

Dokumentation Robocar

Mobiles Robotikprojekt I

Lars Schwarz

2024-01-14

Table of contents

Einführung	3
Übersicht	3
Handbuch	4
Benötigte Komponenten	4
Hardware	4
GUI	5
Spass haben	8
Hardware	9
Elektronik	11
Blockschaltbild	13
Schema & PCB-Design	14
Messungen BUC-Konverter	15
Wirkungsgrad	15
Spannungsrippel	15
Software	16
Softwareaufbau	16
Flussdiagramm	17
Funktionalität	17
Modulbeschreibungen	19
ToDo-Software	19
Stand der Dinge	19
Fernsteuerung	19
Implementation X-Box-Kontroller	19
Kommunikation zwischen den Mikrocontrollern	19
ToF-Kamera	19
Visualisierung	19
Parameterhandler	19
Bekannte Probleme	20
Reset des Robocars	20
Verbindungsprobleme mit GUI	20

List of Figures

1	Robocar	3
2	Kabelquetschung	5
3	GUI Robocar	6
4	Betriebsmodi	8
5	Fusion-Export	9
6	VFS-Beleuchtung	10

7	ToF-Spot Halterung	11
8	PCB	12
9	Blockschaltbild	13
10	Schema	14
11	Spannungsrippel	15
12	Useless comment	18

Einführung

Diese Dokumentation dient zur Bedienung und Weiterentwicklung des “Robocars”, das im Rahmen des [Mobile Robotics Projekt 1](#) an der [FHGR](#) während des dritten Semesters entwickelt wurde.



Figure 1: Robocar

i Info

- Diese Dokumentation soll als Zusatz dienen und nicht als Ersatz für die Sourcecode Dokumentation.
- Dieses Dokument wurde mit [Quarto-Markdown](#) erstellt und ist sowohl in PDF (ohne Videos) als auch in HTML (bevorzugt) verfügbar.

Übersicht

Das Robocar basiert auf einem stark modifiziertem Chassis von [YF-Robot](#) und verwendet einen [STM-32F446RE](#) Mikrocontroller, welcher auf einem eigens dafür entworfenem PCB mit verschiedenen Peripherien ([M5Stack ESP32-S3](#), [IMU](#), [ToF-Kamera](#), ToF-Sensor, [Visual-Flow-Kamera](#)) kommuniziert.

Das Robocar soll mittels sehr günstiger Hardware ferngesteuert werden und als weiterführende Ziele gewisse Assistenzfunktionen wie Kollisionsdetektion und Spurhalteassistenz verfügen. Weiter soll eine einfache Kollisionsverhinderung mittels Ausweichmanöver implementiert sein.

Handbuch

Benötigte Komponenten

- Robocar
- Geladene Batterien!
- Laptop mit GUI-Applikation

💡 Tip

Sicherstellen, dass die Batterien ganz geladen sind.

Hardware

1. Geladene Batterien in den Batteriehalter einsetzen.
2. Batteriehalter in die vorgesehene Halterung einsetzen.
3. Batteriestecker XT30 mit dem PCB verbinden.
4. Robocar am Hauptschalter einschalten.

❗ Vorsicht

Beim Einsetzen der Batteriehalterung in die vorgesehene Position auf dem Robocar, muss darauf geachtet werden, dass die Litzen des ToF-Spot Sensors nicht gequetscht werden!

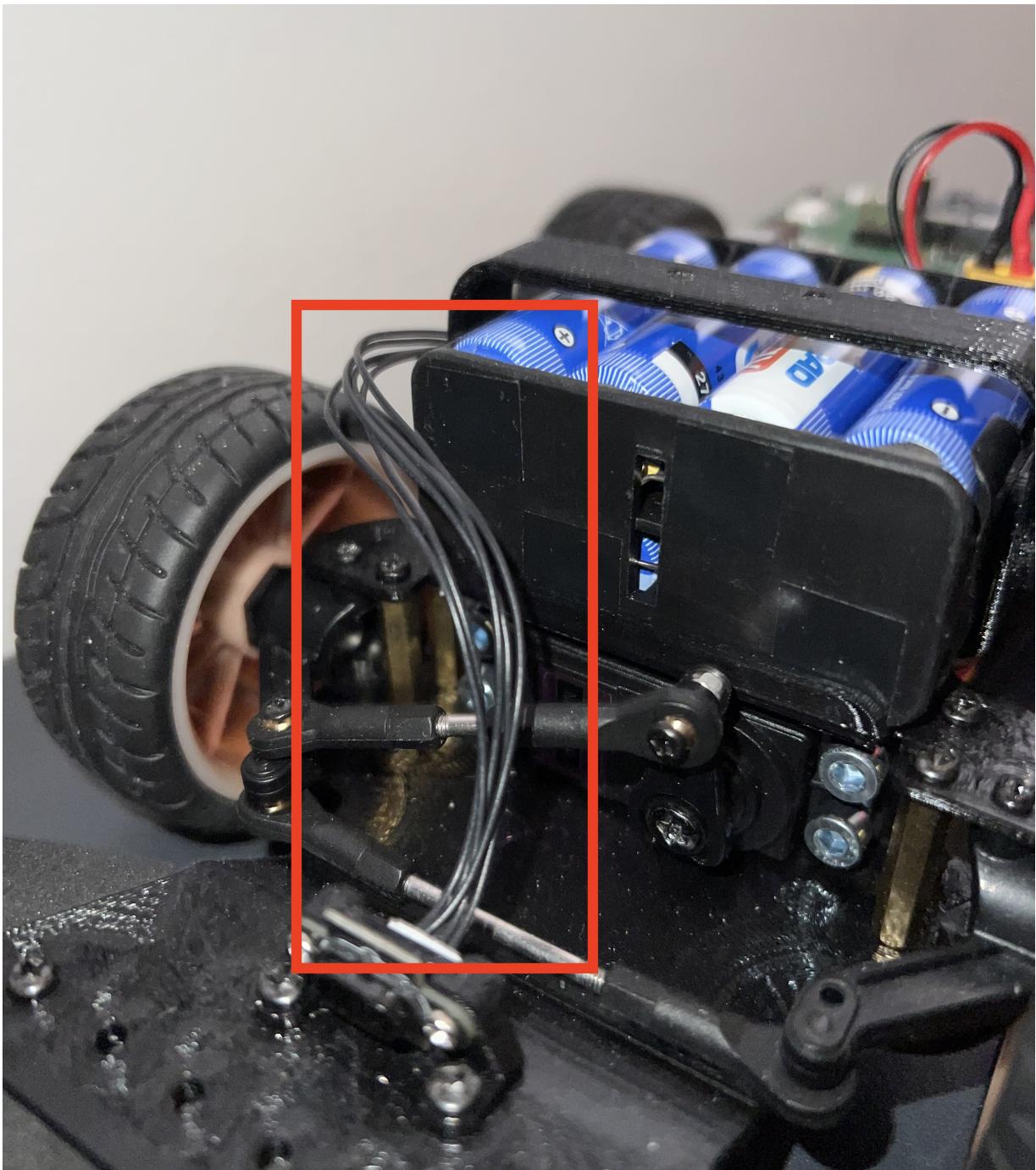


Figure 2: Kabelquetschung

GUI

Über die GUI können alle Sensordaten live abgefragt werden und alle Parameter persistent verändert werden.

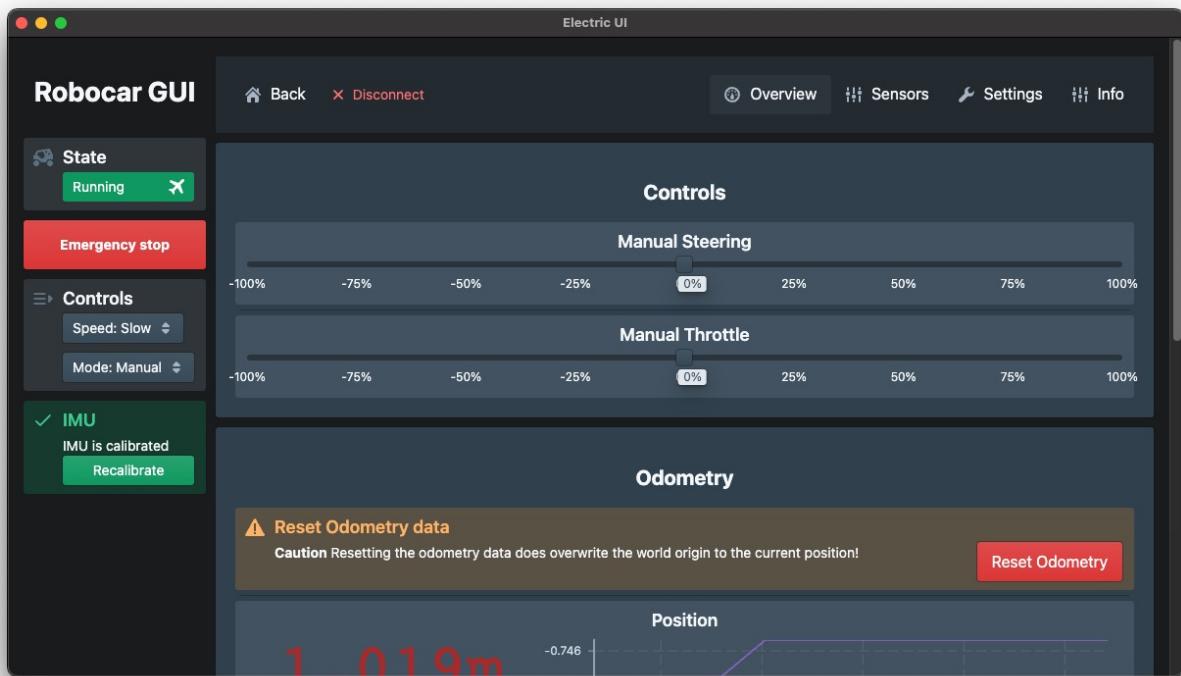
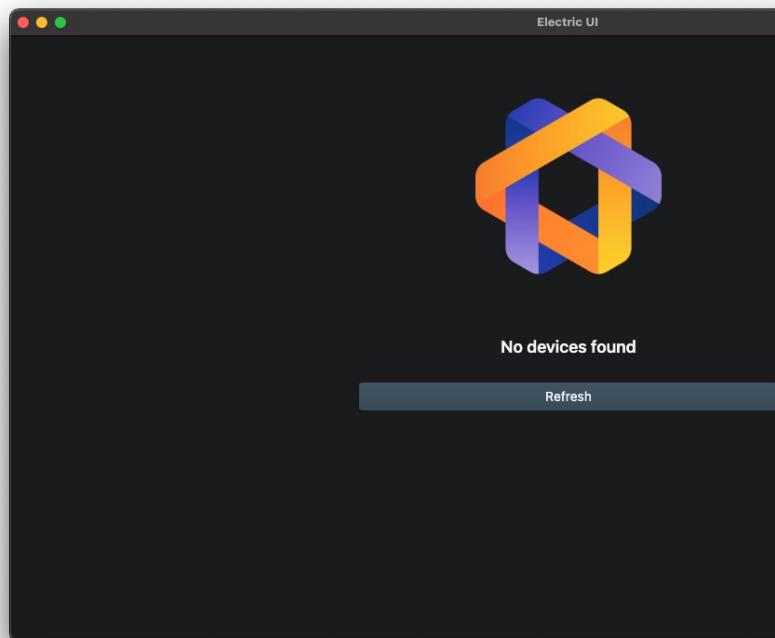
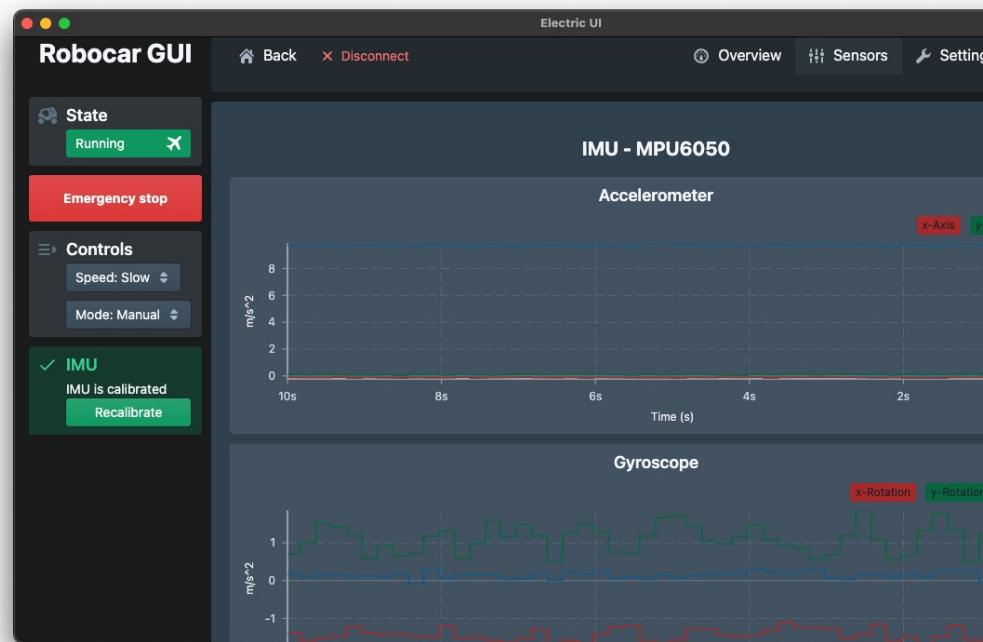
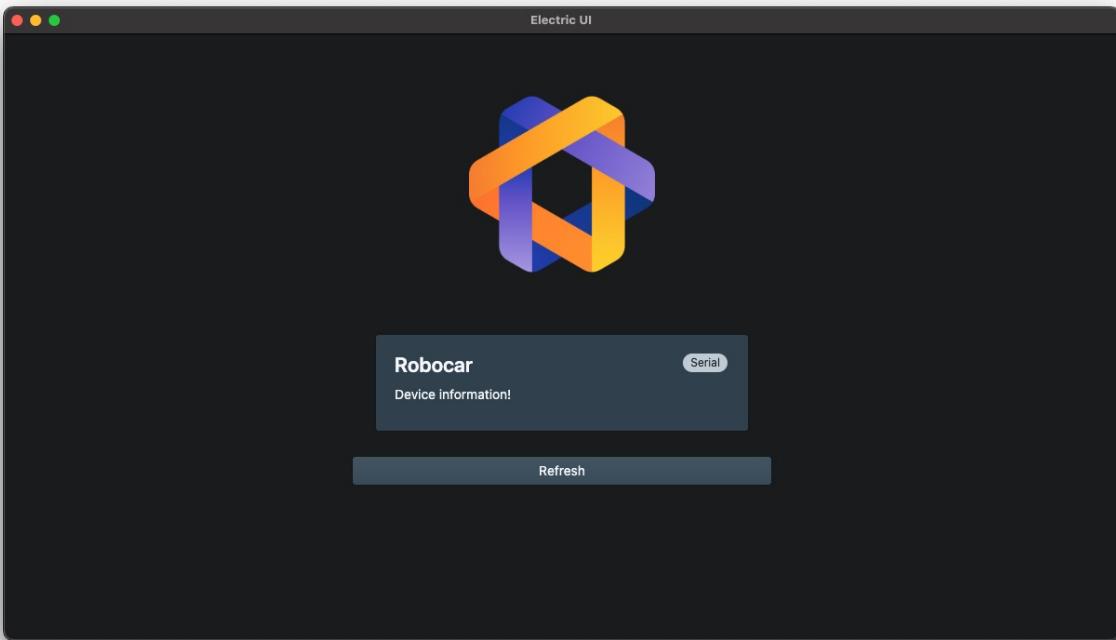


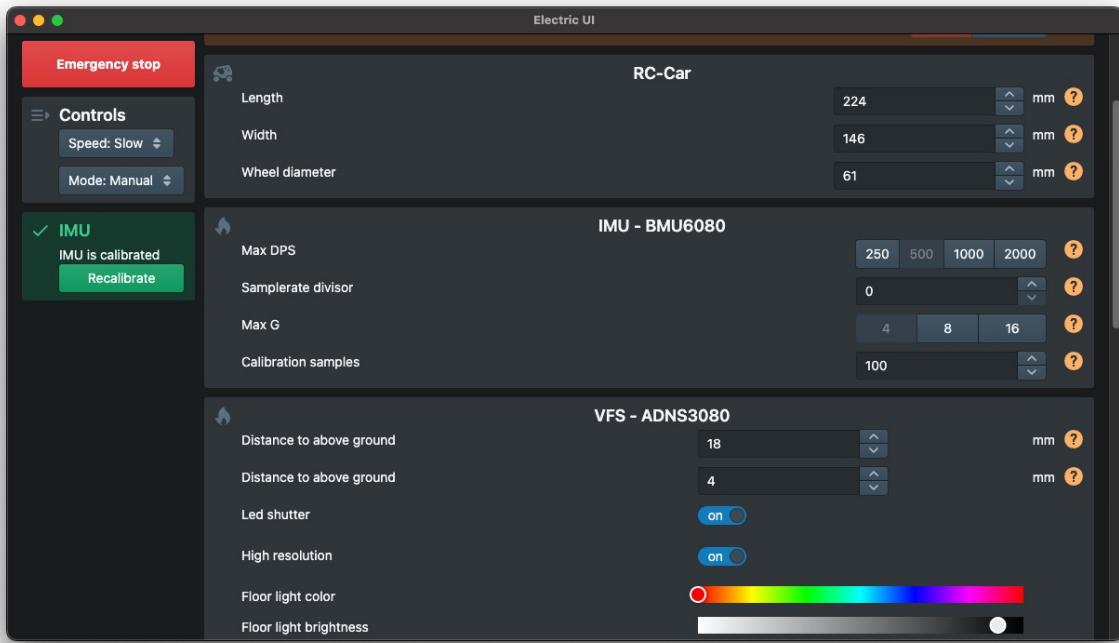
Figure 3: GUI Robocar



1. GUI-Applikation starten
2. Verbindung mit dem Robocar über WIFI/Bluetooth oder Serial (kabelgebunden) herstellen.



3. Werte und Parameter abfragen



Spass haben

Folgende Betriebsmodi wurden implementiert:

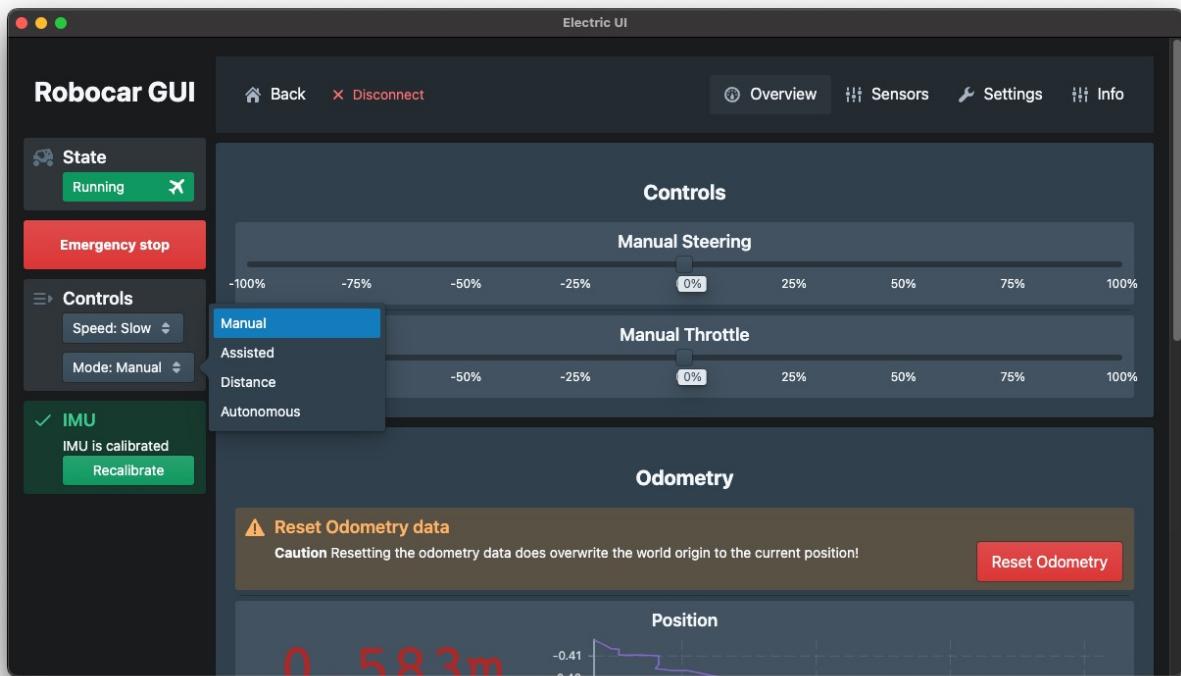


Figure 4: Betriebsmodis

- **Manuelles Verfahren:** Das Robocar kann mittels Slider in der GUI oder dem Controller bewegt werden.
- **Assistiertes Verfahren:** Das Robocar kann mittel Slider in der GUI oder dem Controller, unter Berücksichtigung der Spurhaltefunktion, bewegt werden.
- **Distanz-Modus:** Das Robocar fährt auf ein Objekt zu und hält bei einer parametrisierten Distanz mit einstellbarer Tolleranz vor dem Objekt automatisch an.

- **Kollisionsverhinderung:** Das Robocar umfährt selbstständig Hindernisse ohne Eingreifen einer Drittperson.

Hardware

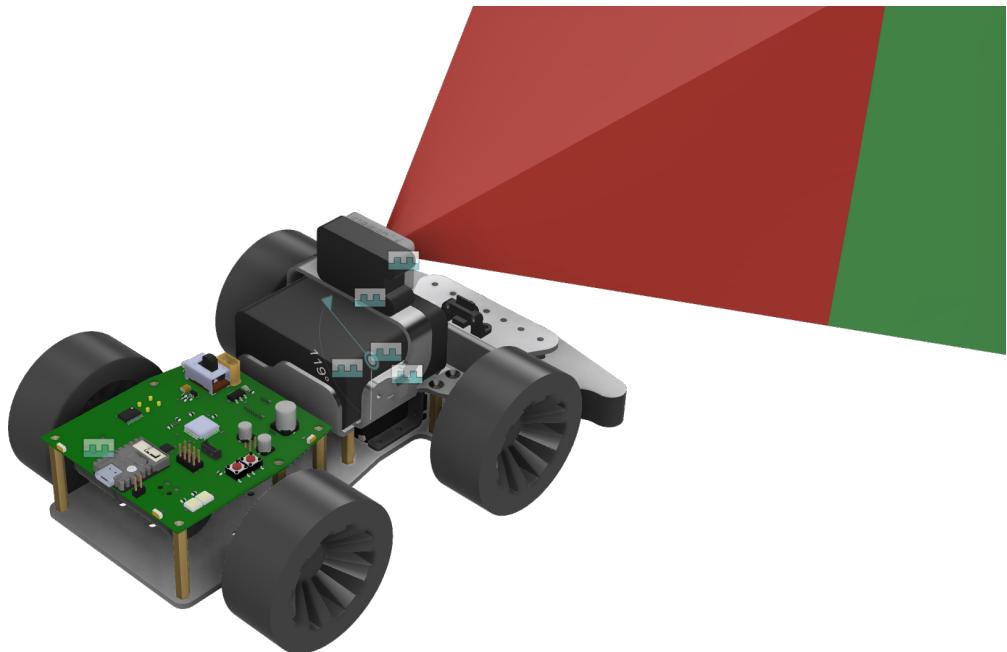


Figure 5: Fusion-Export

Für das Robocar wurden folgende Komponenten entworfen:

- **Grundplatte**

Bindeglied für alle Komponenten und massgebend für die Steifigkeit des Fahrzeuges. Der Radabstand konnten im Vergleich zum Original-Chassis deutlich verringert werden.

[res/data/Groundplate.mp4](#)

- **Batteriehalter**

Der Batteriehalter wurde über der Vorderachse angebracht, um den Schwerpunkt nach vorne zu bringen. Die Batteriehalterung dient zudem als Halter für die Kamera.

[res/data/Batteriehalter.mp4](#)

- **VFS mit Beleuchtungsring**

Mechanisch stellte der Visual-Flow-Senser die grösste Herausforderung dar. Es musste darauf geschaut werden, dass der Sensor ein scharfes Bild des Boden aufnimmt und zudem musste der Boden noch zusätzlich beleuchtet werden.

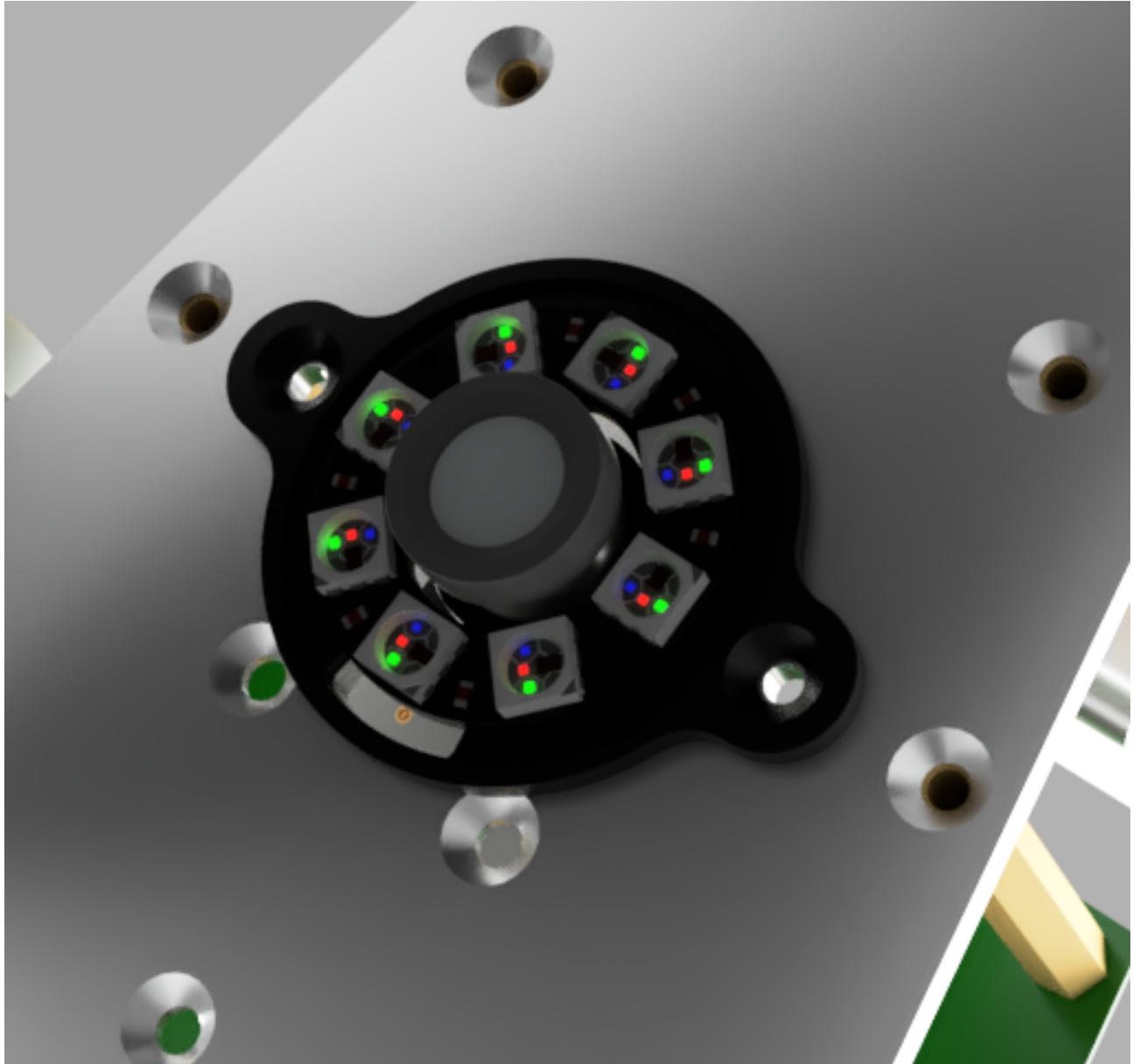


Figure 6: VFS-Beleuchtung

- **ToF-Spot Halterung**

Der ToF-Spot Sensor wurde so auf dem Chassis montiert, dass er möglichst nahe über den Boden hinweg schaut, um so auch niedrige Objekte zu detektieren.

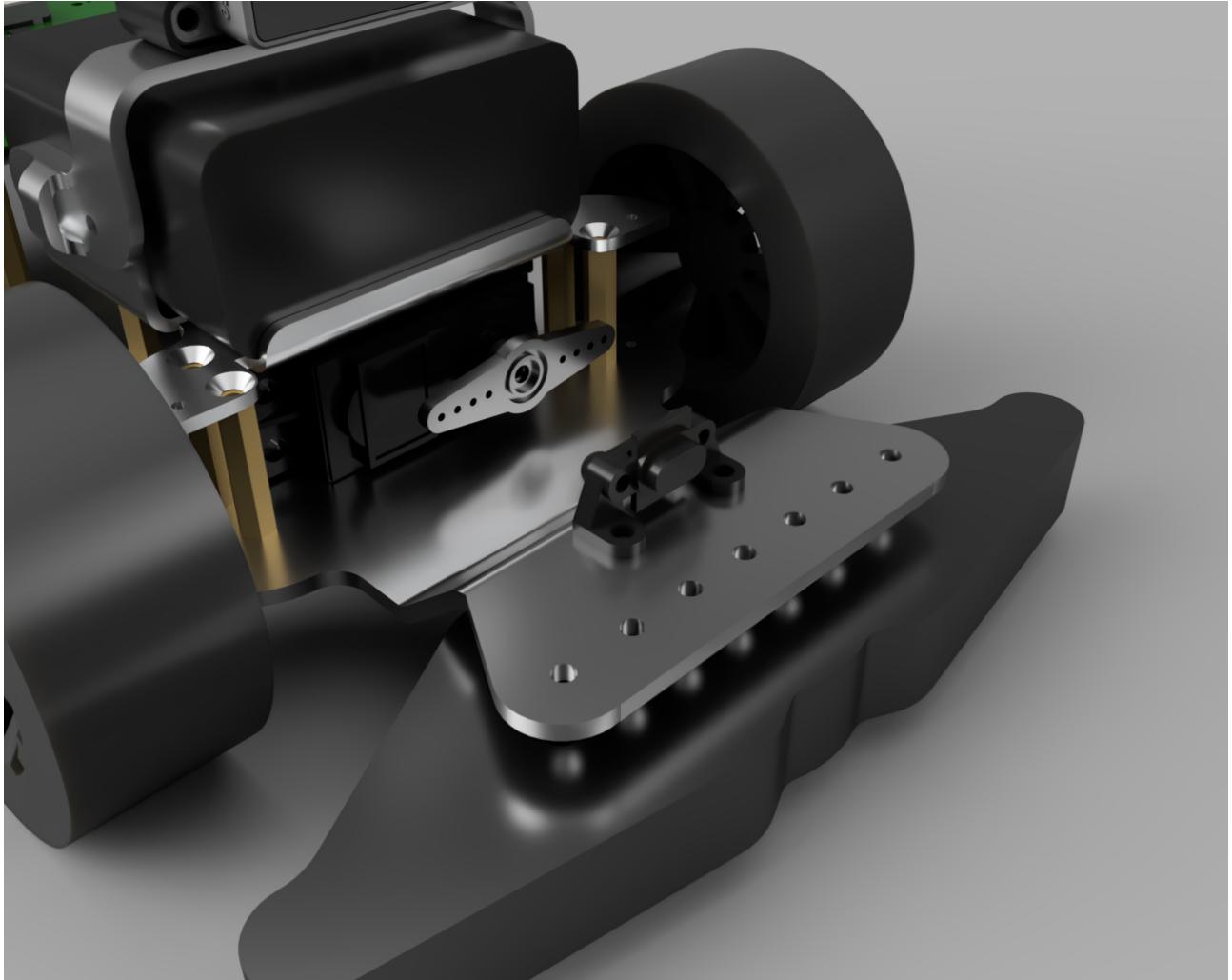
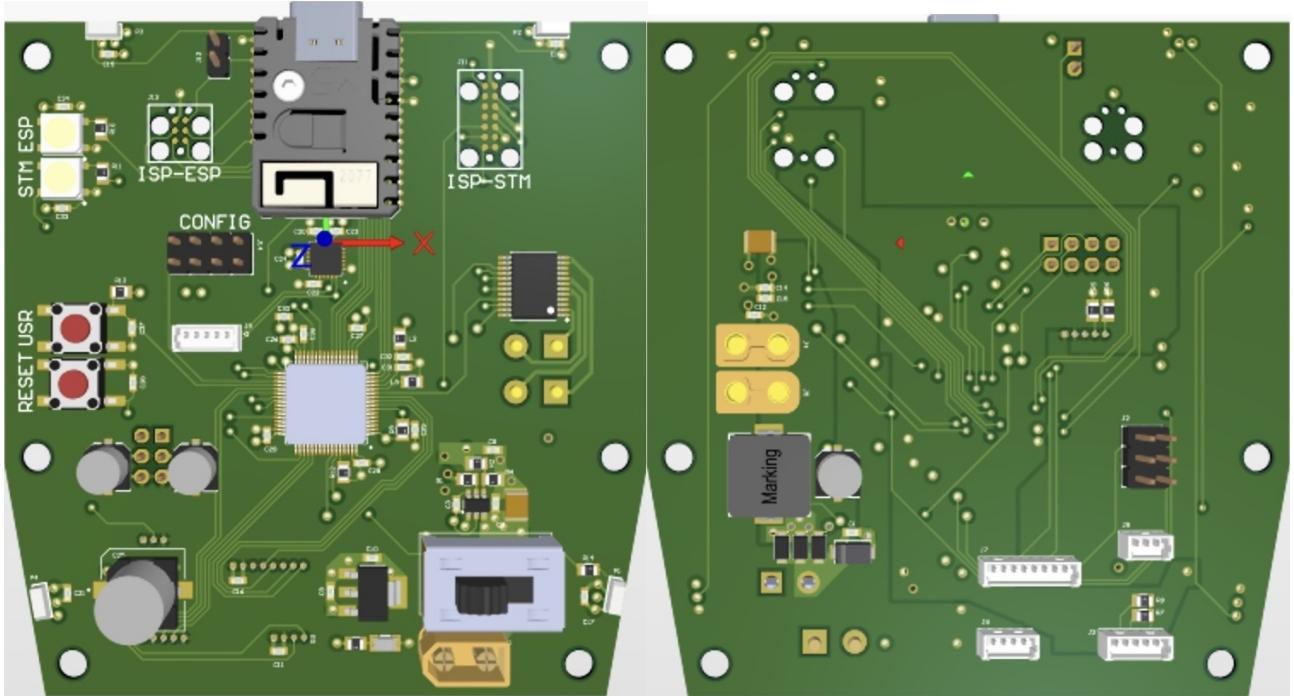


Figure 7: ToF-Spot Halterung

Elektronik

Die Elektronik basiert auf einem eigens entworfenen PCB. Auf dem PCB wird die gesamte Spannungsregelung sowie Spannungverteilung an die jeweiligen Peripherien geregelt sowie die Kommunikation mit dem Mikrocontroller realisiert.



(a) Vorderseite

(b) Rückseite

Figure 8: PCB

Das Robocar wird durch 8xAA Batterien versorgt und verfügt über ein zweistufiges Spannungswandlersystem, um Spannungslevels für 12 VDC (Endstufe Motorensteuerung), 5 VDC (Peripherie) und 3.3 VDC (Mikrocontroller) zu erzeugen. Über einen Schalter kann die Batterie und somit die Versorgungsspannung (9 V – 14 V) vom PCB getrennt werden.

Blockschaltbild

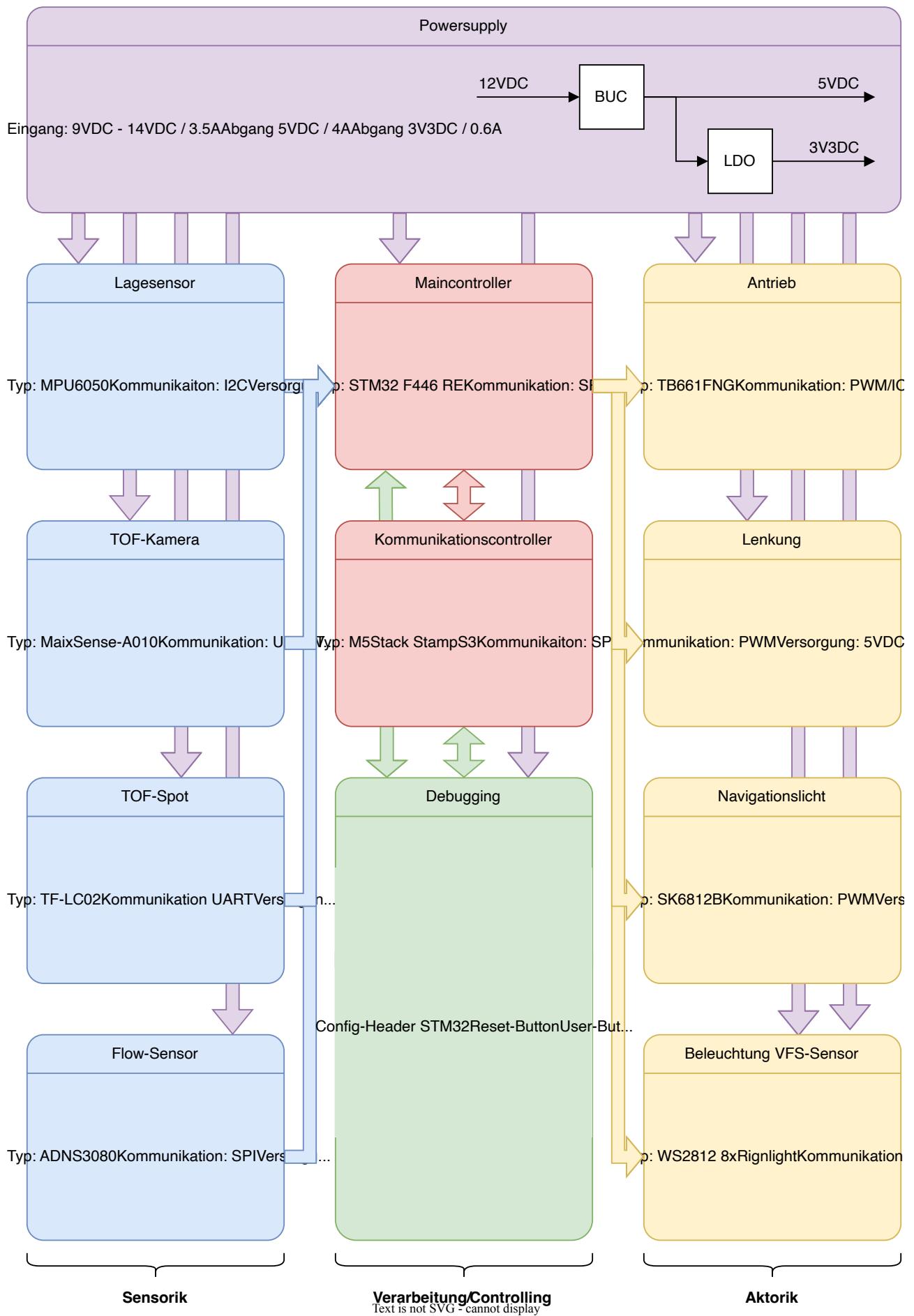


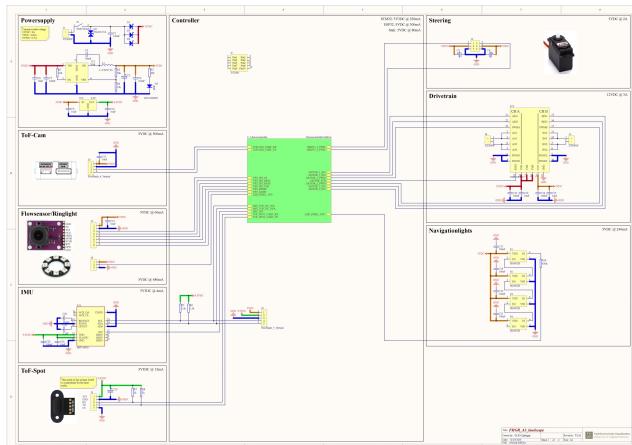
Figure 9: Blockschaltbild

Schema & PCB-Design

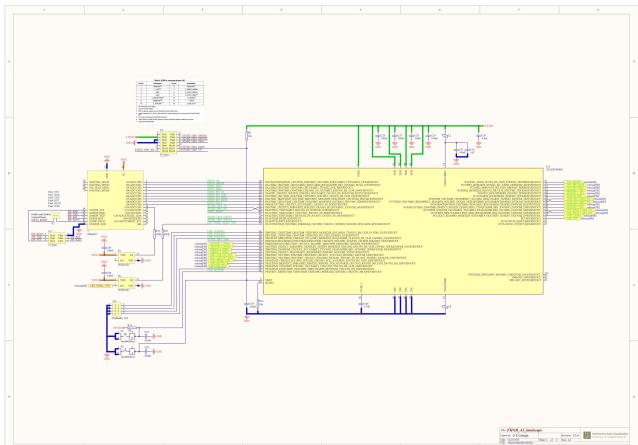
Das PCB verfügt über die folgenden Komponenten:

- Spannungsversorgung
 - 12 VDC Eingang mit ESD- und Polaritäts-Schutz
 - 12 VDC \Rightarrow 5 VDC **BUC-Konverter**
 - 5 VDC \Rightarrow 3.3 VDC **LDO-Spannungsregler**
- Mikrocontroller
 - Maincontroller STM32
 - Kommunikationscontroller ESP32
- Debugging
 - 1 Power-ok LED
 - Reset-Button
 - Konfigurierbarer User-Button
 - 2 RGB Statusled für Mikrocontroller (WS2812)
 - Tag-Connect 6-Pol für ESP32
 - Tag-Connect 14-Pol für STM32
- Peripherie
 - Lagesensor (MPU6050)
 - ToF-Kamera (MaixSense-A0101)
 - ToF-Spot (TF-LC02)
 - Fluss-Sensor (ADNS3080)
 - Motorentreiber (TB6612FNG)
 - Navigationslichter (SK6812)
 - Ringbeleuchtung Fluss-Sensor (8xWS2812)
- Stecker
 - 1 XT30 Stecker für den Batterieanschluss
 - 2 XT30 Buchsen für Motoren
 - 2 3-Pin Header für Servos
 - 4 GPIO Pins mit GND

Dem Schema sind alle weiteren Details bezüglich Auslegung und Dimensionierung zu entnehmen.



(a) Mikrocontroller



(b) Peripherie

Figure 10: Schema

Info

Für Komponenten, die nicht der FHGR-Altium Bibliothek entnommen werden konnte, wurde jeweils eine eigene Bibliothek erstellt. Diese ist unter **PCB/lib** zu finden.

Messungen BUC-Konverter

Wirkungsgrad

Der Wirkungsgrad des BUC-Konverter wurde mithilfe eines Messwiderstandes von $R = 10 \Omega$ gemessen, was zu einem theoretischen Strom von $I = 500 \text{ mA}$ auf der Abgangsseite führt.

Messungen

	Spannung	Strom
Eingangseite	$U_{in} \approx 12.24 \text{ V}$	$I_{in} \approx 290 \text{ mA}$
Ausgangseite	$U_{out} \approx 4.89 \text{ V}$	$I_{out} \approx 510 \text{ mA}$

Berechnung

$$\eta = \frac{P_{out}}{P_{in}} = \frac{U_{out} \cdot I_{out}}{U_{in} \cdot I_{in}} \approx 72 \%$$

$U_{in} = 12.24$

$I_{in} = 0.29$

$U_{out} = 4.98$

$I_{out} = 0.51$

```
print(f'P_in = {(P_in := U_in*I_in):.2f} W')
print(f'P_out = {(P_out := U_out*I_out):.2f} W')
print(f'n = {P_out/P_in*100:.2f} %')

P_in = 3.55 W
P_out = 2.54 W
n = 71.55 %
```

Note

Bei der Messung wurden die folgenden Komponenten nicht entfernt, da diese aufwendig herausgelöst werden müssten:

- Mikrocontroller STM32
- Kommunikationscontroller ESP32
- IMU MPU6050

Aus diesem Grund fliesst immer ein gewisser Strom, der bei der Abgangsseite nicht erfasst werden kann. Dies führt zu einem Querstrom, welcher nicht erfasst werden kann und eine genaue Messung verunmöglicht sowie den tiefen Wirkungsgrad ausmacht.

Spannungsrippel

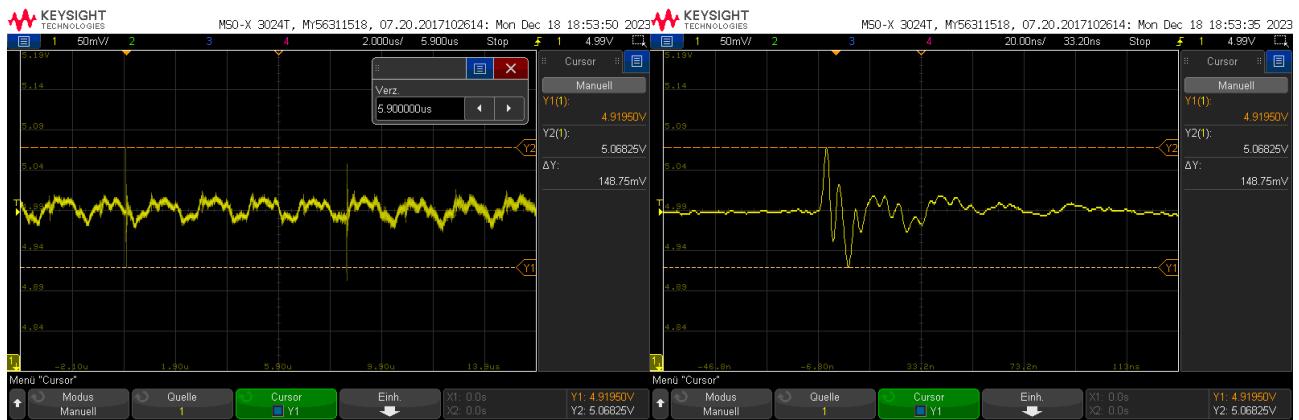


Figure 11: Spannungsrippel

Der Spannungsrippel beträgt ca. $\Delta U_5 \approx 150 \text{ mVDC}$, was 3 % von 5 VDC entspricht.

i Note

Ein Grossteil der Spannungsrippe macht hierbei das ESP32 aus. Wird dieses ausgelötet und die Messung wiederholt, so sind um einiges bessere Werte zu erwarten.

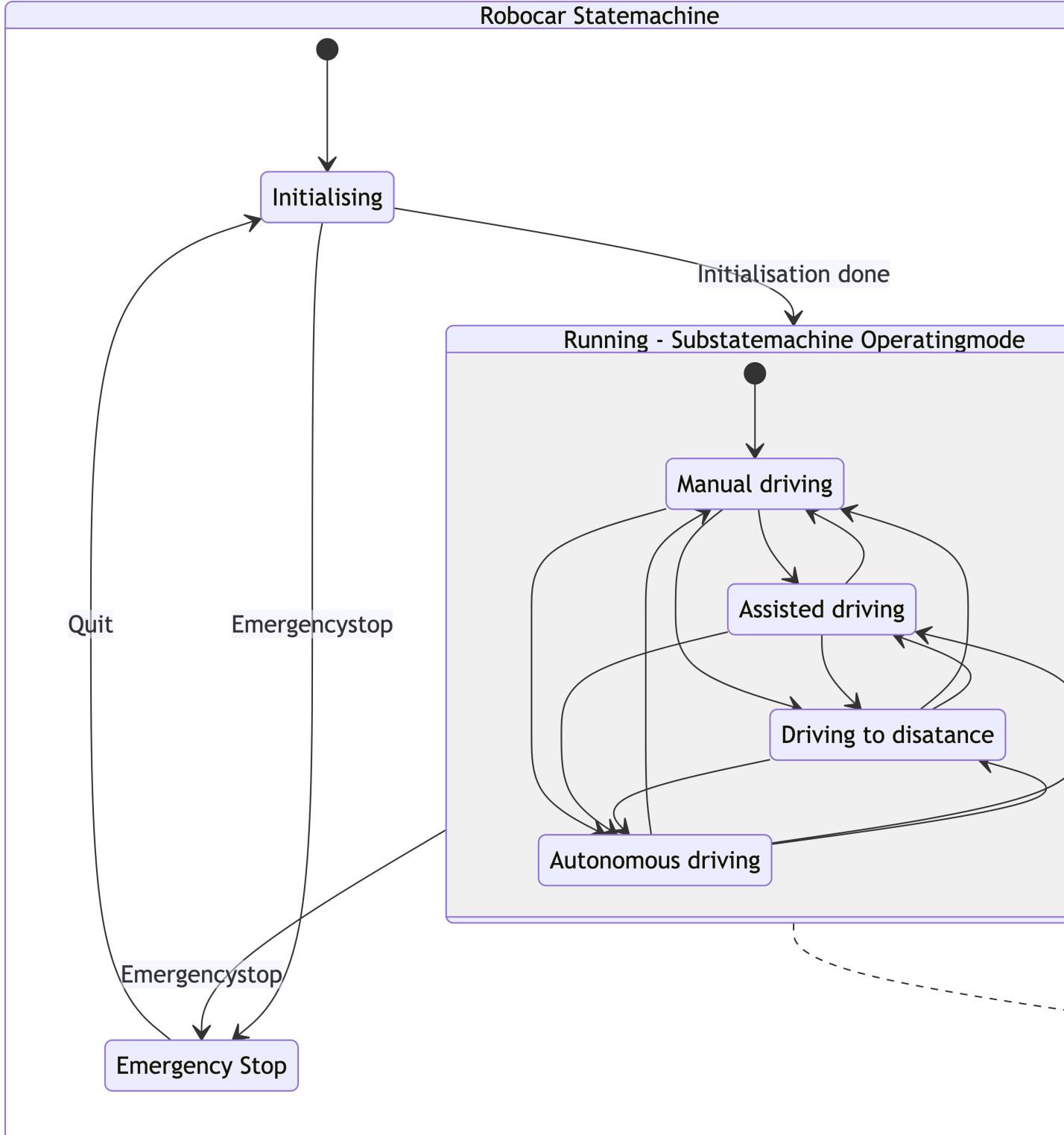
Software

Softwareaufbau

Das Robocar wurde aufgrund der erhöten Übersichtlichkeit und Abkapselung der einzelnen Module mit [C++](#) programmiert. Dafür wurde für jede Komponente ein eigenes Modul entwickelt und implementiert.

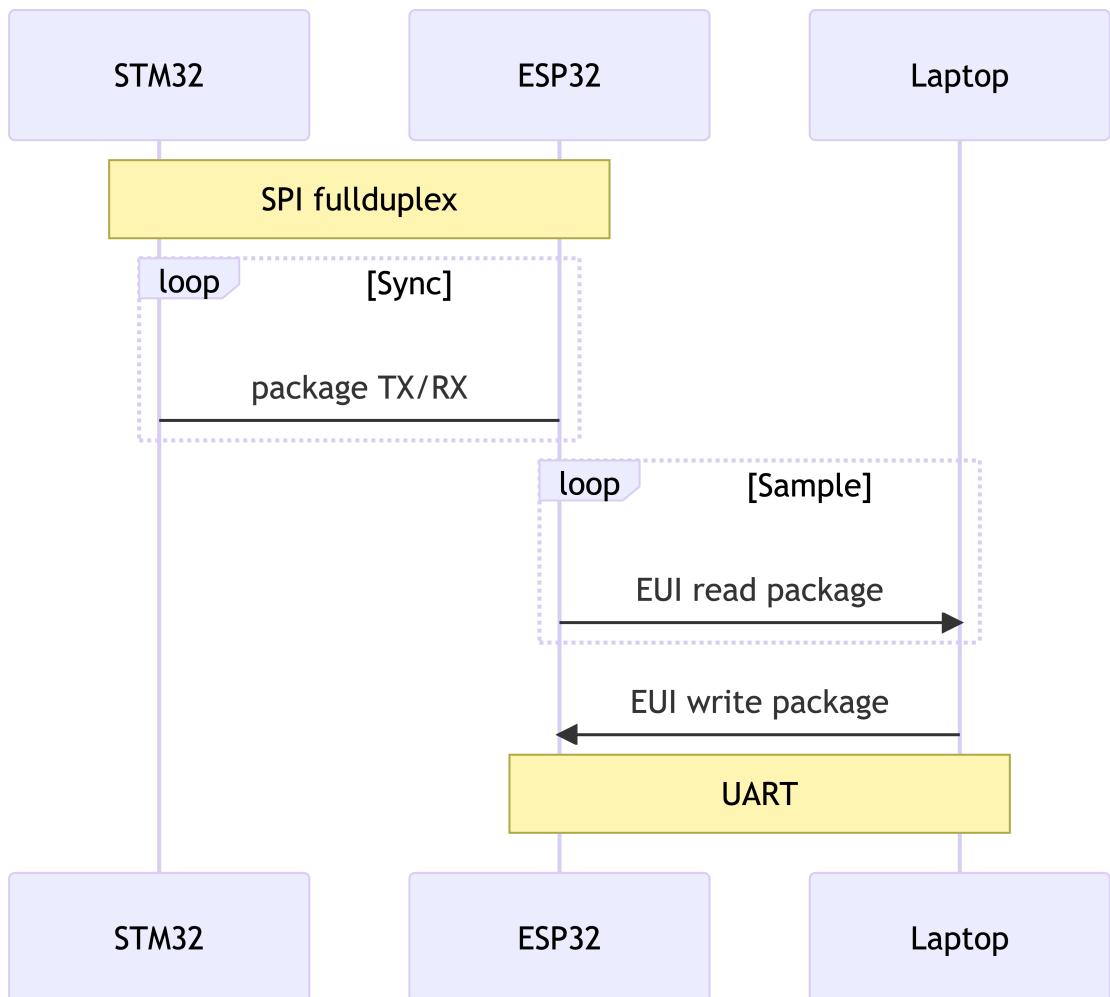
Komponente	Modulname	Softwaremodul
IMU	MPU60X0	Github
ToF-Kamera	A010	Github
ToF-Sensor	TFLC02	Github
Visual-Flow-Kamera	ADNS3080	Github
Neopixel	WS2812/SK6812	Github
Servo	MG90	Github
H-Brücke	TB6612FNG	Github

Flussdiagramm



Funktionalität

Die Hauptschwierigkeit bei der Implementierung ist die Synchronisation der [Datenstrukturen](#) zwischen den beiden Mikrocontrollern. So müssen Überschreibungen vermieden werden und requests/acknowledges passend gesetzt und gelöscht werden. Zwischen dem ESP32 und dem STM32 wurde eine SPI-Kommunikation eingerichtet und das ESP32 kommuniziert mit dem Laptop über UART. Über ein [eigenes Protokoll](#) zwischen STM32 und ESP32 wird in jedem Zyklus, welcher durch das ESP32 initiiert wird die Variablen zwischen den Mikrocontrollern synchronisiert. Dabei wird von simultan aus/in den DMA Speicher auf beiden Mikrocontrollern geschrieben.



i Note

Bei einer Hardware-Revision würde es Sinn machen die SPI-Kommunikation durch eine UART-Kommunikation zu ersetzen und das ESP32 nur als UART-Bridge einzusetzen.

Der Rest der Software ist trivial und [selbsterklärend](#).

💡 Tip



Figure 12: Useless comment

- A comment is a failure to express yourself in code. If you fail, then write a comment; but try not to fail. [Robert Martin](#)
- The comments will inevitably become out of date, and untrustworthy comments are worse than no comments at all. [The Pragmatic Programmer](#)

- Make your code explain itself. If it has too many comments, you're doing it wrong!

Modulbeschreibungen

Die einzelnen Modulbeschreibungen können den jeweiligen .hpp Source-Dokumenten als [Doxygen](#) entnommen werden und werden hier nicht explizit aufgeführt!

ToDo-Software

ToDo's für einzelne Module den einzelnen Doxygen-Kommentaren zu entnehmen und werden hier nicht explizit gelistet.

- Generell muss ein [Refactoring](#) getätigt werden.
- Implementierung eines [Watchdog](#).
- Implementierung des Kollisionsverhinderungs-Modus
- Umschalten auf die Odometry-Daten des IMU wenn die Bodequalität des VFS zu schlecht wird.
- "Richtige" Implementation eines PI-Speedcontrollers -> Jetztiger oszilliert praktisch ungedämpft.

Stand der Dinge

Fernsteuerung

Das Robocar ist zur Zeit nicht in der Lage drahtlos ferngesteuert zu werden. Dies kann jedoch mit geringem Aufwand implementiert werden, sobald das nächste Update von [Electricui](#) puliziert wird. (Anfangs des Projektes wurde bei den Entwicklern nachgefragt und ein Release sollte bis Ende Dezember 2023 sichergestellt sein.)

Implementation X-Box-Kontroller

Das Robocar ist zur Zeit nicht in der Lage über einen X-Box-Kontroller ferngesteuert zu werden. Dies kann jedoch mit geringem Aufwand implementiert werden, sobald das nächste Update von [Electricui](#) puliziert wird. (Anfangs des Projektes wurde bei den Entwicklern nachgefragt und ein Release sollte bis Ende Dezember 2023 sichergestellt sein.)

Kommunikation zwischen den Mikrocontrollern

Die Kommunikation zwischen den Mikrocontrollern ist hochgradig ineffizient und könnte um einiges datensparender implementiert werden. Dadurch würden sich auch die Zugriffszeiten von der GUI zum ESP32 verringern, was eine höhere Aktualisierungsrate ermöglicht.

Zur Zeit werden immer alle Bytes der gesamten Datenstruktur übertragen, auch wenn nur ein Teil der Datenstruktur angefragt wurde. Ein Grossteil davon macht das Kamerabild der ToF-Kamera aus, was bis zu 10'000 Bytes betragen kann.

ToF-Kamera

Leider gelang es nicht, die Pixeldaten der Kamera auf den STM32 zu übertragen. Die UART-Schnittstelle scheint auf zwei verschiedenen Kameras immer deaktiviert zu sein. Auch wenn diese über das Konfigurationstool aktiviert wird. Aus diesem Grund wurde der Kollisionsverhinderungs-Modus auch nicht implementiert.

Visualisierung

Die Visualisierung wurde nicht kompiliert, um ohne Entwicklungsumgebung zu funktionieren. Dies kann jedoch nachträglich für alle gängigen Plattformen (Windows, MacOS, Linux) gemacht werden.

Parameterhandler

Zur Zeit können keine Parameter persisteng gespeichert werden. Bei einem Neustart gehen alle Werte verloren. Für den STM32F401RE funktioniert der Code und ein Registermap des Flash-Speichers ist im Web zu finden. Für den STM32F446RE ist keine Dokumentation über Den Flashaufbau zu finden und die Pages und Addressen stimmen nicht mit denen des STM32F401RE überein.

Bekannte Probleme

Reset des Robocars

Wird über den Reset-Button ein Reset-Event ausgelöst, so ist es möglich, dass das IMU über den I2C Bus nicht gefunden wird und die Software nach einer parametrisierten Anzahl an Pingversuchen automatisch den Error-Handler aufruft.

```
if (imu.init(3) != SUCCESS) Error_Handler();
```

Verbindungsprobleme mit GUI

Zum Teil wird das Robocar von der GUI nicht erkannt und kann somit nicht ferngesteuert werden. In diesem Fall muss nach folgendem Ablauf vorgegangen werden:

1. Robocar vom Laptop trennen
2. Robocar ausschalten
3. GUI “refresh” drücken
4. Robocar erneut mit dem Laptop verbinden
5. Wird das Robocar erkannt, so muss auf dem Robocar der Reset-Button gedrückt werden
6. GUI “connect” drücken