
# Diplomarbeit

SwarmBots

Schuljahr 2024/25

Betreuer: **Dipl.-Ing Erich Erker**

| | | |
|---|---|---|
| Gruppenmitglieder: | **Arthur Burjak** | 5AHEL |
| : | **Leander Gastgeber** | 5AHEL |
| : | **Jones Soliman** | 5AHEL |
| : | **Mihael Stojkovic** | 5AHEL |

# Erklärung über die eigenständige Verfassung der Diplomarbeit

Wir, die Herren Arthur Burjak, Leander Gastgeber, Jones Soliman und Mihael Stojkovic, Schüler der Klasse 5AHEL der Höheren Technischen Bundeslehranstalt Wien 10, erklären hiermit an Eides statt, dass wir die vorliegende Diplomarbeit selbständig 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 11.04.2025

Verfasser:

| | | |
|---|---|---|
| Arthur Burjak | 5AHEL | U:_____ |
| Leander Gastgeber | 5AHEL | U:_____ |
| Jones Soliman | 5AHEL | U:_____ |
| Mihael Stojkovic | 5AHEL | U:_____ |

# Inhaltsverzeichnis

# 1 Zusammenfassung

Ziel dieser Diplomarbeit ist es, drei Roboter zu entwickeln, welche kooperativ die Umgebung erkunden können. Hierbei ist ein Roboter ("Guide") mit einem LiDAR-Sensor ausgestattet, welcher Entfernungsmessungen durchführt. Die anderen beiden Roboter (getauft "Tamerlan" und "Bambi") sollen komplett "blind" sein. Koordiniert wird das ganze über einen zentralen Server, welcher die gesammelten Daten zusätzlich über ein Webinterface darstellt. Als zusätzliche Aufgabe sollen sich die Roboter auf nur einer Achse balancieren, da wir Kits für balancierende Roboter verwenden, welche wir für unsere Zwecke modifiziert haben. Die Software basiert auf den Robotern, die wir letztes Jahr im Zuge der Projektwoche als Vorbereitung auf die Diplomarbeit gebaut haben.

## 1.1 Abstract (English)

Goal of this diploma thesis is to develop three robots which can explore the environment cooperatively. One robot ("Guide") is equipped with a LiDAR sensor, which carries out distance measurements. The other two robots (named "Tamerlan" and "Bambi") are to be completely "blind". The diploma thesis is coordinated via a central server, which also displays the collected data via a web interface. As an additional task, the robots should balance themselves on only one axis, as we use kits for balancing robots, which we have modified for our purposes. The software is based on the robots that we built last year during the project week in preparation for the diploma thesis.

# 2 Vorgeschichte

Im Zuge der Projektwoche im Schuljahr 2023/24 haben wir bereits begonnen, einen ersten Prototypen unseres SwarmBots-Systems zu bauen. Dieser Prototyp bestand aus nur zwei Robotern, welche auf Basis eines fertigen Fahrgestells zu sehr wackligen Gefährten wurden. Sehr viel Autonomie hatten die früheren Roboter auch noch nicht, der LiDAR war noch nicht funktionstüchtig, stattdessen wurden die Roboter mittels Videospiel-Controller ferngesteuert. Diese Fernsteuerung wurde aber auch schon damals über eine Websocket-Verbindung implementiert. Außerdem wurde während der Projektwoche der Code für die ESP32-CAMs fast komplett fertiggestellt, diese verwenden jetzt immer noch größtenteils das gleiche Programm. Trotz der Schwächen des damaligen Systems wurde unserem Projekt der erste Preis in der Kategorie "Experten" verliehen.

# 3 Technischer Überblick

## 3.1 Unterstützende Programme

### 3.1.1 Git

Zur Versionskontrolle der Software und der Diplomarbeit selber (siehe Abschnitt 3.1.5) haben wir Git eingesetzt. Das Repository ist auf GitHub, aber aktuell privat.

### 3.1.2 Syncthing

Zum Teilen von anderen Dateien (Weekly Reports, Zeiterfassung, etc.) zwischen Personen und Geräten haben wir ein FOSS[1] Programm names Syncthing [1] genutzt. Syncthing synchronisiert Dateien in einem Ordner dezentralisiert mittels Peer-to-Peer Verbindungen zwischen den Geräten. Dadurch ist das Teilen von Dateien nicht von zentralisierter proprietärer Infrastruktur abhängig, kann aber durch "zentrale" (und gut erreichbare) Server unterstützt werden. Weiters stellt Syncthing eine rudimentäre Art der Versionskontrolle dar, da es die Möglichkeit gibt, bei Änderungen der Dateien eine alte Version der Datei für eine gewisse Zeit zu behalten.

Da Syncthing dezentralisiert ist, und es somit keinen SPOT[2] gibt, kann es (selten, aber doch) zu Konflikten der Dateiversionen kommen (z.B. wenn zwei Personen die selbe Datei gleichzeitig bearbeiten). Bei solchen Konflikten erstellt Syncthing eine Kopie von einer der Versionen der Datei, und der Nutzer muss selber entscheiden, welche Version die "richtige" ist.

### 3.1.3 Visual Studio Code

Zur Bearbeitung der Quellcodes und der Diplomarbeit verwenden wir Visual Studio Code (auch VSC oder VSCode) mit einigen Erweiterungen. Nützlich ist für uns auch die Git-Integration zur Versionskontrolle.

| Erweiterung | Verwendungszweck |
|---:|---|
| Python | Programmierung des Backends |
| PlatformIO IDE | Programmierung der Roboter |
| vscode-proto3 | Syntax-Highlighting für Protocol Buffers |
| LaTeX Workshop | LaTeX Unterstützung für VSC |
| Code Spell Checker | Rechtschreibprüfung |

Tabelle 1: Verwendete VSCode-Erweiterungen

[1] Free and Open Source Software
[2] Single Point of Truth

### 3.1.4 PlatformIO

Für die Programmierung der Mikrocontroller verwenden wir PlatformIO [2] in Verbindung mit dem PlatformIO-Plugin für VSCode. PlatformIO ist eine Alternative zur Arduino-IDE, welche mithilfe von Plugins in viele IDEs wie z.B. VSCode oder CLion integriert werden kann. Alternativ kann man PlatformIO auch über die Kommandozeile bedienen. Vorteile von PlatformIO gegenüber der Arduino-IDE sind u.A. schnelleres Kompilieren, ordentliche Auto-Vervollständigung, statische Code-Analyse (Linting), und ein schön geregeltes System für externe Bibliotheken.

### 3.1.5 LaTeX

Anstatt von WYSIWYG[3]-Editoren wie Microsoft Word oder LibreOffice Writer verwenden wir zum Erstellen dieser Diplomarbeit ein plattformunabhängiges Plaintext-Format namens LaTeX. Eine LaTeX-Datei ist (ähnlich wie Markdown) einfach nur eine Textdatei mit zusätzlichen Befehlen. Deshalb können wir `.tex`-Dateien sehr gut mit Versionskontrollsoftware wie Git verwenden und somit genau Änderungen rückverfolgen und gegebenenfalls zurücksetzen. Weiters können wir mit LaTeX unsere Arbeit in mehrere Dateien aufteilen, was das Ganze um einiges übersichtlicher macht. Der größte Vorteil von LaTeX ist aber wohl, dass die Formatierung einheitlich für den Nutzer übernommen wird, sodass sich dieser besser auf den Inhalt konzentrieren kann. Wenn man aber dann doch mal selber etwas anders formatieren will, ist das mit LaTeX durch die Verwendung von Textbefehlen aber immer noch schneller als in MS Word zur Maus zu greifen und sich durch GUIs durchzuklicken.

## 3.2 Funktionsbeschreibung

### 3.2.1 Guide

Der Guide hat die Aufgabe, mit dem nachgerüsteten LiDAR-Sensor die Umgebung nach Hindernissen abzusuchen. Hierbei ist allerdings zu beachten, dass der Guide die Datenverarbeitung nicht selbständig durchführt, sondern die rohen Sensordaten einfach an das Backend weiterleitet. In diesem Kontext sind Tamerlan und Bambi also nichts anderes als Hindernisse.

### 3.2.2 Tamerlan & Bambi

Tamerlan und Bambi sind "blind" im dem Sinne, dass sie über keine Sensoren zur Hinderniserkennung verfügen. Sie sind komplett von Anweisungen des Backends abhängig.

[3] What You See Is What You Get

### 3.2.3 Backend

Das Backend ist das "Gehirn" des Projekts. Es empfangt die LiDAR-Daten vom Guide, verarbeitet sie, und sendet Befehle an die Roboter. Die LiDAR-Daten und die Ergebnisse der Verarbeitung werden auch über eine Websocket-Verbindung ans Frontend weitergeleitet. Außerdem empfängt es die Sensordaten der Gyroskope und Accelerometer der Roboter, und auch ans Frontend weiter. Gegebenenfalls empfängt das Backend auch Befehle vom Frontend, und leitet diese an die Roboter weiter.

### 3.2.4 Frontend

Das Frontend empfängt Sensordaten vom Backend und stellt diese dar. LiDAR-Daten werden auf einer Karte dargestellt, auf der auch die Ergebnisse der Hinderniserkennung am Backend zu sehen ist. Die Daten der Gyroskope und Accelerometer werden in einem sich laufend aktualisierendem X/Y Diagramm dargestellt. Zusätzlich gibt es im Frontend die Möglichkeit für Nutzer, einen oder mehrere Roboter auf manuelle Steuerung umzuschalten, und diese(n) dann fernzusteuern. Das Frontend ist eigentlich zum autonomen Betrieb nicht nötig, es wird nur für Debugging-Zwecke und menschliche Intervention benötigt.

### 3.2.5 Kommunikationswege

TODO bessere Grafik(!!!) & Beschreibung

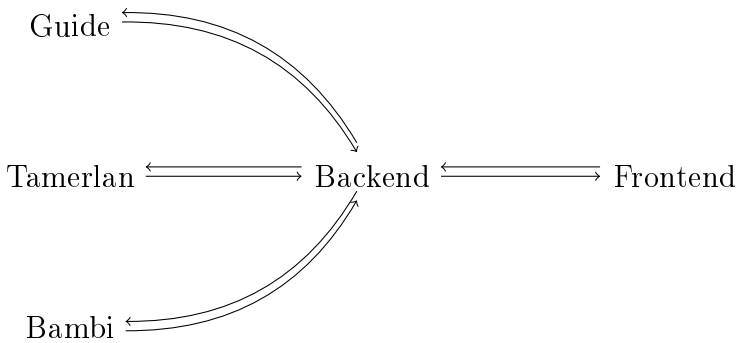


Abbildung 1: Kommunikationswege

## 3.3 Protocol Buffers

Protocol Buffers [3] ("protobufs") sind ein binäres Übertragungsformat, welches von von Google entwickelt und veröffentlicht wurde. Gegenüber Datenformaten wie JSON und XML gibt es drei wesentliche Vorteile:

1. Da Protocol Buffers auf Binärdaten anstatt von Text basieren, ist die Übertragung viel effizienter [4]. Insbesondere bei der Verwendung mit Mikrocontrollern ist das ein enormer Vorteil.

2. Bei Protocol Buffers gibt es explizit definierte Datenstrukturen. Diese Datenstrukturen sind (bei korrekter Verwendung) mit älteren Versionen rückwärtskompatibel.

3. Für wie Verwendung mit unterschiedlichen Programmiersprachen kann (und soll) man aus protobuf-Definitionen Wrapper-Bibliotheken generieren. Diese Wrapper-Bibliotheken können ohne weiteren Aufwand direkt verwendet werden, um auf die Datenstrukturen zuzugreifen.

**Effizienz** Wie schon oben erwähnt erreichen Protocol Buffers, insbesondere bei kleinen Nachrichten, einen viel kleineren Overhead als textbasierte Formate wie JSON oder XML. TODO mehr infos; Tatsächlicher Vergleich mit Zahlen.

**Datenstrukturen** TODO

**Wrapper-Bibliotheken** TODO

## 3.4 Sensoren

### 3.4.1 LiDAR

### 3.4.2 Gyroskop

### 3.4.3 Dreh-Encoder

# 4 Hardware

## 4.1 Elegoo Tumbller Kit

Um den Hardwareaufbau so einfach wie möglich zu gestalten, entschieden wir uns dafür, fertig entwickelte Kits online zu bestellen und dann zu modifizieren. Die Wahl des Kits fiel letztendlich auf den "Tumbller" von Elegoo (Siehe Abbildung 2). Der Tumbller ist ein zweirädriger Roboter, welcher auf einer Ache balanciert. Zur Kontrolle des unmodifizierten Kits gibt es eine Smartphone-App, welche die Roboter über Bluetooth fernsteuern kann. Da wir die Roboter über WLAN steuern wollten, und die Tumbller-Kits standardmäßig nur eine Bluetooth-Erweiterung eingebaut haben, haben wir die mitgelieferten Arduino Nano durch ESP32-Boards im Arduino Nano-Format ersetzt.

Abbildung 2: Rendering des Elegoo Tumbller

## 4.2 Guide

Die Aufgabe von *Guide* ist es, mithilfe eines LiDAR-Sensors (Siehe Kapitel 3.4.1) die Umgebung nach Hindernissen und den anderen Robotern abzusuchen. Die vom LiDAR gesammelten Abstandsdaten werden über eine TCP/IP Websocket-Verbindung an einen

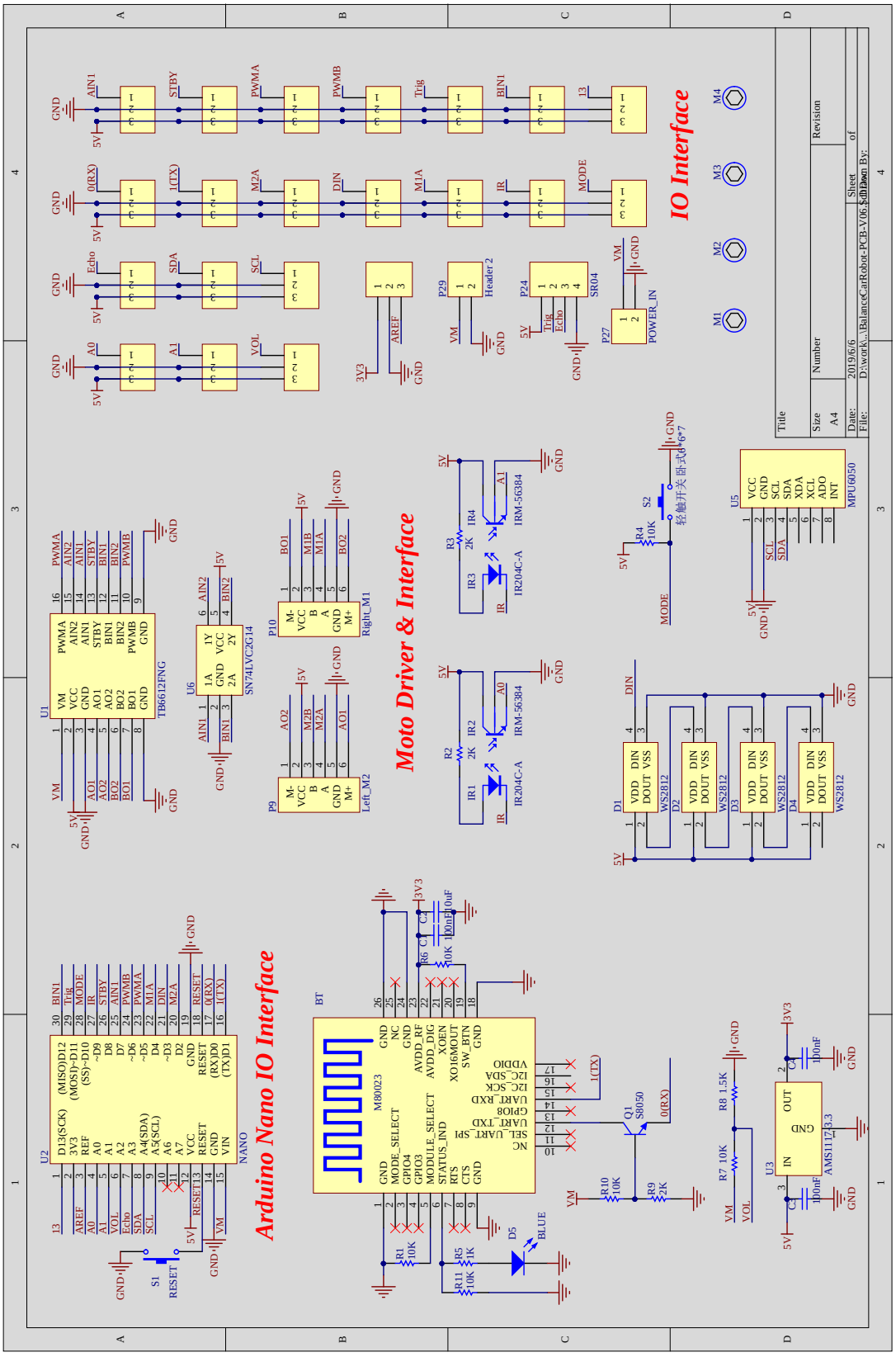


Abbildung 3: Originaler Schaltplan des Tumbllers (TODO: neu zeichnen)

zentralen Server weitergegeben, welcher diese weiter verarbeitet. Um den LiDAR-Sensor zu montieren, haben wir das Fahrgestell leicht modifiziert und die oberste Ebene (mit dem Akku) erhöht, um Platz für Schrauben zu schaffen.

## 4.3 Tamerlan & Bambi

# 5 Software - Roboter

Was die Ansteuerung der einzelnen Roboter angeht, folgen wir so gut wie möglich dem DRY-Prinzip ("Don't repeat yourself"), um Redundanzen zu vermeiden und dadurch den Wartungsaufwand möglichst gering zu halten.

## 5.1 Kameras

Wir verwenden jeweils ein ESP32-CAM AI-Thinker Modul zur erweiterten Fernüberwachung der Roboter. Dieses verbindet sich über WLAN mit dem `IoT`-Netzwerk und bietet über HTTP einen MJPEG-Videostream an. Die drei Videostreams (einer pro Roboter) werden dann im Web-Interface zusätzlich zu den LiDAR-Umgebungsdaten angezeigt, um das räumliche Vorstellungsvermögen der Benutzer zu unterstützen. Die Videodaten werden also nicht automatisch verarbeitet und zur autonomen Steuerung verwendet, sondern dienen nur als weiterer Input für die Bediener.

Abbildung 4: Erstes empfangenes Bild der ESP32-CAM

## 5.2 Core-Bibliothek

Die Core-Bibliothek bildet eine weiter Abstrahierungsschicht zu den Funktionalitäten der Roboter-Hardware. In unserem Modell stellt die Arduino-Bibliothek die erste Abstrahierungsschicht dar, während die Code-Bibliothek diese weiter vereinfacht. Durch diese Abstrahierung werden die Programme für die einzelnen Roboter um ein Vielfaches vereinfacht und dementsprechend übersichtlicher. Wir können also die hier definierte Logik für jeden Roboter wiederverwenden, anstatt drei mal fast genau das Gleiche

zu programmieren. Die Core-Bibliothek übernimmt alles von der WLAN-Verbindung, über die Fernsteuerung per WebSockets, bis hin zum Umschalten der einzelnen Pins zur Ansteuerung der Motortreiber. Die Programmierung der einzelnen Roboter beschränkt sich also darauf, die einzelnen Komponenten zu konfigurieren und miteinander zu verbinden. Da Herr Gastgeber während der Projektwoche 2023/24 (Siehe Abschnitt 2) bereits viel Zeit darin investiert hat, die Bibliothek so modular und wiederverwendbar wie möglich zu gestalten, konnten wir diese mit nur wenigen Modifikationen für unsere Diplomarbeit nutzen, obwohl wir einen ganz anderen Hardwareaufbau verwenden.

## 5.3 Guide

## 5.4 Tamerlan

## 5.5 Bambi

# 6 Software - Backend

## 6.1 Datenverwaltung

## 6.2 Steuerung der Roboter

# 7 Software - Frontend

## 7.1 LiDAR-Karte

## 7.2 Fernüberwachung per Kamera

## 7.3 Fernsteuerung

## 7.4 Anzeigen der Sensordaten

# 8 Probleme

## 8.1 Docker

Am Laptop von Herr Gastgeber war es am Beginn des Schuljahres nicht möglich, jegliche Geräte im Netzwerk der HTL zu erreichen. Grund dafür war eine Software namens Docker, welche außerhalb dieses Projekts zur Containerisierung von anderen Anwendungen dient. Docker war standardmäßig so konfiguriert, dass es containerspezifische Subnets im IPv4-Adressbereich `172.117.0.0/16` erstellt. Diese Subnets hatten dann am Laptop eine höhere Priorität als das Schulnetzwerk, was dafür sorgte, dass das weltweite Internet noch erreichbar war, nicht aber das lokale Schulnetzwerk. Um Docker einen anderen IP-Adressbereich zuzuweisen, wurde der Docker-Daemon mithilfe der Datei `/etc/docker/daemon.json` wie in Listing 1 konfiguriert.

```
{
      "default-address-pools":
      [
        {"base":"10.10.0.0/16","size":24}
      ]
}
```

Listing 1: Konfiguration für den Docker-Daemon

## 8.2 Modifikationen am Tumbller

Als wir das Projekt geplant haben, war die Idee, einfach ein fertiges Kit ein bisschen zu modifizieren, fast schon zu schön um wahr zu sein. Und das war es dann auch. Bei den Hardware-Modifikationen am Tumbller (siehe Kapitel 4.1) gab es zwei relativ große Probleme:

### 8.2.1 Falsche Betriebsspannungen

Beim Austausch des mitgelieferten Arduino Nano mit einem ESP32 im Nano-Format haben wir einen wichtigen Faktor übersehen: Der Arduino Nano hat eine Betriebsspannung von 5V, während der ESP32 mit 3.3V arbeitet. Glücklicherweise funktionieren die meisten Komponenten des Elegoo Tumbllers auch mit 3.3V. Die einzigen Bauteile, welche eine höhere Spannung benötigen, sind die auf der Platine angelöteten farbigen Leuchtdioden. Um nicht das ganze Projekt von Grund auf neu aufbauen zu müssen, haben wir uns entschieden, dieses kleine optische Detail fürs Erste auszulassen. In der originalen Beschaltung (siehe Abbildung 3) wurde das Bluetooth-Modul mithilfe des Spannungswandlers `U3` mit 3.3V versorgt. Diesen Spannungswandler haben wir ausgebaut und mittels einer Lötbrücke $V_{in}$ mit $V_{out}$ verbunden. Allerdings erwies sich das Bluetooth Modul später als problematisch (siehe Abschnitt 8.3), weshalb die Überbrückung eigentlich nicht notwendig ist.

Außerdem war es beim Guide-Roboter aufgrund des Wechsels auf 3.3V nicht mehr möglich, den LiDAR-Sensor direkt mit dem Spannungswandler des Mikrocontroller-Boards zu versorgen. Deshalb haben wir für den Guide einen zusätzlichen Step-Down-Konverter eingebaut, welcher die Versorgungsspannung des Akkus auf 5V für den LiDAR herunterregelt.

## 8.3 Blockierte UART-Schnittstelle

Als das Problem der inkompatiblen Spannungen gelöst, und die erste Version des Programmes zum Testen der einzelnen Komponenten geschrieben war, sind wir auch schon auf das nächste nennenswerte Problem gestoßen: Das externe Bluetooth-Modul, was bereits auf der Platine verlötet war, belegt die UART-Schnittstelle des eingebauten ESP32. Das führte dazu, dass der ESP32 nur neu programmiert werden konnte, wenn das ESP32-Devboard aus dem Roboter ausgebaut war. Außerdem wurde dadurch das Debuggen mittels UART über USB unmöglich gemacht.
Da wir die Roboter über WLAN steuern, und der ESP32 auch ohne externe Erweiterungen bereits sowohl über WLAN- als auch Bluetooth-Kapazitäten verfügt, haben wir entschlossen, die mitgelieferte Platine weiter zu modifizieren, indem wir das Bluetooth-Modul entfernen. Außerdem haben wir den Transistor `Q1` und den Spannungsteiler, welcher aus `R9` und `R10` besteht entfernt, um sicherzustellen, das die UART-Verbindung keinesfalls beeinflusst wird (siehe Original-Schaltplan in Abbildung 3).

# Abbildungsverzeichnis



# Tabellenverzeichnis



# Literatur

[1] Syncthing Foundation. *Syncthing*. URL: `https://syncthing.net/` (besucht am 24. 12. 2024).

[2] PlatformIO Labs. *Your Gateway to Embedded Software Development Excellence · PlatformIO*. URL: `https://platformio.org/` (besucht am 25. 12. 2024).

[3] Google LLC. *Protocol Buffers Documentation*. URL: `https://protobuf.dev/` (besucht am 24. 12. 2024).

[4] Srđan Popić u. a. „Performance evaluation of using Protocol Buffers in the Internet of Things communication“. In: *2016 International Conference on Smart Systems and Technologies (SST)*. 2016, S. 261–265. DOI: `10.1109/SST.2016.7765670`.