

Hege Rishovd & Ane Sofie Smith Kristiansen

Improvements of the shuttlecock launcher robot BADDY

Master's thesis in Engineering and ICT
Supervisor: Amund Skavhaug
June 2021

Hege Rishovd & Ane Sofie Smith Kristiansen

Improvements of the shuttlecock launcher robot BADDY

Master's thesis in Engineering and ICT
Supervisor: Amund Skavhaug
June 2021

Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering





Kunnskap for en bedre verden

DEPARTMENT OF MECHANICAL AND INDUSTRIAL ENGINEERING

TPK4960 - ROBOTICS AND AUTOMATION, MASTER'S THESIS

Improvements of the shuttlecock launcher robot BADDY

Authors:

Hege Rishovd

Ane Sofie Smith Kristiansen

Supervisor:

Amund Skavhaug

June, 2021

Preface

We would like to thank our supervisor Amund Skavhaug for all the help, feedback, and support he have provided us with during this project, and for the loan of equipment.

Thanks to Benoit Greslebin, founder of BADDY, who provided us with useful information and help through interviews and the troubleshooting of the BADDY launching problem with us.

We want to thank our fellow students that helped with some fun testing of BADDY. Jostein Løwer for giving us an Arduino Nano board and some good advice, Kristoffer Opsahl for helping us with JSON expertise, and Jørgen Rishovd for giving feedback on the report.

This particular project was chosen because we had some experience with badminton and were curious about building robots. We wanted a clearly defined project with few limitations to what could be done. Making the remote controller appealed to us because we wanted to gain knowledge about both hardware and software development, perform 3D printing, and having a high chance of realizing the project. The work of this thesis can easily be extended and developed further by others.



Hege Rishovd



Ane Sofie Smith Kristiansen

June 2021

Executive summary

This report describes the development process of improving an already existing badminton shuttlecock launcher robot, named BADDY. To gain necessary background information, an interview with the founder of the BADDY project, and a literature study have been performed. In a prestudy, different improvements of the robot were considered and discussed with regards to three focus areas; security, performance, and ease of use. This report is based on further development of one of the presented improvements, which was making a remote controller for controlling the BADDY robot. This was done to replace the need of using a smartphone when playing badminton.

To make the remote controller, three different design processes were performed. These were case designing, hardware development, and software development. A working prototype has been developed using 3D printing, Bluetooth Low Energy (BLE) technology, and Arduino coding.

In addition to this, a complete BADDY robot was built, tested, reviewed, and modified to work with the remote controller. The mounting process of the robot has been carefully reviewed and explained throughout this report. The BADDY code was reviewed and possible ways to improve the code have been discussed.

The BADDY robot can help improve the quality of badminton training, especially for young players, and players who are in their learning phase. The remote controller will add a new layer of ease of use to BADDY, and the making of a controller will also add new aspects of robot building to the BADDY project.

Sammendrag

Denne rapporten tar for seg en utviklingsprosess og forbedring av en allerede eksisterende robot, kalt BADDY. BADDY er en badmintonrobot som skyter ut fjærballer. Den nødvendige bakgrunnsinformasjonen har blitt hentet inn ved bruk av ulike metoder. Et intervju med grunnleggeren av BADDY-prosjektet, og en litteraturstudie har blant annet blitt gjennomført. Mulige forbedringer av BADDY innen tre hovedområder; sikkerhet, ytelse og brukervennlighet, har blitt undersøkt og diskutert i en tidligere studie. En av de foreslåtte forbedringene i denne forstudien var å lage en fjernkontroll som kunne kontrollere BADDY. Denne rapporten tar for seg videreutviklingen av en slik fjernkontroll, samt implementasjonen av denne. Formålet med en fjernkontroll var å erstatte behovet for å holde en stor smarttelefon i hånden når man spiller med BADDY-roboten.

Utviklingen av denne fjernkontrollen ble gjennomført i tre ulike prosesser. Disse var design av selve fjernkontrollen, utvikling av maskinvaren og programvareutvikling. En fungerende prototype ble utviklet og ferdigstilt ved bruk av 3D printing, Bluetooth lavenergiteknologi (BLE) og programmering i Arduino.

I tillegg til å utvikle en fjernkontroll, ble en BADDY-robot bygget, testet, evaluert og modifisert for å kunne fungere sammen med fjernkontrollen. Den eksisterende koden som kan hentes fra BADDY-prosjektets GitHub side, ble gjennomgått og mulige forbedringer av koden har blitt diskutert.

En BADDY-robot kan bidra til å forbedre kvaliteten på badmintontreninger, spesielt for unge spillere og nybegynnere. Brukervennligheten til BADDY kan forbedres ved å ta i bruk fjernkontrollen, i tillegg vil implementasjonen av en fjernkontroll legge til nye aspekter ved robotbygging i BADDY-prosjektet.

Table of Contents

List of Figures	vii
List of Tables	xi
1 Introduction	1
1.1 Motivation	1
1.2 Objectives	2
1.3 Problem description	3
1.4 Limitations	4
1.5 Report structure	4
2 Background material	6
2.1 Badminton	6
2.2 The BADDY robot	8
2.3 Construction of BADDY	10
2.4 Previous work	13
2.5 Bluetooth and Bluetooth Low Energy technology	13
2.6 HC-08 Bluetooth module	14
2.7 ESP8266 WiFi module	15
2.8 Arduino technology	16
2.9 3D printing	17
3 Method	19
3.1 Interview	19
3.2 Literature study	19
3.3 Previous work	20
3.4 Mounting, testing, and troubleshooting	20
4 Building and initial testing of the BADDY robot	22
4.1 Building BADDY	22
4.2 Testing BADDY	30
4.2.1 Functionality test	30

4.2.2	Observing and fine-tuning of the switch and retainer	32
4.2.3	Performance test	33
4.2.4	Troubleshooting	37
4.2.5	Final test	38
4.3	BADDY code review	45
4.3.1	Code improvements	51
5	Modifications to BADDY	57
5.1	Wiring a push button to BADDY's PCB	57
5.2	Wiring the Bluetooth module to BADDY's PCB	60
5.3	Getting a shuttlecock to launch	63
6	Further implementation of the remote controller	67
6.1	Case design process and improvements	67
6.2	Hardware development	77
6.2.1	Remote controller with Arduino Nano and Nano Every	80
6.2.2	Components	84
6.2.3	Assembling the components	85
6.3	Software development for the controller	89
7	Discussion and conclusion	94
7.1	Discussion	94
7.1.1	The remote controller	94
7.1.2	The BADDY robot	96
7.2	Conclusion	99
7.3	Afterword	100
	Bibliography	101
	Appendix	108
A	Changes done to BADDY's code	108
A.1	Wiring a push button to BADDY's PCB	108
A.2	Wiring the Bluetooth module to BADDY's PCB	109
A.3	Getting a shuttlecock to launch	110

B	Controller code	111
---	---------------------------	-----

List of Figures

1	The first version of BADDY [19]	1
2	A cone-shaped shuttlecock made out of feathers attached to a cork base [45, 91] . .	2
3	A remote controller communicating with BADDY using Bluetooth	3
4	The men’s double final of the 2012 Olympics in London between China and Denmark [87]	6
5	Children waiting in line while teacher is illustrating an exercise [79]	7
6	A Siobasi badminton shuttlecock robot [6]	7
7	BADDY, the open-source badminton robot and the founder, Benoit Greslebin [19]	8
8	The BADDY application available on Google Play Store [90]	9
9	View of BADDY taken from Janton’s CAD model [63]. BADDY consists of a feeder tube (1), fire room (2), and frame (3)	10
10	View of the fire room taken from Janton’s CAD model of BADDY [63]. The inside of BADDY consists of a feeder hole and magnet slots (1), neck (2), servomotors (3), retainer (4), and wheels (5) [19]	11
11	The wheels used inside BADDY [19]	11
12	Illustration of the court coverage provided with BADDY’s 9 different types of strokes.	12
13	The electronic components used in BADDY. The circuit board on the left is a special designed circuit board with Arduino embedded, and the Arduino LED panel is shown on the right [19]	12
14	CAD model of a four-barrel rotary feeder Janton designed for his report [63]	13
15	The HC-08 module [84]	15
16	The ESP8266 board [46]	16
17	Original Prusa i3 MK3S+ 3D printer [17]	17
18	The glue and small screwdrivers used for the mounting process	21
19	The finished mounted fireroom (a), and frame (b)	23
20	<i>PlatformIO</i> opened in <i>Visual Studio Code</i>	23
21	BADDY’s printed circuit board	24
22	Servo motors, motors and battery attached to the fireroom and frame	25
23	The inside of BADDY mounted with all the parts	26
24	Sandpaper used to remove excess material (a), and the power button and charging cable correctly placed on the underside of BADDY (b)	27
25	BADDY with the box	28

26	The parts for the three feet (a), one of the finished assembled feet (b)	28
27	Plastic tube	29
28	Magnets added to BADDY and fastened with a plastic plate on top of BADDY . .	29
29	BADDY finished mounted with feeder and shuttlecocks	30
30	Three shuttlecocks trapped inside the fire room	31
31	The servomotor hanging in the air (a), and a temporarily solution with a piece of cardboard under the motor (b)	32
32	BADDY placed at the intersection between the short service line and the center line (a), and the incline of BADDY (b)	34
33	Recommended maximal inclination (a), and BADDY with decreasing inclination for straight drives shown in a YouTube video [24] (b)	35
34	Illustration of the court coverage provided with BADDY's 9 different types of strokes.	35
35	Due to the vibrations from the motors, a screw from the underside of BADDY loosened	37
36	Wheels mounted unevenly	38
37	Placement of the shuttles from the first tube with new shuttles	39
38	Placement of the shuttles from the second tube with slightly used shuttles	40
39	Placement of the shuttles from the third tube with the oldest shuttles	40
40	Placement of the left, right and middle drop shots circled in	41
41	BADDY placed on the tripod with an uneven weight distribution, leaning tho the left	42
42	Placement of the left drops (at the bottom), drives (in the middle) and clears (at the top)	43
43	A player using the remote controller in the left hand	44
44	The hand doesn't need to be closed for the controller to stay safe in the hand . . .	44
45	The weather conditions in Trondheim for the third test, on March 10 [92]	45
46	The weather conditions in Trondheim for the fourth test, on May 13 [93]	45
47	Folder structure for the BADDY files	46
48	Simple illustration of BADDY's code flow	47
49	Flowchart representing BADDY's code	48
50	Five string formatting functions declared in the BADDY code. <i>urldecode()</i> uses <i>h2int()</i>	49
51	Functions used in the setup-function	50
52	Functions used in the loop-function	51
53	SPIFFS warning	52

54	Wired master circuit from the prestudy	57
55	Indication of used and available pins on BADDY's ESP board. Original illustration made by Circuitspecialists [40]	58
56	Circuit diagram for the button and ESP8266 to the left, and wired circuit to the right	59
57	BADDY is smiling when the button is pressed	59
58	The button is wired with a short time solution to the ESP board, one wire to ground (yellow) and one wire to D4 (green) where internal pull-up for D4 is enabled	60
59	The HC-08 module is working with the ESP board	61
60	BADDY is smiling when the button on the controller is pressed	61
61	The HC-08 wired to the ESP board using a breadboard as a temporary solution. One wire to ground (yellow), one wire to D4 (green) and one wire to 3V (green)	61
62	HC-08 module soldered to the BADDY ESP board	62
63	BADDY's ESP board with the soldered HC-08 module placed inside BADDY	62
64	The prototype (a), and the original sketched idea (b)	67
65	A small test print	68
66	The friction surface area is highlighted	68
67	Bigger test print with working closing mechanism where the parts were printed on different printers	69
68	The first prototype	69
69	The first prototype attached to a hand and using the notch to open it	70
70	Close-up of the face with removed support material	70
71	Two different printing jobs, where one case was printed with support structure (left) and the unfinished printed case without support structure (right)	71
72	Alternative 3D drawing	71
73	Sketch of a push button with a spring	72
74	Some of the buttons available	72
75	Combination of 3D printed and manufactured button	73
76	The buttons used, push button to the left and switch button to the right	73
77	The case with the manufactured buttons implemented. A bit of sandpaper was needed to make room for the buttons	74
78	Final 3D drawings	75
79	Dimensions of the case	75
80	The final prototype	76

81	Two possible ways of pressing the launching button	76
82	A button on top of the case can in future development be used to e.g. switch between different modes	77
83	Case development	77
84	Wired master module to the left and slave module to the right	78
85	Illustration of signal transmission with two Arduino boards	78
86	Illustration of the signal transmission between a controller and BADDY. The modules are part of the controller and BADDY	79
87	Circuit diagram for the slave module (a), and the wired circuit with an Arduino Nano and HC-08 Bluetooth module (b)	81
88	Arduino Nano soldered with the 2x3 pin headers on incorrect side	82
89	A desoldering pump [9] (a) and copper mesh used for desoldering [97] (b)	82
90	The Nano Every module wired with all components needed for the controller	83
91	Final circuit of all the necessary components for the controller	83
92	Double coin battery holder	84
93	The buttons used, push button to the left and switch button to the right	84
94	Adding a slide switch to the circuit	85
95	Final circuit of all the components inside the controller	86
96	Soldered loop with battery and switch button placed inside the case	87
97	HC-08 module soldered to a small breadboard with wires	88
98	One of the legs on the LED lamp came loose	88
99	The inside of the final controller	89
100	Folder structure for the Arduino files used for the controller	90
101	Choosing the correct board in the <i>Arduino IDE</i>	93
102	BADDY used for fun together with fellow graduation students	100

List of Tables

1	Pros and cons for the different hardware solutions	80
2	Component data	85

1 Introduction

BADDY is the name of an ongoing, open-source robotics project. The project includes both the hardware and software required to build a badminton shuttlecock launcher, named BADDY, shown in Figure 1. There exists a worldwide BADDY community, where people share personal experiences and problems they have encountered with their BADDY robots. The community encourages others to mount, improve, and develop their own robot. The documentation is open and the source code is free to use and modify.

The authors of this thesis have taken part in the BADDY community and developed a product that could improve BADDY. The purpose of this project was to give something back to the community and provide both new and former developers with useful information and help to continue developing the robot. This report documents the development of a new product based on an existing project, as well as the experiences of familiarizing oneself with previous work.¹



Figure 1: The first version of BADDY [19]

1.1 Motivation

In badminton, you want to practice your stroke. The easiest way to do so is to get a shuttlecock thrown perfectly at you repeatedly. A big problem in badminton versus other sports, like tennis, is the shape of the ball. In tennis, the ball is spherical with the center of mass in the middle of an even, relatively heavy ball. In badminton, a shuttlecock is used. A shuttlecock is made out of a cork base with feathers attached, forming an open, conical shape, as shown in Figure 2. This kind of ball is light, with a total weight of 5 grams [91], and hard to throw because it needs a high output speed of approximately 100 km/h. A human will never be able to reach that speed by throwing the shuttlecock, and it can therefore be an advantage to have a robot that can do just that.

¹This chapter include parts from the authors' own prestudy report [67]



Figure 2: A cone-shaped shuttlecock made out of feathers attached to a cork base [45, 91]

Another problem appearing in badminton is that two beginners are not fit to practice with each other. They would lack the basic skills needed in order to hit the shuttlecock so the other player could receive it. This usually leads to either the need for one trainer for each player, or inactivity among the waiting players. To solve both of these problems, monotonous repetition of a specific stroke and teaching new players, a badminton shuttlecock launcher machine could be used.

Many different shuttlecock launcher machines exist on the market today, but most of these are big and expensive. In order to make the badminton machines more accessible to regular people and badminton clubs, an individual company called SPORTVATION, have developed a low-cost, shuttlecock launcher robot, called BADDY, and are selling “do it yourself” (DIY) kits so you can make your own robot [19].

BADDY is continuously being improved and changed, and there are many different implementations and additions that could improve BADDY. The prestudy for this report looked at different aspects of BADDY, found potential areas for improvement, and started the development of a solution for one of the suggested improvements, which was to create a remote control. The remote controller has been further developed and a working prototype has been documented and presented in this report.

1.2 Objectives

The objective of this study is to build and develop the already existing BADDY shuttlecock launcher robot, with the results from the prestudy in mind. To achieve this, a BADDY robot shall be built and carefully documented along with the possible pitfalls one may encounter along the way. This would provide a basis for a review of the product and a better understanding of the embedded source code. The BADDY project lacks documentation, so missing documentation for both the existing project and new features will be written and presented in this report.

Sufficient knowledge and background theory have to be acquainted in order to gain a better understanding of open-source projects and how to further develop the work of others. An essential part of this background material will be to get acquainted with previous work that has been done to BADDY.

The goal for this project is to come up with and develop a working prototype that can improve the functionality of BADDY, and possibly be included in the open-source BADDY project in the future. In order to achieve this, the three main goals of the original BADDY project have to be

maintained.

- Open-source
- Affordable
- Sustainable

This means that when the developed improvement, described in this report, is integrated with BADDY, you can still make and build BADDY at home at a low cost without the need for advanced or expensive equipment. In order for other users to benefit from the solution presented in this report, it has to be shared with the community. This can be done either by posting it to the forum as a private person or by getting it approved by the founders and added to the official open-source project.

In the prestudy, one of the suggested improvements was chosen to proceed with. This improvement was to build a remote controller that could replace the need for a smartphone when communicating with BADDY. The main goal for this project is therefore to further develop the remote controller and get a working prototype that can communicate with BADDY. BADDY's code and software have to be reviewed and the necessary changes have to be made in order to achieve the desired functionality for the controller. This functionality would be to make BADDY launch a shuttlecock when a button on the remote controller is pressed. An illustration of a remote controller communicating with BADDY using Bluetooth is provided in Figure 3.

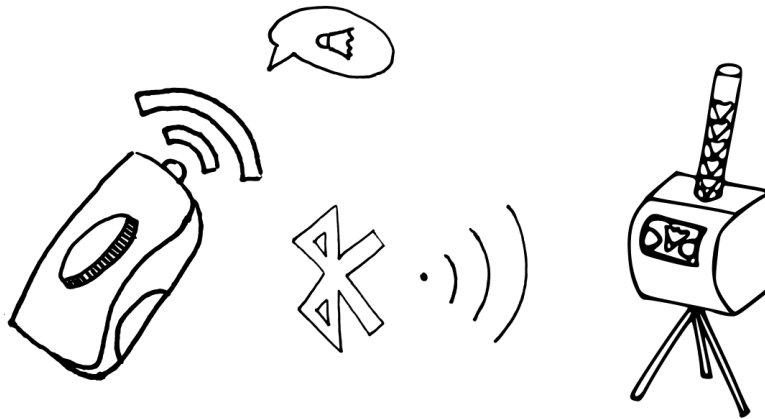


Figure 3: A remote controller communicating with BADDY using Bluetooth

1.3 Problem description

Contribute to the BADDY community by further developing the project. Improve the BADDY robot within the original goals of the project, and contribute to increasing the quality of the project. The information is spread across several platforms and is hard to obtain. Gathering this information will make it easier for others to familiarize themselves with previous work. With the prestudy in mind, develop a remote controller, that will improve training with BADDY. To do this, a BADDY robot has to be built for testing and development purposes. The process has to be documented and consists of the following steps:

- Detailed instructions for the mounting process of a BADDY robot, along with possible pitfalls

one may encounter and how to avoid and solve these.

- A review of the original source code, along with recommended improvements.
- Gathering of information about the existing BADDY project, historical data, and hardware components in order to find possible ways to improve the robot.
- Documentation of the gathered information.
- Development of a remote controller by performing design, hardware, and software processes.
- Documentation and detailed instructions for making the controller so that others easily can familiarize themselves with the work.
- A discussion as well as recommendations for further work on the BADDY project.

1.4 Limitations

In consultation with the supervisor, this report has been limited to study the BADDY project within the focus areas of the specialization course, TPK4960 - Robotics and Automation, Master's Thesis, at the Norwegian University of Science and Technology. This is a 30 credits course that focuses on knowledge of robotics, development, and implementation of robotic systems and mechatronics [75]. The course lasted 20 weeks during the spring of 2021. During this time period, the worldwide pandemic, COVID-19, influenced the whole society. The pandemic resulted in several restrictions, of which the use of labs was limited. Both the equipment and workspace needed were affected and less available for use. In addition to this, the delivery time for ordered equipment was also increased. These were all factors that affected the work and progress of this report.

This report was written within some predefined limitations. The remote controller developed and reviewed throughout this report was not intended as a commercial solution, rather as a prototype. It was developed to demonstrate the feasibility and usability of replacing a smartphone with a remote controller. Optimization has not been a priority, so areas like total cost, performance, and functionality can be improved.

1.5 Report structure

The target group of this report is, among others, people that want to build BADDY, develop it further, or use the same technology used in the BADDY project. As well as people with interests in open-source projects, development processes, or badminton in general. Due to this large and wide target group, it is not expected that the reader is familiar with all aspects of building and developing a badminton robot. Therefore, section 2 presents the necessary background information for people without prior knowledge within the areas of 3D printing, Arduino technology, and Bluetooth Low Energy. Information about badminton in general, the BADDY project, and the specific modules used, have also been included in this section.

Section 3 presents a review of the different methods that were used to gather information and knowledge. These were interviews, literature study, previous work, testing, and troubleshooting.

Section 4 goes into depth on the mounting process of BADDY and the different tests that were performed. This section can be used as guidance for others building BADDY, with detailed pictures

and information about mistakes that are easily made. A review of the source code is also provided in this section, so other developers can more easily understand the code and/or improve it.

A description of the modifications done to BADDY is provided in section 5. The different steps of the development and the final solution are carefully reviewed and discussed. The solution is presented and explained so others can modify their BADDY with the same extension.

Section 6 is a detailed description of the development process for the remote controller. All choices made during this process, are discussed separately as they appear, in order to help with readability and to provide an understanding of why the different choices were made. This section can be used as a guidance to create and mount the remote controller, and/or develop it to suit your own needs.

At the end of the report, in section 7, an overall discussion and conclusion are presented, along with suggestions for future work.

2 Background material

This section addresses background information essential for understanding the content of this study, as well as the choices that have been made. It includes information about badminton in general, the BADDY project, 3D printing, Arduino technology, Bluetooth Low Energy, and the modules used. ²

2.1 Badminton

Badminton is a racket sport played by people all over the world. The sport is played by either two or four players on a court which is divided by a net. It is “the fastest racquet sport in the world” [39] where shuttles can reach a speed of more than 400 km/h [73]. Badminton uses a special “ball” that is called a shuttlecock. This is shaped like a cone and is made out of feathers attached to a leather-drawn cork tip. The combination of short, high-intensity rallies and longer rallies with moderate intensity make badminton a challenging sport. It requires good technique, good reactivity, flexibility, strength, and endurance. A match is played as best of three sets, where each set is played to 21 points. To win a match, a player or team has to win 2 sets of 21 points each [81].



Figure 4: The men’s double final of the 2012 Olympics in London between China and Denmark [87]

A well-known problem for beginners is the lack of technique and the ability to keep a shuttlecock in play for a longer period of time. In order to develop the correct, and necessary technical skills to later be able to hit hard, either with long clear shots or smashes, a player must practice hitting shuttles that approach the player at a height of at least 0.5 meters above his own height. A beginner will find it difficult to send a shuttle in a controlled way to its teammate so that they can practice different strokes precisely because of the lack of technique. This will not be beneficial for either of the players since neither of them will improve their game. This is often solved by the coach hitting the shuttlecocks towards one player at a time, often named “feeding”. This leads to inactivity among the other players. Figure 5 shows inactivity among the children as the coach is illustrating an exercise.

²This chapter includes parts from the authors’ own prestudy report [67]



Figure 5: Children waiting in line while teacher is illustrating an exercise [79]

“In teaching badminton one of the lesson objectives would be to keep a shuttle in play at the beginning level, but this is not an easy task for beginners. Because of unpredictable shuttlecock placements, beginners may have difficulty keeping the shuttle in play” [66]. A badminton shuttlecock training machine that launches shuttlecocks with high precision and accuracy would be suitable for this problem. By browsing today’s market, it is possible to find a wide range of different shuttlecock launcher machines, but many of these are expensive and difficult to handle in terms of weight and size. It is possible to purchase robots in different price ranges and with different specifications, but the cheapest ones, comes at a price of around 1000 US dollars. Figure 6 illustrates a Siobasi badminton shuttlecock robot, one of the cheaper and more lightweight robots one can purchase today, and comes at a price of 1620 US dollars (whiteout shipping and taxes). The robot has a total weight of 28 kg, where the tripod in itself is 145 cm tall [6]. This is a high-performance robot that comes with among other things, a built-in power supply system, wide range of shots and functions, and a remote controller. Due to the dimensions of the robot, it is difficult to transport and set up. A more affordable and sustainable alternative exists in BADDY, which was developed to make such a robot more accessible to the public. This is a robot that you can mount and maintain for a small amount of money.



Figure 6: A Siobasi badminton shuttlecock robot [6]

2.2 The BADDY robot

BADDY is an open-source badminton shuttlecock launcher and can be seen in Figure 7, along with the founder of the project, Benoit Greslebin. It was created by a team in France, with the purpose of creating a training partner for everyone to use. The developers had three main goals for the project. First, it should be accessible to all. The software and code behind BADDY are open-source, which means that it is publicly accessible, and can be both modified and used by anyone. Secondly, it should be affordable. As discussed in 2.1, there are several shuttlecock launcher robots on the market today, but these are all expensive with a cost anywhere from 1000 US dollars and up. BADDY was created as an alternative to these expensive robots, therefore the cost had to be significantly decreased. Finally, the robot was meant to be sustainable. If any parts were damaged, it should be possible to fix them at home. Either by purchasing new parts or by 3D printing or laser cutting them. By being able to fix the robot, it is easy to maintain it. The user only has to change the broken part locally instead of having to submit the entire robot for service. This makes BADDY more sustainable both for the user and the environment. The user won't have to spend money and time submitting it, and the environment is spared the potentially unnecessary pollution that comes with the shipment. The total cost of purchasing a full BADDY DIY kit is less than 500 USD. The BADDY founders have calculated that the average cost of maintaining BADDY is about 50 USD a year, used for replacing old and worn parts with new ones [20]. The maintenance can be performed by the owner. BADDY is also quite small with the dimensions of 25 x 20 x 20 cm without the feeder [26], which makes it portable and easy to bring along.



Figure 7: BADDY, the open-source badminton robot and the founder, Benoit Greslebin [19]

In addition to BADDY being open-source, there is also a BADDY community. People from all over the world come together via Facebook or Discord to exchange tips and tricks regarding the use of BADDY, mounting, and general questions.

It is possible to purchase two different DIY kits. One, where all parts are provided and the buyer only has to mount the robot, and one where some parts are not included and have to be 3D printed or laser cut by the buyer. The later kit is cheaper than the full DIY kit and comes at a cost of 265 USD [25]. The design files and software are available on BADDY LAB's Github page [29], and the BADDY team has created and made several "follow along" videos available on their YouTube channel [27].

BADDY is embedded with a WiFi microchip, ESP8266, which turns BADDY into a portable

WiFi web server where BADDY works as an access point (called AP mode). This means that when BADDY is turned on, a free BADDY network without a password is created. In order to use BADDY, it is necessary to connect to the network with a WiFi client, either with a smartphone or a computer. For the earlier firmware versions, this was the only way to communicate with, and use BADDY. The latest version, had in addition to WiFi, also been embedded with hotspot mode. This allows for a user to set up a WiFi hotspot from a smartphone and connect BADDY directly to this. Hotspot mode was included in the latest firmware update in order to make it more straightforward to connect two or more BADDYs together. Even though the newest version was embedded with both WiFi and hotspot, WiFi was only used for configuration purposes, not controlling BADDY. In order to use BADDY, it has to be connected to a hotspot mode even when only one BADDY is used. The WiFi connection modes are carefully explained in a video on BADDY's YouTube channel [23] and how to set up a hotspot network is shown in another video [22].

Along with the robot, there exists a mobile application called BADDY that is available for both iOS and Android. The app allows the user to create playing sequences where the fire rate, type of shot, placement on the court and more, can be both defined and changed. Some of these features can be seen in Figure 8. Where Figure 8a illustrates how the timeline of a playing sequence can be edited considering the time, in seconds, until the next shot in the sequence is launched. Figure 8b illustrates how a sequence with shots can be made and Figure 8c gives a complete overview of the created sequence with the different strokes selected and the time between the shots. The newer version of BADDY, BADDY V2, communicates with the app through a WiFi connection, different from the first version that used Bluetooth.

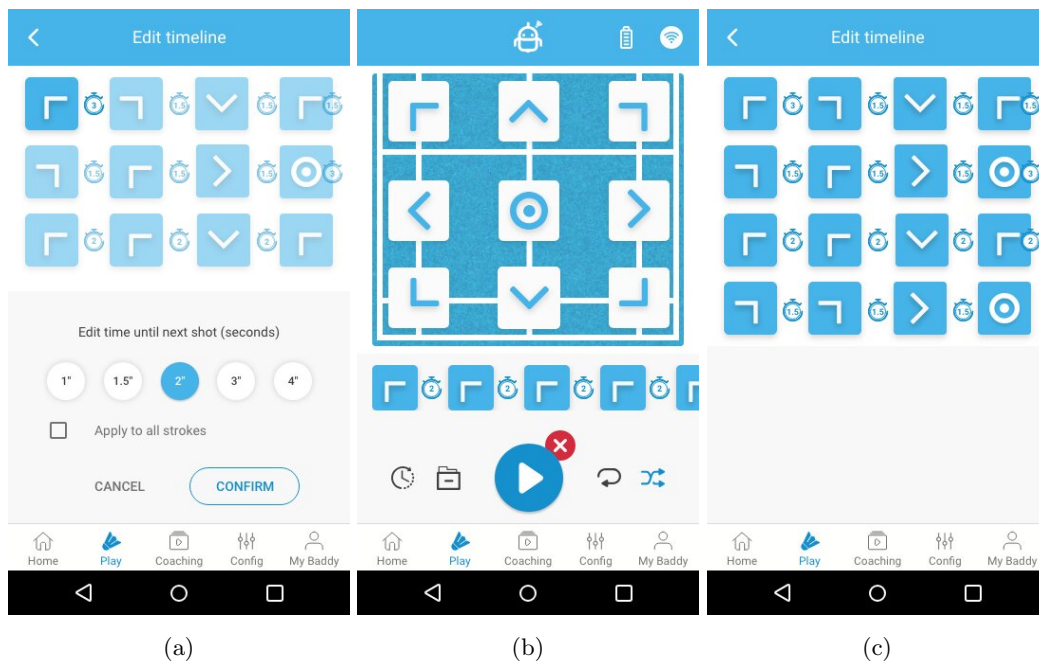


Figure 8: The BADDY application available on Google Play Store [90]

Considering the three main goals and intentions for the BADDY project, accessible, affordable, and sustainable, the use of an application supports an affordable solution. Most people already have a smartphone, and if not, multiple cheap options exist on the market. It is preferable for the user to hold the phone while playing in order to control BADDY from the app. This may be a problem, especially considering the size of smartphones. If the phone is too big, it may compromise

the quality of the practice, and there would be a risk of dropping the phone while playing. When using the application to control BADDY, it would require the player to look at the phone. This implies that the player would have to stop the session, and the use of a smartphone would therefore disturb the training even more.

2.3 Construction of BADDY

BADDY consists of three main components, the "Go Baddy" application presented in section 2.2, the robot itself, and the electronics. The robot itself consists of multiple components. These components are mainly the frame, the fire room, and the feeder tube, and can be seen in Figure 9. The feeder is placed on top of BADDY and is a long tube that holds the shuttlecocks that feed the launcher. It has a capacity of 30 shuttlecocks. The frame consists of a box and holds BADDY together. The frame can be made of different types of material, where both MDF and plastic have been used. This is a part that can be 3D printed or laser cut by the user itself.

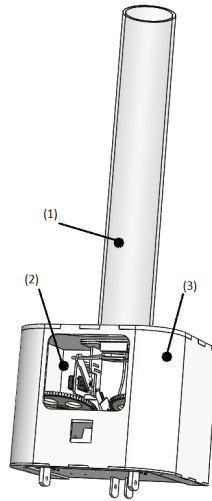


Figure 9: View of BADDY taken from Janton's CAD model [63]. BADDY consists of a feeder tube (1), fire room (2), and frame (3)

The fire room is located inside the frame, and this room contains a firing system that launches the shuttlecocks in addition to the motors and electronics for the robot. The firing room consists mainly of a neck, switch, retainer, and launcher, as seen in Figure 10. Both the switch and retainer are operated by servo motors, which again control whether a shuttlecock can be launched or not. These parts can either be in an open or closed position. When they are closed, no shuttlecock will be let through to the launcher. Once they are opened, a shuttlecock will be led from the feeder, towards the retainer and switch, before it reaches the launcher. The launcher is provided with two rotating wheels with separate motors that can spin up to 5000rpm. This allows for the two wheels to rotate at different speeds, which is necessary in order for BADDY to launch different types of shots. The wheels can be seen in Figure 11.

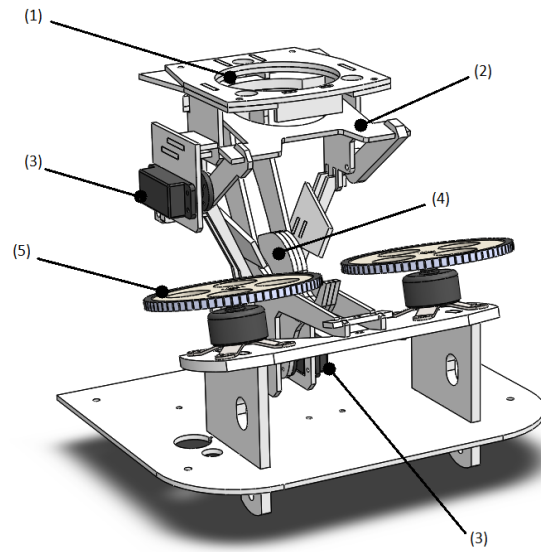


Figure 10: View of the fire room taken from Janton's CAD model of BADDY [63]. The inside of BADDY consists of a feeder hole and magnet slots (1), neck (2), servomotors (3), retainer (4), and wheels (5) [19]

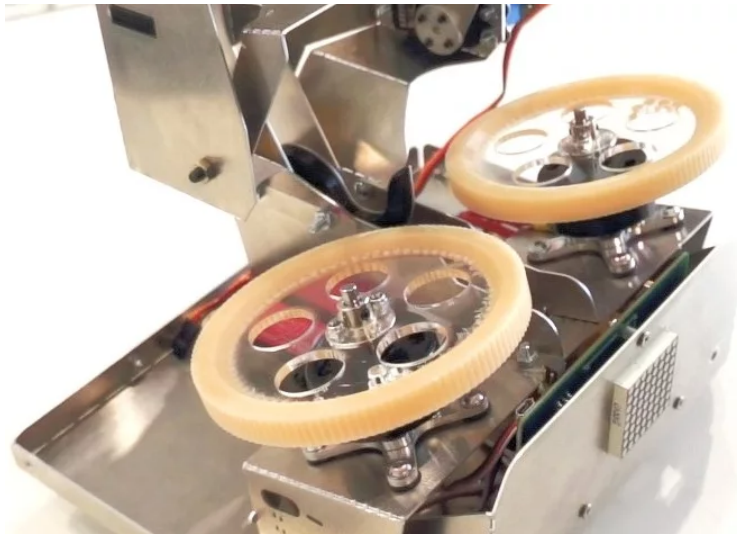


Figure 11: The wheels used inside BADDY [19]

BADDY is provided with 9 different strokes illustrated in Figure 12. These are drops, drives, and clears, where all of these can be fired in 3 different directions, left, center, and right.

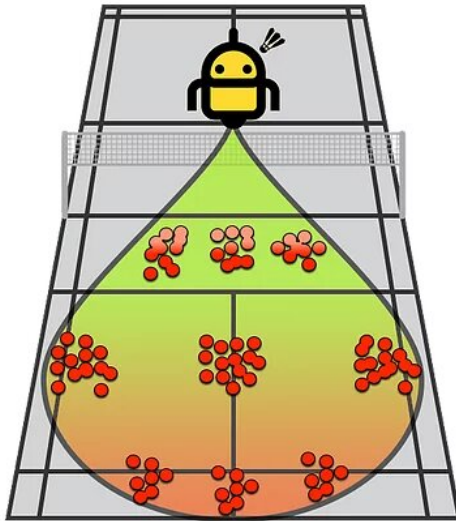


Figure 12: Illustration of the court coverage provided with BADDY's 9 different types of strokes.

BADDY is equipped with a Printed Circuit Board (PCB) and an Arduino LED panel, as shown in Figure 13. The circuit board holds an ESP8266 WiFi-based Arduino board with different connectors. To provide power to the servomotors and the motors for the wheels, they have to be connected to the circuit board when mounting BADDY. The neck and retainer must be positioned correctly. BADDY also comes with a 12 V Li-ion battery and a charger. The LED panel is placed in front of BADDY and is used for giving feedback to the user. It can notify the user when the battery level is under a critical threshold and BADDY needs to be charged, if something happens to the network connection, as well as other things.

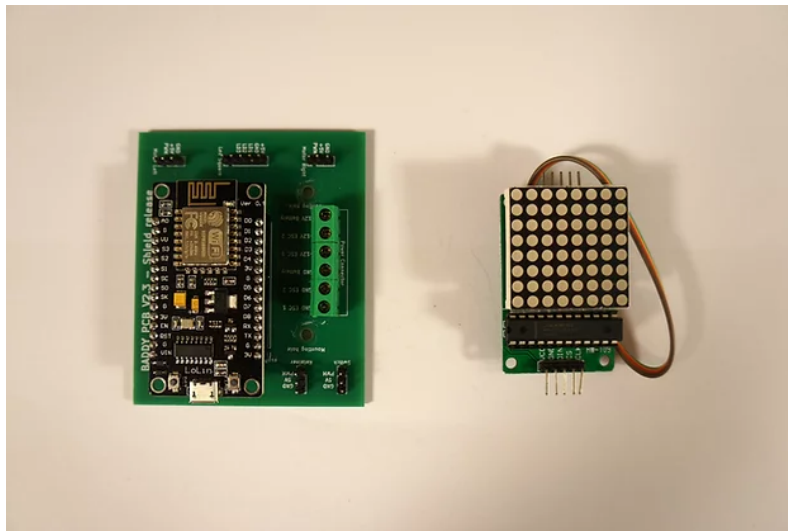


Figure 13: The electronic components used in BADDY. The circuit board on the left is a special designed circuit board with Arduino embedded, and the Arduino LED panel is shown on the right [19]

2.4 Previous work

It doesn't exist any official reports or documentation on the BADDY project other than the website [19]. Except, a report from 2019 written by Hugo Janton [63]. It addresses, at that time, the latest version of BADDY and possible ways to improve the robot. The main aspect of his report was that the suggested improved design of BADDY should be both affordable, transportable, and usable for everyone. As documented in his report, several elements were considered for improvement where three of these improvements were suggested and are listed below.

1. Designing a new firing system that would handle variations in the initial angle
2. Increasing the capacity of the feeder to allow for an increased amount of shuttlecocks
3. The implementation of a safety system in order to create a safe environment around BADDY

An example of his work is shown in Figure 14. Janton used the .dxf files for laser cutting parts, available on BADDY LAB's GitHub, to create his own CAD files.

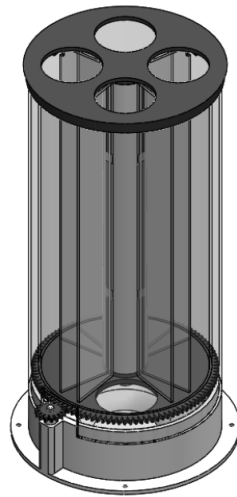


Figure 14: CAD model of a four-barrel rotary feeder Janton designed for his report [63]

2.5 Bluetooth and Bluetooth Low Energy technology

The first version of BADDY used Bluetooth to communicate with the smartphone application, but this was replaced with WiFi for the next version. For a wireless remote controller, it would be preferable to re-implement the use of Bluetooth, but with a new technology that is more optimized for the use of a remote controller.

”Bluetooth is a wireless technology that allows the exchange of data between different devices” [64]. It is a personal area network that works within a short range and allows for communication between different devices [71]. Bluetooth Low Energy (BLE) was released in 2010 as a part of the core specifications of Bluetooth 4.0. BLE is not an upgraded version of the original Bluetooth, but rather a new technology that focuses on applications and devices that are related to, and uses the Internet of Things (IoT) [49]. These applications only transfer a small amount of data with low speed, which implies that they require less energy than other devices to transfer data. The

technology is mainly targeting devices that communicate within a short range of each other and periodically transfer data [3]. One of the features of BLE devices is that they are kept in a sleep state most of the time. This reduces the power consumption of the device to a minimum. When an event, or update, takes place, the device awakens and transfers a short message before it goes back in sleep mode. “The active power consumption is reduced to a tenth of the energy consumption of classic Bluetooth” [86]. BLE is therefore optimized for i.e. home automation applications, fitness trackers or smartwatches, which all require low energy and only transmit small amounts of data [49].

2.6 HC-08 Bluetooth module

HC-08 is a Serial Bluetooth module. This module was a part of the Bluetooth 4.0 release and is embedded with BLE, which was described in section 2.5 above. HC-08 is a long-distance communication module and can communicate with other Bluetooth devices within the range of approximately 80 meters. The HC-08 Bluetooth module is easy to implement in your own project as it is small in size, with dimensions of 26.9 x 13 x 2.2 mm and can be seen in Figure 15. It uses a CC2540 chip, supports AT commands, and the role and serial baud rate can easily be changed by the user [82].

HC-08 uses a serial interface and UART communication. Universal Asynchronous Receiver/Transmitter (UART) is a protocol for serial communication with the purpose of receiving and transmitting serial data. This protocol transmits data, one bit at a time. The data that is transmitted is organized in packets. Each packet contains one starting bit, a data frame of 5 to 9 bits with the actual data, an optional parity bit for error checking, and one or two stop bits [37]. By using an HC-08 module in the remote controller, it will be possible to transmit various commands between the controller and BADDY, even with only one button on the remote controller. By defining different types of clicks on the button, different data can be sent to BADDY, which will trigger different shots. One single press on the button will send the data '1', and the specified shot for this value will be launched. Other shots or sequences of shots can be launched i.e. by holding the button for a longer time or double-clicking and therefore sending another data value to BADDY. This increases both the flexibility and functionality of the controller and makes it possible for other users to continue developing the product and customize it for their own needs and desires. Serial communication is a low-cost interface that only requires two wires for setup, and it is an interface that is easy to implement. A disadvantage for this type of communication is that the size of transmitted data is limited to 9 bits something that can be problematic if it is needed to send more data at a time. In addition to this, it can only operate with a slave/master system, which means that you can only connect two modules together at a time [37].

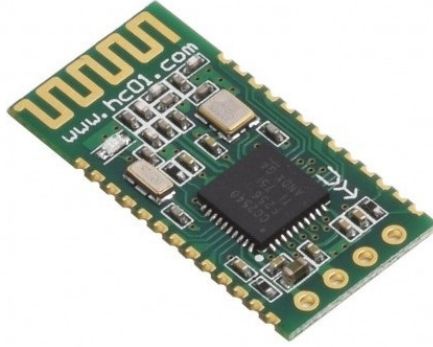


Figure 15: The HC-08 module [84]

In order to get the remote controller to communicate with BADDY, two Bluetooth modules were needed, one in the remote controller and one in BADDY. BADDY’s electronic board consists of an ESP8266 WiFi module, see section 2.7. This module does not support Bluetooth, therefore was it necessary to use a set of HC-08 modules to implement the remote controller. One of the HC-08 modules was connected to BADDY while the controller module, was connected to an Arduino Nano. The small size of the modules made it easy to place them, both to BADDY’s electronics and to create a small enough case for the controller. Due to the UART communication, it was also easy to incorporate the needed code into the already existing BADDY code to make the modules communicate.

2.7 ESP8266 WiFi module

An ESP8266 WiFi module, as shown in Figure 16, is the microchip used in BADDY today. It comes with a microcontroller and TCP/IP stack. The module makes it possible to connect microcontrollers to a WiFi network and allows for communication over the network. It was first produced by Espressif Systems in 2014 [50]. The module is low cost, have few external components which makes it small [33] and has power-saving architecture. This makes it popular for IoT and mobile electronics, like BADDY. ESP8266 can be programmed directly by different SDKs [34], where one option is the Arduino SDK used by BADDY. The ESP8266 module in BADDY is used for making BADDY a network, such that phones can connect and communicate with it.

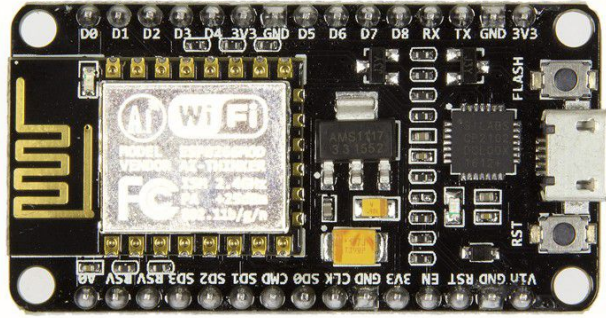


Figure 16: The ESP8266 board [46]

Since this report addresses the process of making a Bluetooth controller, it would be practical if the module integrated with BADDY had Bluetooth in addition to WiFi. A possible upgrade would be an ESP32-C3 Series Modules with both WiFi and BLE [52]. The price difference between this and the ESP8266 would be about 100 NOK if bought in Future Electronics online store [47, 48].

2.8 Arduino technology

The Arduino platform has been developed since 2005 and is used for building digital devices, often prototypes, by using microcontrollers along with different types of sensors and actuators [68]. The platform consists of hardware microcontrollers, the Arduino programming language and the *Arduino IDE (Integrated Development Environment)* for software development. “Arduino is an open-source electronics platform based on easy-to-use hardware and software” [10].

The Arduino software is suitable for both beginners and more advanced users, as it can be adapted to the users’ experience and the complexity of the project [10]. The programming language is a combination of C and C++, and the IDE uses GCC (GNU Compiler Collection) to compile the program. *Arduino IDE* create Arduino files that end with .ino and require the code file to be placed in a folder with the same name.

Arduino IDE can be used with different operating systems such as Linux and Windows. The software is free and it is not required to purchase any license to use it. The programming environment is simple and flexible which makes it suitable for both beginners and more advanced users. It is also a great tool for students and teachers for educational purposes. In addition to these advantages, it is also an open-source software which makes it available for everyone [10].

BADDY is an open-source project that is supposed to be affordable and sustainable. By using the Arduino platform, it supports the affordable and open-source aspect of the project. The software is easy to understand and work with, something that makes it beginner-friendly.

Arduino comes with multiple advantages compared to other microcontroller platforms and it makes the process of working with microcontrollers easier. One of the advantages Arduino provides is the low cost. It is possible to purchase Arduino modules for less than 10 USD [44].

The ESP8266 WiFi module that is used in BADDY is not an Arduino board, but the embedded code for BADDY uses Arduino as a framework. By having an Arduino framework, it “allows

writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces, or physical experiences” [69]. This implies that the ESP8266 board can be programmed as an Arduino, and the module is compatible with other Arduino boards.

2.9 3D printing

3D printing, also referred to as additive manufacturing or digital fabrication technology, is a process of creating a three-dimensional physical object from a digital file. The 3D printer creates an object by layering successive layers of material until the object is complete [2]. The digital file is created using a 3D modeling software. Many different software tools are available online, both free, open source tools and more advanced ones in different price ranges. *Tinkercad*, *FreeCAD* and *Fusion 360* are some examples of free modeling software [1], of which *Fusion 360* is the software that has been used for this project. The 3D model has to be sliced using slicing software that splits the model into many thin layers before it can be printed. *PrusaSlicer* was the program used for the slicing, due to the compatibility with the Prusa printers. A Prusa printer can be seen in Figure 17.

There exist many different 3D printers on the market today. These vary both in terms of price and quality. Budget-friendly printers can be purchased for less than 500 USD [60], while the most expensive 3D printers can be purchased for 2.5 million USD [78].

3D printers are becoming more popular and the interest in these is growing. 3D printers are easily accessible and come with many benefits and advantages. Some of these advantages are that it allows for more flexible and complex designs, parts are produced with strong and light materials, it is cost-effective and accessible. There are, however, also some drawbacks to 3D printing. There are some limitations in regard to which materials can be used and many of the ones that are printable, can not be recycled. Most printers have a small printing chamber which limits the size of the printed parts. The final printed parts can be more brittle than parts constructed using other methods due to the layering structure. In addition to this, there may be problems with the actual printer and inaccurate design [94].

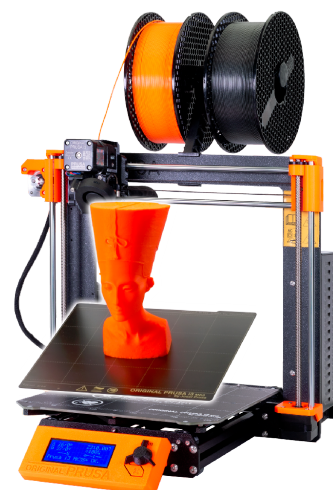


Figure 17: Original Prusa i3 MK3S+ 3D printer [17]

For the BADDY robot, it is possible to print the necessary parts by either using a 3D printer or a laser cutter. To keep BADDY affordable and avoid making the building process more complex or complicated, the remote controller should also be made by one of these. A 3D printed remote controller would be preferable as it is easy to use, and they are easier to obtain than laser cutters.

3 Method

This chapter addresses how the information provided in this report has been gathered, which research methods have been used, and how they were conducted. This chapter also addresses why the following methods were used. ³

Several methods were used to provide the necessary knowledge in order to improve BADDY. A prestudy was made to obtain the necessary background knowledge. The prestudy was conducted based on research and a feasibility study due to little, to none, prior knowledge about the BADDY project. In addition to this, an interview with the founder was arranged and another student's previous project report [63] was examined.

When going through the background material, the candidates have become familiar with aspects of badminton, problems that arise during the learning phase, and the equipment needed to play. The construction and operation of BADDY have been carefully examined, as well as the history and motivation behind the project have been examined more deeply. Knowledge regarding the use of Bluetooth Low Energy (BLE), has been acquired and the benefits such technology can provide, compared to other wireless technology such as i.e., WiFi or earlier Bluetooth versions. Additionally, knowledge about 3D printing, 3D modeling, and design processes has been obtained.

3.1 Interview

In order to achieve a deeper understanding of the idea and work behind BADDY, an interview with the founder, Benoit Greslebin, was arranged. Due to distance, given his settlement in France, and COVID-19, the interview was conducted using Skype. Prior to the interview, Mr. Greslebin was provided with some questions in order for him to prepare himself for the interview. The interview with Mr. Greslebin addressed different topics around BADDY, such as the background and idea behind the BADDY project, future plans, how to develop BADDY further, the difference between the versions, and why Bluetooth communication was changed to WiFi communication.

On the question regarding WiFi and Bluetooth communication, Mr. Greslebin explained that they had experienced some troubles with the use of Bluetooth communication. When new phones with new Bluetooth technology were released to the market, some problems with the compatibility with BADDY had occurred. It was therefore necessary to upgrade the hardware of BADDY to match the new Bluetooth devices. Checking with supervisors, this should not be necessary, as Bluetooth is backward compatible. This led to some uncertainty of what would be the best hardware solution for this study's improvement, and is discussed in section 7.

3.2 Literature study

A literature study was used to acquire the necessary knowledge regarding BADDY and relevant topics for the study. This was mainly implemented at the beginning of the project, but also throughout the project as new information emerged. A lot of information was available online, so it was important to find and use credible sources. NTNU's digital library service "Oria" was mainly used to find relevant articles and information. Subjective opinions and input have contributed to the study and the end result by introducing new ideas and points of view.

³This chapter include parts from the authors' own prestudy report [67]

The CRAAP test was used to assess the trustworthiness of the sources obtained. “The CRAAP acronym [...] stands for the components of the evaluation process: currency, relevance, authority, accuracy, and purpose” [77]. The CRAAP test is based on a set of questions and guidelines to help evaluate the sources and information collected. Sources were evaluated based on their reliability and validity. The reliability is closely related to whether the information is gathered in a trustworthy way or not. Here, both the publication date and author have been assessed as well as the information has been verified with other sources. The validity of the source says something about whether the information obtained is relevant to the study or not. Finally, the purpose of the source has been taken into account. The purpose of the article or the intent of the author has also been part of the overall evaluation of the source.

3.3 Previous work

Parts of a previous student’s project report were made available and used as inspiration and as part of the research for this study. This was the report made by Hugo Janton mentioned in section 2.4. The report was used to get an insight into what has been done previously and how an open-source project can be further developed by others than the founders.

Additionally, a report was made as a prestudy for this master thesis, *Improvements of the shuttlecock launcher robot BADDY* [67]⁴. The main objective of the prestudy consisted of looking at the already existing BADDY shuttlecock launcher robot and finding possible improvements that could increase the performance of the robot. Some possible improvements were considered before they were evaluated based on several factors. One of the suggested improvements was chosen to proceed with and the subsystem was implemented to the extent time allowed. Finally, the usability of the resulting solution was assessed and evaluated.

During the research of BADDY, and considering potential improvements that could be implemented, three areas were emphasized. These were security, performance, and ease of use. The suggested improvements were compared, and one of the improvements was chosen to be partly implemented within the restrictions and limitations of available material, tools, and time.

The result of the prestudy and information gathered while working with it, have formed the basis for this report. The improvement chosen in the prestudy has been further developed in this thesis.

3.4 Mounting, testing, and troubleshooting

Several tutorial videos are available on Baddy’s YouTube channel [27]. These were used as guidance during the mounting of BADDY. The tutorial videos for BADDY V2 exist in two languages, English, which was used for this thesis, and French. The English playlist consists of twelve videos that guide you through the unpacking and physical mounting of BADDY. They also guide you through the software setup, final check, and how to connect to the finished BADDY. The total video time is around an hour. The actual mounting time is depending on your skill level, but according to the BADDY website, it may take between three to five hours [20].

Different tools were needed to mount BADDY. Prior to the mounting process started, a pair of pliers, a small screwdriver set, and glue were purchased. Figure 71 shows the glue and small screwdriver used. To access all the necessary places inside BADDY, it was necessary to have a

⁴The prestudy report is included in the digital appendix (attached files) as a reference for the reader

shorter screwdriver than usual, as there were several places that were difficult to access. It is an advantage to have access to a workbench since the metal parts can scratch the surface underneath. In addition, you need a tiny hex key, which comes with BADDY.



Figure 18: The glue and small screwdrivers used for the mounting process

The testing of BADDY was performed in several steps, which have been reviewed in section 4.2. Small tests were carried out during the construction phase to make sure each building step was made correct. When the building of BADDY was completed, an initial, basic functionality test was performed. This was done to make sure the assembly of BADDY was correct and that the connection to the phone worked. During this test, a problem with launching several shuttlecocks at the same time was detected.

The further testing of BADDY required big, empty rooms without people (outdoor was not a possibility due to the winter season in Trondheim). Lack of people was necessary for their own safety and to conduct the test without interruption. Hence, the second test was a test of functionality that was conducted in a gym. A big, empty room made it possible to check if BADDY managed to launch the long shots.

The third and fourth tests were performed on a badminton court with a net to test the different types of shots, the flight path, and placement of these in relation to the lines of the court. For these tests, several measuring tapes were used to measure how far the shuttlecocks were shot and the possible deviation from their expected landing position. The third test had several sources of error that may have affected the results, including the usage of old shuttlecocks, the type of shuttlecocks, and not measuring velocity, temperature, air pressure, and humidity. Hence, the fourth test was done, trying to decrease these sources of errors.

To troubleshoot the problems that appeared during testing, YouTube videos were used to make sure that the mounting and setup were performed correctly. When that didn't help, the BADDY forum on discord and Facebook group were used. It seemed like the forum were not so often used as the response time was quite long. Hence, an email was sent directly to BADDY Space and a Skype meeting was arranged. During the Skype meeting with Mr. Greslebin, the problem with multiple shuttles being launched at the same time, was inspected and fixed.

4 Building and initial testing of the BADDY robot

BADDY is, as previously stated, an open-source project where all the parts needed to mount the robot as well as the code files, are either included in a DIY kit or available online. This chapter addresses the actual mounting process of the robot. The different steps that were performed, from the unboxing of the DIY kit, until BADDY was finished and in working condition, are reviewed and discussed. After BADDY was completed, several tests were conducted in order to test different aspects of the robot. The initial objectives of the tests were different, but some problems were addressed and checked in more than one test. To solve some of the problems that occurred during the tests, it was necessary to perform some troubleshooting. This was done in collaboration with Mr. Greslebin through a Skype meeting. Finally, BADDY's source code, which is available on BADDY LAB's GitHub page [29], has been reviewed and discussed at the end of this chapter.

4.1 Building BADDY

An actual BADDY robot was necessary to test and implement the chosen improvements. A BADDY V2 DIY kit was purchased from the online BADDY shop [18]. There was no need to use neither 3D printers nor laser cutters to make the parts as the kit contained everything needed. This made it possible to start building the robot despite the restrictions regarding the use of workshops, given COVID-19.

However, the kit did not include the tools for mounting the robot. The tools needed were different screwdrivers, a pair of pliers, some Hex keys, and glue. A computer with WiFi enabled was also needed to adjust the switch and retainer's positions using the BADDY web server. The computer was additionally used with a micro USB cable enabled with data transfer, to upload the firmware to BADDY's PCB.

BADDY is, as mentioned before, an open-source project. The founders have made all the code behind BADDY available online, and they have created multiple tutorials on their YouTube channel, Baddy [27]. These tutorials were followed throughout the entire mounting process, from unpacking the parcel to assembling the parts and connecting BADDY to a phone.

The first thing that was performed, was the unpacking of the BADDY DIY parcel, get an overview of the different parts, and sort out which parts belonged to the different steps of the mounting processes. Then, the fireroom was built. All the parts were flat upon delivery, which meant that they had to be bent to their correct positions before they could be assembled. The parts for both the fireroom and frame were made out of aluminum and the parts that were to be bent were half-perforated. This allowed for them to be easily bent to their correct positions without the use of any tools. The same process was performed for the frame of BADDY. The parts had to be bent and assembled. Figure 19a is a picture of the finished assembled fireroom and Figure 19b shows the finished frame.



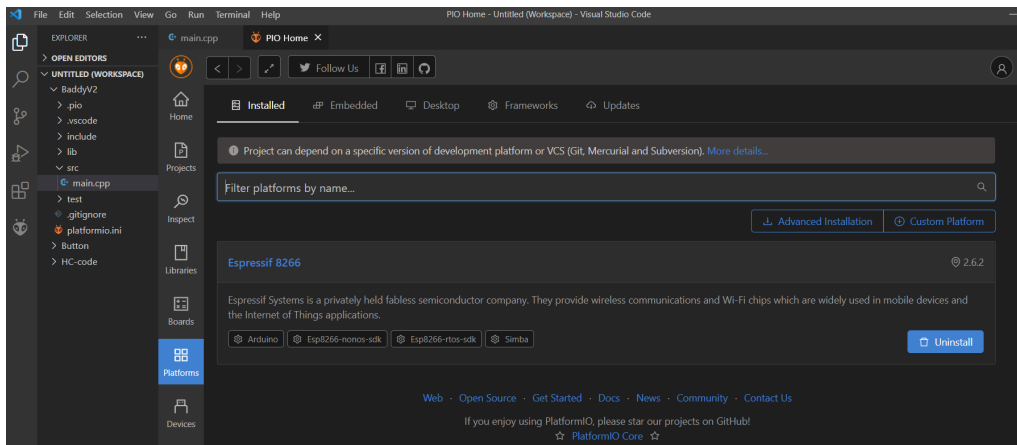
(a)



(b)

Figure 19: The finished mounted fireroom (a), and frame (b)

In order to use BADDY, the correct firmware had to be downloaded and installed. Both *Visual Studio Code* [72] and *PlatformIO* [70] had to be installed on a computer. Figure 20 shows *PlatformIO* opened in *Visual Studio Code* with the BADDY project opened on the left side. A new project was created in *Visual Studio Code*. The name for the project was defined, the board was set to *NodeMCU 1.0 (ESP-12E Module)* and Arduino was selected as the framework. It was necessary to edit two of the files in this project, *main.cpp* and *platformio.ini*. The code needed to implement in these files was provided on BADDY LAB's GitHub page [29] with the same, respective names. Some additional libraries had to be installed as BADDY is referring to, and using, functions from the open-source libraries in order to make different components work. The libraries needed were version 1.0.6 of *LedControl*, version 1.2.0 of *ESPAsyncTCP*, and version 1.2.0 of *ESP Async Web-Server*. The project was then built, and the code uploaded to the BADDY V2 electronic board. The circuit board embedded in BADDY is shown in Figure 21.

Figure 20: *PlatformIO* opened in *Visual Studio Code*

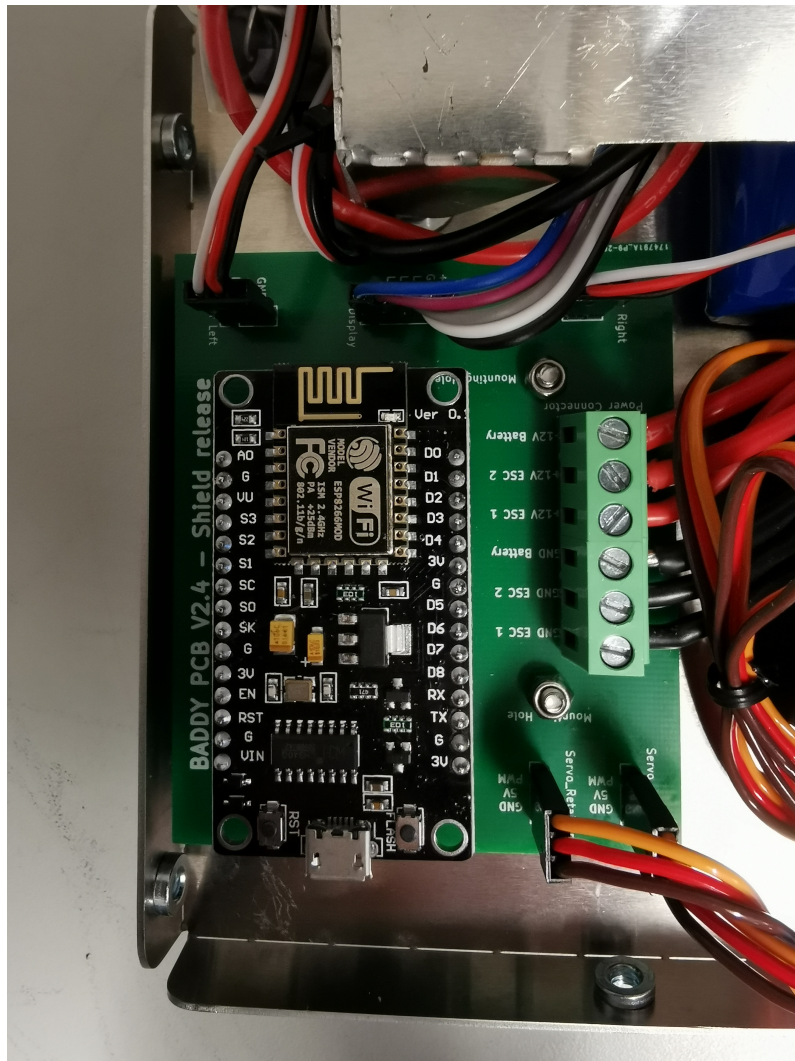


Figure 21: BADDY's printed circuit board

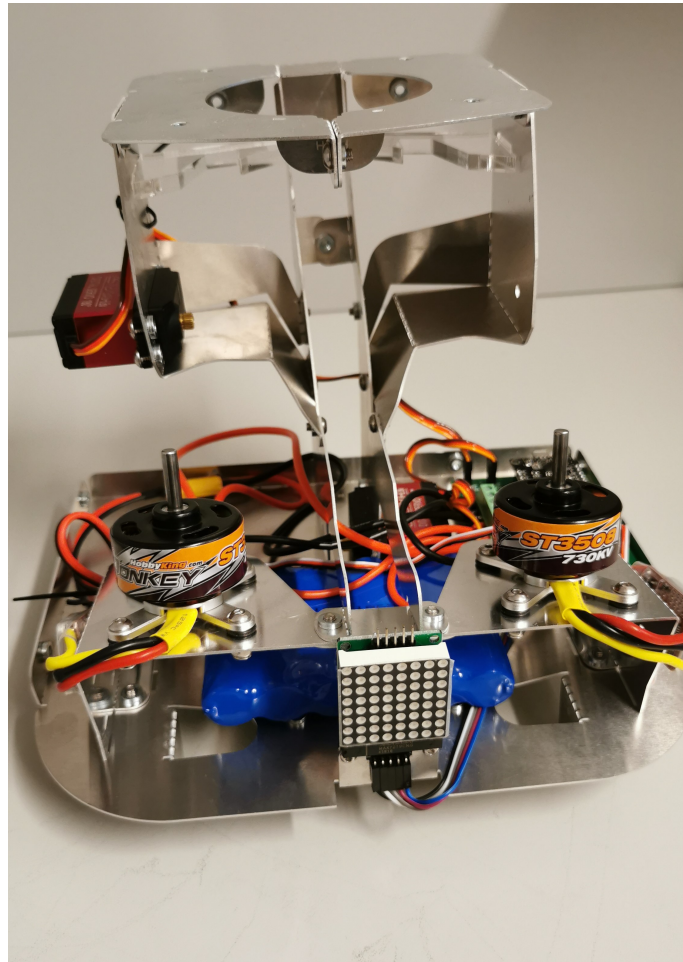


Figure 22: Servo motors, motors and battery attached to the fireroom and frame

Next up were the electronics and motors. The servo motors were attached to their respective places and the wires were fastened using strips. The motors for the wheels were then attached as well as their wires. The different wires had to be inserted into the correct pods on the circuit board. This raised some problems as the tip of one of the wires was soldered in such a way that it did not fit into its respective pod on the board. It had to be bent and formed with pliers to fit. Both the switch, battery, and led display were then placed in their respective places and connected to the circuit board, as seen in Figure 22. It was important to assure that the wires were fastened and positioned in such a way that they didn't get in the way of the wheels or any of the moving parts inside BADDY. The wheels were then placed on the upper side of the motor shaft in order to achieve maximum energy transfer. The complete mounting of the inside of BADDY is shown in Figure 23.

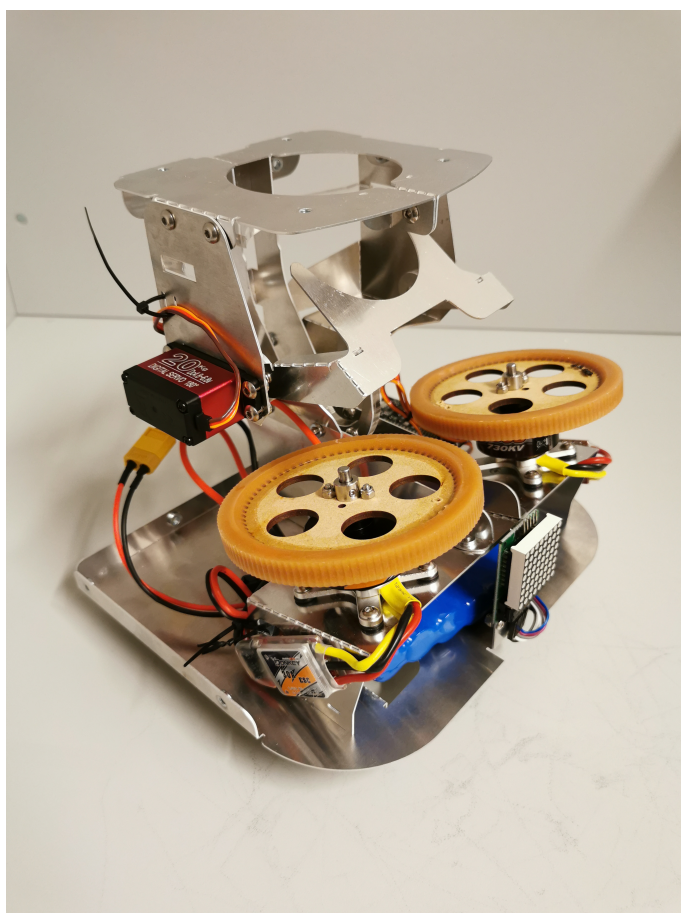
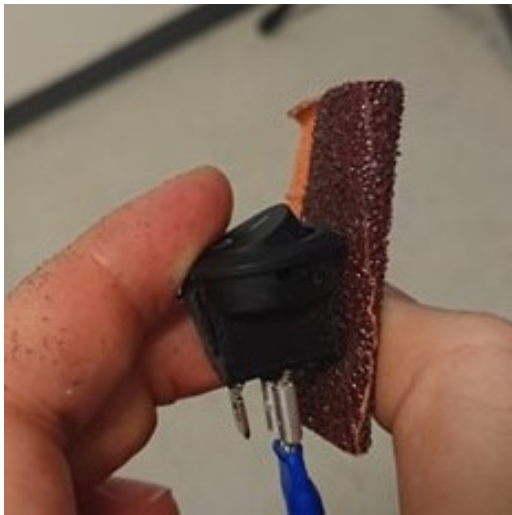


Figure 23: The inside of BADDY mounted with all the parts

BADDY comes with a power button which is necessary in order to turn the robot on and off. This button is placed on the underside of BADDY where there is a hole in the frame. The button has to be fastened from the outside of the frame and the wires connected to the button were threaded through the hole to be able to connect these to BADDY's PCB. When placing the button, it was discovered that the button was too big to fit within the hole and some adjustments had to be made. Figure 24a illustrates how sandpaper was used to sand away any excess material that prevented the button from getting through the hole. The placement of the power button, as well as the charging cable on the underside of BADDY, is shown in Figure 24b.



(a)



(b)

Figure 24: Sandpaper used to remove excess material (a), and the power button and charging cable correctly placed on the underside of BADDY (b)

The retainer and switch are the only things, except the wheels, that are moving inside the robot. These had to be attached to their respective servo motors and fastened using a screwdriver and a pair of pliers. The initial positions of these were adjusted by connecting a computer to the local BADDY WiFi network, see section 2.2, by sending the requests given in Listing 1 to the ESP.

```
http://192.168.1.2/switch_forward  
http://192.168.1.2/switch_backward  
http://192.168.1.2/retainer_up  
http://192.168.1.2/retainer_down
```

Listing 1: Requests used to adjust the initial position of the switch and retainer

By sending these requests, BADDY adjusted the position of the respective part and it was therefore possible to set the correct start positions for the switch and retainer. Here, it is important to notice that because of a mix-up of the wiring that was not noticed until much later in the project, the retainer changed position when using the command for the switch and vice versa. This caused some misunderstanding regarding the names of the parts, and therefore the wrong assumption that the retainer part's name was "switch", and the switch part's name was "retainer". This is important for the second test that is reviewed in section 4.2.2.

BADDY's box and feet are different from the rest of the robot as almost all the parts inside BADDY are made of metal. The box is made of medium-density fiberboard (MDF) or engineered wood [18], while the feet are made of plastic. The mounting of these parts required the use of glue. The parts for the box were made with a "clip-on" mechanism so that the front and top parts only had to be clipped together. The joint between these had to be glued to ensure that the parts were attached to each other. In order to attach the box to the rest of BADDY, it had to be threaded over the frame and fireroom and screwed into the frame, as shown in Figure 25.



Figure 25: BADDY with the box

BADDY has three feet, all of which consist of four plastic parts. Figure 26a shows the parts of all the feet and that they consist of two longer parts and two smaller ones. These had to be glued together to create a solid foundation for BADDY when it is in use. As seen in Figure 26, all three feet have a hole in one of the ends. These holes were used to screw the legs to the underside of the robot. It was therefore important to make sure that the parts were glued together precisely, with all the holes aligned. When the parts were glued together and the glue had dried, they were fastened to the underside of BADDY with a screwdriver and a pair of pliers.



(a)



(b)

Figure 26: The parts for the three feet (a), one of the finished assembled feet (b)

The feeder is, similar to the feet, also made of plastic. This part is not mounted onto BADDY, but attached using three strong magnets. The tube for the feeder is glued to a base consisting of two rings that were screwed together. These screws are magnetic and will interact with the magnets attached to the top of BADDY. Figure 27 shows the feeder and the base for the feeder with the screws attached. There are many holes on the top plate of BADDY. One large one, where the shuttlecocks will come through and feed the retainer, and some smaller ones placed around the big

hole. The three magnets were placed in their respective holes and fastened by a plastic plate that was screwed onto BADDY, as seen in Figure 28. In order to attach the feeder, simply place the feeder on top of BADDY with the magnets and screws aligned. Due to these magnets, it is easy to both remove and attach the feeder to BADDY.



(a) The base for the feeder with magnetic screws



(b) The feeder glued to the base

Figure 27: Plastic tube

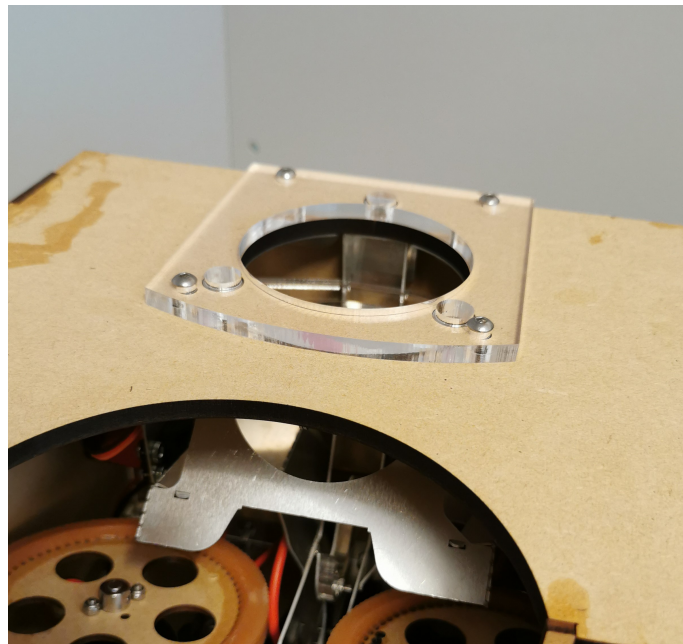


Figure 28: Magnets added to BADDY and fastened with a plastic plate on top of BADDY

The retainer and switch had to be adjusted after the mounting was finished. To find the correct positions for optimal shooting, BADDY had to be tested on a badminton court. This was postponed due to the restrictions given COVID-19, but was performed later.

Figure 29 shows the finished mounted BADDY placed on the tripod with the feeder attached and filled with shuttlecocks.

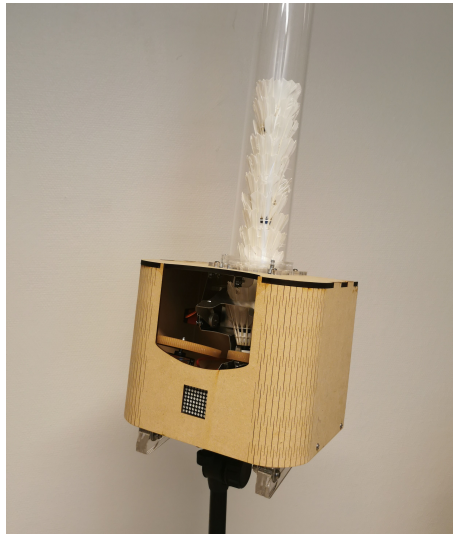


Figure 29: BADDY finished mounted with feeder and shuttlecocks

4.2 Testing BADDY

After BADDY was finished mounted, some tests had to be performed in order to adjust the switch and retainer and to check that the robot actually worked. Due to COVID-19, the gyms were closed for a period of time, which led to delays in the testing.

4.2.1 Functionality test

The first test was executed without a badminton net and court. BADDY was mounted on top of the tripod and turned in a direction with empty space ahead. When turning the robot on, a phone could connect to the BADDY WiFi network and communicate with BADDY using the phone application. Several different tests were performed, with one shuttle, multiple shuttles, and without any shuttles. With one shuttle in the feeder, BADDY worked as expected, and the shuttle was launched and followed the expected flight path. When multiple shuttles were placed in the feeder, some problems occurred. For every single shot, several shuttles were fired at the same time, when only one was supposed to launch. The first shuttle had the expected speed and flight path, while the retainer failed to hold back the next shuttles in the feeder. These next shuttles fell out, unintentionally, and landed in random places close to BADDY. Some of the shuttlecocks also ended up being "trapped" inside the fire room of BADDY, as shown in Figure 30.



Figure 30: Three shuttlecocks trapped inside the fire room

The robot was also tested without any shuttles in order to observe the functions and movements of the retainer and switch on a close hold, without the risk of getting injured by the launching shuttles. The conclusion of this test was that the retainer was unable to hold back the next shuttles which resulted in multiple shuttlecocks being launched at the same time, making it impossible to use BADDY for playing. Another observation that was made, was that the switch was not completely parallel to the frame, and the screws used on the switch were too long and touched the frame when launching a new shuttle. No damage was done to either the frame or the switch during this small test, but this could wear on both of these components and cause damage to them when using the robot over time.

By going through previous posts on the online BADDY forum, it was discovered that others have had the same problem with their BADDY. A solution to this was to change the embedded code for BADDY by adjusting the time for the retainer so that it managed to hold back the next shuttle while the first shuttle was launched. All parameters needed to control BADDY are defined and configured at the beginning of BADDY's code. Even though these parameters are named according to their functions and they are provided with small comments explaining what they do, it was difficult to understand what the different parameters were used for. The entire BADDY code is approximately 3800 lines long, and it was hard to find out where in the code the different parameters were used.

Before the second test was performed, the observation of the long screws and the unparallel switch had to be fixed. When removing the box of BADDY, it was observed that the switch was not parallel to the frame due to the position of the servomotor. The servomotor is quite heavy and only attached to the frame on one side, leaving the rest of the servomotor hanging in the air as seen in Figure 31a. This created an uneven load distribution resulting in an oblique position of the switch. Figure 31b shows a temporary solution to this problem by placing a piece of cardboard under the servomotor, allowing the switch to be aligned in parallel with the frame. A more permanent solution could be to attach a custom piece of e.g. plastic with glue under the servomotor to support it.

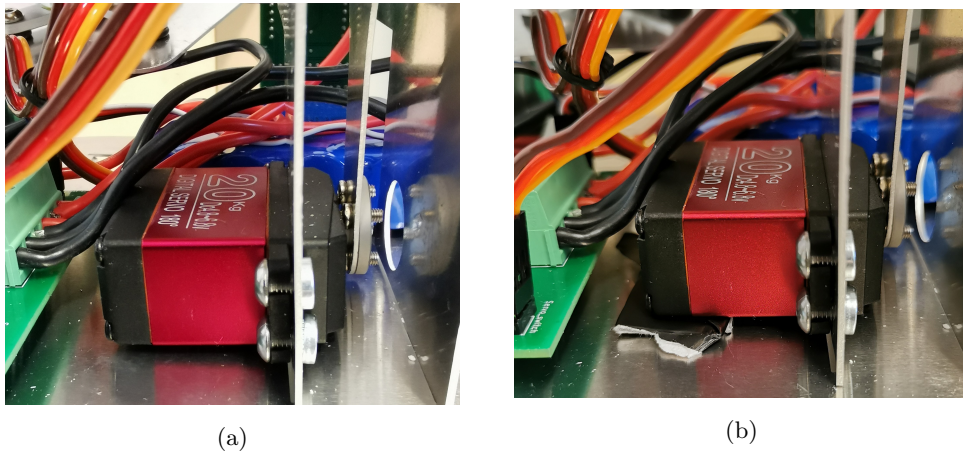


Figure 31: The servomotor hanging in the air (a), and a temporarily solution with a piece of cardboard under the motor (b)

To replace the screws on the switch, the whole switch had to be removed from the servomotor and be detached from BADDY. The screws, originally used to mount the switch, were 12 mm long. These were, as previously stated, too long, causing them to scratch the frame when BADDY launched shuttles. All the screws provided in the DIY kit came in one big plastic bag and were therefore mixed up. The long screws were replaced with 10 mm long screws, providing a more functional and safe movement of the switch to avoid damaging the switch itself, and the frame. The combination of changing the screws, and correcting the position of the switch by providing extra support to the servomotor, improved the functionality of BADDY to a great extent. However, the problem regarding multiple shuttles being launched for a single shot still remained.

4.2.2 Observing and fine-tuning of the switch and retainer

The second test was performed with the intention of solving the problem regarding multiple shuttles being launched at the same time. The hypothesis was that adjusting the timing of the retainer would help hold back the following shuttles in the feeder, and that would solve the problem. The setup of this test was different from the first one and it was desirable to observe the alignment and movement of the switch and retainer closely. This was possible by removing the box around the frame and providing better access and insight for the different components.

A closeup video of the function, and timing between the switch and retainer had previously been published in the public Facebook group, *BADDY community Group* [53]. This video showed how the two components should move related to each other to optimize the efficiency of the launching. By comparing this video and the movement of BADDY, it was observed that the retainer was moving too fast, not pausing at the top position. This prevented the retainer to hold back the next shuttle in the feeder and therefore allowing multiple shuttles to launch for each shot. Both the movement and timing of the components are defined in the code along with descriptive names for the parameters. The code was also provided with comments explaining the different parts of the code. Under the configuration parameters, the factory settings for the different shots, movement, and timers for the switch and retainer were provided, as well as other necessary parameters. The movement and timing of the switch and retainer are configured with eight different parameters with values as given in Listing 2.

```
int SWITCH_LONG_POSITION = 75;
int SWITCH_SHORT_POSITION = 140;
int SWITCH_WAIT_POSITION = 90;
int SWITCH_TIMER = 300;
int SWITCH_SPEED = 600;
int RETAINER_UP_POSITION = 115;
int RETAINER_DOWN_POSITION = 158;
int RETAINER_TIMER = 160;
```

Listing 2: Configuration parameters for the movement and timers for the switch and retainer

In order to make the retainer stop in the top position and hold back the next shuttle, some of these values had to be changed. The functions for each of these parameters are not explained in any other way than through their names. This caused some uncertainty regarding which parameter to change in order to make the retainer stay in the top position for a longer time. Given this uncertainty, it was concluded that different values had to be changed and tested to find one that improved the launches. They were tested with both increased and decreased values. Due to the purpose of this small test, which was to figure out what the different parameters did, the values were replaced with either very high or very low values to see the outcome of the change. Because of the misunderstanding explained in section 4.1, where the switch and retainer names were mixed up, only the switch parameters were changed.

After multiple changes to all the switch parameters, it was concluded that the parameter that made the most improvement was the parameter *SWITCH_SHORT_POSITION*. When this was changed from 140 to 300 the launching went a bit better. In retrospect, this was probably a coincidence. Changing the parameters would not help in the launching process, since the switch and retainer's servomotors were connected to the opposite pins on the electronic board, and their movements were not customized for the operation. This was not detected until later.

The embedded code and the technology behind BADDY are complex and consist of many parts that work together to optimize the performance of the robot. By just looking at a small part of the code, in this case, the parameters and values for the switch (that should have been the retainer), the solution found may not be the best or most optimal. The switch and retainer work together in order to launch the shuttles. Therefore, it might be possible to find a better solution than the one found, by changing multiple values simultaneously, e.g. one parameter for the switch and one for the retainer. After this test was performed, it was discovered that the retainer and switch were connected incorrectly. If they had been connected correctly, there would have been no need to change any of the parameters. This shows the complexity of BADDY, both in terms of the code and the wiring of the components. If something is wrong, it will affect the entire performance of BADDY and it can be hard to see if the problem is software or hardware related.

4.2.3 Performance test

Despite the problem of several shuttles being launched at the same time, still remained, this should not affect the performance of the test, as only one shuttle was launched at a time. The main purpose of this test was to observe and retain information about the different strokes. Especially considering the accuracy of the shots and where the shuttles landed. By having only one shuttle in the feeder at any given time, the shuttles that were launched would not be affected by the fact that BADDY was unable to withhold the next shuttles.

BADDY was attached to the tripod and placed at the intersection between the short service line

and the center line, about 2 m from the net, as can be seen in Figure 32a. The height of a badminton net is 1.55 m [80]. According to the report *Improvement of BADDY the shuttlecock launcher* written by Hugo Janton [63], you achieve the greatest reach of the shuttles when BADDY is placed on a tripod with an inclination between 30° and 40° . Janton states that it is necessary to “[...] dispose BADDY on its tripod to make it able to perform the widest range of shots” [63]. Therefore, BADDY was placed on top of the tripod so that the launch hole of BADDY was 1.35 m above the ground with an angle of 35° , shown in Figure 32b.

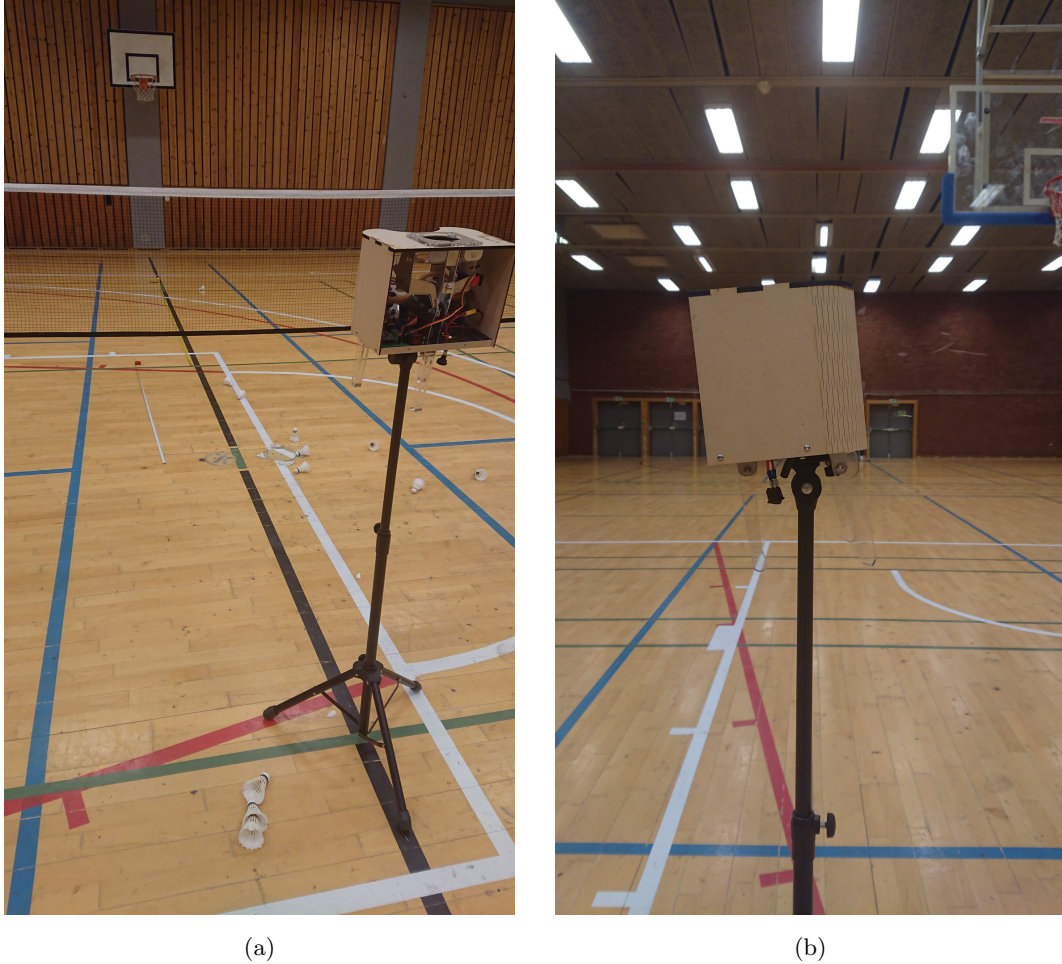


Figure 32: BADDY placed at the intersection between the short service line and the center line (a), and the incline of BADDY (b)

The type of shots that are launched depends on the height and inclination of BADDY. By increasing the height, BADDY will launch higher clears, while putting BADDY directly on the ground would result in more realistic drop shots. Figure 33a illustrates the recommended maximal inclination of BADDY. By increasing the inclination, the strokes will be more ascending, but BADDY can also launch powerful drives that go straight over the net. This can be done by decreasing the height and incline the robot forward as seen in Figure 33b. According to a badminton rule that was introduced in March 2018, a serve has to be shot from a maximum height of 1.15 m [8]. By being able to adjust the height according to which shots are desired, the shots will not mimic a serve.



(a)



(b)

Figure 33: Recommended maximal inclination (a), and BADDY with decreasing inclination for straight drives shown in a YouTube video [24] (b)

Figure 34 illustrates the positions of the shuttles for the different strokes, with BADDY placed at the same location as it was placed when performing this test. The expected output for this test was that the shuttles would land in approximately the same places as shown in the figure. However, the length of the clear shots deviated somewhat from what was expected, as they landed in front of the doubles back service line.

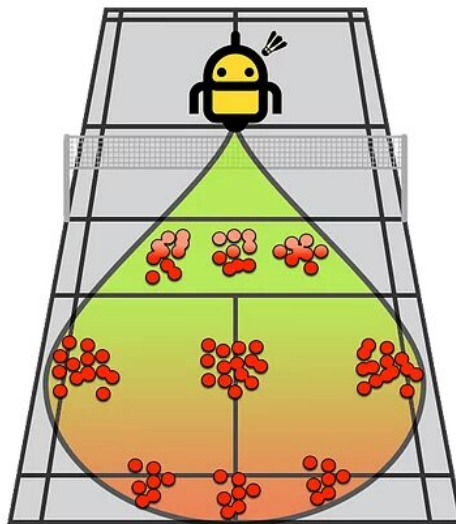


Figure 34: Illustration of the court coverage provided with BADDY's 9 different types of strokes.

With only one shuttle in the feeder, BADDY was able to launch the different shots with good precision each time. Meaning that for each shot of the same stroke, the shuttles landed within the same area with a radius of 0.5 m. The placement of the strokes on the court however, was not as expected. For each stroke, about 5 to 8 shuttles were launched in a row to check the precision of the placement.

After testing all the different strokes, it was concluded that the three different clear strokes were placed almost as expected on the court. The middle shot landed slightly to the right of the intersection between the center line and the long service line for doubles. These strokes were provided with both good height and a length of approximately 5.5 - 6 m from the net. The right clear shots were as expected a little shorter than the middle stroke and landed about 4.5 m from the net. This matches the illustration of the strokes in Figure 12. The left clear stroke was the one

that had the largest deviation from what was expected. In terms of length, the shots landed about 5 - 5.2 m from the net, which were roughly midways between the length of the center strokes and the strokes to the right. The left strokes landed closer to the center line compared to the right strokes with about 0.5 - 0.7 m. The length of the strokes was generally shorter than expected. There may be several reasons for this, but some sources of error may be because the shuttles that were used may have been old and dry. The quality of the air would also have an effect on the strokes, where both temperature and the humidity in the air were relevant factors, as well as the air pressure.

Even though BADDY was placed with the launching hole facing directly against the net, a common feature for all the shots, were that they ended up to the right of where they were expected. The same tendencies that were observed for the clears also applied to the drive shots. The middle drive shots landed about 3.5 m from the net, but compared to the middle clear shots, the drives landed about 0.5 - 1 m further to the right than the long shots. The placement of the right drives considering the distance from the center line was almost as expected. They all landed within 0.5 m from the right side line for singles. The length of the right drive shot was approximately 3 m from the net. Based on Figure 34, it was expected that all the drive shots would land with the same distance to the net. The left drive shots had about the same distance to the net as the right shots. But these, like the left clears, landed to the right from where they were expected. All the shuttles launched for the left drive stroke landed around 1 m from the center line, which gives a deviation of approximately 1.5 m from the left side line for singles.

The drop shots were the ones that had the largest deviation from what was expected. A common factor that was observed for the three different drop shots was that all shuttles landed to the right for the center line, even the left shots. The placement of the middle drop shots varied for each shuttle launched. The length of these shots varied from about 1.5 - 2.2 m from BADDY, which implies that some of the shuttles didn't even pass the net. They all landed slightly to the right of the center line. None of the shuttles launched for the right drop shot passed the net. The length of the launched shuttles was approximately 1 m from the net, and the height of these was approximately the height of the net. Based on the observation that most of the shuttles for all strokes landed to the right from where they were expected to land, it was expected that the right drop shots would also land further to the right than what Figure 34 shows. But these shots landed about 1 m to the right from the center line, which was close to the position of the middle drop shots. The left shots, as mentioned, also landed to the right for the center line. These landed about 0.2 m to the right of the center and 0.5 m in front of the short service line. Considering the placement of the left drop shots and the flight path for the shuttles, these shots were more like a center drop shot than a left shot.

After the test was completed, it was concluded that all strokes landed, with varying degrees, to the right of what was expected considering the illustration of the court coverage shown in Figure 34. This may be due to poor accuracy when mounting BADDY. Some parts may not have been placed correctly in relation to other parts so that the entire robot became skewed, causing all strokes to move to the right. Another reason for this could be that the placement and functionality of the wheels were not symmetrical. If one of the wheels was placed higher than the other, it could have given the shuttles a different angle when launching than if the wheels were placed perfectly in relation to each other. After the test, it was also observed that a screw on the underside of BADDY had loosened. This may be due to vibrations created by the motors while BADDY was launching shuttles. The loosened screw can be seen in Figure 35. A possible way to prevent this from happening again could be by using a lock washer or *Loctite* to secure the screw.

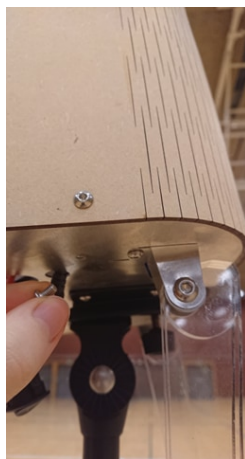


Figure 35: Due to the vibrations from the motors, a screw from the underside of BADDY loosened

The length and flight path of the shuttles for the clear shots were generally good, and these shots could be used for training. Even though they all moved a bit to the right, these strokes were, as stated earlier, the ones that were most precise and had the least deviation from what was expected. The drives were also good in terms of the flight path, the left and right drives were a little too short and the left drives were closer to the center of the court than the side line which is not optimal when training. It would not be possible to play or train with the drop shots. The left drop shot could replace the middle drop shot, but considering that all the shuttles for the right shots and most of the middle shots didn't pass the net, these would be useless in a training situation. An easy solution to this problem could have been to move BADDY one meter forward, but by doing so, the clear shots would be too long and end up on the outside of the back line of the court, which would not be optimal. Another adjustment that could have been performed, was to rotate BADDY towards the left to compensate for the shuttles being launched towards the right. This adjustment could make the landing position for some of the shots more accurate and as expected, but since the different shots varied to a great extent how much to the right of their expected position they went. This would only solve the problem for some of the shots and would only work as a temporary solution. However, it can be discussed to what extent this would be a problem. If great demands are made in terms of precision and perfect positioning of the shuttles, this would be a big problem and could compromise the effectiveness of the training. But if the purpose of the training is to use BADDY as a training assistant to practice simple strokes or provide beginners with good shuttles, it would not be a problem that the shuttles move sideways.

4.2.4 Troubleshooting

After the third test was completed, several adjustments had to be made to BADDY. A Skype meeting with the founder of BADDY, Mr. Greslebin, was arranged and conducted so that he could help make BADDY work correctly. With assistance from Mr. Greslebin, it was concluded that the servomotors for the retainer and switch had been exchanged and wired incorrectly to the PCB. This explained the misunderstandings regarding the names and why changing the parameters in the second test did not work as expected. To fix this, both the retainer and switch had to be detached from the servomotors before the wires connected to the PCB could be connected to the correct pins. The retainer and switch then had to be reattached to their respective motors and screwed on. BADDY was then turned on and tested both with and without shuttles. Some additional changes had to be made to the starting position for the retainer and switch, but BADDY was now

successfully able to launch only one shuttle at a time, with several others in the feeder. This entire process was conducted during the meeting with Mr. Greslebin, and while testing BADDY with shuttles, it emerged that the shuttles went to the right when launched. According to him, this happened because of an unalignment between the wheels. Figure 36 shows the two wheels mounted on each motor, where the left wheel is positioned lower than the right wheel when looking at the rods they are placed on. The left wheel was adjusted downwards after the previous test because it seemed that it was positioned higher than the right wheel. After further examination, it appeared that the entire board the motors are mounted on was unevenly mounted to the frame. Some of the parts had slipped slightly apart so that the motor board was not straight. By correcting this, BADDY will probably shot more accurately.

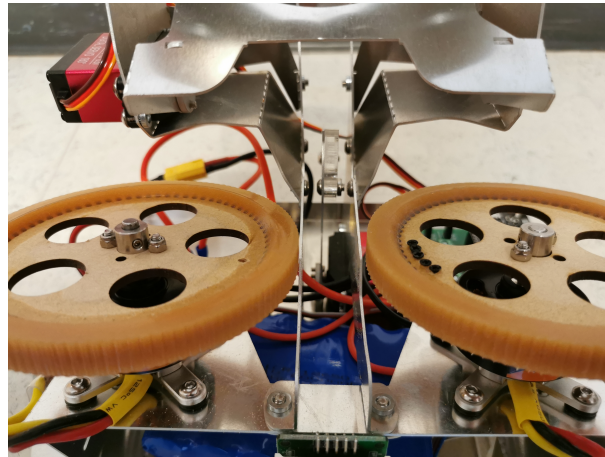


Figure 36: Wheels mounted unevenly

4.2.5 Final test

A final test was performed to ensure that everything was in order after the troubleshooting to make sure the corrections described in section 4.2.4, fixed the problems. This test was conducted with the purpose of checking whether BADDY could launch one shuttle at a time, even when multiple shuttles were placed in the feeder. During the troubleshooting, the motor board was straightened in an attempt to get BADDY to launch the shuttles more straight ahead, not drifting to the right. In addition to testing whether these adjustments improved the functioning of BADDY, the remote controller was also to be tested, both in regards to making BADDY launch a shot, but also to test it while playing.

Section 4.2.3 discusses some reasons that may have caused the shots to be shorter than expected. One of these was that the shuttles were old and dry. In order to check whether this was indeed one of the reasons, three different tubes of shuttles were used for this test. The first tube contained new shuttles with a weight of 62 grams for twelve shuttles, which gives a weight of 5.167 grams per shuttle. The second tube consisted of slightly older shuttles that have been used a bit, with a total weight of 60 grams for twelve shuttles, giving each shuttle a weight of 5 grams. The third, and final tube contained old and dry shuttles that have been used many times and were in worse conditions than the other shuttles.

BADDY was set up in the same way as for the third test, attached to the tripod, and placed about 2 m from the net, at the intersection between the short service line and the center line. The height of BADDY was set to 1.15 m. The inclination of BADDY was set to 40° since Janton's states

in his report that the maximal reach for the clear shots would be achieved when BADDY had an angle between 30° and 40° . Several centered clear shots were launched to test both the setup and whether BADDY could launch one shuttle at a time. The location for this test was a regular gym hall, with a decent height under the roof. However, with the given setup, about 80% of the launched shuttles was shot in the ceiling. Due to this, the angle of BADDY was reduced to 35° . The flight path of the shuttles was still quite high, but they did not touch the ceiling.

The first thing that was checked during this test, was whether the quality of the shuttles had any effect on the quality of the shots. In order to compare the different shuttles in the best way, multiple shuttles were launched with the same type of shot, a center clear. The first tube with the newest shuttles was tested first. 10 shuttles were launched and the placement of the shuttle can be seen in Figure 37. The yellow measuring tape is placed 5 m from the net, meaning that most of the shuttles landed around 5.2 m to 5.6 m from the net, with the exception of two shuttles that landed approximately 4.8 m from the net. As seen in the figure, all shuttles landed within an area with a diameter of 1 m.



Figure 37: Placement of the shuttles from the first tube with new shuttles

The same procedure was performed with the second tube. 10 shuttles were launched and the placement of the 10 shuttles can be seen in Figure 38. As seen in the figure, these shuttles were more spread out and they all landed within an area with a diameter of 1.5 m. Some of the shuttles went shorter than the others, but generally, the length of the shuttles was almost the same as for the first tube. However, the shots launched with the first tube were more accurate and centered than the shots with the second tube.



Figure 38: Placement of the shuttles from the second tube with slightly used shuttles

The final and third tube, as mentioned, consisted of old and dry shuttles. The placement of these can be seen in Figure 39. The old shuttles landed, similar to the shuttles in the second tube, more spread out than the newest shuttles. But the old ones landed closer to the net than the shuttles in both tube one and two. This was expected as they were in a worse condition than the other shuttles and this did in fact affect the quality of the shot. Even though most of these shuttles went shorter than for the previously tested shuttles, they all landed relatively close to each other, within 1 m, with the exception of two shuttles that landed at around 4 m from the net. Based on the results from these three tests, it was concluded that the quality of the shuttlecock did have an effect on the shots. The shuttles that were old and dry went shorter than the newer shuttles. Even though the length of the different shuttles varied, the accuracy of the shots was relatively good. For all three tubes, BADDY launched the 10 shuttles within a small area. This implies that even though shuttles with poor quality are used, BADDY will launch accurate shuttles for the different shots. It was concluded that BADDY is quite precise in terms of the placement on the court regardless of the quality of the shuttles. The placement of the shuttles, for all the tubes, may not be an accurate representation of how precise BADDY's launching is. The shuttles are, as mentioned in section 2.1, made out of a leather-drawn cork tip with feathers attached to it. This could result in the shuttles bouncing around after hitting the ground and increasing the distance between them.



Figure 39: Placement of the shuttles from the third tube with the oldest shuttles

After testing whether BADDY could launch one shuttle at a time, the placement on the court for the different shots was tested. To get the most reliable result, the first tube with the newest shuttles was used for this. 12 shuttlecocks were placed in the feeder. The drop shots were tested first, where 4 shuttles were launched for the right drop shot, 4 in the middle, and 4 to the left. Generally, all the 12 shuttles landed to the right of what was expected, as shown Figure 40, similar to the third test. The reason for this may have been that the motor board was not straightened enough, or that BADDY was mounted skewed. When BADDY was placed on top of the tripod, it was observed that BADDY was slightly leaning to the left, as can be seen in Figure 41. This could be the reason why the shots went to the right. The PCB is placed on the left side of BADDY, and this could create an uneven weight distribution causing BADDY to lean to the left. Regardless of this, the different types of drives were quite precise. As seen in Figure 40, they all landed about 1.5 m from the net and within an area with a diameter of 0.5 m. The placement of the shots is indicated with the white circles in the figure. One of the shuttles for the middle shots, indicated in the red circle, deviated from the others as it didn't even pass the net. This reinforced the observations previously made in the test, in regards to BADDY being precise for the different shots.



Figure 40: Placement of the left, right and middle drop shots circled in

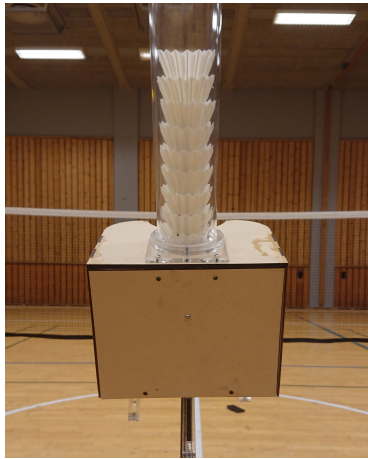


Figure 41: BADDY placed on the tripod with an uneven weight distribution, leaning tho the left

This was repeated for both the drives and the clears, where 4 shuttles were launched for the right, center, and left shots. For most of the different shots, one shuttle deviated a bit from the three others, but they were all placed close to each other. The drives and clears also landed to the right of what was expected, but as stated, this did not affect the precision of the shots. For both the drives and clears, the left and right shots landed closer to the net than the center shots.

Finally, the three different shots were launched for the left, center, and right side, to see the placement of the shots in relation to each other. The left shots can be observed in Figure 42, where the drop shots landed just in front of the service line and the clears landed about 1.5 m in front of the long service line for doubles. As seen in the figure, 4 shuttles were launched for each stroke. The placement of the shuttles for the different strokes was almost as expected based on Figure 34, which illustrates the court coverage for the different strokes, with the exception of the clears. These went much shorter than expected, as it was expected for them to land between the two back lines instead of 1.5 m in front of the shortest back line. How much this would influence the training depends on the player. If the players are adults or senior players, the length of the clear shots shown in Figure 42, would not be sufficient and the outcome of the practice would be lower than desired. However, if the players are younger or inexperienced, these short clears would not be a problem as the players would be challenged enough. Some changes could have been made in order for BADDY to launch longer clears, e.g. by moving BADDY forward or changing the height or incline. By moving BADDY forward, the drop shots would have been too long, something that would also have a negative effect on the training. For this test, BADDY was set up with an angle of 35° and a height of 1.15 m which, according to Janton [63], would be optimal for long shots. In order to achieve longer clear shots, the only factor left to change was the speed of the motors. It has been recommended by the developers not to change any of the parameters in the code, including the speed. Therefore, no changes were made to these, and the shots could not be improved in order to satisfy the senior players. BADDY might not be the perfect robot for them if they want to be challenged at all levels, but can be used for practicing one shot at a time.



Figure 42: Placement of the left drops (at the bottom), drives (in the middle) and clears (at the top)

The remote controller was also tested during this test. Before this test was conducted, the remote controller had been soldered and completed. The HC-08 module had also been soldered to BADDY's PCB, and part of this test was to check and make sure that the controller worked properly. The remote controller was placed in the hand of a player, as seen in Figure 43, so that the player was able to launch a preprogrammed shot when pressing the button. The remote controller is small and lightweight, and due to the elastic band, it can be firmly attached to the non-racket hand of the player. The non-racket arm has a big impact on the game. It is mainly used for balance, but also to increase the power of the hit, and aiming where to hit [32, 36, 98]. It would be possible to improve both the player's experience and training by replacing a smartphone with the remote controller, as the player can use the controller without affecting the game or technique. When using a smartphone, the player would have to hold on to the phone during the entire session and look at it to launch shots. It is natural to use the non-racket arm for aiming at the ball, with an open hand, something that would be impossible when holding on to a phone. The use of a phone could therefore be a distraction for the player. By using the controller that is attached to the hand, the player could move in a more natural way and even open the hand without losing it. This is discussed in the prestudy report [67] in section 5.2. A closeup of the controller attached to a hand can be seen in Figure 44.

The controller could also be used by the trainer, something that could improve the quality of the feedback as the trainer would be able to observe the player more closely than if he had to stand on the other side of the net, launching shuttles himself, and only observing the player from a distance.



Figure 43: A player using the remote controller in the left hand



Figure 44: The hand doesn't need to be closed for the controller to stay safe in the hand

After the tests, the weather data for the time and days the third and fourth tests were carried out, retrieved from an online weather channel. The third test was conducted on March 10 and the temperature, humidity, and air pressure for the relevant times of the test are given in Figure 45. The temperature was -1°C , the humidity was 55% and the air pressure was around 1007 mbar [92]. The fourth test was performed about two months later on May 10 and the weather conditions for this day can be seen in Figure 46. The temperature was 8°C , humidity around 95%, and air pressure of 1013 mbar [93]. These data were measured at a weather station in Trondheim and show the conditions outdoors. Both tests were conducted in an indoor hall with a higher temperature than shown in the figures, but it can be assumed that the temperature inside was lower on March 10 when it was colder outside than on May 10.

When the third test was performed, the humidity was 55%, while it was around 95% during the fourth test. This implies that the water vapor in the air was higher for the fourth test than for the third. With a lower temperature and humidity outside, the air indoors will be affected and result in less optimal conditions for the use of badminton shuttles. The shuttles are made of feathers and the flight path and length for these could be reduced with a lower temperature, humidity, and air pressure. While for a higher temperature, as it was for the fourth test, the shuttles would have

a more optimal and expected behavior. The shuttles would most likely have a longer and higher flight path and the overall playing conditions would be improved.




Time	Temperature	Humidity	Air pressure
11:20	 -1 °C	55%	1008 mbar
11:50	 -1 °C	55%	1007 mbar
12:20	 -1 °C	55%	1007 mbar

Figure 45: The weather conditions in Trondheim for the third test, on March 10 [92]




Time	Temperature	Humidity	Air pressure
11:20	 8 °C	100%	1013 mbar
11:50	 8 °C	93%	1013 mbar
12:20	 8 °C	93%	1013 mbar

Figure 46: The weather conditions in Trondheim for the fourth test, on May 13 [93]

After BADDY was finished mounted and tested, and the prototype of the remote controller was fully developed, BADDY was borrowed by a Badminton coach in *Trondheim Badmintonklubb*. The robot was used during a training session for twelve year old players. The training session lasted around one hour, and BADDY was used throughout the entire session. The feedback given from both the coach and the players was positive and valuable in terms of how it is to train with a BADDY robot. It was also observed that many were interested in BADDY and wanted to get a closer look at it. This challenged the security aspects of the robot, as people were right in front of the launching hole and wanted to look inside BADDY. This will be further discussed in section 7.1.

4.3 BADDY code review

BADDY's source code can be seen on BADDY's GitHub page [31], and the project structure can be seen in Figure 47. The code used ⁵, was the third, and latest version of BADDY's code, at the time. BADDY V1 was the first version launched. This version was embedded with a Bluetooth connection and used an HC-03 module to connect with other Bluetooth devices [74]. The next

⁵The original BADDY code used is included in the digital appendix (attached files) as a reference for the reader. For further development, get updated code from the BADDY GitHub [29]

version, V2, used WiFi to communicate with other devices. This opened up for the possibility of connect two BADDYs (BADDY and BUDDY) together. For the latest version, V3, a hotspot mode was added, and *PlatformIO* was introduced for the firmware part of BADDY. The hotspot is used instead of an access point and a WiFi client (station), see section 2.2. The introduction of *PlatformIO* required the usage of *Visual Studio Code* instead of *Arduino IDE*. *Visual Studio Code* is a more full-fledged editor with more features included and it is more suitable for more extensive code.

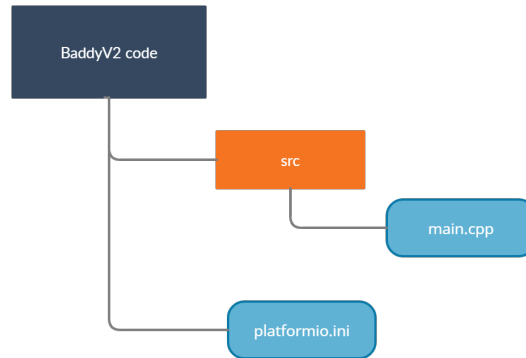


Figure 47: Folder structure for the BADDY files

The latest version of the code consists of two files, a C++ file called `main.cpp` and a *PlatformIO* file called `platformio.ini`. The main file is 3824 lines long and the entire functionality for BADDY is declared, configured, and used in this file. In the video “BADDY V2 Tutorial - Install BADDY Firmware” at time 7:30, it is explained why the entire code is placed in one large file. According to the video, this method was used to make a simple setup for the project’s codebase [28]. The method chosen to represent and publish the code base for the project may be easy to implement, but it is neither elegant nor easy to work with. It is difficult to get an overview of the different functions and methods, as well as to understand what the different parts of the code actually do. In the tutorial video, it is said that future releases will include a `library.h` file to make the code setup more elegant and readable [28].

As stated in section 2.8, BADDY’s code is made of Arduino code, which is simplified C++ code. Arduino code has a standard code structure and can be divided into four parts; initialization of variables, declarations of functions, setup-function, and loop-function. Figure 48 illustrates these parts and shows how BADDY launches shuttlecocks in the infinite loop-function. The variable `flag_new_sequence` will change to true if BADDY receives signals from the user to launch shuttlecocks.

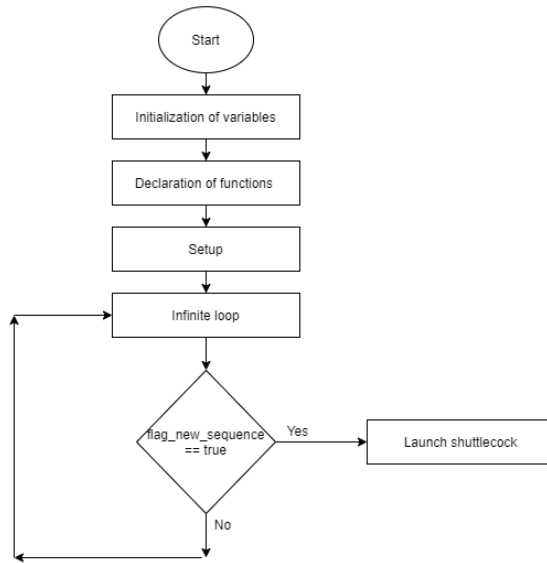


Figure 48: Simple illustration of BADDY's code flow

Big programs that are written in C++ code, often split the code into multiple classes and files to make the code more understandable and readable. It is also common to implement header files that contain declarations of the variables and functions. The use of a header file reduces dependencies and can reduce the compilation time. Related declarations will only appear in one place, something that makes it easier to change the code both in terms of time consumption and error pruning [58]. Arduino projects usually only consist of one file as they are often small projects with little code. As mentioned, BADDY's entire code is placed in one large file to simplify the process of setting up the firmware. Since BADDY has been developed in several versions, over time, with additional features, the code has grown longer and longer. For future development, it would be beneficial to split the code into several files and add header files.

The first lines of BADDY's main.cpp file are used to include the libraries needed and defining the used board pins, followed by all the variables and functions needed. By moving all these declarations of variables and functions to a header file, the code would be more structured and easier to understand. After these declarations, the two mandatory main functions of Arduino code follow; *void setup()* and *void loop()*. The setup-function defines BADDY's initial state and is run only once when BADDY is turned on. The loop-function is run after the setup-function is finished, and run through the code until BADDY is turned off. This function contains all the main logic for the program and calls for the necessary methods. BADDY's code consists of many parts and methods, and a detailed flowchart representing the code can be seen in Figure 49. The figure also includes the steps where AP or hotspot mode and BADDY/BUDDY mode are chosen.

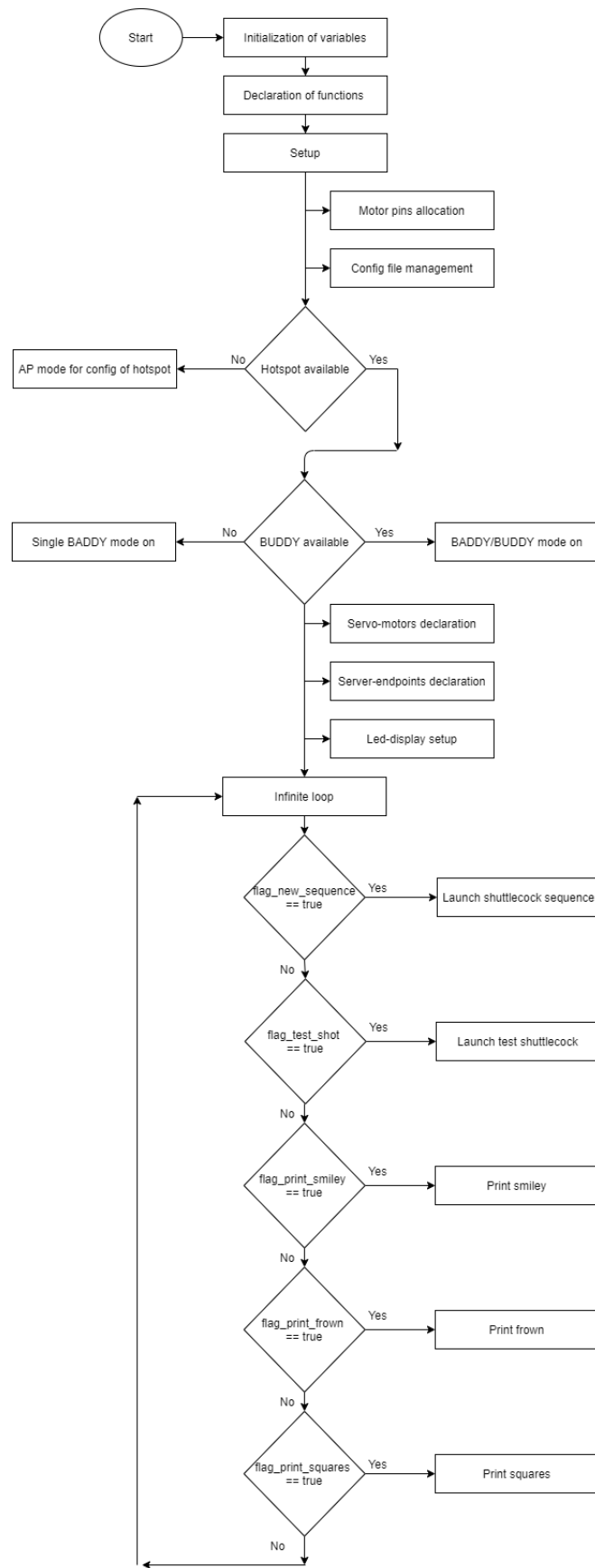


Figure 49: Flowchart representing BADDY's code

A total of 36 functions are declared within the "main.cpp" file. One of these functions, *ipToString()*, is never used and is most likely forgotten to be removed after a version update. Four functions in addition to the *ipToString()*, are illustrated in Figure 50, and can be considered as help functions for formatting strings.

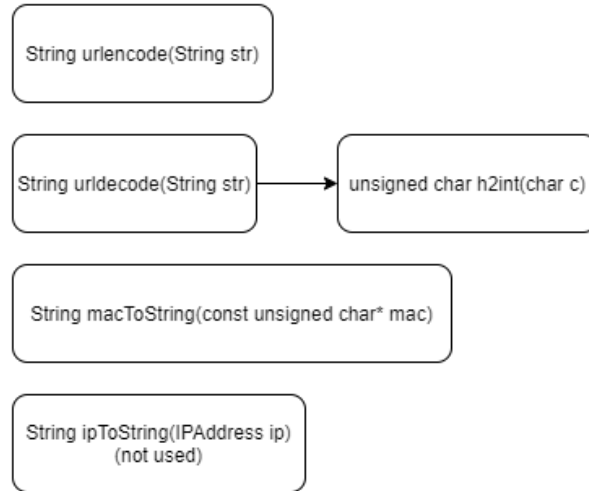


Figure 50: Five string formatting functions declared in the BADDY code. *urldecode()* uses *h2int()*

Figure 51 and 52 give a detailed overview of the different functions used in the setup- and loop-functions, in the same order as they are called. The arrows point from a function using other functions (left), to the function used (right). BADDY's code consists of many functions and most of these, use other functions. The figures illustrate how the functions are used and that many functions are called several times in different functions. The functions used only once, have a white background color, while the functions used several times have been indicated with their own background color. For example, the function *dump_play_mode()* has been indicated with a light red background color in Figure 51. All the boxes with the light red color show where this function has been used. In addition to the functions shown in the illustration, the BADDY code uses several functions from the libraries included, such as Servo, LedControl, and WiFi-functions.

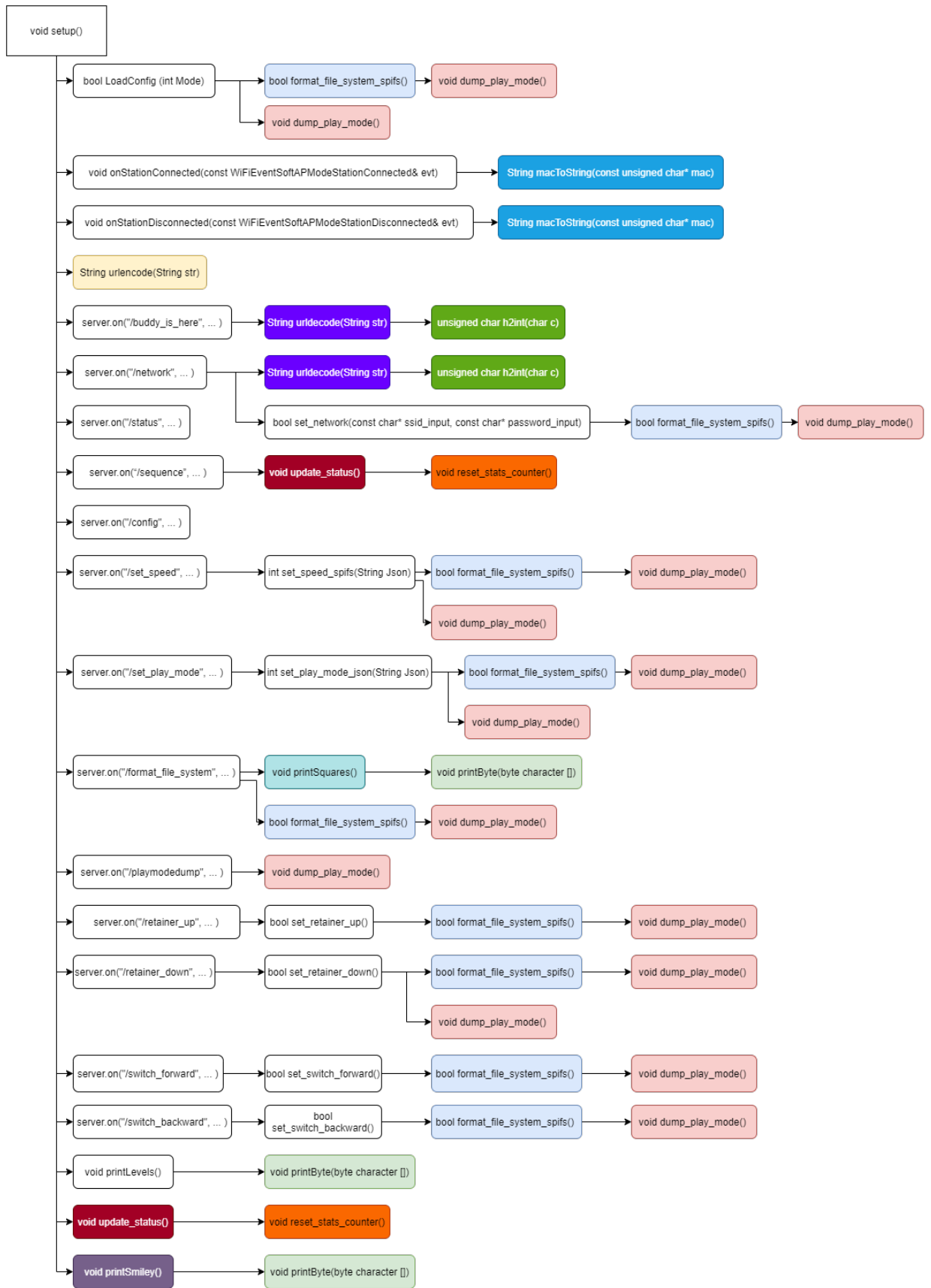


Figure 51: Functions used in the setup-function



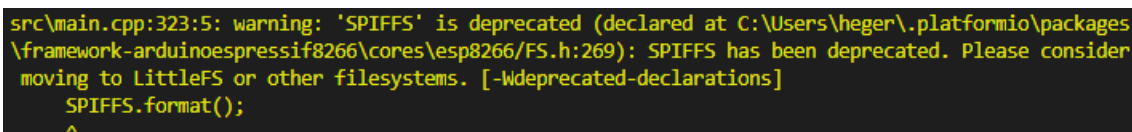
Figure 52: Functions used in the loop-function

4.3.1 Code improvements

Many of the parameters are defined as variables. A possible improvement of the code could be to define these as *#define*. Constants that are defined like this, do not take up any memory space on the chip and the compiler will replace any references to these constants with the value that is

defined when the code is compiled. However, this can cause some problems. If a constant name that is *#define* is included in another constant or variable name, the value for this will be replaced by the *#define* constant's value [13]. Constants can be defined by the use of the keyword *const* instead of *#define*. The use of this keyword will change the behavior of the variable so that it is *read-only*, meaning that its value is unchangeable. Such constants obey the rules of variable scoping, making this method superior to the use of *#define* [12]. At the same time, BADDY's code has many initialization values. These are values that don't need variable scoping, since they can have a constant as variable that will remain unchanged. It could therefore be preferable to define the values as *#define* and store them all in a separate file. Instead of defining each variable in the actual code file, one can rather include the file where all the variables are defined.

All libraries used in the code, are included at the beginning of the main.cpp file. One of the file systems needed to make BADDY work is the ESP8266 SPIFFS file system. The library FS.h is included in line 67 in the main.cpp file with the command *#include "FS.h"* to declare SPIFFS. When building the project in *Visual Studio Code*, multiple warnings are displayed in the terminal, one of which is shown in Figure 53, and states that "SPIFFS' is deprecated".



```
src/main.cpp:323:5: warning: 'SPIFFS' is deprecated (declared at C:\Users\heger\.platformio\packages
\framework-arduinoespressif8266\cores\esp8266/FS.h:269): SPIFFS has been deprecated. Please consider
moving to LittleFS or other filesystems. [-Wdeprecated-declarations]
  SPIFFS.format();
  ^
```

Figure 53: SPIFFS warning

This warning did not prevent the code from being uploaded to the BADDY PCB, nor did it prevent BADDY from working, but it might not be optimal to use this library. The warning states as seen that 'SPIFFS' is deprecated, this is because the file system is no longer supported by the developer as a new file system, "LittleFS", is under development. LittleFS focuses more on directory support and higher performance than SPIFFS, which is a file system optimal for applications with limited space and RAM [59]. In addition to adding header files in future releases, the SPIFFS file system should be replaced with LittleFS as LittleFS is under development, supports real directories, and is faster than SPIFFS. Another reason for moving to LittleFS, is that SPIFFS currently is deprecated, and may not be a part of future releases of the core [59].

When reviewing BADDY's code, many code smells are observed. Code smells are "absolute violations of the fundamentals of developing software that decrease the quality of code" [42]. A code can have multiple code smells and still provide an output, these are not bugs or errors that will cause the code to crash. But by having code smells, the program would according to Codegrip run slower, the risk of failure and errors would increase and the program would be more vulnerable to possible bugs in the future [42]. There are many different types of code smells, and which smells that occur vary from program to program, based on the developer and organization behind the development. Some common code smells are duplication of code and using unnecessarily complicated design patterns. Long methods that are hard to understand and responsible for more than one behavior, as well as too long or short identifiers and too many parameters are also common smells. The BADDY code is, as previously mentioned, long, complicated, and confusing. Code smells like duplicated code, many parameters, long functions, and unused code are repeated in the code.

The *server.on("/status", ...)*-function in BADDY's code is given in Listing 3. From line 7 to 23 in the listing, BADDY's battery level is read and processed. A possible improvement for this part

of the code could be to create a new function called e.g. *readBatteryLevel()* to make it easier to understand the code.

```
1 server.on("/status", HTTP_ANY, [](AsyncWebServerRequest *request){
2
3     DynamicJsonBuffer jsonBuffer;
4     JsonObject& status = jsonBuffer.createObject();
5     // check battery level
6
7     int battery_level = analogRead(ANALOG_READ_BATTERY);
8
9     // Debug battery level management
10    Serial.print("Battery level read on A0: ");
11    Serial.println(battery_level);
12    // End debug
13
14    if (battery_level > 195)
15    {
16        battery_level = 195;
17    }
18    if (battery_level < 135)
19    {
20        battery_level = 135;
21    }
22
23    battery_level = map(battery_level, 135, 195, 0, 100);
24
25    status["Running"] = running_status;
26    status["Battery"] = battery_level;
27    status["Firmware_version"] = baddy_firmware_version;
28    status["Connect_mode"] = connect_mode;
29    JSONArray& Stats = status.createNestedArray("Stats");
30    Stats.add(COUNTER_DROP_LEFT);
31    Stats.add(COUNTER_DROP_CENTER);
32    Stats.add(COUNTER_DROP_RIGHT);
33    Stats.add(COUNTER_DRIVE_LEFT);
34    Stats.add(COUNTER_DRIVE_CENTER);
35    Stats.add(COUNTER_DRIVE_RIGHT);
36    Stats.add(COUNTER_CLEAR_LEFT);
37    Stats.add(COUNTER_CLEAR_CENTER);
38    Stats.add(COUNTER_CLEAR_RIGHT);
39
40    //Serial.println("Json status object dump");
41    //status.prettyPrintTo(Serial);
42    String json_status = ""; // clean before print
43    status.printTo(json_status);
44    //Serial.println(json_status);
45
46    request->send(200, "text/plain", json_status);
47 }
```

Listing 3: *server.on()* - function with logic for reading BADDY's battery level

By reviewing the code for the different versions, it appears that many parts are very similar. When the code was upgraded, the changes were made directly to the code of the previous versions. Some parts were removed, changed and some new functions were added, such as WiFi replacing Bluetooth, and the code necessary to make WiFi work was added. When reusing and further develop old code, it is easy to overlook and miss different parts of the code that are no longer in use. Several examples of this can be found in BADDY's code, such as the function *ipToString()* which is defined in lines 1104 to 1109. This function is never used and should be removed from

the code. The code also contains several lines that are commented out. Many of these lines are *Serial.println()*, these are probably used for debugging, and are not an essential part of the program. Listing 4 shows some lines of code and line numbers for code that is unused and should be removed.

```
Line 1152 //encodedString+=code2;
Line 1183 //yield();
Line 1225 //buddy.sendRequest("POST", "/status");
Line 1226 //buddy.addHeader("content-type", "text/plain");
Line 2515 //update_status();
```

Listing 4: Code lines with associated line numbers that are never used in BADDY's code

There are some guidelines and standards that should be followed to achieve good code practice. Some of these standards include writing readable and efficient code. This makes it easy to understand the code while the run time and required space for the program are optimized. Commenting and documenting the code is also a big part of a good code practice, as well as using descriptive names on variables and functions. This will not only help the developer during the development process, but also make the code more understandable to others. The BADDY code does have descriptive names and many comments describing the functionality of the code. However, these comments are the only form of documentation for the code that exists.

Several naming conventions exist to adhere to a set of "rules" when naming functions and variables throughout the code. The functions and variables in BADDY's code do have descriptive names, but they are not written in a consistent way and the way they are defined varies throughout the code. Naming conventions used are flat case, camelCase, PascalCase, snake_case, and SCREAMING_SNAKE_CASE. The use of all these different conventions may be a result of several people writing the code and a combination of many coding cultures. Another reason for this might be if the code has been rewritten for the newer version updates, such that parts of the code are older than other parts of the code.

Listing 5 shows five boolean variables. These are flag variables that indicate when a status changes or when a signal is received from the BADDY application. All these flag variables are used in the loop-function, as shown in Listing 6, where an action is executed if either of these variables changes the value to *true*. From line 32 in the listing, if *flag_print_squares* is *true*, the *printSquare()*-function is called, making the led screen on BADDY print a square, before the variable is set back to 0 in line 35. The flag variables are used this way throughout the entire code, except for in line 3668 in BADDY's code. Here, the *printSquares()*-function is called directly in the setup-function instead of changing the flag first. This might have been overlooked by mistake during the upgrade of the code and should be changed because it makes the code inconsistent. As this example shows, the code is also inconsistent in regards to using boolean variables or the numbers 0 and 1. Since the flag variables are defined as *bool* variables, they should be set to *true* or *false*. Instead, they are set to the numbers 0 and 1 in many places. This is accepted by the compiler because the boolean values in C++ are actual stored as integers [62], but it should have been defined in one way and used consistently throughout the code.

```
bool flag_new_sequence
bool flag_test_shot
bool flag_print_smiley
bool flag_print_frown
bool flag_print_squares
```

Listing 5: Flag variables defined as booleans

```

1 void loop() {
2     //ArduinoOTA.handle();
3     MDNS.update();
4
5     if (flag_new_sequence)
6     {
7         sequencefetch(JsonSequence);
8         ReadSequence();
9     }
10
11    if (flag_test_shot)
12    {
13        delay(1500); // wait till motors get their nominal speed
14        Serial.println("Shooting shuttlecock...");
15        servo_fire();
16        flag_test_shot = 0;
17        MotorRight.writeMicroseconds(STOP);
18        MotorLeft.writeMicroseconds(STOP);
19        Serial.println("Stop motors!");
20    }
21
22    if (flag_print_smiley)
23    {
24        printSmiley();
25        flag_print_smiley = 0;
26    }
27    if (flag_print_frown)
28    {
29        printFrown();
30        flag_print_frown = 0;
31    }
32    if (flag_print_squares)
33    {
34        printSquares();
35        flag_print_squares = 0;
36    }
37 }

```

Listing 6: The loop-function

As stated earlier, good code practice is about, among other things, avoiding duplicating code. If code is duplicated, it is much more beneficial to create a separate function or method for this part and call for this method where needed. Creating one separate function for code that is duplicated, will simplify the code structure and reduce the length of the code. It would also be easier and cheaper to support and maintain the code. If the code needs to be changed, it would only be necessary to change the code in one place, instead of finding all the different places the code is used. This would also be beneficial in terms of reducing the risk for errors and bugs. Listing 7 shows a code snippet from BADDY's code that is reused seven times throughout the code. This could have been placed in a function and instead of writing the same code seven times, it would only be necessary to define the functionality in one function and call on the function in the necessary places.

```

1 if (!file){
2     Serial.println("No BADDY config file exists");
3     file.close();
4     format_file_system_spifs();
5     return false;
6
7 }

```

Listing 7: This code appears seven times in the BADDY code

The BADDY code was clearly under construction and continuously upgraded. An example that illustrates this can be seen in line 1877 in the code and its following lines that are given in Listing 8. This part of the code is commented out and clearly not in use. BADDY is continuously being improved and upgraded with new features and better functionality. As long as BADDY is being developed, the code will also be upgraded and developed.

```

1 // Here, we manage BADDY BUDDY time considerations between strokes - section kept
  out as i needed more tangible finetuning
2 /*
3  if ((next_type == 11)|| (next_type ==12)|| (next_type ==13)){
4      return 0; //we do nothing
5  }
6
7  if ((next_type == 14)|| (next_type ==16)){
8      return 0; //we do nothing
9  }
10
11  if ((next_type == 17)|| (next_type ==19)){
12      return 0;
13  }
14  */
15 // End of BADDY BUDDY timer considerations

```

Listing 8: Unused code in BADDY

The challenges of open-source projects are often the same as the examples discussed throughout this section. With many people contributing, different coding cultures are represented, and it would be hard to establish some common standards and convey these to everyone contributing. Such communities are often built on trust and voluntary work, which makes the projects more vulnerable to errors and slow progress [61]. All the commits that had been sent to the project on BADDY LAB's GitHub page, were sent by the BADDY team. This could mean that the code for BADDY has been written by a few people, all within the BADDY team. But two pull requests have been sent to the project from two outside individuals. However, both of these requests were, at the time of this report, more than one year old, and the latest commit was dated November 20, 2020 [21]. This commit was, at the time, more than 6 months old, something that indicated that the progress and development of the code were slow. The rare commits along with the poor response on the BADDY forum made it hard to tell whether anyone was working on the code at that moment.

5 Modifications to BADDY

BADDY is controlled by the microcontroller ESP8266 WiFi module (2.7). To connect BADDY to the remote controller using Bluetooth communication, it was necessary to extend the microcontroller with a Bluetooth module. This modification consists of hardware and software modifications and was done in several steps. As mentioned in section 1.2, the final goal for this project was to launch a shuttlecock when pressing the button on the remote controller.

To give BADDY the possibility to communicate with Bluetooth, a module had to be integrated with BADDY's PCB. The last result from the prestudy consisted of an Arduino Uno board connected to an HC-08 Bluetooth module and a LED lamp. It could receive a signal from another HC-08 module and turn the LED on. This was called the master setup due to the use of the master HC-08 module in this end. The master setup can be seen in Figure 54.

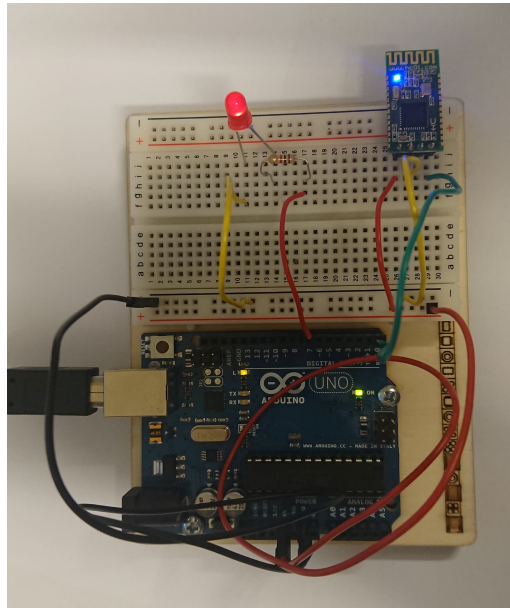


Figure 54: Wired master circuit from the prestudy

The master setup had to be changed from using an Arduino Uno board to use the ESP8266 board in BADDY. To make the code modification as simple as possible and check which pins on the ESP that were usable, the Bluetooth module was switched with a button in the first modification step. The final result did not include the LED, but it was used for some testing and to check if the modifications done to the wiring were performed correctly. The slave and the master code from the prestudy had to be combined and modified to make this work.

5.1 Wiring a push button to BADDY's PCB

The printed circuit board that comes with BADDY is a custom-made circuit board created for the purpose of wiring BADDY's electronics in the best possible way. It is embedded with an ESP8266 WiFi module. Given that the board is custom made, it implies that most of the pins are already assigned to different outputs and wired directly to the components inside BADDY. The source code provided on Baddy Lab's GitHub page [29] includes the ESP Pin configurations and what the different pins are used for, shown in Listing 9. The code implies that 9 pins, in addition to GND

and Vin, are used to control the different components in BADDY, leaving only a few available for further work.

```

1 // ESP Pin configurations
2 #define LED_DIN D2
3 #define LED_CS D1
4 #define LED_CLK D3
5 #define MOTOR_RIGHT_PIN D5
6 #define MOTOR_LEFT_PIN D7
7 #define SHUTTLE_SWITCH_PIN D6
8 #define SHUTTLE_RETAINER_PIN D8
9 //In case you want to use the EXT pin header for your Retainer servo motor, comment
   the line above and uncomment the line just below
10 // #define SHUTTLE_RETAINER_PIN D0
11 #define ANALOG_READ_BATTERY A0

```

Listing 9: The source code shows that the following pins are already in use

By the inspection of BADDY’s original source code, along with the study of the Gerber files ⁶ of the PCB, unused pins were detected. Figure 55 shows the Pin Out Diagram for an ESP8266 board and was used to gain information about the different pins on the board. Based on the information retrieved from this diagram, the pins available for use could be separated from the ones that were already assigned to other components. The red squares on the figure indicate the pins that were unavailable, while the green square indicates the one pin that was available and possible to use. None of the regular GPIO pins was free to use without changing BADDY’s original wiring. The only free pin that was possible to use was D4 (GPIO2). It has a pull-up resistor and can also be used as TX-pin [51]. This pin can be used with the push button, but the pull-up resistor will invert the logic of the button. A push button connects two points in a circuit when they are pressed down. The internal pull-up makes the pin connected to a 5 V and it will read as HIGH even if the button is not pressed. The button needs no reference to voltage and only has to be wired to pin 2 and ground without any resistor. When the button is pressed, pin 2 will be connected to ground and reads as LOW. This is the opposite of regular push button behavior [16]. A picture and circuit diagram of the wiring can be seen in Figure 56. A LED was wired temporarily to the D1 pin just to get a feedback when the button was pressed. The D1 pin was suitable for the LED even without a resistor.

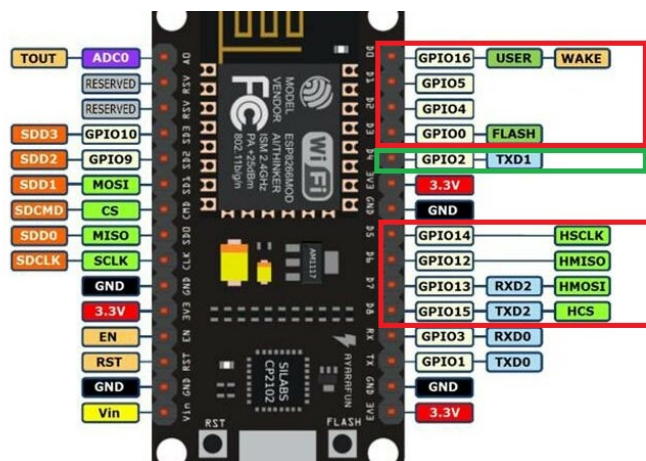


Figure 55: Indication of used and available pins on BADDY’s ESP board. Original illustration made by Circuitspecialists [40]

⁶The Gerber files used are included in the digital appendix (attached files) as a reference for the reader. For further development, get updated files from the BADDY community [19]

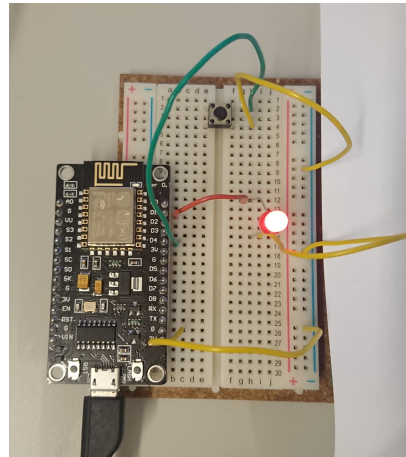
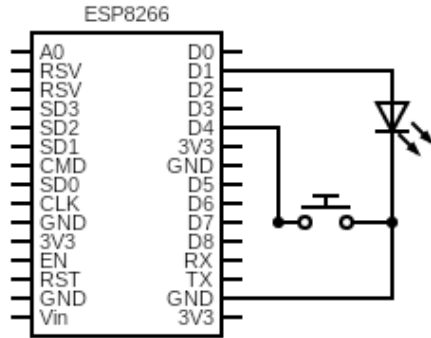


Figure 56: Circuit diagram for the button and ESP8266 to the left, and wired circuit to the right

For this to work, the Arduino code for the master and slave module were combined and modified a bit. The code for this can be seen in Listing 21 in Appendix A. The LED had to switch on and off when the button state was read as LOW instead of HIGH, and the pinMode in the setup section had to be specified to be a pullup, by writing `INPUT_PULLUP` [14]. This was done with the command shown in Listing 10.

```
1 pinMode(buttonPin, INPUT_PULLUP);
```

Listing 10: Specifying buttonPin as an INPUT_PULLUP in the slave code

The next step was to modify BADDY's source code to work with these modifications. The logic for the button had to be added to the rest of the code without affecting BADDY's behavior. The entire source code for BADDY is approximately 3800 lines long. As *Visual Studio Code*, together with *PlatformIO*, was used to configure BADDY with the original source code, it was natural to continue the development in *Visual Studio Code* rather than *Arduino IDE*. *Visual Studio Code* is more advanced and comes with practical functions that *Arduino IDE* doesn't have, and therefore more preferable when working with more complex projects. The modifications of the code can be seen in Listing 22 in Appendix A. Instead of making the LED in Figure 56 turn on when the button was pressed, a "smile" function called `printSmiley()`, creating a smiley face on BADDY's LED screen, was invoked. Figure 57 shows that BADDY is smiling when the button is pressed.

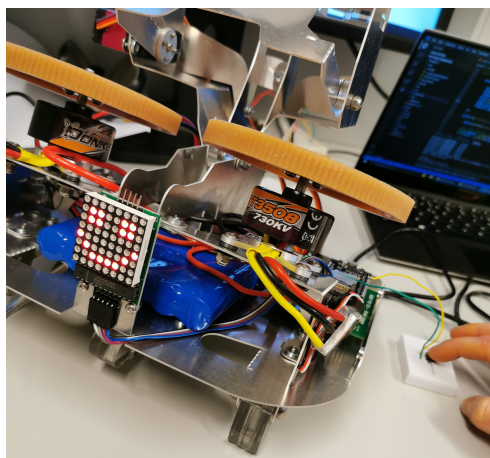


Figure 57: BADDY is smiling when the button is pressed

Figure 58 shows how the button was integrated with BADDY's PCB. There was no need for a separate pull-up resistor wired to the button, as the D4 pin on the ESP had an internal pull-up.

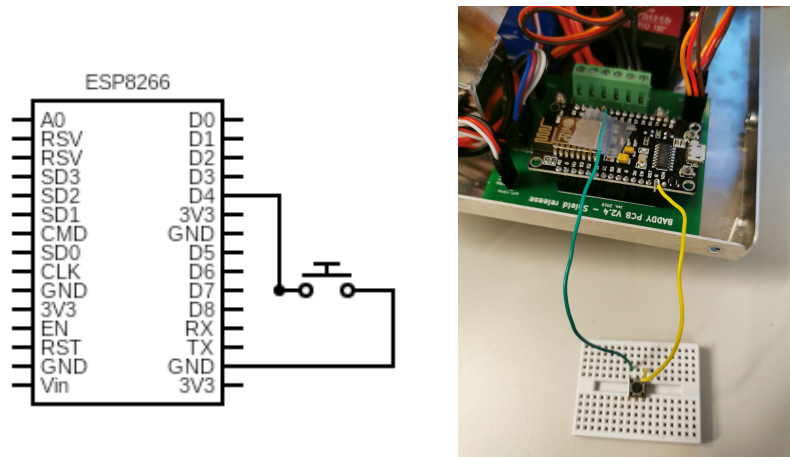


Figure 58: The button is wired with a short time solution to the ESP board, one wire to ground (yellow) and one wire to D4 (green) where internal pull-up for D4 is enabled

This step made sure the code was modified correctly, and the attached wires were not affecting other parts of BADDY's behavior. Some additional modifications had to be done to the code before the button could actually launch a shuttlecock. This was done later, but for now, the smiley face was enough confirmation that everything worked as wanted.

5.2 Wiring the Bluetooth module to BADDY's PCB

The second step was to implement the HC-08 module to the PCB instead of the wired button and change the code accordingly. This was done in the same way as with the button, the Bluetooth code was first made to work with the ESP board alone without any of the BADDY code, before it was implemented into BADDY's code. The Bluetooth code can be seen in Listing 23 in Appendix A, and the changes added to BADDY's code can be seen in Listing 24 in the same Appendix.

The master module was used and wired to the ESP board along with a LED lamp. This is shown in Figure 59. It worked as intended when the LED was turned on by pressing the controller button. The next step is shown in Figure 60. The LED was no longer necessary, instead, the LED board on BADDY was used for confirmation of the received signal. Figure 61 shows the temporary wiring.

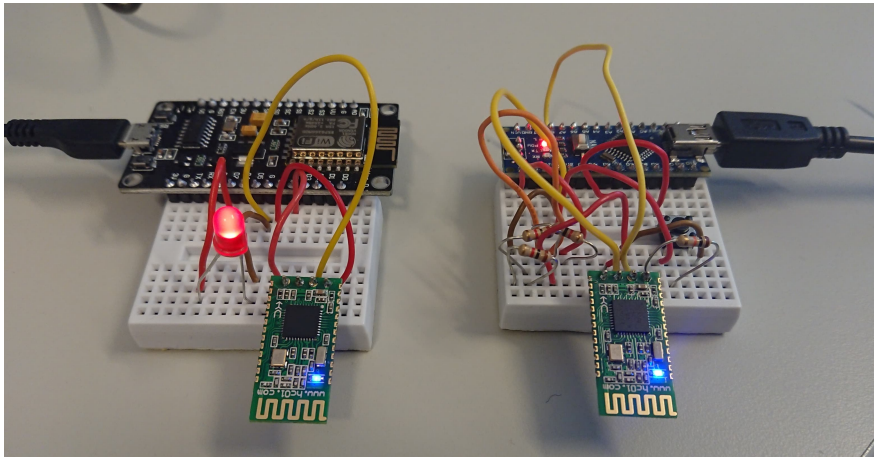


Figure 59: The HC-08 module is working with the ESP board

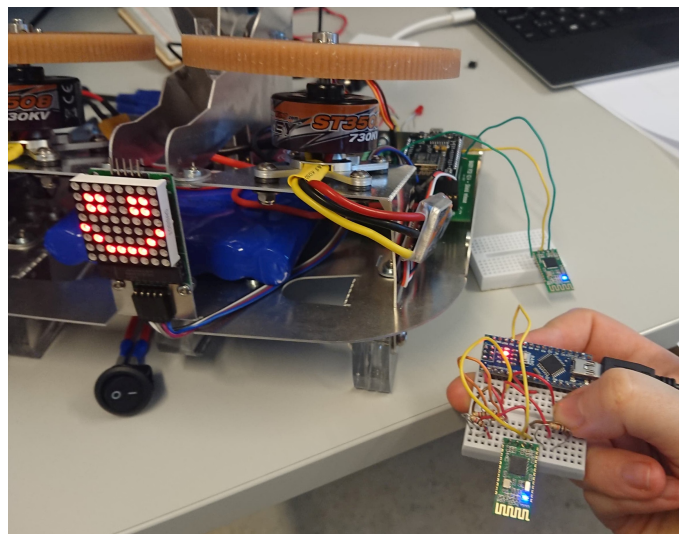


Figure 60: BADDY is smiling when the button on the controller is pressed

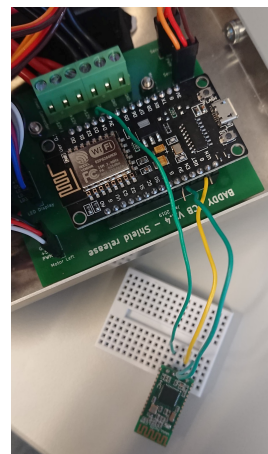
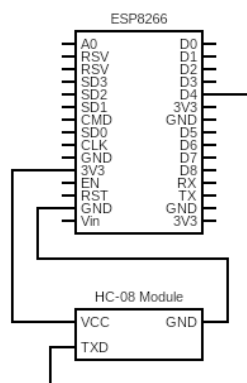


Figure 61: The HC-08 wired to the ESP board using a breadboard as a temporary solution. One wire to ground (yellow), one wire to D4 (green) and one wire to 3V (green)

The wired setup with a breadboard was only a temporary solution that would not be optimal in the long run. Due to the vibrations occurring inside BADDY when shuttlecocks were launched, this setup would risk getting broken. A more solid solution was therefore needed. Three female jumper wires were attached to three of the pins, ground, TX, and VCC, on the HC-08 module. While the other end of these wires was soldered onto three pins on BADDY's PCB, as can be seen in Figure 62. The use of female jumper wires makes it easy to detach the HC module from BADDY if it needs to be reprogrammed. The finished setup with BADDY's PCB and the HC-08 module inside BADDY can be seen in Figure 63. An optimal solution would be to integrate the HC-08 module on the PCB, but as discussed in section 7, replacing the ESP8266 with another one with Bluetooth, might be preferable.

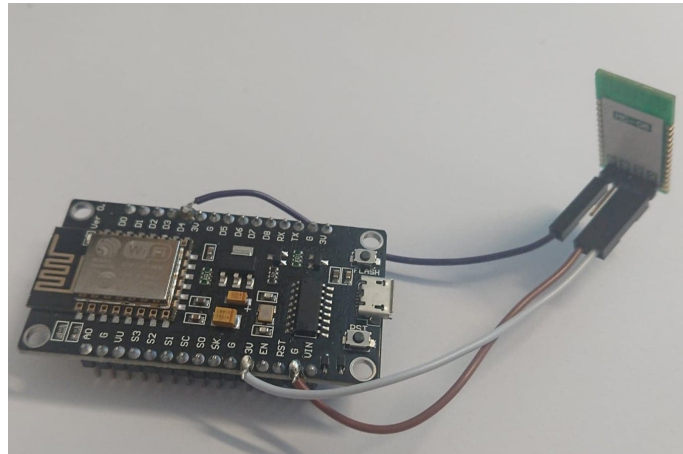


Figure 62: HC-08 module soldered to the BADDY ESP board

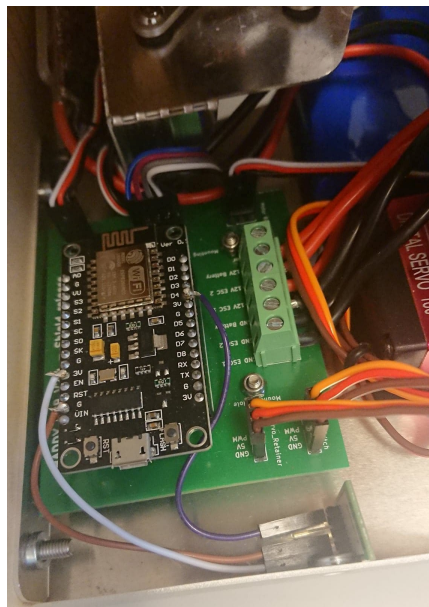


Figure 63: BADDY's ESP board with the soldered HC-08 module placed inside BADDY

5.3 Getting a shuttlecock to launch

As mentioned in section 4, the software files used in the BADDY project are available on the BADDY GitHub page [29]. To make it easy for everyone to build BADDY [83], the developers have put all the code in one .cpp file with the name *main*⁷. This file has to be modified to make the controller work with BADDY.

The HC-08 module was attached to BADDY without interrupting other functions and the ESP8266 could receive signals from the controller. Instead of making a smiley, a shuttlecock had to be launched. To do this, the BADDY source code had to be examined further. As for now, a shuttlecock is launched by sending a JSON object from the phone application to BADDY over the WiFi network. This had to change.

JSON stands for JavaScript Object Notation and is used for both storing and exchanging data [95]. JSON is formatted as a string and can be structured in two different ways. Either as a collection of pairs with names and values, such as objects, hash tables, and dictionaries. Or as an ordered list with values like arrays, vectors, and lists [65]. A JSON object is an unordered set of pairs with names and values. A JSON object is defined within a set of curly braces where the names are strings and the values are a valid JSON data type. The names and values are separated with a colon and the name/value pairs with a comma [96].

The JSON object that is sent from the application to BADDY, consists of four names with associated values. The format of this object can be seen in Listing 11. “Type” is either set to “ABT” or “SEQ”. When “Type” is defined as “ABT”, BADDY receives an abort message and the motors stop, while “SEQ” indicates that a sequence of shots is sent to BADDY. The next name in the JSON object is “Strokes”. The value for this is given as a list with numbers from 0 to 9. If the value is equal to 0, no stroke is configured and the sequence will end. The different values from 1 to 9 define different types of strokes and are defined in the `set_stroke(int stroke)` function from line 1430 in the code. “Speed” is also given as a list, where the list for “Strokes” and “Speed” are of the same length. “Speed” have values from 0 to 5, and define the time before the shot is launched. Here, the values from 0 to 5 respectively stand for launching every 0.5 second, 1st, 1.5, second, third, and fourth second. Listing 11 shows an example of a JSON object that can be sent to BADDY. In this object, the numbers defined for the speeds are within the range of 1 to 9 even though it has been stated that speeds only can have values from 0 to 5. Mr. Greslebin said in one of the interviews that the values outside this range, such as 6, 7, 8, and 9, as given in the example in Listing 11, are just placeholders for future releases in case it would be necessary to reallocate the speeds differently.

```
1 {  
2   "Type": "SEQ",  
3   "Strokes": [ 1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 11, 12, 13, 14, 15, 16, 17, 18,  
4     19, 2],  
5   "Speeds": [ 1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 1],  
6   "LoopMode": 0  
}
```

Listing 11: Format of a JSON object sent from the BADDY application to BADDY [30]

“LoopMode” is the final name that is defined in the JSON object. The value for this name is one single number. This number can be either set to 0, 1, or 2. Everything except these values is

⁷The original BADDY code used is included in the digital appendix (attached files) as a reference for the reader. For further development, get updated code from the BADDY GitHub [29]

undefined. When “LoopMode” is set to 0, this indicates no loop, and the shoots will be launched one time each. The number 1, indicates a loop, and 2 means a random loop where the shoots are launched in random order.

The first step of making the controller work with BADDY was to make BADDY launch one standard shot when the button on the controller was pressed. Further development would include the possibility of launching different shots depending on how the button is pressed, as stated in Section 2.6. The intention was to make the remote controller work with the same functionality as the application. To make them work the same way, a JSON object would have to be sent via Bluetooth from the remote controller to BADDY. At this point in the development phase, the controller had only been tested by sending one single character when the button was pressed, and then either making BADDY smile or light a LED lamp. The transmission of this character was done by using the code line given in Listing 12 in the code for the slave module.

```
1 Serial.print(bluetooth.write('1'));
```

Listing 12: Sending the character '1' via Bluetooth from the slave module

Several methods were tested in order to find a way to launch a shot when pressing the button on the controller. The first method that was tested, was based on the functionality of the application. The application creates, and transmits, a JSON object to BADDY when a sequence is created. The code in BADDY receives this object and processes it. Instead of sending a single character from the slave code, as shown in Listing 12, a JSON object was created as a string and sent to BADDY via Bluetooth. Listing 13 shows the code that was implemented in the slave code for creating a JSON object and sending it. For this object, the lists for both “Strokes” and “Speeds” contained only two values, where the first was four. This would initiate a shot to the middle of the court after 3 seconds. The second value in both lists was zero. As mentioned before, when “Strokes” is assigned the value zero, the sequence would end since no stroke had been configured for this value.

```
1 char json[] = "{\"Type\":\"SEQ\",\"Strokes\":[4,0],\"Speeds\":[4,0],\"LoopMode\":0}";
2
3 void loop()
4 {
5     Serial.print(bluetooth.write(json));
6 }
```

Listing 13: Creating a JSON object and sending it via Bluetooth

This caused some problems in terms of how to actually transmit and receive this object. Since the UART communication only allows for one character to be sent at a time, sending the whole JSON object would be cumbersome as the entire object would have to be split up into characters and send one at a time. The solution of sending an actual JSON object via Bluetooth was therefore rejected, and another method was tested.

The next method that was tested, was to move the creation of the JSON object from the slave code to BADDY’s code. By implementing the JSON object and initiating this object in a function in BADDY’s code, it was possible to go back to sending only a character from the remote controller. When BADDY receives this character, it would launch the defined shot or sequence of shots for this value.

BADDY is embedded with an ESP8266 WiFi module that does not support Bluetooth. In order to make BADDY receive data through the HC-08 Bluetooth module, the SoftwareSerial library

had to be included and configured in BADDY's code and the command "BTserial.read()" was needed to receive data from the remote controller. The code that was implemented into BADDY's loop()-function for checking when the controller was transmitting data, can be seen in Listing 14. If the controller was transmitting, a new function *void readDataFromController()* seen in Listing 15, is called. The code worked as intended, and shuttles could be launched by using this method of implementation.

```
1 void loop() {
2     //If the controller is sending data, read the data
3     while(BTserial.available() >0){
4         readDataFromController();
5     }
6     //ArduinoOTA.handle();
7
8     MDNS.update();
9
10    if (flag_new_sequence)
11    {
12        sequencefetch(JsonSequence);
13        ReadSequence();
14    }
15
16    if (flag_test_shot)
17    {
18        delay(1500); // wait till motors get their nominal speed
19        Serial.println("Shooting shuttlecock...");
20        servo_fire();
21        flag_test_shot = 0;
22        MotorRight.writeMicroseconds(STOP);
23        MotorLeft.writeMicroseconds(STOP);
24        Serial.println("Stop motors!");
25    }
26
27    if (flag_print_smiley)
28    {
29        printSmiley();
30        flag_print_smiley = 0;
31    }
32    if (flag_print_frown)
33    {
34        printFrown();
35        flag_print_frown = 0;
36    }
37    if (flag_print_squares)
38    {
39        printSquares();
40        flag_print_squares = 0;
41    }
42
43 }
```

Listing 14: BADDY's loop-function with additional code for receiving Bluetooth signals on line 2 to 5

The below code snippet shows that different cases could be defined, where different shots or sequences could be initiated when receiving different values from the remote controller. If the controller sends the character '1', BADDY would launch a drive shot, and if the controller sends '2', BADDY would launch four drive shots in a sequence. Further development of the remote controller could include, as previously mentioned, adding more functionality to the button and extend the

number of cases possible.

```
1 //Function for reading data from a remote controller
2 void readDataFromController() {
3     char data = 0; // variable for received data
4     data = BTserial.read(); // Reads the data from the serial port
5     //Different types of strokes according to the data sent from the controller
6     switch(data){
7         case '1': //One short press gives one middle drive shot
8             JsonSequence = "{\"Type\": \"SEQ\", \"Strokes\": [4,0], \"Speeds\": [4,0], \"
LoopMode\": 0}";
9             break;
10        case '2': //One long press gives four middle drive shot
11            JsonSequence = "{\"Type\": \"SEQ\", \"Strokes\": [4,4,4,4,0], \"Speeds\":
[4,4,4,4,0], \"LoopMode\": 0}";
12            break;
13
14        }
15        data = 0; // variable for received data changes back to 0
16        flag_new_sequence = true; //Flag is changed, and will trigger the functions
sequencefetch(JsonSequence) and ReadSequence()
17 }
```

Listing 15: A function implemented in BADDY's code to receive data via Bluetooth and initializing a sequence when different data is received

Listing 16 shows the creation of an instance of a SoftwareSerial object at the beginning of BADDY's code, where the Tx (transmission line) and Rx (receiver line) are initialized first. This is a one-way communication where BADDY only receives data, but the SoftwareSerial object requires both variables to be created.

```
1 int bluetoothTx = 1; // TX pin
2 int bluetoothRx = 2; // RX pin, D4 on ESP board
3 SoftwareSerial BTserial(bluetoothRx, bluetoothTx); // Creating a SoftwareSerial
object - Rx, Tx
```

Listing 16: Creation of a SoftwareSerial instance in the beginning of BADDY's code

Listing 17 shows a line of code needed in the setup()-function for setting the data rate in bits per second (baud) for Bluetooth transmission.

```
1 BTserial.begin(9600); // Begin The Bluetooth Module, defaults 9600bps
```

Listing 17: Setting the transmission baud rate in the setup()-function

By adding these changes to BADDY's original source code, the remote controller works with BADDY, both with and without a phone connected to BADDY. The BADDY application on a phone could be used at the same time as the controller is connected. Everything that has been changed and added to BADDY's code can be seen in Listing 25 in Appendix A.

6 Further implementation of the remote controller

A prototype of a case for the remote controller was developed in the prestudy and documented in the report [67]. As neither the case design nor the hardware were completely defined in the prestudy, several modifications had to be done to each of these parts in order to achieve the intended functionality. The software had to be adapted to these modifications and needed further development. Further development of the case had to comply with the technical requirements stated in section 5.1 in the prestudy report [67]. The necessary modifications are listed below.

1. Create an ergonomic design for the case
2. Add an open and closing mechanism to the case
3. Design the case with respect to additional buttons and a LED lamp
4. Choose the final hardware components with regards to the total volume
5. Develop a final wired circuit with all components
6. Install the components in the case
7. Develop software to work with the chosen components
8. Add functionality to the controller by developing the software further

6.1 Case design process and improvements

The prototype case from the prestudy and the original sketch idea can be seen in Figure 64. Several factors separate these two. The initial prototype was designed to be as easy to print as possible, as it was designed to test basic concepts of buttons, size of the case, and possible ways to attach it to a hand. The next steps consisted of improving the design to make it better to hold on to, adding a closing mechanism, and adding a power button. The 3D drawings were created using the program *Autodesk Fusion 360*, and *Prusa Slicer* was used to make the drawings ready for printing. How these programs are used, is described in section 5.2.3 in the prestudy report [67].

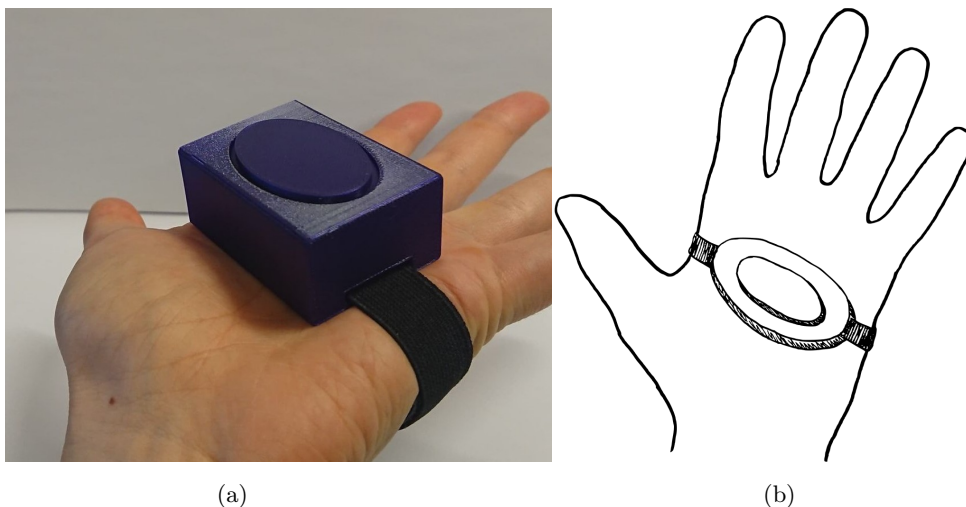


Figure 64: The prototype (a), and the original sketched idea (b)

In the 3D printer Lab, three types of printers were available; Original Prusa i3 MK3, Original Prusa i3 MK3S, and Original Prusa i3 MK2.5. The difference between these printers is not very notable in terms of the quality of the printing, but the setup of the machines was quite different. The three different types were tested and gave quite similar results.

The first attempt to design a spherical case is shown in Figure 65. This was designed to be a quite small case so that the 3D printing would go fast. This case was also designed with a simple closing mechanism by creating an extension of the edge for both parts. These extensions would fit inside each other as seen in Figure 66. The small design did result in a short printing time, but the printed case was too small to function for its purpose.



Figure 65: A small test print

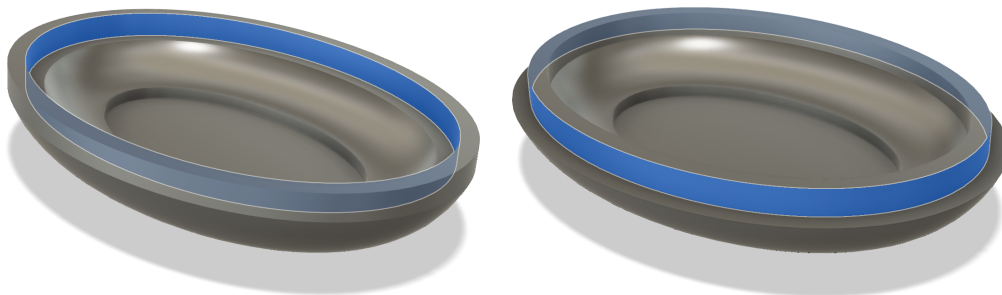


Figure 66: The friction surface area is highlighted

Further development of the design of the case, included getting the proper dimensions so that the controller would fit comfortably inside a hand. The overall size of the case and thickness of the walls was increased. By increasing the thickness of the walls, the closing mechanism would have a larger contact surface with increased friction between the two parts. This led to the parts sitting better together and being able to withstand more, like being dropped to the ground. The closing mechanism with edges that fitted inside each other and hold the part together can be seen in Figure 66. The idea was that the friction between the two parts would keep them together, and with a bigger overlapping area, they would not fall apart so easily. After testing this design, it was concluded that the increased thickness of the walls did not affect the functionality of the closing mechanism. Something that did affect this, was how far these half edges, illustrated in blue in Figure 66, were extended from the walls of the case.

The printed parts fitted well together and the friction along the contact surfaces was large enough to hold the parts together. It was also easy to take them apart. As seen in Figure 67, the parts

for the same case were printed in two different printing jobs, with two different printers. By more testings and prints, it was learned that it would be preferable for the two parts to be printed in the same job. This would not only require less equipment, but it would also improve the end result.

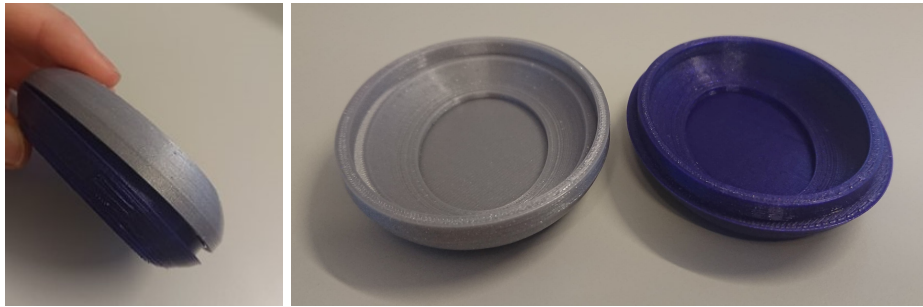


Figure 67: Bigger test print with working closing mechanism where the parts were printed on different printers

An alternative to this closing and opening mechanism could be the usage of screws. Screws would provide a solid and trustfully solution, but it would also increase the need for more tools and materials. In the prestudy, a case with screw holes was 3D printed, and the experience showed that screw holes were filled with support material which was almost impossible to remove. Another disadvantage of using screws would be the need for a screwdriver to open the case.

To easily open the case, the next step for the development of the prototype was to add a small notch along the edge of one of the parts. As previously mentioned, the closing mechanism was not improved by the increased thickness of the walls. The walls were therefore made thinner to increase the space inside the controller and accommodate all the components. It was also created holes for the rubber band, and the launching button was added. From the prestudy, the result showed that the separate button solution worked better than a half attached solution. Therefore, a separate button was created this time. The button solution will be discussed in more detail later. This print was the first prototype and can be seen in Figure 68 and 69. When placing the case in a hand, a natural place for the button could be on the side of the case, along the edge between the part. By placing a button here, it would be easy to reach with the thumb. This could be an improvement for later. The dimensions for the case were changed a few times, in order to find a good balance between an ergonomic shape that fits well in the hand, fast printing time, and solidness. The size of the controller case that has been developed during this project has been adapted to the hand of a female adult at the same time as the size of the components has been taken into account.

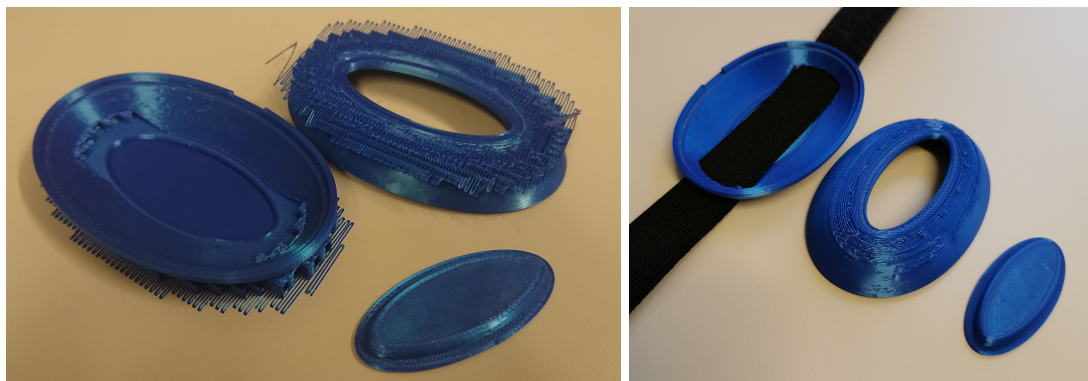


Figure 68: The first prototype



Figure 69: The first prototype attached to a hand and using the notch to open it

Figure 70 is a close-up of the printed controller, where the support material has been removed. The surface that was printed with support structure is rough and not as even as the parts that did not need support during printing. The roughness of the surface would not be a problem in terms of using the controller. However, this is something that can be improved either by designing a case that does not need support structure when printing or by sanding the surface.

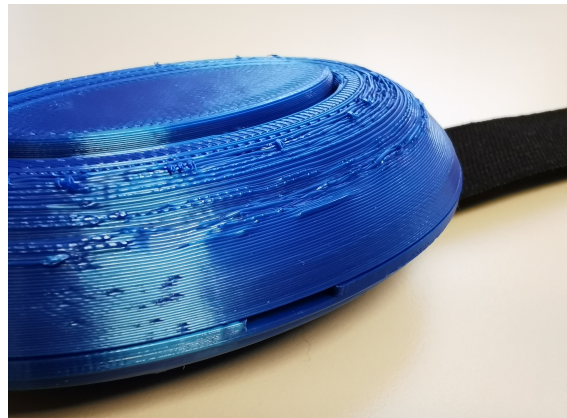


Figure 70: Close-up of the face with removed support material

Previous experience has shown that printing with support structure takes more time and can be difficult to remove without breaking the print afterward. The surfaces with support material will also be more uneven than the surfaces without. In the printing lab, there were examples of prints made without support with decent results. It was therefore decided to print a new case without the use of any support structure. Figure 71 shows the difference between printing with and without support structure. The surface for the case without support structure was less even than the surfaces of the cases printed with support. The printing of this case was monitored, and it was observed that the result was worse than the case printed with support. The printing job was therefore aborted before it was finished. After this test, it was concluded that in order to achieve the best result, further printing would be performed with the use of support structure, even considering the extra printing time.

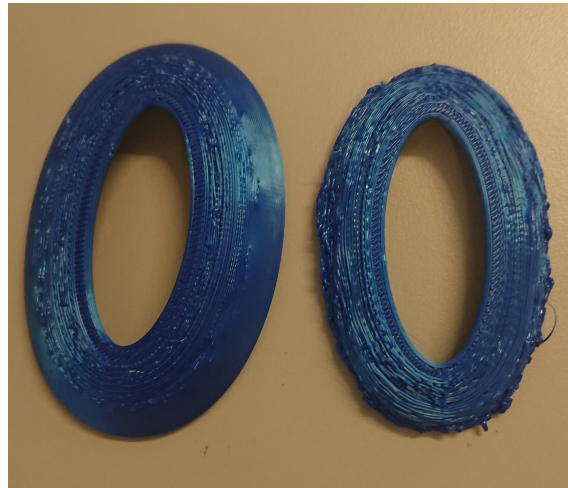


Figure 71: Two different printing jobs, where one case was printed with support structure (left) and the unfinished printed case without support structure (right)

The printing without support, shown to the right in Figure 71, turned out bad as the angle between the walls and the vertical plane was too large. Since the printing was performed without support structure, the first layer had to cover large enough areas so that the second layer could be supported by the first layer. Due to the large angle, this was not the case and the second layer was not provided with the necessary support. This became a recurring problem for the next layers as well and resulted in a poor result.

To avoid using support in the printing, overhang structures with a larger angle than 45° from the vertical had to be avoided [38]. An alternative way to design the case was to decrease the angle between the walls and the vertical axis so that it was less than 45° , as shown in Figure 72. The design of this case was pointy and the 3D drawing program was not able to add the closing mechanism to the edges of the walls. This alternative was therefore rejected in favor of printing with support.

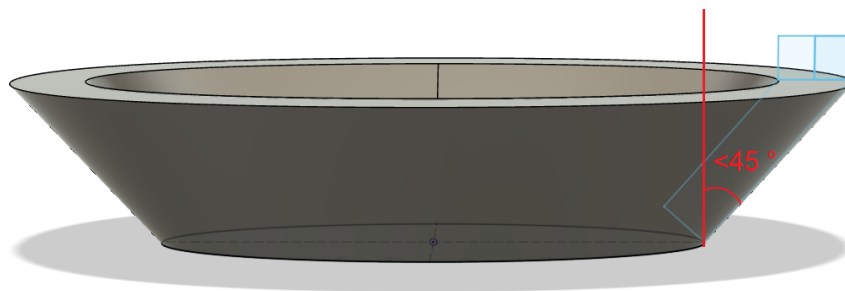


Figure 72: Alternative 3D drawing

By trying to avoid the use of support structure during printing, the design and modeling of the case became more complicated and the finished product would have been less ergonomic in terms of holding and using the case. It was therefore decided to go back to the original design with curved walls and use support structure when printing. By using different tools, the support was removed and the surfaces were polished.

The next step was to establish the button solutions. A power button was needed to turn the

controller on and off, and the launching button needed further development. The solution for the launching button that was created for the first prototype, would be in need of some sort of sliding guidelines and a spring, as illustrated in Figure 73, in order to work as a push button. This could have been performed by using 3D printers, but was not done due to lack of available printers. An alternative way to create a functioning button was to purchase and use a manufactured push button. Such a push button is created with a simple switch mechanism and is easy to incorporate into own projects. There are many different types of buttons on the market today, and a few different ones were purchased to find the one that was best suited for the controller. Some of the buttons that were available and purchased, can be seen in Figure 74.

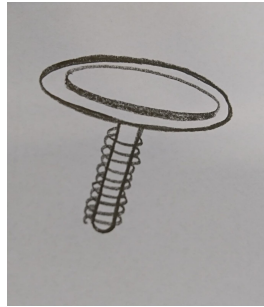


Figure 73: Sketch of a push button with a spring

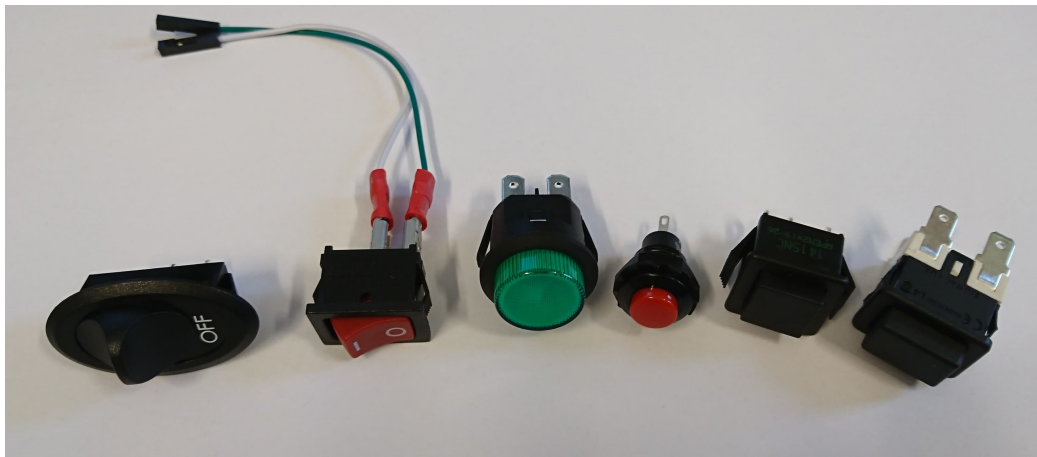


Figure 74: Some of the buttons available

It was hard to find a button with the right proportions for the case. Many of the purchased buttons had a small push area and were too long to fit inside the case, even when the case was expanded a few millimeters. A possible way of solving the problem with the small push area was to combine a 3D printed part and a manufactured button as shown in Figure 75. This option would provide the button with a bigger push area, but after some testing, it was concluded that the purchased button was big enough and easy to hit.

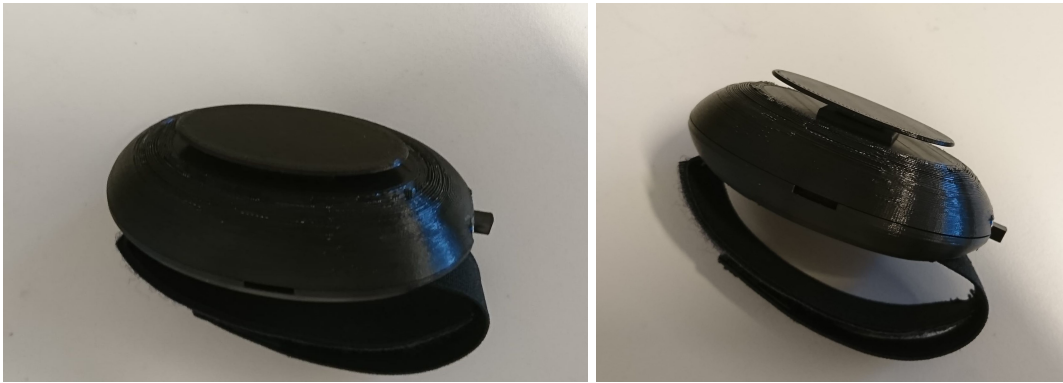


Figure 75: Combination of 3D printed and manufactured button

The switch button was chosen because of the small size. It has three positions, ON - OFF - ON, which is one more than the controller needed at this point. The extra position opens up the possibility of adding extra functionality later, e.g. adding a different mode for the controller. The buttons that were chosen to use, are shown in Figure 76.

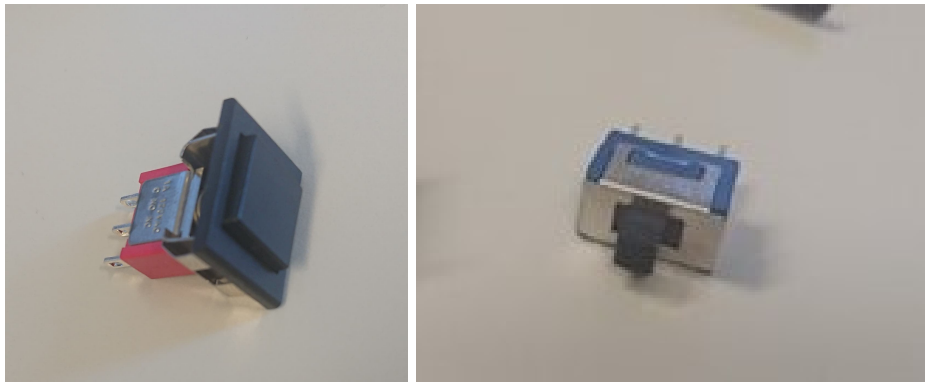


Figure 76: The buttons used, push button to the left and switch button to the right

The holes for the buttons were designed so that the buttons would fit perfectly. After 3D printing the case it was observed that some of the support structure was left behind in the holes after everything had been removed. This extra support structure had to be removed with sandpaper to be able to place the buttons in the holes.

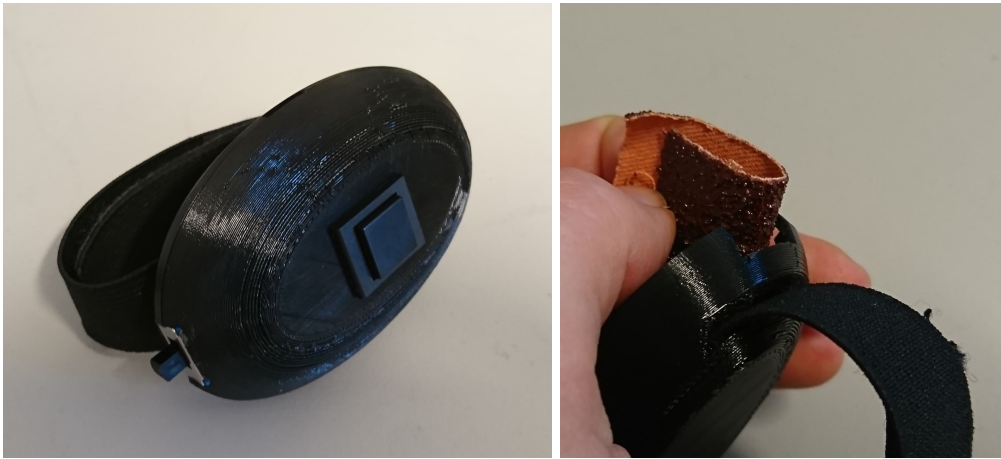


Figure 77: The case with the manufactured buttons implemented. A bit of sandpaper was needed to make room for the buttons

The entire designing process for the case, both in terms of the case itself, but also with regard to the buttons, was an iterative process. When developing the solutions for the buttons, the thickness of the case had to be changed and the correct dimensions of the holes for the buttons had to be found. This was done by designing, printing, testing, and redesigning multiple times. This implied that the 3D drawings had to be modified and printed several times. The final 3D drawings can be seen in Figure 78 and the dimensions can be seen in Figure 79. In addition to the holes for the buttons, a circular hole was placed at the opposite side of the power button, as seen in Figure 78. This hole was created in order to include a LED lamp in the controller. This lamp could be used as an indicator to show whether the controller was on or off and if it was connected to BADDY. Further development of the controller could consist of using the LED for other functionalities.

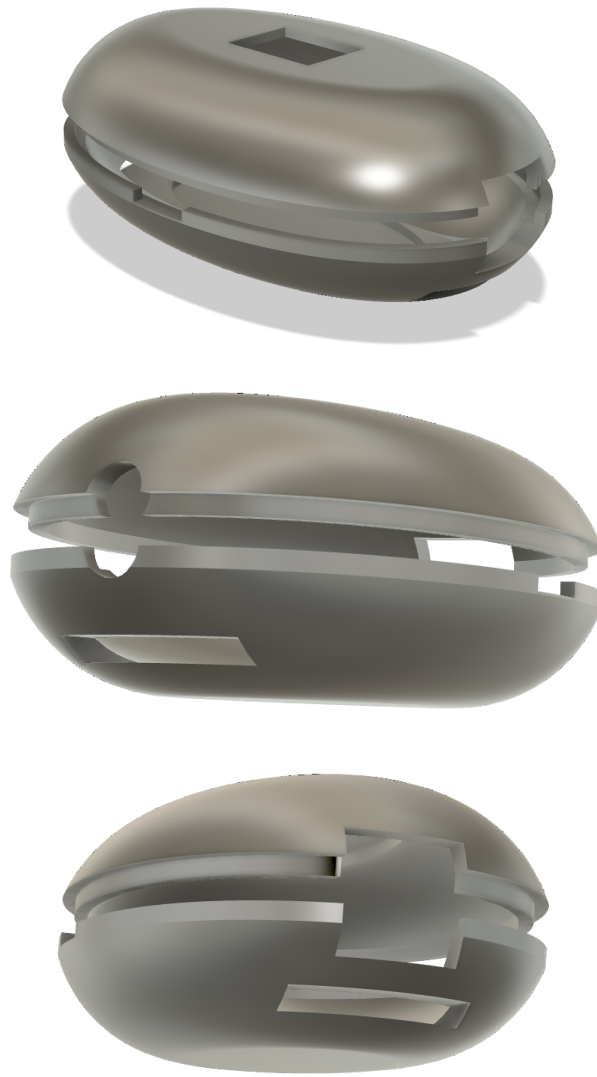


Figure 78: Final 3D drawings

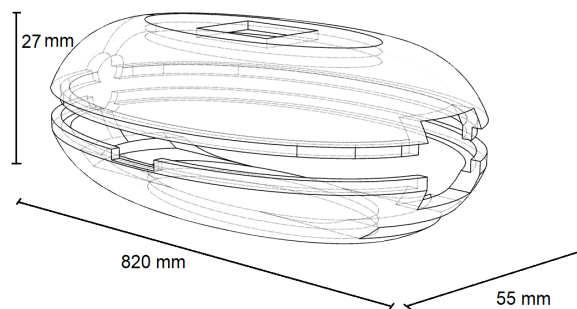


Figure 79: Dimensions of the case

The final case can be seen in Figure 80. Two different ways of pressing the launching button are shown in Figure 81, with the thumb or with the other fingers. Figure 82 shows the use of the

power button and how it is easily accessible with the thumb.



Figure 80: The final prototype

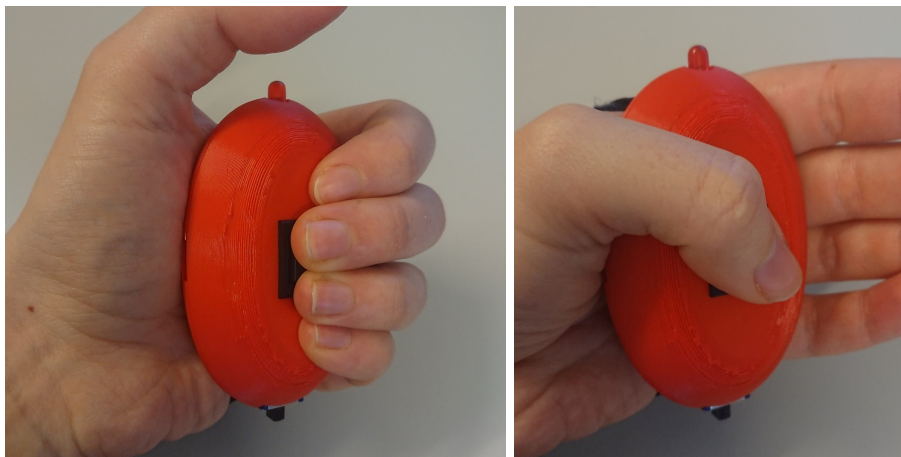


Figure 81: Two possible ways of pressing the launching button



Figure 82: A button on top of the case can in future development be used to e.g. switch between different modes

The development of the case has, as stated, been an iterative process and various designs have been tested out. Figure 83 is an overview image showing the entire case development process from the very first 3D print performed in the prestudy, to the final prototype.

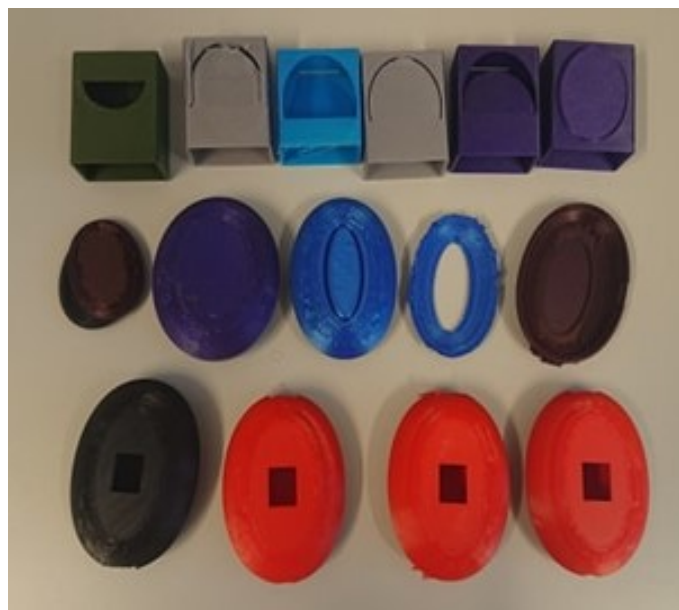


Figure 83: Case development

6.2 Hardware development

The hardware developed during the prestudy consisted of two HC-08 modules connected to two separate Arduino Uno boards. These modules were connected via Bluetooth and were able to light a LED lamp connected to the master module when pressing a button connected to the slave module. The setup of the wired master and slave circuits can be seen in Figure 84, and an illustration of the signal transmission can be seen in Figure 85. This was used as a starting point and developed further in order to make the controller work with BADDY.

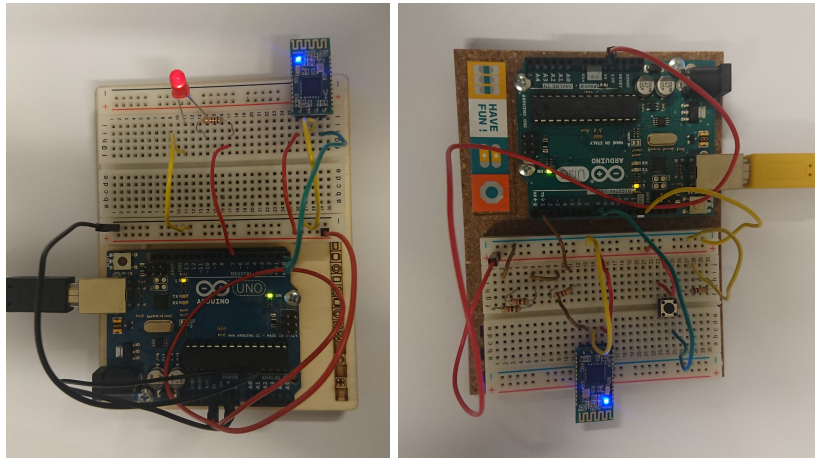


Figure 84: Wired master module to the left and slave module to the right

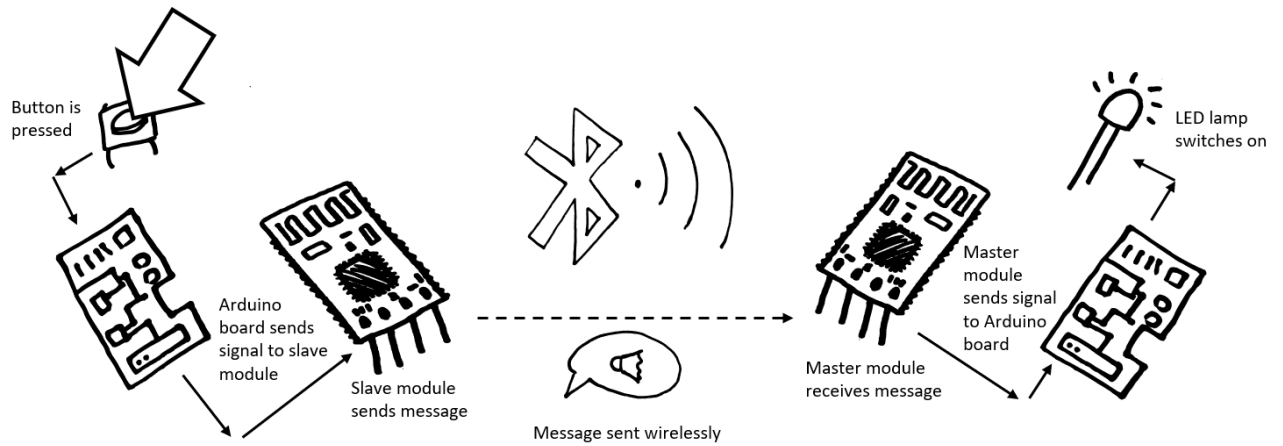


Figure 85: Illustration of signal transmission with two Arduino boards

In order to make this hardware setup work with BADDY, several modifications and changes had to be done. Figure 86 shows an illustration of the intended signal transmission between a remote controller and BADDY. As shown in the illustration, the slave module would be embedded with the launching button, and the entire setup for this part would have to fit inside the controller case. A possible way of making the wired slave circuit smaller was to replace the Arduino Uno board with a smaller one. In addition to changing the board, a switch button and LED lamp had to be added to the slave circuit, and all the components had to be soldered together and placed inside the case. The placement and soldering of the components had to be performed in a secure and robust way so that the controller could withstand shocks from falling to the ground and rough treatment. The master module had to be integrated with the BADDY PCB and was done in section 5.

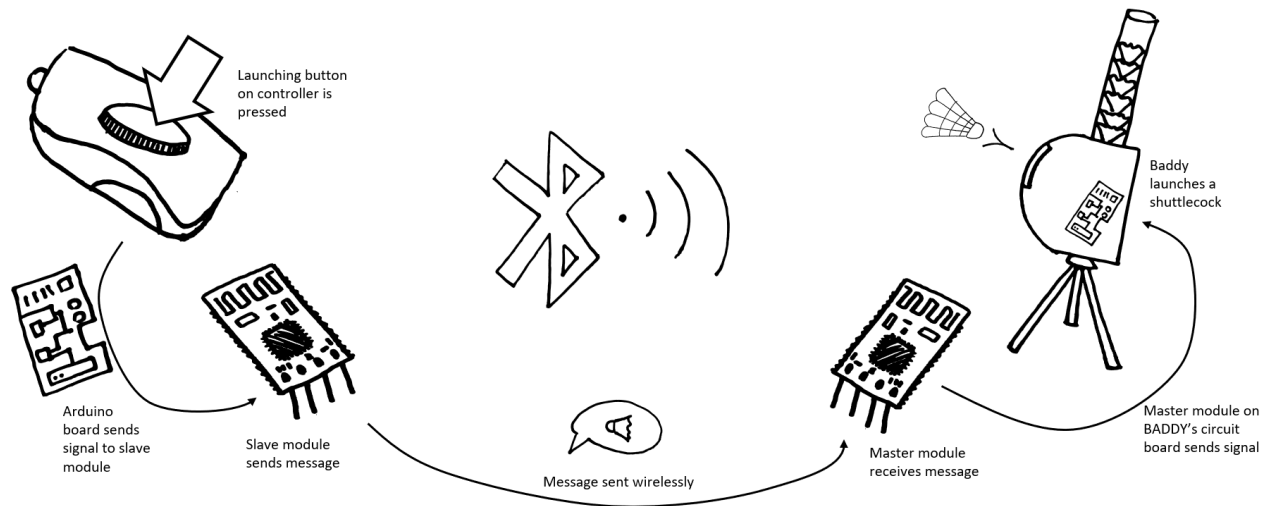


Figure 86: Illustration of the signal transmission between a controller and BADDY. The modules are part of the controller and BADDY

It was desirable to have the controller as small as possible, which meant that the components also had to be as small as possible to fit inside the case. Figure 84 shows the wired circuit with an HC-08 module and an Arduino Uno board. From the figure it can be observed that the Arduino board is the component that limits the size of the final setup. By changing the Arduino Uno to a smaller board, the overall size of the controller could be decreased significantly. This would be desirable so that the controller would be an improvement from the use of a smartphone. It was concluded that the board either had to be replaced with a smaller one or removed. Some of the different alternative options are listed below.

- Keep using the HC-08 module
 - Program the CC2540 chip
 - Add an Arduino Nano board that is small enough
- Use another module than HC-08
 - Program an nRF52840 Dongle to work without the need of an Arduino
 - Find another module
- Switch from HC-08 Bluetooth modules to an Arduino Nano 33 BLE or similar

The difficulty of implementing, size, and the price of the components are factors that should be considered when choosing a solution. In Table 1, the pros and cons of the five different options are listed.

Solution	Pros	Cons
Program the CC2540 chip	Only needing one component, already have it in possession	Have to program everything again and in a new language
Add an Arduino Nano	Can use the code from previous	Need of two components, a bit bigger
Program the nRF52840 Dongle	Only needing one component, small, already have it in possession	Completely new component with a new language, got advice of not using it (quite complex)
Find another module	Can possible find a better, more simple solution to a lower cost	Have to start over, do research, waiting for the component to arrive, more uncertainty
Arduino Nano 33 BLE	Only needing one component	A bit bigger, have to order it, small changes to code

Table 1: Pros and cons for the different hardware solutions

The second option from Table 1, using an Arduino Nano board with the HC-08 module, was the solution chosen to proceed with. It seemed like the best alternative both in terms of accessibility, time frame, skill range, and re-usage of the code. It was not the smallest alternative, but considering the final case prototype that was developed, see section 6.1, this solution would fit inside the controller. The already written code, would not need much modification in order to work with the new board. An Arduino Nano board was also available and ready to use, so there was no need to purchase new ones and wait for them to arrive.

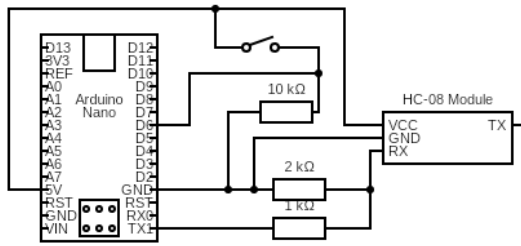
6.2.1 Remote controller with Arduino Nano and Nano Every

The slave module was first modified and wired to work with the Arduino Nano, which later was replaced with an Arduino Nano Every. Neither the Arduino Nano nor Nano Every is embedded with Bluetooth or BLE. This required both the Arduino boards to be wired to the HC-08 Bluetooth module in order to communicate with BADDY. The modifications done on the slave module mainly consisted of replacing the previously used Arduino Uno with an Arduino Nano, later replaced with a Nano Every, and switching to a smaller breadboard. As seen in Figure 87, the Arduino Nano is connected to the HC-08 module and an external push button. As stated in the prestudy and shown in Figure 87, a voltage divider had to be used for the connecting line between the Arduino and the HC-08 module to prevent the module from burning [67]. The wired circuits for the Uno and Nano boards were exactly the same, and the same pins were used for both boards. Due to this, no changes had to be done to the code for the slave module when changing the Arduino board. The push button needed a pull-down resistor since the used pin didn't have an internal pull-down.

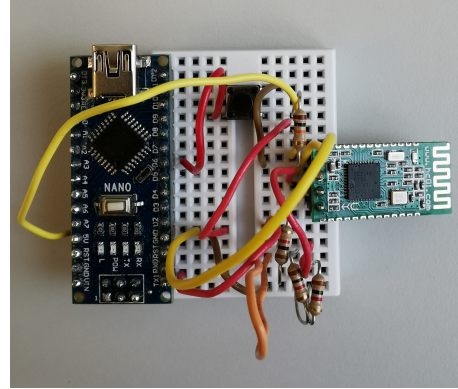
The input pins on the Arduino boards could be used with an internal pull-up resistor [43]. The internal pull-up would be activated in the code by writing `pinMode(pinNumber, INPUT_PULLUP);`. By implementing this to the button pin would make the wired 10 k Ω resistor unnecessary, and the wired circuit would have been simplified. This was discovered after the soldering of the components had taken place.

The Arduino TX pin operates with 5 V [11], while the Bluetooth module operate with a logic voltage level of 3.3 V for the RX pin [82]. To prevent the module from burning, a voltage divider had to be used to separate the TX pin (HC-08 module) from the RX pin (Arduino Nano). Formula 1 calculates the output voltage based on the source voltage and the resistance from two resistors. In order to achieve an output voltage, V_{out} , of 3.3 V, with a source voltage, V_{in} , of 5 V from the Arduino, the resistors, R_1 and R_2 , must have the values of 1000 Ω and 2000 Ω , respectively.

$$V_{out} = V_{in} * \frac{R_2}{R_1 + R_2} \quad (1)$$



(a)



(b)

Figure 87: Circuit diagram for the slave module (a), and the wired circuit with an Arduino Nano and HC-08 Bluetooth module (b)

Some problems occurred when re-wiring the slave module. First off, the workshop was closed due to COVID-19, which meant that the necessary equipment needed to get started with the slave module was not available. Second, the Arduino Nano was soldered incorrectly with the 2x3 pin, as seen in Figure 88. This caused some problems when trying to connect the Arduino Nano to the breadboard and a different solution had to be used. Given the wrongly soldered headers on the Arduino Nano, it was only possible to connect the pins on one side of the Arduino to the edge of the board as seen in Figure 87b. An attempt of removing the misplaced 2x3 pins by desoldering them was carried out without success. Both a desoldering pump and copper mesh, as seen in Figure 89, were used for the attempt. A reason why the desoldering attempt was unsuccessful, might have been because all the 6 pins were fastened together. If one of the pins were to come loose, the other 5 pins would still be fastened to the Arduino board. This could have been solved by using small pliers to cut the pins from each other and drag out the pins, one by one. As this Arduino Nano was only used for testing and during the development process, the desoldering attempt was aborted, and a new Arduino Nano Every without pins was purchased for use inside the controller.

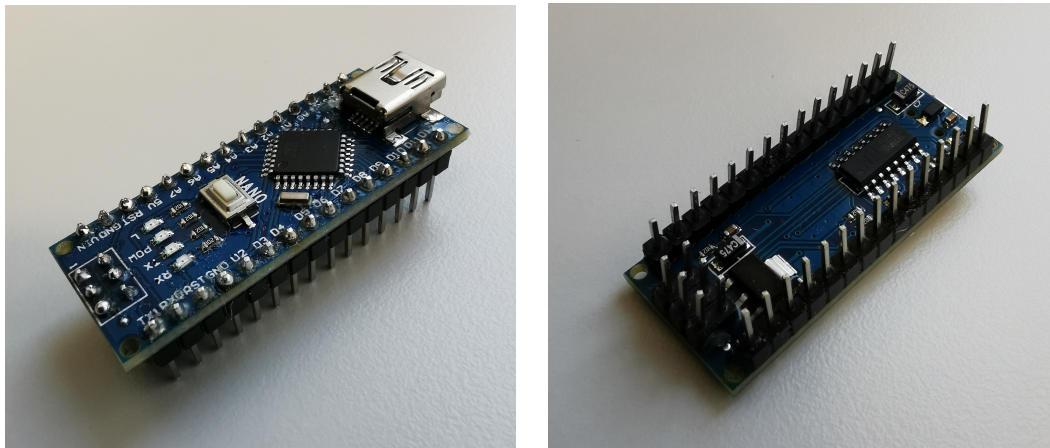


Figure 88: Arduino Nano soldered with the 2x3 pin headers on incorrect side

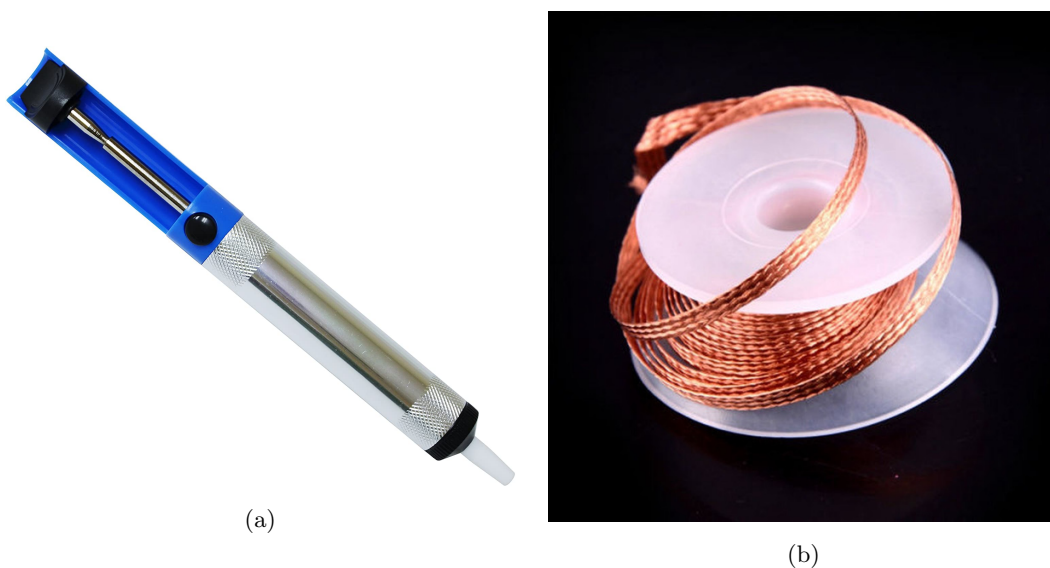


Figure 89: A desoldering pump [9] (a) and copper mesh used for desoldering [97] (b)

It was difficult to get hold of an Arduino Nano that didn't have the pins soldered to the board, as none of the websites NTNU uses for purchases had this. This was an unexpected problem. As previously tested, desoldering the pins was not an easy task and would have been time-consuming. Even with a Nano soldered in the correct way, it would not fit inside the controller case as the pins would take up too much space and get in the way of the other components.

After some research on both the Nano and other boards, the Arduino Nano Every seemed like a good alternative as this board was much easier to get hold of without the soldered pins attached to it. According to the Arduino Team [88], Every is backward compatible with Nano. This meant that it would be possible to switch from an ordinary Arduino Nano board to an Every board without making any changes to the code or wiring. The Every is cheaper than the Nano [15], so the change would decrease the total price of the controller. Every is also embedded with more memory and has a more powerful processor than the Nano. However, this is not an upgrade that is needed for the controller, as the code neither uses much memory nor includes time-consuming tasks.

The controller was in need of a power button and a separate power supply. The setup for this was tested with a breadboard in order to get a better overview of the wiring and to be able to fit all the components. The setup for this can be seen in Figure 90. In order to make all the components fit inside the controller, the breadboard had to be removed and the components had to be soldered directly to each other. The size of the breadboard was therefore irrelevant during the testing process.

A double coin battery holder was added to the Nano's VIN (input voltage) and GND (ground) pins, and a switch button was placed between the battery holder and the VIN pin, as can be seen in both Figure 90 and 91. The circuit is open when the switch is off and closed when the switch is on, providing the Nano with enough power to turn on. The push button was replaced with a bigger one, and a LED lamp was added and connected to a pin.

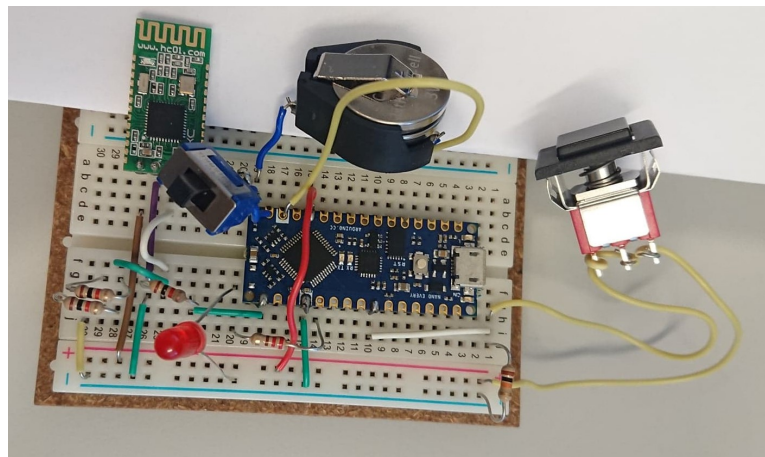


Figure 90: The Nano Every module wired with all components needed for the controller

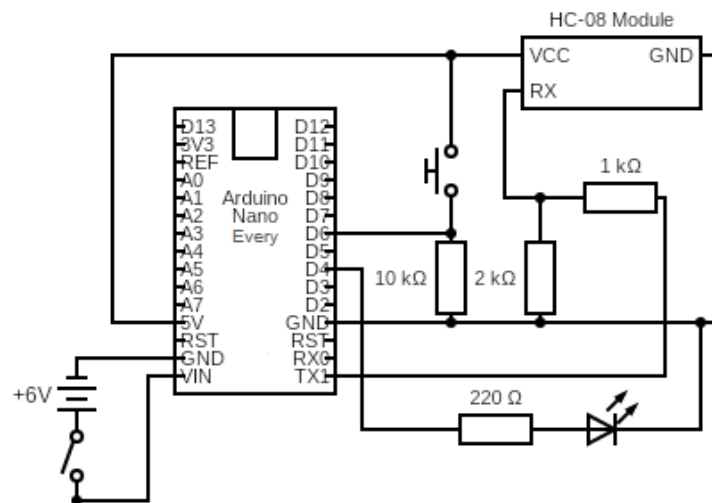


Figure 91: Final circuit of all the necessary components for the controller

6.2.2 Components

All Arduino boards that have been used during the development process, the Uno, Nano, and Nano Every, required an input voltage of 6 V to operate reliably. Meaning that the single coin battery used in the prestudy did not provide the boards with enough power. By adding an extra coin battery, the boards would get enough power and this solution would still fit inside the case. The two batteries were connected in series with a battery holder, which led to the voltage level being doubled from 3 to 6 V [85]. An alternative would have been to replace the Arduino Nano with an Adafruit Pro Trinket 3V [5], which can operate with an input voltage of 3 V. This was not done due to the limited time scope.



Figure 92: Double coin battery holder

The process of choosing buttons is explained in section 6.1, and the buttons that were used for the final prototype, are shown in Figure 93. The push button switch has three legs, a Momentary SPDT button, but only two legs are used. The NO (normally open) and C (common) [76] legs are used to make the circuit closed when the button is pressed. The button has (ON)-OFF functionality [41], which means that when you release the button, it returns to a normally open circuit position. The switch button is also an SPDT button with an ON-OFF-ON functionality. Two legs are used for controlling the power supply to the controller. When the toggle is placed in the middle, the circuit is open and power is off. When the toggle is placed to the right, the circuit is closed and the controller is on.

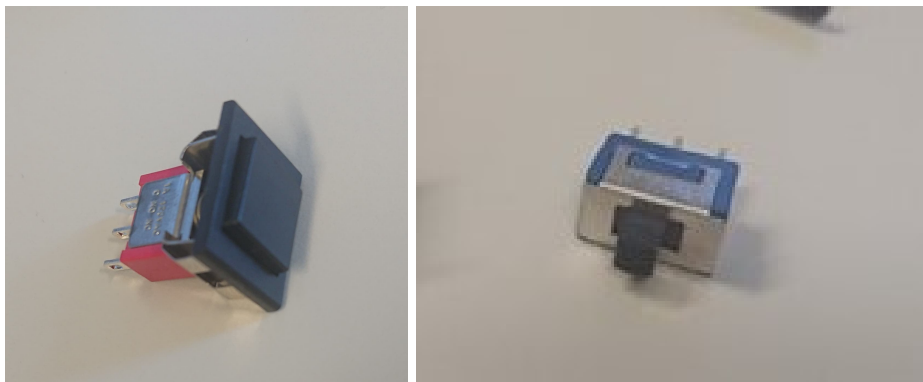


Figure 93: The buttons used, push button to the left and switch button to the right

As it was chosen to implement the switch button at a later time than the launching button, a simple circuit shown in Figure 94, was created to test its functionality.

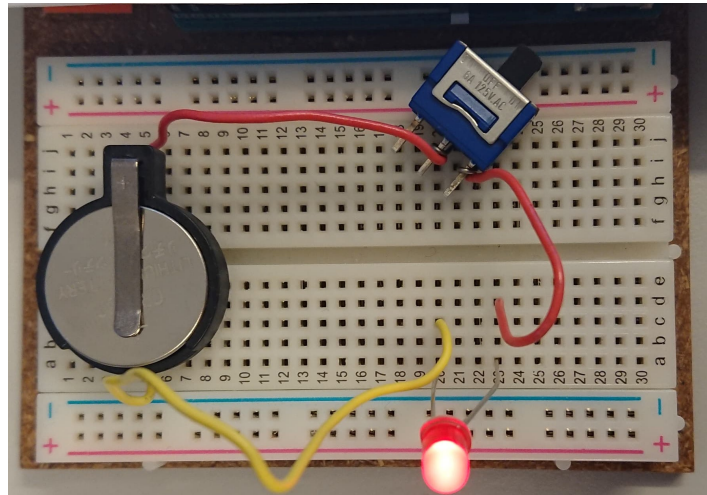


Figure 94: Adding a slide switch to the circuit

Component name	Where to find it	Approximately price	Quantity
HC-08 Serial Bluetooth Module	Smart prototyping [84]	50 NOK	2
Arduino Nano Every	Elfa Distrelec [44]	102 NOK	1
CR2032 button battery	Farnell [54]	7 NOK	2
Battery holder	Farnell [55]	48 NOK	1
LED lamp	Farnell [56]	5 NOK	1
Push button	AliExpress [7]	14 NOK	1
Slide switch button	Farnell [57]	34 NOK	1

Table 2: Component data

The total cost of the components needed for the controller is about 320 NOK or 38 USD without the wires, resistors, and rubber band. There is a possibility to buy the components in bigger quantities so that the price per part decreases. In addition, some costs might appear when soldering and 3D printing. Making the controller will increase the total cost of BADDY by 7-8%, which is about the same price as a tube with quality shuttlecocks.

6.2.3 Assembling the components

The case for the controller was developed with the aim of making it as small as possible, but at the same time being able to fit all the components inside. Due to this, it was not possible to make room for a breadboard inside the case and the components had to be soldered directly to each other using wires. During the soldering of the components, it was important to make sure that the wires were correctly connected. In addition, it was also important to solder the components so that they would fit inside the case while still being able to get the lid on. In section 6.1, the final prototype of the case along with the push button, switch button, and LED lamp is shown in Figure 80. These three components cannot be moved and became the foundation for how the other

components were to be placed. Since the push button is quite long, this was used as a reference point for the highest point of all the components during the soldering. By allowing this to be the highest point, it would still be possible to attach the lid and close it.

The remote controller is powered by two batteries placed in a battery holder, located inside the case. In order to make the controller sustainable and easy to use, it should be possible to replace these batteries when needed. This had to be taken into account during the soldering and placement of the components. To enable the user to update and further develop the controller, the micro-USB connector would have to be easily accessible, which also had to be taken into account when placing the component.

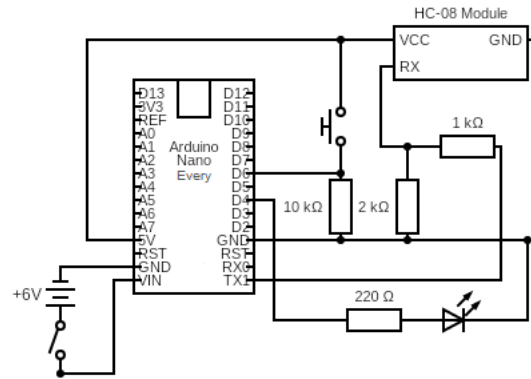


Figure 95: Final circuit of all the components inside the controller

Figure 95 shows the circuit diagram for the controller. As seen in this diagram, there is one smaller loop with the battery and switch button, and one more complex with the push button, led light, and HC-08 module. The smallest loop was soldered first and can be seen in Figure 96. To make the whole process easier, tiny wires were soldered to the different pins that were to be used on the Arduino Nano Every. Wires were also soldered to the legs of the two buttons as seen in the figure.

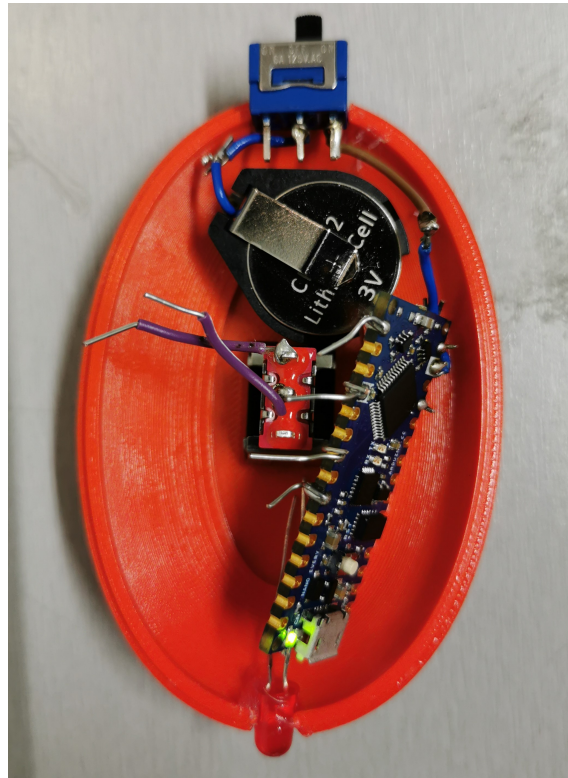


Figure 96: Soldered loop with battery and switch button placed inside the case

The next step was to solder the HC-08 module and LED lamp to the Arduino board. This was performed in several smaller steps. First, the HC-08 module was soldered to a small breadboard with wires as seen in Figure 97. This made it easier to access the needed pins on the module without risking touching the wrong pins. The LED lamp was then soldered to the D4 pin at the Arduino board through a resistor. When the other leg on the lamp was to be soldered, it detached from the lamp as seen in Figure 98. The entire LED had to be replaced, but the already connected wire between the Arduino and the LED had to be removed before this could happen.

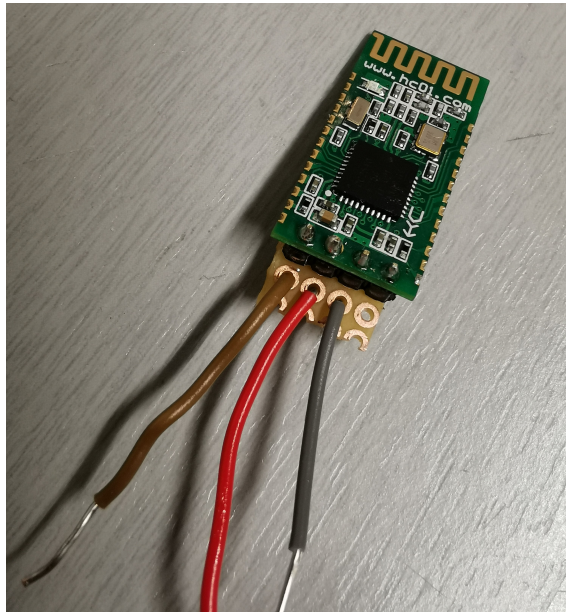


Figure 97: HC-08 module soldered to a small breadboard with wires

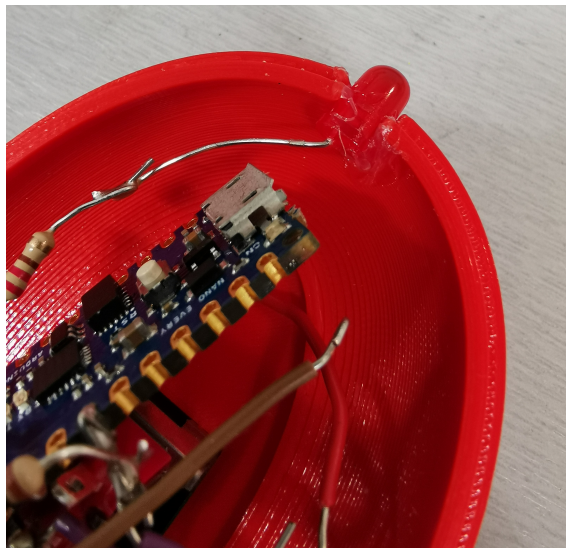


Figure 98: One of the legs on the LED lamp came loose

Furthermore, the HC-08 module was connected to both the Arduino board and the push button. It was initially intended that the switch button and the LED lamp should be glued to the case since these are the only components that have a fixed place in the case. But after soldering all the components together and placing them inside the case, it was discovered that it was not necessary to do so. All the components got a natural placement in the case and they fitted well together. It was also concluded that it would be easier to replace the battery and parts if it was possible to remove all the parts, except the push button. The finished soldered case can be seen in Figure 99.

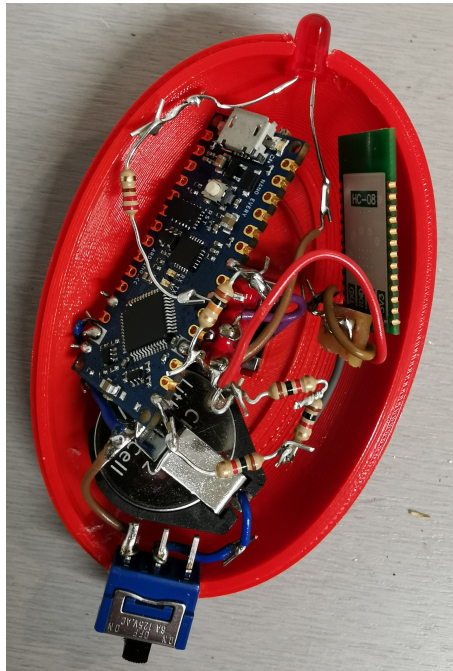


Figure 99: The inside of the final controller

As seen in Figure 99, there are many wired connections between the components, and several of these can cause errors if the wires come into contact with each other or other pins. Several methods can be used to avoid this from happening. One possibility could be to use electric tape around the exposed parts and wires. Another solution could be to use rubber or glue to create a film around the wires. The use of electric tape may be an easier solution in terms of feasibility, but glue or rubber may be a better solution in terms of durability.

6.3 Software development for the controller

The controller code was written with *Arduino IDE*, which can be read about in section 2.8. *Visual Studio Code* and *PlatformIO* could have been used for developing this code, but as the code was short and uncomplicated, *Arduino IDE* was sufficient. The code was started developed in the prestudy and can be read about in the prestudy report, section 6.3 [67]. The prestudy explains how AT commands were used to program the modules so that they would automatically connect to each other. The code for this is can be seen in Listing 26 in Appendix B. The Bluetooth module working as a slave was placed in the controller for battery-saving purposes. This lead to the code called *slave_code* in the prestudy being renamed to *controller_code* to make it more descriptive.

The two files needed to program the controller and their folder structure can be seen in Figure 100. As mentioned in section 2.8, an Arduino .ino file has to be placed in a folder with the same name. This makes it possible for the *Arduino IDE* to retrieve the file and use it.

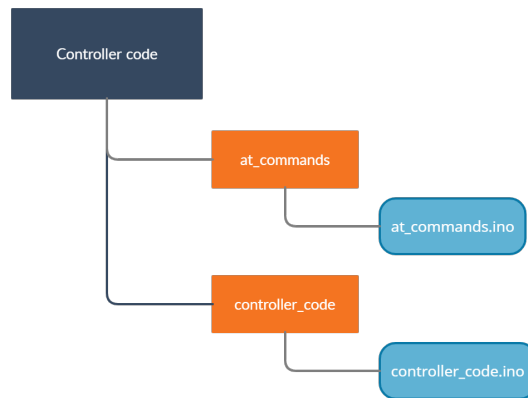


Figure 100: Folder structure for the Arduino files used for the controller

The controller code from the prestudy had to be changed due to new hardware components and to provide the controller with more functionality. The only modifications that had to be done to the code when changing from an Arduino Uno to an Arduino Nano (and later, Nano Every), was to redefine which pin number was used for the button. When the LED lamp was added later, the code had to configure the pin used as an output, and the logic for the LED was written in the loop-function. The LED pin was defined and configured as seen in Listing 18. The only function of the LED, as for now, was to tell if the controller was on or off. When the controller was provided with power and turned on, the LED lamp would light up, and when the controller was turned off, the LED would also turn off. This function was implemented by placing a command for lighting the LED, as seen in line 6 in Listing 18, as the first line in the loop-function.

```

1 #define led 4      // the number of the led pin
2 void setup(){
3     pinMode(led, OUTPUT); // Configure the led pin as an output
4 }
5 void loop(){
6     digitalWrite(led, HIGH); // Led turns on when the controller turns on
7 }
  
```

Listing 18: Code to control the led lamp

Different approaches were tested out in order to launch a shuttlecock. As stated in section 5.3, it was first attempted to send a JSON object from the controller to BADDY. A different approach was chosen instead of this, where only one character was sent to the Bluetooth module in BADDY when the button was pressed. This solution required minimal changes to the controller code and the code for sending this character can be seen in Listing 19.

```

1 Serial.print(BTserial.write('1')); // Send signal to BADDY
  
```

Listing 19: Sending a character to BADDY

The next step was to add more functionality to the controller and make it possible to send different signals to BADDY. Different clicks on the button would send different characters, where each character would trigger a predefined action in the receiving code. The controller was created with only one button, something that limits the amount of different, convenient clicks. Some of these clicks might be a short button press, a long button press, and a double press. By adding one more button to the controller, the combination of clicks could increase the number of actions. At the time of this report, a short press and a long press were implemented. The button press logic can

be seen from line 32 to 52 in Listing 20. Four variables had to be added, *buttonState* to store the last status of the push button, *longPressActive* to keep track of whether the button press has been treated as a long press *buttonTimer* tracks the time of a button push, and *longPressTime* to define how many milliseconds a long press has to last.

The logic for the button presses is written in the loop-section of the code. There are many possible solutions for writing the logic behind the long and short button presses. The implemented solution first checks if the current state of the button is pressed, if so, the timer start. When the button is released, the length of the button press is compared to *longPressTime*, and it checks whether a long button press has been triggered or not. If it was not, a short button press is activated. At the same time, it uses the *buttonState* to detect any changes since the last iteration. With this logic, a long button press is detected while the button is being pressed, and a short button press is detected when the button is released. To make BADDY react differently for the two button presses, the line from Listing 19 had to be modified and duplicated. When a short button press is perceived, it will send the character '1', and when a long button press is perceived, it will send the character '2'. Then the *readDataFromController()*-function in BADDY's code will handle the two characters differently and trigger two different actions. The code for receiving the data sent from the controller can be seen in section 5.3.

```

1  /* This code is written with inspiration from
2  *  Dejan Nedelkovski, www.HowToMechatronics.com
3  *  and xn1ch1,
4  *  https://www.instructables.com/Arduino-Dual-Function-Button-Long-PressShort-
      Press/
5  *
      == Controller Code ==
6  */
7  #include <SoftwareSerial.h>
8  int bluetoothTx = 1; // TX pin of HC-08 Bluetooth Module, Arduino D1
9  int bluetoothRx = 0; // RX pin of HC-08 Bluetooth Module, Arduino D0,
10 // Rx is not needed since communication is only going one way
11
12 //create an instance of a SoftwareSerial object
13 SoftwareSerial BTserial(bluetoothRx, bluetoothTx); //Rx,Tx
14 #define led 4          // the number of the led pin
15 #define button 6      // the number of the pushbutton pin
16
17 boolean buttonState = false; // variable for storing last status of the pushbutton
18 boolean longPressActive = false; // variable for long press
19 long buttonTimer = 0; // variable for time tracking
20 long longPressTime = 500; // Time of a long press i milliseconds
21
22 void setup(){
23   pinMode(button, INPUT); // Configure the button pin as an input
24   pinMode(led, OUTPUT);   // Configure the led pin as an output
25   Serial.begin(9600);     //Default communication rate of the Bluetooth module
26   BTserial.begin(9600);  // Begin The Bluetooth Module, default is 9600bps
27 }
28
29 void loop(){
30   digitalWrite(led, HIGH); // Led turns on when the controller turns on
31   if (digitalRead(button) == HIGH) { // The button is pressed
32     if (buttonState == false) { // Detect if buttonState has changed since the last
          iteration
33       buttonState = true; // A buttonpress is detected
34       buttonTimer = millis(); // Record the time of the buttonpress in milliseconds
35     }
36     // Check if current time subtracted from the time of the first buttonpress is
          longer than time for a long press, and if long press have been activated since
          last iteration
37     if ((millis() - buttonTimer > longPressTime) && (longPressActive == false)) {
38       longPressActive = true; // variable changed to true to stop repeated
          activation of long press
39       Serial.print(BTserial.write('2')); // Send long press signal to BADDY
40     }
41   } else { // Button is not pressed anymore
42     if (buttonState == true) { // The button has been pressed
43       if (longPressActive == true) { // Checking if a long press was activated
44         longPressActive = false; // Changed to false to allow new long press next
          iteration
45       } else { // Long press was not activated
46         Serial.print(BTserial.write('1')); // Send short press signal to BADDY
47       }
48       buttonState = false; //Changed to allow new press in next iteration
49     }
50   }
51   delay(50);
52 }

```

Listing 20: The complete code for the controller

To upload the code to the Arduino Nano Every board in the controller, the board has to be installed in the *Arduino IDE's* board manager. The *Arduino megaAVR Boards* package was installed, and the board was chosen from the "Tools" menu in the top toolbar, along with the USB port the board was connected to. This is illustrated in Figure 101. The complete code can be seen in Listing 27 in Appendix B.

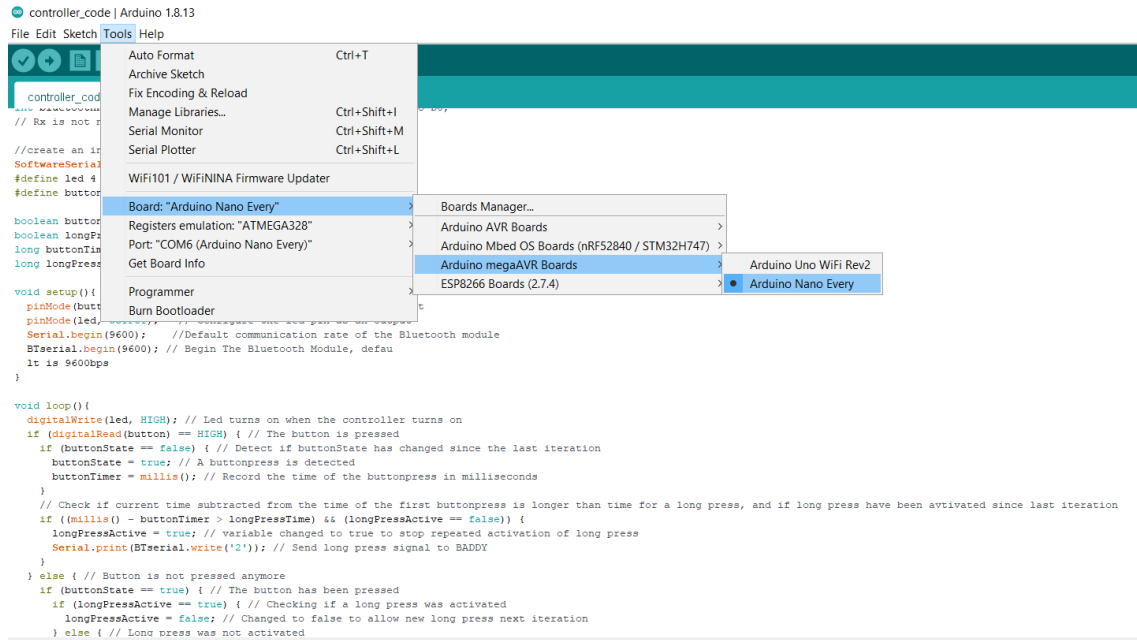


Figure 101: Choosing the correct board in the *Arduino IDE*

7 Discussion and conclusion

This study has consisted of further development and improvement of an already existing badminton robot, where previous work has been conducted by others. This meant that a large part of this study consisted of getting acquainted with work that had already been done to the BADDY project. The research was complicated due to lack of documentation. If the documentation and information for the BADDY project had been sufficient, more time could have been used for the further development of the robot. Instead, missing documentation had to be written. Due to this, a big part of this study has been to provide documentation for the BADDY project. The documentation has been made, based on the experience of building BADDY, a code review, information gathered from different websites, and through interviews with the founder.

Poorly documented projects could result in less attention from others, as it would take longer to understand what previously has been done. The lack of documentation could also make it difficult to ensure the quality of the upgrades and developments. Especially if these have been performed by others than the original developers. On the other side, too much information could also be negative, as it might be overwhelming and time-consuming to find the specific information needed. Finding the correct balance could be hard, but it is an important aspect of today's development. Most projects that exist today are collaborative projects or projects that can be further developed by others. This increases the necessity and importance of good documentation, now more than ever. The world is constantly changing and new ideas are often based on already existing ideas and projects, so documentation is essential for allowing others to continue the development of products without the need for excessive work.

7.1 Discussion

7.1.1 The remote controller

The remote controller has been developed within some limitations mentioned in section 1.4. These limitations could have affected the final result of the wired circuit for the controller. Both in terms of the selected components, resistors used, which pins on the boards were used, the final price for the entire controller, and more. By using other components, other functionalities could have been possible to achieve, and the setup and implementation would have been done differently. This could result in a more optimized prototype, but was not prioritized due to the limitations of the project. The optimization of the remote controller, in terms of decreasing the price, choosing better components and pins for the wiring, has been left for future work along with the possibility of adding more functionality to the remote controller.

The process of creating a 3D drawing, printing, redoing the drawing, and printing again took longer than expected. There were several reasons for this, where COVID-19 has been a contributing factor to, among other things, the lack of access to the lab and working 3D printers. Some 3D printers were provided, but these were available for multiple other students. Meaning that many of the printers were in use most of the time, in addition to the fact that several of these did not work and necessary material was missing. After some time, the knowledge of how to calibrate the printers and changing filament was gathered. This helped a bit, but the printing lab was still not up and running most of the time. Another reason for the delayed progress was due to late access to the mechatronics lab. This lab had, among other things, possibilities for soldering, available components, different tools, and a workbench. It would have been an advantage to have had access

to all of this earlier than what was provided.

The Arduino Nano Every, along with an HC-08 Bluetooth module, was chosen as the final components to use in the controller. The Nano Every is not embedded with BLE, so a Bluetooth module was necessary to make the controller communicate with BADDY. When using BLE as a means of communication, several changes had to be made to both BADDY's PCB and BADDY's code. A possible way to avoid this could have been to replace the Arduino and Bluetooth module in the remote controller, with a WiFi microchip. There exist small ESP8266 boards that could be placed in the controller. By doing so, it would only be necessary to change the code for the controller, not BADDY's code, as BADDY already had all the necessary WiFi configurations and an interface for it. By using WiFi in the controller, it would make it easier and require less work to connect the controller to BADDY. A small ESP8266 module would most likely simplify the wired circuit for the controller as well as make it smaller than the chosen solution, as the HC-08 module, along with some of the wiring, would be removed. On the other side, the use of WiFi in the remote controller might not be needed or wanted. WiFi has a higher power consumption than BLE, something that would require a more frequent replacement of the batteries. WiFi is designed for using high bandwidth and transfer big amounts of data constantly, which is more than what is required for the controller.

The use of BLE for communication between BADDY and the controller would therefore be beneficial. A possible improvement could be to replace the Arduino Nano Every and HC-08 module with an Arduino Nano 33 BLE. This is an upgraded version of the Arduino Nano and comes with BLE, and the use of this board would result in a smaller and easier wired circuit for the controller. The Nano 33 BLE includes many other libraries and more functionality than the Nano and Nano Every. By replacing the Nano Every and HC-08 module with a Nano 33 BLE, most parts of the controller code would have to be changed. As the Nano 33 BLE was released in 2019 [89], it was, at the time of this report, relatively new to the market, and few relevant sample codes existed. This would have made the development and coding of the remote controller harder and require more time than using an Arduino Nano Every and an HC-08 module.

An Arduino Nano Every was used for the final prototype for the remote controller. This board was chosen due to its small size and since it was easy to get a hold of without the pins soldered to the board. A disadvantage of the Nano Every is that it requires an input voltage of 6 V. In order to provide the board with enough power to function, two coin batteries had to be used. A disadvantage of this was that the extra battery required more space inside the case. To avoid this, the Arduino Nano Every could be replaced with an Adafruit Pro Trinket - 3V, which only requires an input voltage of 3 V. Replacing the Arduino Nano Every with an Adafruit would result in a smaller and cheaper solution as the Adafruit is smaller in size and cheaper than the Nano Every. Due to the low input voltage for this board, only one coin battery would be needed, which would also reduce the total cost of the controller. The Adafruit Pro Trinket "works with 99% of existing Arduino sketches" [4], meaning that the code written for the Arduino Nano Every would most likely work with the Adafruit, and no changes would have to be made to the code.

The remote controller could be improved by either implementing an Arduino Nano 33 BLE or an Adafruit Pro Trinket. By using the Nano 33 BLE, the Bluetooth module could be removed, but it would require a more comprehensive change to the code. By switching to the Adafruit, which is not embedded with Bluetooth, the HC-08 module would be necessary. However, the implementation of an Adafruit would only require one coin battery, and it would most likely not be necessary to change the code.

The 3D printed case for the controller was featured with a simple closing mechanism that worked well when the case was empty. After all the components were soldered together and placed inside the case, the closing mechanism was weakened and the risk of it opening up, increased. This could be improved by adding some small adjustments to the case, and there are many possible ways to do this. Adding screw holes could be one option, but as mentioned before, this is hard to 3D print. Another possibility could be to implement magnets. The design would have to be adjusted in order to make room for the magnets. Small and strong magnets are already used to fasten the tube on top of BADDY and could be a more optimal solution. This would increase the durability and quality of the case and closing mechanism.

The remote controller has been implemented sideways with the phone connection. Meaning that it would be possible to switch back and forth between the controller and a phone without breaking the connection between BADDY and any of these. The controller and a phone could also be used independently of each other, one at a time. This makes the implementation of the remote controller flexible, and it would be up to the user whether the controller should be used or not after the implementation. An advantage of the sideways implementation would be that it makes it possible for both the player and the coach to control BADDY at the same time. The player could use the remote controller while the coach could set up a sequence from the phone.

Holding on to the remote controller could lead to a more natural way of playing compared to using a big phone. The remote controller is fastened to the player's hand with a strap, allowing the player to use the non-racket arm to aim at the shuttle with an open hand. This would be beneficial for the players natural movement and increase the profits of the training session.

There are many possible ways to program the controller and different functionality could be implemented. One possibility could be to combine the controller with the phone application. The player would be able to create a sequence in the application prior to the training, before putting the phone away. The controller could then be used for either launching the next shot in the sequence, or a button press could initiate the entire sequence. This could be done by modifying the *ReadSequence()*-function (the triggering function in BADDY's code) to check for signals from the controller.

As for now, the pairing of the two Bluetooth modules has been done as described in section 5.4 in the prestudy report [67]. The Bluetooth address for the slave module was saved in the master module, so the master module automatically connects to this address when turned on. This might not be an optimal solution in terms of connecting other Bluetooth modules to BADDY, as the addresses of these devices would have to go through the same process. The final prototype documented in this report does not include any functionality for pairing new devices to either BADDY or the controller other than the one described above. This is something that could be implemented in the future to improve the remote controller and BADDY even more.

7.1.2 The BADDY robot

The first version of BADDY, BADDY V1, was embedded with Bluetooth for communication purposes, but this was replaced with WiFi communication for the newer versions. Because of this, BADDY's PCB was switched to an ESP8266 WiFi microchip that doesn't support Bluetooth. By creating a remote controller with a Bluetooth module and making it connect with BADDY, an external HC-08 Bluetooth module had to be soldered onto BADDY's PCB. A possible upgrade could be to replace the ESP8266 in BADDY, with a board that supports both Bluetooth and WiFi, such as the ESP32-C3 Series Modules [52]. This upgrade was not tested, as the possibility of changing

BADDY's PCB was discovered late in the project, with little time left. The implementation of an ESP32-C3 board would also require extensive changes in BADDY's code, something that would be time-consuming.

It requires both time and effort when joining ongoing projects in terms of familiarizing oneself with existing work. As the BADDY project is poorly documented, this has been particularly challenging. After building and testing BADDY, it was concluded that the overall quality of the robot and project could be improved. Small adjustments to the robot could lead to great improvements considering quality and expected lifetime of BADDY. During the short time of this project, it has been observed that multiple screws have loosened, some parts have been detached and other parts have broken. Some aspects of the BADDY robot didn't meet the expectations prior to the purchase of the DIY kit. An example of this is that the fastening mechanism between the tripod and BADDY is unstable and it is hard to fine-tune the angle of the robot. When BADDY was placed on the tripod, it leaned to the left side, most likely because of the placement of components inside creating an uneven load distribution. Some of the problems that occurred could be solved by making small adjustments to increase the quality, but they might also occur due to poor mounting of the robot.

The BADDY robot would benefit from being a bit more robust and stable considering both usability and sustainability. Small changes and adjustments could improve the durability of the robot. An example of this could be to use *Loctite* glue on the screws during the mounting process. The tripod could be replaced by one that is more stable and has a better fastening mechanism. By switching to a tripod of higher quality, the risk of breaking parts and even breaking BADDY would decrease. The fact that small adjustments could make some big differences in performance, indicates that it is important to be thorough when mounting BADDY. In order to make BADDY work perfectly, it would have to be mounted correctly with high accuracy. The only instructions for how to do the mounting are provided in the YouTube videos. Even when following the videos, it could be hard to assemble every part correctly. A written guide, such as the one presented in section 4, could be beneficial and make it easier to mount BADDY.

BADDY was tested during a training session for young players, arranged by *Trondheim Badmintonklubb*. BADDY was popular and attracted much attention. Based on the feedback given, BADDY had a positive effect on the session for both the coach and the players. BADDY also worked as a motivating factor and contributed to both joy and efficiency during the session. The coach could more easily walk around the court and observe the players from new angles, as the coach didn't need to launch the shuttles himself. The feedback from both the coach and the players was positive and valuable in terms of the effect a BADDY robot could have on a training session. One of the downsides that came out of this test was that it was difficult for children to reach the top of the tube and fill it with shuttles. As a result of trying to do so, the tube came loose and fell to the ground. It broke, but was glued back together. Another feedback that was given, was that BADDY should give a sign or notification prior to launching a shuttle. This would give the player a heads up, and get ready for the next shuttle. This would also make training with BADDY more similar to a real game.

A problem that occurred during this test, was the curiosity of the people around. BADDY attracted attention, and many people wanted to take a closer look at BADDY. They placed themselves in front of BADDY, wanting to look inside it, even when it was turned on and launched shuttles. As of today, the only security aspect of BADDY consists of a written warning on the BADDY website [19]. This is not sufficient and considering the amount of attention BADDY received during the training session, some safety mechanism should be implemented. This could be solved by either

adding a mechanical feature, changing the software, or a combination of these, and make it safer to be around the robot.⁸

During this test, both the controller and a smartphone were used to control BADDY. As the controller has less functionality than the application, a smartphone was therefore used to test the different shots and to create longer sequences. The coach found it difficult to set the correct timing between the shots in a sequence. This could be simplified by making the controller work with the application. A sequence could be created in the application and a button press on the controller could launch the next shot in the sequence. In addition to this, it was also mentioned that it should be possible to save sequences in the application so that they could be reused at a later time. The feedback given after the training session indicated that the robot and the controller could be a valuable addition to training for both new and experienced players.

BADDY is a stationary and static robot considering that it has to be manually adjusted to change its height, incline, etc. BADDY can be placed both on the ground and on a tripod, and it is possible to increase or decrease the incline of BADDY when it is placed on the tripod. These adjustments allow BADDY to launch different and realistic shots. However, it would be both time-consuming and cumbersome to constantly have to change these factors in order to get the wanted shots from BADDY. In addition to interrupting the training session, it was difficult to get the wanted angle and height by manually having to change these. It could be a possibility to add some functionality that allows BADDY to move, either sideways, up and down, rotating or change the incline, mechanically by the use of e.g. a remote controller. This would not only make it easier to get the wanted angle and height, but it could also increase the shooting range and quality of the different shots, resulting in a smoother training session.

There are many possible ways to implement such functionality, both in terms of adding an external part that could be attached to BADDY or by fastening a mechanism directly onto BADDY. Both of these solutions would increase the price of BADDY. By adding an external part, BADDY would be unchanged and this part could be purchased or created independently of BADDY. An example of this could be to create some sort of a cradle that BADDY could be placed in, and control the movement of this cradle so that BADDY's height, incline, and rotation could be changed remotely. Such a solution would result in the need for an external power supply and a way to control the added part. This could cause some problems in regards to the communication between BADDY, the external part, and the controller. BADDY and the new part would be two separate robots that would have to be controlled independently of each other. By adding such a mechanism that has to be controlled separately of BADDY, it would result in poorer usability and make BADDY more complicated to use. However, by implementing an external part, it would be up to the user to decide if and how often this part should be used.

Another way to control the adjustment of these factors could be to mount some mechanism directly onto BADDY. An example of this could be to attach gears on each side of the robot, as well as some functionality to use these gears. This would make it possible to adjust the angle of BADDY as well as rotate it in order for the shots to cover larger parts of the court. By implementing such a mechanism directly onto BADDY, this mechanism could be wired to BADDY's own battery and there would be no need for an external power supply. A disadvantage of implementing a mechanism like this could be that it would have to be physically screwed onto BADDY. Meaning that you have to bring it with BADDY, regardless of whether it should be used or not. This would result in an increased weight of BADDY itself and could make the mounting of the robot more complex and time-consuming.

⁸Different security aspects considering BADDY is written about in section 4.1.1 in the prestudy report [67]

7.2 Conclusion

This study has consisted of mounting and testing a BADDY robot, in addition to the development of a working remote controller that could improve BADDY. Several problems and areas of improvement for the current design of the robot were found and discussed in a prestudy report [67]. One of the improvements presented in this prestudy has been further developed and the implementation of this is documented in this report.

The first part of this study involved the mounting of BADDY, as well as discovering possible pitfalls one may encounter during the process. The documentation of this was intended as an instruction for others to follow when mounting their robot. The finished mounted BADDY was tested based on functioning, performance, accuracy, precision, and range of shots. During these tests, multiple problems were discovered and various attempts were made in order to correct these problems.

The BADDY website consists mostly of information about the latest version of the BADDY robot. It was difficult to find and retrieve information about the previous versions and the BADDY project itself. Due to this, missing documentation for the robot has been written and information from the BADDY project gathered. As a result of this, a product review and assessment have been made of the robot itself and the source code.

The second part of this study documents the process of creating an alternative way to control BADDY. As of today, BADDY is controlled by the use of a smartphone application. The use of a smartphone may not be optimal due to its big size, but also due to the price and complexity of such a phone. A working remote controller has been developed using 3D printing, soldering, Arduino coding, and Bluetooth Low Energy (BLE). In order to integrate the controller with the BADDY robot, BADDY's hardware and software had to be modified. An HC-08 Bluetooth module was soldered to BADDY's PCB and additional code had to be written and implemented. The controller could either be used as a replacement for the smartphone or together with the smartphone to improve the functionality of controlling BADDY. In order to modify BADDY, the embedded source code had to be examined and reviewed. As a result of this review, possible ways to improve the already existing code have been presented.

The remote controller is not optimized in terms of price, performance or functionality, due to the limitations defining the study. It would be possible to further develop the remote controller, both in terms of design, software, and hardware, and adding more functionality to it. The controller has been tested considering range, usability, and ease of use. It is only a prototype and could be improved in many possible ways, one of which could be to implement a more durable closing mechanism.

The work that has been done throughout the course of this study, will be valuable to those who want to either build BADDY, help develop the robot further, or just get acquainted with the project and previous work. The remote controller illustrates how new aspects could be added to BADDY, and this could inspire others to further develop their BADDY.

This study has been carried out with the purpose of making it easier for others to familiarize themselves with the BADDY project. The work that has been done, will make it easier to mount a BADDY robot and continue the development of the robot. The authors of this report recommend that the presented solution is implemented in the BADDY project, so others can benefit from the controller and the documentation. This report has been sent to the founder of BADDY and published to the community for others to use freely.

7.3 Afterword

BADDY is a robot designed and created for training purposes, but it can also be used for other purposes. For example having some fun and good times, as can be seen below.



Figure 102: BADDY used for fun together with fellow graduation students

Bibliography

- [1] 3DPrinting.com. An Overview Of The Best 3D Printing Software Tools, . URL <https://3dprinting.com/software/#3D-MODELING-SOFTWARE>. Accessed: 06.04.2021.
- [2] 3DPrinting.com. What is 3D Printing?, . URL <https://3dprinting.com/what-is-3d-printing/>. Accessed: 06.04.2021.
- [3] Mathieu Abet. WHAT IS BLUETOOTH LOW ENERGY? URL <https://elainnovation.com/what-is-bluetooth-low-energy/?lang=en>. Accessed: 07.12.2020.
- [4] Adafruit. Adafruit Pro Trinket - 3V 12MHz, . URL <https://www.adafruit.com/product/2010>. Accessed: 11.05.2021.
- [5] Adafruit. Adafruit Pro Trinket - 3V 12MHz, . URL https://www.adafruit.com/product/2010?gclid=CjwKCAjwy42FBhB2EiwAJY0yQkutVD-MIPKUN2nGtcyMAw-DT46Njwpi9s50AyA2fb536xqoG6TzExoCfKEQAvD_BwE. Accessed: 18.05.2021.
- [6] Alibaba.com. Remote control portable badminton robot shuttlecock robot. URL https://www.alibaba.com/product-detail/Remote-control-portable-badminton-robot-shuttlecock_60791626592.html?spm=a2700.7724857.normalList.72.22952371FzR801. Accessed: 16.11.2020.
- [7] AliExpress. SH P8701-F22A Square Snap-in With Frame 3Pin Momentary SPDT Mini Push Button Switch. URL <https://www.aliexpress.com/i/32816420226.html>. Accessed: 05.05.2021.
- [8] Gayle Alleyne. Experimental service law from march 2018. URL <https://bwfbadminton.com/news-single/2017/11/29/experimental-service-law-from-march-2018>. Accessed: 10.05.2021.
- [9] amazon. Solder Sucker - desoldering pump. URL <https://www.amazon.com/WEmake-WM-SP4-Solder-Sucker-desoldering/dp/B0002KRAAG>. Accessed: 28.04.2021.
- [10] Arduino. What is Arduino?, . URL <https://www.arduino.cc/en/Guide/Introduction>. Accessed: 04.03.2021.
- [11] Arduino. Arduino nano, . URL <https://store.arduino.cc/arduino-nano>. Accessed: 17.02.2021.
- [12] Arduino. const, . URL <https://www.arduino.cc/reference/en/language/variables/variable-scope-qualifiers/const/>. Accessed: 03.05.2021.
- [13] Arduino. define, . URL <https://www.arduino.cc/reference/en/language/structure/further-syntax/define/>. Accessed: 03.05.2021.
- [14] Arduino. Digital Pins, . URL <https://www.arduino.cc/en/Tutorial/Foundations/DigitalPins>. Accessed: 25.02.2021.
- [15] Arduino. Arduino Nano Every, . URL <https://store.arduino.cc/arduino-nano-every>. Accessed: 12.04.2021.
- [16] Arduino. InputPullupSerial, . URL <https://www.arduino.cc/en/Tutorial/BuiltInExamples/InputPullupSerialf>. Accessed: 25.02.2021.

-
- [17] Prusa Research a.s. Original Prusa i3 MK3S+ 3D printer. URL <https://shop.prusa3d.com/en/3d-printers/181-original-prusa-i3-mk3s-3d-printer.html>. Accessed: 07.04.2021.
- [18] BADDY. Baddy shop, . URL <https://www.baddy.space/badminton-feeding-machine-diy-robot>. Accessed: 03.02.2021.
- [19] BADDY. Baddy website, . URL <https://www.baddy.space>. Accessed: 15.11.2020.
- [20] BADDY. Frequently Asked Questions, . URL <https://www.baddy.space/baddy-badminton-feeding-machine-diy>. Accessed: 28.10.2020.
- [21] BADDY. Pull requests, . URL <https://github.com/baddy-lab/baddy-makers-edition/pulls>. Accessed: 18.05.2021.
- [22] BADDY. BADDY app V3 - how to connect, . URL <https://www.youtube.com/watch?v=32rZ1G6EdBk>. Accessed: 03.05.2021.
- [23] BADDY. BADDY V3 - WiFi connection explained, . URL <https://www.youtube.com/watch?v=9iyRco0qFvk&t=127s>. Accessed: 03.05.2021.
- [24] BADDY. How to use BADDY, . URL <https://www.youtube.com/watch?v=f6MntnNkGPY>. Accessed: 03.05.2021.
- [25] BADDY. BADDY V2 - Maker kit, . URL <https://www.baddy.space/product-page/baddy-v2-maker-kit>. Accessed: 17.11.2020.
- [26] BADDY. Specifications, . URL <https://www.baddy.space/baddy-badminton-diy-robot-specifica>. Accessed: 28.10.2020.
- [27] Baddy. Home [YouTube Channel], . URL <https://www.youtube.com/channel/UCYEWljJYV7ifch94jTnyJbA/featured>. Accessed: 17.11.2020.
- [28] Baddy. BADDY V2 Tutorial - Install BADDY Firmware [Video]. YouTube, . URL <https://www.youtube.com/watch?v=81ajkughvqE&t=755s>. Accessed: 08.04.2021.
- [29] BADDY. baddy-lab/baddy-makers-edition, 2020. URL <https://github.com/baddy-lab/baddy-makers-edition>. Accessed: 27.01.2021.
- [30] BADDY. baddy-makers-edition/embedded source code/baddy v3/main.cpp, 2020. URL <https://github.com/baddy-lab/baddy-makers-edition/blob/master/embedded%20source%20code/BADDY%20V3/main.cpp>. Accessed: 09.04.2021.
- [31] BADDY. baddy-lab/baddy-makers-edition, 2020. URL <https://github.com/baddy-lab/baddy-makers-edition/tree/master/embedded%20source%20code/BADDY%20V3>. Accessed: 09.03.2021.
- [32] Master Badminton. Badminton Body Balance. URL <https://www.masterbadminton.com/badminton-body-balance.html>. Accessed: 29.09.2020.
- [33] Brian Benchoff. THE CURRENT STATE OF ESP8266 DEVELOPMENT, . URL <https://hackaday.com/2014/09/06/the-current-state-of-esp8266-development/>. Accessed: 14.04.2021.
- [34] Brian Benchoff. AN SDK FOR THE ESP8266 WIFI CHIP, . URL <https://hackaday.com/2014/10/25/an-sdk-for-the-esp8266-wifi-chip/>. Accessed: 14.04.2021.
-

-
- [35] Jim Blom. Using the BlueSMiRF. URL <https://learn.sparkfun.com/tutorials/using-the-bluesmirf/example-code-using-command-mode>. Accessed: 04.12.2020.
- [36] A. M. Cabrero. BADMINTON TECHNIQUES. *KSU Kent State University*, pages 1–5, 2017.
- [37] Scott Campbell. BASICS OF UART COMMUNICATION. URL <https://www.circuitbasics.com/basics-uart-communication/>. Accessed: 14.04.2021.
- [38] Dibya Chakravorty. 3D Printing Support Structures – The Ultimate Guide. URL <https://all3dp.com/1/3d-printing-support-structures/>. Accessed: 17.11.2020.
- [39] Olympic Channel. Badminton: How to play, rules, and all you need to know. URL <https://www.olympicchannel.com/en/stories/features/detail/badminton-guide-how-to-play-rules-olympic-history/>. Accessed: 15.11.2020.
- [40] Circuitspecialists. ESP Microcontroller Quick Start Guide. URL <https://www.circuitspecialists.com/blog/esp-microcontroller-quick-start-guide/>. Accessed: 25.02.2021.
- [41] Electronics club. Switches. URL <https://electronicsclub.info/switches.htm>. Accessed: 05.05.2021.
- [42] Codegrip. Everything you need to know about Code Smells. URL <https://www.codegrip.tech/productivity/everything-you-need-to-know-about-code-smells/>. Accessed: 18.05.2021.
- [43] Benne de Bakker. Arduino Nano Board Guide. URL <https://www.makerguides.com/arduino-nano/#arduino-nano-every>. Accessed: 28.05.2021.
- [44] Elfa Distrelec. ABX00028 - Arduino Nano Every. URL <https://www.elfadistrelec.no/no/arduino-nano-every-arduino-abx00028/p/30150884?q=arduino&pos=8&origPos=59&origPageSize=50>. Accessed: 23.04.2021.
- [45] Chris Eason. Shuttlecock. URL <https://www.flickr.com/photos/mister-e/3294913646/in/photolist-62ai2Y-EZB7S-EZC3B-EZCVe-DwfP8b-EZDon-EZAB5-EZAA4-EZCUq-9YVWkr-pXd7RR-5FuYzV-uK7tG-pWZhbQ-jhYkUJ-2j5DPY1-4rabgQ-9PokFv-6aNPCP-6a4uES-7RjhBZ-5hxmMU-adCdSw-2i9sA3c-7moYP9-3b6B3R-CEWFjj-7v9kzX-4jQwu-4LLBEV-ewKB3E-nknj4M-9PG6eu-8hziEe-TmXnq8-wrysRr-5AxAzj-4hrTxP-8EGtrr-5xGCCf-RrH6ZM-arSBRz-DAwccM-rWeoY-jGFVK-8kwsis-22aFqmQ-7i1eWS-9i3Y3-7AQSfq>. Accessed: 15.11.2020.
- [46] Elektorstore. NodeMCU microcontroller board with ESP8266 and Lua. URL <https://www.elektor.com/nodemcu-microcontroller-board-with-esp8266-and-lua>. Accessed: 26.04.2021.
- [47] Future Electronics. Esp32 development board (wifi - bluetooth), . URL https://store.fut-electronics.com/products/esp-32?_pos=1&_sid=fb36146fa&_ss=r. Accessed: 05.02.2021.
- [48] Future Electronics. Nodemcu (esp8266 wifi programming development kit), . URL <https://store.fut-electronics.com/products/nodemcu-esp8266-programming-and-development-kit>. Accessed: 05.02.2021.
- [49] Ellisys. Ellisys Bluetooth Video 1: Intro to Bluetooth Low Energy [Video]. YouTube. URL <https://www.youtube.com/watch?v=eZGixQzBo7Y>. Accessed: 07.12.2020.
-

-
- [50] Espressif. Esp8266, . URL <https://www.espressif.com/en/products/socs/esp8266>. Accessed: 05.02.2021.
- [51] Espressif. ESP8266 Hardware Design Guidelines, . URL https://www.espressif.com/sites/default/files/documentation/esp8266_hardware_design_guidelines_en.pdf. Accessed: 25.02.2021.
- [52] Espressif. Wifi and bluetooth modules, . URL <https://www.espressif.com/en/products/modules>. Accessed: 05.02.2021.
- [53] Facebook. BADDY community Group. (n.d.). *Home* [Facebook group]. URL <https://www.facebook.com/groups/519469218529907>. Accessed: 25.02.2021.
- [54] Farnell. Multicomp CR2032, . URL <https://no.farnell.com/multicomp/cr2032/cell-lithium-cr2032-210mah-3v/dp/2065171?ost=2065171>. Accessed: 30.11.2020.
- [55] Farnell. Battery Holder, Coin Cell - 20mm x 2, PCB, . URL <https://no.farnell.com/keystone/1062/battery-holder-2-cell-20mm/dp/1282703?st=doble%20button%20battery%20holder>. Accessed: 27.04.2021.
- [56] Farnell. LED, Red, Through Hole, T-1 3/4 (5mm), 10 mA, 1.9 V, 626 nm, . URL <https://no.farnell.com/broadcom-limited/hlmp-3301/led-5mm-red-5-4mcd-626nm/dp/1003211?st=led>. Accessed: 27.04.2021.
- [57] Farnell. Slide Switch, . URL <https://no.farnell.com/nidec-copal/8ss1022-z/slide-switch-spdt-6a-125vac-th/dp/2854816?st=slide%20switch>. Accessed: 12.10.2020.
- [58] Gcc gnu. Header files. URL <https://gcc.gnu.org/onlinedocs/cpp/Header-Files.html>. Accessed: 14.05.2021.
- [59] Ivan Grokhotkov. Filesystem. URL <https://arduino-esp8266.readthedocs.io/en/latest/filesystem.html#spiffs-deprecation-warning>. Accessed: 11.03.2021.
- [60] Tony Hoffman. The Best 3D Printers for 2021. URL <https://www.pcmag.com/picks/the-best-3d-printers>. Accessed: 07.04.2021.
- [61] David Hurley. 12 challenges for open source projects. URL <https://opensource.com/life/14/6/12-challenges-open-source-projects>. Accessed: 18.05.2021.
- [62] Alex in LearnCpp. Boolean values. URL <https://www.learncpp.com/cpp-tutorial/boolean-values/>. Accessed: 20.05.2021.
- [63] Hugo Janton. Improvement of baddy the shuttlecock launcher. 2019.
- [64] Steven John. 'What is Bluetooth?': A beginner's guide to the wireless technology. URL <https://www.businessinsider.com/what-is-bluetooth?r=US&IR=T>. Accessed: 06.12.2020.
- [65] Json. Introducing JSON. URL <https://www.json.org/json-en.html>. Accessed: 17.03.2021.
- [66] Insook Kim. Teaching Badminton through Play Practice in Physical Education. *Journal of physical education, recreation dance*, 88(8):7–14, 2017. ISSN 0730-3084. doi: 10.1080/07303084.2017.1356768.
- [67] A. Kristiansen and H. Rishovd. Improvements of the shuttlecock launcher robot BADDY. 2020.
-

-
- [68] David Kushner. The Making of Arduino. URL <https://spectrum.ieee.org/geek-life/hands-on/the-making-of-arduino>. Accessed: 02.06.2021.
- [69] PlatformIO Labs. Frameworks, . URL <https://platformio.org/frameworks>. Accessed: 04.03.2021.
- [70] PlatformIO Labs. Platformio ide, . URL <https://platformio.org/platformio-ide>. Accessed: 27.01.2021.
- [71] MAKERDEMY. What is BLE? (2020) — Bluetooth Low Energy — Learn Technology in 5 Minutes [Video]. YouTube. URL <https://www.youtube.com/watch?v=NSkIHdV6NoY>. Accessed: 07.12.2020.
- [72] Microsoft. Visual studio code. URL <https://code.visualstudio.com>. Accessed: 27.01.2021.
- [73] Mark Nadolny. Shuttlecock and balls: The fastest moving objects in sport. URL <https://olympic.ca/2014/09/11/shuttlecock-and-balls-the-fastest-moving-objects-in-sport/>. Accessed: 07.05.2021.
- [74] Newswire. BADDY - Open Source Badminton Robot! URL <https://www.newswire.com/news/baddy-open-source-badminton-robot>. Accessed: 08.04.2021.
- [75] NTNU. TPK4960 - Robotics and Automation, Master's Thesis. URL <https://www.ntnu.edu/studies/courses/TPK4960#tab=omEmnet>. Accessed: 07.05.2021.
- [76] OMRON. Basic Switch: NO, NC and COM Contact Terminal. URL https://www.omron.com.au/service_support/FAQ/FAQ03206/index.asp. Accessed: 05.05.2021.
- [77] Lauree Padgett. Keeping promises and all that CRAAP. *Information today*, 34(4):19, 2017. ISSN 8755-6286. Accessed: 14.11.2020.
- [78] Garrett Parker. The Five Most Expensive 3D Printers on the Market Today. URL <https://moneyinc.com/the-five-most-expensive-3d-printers-on-the-market-today/>. Accessed: 07.04.2021.
- [79] Cork Sports Partnership. URL https://www.corksports.ie/index.cfm/page/schools-badminton?fbclid=IwAR0movOQBKAFZEK6TtNBY1tSkztF4_OWud02-7dw1VffXewgk-ZaxDBngSw. Accessed: 15.11.2020.
- [80] Naveen Peter. Everything you wanted to know about a badminton court. URL <https://www.olympicchannel.com/en/stories/features/detail/badminton-court-size-dimension-measurement-length-width-net-height-service-line/>. Accessed: 16.03.2021.
- [81] Michael Phomsoupha and Guillaume Laffaye. The Science of Badminton: Game Characteristics, Anthropometry, Physiology, Visual Fitness and Biomechanics. *Sports Medicine*, 45(4):473-495, 2015. ISSN 1179-2035. doi: 10.1007/s40279-014-0287-2. URL <https://doi.org/10.1007/s40279-014-0287-2>. Accessed: 15.11.2020.
- [82] professorcad. HC-08 BLUETOOTH UART COMMUNICATION MODULE V3.1 USER MANUAL . URL <https://www.professorcad.co.uk/Bluetooth/HC-08A%20version%20english%20datasheet.pdf>. Accessed: 12.10.2020.
- [83] BADDY project. Baddy discord server, baddy project. URL <https://discord.com/invite/YH3QC2V>. Accessed: 25.02.2021.
-

-
- [84] Smart Prototyping. HC-08 SERIAL BLUETOOTH MODULE. URL <https://www.smart-prototyping.com/HC-08-4-0-BLE-Serial-Bluetooth-Module-CC2540>. Accessed: 30.11.2020.
- [85] V. Ryan. BUTTON BATTERIES /COIN CELLS AND SIMPLE CIRCUITS. URL https://technologystudent.com/elec_flsh/button1.html. Accessed: 27.04.2021.
- [86] Embedded Staff. Bluetooth low energy (BLE) fundamentals. URL <https://www.embedded.com/bluetooth-low-energy-ble-fundamentals/>. Accessed: 23.04.2021.
- [87] Antony Stanley. Cai Yun and Fu Haifeng vs. Mathias Boe and Carsten Mogensen In The Men's Doubles Badminton Final. URL <https://www.flickr.com/photos/antonystanley/8172623247/in/photolist-dsbQan-cPCg3f-cPChvC-cPC9RQ-dsbQXH-cPC6T5-cJYrML-cNKKz1-cPCcfd-cJYkUj-cPCb5s-cPCdkm-dsbWZL-cK1DPN-cPC4CL-cJYq4b-cPCesN-cK1oVE-dc23H7-cK1wJJ-cJYpbw-cK1B2E-cJYrjo-cJYsh7-cJYkqm-cK1Ae5-cJYxQu-cK1mTb-cJZpM3-cK1tpJ-cJZ7Z1-cJYpKE-cK1BoJ-cK1ufq-cJYUB9-cJZyXw-d9YQpu-cJYYCN-cJYScG-cJYDe1-cK1d8S-cJYLT1-cJZdTq-cJYvWW-cJZ4Ym-cJZ3fE-cJZtz1-cJYzQs-cJZ2Gb-cJZzDs>. Accessed: 15.11.2020.
- [88] Arduino Team. Deep dive with Dario: Get to know the Arduino Nano Every, . URL <https://blog.arduino.cc/2019/05/31/getting-started-with-the-new-arduino-nano-every/>. Accessed: 28.04.2021.
- [89] Arduino Team. The Arduino Nano 33 BLE and BLE Sense are officially available!, . URL <https://blog.arduino.cc/2019/07/31/the-arduino-nano-33-ble-and-ble-sense-are-officially-available/>. Accessed: 28.05.2021.
- [90] BADDY team. Baddy. URL <https://play.google.com/store/apps/details?id=com.baddy&hl=no>. Accessed: 19.11.2020.
- [91] Thebadmintonguide. Badminton Shuttle – All You Need to Know About the Badminton Projectile. URL <https://www.thebadmintonguide.com/badminton-shuttle/>. Accessed: 28.11.2020.
- [92] timeanddate.no. Været som var i Trondheim, Norge - mars 2021, . URL <https://www.timeanddate.no/vaer/norge/trondheim/siste-uke?month=3&year=2021>. Accessed: 21.05.2021.
- [93] timeanddate.no. Været som var i Trondheim, Norge - mai 2021, . URL <https://www.timeanddate.no/vaer/norge/trondheim/siste-uke?month=5&year=2021>. Accessed: 21.05.2021.
- [94] TWI. What Are the Advantages And Disadvantages of 3D Printing? URL <https://www.twi-global.com/technical-knowledge/faqs/what-is-3d-printing/pros-and-cons>. Accessed: 07.04.2021.
- [95] w3schools.com. JSON - Introduction, . URL https://www.w3schools.com/js/js_json_intro.asp. Accessed: 24.03.2021.
- [96] w3schools.com. JSON Objects, . URL https://www.w3schools.com/js/js_json_objects.asp. Accessed: 24.03.2021.
- [97] Wish. 3.5mm 1.5M Desoldering Braid Solder Remover Wick. URL <https://wish.link/3ipZ9Zt>. Accessed: 28.04.2021.
-

-
- [98] B. Wan P. Visentin Q. Jiang M. Dyck H. Li G. Shan Z. Zhang, S. Li. The Influence of X-Factor (Trunk Rotation) and Experience on the Quality of the Badminton Forehand Smash. *Journal of Human Kinetics*, 53:10, 2016.

Appendix

A Changes done to BADDY's code

A.1 Wiring a push button to BADDY's PCB

```
1 const int buttonPin = 2;    // the number of the pushbutton pin
2 const int ledPin = 13;     // the number of the LED pin
3 bool buttonState = LOW;
4
5 void setup() {
6   // initialize the LED pin as an output:
7   pinMode(ledPin, OUTPUT);
8   // initialize the pushbutton pin as an input:
9   pinMode(buttonPin, INPUT_PULLUP);
10 }
11
12 void loop() {
13   // read the state of the pushbutton value:
14   buttonState = digitalRead(buttonPin);
15   // check if the pushbutton is pressed. If it is, the buttonState is LOW:
16   if (buttonState == LOW) {
17     // switch the state of the LED:
18     digitalWrite(ledPin, !digitalRead(ledPin));
19     delay (1000);
20     // Delay needed such that the signal is not sent more than once on the time you
21     // hit the button
22     buttonState = HIGH; //Set buttonState back to HIGH
23   }
24 }
```

Listing 21: ESP8266 board working with a LED

```
1 //Line 77:
2 const int buttonPin = 2;    // the number of the pushbutton pin
3 bool buttonState = LOW;    // buttonState constant
4
5 //Line 3293:
6 void setup() {
7   // initialize the pushbutton pin as an input:
8   pinMode(buttonPin, INPUT); // _PULLUP not needed
9
10 //Line 3792:
11 void loop() {
12   // read the state of the pushbutton value:
13   buttonState = digitalRead(buttonPin);
14   // check if the pushbutton is pressed. If it is, the buttonState is LOW:
15   if (buttonState == LOW) {
16     // print smiley:
17     printSmiley();
18     delay (1000);
19     // Delay needed such that the signal is not sent more than once on the time
20     // you hit the button
21     buttonState = HIGH; //Set buttonState back to HIGH
22   }
23 }
```

Listing 22: Edited parts of BADDY's code when using a button to make a smiley

A.2 Wiring the Bluetooth module to BADDY's PCB

```
1 #include <SoftwareSerial.h> //Serial communication library
2 int bluetoothTx = 1; // TX pin
3 int bluetoothRx = 2; // GPIO2 is the D4 pin on ESP8266
4 // Tx is not needed since communication is only going one way
5
6 //create an instance of a SoftwareSerial object
7 SoftwareSerial bluetooth(bluetoothRx, bluetoothTx); //Rx, Tx
8
9 int ledPin = 13; // the number of the LED pin
10 int data = 0;
11
12 void setup() {
13     delay(50);
14     // Default communication rate of the Bluetooth module
15     Serial.begin(9600);
16     // Begin The Bluetooth Module, defaults 9600bps
17     bluetooth.begin(9600);
18     // initialize the LED pin as an output:
19     pinMode(ledPin, OUTPUT);
20 }
21
22 void loop() {
23     // Checks whether data is coming from the serial port
24     if(bluetooth.available() >0){
25         data = bluetooth.read(); // Reads the data from the serial port
26     }
27     // Controlling the LED. If the received data equals the message for button
28     // clicked
29     if (data == '1') {
30         // The LED lamp will switch state
31         digitalWrite(ledPin, !digitalRead(ledPin));
32         delay(50);
33         data = 0; // variable for received data changes back to 0
34     }
35 }
```

Listing 23: ESP8266 board working with HC-08 Bluetooth module and a LED

```
1 //Line 76:
2 #include <SoftwareSerial.h> //Serial communication library
3
4 int bluetoothTx = 1; // TX pin
5 int bluetoothRx = 2; // GPIO2 is the D4 pin on ESP8266
6
7 SoftwareSerial bluetooth(bluetoothRx, bluetoothTx); //Rx, Tx
8
9 int data = 0; // constant for received data
10
11 //Line 3298:
12 void setup() {
13     // Begin the Bluetooth Module, defaults 9600bps
14     bluetooth.begin(9600);
15
16 //Line 3797:
17 void loop() {
18 // Checks whether data is coming from the serial port
19     if(bluetooth.available() >0){
20         data = bluetooth.read(); // Reads the data from the serial port
21     }
22 }
```

```

22
23 // Controlling the LED and smiley. If the received data equals the message for
    button clicked
24 if (data == '1') {
25     printSmiley(); // printing LED smiley on BADDY
26     delay(50);
27     data = 0; // variable for received data changes back to 0
28 }

```

Listing 24: Edited parts of BADDY's code when using a HC-08 module to make a smiley

A.3 Getting a shuttlecock to launch

```

1 //Line 76:
2 #include <SoftwareSerial.h>
3
4 int bluetoothTx = 1; // TX pin
5 int bluetoothRx = 2; // RX pin, D4 on ESP board
6 SoftwareSerial BTserial(bluetoothRx, bluetoothTx); // Creating a SoftwareSerial
    object - Rx, Tx
7
8 //Line 3295:
9 //Function for reading data from a remote controller
10 void readDataFromController() {
11     char data = 0; // variable for received data
12     data = BTserial.read(); // Reads the data from the serial port
13     //Different types of strokes according to the data sent from the controller
14     switch(data){
15         case '1': //One short press gives one middle drive shot
16             JsonSequence = "{\"Type\": \"SEQ\", \"Strokes\": [4,0], \"Speeds\": [4,0], \"
                LoopMode\": 0}";
17             break;
18         case '2': //One long press gives four middle drive shot
19             JsonSequence = "{\"Type\": \"SEQ\", \"Strokes\": [4,4,4,4,0], \"Speeds\":
                [4,4,4,4,0], \"LoopMode\": 0}";
20             break;
21     }
22     data = 0; // variable for received data changes back to 0
23     flag_new_sequence = true; //Flag is changed, and will trigger the functions
        sequencefetch(JsonSequence) and ReadSequence()
24 }
25
26 void setup() {
27     BTserial.begin(9600); // Begin The Bluetooth Module, defaults 9600bps
28
29 //Line 3805:
30 void loop() {
31     //If the controller is sending data, read the data
32     while(BTserial.available() >0){
33         readDataFromController();
34     }

```

Listing 25: Final additions to BADDY's code

B Controller code

```
1 #include <SoftwareSerial.h> //include library for UART communication
2
3 int bluetoothTx = 2; // TX pin of HC-08 Bluetooth Module, Arduino D2
4 int bluetoothRx = 3; // RX pin of HC-08 Bluetooth Module, Arduino D3
5
6 //create an instance of a SoftwareSerial object
7 SoftwareSerial bluetooth(bluetoothTx, bluetoothRx);
8
9 void setup()
10 {
11   Serial.begin(9600); // Begin the serial monitor at 9600bps
12   // Begin The Bluetooth Module, default is 9600bps
13   bluetooth.begin(9600);
14   bluetooth.print("AT"); // Test command
15   delay(100); // Short delay, wait for the Module to send back OK
16   bluetooth.print("AT+RX"); // Check the basic parameters
17   delay(100);
18   bluetooth.print("AT+NAME=HC-MrMaster"); // Change name on the module
19   delay(100);
20   bluetooth.print("AT+ROLE=M"); // Change role from slave to master
21   delay(100);
22   // Set connect ability so it can be connected, default is 0
23   bluetooth.print("AT+CONT=0");
24   delay(100);
25   // Binding to slave for secure and automatic connection
26   bluetooth.print("AT+BIND=F8,33,31,A9,CB,D4");
27   delay(100);
28   bluetooth.print("AT+RX"); // Check new basic parameters
29 }
30
31 void loop()
32 {
33   if(bluetooth.available()) // If the bluetooth sent any characters
34   {
35     // Send any characters the bluetooth prints to the serial monitor
36     Serial.print((char)bluetooth.read());
37   }
38   // If anything is typed into the Serial monitor
39   if(Serial.available())
40   {
41     // Send any characters the Serial monitor prints to the bluetooth
42     bluetooth.print((char)Serial.read());
43   }
44 }
```

Listing 26: Code using UART commands to set up HC-08 modules. The code is based on a sketch made by *Jim Lindblom* [35]

```

1  /* This code is written with inspiration from
2  *  Dejan Nedelkovski, www.HowToMechatronics.com
3  *  and xn1ch1,
4  *  https://www.instructables.com/Arduino-Dual-Function-Button-Long-PressShort-
      Press/
5  *
      == Controller code ==
6  */
7  #include <SoftwareSerial.h>
8  int bluetoothTx = 1; // TX pin of HC-08 Bluetooth Module, Arduino D1
9  int bluetoothRx = 0; // RX pin of HC-08 Bluetooth Module, Arduino D0,
10 // Rx is not needed since communication is only going one way
11
12 //create an instance of a SoftwareSerial object
13 SoftwareSerial BTserial(bluetoothRx, bluetoothTx); //Rx,Tx
14 #define led 4          // the number of the led pin
15 #define button 6      // the number of the pushbutton pin
16
17 boolean buttonState = false; // variable for storing last status of the pushbutton
18 boolean longPressActive = false; // variable for long press
19 long buttonTimer = 0; // variable for time tracking
20 long longPressTime = 500; // Time of a long press i milliseconds
21
22 void setup(){
23   pinMode(button, INPUT); // Configure the button pin as an input
24   pinMode(led, OUTPUT);   // Configure the led pin as an output
25   Serial.begin(9600);     //Default communication rate of the Bluetooth module
26   BTserial.begin(9600);  // Begin The Bluetooth Module, defau
27   lt is 9600bps
28 }
29
30 void loop(){
31   digitalWrite(led, HIGH); // Led turns on when the controller turns on
32   if (digitalRead(button) == HIGH) { // The button is pressed
33     if (buttonState == false) { // Detect if buttonState has changed since the last
          iteration
34       buttonState = true; // A buttonpress is detected
35       buttonTimer = millis(); // Record the time of the buttonpress in milliseconds
36     }
37     // Check if current time subtracted from the time of the first buttonpress is
          longer than time for a long press, and if long press have been avtivated since
          last iteration
38     if ((millis() - buttonTimer > longPressTime) && (longPressActive == false)) {
39       longPressActive = true; // variable changed to true to stop repeated
          activation of long press
40       Serial.print(BTserial.write('2')); // Send long press signal to BADDY
41     }
42   } else { // Button is not pressed anymore
43     if (buttonState == true) { // The button has been pressed
44       if (longPressActive == true) { // Checking if a long press was activated
45         longPressActive = false; // Changed to false to allow new long press next
          iteration
46       } else { // Long press was not activated
47         Serial.print(BTserial.write('1')); // Send short press signal to BADDY
48       }
49       buttonState = false; //Changed to allow new press in next iteration
50     }
51   }
52   delay(50);
53 }

```

Listing 27: Arduino code for the controller

