2021

Projektarbeit - Arducraft

ARDUINO & HOVERCRAFT

MIRKO CORDES

LEHRKRÄFTE | Herr Hartmann & Frau Wenke

Inhaltsverzeichnis

Vorwort

Zu Beginn der Projektarbeit möchte ich meinen Freunden danken, die mich bei Fragen unterstützt haben. Ob ich dieses „Projekt“ auch nach dem Abgabedatum weiterführe, steht noch offen. In dem Projekt bin ich auf ein großes Problem gestoßen, welches auch andere Personen betrifft, die sich mit einer Bluetooth Verbindung und dem Arduino beschäftigt haben. Dadurch ist meine komplette Projektarbeit praktisch zum Erliegen gekommen und fokussiert sich hauptsächlich theoretisch mit der bevorstehenden Thematik.

Einleitung

Das Rahmenthema dieser Projektarbeit ist die Erstellung eines Vehikels. Wäre es nicht schön, mit einem Fahrzeug durch die Gegend zu schweben? Zumindest ist diese Vorstellung im kleinen Format möglich. Ein Hovercraft (übersetzt: „Luftkissenboot“) ist ein Fahrzeug, welches mit Hilfe von Turbinen, also Motoren, über dem Boden schwebt und sich so fortbewegt. Heutzutage finden sogar eigene Rennen mit selbstgebauten Luftkissenbooten statt. Ein weiterer Vorteil ist auch, dass sie sogar über Wasser gleiten und deshalb auch im maritimen Bereich Anwendung gefunden haben.

Mein Ziel ist es in dieser Projektarbeit ein Mini-Hovercraft zu erstellen, welches durch eine Android Applikation gesteuert werden kann. Dieses Mini-Hovercraft werde ich im Anschluss dem Cousin meiner Freundin zum Geburtstag schenken, da ich sonst keine Verwendung für das Spielzeug finde. Wahrscheinlich werde ich dabei die benötigte Zeit von sechs Wochen nicht einhalten können, da ich alles selbst „erstellen“ werde. Das spart zwar keine Zeit, jedoch weiß ich so genau, wie das Hovercraft aufgebaut ist und kann bei Problemen in der Zukunft besser handeln. Ein gewisser Lernfaktor ist also auch mit dabei.

Zum Erstellen eines Hovercrafts werde ich ein Arduino verwenden, welches gesendete Daten des Handys empfängt und auswertet. Nach der Auswertung der Daten wird ein Motor, so wie eine Servolenkung angesteuert. Um die Daten zu empfangen, benötige ich zusätzlich noch ein Modul. Meine Wahl fiel dabei auf ein entsprechendes Bluetoothmodul, da Bluetooth eine gute Reichweite hat und die meisten Smartphones immer über Bluetooth Schnittstellen verfügen. Ein Internetmodul benötigt außerdem eine Sim-Karte und somit auch einen Mobilfunkvertrag, welches für ein Spielzeug nicht notwendig sein sollte. Letztendlich bestünde auch die Möglichkeit, ein Frequenzmodul zu verwenden (Radiofrequenzen). Jedoch befindet man sich mit solchen Modulen entweder in der Grauzone, da beispielsweise Radiofrequenzen angemeldet werden müssen oder man empfängt Daten von anderen Geräten. Bei Bluetooth haben wir diese Probleme nicht. Durch einen sogenannten Socket wird eine gesicherte und relativ verlustfreie Verbindung zum Endgerät hergestellt.

Beginnen wir mit dem technischen Aufbau.

Technischer Aufbau

Um das Hovercraft nun zu erstellen, benötige ich erstmal Hardware. Dieses Hovercraft wird durch eine eigens entwickelte App und dem Arduino UNO R3 gesteuert. In der folgenden Auflistung sehen sie die benötigte Hardware:

• 1x 6V Gleichstrommotor • 1x 5V Servomotor • 2x Kippschalter • 3x zweier Batteriepack inkl. Batterien • 1x 3.3V HC-05 Bluetooth Modul	• 1x 5V Voltage Booster • Ca. 10x (Jumper-)Kabel • 1x Transistor • 1x 3D-Modelle • 2-3x Widerstände

Tabelle 1: Benötigte Hardware

Außerdem werden folgende Programme, beziehungsweise Software, benötigt:

- Arduino Programm
- Android Studio
- Bluetooth-Client

Arduino mit Strom versorgen

Um den Arduino später ohne externe Stromquelle nutzen zu können, wird ein Batteriepack an das Arduino geschlossen. Da allerdings zwei Batterien nur eine maximale Spannung von 3V abgeben, schalte ich einen „Voltage Booster“ zwischen die Batteriepacks. Dadurch wird die Spannung auf 5V USB-Spannung gehoben, die der Arduino für den Betrieb benötigt. Dieser Voltage Booster fungiert ähnlich wie ein Transformator. Allerdings braucht ein Transformator eine bestimmte Eingangsspannung, um die Ausgangsspannung zu regulieren. Der Voltage Booster hingegen regelt die ausgehende Spannung unabhängig von der Eingangsspannung auf 5 Volt. Vor dem Voltage Booster muss allerdings auch ein Kippschalter in den Schaltkreis eingebaut werden, um das Arduino später ausschalten zu können.

Motor schalten

Da der verwendete Motor, der im späteren Verlauf einen ausreichenden Luftstrom erzeugen wird, damit das Hovercraft gleitet, eine 6V Spannungsquelle benötigt, habe ich einen zweiten Schaltplan entwickelt. Dieser zweite Schaltplan verfügt über eine zusätzliche Stromquelle (zwei Batteriepacks). Dadurch kann die 6V Stromquelle gewährleistet werden, die der Motor benötigt. Wie auch der Schaltplan des Arduinos, verfügt diese Stromquelle über einen Kippschalter. Somit werden insgesamt zwei Kippschalter verwendet. Folgende Abbildung zeigt den Schaltplan des Motors.

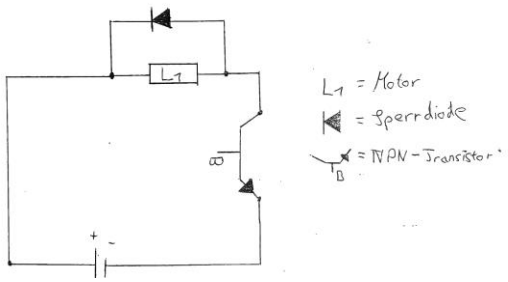

Abbildung 1: Motorschaltplan

In diesem Schaltplan werden ein Motor, Transistor, Widerstand und

eine Schutzdiode verbaut. Durch den Transistor ist es möglich, den Stromfluss des Motors zu unterbrechen und so den Motor indirekt zu steuern: „[...] der Transistor wird entweder hochohmig oder komplett in die Sättigung geschaltet, so dass er annähernd wie ein geschlossener Schalter wirkt“ (1). Dadurch kann der Motor später ein- / ausgeschaltet werden. Im Arduino Programm werde ich die Basisschnittstelle über ein PWM-Signal ansteuern. PWM bedeutet „Pulsweitenmodulation“ (2) und sendet in einem bestimmten Abstand eine logische 1 oder 0 an die Schnittstelle: „Die digitale Steuerung wird nur im binären Format eingeschaltet (volle 5 V) oder ausgeschaltet (0 V)“ (2). Dieses Prinzip finden wir in vielen Geräten wieder. Dabei verwende ich im späteren Verlauf die Funktion „analogWrite(pin, value)“ (3). Auf diese Funktion gehe ich später noch ein.

So kann durch schnelles Ein- und Ausschalten eine LED verdunkelt werden: „Kann z.B. benutzt werden, um eine LED mit verschiedener Helligkeit leuchten zu lassen oder einen Motor mit unterschiedlicher Geschwindigkeit laufen zu lassen“ (3). Dabei sieht man diese Veränderung nicht durch das bloße Auge. Durch eine Kamera kann man dieses Flackern jedoch sichtbar machen.

Parallel zum Motor wurde ebenfalls eine Diode in Sperrrichtung der Stromquelle geschaltet. Solange der Strom dabei fließt und der Transistor geöffnet ist, hat diese Diode keinen Nutzen. Allerdings kann eine Induktion im Motor entstehen, wenn die Basis des Transistors schnell unterbrochen wird: „Sobald sich der Motor dreht, fungiert er auch als Generator und erzeugt somit eine Spannung“ (1). Dies ist auch bei mir der Fall, da ich (wie oben beschrieben) die Pulsweitenmodulation verwende. Diese Induktion führt zu einem sog. „Peak“ (Hochpunkt) der Spannung: „So können kurzzeitig mehrere tausend Volt induziert werden, die einen Zündfunken erzeugen“ (1). Dabei kann der Transistor zerstört werden. Abhilfe schafft eine Schutz- oder auch Freilaufdiode, durch die der Reststrom zurückgeleitet wird: „Durch eine in Sperrrichtung parallel geschaltete Diode wird dem Strom ermöglicht, durch die Diode weiterzufließen“ (1). Somit wird der Transistor entlastet und das Magnetfeld des Motors kann langsam abgebaut werden.

Die folgende Tabelle zeigt die Schnittstellen, die für den Arduino und die zusätzlichen Komponenten, die an dem Arduino angeschlossen werden, verwendet werden.

Komponenten	Schnittstellen	Arduino Schnittstellen
Transistor:	Basis	PD5 (Digital 5)
HC-05 Modul:	GND	GND
	VCC	3.3V
	RXD	TXD
	TXD	RXD
Servomotor:	GND	GND
	VCC	5V
	Digital IN	PD6 (Digital 6)

Tabelle 2: Schaltung der Schnittstellen vom Arduino mit den einzelnen Komponenten.

Der Transistor wird mit der digitalen Schnittstelle 5: „D (digital pins 0 to 7)“ (4) des Arduinos verbunden und kann später durch die Bezeichnung PD5 bzw. die Nummer 5 angesteuert werden. Die Ansteuerung über diese Bezeichnung wurde uns im Unterricht der 11. Klasse gezeigt, beziehungsweise erklärt.

Das HC-05 Modul ist das Bluetoothmodul, durch das am Ende die Daten der Controller-App empfangen werden (5). Er benötigt lediglich eine Spannung von 3.3V: „3,3-6 V Spannungstoleranz“ (5) und kann deshalb direkt an die 3.3V Schnittstelle angeschlossen werden. Zudem gilt standardmäßig Masse an Minus. Um Daten empfangen und senden zu können, besitzt das Modul die Schnittstellen RxD und TxD. RxD steht für „Receive data“ (6) und somit für das Empfangen von Daten. Beide Schnittstellen müssen kreuzweise verbunden werden. Die RxD Schnittstelle des Moduls muss mit TxD des Arduinos verbunden werden: „Connect Rx pin with Tx of Arduino“ (7). Ebenfalls muss die TxD Schnittstelle des Moduls mit RxD des Arduinos verbunden werden: „Connect Tx pin with Rx of Arduino“ (7). Dabei steht TxD für „Transmit data“ (6), also für das Senden von Daten.

Der Servomotor „SG90“ (8) wird zunächst standardmäßig mit dem Minuspol des Arduinos verbunden. Der VCC Eingang geht schließlich zum 5V Ausgang des Arduinos, da dieser die benötigte Betriebsspannung für den Motor braucht (vgl. „Betriebsspannung: 4.8V - 6V“ (8)). Zum Schluss wird der digitale Eingang mit der digitalen Schnittstelle 6 des Arduinos verbunden. So kann der Servomotor später über die Bezeichnung PD6: „D (digital pins 0 to 7)“ (4) angesteuert werden.

Da die Elektronik nun installiert wurde, kann es jetzt damit losgehen das Arduino zu programmieren, so wie die App, über die das Arduino später angesteuert wird.

Programmierung der Arduino-Logik

Nun wird das Arduino programmiert. Dazu benötige ich zuerst Bibliotheken, die in der Arduino Umgebung enthalten (8) ist und so nicht zusätzlich runtergeladen werden müssen. In der folgenden Abbildung sehen Sie die Inkludierung der Servo-Bibliothek.

```
1   #include <Servo.h>              // Libarie für Servomotor holen
```

Abbildung 2: Bibliotheken hinzufügen

„Servo.h“ ist die Bibliothek, mit der wir den Servomotor ansteuern werden.

```
3   Servo servoMotor;               // Servoobjekt erstellen
4
5   //nötigen Variablen vordefinieren
6   long gotDatas, motorLeistung, servoLeistung;
```

Abbildung 3: Objekte und Variablen erstellen

Im nächsten Schritt erstellen wir ein Objekt und drei integer. Erst wird ein Objekt (servoMotor) für die Servolenkung erstellt. Danach habe ich die long integer Variablen „gotDatas“,

„motorLeistung“ und „servoLeistung“ erstellt, da die empfangenen Werte später in dem reservierten Speicherplatz gespeichert werden. „motorLeistung“ und „servoLeistung“ arbeiten dann mit den Werten von „gotDatas“.

Jetzt geht es an die Main-Methode oder auch Setup-Methode genannt.

```
 8  void setup() {
 9    //HC modul
10    Serial.begin(9600);
11    pinMode(3, OUTPUT);
12
13    servoMotor.attach(5);     //Servo auf 5. PWM-Pin einstellen
14    //Alle Werte standartmäßig auf 0 setzen
15    gotDatas = 0;
16    motorLeistung = 0;
17    servoLeistung = 0;
18
19    return 0;
20  }
```

Abbildung 4: Startmethode

In Zeile 10 habe ich die Baudrahte des Bluetooth Moduls eingestellt. Das ist wichtig, da sonst keine Daten oder falsche Daten empfangen werden. Denn die Baudrahte von 9600 bestimmt, „dass [die Ein- und Ausgangspins] bis zu [9600] Mal von Null zu Eins oder Eins zu Null wechseln [können]“ (9).

Mit „pinMode(3, OUTPUT)“ (10) wird die dritte Schnittstelle als Ausgangspin festgelegt. Das heißt, dass Strom über den Pin geschaltet werden kann. Über „servoMotor.attach(5)“ wird dem Servomotor die fünfte Schnittstelle zugewiesen (8). Diese Schnittstelle ist wichtig, da es sich um eine PWM-Schnittstelle handelt.

Dann werden alle integer auf 0 gesetzt, um ohne zugewiesene Werte zu starten.

Zu guter Letzt wird 0 ausgegeben, da es sonst zu Problemen kommen kann.

Jetzt geht es an den wichtigsten Teil - die Schleife. In der Laufzeitumgebung des Arduinos ist eine Loop Schleife obligatorisch, denn über sie werden Befehle immer wieder ausgeführt. Vergleichbar ist die Schleife mit einer While Schleife, die nur auf true [while(true)] gesetzt ist. Also eine Schleife, die niemals endet.

```
22  void loop() {
23    if (Serial.available())
24    {
25      gotDatas = Serial.parseInt();                    //Einlesen der Gesamtzahl in (maximal 255.255)
26      Serial.println(gotDatas);
27
28      motorLeistung = gotDatas / 1000;                 //Ersten drei Zahlen für Motor reservieren (255213 / 1000 = 255)
29      servoLeistung = gotDatas - (motorLeistung * 1000);   //Letzten drei Zahlen für Servolenkung reservieren (255213 - (255 * 1000) = 213)
30
31    analogWrite(3, motorLeistung);                     //Leistung des Motors über Pin 5 (PD5)
32    servoMotor.write(servoLeistung);                   //Leistung des Motors über Pin 6 (PD6)
33    delay(300);
34    }
35  }
```

Abbildung 5: Loopschleife

Gehen wir die einzelnen Codezeilen einmal durch. In Zeile 23 startet die Schleife mit der Bedingung „Serial.available()“ (5). Das bedeutet, es wird in jedem Durchgang gefragt, ob Daten im seriellen Eingang - also RxD - zur Verfügung stehen. Wenn keine Daten empfangen werden, wird die Schleife einfach beendet und wieder ausgeführt.

In der Zeile 25 werden die Daten nun der long Variable „gotDatas“ zugewiesen. Eine long Variable wird verwendet, um die benötigten Zahlen bis 255180 zu empfangen. Eine int Variable wird lediglich auf einen Datenwert von „32Bit“ (11), zuzüglich negativer und positiver Zahlen, gesetzt. Da man jeweils 2^{15} für die negativen und positiven Zahlenwerte - und 2^{0} für die 1 - rechnet, könnte ich einen maximalen Zahlenwert von –32768 bis 32767 auf die Variable beziehen. Mit long sind Zahlenwerte im „64Bit“ (11) Bereich möglich. Also Zahlen von -2147483648 bis 2147483647. Das reicht also vollkommen aus.

Durch „Serial.println()“ gibt das Arduino Daten über die serielle Schnittstelle TxD aus (5). So können wir prüfen, ob die Zahl richtig empfangen wurde.

In den Zeilen 28-29 werden die ersten Tausenderstellen durch 1000 geteilt und so der Motorleistung zugewiesen. Dadurch fallen die Hunderterstellen weg. Im nächsten Schritt wird der Empfangene Wert von der Motorleistung, die mal 1000 gerechnet wird, subtrahiert. Dadurch erhalten wir die Hunderterwerte und können diese der Servomotorleistung zuweisen.

In den Zeilen 31-32 werden die entsprechenden Zahlen dem Motor, sowie dem Servomotor zugewiesen. Dabei wird für den Motor die 3. Schnittstelle verwendet, da diese eine PWM-Schnittstelle ist. Über „analogWrite(3, motorLeistung)“wird nun der Zahlenwert von 0 bis 255 in einen analogen Wert umgewandelt und über Pin drei ausgegeben (3). Das bedeutet, dass über einen bestimmten Zeitraum Strom oder kein Strom fließt. Die Servo library beinhaltet eine eigene Methode, zum Übermitteln von Daten an den Servomotor. In dieser „write()“ Methode wird nun ein Wert zwischen 0 und 180 Grad übermittelt (8).

Mit dem Delay von 300 Millisekunden in der Zeile 33 wird verhindert, dass die Schleife zu schnell durchläuft. Außerdem wird dem Servomotor so genug Zeit gegeben, sich auszurichten, denn ohne einen Delay würde der Servomotor, nach eigener Erfahrung, überdrehen.

Programmierung der Controller-App

Layouten mit Hilfe von XML

In der folgenden Grafik sehen sie die App „Arducraft“, welche ich mit Android Studio entwickelt habe. Android Studio ist sehr benutzerfreundlich, da XML als „Auszeichnungssprache“ (12) verwendet wird.

Aussehen und Funktionalität der App

Abbildung 5 beschreibt dabei das Aussehen am Anfang. Hier kann manuell über einen Button eine Verbindung aufgebaut werden.

Abbildung 6: Verbindungsansicht

In Abbildung 6 ist die eigentliche Controlleransicht zu sehen. Über dieses Layout kann man nun das Hovercraft steuern, beziehungsweise die Motoren ansteuern. Oben links in der Ecke wird angezeigt, ob das Smartphone mit dem Hovercraft verbunden ist. Über den linken Regler kann die Leistung des Motors eingestellt werden. Rechts in der Abbildung 6 kann der Servomotor von 0° bis 180° über einen Regler eingestellt werden. Der Schalter unten links setzt die Ausrichtung auf 0° und sperrt den Regler. So kann beispielsweise verhindert werden, dass das Hovercraft nach vorne getrieben wird, da die angetriebene Luft blockiert und nur unter das Gerüst gerichtet wird.

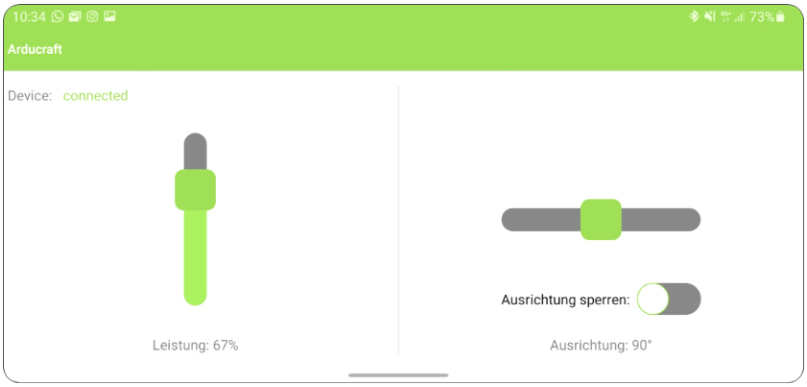

Abbildung 7: Controlleransicht

Was ist XML

XML steht für „eXtensible Markup Language“ und ist der Nachfolger von HTML und SGML, da die Sprache wie HTML einfach anzuwenden so flexibel wie SGML ist (12). Durch hierarchische Verschachtelungen und Anfangs-, sowie End-Tags, ähnelt die Auszeichnungssprache HTML sehr (s.V.).

Aufbau des Layouts

Im folgenden Abschnitt gehe ich näher auf einige Zeilen der entwickelten Layoutdatei ein.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      tools:context=".MainActivity">
8
9
10     <SeekBar
11         android:id="@+id/leistungBar"
12         android:layout_width="220dp"
13         android:layout_height="250dp"
14         android:max="255"
15         android:progress="0"
16         android:progressDrawable="@drawable/progress_bar_leistung"
17         android:rotation="-90"
18         android:thumb="@drawable/seekbar_thumb"
19         app:layout_constraintBottom_toBottomOf="parent"
20         app:layout_constraintEnd_toStartOf="@+id/divider"
21         app:layout_constraintStart_toStartOf="parent"
22         app:layout_constraintTop_toTopOf="parent"
23         app:layout_constraintVertical_bias="0.496" />
```

Abbildung 8: XML-Datei der App "Arducraft"

In der ersten Zeile wird die „Kopfdefinition“ (12) beschrieben, also dass es sich um ein XML-Format handelt. In Zeile zwei werden „[...] die Datentypen der Elemente und Attribute [...]“ (12) eines externen Schemas verwendet. Es handelt sich hier um das Android-Schema, welches Elemente wie zum Beispiel „SeekBar“ (Regler) oder „TextView“ (Textfeld) beinhaltet.

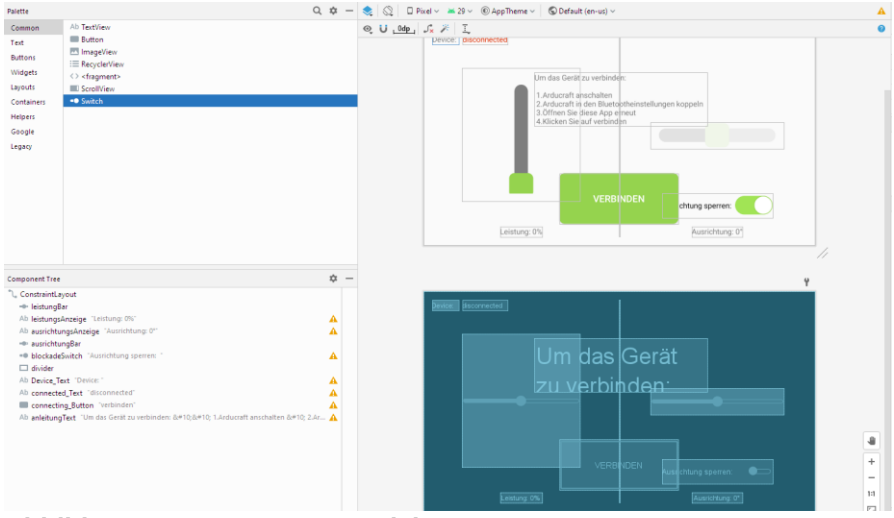

Abbildung 9: Drag & Drop Ansicht

Die Elemente können in Android Studio einfach über die Drag & Drop Funktion in eine Vorschauansicht eingebunden werden. Abbildung 8 zeigt diese Ansicht.

„Ein Namensraum wird als Attribut an ein Element (typischerweise das Wurzelelement) gebunden und kann dann von allen Elementen verwendet werden“ (12). So können wir mit Namensräumen beispielsweise eine ID vergeben. Dieses Element kann später mit Hilfe der ID in Java aufgegriffen werden. So wird die ID der „SeekBar“ in Abbildung 7 „leistungBar“ (Z.11) genannt. Aber auch Aspekte, wie zum Beispiel das Layout, können mit Namensräumen behandelt werden.

Zusammenfassung und Ausblick

Zum Schluss kann ich sagen, dass ich dem Ziel, ein Mini-Hovercraft fertigzustellen, nicht gerecht wurde. Die Programmierung wurde nicht erfolgreich beendet, da ich den Nutzen von Threads anfangs nicht verstanden habe. In diesem Projekt wird allerdings auch deutlich, dass Anwendungen oder Ziele, die mit einer Hardware verbunden sind, schnell zum Scheitern verurteilt sind, sobald die Hardware nicht mehr ordnungsgemäß funktioniert. Durch Sicherungen von alten Programmbeständen kann eine Software immer zurückgesetzt werden. Das ist bei Hardware nicht möglich. Zwar kann eine Verbindung zwischen Bluetoothmodul und Endgerät hergestellt werden, jedoch können die benötigten Daten nicht empfangen werden.

Im späteren Verlauf werde ich die App weiterentwickeln. Des Weiteren werde ich ein Gerüst erstellen und die Hardware in diesem Gerüst einbauen.

Literaturverzeichnis

1. **Heise.** Make Magazin. *Heise.* [Online] 06 2016. [Zitat vom: 21. 03 2021.] https://www.heise.de/select/make/2016/6/1482398401198797.

2. **EXP GmbH.** Arduino Tutorial - Pulsweitenmodulation (PWM). *exp-tech.de.* [Online] 24. 09 2018. [Zitat vom: 21. 03 2021.] https://www.exp-tech.de/blog/arduino-tutorial-pulsweitenmodulation-pwm.

3. **Arduino.** analogWrite(). *arduino.cc.* [Online] 21. 03 2021. [Zitat vom: 21. 03 2021.] https://www.arduino.cc/reference/de/language/functions/analog-io/analogwrite/.

4. —. Port Registers. *arduino.cc.* [Online] 21. 03 2021. https://www.arduino.cc/en/Reference/PortManipulation.

5. **Az-Delivery.** HC-05 6 Pin Wireless Bluetooth Transceiver Modul für Arduino - Deutsch. *az-delivery.de.* [Online] 21. 03 2021. https://www.az-delivery.de/a/downloads/-/52b83b126d06c3c1/5229d1c25e1791cc.

6. **DATACOM Buchverlag GmbH.** RxD (receive data). *itwissen.info.* [Online] 30. 10 2019. [Zitat vom: 21. 03 2021.] https://www.itwissen.info/receive-data-RxD-Empfangsdaten.html#:~:text=Receive%20Data%20(RxD)%20und%20Transmit,einer%20Datenendeinrichtung%20(DTE)%20%C3%BCbertragen..

7. **Tariq, Hammad.** Remote Controlled LED Using HC-05 Bluetooth, Arduino and Mobile Phone App. *instructables.com.* [Online] 21. 03 2021. https://www.instructables.com/Remotely-Control-LED-using-HC-05-Bluetooth-Arduino/.

8. **Az-Delivery.** Micro Servo SG90 - DEUTSCH. *az-deliver.de.* [Online] 21. 03 2021. https://www.az-delivery.de/a/downloads/-/c5a5d2a7b95b820c/a7fb71650d86f453.

9. **Eckert, Michael.** Was ist der Unterschied zwischen Bitrate und Baudrate? *computerweekly.com.* [Online] 29. 01 2016. [Zitat vom: 21. 03 2021.] https://www.computerweekly.com/de/antwort/Was-ist-der-Unterschied-zwischen-Bitrate-und-Baudrate.

10. **Arduino.** pinMode(). *arduino.cc.* [Online] 21. 03 2021. https://www.arduino.cc/reference/de/language/functions/digital-io/pinmode/.

11. **MPohlig.** int_double. *pohlig.de.* [Online] 2002. [Zitat vom: 21. 03 2021.] https://www.pohlig.de/Unterricht/Inf2002/Tag6/int_double.htm.

12. **Ullenboom, Christian.** Einführung in <XML>. *Java ist auch eine Insel.* https://www.rheinwerk-verlag.de/java-ist-auch-eine-insel/ : Rheinwerk Computing, 2016, S. 1123-1135.

Abbildungsverzeichnis

Tabellenverzeichnis

Eidesstaatliche Erklärung

Ich erkläre, dass ich die Projektarbeit ohne fremde Hilfe angefertigt und nur die im Literaturverzeichnis angeführten Quellen und Hilfsmittel benutzt habe und dass ich die Stellen der Projektarbeit, die im Wortlaut oder im wesentlichen Inhalt aus anderen Werken entnommen wurden, mit genauer Quellenangabe kenntlich gemacht habe. Verwendete Informationen aus dem Internet sind der Fachlehrerin bzw. dem Fachlehrer vollständig ausgedruckt zusammen mit einer Liste der erforderlichen Zugangsadressen (URLs) zur Verfügung gestellt worden.

_____ _____

Ort, Datum Unterschrift