dARts

Jun 22, 2022

Contents

Dies ist die Doku für das Projekt dARts. Eine Arduino und Raspberry Pi gesteuerte elektronische Dartscheibe in Automatenqualität. Gestützt vom Scoreboard System

CHAPTER 1

Meine Projektmotivation

Dieses Projekt ist ein Folgeprojekt aus dem Scoreboardprojekt *Dart-O-Mat 3000*. Der DOM3000 soll ein Scoreboard sein für Verwertung von Wurfdaten, die eine Bilderkennung während eines Steeldart Spieles sendet. Da das Scoreboard via RESTful API arbeitet kann es leicht adaptiert werden. Ich selbst spiele ebenso gerne E-Dart, wie auch Steeldart. Dennoch ist es meist frustrierend, wie schlecht verarbeitet und dafür verhältnismäßig teuer elektronische Dartscheiben sind, die man so im Internet oder im Fachhandel erhalten kann. Die Löwenautomaten dagegen jedoch sind von der Qualität spitze. So war schnell die Idee geboren eine E-Dart Scheibe in Form eines "Automaten" zu bauen und hierfür aus dem Ersatzteilelager von Löwen echte Automatenkomponenten zu verwenden. Dieses Projekt ist demnach mein Versuch für ein günstiges Geld eine elektronische Dartscheibe zu erstellen, die eine Automatenqualität aufweist. Und das beste ist, dass ich den Dart-O-Mat 3000 so noch ein zweites mal verwenden konnte.

CHAPTER 2

Hardware

Nachfolgend wird die Hardware aufgeführt, die zum Bau des Automaten notewendig ist.

2.1 Gekauft habe ich

Die nachfolgenden Teile sind eine Empfehlung und entsprechen den Teilen, die ich gekauft habe:

- Ardunio Mega 2560 (China Nachbau):
- Raspberry PI 3B+:
- Knopf:
- Piezo Sensoren:
- Ultraschall Sensor:
- Löwen Komplett Set inklusive Matrix:
- Schloss:

2.2 Sonstiges hatte ich noch

Was man sonst noch so braucht, habe ich noch rumliegen gehabt:

- Lötkolben
- Litze
- Prototyping Steckplatinen
- Lochraster Lötplatinen
- Widerstandsset
- Schrumpfschlauch

- Jumper Kabel
- LED Band 12V

CHAPTER 3

Einkaufsliste Holz mit Vermaßung

Ich habe einen Bauplan entworfen, der sich stark an orientiert. Die nachfolgenden Maße an Holz haben für mich gut funktioniert:

3.1 MDF

- Segmentbrett: 480mm x 520mm x 12mm

3.2 Sperrholz/Multiplex

- Matrixbrett: 520mm x 520mm x 18mm
- Türe: 600mm x 750mm x 12mm
- Rückwand: 600mm x 750mm x 12mm
- 3x Stirnbretter: 750mm x 200mm x 12mm
- Boden: 750mm x 200mm x 12mm

CHAPTER 4

Hochrechnung aller Kosten

- Arduino: 16,00 €
- Header Platine: 17,50 €
- Raspberry Pi: 32,00 €
- SD Karte: 7,00 €
- Knopf: 8,00 €
- Piezos: 10,00 €
- Ultraschall Sensor: 4,00 €
- Löwen Komplett Set inklusive Matrix: 190,00 €
- Schloss: 10,00 €
- MDF: 2,50 €
- Sperrholz/Multiplex: 45,00 €
- Farbe: 9,00 €
- Walzen: 4,50 €
- Klebezahlen: 20,00 €

4.1 Nicht berücksichtigt wurden

- LED Band: 15,50 € ()
- Netzteil (Kombination aus Steckernetzteil und Adapter): 20,00 € (und)
- Kleinteile, wie zum Beispiel Litze, Widerstände, Schrumpfschlauch, . . . : 25,00 €

4.2 Summe

Die Summe beläuft sich demnach auf: 436,00 €

CHAPTER 5

Pinbelegung

Komponente	PIN	PIN Arduino
Push Button	LED +	3
	LED -	GND
	Signal +	2
	Signal -	GND
Ultraschall Sensor	VCC	5V
	TRIG	9
	ECHO	8
	GND	GND
Piezo1	S	A0
	-	GND
Piezo2	S	A1
	-	GND
Matrix Stecker	1	22
	2	24
	3	26
	4	28
	5	30
	6	32
	7	34
	8	36
	9	38
	10	40
	11	42
	12	44
	13	46
	14	48
	15	50
	16	52
	17	53

Continued on next page

Table 1 – continued from previous page

	18	51
	19	49
	20	47

CHAPTER 6

Bauanleitung

In der nachfolgenden Anleitung werden alle Schritte erläutert, die ich durchgeführt habe, um den Automaten als fertiges Produkt zu erhalten.

6.1 Schritt 1: Segmentbrett

Das Segmentbrett besteht aus MDF und ist 480mm auf 520mm und 12mm stark.

Auf diesem Brett muss zuerst die Mitte ermittelt werden und dann eine Bohrung für den Fräszirkel gesetzt werden.

Zuerst wurde eine Probefräsung mit dem selbstgebauten Fräszirkel gemacht. Danach wird erst das Werkstück gefräst.

Als erstes wird der äußerde Durchmesser der Spinne für 2mm versenkt (eingelassen) und anschließend wird der innere Durchmesser durchgestochen. Hierbei ist darauf zu achten, dass man die Fräsung für des Loch von oben bis zur halben Materialstärke macht, dann das Brett wendet und den Durchbruch von der anderen Seite fräst. Somit bekommt man saubere Kanten.

Nun kann die Spinne durch das Loch gesteckt werden, ausgerichtet werden und mit Schrauben fixiert werden. Sie schließt plan ab und kann so nachher auf das Matrixbrett geschraubt werden.

Abschließend werden noch die Catchsegmente von der Hinterseite verschraubt und fertig ist die Konstruktion des Segmentbretts.

6.2 Schritt 2: Matrixbrett

Das Matrixbrett besteht aus Sperrholz Multiplex und ist 520mm auf 520mm und 18mm stark.

Durch Einmessen der Mitte und Zeichnen eines Kreuzes zur Markierung der Mitte ist es möglich die Matrix auszurichten. Die Matrix selbst ist trotz Trägerpapier etwas durchscheinend. Zur Orientierung kann man auch die Druckpunkte der Matrix verwenden. Wichtig ist, dass die Matrix perfekt mittig aufgebracht wird.

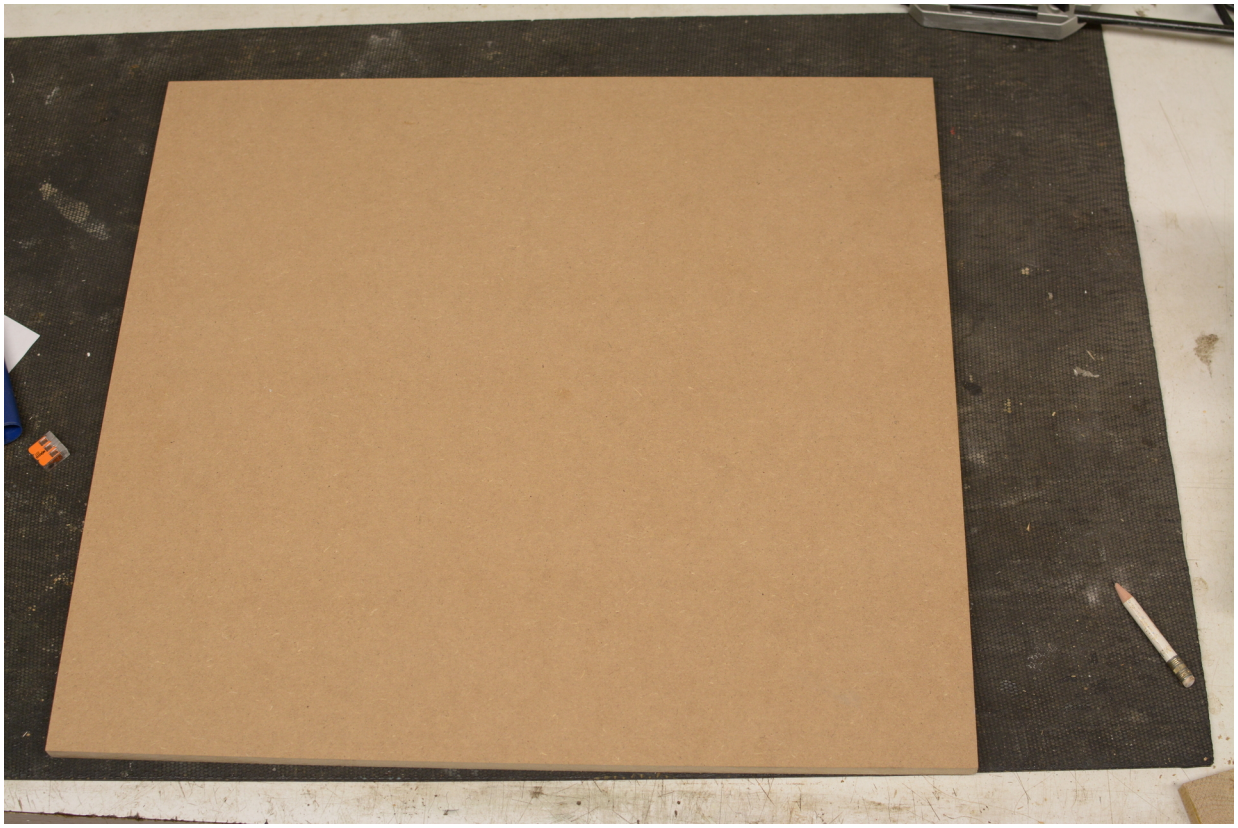


Fig. 1: Brett für die Spinne (480mm x 520mm x 12mm MDF)

Fig. 2: Mitte für Fräszirkel

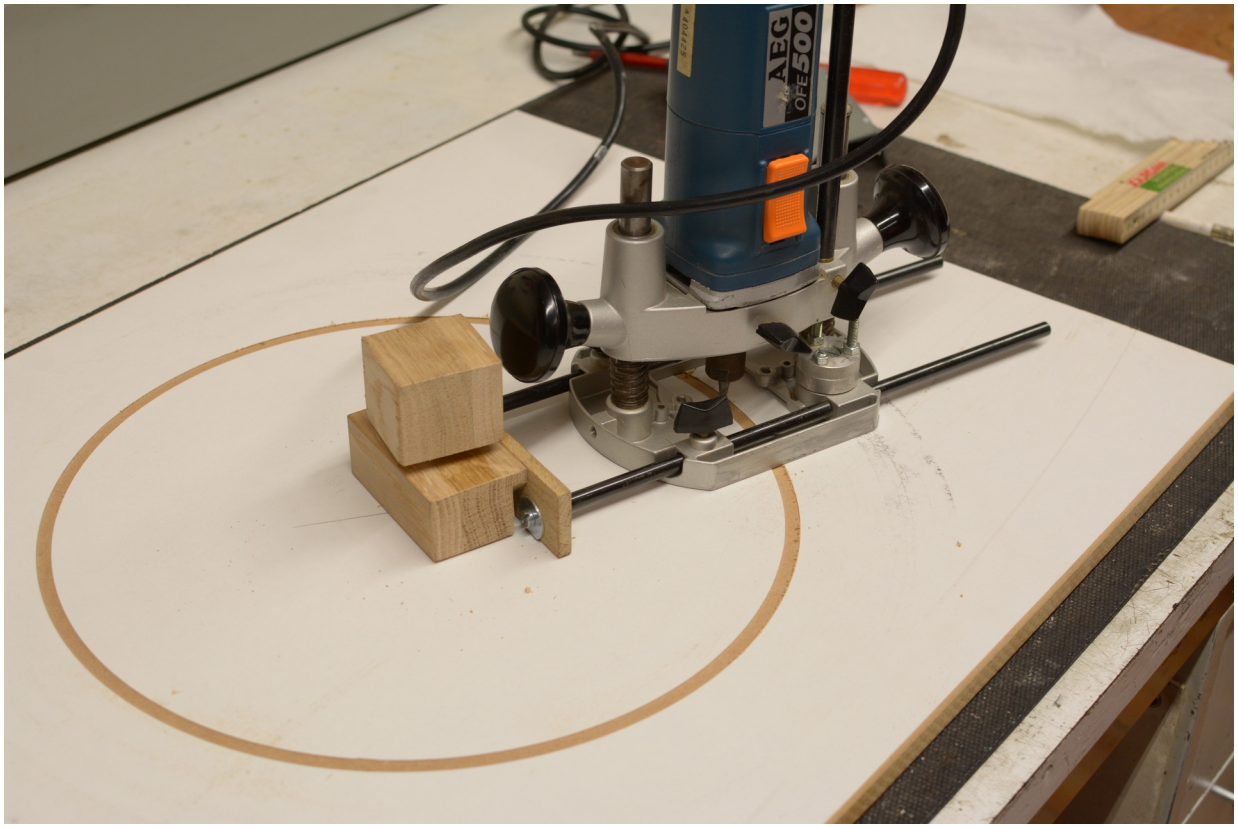


Fig. 3: Probefräsung auf Abfallplatte

Fig. 4: Einlassen des äußeren Durchmessers (2mm)

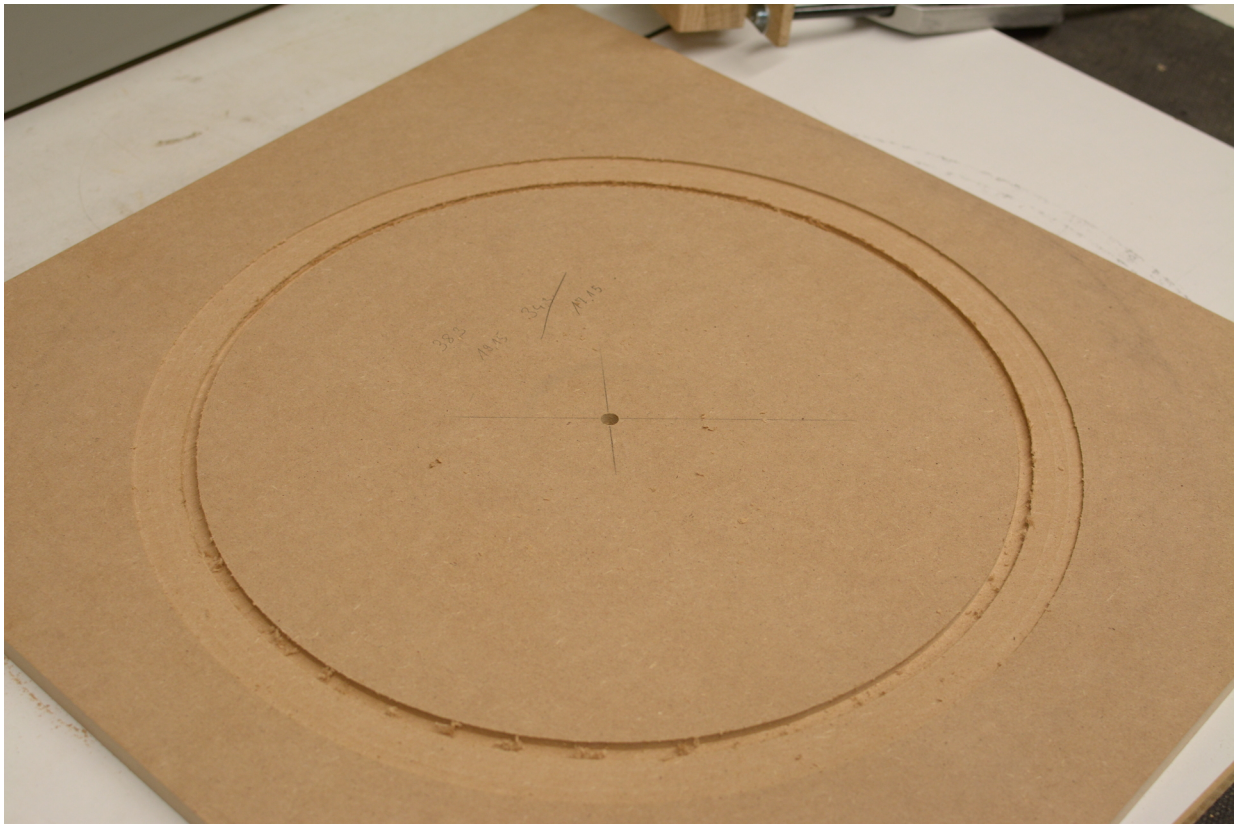


Fig. 5: Die Spinne ist eingelassen

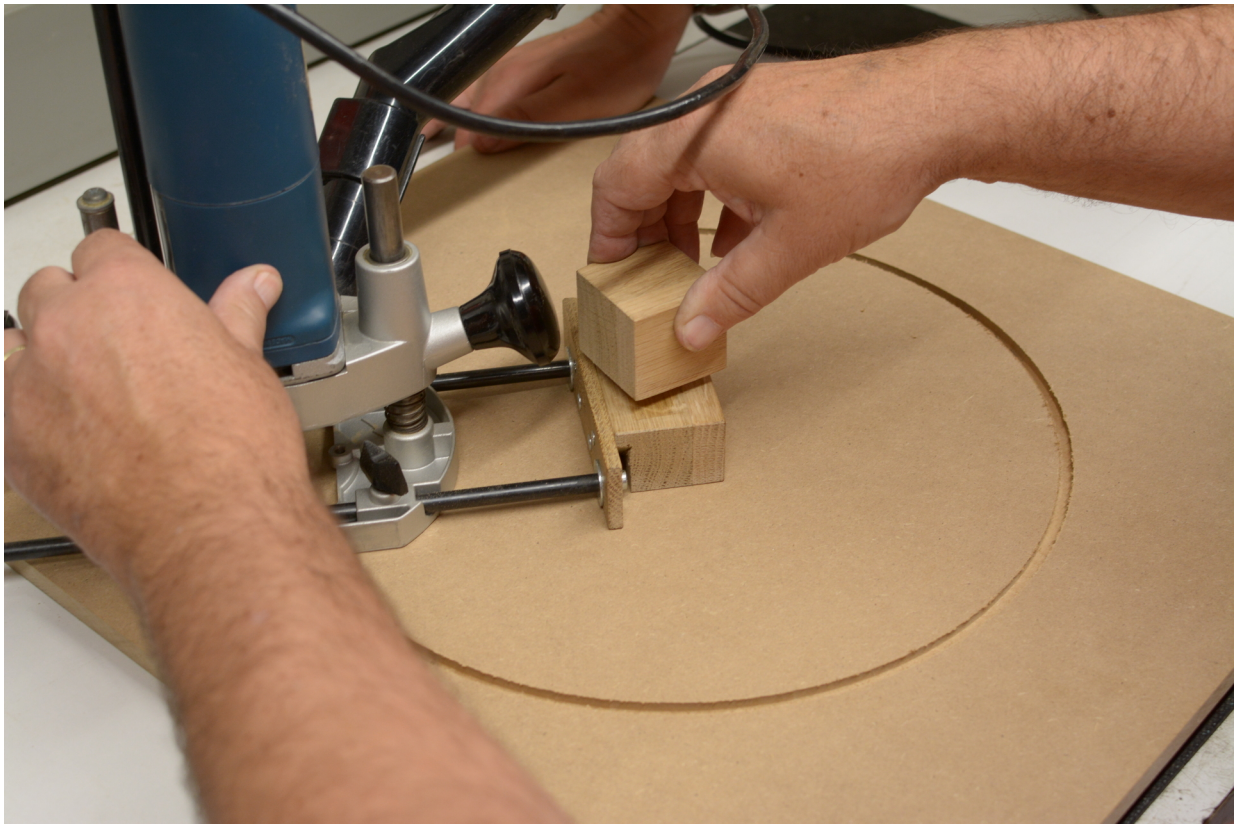


Fig. 6: Durchfräsen von der anderen Seite

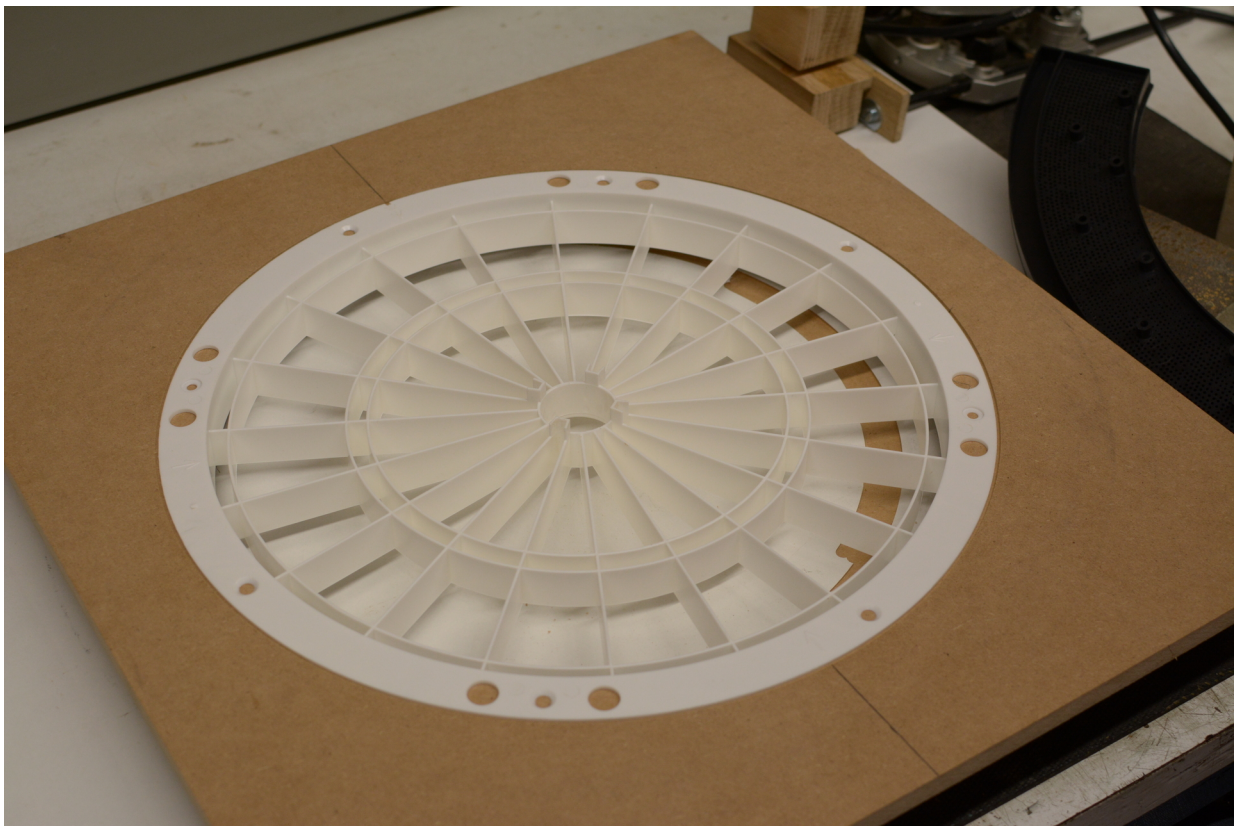
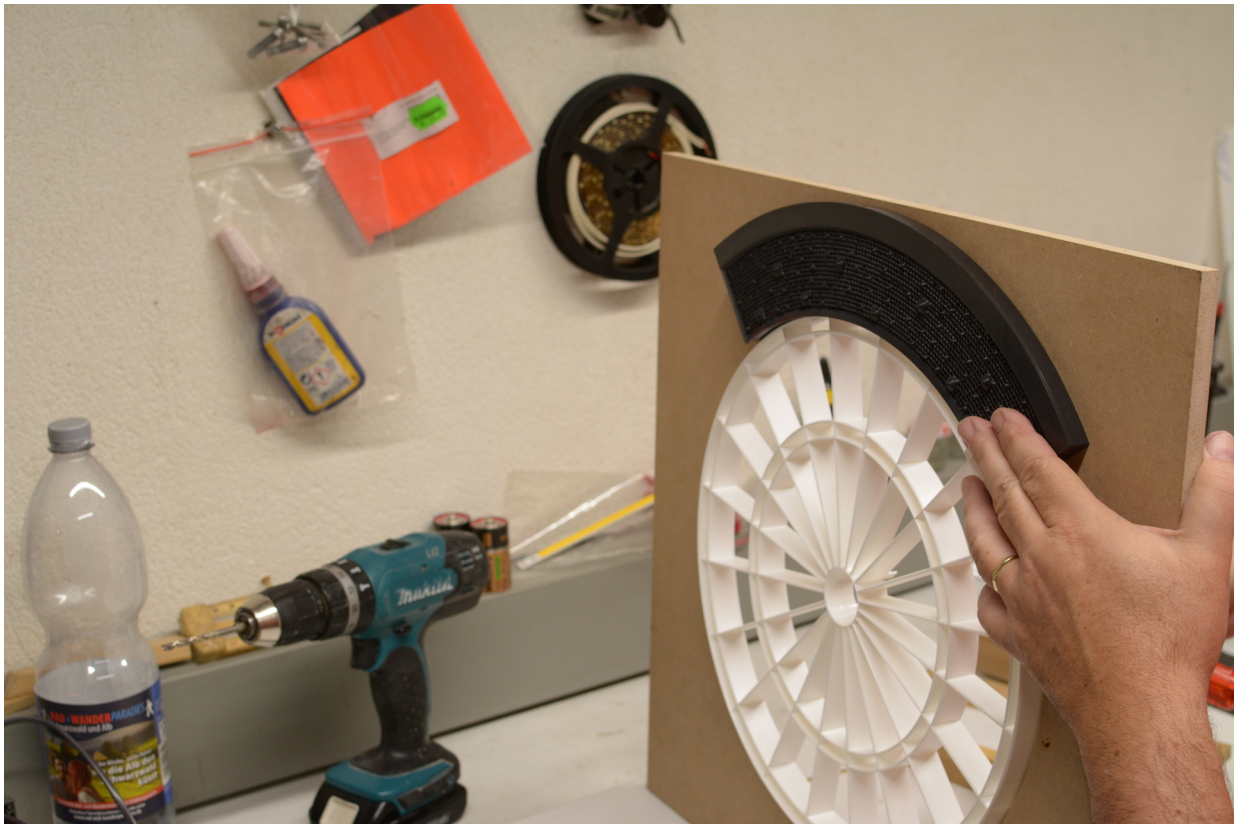


Fig. 7: Die Spinne passt



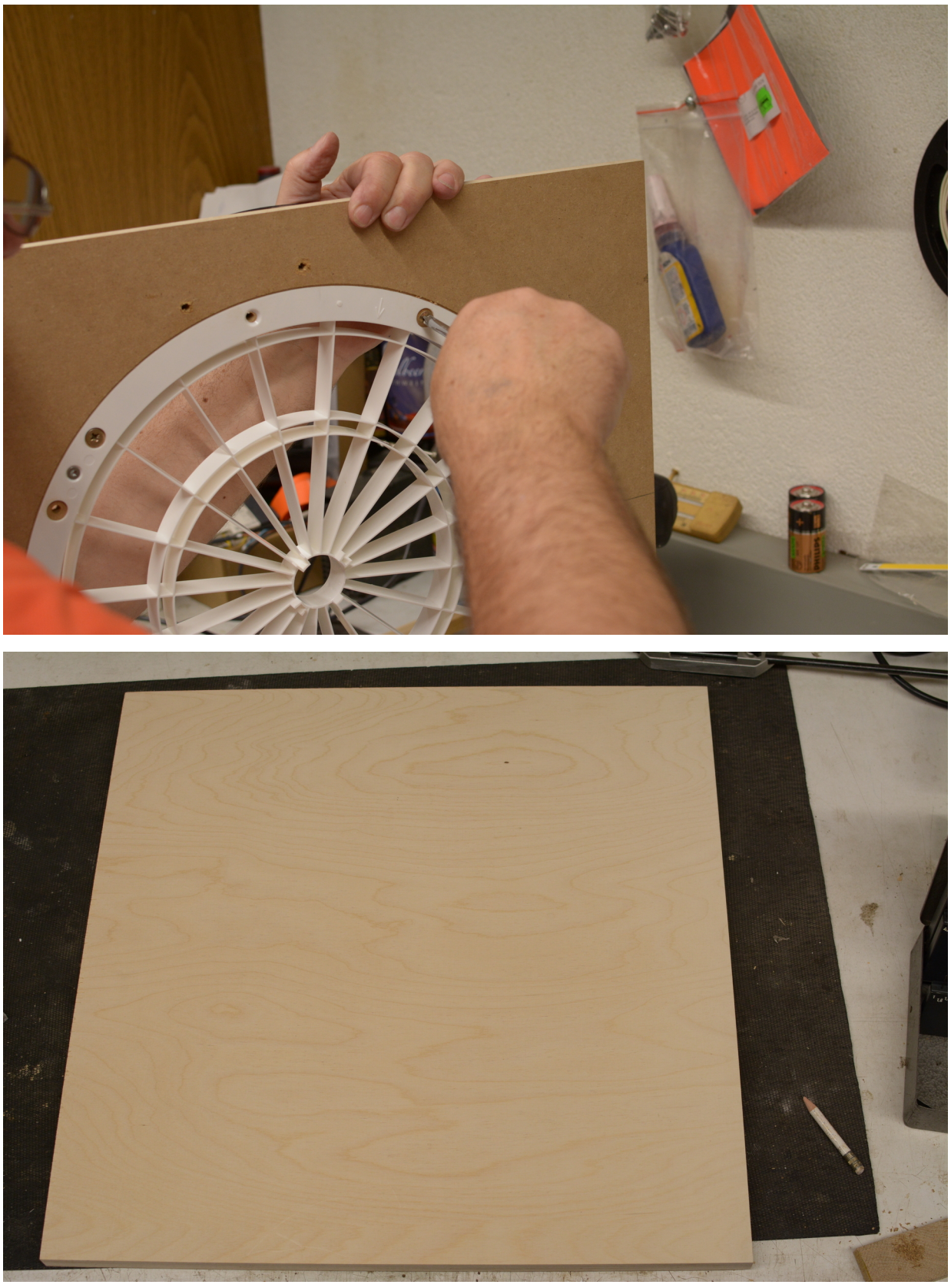


Fig. 8: Brett für die Matrix (520mm x 520mm x 18mm)

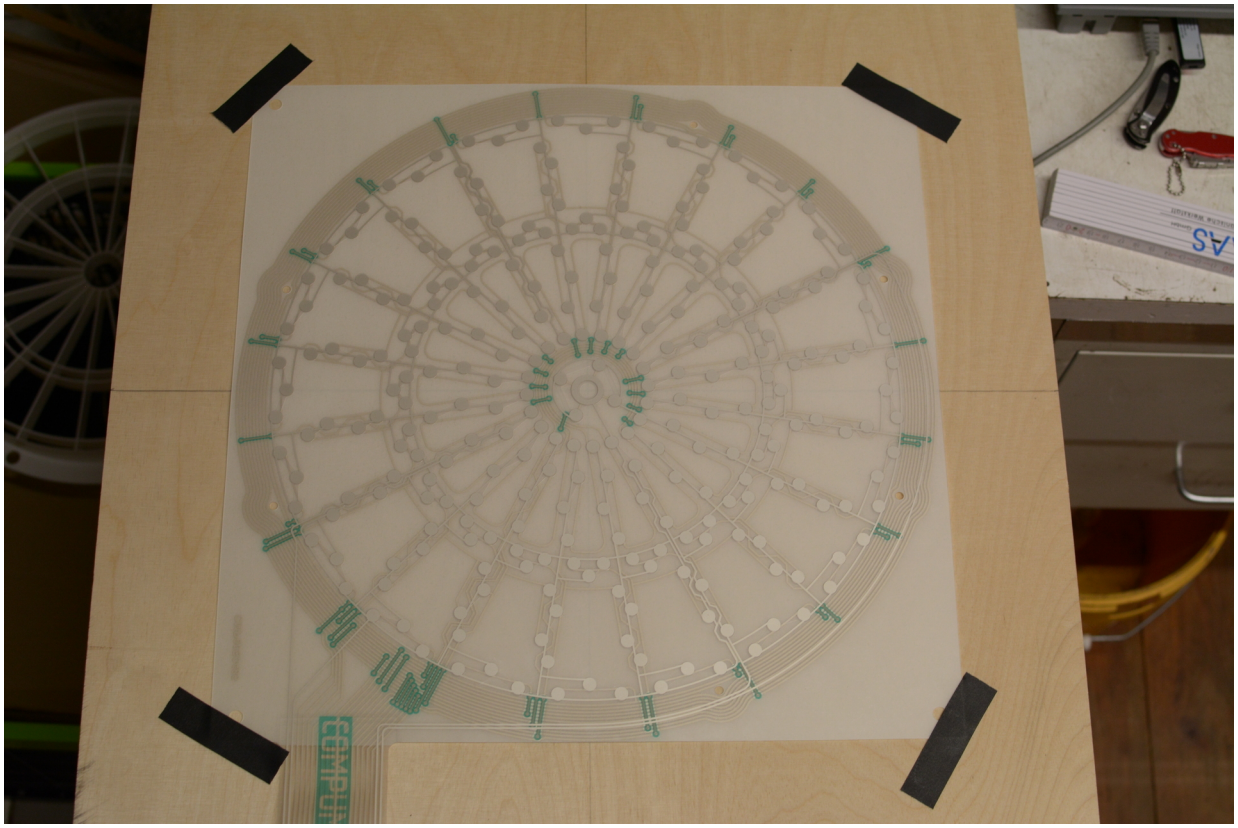


Fig. 9: Matrix ist mittig auf dem Brett fixiert

Wichtig ist es die selbstklebende Matrix noch nicht aufzukleben. Zuerst fixieren wir sie mit Klebeband, setzen das Segmentbrett (mit ein paar eingelegten Segmenten) oben auf und testen, ob bei der aktuellen Positionierung alle Felder korrekt erkannt werden.

Ist das der Fall kann man die Matrix mit einem Edding oder Bleistift umranden, das Trägerpapier auf der Rückseite entfernen und dann die Matrix auf das Brett aufkleben und glatt streichen. Achtung, die Positionierung muss genau eingehalten werden!

Anschließend kann man das Segmentbrett wieder oben auflegen und mit 4 Schrauben auf das Matrixbrett fixieren.

Fig. 10: Fertige Einheit

Es ist darauf zu achten, dass die Schraubenpositionen so gewählt werden, dass die Matrix nicht durchbohrt wird.

6.3 Schritt 3: Rückwand und Position der Einheit

Die Rückwand besteht aus Sperrholz Multiplex und ist 750mm auf 600mm und 12mm stark.

Auf der Rückwand kann nun mit einem Rahmen die genaue Position der Einheit ermittelt werden. Unten muss Platz für die Technik bleiben und die Einheit sollte mittig sitzen.

Dann bohrt man durch den Überstand der Matrixplatte und die Rückwand je 1 Loch in jeder Ecke. Dort habe ich bei diesem Automaten dann Gewindeschrauben eingeklebt um die fertige Einheit auf der Rückwand mit Gewindehülsen befestigen zu können.

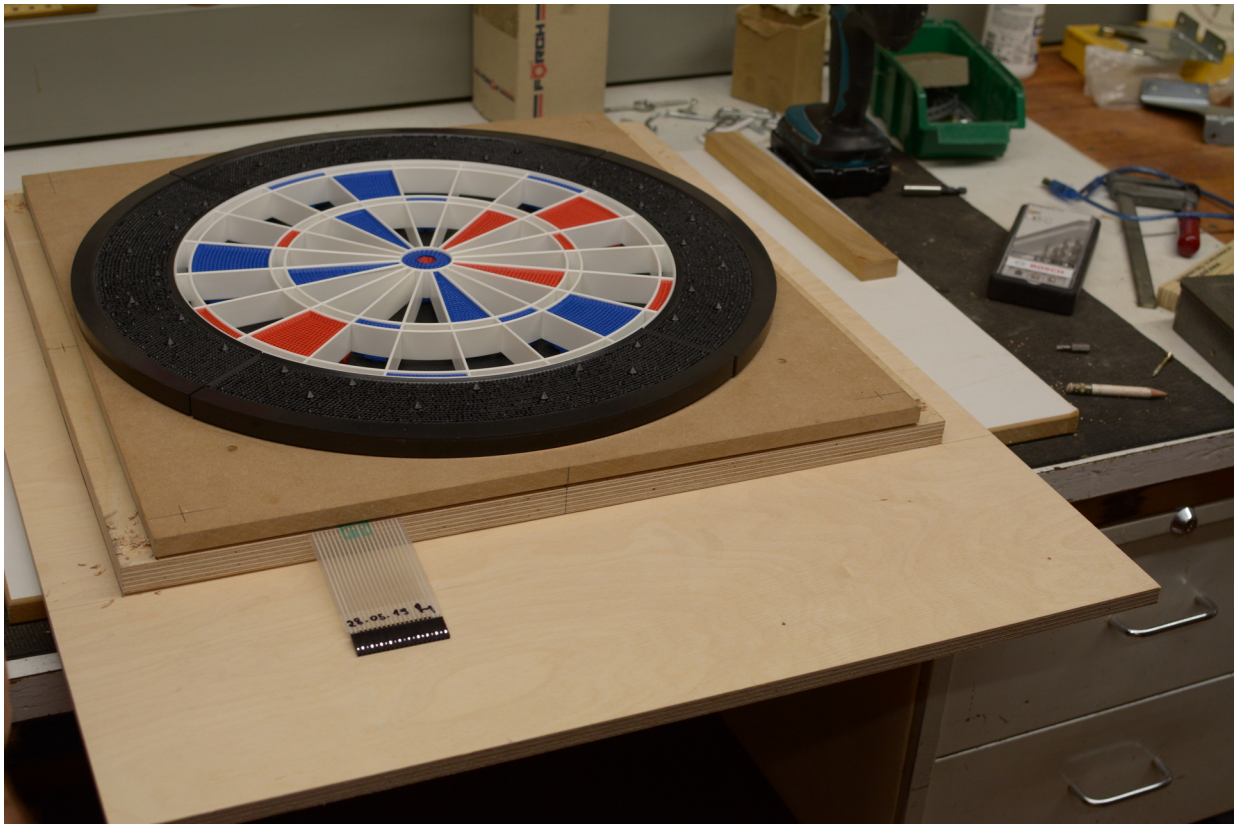


Fig. 11: Fertige Einheit sitzt passgenau auf der Rückwand

Fig. 12: Einkleben der Schrauben von der Rückseite aus

Fig. 13: Schrauben sind fertig für die Einheit

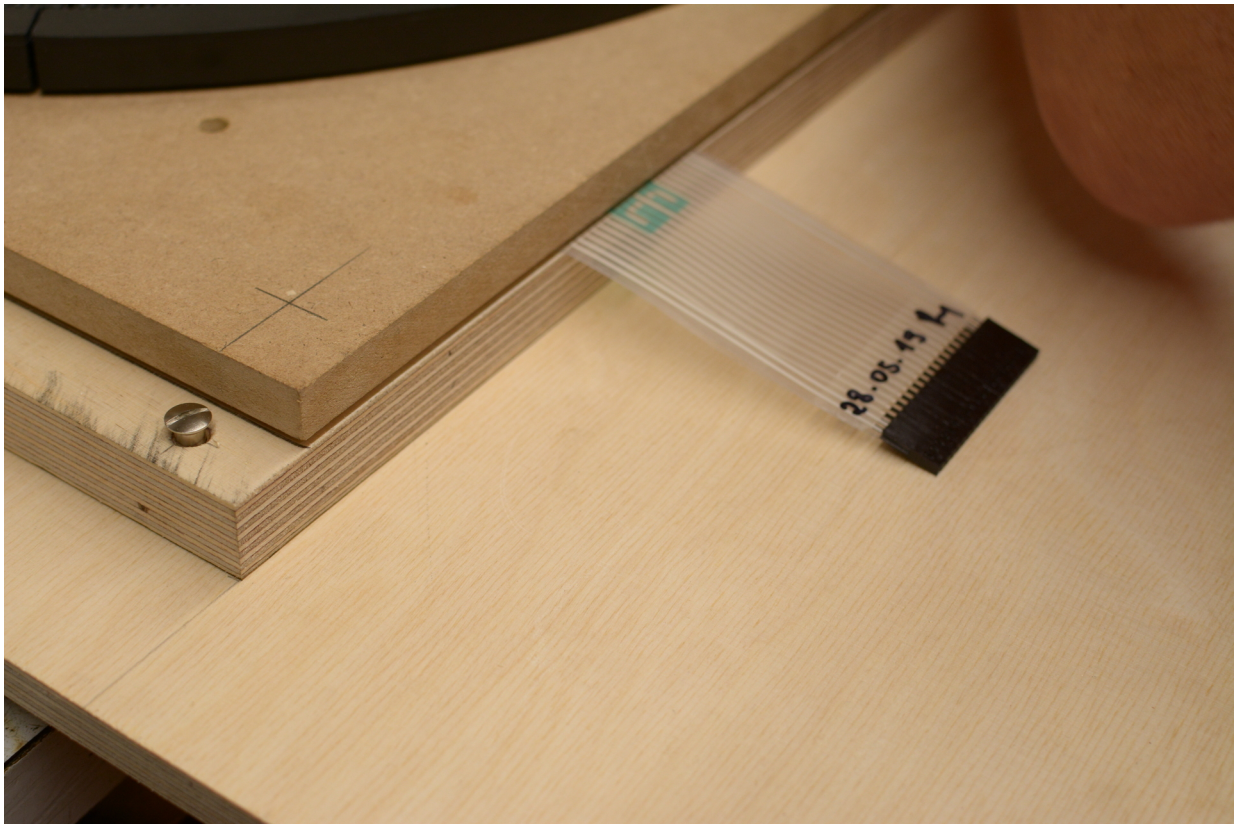


Fig. 14: Einheit wird mit Hülsen auf der Rückwand befestigt

6.4 Schritt 4: Türe

Die Türe besteht aus Sperrholz Multiplex und ist 750mm auf 600mm und 12mm stark.

Die Türe bekommt den gleichen Rahmen eingezeichnet, wie die Rückwand, sodass die Umrisse der fertigen Einheit auch auf der Türe zu sehen sind. Nun kann man die Mitte des Rahmens und somit die Mitte der Scheibe ermitteln.

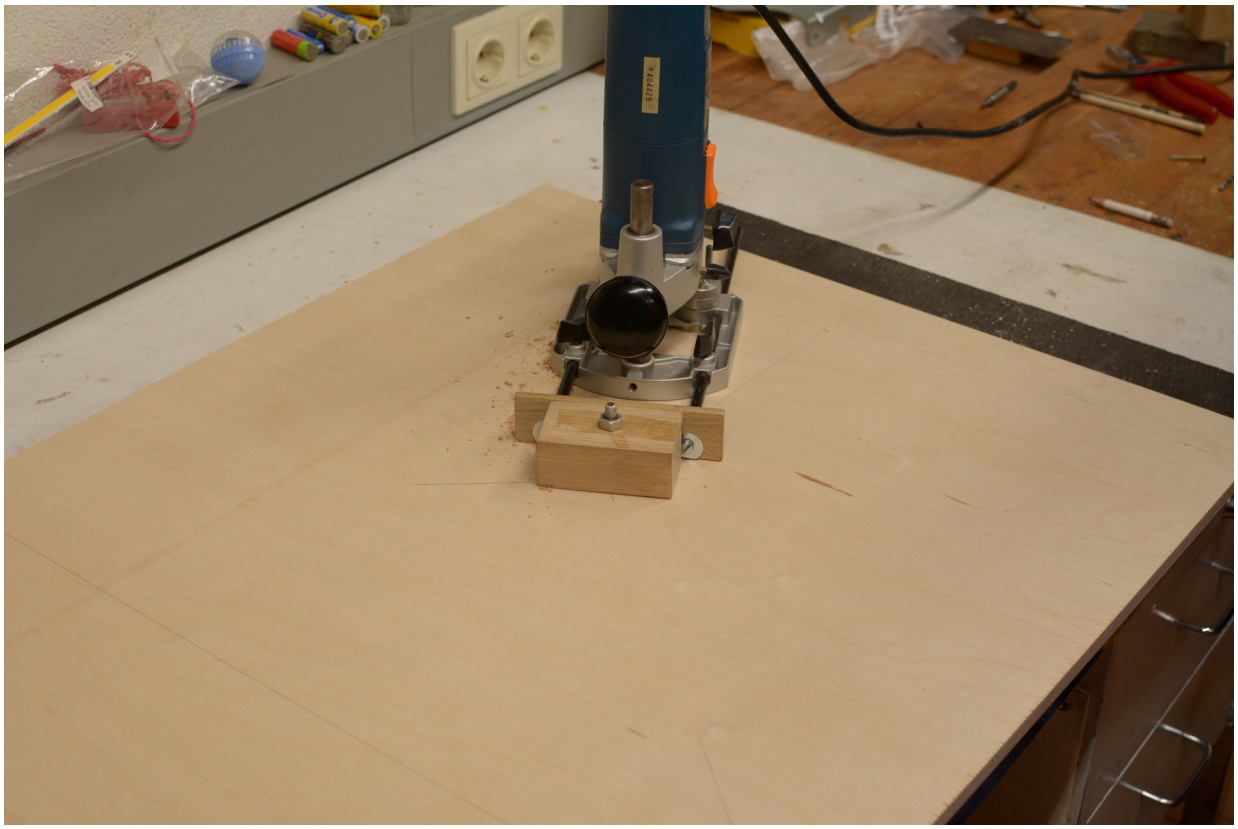


Fig. 15: Mittelbohrung für Fräszirkel in der Türe

Anschließend kann man mit dem Fräszirkel ein Loch in die Türe für die Scheibe machen. Hierbei sollte darauf geachtet werden, dass der Radius so gewählt wird, dass etwa 1,5mm Luft ist, um anschließend noch den Kantenschutz einbringen zu können.

Anschließend kann man die Türe für eine Passprobe auf das Werkstück legen.

Abschließend wird noch ein Loch für den Knopf eingelassen.

6.5 Schritt 5: Montage Seitenwände

Die Seitenwände bestehen aus Sperrholz Multiplex und sind 750mm auf 200mm und 12mm stark.

Die Seitenwände werden auf die Stirnseiten der Rückplatte befestigt. Zuerst die Seiten dann das Oberteil. Unten wird vorerst keine Platte befestigt.

Hierbei ist es wichtig alle Schraubenlöche vorzubohren. Die Köpfe der Schrauben können auch versenkt werden. So sieht der fertige Automat nachher noch besser aus.

Fig. 16: Türloch wird eingefräst

Fig. 17: Türe passt mit Kantenschutz

Fig. 18: Knopf in Türe

Fig. 19: Seitenteil wird befestigt

6.6 Schritt 6: Streichen

Als nächsten Schritt habe ich die Teile gestrichen. Dazu verwende ich einen matten, schwarzen Lack, den ich mit einer Lackierwalze (glatt und aus Schaumstoff) aufgebracht habe. Nach einer Schicht kommt noch leicht die Maserung des Holz durch, was ich ziemlich schön finde. Für ein deckendes Ergebnis muss man das Holz eher zweimal streichen. Man sollte hierbei unbedingt die Trocknungszeit beachten.

Fig. 20: Türe schwarz gestrichen

6.7 Schritt 7: Trocknungszeit nutzen, Einheit bestücken

Während der Kasten und die Türe trocknen kann man die Einheit bestücken.

Dazu muss man die Einheit zuvor erneut zerlegen und dann die Segmente korrekt einsetzen. Ich habe dann noch zwei Piezo Sensoren hinter dem Catchring auf der Rückseite des Segmentbretts mit Klebeband fixiert und die Kabelzuleitungen seitlich heraushängen lassen.

6.8 Schritt 8: Technik - Stecker, Kabel, Arduino, Pi

Ich habe mich entschieden Jumperkabel zu nehmen und eine Seite mit Schrumpfschlauch zu Steckern zu schrumpfen. Diese Stecker können später einfach auf der individuellen Platine aufgesteckt werden.

Die Technik habe ich an der richtigen Stelle angezeichnet. Dann habe ich von hinten Abstandshalter verschraubt, auf der die Technik fixiert werden kann. So sitzen Arduino und Pi anschließend sicher im Gehäuse.

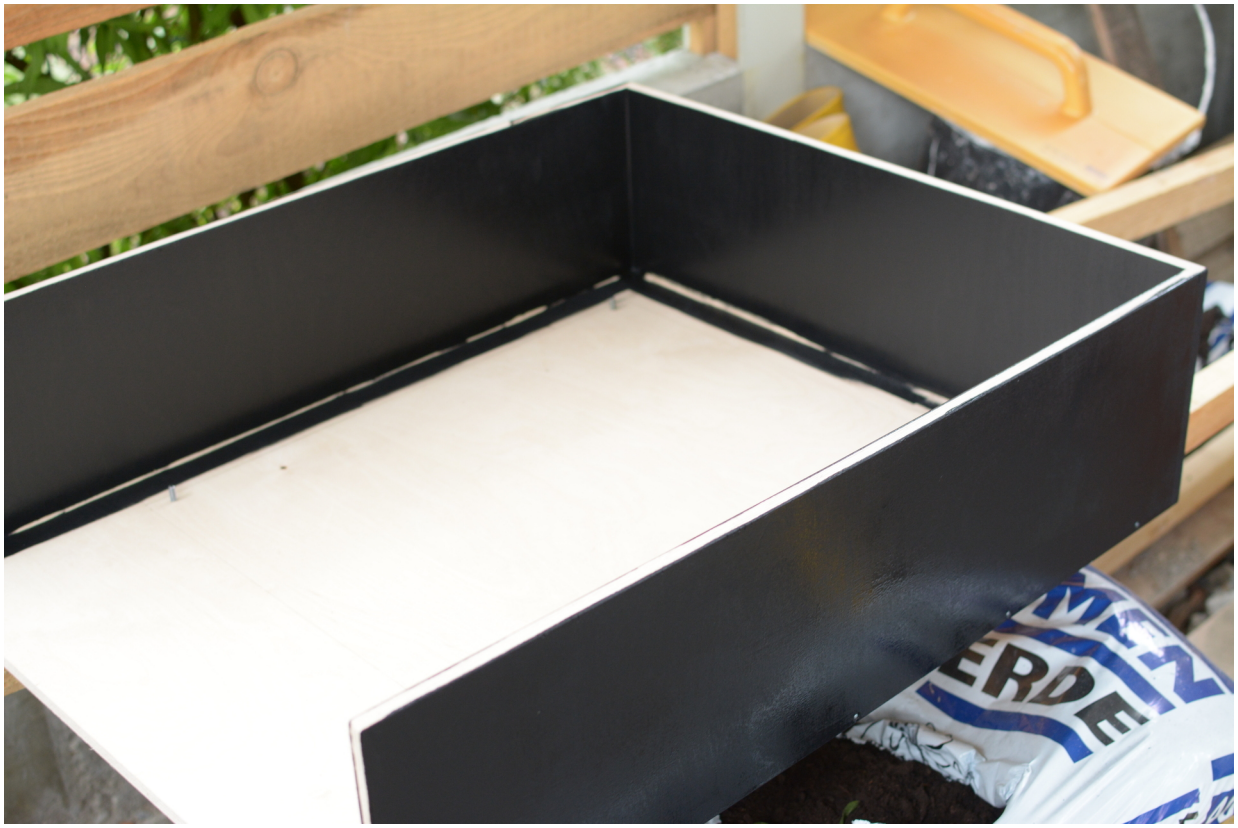


Fig. 21: Kasten schwarz gestrichen

Fig. 22: Fertig bestückte Spinne

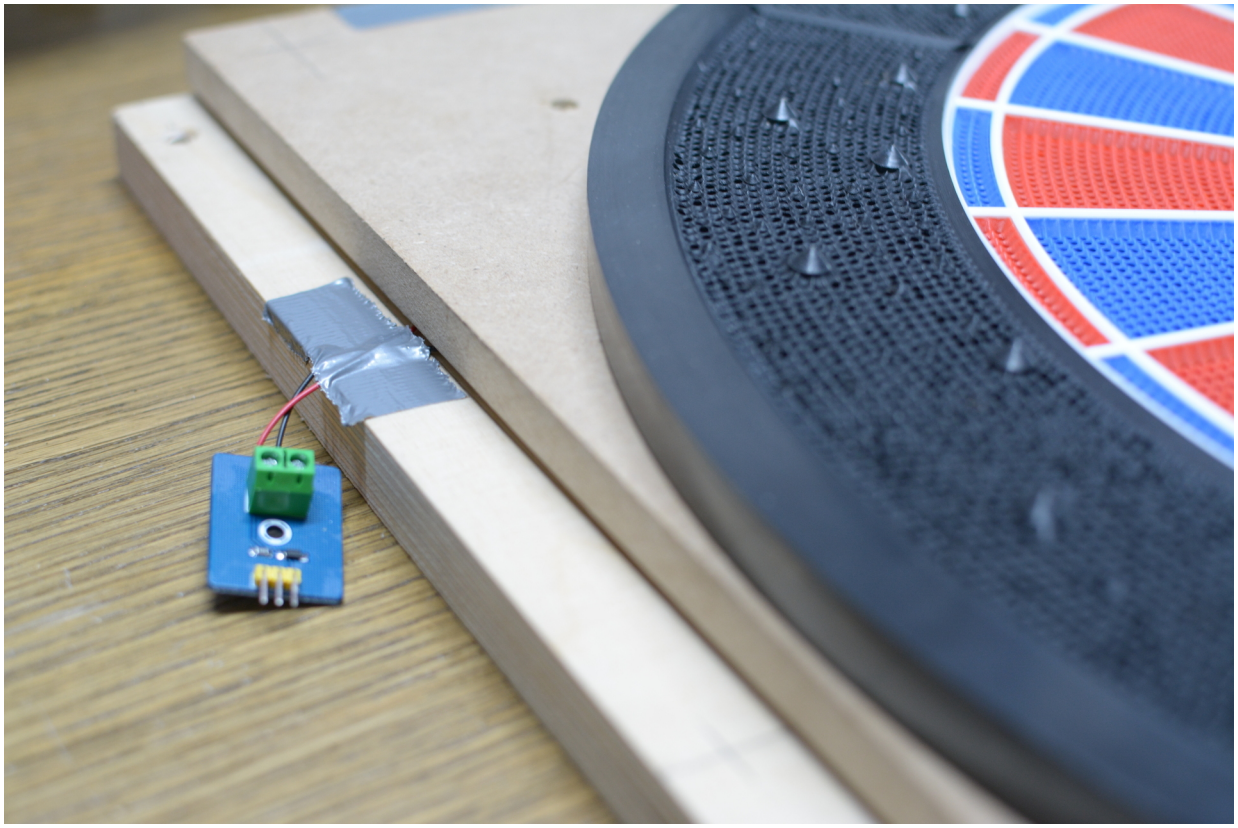


Fig. 23: Kabelzuleitung Piezosensor

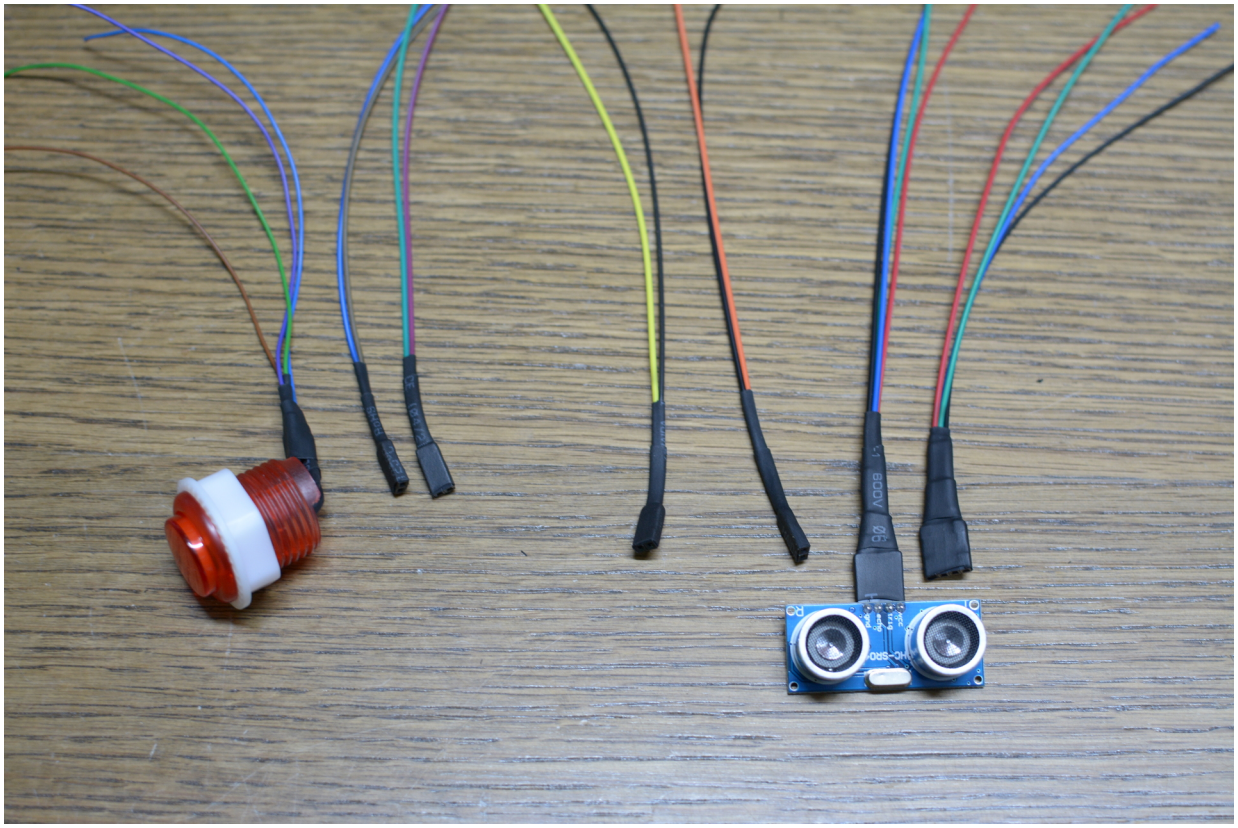


Fig. 24: Jumperkabel wurden zu Steckern gefertigt

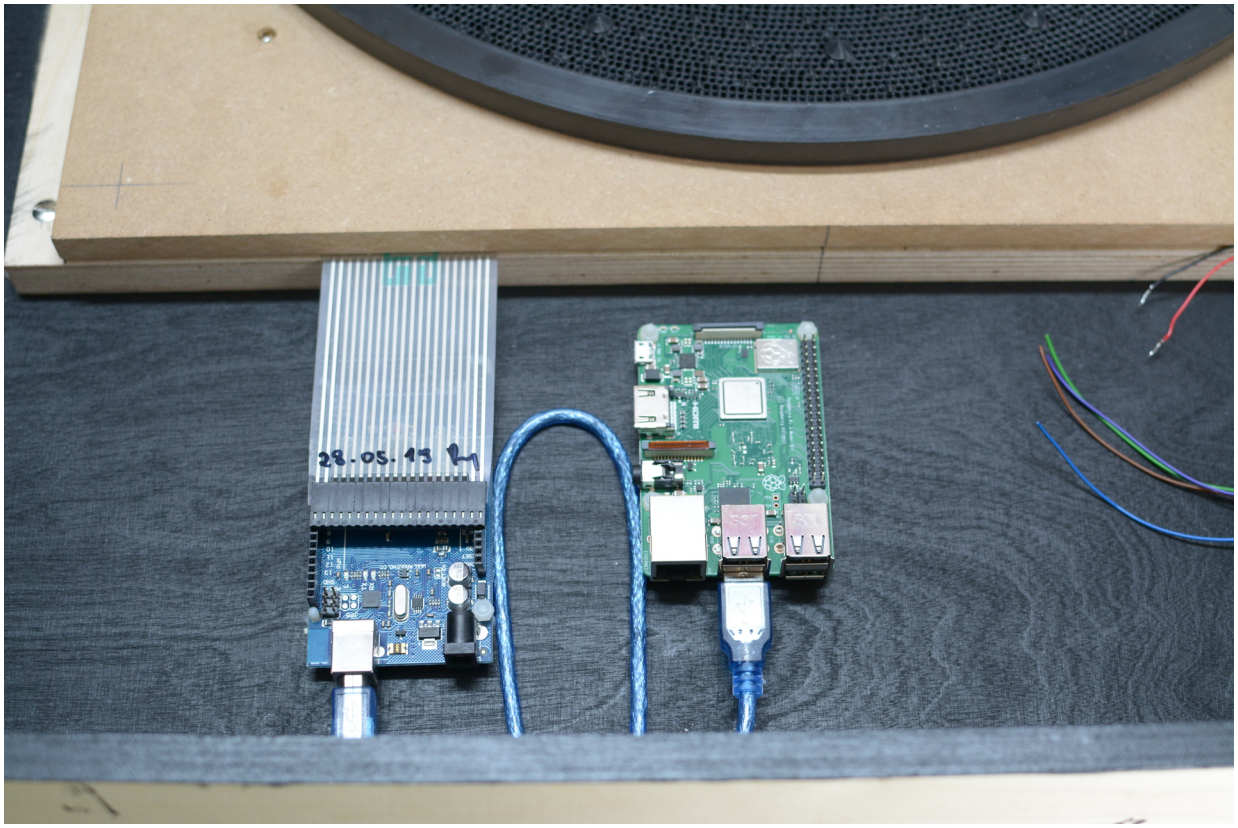


Fig. 25: Arduino und PI sitzen sicher im Gehäuse

Abschließend kann auch die Einheit wieder eingesetzt werden, sodass man die Kabelführung besser planen kann.

Fig. 26: Die Einheit ist wieder auf der Rückplatte befestigt

6.9 Schritt 9: Kabel verlegen

Jetzt können die Kabel verlegt werden und die Stecker in Position gebracht werden.

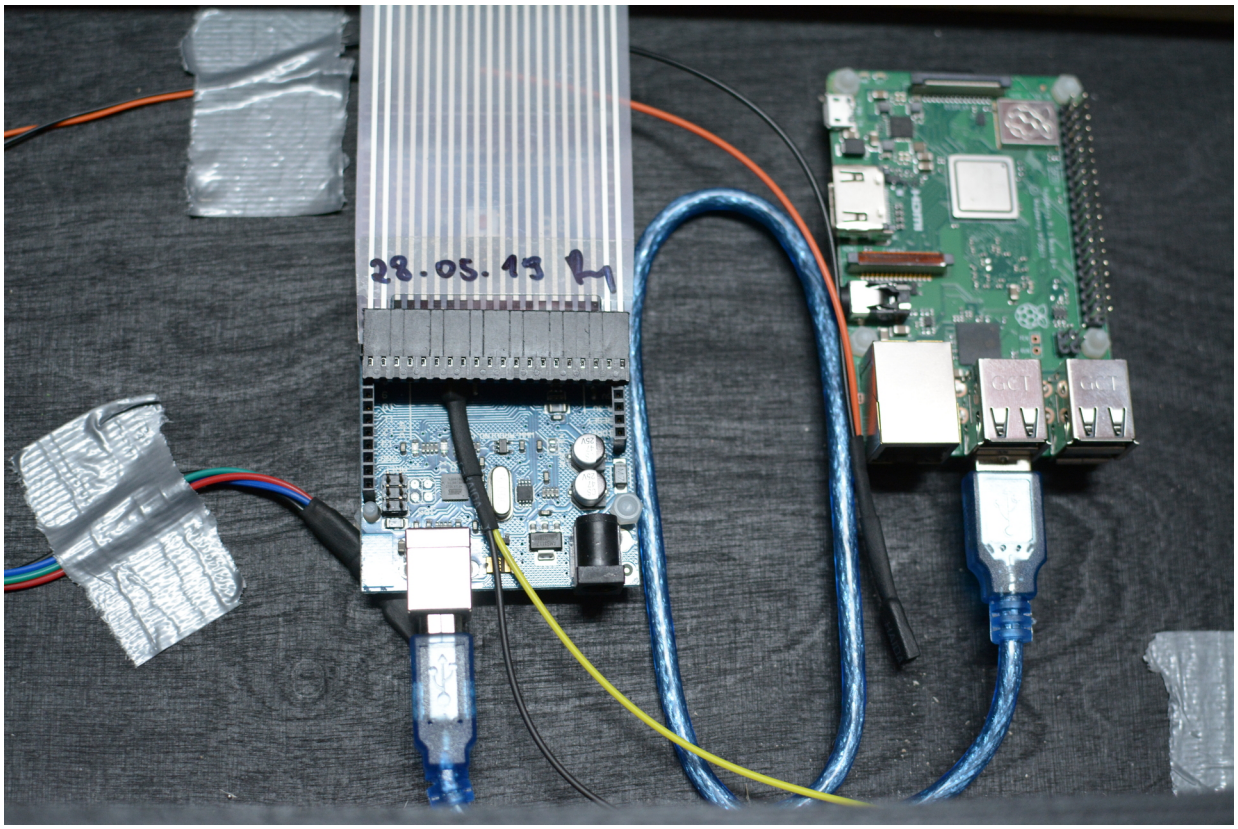
Hierbei habe ich den Ultraschallsensor von oben angebracht (durchgebohrt) und die Kabel über die Oberseite unsichtbar verlegt. Außerdem habe ich die Kabelstrecken immer wieder mit Klebeband vor dem Verrutschen fixiert.

6.10 Schritt 10: Türe inklusive Verriegelung

In das Loch in der Türe wird der Kantenschutz mit einem Tacker eingebracht. Ich musste dazu noch die Führungsnut des Kantenschutzes mit einem Messer abschneiden.

Dann kann erneut geprüft werden, ob die Türe passt. Durch die Lackierung und das zu genaue Arbeiten war ich gezwungen die Türe nochmals an den Seiten und unten abzuschleifen, damit sie passt.

Nun habe ich von oben und einmal seitlich ein kleines Loch gebohrt, in das ein Nagel gesteckt werden kann. Dann wird die Türe vor dem Verrutschen gesichert. Im vierten Eck wird das Schloss eingebaut und verriegelt die Türe. Außerdem dient es zum Ausziehen der Türe für Wartungsarbeiten. Hier muss genau gearbeitet werden.


28.05.13

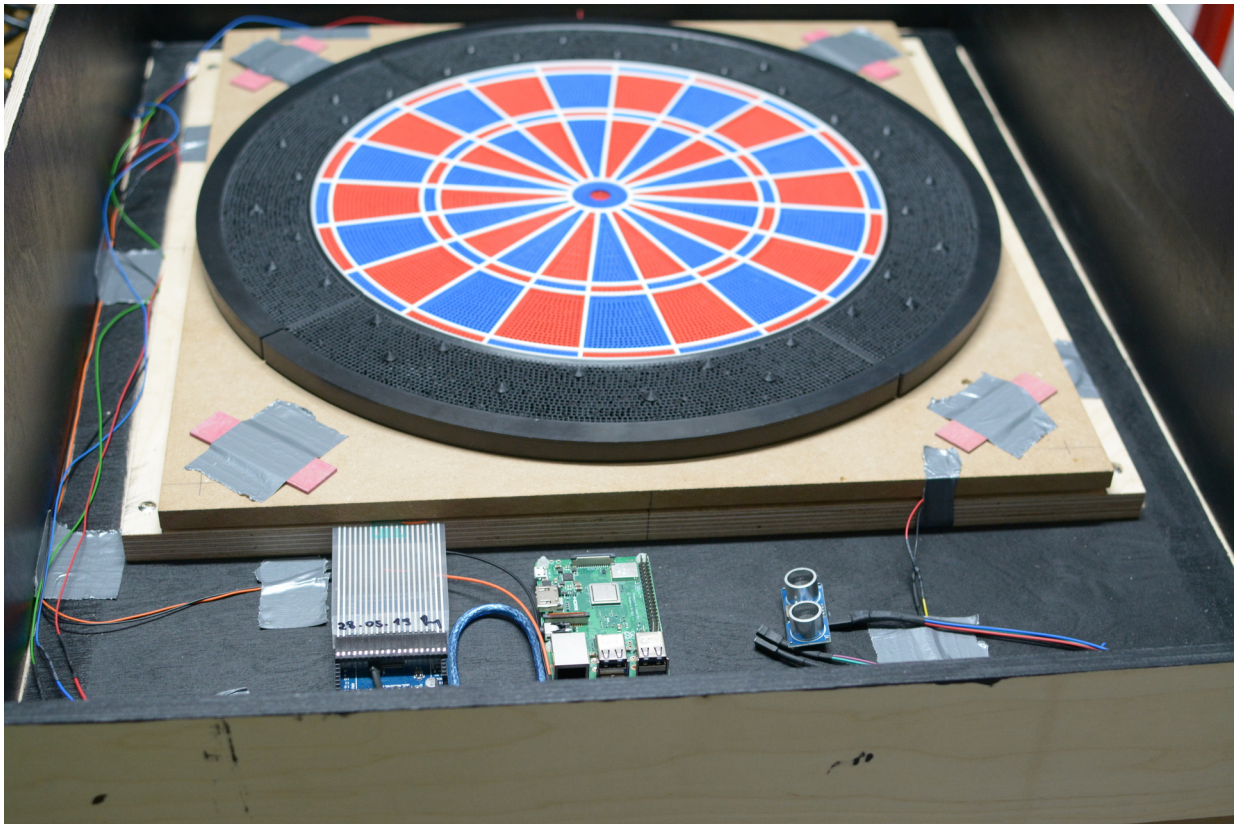


Fig. 27: Ultraschall Sensor ist in Position

Fig. 28: Nagel von Oben durch die Decke in die Türe

Fig. 29: Schloss unten links in der Türe

6.11 Schritt 11: Blenden herstellen und leimen

Die Stirnblenden habe ich aus Winkelholz geschnitten. Es hat die Maße 30mm x 30mm und ist 5mm dick. Ich habe die Blenden ebenfalls schwarz gestrichen und auf Gehrung geschnitten und eingepasst.

Anschließend werden die Leisten an die Seiten geleimt. Unter den Blenden werden später die LEDs angebracht.

Fig. 30: Blenden sind geleimt und mit Schraubzwingen fixiert

6.12 Schritt 12: Netzteil und LEDs

Die LEDs für unter die Blenden habe ich vorab abgelängt und mit ausreichend langen Kabeln versehen.

Die LEDs habe ich dann unter den Blenden aufgeklebt. Ich habe zusätzlich Holzleim verwendet, da mit solche LED Bänder meist abgefallen sind. Ich habe die LEDs außerdem vorübergehend mit Klebeband fixiert.

Die gelben und schwarzen Kabel am rechten Kastenrand sind die LED Kabel, die ich ebenfalls über den Deckel verlegt, im Gehäuse zusammengeführt und dann zum Netzteil nach unten verlegt habe.

Das Netzteil hat einen 12V und einen 5V Ausgang und liefert ausreichend Strom für die Technik und die LEDs.

Das Stromkabel wird durch ein Loch im Boden nach unten herausgeführt und kann bei Bedarf auch abgenommen und wieder eingesteckt werden.

Am Netzteil sind die LEDs (Gelb/Schwarz) und ein Micro-USB Stecker (Rot/Schwarz) für den Pi angelötet.

Fig. 31: Die Blenden sind fertig

Fig. 32: Die LEDs kleben unter den Blenden

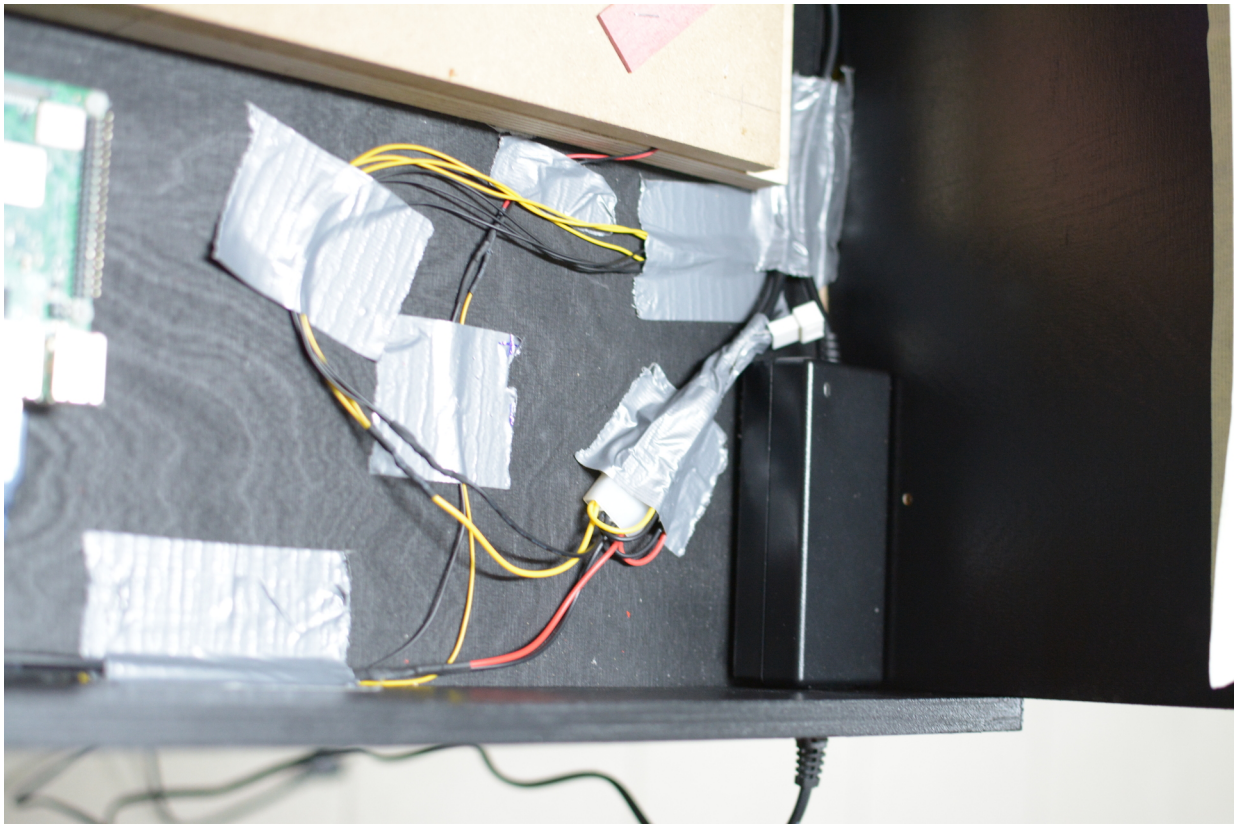


Fig. 33: Netzteil ist im Kasten fixiert

6.13 Schritt 13: Zahlen aufkleben

Zuerst werden alle Zahlen, die benötigt werden ausgeschnitten und ausgelegt. Hier gilt es besonders oft zu kontrollieren, ob die Reihenfolge stimmt.

Dann habe ich eine Schablone gefertigt. Diese wird mittig im Bulls Eye angebracht mithilfe eines Pfeils und zeigt die Mitte des Feldes an. So können die Zahlen besser ausgerichtet werden.

Nun ist Geduld und ein gutes Auge gefragt. Abschließend hat man es geschafft und alle Zahlen sind aufgeklebt.

6.14 Schritt 14: Platine löten und einbauen

Die Platine habe ich selbst designed und dann online bestellt. Man muss praktisch nur noch Stiftleisten nach unten gerichtet auflöten für die Steckverbindung zum Arduino.

Oben lötet man abgewinkelte Steckleisten auf, für die Steckerverbindungen zu Sensoren, Matrix und Knopf. Außerdem muss man noch zwei 1 MOhm Widerstände für die Piezos einlöten.

Außerdem habe ich die abgewinkelten Steckleisten in einem Winkel angebracht, sodass die Stecker später besser draufpassen.

Dann kann alles zusammengesteckt werden und die Funktion getestet werden.

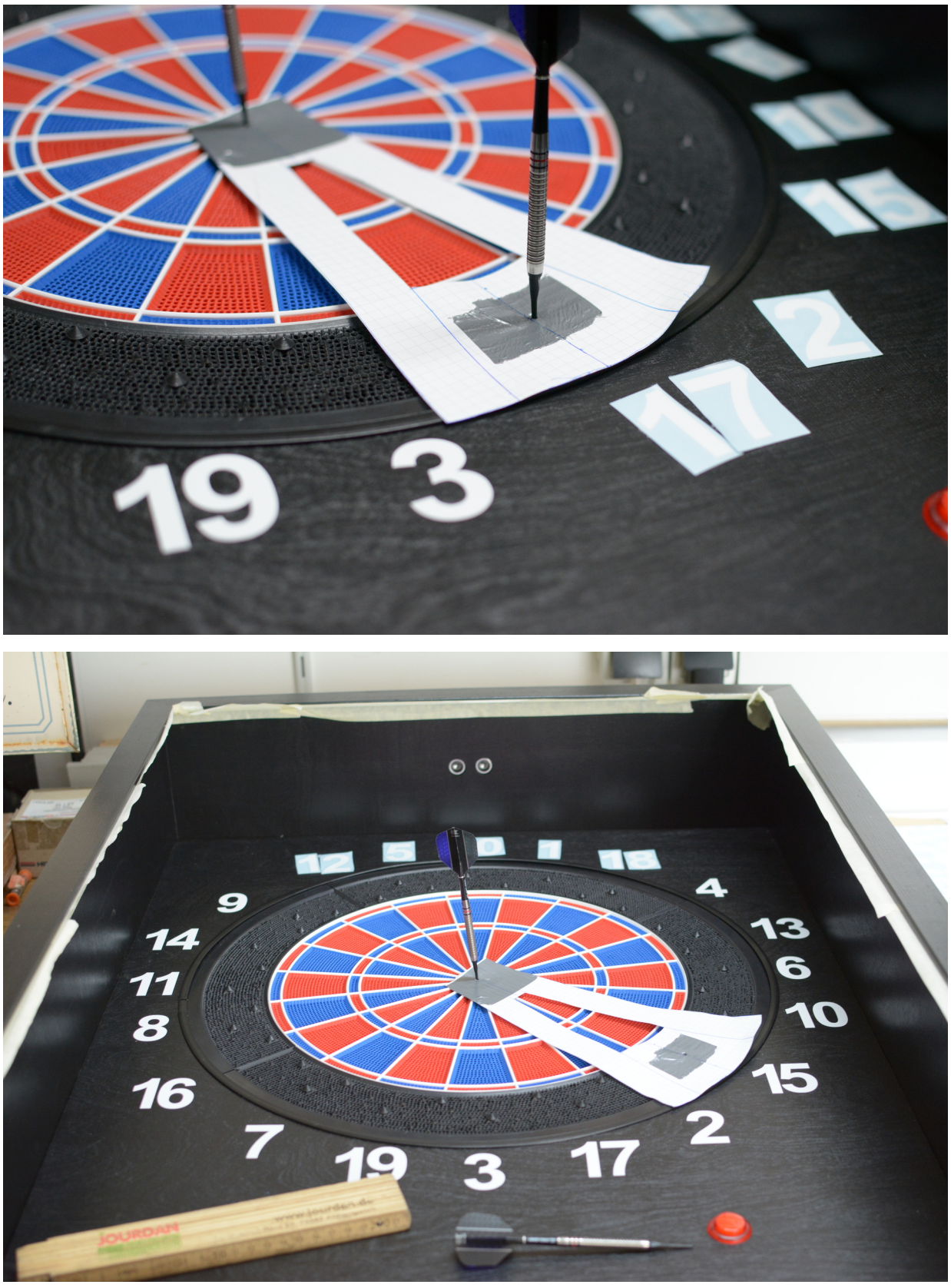


Fig. 34: Fast geschafft!!

Fig. 35: Alle Zahlen sind aufgeklebt und LEDs funktionieren

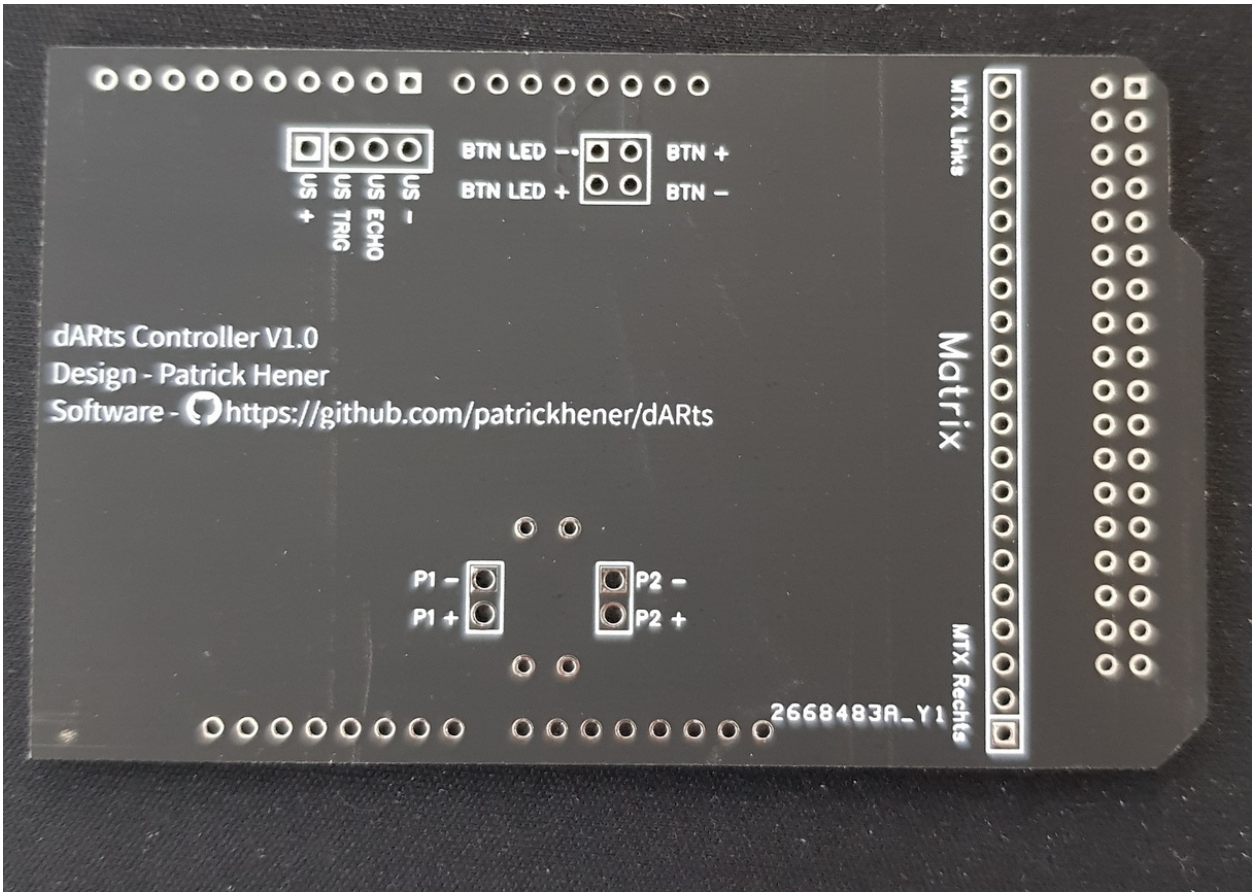


Fig. 36: Platine ungelötet Vorderseite

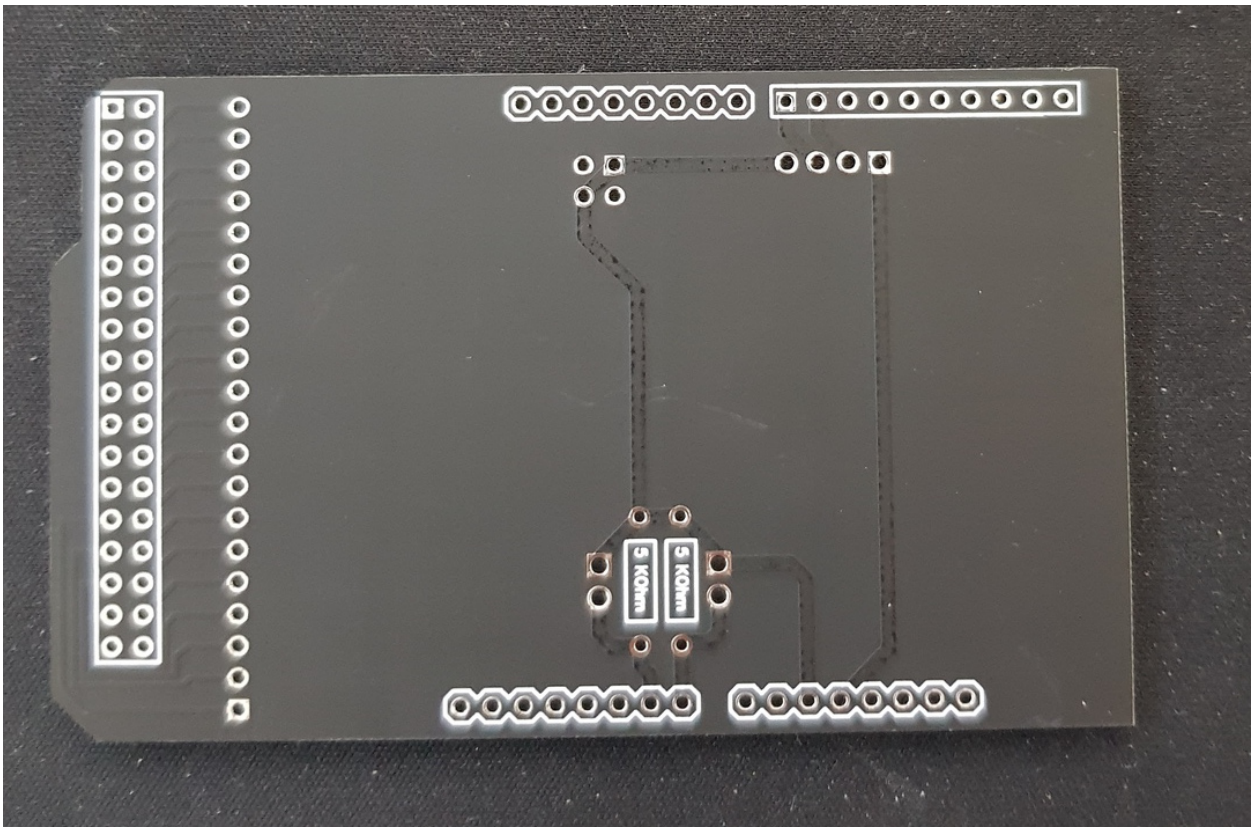


Fig. 37: Platine ungelötet Rückseite

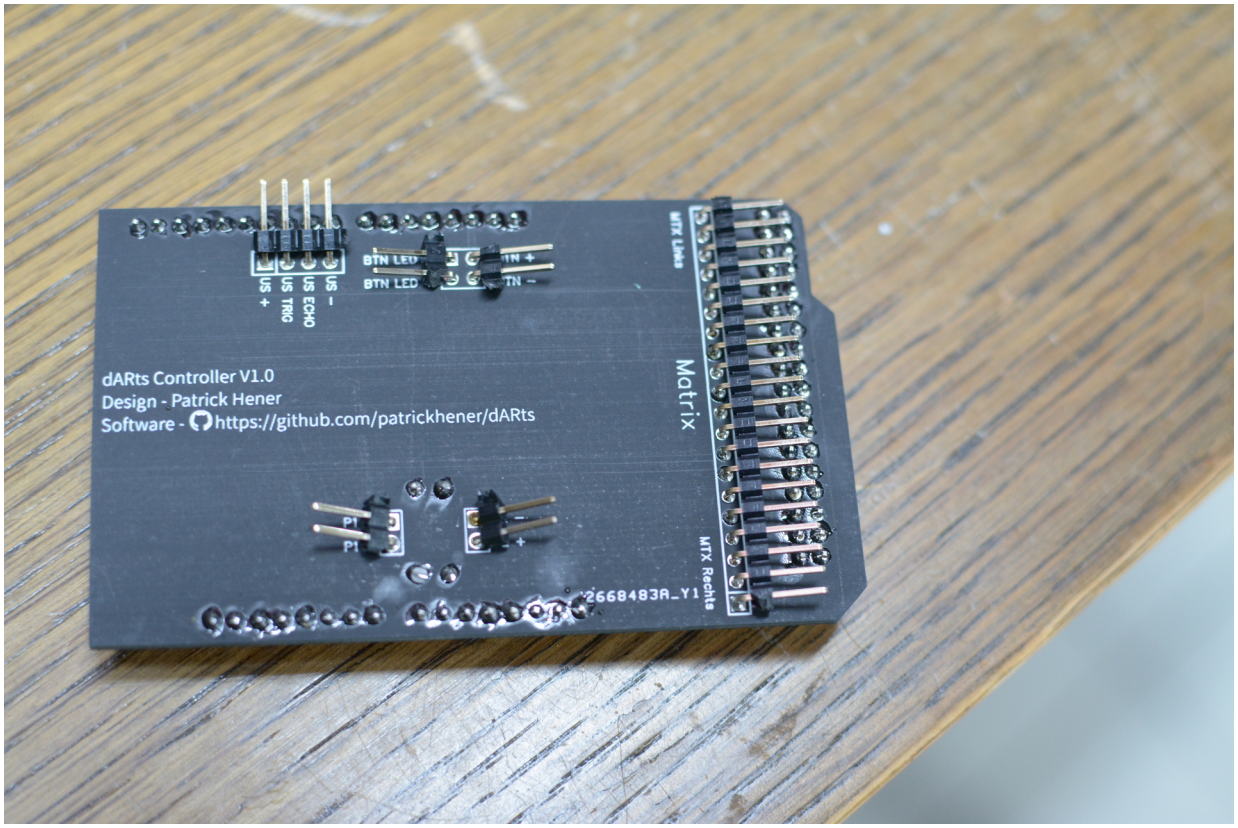


Fig. 38: Fertige Platine von oben

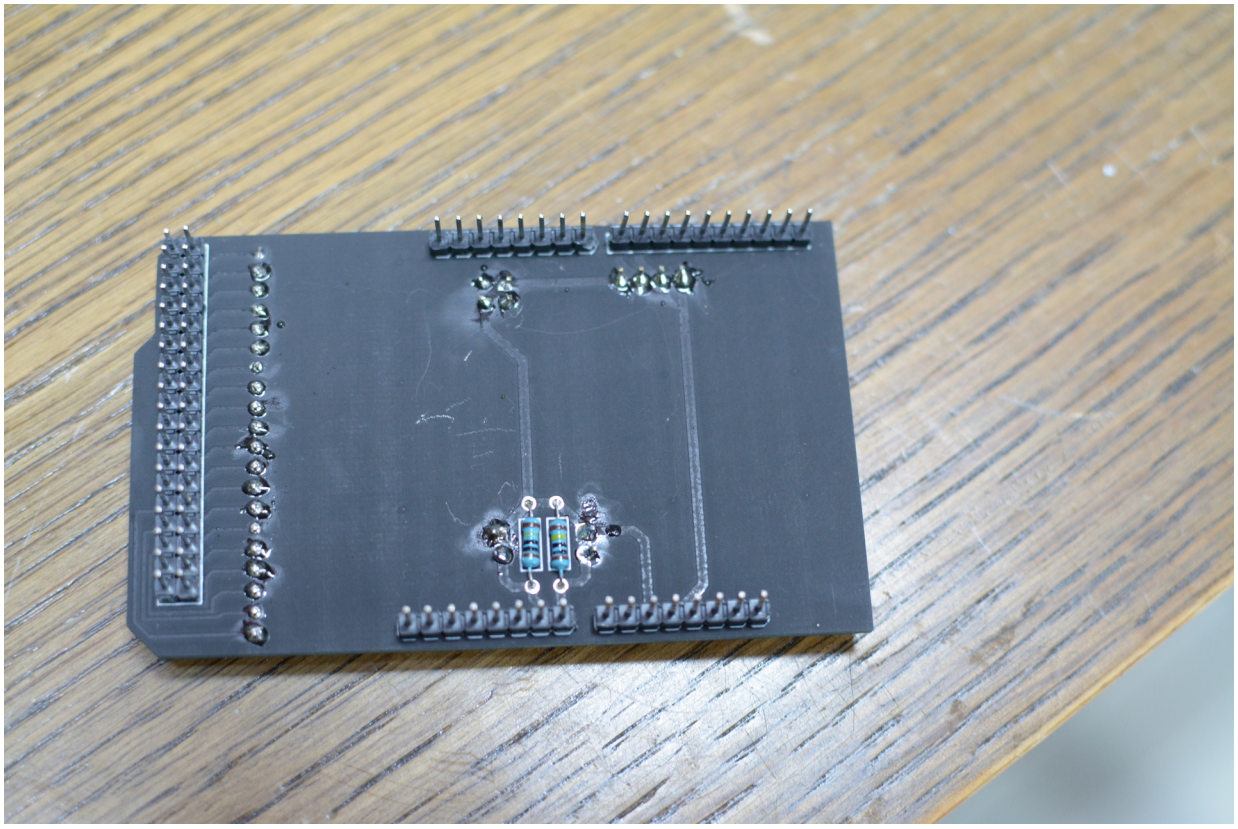
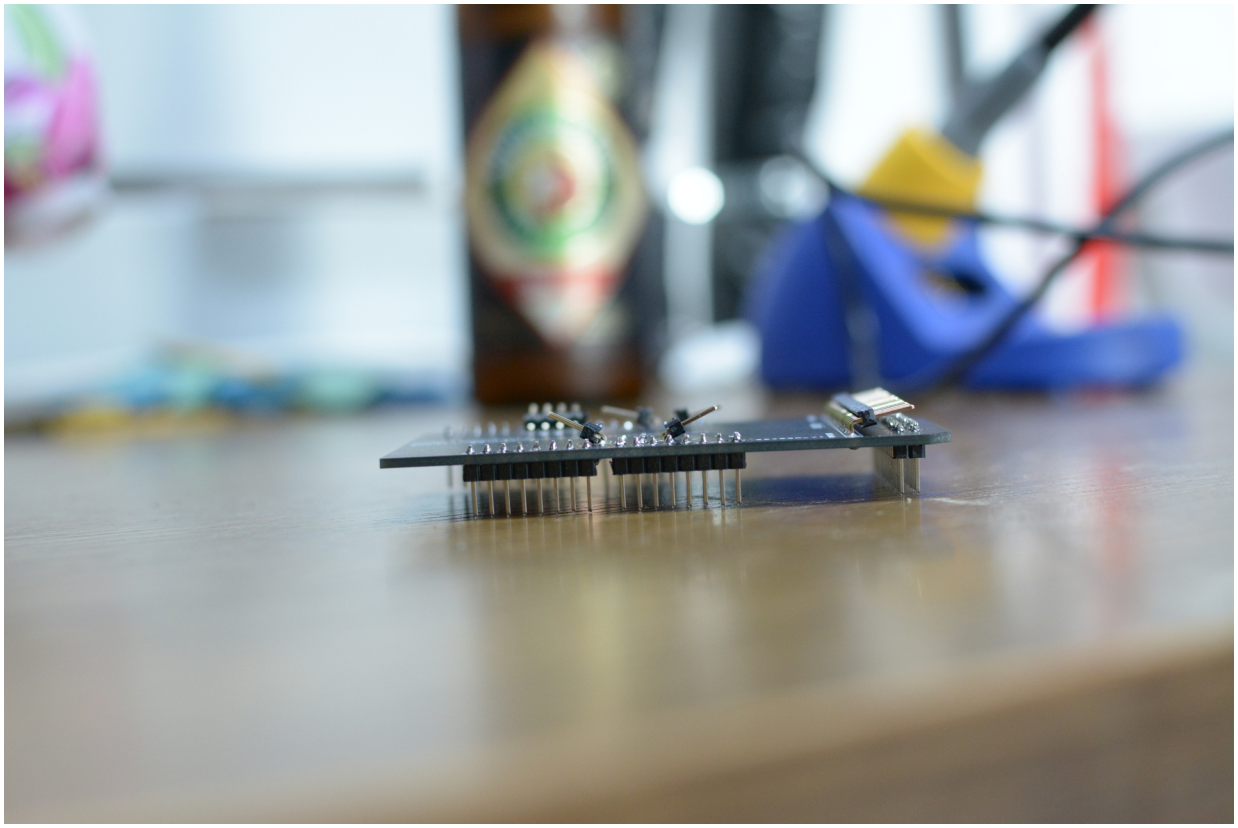


Fig. 39: Fertige Platine von unten



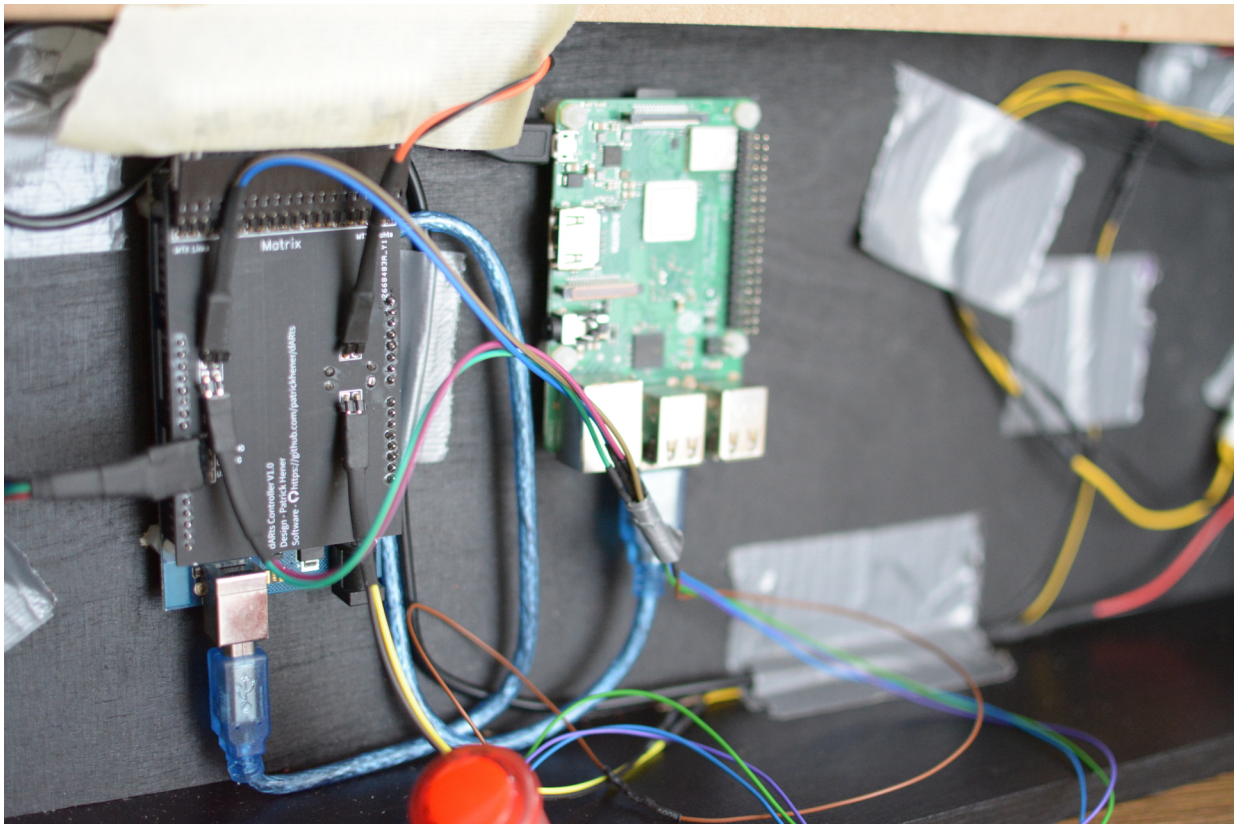


Fig. 40: Die Platine ist eingebaut

6.15 Schritt 15: Abschlusstests

Für den Abschlusstest verwende ich mein Dart-O-Mat 3000 Scoreboard und ein Tablet als Anzeige. Das Scoreboard läuft mit auf dem Pi. Ich habe den Pi so konfiguriert, dass er ein eigenes WLAN ausstrahlt, mit dem man das Tablet zur Anzeige verbinden kann.

Nun müssen alle Funktionen getestet werden (Fehlwurf, Wurferkennung, Knopf, Ultraschallsensor, . . .)

6.16 Abschließende Demo

Hier nun noch die passenden Demo Videos.

6.16.1 Spiel einstellen

6.16.2 301

6.16.3 Cricket

CHAPTER 7

Arduino Sketch

Nachfolgend wird der verwendete Arduino sketch beschrieben

7.1 Der Sketch

```
/************************/
/*        PINS          */
/************************/
// Ultraschall
const int echoPin = 8;
const int trigPin = 9;
// Piezo
const int piezoPin[2] = { A0, A1 };
// Button
const int buttonPin = 2;
const int buttonLedPin = 3;

// Matrix
// Output Pins
const int PO_[4] = { 22, 24, 49, 47 };
// Input Pins
const int PI_[16] = { 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 53, 51 }
↪;

/************************/
/*        VARIABLEN     */
/************************/
// Matrix Werte
// 120 = single 20, 220 = double 20, 320 = triple 20
// 125 = single Bull, 225 = doppel bull
// 4x16 = 64, aber nur 62 Zahlenwerte, deshalb kommt zweimal 000 vor
// usw..
const int FAKTOR_WERTE[4][16]={
```

(continues on next page)

(continued from previous page)

```
        {212, 112, 209, 109, 214, 114, 211, 111, 208, 108, 000, 312, 309, 314, 311,␣
↪308},
        {216, 116, 207, 107, 219, 119, 203, 103, 217, 117, 225, 316, 307, 319, 303,␣
↪317},
        {202, 102, 215, 115, 210, 110, 206, 106, 213, 113, 125, 302, 315, 310, 306,␣
↪313},
        {204, 104, 218, 118, 201, 101, 220, 120, 205, 105, 000, 304, 318, 301, 320,␣
↪305}
};

bool bI[4][16];
bool bHitDetected = false;
const int iHitPause = 300;
unsigned int iLastThrow = 0;

// Piezo
int wert[2] = { 0, 0 };
int schwelle = 20;
bool bFehlwurf = false;
// Button
int buttonState = 0;
// Ultraschall
long duration;
int distance;
int ultraschwelle = 0;
bool bUsAn = false;
bool bBewegungErkannt = false;
bool bSchwelleDefiniert = false;
// Input String von PI zu Arduino
String inputString = "";
boolean stringComplete = false;
//
int wurfanzahl = 0;

/***********************/
/*       FUNKTIONEN    */
/***********************/
void Ultraschall() {
        /* Ultraschall */
        // trigPin frei machen
        digitalWrite(trigPin, LOW);
        delayMicroseconds(2);
        // trigPin an für 10 micro secs
        digitalWrite(trigPin, HIGH);
        delayMicroseconds(10);
        digitalWrite(trigPin, LOW);
        // Lesen des Echo Pins
        duration = pulseIn(echoPin, HIGH);
        // Berechnung der Entfernung
        distance = duration*0.034/2;
        // Prüfen ob sich jemand genähert hat
        if (distance > ultraschwelle) {
                // Bislang ausgabe Seriell
                Serial.println("PFEILE");
        }
}
```

(continues on next page)

(continued from previous page)

```
int UltraschallMessen() {
        /* Ultraschall */
        // trigPin frei machen
        digitalWrite(trigPin, LOW);
        delayMicroseconds(2);
        // trigPin an für 10 micro secs
        digitalWrite(trigPin, HIGH);
        delayMicroseconds(10);
        digitalWrite(trigPin, LOW);
        // Lesen des Echo Pins
        duration = pulseIn(echoPin, HIGH);
        // Berechnung der Entfernung
        distance = duration*0.034/2;
        // Delay
        delay(100);
        // Entfernung zurückgeben
        return distance;
}

void checkButton() {
        /* Button */
        // Button einlesen
        buttonState = digitalRead(buttonPin);
        // Wenn gedrückt Ausgabe
        if (buttonState == LOW) {
                Serial.println("KNOPF");
                delay(500);
        }
}


void WurfSchicken(int x, int y) {
        Serial.println(FAKTOR_WERTE[x][y]);
        delay(iHitPause);
        bHitDetected = true;
        // Zur Sicherheit - Bulls auf NULL
        bI[1][10] = 1;
        bI[2][10] = 1;
}

void WurfAuswerten() {
        /* Matrix */
        bHitDetected = false;

        // Reihe 0-3 auslesen in Bool Array
        for (int x=0; x<4; x++) {
                digitalWrite(PO_[0], HIGH);
                digitalWrite(PO_[1], HIGH);
                digitalWrite(PO_[2], HIGH);
                digitalWrite(PO_[3], HIGH);
                digitalWrite(PO_[x], LOW);

                for (int y=0; y<16; y++) {
                        bI[x][y]  = digitalRead(PI_[y]);
                        // Wenn getroffen dann
                        if (bI[x][y] == 0) {
                                WurfSchicken(x, y);
```

(continues on next page)

(continued from previous page)

```
                        }
                }
        }
}

void FehlwurfErkennen() {
        /* Piezo */
        // Fehlwurf auf false setzen
        bFehlwurf = false;

        // Piezos in Schleife auslesen
        for (int i = 0; i < 2; i++) {
                // Zweifach auslesen um Störungen zu minimieren
                wert[i] = analogRead(piezoPin[i]);
                wert[i] = analogRead(piezoPin[i]);

                if (wert[i] >= schwelle) {
                        bFehlwurf = true;
                }
        }

        // Wenn kein Wurf erkannt wurde ist es ein Fehlwurf
        if (!bHitDetected) {
                if (bFehlwurf) {
                        // Bislang nur Ausgabe
                        Serial.println("FEHLWURF");
                        delay(200);
                }
        }
}

void Blinken(int Anzahl) {
        for (int i=0; i<Anzahl; i++) {
                digitalWrite(buttonLedPin, HIGH);
                delay(250);
                digitalWrite(buttonLedPin, LOW);
                delay(250);
        }
}

void SchwelleDefinieren() {
        int abstand = UltraschallMessen();
        ultraschwelle = abstand + 3;
        bSchwelleDefiniert = true;
}

void ReagiereAufSerialString() {
        if ( inputString.indexOf("BAN") != -1) {
                digitalWrite(buttonLedPin, HIGH);
                bUsAn = true;
                SchwelleDefinieren();
        } else if (inputString.indexOf("BAUS") != -1) {
                digitalWrite(buttonLedPin, LOW);
                bUsAn = false;
                bBewegungErkannt = false;
                bSchwelleDefiniert = false;
        } else if (inputString.indexOf("PERK") != -1) {
```

(continues on next page)

(continued from previous page)

```
                bBewegungErkannt = true;
        } else if (inputString.indexOf("NEXT") != -1) {
                bSchwelleDefiniert = false;
        }
        inputString = "";
        stringComplete = false;
}

/**********************/
/* Die einzelnen Loops */
/**********************/

/* Setup loop */
void setup() {
        // Pins für Ultraschall
        pinMode(trigPin, OUTPUT);
        pinMode(echoPin, INPUT);
        // Pin für Button
        pinMode(buttonPin, INPUT_PULLUP);
        // Pin für Button LED
        pinMode(buttonLedPin, OUTPUT);
        digitalWrite(buttonLedPin, LOW);
        // Matrix
        for (int i=0; i<4; i++) {
                pinMode(PO_[i], OUTPUT);
        }

        for (int i=0; i<16; i++) {
                pinMode(PI_[i], INPUT_PULLUP);
        }

        // Button blinken lassen
        Blinken(3);
        // Beginne serielle Übertragung an PC
        Serial.begin(9600);
}

/* Main loop */
void loop() {
        if (!bUsAn) {
                WurfAuswerten();
                FehlwurfErkennen();
                checkButton();
        } else if (bUsAn) {
                if (bSchwelleDefiniert) {
                        checkButton();
                        if (!bBewegungErkannt) {
                                Ultraschall();
                        } else {
                                Blinken(1);
                        }
                } else {
                        SchwelleDefinieren();
                }
        } else {
                Serial.println("Error in Main Loop");
        }
```

(continues on next page)

(continued from previous page)

```
        if(stringComplete)
                ReagiereAufSerialString();
}

/* Für die Kommunikation von PI zu Arduino */
void serialEvent() {
  while (Serial.available()) {
    char inChar = (char)Serial.read();
    inputString += inChar;
    if (inChar == '\n') {
      stringComplete = true;
    }
  }
}
```

7.2 PINS

In diesem Abschnitt des Sketches werden die Pin Definitionen gemacht. Wie unter *Pinbelegung* zu erkennen ist definiere ich hier Die Ultraschall Pins (echo = 8, trigger = 9), die PiezoPins A0 und A1 als array, den Button Pin 2 und den Button LED Pin 3, sowie die Output und Input Pins der Matrix.

7.3 VARIABLEN

In diesem Abschnitt des Sketches werden unterschiedliche Variablen definiert. Sicherlich am interessantesten ist die Definition der Wertematrix für das Auslesen des Matrix als zweidimensionales Array.

7.4 FUNKTIONEN

Nachfolgend werden die Funktionen des Sketches beschrieben.

7.4.1 Ultraschall()

In dieser Funktion wird der Ultraschall Sensor ausgelesen. Zuerst wird der Sensor durch aus- und anschalten frei gemacht, dann wird eine Dauer des Echos ermittelt und so verrechnet, dass sich eine Abstandseinheit in cm ergibt. Unterschreitet dieser Abstand eine Schwelle, so hat der Spieler seine Pfeile geholt und auf der seriellen Konsole wird das Wort “PFEILE” geschrieben. Danach erfolgt eine Wartezeit.

7.4.2 checkButton()

In dieser Funktion wird der Knopf ausgelesen. Wird er gedrückt wird das Wort “KNOPF” auf der seriellen Konsole geschrieben. Es erfolgt eine Wartezeit.

7.4.3 WurfSchicken(int x, int y)

In dieser Funktion wird der ermittelte Wurf auf die serielle Konsole geschrieben. Der Funktion wird ein Wert des zweidimensionalen Matrixarrays übergeben. Dieser wird dann im Array nachgeschaut und auf die Konsole geschrieben. Außerdem wird die Variable bHitDetected (also Wurf erkannt) auf true gesetzt. Im Falle, dass ein Piezo ausgelöst wurde wird so verhindert, dass ein Fehlwurf anstatt des erkannten Wurfwertes gebucht wird.

7.4.4 WurfAuswerten()

In dieser Funktion wird die Matrix des Automaten ausgelesen. Hierbei wird nacheinander auf den Output Pins der Matrix ein Signal gegeben und dann auf jedem Input Pin der Matrix geprüft ob der Kontakt geschlossen ist. Ist dies der Fall wird genau dieser Wert des zweidimensionalen Matrixarrays an die Funktion WurfSchicken(int x, int y) übergeben.

7.4.5 FehlwurfErkennen()

In dieser Funktion werden die Piezos ausgelesen und evaluiert, ob es sich um einen Fehlwurf handelt. Dafür wird nacheinander jeder Piezo zweimal in Folge ausgelesen und wenn der ausgelesene Wert den Schwellwert übersteigt, so wird die Variable bFehlwurf auf true gesetzt. Anschließend wird geprüft, ob ein Wurf erkannt wurde anhand des Status der Variablen bHitDetected. Ist dieser false, also es wurde kein Wurf erkannt, aber bFehlwurf ist true, also ein Fehlwurf wurde erkannt so wird das Wort “FEHLWURF” auf die serielle Konsole geschrieben.

7.4.6 Blinken(int Anzahl)

In dieser Funktion kann der Knopf zum blinken gebracht werden. Der Funktion wird eine Anzahl für das Blinken übergeben. Dann blinkt der Knopf, indem er an- und wieder ausgeschaltet wird.

7.4.7 ReagiereAufSerialString()

In dieser Funktion können Befehle verarbeitet werden, die dem Arduino auf die serielle Konsole geschrieben werden. Aktuell wird geprüft, ob jemand “BAN”, “BAUS” oder “BB” schreibt. Die Abkürzungen stehen für “Button AN”, also Knopf anschalten, “Button AUS”, also Knopf ausschalten oder “Button BLINKEN”, also einmal blinken lassen mithilfe der Funktion Blinken(int Anzahl).

7.5 LOOPS

Nachfolgend werden die Loops des Programms beschrieben

7.5.1 Setup

Der Setup Loop wird immer beim Boot des Arduino ausgeführt, einmalig. Hier werden die einzelnen Pins als Input oder Output Pin definiert. Außerdem blinkt der Knopf dreimal mithilfe der Funktion Blinken(int Anzahl) und eine serielle Übertragung wird gestartet. Diese Übertragung wird vom *Python Koppler* ausgelesen und behandelt.

7.5.2 Main

Der Main Loop bildet praktisch einen cycle des Prozessors ab. Dieser Loop läuft unendlich lange und wird immer wieder wiederholt. Es werden hier nacheinander die Funktionen Ultraschall(), checkButton(), WurfAuswerten()und FehlwurfErkennen() ausgeführt. Anschließend wird noch geprüft, ob durch den Loop serialEvent() ein fertige String empfangen wurde. Wenn ja wird er mithilfe der Funktion ReagiereAufSerialString() ausgewertet und behandelt. Danach startet der cycle von vorne. Ein cycle benötigt ohne Wartezeiten der Funktionen etwa 20 Millisekunden.

7.5.3 serialEvent

In diesem Loop wird überprüft, ob auf der seriellen Konsole ein String empfangen wurde und ob dieser schon zuende geschrieben wurde (erkennbar an einem Zeilenumbruch). Dieser string wird dann in der Funktion ReagiereAufSerial-String() behandelt.

CHAPTER 8

Python Koppler

In diesem Kapitel wird der verwendete Python Koppler beschrieben

8.1 Programmcode

```
#!/usr/bin/env python3

# Imports
import serial
import time
import yaml
import urllib.request
import urllib.parse
import sys
import logging
import os
import signal
import datetime

# Tracebacks unterdrücken
#  sys.tracebacklimit = 0

# Setup Logging
#  if os.path.isfile("dARts.log"):
    #  os.remove("dARts.log")
logging.basicConfig(filename='dARts.log', level=logging.DEBUG, format='%(asctime)s -
↪%(levelname)s - %(message)s')
logging.info("Start der Applikation")

#  systemd stuff
def signal_handler(signal, frame):
    logging.info("Beende Programm!")
    ser.close()
```

(continues on next page)

(continued from previous page)

```
    logging.info("Serielle Kommunikation gestoppt")
    logging.info("ENDE")
    sys.exit(0)

signal.signal(signal.SIGTERM, signal_handler)
signal.signal(signal.SIGINT, signal_handler)

# Config lesen
try:
    with open ("config.yml", 'r') as ymlconfig:
        cfg = yaml.load(ymlconfig, Loader=yaml.BaseLoader)
except:
    logging.error("Keine Konfigurationsdatei gefunden!")
    sys.exit("ERROR: Keine Konfigurationsdatei gefunden!")

# Scoreboard variablen aus Config
if cfg:
    host = cfg['scoreboard']['host']
    port = cfg['scoreboard']['port']

    # General variablen aus Config
    pfeilzeit = cfg['general']['pfeilzeit']
    serialport = cfg['general']['serial']

    # Serielle Konfiguration
    ser = serial.Serial()
    ser.port = serialport
    ser.baudrate = 9600
    ser.timeout = 1

# Port öffnen
try:
    ser.open()
    logging.info("Serielle Kommunikation gestartet")
except(serial.serialutil.SerialException):
    logging.error("Serielle Kommunikation nicht möglich!")
    sys.exit("ERROR: Serielle Kommunikation nicht möglich!")

# Variablen
matrix_dict = {
    '120': '/20/1',
    '220': '/20/2',
    '320': '/20/3',
    '101': '/1/1',
    '201': '/1/2',
    '301': '/1/3',
    '118': '/18/1',
    '218': '/18/2',
    '318': '/18/3',
    '104': '/4/1',
    '204': '/4/2',
    '304': '/4/3',
    '113': '/13/1',
    '213': '/13/2',
    '313': '/13/3',
    '106': '/6/1',
    '206': '/6/2',
```

(continues on next page)

(continued from previous page)

```
    '306': '/6/3',
    '110': '/10/1',
    '210': '/10/2',
    '310': '/10/3',
    '115': '/15/1',
    '215': '/15/2',
    '315': '/15/3',
    '102': '/2/1',
    '202': '/2/2',
    '302': '/2/3',
    '117': '/17/1',
    '217': '/17/2',
    '317': '/17/3',
    '103': '/3/1',
    '203': '/3/2',
    '303': '/3/3',
    '119': '/19/1',
    '219': '/19/2',
    '319': '/19/3',
    '107': '/7/1',
    '207': '/7/2',
    '307': '/7/3',
    '116': '/16/1',
    '216': '/16/2',
    '316': '/16/3',
    '108': '/8/1',
    '208': '/8/2',
    '308': '/8/3',
    '111': '/11/1',
    '211': '/11/2',
    '311': '/11/3',
    '114': '/14/1',
    '214': '/14/2',
    '314': '/14/3',
    '109': '/9/1',
    '209': '/9/2',
    '309': '/9/3',
    '112': '/12/1',
    '212': '/12/2',
    '312': '/12/3',
    '105': '/5/1',
    '205': '/5/2',
    '305': '/5/3',
    '125': '/25/1',
    '225': '/25/2',
}

valide_wurfzaehler = ["0", "1", "2", "3"]

global pfeile_holen
global knopf_an
global pfeile_abgezogen
global won
global last_throw
global last_hit_time
global stuck_threshold
```

(continues on next page)

(continued from previous page)

```
pfeile_holen = False
knopf_an = False
pfeile_abgezogen = True
won = False
last_throw = "0"
last_hit_time = None
stuck_threshold = datetime.timedelta(0,0,500000)

# Funktionen
def makeRequest(urlpart):
    try:
        url = host + ":" + port + "/game/throw" + urlpart
        response = urllib.request.urlopen(url)
        response_text = response.read().decode('utf-8')
        if "Dart" in response_text:
            set_pfeile_holen(True)
            button_on()
        if "Sieger" in response_text:
            set_won(True)
            button_on()
        if "Winner" in response_text:
            set_won(True)
            button_on()
        logging.info("SCOREBOARDANTWORT: {}".format(response_text))
        return response_text
    except:
        logging.error("Exception in makeRequest")


def requestRematch():
    try:
        url = host + ":" + port + "/game/rematch"
        response = urllib.request.urlopen(url)
        response_text = response.read().decode('utf-8')
        logging.info("SCOREBOARDANTWORT: {}".format(response_text))
        button_off()
        set_won(False)
        return response_text
    except:
        logging.error("Exception in requestRematch()")


def requestStuck():
    try:
        url = host + ":" + port + "/game/stuck"
        response = urllib.request.urlopen(url)
        response_text = response.read().decode('utf-8')
        logging.info("SCOREBOARDANTWORT: {}".format(response_text))
        button_on()
        time.sleep(.25)
        button_off()
        time.sleep(.25)
        return response_text
    except:
        logging.error("Exception ind requestStuck()")
```

(continues on next page)

(continued from previous page)

```
def nextPlayer():
    try:
        url = host + ":" + port + "/game/nextPlayer"
        response = urllib.request.urlopen(url)
        response_text = response.read().decode('utf-8')
        if "Dart" in response_text:
            set_pfeile_holen(True)
            button_on()
        outputString = "NEXT\n"
        ser.write(outputString.encode('utf-8'))
        logging.info("SCOREBOARDANTWORT: {}".format(response_text))
        return response_text
    except:
        logging.error("Exception in nextPlayer()")


def get_wurfzaehler():
    try:
        url = host + ":" + port + "/game/getThrowcount"
        response = urllib.request.urlopen(url)
        response_text = response.read().decode('utf-8')
        global pfeile_holen
        if not pfeile_holen:
            check_button_on(response_text)
        return response_text
    except:
        logging.error("Exception in get_Wurfzaehler()")


def check_button_on(wurfzaehler):
    try:
        global knopf_an
        if wurfzaehler == "2":
            if not knopf_an:
                button_on()
        else:
            if knopf_an:
                button_off()
    except:
        logging.error("Exception in check_button_on()")


def button_on():
    global knopf_an
    outputString = "BAN\n"
    ser.write(outputString.encode('utf-8'))
    knopf_an = True


def button_off():
    global knopf_an
    outputString = "BAUS\n"
    ser.write(outputString.encode('utf-8'))
    knopf_an = False


def read_serial():
```

(continues on next page)

(continued from previous page)

```
    string = ser.readline()
    if string:
        string = string[:-2]
        string = string.decode()
        return string
    else:
        return ""


def get_pfeile_holen():
    global pfeile_holen
    return pfeile_holen


def set_pfeile_holen(status):
    global pfeile_holen
    pfeile_holen = status
    return get_pfeile_holen()


def get_pfeile_abgezogen():
    global pfeile_abgezogen
    return pfeile_abgezogen


def set_pfeile_abgezogen(status):
    global pfeile_abgezogen
    pfeile_abgezogen = status
    return get_pfeile_abgezogen()


def get_won():
    global won
    return won


def set_won(status):
    global won
    won = status
    return get_won()


def get_last_throw():
    global last_throw
    return last_throw


def set_last_throw(string):
    global last_throw
    last_throw = string
    return get_last_throw()


def get_last_hit_time():
    global last_hit_time
    return last_hit_time
```

(continues on next page)

(continued from previous page)

```
def set_last_hit_time(time):
    global last_hit_time
    last_hit_time = time
    return get_last_hit_time()


def main():
    if get_wurfzaehler() == "3":
        set_pfeile_holen(True)
        set_pfeile_abgezogen(False)
        button_on()

    while True:
        string = read_serial()
        if string:
            current_time = datetime.datetime.now()
            logging.info("string ist: {}".format(string))
            if string in matrix_dict:
                wurfzaehler = get_wurfzaehler()
                if not get_pfeile_holen():
                    if not wurfzaehler == "3":
                        if get_pfeile_abgezogen():
                            if string == get_last_throw():
                                # Zeit prüfen
                                global stuck_threshold
                                if (current_time - get_last_hit_time()) < stuck_
↪threshold:
                                    logging.error("Dart steckt fest")
                                    antwort = requestStuck()
                                else:
                                    antwort = makeRequest(matrix_dict[string])

                                set_last_hit_time(datetime.datetime.now())

                            else:
                                antwort = makeRequest(matrix_dict[string])
                                set_last_hit_time(datetime.datetime.now())

                            set_last_throw(string)

            elif string == "FEHLWURF":
                wurfzaehler = get_wurfzaehler()
                if not get_pfeile_holen():
                    if not wurfzaehler == "3":
                        if get_pfeile_abgezogen():
                            antwort = makeRequest('/0/1')

            elif string == "KNOPF":
                if get_won():
                    antwort = requestRematch()
                else:
                    wurfzaehler = get_wurfzaehler()
                    if wurfzaehler in valide_wurfzaehler:
                        if not get_pfeile_holen():
                            while not wurfzaehler == "3":
                                antwort = makeRequest('/0/1')
```

(continues on next page)

(continued from previous page)

```
                                wurfzaehler = get_wurfzaehler()
                                button_on()
                            set_pfeile_abgezogen(False)

                        else:
                            antwort = nextPlayer()
                            set_pfeile_holen(False)
                            set_pfeile_abgezogen(True)
                            button_off()

            elif string == "PFEILE":
                #wurfzaehler = get_wurfzaehler()
                #if get_pfeile_holen() and wurfzaehler == "3":
                if get_pfeile_holen():
                    outputString = "PERK\n"
                    ser.write(outputString.encode('utf-8'))
                    time.sleep(int(pfeilzeit))
                    antwort = nextPlayer()
                    set_pfeile_holen(False)
                    set_pfeile_abgezogen(True)
                    button_off()

# Main Loop
if __name__ == "__main__":
    main()
```

8.2 Imports

In diesem Abschnitt werden notewendige Imports gemacht. Darunter zum Beispiel die Programmbibliothek *serial*, die den Arduino via serieller Konsole ausliest.

8.3 Systemd stuff

Dieser Abschnitt ist notwendig, um den Programmcode sauber beenden zu können, wenn er als systemd Dienst verwendet wird. Andernfalls würde beim Stoppen des Dienstes die serielle Kommunikation nicht sauber beendet.

8.4 Config

In diesem Abschnitt wird die config.yml ausgelesen und interpretiert.

8.4.1 Scoreboard variablen

Die Variablen über den Host und den Port, wo der Dart-O-Mat 3000 zu finden sind werden hier gespeichert.

8.4.2 General variablen

Die Variablen über die Auszeit beim Pfeile Abziehen und den seriellen Port werden hier gespeichert.

8.4.3 Serielle Konfiguration

Es wird eine Instanz ser der Klasse serial.Serial() erzeugt und konfiguriert.

8.5 Ports öffnen

In diesem Abschnitt wird versucht die serielle Kommunikation mit dem Arduino zu starte. Schlägt das fehl wird der Code beendet mit einer Fehlermeldung.

8.6 Variablen

In diesem Abschnitt werden die notwendigen Variablen definiert, die zum Abprüfen von Zuständen verwendet werden. Außerdem werden die Daten aus dem Matrix Array in das API Format vom Dart-O-Mat 3000 übersetzt, sodass zum Beispiel der Wert 320 (Triple 20) einen URL Anteil von 20/3 (Triple 20) ergibt. Dieses Dicitionary wird später von der Funktion makeRequest(urlpart) verwendet.

8.7 Funktionen

8.7.1 makeRequest(urlpart)

Diese Funktion konstruiert aus der *host* und *port* Variablen, sowie dem übergebenen *urlpart* einen Request an das Scoreboard. Dann wird die Antwort ausgewertet. Enthält sie das Wort "Dart" wird die Variable *pfeile_holen* auf **True** gesetzt. Beim Wort "Sieger" oder "Winner" wird die Variable *won* auf **True** gesetzt. Außerdem wird in beiden Fällen das Licht am Knopf aktiviert, indem mithilfe der Funktion button_on() das Wort *BAN* auf die Konsole geschrieben wird.

Die aufgerufenen URL an der Scoreboard API lautet: http://IP:PORT/game/throw/Wurfwert(z.B. /20/3 für triple 20)

8.7.2 requestRematch()

Diese Funktion wird ausgeführt, wenn die Variable *won* auf **True** steht und der Knopf gedrückt wird. So kann schnell ein neues Match mit den gleichen Einstellungen gestartet werden.

Die aufgerufenen URL an der Scoreboard API lautet: http://IP:PORT/game/rematch

8.7.3 requestStuck()

Diese Funktion wird aufgerufen, wenn in kürzester Zeit mehrmals die gleichen Werte auf der Konsole empfangen werden. Hierbei darf man davon ausgehen, dass ein Dartpfeil stecken geblieben ist. Das Scoreboard wird audiovisuell reagieren, sodass der Spieler auf den steckengebliebenen Pfeil aufmerksam gemacht wird.

Die aufgerufenen URL an der Scoreboard API lautet: http://IP:PORT/game/stuck

8.7.4 nextPlayer()

Diese Funktion schaltet zum nächsten Spieler um.

Die aufgerufenen URL an der Scoreboard API lautet: http://IP:PORT/game/nextPlayer

8.7.5 get_wurfzaehler()

Diese Funktion ermittelt die Anzahl der Würfe, die ein Spieler in der aktuellen Wurfrunde (0-3) bereits gemacht hat. Die Wurfanzahl wird an mehreren Stellen ausgewertet und ist entscheiden dafür, wie sich der Python Koppler verhält. Außerdem wird über diese Funktion gesteuert, ob der Knopf angeschaltet werden muss, zum Beispiel wenn schon 3 Würfe gemacht wurden. Dies ist das Signal für den Spieler, dass er seine Pfeile holen muss.

Die aufgerufenen URL an der Scoreboard API lautet: http://IP:PORT/game/getThrowcount

8.7.6 check_button_on()

In dieser Funktion wird anhand der Anzahl der bereits geworfenen Darts entschieden, ob der Knopf aus oder an sein muss.

8.7.7 button_on()

Mithilfe dieser Funktion wird das Wort “BAN” auf die serielle Konsole geschrieben und der Arduino schaltet so die LED des Knopfs an.

8.7.8 button_off()

Mithilfe dieser Funktion wird das Wort “BAUS” auf die serielle Konsole geschrieben und der Arduino schaltet so die LED des Knopfs aus.

8.7.9 button_blinken()

Mithilfe dieser Funktion wird der Knopf zum Blinken gebracht

8.7.10 read_serial()

Diese Funktion liest einen String von der seriellen Konsole aus und formatiert ihn (Entfernt Zeilenumbrüche). Dann gibt sie den formatierten Wert zurück.

8.7.11 GET- und SET-Methoden

Die unterschiedlichen GET- und SET-Methoden sind dazu da die globalen Variablen für die Spielsteuerung zu schreiben oder auszulesen.

8.7.12 main() Als Hauptprogrammloop

Diese Funktion ist eine endlosschleife, wie im Arduino Sketch und steuert zyklisch die Kommunikation zwischen Arduino und dem Scoreboard Dart-O-Mat 3000.

Im Schritt 1 wird ein String von der Konsole empfangen mithilfe der Funktion read_serial(). Wenn ein String gelesen werden konnte wird die aktuelle Zeit als Zeitstempel ermittelt (Dient der Stuck Dart Erkennung).

Im Schritt 2 wird ausgewertet, ob der erkannte String im Dictionary der Dart Matrix steht. Wenn ja und weder *pfeile_holen* **True** ist oder der Wurfzähler 3 beträgt so wird anschließend auf Stuck Dart geprüft. Der Stuck Dart wird ermittelt, indem verglichen wird welches der letzte Wurfwert war. Ist es derselbe, wie der aktuelle Wurfwert wird das

Zeitdelta verglichen zwischen letztem Empfang des Wertes und dem aktuellen. Ist dieser kleiner als die Zeitschwelle (halbe Sekunde) wird von einem Stuck Dart ausgegangen und die Funktion requestStuck() aufgerufen. Andernfalls wird der Wert an das Scoreboard gesendet via makeRequest() und der Wurf wird verbucht.

Ist der String kein Wurfwert wird in Schritt 3 kontrolliert, ob es das Wort "FEHLWURF" ist. Wenn ja und werder *pfeile_holen* **True** ist noch der Wurfzähler 3 beträgt und die Variable *pfeile_abgezogen* **True** ist, so wird via makeRequest() der Wert 0, also Fehlwurf an das Scoreboard geschickt.

Ist der String kein Fehlwurf wird in Schritt 4 kontrolliert, ob es das Wort "KNOPF" ist. Wenn ja, wird geprüft, ob *won* auf **True** steht. Ist das der Fall wird via requestRematch() ein neues Match angefordert. Andernfalls wird der Wurfzähler geprüft. Abhängig davon welchen Wert er hat werden entweder die Würfe mit Fehlwürfen (makeRequest('/0/1')) aufgefüllt oder auf den nächsten Spieler gewechselt (nextPlayer()).

Ist der String nicht Knopf wird in Schritt 5 kontrolliert, ob es das Wort "PFEILE" ist. In diesem Fall und dem Fall, dass sowohl *pfeile_holen* **True** ist und der Wurfzähler 3 beträgt wird die Pfeileholzeit lang gewartet und dann auf den nächsten Spieler geschaltet.

Dann startet der cycle von vorne.

8.8 Logging

Alle Events werden zentral in der Datei dARts.log festgehalten. Man kann sich die Datei zu Debug zwecken also auf der Konsole des Pi's anschauen, indem man folgenden Aufruf im Ordner des Python Kopplers startet:

```
tail -f dARts.log
```

CHAPTER 9

Setup des Systems

In diesem Kapitel wird beschrieben, wie man das System aufsetzen kann

9.1 System Basissetup

Ich habe auf dem Raspberry Pi ein Arch Linux installiert. Eine gute Anleitung dazu findet man hier: . Es ist wichtig darauf zu achten, dass Ihr das 32bit Image installiert. Außerdem benötigt Ihr und solltet die installieren.

9.2 Arduino flashen

Aus Gründen der Bequemlichkeit und der Kompatibilität mit *vim*, den ich als Texteditor verwende nutze ich das *Makefile* aus Projekt. Man clont das Repo, wie in meinem Fall unter **~/.arduino_mk** und kann dann ein Makefile im Ordner mit dem Sketch verwenden. Dazu könnt Ihr einfach mein Makefile entsprechend auf eure Orderstruktur anpassen.

```
ARDUINO_DIR = /usr/share/arduino
BOARD_TAG = mega2560
include $(HOME)/.arduino_mk/Arduino.mk
```

Das ARDUINO_DIR gibt an, wo die IDE zu finden ist. Die include Zeile inkludiert das Makefile aus dem oben erwähnten Git Repo. Anschließend könnt Ihr mit drei Befehlen das Flashen des Arduinos steuern:

```
# Kompilieren des Codes
make

# Hochladen des Codes auf den Arduino
make upload

# Seriellen Monitor starten
make monitor
```

Und weil ich tippfaul bin habe ich mir alias Einträge in meinem profil hinterlegt: *m*, *mu* und *mm*.

9.3 Python Koppler als systemd Dienst

Der Python Koppler kann als systemd Dienst installiert werden, sodass man ihn beim Start des Pi's bequem automatisch starten lassen kann. Dazu habe ich im Ordner *python* ein service file inkludiert.

```
[Unit]
Description=dARts.py E-Dart Kopplungs service
After=multi-user.target

[Service]
Type=simple
WorkingDirectory=/home/patrick/projects/dARts/python
ExecStart=/usr/bin/python3 /home/patrick/projects/dARts/python/dARts.py

[Install]
WantedBy=multi-user.target
```

Wichtig ist, dass Ihr die Pfadangaben an eure Orderstruktur anpasst. Dann kopiert Ihr die Datei nach */home/<euer Benutzername>/.config/systemd/user/dARts.service*. So wird der Service als Benutzerservice zur Verfügung gestellt.

Anschließend könnt Ihr den Service wie jeden systemd Service nutzen:

```
# Service starten:
systemctl --user start dARts.service

# Service stoppen:
systemctl --user stop dARts.service

# Autostart bei boot:
systemctl --user enable dARts.service
```

9.4 Hostapd und Dnsmasq als WLAN Access Point

Den Pi habe ich zum WLAN-Hotspot gemacht. So kann eine Spielergruppe einfach ein Anzeigegerät für den Dart-O-Mat 3000 per WLAN einbuchen. Auch das Smartphone kann so als Game Controller einfach genutzt werden. Ich nutze dazu hostapd. Um IP Adressen und DNS kümmert sich bei mir Dnsmasq. Damit die Geräte nach dem Einwählen direkt umgeleitet werden (Captive Portal, wie zum Beispiel im Hotel) wird nginx eingesetzt. Eine gute Anleitung, wie man das einrichtet findet man beispielsweise .

Für mich haben folgende Konfigurationseinstellungen funktioniert.

9.4.1 hostapd

Die Datei kommt nach */etc/hostapd/hostapd.conf*

```
interface=wlan0
ssid=dARts
hw_mode=g
channel=6
auth_algs=1
wmm_enabled=0
```

Der Inhalt von */etc/default/hostapd* sollte so aussehen:

```
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

9.4.2 dnsmasq

Die Datei kommt nach */etc/dnsmasq.conf*

```
bogus-priv
server=/darts/10.41.18.20
local=/darts/
address=/#/10.41.18.20
interface=wlan0
domain=localnet
dhcp-range=10.41.18.50,10.41.18.200,24h
dhcp-option=3,10.41.18.20
dhcp-option=6,10.41.18.20
dhcp-authoritative
```

9.4.3 nginx

Die Datei kommt nach */etc/nginx/nginx.conf*

```
#user html;
worker_processes  1;

#error_log  logs/error.log;
#error_log  logs/error.log  notice;
#error_log  logs/error.log  info;

#pid        logs/nginx.pid;


events {
    worker_connections  1024;
}


http {
    include       mime.types;
    default_type  application/octet-stream;

    #log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
    #                  '$status $body_bytes_sent "$http_referer" '
    #                  '"$http_user_agent" "$http_x_forwarded_for"';

    #access_log  logs/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    #keepalive_timeout  0;
    keepalive_timeout  65;

    #gzip  on;
```

(continues on next page)

(continued from previous page)

```
    include /etc/nginx/sites-enabled/*;

}
```

Und die Datei kommt nach /etc/nginx/sites-available/hotspot.conf

```
server {
    # Listening on IP Address.
    # This is the website iptables redirects to
    listen       80 default_server;
    root         /usr/share/nginx/html/portal;

    # For others
    location / {
        return 302 http://dart-o-mat-3000.darts/;
    }
 }


server {
     listen       80;
     server_name  dart-o-mat-3000.darts;
     root         /usr/share/nginx/html/portal;

     location / {
         try_files $uri $uri/ /index.html;
     }
}
```

Die Hotspotseite wird aktiviert, indem man die Datei mit einem Symlink nach sites-enabled verlinkt:

```
ln -s /etc/nginx/sites-available/hotspot.conf /etc/nginx/sites-enabled/hotspot.conf
```

9.4.4 /etc/hosts

In der */etc/hosts* sollte noch folgender Eintrag unten angefügt werden.

```
10.41.18.20   dart-o-mat-3000.darts
```

9.4.5 Autostarts

Wenn die Dienste alle korrekt laufen kann man sie mit den folgenden Befehlen noch zum Autostart hinzufügen:

```
sudo systemctl enable hostapd
sudo systemctl enable dnsmasq
sudo systemctl enable nginx
```