



DIPLOMARBEIT

SumoBots

Schuljahr 2023/24

Ausgeführt im Schuljahr 2023/24 von:

EISMANN Bastian	5AHEL
LUCUT Ioan Lukas	5AHEL
ZICKLER Yannick	5AHEL

Betreuer/Betreuerin:

DI (FH) Michael Zatl

Erklärung über die eigenständige Verfassung der Diplomarbeit

Wir, die Herren Bastian EISMANN, Ioan Lukas LUCUT und Yannick ZICKLER, Schüler der Klasse 5AHEL der Höheren Technischen Bundeslehranstalt Wien 10, erklären hiermit an Eides statt, dass wir die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst haben, einschließlich auch andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die benutzten Quellen, wörtlich und inhaltlich entnommenen Stellen, als solche erkenntlich in der Diplomarbeit gekennzeichnet haben.

Wien am 31.05.2024

Verfasser:

Bastian EISMANN | 5AHEL U: _____

Ioan Lukas LUCUT | 5AHEL U: _____

Yannick ZICKLER | 5AHEL U: _____

1 Danksagung

Mit den folgenden Zeilen möchten wir die Gelegenheit nutzen, uns bei allen Unterstützern unserer Diplomarbeit zu bedanken.

Wir möchten unserem Betreuer unseren aufrichtigen Dank aussprechen. Sein Vertrauen in unsere Fähigkeiten sowie sein kontinuierliches Feedback zu den eingereichten Protokollen und Versionen der Diplomarbeit haben wesentlich zum Fortschritt und zur Entwicklung des Projekts beigetragen.

Unser Dank gilt ebenfalls den Werkstätten-Lehrern Weiss und Helfer, die uns während des gesamten Jahres, insbesondere bei der Herstellung der Platinen wertvolle Unterstützung zukommen ließen. Ihre Fachkenntnisse und Hilfsbereitschaft haben entscheidend zu unserem Projekterfolg beigetragen.

Ein besonderer Dank gebührt Professor Jahn, der uns im Bereich Mikrocontroller kontinuierlich mit Hilfe und Ratschlägen unterstützte. Insbesondere in anspruchsvollen Phasen erwies sich seine motivierende Unterstützung als wertvoll und trägt ebenfalls einen großen Bestandteil zur Vollendung der Diplomarbeit bei.

Ebenfalls möchten wir unsere Dankbarkeit gegenüber der Direktion der HTL Wien 10 ausdrücken, die uns großzügigerweise Zugang zu den Räumlichkeiten und Einrichtungen gewährt hat. Ihre Unterstützung hat maßgeblich zum Gelingen unseres Projekts beigetragen.

Abschließend möchten wir unseren Familien von Herzen danken. Selbst in den schwierigsten Phasen, geprägt von den Höhen und Tiefen des schulischen Stresses, haben sie uns stets unterstützt und motiviert.

Inhaltsverzeichnis

1	DANKSAGUNG	3
2	ZUSAMMENFASSUNG (DEUTSCH)	7
2.1	ABSTRACT (ENGLISH)	8
3	EINLEITUNG – PROJEKTBESCHREIBUNG.....	9
3.1	VORGESCHICHTE – AUSGANGSLAGE.....	12
3.1.1	<i>Projektwoche 2023</i>	12
3.1.2	<i>CLIL 2023</i>	12
4	STRUKTUR DES PROJEKTS – ARBEITSPAKETE	13
4.1	OBJEKTSTRUKTURPLAN (OSP) / ARBEITSPAKETE	13
4.2	PROJEKTSTRUKTURPLAN (PSP)	14
4.3	ARBEITSPAKET 1 3D-DESIGN (EIS).....	14
4.4	ARBEITSPAKET 2 HARDWARE-ENTWICKLUNG (LUC)	14
4.5	ARBEITSPAKET 3 SOFTWARE-ENTWICKLUNG (ZIC).....	15
4.6	ARBEITSPAKET 4 PROJEKTMANAGEMENT	15
4.7	ARBEITSPAKET 5 BEDIENUNGSANLEITUNG	15
5	3D-DESIGN UND KONSTRUKTION (EIS)	16
5.1	3D-DRUCKER BEGRIFFSERKLÄRUNGEN	16
5.1.1	<i>Vom CAD-File zum Drucken</i>	16
5.1.2	<i>3D Drucker</i>	18
5.2	MATERIALEIGENSCHAFTEN	19
5.2.1	<i>PLA, PLA+</i>	19
5.2.2	<i>ABS</i>	20
5.3	KONSTRUKTIONSVORGÄNGE.....	21
5.3.1	<i>Antriebsarten</i>	21
5.3.1.1	<i>Zahnriemen</i>	21
5.3.1.2	<i>Zahnrad</i>	22
5.3.1.3	<i>Andere Prototypen</i>	24
5.3.1.4	<i>Antriebs- und Passivräder</i>	25
5.3.2	<i>Rahmen des Roboters</i>	26
5.3.2.1	<i>Gehäuse des Controllers</i>	33
5.3.3	<i>Verschraubungen und Normen</i>	36
5.3.4	<i>Embedded Components (EC)</i>	37
6	HARDWAREENTWICKLUNG (LUC).....	39
6.1	ENTWICKLUNGSABLAUF	39
6.2	HARDWARE-DESIGN ROBOTER REVISION A	40
6.2.1	<i>Roboter Revision A V1.00 PCB-Design</i>	41
6.2.2	<i>Roboter Revision A V1.00 Erkenntnisse</i>	42
6.3	FUNKMODUL NRF24L01	43
6.3.1	<i>Spannungsversorgung des Funkmoduls</i>	44
6.3.2	<i>Spannungsschwankungen des NRF24L01</i>	45
6.3.3	<i>Störungen beim NRF24L01+</i>	46
6.4	KOMPONENTEN DES ROBOTERS REVISION A	46
6.4.1	<i>Motorshield</i>	46
6.4.2	<i>H-Brücke</i>	47
6.4.3	<i>Logik-Pegel-Wandler</i>	48
6.4.4	<i>Serial Peripheral Interface Bus</i>	50
6.4.5	<i>Pierce-Oszillatator</i>	51
6.4.6	<i>ESP32 mit Kamera</i>	52
6.5	HARDWARE-DESIGN ROBOTER REVISION B	53
6.5.1	<i>Roboter Revision B V1.00 PCB</i>	55

6.5.2	<i>Roboter Revision B V1.00 Erkenntnisse</i>	56
6.6	KOMPONENTEN DES ROBOTERS REVISION B	56
6.6.1	<i>I²C-Bus</i>	57
6.6.2	<i>OLED Display SSH1106</i>	58
6.6.3	<i>Vibrationssensor SW-420</i>	59
6.6.4	<i>Farbsensor TCS3200</i>	59
6.6.5	<i>Infrarotsensor Sharp GP2Y0A21YK0F</i>	60
6.6.6	<i>Gyroskop MPU6050</i>	61
6.7	HARDWARE-DESIGN CONTROLLER	63
6.8	CONTROLLER PCB-DESIGN	64
6.9	CONTROLLER ERKENNTNISSE	65
6.10	HARDWARE-KOMPONENTEN CONTROLLER	65
6.10.1	<i>Trigger</i>	65
6.10.2	<i>LCD-Display</i>	66
6.10.3	<i>5V-Längsregler 7805</i>	66
6.10.4	<i>Joystick</i>	67
6.10.5	<i>Raspberry Pi Zero 2W und Touchscreen-Display</i>	67
7	SOFTWAREENTWICKLUNG (ZIC)	68
7.1	STYLEGUIDE	68
7.2	NETZWERK	69
7.2.1	<i>Funkmodul</i>	70
7.3	CONTROLLER	72
7.4	ROBOTER	74
7.4.1	<i>Motorsteuerung</i>	74
7.4.2	<i>Sensoren</i>	77
7.4.3	<i>Logging</i>	78
7.5	SERVER	78
7.5.1	<i>Empfangen der Sensordaten</i>	79
7.5.2	<i>Website</i>	80
7.6	DEVELOPMENT	84
7.6.1	<i>Programmierung</i>	84
7.6.2	<i>Probleme</i>	85
8	PROJEKTMANAGEMENT	86
8.1	VERWALTUNG	86
8.2	ERFASSUNG DER ARBEITSZEITEN UND REPORTING	86
8.2.1	<i>Arbeitszeiterfassung 04.09.2023 – 04.02.2024</i>	87
8.2.2	<i>Arbeitszeiterfassung 12.02.2024 - 05.04.2024</i>	88
9	BEDIENUNGSANLEITUNG	89
10	BUSINESS CASE	89
11	FINALE ERGEBNISSE	90
12	TABELLEN UND ABBILDUNGSVERZEICHNIS	91
13	LITERATURVERZEICHNIS UND QUELLENANGABEN	92
14	ANHÄNGE	94
14.1	ARBEITSZEITVERFASSUNG 2023/24	94
14.2	PROJECT-LIBRE NETZPLAN	95
15	3D-MODELLE UND KONSTRUKTIONSZEICHNUNGEN	96
16	SCHALTPLÄNE UND PCB DESIGNS	103
16.1	ROBOTER REV A V1.00 SCHALTPLAN	103
16.2	ROBOTER REV A V1.00 PCB	104
16.3	ROBOTER REV B V1.00 SCHALTPLAN	105
16.4	ROBOTER REV B V1.00 PCB	106

16.5	CONTROLLER REV A V1.00 SCHALTPLAN.....	107
16.6	CONTROLLER REV A V1.00 PCB.....	108
17	PROGRAMM-CODES.....	109
17.1	CONTROLLER.H	109
17.2	CONTROLLER.INO	110
17.3	ROBOTER.H	113
17.4	ROBOTER.INO.....	114
17.5	SERVER.PY	118
17.6	STYLE.CSS	121
17.7	INDEX.JS.....	123
17.8	INDEX.HTML.....	129
17.9	ABOUT.HTML.....	131

2 Zusammenfassung (Deutsch)

Das Ziel dieser Diplomarbeit bestand darin, mehrere Fahrzeugroboter zu entwickeln, die in einer Arena sich gegenseitig aus der Arena stoßen sollen. Die Steuerung der Roboter erfolgt durch einen Nutzer mithilfe eines Controllers, der über eine Funkverbindung mit den Robotern kommuniziert. Um sicherzustellen, dass die Roboter die Arena nicht verlassen, ist jeder Roboter mit einem Farbsensor ausgestattet, der erkennt, wenn der Roboter den Bereich der Arena verlässt. Zusätzlich sollte die Arena über weitere Funktionen verfügen, wie beispielsweise eine Punkteanzeige, die jedoch nicht im Rahmen dieser Diplomarbeit behandelt wurden.

Hierbei wird die Elektronik sowohl für den Roboter als auch für den Controller mithilfe von KiCAD entworfen. Dabei werden Schaltpläne und Leiterplatten (PCBs) entwickelt, die anschließend von einem externen Lieferanten (Aisler) hergestellt werden. Die Fernsteuerung des Roboters wird durch das NRF24L01 Funkmodul ermöglicht. Beide Platinen sind mit AVR-Mikrocontrollern ausgestattet, was die Programmierung über die Arduino-IDE ermöglicht. Die Programmierung des ATMEGA2560 für den Roboter erfordert die Verwendung des SPI-Busses, da es sich bei diesem um ein SMD-Mikrocontroller handelt.

Der Code des Controllers ist dafür verantwortlich, die Eingabeelemente zu lesen und die Daten an den Roboter zu übermitteln. Der Roboter wiederum liest diese Werte ein, berechnet die erforderliche Motorleistung für die linken und rechten Motoren und gibt diese Werte als PWM-Signal an das Motorshield weiter. Darüber hinaus ist die Schaltung des Roboters mit verschiedenen Sensoren ausgestattet, deren Messwerte an einen Raspberry Pi übertragen werden. Dort werden die Daten grafisch auf einem Webserver dargestellt.

In der Diplomarbeit spielten die Auswahl des richtigen Materials und die Kalibrierung der 3D-Drucker eine zentrale Rolle. Die Gewährleistung des Druckerfolgs erforderte wesentliche Verbesserungen am Drucker. Ein weiterer entscheidender Schwerpunkt lag auf der Modellierung von Gehäusen, Rädern, Antriebssystemen und dem Controller. Hierbei mussten zahlreiche Variablen berücksichtigt werden, um den Anforderungen verschiedener Abteilungen gerecht zu werden.

2.1 Abstract (English)

The aim of this thesis was to develop multiple vehicle robots that are supposed to push each other out of the arena. The control of the robots is done by a user using a controller, which communicates with the robots via a wireless connection. To ensure that the robots do not leave the arena, each robot is equipped with a color sensor that detects when the robot exits the arena area. Additionally, the arena was intended to have additional features, such as a score display, which however were not addressed within the scope of this thesis.

Here, the electronics for both the robot and the controller are designed using KiCAD. Circuit diagrams and printed circuit boards (PCBs) are developed, which are subsequently manufactured by an external supplier (Aisler). The remote control of the robot is enabled by the NRF24L01 radio module. Both boards are equipped with AVR microcontrollers, allowing programming via the Arduino IDE. Programming the ATMEGA2560 for the robot requires the use of the SPI bus, as it is an SMD microcontroller.

The code of the controller is responsible for reading the input elements and transmitting the data to the robot. The robot, in turn, reads these values, calculates the required motor power for the left and right motors, and outputs these values as PWM signals to the motor shield. Additionally, the robot's circuitry is equipped with various sensors, the readings of which are transmitted to a Raspberry Pi. There, the data is graphically represented on a web server.

The thesis focuses on the selection of appropriate materials and the calibration of 3D printers, which are crucial for ensuring successful printing. Significant improvements to the printer were necessary to guarantee printing success. Another key emphasis is placed on modeling the robot's housing, wheels, propulsion systems, and the controller. This involves considering numerous variables to meet the requirements of different departments.

3 Einleitung – Projektbeschreibung

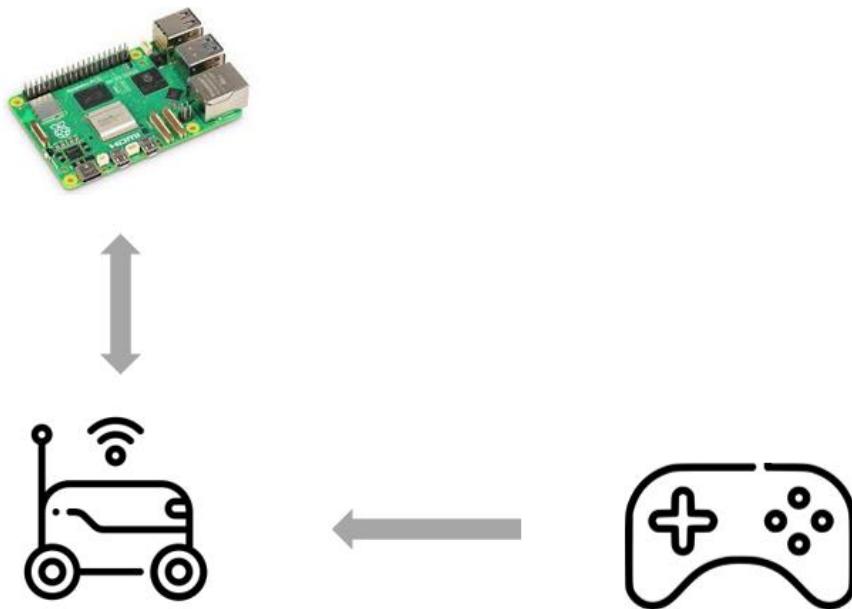


Abbildung 1: Aufbau des Projekts

Das Projekt SumoBots umfasst drei miteinander kommunizierende Geräte. Der Controller sendet Informationen an den Roboter, um dessen Richtung und Geschwindigkeit zu steuern. Der Roboter verfügt über verschiedene Sensoren, deren aufgenommene Werte an einen Raspberry Pi 3 übermittelt werden. Diese Daten werden auf einem Webserver grafisch dargestellt, der auch die verbundenen Geräte auflistet.

Aufgrund der umfassenden Aufgabengebiete des Controllers und des Roboters wird deren Systemarchitektur einzeln dargestellt und erläutert.

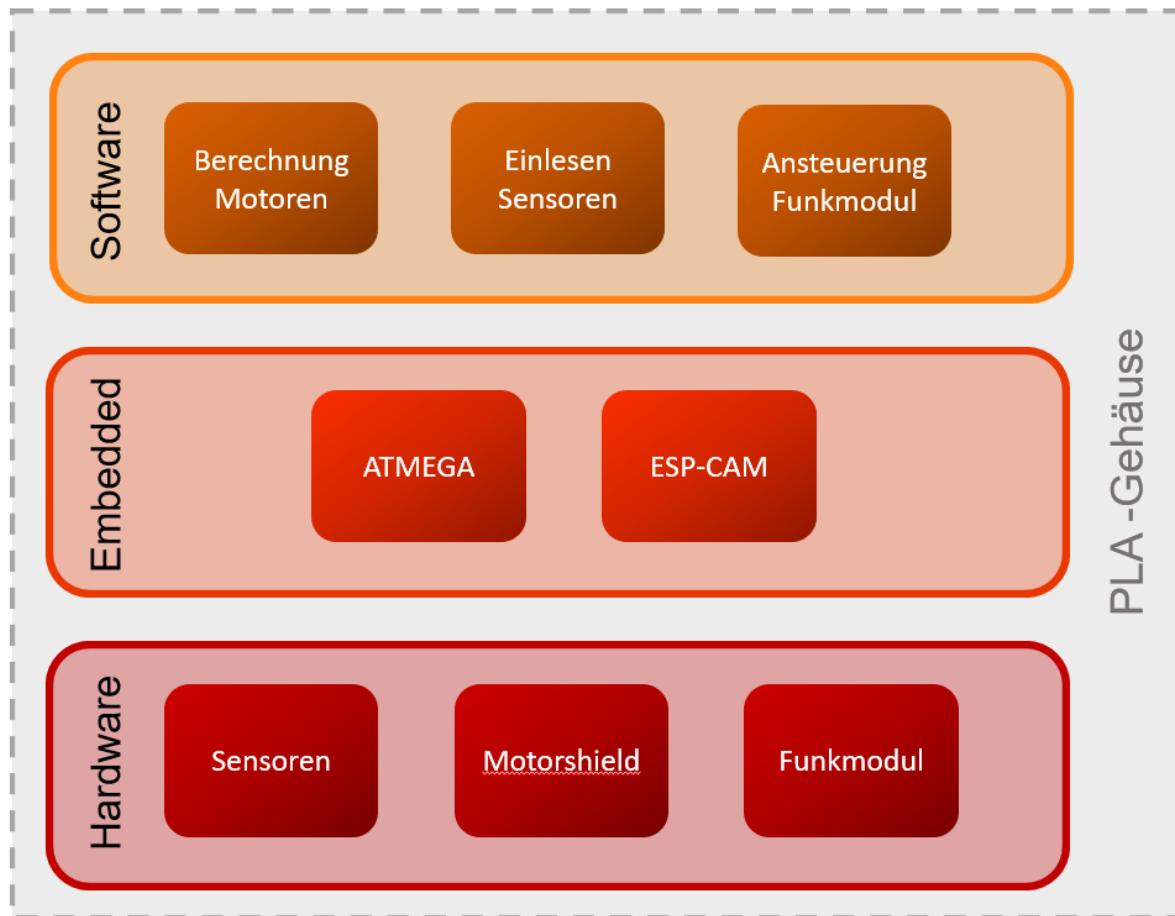


Abbildung 2 Systemarchitektur des Roboters

Im Hardwarebereich umfasst der Roboter drei Hauptkomponenten: Sensoren, die im Verlauf der Diplomarbeit detaillierter erläutert werden, das Motorshield und das Funkmodul. Im Embedded-Bereich wird der Atmega2560 als Mikrocontroller verwendet, der als Kern des Roboters fungiert und sämtliche Prozesse und Berechnungen ausführt. Als Erweiterung wird ein ESP32 mit einer Kamera hinzugefügt. Diese Embedded-Komponenten werden programmiert, um die Steuerung der Motoren durch Berechnungen durchzuführen. Die Sensorwerte werden im Mikrocontroller mithilfe von Bibliotheken ermittelt. Der Roboter kommuniziert mit dem Controller und dem Raspberry Pi, was durch ein Funkmodul ermöglicht wird.

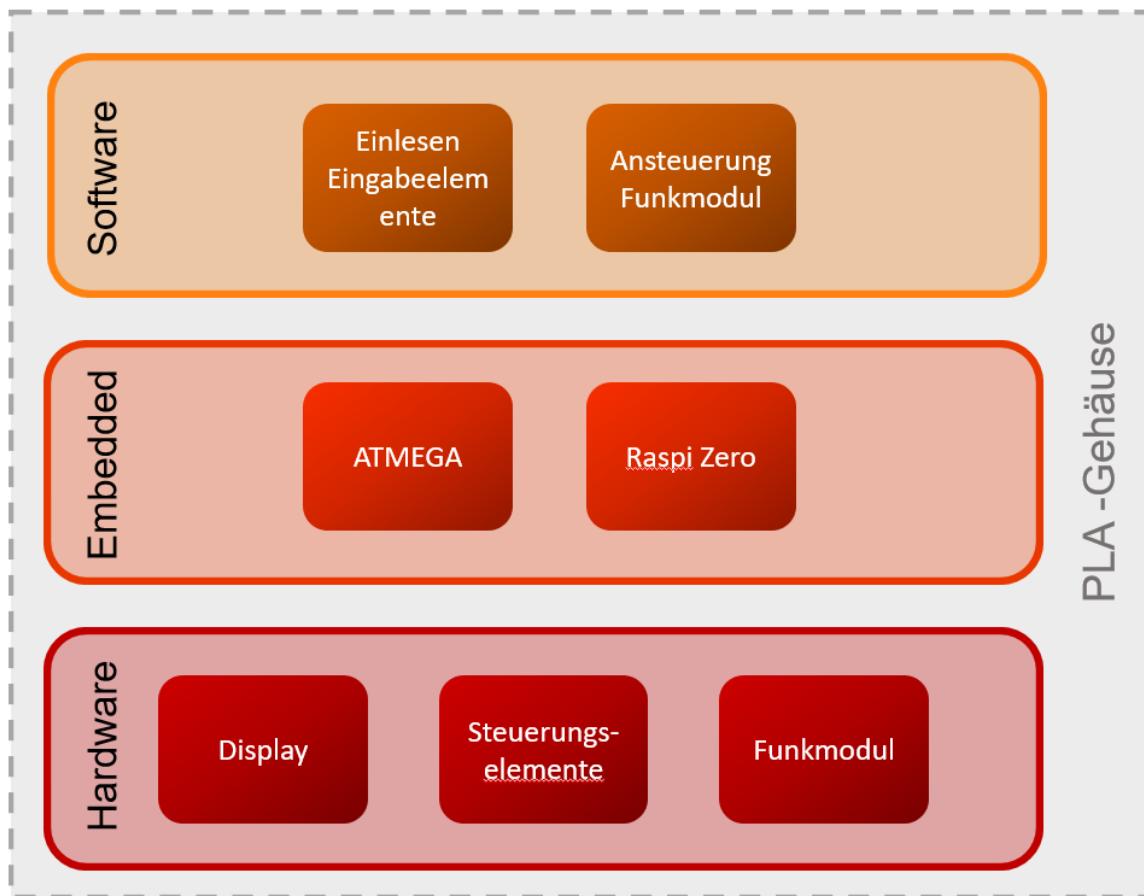


Abbildung 3 Systemarchitektur des Controllers

In der Hardware-Schicht werden Steuerungselemente wie Trigger und Joystick eingelesen, die es dem Nutzer ermöglichen, die Richtung und Geschwindigkeit des Roboters zu steuern. Zusätzlich werden Anzeigen in Form von LCD- und Touchscreen-Displays verbaut. Im Embedded-Bereich wird der Atmega328p als Mikrocontroller verwendet, und ein Raspberry Pi Zero wird hinzugefügt, um das Touchscreen-Display anzusteuern. Sämtliche einstellbare Werte von Trigger und Joystick werden eingelesen und anschließend ausschließlich an den Roboter über ein Funkmodul gesendet. Der Controller selbst kommuniziert nicht mit dem Server.

3.1 Vorgeschichte – Ausgangslage

3.1.1 Projektwoche 2023

Im Zuge der vergangenen Projektwoche wurde bereits ein Roboter und eine Fernsteuerung entwickelt und in Betrieb genommen. Beide Prototypen wurden von dem AVR-Mikrocontroller ATMEGA 328p angesteuert. Die Kommunikation zwischen den beiden, wurde durch das Funkmodul NRF24L01+ ermöglicht, welches die Daten vom Controller an den Roboter sendet. Mit diesen Daten steuert das implementierte Motorshield Cytron MDD3A die Motoren. Als Spannungsquelle wurde ein 12V Bleiakku genutzt.

3.1.2 CLIL 2023

Während des CLIL-Unterrichts 2023, wurde als Ziel die Weiterentwicklung der entwickelten SumoBots, sowie die Verbesserung von Konstruktion und Hardwarefehlern gesetzt, um die Milestones der Diplomarbeit genau definieren zu können. Bei der Konstruktion wurde auf das Arbeiten mit Fusion 360 gesetzt, um verbesserte und professionellere 3D-Modelle konstruieren zu können. In der Hardwareentwicklung wurde ein Fehler, welcher im Zusammenhang mit den unterschiedlichen Logikpegeln des Mikrocontrollers und Funkmoduls steht, durch einen Logic-Level-Converter verbessert. Des Weiteren wurden einige Recherchen, sowie eine Testschaltung entwickelt. In Bereich der Softwareentwicklung wurde der Fokus auf In-System-Programming gelegt, in welchen einige Testcodes geschrieben wurden.

4 Struktur des Projekts – Arbeitspakete

4.1 Objektstrukturplan (OSP) / Arbeitspakete

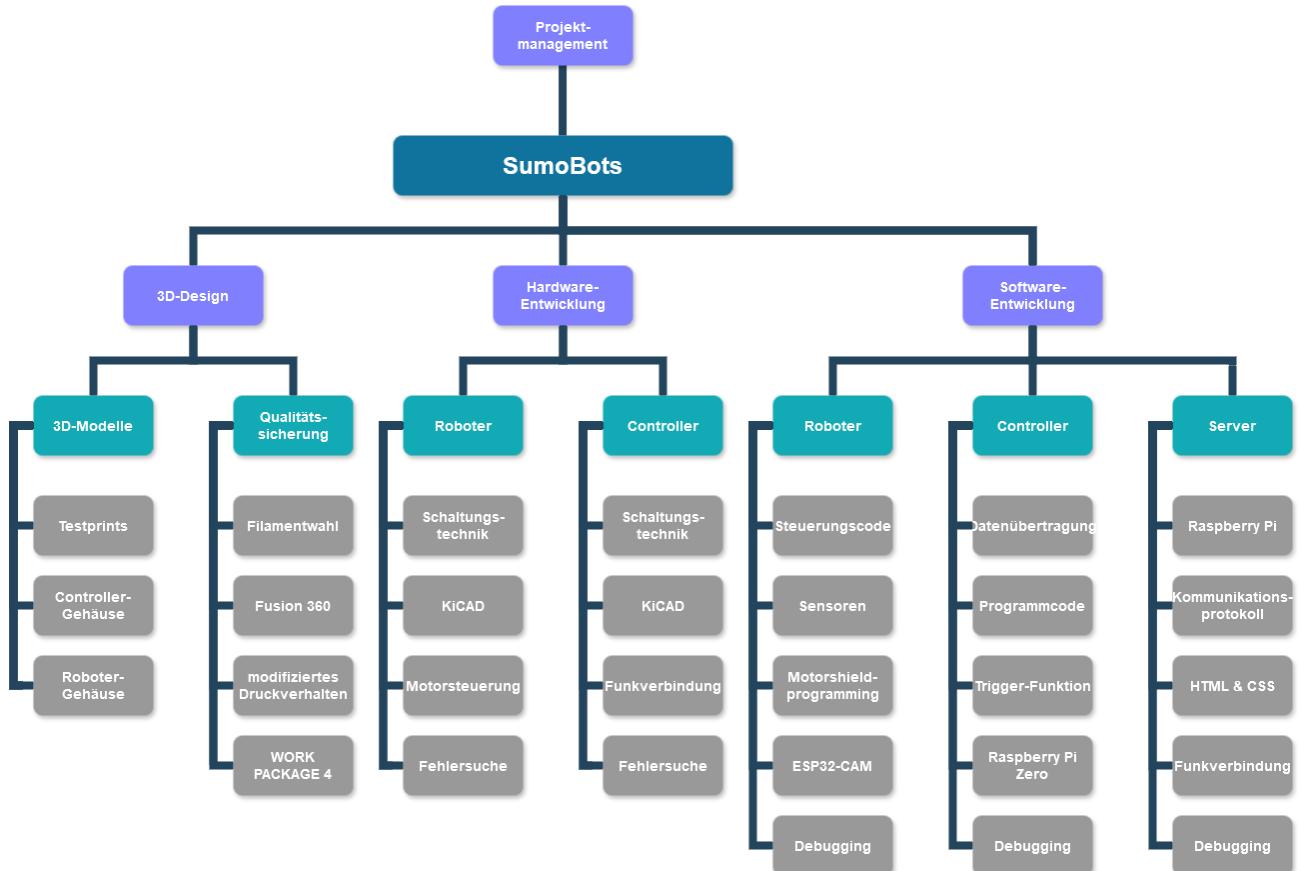


Abbildung 4: Objektstrukturplan (OSP)

Um einen besseren Einblick der Tätigkeiten, sowie deren Aufteilung in den Arbeitspaketen zu gewähren, wird die Diplomarbeit in einem Objektstrukturplan dargestellt. Hier wurden die dazu gehörigen Hilfsmittel, Aufgaben, sowie die benötigten Fähigkeiten für das jeweilige Arbeitspaket genau beschrieben.

4.2 Projektstrukturplan (PSP)

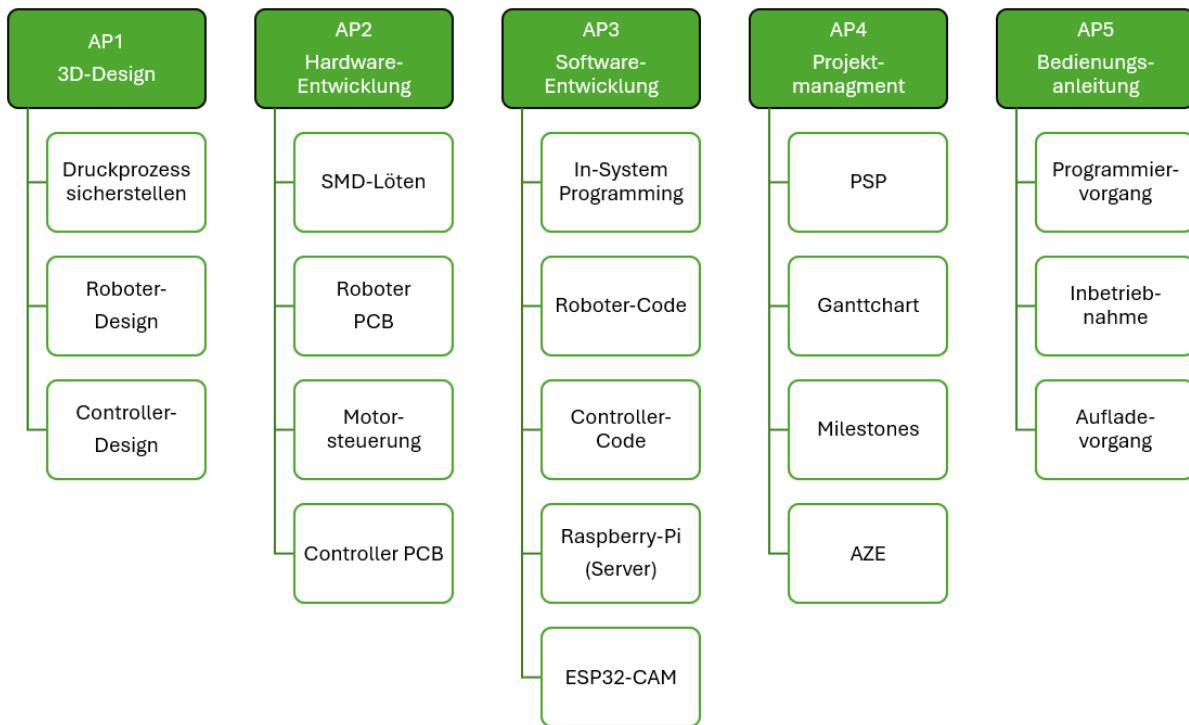


Abbildung 5: Projektstrukturplan (PSP) AP1-AP6

Die im oberen Diagramm beschriebenen Tätigkeiten wurden in Arbeitspakete und daraufhin auf die Diplomanden aufgeteilt.

4.3 Arbeitspaket 1 3D-Design (EIS)

Im Arbeitspaket „3D-Konstruktion“ wird neben den 3D-Modellen, ebenfalls Qualitätssicherung benötigt, da dieses Aufgabengebiet die Stabilität und Qualität der 3D-Prints sicherstellen.

4.4 Arbeitspaket 2 Hardware-Entwicklung (LUC)

Die Schaltungsentwicklung wird im Arbeitspaket „Hardware-Entwicklung“ bearbeitet, in welchen der Entwicklungsprozess des Roboters und des Controllers schrittweise abgebildet ist. Der Hauptfokus liegt hierbei in der Entwicklung des Schaltplans und des darauffolgenden Platinenlayouts, welches in KiCAD designet wird. Es werden hierbei doppelseitige Platten erstellt. Diese Platten werden zum Großteil mit SMD-Bauteilen bestückt, daher ist das SMD-Löten ein wichtiger Bestandteil und stellt damit auch ein Risikofaktor des Arbeitspaketes da.

4.5 Arbeitspaket 3 Software-Entwicklung (ZIC)

Das Arbeitspaket „Software-Entwicklung“ beinhaltet Softwaredesign, Debugging, Ansteuerung der SPI und I2C Buse zur Programmierung von µC und einlesen verschiedener Sensormodule und die Kommunikation mithilfe von Funkmodulen zwischen Controller und Roboter sowie Roboter und Server.

4.6 Arbeitspaket 4 Projektmanagement

Die Struktur, Koordination und Organisation der Diplomarbeit wird im Arbeitspaket „Projektmanagement“ bearbeitet und beinhaltet sämtliche Pläne, Zielsetzung, sowie der dazugehörigen erstellten Milestones-Pläne und Arbeitszeiterfassungen. Um die Arbeitszeiten, samt den Tätigkeiten der Diplomanden im Blick zu behalten, werden diese erfasst und in eine Arbeitszeiterfassung AZE dokumentiert. Des Weiteren werden die Arbeitspakete in einem Projektstrukturplan auf die Diplomanden aufgeteilt, die die Tätigkeiten der Diplomanden auf gewählte Bereiche einschränken. Dazu wird der Fortschritt des jeweiligen Arbeitspaketes in einem Ganttchart in Projekt Libre als Graph dargestellt.

4.7 Arbeitspaket 5 Bedienungsanleitung

Um die fertige Diplomarbeit nach Abgabe in Betrieb nehmen zu können, wurde eine Bedienungsanleitung geschrieben, in welcher die Inbetriebnahme, samt weiteren Konfigurationseinstellung, dokumentiert wird.

5 3D-Design und Konstruktion (EIS)

Dieses Kapitel behandelt und beschreibt den Aufbau, sowie die Konstruktion der Mechanik des Roboters und der Controller-Fassung. Für den Bau dieser Komponenten wird das CAD-Programm Autodesk Fusion 360 in Verbindung mit dem Slicer-Programm Cura 3D angewandt. Der gesamte Aufbau des Roboters, sowie des Controllers soll mit eigenerstellten STL-Files 3D-gedruckt werden. Dabei liegt ein beträchtlicher Fokus auf der Wahl des richtigen Materials, beim 3D-Drucken Filament genannt, da diese zwischen den unterschiedlichen Werkstoffen viele differente Eigenschaften aufweisen können.

5.1 3D-Drucker Begriffserklärungen

5.1.1 Vom CAD-File zum Drucken

Der erste Schritt im 3D-Design und Druckvorgang beinhaltet die Wahl des gewünschten CAD oder auch Computer Aided Design Programmes, welches zum Erstellen der benötigten Dateien benutzt wird. Zur Auswahl stehen hier eine große Auswahl an verschiedensten CAD-Programmen für den Heim- als auch Firmengebrauch. Durch persönliche Erfahrungen mit dem Programm Fusion 360 von Autodesk, wurde dies für die gesamte Konstruktion von Roboter als auch Controller benutzt. Das verwendete Dateiformat für die Modellierten Teile wurde auf. STL-Files festgelegt, da diese am einfachsten durch ein Slicer-Programm weiterzuverarbeiten sind.

Ein sogenannter Slicer wird benötigt um die. STL-Dateien auf eine Koordinatenorientierte Programmiersprache mit dem Namen G-Code zu übersetzen. 3D Drucker können diesen ge-sliced G-Code dann auslesen und nach Vorgaben der ursprünglichen. STL-Datei nachdrucken. Hierbei arbeitet der Drucker mit Seinen 3 Achsen X-, Y- und Z-Achse, um die vorgegebenen Koordinaten und befehle im G-Code abzufahren, wodurch das gewünschte Modell entsteht. Ein anderes Feature der Slicer ist es verschiedene Druckeigenschaften für den bestimmten Druckvorgang zu konfigurieren, wodurch zum Beispiel die Geschwindigkeit, Temperatur, Startposition sowie viele anderen Faktoren individuell für jeden Druck geändert werden können. Als Slicer für alle 3D-Modelle fiel die Wahl auf Ultimaker Cura 3D, wegen seiner vielfältigen Einstellungsmöglichkeiten und Optionen.

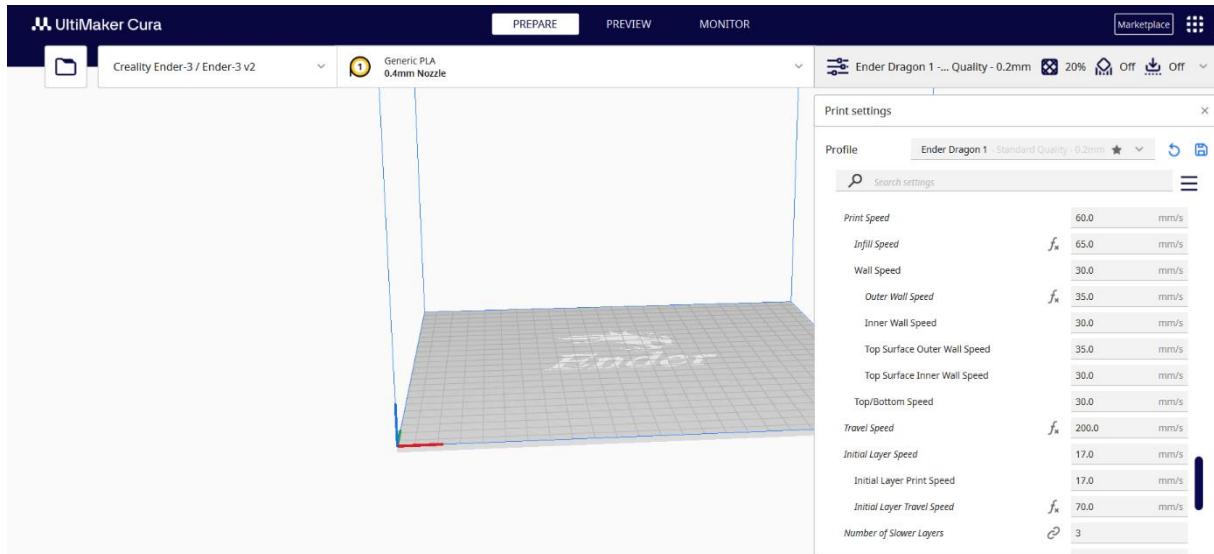


Abbildung 6 Cura 3D Interface

Wie in Abb. 6 zu sehen ist, verfügt der Slicer über eine 3D-Ansicht, in welcher der geplante Druck inspiziert und verschoben werden kann. Als Beispiel für die Einstellungsmöglichkeiten sind auf der rechten Seite der Abbildung die Geschwindigkeit Settings zu sehen welche den Travel Speed des Hotends sowie der restlichen Stepper-Motoren kontrolliert. Zu hohe Geschwindigkeiten können einen unsauberen Druck auslösen oder im schlimmsten Fall sogar zum kompletten Fehldruck, siehe Abb. 7, führen, welcher auch umgangssprachlich als „Spaghetti“ bekannt ist.



Abbildung 7 Kompletter Druckfehler

5.1.2 3D Drucker

Im Laufe dieses Kapitels werden verschiedene Begriffe und Beschreibungen für einzelne Komponenten eines typischen 3D Druckers verwendet. In diesem Abschnitt sollen die wichtigsten Begriffe aufgezeigt und erklärt werden. Alle benötigten Komponenten der SumoBots wurden mit einem privaten Creality Ender 3 V2 gedruckt welcher über mehrere Upgrades und Erweiterungen verfügt.

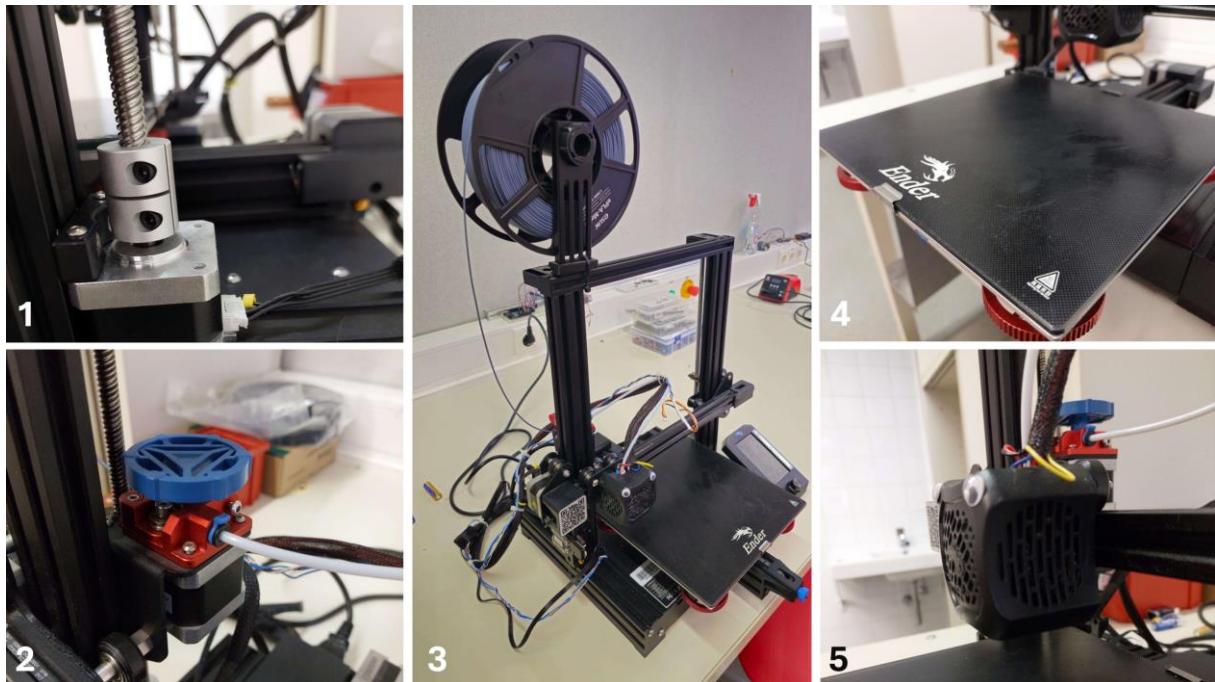


Abbildung 8: Drucker-Komponenten Überblick

Auf Bild 1 der Abbildung 8: Drucker-Komponenten Überblick ist ein Stepper-Motor zu sehen der für den hub der Z-Achse verantwortlich ist. Stepper-Motoren werden bei 3D Druckern und anderen Präzision Maschinen häufig dank ihres genauen und stabilen Betriebs gewählt, bei diesem Drucker allein sind 5 Stepper-Motoren verbaut, welche jede mögliche Achsen Bewegung und das Laden des Filaments übernehmen.

In Bild 2 der Abbildung ist der sogenannte Extruder sichtbar. Seine Aufgabe ist das Laden des Filaments und das Einschieben von mehr Material hin zum Hotend (siehe Bild 5) während des Druckvorgangs. Im Extruder verbaut sind zwei Zahnräder, wobei eines mit einem Stepper-Motor angetrieben wird und Filament durch die „Feeding-Tube“ ins Hotend schiebt. Der verbaute Extruder ist jedoch ein „Tuning“ Teil das extra für den Druck des Roboters installiert wurde um bessere Drucke zu garantieren.

Bild 4 zeigt die Buildplate oder Druckfläche, auf der das Filament vom Hotend aufgeschichtet wird. Dieser Drucker verfügt über eine, separat vom Hotend beheizte Buildplate welche eine bessere Haftung ermöglicht und Fehler Effekten wie Warping entgegenwirken kann. Es ist ratsam diese regelmäßig zu säubern, um Druckfehlern vorzubeugen und gute Haftung zu garantieren. Ein weiterer wichtiger Punkt ist das „Leveln“ der Buildplate bei dem man mit Hilfe von vier Einstellschrauben unter der Platte die Höhe der Buildplate verstellt, um überall den

gleichen Abstand zum Hotend zu haben. Dieser Leveling-prozess kann bis zu 40min dauern da man die Schrauben im Millimeterbereich drehen muss, um ein gutes Level zu erreichen.

Bild 5 veranschaulicht das Hotend des Druckers, welches das Hauptheizelement des 3D Druckers ist. In seinem Gehäuse befinden sich ebenfalls zwei Lüfter, um das Heizelement auf einer konstanten Temperatur zu halten. Ebenfalls sind die Lüfter für das Kühlen des Werkstücks verantwortlich damit das Flüssige Filament nach dem Austritt aus dem Hotend sofort härtet. Das originale Hotend konnte bis zu 190°C Heizen, jedoch wurde mittlerweile ein Upgrade Kit installiert, welches Temperaturen bis zu 260°C ermöglichen, sollten diese benötigt werden.

Diese Begriffe werden im Laufe des Kapitels mehrfach benutzt, um Prozesse und Vorgänge genauer erklären zu können

5.2 Materialeigenschaften

Die Wahl des richtigen Filaments kann massive Auswirkungen auf sowohl Druckqualität wie auch Druckstabilität haben. Dadurch eignen sich manche Werkstoffe besser für gewisse Anwendungen als andere und können anhand verschiedener Anforderungen gewählt werden. Bei den 3D-Drucken verwendete Filamente sind PLA, PLA+, ABS und PETG, welche für diverse Anwendungen gebraucht oder getestet wurden.

5.2.1 PLA, PLA+

PLA auch Polylactide genannt, sind aus natürlichen Rohstoffen wie Maisstärke oder Zuckerrohr bestehende Polyester und sind eines der populärsten Filamente der Welt des 3D-Druckens. Das Material bietet eine, zu anderen Werkstoffen verglichen, niedrigere Schmelztemperatur was für einen besseren Druckvorgang und Druckqualität bei Temperaturschwankungen im Hotend des Druckers oder der Umgebungsluft.

Das Drucken von Standard PLA Filament ist theoretisch schon mit einer Temperatur von 190°C möglich, wird jedoch aus gesammelten Erfahrungen am besten mit einer Temperatur von 210-213 °C gedruckt. Dabei entwickelt das Filament die beste Haftung an der sogenannten Buildplate des 3D-Druckers, wodurch die Qualität und Stabilität des Drucks um ein Vielfaches gesteigert wird. Negative Aspekte des Filaments sind jedoch auch vielfach vorhanden. PLA ist durch seine Zusammensetzung aus natürlichem Polyester in der Theorie biologisch abbaubar, jedoch lässt sich aus Erfahrungswerten sagen, dass die Abbauzeit der Konstruktion des Controllers, oder der Roboter, keine beachtliche Rolle spielt da diese selbst mit Einwirkung von Regen, Wind und Sonne bis zu 80 Jahre dauern kann. Allerdings ist das Filament für tragende Anwendungen nicht geeignet, da die Festigkeit des Materials mit der Härte von ABS-Plastik nicht mithalten kann.

Zur Lösung dieses Problems gibt es den Stoff PLA+, welcher auf dem originalen PLA basiert, jedoch aber mit einer Reihe von anderen Stoffen kombinierten wurde, um seine positiven Druckeigenschaften zu verbessern und die Negativen, wo möglich, zu entfernen. So bietet dieses Material eine bis zu 10-mal höhere Schlagfestigkeit ohne die negativen Aspekte eines Stoffes wie ABS-Filament mit sich zu bringen.

5.2.2 ABS

ABS-Filament oder auch Acrylnitril-Butadien-Styrol ist ebenfalls ein Polyester, der jedoch auf synthetischer Basis hergestellt wird. Dadurch besitzt er nicht den Nachteil biologisch abbaubar zu sein, wie PLA-Filament. Doch in der Praxis ist der bemerkenswerteste Unterschied die Schlagfestigkeit des Filaments. ABS besitzt einen viel höheren Härtegrad als das übliche PLA-Filament und kann daher für mehr tragende Rollen eingesetzt werden als PLA.

Jedoch bietet dieses Material einen gewaltigen Nachteil im Gegensatz zu anderen Filamenten in Form eines weitaus schwierigeren Druckverfahrens. Die benötigte Temperatur um ABS-Filament stabil und mit guter Qualität Drucken zu können ist mit bis zu 260 °C weitaus höher und schwieriger konstant zu halten als bei vergleichbaren Stoffen, wodurch sich die Anwendung bei unserer Diplomarbeit als praktisch unmöglich herausstellte. Durch diese hohe Drucktemperatur ist das Filament äußerst anfällig für äußere Temperaturschwankungen der Umgebungsluft oder auch Druckplatte, wodurch ein Fehler namens Warping auftreten kann, siehe Abb. 9

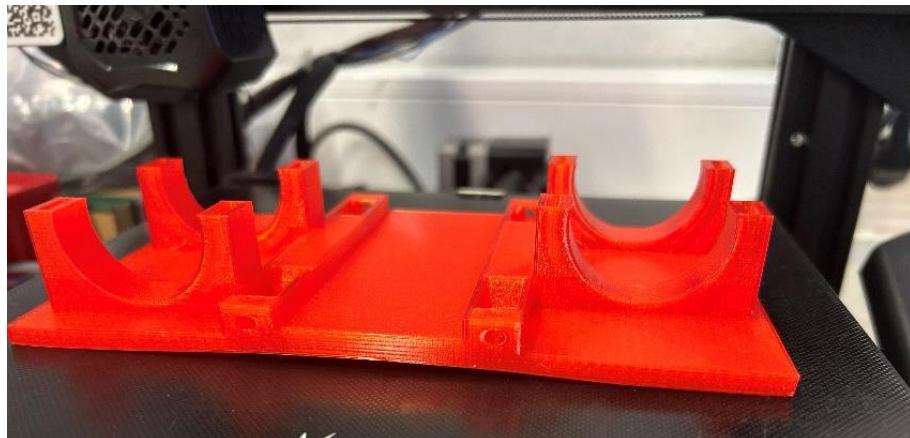


Abbildung 9 Warping

Hierbei „Warped“ oder verformt sich das Filament aufgrund von Temperaturschwankungen und vor allem große Flächen können deformiert und bis zur Unbrauchbarkeit verbogen werden. In der Theorie kann jeder Filament Stoff solche Ergebnisse hervorbringen, jedoch treten diese Fehler am häufigsten bei hohen Drucktemperaturen wie bei denen von ABS auf. Diese Instabilitäten kann durch konstante Temperaturhaltung ausgeglichen werden, jedoch erwies sich dies als unnötig für unsere Anwendungen da für ein solches Vorhaben eine Art „Druck-Behälter“ oder Kammer gebraucht wird.

5.3 Konstruktionsvorgänge

Ein wichtiger Bestandteil der 3D Konstruktion besteht aus der Wahl der richtigen Teilkomponenten und passenden Verbindungsart.

5.3.1 Antriebsarten

Die Wahl der gewünschten Antriebsart wurde durch mehrere Tests und Prototypen festgelegt und durchlief teils massiven Veränderungen. Als Anfangskonzept wurde ein symmetrisches Motorlayout gewählt, bei dem der Roboter zwar nur über zwei Motoren verfügt, diese jedoch mit einer Übersetzung die anderen beiden Räder mitdrehen, um somit einen Allrad Antrieb zu haben der auf einem Ähnlichen Prinzip eines Kettenfahrzeugs oder Panzers basiert. Durch die Symmetrie der beiden Roboterhälften wurde sowohl der Arbeitsaufwand als auch die Druck-Schwierigkeit enorm verringert.

5.3.1.1 Zahnriemen

In den frühen Entwicklungsstadien dominierten Zahnriemen als bevorzugte Lösung für den Antrieb. Sie ermöglichten eine effiziente Kraftübertragung und hielten das Design des Roboters kompakt, um die Nutzung standardisierter Zahnriemenlängen zu gewährleisten. Um dennoch eine gewisse Flexibilität in der Größenanpassung zu bieten, wurde ein innovativer Riemen-Spanner entwickelt, der es bis in die endgültigen Designversionen schaffte. Trotz seiner Vorteile offenbarte dieses Antriebssystem signifikante Schwächen, wie die begrenzte Lebensdauer der Zahnräder, die durch den ständigen Betrieb verschlissen und letztlich ersetzt werden mussten, und die inhärente Einschränkung durch die Länge der Zahnriemen, was das System in seiner Anpassungsfähigkeit limitierte.

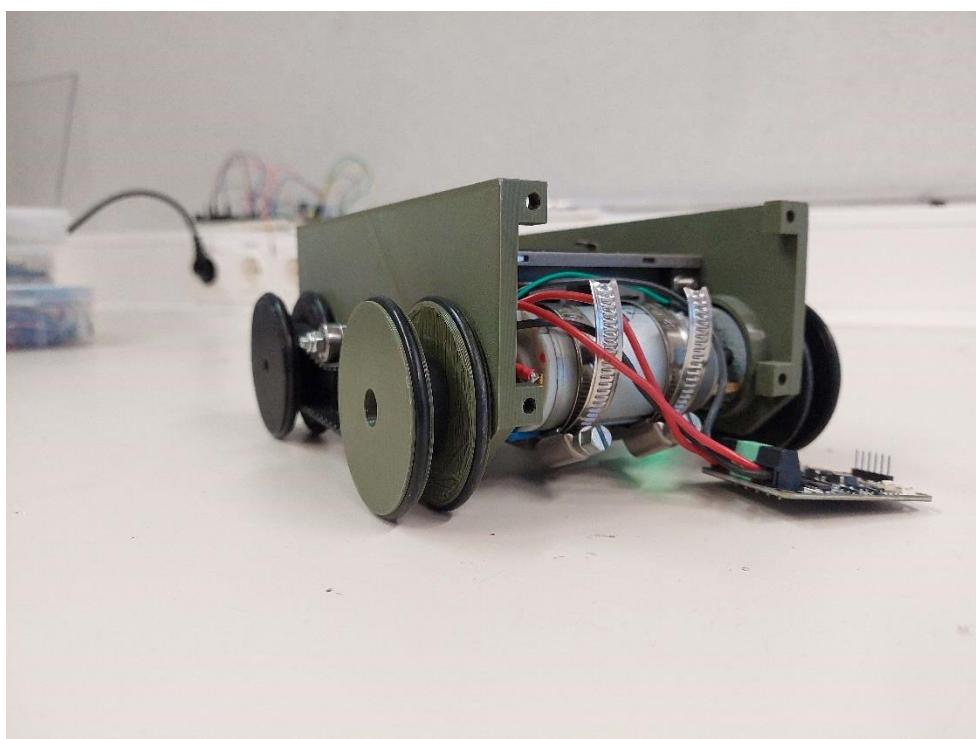


Abbildung 10: Zahnriemen-Prototyp

5.3.1.2 Zahnrad

Die Evolution führte zum Zahnradkonzept als nächstem Schritt in der Antriebstechnologie des Roboters, welches sich durch seine außergewöhnliche Anpassungsfähigkeit und Fehlerresistenz auszeichnete. Die Zahnräder wurden mit einem Laser-Cutter hergestellt, was nicht nur eine schnelle Produktion ermöglichte, sondern auch die Kosten und den Zeitaufwand im Vergleich zu anderen Methoden erheblich reduzierte. Die robuste Übersetzung der Zahnräder bewies eine beeindruckende Langlebigkeit und Zuverlässigkeit, selbst nach langen Betriebsstunden, was den Zahnradantrieb zu einer der überlegensten Antriebslösungen machte.

Durch kontinuierliche Verbesserungen und Anpassungen dieser Antriebssysteme wurde die Leistungsfähigkeit und Zuverlässigkeit des Roboters signifikant gesteigert. Jedes System, von Zahnriemen bis Zahnrad, trug durch seine einzigartigen Eigenschaften und die daraus resultierenden Erkenntnisse entscheidend zur Entwicklung effizienter und optimierter Antriebslösungen bei. Diese evolutionäre Entwicklung spiegelt die fortwährende Suche nach dem idealen Gleichgewicht zwischen Leistung, Effizienz und Anpassungsfähigkeit im Roboterdesign wider.

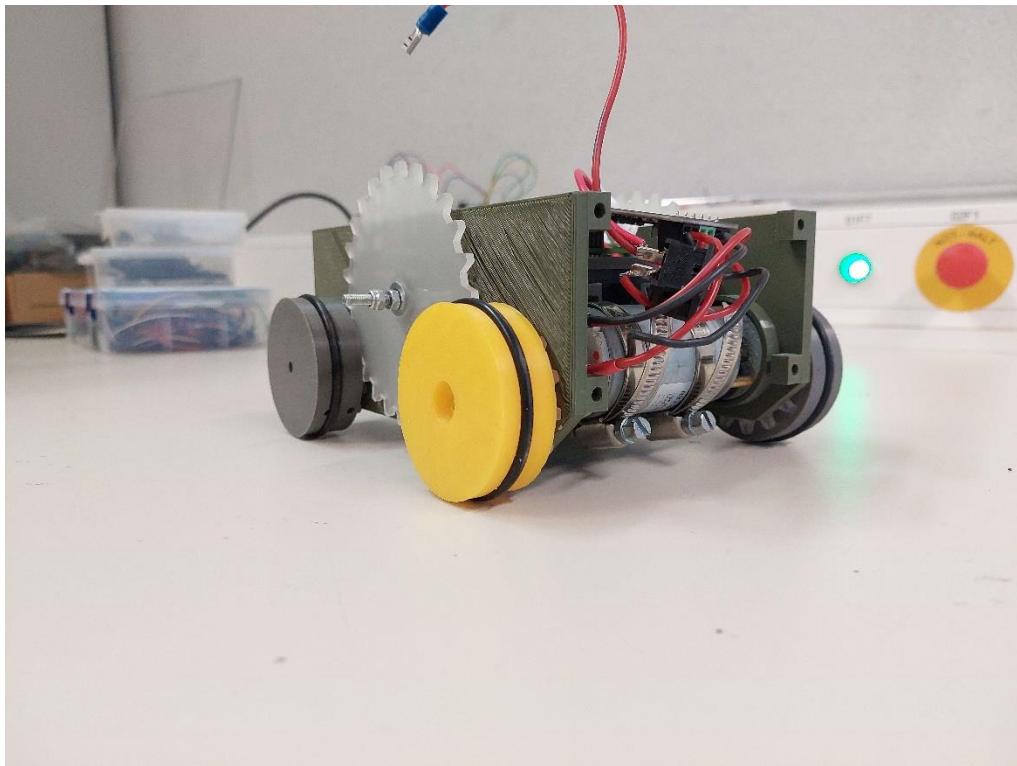


Abbildung 11: Zahnradantrieb-Prototyp

Bei der Überarbeitung des Zahnradantriebs wurden umfangreiche Anpassungen vorgenommen, um die Effizienz und die Kraftübertragung des Systems zu verbessern. Die Zahnradgeometrie, einschließlich Größe, Durchmesser, Zahanzahl sowie die Form und Zuspitzung der Zähne, wurde durch Berechnungen neu definiert. Ziel dieser Optimierungen war es, die Interaktion zwischen den Zahnräder zu verfeinern und somit eine effektivere und reibungslose Kraftübertragung zu gewährleisten.

Durch die Neuberechnung der Zahnräder konnten Verbesserungen in Bezug auf die Übertragungseffizienz und die Langlebigkeit des Antriebssystems erzielt werden. Die spezifische Anpassung der Zahnform und -zuspitzung spielte dabei eine entscheidende Rolle. Eine optimierte Zahngeometrie fördert einen sanfteren Eingriff der Zahnräder, reduziert den Verschleiß und minimiert das Risiko von Zahnausfällen unter Last. Zudem ermöglichte die präzise Abstimmung der Zahanzahl eine verbesserte Lastverteilung und eine gesteigerte Kraftübertragungskapazität, wodurch die Gesamtleistung des Antriebssystems erhöht wurde.

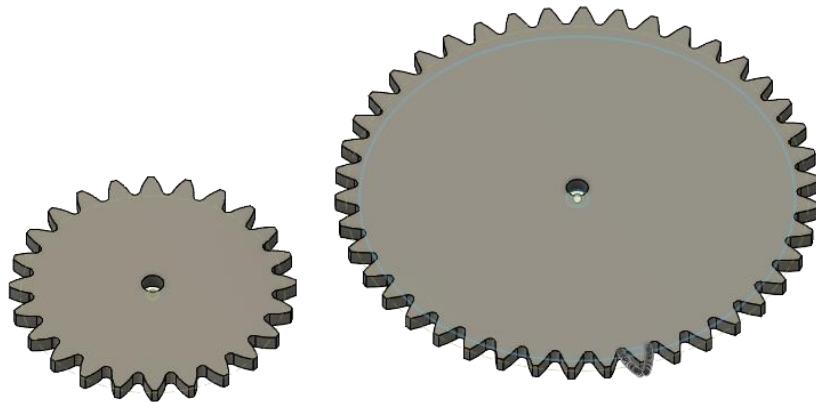


Abbildung 12: neue Zahnräder

5.3.1.3 Andere Prototypen

Im Laufe der Entwicklung des Roboters wurden neben den Zahnrad- und Zahnriemenantrieben auch alternative Antriebssysteme wie Kettenantriebe und Stangensysteme getestet. Diese Prototypen hatten zum Ziel, die Palette möglicher Lösungen für die Antriebstechnik zu erweitern und zu diversifizieren. Jedoch stellten sich diese Alternativen als nicht praktikabel heraus.

Das Stangensystem, inspiriert von der Konstruktion klassischer Zugsysteme, bei denen einzelne Räder durch Stangen an beweglichen Punkten verbunden sind, erwies sich als besonders problematisch. Die Idee hinter diesem Ansatz war es, eine direkte und robuste Übertragung der Motorleistung auf die Räder zu ermöglichen. Doch die Realität zeigte schnell die Schwächen dieses Systems auf. Die erforderlichen Toleranzen für eine präzise Drehbewegung der Räder waren niedrig, sodass schon geringfügige Fertigungsungenauigkeiten zu einem Stillstand des Systems führen konnten.

Ein besonders aufschlussreicher Vorfall war ein Test, bei dem ein solches Stangensystem zu einem Lock-up des Motors führte. Die resultierenden hohen Ströme, die durch den blockierten Motor flossen, verursachten ein Bürstenfeuer – ein ernsthaftes Sicherheitsrisiko, das nicht nur die Zuverlässigkeit, sondern auch die Sicherheit des gesamten Roboters in Frage stellte. Angesichts dieser Schwierigkeiten und der offensichtlichen Risiken wurde die Entwicklung und Integration des Stangensystems in den Roboter aufgegeben.



Abbildung 13: Stangenantrieb Beispiel (https://de.wikipedia.org/wiki/Stangenantrieb_%28Eisenbahn%29)

5.3.1.4 Antriebs- und Passivräder

Im Zuge der Entwicklung des Roboters wurde eine klare Unterteilung der Räder in zwei Kategorien vorgenommen, die sich nach ihrer Funktion im Antriebssystem des Roboters richten: passive Räder und Antriebsräder. Passive Räder, die nicht direkt mit dem Motor verbunden sind, spielen eine unterstützende Rolle und arbeiten zusammen mit den Antriebsrädern, um einen effizienten Allradantrieb zu ermöglichen. Die Antriebsräder hingegen sind fest mit dem Motor verbunden, was durch den Einsatz von rechteckigen Flachmuttern und Madenschrauben erreicht wird, die eine robuste Montage an der Motorachse gewährleisten.

Bei der Konstruktion der Räder wurde die Technik der Embedded Components angewendet, um Kugellager direkt in die Räder einzubetten. Diese Methode trägt nicht nur zur Effizienz der Kraftübertragung bei, sondern erhöht auch die allgemeine Robustheit und Langlebigkeit der Räder. Zusätzlich verfügen die Räder über Aussparungen für Gummidichtungen an der Außenseite, die die Haftung mit dem Boden signifikant verbessern. Diese Gummielemente, flexibel in der Anzahl der einsetzbaren Rillen, ermöglichen eine individuelle Anpassung der Bodenhaftung. Durch das Einfügen verschiedener Anzahlen und Anordnungen von Gummidichtungen in die Räder kann die Traktion des Roboters auf unterschiedlichen Untergründen optimiert werden.



Abbildung 14: Räder

5.3.2 Rahmen des Roboters

Während der Durchführung der Diplomarbeit wurden entscheidende Anpassungen am ursprünglichen Design des Roboter-Rahmens vorgenommen, um dessen Leistungsfähigkeit zu optimieren. Diese Änderungen basierten auf tiefgreifenden Einblicken, die durch das Testen von Prototypen und die Analyse verschiedener Druckverfahren gewonnen wurden. Ziel war es, den Rahmen so zu gestalten, dass er eine maximale Stabilität bietet und gleichzeitig die Druckqualität verbessert. Die Anfangsmodelle des Roboters zeichneten sich durch eine Konstruktion aus zwei identischen Hälften aus. Dieser Ansatz hatte den klaren Vorteil, dass keine Unterscheidung zwischen der rechten und linken Seite des Roboters getroffen werden musste, was das Gesamtdesign erheblich vereinfachte. Die ausgewählten Rahmenhälften waren nicht nur leicht mittels 3D-Druck herzustellen, sondern boten auch eine beachtliche Stabilität.

Die Einfachheit dieses Designs hatte jedoch ihren Preis. So führten Änderungen an bestimmten Dimensionen zu weitreichenden Auswirkungen auf das gesamte Design des Roboters. Zusätzlich erschweren erforderliche Modifikationen wie zusätzliche Löcher, Aushebungen oder andere einseitige Anpassungen die Aufrechterhaltung der symmetrischen Konstruktion. Trotz dieser Herausforderungen bewies der Ansatz der symmetrisch gespiegelten Konstruktion seine Vorteile in Bezug auf Stabilität, Einfachheit und Robustheit, was ihn zu einem prägenden Element im Konstruktionsprozess der Roboter machte.

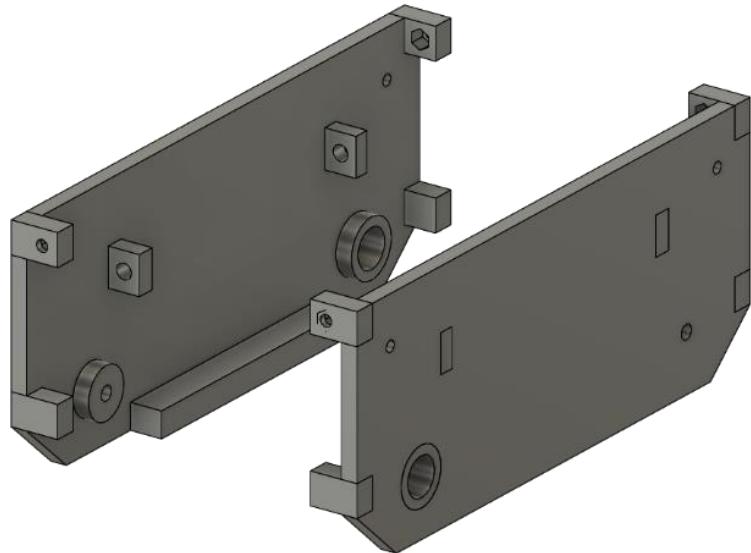


Abbildung 15: Prototyp Seitenteile

Ein weiterer Kritikpunkt war die Verbindung der beiden Hälften mithilfe der Motorhalterungen. Diese Halterungen sollten die Motoren sicher am Gehäuse befestigen. Allerdings offenbarten sich bei diesem Design mehrere Schwierigkeiten, die zu einem Übergang zu einem moderneren Konzept führten. Ursprünglich verwendete man Kabelbinder, um die Motoren zu fixieren, welche jedoch nicht die erforderliche Stabilität und Flexibilität boten. Bei jeglicher Änderung des Rahmens mussten die Kabelbinder entfernt und ersetzt werden, was langfristig keine zufriedenstellende Lösung darstellte. Daher erfolgte der Wechsel zu wiederverwendbaren, metallenen Rohrklemmen. Diese boten aufgrund ihrer Robustheit die gewünschte Stabilität und konnten zudem jederzeit mit Hilfe eines Schraubendrehers gelöst und wieder angebracht werden. In späteren Entwürfen etablierten sich Rohrklemmen als ideales Befestigungsmittel.

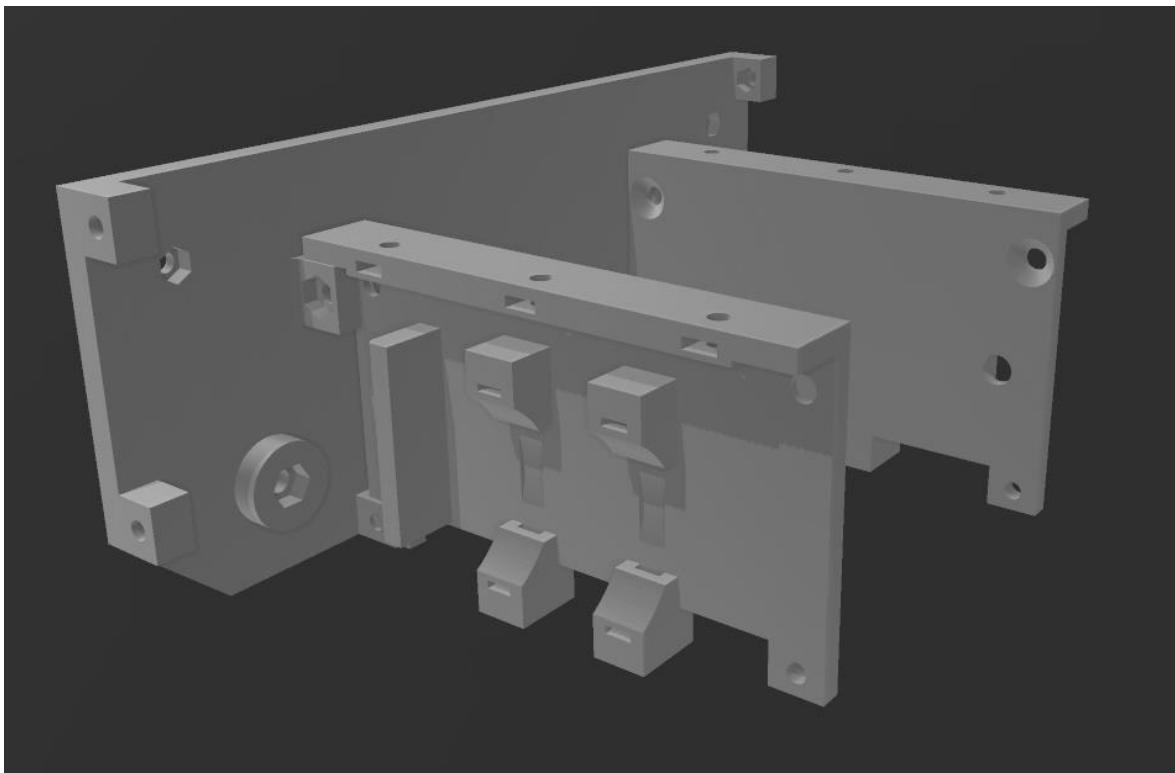


Abbildung 16: Motorhalter

Der ehemals verfolgte Ansatz symmetrischer Seitenteile erwies sich mit der Erweiterung der Funktionalitäten des Roboters zunehmend als unrealistisch, was zu einem Design führte, das nicht länger auf einem einheitlichen Aufbauprinzip beruht. Die Motorhalterung wird nun von einer zentralen Bodenplatte übernommen, die alle erforderlichen Schraubenlöcher und Erweiterungsoptionen für zusätzliche Hardware, Sensoren und weitere Aussparungen aufweist. Diese zentrale Platte hat den Druckprozess wesentlich effizienter gestaltet und die Wartung des Roboters vereinfacht, da im Rahmen selbst mehr Raum für diverse Halterungen, für die Platine, den Akku und andere elektronische Komponenten, geschaffen wurde. Ein zusätzlicher Vorteil ist die erhöhte Stabilität des Roboters sowie der verbesserte Abstand zum Boden. In diesem Design wurden weiterhin die robusten Rohrklemmen aus früheren Prototypen verwendet, und es wurden mehrere bewährte Konstruktionsansätze integriert.

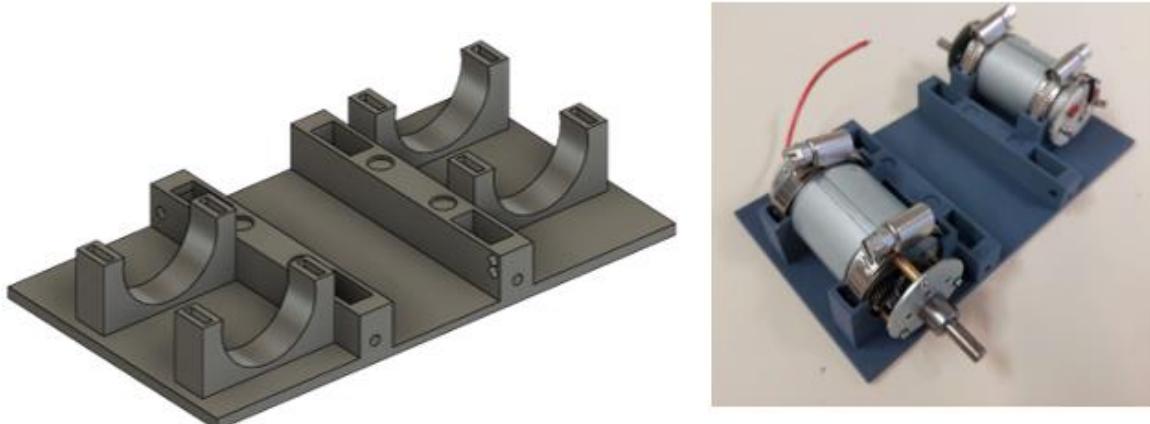


Abbildung 17: Bodenplattendesign

Bei Betrachtung des 3D-Modells ist zu erkennen, dass die zentrale Bodenplatte so gestaltet ist, dass sie als Kernstück für die Montage und Integration der verschiedenen Komponenten dient. Ihre Struktur weist eine Serie von gekrümmten Aussparungen auf, die genau auf die Form der zu befestigenden Motoren abgestimmt sind, sowie eine Reihe von vorgefertigten Schraubenlöchern für eine präzise Befestigung. Diese Konfiguration ermöglicht nicht nur eine schnelle und sichere Montage der Motoren, sondern auch eine einfache Skalierbarkeit des Systems durch Hinzufügen weiterer Module. Durch die strategische Platzierung der Schraubenlöcher und Aussparungen ist auch sichergestellt, dass der Roboter leichter anpassbar und für zukünftige Upgrades vorbereitet ist.

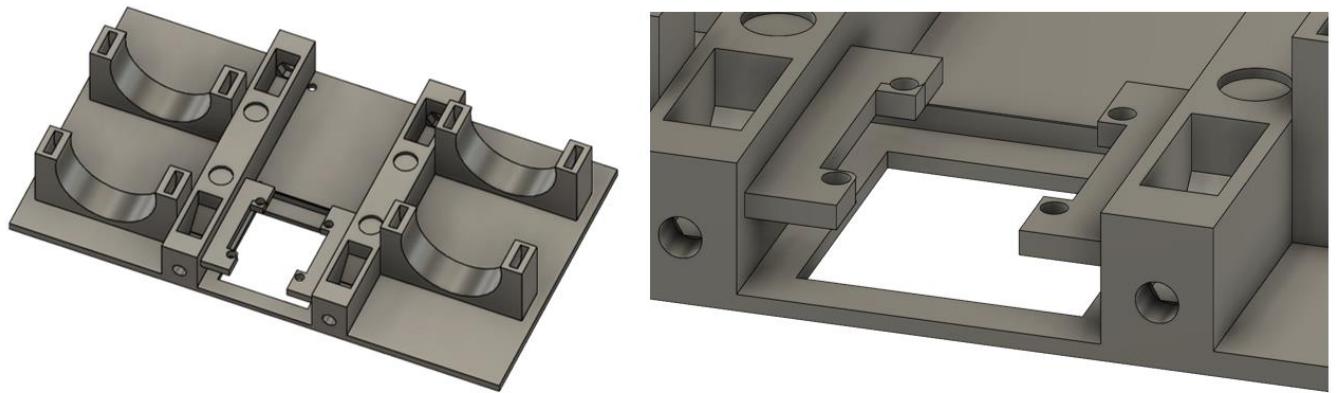


Abbildung 18: Sensorhaltung

Eine signifikante Weiterentwicklung im Design des Roboters war die Integration von Aussparungen für 4-mm-Neodym-Magnete, welche eine innovative Lösung zur Befestigung der Abdeckung über dem darunter montierten Farbsensor und dem fest installierten Motor Shield darstellten. Auf dieser magnetisch befestigten Abdeckung wurde zudem eine Abgrenzung implementiert, die den Akku sicher an seinem Platz hielt. Die Entscheidung, Magnete zu verwenden, erwies sich als durchdachter Schritt, um den Zugang zu den internen Komponenten des Roboters zu vereinfachen und gleichzeitig die Notwendigkeit traditioneller Befestigungselemente wie Schrauben und Muttern zu eliminieren.

Diese Designinnovation bot nicht nur den Vorteil eines schnellen und werkzeuglosen Zugriffs auf wichtige elektronische Bauteile für Wartungs- oder Upgrade-Zwecke, sondern trug auch zur Gesamtästhetik und Sauberkeit des Roboterdesigns bei. Indem man die Magnete als Befestigungslösung wählte, wurde eine flexible und dennoch robuste Methode etabliert, die es ermöglichte, die Abdeckung sicher zu befestigen und gleichzeitig einen einfachen Mechanismus für den Zugang zu den unterliegenden Modulen zu bieten.

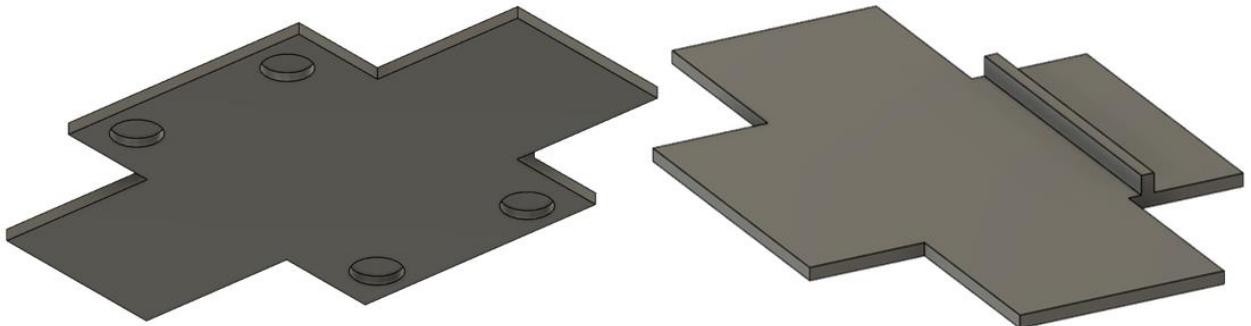


Abbildung 19: Abdeckung

Die Neugestaltung der Wände des Roboters erfolgte in enger Abstimmung mit der Bodenplatte, um eine nahtlose Integration und ein kohärentes Gesamtdesign zu gewährleisten. Diese Konstruktionsweise wurde speziell gewählt, um Flexibilität für spätere Anpassungen zu ermöglichen, sei es für den Einbau zusätzlicher Sensoren, Schalter, LEDs oder anderer funktionaler Komponenten, die zur Erweiterung der Fähigkeiten des Roboters beitragen könnten. Durch vorgeformte Optionen in den Wänden wurde zudem die Möglichkeit geschaffen, ein äußeres Gehäuse am Roboter zu montieren, was den Schutz der internen Komponenten verbessert und dem Roboter ein fertiges, professionelles Aussehen verleiht.

Die Entscheidung gegen eine symmetrische Bauweise führte zur Konzeption separater Teile für die rechte und linke Wand des Roboters, was eine individuelle Fertigung und Montage dieser Komponenten notwendig macht. Diese differenzierte Herangehensweise ermöglicht eine präzisere Anpassung an spezifische Anforderungen und Funktionen auf beiden Seiten des Roboters, bietet aber gleichzeitig die Herausforderung, dass jedes Teil einzeln gedruckt werden muss. Diese Konstruktionsentscheidung spiegelt das Bestreben wider, einen hochgradig funktionalen und anpassungsfähigen Roboter zu entwickeln, der in der Lage ist, eine Vielzahl von Aufgaben in unterschiedlichen Umgebungen effektiv zu erfüllen.

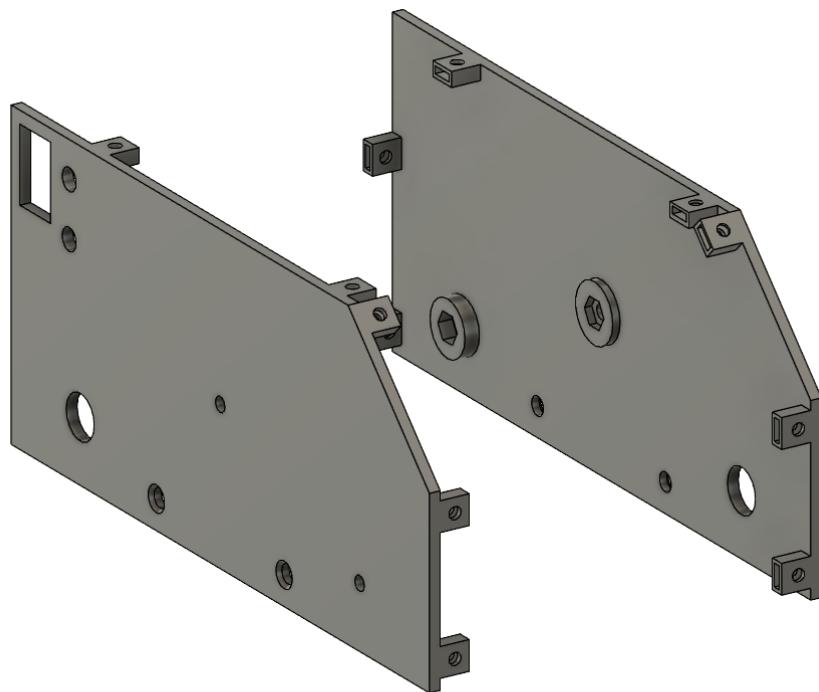


Abbildung 20: Seitenteil Rechts und Links

Das äußere Gehäuse des Roboters wurde mit besonderer Sorgfalt entworfen, um die Integration und Montage eines geplanten LCD-Bildschirms sowie einer Raspberry Pi Kamera zu ermöglichen. Dieses Design ermöglicht es, die fortgeschrittenen Funktionen und die Interaktivität des Roboters erheblich zu erweitern, indem Benutzern visuelle Feedbackmöglichkeiten und Bildverarbeitungsfunktionen geboten werden. Das Gehäuse wird präzise mit den vorbereiteten Löchern an den Wänden des Roboters verschraubt, wodurch ein semi-geschlossenes System entsteht, das die internen Komponenten schützt, während es gleichzeitig Zugang für Wartung und Upgrades ermöglicht.

Besondere Aufmerksamkeit wurde auch der Funktionalität und der Benutzerinteraktion gewidmet, indem speziell Platz für die Montage einer Antenne und einer Status-RGB-LED vorgesehen wurde. Diese Konstruktionsentscheidung trägt dazu bei, dass der Betriebszustand des Roboters auch dann noch klar erkennbar ist, wenn das Gehäuse vollständig geschlossen ist. Die Antenne gewährleistet eine zuverlässige drahtlose Kommunikation, was besonders für Fernsteuerungsaufgaben oder Datenübertragung wichtig ist, während die RGB-LED dem Benutzer wichtige Informationen über den Zustand des Roboters in Form von visuellen Signalen vermittelt. So kann zum Beispiel der Ladestatus der Batterie, die Verbindungsreichweite oder etwaige Fehlermeldungen effektiv kommuniziert werden.

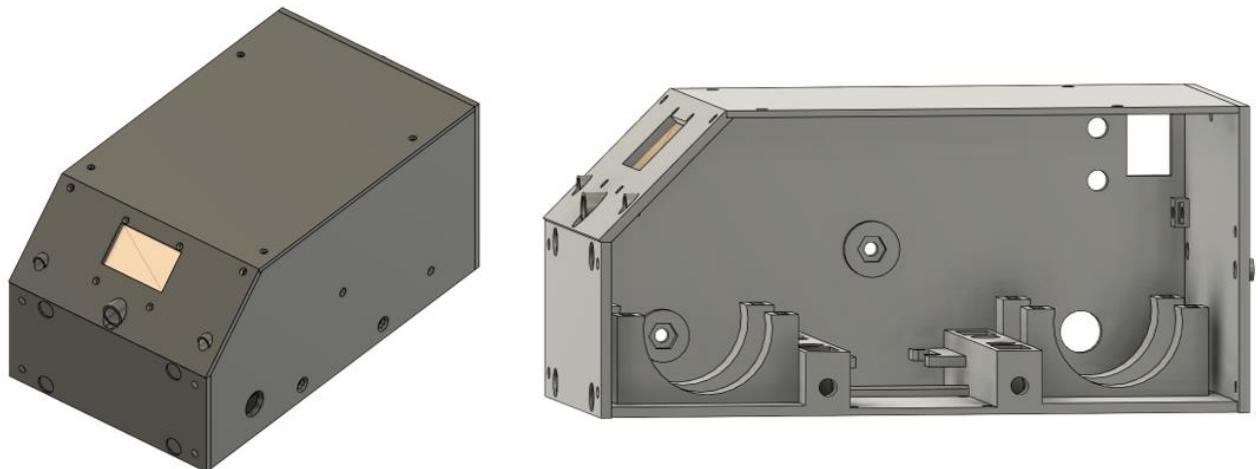


Abbildung 21: Gehäuse-Design

Weitere Verfeinerungen wurden an der Gehäusestruktur des Roboters vorgenommen, insbesondere durch die Integration von Aussparungen für 4-mm-Neodym-Magnete an der Front des Roboters. Diese innovative Ergänzung eröffnet eine Vielzahl neuer Möglichkeiten für die Anpassung und Erweiterung des Roboters durch magnetisch befestigte Extras wie zusätzliche Sensoren, Rammböcke oder andere Aufsätze.

Durch diese magnetischen Befestigungspunkte können Benutzer den Roboter schnell und einfach umkonfigurieren, je nach den spezifischen Anforderungen der Aufgabe oder des Einsatzumfeldes. Zum Beispiel könnten für Erkundungsmissionen in unbekanntem Terrain Sensoren angebracht werden, die das Gelände scannen und Hindernisse erkennen. Für Wettbewerbe oder Demonstrationszwecke könnten Rammböcke oder ähnliche mechanische Aufsätze befestigt werden, um die Interaktivität und das Engagement des Publikums zu steigern.

Das virtuelle Bild des Roboters zeigt das durchdachte Design und die komplexen Konstruktionsdetails, die bei der Entwicklung berücksichtigt wurden. Die Ansicht gibt Aufschluss über die Anordnung der Komponenten und wie sie im Rahmen des Robotergehäuses zusammenpassen. Mit dieser Visualisierung lassen sich potenzielle Herausforderungen im Druck- oder Konstruktionsprozess erkennen, wie etwa die Platzierung der Magneten, die Einbettung von Kugellagern oder die Integration der Elektronik.

Solche virtuellen Modelle sind äußerst nützlich, um bereits vor dem physischen Prototyping mögliche Schwachstellen oder Konstruktionsfehler zu identifizieren. Eventuelle Komplikationen beim 3D-Druck – beispielsweise überhängende Teile, die Stützmaterial erfordern, oder die Zugänglichkeit für das Einsetzen von Embedded Components – können so im Vorfeld erkannt und angepasst werden.

Das Modell zeigt auch, wie die externen Anschlüsse und Bedienelemente wie LCD-Display, Kamera, Antennenplatz und LED-Anzeigen zugänglich gemacht wurden, während gleichzeitig ein geschlossenes System zur Sicherung der internen Komponenten gewährleistet ist. Dieser ganzheitliche Ansatz in der virtuellen Darstellung ist entscheidend für die Fehlersuche und Optimierung des Designs, bevor wertvolle Ressourcen für den Bau physischer Modelle eingesetzt werden.

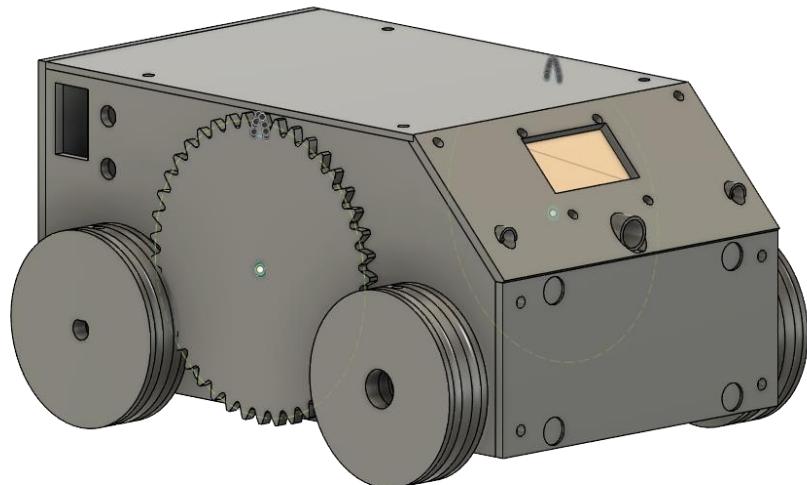


Abbildung 22: Volles Abbild des Roboters

5.3.2.1 Gehäuse des Controllers

Die Konstruktion des Controllers war ein komplexes Unterfangen, das viele entscheidende Designentscheidungen erforderte. Ein kritischer Aspekt dabei war die Entwicklung des Trigger-Mechanismus für die Geschwindigkeitskontrolle. Ein präziser und kontinuierlicher Gasanstieg ist für die feinfühlige Steuerung des Roboters essentiell, und in typischen Controllern wird dies durch ein Potenziometer erreicht, welches die Position des Triggers in ein entsprechendes Signal umwandelt. Dies ermöglicht eine variable Geschwindigkeitsanpassung weit über die simple binäre Auswahl zwischen Stillstand und Vollgas hinaus, die man mit einem Schalter oder Knopf hätte.

In den ersten Controller-Prototypen wurde ein Drucksensor verwendet, um die Kraft, mit der der Trigger gedrückt wird, in einen entsprechenden Widerstandswert zu übersetzen. Dieser Ansatz sollte die gleiche stufenlose Kontrolle ermöglichen, die ein Potenziometer bietet, stellte sich jedoch als instabil und unzuverlässig heraus. Schwankungen im Sensorausgang oder Schwierigkeiten bei der genauen Messung der Druckstärke können dazu führen, dass die Geschwindigkeit nicht präzise gesteuert werden kann.

Daher wurde die Entscheidung getroffen, auf robustere und zuverlässigere Alternativen umzusteigen. Die Verwendung eines Potenziometers, das oft in Gamecontrollern für diese Funktion verwendet wird, bietet eine bewährte Lösung. Es ermöglicht eine präzise und kontinuierliche Eingabe, was für die Steuerung des Roboters entscheidend ist. Das Potenziometer wandelt die mechanische Bewegung des Triggers in ein variables elektrisches Signal um, das dann vom Controller-Programm interpretiert und in entsprechende Geschwindigkeitsbefehle an den Roboter übertragen werden kann.

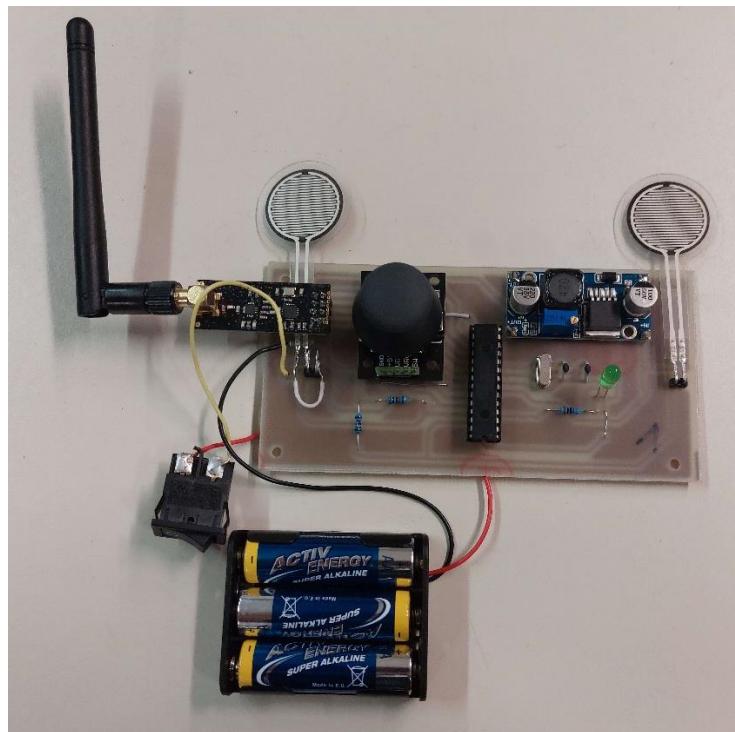


Abbildung 23: Prototyp des Controllers mit Drucksensoren

Für den Aufbau des Controllergehäuses wurde ein minimalistisches und funktionales Design gewählt, das die notwendigen Komponenten in einer einfach zu öffnenden Einheit unterbringt. Dieser Ansatz dient nicht nur der Benutzerfreundlichkeit, indem er den Zugang für Wartung, Reparatur oder Upgrade der internen Teile erleichtert, sondern unterstützt auch die Effizienz des Herstellungsprozesses.

Das Gehäuse wurde so gestaltet, dass es sich intuitiv öffnen lässt, ohne dass eine Vielzahl von Werkzeugen erforderlich ist, was Zeit spart und die Zugänglichkeit für Benutzer mit unterschiedlichem technischem Hintergrund erhöht. Die Konstruktion des Gehäuses fokussiert auf eine klare und übersichtliche Anordnung der internen Komponenten.

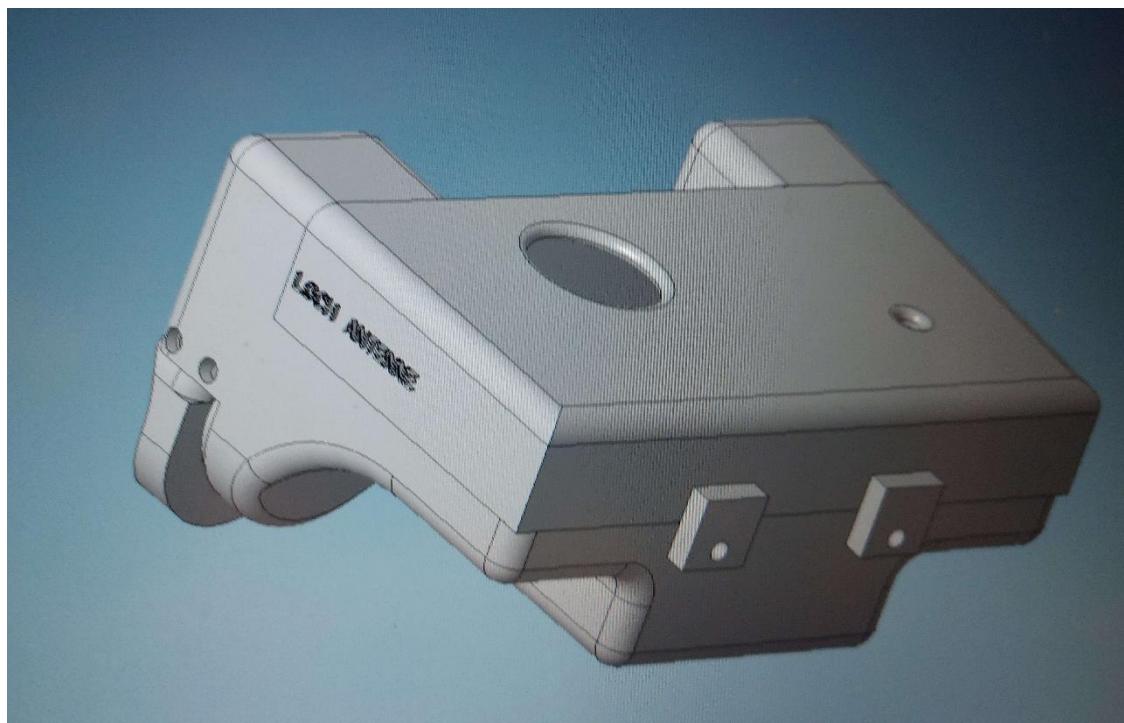


Abbildung 24: Erstes Gehäuse Design

Die Überarbeitung der Steuerung für den Roboter beinhaltete eine eingehende Untersuchung und das Auseinandernehmen des Trigger-Systems eines ferngesteuerten Autos, um dessen Mechanismus als Inspirationsquelle für einen ähnlichen Aufbau zu nutzen. Das Ziel war es, ein präzises und fein abgestimmtes Trigger-System zu entwickeln, das eine stufenlose Kontrolle über die Geschwindigkeit des Roboters ermöglicht. Allerdings stellte sich heraus, dass der Mechanismus des ferngesteuerten Autos für die geplante Anwendung zu komplex war. Insbesondere der Aufbau des Triggers passte nicht zu den Anforderungen des Controllers, da er die Benutzerfreundlichkeit und die intuitive Bedienbarkeit beeinträchtigte.

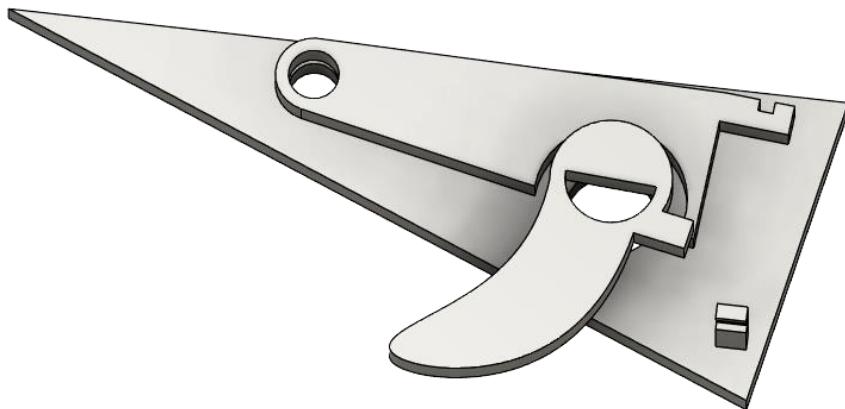


Abbildung 25: Mechanischer Trigger

Letztendlich wurde die Entscheidung getroffen, den Trigger-Mechanismus eines Akkuschraubers zu adaptieren. Dieser Ansatz bot eine vereinfachte Lösung, die sowohl die nötige Feinfühligkeit als auch die erforderliche Stabilität für den Controller lieferte. Zudem wurde ein Tiefpassfilter in das System integriert, um die Gaskurve (Geschwindigkeit) zu glätten. Diese elektronische Schaltung sorgt dafür, dass das Signal, welches den Motor steuert, frei von plötzlichen Sprüngen oder Rauschen ist, was eine gleichmäßige Beschleunigung ohne abrupte Geschwindigkeitsänderungen ermöglicht.

5.3.3 Verschraubungen und Normen

Um die Konstruktion des Roboters und des Controllers zu vereinfachen und eine einheitliche Gestaltung zu gewährleisten, wurde ein besonderes Augenmerk auf die Standardisierung der verwendeten Komponenten wie Schrauben, Muttern, Beilagscheiben, Kugellager und Kabel gelegt. Indem für die gesamte Konstruktion möglichst wenige verschiedene Hardwarekomponenten verwendet wurden, konnte nicht nur die Material- und Arbeitseffizienz gesteigert, sondern auch die Komplexität des Zusammenbaus reduziert werden. Die konsequente Anwendung einheitlicher Größen und Normen erleichterte die Montage, Wartung und eventuelle Reparaturen des Systems erheblich.

Die vorrangig verwendeten Schraubengrößen im M3- und M4-Format, jeweils ergänzt durch passende Muttern und selbstsichernde Muttern, bildeten das Fundament der Hardwareauswahl. Selbstsichernde Muttern kamen insbesondere bei beweglichen Teilen des Roboters zum Einsatz, um eine dauerhafte und zuverlässige Verbindung auch unter den Belastungen langer Betriebsstunden zu gewährleisten. Diese Wahl trug maßgeblich zur Betriebssicherheit und Langlebigkeit des Roboters bei.

Für die Konstruktion wurden vier verschiedene Schraubenlängen verwendet, wobei ein identisches Schraubkopfdesign für eine weitere Vereinheitlichung sorgte. Diese Konsistenz in der Auswahl ermöglichte eine effiziente Lagerhaltung und erleichterte den Montageprozess, da die Verwechslungsgefahr von Komponenten minimiert wurde. Bei den meisten Verschraubungen wurden normale Muttern eingesetzt, oft ohne den Einsatz von Beilagscheiben, was durch präzise Einlassungen im Gehäuse des Roboters ermöglicht wurde. Diese Einlassungen gewährleisteten eine sichere Lage der Muttern und trugen zu einer sauberer, ästhetisch ansprechenden Konstruktion bei.

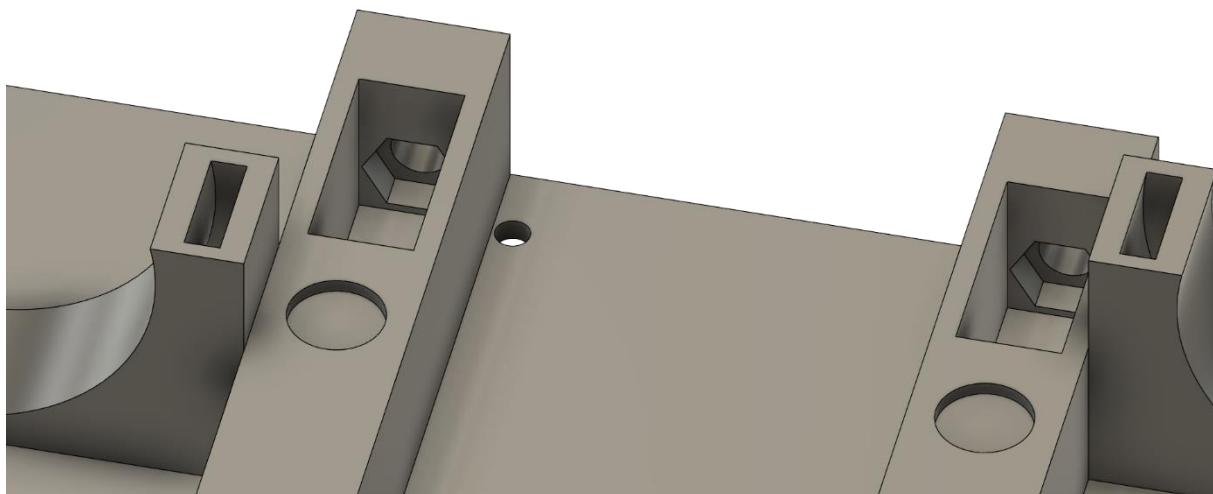


Abbildung 26: Muttern Löcher

5.3.4 Embedded Components (EC)

Embedded Components (EC) im 3D-Druck eröffnen neue Horizonte in der Herstellung komplexer Bauteile durch das direkte Einbetten von Objekten wie Muttern, Schrauben und Kugellagern in die 3D-gedruckten Teile. Diese innovative Technik steigert nicht nur die Funktionalität und strukturelle Integrität der Bauteile, sondern reduziert auch den Montageaufwand und die Notwendigkeit zusätzlicher Befestigungselemente erheblich. Die Implementierung von EC stellt jedoch technische Herausforderungen dar, insbesondere bei der exakten Positionierung der einzubettenden Teile, um sicherzustellen, dass sie fest im fertigen Bauteil verankert sind, ohne dessen Integrität oder ästhetische Qualität zu beeinträchtigen.



Abbildung 27: Eingebettetes Kugellager

Ein Lösungsansatz besteht darin, den Druckprozess an spezifischen Punkten zu pausieren, um die Komponenten manuell einzulegen und dann den Druck fortzusetzen, was eine präzise Steuerung des 3D-Druckers und möglicherweise die Anpassung des G-Codes erfordert. Die Anwendungsbereiche für EC sind vielfältig und reichen von der Luft- und Raumfahrt über den Automobilbau bis hin zur Medizintechnik, wo durch diese Technik Bauteile hergestellt werden können, die leichter, stärker und funktionaler sind als traditionell gefertigte Teile. Für eine erfolgreiche Implementierung sind Designüberlegungen bereits in der Frühphase wichtig, einschließlich der genauen Festlegung der Positionen und Abmessungen der einzubettenden Komponenten unter Berücksichtigung der thermischen Eigenschaften der verwendeten Materialien. Zudem ist die Anpassung des G-Codes für die Integration der Pausen und die Steuerung der Temperaturänderungen notwendig, wofür Open-Source-Software und spezialisierte Slicing-Programme oft bereits integrierte Funktionen oder Plugins bieten. Nach dem Druck ist eine sorgfältige Qualitätskontrolle erforderlich, um die Festigkeit der eingebetteten Komponenten und die gewünschte Funktionalität zu gewährleisten.

Embedded Components im 3D-Druck bieten ein enormes Potenzial für die Fertigungsindustrie, indem sie die Herstellung von Bauteilen mit höherer Funktionalität, verbesserten mechanischen Eigenschaften und reduzierten Fertigungskosten ermöglichen. Die erfolgreiche Anwendung dieser Technik setzt jedoch ein tiefes Verständnis der 3D-Druckprozesse, der Materialwissenschaften und der präzisen Steuerung der Druckparameter voraus. Mit der Weiterentwicklung der Technologie und zunehmender Erfahrung wird diese Methode voraussichtlich eine immer wichtigere Rolle in der modernen Fertigung spielen.

6 Hardwareentwicklung (LUC)

Im folgenden Kapitel werden die Funktionen, sowie die jeweiligen Entwicklungsschritte der Hardware des Roboters und Controllers erklärt. In Zuge dessen wird das Konzept der Steuerung, sowie der Kommunikation mit der dazugehörigen elektronischen Schaltung aufgegriffen.

6.1 Entwicklungsablauf

Da in diesen Teil des Arbeitspakets mehrere Fehler auftreten können, wurde ein Flussdiagramm erstellt, das die Abläufe der Entwicklung, sowie weitere optionale Schritte darstellt. Das Vorplanen der Tätigkeiten ist hier besonders wichtig, aufgrund der aufkommenden Kosten, welcher durch die PCB-Bestellungen von externen Firmen entsteht.

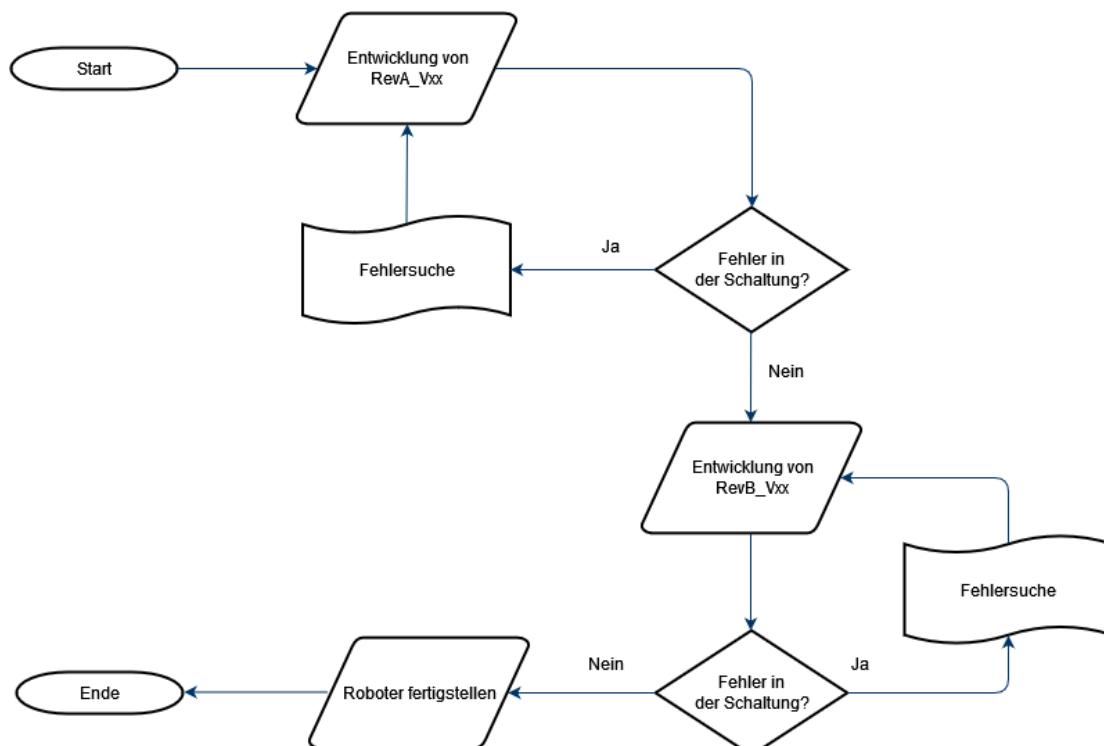


Abbildung 28: Entwicklungsablauf der Roboter Hardware

Dabei wurde die Hardware des Roboters in Revision A (RevA) und Revision (RevB) geteilt. In Revision A wird eine Platine mit den elektronischen Grundkomponenten entworfen, die benötigt werden, um den Roboter zum Fahren zu bringen. Bei Revision A wird der Fokus auf dem Ermitteln und Korrigieren von vermeintlich auftretenden Fehlern. Falls in der ersten Version des PCBs gravierende Fehler auftreten, die das in Betrieb nehmen der Schaltung nicht ermöglichen, wird eine weitere Version von Revision A entwickelt. Wenn dieser optionale Fall nicht eintritt, wird die Revision B Schaltung entwickelt, bei der die funktionsfähige Schaltung von Revision A mit Sensoren und weiteren Bauelementen aufgewertet wird.

6.2 Hardware-Design Roboter Revision A

Wie bereits beschrieben, beschränken sich die Hauptaufgaben dieses Schaltungsentwurfs auf die Kommunikation mit einem Controller, die Steuerung der Motoren durch diesen sowie die In-System-Programmierung.

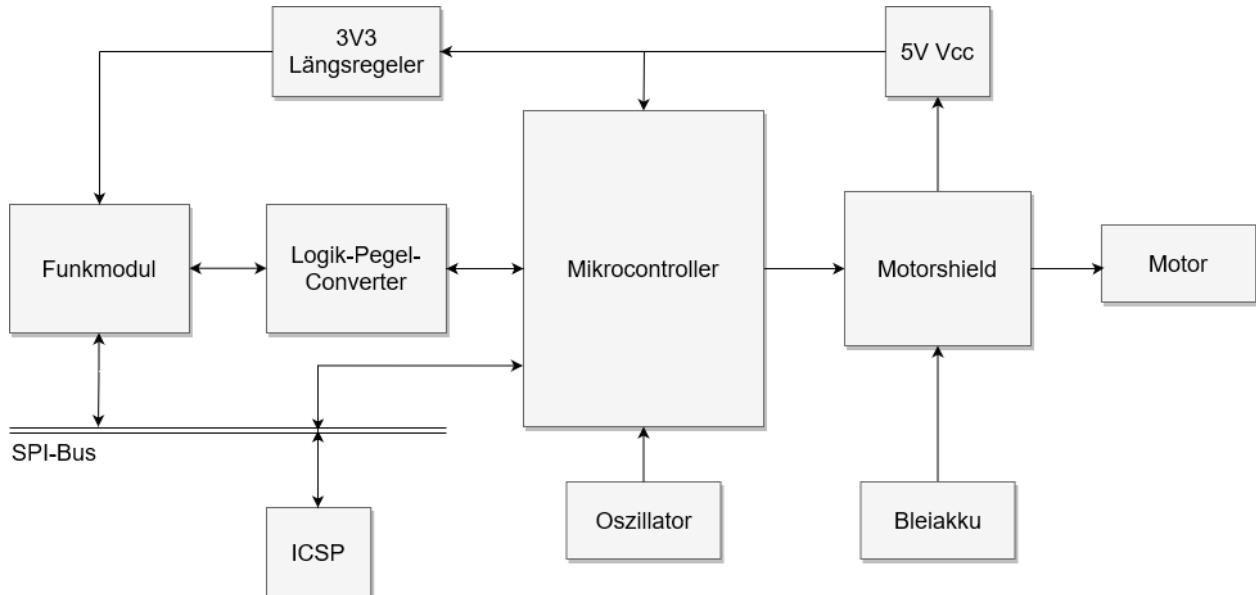


Abbildung 29: Blockschaltbild von Roboter Rev A V1.00

Als Mikrocontroller wird der ATMEGA2560 eingebaut, der als essenzieller Teil der Schaltung gilt. Damit dieser funktioniert, muss eine 5V Gleichspannungsquelle und ein digitales Taktsignal vorhanden sein. Diese Versorgungsspannung wird durch das Motorshield ermöglicht, da auf diesem ein Spannungsregler verbaut ist, der die 12V vom Bleiakku in 5V umwandelt. Die 12V vom Bleiakku werden benötigt, um die Motoren zu versorgen. Das Taktsignal wird durch einen externen Quarzoszillatator mit 16MHz generiert, der die Geschwindigkeit der Programmabläufe beeinflusst. Des Weiteren wird eine 3.3V Spannungsquelle benötigt, um das Funkmodul NRF24L01+ in Betrieb zu nehmen. Das Funkmodul arbeitet hierbei mit unterschiedlichen Logikpegeln als der Mikrocontroller. Um Fehler bei der Übertragung von Daten zu vermeiden, wird ein Logik-Pegel-Wandler (Logic Converter) dazwischengeschaltet, der den Pegeln von 3.3V auf 5V umwandelt. Da es sich hierbei um ein selbstentwickeltes PCB handelt, ist keine serielle Schnittstelle vorhanden. Um den Mikrocontroller programmieren zu können, wird über den SPI-Bus programmiert. Die SPI-Pins (MOSI, MISO, SCK) sind auf den ICSP-Header vorhanden.

6.2.1 Roboter Revision A V1.00 PCB-Design

Das im Kapitel 7.3 illustrierte Blockschaltbild wurde in KiCAD als Schaltplan, sowie als Platinenlayout realisiert. Aufgrund der dünnen Bahnen von 0,25mm können beim Ätzen Fehler auftreten. Daher wird für die Roboterplatten auf den schulexternen Hersteller Aisler zurückgegriffen.

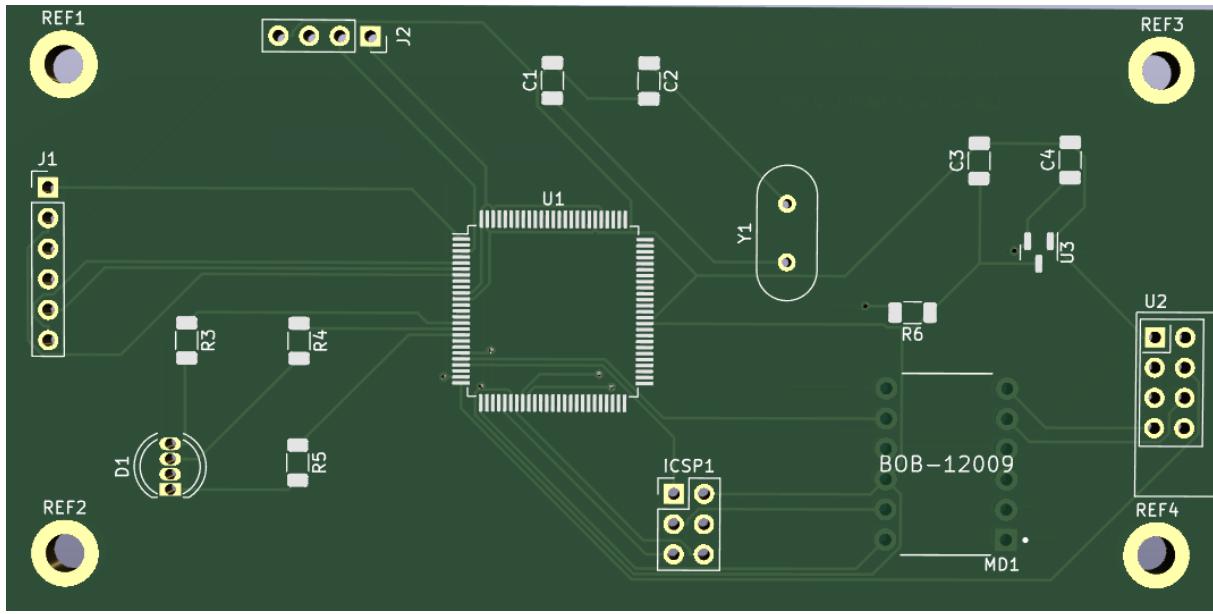


Abbildung 30: Roboter Rev A V1.00 PCB

Da hier das Löten mit SMD-Komponenten im Vordergrund stand, wurde der Platz zwischen den Bauteilen vergrößert, um leichter Fehler, die durch das Löten mit Heißluft zustande kommen, auszubessern. Im Fokus lag hierbei der Mikrocontroller, der durch die Anzahl von 100 Pins, welche jeweils 0,25mm breit sind, fehleranfällig auf Kurzschlüsse ist. Der Quarz Y1, der als Taktgeber für den Mikrocontroller dient, sollte möglichst nah platziert werden, um kurze Signalwege zu gewährleisten. Durch die Minimierung von Leitungslängen werden Laufzeitverzögerungen reduziert, was die präzise Synchronisation des Mikrocontrollers verbessert. Gleichzeitig verringert dies die Anfälligkeit für elektromagnetische Störungen und trägt zur Gesamtstabilität und Leistungsfähigkeit des elektronischen Systems bei. Da das Motorshield über keine eigene KiCAD-Bibliothek verfügt, wurde stattdessen ein 6x1-Pinheader J1 implementiert, auf welchen dieser draufgelötet wird. Ein weiterer Grund für die Positionierung lag an seiner Größe und den Verbindungen, bei denen breitere Drähte mit den Motoren und dem Bleiakkumulator verbunden werden müssen. Der 4x1 Pinheader J2 besteht aus zwei digitalen Pins, Vcc und Masse. Diese waren dazu vorgesehen, weitere Komponenten anzuschließen und auf Fehler zu überprüfen. Um das In-System-Programming zu ermöglichen wurde dieser ICSP-Header als 2x3 Pinheader ICSP1 realisiert. Dabei wurde die Reihenfolge der Pins vom Arduino UNO übernommen, um Fehler zu vermeiden. Das Funkmodul U2 wurde aufgrund seiner herausstehenden Antenne, am Rand der Platine positioniert, sodass ein Großteil des Gehäuses sich außerhalb des PCB befindet. Da für die Kommunikationskomponente eine Bibliothek existiert, gibt KiCAD bei dem Design Rule Check (DRC) eine Warnung aus, dass die Lötstopfmaske geschnitten wird. Aus diesem Grund wurde, wie in Abbildung 30: Roboter Rev A V1.00 PCB dargestellt, nur ein Teil des Umrisses (Front

Silkscreen) auf der Platine realisiert. Für den Logik-Pegel-Wandler wurde eine Bibliothek genutzt, die ein anderes Modell, aber dasselbe Footprint mit derselben Reihenfolge von Pins, beinhaltet. Wie in Abbildung 30: Roboter Rev A V1.00 PCB zu erkennen ist, sind die Pin-Verbindungen nicht vergoldet. Dies hat zur Folge, dass es nicht möglich ist, den Logik-Pegel-Wandler an die Platine zu löten. Der Grund hierfür liegt an dem vorkonfiguriertem Footprint, dessen Pins keinem Layer (Front Mask/ Back Mask) zugewiesen wurden.

Der Mikrocontroller wurde zunächst auf den Lötpads positioniert, und anschließend wurde jeder einzelne Pin mittels einer dünnen Lötspitze mit Flussmittel verlötet. Durch diesen Prozess konnten die Anzahl der Kurzschlüsse minimiert werden und leichter ausgebessert werden. Die restlichen SMD-Komponenten (Widerstände, Kondensatoren, Längsregler) wurden mit einer Lötpaste auf die Lötpads platziert, welche durch eine Erhitzung einer Heißluftstation, in den Schaltkreis implementiert wurde. Da es sich bei den restlichen Komponenten um THT-Bauteile handelt, wurden diese mit einem herkömmlichen Lötkolben verlötet.

6.2.2 Roboter Revision A V1.00 Erkenntnisse

Nach dem Erhalt der Platinen machte sich der Fehler, welcher durch den Logik-Pegel-Wandler zustande kam, bemerkbar. Dadurch war das Funkmodul nicht mit dem SPI-Bus verbunden. Da jedoch der ICSP-Header mit den benötigten Pins des Buses verbunden ist, wurden vom Funkmodul an den Pins MOSI, MISO, SCK Drähte an die entsprechenden Pins des ICSP-Headers verlötet. Zusätzlich wurden zwei digitale Pins (CE, CSN) benötigt, welche durch den Pinheader J2 zur Verfügung standen. Diese wurden ebenfalls mit einer Drahtbrücke miteinander verbunden. Zunächst wurde der Logik-Pegel-Wandler dazwischen gelötet, um den entwickelten Schaltplan zu befolgen. In diesem wird beschrieben, dass MISO, MOSI und SCK auf 5V konvertiert werden. Nach diesem Aufbau konnte das Funkmodul zwar eine Verbindung herstellen, bekam allerdings nur zufällige fehlerhafte Werte.

Der Fehler lag hierbei an dem MISO-Pin, welcher nicht umgewandelt werden durfte. Als dieser Fehler verbessert wurde, konnten die richtigen Werte erhalten werden. Nach diesem Test wurde aus dem Datenblatt die Information entnommen, in welcher beschrieben wird, dass kein Logik-Pegel-Wandler benötigt wird, da der NRF24L01+ 5V tolerant ist. Daher wurde für einen weiteren Test der Wandler nicht mit eingebaut.

Mit der anschließenden bestückten Platine konnte die Kommunikation, samt der Fernsteuerung des Motorshield getestet werden und resultierte eine erfolgreiche Schaltung.

6.3 Funkmodul NRF24L01

Die Kommunikation zwischen dem Controller und dem Roboter wird durch das Funkmodul NRF24L01+ ermöglicht. Das Funkmodul arbeitet auf einem 2.4GHz ISM-Frequenzband, mit welchen ein drahtloser Austausch ermöglicht wird. Entwickelt von der Firma Nordic Semiconductor, zeichnet sich der NRF24L01+ durch seine hohe Effizienz und niedrige Leistungsaufnahme. Zusätzlich zur integrierten Antenne, die die Grundlage für die drahtlose Kommunikation bildet, besteht die Möglichkeit, externe Antennen anzuschließen, um die Reichweite des Moduls zu erweitern. Im Vergleich zum Vorgänger NRF24L01 bietet dieses Modell eine größere Reichweite und unterstützt höhere Datenraten bis zu 2Mbit/s, je nach Konfiguration.



Abbildung 31 : NRF24L01+ Funkmodul

Die Funktionsweise kann aus folgenden Blockschaltbild abgelesen werden:

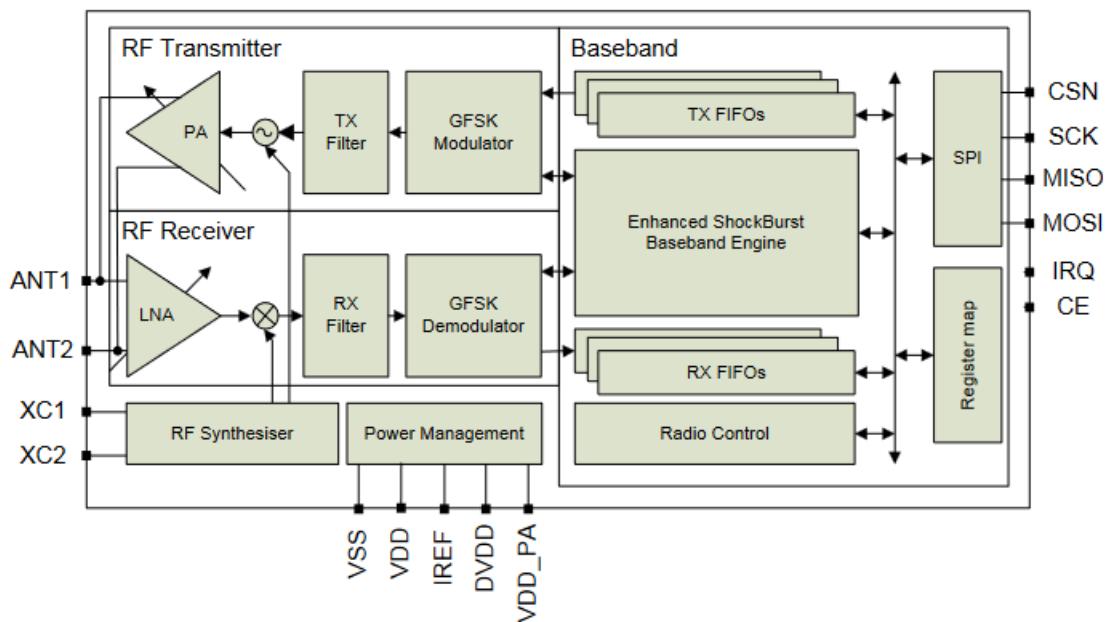


Abbildung 32: NRF24L01 Blockschaltbild

(https://www.sparkfun.com/datasheets/Components/SMD/nRF24L01Plus_Preliminary_Product_Specification_v1_0.pdf)

Die Informationen werden mittels der GFSK-Modulation (Gaussian frequency-shift keying) übertragen. Bei dieser handelt es sich um eine Art der Frequenzmodulation, bei der zwei Frequenzen für die Übertragung von Binärdaten verwendet werden. Hierbei verwendet der Sender RF-Transmitter („Radio Frequency“) das GFSK-Modulationsverfahren, wodurch digitale Daten in ein hochfrequentes Funksignal im 2,4GHz-Band umgewandelt werden. Diese modulierten Signale enthalten Informationen für die drahtlose Übertragung an andere NRF24L01+-Empfänger. Zusätzlich implementiert dieser Kollisionsvermeidungsverfahren, um sicherzustellen, dass mehrere Module im gleichen Netzwerk nicht gleichzeitig senden und kollidieren. Die Möglichkeit automatischer Wiederholungen von Datenpaketen kann bei

Bedarf aktiviert werden, um eine zuverlässige Kommunikation zu gewährleisten. Der RF-Receiver übernimmt die Rolle des Empfängers der Signale. Er dekodiert hochfrequente Funksignale, die durch den RF-Transmitter moduliert wurden. Der Empfänger des NRF24L01+ nutzt das GFSK-Verfahren, um die Frequenzverschiebungen im Signal zu interpretieren und die ursprünglichen digitalen Daten wiederherzustellen. Der Baseband-Block im NRF24L01+ ist für die Verarbeitung der digitalen Daten auf der Basisbandebene verantwortlich. Dieser übernimmt die Aufgabe der Modulation der digitalen Daten in ein hochfrequentes Funksignal, die Implementierung von GFSK, sowie der Demodulation empfangener Signale. Durch den SPI-Bus kann das Modul die empfangenen Daten an einem Mikrocontroller übertragen oder weitere Informationen übertragen und gesteuert werden.

6.3.1 Spannungsversorgung des Funkmoduls

Für den Betrieb des Funkmoduls ist eine Spannungsversorgung von 3,3V erforderlich. Daher wird eine Schaltung benötigt, die die 5V Betriebsspannung des Mikrocontrollers in 3,3V umwandelt. Um dies zu ermöglichen wird der MCP1703A als Längsregler eingebaut, da dieser einen simplen Schaltungsaufbau zur Stabilisierung der Spannung bietet. Aufgrund der Fehler, die durch Spannungsabweichungen beim Übertragen, sowie Senden der Daten auftreten können, können Fehler vorgebeugt werden. Ein Längsregler ist ein Spannungsregler, der überschüssige Energie in Form von Wärme abführt, um die Ausgangsspannung zu stabilisieren. Im Gegensatz zu einem Querregler kann die Ausgangsspannung mittels Dimensionierung von Widerständen nicht geändert werden, da diese vom Hersteller in der Entwicklung festgelegt wurden.

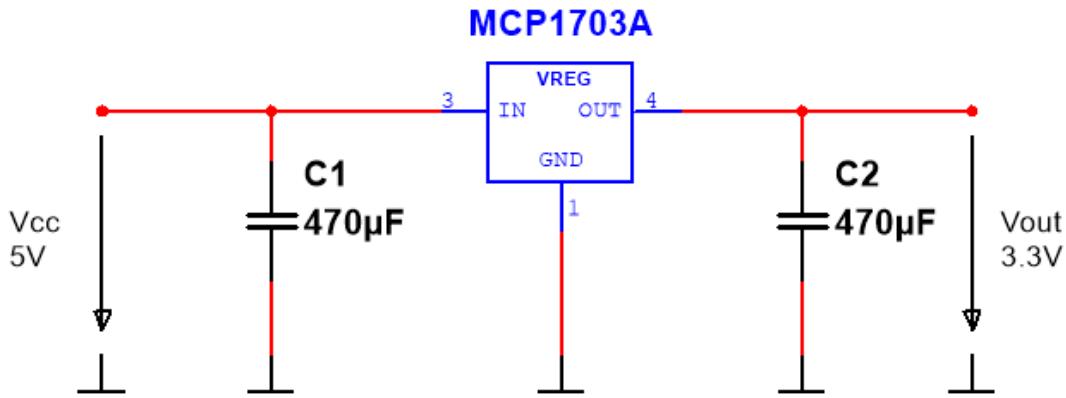


Abbildung 33 3,3V-Längsreglerschaltung

Um Spannungsschwankungen und Rauschaufreten zu verhindern, wird im Datenblatt empfohlen zwei gleichgroße Kondensatoren am Eingang und Ausgang parallel zu schalten. Durch den Kondensator C₁ wird sichergestellt, dass der Regelelement in einem stabilen Betriebszustand bleibt. Der Ausgangskondensator C₂ dient dazu, die Ausgangsspannung zu stabilisieren und Spannungsspitzen zu minimieren, damit die Ausgangsspannung auch bei sich ändernden Lastbedingungen konstant bleibt.

6.3.2 Spannungsschwankungen des NRF24L01

Wenn man das Funkmodul als Receiver betreibt, können Spannungsschwankungen beim Empfangen von Daten auftreten, die den Datenaustausch hindern können.

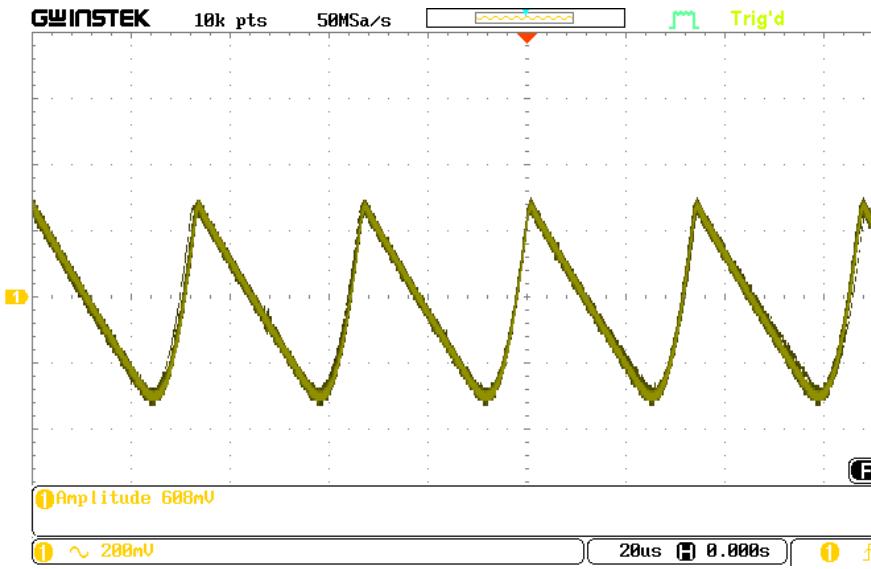


Abbildung 34 NRF24L01 Messung der Spannungsversorgung

In der oberen Messung wurde die Versorgungsspannung des Funkmoduls mit einem Oszilloskop gemessen. Hierbei kann man die Schwankungen, welche von 600mV – 800mV variieren erkennen, die während der Kommunikation zwischen zwei Modulen auftreten. Daher wurde ein C=10 μ F Elko zwischen den der Spannungsversorgung eingebaut. Um die resultierende Messung besser interpretieren zu können, wurde ein GPIO-Pin des Arduinos auf „1“ geschaltet.

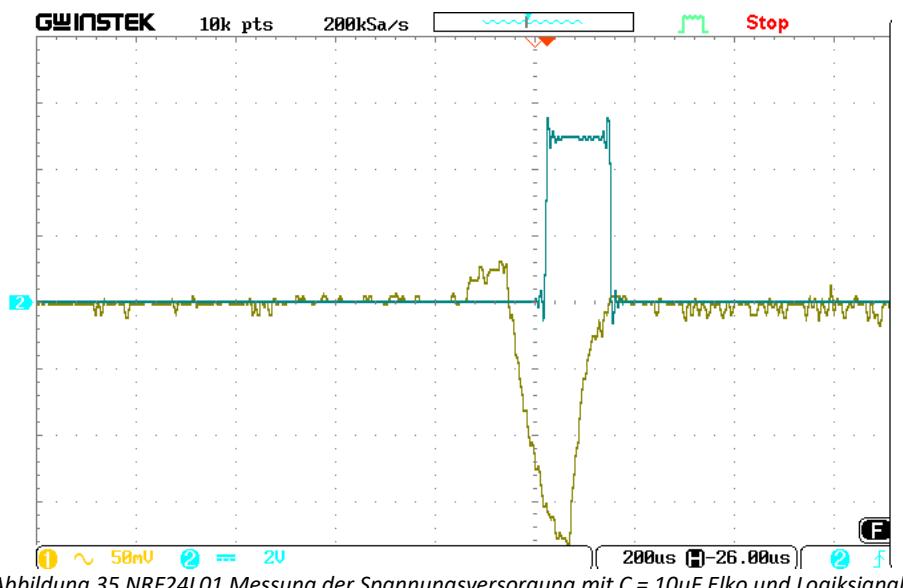


Abbildung 35 NRF24L01 Messung der Spannungsversorgung mit C = 10 μ F Elko und Logiksignal

Die in der oberen Messung erkennbaren Störung tritt nur einmal zu Beginn des Datenaustausch auf. Mit einem größeren Kondensator ist eine Verbindung, samt einen korrekten Datenaustausch möglich.

6.3.3 Störungen beim NRF24L01+

Um Störungen aufgrund von Interferenzen zu umgehen, können beim NRF24L01+ sowohl hardware- als auch softwareseitige Maßnahmen ergriffen werden. Eine Erweiterung im Vergleich zum Vorgänger NRF24L01 besteht darin, dass der NRF24L01+ eine austauschbare Antenne besitzt, während die ältere Version eine auf dem PCB montierte Antenne hat. Trotz dieser Verbesserung kann das Funkmodul aufgrund seiner 2,4-GHz-Frequenz anfällig für Störungen durch WiFi sein, was zu Verbindungsproblemen und einem Ausbleiben des Datenaustauschs führen kann. Dieses Problem wird ebenfalls in der offiziellen Library des Moduls auf GitHub dokumentiert.

Um Interferenzen hardwareseitig zu reduzieren, wäre es ideal, das Funkmodul in einem Faraday'schen Käfig zu platzieren. Da dies jedoch in der Praxis nicht immer möglich ist, besteht eine alternative Lösung darin, den NRF24L01+ Chip auf der Platine mit einer Schicht Aluminiumfolie zu umwickeln. Dies kann dazu beitragen, Interferenzen, die durch WiFi verursacht werden, zu reduzieren und den Datenaustausch zwischen den Geräten zu ermöglichen.

Darüber hinaus besteht die Möglichkeit, die Kanäle in der Software so zu wählen, dass sie nicht mit den Funkmodulen interferieren. Hierbei muss in der Setup-Konfiguration festgelegt werden, auf welchen Kanälen das Modul sendet und empfängt.

6.4 Komponenten des Roboters Revision A

6.4.1 Motorshield

Da die Information, die der Mikrocontroller ausgibt, lediglich 0 oder 1 ist (0V / 5V) kann ein einzelner DC-Motor diese nicht verarbeiten. Bei einer direkten Verbindung zwischen dem Motor und dem Mikrocontroller würde sich dieser nur mit einer konstanten Geschwindigkeit in einer Richtung oder gar nicht bewegen. Deswegen ist es möglich mit Hilfe eines dazwischen geschalteten Motorshields einen Motor mit Software in verschiedenen Modi anzusteuern.

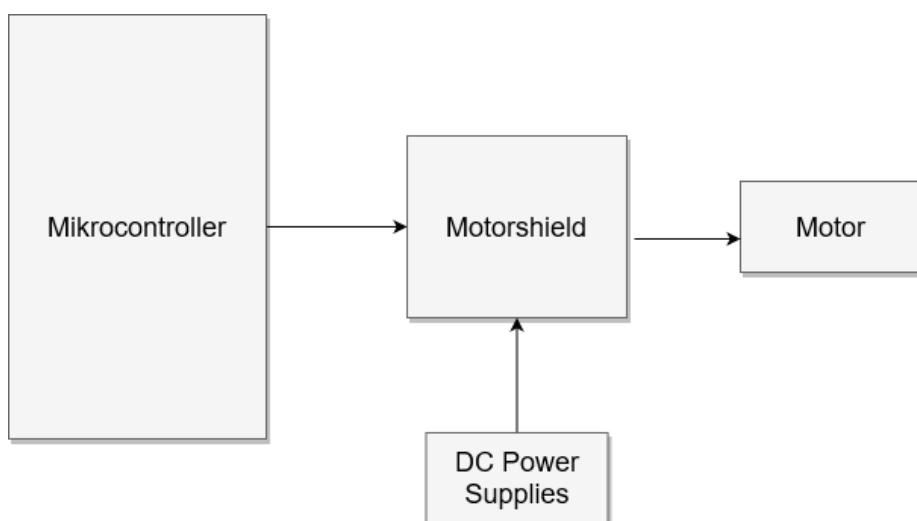


Abbildung 36 Motorshield-Blockschaltbild

Das Motorshield steuert den Motor mittels einer „H-Brücke“ mit PWM-Signalen an. Je nach Motorgröße ist eine zusätzliche Spannungsquelle notwendig, da nicht jeder mit 5V in Betrieb genommen werden kann. Da in der Diplomarbeit ein 12V Motor verwendet wird, wird im Blockschaltbild die entsprechende Spannungsquelle integriert.

6.4.2 H-Brücke

Eine H-Brücke besteht aus vier Transistoren, die in einer „H“-Form geschaltet sind, um den Stromfluss in zwei Richtungen durch den Motor steuern zu können. In diesem Motorshield besteht die H-Brücke aus N- und P-Channel MOSFETs.

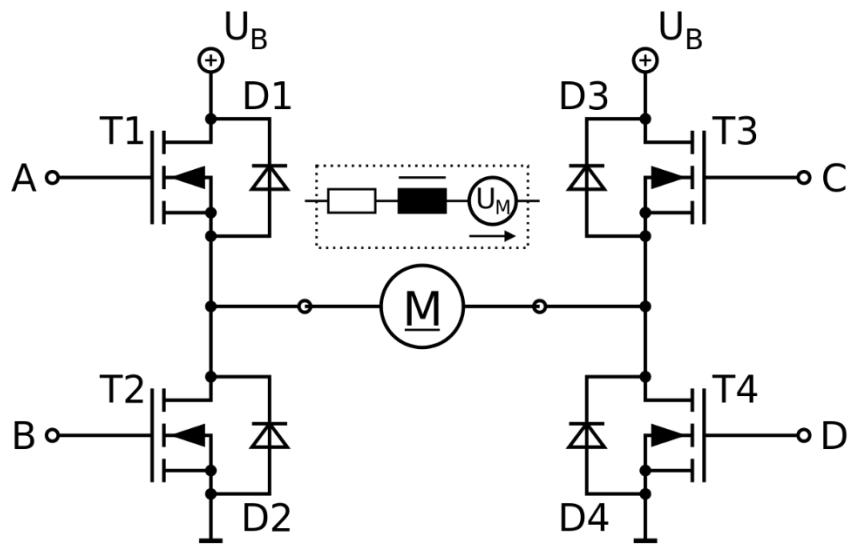


Abbildung 37: H-Brücke Schaltplan (<https://de.wikipedia.org/wiki/Vierquadrantensteller>)

Um die Steuerung eines Motors zu ermöglichen, werden die Gate-Eingänge an den PWM-Pins des Mikrocontroller angeschlossen. Durch das Duty-Cycle-Verhältnis ist es nun möglich den Motor in verschiedenen Drehrichtungen mit einer dazugehörigen Geschwindigkeit rotieren zu lassen.

6.4.3 Logik-Pegel-Wandler

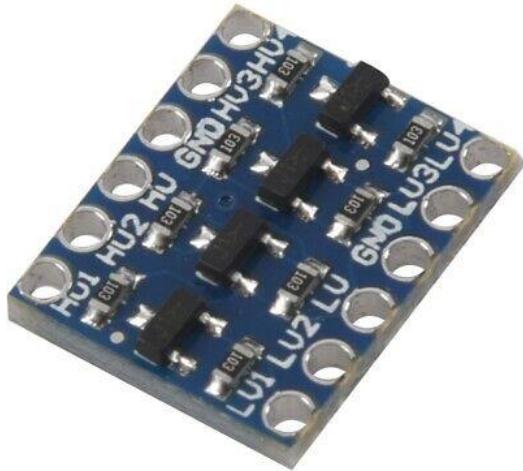


Abbildung 38: Logik-Pegel-Converter

In der digitalen Elektronik kann ein Signal entweder den Zustand „ein“ oder „aus“ haben. Der EIN-Zustand wird als logisch HIGH „1“ bezeichnet, während der AUS-Zustand logisch LOW „0“ ist. Logikpegel repräsentieren definierte Spannungszustände, bei denen ein Signal als hoch oder niedrig klassifiziert wird. Ein Logikpegelwandler oder Logic-Level-Converter ist eine benötigte Schaltung, wenn Geräte mit unterschiedlichen Spannungsanforderungen miteinander verbunden werden. Ein Logikpegelwandler passt die Spannung entsprechend den Anforderungen eines Geräts an, indem er sie erhöht oder verringert. In diesem Zusammenhang wird ein Logikpegelwandler verwendet, um die Kommunikation zwischen einem

Modul, das mit 3,3V-Pegeln arbeitet, und einem 5V-Mikrocontroller zu ermöglichen.

Der Funktionsablauf dieses elektronischen Bausteins lässt sich aus dem folgenden Schaltplan ableiten:

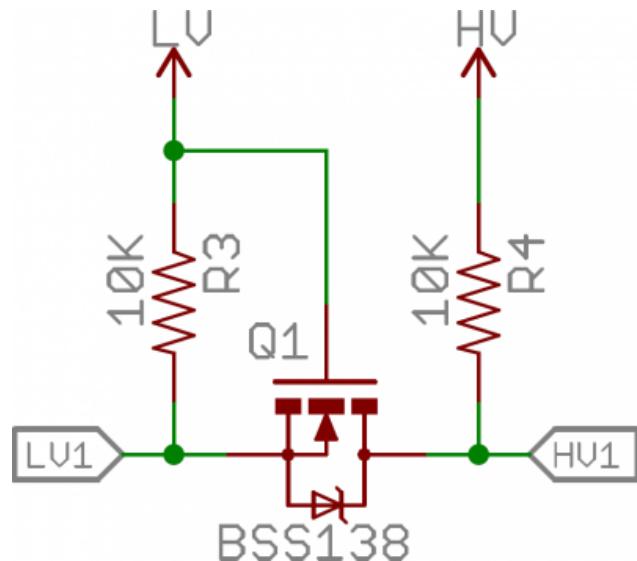


Abbildung 39: Logik-Level-Konverter (<https://learn.sparkfun.com/tutorials/bi-directional-logic-level-converter-hookup-guide/all>)

Ein n-Kanal-MOSFET und zwei Widerstände werden in diesem Schaltungskonzept verwendet, um einen bidirektionalen Logikpegelwandler zu realisieren. Abhängig von den Größenunterschieden werden unterschiedliche Spannungsspeicher mit HV (High Voltage) oder LV (Low Voltage) verbunden. Wenn die Spannung an LV1 3,3 V beträgt, verbleibt der MOSFET im Sperrbereich, da die Spannungsdifferenz zwischen Gate und Source geringer ist als die Schwellenspannung, die für das Einschalten des MOSFET erforderlich ist. Die Spannung an HV1 wird durch den Widerstand R4 auf 5V gezogen. Somit resultiert eine Ausgabe von 3,3V an LV1 und 5V an HV1. Im Falle einer 0V-Spannung an LV1 nimmt der Einschaltwiderstand des MOSFET aufgrund der Spannungsdifferenz zwischen Gate und Source ab. Da der MOSFET in diesem Zustand nahezu, wie ein Kurzschluss wirkt, wird HV1 auf 0V gezogen. Folglich resultiert 0V an beiden Kanälen. Bei einer Spannung von 5V am HV1 bleibt der MOSFET im Sperrbereich, und der Knoten an LV1 wird durch den Pull-Up-Widerstand R3 auf 3,3V gezogen. Wenn HV1 0V beträgt, beginnt die Diode des MOSFET zu leiten, da sie vorwärts gepolt wird. Dies zieht die Spannung am LV1 nahe an 0V. Daher ergeben sich auch in diesem Fall die entsprechenden Spannungen auf der anderen Seite.

6.4.4 Serial Peripheral Interface Bus

Das Serial Peripheral Interface (SPI) ist ein synchroner serieller Bus und ermöglicht eine effiziente bidirektionale Datenübertragung zwischen einem Master-Gerät und mehreren Slave-Geräten. Der Bus besteht typischerweise aus vier Hauptleitungen, wobei jede Leitung eine spezifische Funktion erfüllt.

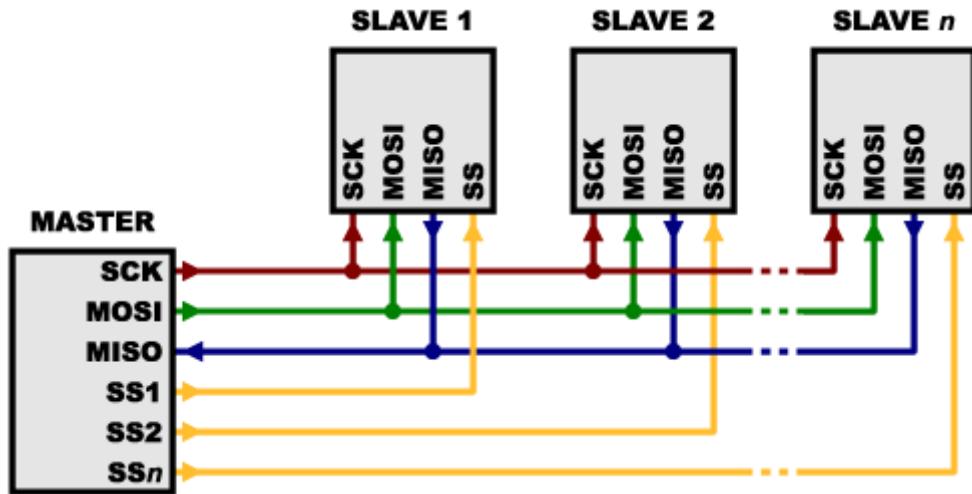


Abbildung 40: SPI-Bus (<https://www.rc-modellbau-portal.de/index.php?threads/spi-schnittstelle-beim-arduino-co.6994/>)

Die MOSI-Leitung (Master Out Slave In) dient dazu, dass der Master Daten an die Slaves übertragen kann. Umgekehrt ermöglicht die MISO-Leitung (Master In Slave Out) den Slaves, Daten an den Master zu senden. Die SCLK-Leitung (Serial Clock) überträgt den Takt vom Master an die Slaves, wodurch eine synchronisierte Datenübertragung möglich wird. Die SS/CS-Leitung (Slave Select/Chip Select) wird für jedes Slave-Gerät separat verwendet, um es auszuwählen und die Kommunikation mit dem Master zu initiieren.

Mode Nr.	CPOL	CPHA
0	0 - LOW	0 - fallende Flanke
1	0 - LOW	1 - steigende Flanke
2	1 - HIGH	0 - fallende Flanke
3	1 - HIGH	1 - steigende Flanke

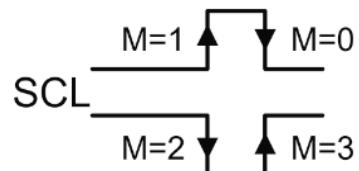


Abbildung 41: SPI-Bus CLK-Konfiguration (<https://edistechlab.com/wp-content/uploads/2020/11/Screenshot-2020-11-22-at-09.37.57.png>)

Des Weiteren muss eine Synchronisation des Protokolls geeinigt werden, welche mit der CLK-Leitung gewählt wird. Dadurch ist es möglich den Zeitpunkt der Datenübertragung, welche durch fallende oder steigende Flanke, sowie des aktiven Zustandes (LOW/HIGH) zu konfigurieren.

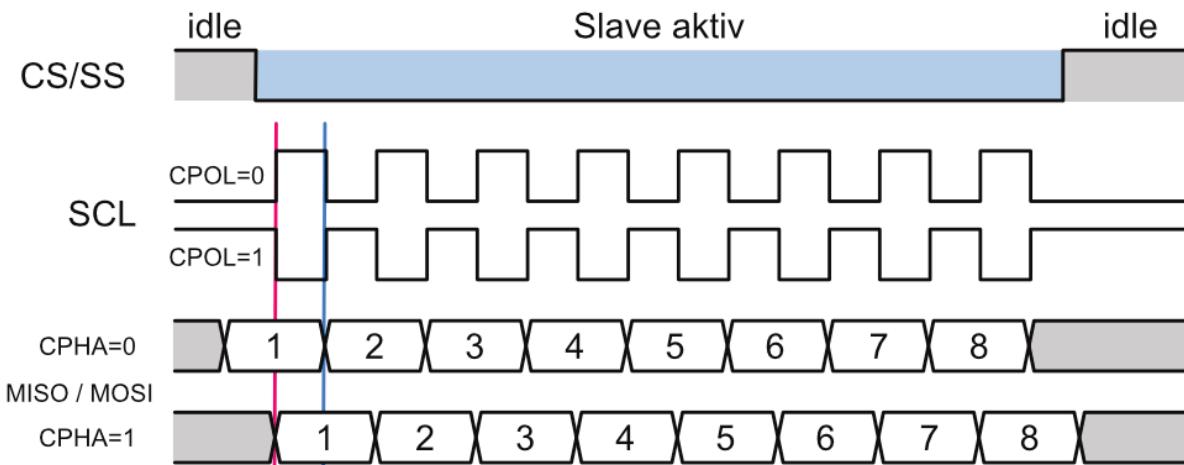


Abbildung 42: SPI-Bus Kommunikation (<https://edistechlab.com/wie-funktioniert-spi/?v=fa868488740a>)

Die Kommunikation erfolgt durch das gleichzeitige Senden und Empfangen von Daten, in einem vorher festgelegten Taktzyklus. Zuerst wird mit der CS-Leitung bestimmt, ob der Master oder der Slave senden darf. Danach kommt es auf die Konfiguration des Clocksignal an, in welcher bestimmt wird, wann die Daten übertragen werden. Als letztes werden die Daten über der MOSI- oder MISO-Leitung Bit für Bit in Serie versendet.

6.4.5 Pierce-Oszillatort

Da für die Inbetriebnahme des Mikrocontrollers ein Taktsignal benötigt wird, ist eine Oszillatorschaltung notwendig. Mit den Informationen aus dem Datenblatt wird ein Quarzoszillatot dazu gebaut, der den ATMEGA2560 mit einem Rechtecksignal mit 16MHz versorgt.

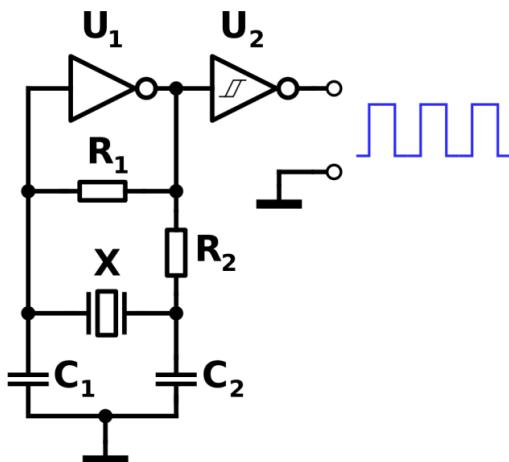


Abbildung 43: Schaltplan Pierce-Oszillatot (<https://de.wikipedia.org/wiki/Quarzoszillatot>)

Die Bauteile sind essenziell, um die Erzeugung und Stabilisierung der Taktfrequenz zu sichern. Der Quarz fungiert als frequenzbestimmendes Element, indem er aufgrund seiner piezoelektrischen Eigenschaften Schwingungen erzeugt, die als präzise Taktfrequenz dient. Die Kondensatoren sind in der Regel parallel zum Quarz geschaltet. Sie dienen dazu, die nötige Kapazität für die Schwingungsbildung des Quarzes bereitzustellen. Die Kondensatorwerte von 22pF wurden gemäß den Angaben im Datenblatt ausgewählt, das die Gleichheit der Kapazitäten für beide vorschreibt. In einigen Schaltungen werden Widerstände parallel zu den

Kondensatoren geschaltet, um die Impedanz des Oszillators zu beeinflussen. Die Widerstände können dazu beitragen, die Schwingungsfrequenz zu stabilisieren und unerwünschte Schwingungsmoden zu unterdrücken. Der Inverter U_1 erzeugt eine um 180° phasenverschobene Ausgangsspannung, die für die positive Rückkopplung und Stabilisierung der Schwingungen im Oszillatorkreis, insbesondere bei Verwendung eines Quarzresonators, entscheidend ist. Der Schmitt-Trigger dient dazu, das Ausgangssignal des Oszillators zu stabilisieren und sicherzustellen, dass das Signal sich im Pegelbereich des Mikrocontroller befindet. Beim ATMEGA 2560 ist der Großteil der Oszillatorschaltung im IC implementiert, daher sieht die Beschaltung wie folgt aus:

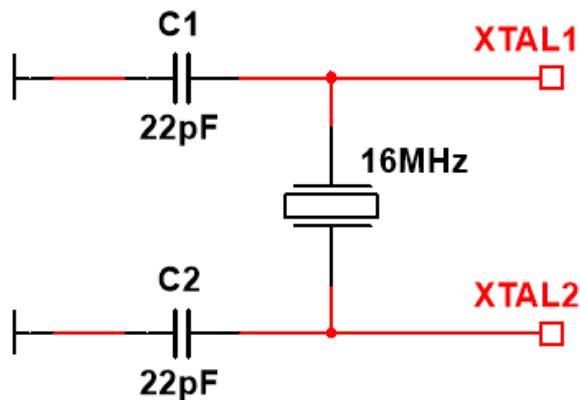


Abbildung 44: Oszillatorschaltung beim ATMEGA2560

Da der Mikrocontroller mit einer Taktfrequenz von 16MHz arbeitet, wird ein Quarz mit diesem Wert eingebaut. Die Pins XTAL1 und XTAL2 stellen hierbei die Anschlüsse des im IC eingebauten Oszillator da. Die Frequenz kann somit mit dem Wert des Quarzes gewählt werden.

6.4.6 ESP32 mit Kamera

Als letzte Erweiterung wurde ein ESP32 mit einer integrierten Kamera verbaut. Der ESP32 verfügt über ein WLAN-Modul, wodurch es möglich ist, einen Webserver auf diesem zu betreiben, ohne den laufenden Server auf dem Raspberry Pi zu belasten. Dieser kommuniziert nicht mit der eigentlichen Roboterplatine, daher wird der Mikrocontroller lediglich mit Vcc = 5V versorgt. Da der Programmer, sowie die USB-Schnittstelle nicht auf der Platine verbaut sind, wurde ein zusätzliches Modul benötigt, die Software auf den Mikrocontroller zu laden. Durch die Implementierung des ESP32 mit der verbauten Kamera, ist es möglich die Fahrt des Roboters aufzunehmen und auf einem Webserver zu übertragen.

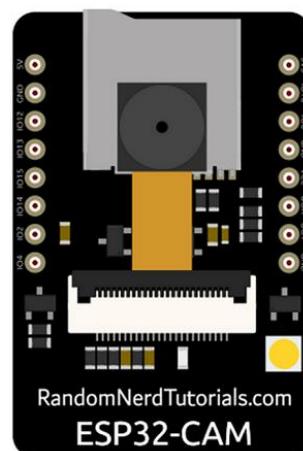


Abbildung 45: ESP32-CAM

6.5 Hardware-Design Roboter Revision B

Aufgrund der funktionierenden Schaltung im Kapitel 6.2 Hardware-Design Roboter Revision A wird der Entwicklungsplan konsequent umgesetzt und ein Nachfolger der Roboterschaltung mit Verbesserungen entworfen. Es wird eine Weiterentwicklung der Roboterschaltung mit gezielten Verbesserungen konzipiert, wobei diese als zusätzliche Features integriert werden, ohne die grundlegende Fahrzeugsteuerung zu beeinträchtigen. Die primäre Zielsetzung besteht darin, den Roboter aufzuwerten. Die implementierten Erweiterungen sind im nachfolgenden Blockschaltbild detailliert dargestellt:

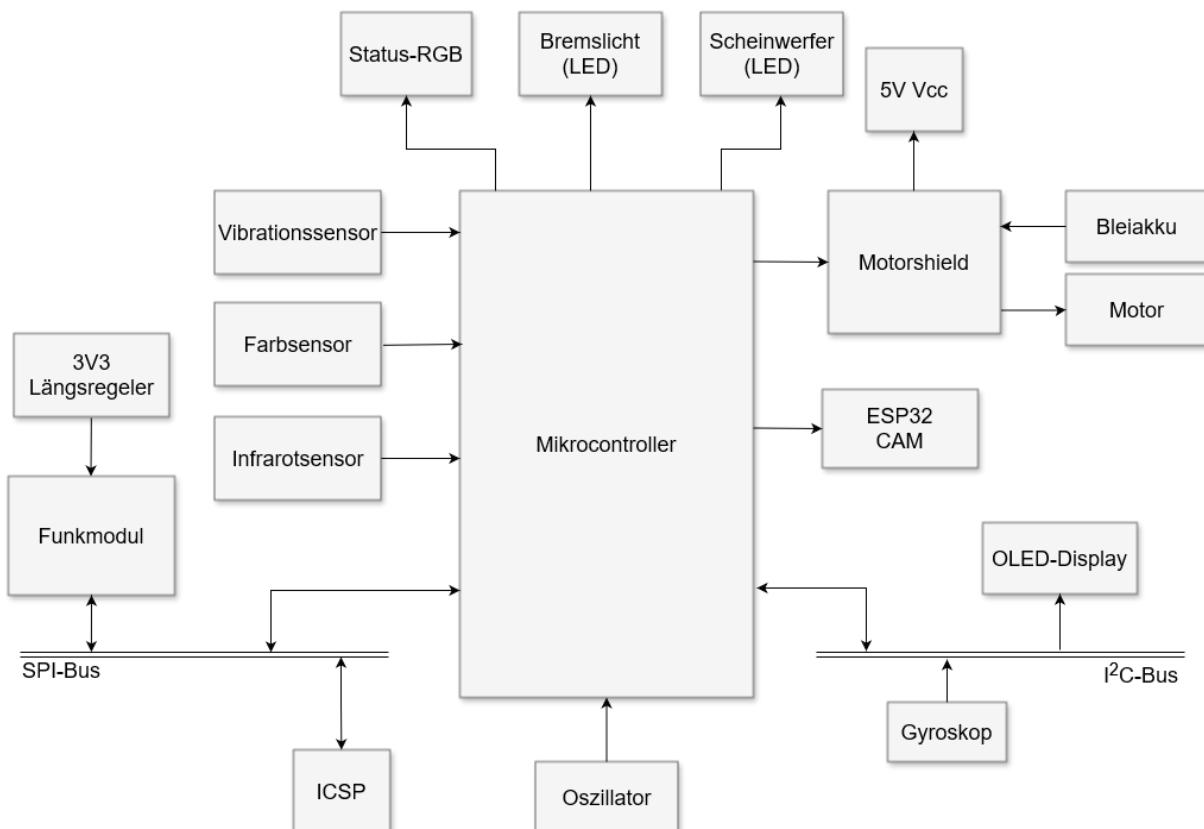


Abbildung 46: Blockschaltbild von Roboter Rev B V1.00

Wie bereits in der vorherigen Version bleiben die Grundkomponenten (Motoransteuerung, Funkmodul, Oszillator), welche für das Ansteuern des Fahrzeugs benötigt werden unverändert. Allerdings wird der Logik-Pegel-Wandler nicht mehr verwendet, da dieser für den Datenaustausch zwischen dem Funkmodul und dem Mikrocontroller nicht notwendig ist. Der genaue Grund für diese Entscheidung wird im Kapitel 0

Roboter Revision A V1.00 Erkenntnisse beschrieben. Um dem Roboter autonomes Fahren zu ermöglichen, werden Infrarotsensoren integriert. Diese Sensoren ermöglichen es dem Fahrzeug, Hindernisse eigenständig zu umfahren, ohne dabei von der Steuerung beeinflusst zu werden. Zu diesem Zweck werden drei Infrarotsensoren eingebaut, um eine präzise Erfassung der Umgebung sicherzustellen. Ergänzend dazu wird ein Farbsensor integriert, der es dem Roboter ermöglicht, Farben in seiner Umgebung zu erkennen und darauf zu reagieren. Dadurch sollen einprogrammierte Software-Funktionen ausgeführt werden. Darüber hinaus

wird im Fahrzeug ein Vibrationssensor eingebaut, der bei intensiven Bewegungen des Roboters ein HIGH-Signal an den Mikrocontroller sendet. Auch mit diesem Sensor können Softwareprogramme aufgerufen werden. Zusätzlich ermöglicht ein Gyroskop eine Messung der Geschwindigkeit, welches durch ein 3-Achsen Accelerometer die Messwerte ermittelt. Des Weiteren ist dieser in der Lage die Temperatur seines Umfelds zu erfassen. Die Kommunikation mit dem Mikrocontroller erfolgt über den integrierten I²C-Bus. Hierfür wird ein OLED-Display eingebaut, das durch einen Softwarealgorithmus auf Sensoren reagiert. Zusätzlich empfängt das OLED-Display Informationen über den I²C-Bus. Um den Roboter mit zusätzlicher Unterhaltungselektronik auszustatten, werden LEDs als Scheinwerfer- und Bremslichter eingebaut. Die Software wird darauf zugreifen und die entsprechenden LEDs aktivieren, abhängig von der Fahrtrichtung des Roboters.

Des Weiteren wurde ein ESP32 mit einer eingebauten Kamera in die Schaltung integriert. Dieser ist dafür verantwortlich, die Perspektive während der Fahrt des Roboters zu erfassen und auf einem Webserver zu übertragen, der unabhängig vom Raspberry Pi betrieben wird.

6.5.1 Roboter Revision B V1.00 PCB

Das Blockschaltbild, wie in 6.5 Hardware-Design Roboter Revision B dargestellt, wurde in KiCAD als Schaltplan und Platinenlayout umgesetzt. Wie bereits bei seinem Vorgänger wird für die Produktion dieses PCBs auf dem externen Dienstleister Aisler zurückgegriffen.

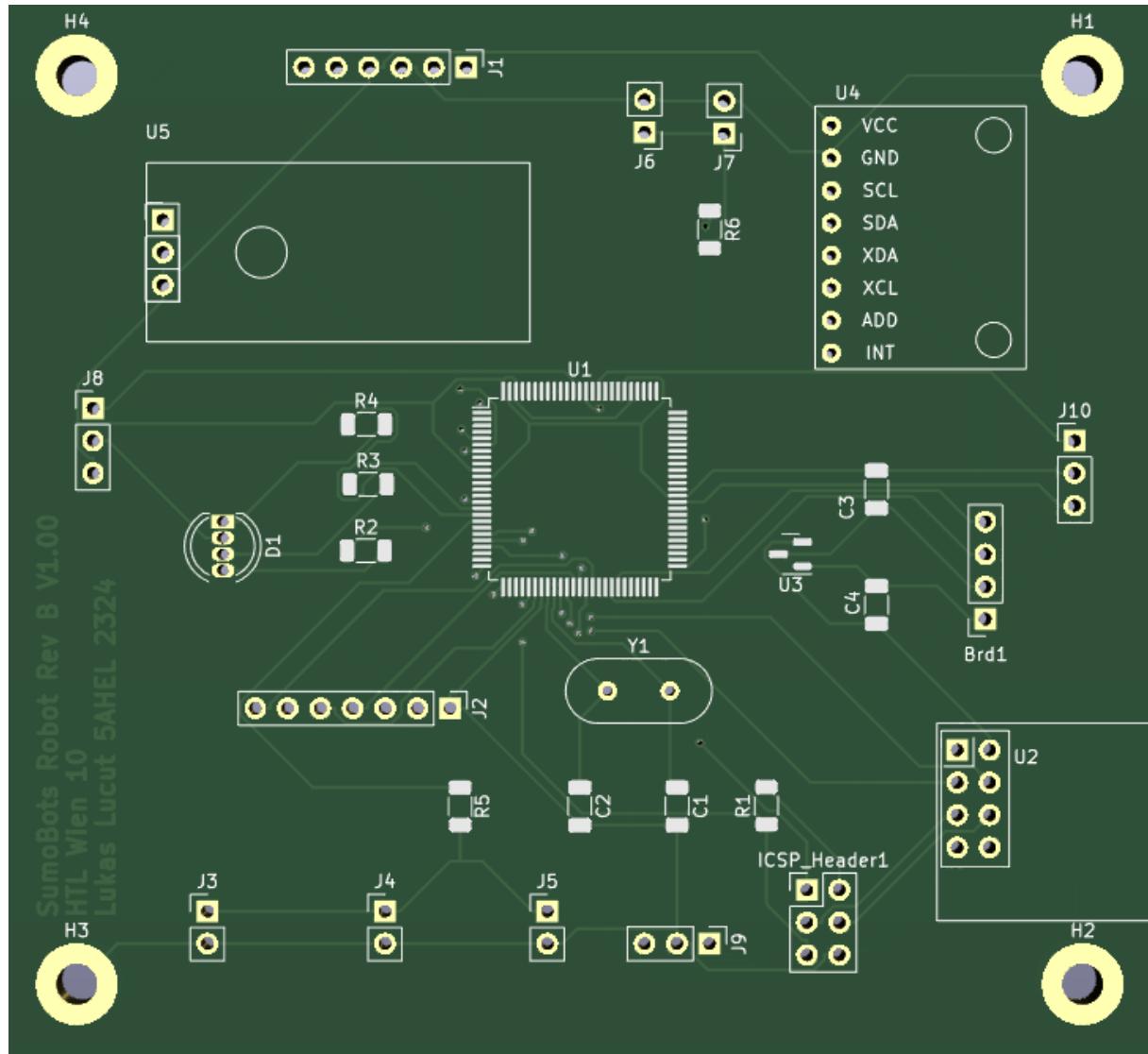


Abbildung 47: Roboter Rev B V1.00 PCB

Durch die gewonnenen Erkenntnisse im Fachbereich SMD-Löten wurde der damit verbundene Risikofaktor eliminiert, was es ermöglichte, die Platine mit kleineren Abmessungen zu dimensionieren. Zusätzlich zu den implementierten Sensoren befinden sich nun mehrere Bauteile auf der Platine. Einige dieser Bauteile, wie das OLED-Display, LEDs, Funkmodul und Farbsensor, müssen am Gehäuse des Roboters befestigt werden, um ihre Funktion sicherzustellen. Diese wurden auf einer Lochrasterplatine verlötet und mit Drähten an der Hauptplatine angeschlossen, weshalb sie als Pinheader auf dem PCB realisiert wurden. Dies führte dazu, dass die Positionierung dieser Bauteile ein zu berücksichtigender Faktor während des Designs wurde. Aus diesem Grund wurden die LEDs J6 & J7, die als Scheinwerfer fungieren sollen, an der Vorderseite des PCB positioniert. Gleiches gilt für die LEDs J3 & J4 & J5, die als Bremslichter dienen sollen. Die RGB-LED bleibt auf der Hauptplatine, da sie dazu dient, das Debuggen zu erleichtern. Der Farbsensor wird an der Unterseite des Robotergehäuses

angebracht, um während der Fahrt die darunter liegenden Farben zu erkennen. Der Vibrationssensor und das Gyroskop werden hingegen auf der Platine befestigt, da sie aufgrund ihrer einstellbaren Sensitivität auf ruckartige Bewegungen reagieren würden und somit eine präzise Messung gewährleisten sollen. Zusätzlich wurden die Bohrlöcher, durch die die Platine am Gehäuse befestigt wird, mit der Masse verbunden.

6.5.2 Roboter Revision B V1.00 Erkenntnisse

Neben den Erweiterungen wurden auch die Fehler, die bei seinem Vorgänger Rev A V1.00 aufgetreten sind, behoben. Im ersten Schritt wurden die Sensoren getestet, um sicherzustellen, dass sie ordnungsgemäß funktionieren. Zuerst wurde der Vibrationssensor in die Schaltung integriert, da dieser eine einfache Funktion hat und sich auch gut für das Debuggen der Software eignet. Im ersten Versuch konnten Daten vom ATMEGA2560 erfasst werden und anschließend auf dem Webserver, der auf dem Raspberry Pi 3 läuft, in einem Graphen dargestellt werden. Danach wurde das Gyroskop verbaut, bei dem zunächst fehlerhafte Informationen übertragen wurden. Der Grund hierfür lag an der SCL-Leitung, da der zugehörige Pin beim Mikrocontroller nicht mit dem Lötpad verbunden war, wodurch eine Datenübertragung nicht möglich war. Allerdings konnte nach dieser Fehlerbehebung dieser ebenfalls in Betrieb genommen werden und die Daten auf dem Webserver dargestellt werden. Außerdem konnte der Farbsensor, erfolgreich implementiert werden, der mittels einer geschriebenen Software die Farben Weiß und Schwarz erkennt und bei Änderungen reagiert. Aufgrund des Zeitmangels und fehlender Komponenten war es nicht möglich, die Infrarotsensoren und ESP32-CAM in den Roboter einzubauen und zu verwenden.

6.6 Komponenten des Roboters Revision B

In diesem Abschnitt werden die Erweiterungen des Roboters in Revision B im Detail erläutert. Da die bereits in Revision A verbauten Komponenten (Oszillator, Funkmodul, Spannungswandler, Motorshield) in Kapitel 6.4 Komponenten des Roboters Revision A beschrieben wurden, wird hier auf eine erneute Beschreibung verzichtet.

6.6.1 I²C-Bus

Da diverse Sensoren über den I²C-Bus kommunizieren, wird diese Peripherie-Komponente des Mikrocontrollers eingesetzt. Der Datentransfer findet hierbei über einem Master initiiert, der über eine angesprochene Adresse einen Slave auswählt und anschließend anspricht. Anders als beim SPI-Bus, ist es hierbei möglich, dass mehrere Master im Bus tätig sind. Des Weiteren arbeitet der I²C-Bus mit positiver Logik.

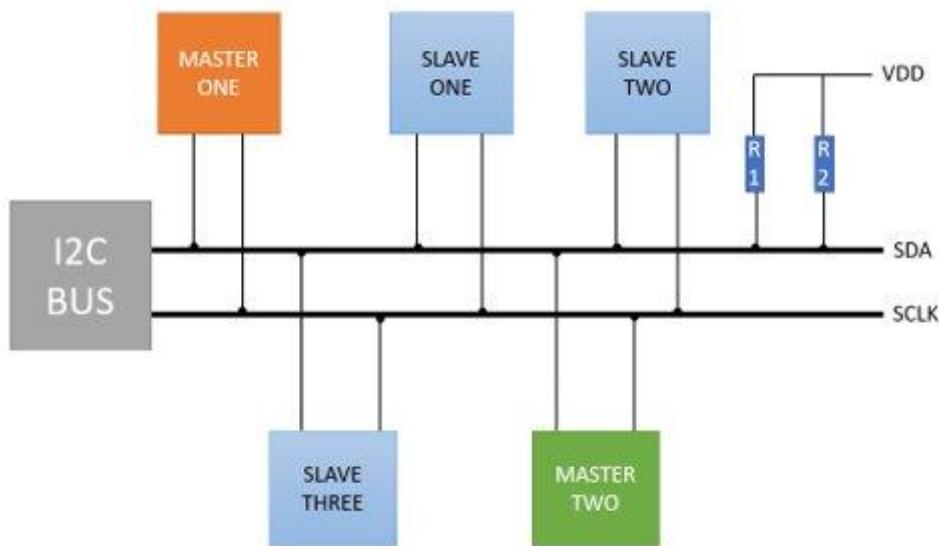


Abbildung 48: I²C-Bus (<https://www.design-reuse.com/articles/54776/i2c-interface-timing-specifications-and-constraints.html>)

In der Abbildung 48 kann man den Aufbau einer Kommunikation mittels I²C-Bus genauer betrachten. Hierbei werden zwei Leitungen benötigt SCLK (Serial Clock) und SDA (Serial Data). Die Taktsignal kommt durch die SCLK-Leitung zustande, während der Datenaustausch auf der SDA-Leitung stattfindet. Die Pull-Up Widerstände R₁ und R₂ dienen zur Sicherstellung, dass die Datenleitungen im Leerlauf auf einen definierten High-Pegel liegen, was durch die Open-Collector-Ausgänge ermöglicht wird. Mit diesem Aufbau ist eine bidirektionale Kommunikation zwischen Master und Slave möglich.

Im I²C-Bus werden Slave-Geräte über ihre eindeutigen 7-Bit-Adressen adressiert, die der Master sendet. Diese Adressen ermöglichen es dem Master, gezielt mit einem bestimmten Slave-Gerät zu kommunizieren. Im folgenden Diagramm wird die Kommunikation des Buses genauer verdeutlicht.

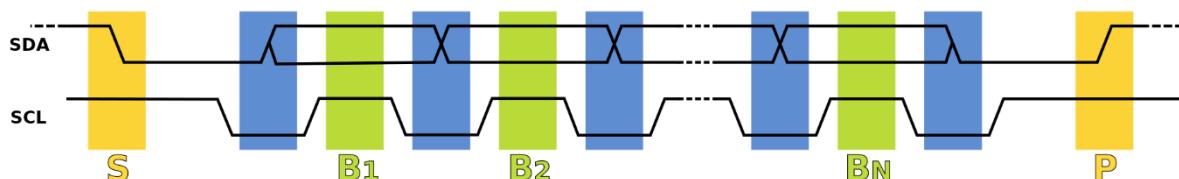


Abbildung 49: I²C-Bus Zeitverhalten (<https://de.wikipedia.org/wiki/I%C2%BCC>)

Die Kommunikation startet mit einem Startbit (S), welches vom Master gesetzt, während das Clocksignal, ebenfalls vom Master gesendet, auf logisch HIGH ist. Als nächstes sendet der Master eine Adresse, um den jeweiligen Slave anzusprechen. Nachdem der Slave ausgewählt wurde, überträgt der Master ein Richtungsbit, um anzugeben, ob es sich um eine Lese- oder Schreiboperation handelt. Nach dieser Initiierung werden die Daten übertragen. Nach jedem empfangenen Byte sendet der Empfänger (Master oder Slave) ein Bestätigungsbit (Acknowledgebit) über die SDA-Leitung, um den erfolgreichen Empfang des Bytes zu signalisieren. Am Ende der Kommunikation sendet der Master eine Stopbit (P), indem er die SDA-Leitung von LOW auf HIGH zieht, während die SCL-Leitung HIGH bleibt. Dies signalisiert das Ende der Übertragung.

6.6.2 OLED Display SSH1106

Ein OLED-Display (Organic Light-Emitting Diode) des Typs SSH1106 ist ein elektronisches Bauteil, das auf organischen Verbindungen basiert, welche bei Anlegen einer elektrischen Spannung Licht emittieren. Der SSH1106 ist als monochromatisches Grafik-Display konzipiert und zeichnet sich durch seine hohe Helligkeit, Kontrastverhältnis sowie schnelle Reaktionszeit aus. Das Display besteht aus einer Matrix von organischen Leuchtdioden, die auf einem dünnen Substrat angeordnet sind. Jede einzelne OLED fungiert als ein Pixel auf dem Display. Der SSH1106 verwendet eine passive Matrix-Anordnung, bei der jede Zeile und Spalte der Matrix mit Treibertransistoren gesteuert wird, um die Helligkeit jedes Pixels zu regulieren. Wenn eine elektrische Spannung an eine bestimmte Zeile und Spalte angelegt wird, werden die organischen Materialien in diesem Pixel aktiviert. Diese Materialien emittieren dann Licht, wodurch das Pixel leuchtet. Durch die präzise Steuerung der Spannung in jeder Zeile und Spalte kann der SSH1106 komplexe Grafiken und Texte auf dem Display darstellen. Die Kommunikation erfolgt über den I²C-Bus.



Abbildung 50 OLED-Display SSH1106

6.6.3 Vibrationssensor SW-420

Das SW-420 Vibrationssensor-Modul besteht aus einem SW-420 Vibrationsschalter und einem LM393 Spannungsvergleicher. Im SW-420 Vibrationsschalter befinden sich eine Feder und ein Stab innerhalb eines Rohres. Bei auftretenden Vibrationen kommt es zum Kontakt zwischen der Feder und dem Stab, wodurch der Stromkreis geschlossen wird. Der integrierte Vibrationssensor im Modul erfasst diese Schwingungen und wandelt sie in elektrische Signale um. Der LM393 Komparator vergleicht diese Signale mit einer Referenzspannung, die über den Trimmer eingestellt wird. Wenn die Amplitude des Signals die Referenzspannung überschreitet, gibt der Komparator ein HIGH-Signal aus, andernfalls ein LOW-Signal. Auf der Platine befinden sich auch LED Anzeigen für die Stromversorgung und den digitalen Ausgangsstatus. Es hat eine einfache und klare 3-Pin-Schnittstelle: VCC, GND und DO (digitaler Ausgang). Es unterstützt eine Stromversorgung von 3,3V oder 5V. Dieses Modul ist mit jedem Mikrocontroller kompatibel, der über einen digitalen Eingang verfügt.



Abbildung 51: Vibrationssensor SW-420

Dieser Sensor soll hauptsächlich das Debuggen der Übertragung der Sensorwerte an den Server ermöglichen. Da hierbei lediglich eine „0“ oder „1“ erkannt wird, eignet sich dieser zur Messung der Laufzeit des Funkmoduls.

6.6.4 Farbsensor TCS3200

Der TCS3200 Farbsensor ist ein elektronisches Bauteil, das in der Lage ist, Farben präzise zu identifizieren und zu erkennen. Dieser Sensor ist in der Lage, Licht in verschiedenen Farben zu erfassen und die Intensität dieser Farben zu messen. Der eigentliche Sensor ist der verbauten IC innerhalb des schwarzen Gehäuses, welches die Genauigkeit der Farberkennung verbessern soll, indem durch die vier weißen LEDs die Umgebung belichtet wird. Da die Sensorik lediglich auf dem verbauten Chip stattfindet, dient die restliche Platine als Zusatz, um die Kompatibilität mit anderen Mikrocontroller sicherzustellen.



Abbildung 52: Farbsensor TCS3200

Die Funktionsweise des Sensors wird anhand der folgenden Grafik erläutert:

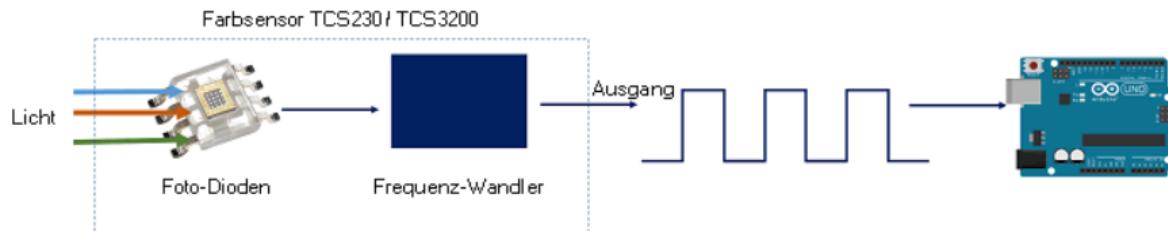


Abbildung 53: Funktionsweise Farbsensor (<https://elektro.turanis.de/html/prj029/index.html>)

Der TCS230/TCS3200 ist ein leistungsfähiger IC zur Farberkennung, der farbiges Licht mithilfe einer 8x8-Matrix von Fotodioden erfasst. Durch einen integrierten Strom-Frequenz-Wandler werden die detektierten Werte direkt in ein Rechtecksignal umgewandelt, das proportional zur erkannten Lichtwellenlänge bzw. Intensität steht. Der an den Ausgang des Sensors angeschlossene Mikrocontroller kann dieses Signal nun analysieren. Das verwendete Modul beinhaltet zusätzlich zum IC vier Leuchtdioden, die das zu erkennende Objekt gleichmäßig beleuchten. Bei genauerer Betrachtung des Sensor-Chips kann man sehen, dass die Foto-Dioden drei verschiedene Farbfilter besitzen. 16 davon haben je einen roten, grünen oder blauen Filter. 16 Foto-Dioden haben gar keinen Filter. Alle diese 16 gleichen Foto-Dioden sind parallel geschaltet und mit den Kontroll-PINs S2 und S3 verbunden, mit denen ausgewählt werden kann, welche Farbe erkannt werden soll. Mit diesen kann man in verschiedenen Kombinationen, verschiedene Farben erkennen. Dasselbe gilt für die zwei weiteren Kontroll-Pins S0 und S1, nur mit dem Unterschied, dass diese für die Skalierung der Ausgangsfrequenz genutzt werden. Diese Skalierungsfunktion wird für die Optimierung für bestimmte Frequenz-Zähler oder Mikrocontroller verwendet. Der Farbsensor wird zur Orientierung in seiner Umgebung benötigt. Der Roboter soll die Farbe unter sich erkennen und anhand einer entwickelten Software feststellen können, ob er sich auf einer einfarbigen Fläche befindet.

6.6.5 Infrarotsensor Sharp GP2Y0A21YK0F

Der Infrarotsensor Sharp GP2Y0A21YK0F ermöglicht eine präzise Messung von Distanzen im Bereich von 10 cm bis 80 cm. Beim Messverfahren sendet der Sensor ein Infrarotlicht aus, das von einer eingebauten Infrarot-LED erzeugt wird, und erfasst die Reflexion innerhalb des Bereichs. Die durch den Sensor erfassten Werte werden durch den Analog-Digital-Converter im Mikrocontroller in digitale Daten umgewandelt, um sie für weitere Verarbeitungsschritte zugänglich zu machen. Zudem wird das Bauteil mit einer Gleichspannung von 5V versorgt.



Abbildung 54: Infrarotsensor Sharp GP2Y0A21YK0F

Die genaue Funktion des Sensors kann aus dem folgenden Blockschaltbild aus dem Datenblatt entnommen werden:

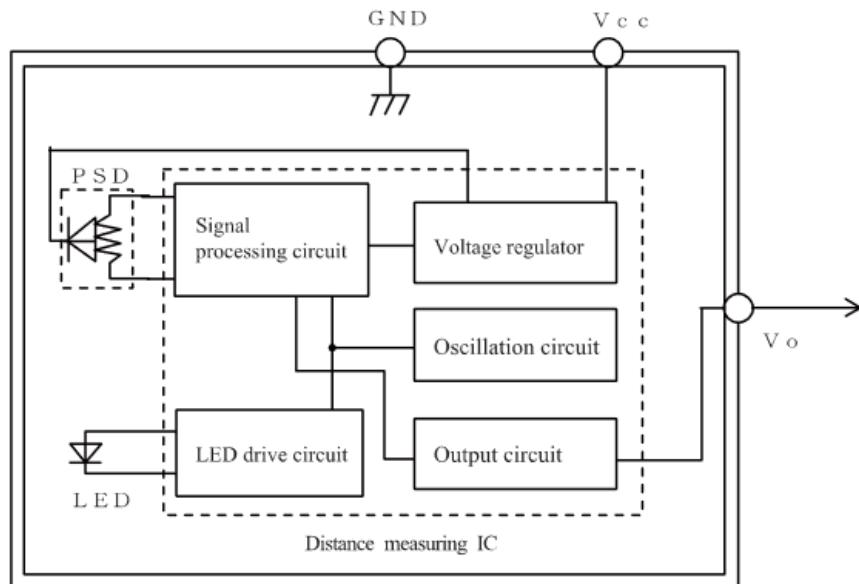


Abbildung 55: Infrarotsensor Blockschaltbild
(https://global.sharp/products/device/lineup/data/pdf/datasheet/gp2y0a21yk_e.pdf)

In dem Gerät ist ein Spannungsregler integriert, der die Signalverarbeitung und die LED-Ansteuerung mit Energie versorgt. Mittels des optoelektronischen Bauelements Infrarot-Positionsempfindlicher Detektor (PSD) erfolgt eine präzise Bestimmung der Position des reflektierten Infrarotstrahls. Da der PSD variable Spannungswerte erzeugt, wird eine spezielle Schaltung für die Signalverarbeitung verwendet, um diese Werte für einen Mikrocontroller bereitzustellen. Um ein synchrones Arbeiten zwischen dem PSD und der LED zu gewährleisten, werden beide von einem Oszillator innerhalb der Schaltung getaktet. Die ausgelesenen Informationen werden über einen Output-Circuit zum Pin VO übertragen.

Durch eine Software soll der Roboter in der Lage sein, die von den Sensoren erfassten Daten auszuwerten und diese anschließend zur Motorsteuerung zu nutzen. Dadurch ist es möglich, dass der Roboter autonom fahren kann, ohne dass eine manuelle Steuerung durch den Nutzer erforderlich ist.

6.6.6 Gyroskop MPU6050

Der MPU6050 – Sensor verfügt über ein 3-Achsen Accelerometer, ein 3-Achsen Gyroskop und einen Temperatursensor. Der MPU6050 besitzt intern einen eigenen DMP (Digital Motion Processor). Der DMP wurde ausgelegt für Sensorfusion und Bewegungserkennung. Der Prozessor kombiniert die Daten des Accelerometer und des Gyroskops, um Fehler innerhalb der einzelnen Sensoren zu minimieren. Er kann die Daten außerdem verrechnen und das Ergebnis in Euler-Winkel umrechnen. Der Algorithmus, auf welche Weise die Daten kombiniert werden, wird vom Hersteller Invensense nicht offen gelegt.

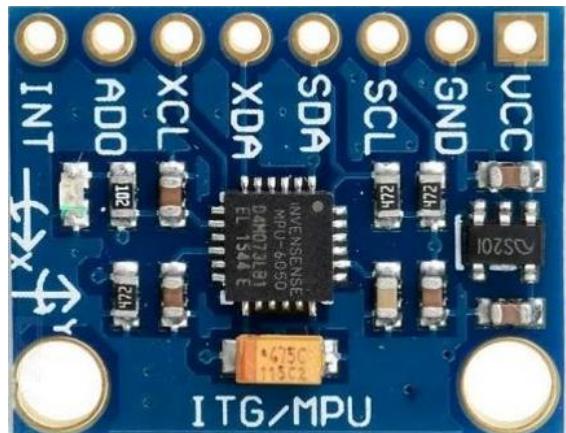


Abbildung 56 Gyroskop MPU6050

Die Daten werden über den I2C – Bus eingelesen. Im Sensor befinden sich MEMS (= „Micro-Electrical-Mechanical Systems“). Das sind winzige elektro-mechanische Systeme die billig und in Masse produziert werden können.

In der unteren Grafik drückt die Beschleunigung nach links die mittlere Kondensatorplatte näher zur rechten. Das verändert die Kapazität zwischen den Platten. Misst man die Kapazitäten kann man so auf die Beschleunigung rückschließen.

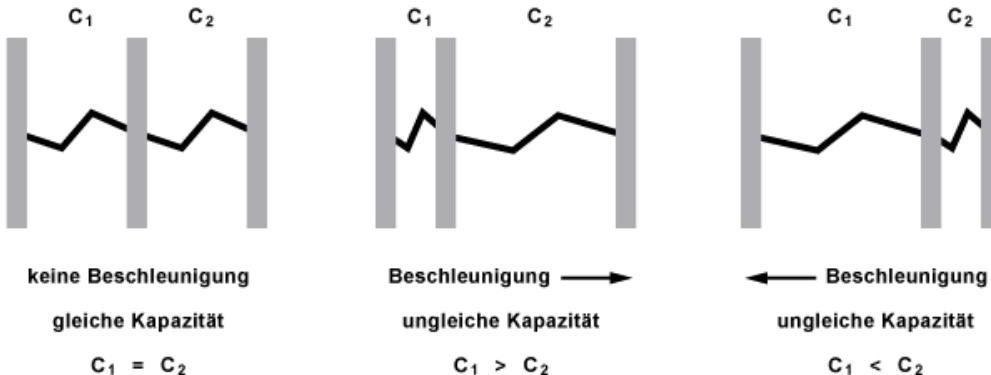


Abbildung 57: Funktionsweise des Accelerometers (<https://www.elektronik-kompendium.de/sites/bau/1503041.htm>)

Der Accelerometer misst Beschleunigungen entlang der x, y, z-Achsen, während das Gyroskop Geschwindigkeiten um die x, y, z – Achsen misst.

Diese Funktionen ermöglichen es dem Roboter, sowohl die veränderte Geschwindigkeit als auch die jeweilige Richtung zu erfassen und die Daten anschließend an den Raspberry Pi zu übertragen. Dort werden die Informationen in einem Graphen dargestellt und stetig aktualisiert.

6.7 Hardware-Design Controller

Die Steuerung des Roboters erfolgt über einen eigens dazu entwickelten Controller. Dieser soll die Funktion erfüllen, den Roboter in kontrollierten Richtungen zu lenken und diverse Software-Funktion auszuführen. Hierzu kann man den Aufbau aus dem folgenden Blockschaltbild entnehmen:

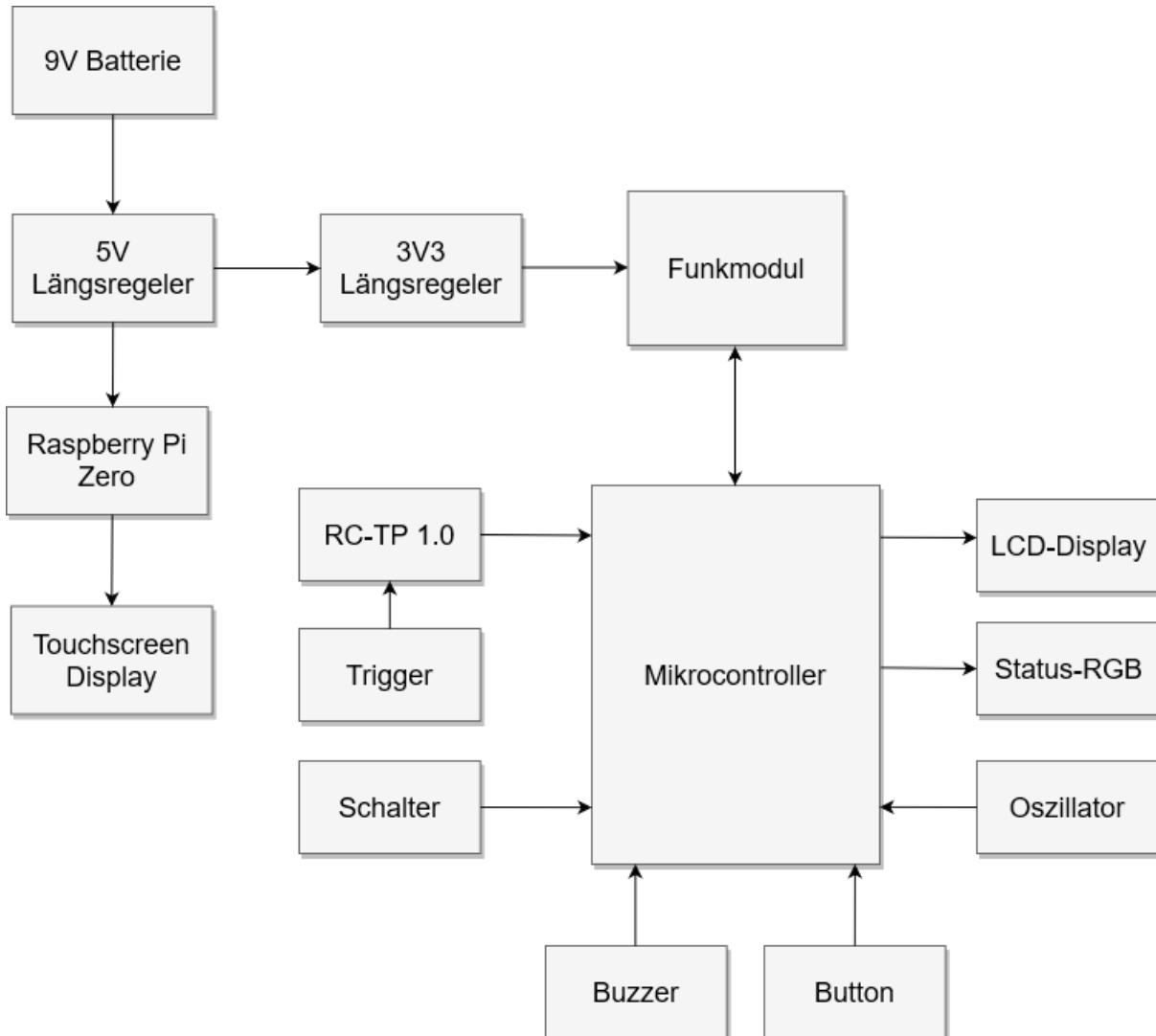


Abbildung 58: Controller Blockschaltbild

Es wird ein ATMEGA 328p als Mikrocontroller verwendet, da nur eine geringe Anzahl von Pins notwendig ist. Die Komponenten, wie der Oszillator, das Funkmodul NRF24L01+, die RGB-LED und der 3,3V-Längsregler MCP1117, wurden bereits in vorherigen Kapiteln erläutert und aus den gleichen Gründen eingebaut. Zur Stromversorgung der Schaltung wird eine 9V-Blockbatterie genutzt. Da der Mikrocontroller jedoch eine 5V Spannungsquelle benötigt, wird diese durch einen 7805-Längsregler von 9V auf die erforderlichen 5V umgewandelt. Für die Steuerung der Richtungen und Geschwindigkeit des Roboters werden zwei Akkuschrauber-Schalter als Trigger verwendet. Ergänzend dazu sind ein Buzzer und zwei Buttons eingebaut, die zusätzliche Funktionen ermöglichen. Eine detaillierte Erläuterung dieser Funktionen findet sich in der Softwarebeschreibung. Zusätzlich ist ein Raspberry Pi Zero 2W verbaut, der die Aufgabe hat, die Perspektive, die durch die Kamera des ESP32 im Roboter erfasst wird, auf

einem 3,5-Zoll-Display darzustellen. Eine detaillierte Erläuterung dazu wird ebenfalls im Kapitel zur Software-Entwicklung gegeben. Darüber hinaus wird auch ein LCD-Display eingebaut, das als Anzeige fungiert und eine bestimmte Zeit benötigt, um eine Leiste zu füllen. Sobald dies geschieht, kann der Joystick betätigt werden, und der Roboter beschleunigt für kurze Zeit.

6.8 Controller PCB-Design

Basierend auf dem Blockschaltbild in Abbildung 58 wurde in KiCAD ein Schaltplan sowie ein Platinenlayout entwickelt. Aufgrund der Verwendung von 1206-SMD-Komponenten als kleinste Bauteile wurde zunächst ein Prototyp in der Werkstatt erstellt, um die Schaltung zu testen. Anschließend wurden auftretende Schaltungsfehler behoben und die endgültige Platine bei dem externen Hersteller Aisler in Auftrag gegeben.

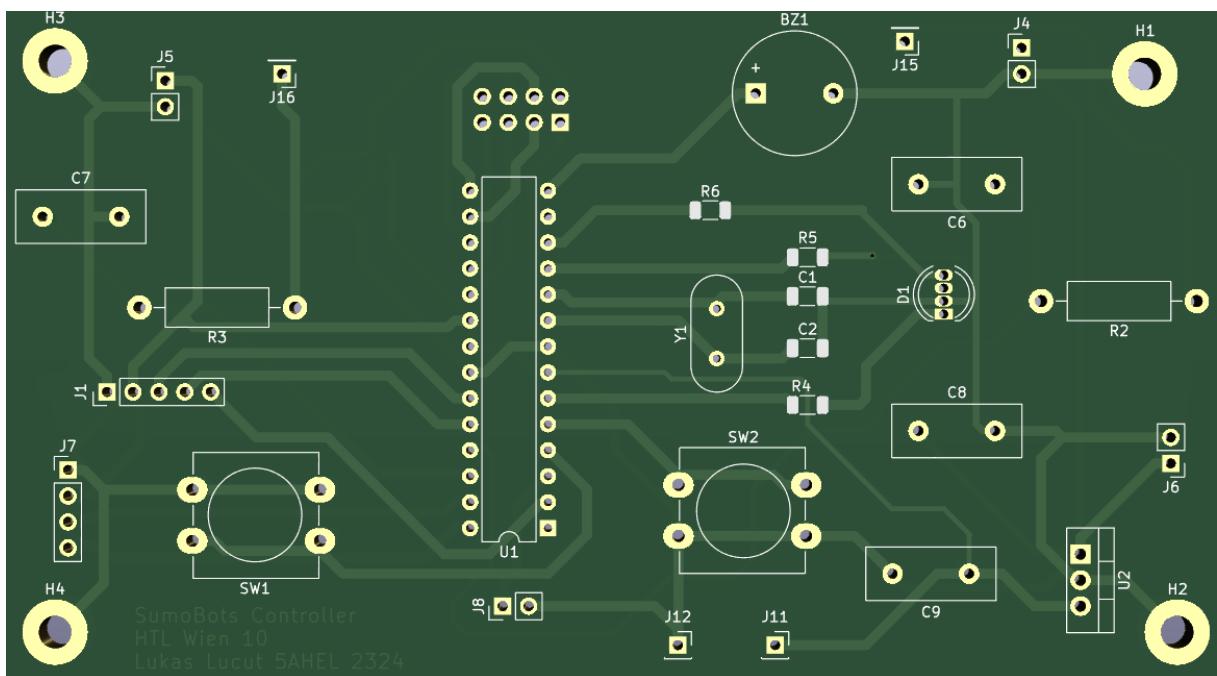


Abbildung 59: Controller PCB

Da die Platine zunächst innerhalb der Schule hergestellt wurde, waren wichtige Faktoren die Leiterbahnbreite von 1 mm, die beim Roboter nicht berücksichtigt wurde, sowie der zu beachtender Abstand, um Leiterbahnunterbrechungen beim Ätzen zu vermeiden. Aufgrund der Montageanforderungen am Gehäuse werden die Trigger im PCB als Pinheader realisiert. Das PWM-Signal des Gas-Triggers wird mit der Schaltung über J15 verbunden, während er über J4 mit einer Betriebsspannung von 5V versorgt wird. Dasselbe gilt für den Reserve-Trigger, der über J16 und J5 angeschlossen wird. Ebenso gilt dies für den Joystick, das LCD-Display, den Raspberry Pi Zero sowie für die Buttons. Die Bauteilbeschriftung ist direkt vom PCB abzulesen. Das Funkmodul NRF24L01+ hingegen wird direkt auf das PCB gelötet, ohne einen aussteckbaren Pinheader, um mögliche Verbindungsprobleme zu vermeiden.

6.9 Controller Erkenntnisse

Bei der Herstellung des Prototyps traten trotz der Einhaltung der Abstandsregeln mehrere Leiterbahnunterbrechungen und Kurzschlüsse zwischen den Leitungen auf. Des Weiteren schien beim Ätzen ein Fehler aufgetreten zu sein, da einige Lötpads am Mikrocontroller vor dem Lötvorgang abgelöst wurden und daher mit einer größeren Menge Lötzinn verbessert werden mussten. Nach einigen Korrekturen konnte die Schaltung jedoch in Betrieb genommen werden. Ein Schaltungsfehler, der sich bemerkbar machte, war, dass beide Ausgänge des Triggers dasselbe Signal ausgaben. Zunächst ging man davon aus, dass ein Anschluss eine Masseverbindung sein sollte, was sich jedoch als falsch herausstellte.

Nachdem die neue Platine von Aisler eingetroffen war, gestaltete sich die Inbetriebnahme als unkompliziert. Allerdings stellte sich bei der ersten Inbetriebnahme heraus, dass das verbaute Funkmodul defekt war, was zu keiner Verbindung führte. Nach dem Austausch konnte jedoch eine erfolgreiche Kommunikation zwischen den Geräten hergestellt werden. Die Funktion des Triggers mit dem dimensionierten Tiefpass wurde erfolgreich getestet und erwies sich als korrekt in Verbindung mit dem Motorshield des Roboters. Die Implementierung der restlichen Funktionen der einzelnen Komponenten wie des LCD-Screens verlief ebenfalls erfolgreich.

6.10 Hardware-Komponenten Controller

6.10.1 Trigger

Um die Geschwindigkeit des Roboters präzise zu steuern, werden anstelle von fehleranfälligen Drucksensoren elektrische Bohrschalter verwendet. Zur besseren Veranschaulichung der Funktionsweise dieses elektrischen Schalters wird das folgende Blockschaltbild verwendet:

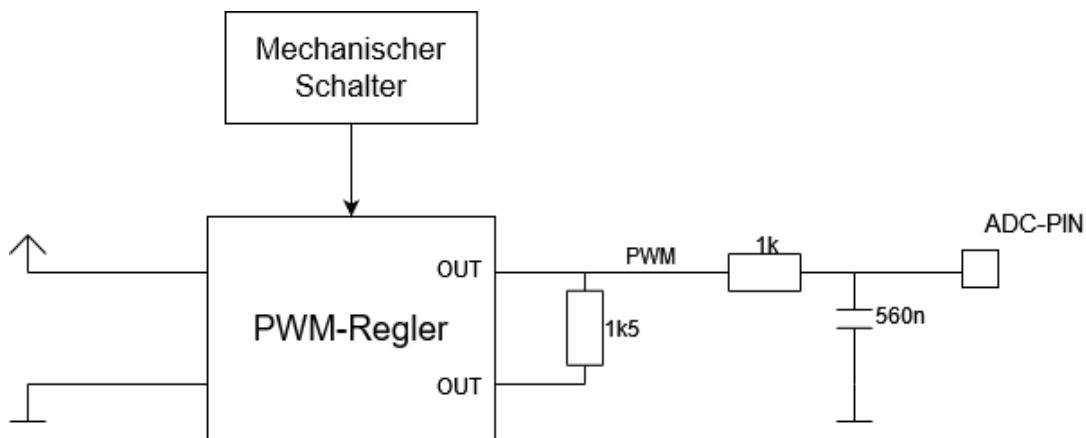


Abbildung 60: Trigger Blockschaltbild

In der Regel wird diese Art von Trigger in Geräten zur direkten Motoransteuerung verwendet, wie beispielsweise Bohrmaschinen. Um sie nun in einem Controller zur Fernsteuerung zu implementieren, muss zuvor die Funktionsweise verständlich sein. Die eigentliche Ansteuerung erfolgt mittels eines PWM-Signals, dessen Pulsweite durch den mechanischen Schalter einstellbar ist. Da hier jedoch kein Motor direkt gesteuert wird, wird ein Widerstand als Last geschaltet, um ein pulsweitenmoduliertes Signal messen zu können. Um das Signal für eine Steuerung mittels der Fernsteuerung zu nutzen, wird ein RC-Tiefpass benötigt. Dieser

filtert aus dem modulierten Rechtecksignal eine Ladekurve heraus, die anschließend mit dem ADC eines Mikrocontrollers erfasst werden kann. Die aufgenommenen Daten werden dann zum Roboter übertragen, um die Motoren anzusteuern.

6.10.2 LCD-Display

Im Controller wird ein 16x2 LCD-Display benötigt, um eine Ladeanzeige darzustellen, welche für eine Beschleunigungsfunktion genutzt wird. Die Hauptfunktion des 16x2 LCD-Displays besteht darin, Informationen in Textform anzuzeigen. Es besteht aus einer Matrix von 16 Spalten und 2 Zeilen von Flüssigkristallzellen, die elektrisch gesteuert werden können. Jede Zelle repräsentiert ein einzelnes Zeichen, das durch die Anwendung von Spannung verändert werden kann, um die Lichtdurchlässigkeit zu beeinflussen. Dieser wird wiederrum über I²C-Bus angesteuert, wodurch lediglich 4-Pins zur Ansteuerung notwendig sind.

6.10.3 5V-Längsregler 7805

Die Funktionsweise eines Längsreglers wurde bereits im Kapitel 6.3.1 Spannungsversorgung des Funkmoduls erklärt. Da im Controller eine 9V-Blockbatterie genutzt wird und der Mikrocontroller ATMEGA328p mit einer Spannung von 5V in Betrieb genommen werden muss, ist ein weiterer Längsregler notwendig.

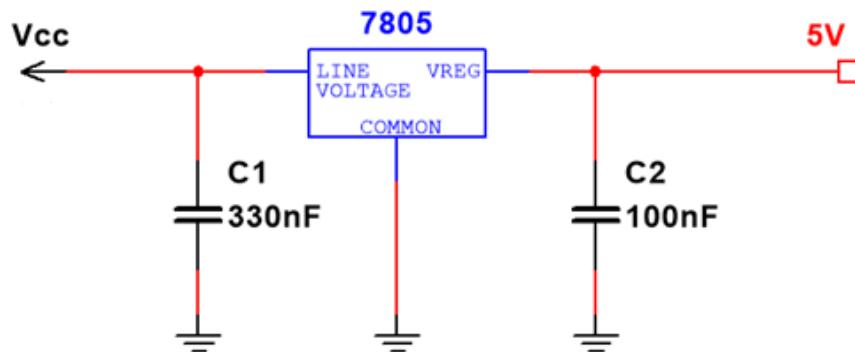


Abbildung 61: 5V-Längsreglerschaltung

Die Kondensatoren werden benötigt, um auftretende Reflektionen zu verhindern und die Ausgangsspannung 5V zu glätten. Falls die 5V-Leitung im PCB-Design als eine längere Leitung realisiert wurde, kann diese induktive Eigenschalten vorweisen und es könnte zu einer Beschädigung des Mikrocontrollers oder des Längsreglers kommen. Daher werden die beiden Kondensatoren C₁ und C₂ verbaut, dessen Werte aus dem Datenblatt entnommen wurden.

6.10.4 Joystick

Die Fahrtrichtung des Roboters wird mittels eines 2-Achsen-Joysticks bestimmt, der sich in allen Richtungen bewegen lässt. Dieser besitzt jeweils ein Potentiometer für die X- und Y-Achse, welche als veränderbare Spannungsteiler fungieren. Um die Daten des Spannungsteilers auslesen zu können, müssen die Pins V_{Rx} und V_{Ry} an den ADC-Pins des Mikrocontrollers verbunden sein. Dafür muss der Joystick zunächst mit 5V versorgt werden, da ansonsten unerwünschte Fehler beim Konvertieren auftreten könnten. Des Weiteren bietet der Joystick einen eingebauten Taster an, der eine logische „0“ oder „1“ ausgibt, daher wird der Pin SW an einem digitalen Pin angeschlossen.

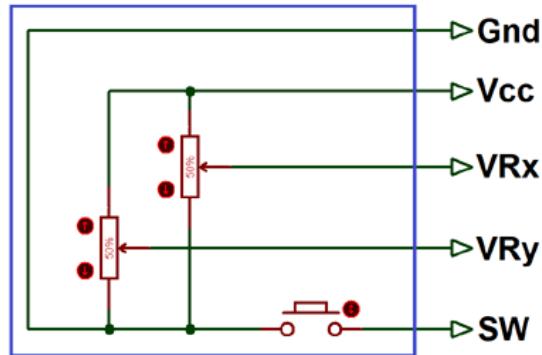


Abbildung 6.2 Joystick Schaltplan

6.10.5 Raspberry Pi Zero 2W und Touchscreen-Display

Die Videoübertragung der ESP32-Kamera im Roboter ermöglicht es, die Sicht des Roboters auf einem Webserver zu streamen. Um diese Übertragung auf einem kleineren Display im Controller anzuzeigen, wurde ein 3,5-Zoll-Touchscreen ausgewählt, der diese Anforderungen erfüllt. Allerdings ist dieser nicht mit dem Atmega328p kompatibel, und der Atmega328p verfügt nicht über die notwendige Hardware für eine Internetverbindung zum Zugriff auf den Webserver. Aus diesem Grund wird ein Raspberry Pi Zero 2W in den Controller integriert, um diese Anforderungen zu erfüllen.

Der Raspberry Pi Zero wird lediglich mit derselben Versorgungsspannung $V_{cc} = 5V$ betrieben. Daher kommuniziert er nicht mit der eigentlichen Controllerplatine, wodurch auch keine weiteren Pins zur Verbindung benötigt werden. Der Datenaustausch zwischen dem Raspberry Pi Zero und dem Touchscreen erfolgt über den SPI-Bus. Im Kapitel Software werden die spezifischen Konfigurationen des Pi näher erläutert.

7 Softwareentwicklung (ZIC)

In diesem Kapitel soll der Prozess der Softwareentwicklung sowie die Software an sich erläutert werden.

Alle Codes, die in diesem Kapitel verwendet werden, sind „Pseudocodes“. Diese sind nicht in einer Programmiersprache verfasst und sollen nur die Funktionsweise des tatsächlichen Codes erklären. Sie werden im Stil der ursprünglichen Programmiersprache geschrieben. Dabei können z.B. Funktionsaufrufe, Variablen Deklarationen, Typecasting usw., ausgelassen werden. Teilweise wird auf Weggelassenes in Fußnoten eingegangen und im Anhang ist eine gut kommentierte Version der tatsächlich verwendeten Codes zu finden.

Manche Unterfunktionen sind in ihrer Funktionsweise einfach genug, um sie beim einmaligen Lesen des Sourcecodes verstehen zu können. In folgenden Kapiteln wird dann nur erklärt, was die Funktion bewirkt anstatt sie – wie sonst – Zeile für Zeile zu behandeln. Der interessierte Leser kann aber jederzeit den Quellcode im Anhang zur Hand nehmen.

Die Software wird, mit anderen Teilen der Diplomarbeit wie z.B. dieses Dokument, auf GitHub gehostet, wie in Kapitel 8.1 beschrieben.

Grundsätzlich beinhaltet das Projekt verschiedene Teilnehmer die alle mit RF24 – Funkmodulen miteinander kommunizieren:

- 1) Bis zu 3 Controller, welche Eingabeelemente einlesen und an den Roboter weiterschicken soll.
- 2) Bis zu 3 Roboter, welche Daten von jeweils einem Controller entgegennehmen, Motoren ansteuern und Sensordaten an den Server weiterschicken, und
- 3) Der Server, welche Sensordaten der Roboter entgegennehmen, und diese auf einer Website darstellen soll.

Zunächst, wird erklärt, welchen Styleguide die Codes befolgen. Dann, wie sie miteinander kommunizieren und wie das Funkmodul angesteuert wird. Erst danach können die einzelnen Komponenten im Detail behandelt werden. Zum Schluss wird dann von einigen Problemen berichtet, die während der Entwicklung aufgetreten sind.

7.1 Styleguide

Alle Variablen und Funktionen sind Englisch¹ und im „Camel Case“ benannt. Statt einem Abstand wird also das nächste Wort großgeschrieben (z.B. `configureRadio`). Konstanten beginnen mit einem Großbuchstaben, C-Makros werden in Caps Lock geschrieben. Makros können daher keinen Camel Case befolgen und verwenden stattdessen Unterstriche.

Alle Pin-Nummern beginnen mit `p_`. Einige Wörter werden so oft verwendet, dass sie mit einzelnen Buchstaben abgekürzt werden dürfen:

`l ... left,` `r ... right`

¹ Die einzigen Ausnahmen sind die `_soll` und `_ist` Suffixe.

b ... backwards, f ... forwards

Die Reihenfolge ist Left / Right dann Backwards / Forwards. Der Pin, mit dem der Rechte Motor nach vorne angesteuert werden kann, heißt somit:

| p_rf

In den Pseudo-Codes der Diplomarbeit sind diese Wörter der Klarheit halber trotzdem oft ausgeschrieben.

7.2 Netzwerk

Es gibt drei verschiedene Komponente, die über das im Kapitel 6.3 beschriebene Funkmodell miteinander kommunizieren. Außerdem sollen bis zu drei verschiedene Roboter miteinander kommunizieren können. Das daraus resultierende Netzwerk kann durch folgendes Diagramm veranschaulicht werden:

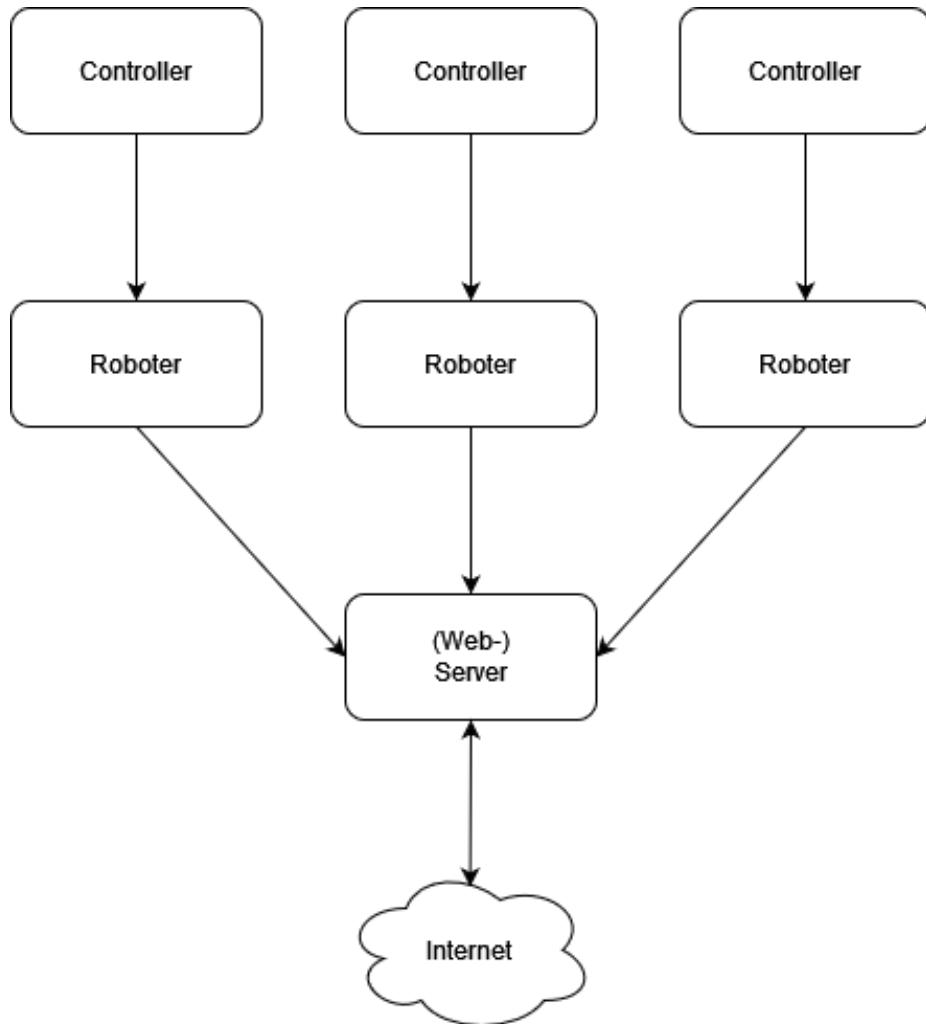


Abbildung 63: Blockschaltbild der Netzwerkarchitektur

Jeder Controller schickt seinem Roboter alle 10ms ein Paket von den Zuständen der verschiedenen Eingabeelemente als Array von Integer-Zahlen:

```
int data[4] = [lSchulter, rSchulter, joyX, joyY]
lSchulter 0 (losgelassen) ... 255 (vollstaendig gedrueckt)
rSchulter 0 (losgelassen) ... 255 (vollstaendig gedrueckt)
joyX     -255 (ganz links) ... 255 (ganz rechts)
joyY     -255 (ganz unten) ... 255 (ganz oben)
```

Der Roboter muss Daten verschiedener Art verschicken, weshalb folgendes struct verwendet wird:

```
struct DataPayload {
    uint8_t zero;

    float gyroX;
    float gyroY;
    float gyroZ;

    float accX;
    float accY;
    float accZ;

    float temp;

    uint8_t controllerConnected;
    uint8_t vibration;
};
```

Auf die Bedeutung der einzelnen Elemente dieses structs wird in Kapitel 7.4.2 genauer eingegangen. Der Roboter gibt auch seinen Zustand mit Log-Nachrichten bekannt. Diese sind ein 32 -Byte großer String, da jedes Paket des RF24 Moduls 32 Byte groß ist:

```
char msg[32];
```

Damit der Server zwischen Lognachrichten und Daten unterscheiden kann, beginnt ein Datenpaket mit einem Null-byte (uint8_t zero), welches niemals das erste Element eines Strings sein kann.

7.2.1 Funkmodul

Software-seitig verwenden wir für die Ansteuerung des Funkmoduls die dazugehörige Library “nRF24L01.h”, welche den OSI-Layer 2 für das Funkmodul implementiert. Einige der anfänglichen Befehle zur Initialisierung des Moduls bleiben bei allen Komponenten gleich.

```
RF24 radio(p_CE, p_CS);
radio.begin();
```

Das Funkmodul unterscheidet zwischen logischen Kanälen (pipes). Von diesen kann er insgesamt eine zum Schreiben („Writing-Pipe“) und 6 zum Senden („Reading Pipe“) geöffnet haben. Gleichzeitig lesen und schreiben kann er immer noch nicht; davor muss jeweils „radio.stopListening()“ oder „radio.startListening()“ aufgerufen werden.

Der Roboter Code hat damit zum Beispiel grob folgende Form:

```
setup() {  
    openReadingPipe(controllerAddress)  
    openWritingPipe(serverAddress)  
    startListening()  
}  
  
loop() {  
    read(inputData)  
  
    if ( 500ms passed ) {  
        stopListening()  
  
        write(sensorData)  
        startListening()  
    }  
}
```

Konfiguriert wird das Funkmodul in allen drei Teilnehmern mit der `configureRadio()` Funktion.

```
void configureRadio() {  
    // Do nothing while radio module not connected  
    while ( !radio.begin() ) {}  
  
    openReadingPipe(controllerAddress)  
    openWritingPipe(serverAddress)  
    setPALevel(RF24_PA_MIN)  
    setChannel(controllerChannel)  
    setRetries(4, 10)  
}
```

Hier werden zunächst die verschiedenen pipes geöffnet. Danach wird konfiguriert, mit welcher Leistung das Radiomodul senden soll. Je nachdem, ob der Code auf einem Batterie- oder Netzbetriebenem Gerät läuft, werden hier höhere oder niedrigere Leistungen gewählt. `radio.write()` erwartet sich nach dem Senden eine Bestätigung vom Empfänger, dass das Packet angekommen ist. Bekommt er das nicht, sendet er die Nachricht erneut. Mit `setRetries()` kann konfiguriert werden, wie oft und in welchem Abstand die Nachricht erneut gesendet werden soll. Mit der Standard-Konfiguration kommt man, wenn kein Packet ankommt, auf einen maximalen Delay von 60ms, was für diese Zwecke zu hoch ist. Deswegen wird mit der Anzahl der Neuversuche hier so weit hinuntergeschraubt, dass der maximale Delay nun 10ms beträgt.

Bricht die Verbindung zwischen µC und Funkmodul ab, setzt die RF24 Library die Variable `radio.failureDetected`. In diesem Fall wird versucht, das Modul neuzukonfigurieren:

```
if (radio.failureDetected) {  
    drive(0, 0);  
  
    configureRadio();  
    logMsg("Radio reconfigured after failure", 32);  
}
```

Die Funktion `logMsg` tritt mit dem Server in Verbindung und wird in Kapitel 7.4.3 vorgestellt.

7.3 Controller

In diesem Kapitel wird der Code des Controllers erklärt. Dieser soll die elektronische Komponente auslesen, die Daten kurz verarbeiten und anpassen, um diese dann an den Roboter weiterzusenden.

Das Projekt besteht aus 3 verschiedenen Controllern, die alle auf einem anderen Kanal senden und dessen Pipe anders benannt ist. Diese Konstanten werden in der Headerdatei mithilfe des Precompilers gesetzt.

```
-- controller.h --  
  
#ifdef controller1  
    robotAddress = "c-r01";  
    robotChannel = 25  
#endif  
  
#ifdef controller2  
    robotAddress = "c-r02";  
    robotChannel = 75  
#endif  
...
```

Vor der Inkludierung des Headers muss dann, je nach dem für welchen Controller das Programm kompiliert werden soll, das jeweilige Makro definiert werden.

```
-- controller.ino --  
  
#define controller1  
#include "controller.h"
```

So können alle Konstanten an nur einer Stelle im Code gesetzt werden. Die Pipes werden hier `c-r0X` genannt, weil dieser eine Verbindung von Controller zu Roboter benennt, und nicht den Empfänger, also den Roboter selbst. So muss die Pipe von z.B. Roboter 1 genauso wie die des Controllers `c-r01` heißen.

Der Controller muss die verschiedenen elektrischen Komponenten, also den Joystick und die Schultertasten auslesen und an den Roboter weiterschicken. Unabhängig von der Hardware wurden die Wertebereiche in Kapitel 7.2 definiert. Vor dem Senden müssen die Werte noch auf den entsprechenden Wertebereich angepasst werden.

```
loop() {
    data[0] = analogRead(p_joyX)/2 - 255
    data[1] = analogRead(p_joyY)/2 - 255
    data[2] = 255 - analogRead(p_lShoulder)/4
    data[3] = 255 - analogRead(p_rShoulder)/4
    write(data)
}
```

Zusätzlich dazu soll ein von dem Videospiel „Rocket League“ inspirierter „Boost“ implementiert werden, der nach einer gewissen Zeit aktiviert werden kann, um den Roboter kurzfristig schneller fahren zu lassen. Wird der Boost aktiviert, wird die Variable „millisBoost“ auf den jetzigen Zeitpunkt gesetzt werden. Die Differenz zwischen dem jetzigen Zeitpunkt und dem Zeitpunkt, wo der Boost aktiviert wurde, sagt daher aus, ob dieser noch aktiviert ist:

```
if (crntMillis - millisBoost <= BOOST_DURATION) {
    data[2] = 255 - (analogRead(p_lSchulter) >> 2);
    data[3] = 255 - (analogRead(p_rSchulter) >> 2);
}

else {
    // Subtract 80, but keep it above 0
    data[2] = max( (175-(analogRead(p_lSchulter) >> 2)), 0);
    data[3] = max( (175-(analogRead(p_rSchulter) >> 2)), 0);
}
```

Ist das der Fall, wird der tatsächliche Wert gesendet, ansonsten wird er mit 80 subtrahiert. Ein LCD-Display zeigt an, wie lange auf den Boost gewartet werden musst. Dieses stellt einen Ladebalken an, der sich langsam (jede Sekunde um ein Element) auffüllt. Ist er voll gibt der Buzzer einen Ton aus und der Boost kann mit drücken des Joysticks aktiviert werden. Zusätzlich wird, um den Zustand erkennen zu können, eine RGB – LED angesteuert. Was diese Komponente genau tun, kann im Quellcode und in der Bedienungsanleitung nachvollzieht werden.

7.4 Roboter

In diesem Kapitel wird der Code des Roboters erklärt. Dieser soll Daten des Controllers empfangen, und damit die Motoren ansteuern. Außerdem sollen Daten verschiedener Sensoren an den Server geschickt werden. Auch hier gibt es eine LED, zu der in der Bedienungsanleitung mehr Informationen zu finden ist.

Hier wird dieselbe Technik wie beim Controller, eingesetzt, um die Konstanten für die verschiedenen Roboter verschieden zu setzen.

```
-- robot.h --  
  
#ifdef robot1  
    controllerAddress = "c-r01"  
    serverAddress = "1-r-s"  
#endif  
  
#ifdef robot2  
    controllerAddress = "c-r002"  
    serverAddress = "2-r-s"  
#endif  
...
```

Da die 3 Reading Pipes im Server sich laut der Dokumentation der RF24 Library nur um das erste Byte unterscheiden dürfen, wird der Index der Pipe in `serverAddress` an die erste Stelle verschoben.

7.4.1 Motorsteuerung

Als Input – Größen bekommt der Roboter die Position des Joysticks und der zwei Schultertasten. Die Ansteuerung der Motoren wird anhand dieser berechnet. Die Schultertaste bestimmt die Geschwindigkeit der Motoren, das Verhältnis zwischen den Motoren bleibt aber unverändert. Je nachdem ob die rechte oder linke Schultertaste gedrückt wird, fährt der Roboter nach vorne oder hinten. Es wird der Wert der stärker gedrückten Schultertasten verwendet. Durch die Wrapper – Funktion, auf die später noch eingegangen wird, laufen Motoren durch negative Werte rückwärts. Die linke Schultertaste gibt also dieselbe Magnitude, aber ein negatives Vorzeichen. Damit kommen wir auf folgenden Code:

```
if (rSchulter > lSchulter)  
    speed = rSchulter  
else  
    speed = -lSchulter  
  
lMotor = lMotorTurn * speed  
rMotor = rMotorTurn * speed
```

`lMotorTurn` und `rMotorTurn` geben das Verhältnis zwischen linkem und rechtem Motor an. Sie sind also Variablen, die von -1 bis 1 gehen und sich aus der x-Achse des Joycons berechnen lassen.

Nimmt man an, das „`joyX`“ eine floating-point-Zahl ist, die von -1 (ganz links) bis +1 (ganz rechts) lauft², kommen wir auf folgendes Diagramm:

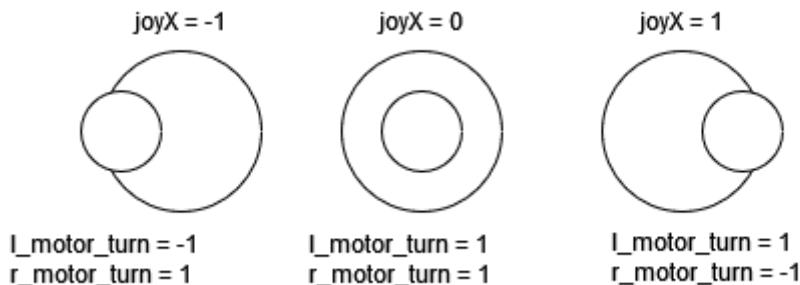


Abbildung 64: Diagramm für Veranschaulichung der Berechnung von `motorTurn` Werten

Ist der Stick ganz links positioniert, soll der Roboter sich im Stand nach links drehen. Deshalb sollen sich die linke Räder Rückwärts und die rechten vorwärts bewegen. Die Anderen Positionen können mit einer ähnlichen Argumentation verstanden werden. Zwischen diesen Extrempositionen sollen die Variablen kontinuierlich und linear alle Zwischenwerte einnehmen. Mittels `map` – Befehl³ werden diese Wertebereiche aneinander angepasst.

```

if (joyX < 0) {
    lMotorTurn = map(joyX, -1, 0, -1, 1)
    rMotorTurn = 1
}
else if (joyX >= 0) {
    lMotorTurn = 1
    rMotorTurn = map(joyX, 0, 1, 1, -1)
}
lMotor = lMotorTurn * speed
rMotor = rMotorTurn * speed

```

Steuert dieser Code die Motoren direkt an und falls der Fahrer nicht vorsichtig ist und schnelle Richtungsänderungen vornimmt, kann das die Motorbrücke überlasten. Deshalb sollen diese Änderungen nur langsam übernommen werden. Der Controller ändert nur die „_soll Werte“, tatsächlich an das Motorshield weitergegeben werden die „_ist Werte“. Sind ein Soll-Wert und Ist-Wert nicht gleich, wird der Ist-Wert nur einen Schritt Richtung Soll-Wert verändert.

² Tatsächlich ist diese nach Kapitel 7.2 eine Integer-Zahl zwischen -255 und 255. Diese wird zuerst auf einen Float gecasted und dann durch 256 dividiert, um auf den Wertebereich [-1; 1] zu kommen.

³ Der Map – Befehl von Arduino verwendet nur Integer Mathematik. Deshalb ist im Code ein eigener floating-point-map Befehl implementiert. IMotor und rMotor werden dann wieder zurück auf einen Integer – Wert gecasted, um als PWM – Signal an das Motorshield weitergeleiten werden zu können.

```
lMotor_soll = lMotorTurn * speed

if (lMotor_ist > lMotor_soll)
    lMotor_ist -= 1

else if (lMotor_ist < lMotor_soll)
    lMotor_ist += 1
```

Dieser Code gilt genauso für den rechten Motor. Diese Werte müssen nur noch an das Motorshield weitergegeben werden. Wie diese funktioniert wurde im Kapitel 6.4.1 bereits genauer beschrieben, wird hier aber noch einmal zusammengefasst. Er hat für beide Motoren zwei Pins die ein PWM – Signal akzeptieren; eins, falls sich der Motor nach vorne drehen soll, eines, falls sich der Motor rückwärts drehen soll. Diese Ansteuerung wurde, wie oben bereits erwähnt, durch eine Wrapper – Funktion „versteckt“ werden. Diese akzeptiert für rechten und linken Motor Werte von -255 bis 255. Negative Werte stehen hier für einen sich rückwärtsdrehenden Motor. Wie man im vorherigen Kapitel erkennen konnte, macht das die Mathematik ein wenig bequemer.

In der Funktion muss dann nur abgeprüft werden, ob der Wert negativ oder positiv ist, und je nachdem auf den Vorwärts- oder Rückwärtspin der absolute Wert⁴ des Arguments als PWM – Signal angelegt werden.

Wie im Kapitel 5 zu sehen ist der linke Motor des Roboters verkehrt herum eingebaut. Das wird kompensiert, indem wir den linken Wert negieren und somit seine Fahrtrichtung umkehren.

```
drive(left, right) {
    left = -left
    if (left > 0) {
        analogWrite(p_leftForward, |left|)
        analogWrite(p_leftBackward, 0)
    }
    else {
        analogWrite(p_leftForward, 0)
        analogWrite(p_leftBackward, |left|)
    }
}
```

Auch dieser Code gilt, abseits der Negation des Wertes, genauso für den rechten Motor.

⁴ Im Code wurde der Einfachheit halber, der positive Wert gelassen und der Negative negiert.

7.4.2 Sensoren

Der Roboter hat folgende Sensoren:

- Gyroskop entlang 3 Achsen
- Accelerometer entlang 3 Achsen
- Temperatursensor
- Vibrationssensor
- 3 Abstandssensoren

Der Vibrationssensor sagt in einem Moment nur über einen Pin aus, ob er gerade Erschütterungen misst oder nicht. Er kann über ein einfaches `digitalRead()` ausgelesen werden. Gyroskop, Accelerometer und Temperatursensor sind alle Teil des MPU6050 Sensors und können mit dessen Library in einem Funktionsaufruf abgefragt werden:

```
| mpu.getEvent(&a, &g, &temp);
```

a, g und temp sind `sensors_event_t` structs und beinhalten nach Funktionsaufruf die verschiedenen Daten. Diese Daten wurden von der Library in gewöhnliche Einheiten umgewandelt. Außerdem können Sensitivität des Sensors sowie die Grenzfrequenz eines eingebauten digitalen Filters eingestellt werden.

```
| mpu.setAccelerometerRange(MPU6050_RANGE_8_G);  
| mpu.setGyroRange(MPU6050_RANGE_500_DEG);  
| mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);
```

Der digitale Sensor „mittelt“ automatisch über mehrere Messdaten. Die Grenzfrequenz bestimmt also einerseits, wie schnell er reagieren kann aber andererseits, wie stabil seine Werte sind. Alle Daten werden in der Funktion `sendSensorData()` gleichzeitig gesendet.

```
int sendSensorData() {  
    stopListening()  
  
    sensors_event_t a, g, temp  
    mpu.getEvent(&a, &g, &temp)  
  
    struct DataPayload payload = {  
        0x00,  
        g.gyro.x, g.gyro.y, g.gyro.z,  
        a.acceleration.x, a.acceleration.y, a.acceleration.z,  
        temp.temperature,  
        controllerConnected,  
        digitalRead(p_vibration)  
    }  
    int response = radio.write(&payload, sizeof(payload))  
  
    startListening()  
    return response  
}
```

Diese Funktion wird alle 100ms aufgerufen. Wäre der Server nicht verbunden, und wäre die Standard-konfiguration von `setRetries` in Kapitel 7.2.1 nicht geändert worden, hätte `radio.write` immer 60ms auf eine Antwort des Server gewartet. Somit würde der Roboter mehr als die Hälfte seiner Zeit in dieser Funktion verbringen!

Am Ende wird auch mit `controllerConnected` angegeben, ob der Controller mit dem Roboter verbunden ist. Dafür wird überprüft, wie viel Zeit seit dem Empfangen des letzten Pakets vergangen ist:

```
if (packet_received) {  
    timeLastPacket = crntTime  
    controllerConnected = true  
}  
  
else if ( crntTime - timeLastPacket > 100)  
    controllerConnected = false
```

Controller und Server überprüfen ihre Verbindungen auf eine ähnliche Art und Weise.

7.4.3 Logging

Sobald Roboter und Server verbunden ist der Roboter nicht mehr auf LEDs angewiesen, um den internen Zustand zu erläutern. Er verwendet dafür log-Nachrichten.

```
int logMsg(char *x, int len) {  
    int resp;  
    radio.stopListening();  
    resp = radio.write( &msg, len );  
  
    radio.startListening();  
    return resp;  
}
```

Der Roboter protokolliert, ob das Initialisieren jedes einzelnen Moduls funktioniert hat. Er gibt auch aus, wenn das Radio Modul neu konfiguriert werden musste.

7.5 Server

Der Server hat zwei Rollen zu erfüllen: Einerseits soll er Daten mit dem Funkmodul empfangen und speichern, andererseits soll er diese auf einem Webserver anzeigen. Beide Teile werden in einem Python-Script „server.py“ implementiert. Für die Website wurde das Python-Framework flask verwendet, für das Funkmodul die Library „pyrf24“ welche nur ein Wrapper der bisher genutzten C-Library ist. Damit beide Teile parallel laufen können, sich aber immer noch Speicher teilen, werden verschiedene Threads verwendet.

```
### Shared Memory ###
gyroX=[deque(maxlen=100),deque(maxlen=100),deque(maxlen=100)]
gyroY=[deque(maxlen=100),deque(maxlen=100),deque(maxlen=100)]
...
accX = [deque(maxlen=100),deque(maxlen=100),deque(maxlen=100)]
...
vibration=[deque(maxlen=10),deque(maxlen=10),deque(maxlen=10)]
temp = [deque(maxlen=10),deque(maxlen=10),deque(maxlen=10)]
log = [deque(maxlen=10),deque(maxlen=10),deque(maxlen=10)]
```

Hier wird „deque“ statt dem Standard Python-Array verwendet. Einerseits sind diese ein wenig schneller, vor allem aber wird das älteste Element gelöscht, wenn man ein neues hinzufügt und die Liste voll ist. Das heißt es verhält sich, ohne jeglichem Mehraufwand, wie ein Ringpuffer. Da nur die Geschichte der letzten paar Sensorwerte gespeichert werden soll, ist das genau das gewünschte Verhalten.

Da die Werte von drei verschiedenen Robotern gespeichert werden muss, ist jede Variable ein Array von drei „deques“. Es bildet somit eine Art zweidimensionales Array.

7.5.1 Empfangen der Sensordaten

Die Konfiguration des Funkmoduls ist nicht anders als beim Roboter. Es müssen nur mehrere Pipes gleichzeitig geöffnet werden.

```
radio.openReadingPipe(0, bytearray( b"1-r-s\x00" ))
radio.openReadingPipe(1, bytearray( b"2-r-s\x00" ))
radio.openReadingPipe(2, bytearray( b"3-r-s\x00" ))
```

Da die Pipes hier genauso wie in `roboter.c` heißen müssen, wird am Ende ein Null-Byte angehängt um die Implementation von Strings in C nachzuahmen.

```
hasPayload, index = radio.available_pipe()
if ( hasPayload ):
    timeLastPacket[index] = timeNow
    robotConnected[index] = True

    rec = radio.read()

    process(index, rec)
```

`timeLastPacket` wird verwendet, um den Status der Verbindung zum Roboter zu ermitteln, ähnlich der Implementation im Roboter selber, wie in Kapitel 7.4.2 erklärt.

`radio.available_pipe` gibt nun nicht mehr nur aus, ob ein Paket eingelesen werden kann, sondern auch, von welcher Pipe es kommt. So kann unterschieden werden, von welchem Roboter ein Paket gekommen ist.

```
def process(i, x):
    if ( x[0] == 0x00 ): #Data
        data = struct.unpack("< B 7f 2B B", x)
        gyroX[i].append(data[1])
        gyroY[i].append(data[2])
        gyroZ[i].append(data[3])

        accX[i].append(data[4])
        accY[i].append(data[5])
        accZ[i].append(data[6])

        temp[i].append(data[7])
        controllerConnected[i] = bool( data[8] )
        vibration[i] = data[9]

    else:
        data = struct.unpack("<32s", x)
        log[i].append( data[0].decode("ASCII") )
```

Vom ersten Byte kann unterschieden werden, ob es sich um eine Log-Nachricht oder ein Datenpayload handelt. Mithilfe der struct library kann das struct „ausgepackt“ werden. Dazu wird mithilfe eines Strings angegeben, um was für Daten es sich handelt und ob sie big- oder little-endian sind. Danach können die verschiedenen Daten in die dazugehörigen Arrays sortiert werden.

Server und Radiomodul sind in einem einzigen Python-Script implementiert. Damit das Radiomodul dem Webserver nicht in die Quere kommt, wird es in einem Thread gestartet:

```
def serviceRadio():
    initRadio()
    while true:
        if ( received_packet ):
            process( packet )

    threading.Thread(target=serviceRadio, args=() ).start()
```

7.5.2 Website

Nun kann ein Webserver diese Daten bekommen und anzeigen. Die Website wurde mit dem gewöhnlichen CSS, HTML, JS Stack programmiert. Im HTML-Header werden zunächst einige Metadaten definiert.

```
<head>
  <title> SumoBots </title>
  <script src= "https://cdnjs.cloudflare.com/ [...] /Chart.js">
  </script>

  <link rel="stylesheet" href="/static/style.css">
  <link rel="icon" type="image/png" href="/static/images/favicon.png">
</link>

</head>
```

Eine JS Library wird importiert, und die Pfade einiger statischen Dokumente wird angegeben. Darauf folgt die Definition der Navigationsleiste mit HTML-Logo.

```
<ul id="navbar">
  <li class="navelement navelementActive"> <a class="navtext" href=""> Home
</a> </li>
  <li class="navelement"> <a class="navtext" href="/about"> About </a> </li>
  <li>  </img> </li>
</ul>
```

Darunter ist sie in drei Spalten, eine Spalte für einen Roboter, eingeteilt. Jede Spalte bekommt zwei Diagramme für Beschleunigung und Gyroskop, sowie zwei Anzeigen, ob der jeweilige Roboter und Controller verbunden sind.

```
<div class="column" id="column1">
</div>
<div class="column" id = „column2“ style="left: 33%">
</div>
<div class="column" id="column3" style="left: 66%">
</div>
```

Mit folgender CSS:

```
.column {
  position: absolute;
  width: 30%;
  height: 100%;
  top: 0px;
  padding: 3%
}
```

So ist jedes div 33% der Seite breit, 100% der Seite lang, und alle 33% voneinander entfernt. Jede Spalte hat zwei Diagramme und zwei Elemente, die die Verbindung von Roboter und Controller anzeigen.

```
<div id="column0" class="column">
    <p style="text-align: center"> Roboter 0 </p>
    <canvas id="chartAcc0"> </canvas>
    <canvas id="chartGyro0"> </canvas>

    <ul id="statusList0">
        <li id="robotConnected0"> </li>
        <li id="controllerConnected0"> </li>
    </ul>
</div>
```

Zunächst müssen die Charts in JS initialisiert werden.

```
const chartVibration0 = new Chart("chartVibration0", {
    type: "line",
    data: {
        labels: [1,2,3,4,5,6,7,8,9,10],
        datasets: [{ 
            borderColor: "blue",
            data: [],
            fill: false
        }]
    },
    options: {
        animation: {duration: 75}
    }
});
```

Nun sollen diese Charts live geupdated werden. Requests an Ressourcen des Webservers werden über die sendRequest Funktion gesendet.

```
function sendRequest(url, callback) {
    let req = new XMLHttpRequest();
    req.responseType = 'json';
    req.open("GET", url);
    req.send();

    req.onreadystatechange = function() {
        if(req.readyState == 4) {
            callback(req)
        }
    };
}
```

Die XMLHttpRequest-Klasse wird instanziert. Über diese Instanz lässt sich zunächst mit .open und .send eine Anfrage auf eine Ressource beginnen. .readyState zeigt an, in welchem Zustand sich die Anfrage befindet, also ob sie z.B. aufgesetzt oder gesendet wurde. Ändert sich dieser Wert, wird die req.onreadystatechange Funktion aufgerufen, welche zunächst überprüft ob der Wert 4 ist und dann, ist das der Fall, die als Argument gelieferte callback

Funktion aufruft. Da nur die `.onreadystatechange` Funktion geändert wird, kann der Client andern JS Code ausführen, während auf die Antwort des Servers gewartet wird.

```
sendRequest("/data/log", function(req) {
    log_text = document.getElementById("log_text")
    log_text.innerHTML = req.response.join("\n");
    log_text.scrollTop = log_text.scrollHeight;
});
```

Hier wird zum Beispiel `/data/log` angefragt. Bekommt der Client eine Antwort, wird der Code, welcher Teil der anonymen Funktion ist, ausgeführt. Aus verschiedenen `sendRequests` setzt sich die Funktion `getData()` zusammen, welche jedes dynamische Element updaten soll. Jedes Element soll 10 mal pro Sekunde upgedatet werden. Die Anfragen werden jetzt gleichmäßig auf die 100ms aufgeteilt:

```
function getData() {
    if (index == 0) {
        sendRequest("data/gyro/0", function(req) {
            chartGyro0["data"] = req.response["x"]
            chartGyro0["data"] = req.response["y"]
            chartGyro0["data"] = req.response["z"]
            chartGyro0.update()
        });
    }

    else if (index == 1) {
        sendRequest("data/acc/0", function(req) {...})
    }

    else if (index == 2) {
        sendRequest("data/status/0", function(req) {...})
    }
    ...
    else if (index == 8) {
        sendRequest("data/status/2", function(req) {...})
    }
}

index++;
if (index == 9) {
    index = 0
}
}

let index= 0;

setInterval(getData, 100/9);
```

Um die neuen Daten in den Diagrammen anzuzeigen muss zum Schluss die `.update()` Methode aufgerufen werden.

Alleine um die Lognachrichten des Roboters anzeigen zu können, benötigt es schon viel funktionierende Software. Da diese auf einem Betriebssystem läuft, konnte sich die Entwicklung aber leicht gemacht werden. Die Teilprogramme konnten ihre Zustände raus schreiben, und die Anfragen des Clients sowie die Antworten des Servers konnten mit den Entwickler-tools des Webbrowsers analysiert werden. Sobald alles funktioniert hat, wurden diese Hilfestellungen jedoch wieder entfernt.

7.6 Development

7.6.1 Programmierung

In diesem Kapitel wird die Weise definiert, wie wir den Code hochladen. Der Roboter und Controller verwenden jeweils den atMega2560 und atMega328p µC. Das sind beides µC die für verschiedene Arduino Boards verwendet werden, weshalb für die Softwareentwicklung die gewöhnliche Arduino Umgebung verwendet werden kann. Ein gewöhnliches Arduino-Board wird mit einem USB – Kabel mit dem Computer verbunden. Ein zweiter µC ermöglicht „Serial over USB“ also eine (virtuelle) serielle Verbindung über die USB – Schnittstelle. Auf dem Arduino läuft ein Bootloader, ein kleines Programm welches beim Hochfahren des Arduinos startet, und das Programmieren über die Serielle Verbindung ermöglicht. Über diese können auch Debug – Nachrichten an den Computer übertragen werden. Wir haben allerdings weder den zweiten µC noch den Bootloader, da wir die µC direkt vom Händler kaufen. Deshalb mussten für Programmierung und Debugging andere Methoden verwendet werden, die folgend näher beschrieben werden.

Um Funktionalitäten von Arduino-Boards verwenden zu können schalten wir für Programmierung und Debugging einen zweiten Arduino dazwischen, wie folgt:

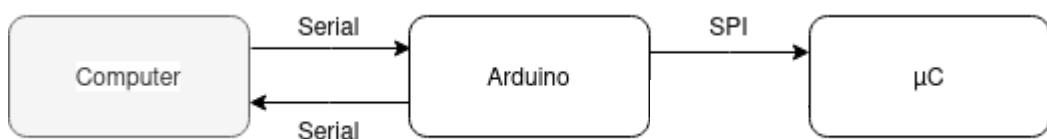


Abbildung 65: Schaltbild In-System-Programming

Programmiert kann der µC über den SPI – Bus werden. Für diesen Zweck haben Arduinos einen veränderten SPI - Bus am ICSP⁵ – Header ausgeführt:



⁵ ICSP (= „In Circuit Serial Programming“), ist eine andere Abkürzung für ISP (= „In System Programming“) und meint das Programmieren eines µC, während dieser im System eingebaut ist.

Abbildung 66: ICSP – Header der Arduinos

Verändert, weil statt dem üblichen „Slave Select“, der „Reset“ Pin ausgeführt ist. Auf unserem PCB ist der gleiche Header implementiert.

Das Programmieren erfolgt wie folgt:

1. Zunächst muss das Programmierer-Board selbst den Code „Programmer.ino“ hochgeladen bekommen.
2. Danach kann man die von uns erstellten ICSP – Stecker miteinander verbinden. Der Stecker mit zusätzlichem Jumper-Draht wird auf die ICSP - Schnittstelle des Programmierer-Boards angesteckt, das zusätzliche Kabel auf Pin 10. Dieser ist mit Pin 5: „reset“ verbunden. Die andere Seite wird auf das ICSP – Interface des zu programmierenden Boards angesteckt. Die schwarz angemalte Seite zeigt nach oben, ist also näher an Pin 1 der Schnittstelle.
3. Nun kann das Programm hochgeladen werden. Als Board wird „Arduino Mega2560“ ausgewählt, als Programmer „ArduinoISP“ und „Upload with Programmer“ betätigt.

7.6.2 Probleme

Im Zuge der Entwicklungsarbeit kam es bereits zu einigen Problemen und Komplikationen. Auf einige davon, sowie ihre Lösung, soll hier eingegangen werden.

1) Das Rf24 Modul

Eigentlich sollte dieses Problem im Hardwareentwicklungsteil stehen. Lange Zeit wurde das jedoch nicht erkannt, ganze Tage wurden damit verbracht Fehler in der Software zu finden, die es gar nicht gab. Das RF24L01+ Modul ist nicht gut gegen Interferenzen geschützt. Mit Isolierband und Alufolie konnte dieses Problem aber gelöst werden. Das zeigt, wie wichtig es ist, mit LEDs oder Lognachrichten über den Zustand der Komponente informiert zu sein.

2) Software oder Hardware?

Im Zuge der Entwicklung war oft nicht klar, ob ein bestimmter Fehler in Soft- oder Hardware lag und ob er in Controller, Roboter, Server oder der Kommunikation zu finden ist. Das erschwerte die Entwicklung erheblich, weshalb seitdem ein Schwerpunkt auf einfacheres Debugging gesetzt wurde.

8 Projektmanagement

In diesem Kapitel wird das Arbeitspaket 4 Projektmanagement beschrieben. Es hatte zum Ziel, die Organisation, Planung und Struktur des Projekts für die Projektteilnehmer und den Betreuer zu erleichtern. Es werden Snapshots der Arbeitszeiten der Diplomanden aus der AZE und andere organisatorische Hilfsmittel präsentiert. Zudem werden die Termine der Meilensteinpläne, die während der Planungsphase der Diplomarbeit erstellt wurden, mit den tatsächlichen Terminen verglichen und erläutert, warum es zu Abweichungen kam.

8.1 Verwaltung

Für eine effiziente Verwaltung von Programmcode, 3D-Dateien, KiCAD-Projekten und Dokumentationen wurde zunächst auf Google Drive gesetzt. Aufgrund vermehrter Probleme bei der Bearbeitung des Codes erfolgte jedoch später der Wechsel zu der Front-End-Software für GIT, SourceTree. Dadurch konnten alle Diplomanden gleichzeitig an der Dokumentation arbeiten und an den Projektdateien arbeiten.

Graph	Beschreibung	Datum	Autor
	v1.0 Revert "Reverting prev Commit." ups.	2 Feb 2024 14:57	RoteRuebe <yannick.zickler@gmail.com>
	Merge branch 'main' of github.com:RoteRuebe/diplomarbeit	2 Feb 2024 14:41	RoteRuebe <yannick.zickler@gmail.com>
	Reverting prev Commit.	2 Feb 2024 14:41	RoteRuebe <yannick.zickler@gmail.com>
	AZE aktualisiert 😊, verbesserte Version der DA hochgeladen 🎉	31 Jän 2024 14:30	Lukas Lucut <lukas.lucut@hotmail.com>
	Neuer server code. Nicht fertig, Roboter code muss angepasst werden. Außerdem: pfadänderungen	30 Jän 2024 15:31	RoteRuebe <yannick.zickler@gmail.com>
	Rounded boxes!	28 Jän 2024 13:31	RoteRuebe <yannick.zickler@gmail.com>
	Neues icon	28 Jän 2024 11:16	RoteRuebe <yannick.zickler@gmail.com>
	Very minor fix	28 Jän 2024 10:56	RoteRuebe <yannick.zickler@gmail.com>
	Merge branch 'main' of github.com:RoteRuebe/diplomarbeit	28 Jän 2024 10:52	RoteRuebe <yannick.zickler@gmail.com>
	Das internet ist fuer uns alle neuLand. Erweiterung website	28 Jän 2024 10:51	RoteRuebe <yannick.zickler@gmail.com>
	AZE aktualisiert	28 Jän 2024 10:05	Lukas Lucut <lukas.lucut@hotmail.com>
	More comments	28 Jän 2024 8:55	RoteRuebe <yannick.zickler@gmail.com>
	New Robo code	26 Jän 2024 16:16	RoteRuebe <yannick.zickler@gmail.com>
	Merge branch 'main' of github.com:RoteRuebe/diplomarbeit	26 Jän 2024 16:09	RoteRuebe <yannick.zickler@gmail.com>
	Deleting log files, new Server code	26 Jän 2024 16:09	RoteRuebe <yannick.zickler@gmail.com>
	no message	26 Jän 2024 16:01	Lukas Lucut <lukas.lucut@hotmail.com>
	Merge remote-tracking branch 'origin/main'	26 Jän 2024 15:26	Lukas Lucut <lukas.lucut@hotmail.com>
	I2C-Bus	26 Jän 2024 15:26	Lukas Lucut <lukas.lucut@hotmail.com>
	Uploading server code	26 Jän 2024 14:43	RoteRuebe <yannick.zickler@gmail.com>
	Deleting useless files	26 Jän 2024 14:20	RoteRuebe <yannick.zickler@gmail.com>
	testetetet	25 Jän 2024 17:05	Lukas Lucut <lukas.lucut@hotmail.com>

Abbildung 67: Git-Verwaltung der Diplomarbeit

In Abbildung 67 kann man den Entwicklungsverlauf beobachten, da bei jeder Bearbeitung die Version aktualisiert werden muss. Außerdem ist es möglich, auf die vergangenen Versionen zuzugreifen, was besonders in der Software-Entwicklung von Bedeutung ist.

8.2 Erfassung der Arbeitszeiten und Reporting

Im Zuge des Projektes wurden die Arbeitszeiten jedes Team-Mitglieds erfasst und den entsprechenden Arbeitspaketen zugeordnet. Aus diesen Daten wurden wöchentlich Fortschrittsberichte zu den einzelnen Arbeitspaketen erstellt. Ebenso wurde wöchentlich ein Weekly Report erstellt, der die Themenbearbeitung pro Woche und Gruppenmitglied zusammenfasste. Die im folgenden wiedergegebenen „Schnappschüsse“ zeigen den Verlauf des Projekts aus PM-Perspektive:

8.2.1 Arbeitszeiterfassung 04.09.2023 – 04.02.2024

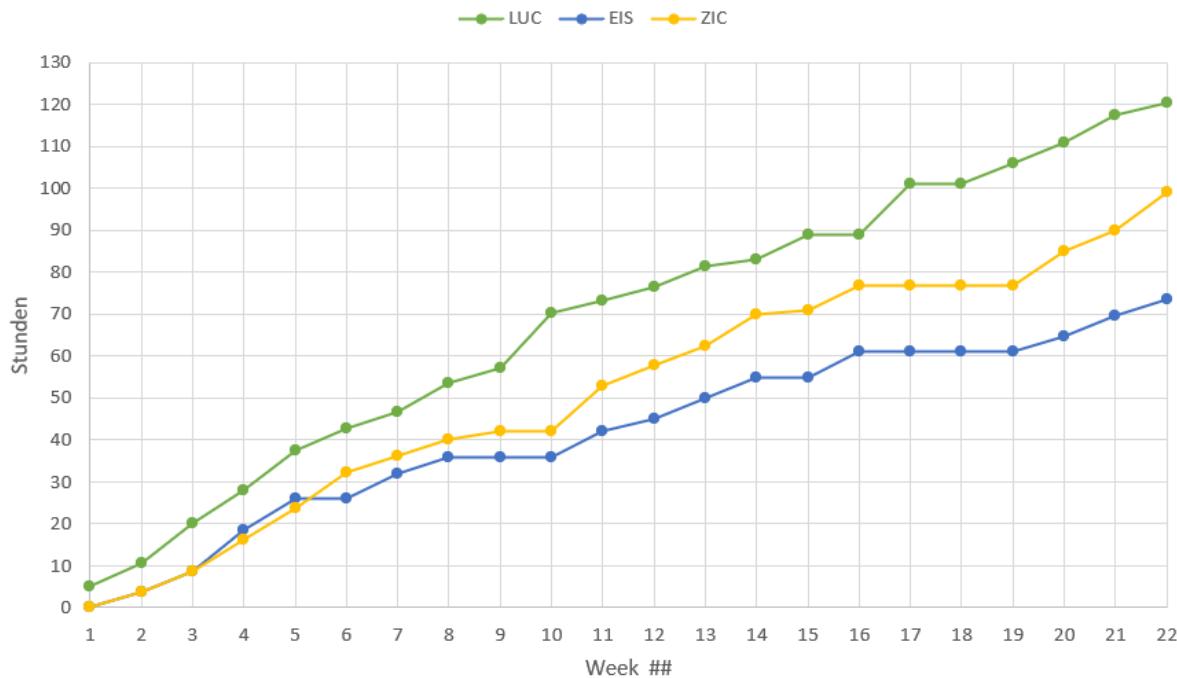


Abbildung 68: AZE Week 1 - Week 22

In der Grafik sind die während des ersten Semesters geleisteten Arbeitsstunden der Diplomanden dargestellt. Bis zum Beginn des Projekts in Week 4 zeigen die Arbeitsstunden der Diplomanden eine vergleichbare Verteilung, die auf die aufgegliederten und geplanten Arbeitspakete für klare Zielsetzungen zurückzuführen ist. Als Projektleiter widmete Lukas Lucut während der Schulzeit verstärkt Zeit der fortlaufenden Dokumentation und dem Projektmanagement. Die Arbeit von Yannick Zickler baute auf dem zweiten Arbeitspaket, der Hardware-Entwicklung, auf, wodurch Vorbereitungen wie Softwarecodes und Recherchen erstellt wurden, die später jedoch überarbeitet wurden.

Ab Week 5 konzentrierten sich Yannick Zickler und Lukas Lucut darauf, die erste Version der Roboterplatine korrekt in Betrieb zu nehmen, während Bastian Eismann parallel ein Prototypengehäuse konstruierte. Diese Aufgaben erforderten Erweiterungen am 3D-Drucker, um die Druckqualität zu verbessern, und wurden bis Week 12 im Dezember abgeschlossen. Während dieser Phase traten jedoch Fehler auf, die behoben werden mussten, und der Fokus auf die Diplomarbeit wurde durch schulische Verpflichtungen beeinträchtigt, was von Week 7 bis Week 14 zu stagnierenden Arbeitsstunden führte. Fehler in Software und Hardware, insbesondere im Zusammenhang mit dem Funkmodul NRF24L01+, erforderten erhebliche Zeit für Fehlersuche und Debugging.

Schwierigkeiten beim 3D-Druck verzögerten ebenfalls den Bauprozess des Prototypen. Von Week 19 bis Week 22 wurden bedeutende Meilensteine erreicht, darunter eine stabile Kommunikation zwischen Controller und Roboter sowie der erfolgreiche Bau eines Prototyps mit anschließenden Testfahrten.

8.2.2 Arbeitszeiterfassung 12.02.2024 - 05.04.2024

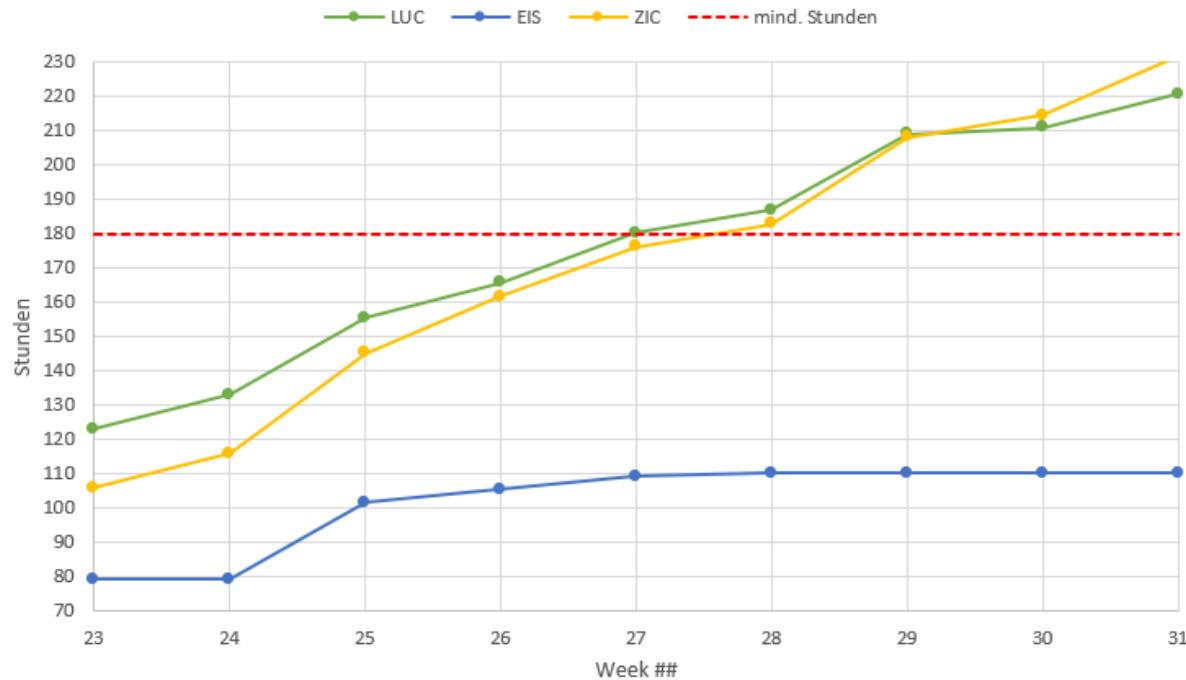


Abbildung 69: AZE Week 23 – Week 32

Zu Beginn des Semesters wurde viel Zeit darauf verwendet, die grundlegende Funktionalität des Roboters - insbesondere die Steuerung - mit den neuen Komponenten umzusetzen. Nach erfolgreichen Testaufbauten konnte dann mit der Entwicklung des Controller-PCBs begonnen werden, was vor allem in Week 24 intensiviert wurde, da zu dieser Zeit die Projektwoche stattfand. Durch den dadurch entstandenen Zeitgewinn konnten erhebliche Fortschritte in der Diplomarbeit erzielt werden. Es wurde auch eine zusätzliche Erweiterung beschlossen, die bei einem Defekt keine Auswirkungen auf die übrigen Funktionen haben sollte. Parallel dazu wurde das erste 3D-Design des Roboters erstellt, das in den folgenden Wochen ausgedruckt werden sollte.

Die verbleibenden Wochen wurden hauptsächlich darauf verwendet, die Verbindung zum Server herzustellen und Verzögerungen zu minimieren, beides erwies sich als zeitaufwändig. In Week 28 wurde dann intensiv daran gearbeitet, die gesamte Hardware und Software fertigzustellen, wobei insbesondere das Problem der Zeitverzögerung gelöst werden konnte. Allerdings erwies sich die Herstellung der Verbindung zwischen dem Server und den drei Robotern als zeitaufwändig.

In Woche 31 wurde das Kommunikationsproblem erfolgreich gelöst, und die verbleibende Zeit wurde genutzt, um die Module zu programmieren, was ebenfalls erfolgreich war. Allerdings konnte das Gehäuse nicht rechtzeitig fertiggestellt werden, weshalb externe Unterstützung erforderlich war, um den Prozess abzuschließen.

9 Bedienungsanleitung

Die Erläuterungen zur Steuerung, zur Inbetriebnahme, zur Verbindungsherstellung mit dem Server sowie zum Hochladen von Codes mittels ICSP wurden in einem separaten Dokument "2324_5AHEL_SumoBots_Bedienungsanleitung" festgehalten.

10 Business Case

Die Diplomarbeit wurde im Rahmen eines Business-Plans analysiert, um den potenziellen Nutzen und die Marktfähigkeit zu bewerten. Die Ergebnisse dieser Analyse wurden in einem separaten Dokument mit dem Titel "2324_5AHEL_DA_SumoBots_BusinessCase" dokumentiert.

11 Finale Ergebnisse

Die Entwicklung und Integration der Hardware verlief erfolgreich und konnte ordnungsgemäß in Betrieb genommen werden, wie bereits beschrieben. Aufgrund von Zeitmangel konnte jedoch die Implementierung der Selbststeuerung mittels Umgebungserkennung der Infrarotsensoren nicht umgesetzt werden. Zusätzlich wurde die Idee, einen ESP32 mit Kamera und einen Raspberry Pi Zero mit Display hinzuzufügen, verworfen, da das Video auf dem Display aufgrund erheblicher Zeitverzögerungen nicht flüssig und erkennbar wiedergegeben wurde.

Der Programmcode für die Motorsteuerung wurde erfolgreich auf dem Controller getestet und konnte bereits im Roboter Rev A angewendet werden. Die Entwicklung des Kommunikationsprotokolls stellte sich als Herausforderung heraus, da verschiedene Teilnehmer unterschiedliche Daten senden mussten. Ein Ansatz war, sie auf verschiedene Frequenzen zu senden, was jedoch zu instabilen Verbindungen führte, da die Funkmodule nicht für schnelles Frequenzwechseln ausgelegt waren. Nun wurde ein effizienteres Protokoll implementiert, das die benötigte Datenrate senkt und es allen Teilnehmern ermöglicht, gleichzeitig zu kommunizieren.

Im Verlauf der Diplomarbeit stellte das NRF24L01+ Funkmodul eine Vielzahl von Problemen dar, die eine zuverlässige Verbindung zwischen den Geräten erschwerten. Daher wäre für eine Diplomarbeit, die die Kommunikation mehrerer Geräte miteinander ermöglicht, die Implementierung von ESP32 die bessere Wahl, da diese weniger Probleme im WLAN-Betrieb aufweisen. Darüber hinaus bietet das ESP-NOW Protokoll eine optimale Lösung für solche Anwendungen, dessen Existenz erst im Verlauf des zweiten Semesters bekannt wurde.

Dennoch konnte das Gehäuse des Controllers nicht fertiggestellt werden. Dies ist darauf zurückzuführen, dass es an den erforderlichen Kenntnissen im Bereich 3D-Modellierung fehlte, was zu Verzögerungen in der Entwicklung des Gehäuses führte und keine ausreichende Zeit mehr für die Konstruktion blieb.

Während der Diplomarbeit wurden drei Roboter sowie drei Controller fertiggestellt, die erfolgreich mit dem programmierten Protokoll mit dem Server kommunizieren und eine Steuerung ermöglichen. Die Erweiterungen der einzelnen Module konnten abschließend programmiert und implementiert werden.

Zusammenfassend lässt sich festhalten, dass zu Beginn der Arbeit mehrere Ziele festgelegt wurden, von denen einige aufgrund zeitlicher Einschränkungen nicht vollständig erreicht werden konnten. Dennoch gelang die erfolgreiche Implementierung des Großteils der geplanten Erweiterungen.

12 Tabellen und Abbildungsverzeichnis

Abbildung 1: Aufbau des Projekts	9
Abbildung 2 Systemarchitektur des Roboters	10
Abbildung 3 Systemarchitektur des Controllers	11
Abbildung 4: Objektstrukturplan (OSP)	13
Abbildung 5: Projektstrukturplan (PSP) AP1-AP6	14
Abbildung 6 Cura 3D Interface	17
Abbildung 7 Kompletter Druckfehler	17
Abbildung 8: Drucker-Komponenten Überblick	18
Abbildung 9 Warping	20
Abbildung 10: Zahnriemen-Prototyp	21
Abbildung 11: Zahnradantrieb-Prototyp	22
Abbildung 12: neue Zahnräder	23
Abbildung 13: Stangenantrieb Beispiel (https://de.wikipedia.org/wiki/Stangenantrieb_%28Eisenbahn%29)	24
Abbildung 14: Räder	25
Abbildung 15: Prototyp Seitenteile	26
Abbildung 16: Motorhalter	27
Abbildung 17: Bodenplattendesign	28
Abbildung 18: Sensorhaltung	29
Abbildung 19: Abdeckung	29
Abbildung 20: Seitenteil Rechts und Links	30
Abbildung 21: Gehäuse-Design	31
Abbildung 22: Volles Abbild des Roboters	32
Abbildung 23: Prototyp des Controllers mit Drucksensoren	33
Abbildung 24: Erstes Gehäuse Design	34
Abbildung 25: Mechanischer Trigger	35
Abbildung 26: Muttern Löcher	36
Abbildung 27: Eingebettetes Kugellager	37
Abbildung 28: Entwicklungsablauf der Roboter Hardware	39
Abbildung 29: Blockschaltbild von Roboter Rev A V1.00	40
Abbildung 30: Roboter Rev A V1.00 PCB	41
Abbildung 31 : NRF24L01+ Funkmodul	43
Abbildung 32: NRF24L01 Blockschaltbild (https://www.sparkfun.com/datasheets/Components/SMD/nRF24L01Pluss_Preliminary_Product_Specification_v1_0.pdf)	43
Abbildung 33 3,3V-Längsreglerschaltung	44
Abbildung 34 NRF24L01 Messung der Spannungsversorgung	45
Abbildung 35 NRF24L01 Messung der Spannungsversorgung mit $C = 10\mu F$ Elko und Logiksignal	45
Abbildung 36 Motorshield-Blockschaltbild	46
Abbildung 37: H-Brücke Schaltplan (https://de.wikipedia.org/wiki/Vierquadrantensteller)	47
Abbildung 38: Logik-Pegel-Converter	48
Abbildung 39: Logik-Level-Konverter (https://learn.sparkfun.com/tutorials/bi-directional-logic-level-converter-hookup-guide/all)	49
Abbildung 40: SPI-Bus (https://www.rc-modellbau-portal.de/index.php?threads/spi-schnittstelle-beim-arduino-co.6994/)	50
Abbildung 41: SPI-Bus CLK-Konfiguration (https://edistechlab.com/wp-content/uploads/2020/11/Screenshot-2020-11-22-at-09.37.57.png)	50
Abbildung 42: SPI-Bus Kommunikation (https://edistechlab.com/wie-funktioniert-spi/?v=fa868488740a)	51
Abbildung 43: Schaltplan Pierce-Oszillator (https://de.wikipedia.org/wiki/Quarzoszillator)	51
Abbildung 44: Oszillatorschaltung beim ATMEGA2560	52
Abbildung 45: ESP32-CAM	52
Abbildung 46: Blockschaltbild von Roboter Rev B V1.00	53
Abbildung 47: Roboter Rev B V1.00 PCB	55
Abbildung 48: I ² C-Bus (https://www.design-reuse.com/articles/54776/i2c-interface-timing-specifications-and-constraints.html)	57
Abbildung 49: I ² C-Bus Zeitverhalten (https://de.wikipedia.org/wiki/I%C2%B2C)	57

<i>Abbildung 50: OLED-Display SSH1106</i>	58
<i>Abbildung 51: Vibrationssensor SW-420</i>	59
<i>Abbildung 52: Farbsensor TCS3200</i>	59
<i>Abbildung 53: Funktionsweise Farbsensor (https://elektro.turanis.de/html/prj029/index.html)</i>	60
<i>Abbildung 54: Infrarotsensor Sharp GP2Y0A21YK0F</i>	60
<i>Abbildung 55: Infrarotsensor Blockschaltbild (https://global.sharp/products/device/lineup/data/pdf/datasheet/gp2y0a21yk_e.pdf)</i>	61
<i>Abbildung 56: Gyroskop MPU6050</i>	61
<i>Abbildung 57: Funktionsweise des Accelerometers (https://www.elektronik-kompendium.de/sites/bau/1503041.htm)</i>	62
<i>Abbildung 58: Controller Blockschaltbild</i>	63
<i>Abbildung 59: Controller PCB</i>	64
<i>Abbildung 60: Trigger Blockschaltbild</i>	65
<i>Abbildung 61: 5V-Längsreglerschaltung</i>	66
<i>Abbildung 62: Joystick Schaltplan</i>	67
<i>Abbildung 63: Blockschaltbild der Netzwerkarchitektur</i>	69
<i>Abbildung 64: Diagramm für Veranschaulichung der Berechnung von motorTurn Werten</i>	75
<i>Abbildung 65: Schaltbild In-System-Programming</i>	84
<i>Abbildung 66: ICSP – Header der Arduinos</i>	85
<i>Abbildung 67: Git-Verwaltung der Diplomarbeit</i>	86
<i>Abbildung 68: AZE Week 1 - Week 22</i>	87
<i>Abbildung 69: AZE Week 23 – Week 32</i>	88

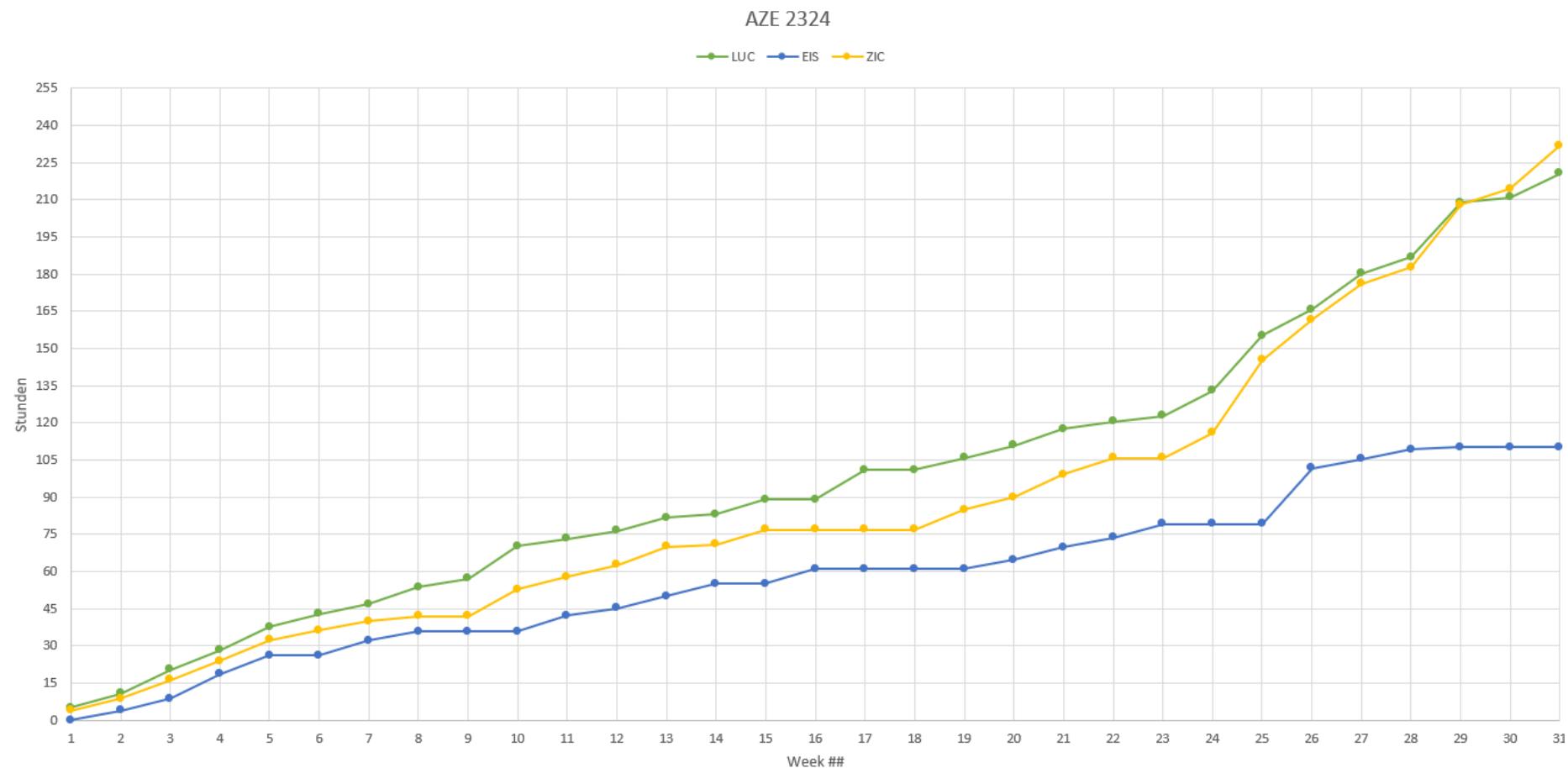
13 Literaturverzeichnis und Quellenangaben

- <https://collab.dvb.bayern/pages/viewpage.action?pagId=69683490>
<https://forum.arduino.cc/t/nrf24l01-can-long-wires-cause-this-problem/481015>
<https://arduino.stackexchange.com/questions/43267/more-missed-messages-with-longer-cables-with-nrf24l01>
<https://www.mikrocontroller.net/topic/431097>
https://www.elektronik-kompendium.de/sites/praxis/bauteil_ky023-joystick.htm
<https://embedds.com/adc-on-atmega328-part-1/>
<https://www.elprocus.com/nrf24l01/>
https://en.wikipedia.org/wiki/RF_module
<https://www.rc-modellbau-portal.de/index.php?threads/spi-schnittstelle-beim-arduino-co.6994/>
<http://www.knap.at/datenblaetter/all/pierceos.pdf>
<https://elektro.turanis.de/html/prj029/index.html>
https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf
https://ww1.microchip.com/downloads/en/devicedoc/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf
https://www.sparkfun.com/datasheets/Components/SMD/nRF24L01Pluss_Preliminary_Product_Specification_v1_0.pdf
https://cdn.shopify.com/s/files/1/1509/1638/files/1_3_Zoll_Display_Datenblatt_AZ-Delivery_Vertriebs_GmbH_rev.pdf?v=1606164520
<https://www.canadarobotix.com/products/1886>
<https://www.elektronik-kompendium.de/sites/bau/1503041.htm>

<https://elektro.turanis.de/html/prj029/index.html>
<https://de.wikipedia.org/wiki/I%C2%B2C>
<https://de.wikipedia.org/wiki/Vierquadrantensteller>
<https://www.design-reuse.com/articles/54776/i2c-interface-timing-specifications-and-constraints.html>
<https://de.wikipedia.org/wiki/Quarzoszillator>
<https://www.rc-modellbau-portal.de/index.php?threads/spi-schnittstelle-beim-arduino-co.6994/>
<https://edistechlab.com/wp-content/uploads/2020/11/Screenshot-2020-11-22-at-09.37.57.png>
<https://nrf24.github.io/RF24/classRF24.html#a025fcbad6f062d18252485c1d6ba574f>
https://global.sharp/products/device/lineup/data/pdf/datasheet/gp2y0a21yk_e.pdf
<https://ww1.microchip.com/downloads/en/DeviceDoc/30277d.pdf>
<https://docs.arduino.cc/built-in-examples/arduino-isp/ArduinoISP/>
<https://arduino.stackexchange.com/questions/38699/programming-arduino-via-icsp>
<https://github.com/nRF24/RF24>
<https://nrf24.github.io/RF24/>
<https://wolles-elektronikkiste.de/mpu6050-beschleunigungssensor-und-gyroskop>
<https://www.w3schools.com/>
<https://forum.arduino.cc/t/solved-why-cant-i-print-a-float-value-with-sprintf/367971>
<https://stackoverflow.com/questions/1224463/is-there-any-way-to-call-a-function-periodically-in-javascript>
<https://support.microsoft.com/de-de/office/konfigurieren-von-kopf-und-fu%C3%9Fzeilen-%C3%BCberschiedliche-abschnitte-eines-dokuments-94332643-a6e9-46aa-ab29-064f1d356db6>

14 Anhänge

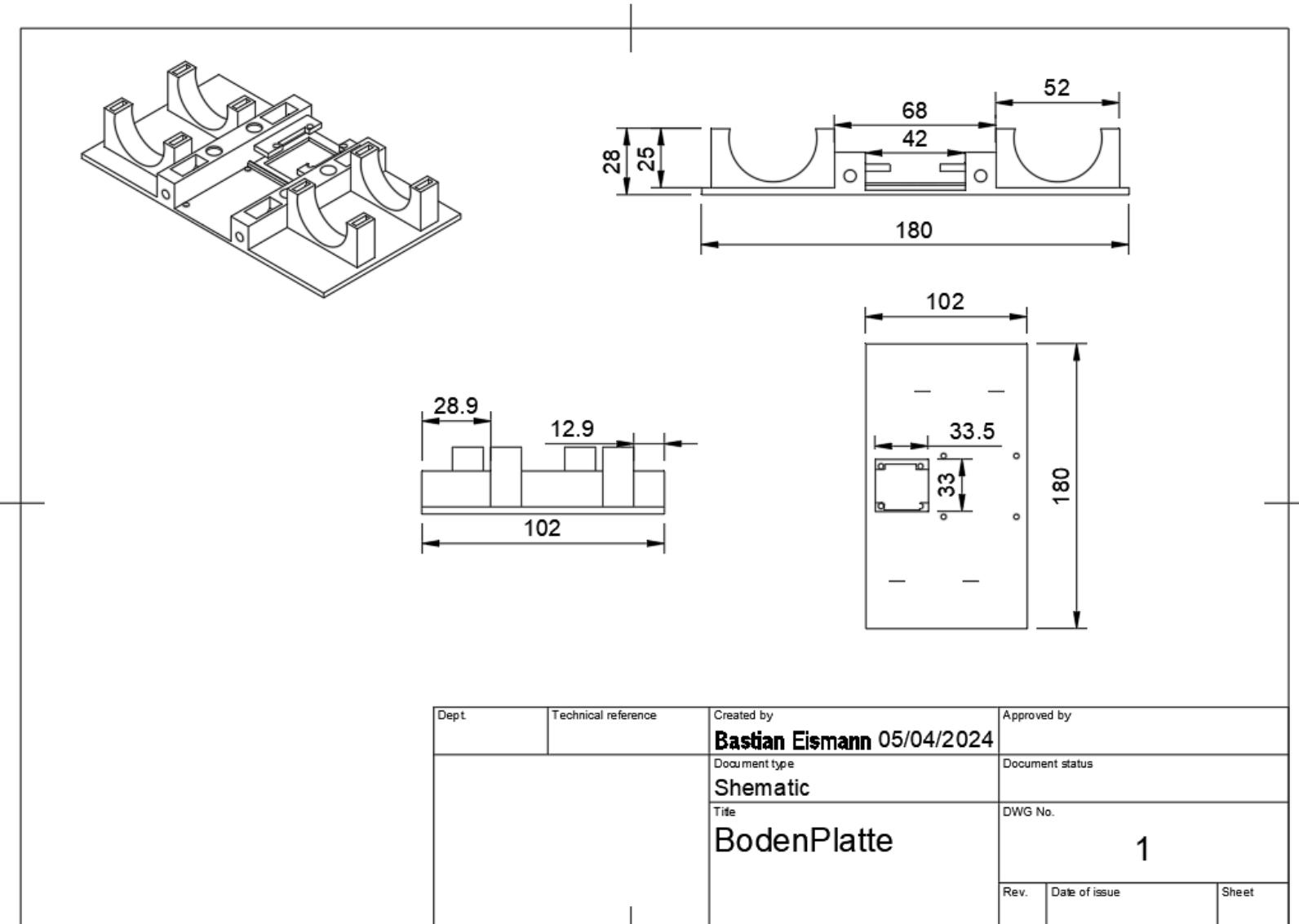
14.1 Arbeitszeitverfassung 2023/24

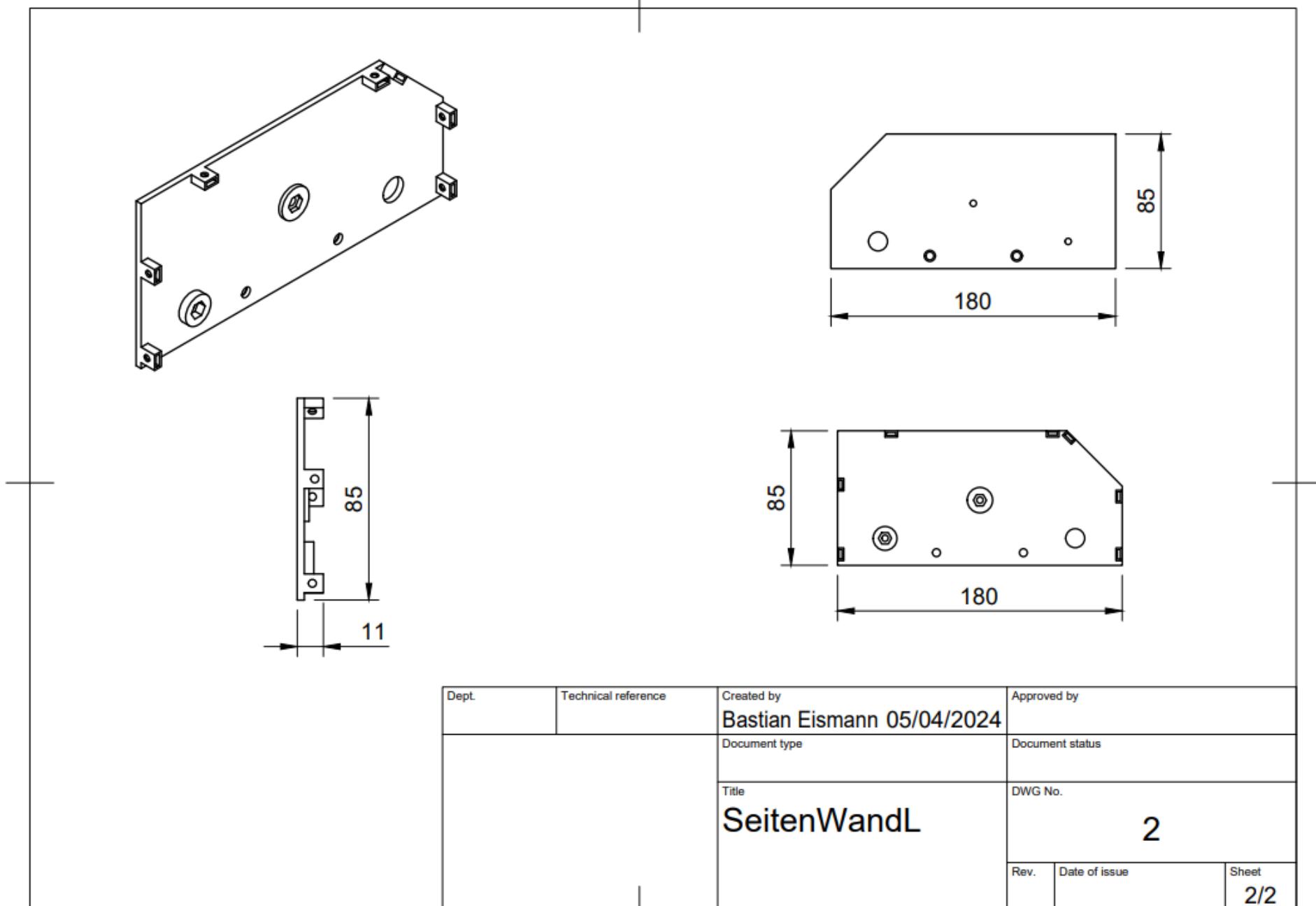


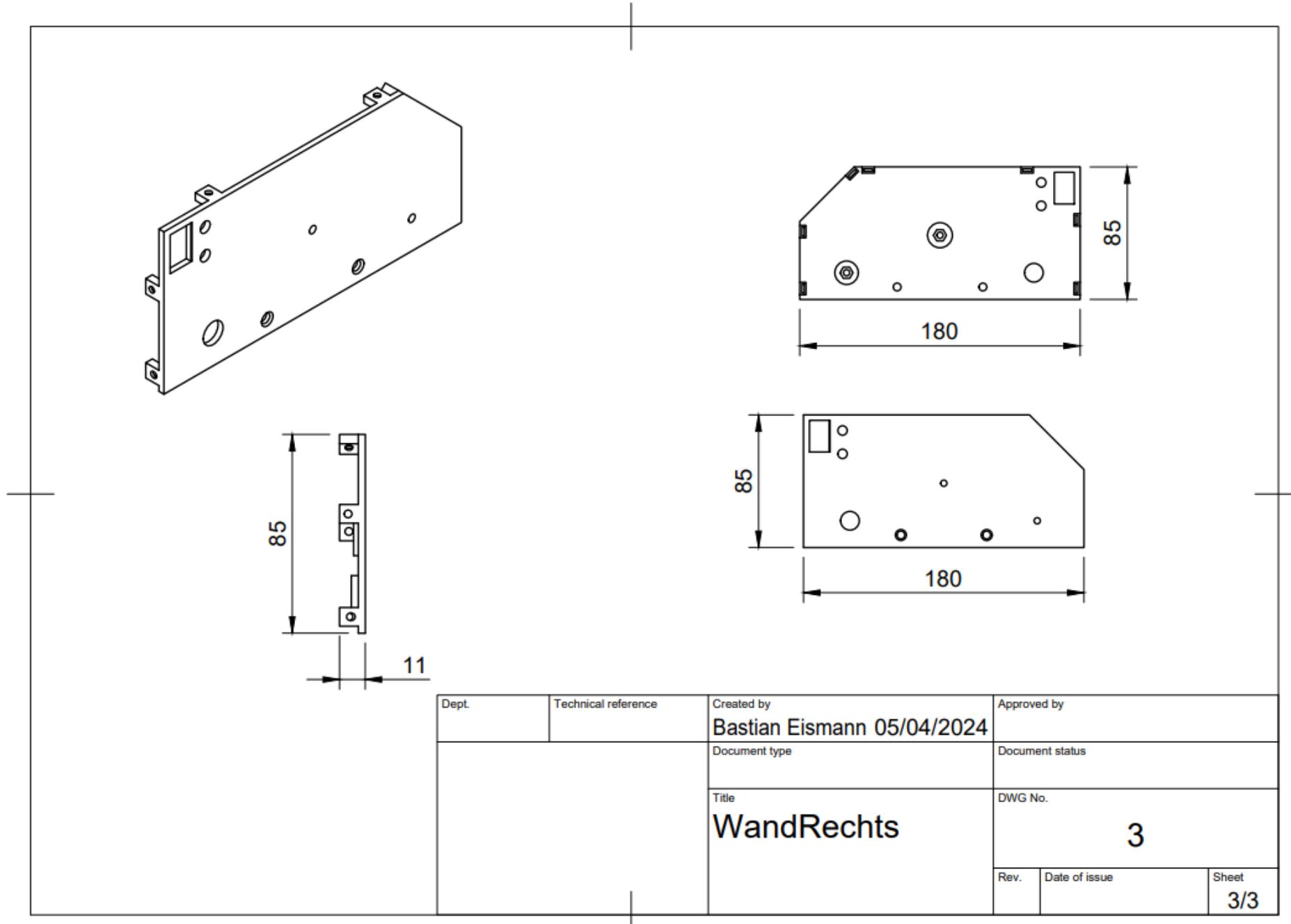
14.2 Project-Libre Netzplan

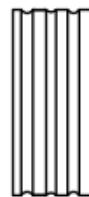
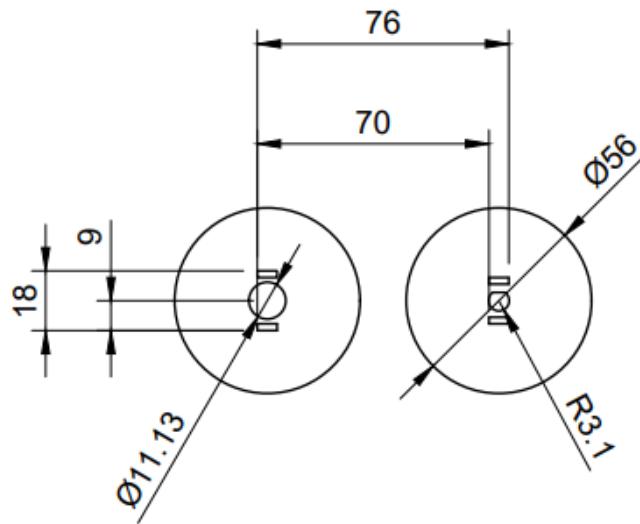
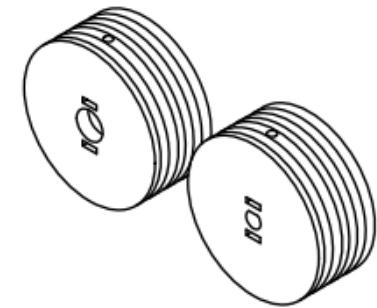


15 3D-Modelle und Konstruktionszeichnungen

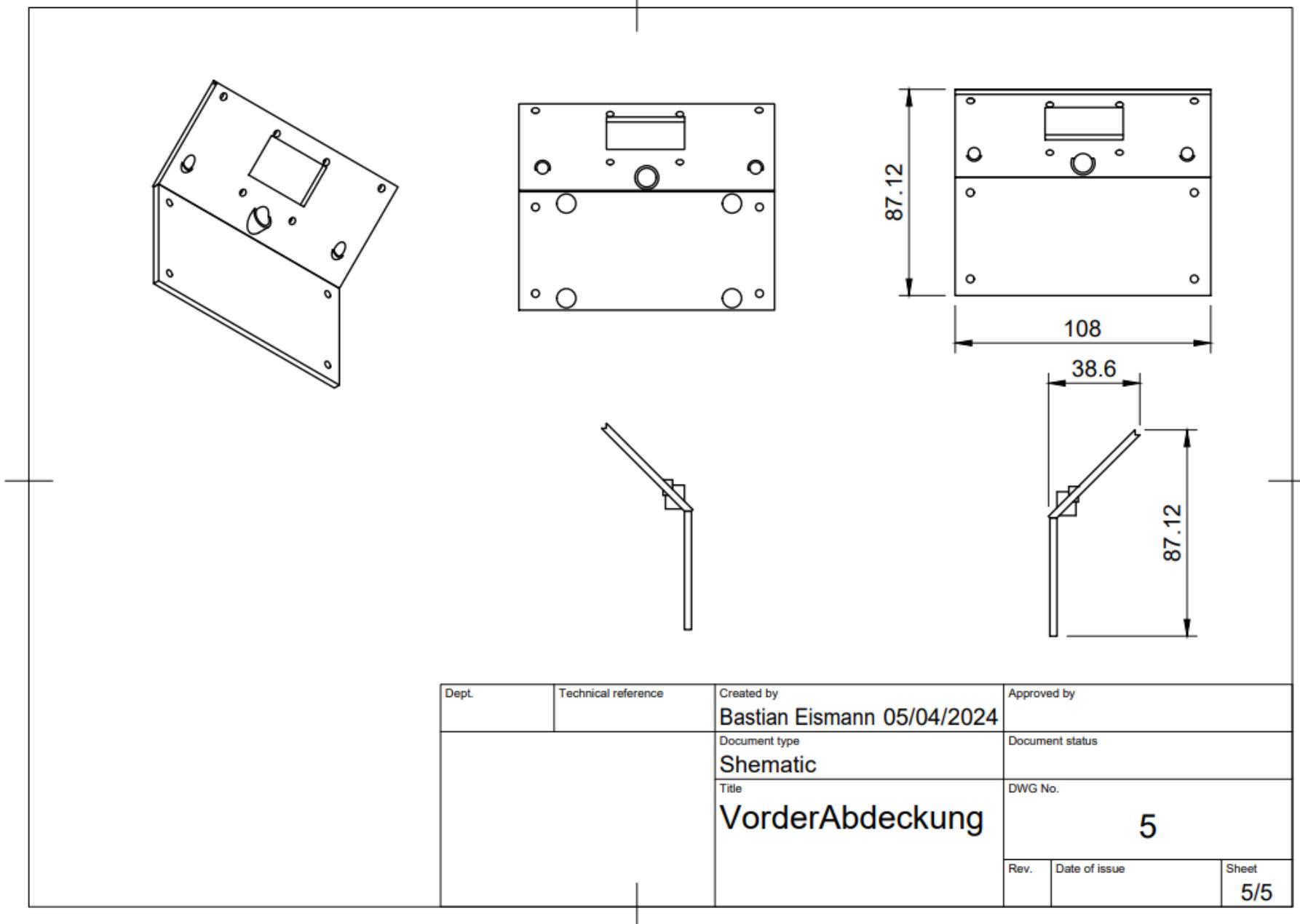


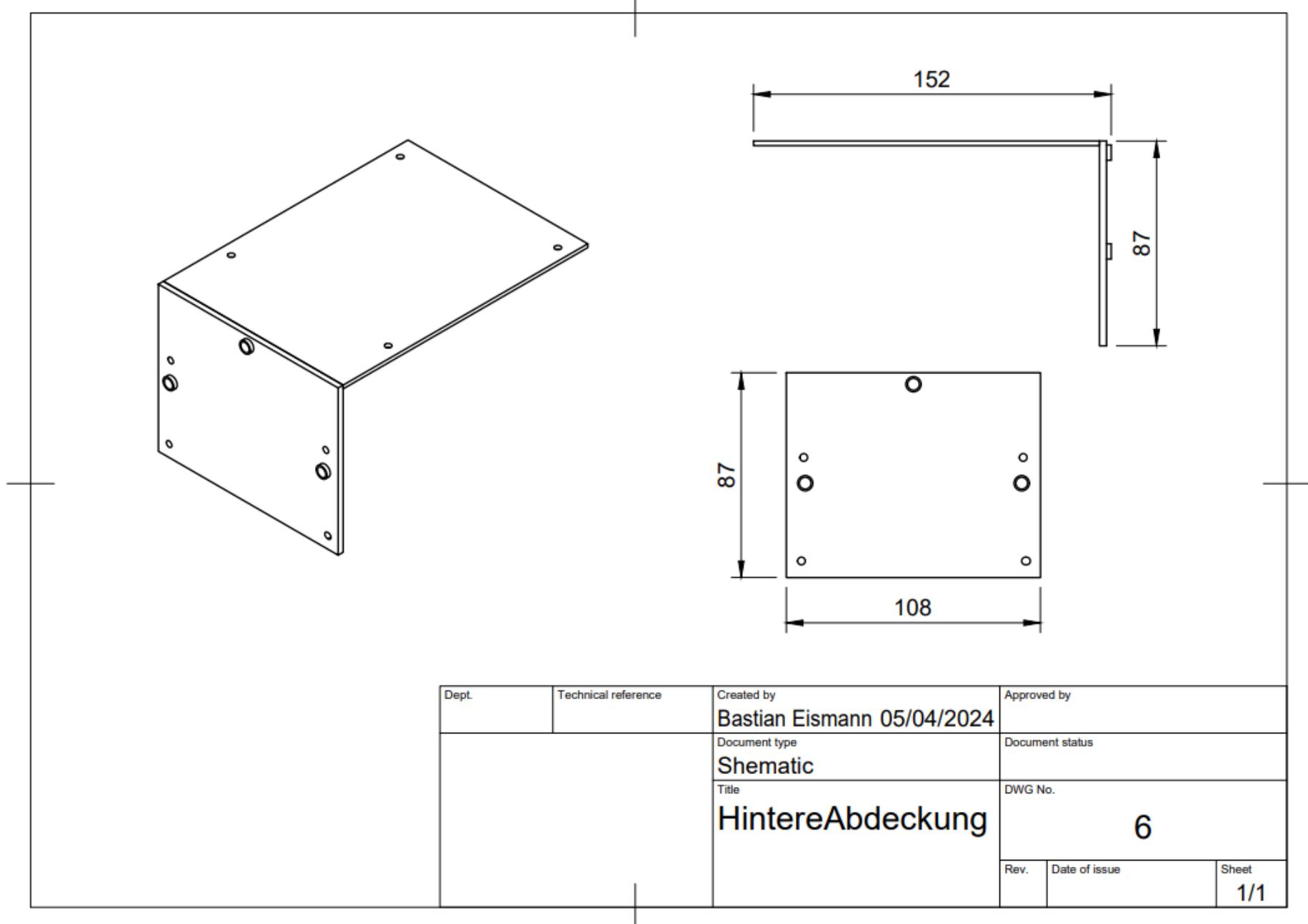


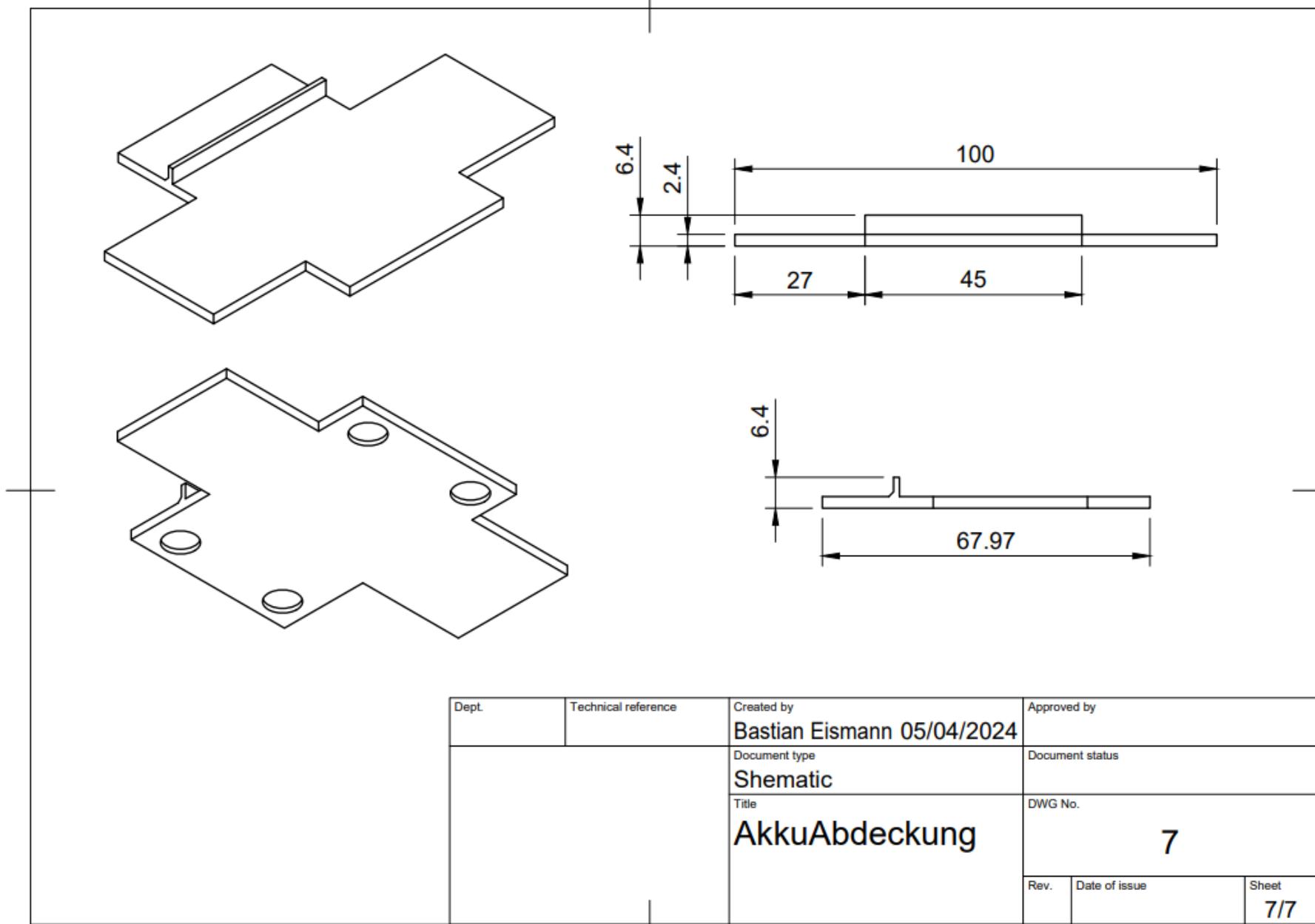




Dept.	Technical reference	Created by Bastian Eismann 05/04/2024	Approved by
	Document type		Document status
	Title Räder		DWG No. 4
	Rev.	Date of issue	Sheet 4/4

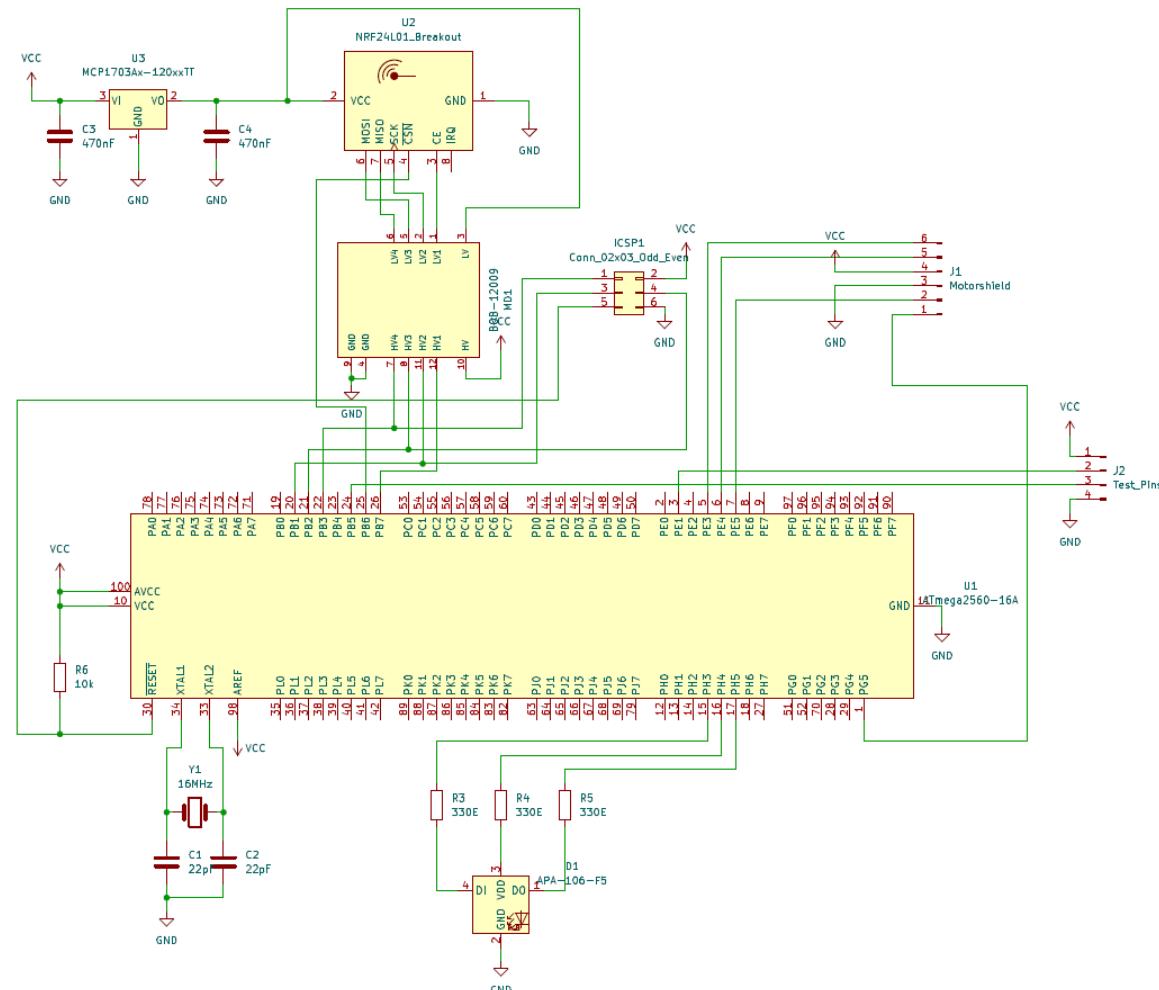




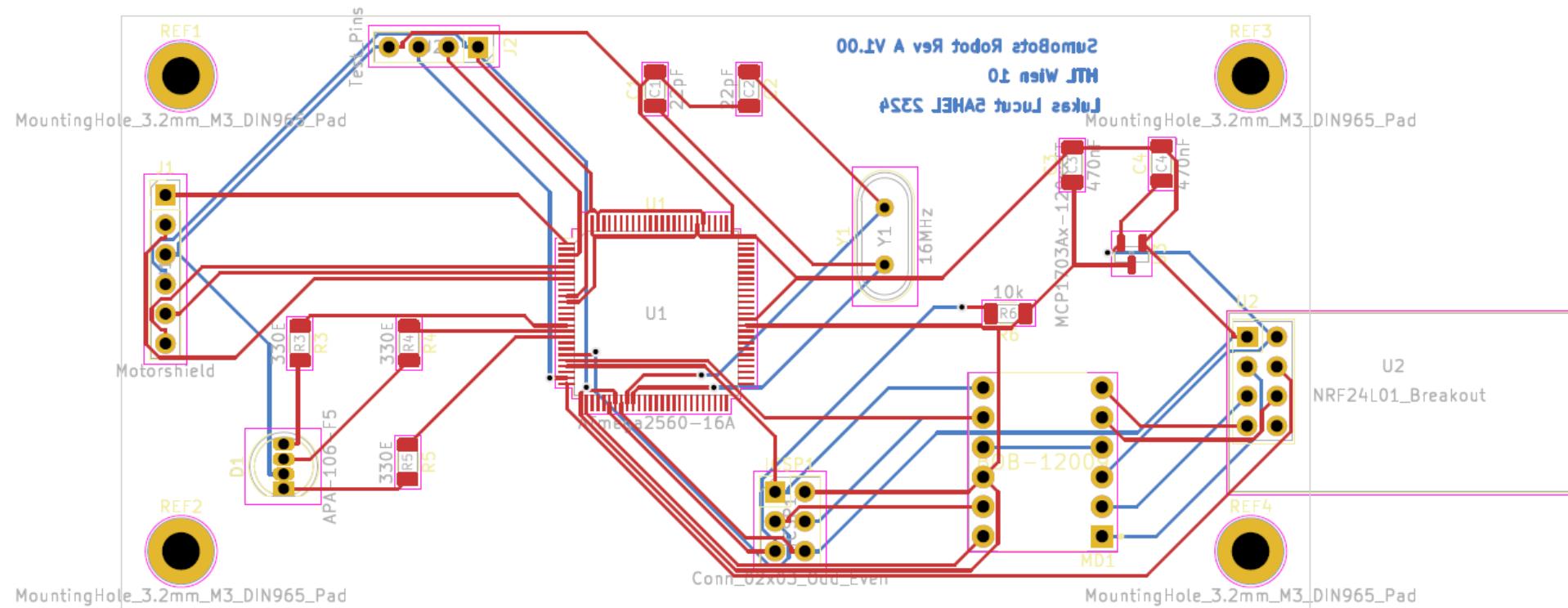


16 Schaltpläne und PCB Designs

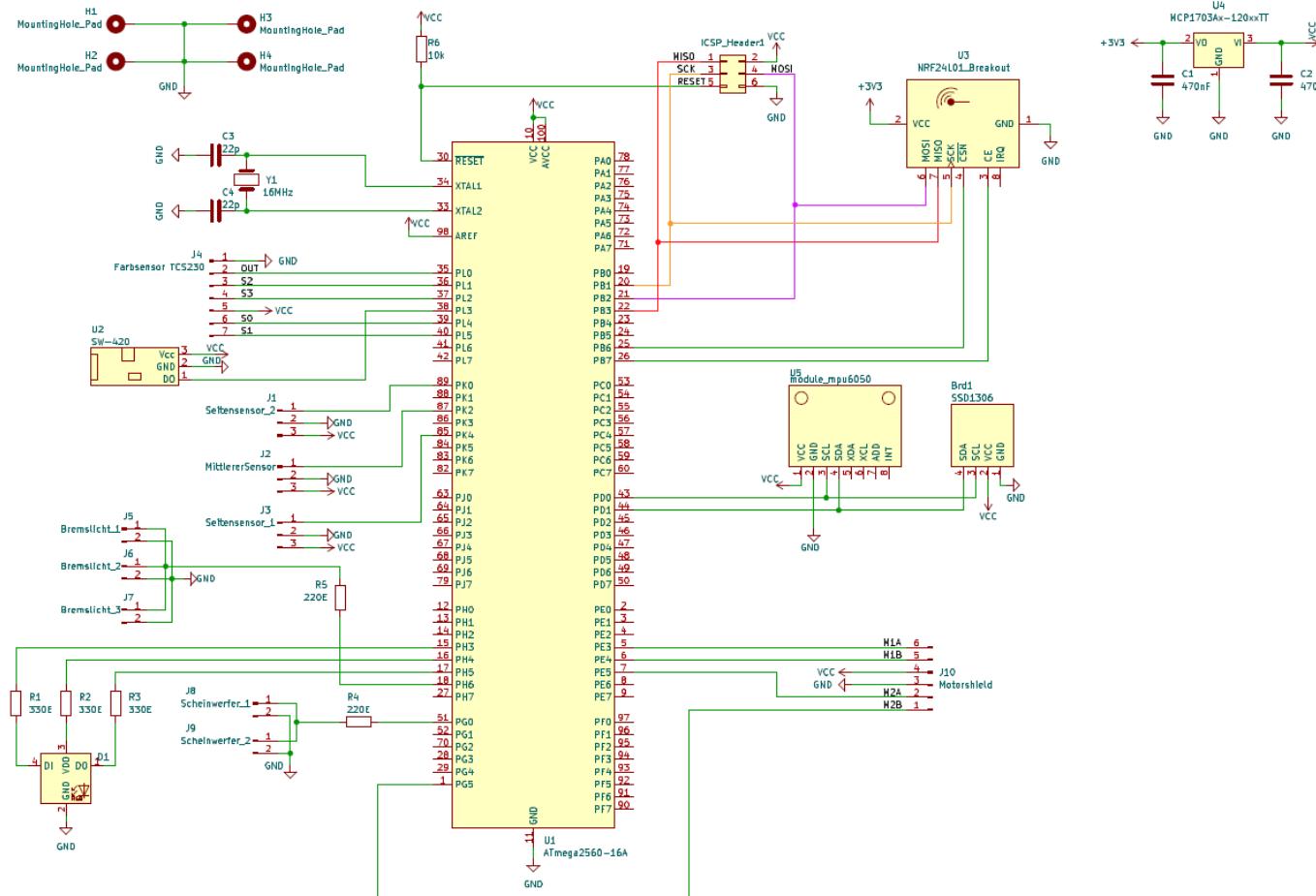
16.1 Roboter Rev A V1.00 Schaltplan



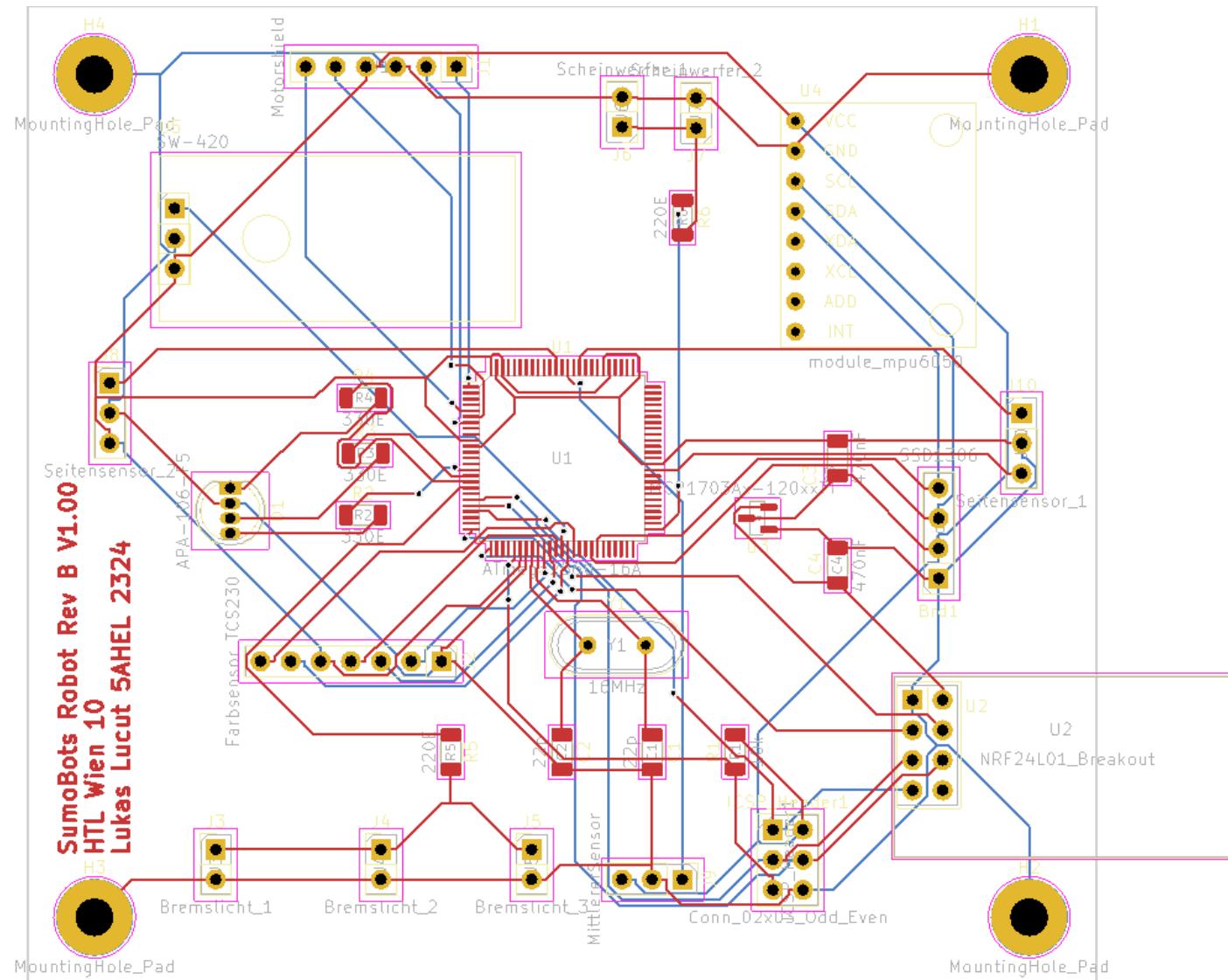
16.2 Roboter Rev A V1.00 PCB



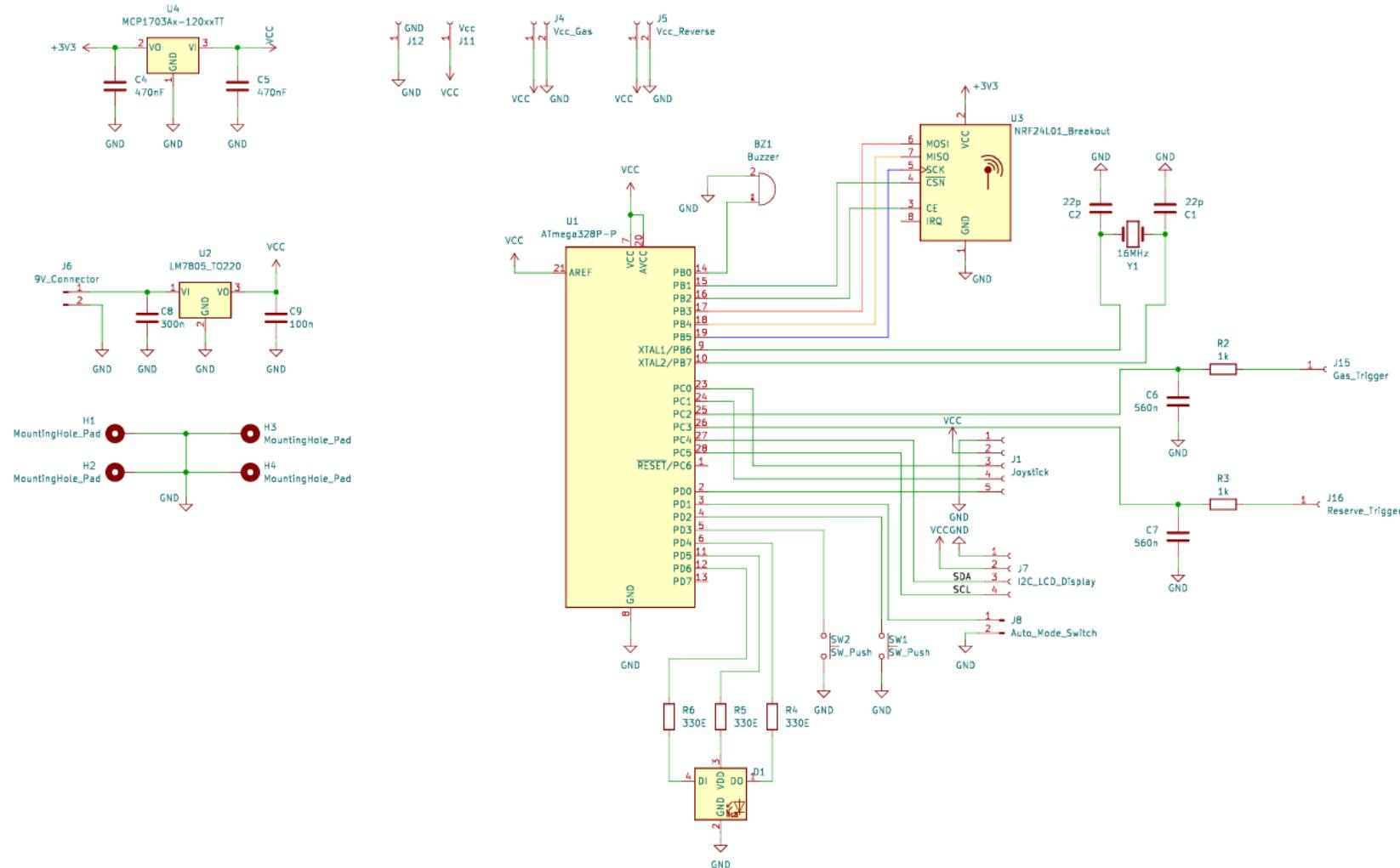
16.3 Roboter Rev B V1.00 Schaltplan



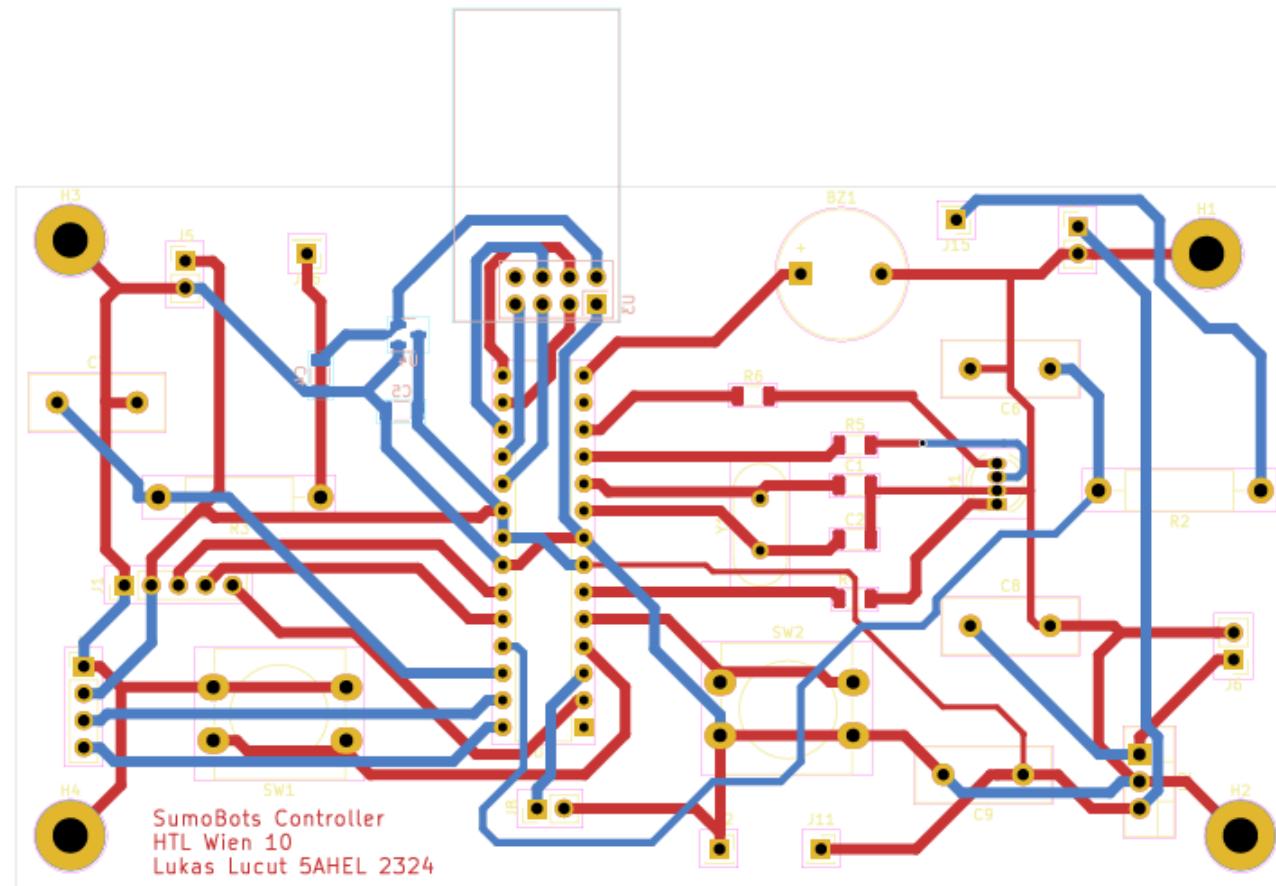
16.4 Roboter Rev B V1.00 PCB



16.5 Controller Rev A V1.00 Schaltplan



16.6 Controller Rev A V1.00 PCB



17 Programm-Codes

17.1 controller.h

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>

#define p_lSchulter A2
#define p_rSchulter A3
#define p_joyX A1
#define p_joyY A0

#define p_red 4
#define p_green 5
#define p_blue 6

#define p_joyButton 0
#define p_buzzer 8

#ifndef controller1
  const byte robotAddress[6] = "c-r01";
#endif

#ifndef controller2
  const byte robotAddress[6] = "c-r02";
#endif

#ifndef controller3
  const byte robotAddress[6] = "c-r03";
#endif

int crntMillis, millisLastPacket, robotConnected, prevMillisLoading, millisBoost;
int loading;
#define BOOST_DURATION 5000

#include <LiquidCrystal_I2C.h>

#define I2C_ADDR      0x27
#define LCD_COLUMNS 16
#define LCD_LINES   2
LiquidCrystal_I2C lcd(I2C_ADDR, LCD_COLUMNS, LCD_LINES);

void lcdSwish();
void rgbWrite(int r, int g, int b);
```

17.2 controller.ino

```
/**  
 *   Auth: Yannick Zickler  
 *  
 *   Desc: Code for both controllers. Reads button-values and sends them to Robot.  
 */  
  
#define controller1  
#include "controller.h"  
  
/** Parameters **/  
  
// CE, CSN  
RF24 radio(10, 9);  
  
void rgbWrite(int r, int g, int b);  
  
/** Initialization */  
int data[] = {0, 0, 0, 0};  
  
void setup() {  
    pinMode(p_red, OUTPUT);  
    pinMode(p_green, OUTPUT);  
    pinMode(p_blue, OUTPUT);  
  
    rgbWrite(1, 0, 0);  
  
    while (!radio.begin()) {}  
    rgbWrite(0, 0, 1);  
  
    radio.openWritingPipe(robotAddress);  
    radio.stopListening();  
    radio.setPALevel(RF24_PA_MIN);  
  
    pinMode(p_lSchulter, INPUT);  
    pinMode(p_rSchulter, INPUT);  
    pinMode(p_joyX, INPUT);  
    pinMode(p_joyY, INPUT);  
    pinMode(p_joyButton, INPUT_PULLUP);  
  
    lcd.init();  
    lcd.clear();  
    lcd.backlight();  
    delay(100);  
    lcdSwish();
```

```
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("Hello, World!");
}

/** Main Loop **/


void loop() {
    crntMillis = millis();
    data[0] = analogRead(p_joyX)/2 - 255;
    data[1] = analogRead(p_joyY)/2 - 255;

    if (crntMillis - millisBoost <= BOOST_DURATION) {
        data[2] = 255 - (analogRead(p_lSchulter) >> 2);
        data[3] = 255 - (analogRead(p_rSchulter) >> 2);
    }

    else {
        // Subtract 80, but keep it above 0
        data[2] = max( (175-(analogRead(p_lSchulter) >> 2)), 0);
        data[3] = max( (175-(analogRead(p_rSchulter) >> 2)), 0);
    }

    if( radio.write(data, sizeof(data)) ) {
        millisLastPacket = crntMillis;
        robotConnected = 1;
    }

    else if (crntMillis - millisLastPacket >= 100) {
        robotConnected = 0;
    }

    if (crntMillis - prevMillisLoading >= 1000) {
        prevMillisLoading = crntMillis;

        if (loading < 16)
            loading++;
        else {
            tone(750, p_buzzer, 500);
        }

        for(int i = 0; i < 16; i++) {
            if (i < loading)
                lcd.setCursor(i, 0);
            lcd.print("#");
            lcd.setCursor(i, 1);
            lcd.print("#");
        }
        else {
            lcd.setCursor(i, 0);
        }
    }
}
```

```
    lcd.print(" ");
    lcd.setCursor(i, 1);
    lcd.print(" ");
}
}

if ( !digitalRead(p_joyButton) && loading == 16 ) {
    millisBoost = crntMillis;
    loading = 0;
}

if (robotConnected)
    rgbWrite(0, 1, 0);
else
    rgbWrite(0, 0, 1);

// Avoid crosstalk with other controllers / robots
delay(10);
}

void rgbWrite(int r, int g, int b) {
    digitalWrite(p_red, r);
    digitalWrite(p_green, g);
    digitalWrite(p_blue, b);
}

void lcdSwish() {
    for (int i = 0; i < 16; i++) {
        lcd.setCursor(i, 0);
        lcd.print("#");
        lcd.setCursor(i, 1);
        lcd.print("#");
        delay(10);
    }
    for (int i = 0; i < 16; i++) {
        lcd.setCursor(i, 0);
        lcd.print(" ");
        lcd.setCursor(i, 1);
        lcd.print(" ");
        delay(10);
    }
}
```

17.3 Roboter.h

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <string.h>

#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>

// Different channels and pipes for every robot
#ifndef robot1
    const byte controllerAddress[6] = "c-r01";
    const byte serverAddress[6] = "1-r-s";
#endif
#ifndef robot2
    const byte controllerAddress[6] = "c-r02";
    const byte serverAddress[6] = "2-r-s";
#endif
#ifndef robot3
    const byte controllerAddress[6] = "c-r03";
    const byte serverAddress[6] = "3-r-s";
#endif

#define p_red 8
#define p_green 7
#define p_blue 6

#define p_lf 5 //Left Forward Pin
#define p_lb 2
#define p_rf 3
#define p_rb 4 //Right Backward Pin

#define p_vibration 46

const int pushDataTimestamp = 100;

//CE, CSN
RF24 radio(13, 12);
void configureRadio();
void drive(int left, int right);
void rgbWrite(int r, int g, int b);
float fmap(float x, float in_min, float in_max, float out_min, float out_max);
int logMsg(char *x, int len);
int sendSensorData();

int speed;
```

```
float turn;
float lm_turn, rm_turn;
int crntMillis, prevMillisData, millisLastPacket;
int lm_ist, rm_ist, lm_soll, rm_soll;
uint8_t controllerConnected, serverConnected;

int inputs[4];
#define JOY_X inputs[0]
#define JOY_Y inputs[1]
#define SHO_L inputs[2]
#define SHO_R inputs[3]

Adafruit_MPU6050 mpu;

struct DataPayload {
    uint8_t zero; // should be 0x00 to signify to server it isn't a log message

    float gyroX;
    float gyroY;
    float gyroZ;

    float accX;
    float accY;
    float accZ;

    float temp;

    uint8_t controllerConnected;
    uint8_t vibration;
};

};
```

17.4 Roboter.ino

```
/***
 *  Auth: Yannick Zickler
 *
 *  Desc: Code for the robot. Handles communication, drives motors serial Interface and LED.
 */

#define robot3
#include "roboter.h"

/** Initialization **/


void setup() {
    // Setup Pins
    pinMode(p_lf, OUTPUT);
    pinMode(p_lb, OUTPUT);
```

```
pinMode(p_rf, OUTPUT);
pinMode(p_rb, OUTPUT);

pinMode(p_red, OUTPUT);
pinMode(p_green, OUTPUT);
pinMode(p_blue, OUTPUT);

pinMode(p_vibration, INPUT);

rgbWrite(1, 0, 0); // red
configureRadio();

if(mpu.begin()) {
    logMsg("Gyro Initialized.", 17);

    mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
    mpu.setGyroRange(MPU6050_RANGE_500_DEG);
    mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);
}
else {
    logMsg("Gyro not connected.", 19);
}

logMsg("Initialization complete.", 24);
}

void loop() {
    crntMillis = millis();

    if (radio.failureDetected) {
        drive(0, 0);

        //blocks until radio configured
        configureRadio();
        logMsg("Radio reconfigured after failure", 32);
    }

    if (radio.available()) {
        radio.read(inputs, sizeof(inputs));
        controllerConnected = 1;
        millisLastPacket = crntMillis;
    }

    else if (crntMillis - millisLastPacket >= 500) {
        controllerConnected = 0;
    }

    if (crntMillis - prevMillisData > pushDataTimestamp) {
        serverConnected = sendSensorData();
```

```
prevMillisData = crntMillis;
}

if (serverConnected && controllerConnected)
    rgbWrite(0, 1, 0); // green
else if (!serverConnected && controllerConnected)
    rgbWrite(0, 0, 1); // blue
else if (serverConnected && !controllerConnected)
    rgbWrite(1, 1, 0); // yellow
else if (!serverConnected && !controllerConnected )
    rgbWrite(0, 1, 1); // cyan

// Calculate Motor speeds
if (SHO_R > SHO_L)
    speed = SHO_R;

else
    speed = -SHO_L;

turn = (float)JOY_X / 256.0;
if (JOY_X < 0) {
    lm_turn = fmap(turn, -1.0, 0.0, -1.0, 1.0);
    rm_turn = 1.0;
}
else if (JOY_X >= 0) {
    lm_turn = 1.0;
    rm_turn = fmap(turn, 0.0, 1.0, 1.0, -1.0);
}

lm_soll = (int)(lm_turn * speed);
rm_soll = (int)(rm_turn * speed);

// Slow down acceleration of motors
if (controllerConnected) {
    if (lm_ist > lm_soll)
        lm_ist -= 1;

    else if (lm_ist < lm_soll)
        lm_ist += 1;

    if (rm_ist > rm_soll)
        rm_ist -= 1;

    else if (rm_ist < rm_soll)
        rm_ist += 1;
}

drive(lm_ist, rm_ist);
}
```

```
    else {
        drive(0, 0);
    }
}

/** Functions **/


void configureRadio() {
    // Do nothing while radio module not connected
    while ( !radio.begin() ) {}

    radio.openReadingPipe(1, controllerAddress);
    radio.openWritingPipe(serverAddress);
    radio.setPALevel(RF24_PA_MIN);
    radio.setRetries(4, 10);
}

void rgbWrite(int r, int g, int b) {
    digitalWrite(p_red, r);
    digitalWrite(p_green, g);
    digitalWrite(p_blue, b);
}

void drive(int left, int right) {
    left = -left;
    if (left > 0) {
        analogWrite(p_lf, left);
        analogWrite(p_lb, 0);
    }
    else {
        analogWrite(p_lb, -left);
        analogWrite(p_lf, 0);
    }

    if (right > 0) {
        analogWrite(p_rf, right);
        analogWrite(p_rb, 0);
    }
    else {
        analogWrite(p_rb, -right);
        analogWrite(p_rf, 0);
    }
}

int logMsg(char *x, int len) {
    radio.stopListening();

    int response = radio.write( x, len );
```

```
radio.startListening();
    return response;
}

int sendSensorData() {
    radio.stopListening();

    sensors_event_t a, g, temp;
    mpu.getEvent(&a, &g, &temp);

    struct DataPayload payload = {
        0x00,
        g.gyro.x, g.gyro.y, g.gyro.z,
        a.acceleration.x, a.acceleration.y, a.acceleration.z,
        temp.temperature,
        controllerConnected,
        digitalRead(p_vibration)
    };
    int response = radio.write(&payload, sizeof(payload));

    radio.startListening();
    return response;
}

float fmap(float x, float in_min, float in_max, float out_min, float out_max)
{
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}
```

17.5 Server.py

```
import RPi.GPIO as gpio
from pyrf24 import *
from collections import deque
import threading, flask, time, struct

### Shared Memory ###
gyroX = [deque(maxlen=100), deque(maxlen=100), deque(maxlen=100)]
gyroY = [deque(maxlen=100), deque(maxlen=100), deque(maxlen=100)]
gyroZ = [deque(maxlen=100), deque(maxlen=100), deque(maxlen=100)]

accX = [deque(maxlen=100), deque(maxlen=100), deque(maxlen=100)]
accY = [deque(maxlen=100), deque(maxlen=100), deque(maxlen=100)]
accZ = [deque(maxlen=100), deque(maxlen=100), deque(maxlen=100)]

vibration = [deque(maxlen=10), deque(maxlen=10), deque(maxlen=10)]
temp = [deque(maxlen=10), deque(maxlen=10), deque(maxlen=10)]
log = [deque(maxlen=10), deque(maxlen=10), deque(maxlen=10)]
```

```
robotConnected = [False, False, False]
controllerConnected = [False, False, False]

### Rf24 Thread ###
def process(i, x):
    if (x[0] == 0x00): #Data
        data = struct.unpack("< B 7f 2B B", x)

        print(i)
        print(data)
        gyroX[i].append(data[1])
        gyroY[i].append(data[2])
        gyroZ[i].append(data[3])

        accX[i].append(data[4])
        accY[i].append(data[5])
        accZ[i].append(data[6])

        temp[i].append(data[7])
        controllerConnected[i] = bool( data[8] )
        vibration[i] = data[9]

    else:
        data = struct.unpack("<32s", x)
        log[i].append( data[0].decode("ASCII").split( u"\x00" )[0] )

def serviceRadio():
    radio = RF24(25, 0)
    while (not radio.begin()):
        pass

    # Match Null-Terminated C "Strings"
    radio.openReadingPipe(0, bytearray( b"1-r-s\x00" ))
    radio.openReadingPipe(1, bytearray( b"2-r-s\x00" ))
    radio.openReadingPipe(2, bytearray( b"3-r-s\x00" ))

    radio.listen = True
    radio.print_details()

    timeNow = 0
    timeLastPacket = [0, 0, 0]
    index = 0

    print("Listening")
    while True:
        timeNow = time.time()

        hasPayload, index = radio.available_pipe()
```

```
if ( hasPayload ):
    timeLastPacket[index] = timeNow
    robotConnected[index] = True

    rec = radio.read()

    try:
        process(index, rec)
    except Exception as e: print(e)

for i in range(3):
    if ( timeNow - timeLastPacket[i] >= 0.250):
        robotConnected[i] = False
        controllerConnected[i] = False

threading.Thread(target=serviceRadio, args=() ).start()

#gyroX = [ [i for i in range(100)], [50 for i in range(100)], [100-i for i in range(100)] ]

### Webserver ###
app = flask.Flask(__name__)

@app.route("/")
def index():
    return flask.render_template("index.html")

@app.route("/about")
def about():
    return flask.render_template("about.html")

@app.route("/data/<name>/<id>")
def data(name,id):
    id = int(id)
    if name == "log": return list(log[id])
    elif name == "temp": return list(temp[id])
    elif name == "gyro": return { "x": list(gyroX[id]), "y": list(gyroY[id]), "z": list(gyroZ[id])}
    elif name == "acc": return { "x": list(accX[id]), "y": list(accY[id]), "z": list(accZ[id])}
    elif name == "status": return { "robotConnected": str( robotConnected[id] ), "controllerConnected": str( controllerConnected[id] ) }

    else: return "Ressource not found", 404

if __name__ == "__main__":
    app.run(host="0.0.0.0")
```

```
flask.url_for("static", filename="style.css")
flask.url_for("static", filename="source.js")
```

17.6 Style.css

```
/* General */
html, body {
    font-family: helvetica;
    width: 100%;
    height: 100%;
    overflow: hidden;
}

html {
    margin: 0;
    padding: 0;
}

p {
    position: relative;
}

textarea {
    position: relative;
    width: 100%;
    height: 50%;
    margin-top: 50px
}

li {
    list-style-type: none;
}

/* Navbar */
#navbar {
    list-style-type: none;
    height: 40px;
    width: 96%;
    margin: 10px;
    padding: 0;
    border: 1px solid #e7e7e7;
    background-color: #f3f3f3;
}

.navelement {
    float: left;
    display: block;
    color: #666;
    text-align: center;
```

```
padding: 8px 17px;
text-decoration: none;
}

.navtext {
color: #2d2d2d;
text-decoration: none;
position: relative;
z-index: 2;
}

.navtext:hover:not(.active) {
background-color: #ddd;
}

.navelementActive {
color: white;
background-color: #04AA6D;
}

.column {
position: absolute;
width: 30%;
height: 100%;
top: 20px;
margin: 30px;
}

/*
.statusList {
margin-top: 20px;
}
*/

/* Specific */
.chartjs-render-monitor {
margin-top: 20px;
margin-bottom: 20px;
}

.box {
position: relative;
display: inline-block;
border-radius: 30%;

width: 13px;
height: 13px;
top: 0px;
left: 0px;
background-color: red;
```

```
}

#log {
    width: 33%;
    height: 800px;
    white-space: pre-line;
}

#chart {
    width: 100%;
    height: 50%;
}

.statusList {
    margin: auto;
}

#htl-logo {
    position: relative;
    width: 15%;
    margin-top: -1px;
    margin-right: -1px;
    float: right;
}
```

17.7 Index.js

```
Chart.defaults.global.elements.point.pointStyle = "line"

const chartAcc0 = new Chart("chartAcc0", {
    type: "line",
    data: {
        labels: Array.from(Array(100).keys()),
        datasets: [
            {
                label: "Acc. X",
                borderColor: "red",
                data: [],
                fill: false
            },
            {
                label: "Acc. Y",
                borderColor: "green",
                data: [],
                fill: false
            },
            {
                label: "Acc. Z",
                borderColor: "blue",
                data: []
            }
        ]
    }
})
```

```
        fill: false
    }
}],
options: {
    animation: {duration: 50}
}
});

const chartAcc1 = new Chart("chartAcc1", {
    type: "line",
    data: {
        labels: Array.from(Array(100).keys()),
        datasets: [{
            label: "Acc. X",
            borderColor: "red",
            data: [],
            fill: false
        },
        {
            label: "Acc. Y",
            borderColor: "green",
            data: [],
            fill: false
        },
        {
            label: "Acc. Z",
            borderColor: "blue",
            data: [],
            fill: false
        }
    ],
    options: {
        animation: {duration: 50}
    }
});

const chartAcc2 = new Chart("chartAcc2", {
    type: "line",
    data: {
        labels: Array.from(Array(100).keys()),
        datasets: [{
            label: "Acc. X",
            borderColor: "red",
            data: [],
            fill: false
        },
        {
            label: "Acc. Y",
            borderColor: "green",
            data: [],
            fill: false
        }
    ]
});
```

```
        fill: false
    },
    {
        label: "Acc. Z",
        borderColor: "blue",
        data: [],
        fill: false
    }
],
options: {
    animation: {duration: 50}
}
});

const chartGyro0 = new Chart("chartGyro0", {
    type: "line",
    data: {
        labels: Array.from(Array(100).keys()),
        datasets: [
            {
                label: "Gyro X",
                borderColor: "red",
                data: [],
                fill: false
            },
            {
                label: "Gyro Y",
                borderColor: "green",
                data: [],
                fill: false
            },
            {
                label: "Gyro Z",
                borderColor: "blue",
                data: [],
                fill: false
            }
        ]
    },
    options: {
        animation: {duration: 50}
    }
});

const chartGyro1 = new Chart("chartGyro1", {
    type: "line",
    data: {
        labels: Array.from(Array(100).keys()),
        datasets: [
            {
                label: "Gyro X",
                borderColor: "red",
                data: []
            }
        ]
    }
});
```

```
        fill: false
    },
    {
        label: "Gyro Y",
        borderColor: "green",
        data: [],
        fill: false
    },
    {
        label: "Gyro Z",
        borderColor: "blue",
        data: [],
        fill: false
    }
],
options: {
    animation: {duration: 50}
}
});

const chartGyro2 = new Chart("chartGyro2", {
    type: "line",
    data: {
        labels: Array.from(Array(100).keys()),
        datasets: [
            {
                label: "Gyro X",
                borderColor: "red",
                data: [],
                fill: false
            },
            {
                label: "Gyro Y",
                borderColor: "green",
                data: [],
                fill: false
            },
            {
                label: "Gyro Z",
                borderColor: "blue",
                data: [],
                fill: false
            }
        ]
    },
    options: {
        animation: {duration: 50}
    }
});

function sendRequest(url, callback) {
    let req = new XMLHttpRequest();
```

```
req.responseType = 'json';
req.open("GET", url);
req.send();

req.onreadystatechange = function() {
    if(req.readyState == 4) {
        callback(req)
    }
};

function getData() {

    if (index == 0) {
        sendRequest("data/gyro/0", function(req) {
            chartGyro0["data"]["datasets"][0]["data"] = req.response["x"]
            chartGyro0["data"]["datasets"][1]["data"] = req.response["y"]
            chartGyro0["data"]["datasets"][2]["data"] = req.response["z"]
            chartGyro0.update()
        });
    }

    else if (index == 1) {
        sendRequest("data/acc/0", function(req) {
            chartAcc0["data"]["datasets"][0]["data"] = req.response["x"]
            chartAcc0["data"]["datasets"][1]["data"] = req.response["y"]
            chartAcc0["data"]["datasets"][2]["data"] = req.response["z"]
            chartAcc0.update()
        });
    }

    else if (index == 2) {
        sendRequest("data/status/0", function(req) {
            if (req.response["robotConnected"] == "True") {
                document.getElementById("robotConnected0").innerHTML = "<div class='box' style='background-color: green'></div> &nbsp; Robot Connected"
            } else {
                document.getElementById("robotConnected0").innerHTML = "<div class='box'></div> &nbsp; Robot Disconnected"
            }

            if (req.response["controllerConnected"] == "True") {
                document.getElementById("controllerConnected0").innerHTML = "<div class='box' style='background-color: green'></div> &nbsp; Controller Connected"
            } else {
                document.getElementById("controllerConnected0").innerHTML = "<div class='box'></div> &nbsp; Controller Disconnected"
            }
        });
    }
}
```

```
}

else if (index == 3) {
    sendRequest("data/gyro/1", function(req) {
        chartGyro1["data"]["datasets"][1]["data"] = req.response["y"]
        chartGyro1["data"]["datasets"][0]["data"] = req.response["x"]
        chartGyro1["data"]["datasets"][2]["data"] = req.response["z"]
        chartGyro1.update()
    });
}

else if (index == 4) {
    sendRequest("data/acc/1", function(req) {
        chartAcc1["data"]["datasets"][0]["data"] = req.response["x"]
        chartAcc1["data"]["datasets"][1]["data"] = req.response["y"]
        chartAcc1["data"]["datasets"][2]["data"] = req.response["z"]
        chartAcc1.update()
    });
}

else if (index == 5) {
    sendRequest("data/status/1", function(req) {
        if (req.response["robotConnected"] == "True") {
            document.getElementById("robotConnected1").innerHTML = "<div class='box' style='background-color: green'></div> &nbsp; Robot Connected"
        } else {
            document.getElementById("robotConnected1").innerHTML = "<div class='box'></div> &nbsp; Robot Disconnected"
        }

        if (req.response["controllerConnected"] == "True") {
            document.getElementById("controllerConnected1").innerHTML = "<div class='box' style='background-color: green'></div> &nbsp; Controller Connected"
        } else {
            document.getElementById("controllerConnected1").innerHTML = "<div class='box'></div> &nbsp; Controller Disconnected"
        }
    });
}

else if (index == 6) {
    sendRequest("data/gyro/2", function(req) {
        chartGyro2["data"]["datasets"][1]["data"] = req.response["y"]
        chartGyro2["data"]["datasets"][0]["data"] = req.response["x"]
        chartGyro2["data"]["datasets"][2]["data"] = req.response["z"]
        chartGyro2.update()
    });
}
```

```

else if (index == 7) {
    sendRequest("data/acc/2", function(req) {
        chartAcc2["data"]["datasets"][0]["data"] = req.response["x"]
        chartAcc2["data"]["datasets"][1]["data"] = req.response["y"]
        chartAcc2["data"]["datasets"][2]["data"] = req.response["z"]
        chartAcc2.update()
    });
}

else if (index == 8) {
    sendRequest("data/status/2", function(req) {
        if (req.response["robotConnected"] == "True") {
            document.getElementById("robotConnected2").innerHTML = "<div class='box' style='background-color: green'></div> &nbsp; Robot Connected"
        } else {
            document.getElementById("robotConnected2").innerHTML = "<div class='box'></div> &nbsp; Robot Disconnected"
        }

        if (req.response["controllerConnected"] == "True") {
            document.getElementById("controllerConnected2").innerHTML = "<div class='box' style='background-color: green'></div> &nbsp; Controller Connected"
        } else {
            document.getElementById("controllerConnected2").innerHTML = "<div class='box'></div> &nbsp; Controller Disconnected"
        }
    });
}

index++;
if (index == 9) {
    index = 0
}
}

let index= 0;

setInterval(getData, 100/9);

```

17.8 index.html

```

<!DOCTYPE html>

<html>
    <head>
        <title> SumoBots </title>
        <script src = "https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.9.4/Chart.js"> </script>

```

```
<link rel="stylesheet" href="/static/style.css">
<link rel="icon" type="image/png" href="/static/images/favicon.png">
</link>
</head>

<body>
    <ul id="navbar">
        <li class="navelement navelementActive"> <a class="navtext" href="">
Home </a> </li>
        <li class="navelement"> <a class="navtext" href="/about"> About </a>
</li>
        <li>  </img>
</li>
    </ul>

    <div id="column0" class="column">
        <p style="text-align: center"> Roboter 0 </p>
        <canvas id="chartAcc0"> </canvas>
        <canvas id="chartGyro0"> </canvas>

        <ul id="statusList0">
            <li id="robotConnected0"> </li>
            <li id="controllerConnected0"> </li>
        </ul>
    </div>

    <div id="column1" style="left: 33%" class="column">
        <p style="text-align: center"> Roboter 1 </p>
        <canvas id="chartAcc1"> </canvas>
        <canvas id="chartGyro1"> </canvas>

        <ul id="statusList1">
            <li id="robotConnected1"> </li>
            <li id="controllerConnected1"> </li>
        </ul>
    </div>

    <div id="column2" style="left: 66%" class="column">
        <p style="text-align: center"> Roboter 2 </p>
        <canvas id="chartAcc2"> </canvas>
        <canvas id="chartGyro2"> </canvas>
        <ul id="statusList2">
            <li id="robotConnected2"> </li>
            <li id="controllerConnected2"> </li>
        </ul>
    </ul>

</div>
```

```
<script src = "static/index.js"> </script>

</body>
</html>
```

17.9 about.html

```
<html>
  <head>
    <title> SumoBots </title>
    <link rel="stylesheet" href="/static/style.css">
    <link rel="icon" type="image/png" href="/favicon.png"> </link>
  </head>
  <body>

    <ul id="navbar">
      <li class="navelement"> <a class="navtext" href="{{ url_for('index') }}> Home </a> </li>
      <li class="navelement navelementActive"> <a class="navtext" href="{{ url_for('about') }}> About </a> </li>
    </ul>

    <p>
      Das ist die Website des Sumobots, der Diplomarbeit die in der HTL
      Wien 10 von
      <ul>
        <li> 1. Lukas Lucut </li>
        <li> 2. Yannick Zickler </li>
        <li> 3. Bastian Eismann </li>
      </ul>
      entwickelt wurde. Eine Genauere Dokumentation ist <a href
      ="https://www.github.com/RoteRuebe/diplomarbeit"> auf Github zu finden. </a>
    </p>
  </body>
```