Launchbot

Autonomous floorball goalkeeper training

DOUGLAS ERIKSSON

JOEL GREBERG
This page is intentionally left blank.
Launchbot

Autonomous floorball goalkeeper training

DOUGLAS ERIKSSON
JOEL GREBERG

Bachelor’s Thesis at ITM
Supervisor: Nihad Subasic
Examiner: Nihad Subasic

TRITA-ITM-EX 2018:59
Abstract

The purpose of this project was to help floorball goalkeepers acquire gains in reaction time, catch/loss ratio and overall agility by using a ball launching machine. The device had the ability to launch balls in both a straight or curved trajectory and to aim at different parts of the goal.

The project focused on reducing the dispersion of balls launched in order to make the aim as precise as possible. By automating the sequence the training is made rewarding and challenging.

In order to make the machine safe and precise, stable material was required. However, it was found that the stable and heavy material increased the time to set the aim.

The final constructed product had the ability to aim at any part of the goal. The aim was tested at 6m away from the target and at 10m. It was found that 6m away there was a 15cm deviation from the target and at 10m away this deviation was 30cm. The velocity of the floorball at the goal was 15m/s at 75% motor power. The ball curved when spin was applied resulting in the center point of shots moving 42.6cm. The deviation from target with the new trajectory was 20cm from a distance of 6m.
Referat

Launchbot

Syftet med detta projekt var att bygga en maskin som ska hjälpa innebandmålvakter förbättra sina reaktionsförmågor, greppförmåga och rörelse i målet. Enheten hade förmågan att avfyrar bollarna i både en rak och en böjd bollbana mot olika punkter av målet.

Projektet fokuserade på att minska spridningen av bollar med avsikt att göra siktet så exakt som möjligt samtidigt som att automatisera hela sekvensen för att göra träningen både givande och utmanande.

För att göra maskinen säker och precis så var stabilt material en nödvändighet men det stabila och även tunga materialet ledde till en ökad tid för inställning av sikte.

Resultatet var en konstruktion som kunde sikta mot samtliga punkter i målet med en spridning som motsvarade en area av 30 · 30cm från ett avstånd av 6m och 60 · 60cm från ett avstånd av 10m då bollens utgångshastighet var 15 m/s och motorerna kördes på 75%. Med skruv applicerad på bollen så förflyttades träffcentrum 42,6cm och hade en spridning på 40 · 40cm från 6m avstånd.
This page is intentionally left blank.
Acknowledgements

We would like to thank all people who assisted us throughout the project, starting with our supervisor Nihad Subasic for guidance and feedback on both the project and report. We would also like to thank Staffan Qvarnström for his assistance with materials and sharing of knowledge surrounding motors and drivers. We would like to thank Åke Larsson for assisting with ideas, feedback and material. Finally, we would like to thank all assistants and classmates, especially Edvin Kugelberg and Philip Andersson, for their support, help and input throughout the project.

Douglas Eriksson & Joel Greberg
Stockholm May 2016
This page is intentionally left blank.
Nomenclature

\[ D \] Distance between target and launcher
\[ g \] Gravitational constant
\[ h_0 \] Height of the objects launching point
\[ H \] Height of the launcher, distance between ground and the start of the pipe
\[ L \] Length of pipe
\[ T \] Time for the object to leave the nozzle and hit the target
\[ v \] Velocity on the y axis
\[ v_0 \] Total velocity of the launched object
\[ x \] Desired impact point on the x-axis
\[ y \] Desired impact point on the y-axis
\[ \theta_x \] angle between origo and the sought point of impact on the x-axis
\[ \theta_y \] Angle between origo and the sought point of impact on the y-axis
This page is intentionally left blank.
List of Abbreviations

\begin{itemize}
\item DC \quad Direct Current
\item MOSFET \quad Metal Oxide Semiconductor Field Effect Transistor
\item MW \quad Master Writer
\item PWM \quad Pulse Width Modulation
\item ROM \quad Read-Only Memory
\item RAM \quad Random-Access Memory
\item RPM \quad Revolutions Per Minute
\item SR \quad Slave Receiver
\end{itemize}
# Contents

Abstract i  
Referat iii  
Acknowledgements v  
Nomenclature vii  
Abbreviations ix  
List of figures xiii  
List of tables xiv

## 1 Introduction
1.1 Background ............................................. 2  
1.2 Purpose ............................................. 2  
1.3 Scope ............................................. 3  
1.4 Method ............................................. 3  
1.4.1 Launcher construction ......................... 3  
1.4.2 Ball dispersion test .............................. 3  
1.4.3 Controlling the aim ......................... 4

## 2 Theory
2.1 Previous works ............................................. 6  
2.2 Microcontroller ............................................. 6  
2.3 DC-motor control ............................................. 7  
2.4 Stepper motor ............................................. 7  
2.5 Material ............................................. 7  
2.6 Theoretical trajectory prediction ......................... 8

## 3 Demonstrator
3.1 Problem formulation ............................................. 11  
3.2 Prototype development ............................................. 11  
3.2.1 Firing mechanism ................................. 11
List of Figures

1.1 The virtual net applied to a floorball goal. Picture edited in Adobe Photoshop CS6. .............................................. 4

3.1 The Firing mechanism assembled, rendered in Blender 2.79. ............ 12
3.2 The feeder and reloading mechanism, rendered in Blender 2.79. ........ 13
3.3 The parts holding the mechanism stable and allowing it to rotate around the vertical axis. ........................................... 14
3.4 Threaded metal block which results in a upward motion when rotation is applied to the threaded rod, rendered in Blender 2.79. ............ 14
3.5 All motor positions. Rendered in Blender 2.79 and edited in Adobe Photoshop CS6. ..................................................... 15
3.6 Flowchart of the system, made with Adobe Illustrator. .................... 17
3.7 Resulting hit spots from testing with a straight trajectory from 10m, graph made in Microsoft Excel. ................................. 18
3.8 Resulting hit spots from testing with a straight trajectory from 6m, graph made in Microsoft Excel. ................................. 19
3.9 Resulting hit spots from testing with a curved trajectory from 6m, graph made in Microsoft Excel. ................................. 20
List of Tables

3.1 The resulting data from the tests .......................... 20
This page is intentionally left blank.
Chapter 1

Introduction

1.1 Background

Floorball, Soccer and Hockey are a few examples of sports with great popularity. One thing that these three have in common is the individual positions the players can play. Though these positions possess many different names there are especially two main categories they can be divided in, goalkeeper and field player. The latter has the advantage of making it easy for the player to practice their skills by themselves. A ball or puck can be shot towards a target or a wall and you can easily practice with friends playing the same position as yourself. These statements cannot be applied to the goalkeeper. A wall will not suffice a suitable practice partner and there is no target that can improve your skills when it comes to guarding the goal. If you have a partner and you both play the same position as goalkeeper it will not be be easy to practice just the two of you.

To make it easier for goalkeepers to exercise on their own an automatic ball launcher that mimics the action of a shooting field player could be developed. Machines that launch balls such as baseballs [1] and tennis balls [2] already exist on the market but they only strive for automating the firing sequence, not aiming and reloading automatically.

1.2 Purpose

The aim of this project was to design a ball launching machine to help floorball goalkeepers acquire gains in reaction time, catch/loss-ratio and overall agility. The machine was required to aim, shoot and reload without any human interaction in order to make the practice as easy as possible. The main research question that this project was striving to answer was:

*How can we design a self-aiming floorball ball launcher for efficient goalkeeper practice?*
CHAPTER 1. INTRODUCTION

To bring a deeper understanding to the subject at hand the following sub-questions are also answered:

How small can we keep the variations in point of impact of the ball?

Can we compensate for the change in aim needed when applying spin to the ball?

1.3 Scope

To avoid becoming overwhelmed by the complexity of the project, some restrictions were considered necessary. This project focused on launching an object towards a floorball goal with specified dimensions. The launcher was only tested and used indoors in a relatively controlled environment and therefore outside forces such as wind, rain and inconsistencies in temperature were neglected. The surface of a floorball ball is not standardized. Some have dimples much similar to those on golf balls that affects the trajectory of the ball [3]. Since not all floorball balls have dimples, it will be neglected in all calculations.

1.4 Method

This project’s main focus was mechatronics and contained mechanical, electronic and programmable parts. The method consisted of two major steps. The first was the obtaining of mechanical pieces required for the electronic and programmable parts to be implemented. The second step was programming the machine to be user friendly and precise.

1.4.1 Launcher construction

A full size prototype was constructed using an iterative process. Material cost had to be lowered due to a limited budget. However component exchange or upgrades could be implemented if they were necessary. This launcher contained all the mechanical sub parts and was crucial for the project. CAD models were constructed to determine the correct dimensions and relative positions of components to reduce the volume and weight of the final product. Parts that could not be constructed by hand had to be ordered and adjusted for.

1.4.2 Ball dispersion test

By building the launcher and testing its aiming accuracy, a virtual net could be constructed containing virtual targets which the robot would hit. This is visible in Figure 1.1. With targets acquired the correct angles for which the launcher’s nozzle must obtain could be calculated. The calculations for the ball trajectory was firstly made using the formula for projectile motion [4] combined with a constant due to
CHAPTER 1. INTRODUCTION

the cavities presented in *Scope*. The constant will counteract the force obtained from air resistance and the magnus force [5]. The constant was calculated through tests for optimum results. The speed acquired was measured using a *Supido Multi Sports Personal Speed Radar Precision Training Instrument* [6] from a distance of 6 and 10m.

![Figure 1.1: The virtual net applied to a floorball goal. Picture edited in Adobe Photoshop CS6.](image)

1.4.3 Controlling the aim

Once the dispersion was determined the control of the launchers aiming capabilities was constructed. The required angles were calculated, transferred and translated into rotational degrees for the stepper motors controlling the horizontal and vertical aim. This was completed using MATLAB.
This page is intentionally left blank.
Chapter 2

Theory

This chapter contains the theory relevant for this project.

2.1 Previous works

The technique used to accelerate the ball is the same used in the Professional Batting Training Machine [7]. The machine pushes the ball between two spinning wheels which then accelerates the ball to desired velocity depending on the rpm of the wheels. Spinning the wheels at different rpm also gives the ball a spin and curves the trajectory.

To pinpoint the correct landing position accurate calculations and measurements were required. A system using golf ball flight angle, velocity and spin to determine the trajectory and impact point of a ball was developed in 1979[8].

2.2 Microcontroller

A microcontroller was used to control and transmit commands to the motor and user interface. A microcontroller is a computer with one purpose, to run a single program stored in Read-only memory (ROM). The controller also possess a small amount of Random-access memory (RAM) for storage of variables [9]. Microcontrollers also possess input and output ports which, for example, can be used for powering a display, buttons or motors. The output can be either high or low voltage or, with the use of Pulse-width modulation (PWM), simulate a value between the two. This is possible by varying the voltage between full off or full on by changing the signal frequency of on-time and off-time of the signal [10]. One commonly used microcontroller today is the Arduino board that can be programmed, erased and reprogrammed at any given time [11].
CHAPTER 2. THEORY

2.3 DC-motor control

The speed of a DC-motor varies depending on the voltage sent to the motor. Maximum speed is acquired when highest specified voltage is sent in to the motor. This makes electrical current pass through arranged coils that create a magnetic field. This either pushes or attracts a coil in the motor which leads to rotation [12]. To be able to control the speed of a DC-motor running only in one direction, a Metal Oxide Semiconductor Field Effect Transistor (MOSFET) can be used. A MOSFET is a semiconductor-based transistor [13] that can be used to modify a high voltage signal with a low voltage one, such as a PWM-signal from a microcontroller. When the PWM-signal is applied to the signal pin of the MOSFET, the transistor opens and closes the high voltage gate in relation to the duty cycle of the PWM-signal. The voltage of the high voltage output gate becomes adjustable due to the fast opening and closing of the transistor. By modifying the duty cycle of the PWM-signal a corresponding change can be accomplished in the motor supply voltage.

2.4 Stepper motor

To be able to control how far an electric motor rotates an encoder can be fitted to any standard DC-motor. These rotary encoders can be used in a variety of applications and the technology can be adjusted for specific needs, the encoder can be magnetic [14], optical [15] or conductive based [16]. If there is a need for a higher resolution in the rotation than what a standard DC-motor can provide, a stepper motor may be a good alternative [17]. Stepper motors are based on brushless DC-motors but use more than one set of coils. By alternating the current flowing through the different sets of coils in the stepper motor, the outgoing shaft can be rotated in a very precise manner. Speed of rotation, amount of degrees to rotate and position to hold can be sent from a microcontroller to the stepper motor driver card. These are sent as short pulses or high and low outputs, which then controls the motor itself. The rotational speed of the stepper motors is therefore controlled by the frequency of pulses transmitted from the microcontroller. The direction of rotation is controlled by sending 0 or 5 volts through a directional pin on the separate driver. This results in the motor to spin clockwise or counterclockwise by transferring the current in different directions.

2.5 Material

The material used in the structure was required to be stable so that the wheels are kept in place and the risk of harming the users is removed. When the wheels gain rotation speed, their momentum increases exponentially and if some parts would fail it could be a high risk. As the machine is designed to stand on the ground there is no maximum weight for it. This makes the weight not a factor to be considered for mobility when choosing parts.
Due to the high momentum on the wheels, the pipe, plates and all other components must be placed in close proximity to the rotating wheels. This reduces the risk of them clashing during operation and accidentally launching unexpected objects towards users. A failure of one component should not lead to immediate danger.

2.6 Theoretical trajectory prediction

To make it possible for the robot to aim at the target, the correct angles must be calculated. Two angles must be acquired for the launcher to hit the targets, one being vertical and one horizontal. With outside forces neglected, the ball will keep its direction. Therefore the horizontal angle is given by

\[
\theta_x = \tan\left(\frac{x}{D}\right)
\]  

(2.1)

where \( x \) is the desired impact point horizontally, \( D \) is the distance to the goal and \( t \) is the time for the ball remains airborne. The vertical angle was numerically calculated using the formula for projectile motion which is formulated as

\[
y = h_0 + v_0 \cdot t \cdot \sin(\theta_y) - \frac{g \cdot t^2}{2}
\]  

(2.2)

where \( y \) is the desired impact point, \( h_0 \) is the height of the launchpoint, \( v_0 \) the output velocity, \( \theta_y \) the required angle of the nozzle and \( g \) is the gravitational constant. The horizontal angle is being sought, thus, the other variables must be formulated depending on that specific angle. The launchers height, \( H \), plus the length of the pipe, \( L \), gives the starting height for the object leaving the nozzle and is therefore given by

\[
h_0 = H + L \cdot \sin(\theta_y).
\]  

(2.3)

Including all distances and the total velocity of the object defined, the time for the object to reach the target can be written as

\[
t = \frac{D - L \cdot \cos(\theta_y)}{v_0 \cdot \cos(\theta_y)}.
\]  

(2.4)

Including \( h_0 \) and \( t \) eq.2.2 can be formulated as

\[
y = H + L \cdot \sin(\theta_y) + v_0 \cdot \frac{D - L \cdot \cos(\theta_y)}{v_0 \cdot \cos(\theta_y)} \cdot \sin(\theta_y) + \frac{g \cdot \left(\frac{(D-L\cdot\cos(\theta_y))^2}{v_0^2 \cdot \cos^2(\theta_y)}\right)}{2}.
\]  

(2.5)
CHAPTER 2. THEORY

For each y-position in the goal, an angle $\theta_y$ can be numerically calculated. By using eq.2.1 and 2.5 for all positions and velocities, a 3-dimensional matrix containing all information can be constructed.

Through tests a constant, $\epsilon$, can be calculated to compensate for air resistance and *magnus force* that would affect the vertical and horizontal angle [18]. $\epsilon$ was a scalar used on the $x$ and $y$ coordinates. Due to the air resistance being proportional to the speed [19] $\epsilon$ was differentiated depending on the object’s velocity.
This page is intentionally left blank.
Chapter 3

Demonstrator

This chapter formulates the problem, how all parts have been used, how the assembly and manufacturing took place and the final results.

3.1 Problem formulation

Ball firing mechanisms require high rotational speed and forces which means that they have to be constructed thoroughly and robust for both the user and the machine’s safety. With limited budget the previous statement led to mechanism being large and heavy. The solution was therefore not the most slim but it was adequate for the purpose of the prototype. The solution could be divided into 3 major parts:

1. Prototyping consisting of 4 sub-categories:
   a) Firing
   b) Loading
   c) Aiming vertically
   d) Aiming horizontally

2. Electronics: Connecting microcontrollers, motors and drivers.


3.2 Prototype development

All parts but the wheels, motors and bearings had to be constructed or modified to fit the specific application.

3.2.1 Firing mechanism

The firing mechanism consisted of one bottom plate with two holes for bearings and one top plate with two holes concentric to the two in the bottom plate with same
CHAPTER 3. DEMONSTRATOR

type of bearings. Between the plates two wheel axles were mounted in the four
bearings and on each axle one wheel was attached and locked from rotation relative
to the axle. Between the two wheels, a pipe was mounted between two wooden
pipe holders with friction pads. The pipe had a cutout which made it possible for
the wheels to reach the inside of the pipe and act directly on the ball inside. To
minimize vibrations and to lock the pipe and wheel axle in place between the plates,
four threaded rods were used as stabilizers combined with two bent metal plates in
two of the corners. On the topside of the top plate, two DC-motors were mounted
on sliding metal plates. On the motors, a timing belt pulley was attached allowing
for motor rotation to be transferred to the wheel axis through a timing belt. The
complete assembly is shown in Figure 3.1.

![Image](image_url)

**Figure 3.1**: The Firing mechanism assembled, rendered in Blender 2.79.

### 3.2.2 Feeding and reloading

By using a feeding and reloading mechanism the shooting sequence could be auto-
mated without the user ever having to interact with the machine during operation.
The mechanism consisted of three parts, one ball and rack container, one rack and
one stepper motor, see Figure 3.2. The container was placed on the backside of the
firing pipe. The rack would slide in the rack container by the gear when the stepper
motor was rotating. When the rack was in start position it kept the next ball from
falling down in the container. Once a firing sequence was started the rack moved
backward allowing for a ball to fall down and be pushed in between the wheels.
This way, the only limiting factor for how many balls can be fired in sequence is the
size of the ball container.
3.2.3 Aiming

To avoid repeating the same point of impact every time a ball is fired it was desirable to be able to aim the firing mechanism in both horizontal and vertical directions. As the resulting mechanisms differ greatly from each other they are described individually.

Horizontal

Horizontal aiming of the machine was achieved by letting a solid plastic cylinder rotate freely inside a larger metal cylinder using two large diameter bearings. The metal cylinder was then placed on the ground and the plastic one was placed on the bottom of a plate on which the vertical aiming mechanism would be mounted. To control the rotation between the ground and the horizontal aiming plate, gears were mounted to the outside of the metal cylinder. As the span of desired angles horizontally was relatively small, only 90° of the metal cylinder had to be fitted with these gears. These were then laser cut from 10mm plexi and mounted to the cylinder using super glue. A stepper motor fitted with a small gear was mounted to the horizontal aiming plate so that the rotation could be controlled with adequate accuracy.
CHAPTER 3. DEMONSTRATOR

Figure 3.3: The parts holding the mechanism stable and allowing it to rotate around the vertical axis.

Vertical

To be able to aim the machine vertically two hinges were mounted on the horizontal aiming plate. This way, when a upward force was applied at the front end of the firing mechanism, it would move controllably. To generate the required force at the front, a threaded rod was mounted between the upper plate and the horizontal aiming plate. At the top plate a stepper motor fitted with a cardan joint provided rotation to the rod, which ran through a hole in the lower base plate. At the horizontal aiming plate this resulted in upward motion through a threaded metal block, see Figure 3.4. As the block was mounted with one free degree of rotation, the two plates could move in relation to each other in the desired fashion.

Figure 3.4: Threaded metal block which results in a upward motion when rotation is applied to the threaded rod, rendered in Blender 2.79.

3.3 Electronics

To allow for rotation around two axis, launching balls and automatically reloading, a total of five motors were required. Figure 3.5 shows all parts connected with wires via microcontrollers. Two stepper motors for aiming, shown as number 1.
CHAPTER 3. DEMONSTRATOR

One smaller stepper motor for loading, shown as number 2 and two DC-motors for rotating the wheels shown as number 3. The stepper motors can be controlled with high precision but a point of origin has to be determined every time the machine restarts. A sensor at each endpoint of the aim, shown as number 4, allows the machine to set the same origin as every startup. The stepper motors are shown as number 1. The sensors are shown as number 4. How all electronic components are connected and powered is shown in Appendix C.1.

Figure 3.5: All motor positions. Rendered in Blender 2.79 and edited in Adobe Photoshop CS6.

3.3.1 Microcontroller

The type of microcontroller used for both motor control and interaction was an Arduino UNO, programmed using the Arduino IDE. As the three stepper motor drivers required two or four pins each, a separate second Arduino UNO was used for motor control as the number of pins on a single board was insufficient [20]. The first Arduino controlled all other aspects of the machine, such as the screen and buttons. Communication between the two microcontrollers was implemented using the Master Writer(MW)/Slave Receiver(SR) library which had the advantage of only taking up two analog ports on each microcontroller creating a two way communication [21].

3.3.2 DC-motor

The speed of the two wheels used in the firing mechanism was very critical to the calculations of the firing angle to hit the desired impact point. If one wheel was unintentionally spinning faster than the other it would result in the ball having a spin and leaving its predicted path. To ensure that the rotational velocities of the two wheels would be as close to each other as possible it was desirable to use two identical motors. Unfortunately, the available motors were not ideal as they did not meet the required specifications and had to be geared up to reach the set top speed. A gearing ratio of 3:2 was implemented through two sets of 3D-printed wheels and
CHAPTER 3. DEMONSTRATOR

a belt. The motors used were two Maxon RE35 90W motors, see Appendix B.1, running at a maximum of 30V.

3.3.3 Stepper Motor

In order to control the aim and the ball feeder, three stepper motors of two sorts were used. Due to the low forces required to transfer a light ball a Tamagawa TS3214N61 200 step motor was chosen. It required a voltage of 12.5V and a maximum of 0.25A. The motors used for aiming experienced higher force and therefore Sonceboz 6600R165 was suitable. These motors used 30V and up to 6.5A per phase.

3.3.4 Motor Driver

To drive the two large bipolar stepper motors used for aiming two DRV8825 based drivers were used [22]. With a max current throughput of 2.5A they were well suited for the large motors. The stepper motor used for the feeding and reloading mechanism was much smaller and only consumed a maximum of 0.75A, so a TB6612FNG based driver was adequate [23].

To drive the DC-motors used to spin the wheels a BTS412B MOSFET was used for each motor [24]. This model has a current limitation of 5A, which was more than enough for the purpose, as the motors only pull a high amount of current during acceleration. To be on the safe side thermally, both transistors were mounted to a large heatsink, to help avoid overheating during extended operation. The full 30V was supplied to the circuit and then controlled by the MOSFET using a PWM signal from the Arduino. As a result of the two DC-motors being controlled and driven separately it was possible to spin the two wheels at different relative speeds, achieving a spinning ball and curved trajectory.

3.4 Software and interaction

3.4.1 User interface

To make the machine as user friendly as possible, a simple interactive program was written, see Appendix A.2.1 and A.2.2. Figure 3.6 shows the flowchart of the interactive program. The MW Arduino controls the SR Arduino by sending short commands, the SR constantly listens to MR and by using interrupts the response time is shortened. With the two way communication, the MR is able to wait for SR to complete its tasks before continuing. Once power is connected the communication is established and the initialization phase start. When all positions have been defined the SR send a command to the MW to continue with exercise choice. The user was given a choice between 4 different training modes with different ball velocities, randomization of impact points and firing interval resulting in higher or lower difficulties which can be aborted at any given time with a simple press
of a button. Once an exercise is complete or aborted the menu restarts and a new exercise can be chosen. These modes were preinstalled on the Arduino and could be adjusted by updating the imported data. The data was stored in vectors and matrices which reduced the code length and memory storage. All data was calculated beforehand using a MATLAB script.

![Flowchart of the system](image)

**Figure 3.6:** Flowchart of the system, made with Adobe Illustrator.

### 3.4.2 Matlab calculations

To calculate the required launching angles, a MATLAB script was used using the dimensions of the mechanism along with distance to target, desired impact points and ball velocities. Refer to Appendix A.1 for further details. These calculations was made before and imported before usage. The script then writes the output angles to a .txt file as a matrix with the correct format for the Arduino IDE for quick implementation.

### 3.5 Results

#### 3.5.1 Mechanical design

The construction consisted of four parts making it possible to fire balls automatically towards desired parts of the target. The low forces on the feeder made it possible
CHAPTER 3. DEMONSTRATOR

for a precise movement and secure transfer of balls. For horizontal aim a dead-
zone, due to laser cutted gears, made the aim less reliable than desired but from
10m, or shorter, the effect is minimal. One step from the stepper motor moving
the structure horizontally led to 0.225 degrees movement. The vertical aim was
extremely accurate which mostly was due to the threaded rod. One step from the
stepper motor led to 0.0034 degrees movement of the vertical aim.

3.5.2 Performance

To test the performance and accuracy of the system three tests were conducted.
The first aimed to describe the accuracy of the impact point at 75% speed and
10m distance. 75% speed measured up to roughly 15m/s on average. The second
used the same setup, but at a distance of 6m. The third kept the 6m distance but
applied maximum spin to the ball, to show whether the spin affected accuracy and
how much the impact point differed from the straight shot. All measurements were
made by shooting towards a whiteboard and using a marker to mark all hitpoints.

Straight shot

While testing the accuracy of a straight shot, the aim was adjusted to achieve an
impact point 80cm above ground level, straight in front of the machine. To check
for irregularities between different balls the three balls used were numbered and
always loaded in the same order. The test itself was performed as a series of three
balls at a time, ten times over, resulting in a total of 30 shots per test. Hitpoints
for every ball in the 10m test can be seen in Figure 3.7 below, where origo is the
center point and both axis represent a distance in cm.

![Figure 3.7: Resulting hit spots from testing with a straight trajectory from 10m, graph made in Microsoft Excel.](image-url)
For 90% of shots to be within limits a hitbox would in this case be approximately 60 · 60cm. Some shots were off by much more than others, possible causes for this will be discussed in chapter 4.

The same procedure was performed for the 6m test, and the aim was adjusted for an impact point 80cm above ground. At this distance the accuracy improved a lot, as can be seen in Figure 3.8 below.

![Figure 3.8: Resulting hit spots from testing with a straight trajectory from 6m, graph made in Microsoft Excel.]()

For 90% of shots to be within limits a hitbox would in this case be approximately 30 · 30cm, a quarter of that of the 10m test.

Spin applied to ball

When performing the tests with spin applied to the ball, the same distance and aiming point was used as in the 6m straight shot test, but the speeds of the left and right wheel were adjusted to 100% for the right and 30% for the left. The difference in rotational speeds between the wheels gave the ball a strong left spin, the results of which can be seen in Figure 3.9 below. Note that the coordinate system is adjusted for a centerpoint of all shots in this test, to better see the dispersion between shots. This centerpoint is 42.6cm to the left and 7.1cm below the one in the 6m straight test.
In this test a hitbox would be approximately $40 \cdot 40\, cm$ for 90% of shots to be within limits.

In all of the tests the different balls were separated by color in the figures in order to be able to see whether there were any differences between them. In the straight shot tests, a better accuracy off Ball 1 can be seen in the pattern of hitpoints compared to the other balls in the same tests. Possible reasons will be discussed in the following chapter. A summary of the results of the three tests can be seen in Table 3.1 below.

Table 3.1: The resulting data from the tests

<table>
<thead>
<tr>
<th></th>
<th>Velocity [m/s]</th>
<th>Average radius to impact point [cm]</th>
<th>Hitbox size [cm]</th>
</tr>
</thead>
<tbody>
<tr>
<td>10m straight</td>
<td>15</td>
<td>19.99</td>
<td>60 \cdot 60</td>
</tr>
<tr>
<td>6m straight</td>
<td>15</td>
<td>11.00</td>
<td>30 \cdot 30</td>
</tr>
<tr>
<td>6m curved</td>
<td>13</td>
<td>13.84</td>
<td>40 \cdot 40</td>
</tr>
</tbody>
</table>
Chapter 4

Discussion and conclusions

In this chapter the results of the project are discussed.

4.1 Discussion

The largest factors of error during testing were determined to be related to irregularities in ball stiffness, aerodynamic environment and precision in the horizontal aim. These, both individually and combined, can be seen as the main reasons why some balls hit greatly off target. When manufacturing a floorball ball two halves are molded separate and then melted together, creating a stronger section along the seam. If this seam happens to be aligned with the wheels, the ball will not compress as much, and as a result the initial velocity decreases. A lower initial velocity means a lower point of impact, which in the current state would be hard to compensate for. Irregularities in the aerodynamic environment of which the tests were conducted showed to be a greater factor than previously estimated. As the distance between the machine and target increases, the aerodynamic effects become more and more severe. This was especially visible in the differences between the 10m and 6m tests, as the 6m hitbox became 1/4th of the area of the 10m one. As the horizontal aim was actuated using a plexi plate with laser cut gears the precision of theses cuts were very critical. Unfortunately the precision was not as fine as predicted, leaving a small dead zone in the grip of the gears. Even though the dead zone was small and the outer gear only moved a couple of millimeters, this resulted in a dead zone of almost 10cm of horizontal impact point when launching from 10m away.

The vertical aim was very precise, largely due to the high resolution motor step to angle relation. This precision however does not show up as much in the testing results, as the differences in initial velocity is in direct relation to vertical impact point. The only downside with this mechanism for vertical aim was that it was arguably too slow as it undermines the element of surprise for the goalkeeper. If one can easily see where the machine is about to shoot it is not as hard to catch the ball.
Applying spin to the ball produced predictable and desirable results. The hit-point was moved enough to be unpredictable for the goalkeeper, but precise enough for the machine to compensate for. Maximum spin in this case was defined to a 100/30 relation between the speed of the right and left wheel. Tests were performed at 100/0 to achieve an even higher amount of spin. However as the ball was compressed between the two wheels the lack of rotational energy in the stationary wheel resulted in it gaining a backwards spin and the trajectory of the ball became inconsistent. It was found that 100/30 was the highest possible separation of speeds where the trajectory was still consistent.

From the start the goal was to be able to fire the balls at a greater speed than achieved, but due to a lack of availability of motors with the desired specifications the machine had to be fitted with weaker motors. This caused a longer speed up time, lower top speed and greater generation of heat than expected. These factors were however compensated for, and the machine can still be used as a demonstrator for the project.

4.2 Conclusions

In general the results were pleasing and the demonstrator lived up to its expectations. While some areas could definitely be improved the machine as a whole could be used for actual practise.

The main research question was answered by the resulting demonstrator, and it has shown great potential for future versions of this machine. The demonstrator works as desired and can easily be expanded or modified to further improve its performance.

The two sub questions were also answered after the testing of the machine. The variations in point of impact were kept within a hitbox of 60 · 60cm from 10m and 30 · 30m from 6m. This leaves the conclusion that the distance from target is a critical factor when aim is considered. From 6m the average radius to impact point increased by 25% and hitbox area by 77% when spin was applied, concluding that with added spin a deterioration in aim is obtained but possible to adjust for.
This page is intentionally left blank.
Chapter 5

Recommendations and future work

This chapter contains recommendations for optimizing and suggestions for future works.

5.1 Recommendations

With a slight higher budget a stronger pair of DC-motors are recommended as it would increase the output velocities and lead to a minimized the dispersion, as concluded in Discussion. By replacing the 3D-printed gears used for driving the wheels with the same model made in steel or aluminium a higher performance would be obtained. This would be mainly due to the reduction in friction from irregularities in the printed material. By implementing a worm gear for the horizontal aim a more accurate aim would be acquired along with less noise and the self-locking mechanism of a worm gear makes it suitable for this application.

With higher budget an effective weight to strength ratio on the materials is preferred. This would in turn reduce the stress on the stepper motors making them both faster and more silent. It is very important that the top and bottom plate are stable and can handle the stress from the spinning wheels in varying angles.

5.2 Future work

The resulting impact point for each shot is now either randomized or predetermined. By implementing computer vision the mechanism could be programmed to aim at parts of the goal that are left unguarded making the training more rewarding. The floorball goal is painted with a standardized red making so it is possible to be registered by a camera mounted on the machine.

By adding a distance sensor with a range up to 15m the machine could be placed at any distance and all calculations could be done in the initialization phase. With more options in the menu the user can adjust difficulty more easily and each exercise
could be suited for each unique user.

If another wheel was to be added, the curve of the ball could be controlled even further giving it top or bottom spin combined with the side spin.

Adding a speaker or buzzer would allow for more types of exercises and would make the interaction between the user and the machine easier. The machine can be customized depending on the size of balls to be fired. By increasing the distance between the wheels and pipe diameter the same mechanism can be made for other sports.
This page is intentionally left blank.
Bibliography


This page is intentionally left blank.
Appendix A

Software code

A.1 Matlab Code

The Matlab code used for calculation of the required angles of the machine.

```matlab
% % Project information

% Project name: Launchbot
% Project members: Douglas Eriksson & Joel Greberg
% University: KTH
% Date last modified: 2018-05-14

% Description: Calculating the expected impact point, for the launched ball, using different mathematical methods.

clc, clear all

%% Variables

v = 100; % Projectile velocity [km/h]
D = 6; % Distance to target [m]
h = 0.25; % Launchpoint height [m]
g = 9.82; % Gravity constant [m/s^2]
B = 1.600; % Width of target [m]
H = 1.150; % Height of target [m]
L = 0.400; % Length of pipe [m]

air_resistance_constant = 1; % Percentual drop of the ball due to air resistance
spin_constant = 1; % deal with it

%% User input

x = 0; % Value between -0.575 & 0.575 (+/- half of B)
```
APPENDIX A. SOFTWARE CODE

26 \( y = 0; \) %Value between 0 & 1.600 (0 & H)
27
28 \( \text{vel}_\text{start} = 55; \) %[km/h] Lowest velocity
29 \( \text{vel}_\text{max} = 85; \) %[km/h] Highest velocity
30 \( \text{interval} = 10; \) %[km/h] how much the velocity should increase
   by each step
31
32 %This depends on the amount of hitboxes:
33 \( x1=0 \ast \text{spin}_{\text{constant}}; \) %x–hitpoint adjusted for spin
34 \( y1=0.8 \ast \text{air}_{\text{resistance}}_{\text{constant}}; \) %y–hitpoint adjusted for
   \hspace{1cm} \text{air resistance}
35 \( x2=0.20 \ast \text{spin}_{\text{constant}}; \)
36 \( y2=0.18 \ast \text{air}_{\text{resistance}}_{\text{constant}}; \)
37 \( x3=0 \ast \text{spin}_{\text{constant}}; \)
38 \( y3=0.38 \ast \text{air}_{\text{resistance}}_{\text{constant}}; \)
39
40 \%\% Calculations – Analytical
41
42 \( v = v/3.6; \) %Transforms km/h to m/s
43 \( \text{thetaX} = \text{atan}(x/D); \) %Calculates the horisontal angle for
   \hspace{1cm} \text{the first stepper motor (no curve)}
44 \( \text{thetaY}_{\text{linear}} = \text{atan}(((y-h))/D); \) %Calculates the vertikal
   \hspace{1cm} \text{angle for the second stepper motor (all linear motion)}
45 \( \text{thetaY}_{\text{constT}} = \text{asin}(g*((D/v)\hspace{2pt}^2)+2y-2h)/(2*L+2*D); \) %
   \hspace{1cm} \text{Calculates the vertical angle for the second stepper}
   \hspace{1cm} \text{motor using projectile motion but assuming T is constant}
   \hspace{1cm} \text{for all angles}
46
47 \%\% Calculations – Numerical
48
49 \( \text{y}_\text{calc} = y+1; \) %Forcing \text{y}_\text{calc} to be much greater than \text{y},
   \hspace{1cm} \text{therefor upcoming while–loop will run}
50 \( \text{thetaY}_{\text{Numerical}} = -90; \) %All projectiles will be forward, \hspace{1cm} \text{therefor only } -90 \text{ to } 90 \text{ degrees is required}
51
52 \textbf{while abs}(y-\text{y}_\text{calc})>0.01 \&\& \text{thetaY}_{\text{Numerical}}<90
53 \hspace{1cm} \text{y}_\text{calc} = h+L*\text{sin}(\text{thetaY}_{\text{Numerical}}) + v*\text{sin}(\hspace{2pt}
54 \hspace{2pt} \text{thetaY}_{\text{Numerical}})*(D-L*\text{cos}(\text{thetaY}_{\text{Numerical}}))/(v*\hspace{2pt}
55 \hspace{2pt} \text{cos}(\text{thetaY}_{\text{Numerical}}))*g*((D-L*(\text{cos}(\hspace{2pt}
56 \hspace{2pt} \text{thetaY}_{\text{Numerical}}))))/(v*\text{cos}(\text{thetaY}_{\text{Numerical}}))^2)/2;
57 \hspace{1cm} \text{thetaY}_{\text{Numerical}} = \text{thetaY}_{\text{Numerical}}+0.001;
58 \textbf{end}
APPENDIX A. SOFTWARE CODE

58    % Calculate all desired impact points with different velocity (gradually increasing)
59
60    ImpactPoints = [x1, x2, x3; %[m] x-position
61        y1, y2, y3]; %[m] y-position
62
63    vel_start = vel_start / 3.6; %[m/s]
64    vel_max = vel_max / 3.6; %[m/s]
65    interval = interval / 3.6; %[m/s]
66
67    k=1;
68    thetaXY = []; %Matrix that will have 3 dimensions: X-angle, Y-angle at/and respective velocity
69
70    for vel = vel_start:interval:vel_max
71        for i = 1:length(ImpactPoints)
72            X = ImpactPoints(1,i);
73            Y = ImpactPoints(2,i);
74            thetaXY(1,i,k) = atand((X)/D); %Calculates the horizontal angle for the first stepper motor (no curve)
75            y_calc = Y+1; %Forcing y_calc to be much greater than y, therefore upcoming while-loop will run
76            thetaY_Numerical = -90; %All projectiles will be forward, therefore only -90 to 90 degrees is required
77
78            while abs(Y-y_calc)>0.01 && thetaY_Numerical<90
79                y_calc = h+L*sind(thetaY_Numerical)+vel*sind(thetaY_Numerical)*(D-L*cosd(thetaY_Numerical))+(vel*cosd(thetaY_Numerical))/((D-L*(cosd(thetaY_Numerical)))/(vel*cosd(thetaY_Numerical)))*2)/2;
80                thetaY_Numerical = thetaY_Numerical+0.001;
81            end
82
83            thetaXY(2,i,k) = thetaY_Numerical;
84        end
85    end
86
87    thetaXY(2,i,k) = thetaY_Numerical;
88
89    Following writes the matrix to .txt file which then can be implemented to the arduino code:
APPENDIX A. SOFTWARE CODE

```c
file = fopen('results_may5.txt', 'w');
for vel = 1:size(thetaXY,3)
    fprintf(file, '{\t}');
    for i = 1:size(thetaXY,2)
        fprintf(file, ['',{num2str(thetaXY(1,i,vel))},'',{num2str(thetaXY(2,i,vel))}], ', ');
    end
    fprintf(file, '{\\n});
end
fprintf(file, '{\\n);
```
A.2 Arduino Code

A.2.1 Master Writer

1
2 /*
3 Program name: master
4 Authors: Greberg, Joel & Eriksson, Douglas
5 Date modified: 2018/05/19
6 Description: Runs the master controller of the AIMBOT-
7 project
8 */
9
10 #include <Arduino.h> // needed to compile the code with the
11   external compiler
12 #include <LiquidCrystal.h> // needed to communicate with the
13   lcd-display
14 #include <Wire.h> // needed to communicate with the other
15   Arduino
16
17 // initialize the library by associating any needed LCD
18   interface pin
19 // with the arduino pin number it is connected to
20 const int rs = 7, en = 6, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
21 LiquidCrystal lcd(rs, en, d4, d5, d6, d7); // declares the
22   display as lcd, with the correct pins
23 int button_thresh = 900; // the threshold for the analogRead
24   () function to read a button as pressed.
25
26 const int stop = 9; // to send to Slave
27 const int next = 0; //analog button
28 const int ok = 1; //analog button
29 int curr_exerc; // Variable to store the index of the
30   current exercise
31 unsigned long last_change; // variable to store the time in
32   ms from the last change on the display
33 int input; // variable used to store information sent by the
34   slave arduino
35 int info; // variable used as index of info to be displayed
36   at a given moment
37 int changed; // variable used to check if any info has
38   changed
39 int done; // variable used to check if the exercise is done
40 int ready; // variable used to check if the exercise should
APPENDIX A. SOFTWARE CODE

```c
int result; // variable used to store the result of an exercise
int initializing; // variable used to check if the machine is initializing or not
const int speeds = 4; // number of speeds
const int no_exerc = 4; // number of exercises
const int no_settings = 3; // number of settings
const int feeder_stby = 8; // standby pin for the driver of the feeder motor
const int STOP = 0; // 0 is used as a STOP-command when communicating between the arduinos
int transmission; // variable used to store a transmission from the slave arduino

int settings[no_settings][speeds] = {{50, 66, 83, 100},
   {1, 5, 4, 3},
   {0, 20, 50, 100}}; // Speeds

char* names[no_exerc] = {"---KEXPO DEMO---", "---Exercise 2---", "---Exercise 3---", "---Exercise 4---"}; // Names of the exercises
char* props[no_settings] = {"Velocity: ", "Delay: ", "Dispersion: "}; // names of the different settings for each exercise

// confirm_run() checks if the user actually meant to press the run option, and if so runs the selected exercise
// It takes no input arguments. It returns 1 if the option to run was not selected, and 0 if the exercise was executed.

int confirm_run(){
    ready = 0; // reset ready
    lcd.clear(); // clear the display
    lcd.print("Are you sure?"); // asks the user if sure to continue
    lcd.setCursor(0,1); // selects the next row on the lcd
    lcd.print("Yes=ok No=Next"); // gives the user instructions on how to act depending on desired outcome
    while(1){
        if (analogRead(ok) < button_thresh) { // if the ok button is not pressed ready is set to 1. This is to avoid
```
APPENDIX A. SOFTWARE CODE

```c
the function reading the previous ok-press
ready = 1;
delay(100); // a delay of 100ms is set to avoid
interpreting glitches in the button if half-pressed
}
if (analogRead(ok) > button_thresh && ready == 1){ // if
    ready is 1 and the ok button is pressed again the
    exercise should run
    lcd.clear(); // clear the lcd
    lcd.print("Run "); // print the exercise number to
    the lcd
    lcd.print(names[curr_exerc-1]);
    lcd.setCursor(0,1);
    lcd.print("Cancel = any key"); // tell the user that
    it is possible to end the exercise at any time
    digitalWrite(feeder_stby, HIGH); // turn on the
driver for the feeder motor
Wire.beginTransmission(8); // transmit the index of
    the current exercise to the other arduino
Wire.write(curr_exerc);
Wire.endTransmission();
delay(350);

transmission = 0; // reset the transmission variable
while (transmission != 1 ){ // while the exercise is
    running constant checks are made for button
    presses and the master arduino asks the slave if
    the exercise is done
    if (analogRead(next) < button_thresh && analogRead
        (ok) < button_thresh){ // check for pressed
        buttons
        Wire.requestFrom(8,1); // request 1 byte from
        adress 8 (the other arduino)
        while(Wire.available()){ // read the answer
            from the slave arduino
            transmission = Wire.read(); // store it in
            transmission
        }
        delay(150);
    }
else{ // if any button is pressed
    Wire.beginTransmission(8); // send to adress 8
    Wire.write(STOP); // sends a STOP to the other
    arduino
```
APPENDIX A. SOFTWARE CODE

```cpp
Wire.endTransmission();
curr_exerc = 1; // resets the current exercise to default value
changed = 1; // indicates something has changed
lcd.clear(); // clear the lcd and
lcd.print("STOPPED!"); // write STOPPED! on it
delay(2000); // wait for 2s before returning to the main menu
transmission = 1; // resets the transmission variable
return 0; // returns a 0 to indicate that the exercise is done. This returns straight out of the whole function
```

```cpp
}
}
}

lcd.clear(); // clear the lcd
lcd.print("Exercise done"); // Write exercise done on it
transmission = 0; // resets transmission
digitalWrite(feeder_stby, LOW); // turn off the driver for the feeder motor
delay(2000); // wait 2s before returning to the main menu
return 0; // returns a 0 to indicate that the exercise is done. This returns straight out of the whole function
```

```cpp
if (analogRead(next) > button_thresh){ // if the next button is pressed
  curr_exerc = 1; // resets the current exercise to default value
  changed = 1; // indicates something has changed
  lcd.clear(); // clear the lcd
  lcd.print("STOPPED!"); // write STOPPED! on it
delay(2000); // wait for 2s before returning to the main menu
  return 1; // return 1 to indicate the exercise not being run
}
```

```cpp
// setup() initializes the machine. It takes no input arguments and produces no output
```
void setup() { 
    pinMode(feeder_stby, OUTPUT); // define the pinmodes
    digitalWrite(feeder_stby, HIGH); // turn on the driver for
    the feeder motor
    lcd.begin(16, 2); // set up the LCD’s number of columns and
    rows
    Wire.begin(); // Begin communications with the other Arduino
    lcd.print("---LAUNCHBOT---"); // print LAUNCHBOT to the lcd
    lcd.setCursor(2, 1);
    delay(1000); // wait 1s to let the other arduino boot
    initializing = 3; // set initializing to 3
    lcd.print("Initializing..."); // tell the user that the
    machine is initializing
    Wire.beginTransmission(8); // begins a transmission session
    on adress 8 - the slave arduino
    Wire.write(7); // send command 7 to start initializing to
    slave.
    Wire.endTransmission();
    transmission = 1; // sets transmission to 1
    delay(50);
    while (transmission != 0) { // If the other arduino is done
        transmission will be set to 0. This loop runs until that
        happens.
        Wire.requestFrom(8, 1); // request 1 byte from adress 8
        while (Wire.available()) { // read the answer
            transmission = Wire.read(); // and store it in
            transmission. Will be 0 if done
        }
        delay(500); // check every 500ms if other arduino is
        done
    }
    lcd.clear(); // clear the lcd
    lcd.setCursor(0, 1); // set the cursor to the second line
    lcd.print("InitializingDone"); // and write "
    InitializingDone" to it
    digitalWrite(feeder_stby, LOW); // turn off the driver for
    the feeder motor
    delay(1000);
    lcd.clear();
    curr_exerc = 1; // resets variables
    info = 0; // resets variables
    changed = 1; // resets variables
}
APPENDIX A. SOFTWARE CODE

// the main loop of the program. Takes no input arguments
// and produces no output.
void loop() {
  if (analogRead(next) > button_thresh && done == 0) { // if
    the next button is pressed and done is 0
    curr_exerc++; // change to the next exercise
    if (curr_exerc == no_exerc+1) { // wrap around if reached
      the number of exercises available
      curr_exerc = 1;
    }
    info = 0; // resets variables
    changed = 1; // resets variables
    done = 1; // the tasks for this button press has been
    executed
  }
  if (done == 1 && analogRead(next) < button_thresh) { // if
    the tasks are done and the button is released
    done = 0; // done is reset to 0
    delay(100);
  }
  if (analogRead(ok) > button_thresh) { // if the ok button is
    pressed
    result = 0; // result is reset to 0
    result = confirm_run(); // calls the function confirm_run
    () and stores the output in result
    curr_exerc = 1; // resets variables
    changed = 1; // resets variables
  }
  if (changed == 1) { // if the information to be displayed has
    changed
    lcd.clear(); // clear the lcd
    lcd.print(names[curr_exerc-1]); // print the name of the
    exercise
    lcd.setCursor(0,1); // reset the cursor to the second line
    lcd.print(props[info]); // print the current setting name
    to the lcd
    lcd.print(settings[info][curr_exerc-1]); // print the
    value of said setting for this exercise
    last_change = millis(); // store the time of change in
    last_change
    changed = 0; // reset changed
}
APPENDIX A. SOFTWARE CODE

if (millis() > last_change + 2000) { // cycle info about the current exercise
    info++; // next setting
    if (info == 3) { // wrap around if reached the number of settings
        info = 0;
    }
    changed = 1; // indicates that something has changed
    last_change = millis(); // store the time of change in last_change
}
APPENDIX A. SOFTWARE CODE

A.2.2 Slave Receiver

1 /*
2 Program name: slave
3 Authors: Greberg, Joel & Eriksson, Douglas
4 Date modified: 2018/05/19
5 Description: Runs the slave controller of the AIMBOT-
   project
6 */
7
8 #include <Arduino.h> // needed to compile the code with the
   external compiler
9 #include <Wire.h> // needed to communicate with the other
   Arduino
10 #include <Stepper.h> // needed to run the driver for the
   feeder motor
12
   the different intervals used between shots
14 const int vertDir = 3; // declare the pin for vertical aim
   direction
15 const int vertStep = 2; // declare the pin for vertical
   stepping
16 const int horizDir = 7; // declare the pin for horizontal
   aim direction
17 const int horizStep = 4; // declare the pin for horizontal
   stepping
18 const int stepper_stby = 13; // declare the pin for standby
   for the 2 stepper motors used to aim
19 const int ain1 = 8; // declare the 4 pins for controlling
   the driver of the feeder motor
20 const int ain2 = 9;
21 const int bin1 = 10;
22 const int bin2 = 11;
23 const int dc_right = 5; // declare the pin for right dc
   motor mosfet pwm signal
24 const int dc_left = 6; // declare the pin for left dc motor
   mosfet pwm signal
25 const int horizontal_button = 2; // analog button for
   positioning the aim coordinates
26 const int vertical_button = 1; // analog button for
   positioning the aim coordinates
27 const int feeder_button = 3; // analog button to change
direction when the feeder reaches its end

28 const int button_thresh = 900; // the threshold for the
   analogRead() function to read a button as pressed.
29 const int stepper_delay_vert = 1000; // delays in
   microseconds for how long to wait between pulses to the
   step-pin of each motor
30 const int stepper_delay_horiz = 15000; // delays in
   microseconds for how long to wait between pulses to the
   step-pin of each motor
31 int rpm_left = 0; // reset the speeds
32 int rpmCurrent_left = 0; // reset the speeds
33 int rpm_right = 0; // reset the speeds
34 int rpmCurrent_right = 0; // reset the speeds
35 Stepper feeder = Stepper(200, ain1, ain2, bin1, bin2); //
   initialize the feeder motor as feeder, with the correct
   pins
36 int transmission; // variable used to store information
   being sent between the arduinos
37 int num_balls = 3; // number of balls per exercise
38 int exercise_done; // to indicate the exercise being
   finished or not
39 int STOP = 0; // 0 is used as a STOP-command when
   communicating between the arduinos
40 int input; // // variable used to store information sent by
   the master arduino
41 int i; // used to count iterations
42 float vertPos = 0; // positions initially set to 0
43 float horizPos = 0; // positions initially set to 0
44
45 const int no_settings = 4; // number of settings per
   exercise
46 const int speeds = 4; // number of speeds
47 const int paus = 1; // interval between shots
48 const int disperse = 2; // randomization of hitpoint
49 const int num_hitboxes = 3; // number of hitboxes
50 int hitbox; // to store the current hitbox for aiming
51
52 // matrix with information about every exercise
53 int settings[no_settings][speeds] = {
   (255, 136, 170, 204), // speeds for motor 1
   (3000, 1000, 6000, 1000), // Intervals
   {0, 20, 50, 100} // dispersion
APPENDIX A. SOFTWARE CODE

70 {70, 170, 212, 255});
    // speeds for motor 2
71
72 // matrix with every hitbox, formatted as [vel: hitbox: x-y coord]. Calculated in MATLAB
73 float ang_matrix[speeds][num_hitboxes][2] = { {{0,11.581},
    {1.9092,5.584}, {0,7.521}, },
74 {{0,9.716}, {1.9092,3.763}, {0,5.687}, },
75 {{0,8.564}, {1.9092,2.631}, {0,4.549}, },
76 {{0,7.8}, {1.9092,1.877}, {0,3.792}, },
77};
78
79 // setRpm() takes in desired speeds of each motor and ramps its values to match. Produces no output.
80 // Ramping of pwm-values is used to avoid voltage spikes, mechanical stress and wear on the motors.
81 void setRpm(int rpm_left, int rpm_right){ // take an int for every motor value
82 int rpmDiff_left = rpm_left - rpmCurrent_left; // calculate the difference between current and desired speed
83 int rpmDirection_left = (rpmDiff_left)/(abs(rpmDiff_left)); // calculate the direction the change should go in. 1 or -1
84 rpmDiff_left = abs(rpmDiff_left); // total difference
85 int rpmDiff_right = rpm_right - rpmCurrent_right; // calculate the difference between current and desired speed
86 int rpmDirection_right = (rpmDiff_right)/(abs(rpmDiff_right)); // calculate the direction the change should go in. 1 or -1
87 rpmDiff_right = abs(rpmDiff_right); // total difference
88 int rpm_done = 0; // reset to 0 - not done
89 while (rpm_done == 0){ // runs as long as the function is not done
90     while (rpmDiff_left != 0){ // while there is a difference in current and desired speed
91         rpmCurrent_left = rpmCurrent_left + rpmDirection_left; // add or remove 1 from the speed of the left motor depending on direction
92         analogWrite(dc_left, rpmCurrent_left); // write the pwm-value to the mosfet pwm-pin
93         rpmDiff_left--; // remove 1 from the difference in
APPENDIX A. SOFTWARE CODE

```c
  speeds
  while (rpmDiff_right != 0){ // while there is a difference in current and desired speed
    rpmCurrent_right = rpmCurrent_right + rpmDirection_right; // add or remove 1 from the speed of the right motor depending on direction
    analogWrite(dc_right, rpmCurrent_right); // write the pwm-value to the mosfet pwm-pin
    rpmDiff_right--; // remove 1 from the difference in speeds
  }
  if (rpmDiff_left == 0 && rpmDiff_right == 0){ // if there are no differences anymore
    rpm_done = 1; // set done
  }
  delay(7); // this delay adjusts how fast the motors ramp. Lower = faster.
}

// aim() adjusts the machines aim to the desired coordinate. It takes an x- and y-coordinate as input and produces no output.
void aim(float vertCoord, float horizCoord){
  int vertDiff = vertCoord - vertPos; // calculate the difference between actual and desired vertical coordinate.
  int vertSteps = round(vertDiff/0.0034377468); //converts degrees to steps. Calculated factor by hand.
  int horizDiff = horizCoord - horizPos; // calculate the difference between actual and desired vertical coordinate.
  int horizSteps = round(horizDiff/0.225); //converts degrees to steps. Calculated factor by hand.
  if (vertSteps < 0){ // if the aim has to go up, set the direction accordingly
    digitalWrite(vertDir, HIGH);
  } else if (vertSteps >= 0){ // if the aim has to go down, set the direction accordingly
    digitalWrite(vertDir, LOW);
  }
```

45
if (horizSteps < 0) { // if the aim has to go to the right, set the direction accordingly
    digitalWrite(horizDir, HIGH);
} else if (horizSteps >= 0) { // if the aim has to go to the left, set the direction accordingly
    digitalWrite(horizDir, LOW);
}

int aimDone = 0; // reset the done variable
while(aimDone != 1) { // loop until done
    // vertical
    if (vertSteps < 0) { // if the aim has to go down, take one step in that direction
        digitalWrite(vertStep, HIGH);
        delayMicroseconds(stepper_delay_vert); // delay between pulses to the step-pin
        digitalWrite(vertStep, LOW);
        delayMicroseconds(stepper_delay_vert); // delay between pulses to the step-pin
        vertSteps++; // add one to the value of steps left to take
    } else if (vertSteps > 0) { // if the aim has to go up, take one step in that direction
        digitalWrite(vertStep, HIGH);
        delayMicroseconds(stepper_delay_vert); // delay between pulses to the step-pin
        digitalWrite(vertStep, LOW);
        delayMicroseconds(stepper_delay_vert); // delay between pulses to the step-pin
        vertSteps--; // remove one from the value of steps left to take
    }
    // horizontal
    if (horizSteps < 0) { // if the aim has to go to the left, take one step in that direction
        digitalWrite(horizStep, HIGH);
        delayMicroseconds(stepper_delay_horiz); // delay between pulses to the step-pin
        digitalWrite(horizStep, LOW);
        delayMicroseconds(stepper_delay_horiz); // delay between pulses to the step-pin
    } else if (horizSteps >= 0) { // if the aim has to go to the right, set the direction accordingly
        digitalWrite(horizStep, LOW);
        delayMicroseconds(stepper_delay_horiz); // delay between pulses to the step-pin
    }
pulses to the step-pin
horizSteps++; // add one to the value of steps left to take

if (horizSteps > 0){ // if the aim has to go right, take
    one step in that direction
digitalWrite(horizStep, HIGH);
delayMicroseconds(stepper_delay_horiz); // delay between
    pulses to the step-pin
digitalWrite(horizStep, LOW);
delayMicroseconds(stepper_delay_horiz); // delay between
    pulses to the step-pin
horizSteps--; // remove one from the value of steps left
to take
}

if (vertSteps == 0 && horizSteps == 0){ // if there is no
difference left
    vertPos = vertCoord; // set the current position to the
desired position
    horizPos = horizCoord; // set the current position to
    the desired position
    aimDone = 1; // done
}

if (STOP == 1){ // if the STOP-variable has been set to
    one during this loop, get straight out of the function
return;
}

} // feeder_step() runs the feeder one loop. Takes no input
    arguments and produces no output.

void feeder_step(){
while (analogRead(feeder_button) < button_thresh && STOP ==
    0){ // as long as STOP is 0 and the end-button is not
    pressed
    feeder.step(-1); // take one step back
}

if (STOP == 0){ // if STOP is still 0
    feeder.step(700); // take 700 steps forward (pushes the
    ball in between the wheels)
}
APPENDIX A. SOFTWARE CODE

```
171 // runExcercise() runs the selected exercise. Takes the
172   index of the desired exercise as input and produces no
173   output.
174 int runExcercise(int aktuell_ovn) {
175   transmission = 0; // resets the transmission
176   ballCount = 1; // resets the ball count
177   last_shot = millis(); // stores the time of
178   the last shot in last_shot
179   setRpm(settings[3][aktuell_ovn-1], settings[0][aktuell_ovn
180       -1]); // set the rpm of the dc-motors to their respective
181   setting
182   delay(settings[0][aktuell_ovn-1]*80); // delay to let the
183   wheels spin up, longer if the speed is higher.
184   while (STOP == 0 && ballCount <= num_balls){ // while
185       num_balls has not been reached and STOP is 0
186       if (settings[disperse][(aktuell_ovn-1)] == 0){ // if the
187           dispense-setting is 0
188           hitbox = 0; // set the hitbox to 0
189       }
190     }
191     if (settings[disperse][(aktuell_ovn-1)] == 20){ // if the
192         dispense-setting is 20
193         hitbox = random(num_hitboxes*3); // calculate the random
194         hitboxes
195       if (hitbox > num_hitboxes){
196           hitbox = random(num_hitboxes*2);
197           if (hitbox > num_hitboxes*0.66){
198             hitbox = random(num_hitboxes*0.33);
199           }
200         }
201     }
202     if (settings[disperse][(aktuell_ovn-1)] == 50){ // if the
203         dispense-setting is 50
204         hitbox = ballCount-1; // set the hitbox to ballCount-1
205     }
206     if (settings[disperse][(aktuell_ovn-1)] == 100){ // if the
207         dispense-setting is 100
208         hitbox = random(num_hitboxes); // set the hitbox to a
209             random number, up to the number of hitboxes
210     }
211     aim(ang_matrix[(aktuell_ovn-1)][hitbox][1], ang_matrix[}
APPENDIX A. SOFTWARE CODE

    aktuell_ovn-1)[hitbox][0]); // call the aim-function
    with the desired coordinates
202
    while (millis() < last_shot + settings[paus][{aktuell_ovn
203
        -1}]){ // wait until the delay between shots has passed
204         delay(10);
205     }
206
    feeder_step(); // run the feeder one loop
207
    last_shot = millis(); // store the time of this shot in
208
    setRpm(255, 204); // as the wheels slow down slightly when
209
    delay(500); // each shot
210
    setRpm(settings[3][aktuell_ovn-1], settings[0][aktuell_ovn
211
        -1]); // set the rpm back to what it should be for the
212
    ballCount++; // add one to the number of fired balls
213     }
214
    delay(50);
215
    setRpm(0,0); // stop the motors
216
    transmission = 1; // reset transmission to 1
217
    STOP = 0; // reset STOP to be 0
218
    // interrupt function to write transmission to the other
219
    void requestEvent(){
220     Wire.write(transmission); // write the value of
221
    }
222
    // interrupt function to receive data from the other arduino
223
    void receiveEvent(int howMany) { // receive byte as an integer
224     input = Wire.read();
225
    if (input == 0) { // an input of 0 signals an immediate
226         STOP = 1; // STOP variable set to 1
227
    setRpm(0,0); // dc-motors stopped
228
    input = 9; // 9 is the default standby value of input.
229     }
230
    // setup() initializes the machine. Takes no input or output
231
49
void setup() {
  pinMode(2, OUTPUT); // declaring pinmodes for all active pins
  pinMode(3, OUTPUT); // declaring pinmodes for all active pins
  pinMode(4, OUTPUT); // declaring pinmodes for all active pins
  pinMode(7, OUTPUT); // declaring pinmodes for all active pins
  pinMode(8, OUTPUT); // declaring pinmodes for all active pins
  pinMode(9, OUTPUT); // declaring pinmodes for all active pins
  pinMode(10, OUTPUT); // declaring pinmodes for all active pins
  pinMode(11, OUTPUT); // declaring pinmodes for all active pins
  pinMode(13, OUTPUT); // declaring pinmodes for all active pins
  analogWrite(dc_left, 0); // set the motor pwm’s to 0
  analogWrite(dc_right, 0); // set the motor pwm’s to 0
  randomSeed(analogRead(0)); // seed to be used for random calculations is taken from an empty analog pin
  transmission = 2; // transmission is set to 2, signaling a wait to start initializing
  feeder.setSpeed(150); // set the speed of the feeder
  Wire.begin(8); // join bus with address #8
  Wire.onReceive(receiveEvent); // register the interrupts
  Wire.onRequest(requestEvent); // register the interrupts
}

// main loop of the program. Takes no input or output
void loop() {
  if (input > 0 && input < 5) { // an input value of 1-4 signals the different exercises
    STOP = 0; // STOP is reset
    digitalWrite(stepper_stby, HIGH); // turn on the motors used for aiming
    runExcercise(input); // call runExcercise() with the input as input
    transmission = 1; // when done, transmission is set to 1, signaling an exercise being done
}
APPENDIX A. SOFTWARE CODE

264 delay(1000); // delay of 1s to let the other arduino have
time to read this value
265 digitalWrite(stepper_stby, LOW); // turn of the motors
    used for aiming
266 transmission = 0; // transmission reset to 0
267 input = 9; // input reset to default 9
268 }
269
270 if (input == 7){ // initialize the module
271    transmission = 1;
272    digitalWrite(stepper_stby, HIGH); // turn on the motors
        used for aiming
273    delay(1000); //
274    int i = 1; // counter
275    while (analogRead(vertical_button) < button_thresh){ //
        while the end position button is not pressed
276        aim(-1*i, 0.0); // aim down 1 degree
277        i++; // count up the iterations by 1
278        delay(2); // short delay to have time to read sensors,
            this also adjusts the speed of the aim
279    }
280
281    vertPos = 0.0; // this position is now 0, the origin
282    delay(200);
283    aim(5.0, 0.0); //aim 5 degrees up from the origin
284    delay(1000); // wait 1s
285    i = 1; // reset the interactions
286    while (analogRead(horizontal_button) < button_thresh){ //
        while the end position button is not pressed
287        aim(vertPos, 1*i); // aim right 1 degree
288        delay(10); // short delay to have time to read sensors,
            this also adjusts the speed of the aim
289        i++; // add one to the number of iterations
290    }
291
292    horizPos=20; // the right end position is 20 degrees from
    0, so 20 is the new current position
293    aim(vertPos, 0.0); //returns to origo but keeps the
    vertical position
294    digitalWrite(stepper_stby, LOW); // turn off the motors
        used for aiming.
295    delay(500);
296    feeder_step(); // run the feeder one loop to calibrate it
297    delay(1000);
APPENDIX A. SOFTWARE CODE

298    transmission = 0; // signals that the initialization is done
299    input = 9; // default value of 9 set to the input variable
300   }
301   delay(100);
302   }
Appendix B

Datasheets

B.1 DC-motor

Datasheet for the Maxon RE35 90W motors. The used motors are on the left side of the markings on the picture below.
Appendix C

Hardware

C.1 Wiring

All wiring required for driving the motors, buttons and LCD. Made with Fritzing.
This page is intentionally left blank.