Systems and Control Engineering

Name, Vorname	Aufgabe-/n
Frankline chenui	Design, Dokumentation

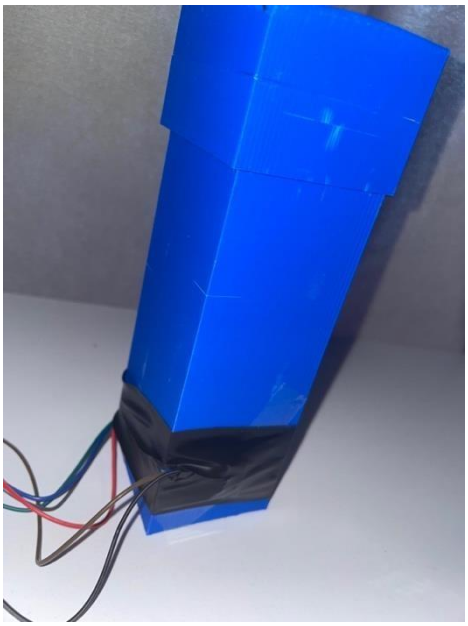
Inhaltsverzeichnis

1. Weintester (1. Projekt)

1.1. Design

Für den Weintester haben wir verschiedene Designideen entwickelt. Dabei war es wichtig, diese Ideen zunächst auf dem Papier umzusetzen, um Material zu sparen und zu überprüfen, ob sie sinnvoll sind. Unsere Hauptidee bestand darin, einen Quader zu bauen, in den die Reagenzgläser passen. Von oben sollte ein Deckel den Quader verschließen, um zu verhindern, dass externes Licht die Messwerte beeinflusst.

Um den Inhalt der Reagenzgläser zu testen und die Farbe des Weins zu bestimmen, haben wir den Lichtsensor und die RGB-LED seitlich etwa zwei Zentimeter von der Unterseite der Reagenzgläser platziert. Durch die tiefe Positionierung der Lichtsensor- und LEDKomponenten wird mit möglichst wenig Außenlicht gearbeitet und die Werte des Lichtsensors können besser differenziert werden. Vor der Konstruktion des Deckels haben wir die Reagenzgläser mit Papier umwickelt, um sicherzustellen, dass sich die Werte unabhängig vom Außenlicht nicht verändern. Unser Enddesign ist recht einfach und erfüllt die Funktionen eines Weintesters.

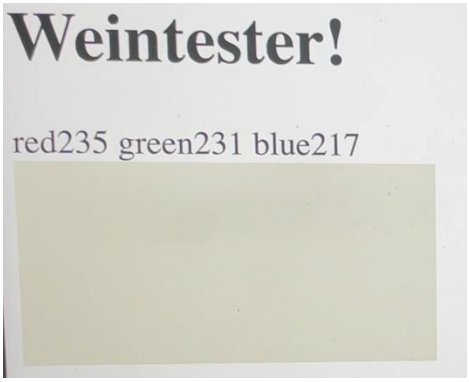


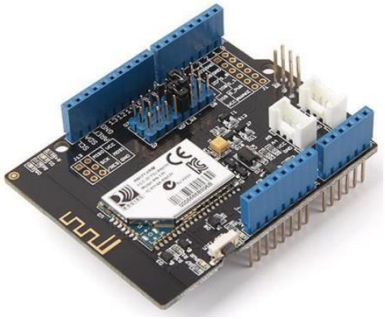
Beim Ausdrucken des Deckels für unseren Quader hatten wir das Problem, dass die Kanten des Deckels zu hoch waren und keine ausreichende Stütze hatten. Dadurch wies der Deckel am Ende eine leichte Neigung von der oberen Seite auf, was nicht wie erwartet war. Obwohl dies die Funktionalität nicht beeinträchtigt hat, haben wir daraus gelernt, wie wir in Zukunft effizienter mit dem Material umgehen können.

Um den Test optimal durchzuführen, haben wir zusätzlich die Reagenzgläser mit einem Stück Papier umwickelt und den Deckel oben darauf platziert. Auf diese Weise konnten wir sicherstellen, dass die Ergebnisse korrekt sind und die Farben genau gemessen werden können. Diese Kombination aus dem umwickelten Reagenzglas und dem Deckel hat dazu beigetragen, eine zuverlässige und genaue Analyse des Inhalts zu ermöglichen.

1.2. Wein erkennen

Um den Inhalt des Reagenzglases zu analysieren, haben wir den Weintester entwickelt. Dieser hilft uns, die Farbe des Inhalts im Reagenzglas zu bestimmen. Der Weintester verwendet einen Lichtsensor mit integriertem Widerstand, um die RGB-Werte des Inhalts zu messen. Dadurch können wir die genaue Farbe erfassen. Der Lichtsensor analysiert das von der RGB-LED ausgestrahlte Licht und gibt entsprechende analoge Werte abhängig von der Helligkeit des Lichts aus. Wir schalten zunächst die rote LED ein, dann die grüne und anschließend die blaue LED. Dadurch erhalten wir drei analoge Werte, die Werte für Grün und Blau werden mit 0.24 multipliziert. Das wird getan, weil die Werte einen Bereich von 0 bis 1023 haben können und die Darstellung der Farbe auf der Website nur Werte von 0 bis 255 annimmt. Da unser ABSQuader die Farbe Blau hatte, mussten wir die blauen Werte mit 0.22 multiplizieren. Ohne diese Berücksichtigung wären die Werte nicht vollständig korrekt.





Die resultierenden Informationen werden über das Arduino WiFi-Shield auf eine Webseite übertragen. Dort werden die RGB-Werte und die Farbe des Materials angezeigt. Durch diese Vorgehensweise ermöglicht der Weintester eine schnelle und zuverlässige Identifizierung der Farbe und des Materials im Reagenzglas. Dadurch erhalten wir am Ende die für uns wichtigen Informationen und wissen, was sich im Reagenzglas befindet.

Um den Weintester zu benutzen, müssen im Code die passenden Daten für ein erreichbares Wlan eingesetzt werden. Anschließend muss der Arduino gestartet werden und ein Gerät, welches sich im gleichen Wlan befindet, kann die Website des Weintesters über die Webadresse, die der Arduino über die serielle Schnittstelle ausgibt, erreichen. Nun kann ein Wein-Reagenzglas in den Weintester gestellt werden. Auf der Website werden nun Messwerte und eine Farbfläche angezeigt, die den getesteten Wein darstellen. Indem die Website im Browser neu geladen wird, werden die Messwerte und die Farbfläche aktualisiert.

1.3. Software

Die Software für unseren Weintester wurde entwickelt, um den Inhalt des Reagenzglases zu analysieren und die entsprechende Farbe der Flüssigkeit zu erkennen. Im Folgenden werden die Funktionen und der Ablauf des Codes erläutert.

Zu Beginn des Codes werden die erforderlichen Pins initialisiert. Pin 11, Pin 12 und Pin 13 werden als Ausgangspins für die Steuerung der RGB-LEDs festgelegt, während Pin A0 als Eingangspin für den Lichtsensor verwendet wird. Außerdem wird das WiFi shield geöffnet.

In der Hauptschleife des Codes werden nacheinander die Farben der RGB-LED eingeschaltet, um das Reagenzglas mit verschiedenen Lichtfarben zu beleuchten. Der Lichtsensorwert wird anschließend ausgelesen und den entsprechenden Farbvariablen zugewiesen. Die Reihenfolge der Beleuchtung ermöglicht es uns, die Farbwerte des Inhalts des Reagenzglases zu erfassen. Indem wir die RGB-Werte des reflektierten Lichts messen, können wir Rückschlüsse auf die Farbe der Flüssigkeit im Reagenzglas ziehen.

Zwischen dem Anschalten der LED, dem Auslesen des Sensors und dem erneuten Anschalten der LED mit der neuen Farbe wird jeweils kurz gewartet, da die Messwerte sonst eine Ungenauigkeit aufweisen. Die Werte werden auf die entsprechenden Variablen skaliert, um sie in einen geeigneten Bereich zu bringen. Abschließend werden die RGB-Werte über das WiFi shield ausgegeben. Die Website, die das WiFi shield öffnet hat eine Überschrift „Weintester!“, die drei Messwerte, die mit ihrer jeweiligen Farbe beschriftet werden und eine Fläche, auf der die drei Messwerte als eine RGB-Farbe ausgegeben werden.

Die Software ermöglicht es uns, den Inhalt des Reagenzglases zu analysieren und die Farbe des Materials zu erkennen. Dies unterstützt uns bei der Untersuchung und Dokumentation der verschiedenen Inhaltsstoffe oder Materialien, die im Reagenzglas enthalten sein können.

1.4. Code

```
#include <SoftwareSerial.h>
#include "WiFly.h"
#define SSID "Joudi"    //HERE WLAN SSID
#define KEY "12345678"     //HERE PASSWORD

#define AUTH WIFLY_AUTH_WPA2_PSK

int flag = 0;
int red = 0; int
```

```
green = 0; int
blue = 0;

// Pins' connection
// Arduino WiFly
// 2 <----> TX
// 3 <----> RX

SoftwareSerial wiflyUart(2, 3);        // create a WiFi shield serial object
WiFly wifly(&wiflyUart);               // pass the wifi shield serial object to the WiFly class

void        setup()        {
pinMode(11,   OUTPUT);
pinMode(12,   OUTPUT);
pinMode(13, OUTPUT);
  pinMode(A0, INPUT);
 wiflyUart.begin(9600); // start wifi shield uart port
Serial.begin(9600); // start the arduino serial port
delay(1000); // wait for initialization of wifly
wifly.reset(); // reset the shield
 delay(1000);

 //set WiFly params
 wifly.sendCommand("set ip protp 18"); // set the local comm port to 80  delay(100);
 wifly.sendCommand("set ip local 80\r"); // set the local comm port to 80  delay(100);
 wifly.sendCommand("set comm remote 0\r"); // do not send a default string when a
connection opens  delay(100);
 wifly.sendCommand("set comm open *OPEN*\r"); // set the string to open connection
delay(100);
 wifly.join(SSID, KEY, AUTH);  delay(5000);
 wifly.sendCommand("get ip\r");  char
c;
 while (wifly.receive((uint8_t *)&c, 1, 300) > 0) { // print the response from the get ip command
Serial.print((char)c);
 }
}

void        loop()        {
analogWrite(11,   255);
analogWrite(12,       0);
analogWrite(13, 0);
  delay(500); // Wait for 500 millisecond(s)
  red = analogRead(A0);
  delay(500); // Wait for 500 millisecond(s)
analogWrite(11,  0);     analogWrite(12,  0);
analogWrite(13, 0);
  delay(500); // Wait for 500 millisecond(s)
analogWrite(11, 0);   analogWrite(12, 255);
analogWrite(13, 0);
```

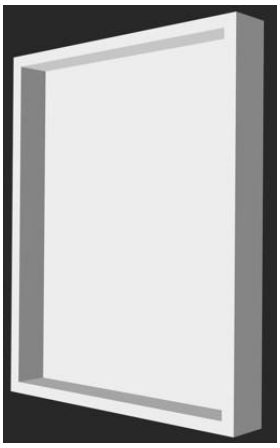
```
  delay(500); // Wait for 500 millisecond(s)  green
= analogRead(A0);
  delay(500); // Wait for 500 millisecond(s)
analogWrite(11,  0);     analogWrite(12,  0);
analogWrite(13, 0);
  delay(500); // Wait for 500 millisecond(s)
analogWrite(11,  0);     analogWrite(12,  0);
analogWrite(13, 255);
  delay(500); // Wait for 500 millisecond(s)  blue
= analogRead(A0);
  delay(500); // Wait for 500 millisecond(s)
analogWrite(11,  0);     analogWrite(12,  0);
analogWrite(13, 0);
  delay(500); // Wait for 500 millisecond(s)

  red=red*0.24;
green=green*0.24;
  blue=blue*0.22;

 if(wifly.available()) { // the wifi shield has data available
 if(wiflyUart.find("*OPEN*")) { // see if the data available by looking for the *OPEN* string
Serial.println("New Browser Request!");
 delay(1000); // delay enough time for the browser to complete sending its HTTP request string
// send HTTP header

 wiflyUart.println("HTTP/1.1 200 OK");
 wiflyUart.println("Content-Type: text/html; charset=UTF-8");
 wiflyUart.println("Content-Length: 250"); // length of HTML code appr. 24 + content
wiflyUart.println("Connection: close");  wiflyUart.println();
 // send webpage's HTML code  wiflyUart.print("<html>");
//6
 wiflyUart.print("<head><style> .farbiges-rechteck { width: 200px;  height: 100px;
background-color: rgb(");  wiflyUart.print(red);
 wiflyUart.print(",");

 wiflyUart.print(green);
wiflyUart.print(",");
wiflyUart.print(blue);
wiflyUart.println("); }</style></head>");
//6  wiflyUart.print("<body>"); //6
 wiflyUart.print("<h1>Weintester!</h1>"); // length 20
wiflyUart.print("red");//3   wiflyUart.println(red);//3
wiflyUart.print("green");//5
wiflyUart.println(green);//3
wiflyUart.print("blue");//4   wiflyUart.println(blue);//3
  wiflyUart.print("<div class=\"farbiges-rechteck\"></div>");
wiflyUart.print("</body>");  //7   wiflyUart.print("</html>");
//7
 }
```

```
}
}
```

2. Linien - Tracer (2. Projekt)

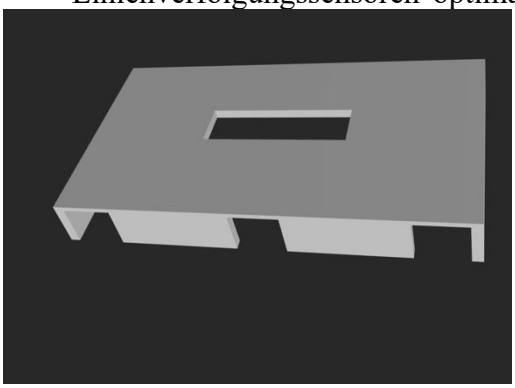
2.1. Design

Das Design für unser zweites Projekt erwies sich als nicht so einfach, wie wir es uns vorgestellt hatten. Obwohl die Idee, einen Linienverfolger zu bauen, nicht allzu kompliziert klang, stellten wir fest, dass es viele Aspekte zu beachten gab, während wir das Design entwickelten.



Ein wichtiger Aspekt war die Gewichtsverteilung, da der Linienverfolger problemlos nach vorne bewegt werden sollte. Die Räder hatten nicht viel Grip, was dazu führte, dass der Roboter sich kaum nach vorne bewegte, wenn das Gewicht nicht richtig verteilt war. Um dieses Problem anzugehen, haben wir die Powerbank als Stromquelle für den Arduino genutzt und sie oben platziert. Wir haben eine Art Hülle konstruiert, um die Powerbank stabil zu umhüllen. Die Position der Powerbank half uns auch dabei, den Schwerpunkt des Gewichts optimal zu verteilen.

Als nächstes haben wir uns Gedanken darüber gemacht, wie wir die drei Linienverfolgungssensoren optimal in das Design integrieren können, damit sie die Linien optimal erkennen können. Dabei ist es wichtig zu beachten, dass die Sensoren nur in einer bestimmten Höhe über dem Boden die Linien erkennen können. Bei uns haben wir die Sensoren etwa fünf Millimeter über dem Boden positioniert.

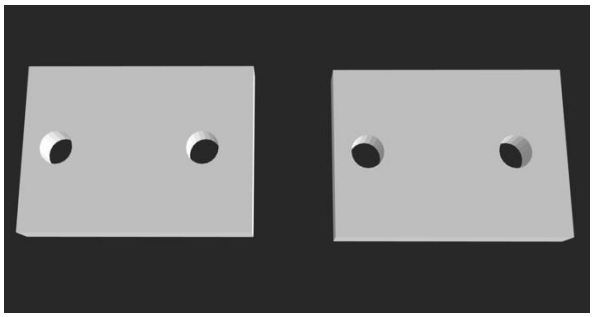


Im nächsten Schritt haben wir eine Frontkonstruktion entworfen, in der die Linienverfolgungssensoren und der

Infrarotsensor eingebaut werden können. Dabei haben wir darauf geachtet, den Infrarotsensor in der Mitte zu platzieren, damit der Roboter anhält, wenn er ein Hindernis vor sich erkennt.

Wir haben die Rückseite der Powerbank-Halterung genutzt, um den Arduino und das Steckbrett dort zu platzieren, um eine saubere und effiziente Verkabelung zu gewährleisten. Dafür haben wir kurze Jumperkabel aus unserem privaten Kit verwendet, da die Distanzen sehr kurz waren und es uns dadurch leichter fiel, den Überblick zu behalten.

Um Material zu sparen, haben wir für die Motoren zwei Halterungen entworfen und ausgedruckt, in die die Schrauben seitlich passen. Die Halterungen haben eine Größe von jeweils 27 Millimetern mal 20 Millimetern und zwei Löcher mit einem Durchmesser von drei Millimetern für die Schrauben. Wir haben die Toleranz für den 3D-Druck berücksichtigt, wodurch die Schrauben perfekt passten. Die Positionierung der Halterungen erfolgt mit einem seitlichen Abstand von ungefähr 20 Millimetern und einer Tiefe von ebenfalls etwa 20 Millimetern. Wir haben zusätzlich Kleberband auf den Rädern geklebt, da der Roboter ansonsten sich kaum nach vorne bewegt hat, somit konnte der Roboter problemlos sich vorwärtsbewegen.



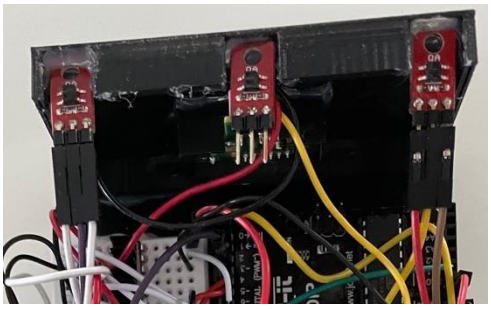
Die Teile haben wir einzeln ausgedruckt und dann mit Heißkleber zusammengefügt. Durch unsere Erfahrungen aus dem ersten Projekt konnten wir unsere Ideen besser umsetzen, da wir uns diesmal intensiv mit den Druckereinstellungen auseinandergesetzt haben. Leider hatten wir zwischenzeitlich das Problem, dass uns der 3D-Drucker (MakerBot) nicht zur Verfügung stand, was dazu führte, dass wir den Druck zeitlich verschieben mussten.

2.2. Linien verfolgen

Der Roboter verwendet zwei Linienverfolgungssensoren, die über der Bodenoberfläche angebracht sind. Diese Sensoren erfassen die Reflexion des Lichts und ermöglichen es dem Roboter, Linien zu erkennen und ihnen zu folgen. Wenn beide Sensoren eine schwarze

Oberfläche erkennen, setzt der Roboter seine Fahrt geradeaus fort. Er bewegt sich in einer geraden Linie und folgt der erkannten Linie.

Obwohl wir ursprünglich ein Design mit drei IR-Sensoren erstellt haben, haben wir festgestellt, dass unser Roboter auch mit nur zwei IR-Sensoren einwandfrei funktioniert. Nach eingehenden Tests und Evaluierungen haben wir festgestellt, dass die Verwendung von zwei Sensoren ausreicht, um die Linie effektiv zu verfolgen. Durch die Optimierung unseres Designs konnten wir dennoch die gewünschte Funktionalität erreichen und sicherstellen, dass unser Roboter zuverlässig die Linie verfolgt. Diese Anpassung hat es uns ermöglicht einige Jumperkabeln zu sparen und wir könnten die Software auch dadurch um einige Zeilen kürzen.



Wenn der rechte Sensor eine weiße Linie erkennt und der linke Sensor keine Linie erkennt, führt der Roboter eine Rechtskurve aus. Dadurch kann er die Linie wieder erfassen und ihr folgen. Umgekehrt, wenn der rechte Sensor keine Linie erkennt und der linke Sensor eine weiße Linie erkennt, führt der Roboter eine Linkskurve aus. Auf diese Weise kann er die Linie wieder erkennen und ihr folgen. Wenn beide Sensoren über einer weißen Oberfläche sind, hält der Roboter an. Der Roboter fährt nach dem Anhalten seine Route weiter, ohne dabei abzubiegen.

Der Infrarotsensor ist ein wichtiger Bestandteil unseres Roboters, da er ihm ermöglicht, Hindernisse in seiner Umgebung zu erkennen. Der Sensor arbeitet auf der Grundlage der Infrarotstrahlung und kann Objekte in seiner Nähe erfassen. Durch die Messung der reflektierten Infrarotstrahlung kann der Sensor feststellen, ob ein Hindernis vorhanden ist oder nicht.

Der Sensor ist mittig auf der Vorderseite des Roboters positioniert, um eine optimale Erfassungsbereich und Sichtlinie zu gewährleisten. Wenn der Infrarotsensor ein Hindernis erkennt, sendet er ein Signal an den Arduino, der daraufhin entsprechende Maßnahmen ergreift. In unserem Fall haben wir programmiert, dass der Roboter stoppt, wenn ein Hindernis erkannt wird. Dies ermöglicht es dem Roboter, Kollisionen zu vermeiden und sicher auf seiner Linie zu bleiben.

Durch die Kombination dieser Aktionen - Geradeausfahren, Rechtskurve, Linkskurve und Anhalten - kann der Roboter die Linie erkennen und ihr folgen, um die Linienverfolgungsfunktionalität zu erreichen.

2.3. Software

Wir steuern die IR-Sensoren mit den Pins R_S und L_S, um die Linien auf dem Boden zu erkennen. Die Motoren werden über die Pins in1, in2, in3 und in4 gesteuert, während die Enable-Pins enA und enB die Motorbridge anschalten. In der Setup-Funktion werden die Pins initialisiert.

In der Hauptschleife (loop) werden verschiedene Bedingungen überprüft, um das Verhalten des Roboters zu steuern. Wenn sowohl der rechte als auch der linke Sensor die schwarze Farbe erkennen, wird die forward-Funktion aufgerufen, um den Roboter vorwärtszubewegen. Wenn der rechte Sensor weiß und der linke Sensor schwarz erkennt, wird die turnRight-Funktion aufgerufen, um den Roboter nach rechts zu drehen. Wenn der rechte Sensor schwarz und der linke Sensor weiß erkennt, wird die turnLeft-Funktion aufgerufen, um den Roboter nach links zu drehen. Wenn beide Sensoren weiß erkennen, wird die stop-Funktion aufgerufen, um den Roboter anzuhalten. Zusätzlich wird überprüft, ob ein Hindernis über den A0-Pin erkannt wird.

Die Funktionen forward(), turnRight(), turnLeft() und stop() steuern die Motoren entsprechend, um die gewünschte Bewegung auszuführen. Die Funktion stop() stoppt den Roboter für drei Sekunden, bevor er wieder vorwärts fährt.

2.4. Code

```
#define in1 6  //Motor1  L293 Pin in1
#define in2 7  //Motor1  L293 Pin in1
#define in3 9  //Motor2  L293 Pin in1
#define in4 10 //Motor2  L293 Pin in1
#define enA 5  //Enable1 L293 Pin enA
#define enB 8  //Enable2 L293 Pin enB
#define R_S 2  //ir sensor Right
#define L_S 3  //ir sensor Left
```

```
void setup() {
pinMode(R_S, INPUT);
pinMode(L_S, INPUT);
pinMode(enA, OUTPUT);
pinMode(in1, OUTPUT);
pinMode(in2, OUTPUT);
pinMode(in3, OUTPUT);
pinMode(in4, OUTPUT);
pinMode(enB, OUTPUT);
digitalWrite(enA, HIGH);
digitalWrite(enB, HIGH);
}

void loop() {
//if Right Sensor and Left Sensor are at Black color then it will call forward function
if ((digitalRead(R_S) == 1) && (digitalRead(L_S) == 1)) { forward(); }

//if Right Sensor is White and Left Sensor is Black then it will call turn Right function   if
((digitalRead(R_S) == 0) && (digitalRead(L_S) == 1)) { turnRight(); }

//if Right Sensor is Black and Left Sensor is White then it will call turn Left function   if
((digitalRead(R_S) == 1) && (digitalRead(L_S) == 0)) { turnLeft(); }


//if Right Sensor and Left Sensor are at White color then it will call Stop function
if ((digitalRead(R_S) == 0) && (digitalRead(L_S) == 0)) { stop(); }     if
(digitalRead(A0) == 1) {stop();}   delay(200);
}
//move forword void forward() {     digitalWrite(in1,
HIGH);  //Right Motor forword Pin

  digitalWrite(in2, LOW);  //Right Motor backword Pin
digitalWrite(in3, LOW);  //Left Motor backword Pin
digitalWrite(in4, HIGH);  //Left Motor forword Pin
```

```
}

void turnRight() {  //turnRight   digitalWrite(in1,
LOW);  //Right Motor forword Pin   digitalWrite(in2,
LOW);  //Right Motor backword Pin   digitalWrite(in3,
LOW);  //Left Motor backword Pin   digitalWrite(in4,
HIGH);  //Left Motor forword Pin
}

void turnLeft() {  //turnLeft   digitalWrite(in1, HIGH);
//Right Motor forword Pin   digitalWrite(in2, LOW);
//Right Motor backword Pin   digitalWrite(in3, LOW);
//Left Motor backword Pin   digitalWrite(in4, LOW);
//Left Motor forword Pin
}

void stop() {  //stop
  digitalWrite(in1, LOW);  //Right Motor forword Pin
digitalWrite(in2, LOW);  //Right Motor backword Pin
digitalWrite(in3, LOW);  //Left Motor backword Pin
digitalWrite(in4, LOW);  //Left Motor forword Pin
delay(3000);              // wait 3 seconds
digitalWrite(in1, HIGH);  //Right Motor forword Pin
digitalWrite(in4, HIGH);  //Left Motor forword Pin
}
```

3. Balancierer (3. Projekt)

3.1. Design

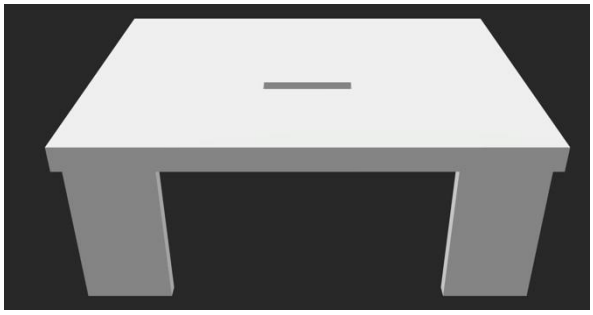
Die Konstruktion unseres Designs besteht aus drei Platten, die mithilfe von Stützen miteinander verbunden sind. Jede Platte wurde separat ausgedruckt, um eine präzise Fertigung zu gewährleisten. Dadurch konnten wir die spezifischen Anforderungen und Funktionen jeder einzelnen Platte besser berücksichtigen. Die Einzelausdrucke ermöglichten uns außerdem eine flexible Montage und Anpassung der Platten, um sicherzustellen, dass sie nahtlos zusammenpassen und die gewünschte Struktur des Roboters bilden. Diese Vorgehensweise erleichterte auch die Handhabung der einzelnen Komponenten während des Montageprozesses und ermöglichte eine effiziente und präzise Zusammenstellung des Designs.

Die Wahl der Stromversorgung für den Arduino war eine wichtige Fragestellung. Wir hatten die Möglichkeit, eine große Powerbank oder einen Batteriebehälter für drei AA-Batterien zu verwenden. Wir haben uns für die Powerbank entschieden. Daraufhin haben wir eine spezielle Hülle entwickelt, ähnlich der des Line-Tracer-Roboters, um die Powerbank sicher zu verstauen. Diese Hülle ist etwa zwei Millimeter dicker an den Rändern, um sicherzustellen, dass die

Motoren den Roboter problemlos vorwärts und rückwärts bewegen können. Auf beiden Seiten der Hülle befinden sich jeweils zwei Löcher mit einem Radius von 1,5 Millimeter und einem Abstand von 18 Millimetern zueinander. Diese Löcher dienen zur Befestigung der Motoren. Die Hülle musste leicht angepasst werden, um Platz für die Schraubenköpfe zu schaffen.

Die zweite Platte ist einfach konstruiert und etwas kürzer in der Länge als die unterste Platte, etwa um 20 Prozent. Sie verfügt über vier Stützen mit einer Länge von jeweils 41 Millimetern und einer Tiefe von drei Millimetern. Auf dieser Platte sind der Arduino und das Steckbrett für die Verbindungselemente platziert. Beide Komponenten liegen nebeneinander und haben ein geringes Gewicht.

Die dritte Platte in unserem Design übernimmt eine wichtige Rolle als Träger für den MPU6050-Sensor, der für die Gleichgewichtsfunktion des Roboters verantwortlich ist. Der



MPU6050-Sensor ist ein leistungsstarker Inertialsensor, der Beschleunigung und Gyroskopdaten misst. Er erkennt Veränderungen in der Lage und Bewegung des Roboters und liefert wertvolle Informationen für die Regelung und Steuerung des Gleichgewichts. Um sicherzustellen, dass der MPU6050-Sen sor optimal funktioniert, haben wir eine kleine Aussparung auf der Platte geschaffen, um Platz für die Kabel des Sensors zu bieten. Dadurch liegt der Sensor flach und stabil auf der Platte, sodass er präzise Messungen durchführen kann. Um sicherzustellen, dass der MPU6050-Sensor optimal funktioniert, haben wir eine kleine Aussparung auf der Platte geschaffen, um Platz für die Kabel des Sensors zu bieten. Dadurch liegt der Sensor flach und stabil auf der Platte, sodass er präzise Messungen durchführen kann.

Durch die Konstruktion mit den drei Platten und dem Schwerpunkt des Roboters am unteren Ende hatten wir folgende Vorteile beim Programmieren: Zum einen konnte die hohe Position des Sensors dazu beitragen ein Umkippen des Balancierers so früh wie möglich zu erkennen und zum anderen musste durch den niedrigen Schwerpunkt nicht so viel kippendes Gewicht aufgefangen werden, wenn der Balancierer fällt.

3.2. Balancieren

Es gab nicht viele Anforderungen an den Balancierer, er sollte mithilfe einer inertialen Messeinheit auf zwei motorbetriebenen Rädern balancieren.

Die Motoren unseres Balancierers werden nicht direkt durch den Arduino angetrieben, sondern über eine Motorbridge. Das hat zum einen den Grund, dass die Motoren sich ohne die Bridge nur in eine Richtung drehen könnten und zum anderen kann über die Bridge der 5 Volt Pin des Arduino benutzt werden, dieser liefert nämlich mehr Stromstärke als die Output-Pins des Arduinos. Die erhöhte Stromstärke hilft dabei die Motoren schneller und reibungsloser laufen zu lassen. Der MPU6050-Sensor ist an 5 Volt, Ground und zwei Analoge Pins des Arduinos angeschlossen.

Da der MPU6050-Sensor über eine Winkelmessung und eine Beschleunigungsmessung in drei Richtungen verfügt, können beide Messungen bei der Erkennung des Ungleichgewichts helfen. Wenn der Balancierer anfängt zu fallen, kann man eine Veränderung der Winkelmesswerte erst zu spät erkennen, um die Motoren noch rechtzeitig anzuschalten. Das bedeutet, dass wir den Beginn des Fallens mit der Beschleunigungsmessung erkennen mussten, allerdings reicht dieser Wert alleine nicht aus, da aus den Beschleunigungsmesswerten nicht hervorgehen würde, ob sich der Balancierer wieder aufgestellt hat.

3.3. Software

Um die Software des Balancierers zu entwickeln, mussten wir zuerst verstehen wie mit dem MPU6050-Sensor gearbeitet werden kann. Der MPU6050-Sensor wird nicht wie die meisten Arduinosensoren über einen oder mehrere Pins angesprochen, sondern zum Beispiel mithilfe der Wire Library. Mit dieser Library kann eine Art Übertragungskanal gestartet werden, über diesen Kanal können dann Daten an den Sensor gesendet und Daten von ihm empfangen werden. Die Daten, die an den Sensor gesendet werden sind zum Beispiel Konfigurations-, Start- oder Stopbefehle. Um die Befehle zu unterscheiden werden sie mit Register-Adressen an den Sensor gesendet. Die Bits innerhalb der Register ergeben dann den Befehl. Im Datenblatt haben wir nachgelesen welche Bitfolgen wir an welche Register schicken mussten um mit dem Sensor arbeiten zu können.

Zu Beginn des Codes wird die Wire Library eingebunden und einige Standartwerte definiert. Zum einen werden Register-Adressen, die oft benutzt werden definiert und zum anderen werden die Pins für die Motorbridge definiert. Anschließend werden zwei Enums definiert, sie sind dazu da die Messreichweiten für die Beschleunigungsmesswerte und die Winkelmesswerte des Sensors zu setzten. Die Messreichweiten werden an in zwei unterschiedlichen Registern mit den Bits drei und vier gesetzt. Die Position des Wertes im jeweiligen Enum wird also beim beschreiben des Konfigurationsregisters binär betrachtet, diese Bitwerte werden dann beim beschreiben des Registers um drei Bits nach links geshiftet um auf die Bits drei und vier zu fallen. Das Passiert dann in den Methoden setAccRange und setGyrRange.

Nach den Enums werden noch zwei 16 Bit Integer Variablen und ein Zeichen Array angelegt. Die Integer Variablen sind für die Speicherung der Beschleunigung in X-Richtung und für den Winkel in Y-Richtung, das sind die beiden Messwerte, die die Fallrichtungen unseres Balancierers darstellen können. Das Zeichen Array ist für die Ausgabe der Zahlen da.

Im Setup beginnt dann die Serielle und die Wire Schnittstelle. Über die Wireschnittstelle wird der Sensor dann aktiviert und die Messreichweiten gesetzt, die Pins für die Motorbridgesteuerung werden ebenfalls im Setup als Output initialisiert.

In der Loop-Schleife des Codes wird eine Datenübertragung über die Wireschnittstelle begonnen, anschließend wird die Register-Adresse 0x3B zum Sensor übertragen und danach wird eine Anfrage nach zwei Registern gestellt. Das bedeutet, dass wir im nächsten Schritt zwei Register auslesen können, unzwar 0x3B und das danach, also 0x3C. Aus diesen beiden Registern kann der Wert für die Beschleunigung in X-Richtung ausgelesen werden. Die beiden Register werden zusammen durch einen veroderten Bitshift in den passenden 16 Bit Integer Wert geschrieben. Der Winkelmesswert wird auf die gleiche Weise ausgelesen, nur mit dem Unterschied, dass das erste Register die Adresse 0x45 hat.

An dieser Stelle im Code wird entschieden, ob der Balancierer stoppen, vorwärts oder rückwärts fahren soll. Wir haben verschiedenste Werte ausprobiert, verschiedenste Kombinationen aus Winkel und Beschleunigung. Ebenfalls haben wir die Messabstände und das Ein- und Ausschalten des Sensors variiert, aber leider konnte der Balancierer bei keiner Kombination wirklich balancieren. Deswegen ist der hier angefügte Code nur eine einfache Version, die bei gewissen Beschleunigungswerten stoppt, vorwärts oder rückwärts fährt.

Am Ende der Loop-Schleife werden die die beiden gemessenen Werte in der seriellen Schnittstelle ausgegeben und abschließend wartet der Arduino 200 Millisekunden.

Nach der Loop-Schleife sind noch einige Methoden implementiert, die alle eine unterschiedliche Aufgabe haben und somit die Loop-Schleife übersichtlicher machen. Die Methoden übernehmen zum Beispiel die Ausgabe der Werte, setzten die Messwertreichweiten, starten den MPU6050-Sensor, schalten den MPU6050-Sensor ab, schreiben Werte in bestimmte Register oder steuern die Motorbridge und damit die Bewegung des Balancierers.

3.4. Code

```
#include "Wire.h"
#define MPU6050_ADDR            0x68
#define MPU6050_GYRO_CONFIG     0x1B
#define MPU6050_ACCEL_CONFIG    0x1C
#define MPU6050_PWR_MGT_1       0x6B
#define MPU6050_SLEEP           0x06
```

```
#define enA 3//Enable1 L293 Pin enA
#define in1 4 //Motor1  L293 Pin in1
#define in2 5 //Motor1  L293 Pin in1
#define in3 7 //Motor2  L293 Pin in1
#define in4 6 //Motor2  L293 Pin in1

#define enB 8 //Enable2 L293 Pin enB  typedef
enum {
  MPU6050_ACC_RANGE_2G,  // +/- 2g (default)
  MPU6050_ACC_RANGE_4G,  // +/- 4g
  MPU6050_ACC_RANGE_8G,  // +/- 8g
  MPU6050_ACC_RANGE_16G // +/- 16g
} mpu6050_acc_range;
typedef enum {
  MPU6050_GYR_RANGE_250,  // +/- 250 deg/s (default)
  MPU6050_GYR_RANGE_500,  // +/- 500 deg/s
  MPU6050_GYR_RANGE_1000, // +/- 1000 deg/s
  MPU6050_GYR_RANGE_2000  // +/- 2000 deg/s
} mpu6050_gyr_range;
int16_t accX, gyroY;
char result[7]; void
setup() {
Serial.begin(9600);
  Wire.begin();
  MPU6050_wakeUp();
setAccRange(MPU6050_ACC_RANGE_16G);
setGyrRange(MPU6050_GYR_RANGE_250);
pinMode(enA, OUTPUT);   pinMode(in1, OUTPUT);
pinMode(in2, OUTPUT);   pinMode(in3, OUTPUT);
pinMode(in4, OUTPUT);   pinMode(enB, OUTPUT);
digitalWrite(enA, HIGH);   digitalWrite(enB, HIGH);
delay(300); } void loop() {
  MPU6050_wakeUp();
```

```
  Wire.beginTransmission(MPU6050_ADDR);
  Wire.write(0x3B);
  Wire.endTransmission(false);
  Wire.requestFrom(MPU6050_ADDR,      2,      true);
accX = Wire.read() << 8 | Wire.read();
  Wire.beginTransmission(MPU6050_ADDR);
  Wire.write(0x45);
  Wire.endTransmission(false);
  Wire.requestFrom(MPU6050_ADDR, 2, true);
gyroY  =  Wire.read()  <<  8  |  Wire.read();
MPU6050_sleep();        if  (accX  <  -400)  {
forward();     }  else  {        if  (accX  >  400)  {
backward();     } else {      stop();     }

  }
  Serial.print("AcX = "); Serial.print(toStr(accX));
  Serial.print(" | GyY = "); Serial.print(toStr(gyroY));
  Serial.println();

  delay(100);
} char* toStr(int16_t i) {
sprintf(result,  "%6d",  i);
return result;
} void setAccRange(mpu6050_acc_range range)
{
  writeRegister(MPU6050_ACCEL_CONFIG, range << 3);
} void setGyrRange(mpu6050_gyr_range range) {
writeRegister(MPU6050_GYRO_CONFIG, range << 3);
}
void              MPU6050_wakeUp()              {
writeRegister(MPU6050_PWR_MGT_1,   0);
delay(30);
}
```

```
void MPU6050_sleep() {    writeRegister(MPU6050_PWR_MGT_1,
1 << MPU6050_SLEEP);
}
void writeRegister(uint16_t reg, byte value) {
  Wire.beginTransmission(MPU6050_ADDR);
  Wire.write(reg);
  Wire.write(value);
  Wire.endTransmission(true);
}
void forward(){   digitalWrite(in1, HIGH); //Right
Motor forword Pin  digitalWrite(in2, LOW);  //Right
Motor backword Pin  digitalWrite(in3, LOW);  //Left
Motor backword Pin  digitalWrite(in4, HIGH); //Left
Motor forword Pin
}

void backward(){  digitalWrite(in1, LOW);  //Right
Motor forword Pin  digitalWrite(in2, HIGH); //Right
Motor backword Pin   digitalWrite(in3, HIGH);
//Left Motor backword Pin  digitalWrite(in4, LOW);
//Left Motor forword Pin
}

void stop(){  digitalWrite(in1, LOW); //Right Motor
forword Pin  digitalWrite(in2, LOW); //Right Motor
backword Pin  digitalWrite(in3, LOW); //Left Motor
backword Pin  digitalWrite(in4, LOW); //Left Motor
forword Pin
}
```