



HOCHSCHULE LANDSHUT
HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN

**HOCHSCHULE FÜR ANGEWANDTE
WISSENSCHAFTEN LANDSHUT**

FAKULTÄT ELEKTROTECHNIK UND WIRTSCHAFTSINGENIEURWESEN

Masterstudium

Elektrotechnik

Projektbericht

Eingebettete Autonome Systeme

**Aufbau und Programmierung eines Modellfahrzeugs für den NXP Cup
Gesamtdokumentation**

vorgelegt von:

Ambrosch Markus, Ecker Christian, Summer Matthias

eingereicht am:

30.07.2021

Betreuer: Prof. Dr. Mathias Rausch

Vorwort

Dieser Projektbericht enthält zusätzlich zur Dokumentation der im Sommersemester 2021 erzielten Ergebnisse auch die des Vorsemesters (Wintersemester 2020/2021). Nachdem im Wintersemester nicht alle der gesetzten Ziele erreicht werden konnten, stellt dieser Bericht die Gesamtdokumentation des über zwei Semester andauernden Projekts „Aufbau und Programmierung eines Modellfahrzeugs für den NXP Cup“ dar.

Zur Unterstützung bei der Erledigung der noch ausstehenden Aufgaben ist zum Sommersemester Herr Markus Ambrosch dem Projekt-Team des Wintersemesters (Ecker Christian & Summer Matthias) beigetreten.

Arne Kullina danken ***Rausch danken***

Inhaltsverzeichnis

Vorwort	II
Inhaltsverzeichnis	III
Abkürzungsverzeichnis	V
1 Einführung	1
1.1 Zielsetzung	1
1.2 Konzept	1
2 Aufbau des Fahrzeugs	3
2.1 Grundaufbau	3
2.2 Anbaukomponenten aus dem 3D-Druck	5
2.2.1 Stoßstange und Ultraschallboard-Halterung	5
2.2.2 Akku-Halterung und Seitenschweller	7
2.2.3 Halterung der Controllerplatine und des Bedienungs-Boards	9
2.2.4 Halterung für die Motorcontroller	12
2.2.5 Einfache Platinenhalterungen	13
2.2.6 Kamerahalterung	14
3 Controllerplatine und ProgrammierTool	16
4 Antrieb des Fahrzeugs	17
4.1 BLDC-Antrieb und Motorcontroller	17
4.2 Montage der Antriebskomponenten	18
4.3 Konfiguration der Motorcontroller	19
4.4 Programmierung des Antriebsbausteins	21
4.5 Drehzahlmessung	24
4.5.1 Erörterung der Notwendigkeit einer Drehzahlmessung	24
4.5.2 Auswahl des Messprinzips	26
4.5.3 Hardware für die Drehzahlmessung	26
4.5.4 Programmierung des Drehzahlmessungsbausteins	26
5 Lenkung des Fahrzeugs	27
5.1 Servoantrieb	27
5.2 Montage der Lenkungskomponenten	28
5.3 Programmierung des Lenkungsbausteins	29
6 Bedienungs-Board	32
6.1 Schaltplan	32
6.2 Programmierung der Anzeige	33

6.3	Programmierung der Steuerelemente	33
7	Streckenerkennung des Fahrzeugs	34
7.1	Kamera des Fahrzeugs	34
7.2	Montage und Ausrichtung der Kamera	34
7.3	Programmierung der Streckenauswertung	34
7.4	Kameralinsen	34
8	Verteilerplatine auf der unteren Fahrzeugebene	35
9	Regelung des Fahrzeugs	36
10	Inbetriebnahme des Fahrzeugs	37
11	Zusammenfassung, Fazit und Ausblick	38
11.1	Zusammenfassung und Fazit	38
11.2	Ausblick	38
	Abbildungsverzeichnis	VI
	Quellenverzeichnis	VIII
	Anhang	IX

Abkürzungsverzeichnis

BLDC-Motor	Brushless Direct Current Motor
DC-Motor	Gleichstrommotor
RC	Remote-Control
ESC	Electronic Speed Controller
PWM	Pulsweitenmodulation
NXP	Next Experience
CPU	Central Processing Unit
ABS-Kunststoff	Acrylnitril-Butadien-Styrol Copolymer
SLA-Drucker	3D-Drucker mit Stereolithographie-Verfahren

1 Einführung

1.1 Zielsetzung

Das Ziel der Projektarbeit ist der Aufbau und die Programmierung eines Modellfahrzeugs für den Next Experience (NXP)-Cup mit einem eigens gebauten, selbstfahrenden Fahrzeug. Bei diesem Wettbewerb müssen die im Maßstab 1:18 angefertigten Fahrzeuge einen Parcours in möglichst kurzer Zeit selbstständig durchfahren. Diese werden von Studenten aus Europa, dem mittleren Osten und Afrika auf Grundbasis eines Bausatzes entwickelt. Die Fahrbahn wird von zwei schwarzen Streifen begrenzt, die auf einem weißen Hintergrund aufgebracht sind. Das Fahrzeug muss diese Begrenzungen erkennen und anhand deren Auswertung die Geschwindigkeit des Fahrzeugs und dessen Lenkwinkel anpassen. Angetrieben wird das Auto mithilfe zweier Brushless Direct Current Motoren (BLDC-Motoren). Die Lenkung wird mittels Servoantrieb und Lenkgestänge realisiert. Auf dem Fahrzeug dürfen beliebig viele Prozessoren und Bauteile von NXP Semiconductors, welche den Wettbewerb ausrichten, verwendet werden [Wil20]. Sind für das eigene Fahrzeug benötigte Komponenten nicht im Portfolio von NXP Semiconductors vorzufinden, können auch eigene oder die Produkte anderer Hersteller verwendet werden.

1.2 Konzept

Zu Beginn soll das Fahrzeug zuerst mit dem Standardbausatz zusammengebaut werden. Die Software wird aus bereits vorhandenen, vorherigen Projekten zusammengesetzt und optimiert. Ein großes Augenmerk liegt dabei auf einem übersichtlicheren Aufbau des Programms und besserer Nachvollziehbarkeit durch Kommentation und eingängigerer Benennung der Funktionen und Parameter.

Die Programmierung soll Stück für Stück vorgenommen werden. Zu Beginn wird die Ansteuerung der BLDC-Motoren bearbeitet. Ist der Punkt erreicht, an dem die Motoren an gesteuert und die Drehzahl über die Puls-Weiten-Modulation variiert werden kann, soll eine Möglichkeit der Drehzahlerkennung erarbeitet werden. Im Anschluss an die Inbetriebnahme der Antriebe wird die Software für den Servomotor der Lenkung erstellt. Die Bedienung des Fahrzeugs soll über ein Display und einen Dreh-Encoder realisiert werden. Zur Streckenerkennung wird eine Zeilenkamera verwendet. Bei Bedarf kann zusätzlich oder ersatzweise eine größere Kamera eingesetzt werden. Nach der Inbetriebnahme der Einzelkomponenten werden deren Ansteuerung und Auswertung mithilfe einer Regelung verknüpft. Die Software des Fahrzeugs soll am Ende so optimiert werden, dass es schnellstmöglich, aber auch sicher durch den Parcours fährt.

Nach der erfolgreichen Entwicklung des Fahrzeugs für das Durchfahren des Parcours soll für eine zusätzliche Wettbewerbsdisziplin eine Objektdetektion mithilfe eines Ultraschallboards realisiert werden, welches bereits in den vorherigen Semestern von anderen Studie-

renden entwickelt wurde. Mithilfe dieses Ultraschall-Boards soll das Fahrzeug Hindernisse erkennen und um diese herumfahren können.

Karosserieteile, wie beispielsweise eine Akkuhalterung oder eine Stoßstange mit der Möglichkeit zur Befestigung des Ultraschallboards, werden mit einem 3D-Druck-Verfahren erstellt. Dafür werden vor allem Teile gedruckt, welche bereits von Herrn Arne Kullina im Rahmen seines Praktikums bei Herrn Prof. Dr. Mathias Rausch im Sommersemester 2020 konstruiert wurden. Für eine übersichtlichere Kabelführung und einfachere De- und Montage wird eine Lochrasterplatine mit Steckkontakten erstellt, an welchen die Einzelmodule angeschlossen werden können.

2 Aufbau des Fahrzeugs

2.1 Grundaufbau

Das Fahrzeug besteht zum größten Teil aus einem Standarbausatz. Auf der unteren Ebene, der Grundplatte, sind die BLDC-Motoren für den Antrieb, der Servomotor und die Reifen montiert (siehe Abbildung 1). In einem weiteren Schritt werden hier auch die Anbauteile des Fahrzeugs aus dem 3D-Druck befestigt, wie beispielsweise die Stoßstange und die Seitenschweller (näheres in Kapitel 2.2).

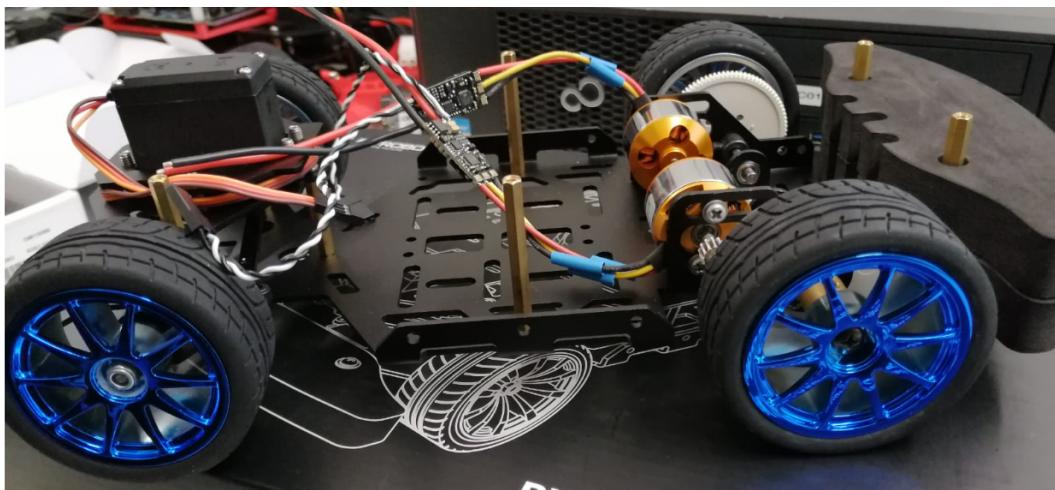


Abbildung 1: Grundplatte des Standardbausatzes für das Fahrzeug mit den bereits montierten BLDC-Motoren, den Motorcontrollern und den Reifen

Zusätzlich ist auf der Grundplatte eine Platine mit Steckmöglichkeiten zum Anschließen der Fahrzeug-Peripherie montiert. Die Steckverbindungen ermöglichen eine einfachere Demontage der verbauten Fahrzeugkomponenten. Auch der Akku findet auf dieser Ebene seinen Platz (siehe Abbildung 2).

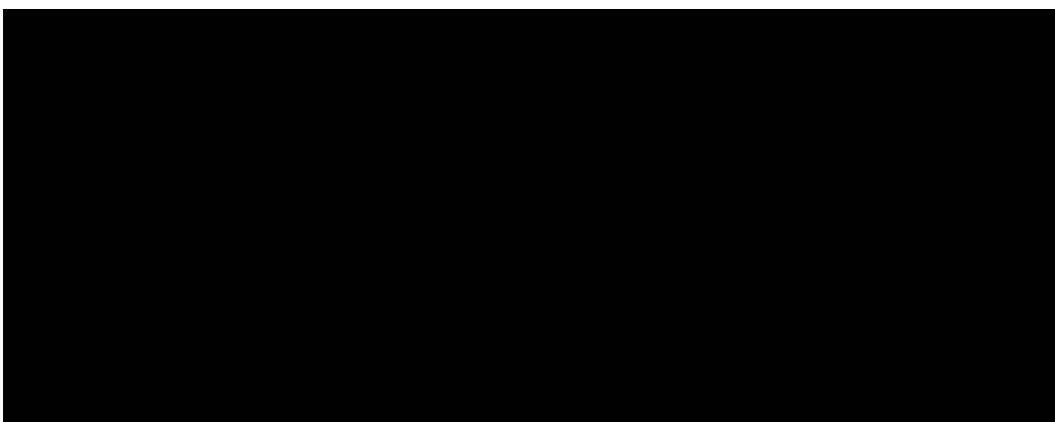


Abbildung 2: Grundplatte des Standardbausatzes für das Fahrzeug mit dem Akku, der Verteilerplatine mit Steckkontakten und der bereits montierten Fahrzeug-Peripherie

Oberhalb der Grundplatte des Fahrzeugs, auf der die Antriebe, die Lenkung, die 3D-Druck Anbaukomponenten und der Akku platziert sind, wird mit einer weiteren Montageplatte eine zweite Ebene aufgespannt (siehe Abbildung 3). Auf dieser oberen Ebene werden sowohl der Controller mit dem Bedienungs-Board, als auch die Kamera montiert (siehe Abbildung 4).

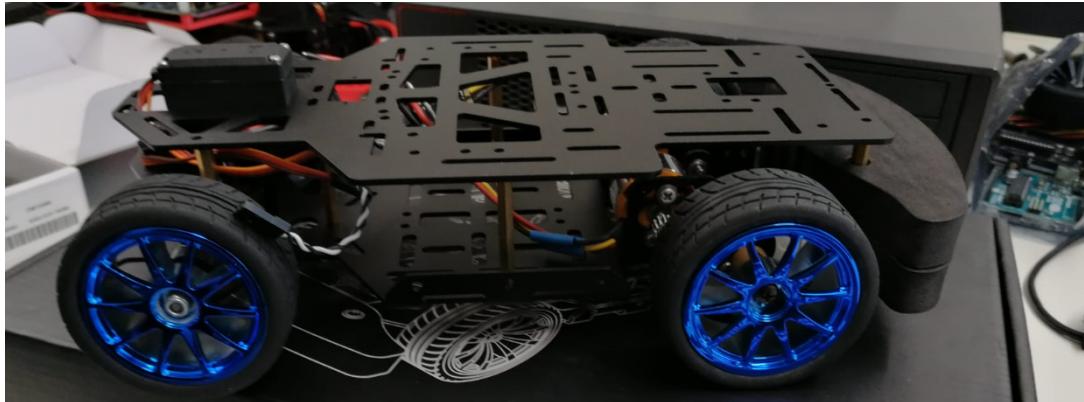


Abbildung 3: Obere Ebene des Standardbausatzes für das Fahrzeug ohne Montage des Controllers und der Kamera

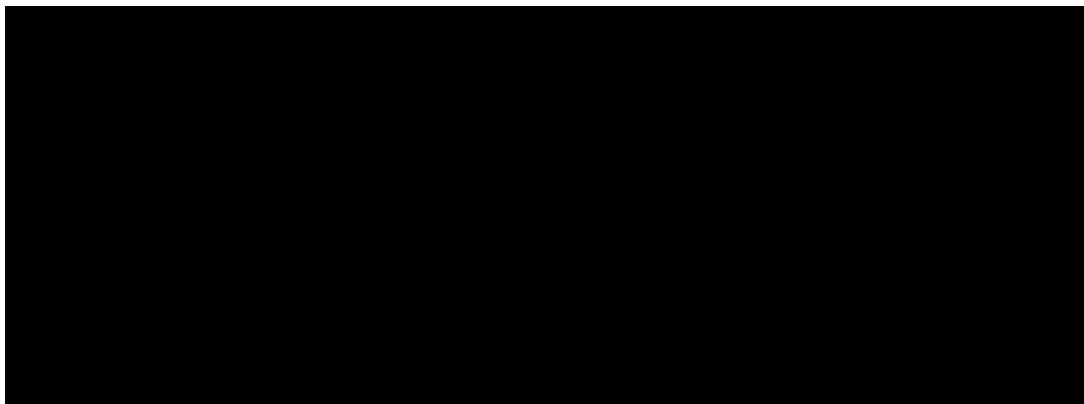


Abbildung 4: Obere Ebene des Standardbausatzes für das Fahrzeug mit befestigtem Controller und Kamera

2.2 Anbaukomponenten aus dem 3D-Druck

Zusätzlich zum Standardbausatz werden auch Anbaukomponenten verbaut, die mithilfe eines 3D-Druckers gefertigt und dann am Fahrzeug angebracht werden. Die Folgekapitel zeigen alle gedruckten Einzelteile und erläutern deren Zweck näher. Die Anbauteile wurden größtenteils von Herrn Arne Kulinna konstruiert, der im Rahmen eines Praktikums bei Herrn Prof. Dr. Mathias Rausch an der Hochschule Landshut bereits eine erste Version des Fahrzeugs mit demselben Bausatz erstellt hat.

2.2.1 Stoßstange und Ultraschallboard-Halterung

Das Fahrzeug benötigt eine Stoßstange, damit bei der Kollision mit einem Hindernis kein wichtiges Teil des Fahrzeugs, wie beispielsweise der Servo-Motor oder das Ultraschall-Board, beschädigt wird. Da das zur Hinderniserkennung zu verwendende Ultraschallboard frontal befestigt wird, müssen in der Stoßstange kegelförmige Aussparungen eingeplant werden, damit die Stoßstange nicht fälschlicherweise als Hindernis erkannt wird. In den Abbildungen 5 und 6 sind die Konstruktionsbilder der beiden Stoßstangenteile abgebildet.

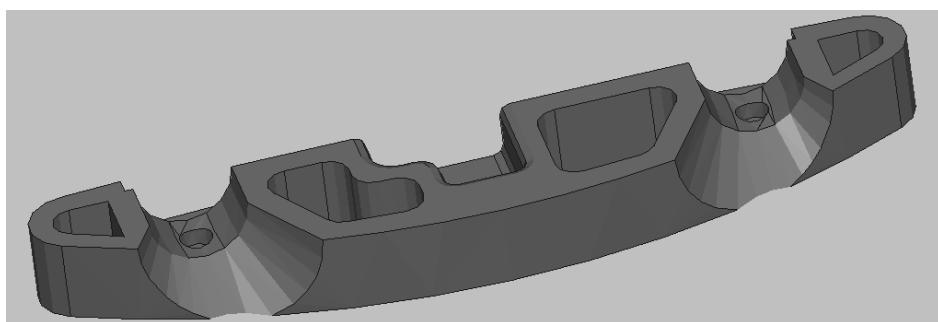


Abbildung 5: Konstruktionsbild des oberen Stoßstangenteils

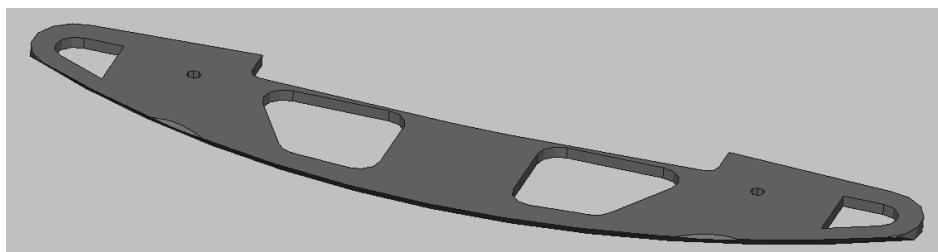


Abbildung 6: Konstruktionsbild des unteren Stoßstangenteils

Abbildung 7 zeigt die fertig gedruckten Stoßstangenkomponenten. Der untere, flache Teil der Stoßstange wird mit zwei Schrauben an der oberen Komponente befestigt. Die gesamte Stoßstange wird ebenfalls an nur zwei Stellen mit der Karosserie verbunden. Zusätzlichen Halt bekommt die Stoßstange von der Halterung des Ultraschallboards.



Abbildung 7: Gedrucktes oberes und unteres Stoßstangenteil

Wie bereits erwähnt, wird das Ultraschallboard frontal am Fahrzeug montiert. Dazu wird eine Aufnahme benötigt, an welcher das Board befestigt werden kann. In den Abbildungen 8 und 9 sind die Konstruktionszeichnung und die fertig gedruckte Halterung abgebildet.

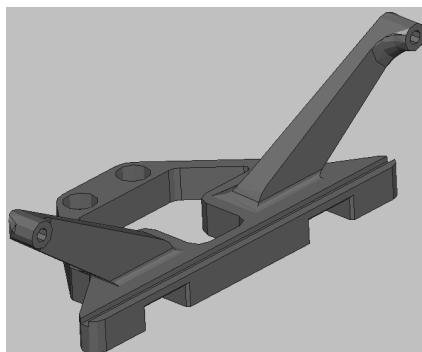


Abbildung 8: Konstruktionsbild der Halterung des Ultraschallboards



Abbildung 9: Gedruckte Halterung des Ultraschallboards

In den Abbildung 10 und 11 sind die am Fahrzeug fertig montierte Stoßstange und Ultraschallboard-Halterung abgebildet. Die Stellen, an denen diese Bauteile am Fahrzeug fixiert sind, sind in den Abbildungen farblich hervorgehoben.

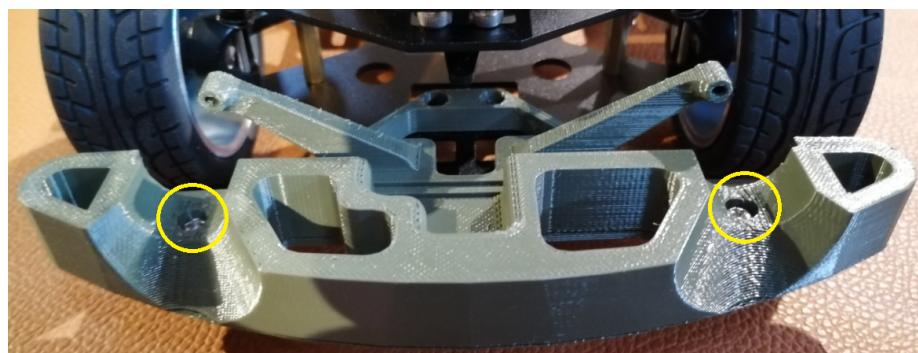


Abbildung 10: Draufsicht der Stoßstange und der Ultraschallboard-Halterung; Befestigungsschrauben der unteren Stoßstangenkomponente in gelb



Abbildung 11: Untersicht der Stoßstange und der Ultraschallboard-Halterung; Befestigungsschrauben der Stoßstange in blau, Schrauben der Ultraschallboard-Halterung in rot und Schrauben zur Befestigung der unteren Stoßstangenkomponente in gelb

2.2.2 Akku-Halterung und Seitenschweller

Der Akku für das Fahrzeug soll auf der unteren Ebene Platz finden, um den Schwerpunkt des Fahrzeugs niedrig zu halten. Damit der Akku einen festen Sitz hat und nicht beim Gasgeben oder Bremsen verrutscht, wird auf der unteren Ebene in der Mitte eine Halterung installiert (Konstruktionszeichnung in Abbildung 12). Die Akku-Halterung fungiert auch als Kabeldurchführung, damit die Drähte und Kabel sauber gebündelt werden können und nicht frei in der Luft geführt werden. In Abbildung 13 ist die fertig gedruckte Akku-Halterung zu sehen. Die Halterung wird von unten mit 4 Schrauben auf der Grundplatte des Fahrzeugs montiert.

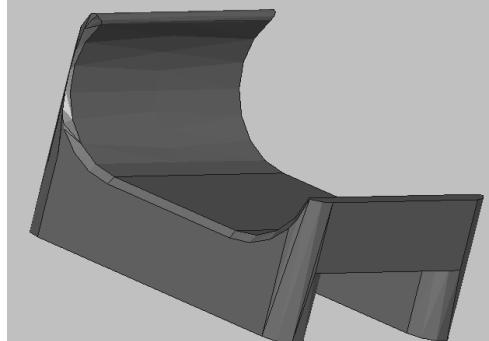


Abbildung 12: Konstruktionsbild der Akku-Halterung mit Kabeldurchführung



Abbildung 13: Gedruckte Akku-Halterung

Während die Akku-Halterung den Akku vor dem nach vorne und hinten rutschen schützt, tragen kleine Aussparungen an den Innenseiten der Seitenschweller zur Sicherung des Akkus zu beiden Seiten bei (Konstruktionszeichnung in Abbildung 14). Sie werden mit je drei Schrauben an der Grundplatte des Fahrzeugs befestigt. Die fertig gedruckten Seitenschweller sind in

Abbildung 15 abgebildet.

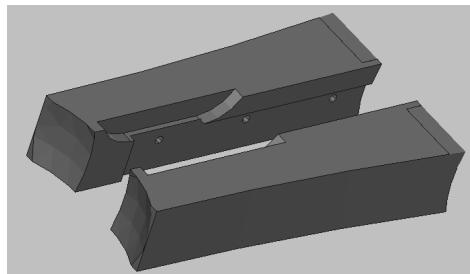


Abbildung 14: Konstruktionsbild der Seitenschweller, welche auch als Sicherung des Akkus zu den Seiten dienen

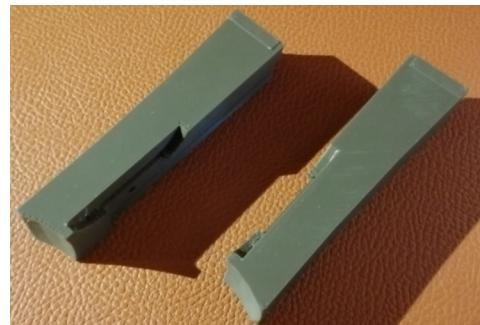


Abbildung 15: Gedruckte Seitenschweller

In Abbildung 16 sind die zum Fixieren des Akkus notwendigen Teile, die Seitenschweller und die Akku-Halterung, am Fahrzeug fertig montiert abgebildet. Der Akku wird quer zur Fahrtrichtung eingesetzt. So bleibt hinter dem Akku noch Platz für eine Verteilerplatine, an der die elektrischen Anschlüsse der verschiedenen Fahrzeugkomponenten angesteckt werden können.

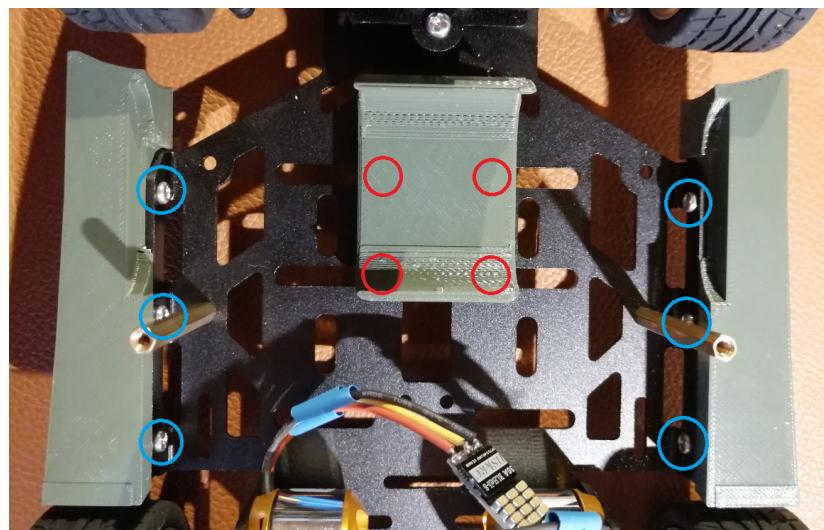


Abbildung 16: Montierte Akku-Halterung und Seitenschweller; Befestigungsschrauben der Seitenschweller in blau und Befestigungsschrauben der Akkuhalterung in rot (von unten, nicht sichtbar)

2.2.3 Halterung der Controllerplatine und des Bedienungs-Boards

Die Controllerplatine wird auf der oberen Fahrzeugplattform befestigt. Dafür wird eine Controller-Halterung benötigt (Konstruktionsbild siehe Abbildung 17). Oberhalb des Controllers findet das Display mit Bedientaster und Inkrementalgeber für die Menüsteuerung Platz (Bedienungs-Board). Zusätzlich zur Halterung der Platine (Konstruktionsbild siehe Abbildung 18) ist eine Abdeckung notwendig (Konstruktionsbild siehe Abbildung 19).

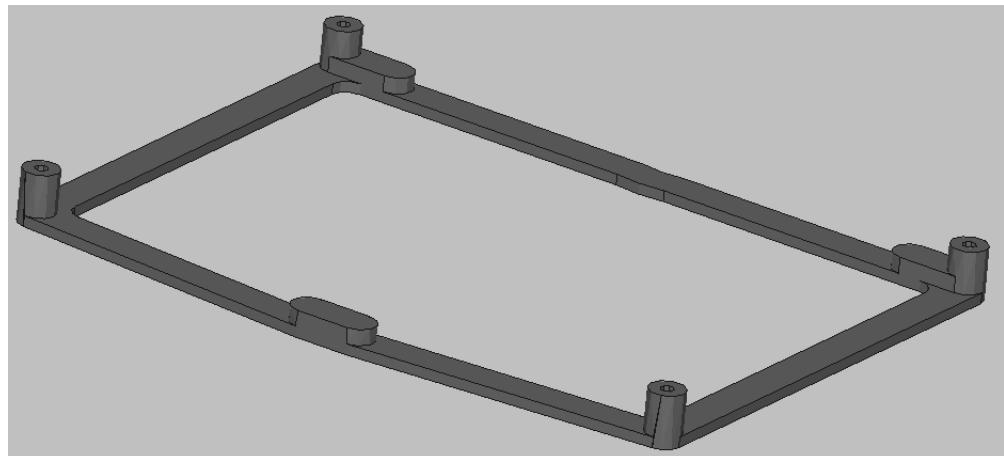


Abbildung 17: Konstruktionsbild der Controller-Halterung

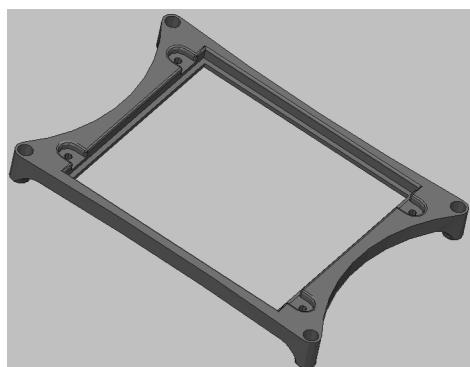


Abbildung 18: Konstruktionsbild der Platinen-Halterung für die Fahrzeugbedienung

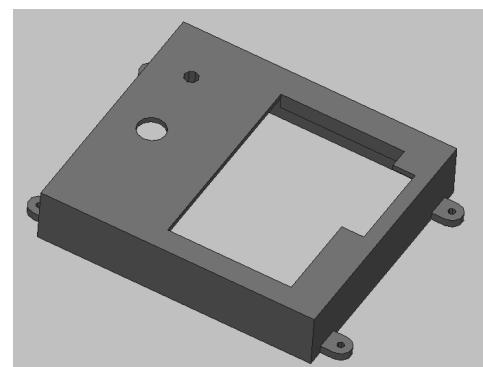


Abbildung 19: Konstruktionsbild der Abdeckung für die Fahrzeugbedienung

Zum Bedienen des Drucktasters auf der Platine wird eine Verlängerung benötigt, damit der Knopf außerhalb der Abdeckung betätigt werden kann (Konstruktionszeichnung siehe Abbildung 20). In den Abbildungen 21 bis 22 sind die fertig gedruckten Teile, die Controller-Halterung, die Platinen-Halterung und die Abdeckung, abgebildet.

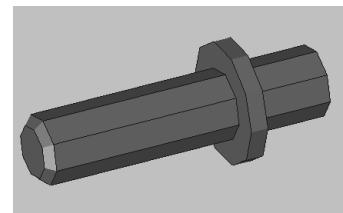


Abbildung 20: Konstruktionsbild der Drucktasters



Abbildung 21: Gedruckte Controller-Halterung



Abbildung 22: Gedruckte Drucktaster-Verlängerung

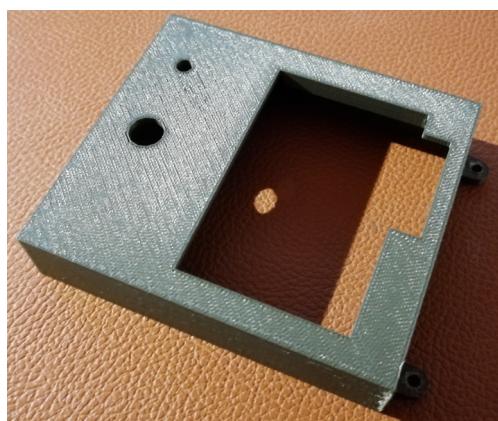


Abbildung 23: Gedruckte Platinen-Abdeckung des Bedienungsboards



Abbildung 24: Gedruckte Platinen-Halterung für das Bedienungs-Board

Die Montage der Teile erfolgt, wie bereits erwähnt, auf der oberen Fahrzeugebene. In den Abbildungen 25 und 26 sind die fertig montierten Komponenten für die Befestigung des Controllers und die der Platine für die Fahrzeugbedienung abgebildet. Damit für die Komponenten auf der Controllerplatine ausreichend Platz ist, wird die Platinen-Halterung des Bedienungs-Boards über Abstandshalterungen montiert. Die Befestigungsschrauben für den Controller sind in den Abbildungen mit roten Kreisen, die der Platinenhalterung mit blauen Kreisen und die der Abdeckung mit gelben Kreisen hervorgehoben.

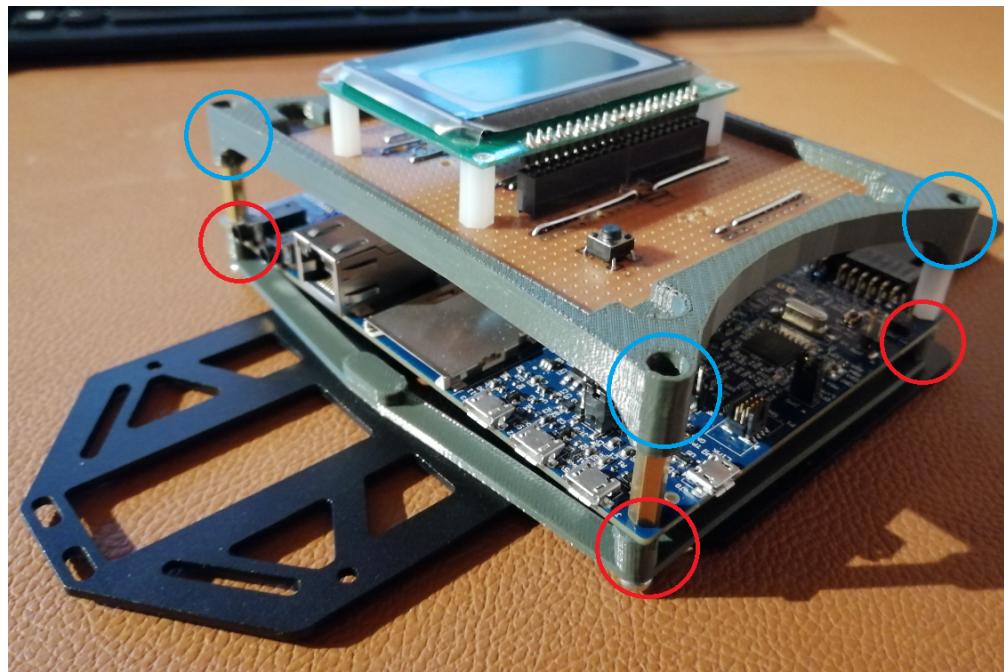


Abbildung 25: Fertig montierte Controller- und Bedienungs-Board-Halterung; Befestigungsschrauben des Controllers in rot, Befestigung der Platinen-Halterung in blau

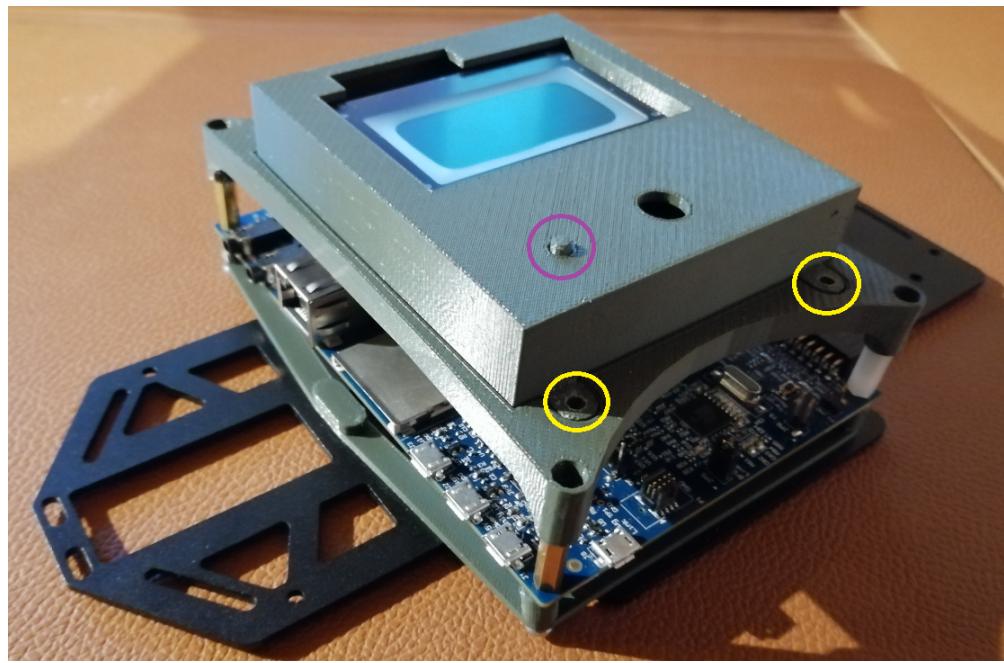


Abbildung 26: Fertig montierter Controlleraufbau mit Controller und Bedienungs-Board sowie deren zugehörige Halterungen und Abdeckung; Befestigungsschrauben der Abdeckung in gelb, Taster in violett

2.2.4 Halterung für die Motorcontroller

Die Motorcontroller, die vor die BLDC-Motoren geschaltet sind, werden auf der Grundplatte befestigt. Mit zwei dreiecksförmigen Halterungen werden die Motorcontroller an jeweils drei Stellen an der Grundplatte angeschraubt. Die Konstruktionszeichnung einer solchen Motorcontroller-Halterung ist in Abbildung 27 und das Druckergebnis in Abbildung 28 einsehbar.

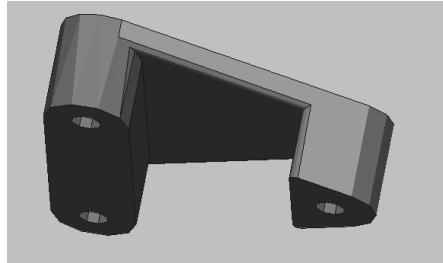


Abbildung 27: Konstruktionsbild einer Motorcontroller-Halterung



Abbildung 28: Gedruckte Motorcontroller-Halterungen

Da die Motorcontroller erst dann am Fahrzeug befestigt werden, sobald alle Komponenten der Grundplatte fertig gestellt wurden und die Drehzahlmessung implementiert wurde, für deren Programmierung an den Ausgängen der Motorcontroller noch Drähte angelötet werden müssen, existiert noch kein Bild der Montage der Motorcontroller mit den Halterungen.

Abbildung 29 zeigt die Motorcontroller-Halterungen im Einsatz. Die Controller sind am Boden fixiert und durch den transparenten Schrumpfschlauch vor Kurzschlüssen über die metallische Grundplatte geschützt.

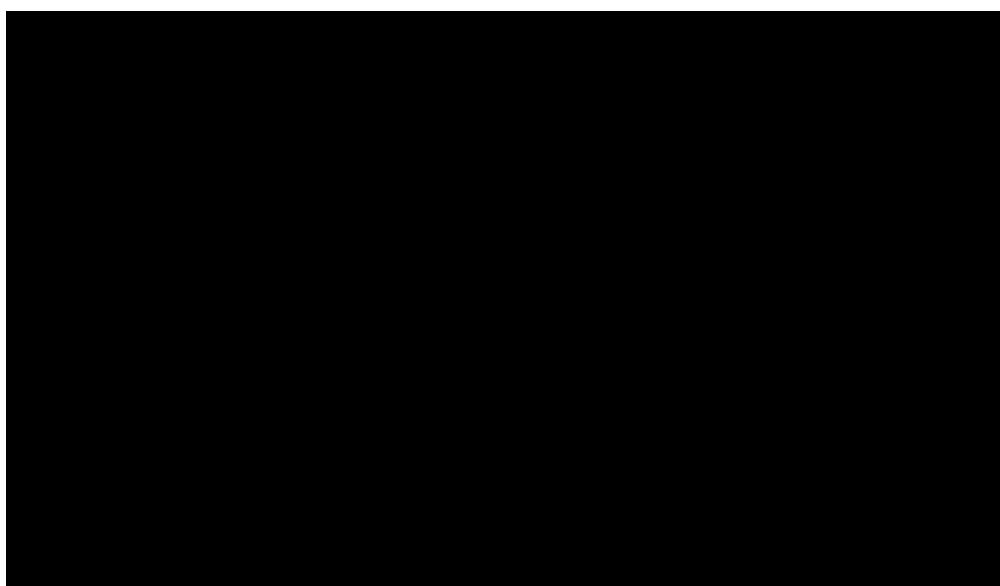


Abbildung 29: Die auf der Grundplatte montierten Motorcontroller

2.2.5 Einfache Platinenhalterungen

Für die Befestigung der Verteiler-Platine auf der unteren Ebene wird eine Halterung benötigt. Mit vier einfachen Halterungen (Konstruktionsbild siehe Abbildung 30) kann die Lochrasierplatine an vier Stellen mit etwas Abstand zur Grundplatte auf dieser befestigt werden. Die fertig gedruckten Platinen-Halterungen sind in Abbildung 31 abgebildet.

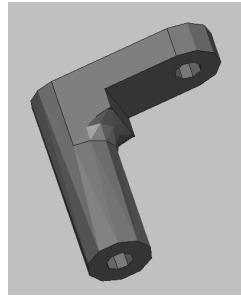


Abbildung 30: Konstruktionsbild einer einfachen Platinenhalterung



Abbildung 31: Gedruckte Platinenhalterungen

Da die Verteilerplatine für die untere Fahrzeugebene, für deren Montage die einfachen Platinenhalterungen gedacht sind, noch nicht fertig gestellt ist, existiert noch keine Abbildung mit den im Einsatz befindlichen Halterungen. In Abbildung 32 ist die fertig montierte Verteiler-Platine sichtbar. Sie ist mit vier einfachen Platinen-Halterungen an der Grundplatte des Fahrzeugs montiert. Der aufgrund der Halterungen resultierende Abstand zur Grundplatte ermöglicht die Montage der Motorcontroller-Halterungen unter der Verteiler-Platine.

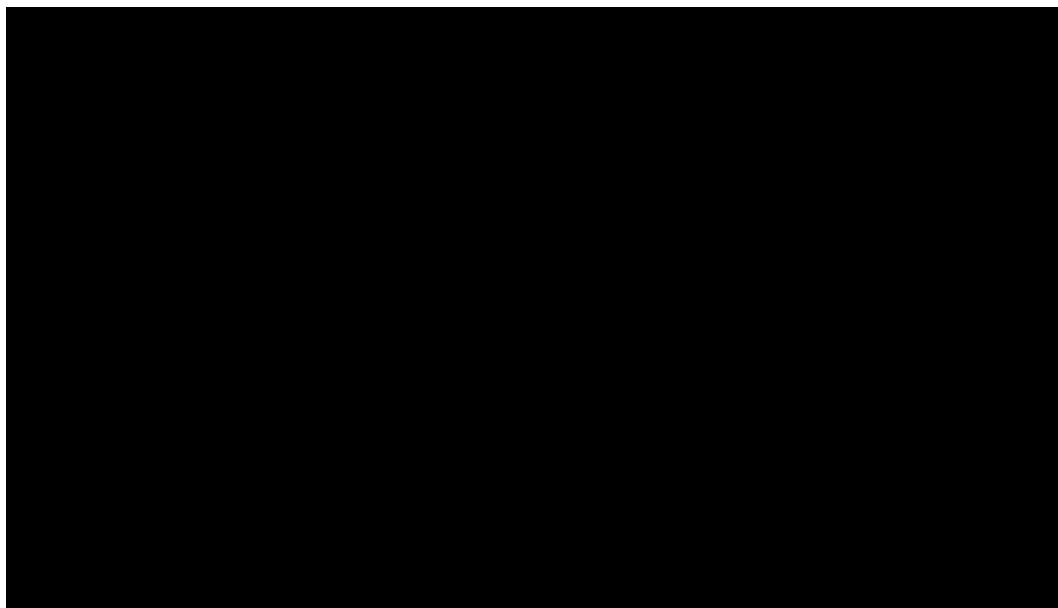


Abbildung 32: Mit den einfachen Platinenhalterungen montierte Verteiler-Platine

2.2.6 Kamerahalterung

Die Kamera soll sich bei Kollisionen nicht in ihrer Position verstellen können. Deshalb wird eine eigens dafür konstruierte Halterung gedruckt. Im Gegensatz zu den übrigen 3D-Druckteilen ist die Kamerahalterung selbst konstruiert und nicht vom Vorgänger übernommen. Die Konstruktionsbilder der beiden Einzelteile der Kamerahalterung sind in den Abbildungen 33 und 34 einsehbar. Die ineinandergreifenden Zacken garantieren dabei eine stabile Kameraposition.

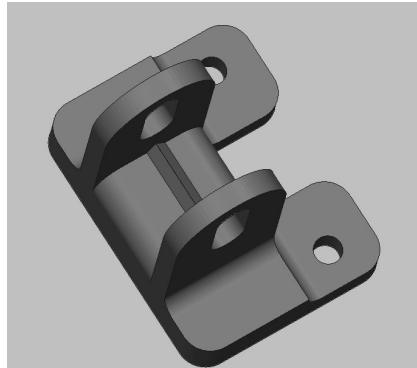


Abbildung 33: Kameramontage-Komponente der Kamera-Halterung

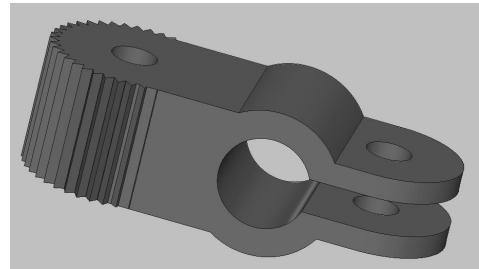


Abbildung 34: Stangenmontage-Komponente der Kamera-Halterung

Schon bei der Konstruktion zeichnet sich ein potentielles Problem für den Druck der Teile ab. Die Verzahnung ist sehr fein gewählt, um die Kamera auch genau einzustellen zu können. Für einen 3D-Drucker kann das bedeuten, dass dieser an seine Grenzen in der Genauigkeit der Fertigung kommt. Die Sorge, dass der verwendete 3D-Drucker, welcher Acrylnitril-Butadien-Styrol Copolymer (ABS-Kunststoff) verwendet, die Verzahnung nicht fein genug fertigen kann, ist zum Teil begründet, da von je zwei gedruckten Teilen eines jeder Komponente von ungenügender Genauigkeit ist. Die beiden in ausreichender Qualität gefertigten Teile sind in Abbildung 35 abgebildet. Mit der Erwartung, genauere Ergebnisse zu erzielen, müsste auf den 3D-Drucker mit Stereolithographie-Verfahren (SLA-Drucker) der Fakultät Maschinenbau zurückgegriffen werden. In diesem Fall ist das allerdings nicht notwendig.

Bei der Montage taucht allerdings ein anderes, unerwartetes Problem auf. Bei der Konstruktion der Teile wurde an einer Stelle entweder ein falsches Maß aufgenommen oder das Maß falsch in die CAD-Zeichnung eingegeben. Die betreffende Stelle ist die Freifläche zwischen den beiden Befestigungsstellen der Kamera, welche um etwa 1,5mm zu kurz geraten ist. Nach einer Anpassung mit einer Säge sind die Teile ohne Probleme verwendbar und die Komponente muss nicht abermals gedruckt werden. Zur Vermeidung dieser Anpassung für Nachfolgemodelle des Fahrzeugs ist die Konstruktionszeichnung angepasst.

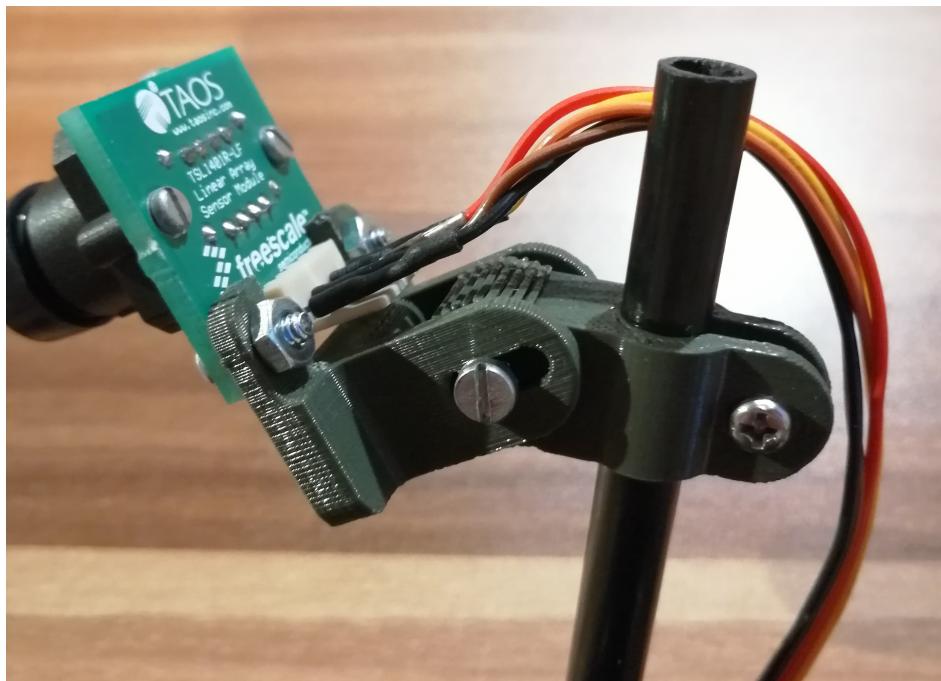


Abbildung 35: Fertig montierte Kamera mit Kamerahalterung

3 Controllerplatine und Programmiertool

Für dieses Projekt wird die Controllerplatine LPCXpresso54608 von NXP Semiconductors verwendet. Das Herzstück dieser Platine ist der auf dem ARM Cortex-M4 basierende Mikrocontroller LPC54608. Dieser Controller wird wegen seiner schnellen Taktfrequenz und seiner großen Anzahl an Peripherie für die Ansteuerung der Motoren und des Servos sowie für die Sensorik zur Bestimmung des Streckenverlaufs verwendet. Der Vorteil in einer fertigen Controllerplatine liegt darin, dass der Schaltplan, um die Funktionsweise des Controllers sicherzustellen, sowie das dazugehörige Layout professionell erstellt sind, wodurch dieser aufwendige Schritt bei der Entwicklung des Autos entfällt. Außerdem ist darauf bereits ein USB basierter J-Link Debugger zur Programmierung und Fehlersuche verbaut.

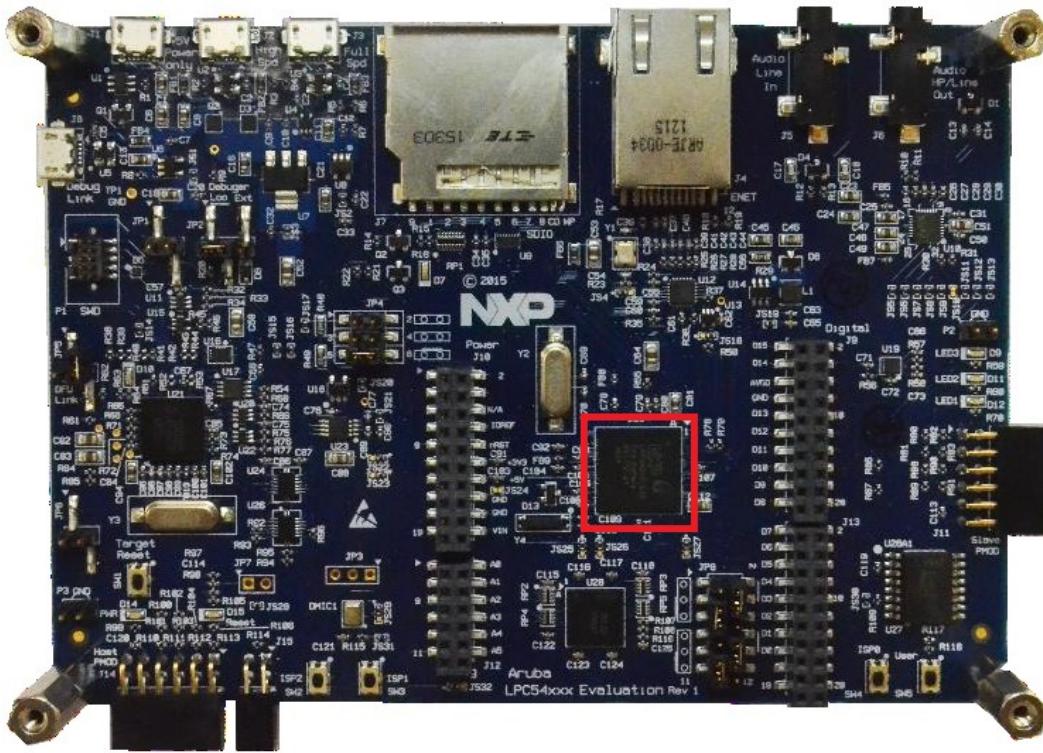


Abbildung 36: Draufsicht auf die Controllerplatine mit dem Mikrocontroller LPC54608 (in rot hervorgehoben) als Hauptbestandteil [Sem19]

Zur Programmierung des Controllers dient das auf Eclipse basierende Tool „MCUXpresso“, welches von NXP Semiconductors zur Verfügung gestellt wird. Darin sind die für den verwendeten Controller notwendigen Konfigurationsdateien, wie beispielsweise das Startup-Skript und einige Beispiele, bereits enthalten. Das erleichtert den Einstieg in die Programmierung erheblich. Außerdem kann das Konfigurationstool „MCUXpresso Config Tools“ verwendet werden. Mit diesem kann die Initialisierung der Peripheriebausteine über eine graphische Benutzeroberfläche vorgenommen werden.

4 Antrieb des Fahrzeugs

Wie in Kapitel 2.1 beschrieben, wird das Fahrzeug von zwei BLDC-Motoren angetrieben, die im Folgenden genauer erklärt werden. Zusätzlich dazu wird in diesem Kapitel auch näher auf die Montage der Antriebskomponenten, die Programmierung des Antriebsbausteins, die Konfiguration der Motorcontroller und die Drehzahlmessung eingegangen.

4.1 BLDC-Antrieb und Motorcontroller

BLDC-Motoren sind im Wesentlichen wie permanent erregte Synchronmaschinen aufgebaut. Sie besitzen Magnete im Rotor und Einzelzahnwicklungen im Stator. Da solche Motoren häufig im Remote-Control (RC)-Bereich Einsatz finden, gibt es zu deren Ansteuerung bereits vorgefertigte Bausteine, sogenannte Electronic Speed Controller (ESC). Diese haben zwei Signalpins für ein Pulsweitenmodulation (PWM)-Signal und zwei Anschlüsse zur Spannungsversorgung des Motorcontrollers, aus welchen die drei Strangspannungen für die Phasen des Antriebs generiert werden. Aus diesen Strangspannungen kann im übrigen auch die Motordrehzahl ermittelt werden, was im Rahmen dieser Projektarbeit auch eines der Ziele darstellt. An den Signalpins wird ein PWM-Signal mit einer Frequenz von 50Hz angelegt, dessen Pulsbreite zwischen 1ms und 2ms liegen darf. Daraus resultiert ein Tastgrad zwischen 5% (Stillstand) und 10% (maximal erreichbare Drehzahl). Die maximale Drehzahl des Antriebs hängt von der Spannung des Akkus ab. Je geringer diese Spannung ist, desto geringer ist die maximal erreichbare Drehzahl. Der Anschluss der Motoren erfolgt nach der in Abbildung 37 gezeigten Weise.

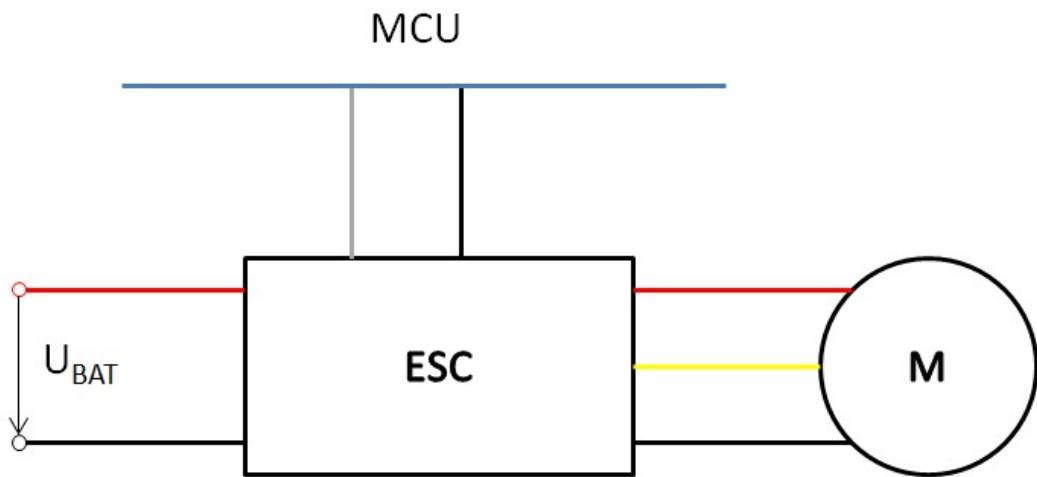


Abbildung 37: Skizze zur Beschaltung eines ESCs und BLDC-Motors; Versorgungsspannung des ESCs in rot und schwarz (Masse), PWM-Signalleitungen in grau und schwarz (Masse) und Anschlussleitungen des BLDC-Motors in rot, gelb und schwarz

4.2 Montage der Antriebskomponenten

Die Komponenten des Fahrzeugantriebs, der über zwei BLDC-Motoren realisiert ist, sind auf der unteren Fahrzeugebene montiert. Die Motoren selbst sind mit zwei Schrauben an der Karosserie befestigt. An der Welle der Motoren befindet sich je ein Zahnrad mit 13 Zähnen, welches ein weiteres Zahnrad mit 90 Zähnen antreibt, das an der Antriebswelle befestigt ist. Das heißt, dass sich die Reifen bei 6,923 Umdrehungen der Motorwelle genau einmal drehen.

In Abbildung 38 sind die montierten Komponenten des linken Antriebs abgebildet. Die Befestigungsschrauben der Motoren sind dabei in rot hervorgehoben, die Befestigungsschrauben des Zahnrads der Antriebswelle in blau und die Befestigung des Reifens in orange. Das Zahnrad der Motorwelle ist durch Erhitzen geweitet und auf die Welle aufgesetzt worden. Zur Sicherheit dient hier etwas Sekundenkleber dazu, dass sich das Zahnrad auf der Welle nicht durchdreht. Sekundenkleber ist hier völlig ausreichend, da aufgrund der Übersetzung nur 1/7 des auf die Reifen wirkenden Moments auf das Zahnrad der Motorwelle wirkt.



Abbildung 38: Montage der BLDC-Motoren und Übersetzung auf die Antriebsachse; In rot die Befestigung des BLDC-Motors, in blau die Befestigung des Zahnrads an der Antriebswelle und in orange die Befestigung des Reifens

4.3 Konfiguration der Motorcontroller

Die beiden Motorcontroller erwarten nach dem Zuschalten der Spannungsversorgung eine Initialisierungssequenz. Das Power-On-Ereignis wird vom ESC mit drei Tönen (tief, mittel, hoch) signalisiert. Im Anschluss daran soll der Wert an der Signalleitung größer als 0% sein. Das bedeutet, dass ein PWM-Signal angelegt werden muss, welches eine Drehzahl $N \geq 0\text{rpm}$ repräsentiert. Der ESC quittiert das Erkennen des PWM-Signals mit einem tiefen Ton. Die Initialisierungssequenz ist allerdings erst dann beendet, wenn das PWM-Signal an der Signalleitung zuerst vergrößert und dann auf 0% verringert wird ($N = 0\text{rpm}$). Dabei gibt der ESC einen letzten, hohen Ton von sich. Der Ablauf der Initialisierungssequenz ist in Abbildung 39 dargestellt. Nach dem letzten Ton ist die Initialisierung beendet und der BLDC-Motor dreht sich in Abhängigkeit des an der Signalleitung anliegenden PWM-Signals.

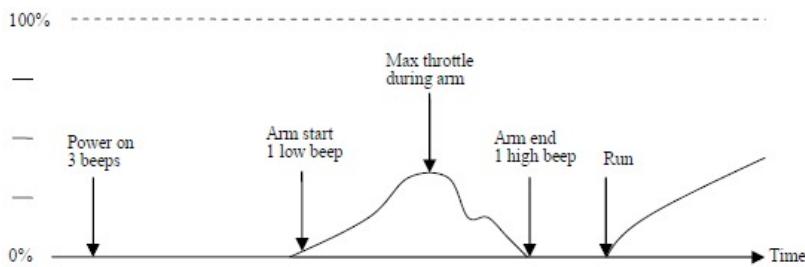


Abbildung 39: Wert an der Signalleitung für die Initialisierungssequenz über der Zeit; 0% entspricht dem PWM-Tastgrad für den Stillstand und 100% dem für die maximal erreichbare Drehzahl

Da die ESCs individuell konfiguriert werden können und in der Dokumentation zu wenige Angaben gemacht werden, ist der Tastgrad für die Werte 0% (Stillstand) und 100% (maximal erreichbare Drehzahl) unbekannt. Deshalb ist es nicht möglich zu wissen, welche PWM-Tastgrade für die Initialisierungssequenz verwendet werden müssen. Über die Konfiguration der ESCs können die Grenzen für 0% und 100% selbst festgelegt werden. Für das Flashen der Motorcontroller wird die Software „BLHeliSuite“ verwendet. Die Verbindung zwischen den ESCs und der Software stellt ein Arduino Nano her.

Der Arduino Nano muss zuvor mit einer neuen Software beschrieben werden. Um die ESCs flashen zu können wird der Signalpin des zu programmierenden ESC mit dem Pin D3 und der Massepin der Signalleitung mit einem Massepin des Arduino Nano verbunden. Danach wird in der Software „BLHeliSuite“ im Reiter „Make Interface“ eine neue Schnittstelle erstellt (siehe Abbildung 40). Auf der rechten Seite des Programm-Fensters wird ein Arduino Nano als Schnittstelle eingerichtet. Mit einem Klick auf den Button „Arduino 4way-interface“ wird die neue Software, durch die die ESCs neu konfiguriert werden können, auf den Arduino Nano geladen.

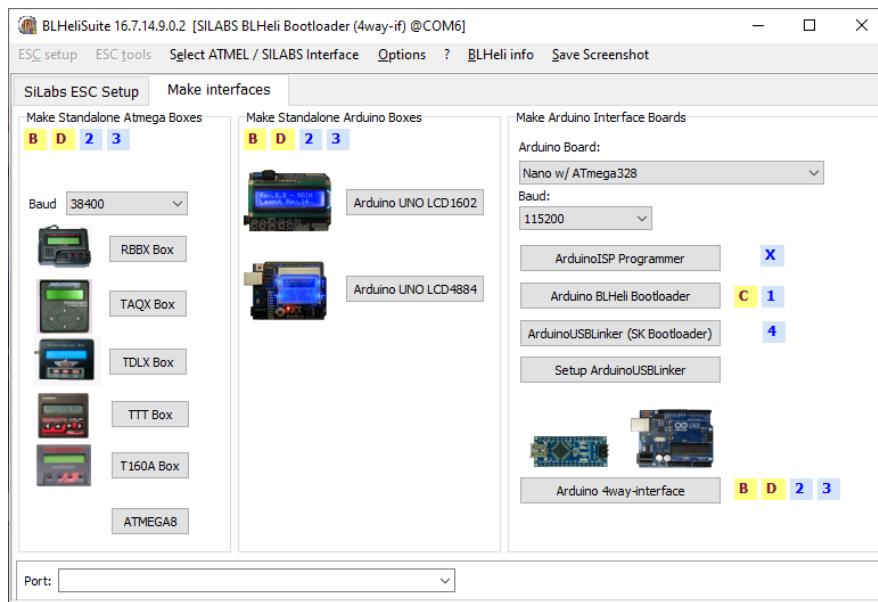


Abbildung 40: Programmierung des Arduino Nano zur ESC-Konfiguration mit der Software BLHeliSuite

Nach dem Programmieren des Arduino Nano wird die Kommunikation der BLHeliSuite-Software mit dem ESC hergestellt. Über die Schaltfläche „Read Setup“ werden die voreingestellten Parameter des ESC ausgelesen. Im nächsten Schritt werden die Zeiten für die Werte „PPM min Throttle“ (0% Aussteuergrad) und „PPM max Throttle“ (100% Aussteuergrad) auf 1100µs und 1900µs angepasst. Alle vorgenommenen Einstellungen sind in Abbildung 41 einsehbar. Die Werte werden über die Betätigung der Schaltfläche „Write Setup“ auf den ESC geladen. Da jetzt die Werte für 0% und 100% Aussteuerung bekannt sind, können die einzelnen Schritte der Initialisierung im Programm des Fahrzeugs abgearbeitet werden.

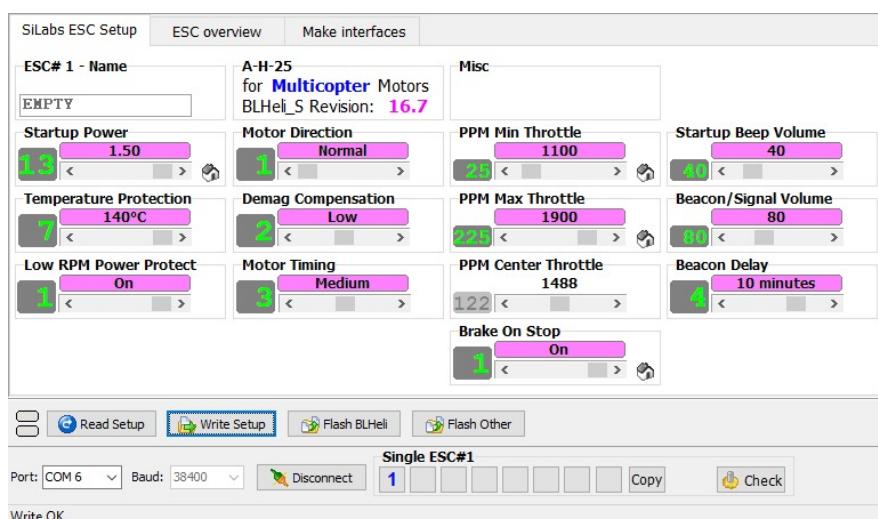


Abbildung 41: Konfiguration der ESCs mit der Software „BLHeliSuite“

4.4 Programmierung des Antriebsbausteins

Der Antriebsbaustein der Software ist in zwei Dateien unterteilt, die Dateien „drive.c“ und „drive.h“. Die Datei „drive.h“ enthält alle relevanten Bibliotheken und Prototypen für die Datei „drive.c“.

Außer der Einbindung der Bibliotheken und der Prototypen der Funktionen aus der Datei „drive.c“ sind hier auch die Parameter für die Initialisierungssequenz der ESCs und für die Initialisierung des Timers für die PWM-Signale hinterlegt (Abbildung 42).

```
#define SOURCE_CLOCK CLOCK_GetFreq(kCLOCK_CoreSysClk)

/* Definition of channel 2 duty --> on ticks*/
#define CTIMER3_PWM_0_DUTY 0

/* Definition of PWM period --> whole period ticks*/
#define CTIMER3_PWM_PERIOD 4400000

/* Definition of channel 0 ID */
#define CTIMER3_PWM_LEFT_CHANNEL kCTIMER_Match_0

/* Definition of channel 2 ID */
#define CTIMER3_PWM_RIGHT_CHANNEL kCTIMER_Match_2

//Init High value for BLDC Init Sequence 1,5ms
#define BLDC_PWM_INIT_HIGH_VALUE 330000

//Init Low value for BLDC Init Sequence 1,0ms
#define BLDC_PWM_INIT_LOW_VALUE 220000

//Stop Throttle value (N = 0rpm) below 1,1ms
#define BLDC_PWM_STOPTHROTTLE 240000

//FULL Throttle value (N = MAX rpm) 1,9ms
#define BLDC_PWM_FULLTHROTTLE 418000
```

Abbildung 42: Relevante Zeilen der Datei „drive.h“ mit den Parametern für die Initialisierungssequenz der ESCs und für die Initialisierung des PWM-Timers

Für die PWM-Periodendauer wird bei der Initialisierung ein Wert von 4.400.000 Taktzyklen festgesetzt, woraus mit einer Central Processing Unit (CPU)-Taktfrequenz von 220MHz (220.000.000 Takte pro Sekunde) eine Periodendauer von 20ms resultiert. Die Pulsbreite wird während des Programmablaufs regelmäßig überschrieben.

Der ESC-Initialisierungswert für den Stillstand („BLDC_PWM_INIT_LOW_VALUE“, 220.000) entspricht hier einer PWM-Pulsbreite von 1,0ms und der Initialisierungswert für die in etwa mittlere Aussteuerung („BLDC_PWM_INIT_HIGH_VALUE“, 330.000) einer Breite von 1,5ms. Die volle Aussteuerung der Motoren wird, wie über die „BLHeliSuite“ festgelegt, bei einer Pulsdauer von 1,9ms erreicht („BLDC_PWM_FULLTHROTTLE“, 418.000). Eine Drehzahl von N = 0rpm wird theoretisch über die bei der ESC-Konfiguration festgelegten 1,1ms ermöglicht (242.000). Da das PWM-Signal leicht abweicht, wird ein etwas geringerer Wert veranlagt („BLDC_PWM_STOPTHROTTLE“, 240.000 = 1,09ms). Damit wird sicher gestellt, dass sich die Räder im Stillstand nicht drehen.

Auch die beiden PWM-Timer (einer je Motorcontroller) benötigen bei der Initialisierung einige Parameter, deren Werte in der Datei „drive.h“ festgelegt sind (PWM-Periodendauer, PWM-Pulsdauer, Channel). Die Channel werden auf das Timer Match Register 2 (rechter Antrieb, „kCTIMER_Match_2“) und auf das Timer Match Register 0 (linker Antrieb, „kCTIMER_Match_0“) festgelegt, was bei dem verwendeten Controller den Pins P0.27 (rechts) und P3.10 (links) entspricht. Der Pin P3.10 wird auf der Controllerplatine über den Pin 7 und der Pin P0.27 über den Pin 12 der Buchsenleiste J13 nach außen geführt. Der Anschluss der Motorcontroller ist über den Anhang „Anhang 1: Schaltplan“ nachvollziehbar.

```

const ctimer_config_t BLDC_config = {
    .mode = kCTIMER_TimerMode, /* TC is incremented every rising APB bus clock edge */
    .input = kCTIMER_Capture_0, /*!< Timer capture channel 0 */
    .prescale = 0 /*!< Prescale value 0 --> */
};

void BLDC_Init(void)
{
    CTIMER3_Init(); //Timer init function
    CTIMER3->MCR |= CTIMER_MCR_MR0RL_MASK;
    //Reload MR0 with the contents of the shadow register when the timer counter (TC) is reset to 0
    IOCON->PIO[3][10] &= 0xFFFFFFFF0; //clear FUNC bits of P3.10
    IOCON->PIO[3][10] |= 0x3; //set FUNC bits to CTIMER3_MAT0 function ALT3 P3.10
    GPIO->DIR[3] |= 1<<10; //set P3.10 to output
    //initialize MSR register with BLDC_INIT_LOW value (1.0 ms on time)
    CTIMER3->MSR[0] = CTIMER3_PWM_PERIOD - BLDC_PWM_INIT_LOW_VALUE;

    CTIMER3->MCR |= CTIMER_MCR_MR2RL_MASK;
    //Reload MR2 with the contents of the shadow register when the timer counter (TC) is reset to 0
    IOCON->PIO[0][27] &= 0xFFFFFFFF0; //clear FUNC bits of P0.27
    IOCON->PIO[0][27] |= 0x3; //set FUNC bits to CTIMER3_MAT2 function ALT3 P0.27
    GPIO->DIR[0] |= 1<<27; //set P0.27 to output
    //initialize MSR register with BLDC_INIT_LOW value (1.0 ms on time)
    CTIMER3->MSR[2] = CTIMER3_PWM_PERIOD - BLDC_PWM_INIT_LOW_VALUE;

    //create initialization task for ESCs
    if (xTaskCreate(BLDC_Init_Task, "BLDC_Init_Task", configMINIMAL_STACK_SIZE + 100, NULL, 1, NULL) != pdPASS)
    {
        LED3_ON();
    }
}

void CTIMER3_Init(void)
{
    /* CTIMER1 peripheral initialization */
    CTIMER_Init(CTIMER3, &BLDC_config); //((CTIMER_Type, ctimer_config_t)
    /* PWM channel 0 of CTIMER3 peripheral initialization */
    CTIMER_SetupPwmPeriod(CTIMER3, CTIMER3_PWM_LEFT_CHANNEL, CTIMER3_PWM_PERIOD, CTIMER3_PWM_PERIOD - CTIMER3_PWM_0_DUTY, false);
    //((CTIMER_Type, ctimer_match_t,uint32_t,uint32_t)
    /* PWM channel 2 of CTIMER3 peripheral initialization */
    CTIMER_SetupPwmPeriod(CTIMER3, CTIMER3_PWM_RIGHT_CHANNEL, CTIMER3_PWM_PERIOD, CTIMER3_PWM_PERIOD - CTIMER3_PWM_0_DUTY, false);
    //((CTIMER_Type, ctimer_match_t,uint32_t,uint32_t)
    /* Start the timer */
    CTIMER_StartTimer(CTIMER3); //Timer starten
}

```

Abbildung 43: Funktionen BLDC_Init und CTIMER3_Init der Datei „drive.c“

Die Datei „drive.c“ enthält die Funktionen zur Initialisierung der für die Verwendung der Antriebe notwendigen Controller-Peripherie (siehe Abbildung 43) und zur Initialisierung der Motorcontroller (siehe Abbildung 44).

In der Funktion BLDC_Init wird zuerst die Funktion CTIMER3_Init aufgerufen, welche die beiden vorher festgelegten Kanäle des Timers C3 („kCTIMER_Match_0“ und „kCTIMER_Match_2“) mit den in der Datei „drive.h“ festgelegten Parametern als PWM-Timer mit einer Periodendauer von 20ms und einer Pulsdauer von 0ms initialisiert. Im Anschluss daran wird einzeln für beide Kanäle festgelegt, dass bei einem Timer-Überlauf die neuen Daten für die Pulslängen aus den Shadow-Registern geladen werden sollen. Zum Ändern

der Geschwindigkeit eines der beiden Antriebe muss deshalb lediglich ein neuer Wert in das Shadow-Register geschrieben werden. Hier muss allerdings aufgepasst werden, da das Register nicht die Pulsbreite (On-Time) sondern die Off-Time erwartet. Deshalb muss der Wert, der eingetragen wird, der Periodendauer abzüglich der Pulsdauer entsprechen. Zusätzlich werden in der Funktion BLDC_Init auch die Pins P3.10 und P0.27 für die Verwendung als PWM-Ausgang des Timers C konfiguriert. Am Ende wird noch der erste ESC-Initialisierungswert in die beiden Shadow-Register geschrieben, bevor die ESC-Initialisierungsfunktion aufgerufen wird (siehe Abbildung 44).

```
//Initialize ESCs
void BLDC_Init_Task(void *pvParameters)
{
    while(1){

        //Initialize Sequence for BLDC-Motors
        //low throttle value
        CTIMER3->MSR[0] = CTIMER3_PWM_PERIOD - BLDC_PWM_INIT_LOW_VALUE;
        CTIMER3->MSR[2] = CTIMER3_PWM_PERIOD - BLDC_PWM_INIT_LOW_VALUE;
        vTaskDelay(2000);
        //half throttle value
        CTIMER3->MSR[0] = CTIMER3_PWM_PERIOD - BLDC_PWM_INIT_HIGH_VALUE;
        CTIMER3->MSR[2] = CTIMER3_PWM_PERIOD - BLDC_PWM_INIT_HIGH_VALUE;
        vTaskDelay(2000);
        //low throttle value
        CTIMER3->MSR[0] = CTIMER3_PWM_PERIOD - BLDC_PWM_INIT_LOW_VALUE;
        CTIMER3->MSR[2] = CTIMER3_PWM_PERIOD - BLDC_PWM_INIT_LOW_VALUE;
        vTaskDelay(3500);

        //suspend Task
        vTaskSuspend(NULL);
    }
}
```

Abbildung 44: Funktion BLDC_Init_Task der Datei „drive.c“ zum Durchlaufen der Initialisierungssequenz der ESCs

Der Task BLDC_Init_Task beginnt mit dem Befüllen der Shadow-Register mit dem ersten Initialisierungswert. Nach einer Wartezeit von 2s wird der zweite Initialisierungswert in die Register geschrieben. Ebenfalls nach einer Zeit von 2s wird dann wieder der erste Initialisierungswert in die Shadow-Register geschrieben und die Initialisierung der ESCs ist abgeschlossen. Die Wartezeiten sind notwendig, damit die ESCs Zeit haben, die Änderungen zu erfassen.

4.5 Drehzahlmessung

4.5.1 Erörterung der Notwendigkeit einer Drehzahlmessung

Die Quellspannung von Akkus verringert sich mit steigender Betriebszeit, weshalb bei vorherigen Fahrzeug-Versionen eine Drehzahlmessung benötigt wird, um über eine Regelung die Spannungsabhängigkeit der Drehzahl bei Gleichstrommotoren (DC-Motoren) ausgleichen zu können. Probleme hat diese Abhängigkeit vor allem beim Überfahren eines Hügels bereitet. Da die in den vorangegangenen Fahrzeugmodellen verwendeten DC-Motoren durch BLDC-Motoren ersetzt wurden, stellt sich allerdings wieder die Frage nach der Notwendigkeit einer Drehzahlmessung. Die Variation der Drehzahl funktioniert bei DC-Motoren über die Änderung der Betriebsspannung, was auch die Abhängigkeit von der Versorgungsspannung erklärt. Die Drehzahlvariation bei einem BLDC-Motor wird hingegen über eine Frequenzänderung der Strangspannungen realisiert. Zur Klärung der Frage nach einer Spannungsabhängigkeit der BLDC-Motoren, wird eine Messung am Prüfstand durchgeführt.

Für die Durchführung der Messung wird ein bereits vorhandener Prüfstand für Modelfahrzeuge verwendet, welcher im Rahmen einer Bachelor-Abschlussarbeit an der HAW Landshut erstellt wurde. In Abbildung 45 ist der Messaufbau zu sehen. Mithilfe eines Drahtes wird das Heck des Fahrzeugs so an der Grundplatte des Prüfstands befestigt, dass die Reifen einen besseren Halt auf den Rollen des Prüfstands haben. An den Vorderreifen ist das Fahrzeug ebenfalls festgeschnallt.

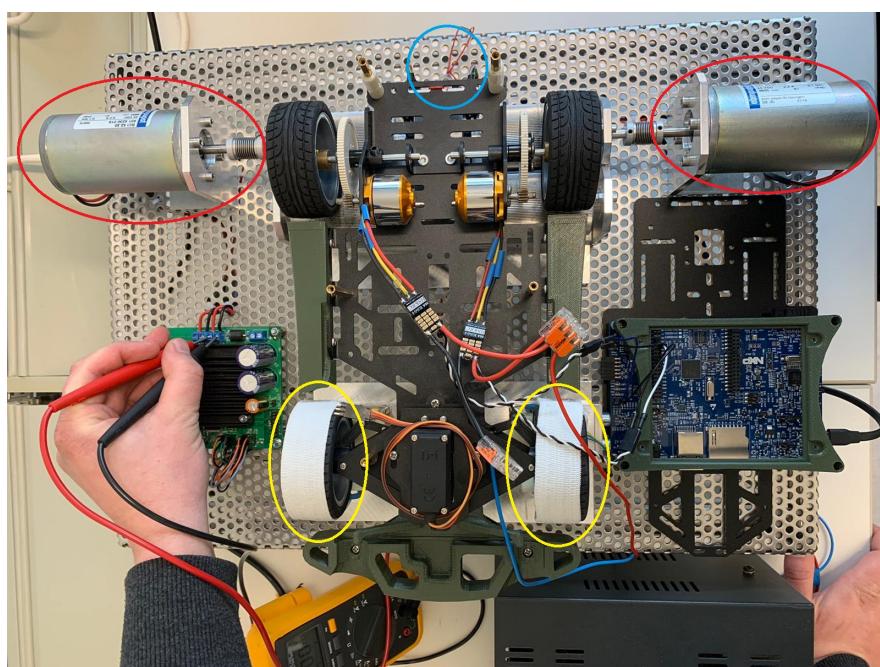


Abbildung 45: Messaufbau zur Prüfung der Spannungsabhängigkeit der BLDC-Motoren; in blau die Befestigung des Hecks an der Grundplatte des Prüfstands, in gelb die Befestigung der Vorderreifen am Prüfstand, in rot die DC-Motoren zur Spannungsmessung

Zur Überprüfung einer eventuellen Drehzahlabhängigkeit der BLDC-Motoren von der Versorgungsspannung wird die Spannung an den DC-Motoren des Prüfstands gemessen, weil diese direkt proportional zur Drehzahl ist (siehe Gleichung 1). Da die Drehmomentkonstante k_i bei hohen Drehzahlen leicht einbricht, ist die Abhängigkeit der gemessenen Spannung von der Drehzahl allerdings nur annähernd linear. Aufgrund der für die Messung konstant eingestellten Pulsbreite von 1,5ms, spielt der Einbruch der Drehmomentkonstante allerdings keine große Rolle, da sich die Drehzahl wenn überhaupt nur sehr gering ändert. Dieser Zusammenhang kann deshalb vereinfachend als linear angenommen werden kann. Die Versorgungsspannung der ESCs wird für die Aufnahme verschiedener Messwerte zwischen 6,1V und 8V variiert. Ändert sich die gemessene Spannung mit der Variation der Versorgungsspannung, so ist das ein hinreichender Beweis dafür, dass die Drehzahl der BLDC-Motoren spannungsabhängig ist. Die Ergebnisse der Messreihe aus Tabelle 1 sind zur einfacheren Auswertung in Abbildung 46 visualisiert.

$$U_A = k_i \cdot N \cdot \frac{\pi}{30} \quad (1)$$

Messung	1	2	3	4	5	6	7	8	9	10
$U_{\text{Versorgung}} [V]$	8.0	7.9	7.8	7.7	7.6	7.5	7.4	7.3	7.2	7.1
$U_{\text{Messung}} [V]$	5.80	5.75	5.65	5.57	5.48	5.40	5.35	5.25	5.18	5.09
Messung	11	12	13	14	15	16	17	18	19	20
$U_{\text{Versorgung}} [V]$	7.0	6.9	6.8	6.7	6.6	6.5	6.4	6.3	6.2	6.1
$U_{\text{Messung}} [V]$	5.05	4.96	4.87	4.82	4.69	4.73	4.66	4.60	4.50	4.45

Tabelle 1: Messreihe 1: Messwerte für Überprüfung einer eventuellen Spannungsabhängigkeit der Drehzahl

Aus Abbildung 46 können einige wertvolle Erkenntnisse abgeleitet werden. Zum einen ist die Drehzahl, welche proportional zu der an den DC-Motor gemessenen Ankerspannung U_{Messung} ist, nicht konstant. Die Drehzahl der BLDC-Motoren ist also von der Versorgungsspannung der ESCs abhängig, andernfalls wäre die resultierende Gerade eine Horizontale.

Die sich aus der Messreihe ergebenen Erkenntnisse machen eine Drehzahlregelung sinnvoll, da während der Durchfahrt des Parcours nicht die volle Drehzahl benötigt wird und eine Regelung es ermöglicht, die vorgegebene Drehzahl auch bei fortschreitender Entladung des Akkus zu fahren. Wird eine feste Geschwindigkeit vorgegeben, kann die Regelung bei fallender Versorgungsspannung der ESCs das PWM-Signal bis zur Stellgrenze (Pulsbreite = 1,9ms) erhöhen. Erst bei Erreichen der Stellgrenze nimmt die Drehzahl linear mit der Versorgungsspannung ab.

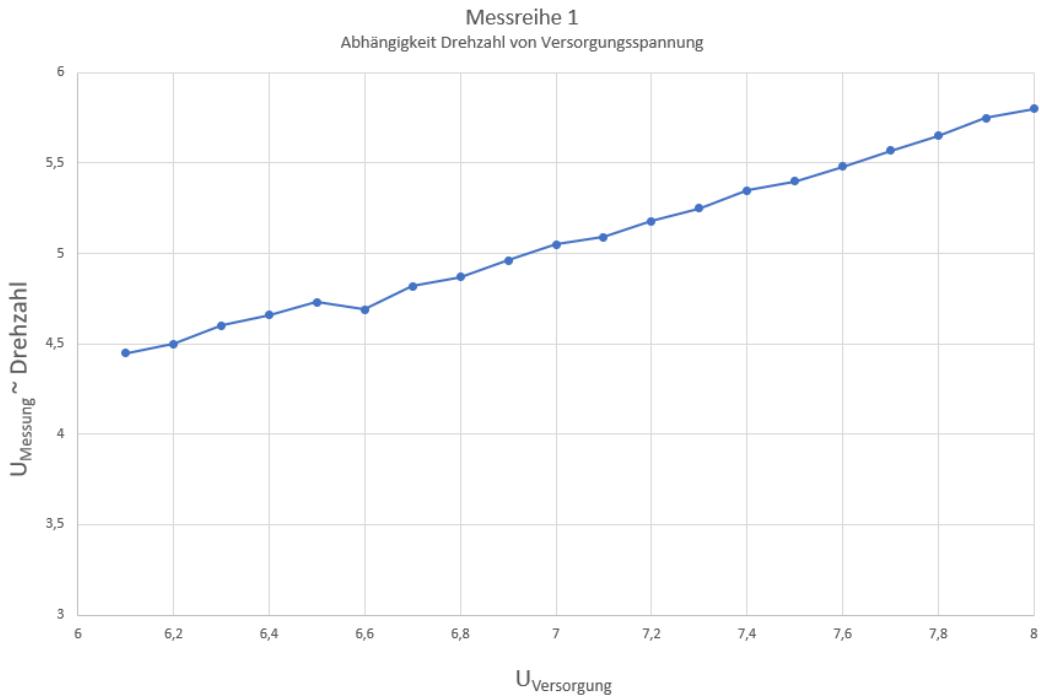


Abbildung 46

Beim NXP-Cup muss das Fahrzeug nach dem Überfahren der Ziellinie innerhalb von zwei Metern anhalten. In der Konfiguration der ESCs bietet sich dafür die Einstellung „Break On Stop“ an, mit der das Fahrzeug bei einer PWM-Pulsdauer von 1ms (Stop-Throttle) aktiv bremst, indem die drei Phasen der BLDC-Motoren auf Masse geschlossen werden. Die Bremskraft reicht aus, um das Fahrzeug innerhalb von ca. 30cm zum Stehen zu bringen.

Problem wegdriften bzw. Rutschen der Reifen weil blockiert -> wegrutschen nach rechts Grund evtl unterschiedlich feste Montage der Reifen

Für später Probleme wegen 1s Strafzeit, wenn nach der Ziellinie die Strecke verlassen wird oder nicht in 2m angehalten wird

4.5.2 Auswahl des Messprinzips

MARKUS

4.5.3 Hardware für die Drehzahlmessung

MARKUS

4.5.4 Programmierung des Drehzahlmessungsbausteins

MARKUS

5 Lenkung des Fahrzeugs

Wie in Kapitel 2.1 beschrieben, wird die Lenkung des Fahrzeugs von einem Servomotor angetrieben, der im Folgenden näher erklärt wird. Auch auf die Montage der Antriebskomponenten und die Programmierung des Lenkungsbausteins wird in diesem Kapitel näher eingegangen.

5.1 Servoantrieb

Zur Lenkung des Fahrzeugs wird ein Servomotor (siehe Abbildung 47) verwendet, der mit den Achsen über ein Lenkgestänge verbunden ist. Ein Servo besteht aus einem Motor, dessen Welle mit einem Potentiometer verbunden ist. Die interne Elektronik regelt automatisch die Istwert-Vorgabe des Potentiometers auf den an der Signalleitung anliegenden Sollwert aus. Dadurch können Lenkwinkelpositionen genau angefahren werden. Die Sollwert-Vorgabe erfolgt wie die Drehzahlvorgabe bei den BLDC-Motoren mit einem 50Hz PWM-Signal und einem Tastgrad von 5% bis 10%, wobei dieser proportional zum Lenkwinkel ist. Bei einem Tastgrad von 7,5% befindet sich die Lenkung in der Mittelstellung. Kleinere Werte bewirken einen Lenkeinschlag nach links und größere Werte einen Einschlag nach rechts. Der PWM-Signalleitung und der dazugehörigen Masseleitung schließt sich noch eine Versorgungsleitung an (5V-Versorgung).



Abbildung 47: Servomotor mit Anschlussleitungen; Versorgungsspannung rote Leitung, PWM-Signal orangene Leitung und Masseleitung in braun [DIY21]

5.2 Montage der Lenkungskomponenten

Die Komponenten der Lenkung, welche über einen Servomotor und ein mit den Reifen verbundenes Lenkgestänge realisiert ist, sind wie die Antriebskomponenten auf der unteren Fahrzeugebene montiert. Der Servomotor ist mit vier Schrauben an einem losen Teil des Chassis befestigt, welches über zwei Schrauben durch Langlöcher von oben am Fahrzeug montiert ist. Über die Langlöcher kann der Servomotor so platziert werden, dass die Vorderreifen parallel zueinander stehen. An der Welle des Servomotors ist dann das Lenkgestänge befestigt, welches mit den Reifen verbunden ist.

In Abbildung 48 sind die montierten Komponenten der Fahrzeuglenkung abgebildet. Die Befestigung des Servomotors ist in violett hervorgehoben, die Befestigung des losen Chassis-Teils in rot, die Lenkgestänge in blau und die Federung in orange.

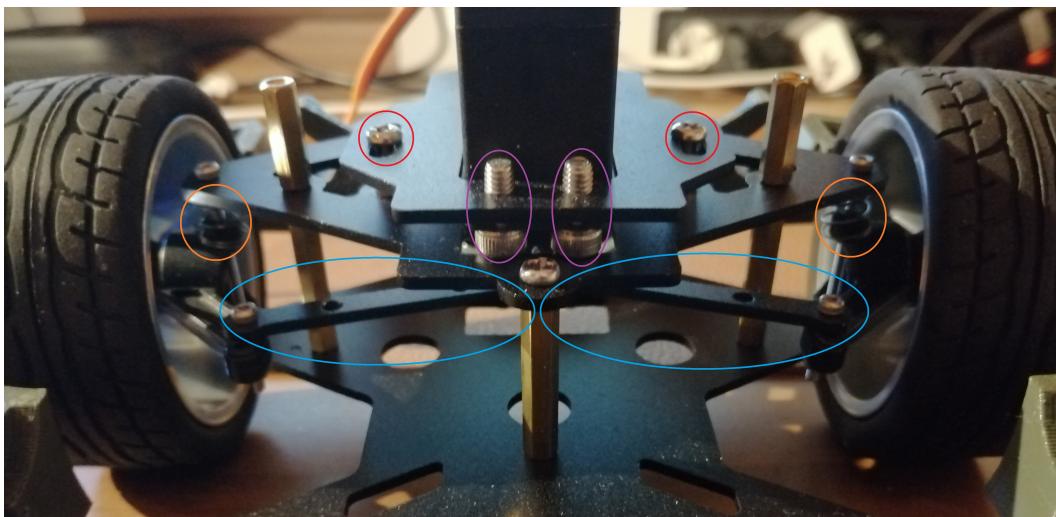


Abbildung 48: Montage des Servo-Motors, des Lenkgestänges und der Vorderreifen; Befestigung des Servo-Motors in violett, Befestigungsschrauben des losen Chassis-Teils in rot, Lenkgestänge in blau und Federung in orange

5.3 Programmierung des Lenkungsbausteins

Der Lenkungsbaustein der Software ist wie der Antriebsbaustein in zwei Dateien unterteilt, die Dateien „servo.c“ und „servo.h“. Die Datei „servo.h“ enthält alle relevanten Bibliotheken und Prototypen für die Datei „servo.c“.

Außer der Einbindung der Bibliotheken und der Prototypen der Funktionen aus der Datei „servo.c“ sind hier auch die Parameter für den maximalen linken und rechten Lenkeinschlag, für die Mittelstellung des Servomotors und für die Initialisierung des Timers für das PWM-Signal hinterlegt (Abbildung 49).

```
/* Definition of channel 2 duty --> on ticks*/
#define CTIMER1_PWM_0_DUTY 4400000

/* Definition of PWM period --> whole period ticks*/
#define CTIMER1_PWM_PERIOD 4400000

/* Definition of channel 2 ID */
#define CTIMER1_PWM_0_CHANNEL kCTIMER_Match_2

//max left steering value //Attention!! If less than 200000 -> 0,91ms--> hardware crash
#define SERVO_PWM_LeftValue 200000

//max right steering value //Attention!! If more than 459000 -> 2,08ms --> hardware crash
#define SERVO_PWM_RightValue 459000

//zero degree steering angle
#define SERVO_PWM_MiddleValue 327000
```

Abbildung 49: Relevante Zeilen der Datei „servo.h“ mit den Parametern für die Initialisierung des PWM-Timers und den Werten für den maximalen linken und rechten Lenkeinschlag und für die Mittelstellung des Servos

Wie schon beim Antrieb, wird für die PWM-Periodendauer bei der Initialisierung ein Wert von 4.400.000 Takten festgesetzt, woraus mit einer CPU-Taktfrequenz von 220MHz (220.000.000 Takte pro Sekunde) eine Periodendauer von 20ms resultiert. Die Pulsbreite wird während des Programmablaufs regelmäßig überschrieben, weshalb für die Initialisierung ein Wert von 0 gewählt werden kann. So bewegt sich der Servomotor bei der Initialisierung nicht und es kommt dabei nicht zum Crash.

Der Wert für den linken maximalen Lenkeinschlag („SERVO_PWM_LeftValue“, 200.000) entspricht hier einer PWM-Pulsbreite von 0,91ms und der des maximalen rechten Lenkeinschlags („SERVO_PWM_RightValue“, 459.000) einer Breite von 2,08ms. Auch wenn der Servomotor einen Wert zwischen 1ms und 2ms erwartet, kann mit den eben genannten Einstellungen noch ein wenig mehr Lenkeinschlag gewonnen werden. Mit dem Wert der Mittelstellung („SERVO_PWM_MiddleValue“, 327.000), welcher durch Probieren angenähert wurde, beläuft sich die Pulsbreite auf 1,486ms.

Auch der PWM-Timer benötigt bei der Initialisierung einige Parameter, deren Werte in der Datei „servo.h“ festgelegt sind (PWM-Periodendauer, PWM-Pulsdauer, Channel). Der

Channel wird auf das Timer Match Register 2 festgelegt („kCTIMER_Match_2“), was bei dem verwendeten Controller dem Pin P3.2 entspricht. Der Pin P3.2 wird auf der Controllerplatine über den Pin 11 der Buchsenleiste J13 nach außen geführt. Der Anschluss des Servos ist über den Anhang „Anhang 1: Schaltplan“ nachvollziehbar.

```

const ctimer_config_t SERVO_config = {
    .mode = kCTIMER_TimerMode, /* TC is incremented every rising APB bus clock edge */
    .input = kCTIMER_Capture_0, /*!< Timer capture channel 0 */
    .prescale = 0 /*!< Prescale value 0 --> */
};

void SERVO_Init(void)
{
    CTIMER1_Init(); //Timer init function
    CTIMER1->MCR |= CTIMER_MCR_MR2RL_MASK;
    //Reload MR2 with the contents of the shadow register when the timer counter (TC) is reset to 0
    IOCON->PIO[3][2] &= 0xFFFFFFFF0; //clear FUNC bits of P3.2
    IOCON->PIO[3][2] |= 0x4; //set FUNC bits to CTIMER1_MAT2 function ALT4 P3.2
    GPIO->DIR[3] |= 1<<2; //set CTIMER1_MAT2 pin to output
    CTIMER1->MSR[2] = CTIMER1_PWM_PERIOD - SERVO_PWM_MiddleValue;
    if (xTaskCreate(SERVO_Demo, "SERVO_Demo", configMINIMAL_STACK_SIZE + 100, NULL, 1, NULL) != pdPASS)
    {
        LED3_ON();
    }
}

void CTIMER1_Init(void)
{
    /* CTIMER1 peripheral initialization */
    CTIMER_Init(CTIMER1, &SERVO_config); //((CTIMER_Type, ctimer_config_t)
    /* PWM channel 2 of CTIMER1 peripheral initialization */
    CTIMER_SetupPwmPeriod(CTIMER1, CTIMER1_PWM_0_CHANNEL, CTIMER1_PWM_PERIOD, 4400000-CTIMER1_PWM_0_DUTY, false);
    //((CTIMER_Type, ctimer_match_t,uint32_t,uint32_t)
    /* Start the timer */
    CTIMER_StartTimer(CTIMER1); //Timer starten
}

void SERVO_Demo(void *pvParameters)
{
    while(1)
    {
        //Steer left
        CTIMER1->MSR[2] = CTIMER1_PWM_PERIOD - SERVO_PWM_LeftValue;
        vTaskDelay(500);
        //Steer right
        CTIMER1->MSR[2] = CTIMER1_PWM_PERIOD - SERVO_PWM_RightValue;
        vTaskDelay(800);
        //Steering angle zero degree
        CTIMER1->MSR[2] = CTIMER1_PWM_PERIOD - SERVO_PWM_MiddleValue;
        vTaskSuspend(NULL);
    }
}

```

Abbildung 50: Funktionen SERVO_Init und CTIMER1_Init und Demonstrations-Task SERVO_Demo der Datei „drive.c“

Die Datei „servo.c“ enthält die Funktionen zur Initialisierung der für die Verwendung des Servos notwendigen Controller-Peripherie. In der Funktion SERVO_Init wird zuerst die Funktion CTIMER1_Init aufgerufen, welche den Timer mit den in der Datei „servo.h“ festgelegten Parametern als PWM-Timer mit einer Periodendauer von 20ms und einer Pulslänge von 0ms initialisiert. Im Anschluss daran wird festgelegt, dass bei einem Timer-Überlauf die neuen Daten für die Pulslänge aus dem Shadow-Register geladen werden sollen. Zum Ändern des Lenkwinkels muss deshalb lediglich ein neuer Wert in das Shadow-Register geschrieben werden. Hier muss allerdings aufgepasst werden, da das Register nicht die Pulsbreite (On-Time) sondern die Off-Time erwartet. Deshalb muss der Wert, der eingetragen wird, der Periodendauer abzüglich der Pulsdauer entsprechen. Zusätzlich wird in der Funktion SERVO_Init

auch der Pin 3.2 für die Verwendung als PWM-Ausgang des Timers C konfiguriert. Am Ende wird der Wert für die Mittelstellung des Servos in das Shadow-Register geschrieben und ein Task für die Demonstration der Lenkung gestartet (SERVO_Demo), welcher nach einmaligem Linkslenken, Rechtslenken und nach der Rückkehr in die Mittelstellung ausgesetzt wird. Der Inhalt der Datei „servo.c“ ist in Abbildung 50 abgebildet.

6 Bedienungs-Board

Das Bedienungs-Board ist auf der oberen Fahrzeugebene über dem Controller verbaut. Es enthält ein Display, einen Drehencoder mit Button, einen separaten Button und einen Summer. Diese Komponenten dienen zum Einen der Eingabe von Parametern und der Bedienung des Fahrzeugs durch den Benutzer und zum Anderen zum Ablesen der Fahrzeugdaten, wie beispielsweise der Streckenerkennungsdaten der Kamera.

6.1 Schaltplan

Das Bedienungs-Board ist eine selbst bestückte Lochrasterplatine, die die Anschlüsse der einzelnen Komponenten auf eine Stifteleiste zusammenführt, welche über ein Flachbandkabel mit der Controllerplatine verbunden ist. In den Abbildungen 51 und 52 ist der Schaltplan des Bedienungs-Boards zusehen. Die Buchsenleisten J3 und J4 sind die Anschlüsse des Displays. An die Buchsenleisten J5 und J6 sind die Pins des Encoders geführt. Dieser gibt zwei Signale für die Erkennung der Drehrichtung und ein Signal für den Taster aus. Die Signale des Displays, des Drehencoders, des Tasters und des Summers werden an die Stifteleisten J1 und J2 geführt, über die die Verbindung zur Controllerplatine hergestellt wird. Die Einbindung des Bedienungs-Boards in das Gesamtsystem des Fahrzeugs ist über den Anhang „Anhang 1: Schaltplan“ nachvollziehbar.

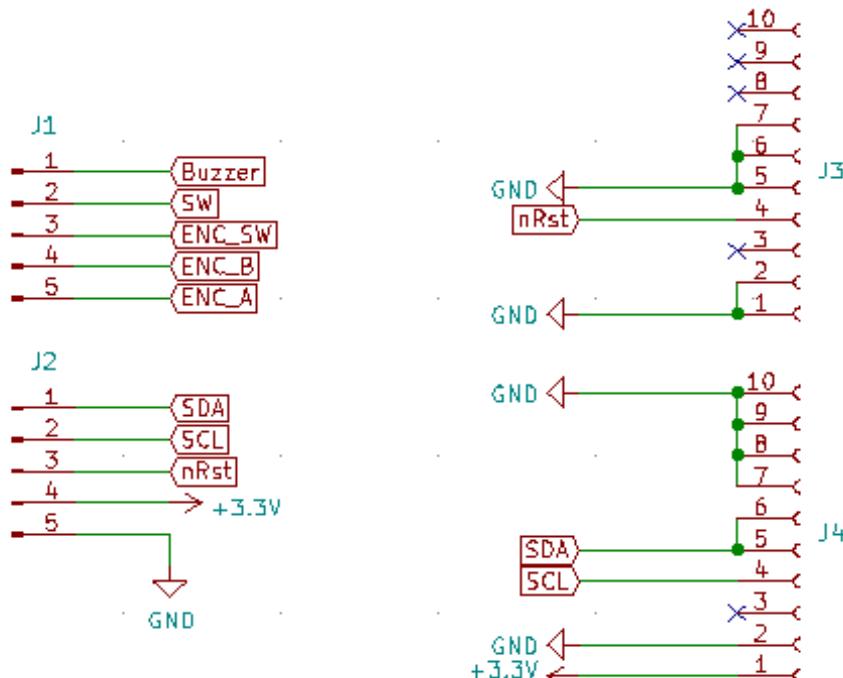


Abbildung 51: Schaltplan des Bedienungs-Boards mit den Stifteleisten J1 und J2 zur Verbindung mit dem Mikrocontroller und den Buchsenleisten J3 und J4 für das Display

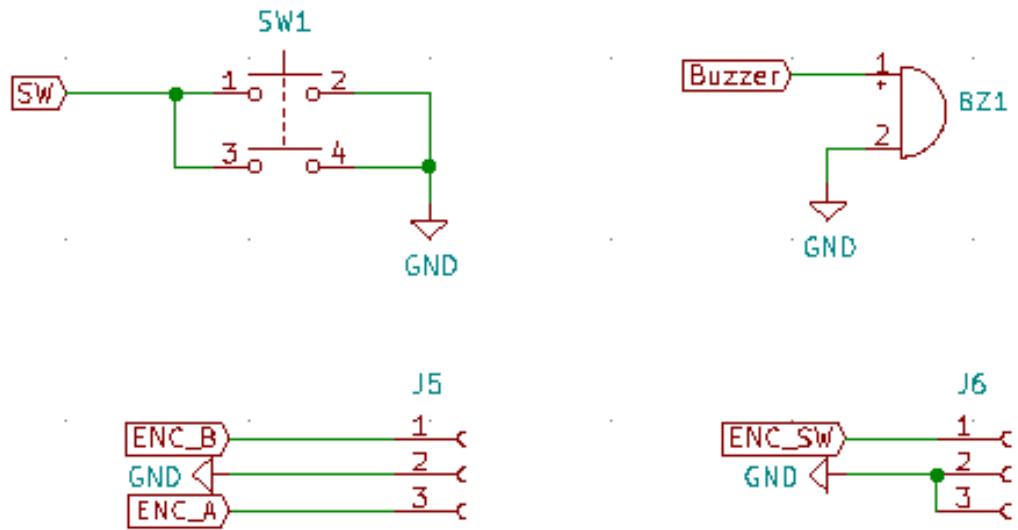


Abbildung 52: Schaltplan des Bedienungs-Boards mit dem Drehencoder, dem Taster und dem Summer

6.2 Programmierung der Anzeige

HIAS

6.3 Programmierung der Steuerelemente

HIAS

7 Streckenerkennung des Fahrzeugs

CHRIS

7.1 Kamera des Fahrzeugs

CHRIS

7.2 Montage und Ausrichtung der Kamera

CHRIS

7.3 Programmierung der Streckenauswertung

CHRIS

7.4 Kameralinsen

CHRIS

8 Verteilerplatine auf der unteren Fahrzeugebene

9 Regelung des Fahrzeugs

10 Inbetriebnahme des Fahrzeugs

11 Zusammenfassung, Fazit und Ausblick

11.1 Zusammenfassung und Fazit

11.2 Ausblick

Abbildungsverzeichnis

1	Grundplatte des Standardbausatzes für das Fahrzeug	3
2	Grundplatte des Standardbausatzes für das Fahrzeug	3
3	Obere Ebene des Standardbausatzes für das Fahrzeug	4
4	Obere Ebene des Standardbausatzes für das Fahrzeug	4
5	Konstruktionsbild des oberen Stoßstangenteils	5
6	Konstruktionsbild des unteren Stoßstangenteils	5
7	Gedrucktes oberes und unteres Stoßstangenteil	6
8	Konstruktionsbild der Halterung des Ultraschallboards	6
9	Gedruckte Halterung des Ultraschallboards	6
10	Draufsicht der Stoßstange und der Ultraschallboard-Halterung	6
11	Untersicht der Stoßstange und der Ultraschallboard-Halterung	7
12	Konstruktionsbild der Akku-Halterung	7
13	Gedruckte Akku-Halterung	7
14	Konstruktionsbild der Seitenschweller	8
15	Gedruckte Seitenschweller	8
16	Montierte Akku-Halterung und Seitenschweller	8
17	Konstruktionsbild der Controller-Halterung	9
18	Konstruktionsbild der Platinen-Halterung für die Fahrzeugbedienung	9
19	Konstruktionsbild der Abdeckung für die Fahrzeugbedienung	9
20	Konstruktionsbild der Drucktasters	9
21	Gedruckte Controller-Halterung	10
22	Gedruckte Drucktaster-Verlängerung	10
23	Gedruckte Platinen-Abdeckung des Bedienungsboards	10
24	Gedruckte Platinen-Halterung für das Bedienungs-Board	10
25	Fertig montierte Controller- und Bedienungs-Board-Halterung	11
26	Fertig montierte Controller- und Bedienungs-Board-Halterung mit Abdeckung	11
27	Konstruktionsbild einer Motorcontroller-Halterung	12
28	Gedruckte Motorcontroller-Halterungen	12
29	Die auf der Grundplatte montierten Motorcontroller	12
30	Konstruktionsbild einer einfachen Platinenhalterung	13
31	Gedruckte Platinenhalterungen	13
32	Mit den einfachen Platinenhalterungen montierte Verteiler-Platine	13
33	Kameramontage-Komponente der Kamera-Halterung	14
34	Stangenmontage-Komponente der Kamera-Halterung	14
35	Fertig montierte Kamera mit Kamerahalterung	15
36	Controllerplatine LPCXpresso54608 [Sem19]	16
37	Skizze zur Beschaltung eines ESCs und BLDC-Motors	17
38	Montage der Antriebskomponenten	18

39	Initialisierungssequenz der ESCs	19
40	Programmierung des Arduino Nano zur ESC-Konfiguration	20
41	Parameter der neuen ESC-Konfiguration	20
42	Relevante Zeilen der Datei „drive.h“	21
43	Funktionen BLDC_Init und CTIMER3_Init der Datei „drive.c“	22
44	Funktion BLDC_Init_Task der Datei „drive.c“	23
45	Messaufbau zur Prüfung der Spannungsabhängigkeit der BLDC-Motoren	24
47	Servomotor mit Anschlussleitungen [DIY21]	27
48	Montage der Lenkungskomponenten	28
49	Relevante Zeilen der Datei „servo.h“	29
50	Funktionen SERVO_Init, CTIMER1_Init und SERVO_Demo der Datei „drive.c“	30
51	Schaltplan des Bedienungs-Boards 1	32
52	Schaltplan des Bedienungs-Boards 2	33

Literatur

- [DIY21] DIYELECTRONICS: *MG996R Servo Motor / 120° Servo Motor – 10kg Stalling Torque.* Website, 2021. – Online erhältlich unter <https://www.diyelectronics.co.za/store/servos/766-towerpro-mg996r-servo-motor.html>; zuletzt abgerufen am 27. März 2021
- [Sem19] *LPC546xx User Manual. : LPC546xx User Manual*, 2019
- [Wil20] WILKENS, Matthias: *Autonomous Driving Robot Student Teams Work from Home to Win the NXP Cup Electromaker Innovation Challenge.* Website, 2020. – Online erhältlich unter <https://www.nxp.com/company/blog/autonomous-driving-robot-student-teams-work-from-home-to-win-the-nxp-cup-electromaker-innovation-challenge>; zuletzt abgerufen am 19. März 2021

Anhang 1: Schaltplan