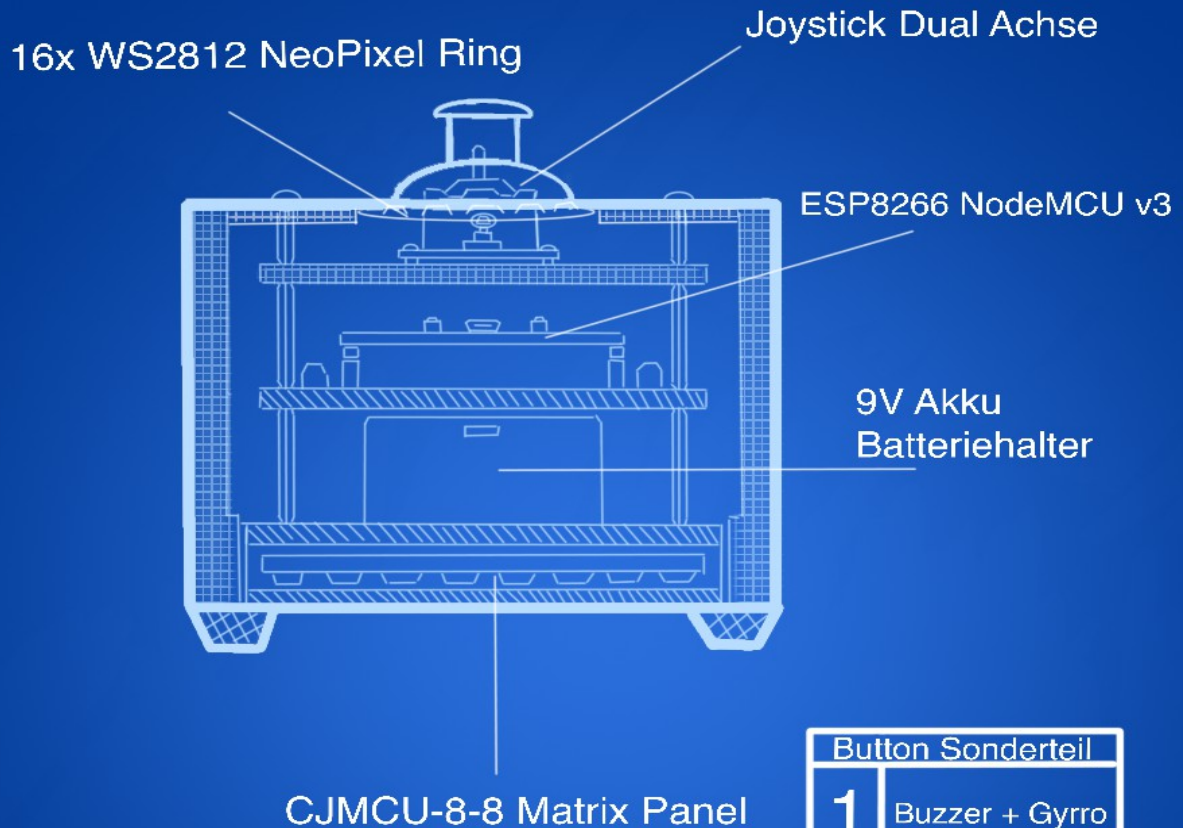


INTELLIGENZBOLZEN

Intelligenter Sport-Button mit Sensoren

Ein Projekt von Manuel und Marco



- Entwicklung der Buttons
- Auslesen der Sensoren
- Kommunikation der Buttons untereinander
- Verbindung der Buttons mit dem Handy
- Entwicklung der App
- Verbindung mit der Uhr

Button Sonderteil	
1	Buzzer + Gyro
2	ToF Sensor
3	
4	

TITEL	Intelligenter Button mit Sensoren	
Datum	03.01.2021	
Marco Selch	M. Selch	

Kurzfassung:

Zuwenig Sport während der Corona-Pandemie ist schlecht für den Körper? Richtig!
Deshalb haben wir uns eine Lösung für dieses Problem überlegt!
Ziel ist es ein System zum Erkennen von Sportübungen zu entwickeln.

Hierzu wird im Rahmen dieser Arbeit eine Handyapp erstellt, welche interaktiv dem Benutzer verschiedene Übungen anbietet und die richtige Durchführung überwacht.
Dabei sind nicht nur Sensoren zum sicheren Erkennen von Sportübungen wichtig, sondern auch eine attraktive App, um den Benutzer zu motivieren.
Zusätzlich soll unsere Erfindung auch in anderen Bereichen als Sport verwendbar sein.

Zentrale Aspekte unseres Projektes sind hierbei

- Entwicklung der Buttons
- Auslesen der Sensoren
- Kommunikation der Buttons untereinander
- Verbindung der Buttons mit dem Handy
- Entwicklung der App

Und letztlich die Verlinkung aller Teilgebiete.

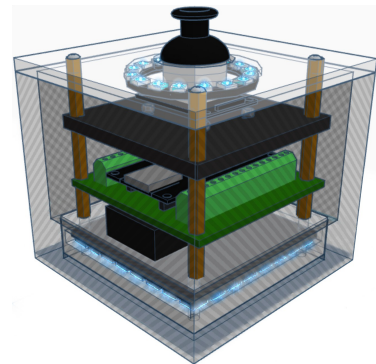


Bild 1: 3D Skizze des Buttons

Bild 1 zeigt beispielhaft das Grundgerüst der vier Buttons. Der Button wird mit dem 3D-Drucker gedruckt und ist so konstruiert, dass er hinreichend stabil ist und die notwendige Hardware aufnehmen kann. Die Buttons sind über ESP-NOW drahtlos miteinander verbunden. Gleichzeitig ist ein Button mit dem Handy über WLAN gekoppelt.



Bild2: Erster Prototyp des Buttons

Bild 2 zeigt eine Fotografie der allgemein im Button befindlichen Hardware. Dabei besitzt natürlich jeder Button noch seine eigenen Funktionalitäten: Beschleunigung-, Temperatur-, Laser- sowie Winkel- und Abstandsmessung über entsprechende Sensoren sind möglich.

Inhaltsverzeichnis

Kurzfassung	2
1. Einleitung	4
1.1 Ideenfindung	4
1.2 Wie kann so eine Lösung realisiert werden?	4
2. Vorgehensweise, Materialien und Methode	5
2.1 Entwicklung der Buttons	5
– Hardware (Grund; Probleme)	
– erkannte Fehlerquellen	
2.2 Auslesen der Sensoren	6
2.2.1 LED-Streifen	6
2.2.2 Fotowiderstand	7
2.2.3 Buzzer	7
2.2.4 Gyro-Sensor	7
2.2.5 ToF-Sensor	9
2.2.6 Entwicklung des Autos	10
2.3 Kommunikation der Buttons untereinander	11
2.4 Verbindung der Buttons mit dem Handy	12
2.5 Design des Buttons	13
3. Entwicklung der Smartphone-App	14
3.1 Programmschema	14
3.2 App Design	15
3.3 Programm Schema	16
4. Zusammenfassung	17
– weitere Verbesserungen/Erweiterungen in der Zukunft	
5. Quellen	17
6. Danksagung	17

1. Einleitung

1.1 Ideenfindung

Durch Recherche im Bereich Sport fanden wir im Internet ein Projekt, welches uns fasziniert hat: Simple, einfach, genial: Buttons mit Leuchteffekten für den Sport. Aber wir wussten: Das geht noch besser! Wie wäre es zum Beispiel mit weiteren Sensoren, Displays, Sounds etc. ?! Folglich erstellten wir schnell unsere eigenen Skizzen, wobei wir es nicht nur auf „Sport“ beruhen ließen. Genauso gut lassen sich damit auch Quizze, Reaktionsspiele, Alarmanlagen, Wecker und vieles mehr umsetzen. Wir streben nun nach einem System, mit welchem man seine eigenen Projekte erstellen kann. Dem User wären somit unendlich viele Möglichkeiten gegeben und er ist nicht nur auf die Fähigkeiten, die das Produkt bereits mit sich bringt, beschränkt.

Da unsere Ursprungsidee dem Thema Sport gewidmet war, fokussieren wir uns hauptsächlich in diesem Bereich und entwickeln 4 Buttons mit entsprechenden Sensoren und einer App.

1.2 Wie kann so eine Lösung realisiert werden?

Zuerst benötigten wir einen Mikrocontroller zum Auslesen der Sensoren. Wir entschieden uns für einen NodeMCU, da dieser besonders preisgünstig ist und über die gängige Arduino IDE mit Arduino C++ programmiert werden kann.

Bild 3 zeigt einen unserer verbauten Mikrocontroller. Der NodeMCU hat einen Esp8266 Chip verbaut, weshalb dieser WLAN basierende Netzwerke aufbauen kann. Eine weitere Besonderheit ist die „soft AP“ Funktion. Diese erlaubt nicht nur das Erstellen von Netzwerken (Access Point) und Einloggen in Netzwerke (Station), sondern auch beides gleichzeitig (soft Access Point).

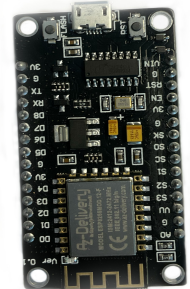


Bild 3: Esp8266 NodeMCU

Ein wesentlicher Bestandteil unserer Forschung ist die Kommunikation mehrerer Buttons miteinander. Mit Buttons sind interaktive Spielgeräte gemeint, die frei im Raum platzierbar sind.

Hierfür probierten wir unterschiedliche Methoden, wie Bluetooth, WiFi, WLAN, Funk etc. Letztendlich verwenden wir nun WLAN, da es uns zum einen am stabilsten erscheint und wir darüber auch einfache HTTP-Requests schicken können, um später mit dem Handy zu kommunizieren. Für die Kommunikation der Buttons untereinander greifen wir auf ESP-NOW zurück, eine sichere und schnellere Lösung.

Deshalb verwenden wir 4 NodeMCU's zum Aufbau von WLAN-basierenden Netzwerken.

Genauer gesagt: Ein Button baut ein eigenes Netzwerk auf, während sich alle anderen darauf einloggen.

Nun fehlen noch Sensoren, die unter anderem Sportübungen erkennen können. Wir einigten uns auf Taster bzw. später Joysticks zum Starten der Übungen, ToF-Sensoren (=Abstandssensoren), Gyrosensoren sowie Matrix-Displays und LED-Ringe für visuelle Effekte. Zur Verbesserung der Dynamik der Übungen sollen die Buttons auch beweglich sein. Dazu haben wir uns ein autonomes Auto überlegt und entwickelt, welches den Button zu einer bestimmten Position bringt, um z.B. Sprints effektiv vorgeben sowie messen zu können, ohne dass der User erst ein Maßband benötigt.

2. Vorgehensweise, Materialien und Methode

2.1 Entwicklung der Buttons

Neben dem Design der Buttons ist deren Hardware natürlich umso wichtiger. Da wir einen 3D-Drucker besitzen, lassen sich Prototypen schnell und unkompliziert herstellen.

Doch nun zur Hardware: Wir nutzen 4 NodeMCUs, die wir anfangs über HTTP-Requests miteinander kommunizieren ließen. Als die Buttons aber durch Sensoren, die ständig ausgewertet werden müssen, umfangreicher geworden sind, merkten wir, wie langsam diese Kommunikation ist.

Aber wir fanden im Internet eine andere, bessere Methode: ESP-NOW. Näheres hierzu später. Allgemein benutzen wir folgenden Algorithmus: Jede Nachricht des Senders (z.B Handy) ist mit einer Id versehen, welche die Nummer des Buttons darstellt. Empfängt ein Button die Nachricht, überprüft er, ob seine feste Id mit der der Nachricht übereinstimmt, ansonsten schickt er den Befehl an den passenden Empfänger weiter.

Wir nutzen für den Anfang Steckbretter mit Jumperkabel, um Schaltpläne schnell erstellen und ändern zu können. Dabei ist es aber trotzdem sinnvoll, die Sensoren zu löten, weshalb es bei uns des Öfteren zu Problemen kommt.

Für das Gehäuse leistet unser 3D-Drucker beste Dienste: Wir können einzelne Ebenen drucken und diese dann aufeinander-stecken. Diese Ebenen bestehen zum Teil aus Kunststoff (PLA), aber auch aus Holz (kleine Holzplatten) oder Platinen. Auf der Platine sitzt der NodeMCU, der über Kabel z.B. den Neopixel LED-Ring ansteuert, welcher zwischen zwei transparenten Plastik-Platten befestigt ist, damit dessen Licht verschwommen wirkt.

Seltsamerweise waren beim Löten des Mikrocontrollers an der Erweiterungsplatine nicht alle Pins korrekt verlötet und mussten mehrmals nachgelötet werden, um letztendlich richtig zu funktionieren.

Auch beim 3D-Drucker klappte nicht alles auf Anhieb. Fast jedes Bauteil musste mehrmals gedruckt werden, bis alle Löcher etc. exakt an den richtigen Stellen sitzen. Außerdem gibt es bei uns ein Problem auf Seiten des Druckers, als dieser mit seinen Rollen kratzt und den ganzen Druck verschiebt, woraufhin dieser natürlich zu ungenau wird. Das Problem ist zum aktuellen Zeitpunkt noch nicht behoben, wir können lediglich den gesamten Druck auf der Heizplatte nach Hinten schieben, denn hier scheint es noch zu funktionieren.

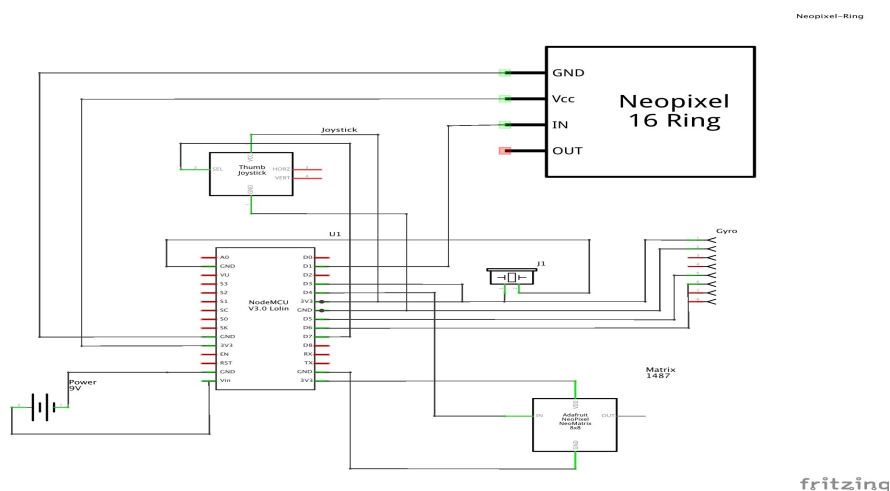


Bild 4: Schaltplan des Buttons

Bild 4 zeigt einen Schaltplan von einem unserer Buttons. Hier ist auch erkennbar, wie knapp die Verfügbarkeit weiterer Digitaler Pins für neue Sensoren ist. Im aktuellen Schaltplan gibt es lediglich noch 2 freie D-Pins, D0 und D2. Alle anderen Pins sind nicht frei belegbar. Sie stehen entweder für Power (VU), Reset (Rst), Serielle Schnittstelle (RX, TX), oder funktionieren nur während des Boots als Input (S1, S2, S3).

2.2 Auslesen von Sensoren und Co.

2.2.1 LED-Streifen

Unserer Meinung nach sind LEDs die beste Möglichkeit für visuelle Effekte. Um diesen Effekt noch einmal zu verstärken, verwenden wir LED-Strips mit IC Chips. Damit lässt sich jede einzelne LED einzeln in verschiedenen Farben und Helligkeiten ansteuern.

Dabei nutzen wir Neopixelringe mit 16x WS812 RGB-LEDs. Das Matrix von AZDelivery ist ebenfalls mit diesen IC-Chips ausgestattet. Das hat unter anderen den Vorteil derselben Ansteuerung. Somit lassen sich die LED-Ringe und die Matrix mit demselben Code und Bibliothek ansteuern.



Bild 5: LED-Ring mit IC Chips

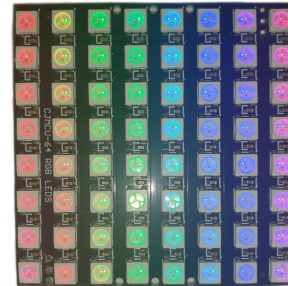


Bild 6: Matrix

Bild 5 zeigt einen LED Ring, dessen LEDs durch die IC Chips in unterschiedlichen Farben leuchten.

Bild 6 zeigt unser Matrix in Regenbogen-Farben. Der richtige Vorteil einer Matrix aber ist das Anzeigen von Zahlen, Symbolen oder sogar Buchstaben. Wir verwenden deshalb die Matrix auch, um z.B. einen Start-Countdown vor dem Sprint anzuzeigen. Hierfür müssen wir zuerst mithilfe eines Pixelart Programms die Zahl erstellen und danach Pixel für Pixel im Code eintragen. Um dabei jedoch noch ein klein wenig Arbeit zu ersparen, färben wir im Code zuerst alle Pixel in einer Farbe und verändern dann nur noch die Pixel für die Zahl. Dadurch müssen wir zwar nicht mehr alle 64 Pixel verändern aber es ist dennoch noch ein großer Aufwand.

Bei der Verkabelung ist es wichtig, auf den Unterschied zwischen Input- und Output Pins zu achten. Unsere Neopixelringe besitzen zwei Seiten mit Kontakten damit mehrere Ringe miteinander gekoppelt werden können. Somit funktioniert nur eine Seite als Input.

Die von Fabrik aus befestigten Kabel sitzen leider nicht sehr gut, weshalb auch hier der Lötkolben zum Einsatz kommt.

```
void neopixel(int r, int g, int b){
  for(int i=0; i<NUMPIXELS; i++) { // For each pixel...
    pixels.setPixelColor(i, pixels.Color(r, g, b));

    pixels.show();    // Aktualisiert die Pixel
  }
}
```

Quellcode 01 zeigt das Ändern der Farbe des Neopixelringes. Der Funktion `neopixel()` werden die 3 Werte `r`, `g` und `b` mitgegeben.

Die For-Schleife ändert nun mit `pixels.setPixelColor()` jede LED in die entsprechende Farbe. Da der Befehl `pixels.show()` nach jedem Pixel ausgeführt wird, könnte man bei einer größeren Anzahl sogar das Ändern einzelner Pixel sehen. Energiesparender ist es deshalb das Aktualisieren nur einmal nach der Schleife durchzuführen.

2.2.2 Fotowiderstand

Wir möchten ebenfalls einen kleinen Laser-Tagger für unseren Button haben. Dabei soll ein beliebiger Button leuchten, damit der User hinrennt und ihn abschießt. Um diesen Laser erkennen zu können, möchten wir das Licht mittels einem Fotowiderstandes messen.

Ein Fotowiderstand ändert seinen eigenen Widerstand in Abhängigkeit der Lichtstärke. Damit lässt sich anhand der an ihr anliegenden Spannung ein Wert für die aktuelle Helligkeit ablesen. Sobald der Laser den Fotowiderstand trifft, empfängt dieser ein helleres Licht, welches sich in Form einer niedrigeren Spannung wieder-spiegelt.

2.2.3 Buzzer

Neben LEDs gibt es noch eine weitere Möglichkeit, dem Nutzer Signale zu geben: Buzzer. Damit lassen sich Töne erzeugen. Es gibt aktive Lautsprecher (active buzzer) und passive Lautsprecher (passive buzzer). Wir verwenden einen passiven Lautsprecher, da dieser die Möglichkeit bietet unterschiedliche Tonhöhen zu generieren. Im passiven Buzzer ist keine Elektronik vorhanden, die den Ton vorgibt. Es können also auch Melodien oder Sirensensignale abgespielt werden.

Die Ansteuerung ist mehr als simpel. Der Buzzer verwendet zwei Kabel für die Spannung. Ground wird direkt mit der Masse des Mikrocontrollers verbunden während die 3V per Code kontrolliert werden. Im Code lässt sich mit einer Funktion der Buzzer mit einer bestimmten Tonhöhe aktivieren und schließlich nach einer gewissen Zeit wieder deaktivieren.

2.2.4 Gyro-Sensor

Mithilfe eines Gyro-Sensors haben wir vor unterschiedliche Übungen anhand der Beschleunigung oder des Winkels zu erkennen.

Unser Sensor erkennt Winkel, Beschleunigungen sowie Temperaturen, da wir einen 3-Achsen Beschleunigungs- und Lagesensor GY-521 verwenden.

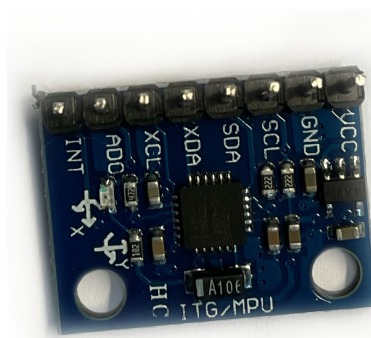


Bild 7: Gyro-Sensor

Anfangs hatten wir Probleme mit der softwareseitigen Erkennung des Sensors. Dieses Problem ist jedoch durch Löten der Kontakte schnell gelöst. Nun gibt es aber für den Arduino verschiedene Bibliotheken zum Steuern des Gyros. Seltsamerweise gaben diese jeweils unterschiedliche Werte an. Das lag daran, dass die Raw-Daten des Gyro-Sensors nicht in Grad (deg), sondern in Grad pro Sekunde (deg/s) geliefert werden und somit erst umgerechnet werden müssen.

```

int matrix_gyro(){
    int deg = mpu.getAngleX();
    //Serial.println(deg);
    int dir = 0;
    if(deg >= 150){          // = 150 - 180 → 180
        dir = 2;
    }else if(deg >= 50){    // = 50 - 150 → 90
        dir = 1;
    }
    // [...]
    return dir;
}

int turn_matrix(int n){
    if(n < 8){
        n = (n+1) * 8 - 1;
    }else if(n < 16){
        n = (n-8+1) * 8 -2;
    }
    // [...]

    return n;
}

int matrix_num(int num){
    int dir = matrix_gyro();
    int n = num;
    switch(dir){
        case 1:
            n = turn_matrix(n);
            break;
        case 2:
            n = turn_matrix(n);
            n = turn_matrix(n);
            break;
        // [...]
    }
}

void example_matrix(){
    matrix.setPixelColor(matrix_num(10), matrix.Color(50, 50, 50));
    matrix.show();
}

```

Quellcode 02 (Arduino C++)

zeigt einen größeren Ausschnitt aus dem Arduino-Script, welcher alle Grafiken, die auf der Matrix abgebildet werden, in die richtige Richtung dreht. Im Ausschnitt werden vier Funktionen gezeigt (jeweils eingerückt). Betrachten wir zuallererst die letzte Funktion *example_matrix()*. Hier färbt der Button das Pixel Nummer 10 der Matrix mit der Farbe Weiß. Sobald jedoch die Matrix gedreht wird, befindet sich dieses Pixel 10 nicht mehr an der richtigen Position. Deshalb wurde eine Funktion *matrix_num()* zum Drehen der Pixel entwickelt. Diese Funktion dreht das Pixel in Abhängigkeit der Funktion *matrix_gyro*, welche je nach Rotation einen Wert zwischen 0 und 3 ausgibt. Die Rotation reicht von null zu 90 bis 180 und nimmt negative Zahlen von -180 zu -90 bis hin zu null an. Somit liegen 180 und -180 direkt nebeneinander. Nun folgt die Funktion *turn_matrix()* mit der einen oder anderen kleinen Rechnung. Für die Herleitung dieser Rechnungen haben wir zuerst beispielhaft einige Zahlen aufgeschrieben, um dann manuell die passende Pixel-Nummer abgelesen, die bei einer Drehung entsteht:

```

/* 0 --> 1* 8 -1      = 7
* 1 --> 2* 8 -1      = 15
* 2 --> 3* 8 -1      = 23
* 7 --> 8* 8 -1      = 63
*
* 8 --> 1* 8 -2      = 6
* 9 --> 2* 8 -2      = 14
* 10 --> 3* 8 -2     = 22
* 15 --> 8* 8 -2     = 62
*
* 16 -->      ...    = 5
* 24 -->      ...    = 4
*/

```

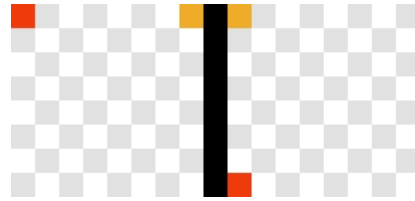


Bild 8: Pixel Grafik der Matrix

Nimmt man das erste Pixel der Matrix (Nummer 0 = oben links) und möchte nun bei einer -90 Grad Drehung an dieser globalen Position (also immer noch oben links) das selbe Pixel haben, so wäre seine neue Position auf einen 8x8-Matrix Position 8 (Nummer 7). Somit ergibt sich am Ende die oben gelistete „Formel“.

Bild 8 soll dies nochmals verdeutlichen: Dort werden zwei Matrize, durch einen schwarzen Strich getrennt, direkt nebeneinander dargestellt. Betrachten wir zuerst die linke Matrix. Hier leuchtet das Pixel 0 (oben links) orange und Pixel 7 gelb (oben rechts). Oben links befindet sich somit Pixel 0. Sobald man aber die Matrix dreht, befindet sich oben links, wie im zweiten Matrix, Pixel 7. Nach genau diesem Schema arbeitet der Code.

2.2.5 ToF-Sensor

Um Abstände möglichst genau und weit messen zu können, verwenden wir die ToF Technologie. ToF (Time of Flight) -Sensoren senden Laser mit Infrarotstrahlung aus und messen die Zeit, bis das Signal wieder empfangen wird. Damit lässt sich mit unserem Sensor eine Entfernung von bis zu 4m mit einer Genauigkeit von $\pm 2\text{mm}$ berechnen.

Je nach Verwendungszweck ist es möglich, unterschiedliche „Modi“ zu wählen (z.B. kurze Distanz / weite Distanz), woraufhin sich der Sensor entsprechend anpasst und sich bestimmte Werte verbessern (z.B. Genauigkeit), sich aber im Gegenzug andere Werte verschlechtern (z.B. Reichweite).

Wichtig ist aber auf jeden Fall das Datenblatt, welches alle Angaben inklusive Schaltplan etc. liefert. Diese erspart einige Unsicherheiten, sofern es im Netz einige (falsche) Informationen aufgrund der Nachahmerprodukte gibt. Unter anderem gibt es unterschiedliche Artikel über dasselbe Produkt, bei denen die einen sagen, es müsse 3V Spannung verwendet werden, 5V wäre zu hoch, andere jedoch meinten, es müsse 5V verwendet werden, sonst funktioniert der Sensor nicht richtig. Ein Blick auf das mitgelieferte Datenblatt gibt Gewissheit: 3V BIS 5V sind möglich.

Bild 9 zeigt unseren Abstandssensor der Marke Pololu.

Von den sieben Pins werden nur vier gebraucht.

Zwei verwenden dabei die Power, zwei andere schicken über den I2C-Bus dem Mikrocontroller die nötigen Informationen. Die zwei gelben Vierecke in der Mitte sind das Herz des Abstandssensors, welcher natürlich nicht verdeckt werden darf.

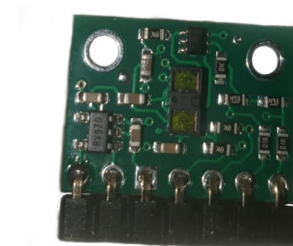


Bild 9: ToF-Sensor

2.2.5 Entwicklung des Autos

Bei der Entwicklung des Autos siehe *Bild 10* besteht die Hauptproblematik in einer sicheren Ansteuerung und Integration mittels Sensoren.

Um Motoren korrekt ansteuern zu können, benötigt man eine sogenannte H-Bridge. Diese erlaubt es uns, die Räder durch Umpolung sowohl vorwärts, als auch rückwärts drehen zu lassen. Nun muss das Auto auch noch die zurückgelegte Strecke ausrechnen. Im Internet fanden wir dazu mehrere Möglichkeiten:

1. GPS: Damit lässt sich die aktuelle Position ausgeben, welche aber unterschiedlich genau bzw. ungenau ist. Bei einer 50m Strecke wären 5m Ungenauigkeit viel zu viel.
2. Abstandssensoren (z.B. Ultraschall oder ToF): Diese Messwerte sind sehr genau, aber erlauben maximal 4m. Wir überlegten uns eine Art Wand, von der aus das Auto 4m weg fährt, sie dann einzieht und dann wieder 4m fährt usw. Dies wäre aber zu aufwendig und langsam.
3. Lichtschranken: Sie bieten sich als einfache und robuste Lösung an. Diese sind die simpelste und beste Methode: Mit einer Lichtschranke könnte man die Radumdrehungen zählen, und dann mithilfe des Radumfanges die Strecke berechnen.

Natürlich findet man die cleversten Arten erst am Ende, aber man findet sie! Damit ist es uns nun möglich ein aus dem 3D-Drucker gefertigtes Auto mit Ladefläche zu erstellen, um die Buttons zu einer bestimmten Position zu fahren.

Wichtig bei der Ansteuerung der Motoren ist, das Datenblatt des entsprechenden Mikrocontrollers korrekt zu lesen. Bei uns nutzt dieser z.B. einen 1023er Cycle anstatt des standardmäßigen 255er-Cycles. Damit ist das PWM-Signal gemeint, mit welchem die Geschwindigkeit kontrolliert wird. Das PWM-Signal kann man sich wie ein Blinken vorstellen. Je höher das Signal, desto schneller das Blinken. Der Cycle gibt das Maximum an, also ab wann 100% erreicht sind. In unserem Fall bedeutet es, bei 255 würde der Motor lediglich Summen statt sich richtig zu drehen.



Bild 10: Transportfahrzeug des Buttons

2.3 Verbindung der Buttons mit dem Handy

Eines der wichtigsten Teilgebiete ist die Verbindung der Buttons. Diese müssen Befehle der App empfangen und anschließend die passenden Messwerte der Sensoren zurücksenden. Damit die Verbindung klappt, darf sich nur ein Button (Master) mit dem Handy verbinden, und die anderen Buttons (Slaves) verbinden sich mit dem Master. Wie vorhin schon in Kapitel 1.2 besprochen, entschieden wir uns für eine WLAN-Verbindung. Da aber die Nutzung auch außerhalb eines öffentlichen Netzwerkes möglich sein soll, erstellt der Master (wie eine Art Router) ein Netzwerk, mit welchem sich das Handy verbindet.

Um die Kommunikation so einfach und flexibel wie möglich zu halten, verwenden wir sogenannte POST-Requests, die Anfragen an den Master senden, und dieser antworten kann.

Gleichzeitig aktivierten wir zusätzlich OTA. Dies ist eine Methode, um Uploads über das WLAN zu tätigen. Sobald also der NodeMCU diesen OTA-Server aufgebaut hat und mit demselben Netzwerk wie der PC verbunden ist, erscheint in der Arduino IDE ein weiterer Port mit der entsprechenden IP-Adresse. In unserem Projekt baut der Master-NodeMCU bereits sein eigenes Netzwerk auf, es bietet sich also an, dieses zu nutzen. Nach dem Umschreiben des Codes ist es nun tatsächlich möglich sich ohne Verbindung zum Internet, also lediglich in einem internen Netzwerk, sich mit den NodeMCUs zu verbinden und ihr Sketch über WLAN zu aktualisieren. Die einzige Voraussetzung hierbei ist, den „OTA-Abschnitt“ im Code unverändert beizubehalten.

```
public void send_sprint(){
    StartCoroutine(http("sprint"));
}
IEnumerator http(string befehl) {
    print("http... ");
    ready = false;
    WWWForm form = new WWWForm();
    switch (befehl)
    {
        case "sprint":
            form.AddField("command","5");
            form.AddField("id","2"); // sprint button
            form.AddField("d5","1"); // start button
            break;
    }
    UnityWebRequest www =
    UnityWebRequest.Post("http://192.168.1.1:81/http", form);
    yield return www.SendWebRequest();
    if(www.isNetworkError || www.isHttpError) {
        Debug.Log(www.error);
    }
    else {
        // Show results as text
        string text = www.downloadHandler.text;
        print(text+ " / <!DOCTYPE HTML><html>true");
    }
    ready = true;
}
```

Quellcode 03 (Unity C#)

zeigt einen beispielhaften Befehl, der von der Smartphone-App zum Button geschickt wird. Zuerst wird die Funktion `http()` über eine öffentliche Funktion `send_sprint()` ausgelöst. Diese Funktion ist einem bestimmten Event-Trigger eines Button innerhalb der App zugeordnet. Daraufhin wird `WWWForm` mit den entsprechenden Variablen und deren Werten gefüllt. Danach wird der Befehl an den Master, welcher den Router mit der IP 192.168.1.1 darstellt, gesendet. Mit `yield` wartet der Code, bis die App eine Gegen-Antwort empfängt und dann mittels `print()` ausgegeben.

2.4 Kommunikation der Buttons untereinander

Wie in Kapitel 2.1 bereits erklärt sind HTTP-Requests für unsere Zwecke zu langsam. Deshalb sind wir auf ESP-NOW umgestiegen.

Mit diesem Protokoll können sich mehrere Geräte unterhalten, ohne Wi-Fi zu verwenden.

Sie benötigen statt der IP- nur die Mac-Adresse. Nachdem sich die Geräte miteinander verbunden haben, ist die Verbindung stabil und Peer-To-Peer. Das bedeutet, sollte ein Arduino abstürzen, kann er sich sofort wieder verbinden und an der Unterhaltung teilnehmen.

```
void send_answer(){
    if(!sended){
        sended = true;
        // Set values to send
        myData.c = i_command;
        myData.n1 = i_d1;
        myData.n2 = i_d2;
        myData.n3 = i_d3;
        myData.n4 = i_d4;
        myData.n5 = i_d5;
        switch(i_id){
            case 2:
                // Send message via ESP-NOW
                esp_now_send(address1, (uint8_t *) &myData, sizeof(myData));
                break;
            case 10:
                esp_now_send(address10, (uint8_t *) &myData, sizeof(myData));
                break;
        }
        lastTime = millis();
    }
}
```

Quellcode 04 (Arduino C++)

zeigt einen Codeausschnitt, in dem grob das Senden von Nachrichten über ESP-NOW verdeutlicht wird. Die Funktion `send_answer` überprüft zuerst, ob die aktuelle Nachricht bereits gesendet wurde, wenn nicht, sendet es die Nachricht, je nach `i_id` dem entsprechenden Button bzw. dessen Mac-Adresse `adress1` / `adress10`, zu.

Fehler kann es bei dieser Methode kaum geben, es sei denn man wechselt einen der Mikrocontroller. Denn dann muss jedem Button wieder die neue Mac-Adresse zugewiesen werden. Dies war bei uns einmal der Fall, als ein Mikrocontroller durch einen Kurzschluss durchgebrannt ist und somit defekt war.

Allerdings wird wie in Kapitel 2.3 immer noch HTTP-Requests zwischen Handy und Button verwendet. Hier gibt es schneller Probleme, wenn sich z.B. die IP-Adresse ändert. Deshalb nutzen wir statische IP-Adressen, wobei ein anderes eingeloggtes Gerät dieselbe IP-Adresse haben könnte. Somit ist die Kommunikation über die MAC-Adresse wesentlich stabiler. Zur Verständnis zeigt *Bild 11* nun diese gesamte Kommunikation vom Handy bis zum einzelnen Button und wieder zurück.

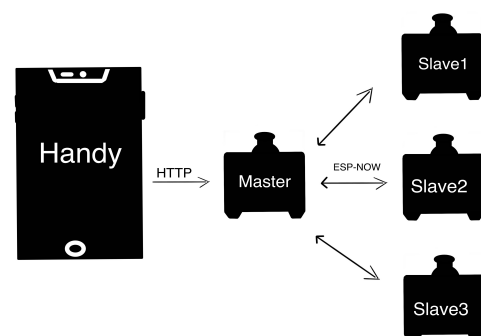


Bild 11: Kommunikation

2.5 Design der Buttons

Für das Erstellen des 3d-Duckmodels wurde zuerst das Programm Blender verwendet, jetzt sind wir aber auf die App Shapr3D gestoßen. Mit dieser lässt sich ganz einfach ein Modell in den passenden Maßen entwerfen. Da wir hierfür keine Farbe oder eine Hochauflösung brauchen ist für uns das Programm kostenlos. Jedoch können wir nur 2 Designs gleichzeitig entwerfen. Shapr3D ist speziell für das iPad und somit perfekt auf den Apple Pen zugeschnitten.

Das Endprodukt soll natürlich möglichst klein sein, aber es muss auch alles bequem hineinpassen. Somit versuchten wir unsere Ebenen möglichst klein und trotzdem gleich groß zu erstellen. Wir kauften quadratische Platinen mit einer Breite von 65mm und quadratische, 65mm breite Holzplatten. Genauso ist auch unser LED-Matrix 65mm groß. Somit können wir alle Ebenen „stapeln“ und dann in eine Box hineinlegen. Die ist aus unserer Sicht sehr praktisch, da alles verschraubt und verkabelt werden kann und erst dann mit ein paar Schrauben in der Box befestigt wird. Dies erlaubt ebenfalls ein leichtes Herausnehmen der Technik, um sie erweitern, zeigen oder reparieren zu können.

Für das äußerliche Design haben wir uns an dem eines klassischen Tasters orientiert.



Bild 12: Taster



Bild 13: Button geschlossen

Bild 12/13 zeigt den Vergleich zwischen einem Taster und unserem Button.

Dieses Design haben wir gewählt um einen anderen Look im Vergleich zu anderen Produkten zu haben. Vieles was es zu kaufen gibt, ist modern und rund designt. Aber warum nicht auch einmal eckig? Zusätzlich ist die Form perfekt an die Elektronik angepasst, welche so simpel verstaut werden kann.

3. Entwicklung der Smartphone-App

3.1 Software, Sprache etc.

Für unsere App nutzen wir die Spiele-Engine Unity, da wir mit dieser schon seit Längerem Erfahrungen teilen. Mit ihr lassen sich eigentlich komplexe 3D-Games entwickeln, für uns eignet sie sich aber auch für diese simple 2D App.

Als Programmiersprache nutzen wir C# mit dem Visual Studio Editor.

Mithilfe der App soll es dem User möglich sein nicht nur zwischen vorgefertigten Übungen/Workouts auswählen zu können, sondern eben auch eigene Funktionen erstellen zu können. Hier ist zum Beispiel ein eigener Trainingsplan vorstellbar.

Auf einen Befehl hin sendet die App einen HTTP-POST-Request an immer dieselbe IP-Adresse, welche den NodeMCU als Router darstellt. Daraufhin bekommt sie eine Antwort, je nachdem ob die Anfrage geklappt hat oder nicht. Das System hatte anfangs jedoch einen kleinen Bug, da es solange gewartet hat, bis es eine Bestätigung bekommen hat, und erst dann wieder eine Variable auf bereit gesetzt hat, um weitere Anfragen zu starten. Dieses Prinzip klappt solange, bis man nicht mehr mit dem richtigen Router verbunden ist, denn dann wird man gar keine Antwort erhalten. Somit musste diese Warteschleife durch einen Timeout unterbrochen werden.

Außerdem wurde der Animation Controller durch LeanTween ersetzt. Genauer gesagt: Um mit Unity Effekte zu erstellen verwendet man häufig den Animation Controller. Mit ihm ist es möglich Animationen mittels Keyframes zu entwerfen. Laut Unity selbst sollte man diese Methode aber nicht für die UI verwenden, da sie prozessorfressend sei. Unsere App besteht aufgrund des 2D Designs hauptsächlich aus UIs (User Interfaces), weshalb wir zu LeanTween umgestiegen bin. LeanTween ist ein Animationstool, welches das Erstellen von Animationen über Code erlaubt. Mit ihm lassen sich zwar nicht komplexe Animationen, wie z.B. für einen Charakter erstellen, sondern eben nur kleine Effekte um z.B. zwischen Tabs zu wechseln. Damit lassen sich Objekte drehen, bewegen, skalieren, pausieren, verfärben etc. Diese Methode arbeitet nicht, wie der Animation Controller, nach Keyframes, wodurch alles in jedem Frame neu gerendert werden müsste. Ich habe zwar keine Veränderung bemerkt, würde aber vermutlich einen Unterschied feststellen, sobald die Software umfangreicher wird. Aber LeanTween verschafft mir trotzdem einen Vorteil gegenüber des Animation Controllers: Die Animationen müssen per Code ausgelöst werden, sobald ein Befehl ausgelöst wird. Da mit LeanTween sowieso per Code animiert wird, lassen sich Animationen auch leichter abspielen, ohne erst für jede Animation einen Controller zu erstellen und diesen dann dem Code über eine public Variable zuweisen zu müssen!

Bild 14 verdeutlicht die Arbeit mit Unity.

Hier wird die Benutzeroberfläche der Engine grob dargestellt. Im Bild unten sind einige C#-Skripte zu sehen, links das Verzeichnis der aktuellen Szene, rechts finden sich Informationen zu dem jeweilig aktivierten Objekt und schließlich in der Mitte die Darstellung der Szene. In der eigentlichen App ist jedoch nur der Bereich bis zu den schwarzen Rechtecken zu sehen.

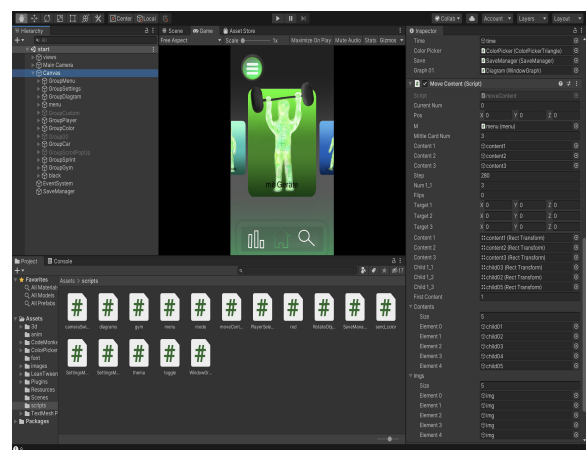


Bild 14: Oberfläche von Unity

3.2 App-Design

Das App-Design wurde anfangs in Pixelart gezeichnet, da wir damit schon Erfahrungen gesammelt haben. Um unsere App aber mit einem moderneren Style zu gestalten, sind wir aber aktuell auf eine „normale“ Grafik gewechselt. Diese Art von Design ist zwar aufwändiger, aber nach unserer Sicht auch lohnenswerter. Dafür verwenden wir die App SketchBook auf dem iPad (Alle Grafiken in dieser App wurden selber designt).

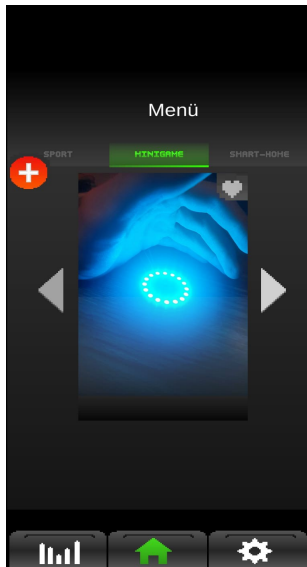


Bild 15: Vorher

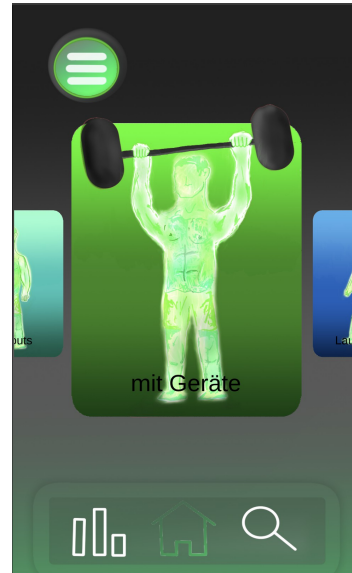


Bild 16: Nachher

Bild 16 zeigt die App Oberfläche unserer App für den User. Für die Grafiken haben wir uns an anderen Apps orientiert und insgesamt einen Schwarz-Grau-Grün-Stil gewählt.

In der App hat der Nutzer die Möglichkeit zwischen 3 Hauptmenüs zu wechseln. Dabei befindet sich in der Mitte das Home Menü. Hier lassen sich unterschiedliche Bereiche auswählen („mit Geräte“, „Sprint“, „Dehnen“, „Workouts“, „Quiz“). Nach und nach sollen aber auch weitere Bereiche wie SmartHome oder Minigames abgedeckt werden.

Das linke Menü (nicht dargestellt) hingegen zeigt Statistiken der unterschiedlichen Bereiche in Form von Diagrammen an. Dabei sendet der Button z.B. die gemessenen Sprint-Zeiten und in der App wird ein Verlauf dieser Werte sowie der Rekord angezeigt. Über die Diagramme ist es dem User möglich Verbesserungen über einen gewissen Zeitraum zu erkennen. Für das Design dieser Diagramme haben wir uns an einer Aktien-Anzeige orientiert. Sollte man sich verbessern, leuchtet es grün, ansonsten rot.

Des Weiteren befindet sich im rechten Menü (nicht dargestellt) eine Art Suchfunktion für Übungen unterschiedlicher Muskeln. Dort wird eine Figur angezeigt, bei welcher die Muskeln durch Drücken ausgewählt werden können.

Unsere App erstellt ein Profil über den Nutzer, um die Übungen optimal anpassen zu können. Hierfür muss der User beim ersten Start ein Trainings-Workout absolvieren. Dabei wird sein Limit anhand der unterschiedlichen Übungen erkannt, wobei ein gewisser Prozentteil dieses Limits für die optimale Anzahl der Übungen steht.

3.3 Programmschema

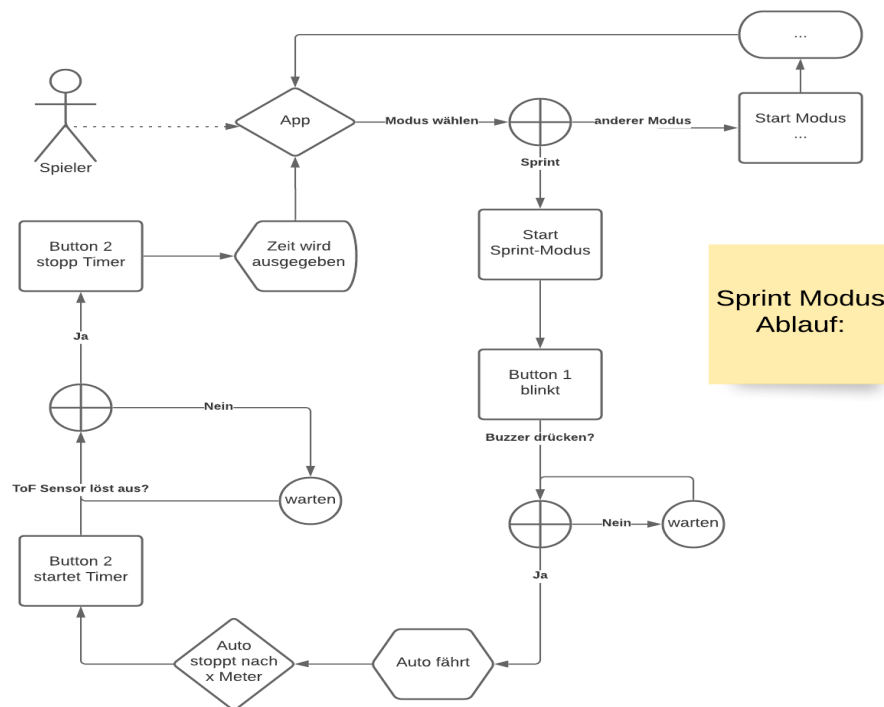


Bild 17: Zustandsdiagramm

Bild 17 zeigt ein Zustandsdiagramm im Sprint Modus.

Sobald der Spieler in der App den Sprint-Modus wählt, wartet Button1 darauf, gedrückt zu werden, woraufhin das Auto losfährt. Wenn das Auto stoppt, misst Button2 die Zeit, bis der Time of Flight Sensor den Spieler erkennt. Danach kann die benötigte Zeit ausgegeben werden.

Ein weiteres Feature unserer App ist das Speichern und visuelles Anzeigen der Übungen. Über Diagramme ist es dem User möglich, Steigerungen und Rekorde zu erkennen. Dabei benötigt das Diagramm eine ordentliche Darstellung um korrekt ablesbar zu sein. Hierzu wurde in Unity eine eigene Klasse *WindowGraph* erstellt. Ihr werden die Höhe und Breite des *graphContainers* mitgeliefert. Damit kann nun ein dynamisches Diagramm in Abhängigkeit einer *valuesList* erstellt werden. Der Code passt dabei die X- und Y-Achse automatisch an. Im Folgendem wird die mathematische aber auch teils codeseitige Lösung beschrieben:

Zuerst muss *yMaximum* und *yMinimum* anhand der gegebenen *valuesList* bestimmt werden. Nun berechnet der Code die Differenz dieser beiden Zahlen, damit er ihnen jeweils 20% hinzufügen bzw. abziehen kann, um etwas Abstand zu erhalten. Zum Anzeigen der X-Achse (*labelX*) wird zuerst *xSize* mit $xSize = graphWidth / valuesList.Count$ errechnet. Jetzt kann die *xPosition* mit $posX = xIndex * xSize$ erfasst werden, wobei *xIndex* mit jedem Durchlauf um eins steigt.

Nun benötigt das Diagramm nur noch die Y-Beschriftung (*labelY*). Hierfür wird eine Schleife gebraucht, die sich sooft wiederholt, bis ihre Wiederholung (*int val*) größer als die Differenz $yMaximum - yMinimum$ ist. In jedem Durchlauf wird *posY* mit $posY = val / (yMaximum - yMinimum) * graphHeight$ errechnet. Da *val* die Anzahl der Wiederholungen wiedergibt, wird sie nie größer als $(yMaximum - yMinimum)$. Somit wird in der obigen Rechnung *graphHeight* mit einer Zahl zwischen null bis eins multipliziert. Dies gibt Sinn, da sich *posY* immer in einer Höhe von null bis hin zu *graphHeight* befindet und nie darüber hinaus geht.

4. Zusammenfassung

– weitere Verbesserungen/Erweiterungen in der Zukunft

Wir haben vor, die Buttons noch mit Kameras und Co zu erweitern. Zusätzlich möchten wir unsere Buttons mit Wireless Charging ausstatten. Dies wäre zwar ein cooles Feature aber eben auch eine Frage des Preises, weshalb wir darauf erstmal verzichten.

Außerdem planen wir, unser Projekt open Source zur Verfügung zu stellen. Das bedeutet, wir würden auf unserer Website alle nötigen Anleitungen, Stl-Files für den Drucker, Unity-Resources und Arduino Skripte hochladen. Wir könnten uns auch vorstellen, Affiliate Links zu den Hardware Produkten, wie den Sensoren und LED-Ringen, anzubieten.

Weitere interessante Features wären z.B Magnete, mit welchen sich der Button an Geräten befestigen lässt. Oder, um die Kids von heutzutage auch zu motivieren, Laser Tagger, mit den Buttons als Ziel. Auch eine neue Idee ist die Verbindung der Buttons mit einer Smartwatch. Hier ist das Senden von Pulsraten oder das Starten der Übung vorstellbar.

Des Weiteren planen wir eine Erweiterung der App mit einem extra Multiplayer-Modus. Denkbar wäre hier eine lokale Variante, bei der jeweils 2 Buttons pro Spieler verwendet werden.

Insgesamt haben wir während des gesamten Projektes viel Neues gelernt und gleichzeitig festgestellt, dass die Möglichkeiten die sich während der Entwicklung ergeben, unendlich sind. Wir haben noch einige weitere Ideen, mussten uns aber explizit auf das Wesentliche konzentrieren, um unsere Forschung noch rechtzeitig fertigstellen zu können. Unsere Fähigkeiten haben sich in der gesamten Arbeit perfekt ergänzt: Während Manuel programmiert hat, designte Marco sowohl die gesamte App als auch alle Grafiken für den 3D-Drucker. Die Technik haben wir gemeinsam ausgewählt und zusammengebaut.

5. Quellen

Funduino: <https://funduino.de/nr-25-das-nodemcu-mit-integriertem-esp8266-modul>

Thingiverse: <https://www.thingiverse.com/thing:3518669>

Fotos: Alle Bilder sind selber fotografiert und bearbeitet

6. Danksagung

Unser Dank gilt unserem Lehrer, Herrn Dr. Andreas Stingl, der uns während des Robotikunterrichtes zu der Idee dieses JuFo Projekt inspiriert und uns immer wieder ermutigt hat, dieses Projekt voranzutreiben und fertigzustellen. Außerdem danken wir unserer Familie für die Bereitstellung der Technik und die Unterstützung bei der Entwicklung und Fertigstellung des Projektes zu Hause.