulate the human eye.

Ultraviolet and Infrared Cameras are two other approaches to sensing something. Contrary to the visible light camera these two camera types, which capture light-wavelengths that are invisible to the human eye - these being ultraviolet and infrared light. This allows for them to not be affected by the room-lighting, making it useful for dimly lit environments. This also means that the sensed items could be painted in a paint invisible to humans, which reflects ultraviolet or infrared, meaning that it only reflects strongly from the desired objects. This creates greater contrast from the background, while not altering the appearance of the object [4].

2.1.2 Sensing Distance

Instead of sensing the puck via a camera, it could be detected as a bump on an otherwise flat surface. To do this, a signal could be sent out with a known speed, and the time until the reflected signal arrived could then be used to calculate the distance, as illustrated in Figure 2.2.

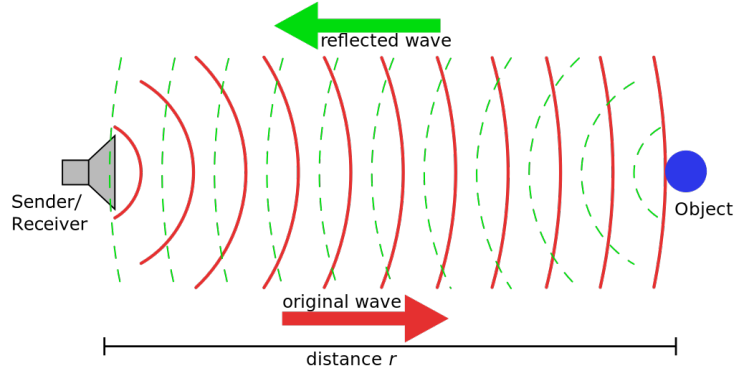


Figure 2.2: A simple reflection based distance sensor. (Diagram by Georg Wiora [5])

Ultrasonic Sensors are sensors used for measuring distance between the sensor and nearby objects. It works by sending a pulse of sound, outside the range of human hearing, and receiving the echo. The distance to nearby objects are determined by the time between sending a pulse and receiving the echo [6].

Light Detection and Ranging (LiDaR) is a type of technology used to measure distances between a given object and the LiDaR sensory equipment. LiDaR technology measures the distance to the object by illuminating it with a laser and processing the light reflected off of the object [7]. This can be utilised in various way, from spanning a singular object to mapping information from pre-defined search area to a 2D array [7].

2.1.3 Processing

After gathering the information another step is needed, namely processing the gathered information. Essentially, the goal is to derive a small amount of specific information about the whereabouts of the puck, from a larger amount of arbitrary information. To illustrate this idea, we consider a simple case where the goal is to determine the position of the puck from an image.

2.1.3.1 Background Subtraction

Background subtraction refers to a technique used for detecting moving objects via a static sensor instead of using the motion between the sensor and the

object. This method separates elements in the foreground from elements in the background by generating a foreground mask. This mask consists of a binary image containing the pixels belonging to moving objects, and is separated from static background. The rationale is that by separating the moving objects from the static background, the robot can find the desired objects that it needs to detect [8]. By defining the background, both types of information, being distance and light, can be simplified. This idea is illustrated in Figure 2.3.

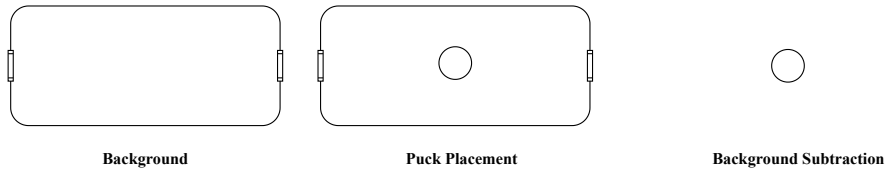


Figure 2.3: An example of determining the position of the puck through background subtraction.

As can be seen, this technique allows the system to isolate the puck, and thereby extrapolate its position. Furthermore, because of the simple nature of the method it is applicable to both the information gained from light and distance sensing. It should be noted however, that during an actual game of air hockey the mallets will also be moving, and therefore complicate the process as they can not be illustrated in the static background. This means that, adaptations are necessary to this method to reliably determine the location of the puck, should this method be chosen as the information gathering technique.

2.2 Prediction

After gathering information about the puck, the robot should be able to consider its next move. For this reason, a way for the robot to predict the final position of the puck should be available. Being able to predict the path of an object is the core aspect necessary to assume the future position of the puck during a game of air hockey.

Once information about the puck has been gathered, the robot should be able to calculate where to move. Using this logic, it would be beneficial for the robot to be able to predict the final position of the puck. Additionally, to know the final position of the puck, requires knowing where it has been, and where it is going. If left to its own devices, meaning applying no force except the initial, the puck follows Newton's first law¹, disregarding friction, air resistance etc naturally. In this case predicting the position of the puck would be simple. However, due

¹an object will move at constant velocity until a force is applied

to the nature of the board and the game, the puck can alter its trajectory. As such a naive attempt to predict the final position of the puck using linear extrapolation will be presented. Furthermore, once the position of the puck has been extrapolated, a pathing problem will be presented to illustrate the importance of an accurate and dynamic prediction method.

2.2.1 Naive Linear Extrapolated Prediction

In order to make predictions about the puck, information about the location of the puck is necessary. For this reason, we assume to know two positions of the puck, P_1 and P_2 . This is to allow for the extrapolation of the puck's position from a history. While knowing the given positions it is possible to predict a naive path connecting the two points as shown in Figure 2.4.

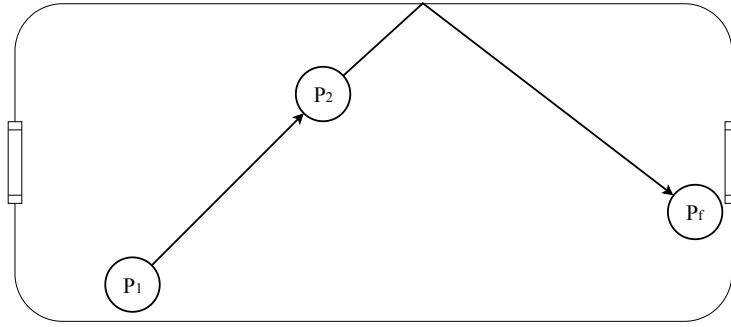


Figure 2.4: A naive attempt to predict the final position of a moving air hockey puck

Using this method, the proposed position to move the mallet to is determined to be P_f . However, this prediction fails to consider many physical attributes of the air hockey board.

2.2.2 Path Problem

An air hockey board can pose influences upon the trajectory of the puck. Specifically, the railing of the board and a player's hit are both concepts in which the trajectory of the puck can change suddenly, and why traditional predictions, as seen in Section 2.2.1, might be inadequate.

2.2.2.1 The Railing of the Board

The railings of the board allows the puck to ricochet and change its trajectory. This means that given two positional states of the puck, the initial position (p_i) and the final position (p_f), the puck could have taken one of five possible routes, which can be seen in Figure 2.5. This is assuming a maximum of a single bounce off the railing, which does not involve the rounded corners.

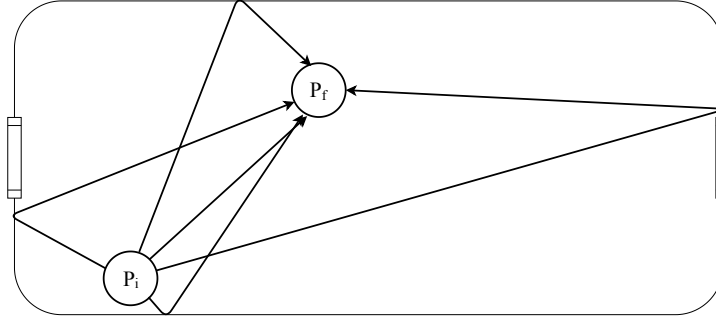


Figure 2.5: All possible routes which maps from p_i to p_f with a maximum of a single ricochet.

At this given moment there is no way to deduce which of the five courses the puck has taken with certainty, meaning the future position of the puck can not be predicted. The railings provide new routes for the puck to take in order to reach p_f and these need to be considered when determining where the mallet should move to.

We shall later devise methods for deduction of which of the five courses the puck has taken. This, however, means that we cannot predict the future position of the puck at current. The railings provide new routes for the puck to take in order to reach p_f and these need to be considered when determining where the mallet should move to.

2.2.2.2 Player Hits

Unless the movement of the players hand is tracked and analysed with respect to the trajectory of the puck, predicting the interaction between opponent and puck is impossible. Naturally, this also implies that a players hit might invalidate any predictions made, as these methods could predict a trajectory using the new puck information, which was changed due to the player hit, and the outdated puck information, meaning the trajectory prior to being hit by the player. Therefore the predictions would be incorrect in predicting the course of

the puck. To illustrate this idea Figure 2.6 shows a scenario where the trajectory of the puck is predicted, the courses caused by ricocheting have been omitted to simplify the figure.

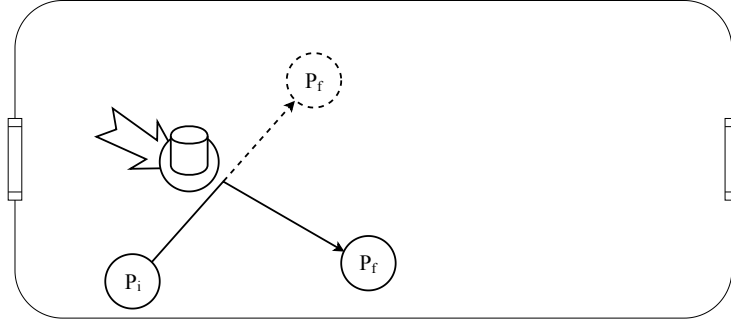


Figure 2.6: Abrupt change in trajectory caused by a player hitting the puck.

This way of affecting the puck means that the history of the positions of the puck can become outdated to predict its future positions. Therefore being able to detect and adapt to this abrupt change in behaviour is necessary, as it will likely occur multiple times over the course of a game.

2.3 Response

After addressing the path problem of Section 2.2.2 the robot should be left with a way to assess the final position of the puck, despite the forces which can be applied to modify the behaviour of the puck during travel. However, in order to play an air hockey game the robot needs to be able to interact with the game as well, rather than simply theorise. This warrants the need for the robot to be able to dictate the position of one of the mallets during play. Naturally, this means the robot should abide by the rules of the game which will be given in this section. Furthermore, possible ways to interact with the game will be presented through adding mobility to the mallet. Lastly, platform options will be considered and compared in order to explore the possibilities available to fully connect the system together.

2.3.1 Legal Shot

Section 1.2 explored the game of air hockey with the intent of giving readers a basic understanding of how the game functions. This section will give a more substantial analysis, with the intent of establishing a set of rules, that any

potential robot must adhere to. This will be done by analysing the rules in the United States Air Hockey Association’s official rules list, and creating a list of legal constraints [9]. The compiled list contains all rules which can influence the design of a robot.

1. If any part of a player’s hand, arm, body [this also refers to the body of the robot], or clothes touches the puck, ...
2. A player may play with only one mallet on the playing surface at a time.
3. The legal bounds of play are the table’s playing surface, the walls of the rails, the front faces of the goals, the interiors of the goals, and the player’s mallets. If the puck touches any other object(s) while it is in play, whether by interference or by foul (unless the foul is nullified) it is considered out of bounds and therefore instantly out of play.
4. No mallet may be altered by sloping the playing surface in order to create an angled striking or defending surface.
5. If interference occurs during a shot which scores and interference is called by referee, the point does not count. Interference is defined as foreign objects on the table or playing surface, ...

Figure 2.7: The United States Air Hockey Associations rules for when a puck is out of play [9].

As the design of the robot is meant to comply with the above-mentioned rules, the approaches are limited. For instance, having any part of the body of the robot make contact with the puck instead of the mallet is prohibited. Similarly, no objects besides the mallets and the puck are allowed to be in contact with the playing surface. These constraints are important to consider if it is required that the robot only performs legal shots.

2.3.2 Moving the Mallet

There exists many approaches to moving objects around via robots. However, with the size of the air hockey mallet taken into account, in addition to the rules described in Section 1.2, that amount decreases. In this section, we will present some possibilities:

Gantry is a common way to move objects around in areas where the space around the object has to be clear. Container ships and harbours usually move

shipping containers via a gantry crane. It allows for high precision, strong forces, and can be extended to move the object in two dimensions. However, moving the gantry along a second axis requires moving the first axis of the gantry. This makes the acceleration required to move the gantry at the same speed higher, which in turn increases the torque required.

For the air hockey board, a mechanism can be made which spans over the top of the board as illustrated in Figure 2.8. This mechanism has a way to hold the mallet, and allow it to move. The mechanism itself is then powered by a motor and a rail to pull the mallet along, or by a hydraulic or pneumatic piston.

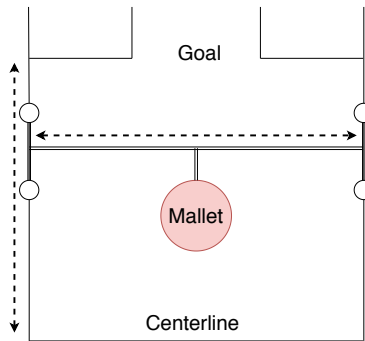


Figure 2.8: An example of a gantry based robot on an air hockey table.

Swinging Boom is a type of pole that rotates around an axle placed in one end of the pole. Boom arms can be used for object movement in various ways, most commonly known in boom cranes used for building and the boom of a sailing ship.

The air hockey mallet can be attached to a boom arm in one end, and the boom arm attached to the middle of a player side of the board as illustrated in Figure 2.9. As such, the boom arm is moved in a circle arc on the board. If the length of the boom arm is half the width of the table, the whole width of the table can be covered by the boom if it is right on the edge of the board. The boom is then moved either by rotation at the pivot joint, or by an extendable piston, attached to some point on the arm.

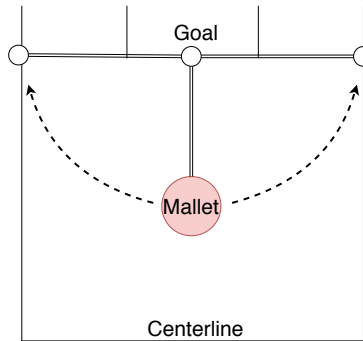


Figure 2.9: An example of a simple boom bot that moves the puck in a circle arc.

Robotic Arm is a widely-used method of robotic movement with the use of arms, often seen in industrial applications [10]. A robotic arm can manipulate the mallet, either by directly attaching the mallet to the arm, or with a grabbing attachment. The arm can be attached to the table by means of a separate platform, fixed to keep the base of the arm still as the arm moves around. The arm is illustrated in Figure 2.10.

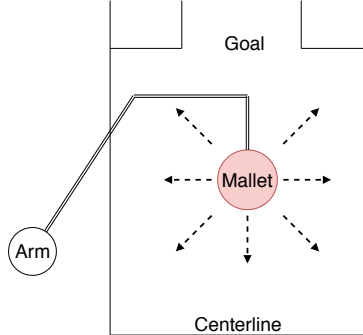


Figure 2.10: An example of an autonomous bot that uses an arm to guide the mallet.

Autonomous Mallet is a movement option that, instead of having a robot pick up and move an object, some objects can be turned into the robots themselves. In this case, making the mallet able to move itself, would allow for less modification of the air hockey board. For instance, this is achieved by adding wheels to the mallet in a way that does not obstruct the interaction with the puck, something that is illustrated in 2.11. These wheels are then driven by electric motors, allowing for quick acceleration and precise control.

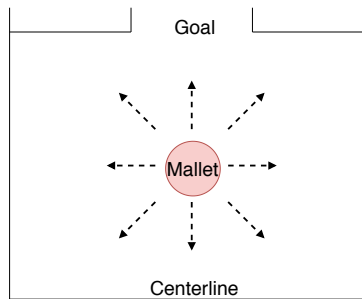


Figure 2.11: An example of some machinery that controls the movement of a mallet directly from the mallet.

2.3.3 Platform

There are many approaches to creating an autonomous robot that can play air hockey. This section will describe a few different computational platforms and their specifications and attempt to weigh pros and cons of the individual platforms. This will be done by comparing the clock speed, memory capacity, inclusion of operating system, price of the various platforms, and their I/O capabilities.

The **clock speed** of a given system is an indicator for the computational speed of the system. A faster clock will allow the processor to execute more instructions in a given time frame. It should be noted that clock speed is not the only indicator for the speed of a system, as the underlying hardware architecture can have greatly varying effect on the efficiency at which instructions are executed.

Memory capacity is an indicator as to the size of programs a system can execute. More memory allows for more instructions and data to be saved, and used simultaneously. A computer vision program might need a lot of memory to store the image frames that are analysed. Different types of memory also come with their own access speeds, which affect how long the processor may have to wait for different amounts of data.

Some controllers come with an **operating system**, and more advanced controllers generally comes with an associated operating system, to act as a resource manager. Operating systems usually incorporates their own schedulers, and as such, causes user programs to run slower and less predictably in some instances. This could potentially cause the user program to not be able to meet hard deadlines.

I/O capabilities for a system indicates what kind of input and/or outputs it can receive and which platform it is available to communicate with.

The table below gives an overview of the specifications for some of the available devices that possibly could be used in the project, namely Arduino Uno, Arduino Mega, the Raspberry Pi 4, and the Lego Mindstorm NXT 2.0 and Lego EV3.

Name	Clock speed	Memory	OS	I/O	Price
Arduino Uno	16 KHz	32 KB	None	26 pins: 20 digital I/O 6 analog input	~150 DKK
Arduino Mega	16 KHz	256 KB	None	70 pins: 54 digital I/O 16 analog input	~260 DKK
Raspberry Pi 4	1500 KHz	1, 2 or 4 GB	Linux-based	40 digital I/O	339-499 DKK
Lego NXT	48 KHz	256 KB + 64 KB	Proprietary (Linux-based)	7 ports: 4 input 3 output	~1900 DKK (full kit)
Lego EV3	300 KHz	16 MB + 64 MB	Linux-based	8 ports: 4 input 4 output	1699 DKK (brick only)

Table 2.1: Table with data and specifications for different computational platforms [11, 12, 13, 14, 15].

As seen in Table 2.1, a Raspberry Pi seems to be a good platform candidate. It is very fast and has a lot of memory, removing the need for programs to be compressed, or optimised, to fit in memory. However, because of the accompanying operating system and the associated system processes, user programs can not fully utilise the CPU. Furthermore, the cache used by the Raspberry Pi, potentially allows for big differences between average- and worst-case execution times for programs running on the device.

The same problems are associated with the other platforms running operating systems, namely the LEGO NXT and EV3. The major difference between the LEGO devices and the Raspberry Pi is that they cost more to acquire, have less memory, and lower clock speed. However, the NXT allows for easier fulfilment of any real-time requirements, due to the simpler processor structure, with no scheduling, and less caching than the Raspberry Pi and EV3. Furthermore, both LEGO platforms have the benefits of built-in hardware for controlling motors, making designing a robot with them simpler.

The Arduino devices are the cheapest of all the platform candidates. However, they also have the the lowest clock speed and the smallest amounts of memory.

This potentially makes it difficult to fit an acceptable program, which deals with everything from motor functions to computer vision, into memory. However, if the computer vision is off-loaded to another processor, the Arduino would be sufficient - provided it only has to handle motor functions. Furthermore, Arduino devices, have the added benefit of only running a singular program on the CPU, making analysis of worst-case execution times easy.

2.4 Problem Formulation

As has been mentioned in the introduction, the purpose of this project is to create an autonomous training assistant that can play air hockey. To be able to play air hockey it must be able to detect, track and hit an air hockey puck. Each of these elements has been examined this chapter.

After considering and exploring each necessary concept to make a functional air hockey robot, an outline of what an autonomous air hockey robot entails are made. However, it is yet to be ascertained whether or not the explored concepts are sufficient to make a practical and functional robot. For this reason, a deeper look into the construction of an autonomous air hockey robot is explored.

How can an autonomous air hockey playing robot, based in the concepts of sensing, prediction, and reaction, be created, such that it can respond to arbitrary shots?

We will apply the presented concepts to an actual board, and determine whether or not the resulting solution will satisfy the problem statement.

2.4.1 Problem Scope Limitation

Since the project is limited by time and resources, we shall focus on making a robot that can block shots autonomously. This limitation does not include the ability to shoot the puck back and try to score. This can be fulfilled by allowing the mallet to move in a single dimension, instead of two.

Chapter 3

Design

3.1 System Requirements

To consider the project a success, we define some requirements for the robot to fulfil. These requirements are based in the concepts presented in Chapter 2.

1. The robot needs to be able to gather data about the position of the puck.
2. The robot needs to be able to make predictions towards the final position of the puck, based on the information gathered.
3. The robot needs to be able to process input and respond in a timely manner.

3.1.1 Performance Criteria

According to the problem statement, the robot must be able to respond reactively to an incoming shot. Therefore, we can lay out a number of performance questions, with the goal of determining how well the robot responds with respect to each of these. This allows us to precisely and reliably analyse the performance measures of each controller.

Block Rate: What is the percentage of shots the robot let through?

Average Distance: How far did the robot travel on average?

In order to put the performance measures on common ground, we define a performance function to quantitatively compare the controllers.

$$Utility = W_{block} \cdot BlockingRate + W_{distance} \cdot AverageDistance \quad (3.1)$$

Given we wish to reward a high block rate while punishing a high average travel distance, the function is based in these concepts. Naturally, we set W_{block} to 1, as this is the most desirable measure. Furthermore, because we wish to punish long travel distances, we assign the average distance travelled a negative weight. We deem a 0.5% decrease of the block rate, as being equivalent to allowing the average travel distance of the robot to increase by 1. This leaves us with an $W_{distance}$ of $\frac{-0.005}{1}$, and finally, the performance formula,

$$Utility = BlockingRate - \frac{0.005}{1} * AverageDistance \quad (3.2)$$

3.1.2 Dependability Requirements

It is necessary to also analyse how dependable the air hockey robot is, in addition to how well it fulfils game-critical functionalities. It matters little how well the robot can defend against a shot, if it only works a fraction of the time, and also endangers its surroundings. As such, we will evaluate attributes, means of finding and handling threats, as well as what a threat to the system means. This analysis will be based in the official dependability requirements set by TC56 of the International Electrotechnical Commission (IEC)[16].

Attributes	Means	Threats
Availability*	Fault Avoidance	Faults
Reliability	Fault Elimination	Errors
Safety*	Fault Tolerance	Failures
Confidentiality*	Fault Forecasting	
Integrity*		
Maintainability		

Note that criteria marked with an asterisk are of lesser importance to this project, and will not be analysed in-depth with regards to the project. They will, however, be given a cursory explanation as to why there were omitted in their respective segments.

3.1.2.1 Attributes

Within the scope of this project, attributes are defined as the basis for measuring the rates and types of failures encountered by the air hockey robot. These requirements provide a clear and concise ways measuring whether the finished product fulfils the criteria specified in Section 3.1.

Availability refers to the probability of the air hockey being operational within an expected time-frame. Due to the nature of this project, this attribute is of less significance as the air hockey table is a physical service that needs to be turned on.

Reliability refers to how well the system deals with running, for an arbitrary amount of time, without encountering any errors. The air hockey robot needs to be able to run unimpeded in a timespan corresponding to a training session. As there is no official definition for the timespan of a training session, within the scope of this project it has been set to 15 minutes. This was believed to be a reasonable amount of time one needs to practice before a potential break.

Confidentiality is one of two parameters that deal with how secure the air hockey system is, in regards to outside tampering. Confidentiality specifically handles security with regards to handling user-information. This requirement is of little relevance, as the air hockey robot holds no information outside of the system code.

Safety is a requirement meant to measure the danger-levels the air hockey robot poses to humans and environments during system-critical failures. As the air hockey robot is not a safety-critical system, this requirement of little importance. The reasoning for this is based in the fact that the worst-case scenario of the air hockey robot, would be for a puck to flip off the table, or for the robot to turn off because of an error.

Integrity like confidentiality, also helps measure how secure the air hockey system is. However, where confidentiality deals with how accessible information is, integrity deals with how easy it is to alter the code used in the air hockey system. It is worth acknowledging that it is possible for users to interfere with the system-code of the air hockey robot, but that doing so would render the product inoperable. As such, users that upload, or change, code are at their own fault of destroying the air hockey robot.

Maintainability refers to the how easy it is for users to repair and modify the air hockey robot and accompanying system. The system is designed with the intention of allowing users to easily repair broken parts.

3.1.2.2 Means

Means measure how well the air hockey system detects and handles failures related to erroneous states. This covers everything from examining the notion of what an error is, how to find them within the system, and ascertaining where they arise.

Fault Avoidance refers to the how well the system was tested, so as to avoid faults or mistakes to be implemented during the development. This is done by the use of peer-programming at regular intervals.

Fault Elimination measures the ability to detect and remove any program defects during development. This is done via simulation of a mathematical model within the UPPAAL model-checking software [17, 18].

Fault Tolerance deals with how prepared the air hockey system is, with regards to dealing with a problem. As such, tolerance measures the safety protocols set in place, that allow the program to keep running in spite of non-critical failures. An example of this could be a misalignment of the mallet. Theoretically the system system could use the camera to ascertain if this is the case, and if not, trigger a recalibration to ensure that the mallet is realigned.

Fault Forecasting deals with how well the air hockey robot accommodates unavoidable problems found while the robot is running. This could, for example, be from camera-noise. This noise could cause the supposed position of the puck to differ from the actual position on the air hockey rink. One way to deal with this, could be create a projected path by extrapolating a curve based off of the rink.

3.1.2.3 Threats

Threats are the problems that the system might encounter throughout execution. This covers the different levels of threats, being hypothesised, and actual instances, of errors, as well as the failures they cause within a system.

As an example, one such failure, within the air hockey robot, could be caused by a misalignment of the mallet. This fault, were it to become a reality, would lead to the system entering into an erroneous state where it makes wrong assumptions regarding position when moving the mallet. This in turn, would increase the risk of a puck passing the mallet and entering the goal-zone, thus failing to uphold the primary intent of the air hockey robot.

3.2 Physical Design of the Table

We will now discuss the decisions made regarding the physical design of the air hockey table based on options presented throughout Chapter 2 and the problem statement, with all its limitations applied. Furthermore, an overview will be given as to the final design of the robot. In addition to this, an explanation of each component will be provided, featuring their role in the system and their relation to one another.

3.2.1 Physical Design Decisions

Chapter 2 explored what machinery was needed for the air hockey robot to be able to gather information, process the information, and act based on the information was established. While multiple options were explored, a final decision needs to be made with respect to the problem statement and its limitations.

3.2.1.1 Picking a Detection Method

Provided a camera is placed at an angle and position where the entirety of the board is visible, the process of detecting the puck, would be relatively simple. The process would revolve around tracking a single object moving on a static, unchanging background. It will primarily be based in sensing the contrast in colour, with the intent of creating a grid that logs the position of the puck. Using sonar equipment could prove problematic, as sonars have an error-margin that increases the smaller an object becomes, paired with the fact that the mallet would be obscured by a human holding it.

However, as the colouration of the rink and puck are already known, it would be easy to pick up on this contrast. This means that utilising a regular camera to log the placement of the puck is ideal. The reasoning behind picking a regular camera over a infrared camera, or a camera that can detect UV-rays is that the playing field emits cold air to elevate the puck, which can obscure the vision of the puck should it be reach the temperature level of the rest of the board which would interfere with the way that an infrared camera works, and would result in the users hands being considered the warmest thing on the board. Additionally, coating the puck in UV-emitting material is not feasible because if the coating gets on the table it will interfere with the camera readings. For these reasons, the final decision is to utilise a regular camera, specifically a camera called a PixyCam. As this specific camera not only fulfils the requirement of being able to gather information, but also because it has the built in functionality of object detection.

3.2.1.2 Picking a Platform

Given the nature of the pacing found in air hockey, Arduino Uno has been deemed a sufficient platform. This is in part due to its lack of associated OS, which allows for complete control over the processor. This level of control, allows for the robot to avoid any potential latency-related problems related to the higher level control of platforms like the Rasperry Pi. While this feature is naturally shared with the Arduino Mega, the availability of the Arduino Uno meant that it was chosen in favor of the Arduino Mega.

3.2.1.3 Picking a Movement Method

Given the nature of the options presented to move the mallet about, the gantry approach has been deemed a reasonable solution. This mechanism is, in relation to the other methods, a simpler implementation which fulfils the same criteria. The gantry allows the best means to gaining free access to the limited playing field necessary to simply block shots, as the movement necessary has been limited to a single dimension.

3.2.2 Overview of the Table

The air hockey table is comprised of a camera (1) functioning as the vision, an arduino (2) functioning as the brain of the system, and finally a mallet mounted to a gantry (3), functioning as the hand.

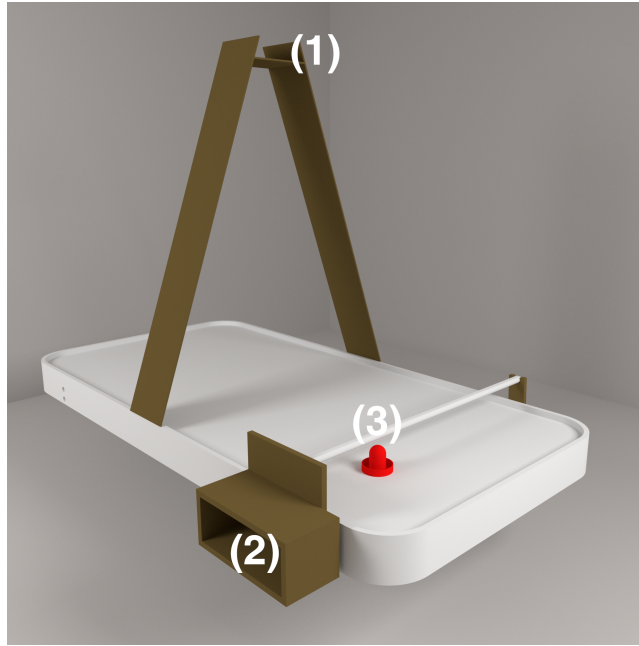


Figure 3.1: Overview of the Table

3.2.2.1 The Camera

The camera is the first component of the system, it allows the robot to gather information about its environment and therefore also the game state. As previously mentioned, the camera used is a PixyCam with built in object detection. This allows it to process the information it has gathered and send only the necessary information, the position of the puck, to the software mounted onto the platform. Which, in turn, allows the computations to be performed without any additional information gathering.

3.2.2.2 The Arduino

The arduino awaits information from the camera, whereafter it analyzes and processes the information in order to give commands to the mallet. Given the limitations enforced on the project the commands available are limited, as the mallet only operates in a single dimension. In order to give reasonable and quick commands to the mallet based on the information from the camera, the arduino is required to effectively and efficiently determine the final position of the puck, such that the mallet is able to respond in time.

3.2.2.3 The Mallet

The mallet is the mechanically active component. The mallet is limited to only moving in a single dimension as it only needs to block incoming shots. However, this is also sufficient in order to fulfil its role in the system. As a component, the mallet is a simple interface for the Arduino that allows it to act upon the world.

3.3 Modelling the System

In order to gain a deeper and more thorough understanding of the system presented in Section 3.2.2, the system is modelled with respect to each component of the system. This approach is based in the 10 fundamental steps of the Model-based design methodology described in [19]. However, due to the natural relationship between our models of computation and the hardware chosen, as well as the relationship between the control algorithm and hardware, the method has been altered to fit the needs of this project.

3.3.1 The Camera

In order to sense the puck, a camera which snapshots the board state every 20 milliseconds is mounted directly above the table. It is essential that the most important part of the playing area will be visible to the camera. The camera is placed approximately 85 centimetres above the board, this ensures the camera will monitor everything between the railings on the long side of the board, without any unnecessary vision outside the field. However, it also means a loss of ~20 centimetres visibility to the table with respect to goal railings (10 centimetres on each end). We deem this loss of vision acceptable, as players will usually hold the mallet that far or further from the goal. Additionally, with the camera closer to the playing surface, image noise is reduced, and object detection may become easier, as the puck will be occupying more pixels. As such, with the camera lens at around 85 centimetres above the table, we can see a sufficient part of the playing area. However, while using the PixyCam allows the robot to gather information about the outside world, the accuracy with which it does this can vary due to the imperfections of the equipment or the structural integrity of the system.

3.3.1.1 Camera Noise

In order to determine the level of error obtained within the received information from the camera, the uncertainty of the camera is tested. We conduct a simple test, where the puck is put directly beneath the camera as a stationary target. The camera then tries to collect data points relating to the position of the puck. The test reveals an expectancy of about one to two pixels offset from the original position.

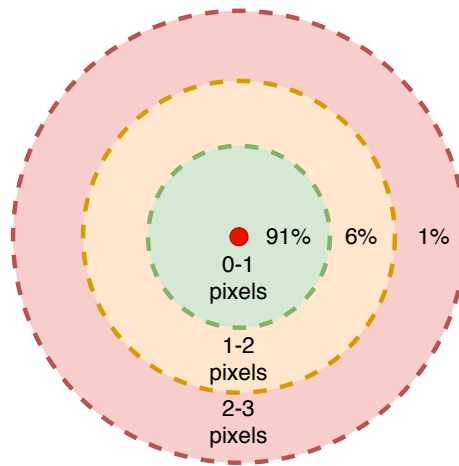


Figure 3.2: Visualisation of the puck detection noise

The test, shown in Figure 3.2 reveals an expectancy of about one to two pixels offset from the original position. This means an uncertainty which affect future predictions about the trajectory of the puck is present, whenever an image is processed.

Rather than letting the puck be represented as a single point, it can be more accurately represented as an area in which at least 50% of the puck will be represented within, naturally this is assuming the PixyCam attempts to identify the centre of the puck.

3.3.2 The Puck

In order to control the system in a way that can reliably and predictably interact with the game, it first needs to be familiar with the air hockey environment it needs to interact with. Many factors influence how an air hockey puck moves during the game but the three primary factors for puck movement are collision with mallets, collision with the rinks railings and the friction from moving on

the playing surface.

Reducing the friction between the table and the puck is the point of the air-cushion design of the air hockey table. The friction between the air and the puck is also taken to be very low because of the pucks flat design. Both these friction parameters will be ignored for this project, as they are assumed to have negligible effect on a shot.

The interaction between the mallet and the puck is another factor that will be ignored, as modelling this complex collision is considered outside the scope of this project and assumed to have insignificant impact on the controllers performance.

When we model the pucks behaviour, consideration will need to be taken with regards to the fact that many air hockey boards exist, all with different design materials, structure, and physical properties. Therefore it warrants the need to create an accurate model of the physical properties of the specific air hockey board.

3.3.2.1 The Top Speed of the Puck

In order to assess the time available to calculate the trajectory of the puck, as well as respond correctly, a limit should be set for the robot. In this case, the limit is given by speed of the puck, the faster it moves the less time the robot will have to respond. For this reason, it should be assessed how fast a human player can shoot the puck, where this speed will function as the baseline for response time. Naturally, if it can be assured that the robot can respond to the fastest shot, it suggests it can respond to any slower shot. To determine this top speed it was a simple case of looking at the fastest recorded shots, using the data where a player was attempting to hit the puck as fast as possible. The data is presented in the Figure 3.3.

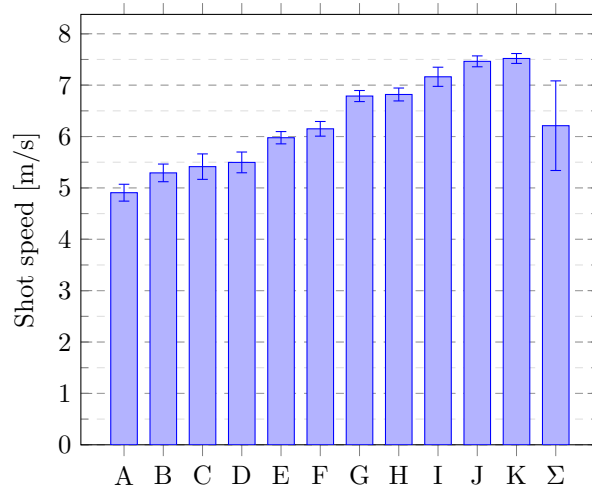


Figure 3.3: Measured speed of shots on an air hockey table with standard derivation imposed.

Here, each letter represents a single shot, while Σ denotes the average shot speed of all shots, while the arrows denote the sample standard deviation. According to the data, the highest achieved speed is about 7.6 m/s. Naturally, this includes uncertainties such as the strength of a player. Because humans vary in strength, it should be possible to create shots that are faster than the found highest speed. In order to compensate for this fact and the natural error margin in measuring, as well as human strength utilisation, the maximum expected speed the system should respond to, is set to 8m/s. It should be mentioned that other human players might be able to shoot the puck even faster, however for the purpose of this project 8 m/s is deemed sufficient as an upper limit.

3.3.2.2 Railing-Rebounds

To measure the behaviour between the puck and the air hockey rink railing, a simple experiment is conducted. A series of shots are taken aiming at an arbitrary point on the edge of the board, which results in a number of various velocities and angles going into the collision. The collisions are recorded at ~120 frames per second and manually analysed frame by frame. This gives the input- and output speed as well as angle of incidence and reflection. One possible source of noise in the data is human error arising from using the program, as placement of tracking points had to be done manually. The data is analysed to answer a number of questions:

- Is the collision close to perfectly elastic?
- Is the angle of incidence and angle of reflection proportionally related?

Elasticity we look at the velocity before and after the collision. to determine if there are any significant losses in the pucks velocity, from hitting the rink and if there are any losses does it depend on the angle of approach.

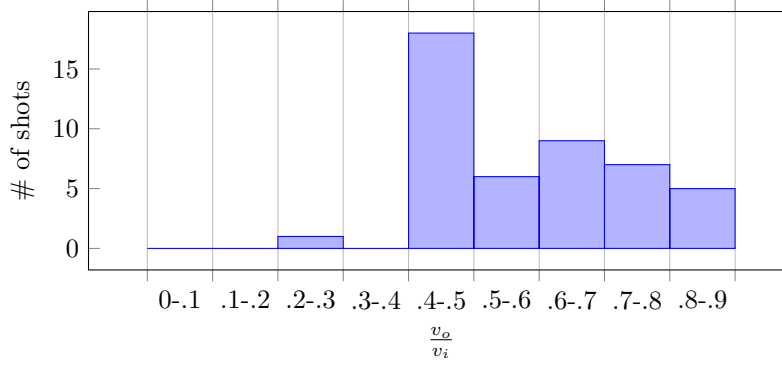


Figure 3.4: The distribution of the shots over how much of the velocity was retained after rebounding (The coefficient of restitution).

From Figure 3.4, it can be observed that loss occurs, and looking at the data shows between 7% and 70% loss of velocity after rebounding. The collision is therefore far from perfectly elastic. The span is enough to justify investigate other parameters that influences the loss of velocity.

Angle We hypothesise that the velocity perpendicular to the rail (v_y) is affected by the rebound and that the velocity parallel to the rail (v_x) is unaffected. The data from the shots is analysed, and by looking at the Pearson correlation of all the different variables of the shot, some of the strongest correlations are found to be between v_y^i and v_y^o at 0.937 and between v_x^i and v_x^o at 0.960 [Table A.1 in Appendix A]. indicates a relation $F : (\mathbb{R} \times \mathbb{R}) \rightarrow (\mathbb{R} \times \mathbb{R})$, however relations from v_x^i to v_y^o and from v_y^i to v_x^o shows low correlation, and relations between the ratio of v_y^i and v_x^i and v_y^o or v_x^o also have low correlations. Therefore we derive two functions: $reboundX(v_x^i) = v_x^o$ and $reboundY(v_y^i) = v_y^o$ by regressing a linear function to fit the measured data.

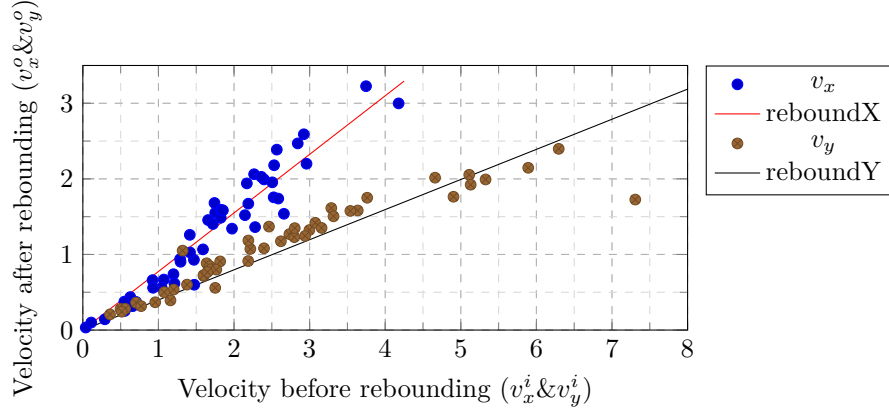


Figure 3.5: The v_x and v_y data points and 2 linear regressions for them.

As can be seen within Figure 3.5, by regressing the formula $f(x) = a * x$ to the two relations, and with a loss of $\sim 33\%$ from v_x^i to v_x^o , our hypothesis about an almost unchanged parallel velocity does not hold. This could be due to friction and velocity transferred to angular momentum. Our hypothesis about less loss in the perpendicular velocity does hold however. There is a roughly 60% reduction in the parallel velocity after a rebound.

3.3.3 The Mallet

As the mallet is the only way for the robot to interact with the game, an in-depth understanding of its movement options is required. Naturally, limiting the system to only blocking incoming shots simplifies this model strongly. However, knowing the general speed and options of the mallet is still essential to determine whether or not the system will be able to respond in time.

3.3.3.1 Mallet Movement

The mallet has limited movement options, due to its restriction to a single dimension. This creates an environment where the only options the mallet has is to await instructions to either move upwards or downwards.



Figure 3.6: The movement options available for the mallet.

While the movement options of the mallet, shown within Figure 3.6, help show whether or not it is possible to reach the final position of the puck, another aspect to consider is the speed of the mallet. Taking this into account enables calculations on whether we can reach the final position in time or not, rather than simply judging by physical obstacles.

3.3.3.2 Mallet Speed

The mallet is moved with a stepper motor, which works by being stepped forward or backward at a certain interval. The specific UPPAAL implementation of the mallet can be seen in Figure 3.7.

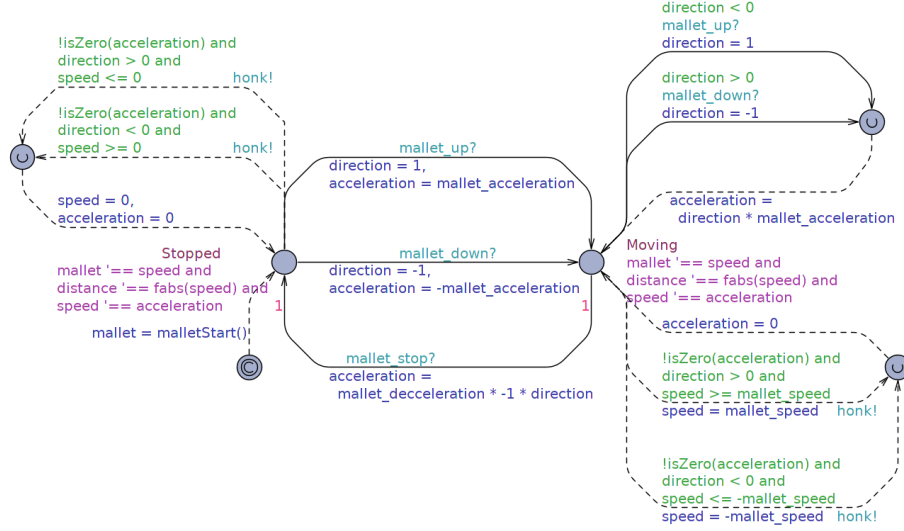


Figure 3.7: The timed-hybrid automata controlling the mallet

Aside from speed, the mallet also takes into account which direction it is moving. This is so that, if the mallet is moving in one direction towards the puck and suddenly has to move the other direction, the motor will first decelerate before reversing direction.

In the model, the mallet accelerates by use of a coefficient set between 1 and -1. 1 would be maximum acceleration, while -1 would be maximum deceleration. As needed, the coefficient can then be set between these values, until a desired speed is reached. At the desired speed, the coefficient can then be set to 0, and the mallet will maintain speed.

The max speed of the mallet is dependent on the minimum step period, along with the number of teeth on the pulley attached to the motor. There are 85 teeth, the pitch of the teeth is known to be 2mm, and the motor is known to take 200 steps per revolution, or rotate 1,8 degrees per step. The minimum step period was found experimentally to be 500 μ s. The max speed is then calculated to 1.42 $\frac{m}{s}$.

3.4 Controllers

Now that a thorough understanding of the system has been achieved, we can model and assess different methods to control the robot by using modelling software such as UPPAAL where we can input the models with different pa-

rameters such as mallet speed and puck speed. We start off by introducing a simple controller, namely the patrolling controller. The goal of this is to provide a baseline controller which can be modified. The controller will be described and simulated, after which the weaknesses will be identified with respect to the performance criteria presented in Section 3.1.1. In these simulations the speed of the puck will always be the top speed, described in Section 3.3.2.1. The results from this simulation forms the basis for a new controller which attempts to improve a specific aspect of the simulated controller. Similarly, the new controller will once again be described, simulated and expanded upon. This process is iteratively applied to the controllers.

3.4.1 Patrolling Controller

The patrolling controller simply moves upwards until it hits the top railing with no option to stop. Once it hits the top, it reverses direction and moves to the bottom. Once it hits the bottom it reverses direction again and repeats. The logic behind this simple controller can be seen illustrated in Section 3.4.1.

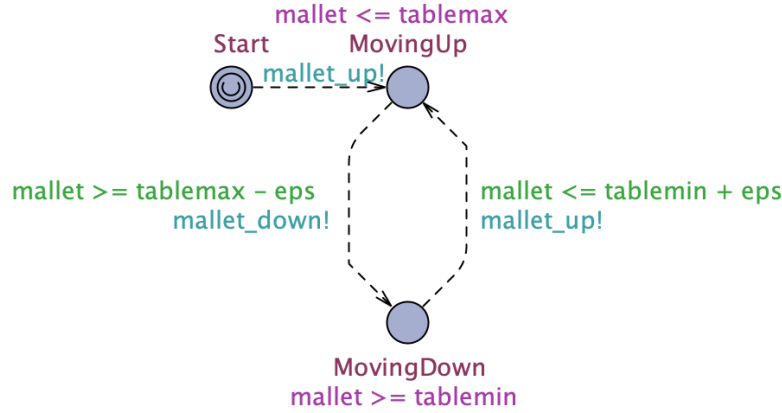


Figure 3.8: The patrolling controller as a hybrid time automaton

We expect this controller to catch a minimal number of pucks, while featuring a high travel distance due to its inability to halt its movement. The results from the simulations can be seen in Table 3.1.

Controller	Block %	Average Distance	Performance
Patrolling	12.9	29.885 ± 7.882	-0.020

Table 3.1: Simulation results for the patrolling controller

As expected, the moving mallet showcases a minimal block rate and a maximal travel distance, resulting in a low performance rating. In order to attempt to increase the performance, we introduce a controller which has the added functionality of moving the mallet with respect to the position of the puck.

3.4.2 Two-step Controller

A two-step, or Bang Bang, controller is a simple feedback controller that can switch between two actions, in this case move up or move down with respect to a simple prediction, this is illustrated in Figure 3.9. The two-step controller functions by utilising a logical expression, which assists in discerning which action to perform.

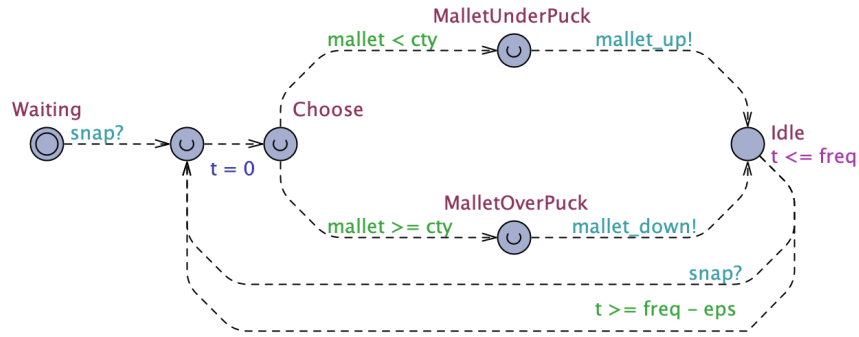


Figure 3.9: The two-step controller as a hybrid time automaton

Due to the logical expression determining its behaviour, the prediction of the two-step controller needs only a single image to determine a supposed position. This process is handled multiple times, as the controller repeatedly tries to predict the final position of the puck. With the goal being to block an incoming air hockey shot, the natural logical expression would be to evaluate whether or

not the position of the puck is above or below the current position of the mallet. This leads to a simple prediction where it is assumed the puck only moves in one dimension.

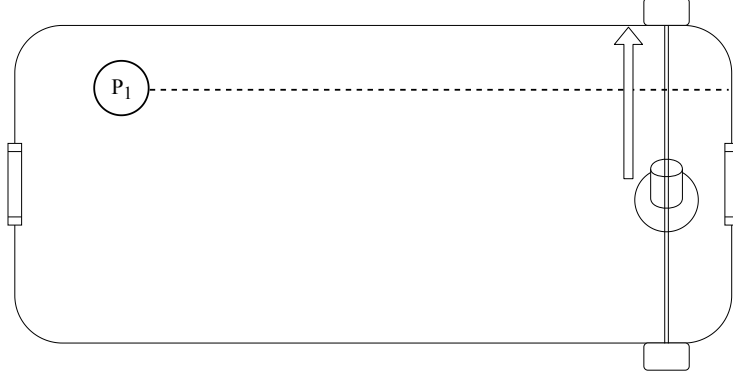


Figure 3.10: A prediction made by the two-step controller.

As can be seen in Figure 3.10, the controller assumes only simple trajectories perpendicular to the mallet axis. Therefore, it is able to make a simple prediction entirely on the height difference between the puck and the mallet. Then, once new information about the puck is gained, a new prediction will be made. Given that the two-step controller never considers the possibility of the puck ricocheting off of the railing, or has a positional history log which a shot can invalidate, it disregards the path problem by disregarding the possible origin methods of the path problem. This naturally means that the two-step controller instructs the mallet to 'chase' the puck, which suggests it might be weak to non-perpendicular shots if the mallet cannot move fast enough. However, as the two-step controller possess the ability to make predictions about the final position of the puck, and move the mallet with respect to this prediction, we expect the two-step controller to feature a higher block rate than what was seen from the patrolling controller. However, as it still shares the weakness of not being able to halt its movement the average travel distance is expected to remain similar. The results from the simulations can be seen in Table 3.2.

Controller	Block %	Average Distance	Performance
Two-step	47.8	26.055 ± 7.914	0.348

Table 3.2: Simulation results for the two-step controller

The two-step controller features a higher block rate, but at the cost of an average travel distance similar to the patrolling controller. Furthermore, it features a performance score of 0.348 which is an increase from the previous controller.

However, it still has a high movement measure. This is due to its inability to stop moving, meaning while it, unlike the patrolling controller, does move with respect to a prediction, it still shares the necessity to constantly move even while at the correct position. In the case of a completely straight shot, the mallet will naturally move away from the correct position, only to move back after receiving information on the puck again. Furthermore, should the shot ricochet off the railing, it means the mallet will chase the puck all the way to the railing, only to move back to a position the mallet most likely passed, or could have gotten to faster by moving there directly.

3.4.3 Three-Step Controller

The two-step controller has no way of lowering its distance travelled due to its inability to halt movement. To reduce the necessary movement, we present the three-step controller. This controller incorporates the same set of actions as the two-step controller, but with an added third action. As can be seen within Figure 3.11, the addition of a third action allows for a new mallet state which halts movement if the puck and mallet positions align on the same y-coordinate.

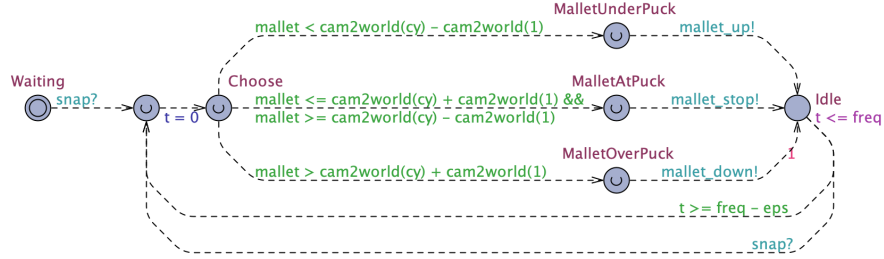


Figure 3.11: The three-step controller as a hybrid time automaton

Essentially, the three-step controller utilises the same prediction model as the two-step controller, meaning it naturally inherits the same prediction weaknesses as the two-step controller. However, a check is also made to determine whether or not the current position is correct, with respect to the prediction, and if it is, the mallet does not move. This added action can be seen in Figure 3.12 together with its prediction model.

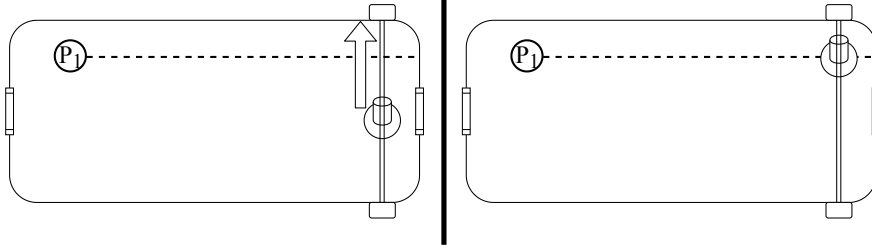


Figure 3.12: The prediction model and actions available to the three-step controller

That is, until it has gained information that the puck is either above or below the mallet, where it will once again mimic the behaviour of the two-step controller until it determines it is in the correct position. As the two- and three-step controllers feature the same prediction model, it is expected that their block rates will be similar. However, with the added functionality to halt movement present in the three-step controller the average travel distance should lessen. The results from the simulations can be seen in Table 3.3.

Controller	Block %	Average Distance	Performance
Three-step	47.9	24.720 ± 8.478	0.355

Table 3.3: Simulation results for the three-step controller

The three-step controller follows many of the same decisions as the two-step controller, thus giving it a similar performance. However, while the controller is able to have the mallet stop moving, the cases in which it will do this, is limited. This is because the puck usually moves in both dimensions available. Therefore while the mallet may stand in the correct position based on its current knowledge of the puck position, it is likely the puck will be moving to a different y-coordinate, which warrants movement from the mallet. So even with the added functionality of stopping the mallet, its usage is too rare because of the inaccuracy of the predictions.

3.4.4 Naive Linear Extrapolation Controller

If the controller is given a small amount of memory, the unreliability of the predictions of the three-step controller can be reduced, as the puck position data can be saved for a few frames. With this, a controller can be extended to perform linear extrapolation, as seen featured in Figure 3.13.

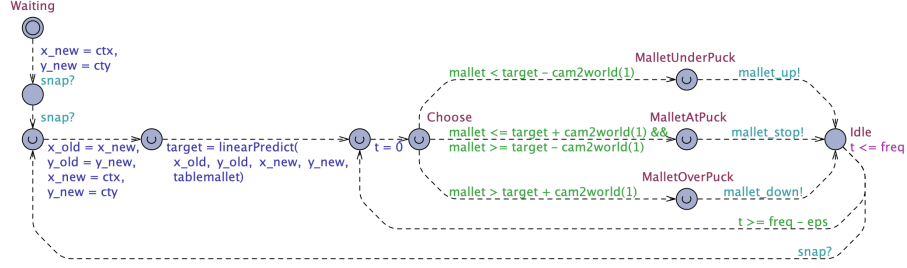


Figure 3.13: The NLEC as a hybrid time automaton

We categorise these types of controllers as linear extrapolation controllers (LEC). Using a LEC means the position of the puck crossing the axis of the mallet can be found as the puck has shown to follow a linear trajectory, see Section 3.3.2. It should be noted however, that the naive LEC (NLEC) naively expects the puck to not have bounced, and to not bounce in the future. This results in a simple LEC which disregards the railing of the board.

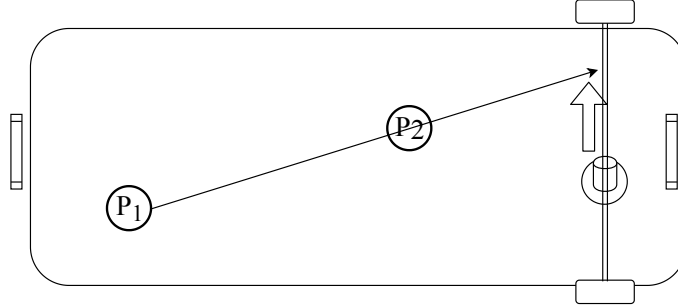


Figure 3.14: NLEC prediction based on two points.

The prediction model, shown in Figure 3.14, considers the possibility of the puck moving in two dimensions, the accuracy of the predictions are expected to be higher than the previously mentioned controllers, which in turn should improve the block rate. The results from the simulations can be seen in Table 3.4.

Controller	Block %	Average Distance	Performance
NLEC	59.1	20.888 ± 7.733	0.487

Table 3.4: Simulation results for the NLEC

We observe an 9.2% increase in the block rate compared to the three-step con-

troller. However, the NLEC is still prone to make erroneous predictions due to ricocheting. This means the predicted position can be incorrect, due to the puck having bounced between the images, and/or because the puck will bounce in the future, which the NLEC will predict as being placed outside of the playing field. Furthermore, as the NLEC has a history of the puck positions, it means it is susceptible to shots from the player which can invalidate the history. However, as the linear extrapolation is repeatedly applied, an invalidated history can be compensated for at the expense of the reaction time available.

3.4.5 Rebounding Linear Extrapolation Controller

As an increase in the reliability of the controller illustrated an increase in performance, we attempt to improve the prediction model of the NLEC. As mentioned before, the NLEC is unable to consider the influence of the railings. The rebounding linear extrapolation controller (RLEC) is a LEC which considers the effect the railing can have on the trajectory of the puck in the future.

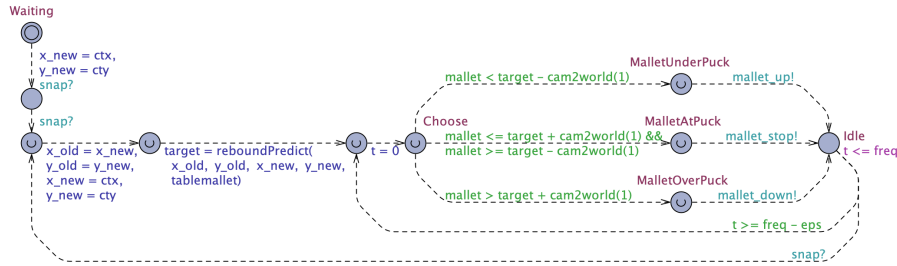


Figure 3.15: The RLEC as a hybrid time automaton

Figure 3.15 shows the logical process the RLEC undergoes in order to react appropriately according to its prediction.

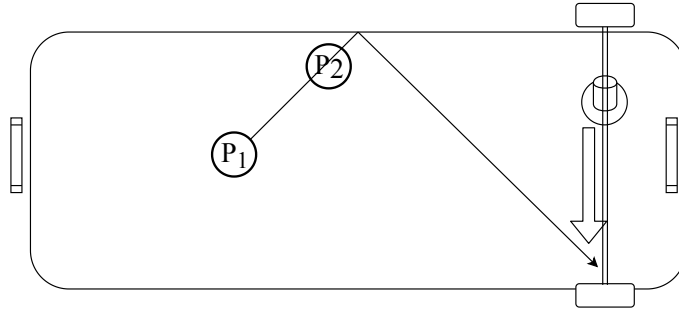


Figure 3.16: RLEC prediction based on two points.

Where the NLEC assumes no bounces will happen in the future, the RLEC will alter the predicted trajectory of the puck in accordance with the possible effects the railings can infer as shown in Figure 3.16. The altered trajectory is with respect to the modelled puck behaviour presented in Section 3.3.2.

Naturally, this means the RLEC is expected to predict the final position of the puck with higher accuracy than the NLEC, as it considers an additional attribute of the board. Therefore, the RLEC should feature a higher block rate than the NLEC. Furthermore, as the RLEC should be more reliable in its predictions, the travel distance is also expected to lessen as less unnecessary movement is done due to faulty predictions. The results from the simulations can be seen in Table 3.5.

Controller	Block %	Average Distance	Performance
RLEC	65.6	19.866 ± 7.318	0.557

Table 3.5: Simulation results for the RLEC

As expected, the RLEC improves on the average travel distance, and the block rate. The RLEC handles the problem with the NLEC by taking rebounds into account and should be able to calculate a path from the two initial pictures. However, while the RLEC considers the possibility of the puck bouncing in the future, it still does not consider the possibility that the puck bounces prior to receiving the second picture. Naturally, as no changes were made to the positional log of the puck, the NLEC may also reduce its reaction time in order to compensate for a invalid log.

3.4.6 Bi Rebounding Linear Extrapolation Controller

In a similar fashion to how the RLEC expanded on the NLEC, the bi rebounding linear extrapolation controller (Bi-RLEC) expands on the RLEC by considering the possibility of the puck bouncing between the images received. This is done by increasing the memory capacity to include an additional image, which should assist in determining the correct trajectory of the puck. As the path problem highlighted, see Section 2.2.2, there exists a multitude of ways the puck can be mapped from one position to another with the help of the railings. However, a lot of these possible trajectories can be disregarded due to the speed limitations of the puck, as well as the rate the PixyCam captures information. Essentially, given the 20ms window where the PixyCam does not capture the board, and the speed limitation of the puck of 8 m/s, specific railings may be impossible to hit before the PixyCam captures an image again. Furthermore, the goal railings can be disregarded as any abrupt changes in the x-axis of the puck does not warrant a change in the position of the mallet. This simplification allows the

Bi-RLEC to evaluate whether or not the puck has ricocheted between any of the received images. This is done by applying linear extrapolation twice, once where the first image received is left out of the extrapolation, and once where the third image is left out of the extrapolation. This process can be seen in Figure 3.17.

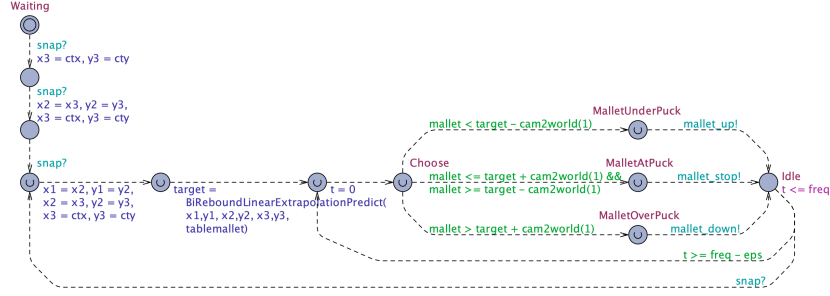


Figure 3.17: UPPAAL Model of the Bi-RLEC

Here the target is calculated through applying two linear extrapolations like this, together with expecting a maximum of a single bounce due to the limitations of speed and the camera capture rate, means there are three possible outcomes.

1. Both linear extrapolations successfully connect all puck positions. In this case, the puck did not ricochet off the railing at any point.
2. Only the linear extrapolation between the first and second puck positions successfully connect all puck positions. In this case, the puck ricocheted between image two and three.
3. Only the linear extrapolation between the third and second puck positions successfully connect all puck positions. In this case, the puck ricocheted between image one and two.

We disregard the possibility of both linear extrapolations failing as this would require the puck to have hit the railings twice, which given the assumptions will not be considered.

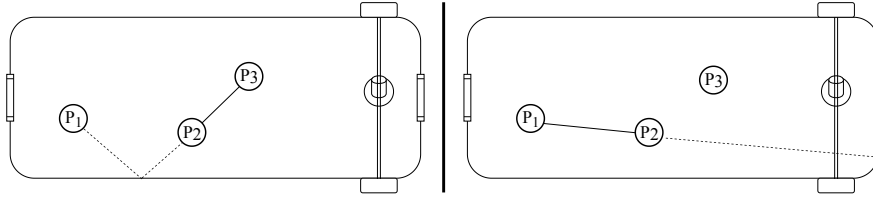


Figure 3.18: An illustration of the Bi-RLEC attempting to determine whether or not the puck hit the railing

Figure 3.18 showcases a scenario where the Bi-RLEC correctly determines the trajectory of the puck in spite of the influence of the railing. The Bi-RLEC notes that linearly extrapolating between the first and second puck positions, results in a trajectory which does not connect with the third puck position. On the contrary, linearly extrapolating between the third and second puck positions, results in a trajectory connecting all three puck positions. Therefore, the correct trajectory must be the lines created by the puck positions given in the third and second image. It should be noted however, that due to the noise from the camera, it is very likely that extrapolations will fail align all puck positions, in cases where they otherwise would have in case of no noise from the camera. In order to compensate for this, we simply select the extrapolation where the omitted puck position has the lowest distance to the drawn trajectory, if the distances are equal the trajectory is predicted from the latest puck positions used, as shown in Figure 3.19.

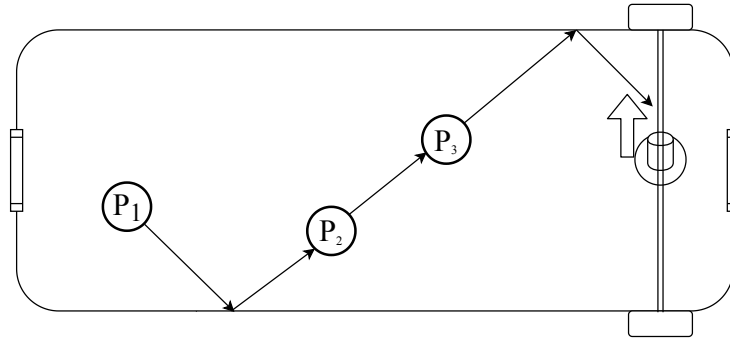


Figure 3.19: Bi-RLEC prediction based on three points

The Bi-RLEC attempts to predict the final position of the puck with even greater precision than the aforementioned LECs. The path can now be predicted more reliably regardless of when, where, and if the puck ricochet off the railing. Essentially, this nullifies the potential effects the railings can introduce to the puck. For this reason, we expect the Bi-RLEC to have the lowest travel distance of the presented controllers, as it should theoretically never fail to move to the

correct position once it starts operating. In addition to this, the block rate is expected to increase as well due to the increased reliability of the predictions, assuming the mallet has enough response time to move. The results from the simulations can be seen in Table 3.6

Controller	Block %	Average Distance	Performance
Bi-RLEC	60.4	17.137 ± 7.308	0.518

Table 3.6: Simulation results for the Bi-RLEC

As expected, the Bi-RLEC travels the least distance on average of the presented controllers. However, this increased precision comes at the price of now requiring three images before making a move, meaning the mallet will idle longer. The lowered block rate from the RLEC is likely due to this increased idle time. Essentially, the mallet is unable to catch the puck in time, due to the speed limitations of the mallet, despite following a correct prediction.

3.4.7 Combining Controllers

The presented controllers can be categorised by the number of pictures they require to begin operating the mallet. As such, the two-step and three-step controllers are mono-image controllers, the LECs are bi-image controllers, with the exception of the Bi-RLEC which is the only showcased tri-image controller. With these categories, combinations can be made across the controllers. For instance, while the Bi-RLEC awaits the necessary three images, it will already operate the mallet with the first image received using the three-step controller. After receiving the second image, it opens up for the possibility of using the RLEC. Finally, after receiving the third, and remaining images, the Bi-RLEC can be utilised. This simply means that whenever a new image is received, the controller switches, and therefore all controllers still have their individual logic. An example of combining controllers in this manner can be seen in Figure A.3.

The combination of controllers allow for the creation of controllers which become more precise the more information is given, whilst operating as soon as possible. This would for instance mean, that the imprecision of the prediction of the three-step controller would be nullified by the prediction of the Bi-RLEC once enough information has been gathered. Naturally, it also means some of the weaknesses will be inherited. For instance, the Bi-RLEC is able to reliably assess the trajectory of the puck with regards to the railings, allowing it to perform very efficiently should the initial prediction remain true. This is because, if the initial prediction of the Bi-RLEC is correct, and all subsequent predictions are correct, the mallet is required to move only once. However, if combined with the two-step controller it is certain to become more inefficient due to the

need for constant movement from the two-step controller. Furthermore, in case an earlier controller made an incorrect puck prediction, it could result in the mallet having to move further, with respect to prediction of the Bi-RLEC, as the predictions have different levels of reliability.

The specific combined controllers are the following: Three-step into NLEC, three-step into RLEC, RLEC into Bi-RLEC, and three-step into RLEC into Bi-RLEC. As aforementioned, while some of the strength of these controllers cover for the weaknesses of others, they will weaknesses which were not present before. For this reason, we test the performance of the combined controllers to see how they fair up against their individual counterparts. The simulation results from these can be seen in Table 3.7.

Controller	Block %	Average Distance	Performance
Three-step \rightarrow RLEC \rightarrow Bi-RLEC	67.4	21.645 ± 7.674	0.566
RLEC \rightarrow Bi-RLEC	65.7	19.063 ± 7.644	0.562
Three-step \rightarrow RLEC	67.2	22.441 ± 7.357	0.560
Three-step \rightarrow NLEC	61.9	23.458 ± 7.789	0.502

Table 3.7: Simulation results for the combination controllers

Some controllers perform better than their individual counterparts, while some perform worse. The three-step into RLEC into Bi-RLEC has the best performance of the combined controllers. The controllers that did not improve performance may have been unable to overcome the faults of their parts.

3.4.8 Picking a Controller

The presented controllers have had their performance rated, following the formula presented in Section 3.1.1. We will now compare all the controllers to each other, given their performance data.

Controller	Block %	Average Distance	Performance
Three-step \rightarrow RLEC \rightarrow Bi-RLEC	67.4	21.645 ± 7.674	0.566
RLEC \rightarrow Bi-RLEC	65.7	19.063 ± 7.644	0.562
Three-step \rightarrow RLEC	67.2	22.441 ± 7.357	0.560
RLEC	65.6	19.866 ± 7.318	0.557
Bi-RLEC	60.4	17.137 ± 7.308	0.518
Three-step \rightarrow NLEC	61.9	23.458 ± 7.789	0.502
NLEC	59.1	20.888 ± 7.733	0.487
Three-step	47.9	24.720 ± 8.478	0.355
Two-step	47.8	26.055 ± 7.914	0.348
Patrolling	12.9	29.885 ± 7.882	-0.020

Table 3.8: Controller performance

As can be seen Table 3.8, the patrolling controller score the lowest. Similarly, the two- and three-step controllers score lower than the other controllers, due to their simplistic predictions. The rest of the controllers show incremental improvement, though some individual controllers perform better than controllers where they are combined with others. Given the results, the chosen controllers that will be used are RLEC into Bi-RLEC, three-step into RLEC, and three-step into RLEC into Bi-RLEC.

3.5 Robustness Analysis

As we now have an understanding of the PixyCam, Mallet and accompanying models, a robustness analysis is to be conducted to ascertain how certain changes might affect the performance of the air hockey robot.

3.5.1 Modifying the Camera Rate

The Original PixyCam has a set speed with which it takes pictures. In the interest of seeing how a faster camera affects the overall performance, an experiment is conducted. This experiment measures the performance of the three controllers that were picked in Section 3.4.8. This experiment will test how a camera operating at twice the speed of the used PixyCam (100 frames per second) will affect the performance of each controller.

Controller	Block Rate (%)	Distance (cm)	Performance
Three-step \rightarrow RLEC	69	24.53 ± 7.24	0.57
RLEC \rightarrow BI-RLEC	68	22.98 ± 7.39	0.57
Three Step \rightarrow RLEC \rightarrow BI-RLEC	69	24.22 ± 7.38	0.57

Table 3.9: Controller performance with a camera two times faster than the PixyCam

As seen in Table 3.9, an improved camera might increase the efficiency of the controllers. Block-rate will increase roughly two to three percentage points, depending on the controller. However, the increased camera speed, will also result in an increase in the distance travelled by the mallet. The reasoning for this is based in the way distance is measured in the model. As it is designed to include noise, the controller will keep moving around the approximate area of the pucks assumed position. This increases the distance as it never reaches the destined coordinate, but rather circles in orbit around it, thus remaining ever in motion. However, this projected increase is not be enough to be deemed disruptive to the performance of the controllers.

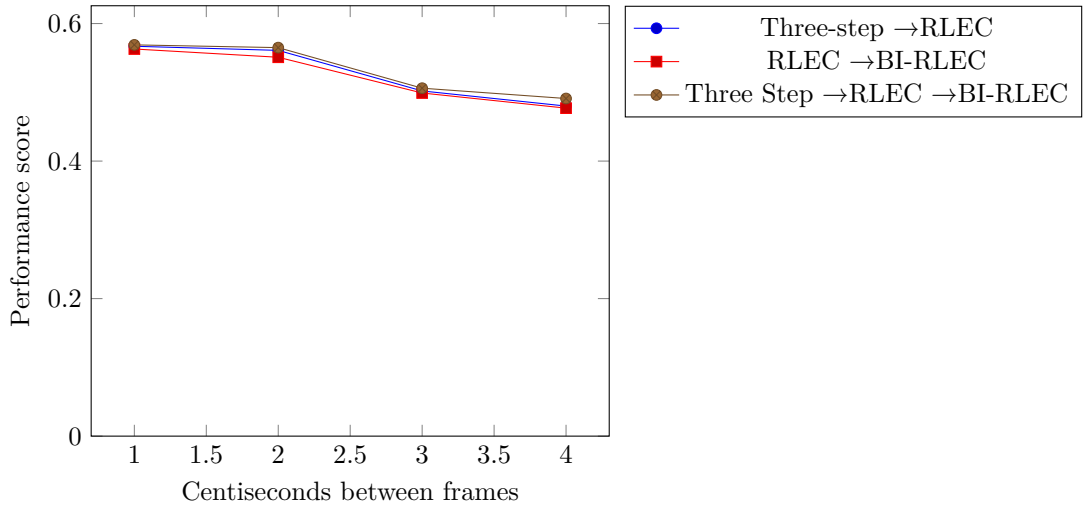


Figure 3.20: Performance score for a slower camera

It is also interesting to see what would happen if the robot utilised a slower camera than the current PixyCam. To test this, an experiment where the camera takes pictures at half rate is also simulated.

Controller	Block Rate (%)	Distance (cm)	Performance
Three-step \rightarrow RLEC	61	20.79 ± 7.13	0.51
RLEC \rightarrow BI-RLEC	58	16.32 ± 7.34	0.50
Three Step \rightarrow RLEC \rightarrow BI-RLEC	61	19.97 ± 7.47	0.51

Table 3.10: Controller performance with a camera half the speed of the PixyCam

As can be seen in the table above Table 3.10, with a camera that is 50% slower at handling the pictures the average performance decreases with around 5-6 percentage points, it is also interesting that the amount of travel decreased with about 2 cm on average. This is probably because the mallet can only hit shoots closer to it.

3.5.2 Modifying the Camera Faultiness

It is assumed that the PixyCams image detection algorithm provides a coordinate for every frame, but it can be interesting to see how the controllers performances change if there is a loss of frames. An experiment is conducted where differing amounts of the pictures processed by the camera does not result in a coordinate. The results can be seen in Figure 3.21, and point to an almost linear correlation between the amount of lost coordinates and the performances of the controllers.

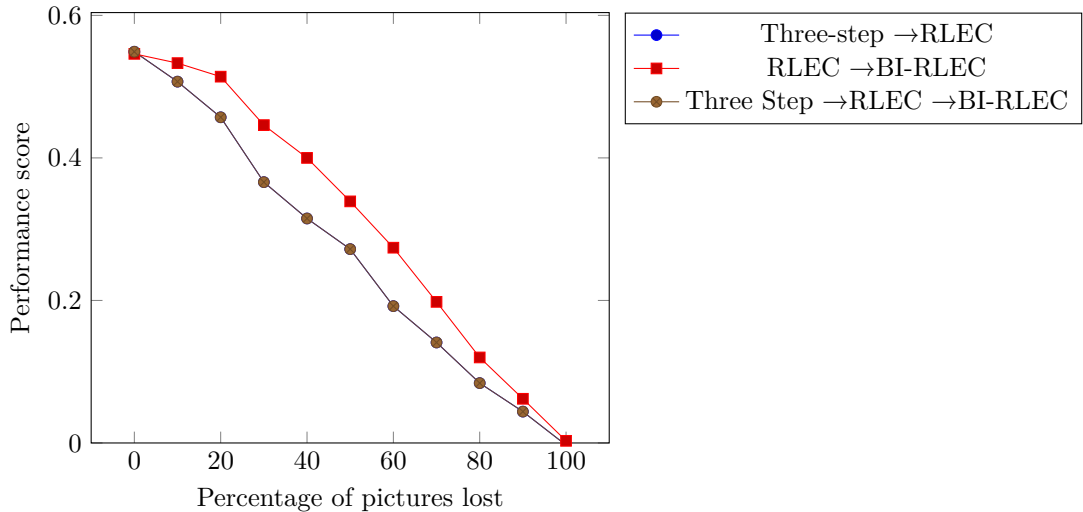


Figure 3.21: Faulty camera reports

The interesting thing is that the combined controllers that start with the three-step controller normally perform better, but in these simulations, they start performing worse. We consider the three-step controllers normal affect on distance travelled, but if we look at Figure 3.22, we can see that the distance are close to the original simulations. Furthermore, Figure 3.22 shows the same change in catch-rate as in performance.

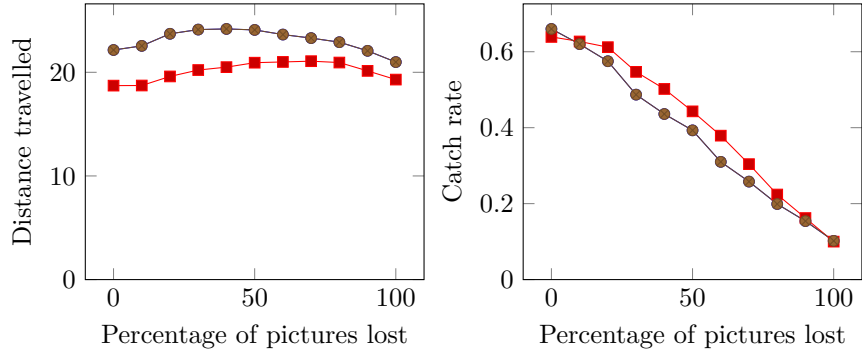


Figure 3.22: The catch rate and distance travelled with lossy cameras

We show a situation in Figure 3.23 that might be the reason for this. If an initial position showing the puck as being above the mallet is received, the three-step controller will begin moving as it is a mono-image controller, while a LEC will wait for the second picture. As can be seen in the diagram, the puck will end up at the bottom, and the three-step will start out moving the mallet away from the final position of the puck, thus decreasing the chance that the controller will be able to catch it. This may not be the sole reason for the difference in catch-rate, but it might be one explanation.

3.5.3 Modifying the Mallet Speed

In addition to the camera, another important part of the robot is the mallet. Alterations to the mallet might change the pacing of the robot drastically.

The primary component of the air hockey robot is the stepper motor that moves the mallet back and forth. The pacing of the robot dictates how well the robot deals with pucks at varying speeds. As such, an experiment will be conducted, that tests the efficacy of the robot at twice, and thrice the speed, as well as at half speed.

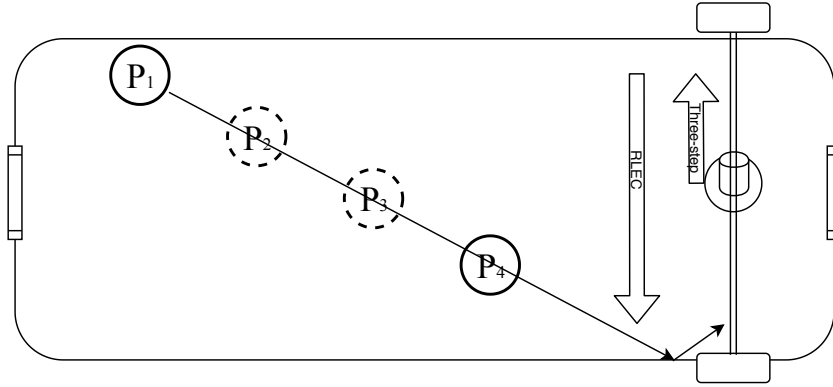


Figure 3.23: The three-step controllers initial reaction with missed positions

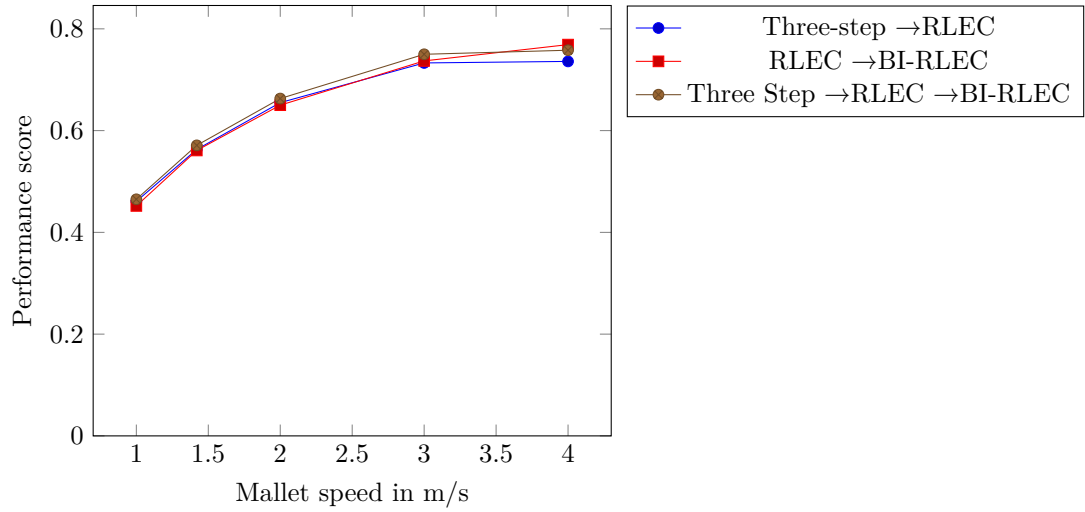


Figure 3.24: A graph showing the performance of the three selected controllers.

As can be seen in Figure 3.24, increasing the mallet speed from the normal $\sim 1,5$ m/s will seemingly increase the performance significantly, up to a cap. As mallet speed approaches four m/s and beyond, the specific performance increase per speed increase decreases. Similarly, decreasing the speed decreases the performance, as expected.

3.5.4 Modifying the Puck Speed

As was originally deliberated on in Section 3.3.2.1, the maximal speed achievable was eight metres per second. However, it is possible that the puck might reach faster speeds if a stronger individual were to play. The same can be said about the reverse, being a puck moving at lower speeds. As such, an experiment that analyses controller efficiency in situations with varying puck speeds.

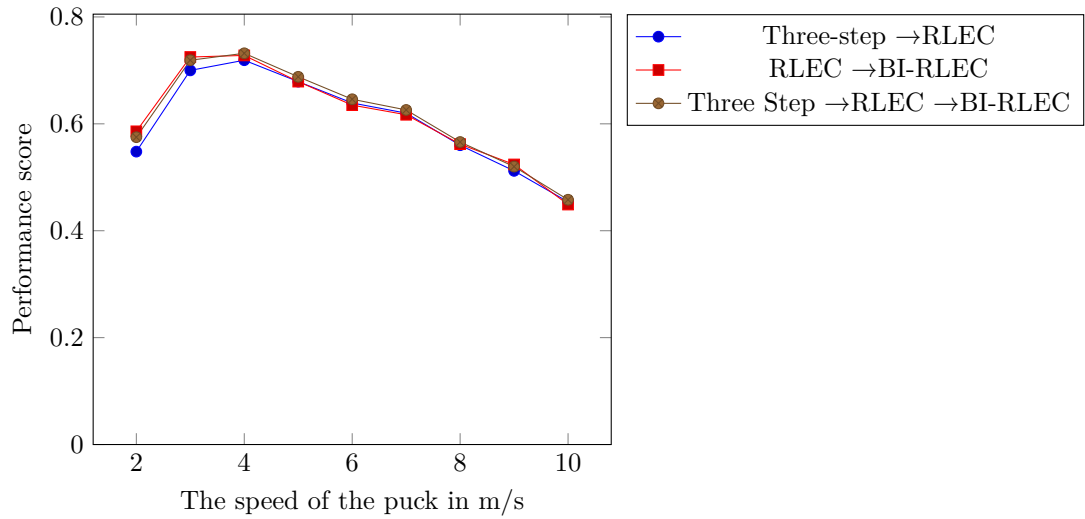


Figure 3.25: The correlation between the speed of the puck and the performance of the selected controllers.

As can be seen in Figure 3.25 the performance decreases as puck speed increases, and increases as puck speed decreases. However, it is of much interest to note that performance decreases significantly from 3m/s to 2m/s. This is most likely because of the noise added in the model. In essence, as two 'clouds' of uncertainty move closer together, the variance between the two actual points becomes greater.

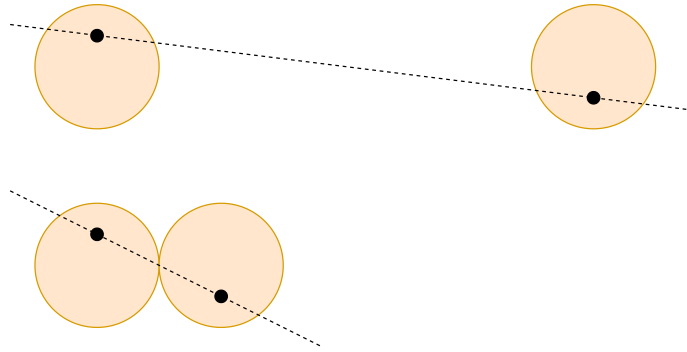


Figure 3.26: Two pairs of uncertainty clouds with reported points.

As a result of this, the predicted mallet-axis crossing of the puck may vary wildly over the course of the shot. Given the low speed, these variances would also cause the mallet to move many times, leading to a lower performance score because of the movement penalty.

Chapter 4

Implementation

4.1 Controlling the Motor

Controlling the stepper motor is integral to creating a functioning air hockey machine. The way the stepper is controlled is by pulsing the motor to move it a step in a direction. The interval between the pulses then controls the speed of the motor as a step is always the same size. The motor in the project can be stepped at a rate of once per 300 μ s, which results in a speed of 1.42m/s of the mallet.

4.2 Controlling the Camera

Controlling the camera is vital, as it is the key component with regards to gathering information about the position. The air hockey robot employs a PixyCam One and its built-in library to snapshot the air hockey playing area. Additionally, the PixyCam utilises built-in object recognition functionality that is calibrated to a colour to keep track of the puck. When the PixyCam takes a photo, it takes note of pre-defined colour objects that sticks out from the background. It turns them into blocks that correspond to a set of coordinates. Said coordinates can then be used to calculate the trajectory of the puck. This in turn, enables the mallet to start reactively moving to a position that corresponds to the same latitude as where it assumes the puck will be. This process is repeatedly conducted as new snapshots are registered. The information that is gathered by the PixyCam is transferred to the Arduino board via a serial peripheral interface (SPI).

The PixyCam is calibrated to take a picture every 20 milliseconds and perform its own object detection on it. The blocks are transferred to the Arduino board with a data rate of 1 Mbit/s. With a block size of 2 bytes, the transfer time is approximately 16 μ s per block.

It is worth noting that the PixyCam has limitations both by design and wear. While the PixyCam have a built-in object recognition functionality, it is of simple design, and has trouble pinpointing advanced shapes. This coupled with a lack of contrast or colour, means that objects that need to be detected, has to conform to some semblance of basic shape, in addition to being of a bright colour contrasting with the background [20]. Additionally, the specific PixyCam used, shows wear from use, making certain parts of the snapshot process inaccurate. However, the degree of inaccuracy is not enough to deem it unusable.

4.3 Scheduling

Now that we have a list of execution costs for the different parts of the implementation, we present an overview of how it all fits together. As can be seen in Table 4.1, there is an immediate problem to a simple scheduling of these parts, namely that all three controllers have a cost that can exceed the minimum periods between stepping the motor. We can solve this by splitting the controllers up into parts that will fit in-between the stepping, though this will require that a form of context switching is implemented.

Task	Execution time	Period	Preemptable?
Step motor	44	≥ 300	No
Get camera data	228	20000	No
Three Step \rightarrow RLEC	<i>leq</i> 20000		Yes
RLEC \rightarrow BI-RLEC	<i>leq</i> 20000		Yes
Three Step \rightarrow RLEC \rightarrow BI-RLEC	<i>leq</i> 20000		Yes

Table 4.1: Table of tasks and their related times. All time measurements are in μ s

A consideration we should make is whether or not snapping pictures with the camera and predicting the trajectory are considered separate tasks. Since the individual outputs of the all three chosen predictors will not change unless the input puck positions change, The predictors do not have periods. As such, it does not make sense to run any of the predictors multiple times between receiving pictures, as this would simply return the same output. Furthermore, it is an advantage to run them as soon as possible after receiving a picture, so we can minimise the delay between sensing and reaction. Therefore, the scheduler

is created with just two tasks specified, the Sensing task and the Reaction task. The Sensing task is comprised of acquiring a position from the camera, and predicting where the puck will end up. The Reaction task is comprised of processing the current position of the mallet, and determining the direction to move with respect to the prediction made. Naturally, this implies the Reaction task has control over the stepper motor.

The Sensing task is split up into parts with costs less than 400 μ s. This means we can now simplify Table 4.1 as illustrated in Table 4.2.

Task Name	Period (T)	Cost (C)	Max part cost (PC)
Sensing	$\leq 20\,000$	$\leq 16\,000$	400
Reaction	≥ 300	50 μ s	50 μ s

Table 4.2: A simpler overview of the tasks

Consider the following example where we use a static period of 500 μ s for the Reaction task. During a 20 000 μ s cycle, we wish to run the Sensing task once, and execute the Reaction task 40 times. If we start at time 0 and execute the Reaction task, 50 μ s will have passed. Which means 450 μ s is left. This is enough time for one of the Sensing parts so we execute it, which leaves us with 50 μ s. 50 μ s is not enough for any task, so we wait until the next Reaction task. This pattern will repeat until there is no more Sensing parts to execute before the next Sensing period. This pattern can be seen in Figure 4.1.

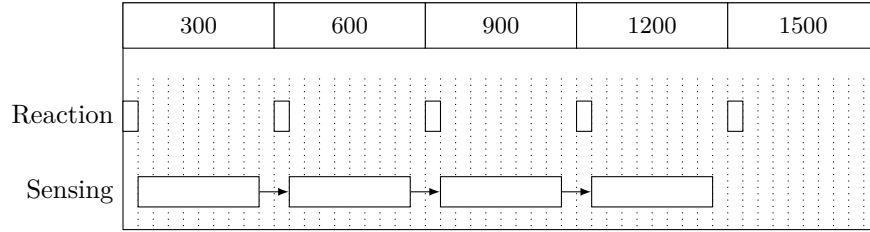


Figure 4.1: Gantt chart showing the fast pulses with the sensing task split up

We can also consider the situation when the mallet is moving slower and our Reaction period increases, namely during acceleration and deceleration. If the reaction period is 1000 μ s, we will start out by running first the Reaction task and then one part of the Sensing task, similar to the process depicted in Figure 4.1. However, this time we are left with 550 μ s, so we have time to execute another part of the Sensing task. When we finish executing the other part, we are left with 100 μ s, which is not enough time to execute anything, so we wait for the next Reaction task. This pattern can be seen below in Figure 4.2.

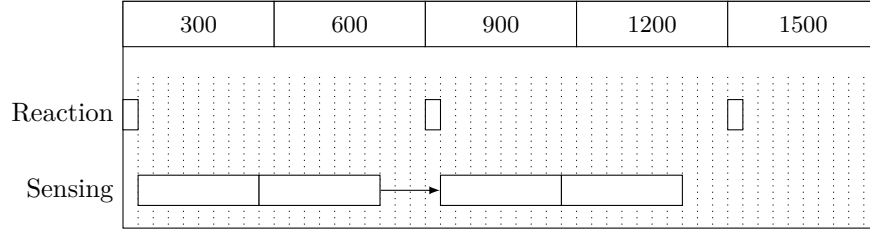


Figure 4.2: Gantt chart showing timing of slow pulses with multiple sense task parts per reaction task

With these two examples, we can work out a simple algorithm, see Algorithm 1, to schedule the tasks. We can constantly see if there is time to run a part of the Sensing task. If there is time to run it, we do it and check again. Otherwise we just wait for the next Reaction task and run it.

```

while running do
   $t \leftarrow$  the current time in microseconds;
  if  $t + PC_{sensing} \leq next_{reaction}$  then
    | execute Sensing task part;
  else
    | wait while  $t < next_{reaction}$ ;
    |  $next_{reaction} \leftarrow t + T_{reaction}$ ;
    | execute Reaction task;
  end
end

```

Algorithm 1: The scheduling algorithm.

This algorithm seems to work, at least in the two examples we went through earlier. To make sure no situations come up that would make this scheduler perform a schedule that does not obey the two task periods, we should verify it.

We can now build a timed automaton that represents the scheduler and the tasks and use UPPAAL [21] as a tool to verify that our scheduler will always be able to schedule the tasks.

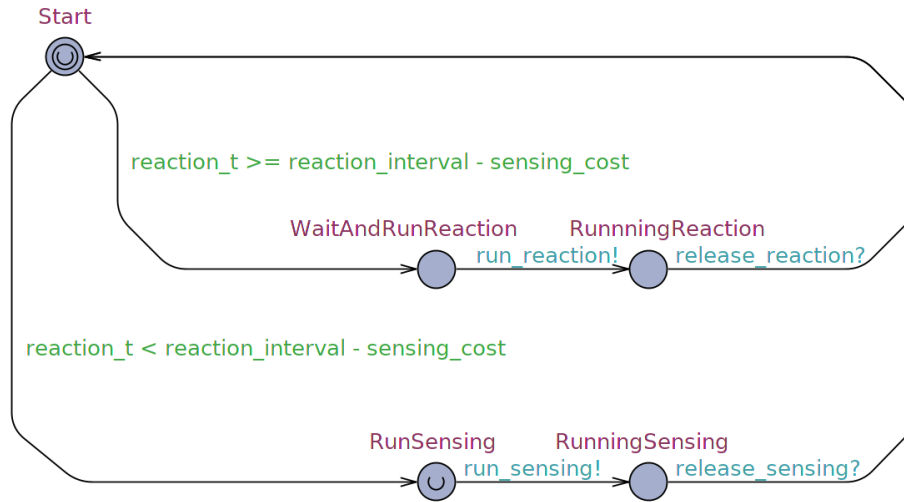


Figure 4.3: Timed automaton of the scheduler.

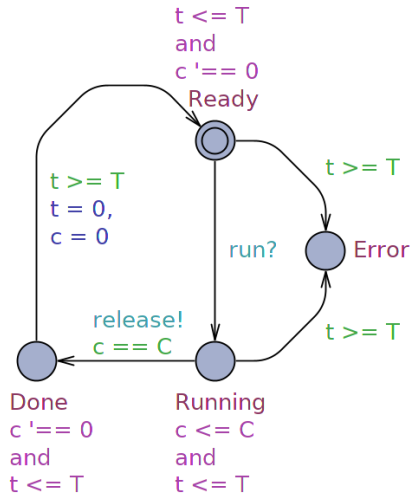


Figure 4.4: Timed automaton of the tasks.

By processing the two automaton seen in Figures 4.3 and 4.4 UPPAAL can tell us that the tasks will always be run within their period in this system, so no periods are violated.

Chapter 5

Evaluation

5.1 Verification

In order to ensure the controllers function as simulated, we conduct a simple experiment to test the validity of the models. The initial testing shows the three-step \rightarrow RLEC only achieves a catch-rate of around 24%, which is a significant reduction compared to the 67.2% that the simulations showed in Section 3.4.8. The robot also shows unexpected behaviour including, not responding to incoming shots or moving seemingly randomly. We expect a perfectly modelled system to behave identically in a simulated environment and the real world, and in extension of this, a closely modelled system behaves similarly to the real world. For this reason, we explore the correlations of the individual components of the system to their respective models to see whether or not they behave identically.

5.1.1 Camera

While the camera has been observed to send the position of the puck conjointly with noise, see Section 3.3.1.1, further testing into the imperfections of the camera is done. Namely, additional noise analysis and image acquisition rate.

5.1.1.1 Camera Noise Analysis

In order to gain a deeper understanding of the noise the PixyCam presents, we shoot the puck, by hand, along a trajectory where we deem the vy of the puck to

be minimal. It is expected that the camera will be able to identify the positions of the puck, such that they can all be connected by a single straight line.

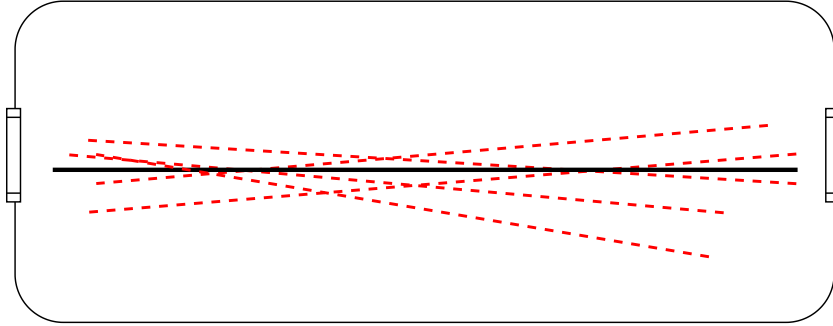


Figure 5.1: A conceptual illustration of how noise affects trajectory predictions.

As illustrated in Figure 5.1, we observe the correct trajectory, the black line, is not fully overlapping with any of the received trajectories, the red lines. Furthermore, as none of the received sets of puck positions can be connected via the use of a single straight line, as the noise from the PixyCam changes the linear extrapolation result depending on which of the puck positions are used.

5.1.1.2 Camera Acquisition Rate

Another possible faulty aspect of the camera is the rate with which we are expecting to receive data. Currently, the model of the camera describes the camera as sending the current puck position every 20 ms, Section 3.3.1.

In order to test whether or not this is the case, the puck is shot at approximately 5m/s along a simple trajectory observing all the information received from the camera.

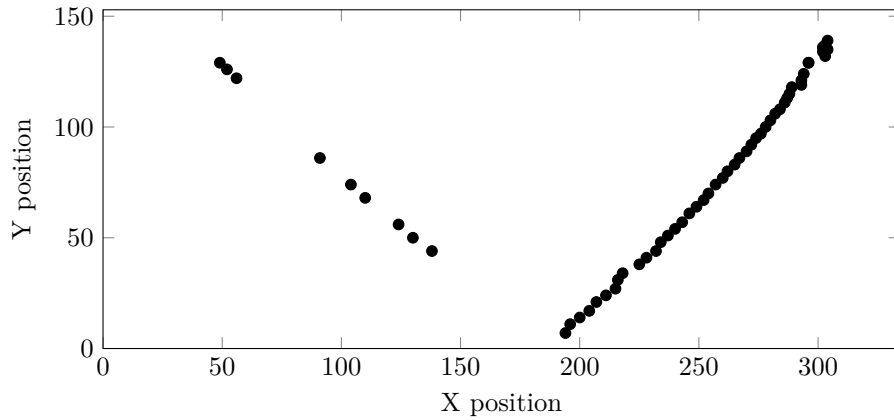


Figure 5.2: The captured positions of a puck in a rebound shot

As showcased in Figure 5.2, the camera sends faulty information not accounted for in the model. The PixyCam is unable to detect the puck positions during the initial phase of the course of the puck, contrary to the positions after hitting the sides of the rink which seem to register more clearly.

5.1.2 Puck

As the puck is not susceptible to the same kinds of hardware issues found in Section 5.1.1. It is still possible for the puck to behave differently than the modelled behaviour. In order to test this correlation, simple puck trajectories are introduced at different areas of the field and explored. We expect that the puck follows a perfectly linear trajectory from the initial position, P_i , to the final position, P_f , Figure 5.3 shows the expected puck position P_e and trajectory, the solid line, compared to the observed, the dashed line.

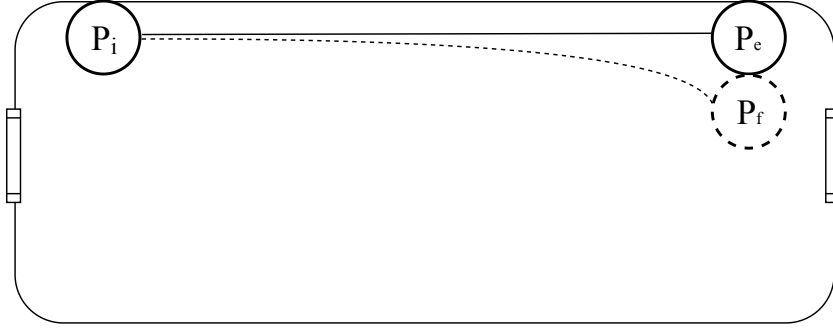


Figure 5.3: A conceptual illustration of how the edge of the railings affect the trajectory of the puck.

According to the test, the curve tendency of the puck increases as the puck approaches the railings. This means, that predicting the puck to move linearly fails, as the puck will attempt to approach the center line with an increasing rate, the closer to the railing the puck is travelling, due to the curving effect the edges of the board have on the trajectory of the puck.

5.1.3 Mallet

As the mallet is a mechanical component of the system, it may be subject to imperfections which affect the real world performance of the controllers. In order to test the correlation between the model and the real world, a simple test is presented which assists in assessing whether or not the mallet behaves correctly. The mallet is required to move a set distance with a specified speed a number of times, after which it resets to its original position and repeats the process. The goals of this test are to determine, whether or not the mallet moves consistently with the specified speed, and how well it considers its own position with respect to a target.

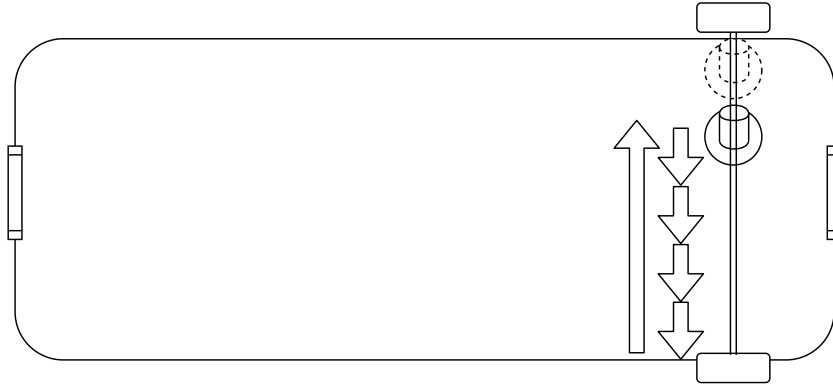


Figure 5.4: A conceptual illustration of how the acceleration and deceleration affect the position of the mallet.

The observed behaviour of the mallet indicates that it can maintain a specified speed well, as described in the model. However, over time the mallet becomes more and more inaccurate, to the point of being so off-position, that a calculated position which would block a shot is not the actual position the mallet ends up at. This issue is also illustrated in Figure 5.4, where the dashed mallet is the off-position the mallet eventually reaches when operating the test.

5.2 Discussion

Given the system simulates a human's way of playing air hockey, it naturally fulfils the system requirements presented in Section 3.1. However, while the robot is able to gather information about the position of the puck, the quality with which it does this is limited. While the effect the noise of the camera introduces to a stationary puck is correctly modelled, the effect the noise has on a predicted trajectory was left to be explored in Section 5.1.1. The results from this analysis indicates, that greater countermeasures against camera noise might need to be installed into the model for better performance. In addition to this, according to the camera model, the PixyCam will deliver information about the position of the puck every 20ms. However, as revealed in Section 5.1.1, the camera may fail to detect the puck at times, and in turn, fail to deliver the necessary information. According to the puck model in Section 3.3.2, the puck loses speed after bouncing off the railing. This implies that the PixyCam struggles to identify the puck during high speeds, but as the puck hits the railing and loses some of its initial speed it becomes easier to detect. This also explains why the detection issue is only present during the first half of the course of the puck. Similarly, the model of the puck is also deemed incomplete, as further testing in Section 5.1.2 reveals that the puck has a tendency to move towards

the middle of the board while travelling along the edges. This could be due to the air pump not being evenly powerful to get the same airflow throughout the entire table, resulting in the middle of the board having a stronger airflow compared to the edges of the rink.

As mentioned in Section 5.1.3, the mallet desynchronises its expected position and actual position, which implies an updating error between the expected position of the mallet and the actual position of the mallet. This desynchronisation, means the some of the dependability requirements, presented in Section 3.1.2, are lower than initially expected. Specifically, the reliability suffers from the desynchronisation, as the longer the game progresses the more disconnected the relationship between the current and expected positions of the mallet will be. In addition to this, the errors from the camera and the prediction of the puck, makes it difficult for the robot to reliably play on the expected level. In spite of this, the maintainability of the system is good. In the prototype built in wood, all components are reachable, and all components can easily be disassembled from the overall machine. Conceivably, a user could change the program running on the Arduino, motor controller, motor, or any combination of these to their liking. The rest of the requirements have been deemed less important, but are either fulfilled or not relevant.

5.3 Conclusion

The motivation behind this project was to create an embedded system. The approach to this was to anchor the problem in a problem related to a specific domain, and a concrete problem:

How can an autonomous air hockey playing robot, based in the concepts of sensing, prediction, and reaction, be created, such that it can respond to arbitrary shots?

In order to accommodate this, a system that acts as a defence against a human player was created. This piece of software is centered around linear extrapolating controllers(LEC), and placed on embedded hardware. LEC's were sufficiently efficient with regards to embedded hardware, because they can function inside a system with a lot of activity. The time restraints in the scheduled order, showcased in Section 4.3, leave little time for calculation and LEC controllers, being divisible into smaller pieces, allow for a fragmented execution that fits the scheduled order.

However, based off of comparisons made between simulated model and the air hockey robot, it can be concluded that the camera has a big impact on linear

extrapolating controllers. The simulated model, the results of which were shown in Section 3.4.8, had vastly better results than the air hockey table experiments shown in Section 5.1. Based off of the information shown in Section 5.1.1, it can be assumed that the real robot has periods of time where it lacks the information necessary to make reliable predictions with regards to puck position. These imperfections were never included in the model, as the design phase of the project, assumed that selected hardware was perfect. To make sufficiently accurate predictions about the system performance, inaccuracies like the camera has to be included in the model.

5.4 Future Work

While the final iteration of the air hockey robot, fulfilled the requirements set for it, there is still room for improvement. There are multiple factors that could weigh in on performance increases, and this section will focus on analysing said aspects with the purpose of gauging how much possible improvements could affect the efficacy of the air hockey robot.

5.4.1 Utilising a Better Camera

As was originally hinted at in Section 3.3.1.1, and further deliberated on in Section 3.5.1, the PixyCam 1 showed signs of ageing, both limiting its features from a technical point of view, but also due to wear from repeated use. This was further deliberated on in section Section 3.5.1 where experimentation into whether a newer model of camera could improve upon the performance of the air hockey robot.

The newer camera, in this case, could be the new PixyCam 2 that has all the functionalities of the previous model, in addition to being able to capture at 60 frames per second. As mentioned briefly in Section 3.5.1 the newer camera, could allow a decrease in shutter-speed, improving upon the overall speed of the air hockey by minimising the time it takes to take pictures. Additionally, a camera like the PixyCam 2 could allow for better object recognition - partly because the pictures taken are of a higher resolution, but also because of the increased amount of recognisable colours.

Furthermore, the PixyCam 2 also has built-in line-tracking [22], which could help in prediction of the trajectory of the puck.

5.4.2 Improving Upon the Model

As was touched upon in Section 5.3, there was a lack of realism when the air hockey robot was modelled in UPPAAL. These inconsistencies made the simulations run at a rate of efficiency that was not representative of the reality it was made to model. As such, there are certain changes that must be implemented in future models, for the model to produce accurate results.

One such change would be to add a measure that accounts for the time it takes to calculate rebound angles and the like. In the current iteration of the air hockey model, it assumes that as soon as a picture is taken, the path prediction will be available. This results in an inaccurate representation of controller efficiency, as it removes the calculation time from slower controllers.

Another thing that should be implemented for the air hockey robot is to account for the fact that the table is uneven. One way to do this could be to add a coefficient of friction that accounts takes the unevenness of the playing field into account, when calculating the path of the puck

5.4.3 Implementing Other Controllers

With regards to implementing models and simulating controllers, the program that, in this project, was used for testing timed automata was UPPAAL. Aside from model verification, UPPAAL is capable of teaching machine learning models [23]. Given the limited scope of control needed, and the clear goal for the mallet, a machine-learning-powered controller could be created and implemented, provided that the hardware to run such a controller was available. In addition, testing the difference in performance between a Q-learning and M-learning controller would also bring insight into whether further development would be beneficial.

This machine-learning controller could, if given the right parameters, likely be made to adapt itself to the specific table being used in this project. This could potentially lead to an increase in performance, due to the machine-learning algorithm adapting to factors like lightning and unevenness on the playing field, to name a few.

Bibliography

- [1] Kumba Sennaar.
Artificial Intelligence in Sports – Current and Future Applications.
URL: <https://emerj.com/ai-sector-overviews/artificial-intelligence-in-sports/>.
Last retrieved 08-10-2019.
- [2] Florian 'Floyd' Mueller et al.
"Airhockey over a Distance".
In: CHI '06 Extended Abstracts on Human Factors in Computing Systems.
CHI EA '06.
Montrécal, Québec, Canada: ACM, 2006,
Pp. 1133–1138.
ISBN: 1-59593-298-4.
DOI: 10.1145/1125451.1125665.
URL: <http://doi.acm.org/10.1145/1125451.1125665>.
- [3] Dave Kennard.
Campanula persicifolia 'Telham Beauty' Vis UV IR comparison.
URL: <https://www.flickr.com/photos/dave-kennard/9727518793/>.
Last retrieved 01-10-2019.
- [4] Chris Woodford.
Luminescence.
2019.
URL: <https://www.explainthatstuff.com/luminescence.html>.
Last retrieved 08-10-2019.
- [5] Georg Wiora.
Distance Sensor.
URL: https://upload.wikimedia.org/wikipedia/commons/thumb/0/07/Sonar_Principle_EN.svg/1024px-Sonar_Principle_EN.svg.png.
Last retrieved 01-10-2019.
- [6] Roderick Burnett.
Understanding How Ultrasonic Sensors Work.

- URL: <https://www.maxbotix.com/articles/how-ultrasonic-sensors-work.htm>.
Last retrieved 30-09-2019.
- [7] Amit Raj Singh.
What is LiDAR and how does it work?
URL: <https://www.geospatialworld.net/blogs/what-is-lidar-and-how-does-it-work/>.
Last retrieved 30-09-2019.
- [8] rishabhsingh1304.
Background Subtraction in an Image using Concept of Running Average.
URL: <https://www.geeksforgeeks.org/background-subtraction-in-an-image-using-concept-of-running-average/>.
Last retrieved 24-09-2019.
- [9] United States Air Hockey Association.
Air Hockey World.
URL: <http://www.airhockeyworld.com/usaarules.asp>.
Last retrieved 20-09-2019.
- [10] Rich Blake.
Industrial Robotic Arms Race Leaves Experts Asking 'What Manufacturing Slowdown?'
URL: <https://www.forbes.com/sites/richblake1/2019/01/09/industrial-robotic-arms-race-leaves-experts-asking-what-manufacturing-slowdown/>.
Last retrieved 10-12-2019.
- [11] Arduino.
Arduino Uno Rev3.
URL: <https://store.arduino.cc/arduino-uno-rev3>.
Last retrieved 08-10-2019.
- [12] Arduino.
Arduino Mega 2560 Rev3.
URL: <https://store.arduino.cc/arduino-mega-2560-rev3>.
Last retrieved 08-10-2019.
- [13] Raspberry Pi Foundation.
Raspberry Pi 4 Tech Specs.
URL: <https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/>.
Last retrieved 08-10-2019.
- [14] Xander Soldaat.
Comparing the NXT and EV3 bricks.
URL: <http://botbench.com/blog/2013/01/08/comparing-the-nxt-and-ev3-bricks/>.
Last retrieved 08-10-2019.
- [15] Brickset.
Mindstorms NXT 2.0.

- URL: <https://brickset.com/sets/8547-1/Mindstorms-NXT-2-0>.
Last retrieved 08-10-2019.
- [16] TC56.
TC56.
URL: <https://tc56.iec.ch/>.
Last retrieved 21-11-2019.
- [17] Alexandre David et al.
“Uppaal SMC tutorial”.
In: International Journal on Software Tools for Technology Transfer 17.4
(Aug. 2015), pp. 397–415.
ISSN: 1433-2787.
DOI: 10.1007/s10009-014-0361-y.
URL: <https://doi.org/10.1007/s10009-014-0361-y>.
- [18] Peter Bulychev et al.
“UPPAAL-SMC: Statistical Model Checking for Priced Timed Automata”.
In: Electronic Proceedings in Theoretical Computer Science 85 (July 2012),
pp. 1–16.
ISSN: 2075-2180.
DOI: 10.4204/eptcs.85.1.
URL: <http://dx.doi.org/10.4204/EPTCS.85.1>.
- [19] J. C. Jensen, D. H. Chang, and E. A. Lee.
“A model-based design methodology for cyber-physical systems”.
In: 2011 7th International Wireless Communications and Mobile Computing Conference.
July 2011,
Pp. 1666–1671.
DOI: 10.1109/IWCMC.2011.5982785.
- [20] Pixy.
Technical Specs.
URL: <https://docs.pixycam.com/wiki/doku.php?id=wiki:v1:overview#technical-specs>.
Last retrieved 05-12-2019.
- [21] Kim G Larsen, Paul Pettersson, and Wang Yi.
“UPPAAL in a nutshell”.
In: International journal on software tools for technology transfer 1.1-2 (1997),
pp. 134–152.
- [22] Pixy.
Line tracking for line-following.
URL: <https://docs.pixycam.com/wiki/doku.php?id=wiki:v2:overview#line-tracking-for-line-following>.
Last retrieved 19-12-2019.
- [23] Manfred Jaeger et al.
“Teaching Stratego to Play Ball: Optimal Synthesis for Continuous Space MDPs”.

In: Automated Technology for Verification and Analysis.
Ed. by Yu-Fang Chen, Chih-Hong Cheng, and Javier Esparza.
Cham: Springer International Publishing, 2019,
Pp. 81–97.
ISBN: 978-3-030-31784-3.

Appendix A

Restitution Influence

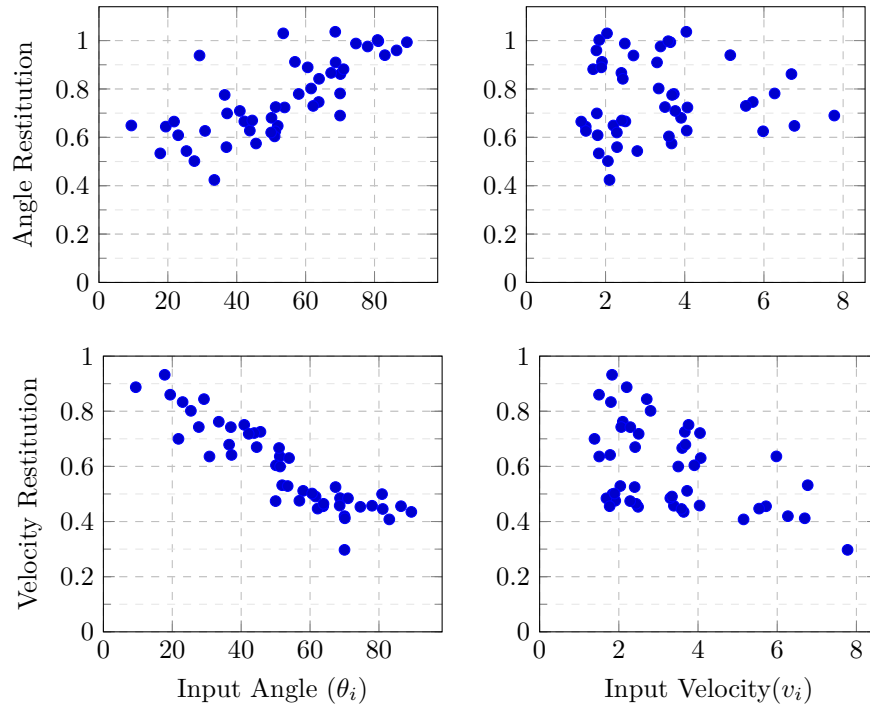


Figure A.1: The influence of θ_i and v_i on the θ_r and v_o .

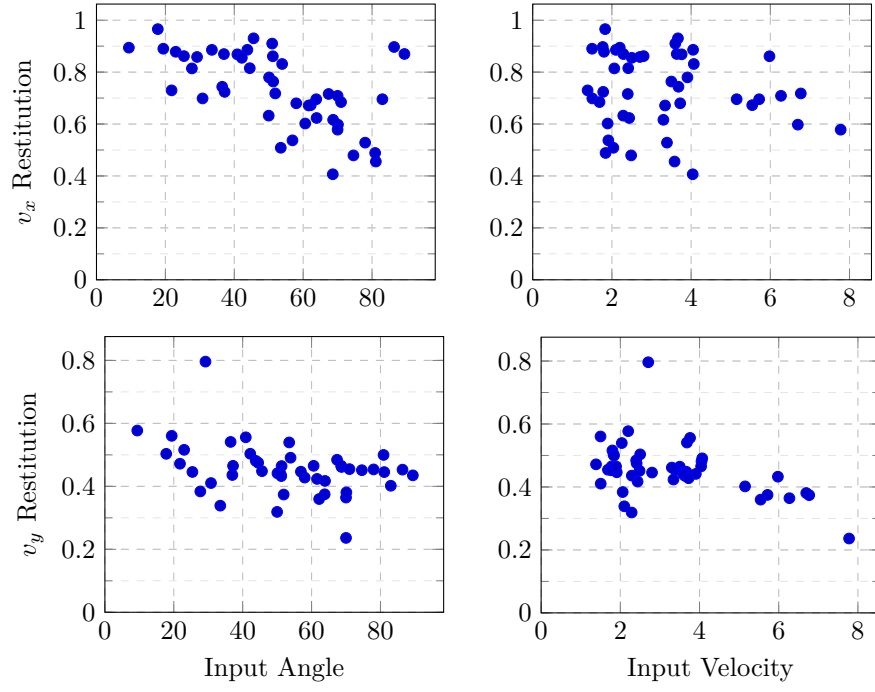


Figure A.2: The influence of θ_i and v_i on the velocity vectors x and y restitution.

0.960396	inangle	outangle	0.519197	speedratio	outvx
0.960309	invx	outvx	0.468970	outangle	invy
0.955555	inspeed	invy	-0.459196	inspeed	speedratio
0.936599	invy	outvy	-0.453019	inspeed	vyratio
-0.901046	inangle	speedratio	-0.434110	inangle	invx
0.892804	inspeed	outvy	0.420367	inspeed	outvx
0.880262	outangle	angleratio	0.379653	inangle	inspeed
0.866805	outspeed	invx	0.373968	invx	vxratio
0.837343	outspeed	outvx	0.372028	angleratio	outvy
-0.834932	outangle	speedratio	-0.372014	inangle	vyratio
0.792223	inspeed	outspeed	0.326549	invx	invy
0.742308	inangle	angleratio	-0.325686	invy	vxratio
0.741160	speedratio	vxratio	0.322493	speedratio	invx
-0.687118	angleratio	vxratio	0.315878	outspeed	vxratio
-0.681270	angleratio	outvx	-0.314763	outspeed	angleratio
-0.680299	speedratio	invy	0.312714	invx	outvy
-0.679407	outangle	outvx	-0.304531	outvy	vxratio
0.658791	outspeed	outvy	-0.298965	outvy	vyratio
-0.647106	outangle	vxratio	0.257230	vxratio	vyratio
-0.646321	angleratio	speedratio	0.240422	angleratio	invy
0.645452	inangle	outvy	0.221415	outangle	inspeed
-0.616388	speedratio	outvy	-0.217929	outangle	outspeed
0.615911	inangle	invy	0.213509	angleratio	vyratio
0.605868	outspeed	invy	-0.182407	outangle	vyratio
-0.587262	inangle	vxratio	0.169178	outvx	outvy
0.582490	outvx	vxratio	0.152219	outvx	invy
-0.580071	angleratio	invx	-0.137094	inspeed	vxratio
-0.577881	outangle	invx	0.103755	outspeed	speedratio
0.574047	inspeed	invx	0.081215	outvx	vyratio
0.568325	speedratio	vyratio	-0.078056	inangle	outspeed
-0.547974	inangle	outvx	-0.076849	outspeed	vyratio
0.536153	outangle	outvy	-0.046075	invx	vyratio
-0.532046	invy	vyratio	0.027635	inspeed	angleratio

Table A.1: Pearson correlations between the different variables of the angled shots.

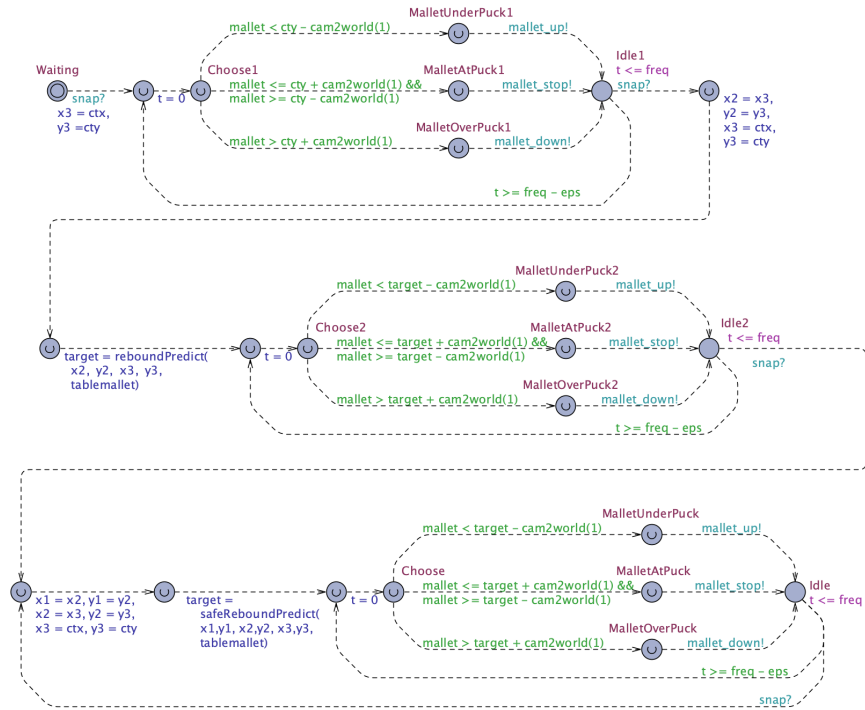


Figure A.3: UPPAAL Model for a combined controller