Freie Universität Berlin
Fachbereich Mathematik und Informatik
Institut für Informatik

Bachelorarbeit

Integration eines elektrischen Antriebs in ein autonomes Modellfahrzeug

Auswahl, Einbau und Implementierung auf einem Mikrocontroller der STM32-Familie

vorgelegt von

Thorben Meier

am 21. November 2012

Gutachter: Prof. Dr. Raúl Rojas
Betreuer: Dr. Hamid Reza Moballegh

Eigenständigkeitserklärung

Ich versichere hiermit, dass ich die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Ich erkläre weiterhin, dass die vorliegende Arbeit noch nicht im Rahmen eines anderen Prüfungsverfahrens eingereicht wurde.

Berlin, den 21. November 2012

Zusammenfassung

Eine zuverlässige Antriebsplattform ist die Grundlage jedes erfolgreichen autonomen Fahrens. Eine intelligente Fahrtenplanung errechnet auf Grundlage von Kamerabildern und anderen Sensoren ein Verhalten, das durch den Antrieb umgesetzt werden muss. Insbesondere bei schwierigen Fahrmanövern hängt der Erfolg daher von der Genauigkeit und Zuverlässigkeit des Antriebs ab.

Diese Arbeit beschäftigt sich mit der Integration eines elektrischen Antriebs in ein autonomes Modellfahrzeug des Berlin United Racing Teams. Neben der Auswahl eines passenden Elektromotors und Reglers – durch detaillierte Berechnungen der benötigten Eigenschaften des Motors anhand der im Vorfeld erhobenen Anforderungen an die Längsdynamik des Modellfahrzeugs – wird in dieser Arbeit auf den Einbau sowie die korrekte Verschaltung und Konfiguration des elektrischen Antriebs eingegangen. Ein weiterer zentraler Punkt dieser Arbeit ist die Implementierung der Kommunikationsschnittstelle zwischen dem elektrischen Antrieb und einem Mikrocontroller aus der STM32-Familie.

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Autonome Fahrzeuge . . . 1
1.2 Das autonome Modellfahrzeug des Racing Teams . . . 2
1.3 Technische Rahmenbedingungen . . . 3
1.4 Ziele und Aufbau . . . 4

2 Auswahl des elektrischen Antriebs **7**

2.1 Einführung in elektrische Antriebe . . . 7
2.1.1 Komponenten des elektrischen Antriebs . . . 7
2.1.2 Elektrische Antriebe im Modellbau . . . 8
2.2 Eingrenzung der Auswahl . . . 9
2.3 Erhebung der Anforderungen . . . 10
2.4 Berechnung der benötigten Motoreigenschaften . . . 11
2.4.1 Berechnung der Drehzahl . . . 11
2.4.2 Berechnung der Leistung . . . 13
2.4.2.1 Fahrwiderstand . . . 13
2.4.2.2 Drehmoment . . . 17
2.4.2.3 Leistung . . . 18
2.5 Ergebnisse und Auswahl . . . 19
2.5.1 Elektromotor . . . 19
2.5.2 Regler . . . 21

3 Einbau, Verschaltung und Konfiguration **23**

3.1 Einbau der Komponenten . . . 23
3.2 Verschaltung der Komponenten . . . 24
3.2.1 Verschaltung von Regler und Elektromotor . . . 24
3.2.2 Verschaltung von Regler und Mikrocontroller . . . 24
3.2.2.1 Signalpegelanpassung . . . 25
3.3 Konfiguration des Reglers . . . 27
3.3.1 Hilfsmittel zur Konfiguration des Reglers . . . 27
3.3.2 Konfiguration der Kommunikationsparameter . . . 27
3.3.2.1 Berechnung der möglichen Datenrate . . . 28
3.3.3 Konfiguration der Parameter . . . 28

4 Implementierung der Kommunikationsschnittstelle 31
4.1 Architektur . . . 31
4.1.1 Framework . . . 31
4.1.1.1 Nachrichtenpumpe . . . 31
4.1.2 Interface zur Antriebsregelung . . . 32
4.2 Informationen vom Mikrocontroller zum Regler . . . 32
4.2.1 Konfiguration des UART . . . 32
4.2.2 Konstanten und Initialisierung des Reglers . . . 33
4.2.3 Erstellung der Befehle zur Drehzahlregelung . . . 35
4.2.4 Versenden der Befehle . . . 37
4.3 Informationen vom Regler zum Mikrocontroller . . . 38
4.3.1 Initialisierung zum Erfassen der Impulse . . . 38
4.3.2 Verhalten bei Über-/Unterlauf . . . 39
4.3.3 Auswertung der Impulse . . . 40
4.4 Ausblick auf weitere Möglichkeiten . . . 40

5 Zusammenfassung und Fazit 43

Abbildungsverzeichnis 45

Tabellenverzeichnis 47

Quellcodeverzeichnis 49

Literaturverzeichnis 51

1 Einleitung

Diese Arbeit beschäftigt sich mit der Integration eines elektrischen Antriebs in ein autonomes Modellfahrzeug, das vom Berlin United Racing Team in der Arbeitsgruppe *Intelligente Systeme und Robotik* unter der Leitung von Prof. Dr. Raúl Rojas an der Freien Universität Berlin entwickelt wird.

1.1 Autonome Fahrzeuge

Unter autonomen Fahrzeugen versteht man Fahrzeuge, die selbständig navigieren können. Um dieses Navigieren zu ermöglichen, benötigt das Fahrzeug Sensoren, mit denen es seine Umwelt wahrnehmen kann. Typische Sensoren hierfür sind (Video-)Kameras, Laserscanner, Radarsensoren, Inertialsensoren und Odometrie. Während Kameras meist für das Erkennen von visuell markanten Objekten wie Fahrspuren, Ampeln etc. verwendet werden, werden Laserscanner und Radarsensoren meist für die Erfassung der Abstände zu den sich in der Nähe befindenden Objekten genutzt. Die Odometrie dient zur Aufzeichnung einer gefahrenen Strecke und wird oft in Kombination mit Inertialsensoren, die die Messung der Beschleunigung und der Drehraten ermöglichen, und GPS[1] zur Selbstlokalisierung auf einer Karte bzw. zur Erstellung eigener Karten verwendet [4].

Abbildung 1.1 zeigt das autonome Fahrzeug MadeInGermany aus dem Projekt AutoNOMOS[2] der Freien Universität Berlin. Deutlich zu erkennen ist der Laserscanner auf dem Dach und die Raddrehzahlsensoren zur Odometrie an den Hinterrädern des Fahrzeugs. Erste erfolgreiche autonome Fahrten konnten damit bereits durch die Innenstadt von Berlin absolviert werden [2].

Die Intelligenz eines autonomen Fahrzeugs liegt in der Interpretation der Sensordaten und dem daraus abgeleiteten Verhalten. Für die tatsächliche Steuerung der Aktoren eines Autos zur Regelung von Motor, Bremsen oder Lenkung werden Schnittstellen benötigt, die diese Steuerung ermöglichen und das gewünschte Verhalten umsetzen. Für eine solche elektronische Steuerung hat sich der Begriff *Drive-by-wire* durchgesetzt, dessen Konzept auch im autonomen Fahrzeug MadeInGermany zum Einsatz kommt. Übertragen werden die elektronischen Signale über den CAN[3]-Bus, an den sämtliche Sensoren und Aktoren angeschlossen sind [4]. Funktionierende Schnittstellen, Regler und letztlich auch mechanische Komponenten bilden daher die Grundlage für autonomes Fahren.

[1] Global Positioning System
[2] http://www.autonomos.inf.fu-berlin.de/
[3] Controller Area Network

Abbildung 1.1: MadeInGermany von AutoNOMOS
Quelle: http://autonomos.inf.fu-berlin.de/made-in-germany
Abruf: 16.11.2012

1.2 Das autonome Modellfahrzeug des Racing Teams

Das autonome Modellfahrzeug des Berlin United Racing Teams (Abb. 1.2) verfügte zum Zeitpunkt dieser Arbeit lediglich über eine nach vorne ausgerichtete (Video-)Kamera und über eine Odometrie. Andere Teams, wie bspw. das FAUST-Projekt[4] der HAW Hamburg oder das Team CDLC[5] der TU Braunschweig, setzen zusätzlich Ultraschall- und Infrarotsensoren zur Erkennung und Abstandsmessung von Hindernissen ein [16, 1]. Gemessen an den Anforderungen, die an das Modellfahrzeug gestellt werden, sollten die Kamera und die Odometrie jedoch zunächst ausreichen. Die Anforderungen an das Modellfahrzeug ergaben sich aus den Disziplinen des Carolo-Cups.

Der Carolo-Cup[6] ist ein Hochschulwettbewerb, in dem studentische Teams mit ihren autonomen Modellfahrzeugen im Maßstab 1:10 in verschiedenen Disziplinen gegeneinander antreten. Zu den Aufgaben gehören das Abfahren eines Rundkurses mit sowie ohne Hindernisse und das parallele Einparken am Seitenstreifen [5].

Entscheidend für das Fahren auf dem Rundkurs ist zum einen das Einhalten der Fahrbahnen, zum anderen das Erkennen von Hindernissen, das in diesem Fall auch über die Kamera möglich ist, da sich alle Hindernisse von der Fahrbahn visuell deutlich absetzen

[4] http://www.informatik.haw-hamburg.de/faust.html
[5] http://www.team-cdlc.de/
[6] http://www.carolo-cup.de/

Abbildung 1.2: Das erste Modellfahrzeug des Berlin United Racing Teams

und so für die *Computer Vision* erkennbar sind, so dass weitere Sensoren nicht zwangsläufig benötigt werden. Für das Einparken war zunächst die Odometrie vorgesehen. Über diese sollte das Fahrmanöver anhand eines Bewegungs-Patterns, das Lenkwinkel und Wegstrecke miteinander verknüpft, implementiert werden. Damit wäre es möglich gewesen, das Fahrmanöver „Einparken“ als komplettes Szenario ablaufen zu lassen, sobald eine geeignete Parklücke erkannt wurde. Bis zu der Entscheidung, auf eine *Omnicam* umzusteigen, wurde ein Bewegungs-Pattern allerdings nicht implementiert. Mit der Omnicam wird es nun möglich sein, die Umgebung im Radius von 360 ° um das Auto zu erfassen und damit auch das Einparken visuell gesteuert und ohne Zuhilfenahme von weiteren Sensoren durchführen zu können, so dass ein Bewegungs-Pattern nicht mehr notwendig sein wird und insbesondere dynamisch reagiert werden kann.

1.3 Technische Rahmenbedingungen

Im Zuge des Softwareprojektes „Entwicklung eines autonomen Modellfahrzeugs“ (01-02/2012), das im Vorfeld des Carolo-Cups 2012 stattfand und aus dem das Berlin United Racing Team hervorging, wurden ein Modellfahrzeug und weitere Bauteile für das autonome Fahren angeschafft (Abb. 1.2). Bei diesem Modellfahrzeug handelte es sich um ein sehr preisgünstiges Komplettmodell, in dem sämtliche Komponenten, u.a. ein einfacher elektrischer Antrieb, vorhanden waren. Mittels *Pulsweitenmodulation* war es möglich, diesen Antrieb durch einen Mikrocontroller zu regeln.

Während der Teilnahme am Carolo-Cup 2012 stellte sich jedoch heraus, dass ein solcher elektrischer Antrieb aus dem einfachen Modellfahrzeugbau für langsames und dosiertes Fahren nicht geeignet ist und dringend ein besser regelbarer Antrieb benötigt wurde. Mit der Entscheidung für einen neuen Antrieb fiel auch die Entscheidung, ein neues Modellfahrzeug bzw. ein neues Chassis anzuschaffen, da das einfache Komplettmodell nicht für das Ersetzen des Antriebs ausgelegt ist.

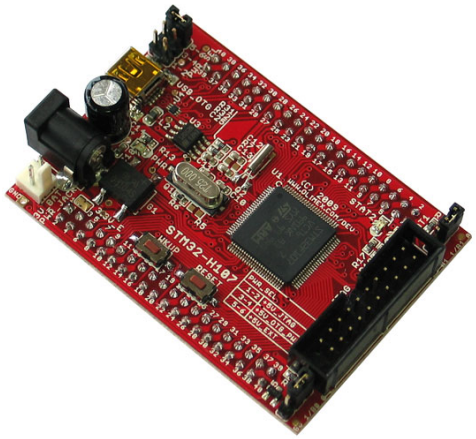


Abbildung 1.3: Der verwendete Mikrocontroller des Herstellers Olimex
Quelle: http://www.olimex.com/Products/ARM/ST/STM32-H107/
Abruf: 16.11.2012

Als Schnittstelle zwischen intelligenter Fahrtenplanung und Steuerung der Hardware des Modellfahrzeugs wird das *Motorboard* verwendet. Das Motorboard regelt einerseits den Antrieb, die Lenkung und die Beleuchtung, andererseits wird auch die Odometrie in Form von Impulsen durch einen Drehgeber ausgewertet, um die zurückgelegte Strecke aufzuzeichnen. Bei dem Motorboard handelt es sich um einen Mikrocontroller der STM32-F107 Baureihe mit einem ARM Cortex-M3 Prozessor, genauer gesagt um ein STM32-H107 Board (Abb. 1.3) des Herstellers Olimex[7].

1.4 Ziele und Aufbau

Das Ziel dieser Arbeit war die Bereitstellung einer zuverlässigen Antriebsplattform für die autonome Fahrt durch die Integration eines neuen elektrischen Antriebs in das (bereits vorhandene) neue Chassis. Diese neue Plattform sollte den hohen Anforderungen an die Regelung für autonomes Fahren gerecht werden. Solche Anforderungen sind das Erreichen und dauerhafte Halten exakter Geschwindigkeitsvorgaben in $^{mm}/_{s}$, die Möglichkeit Beschleunigungs- und Bremsrampen zu erzeugen, mittels derer eine kontinuierliche Beschleunigung bzw. Bremsung stattfinden kann, sowie die Erfassung der zurückgelegten Strecke über eine im Antrieb integrierte Odometrie.

[7] http://www.olimex.com/

In den folgenden Kapiteln wird das Vorgehen zum Erreichen dieses Ziels beschrieben und dokumentiert. Die Kernaspekte dabei sind:

- Berechnung der Fahrzeuglängsdynamik zur Auswahl des richtigen Antriebs
- Einbau, Verschaltung und Konfiguration des Antriebs
- Implementierung der Kommunikation zwischen Motorboard und Antrieb
- Zusammenfassung der Ergebnisse und Fazit

Abbildung 1.4: Logo des Berlin United Racing Teams[8]

[8] http://racing.berlin-united.org/

2 Auswahl des elektrischen Antriebs

2.1 Einführung in elektrische Antriebe

2.1.1 Komponenten des elektrischen Antriebs

Ein elektrischer Antrieb besteht im Allgemeinen aus mehreren Komponenten: dem Steuerelement, dem Stellelement, dem Antriebselement, dem Übertragungselement und dem Wirkelement (Abb. 2.1).

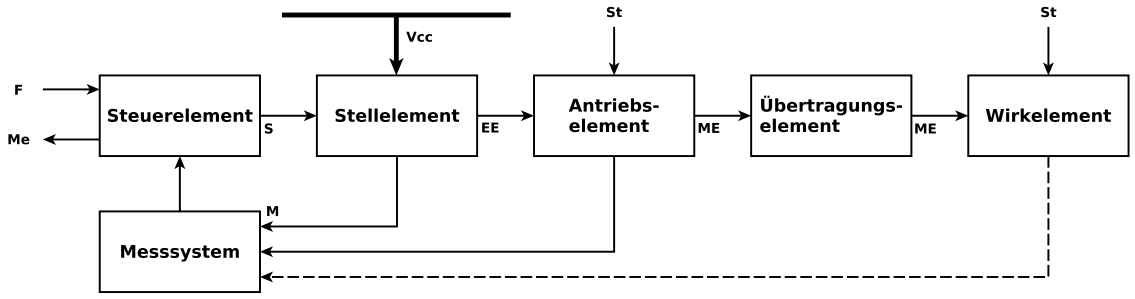


Abbildung 2.1: Schema der Komponenten des elektrischen Antriebs, nach [22, S. 15]

Das Steuerelement vergleicht die durch ein übergeordnetes System vorgegebenen Führungsgrößen (F) (auch Sollwerte, bspw. die Drehzahl) mit den Istwerten aus den Messgrößen (M) und erzeugt somit Stellgrößen (S) für das Stellelement. Sämtliche durch das Messsystem erfassten Messgrößen können als Meldegrößen (Me) einem übergeordneten System zur Verfügung gestellt werden. Aus den Messgrößen lassen sich insbesondere die Störgrößen (St), die auf das Antriebs- und Wirkelement wirken, ableiten. Das Stellelement ist an die Versorgungsspannung (Vcc) angeschlossen und setzt die eingehenden Stellgrößen in diskrete Ausgangsströme für das Antriebselement um. Das Antriebselement ist ein Elektromotor, der die elektrische Energie (EE) der Eingangsströme in mechanische Energie (ME) umwandelt und somit eine rotatorische Bewegung erzeugt, die durch das Übertragungselement an das Wirkelement weitergeleitet wird. Das Übertragungselement ist zumeist ein Getriebe, das die Drehzahl und das Drehmoment des Motors ändert und so dem Wirkelement angepasste Größen zur Verfügung stellt. Das Wirkelement ist schließlich das Element des elektrischen Antriebs, auf das die Energie wirkt und das somit angetrieben bzw. bewegt wird [22, S. 15].

2.1.2 Elektrische Antriebe im Modellbau

Die Komponenten des allgemeinen elektrischen Antriebs finden sich prinzipiell im Modellbau genauso wieder. Es verschmelzen hier jedoch das Steuerelement, das Stellelement und das Messsystem in einer Komponente, die man üblicherweise als Regler bezeichnet. Ein Regler besitzt somit neben den Informationseingängen, die die Sollwerte des übergeordneten Systems und des Messsystems aufnehmen, auch einen Anschluss an die Versorgungsspannung. Der Regler setzt somit die Eingangsinformationen direkt in einen diskreten Strom für den Elektromotor um [11, 9] (Abb. 2.2).

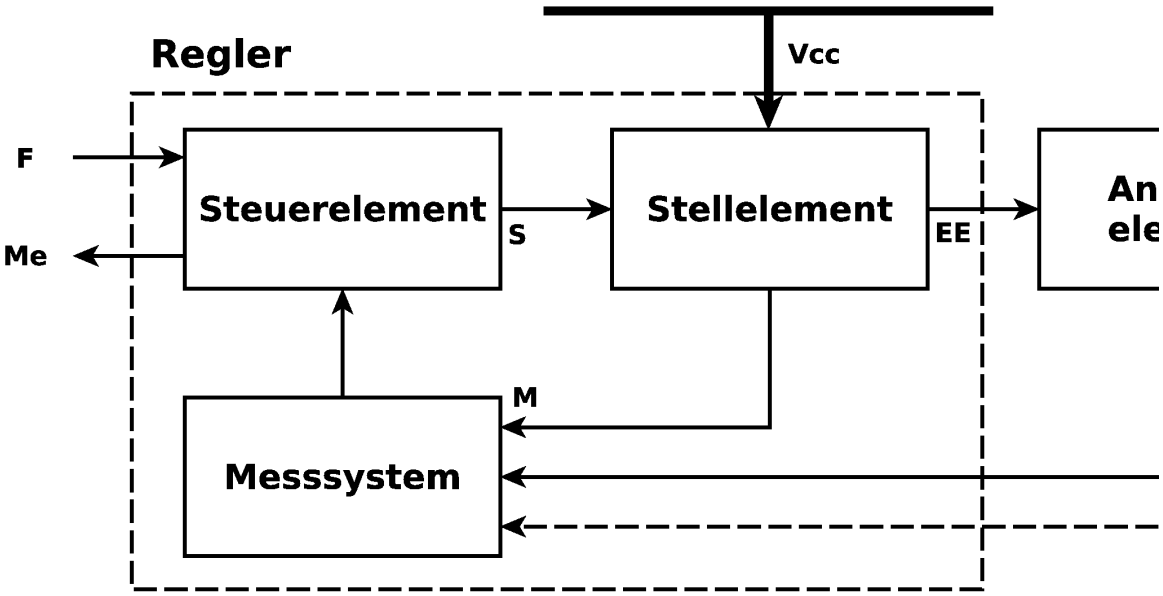


Abbildung 2.2: Der Regler als Komponente des el. Antriebs im Modellbau, nach [22, S. 15]

Aufgrund der üblicherweise verwendeten Akkutechnologie (immer Gleichspannung) werden im Modellbau nahezu ausschließlich Gleichstrommotoren als Antriebselement eingesetzt. Gleichstrommotoren sind hinsichtlich ihrer Eigenschaften und ihres Preises darüber hinaus für den Modellbau sehr gut geeignet. Übliche Spannungsversorgungen für den Modellbau im Maßstab 1:10 sind Lithium-Polymer-Akkus bestehend aus zwei bis vier Zellen mit einer Nennspannung von je $3,7\ V$ [24]. Das Übertragungselement besteht im Modellbau üblicherweise aus einer Kombination von Stirnrädern und bildet somit ein klassisches Getriebe. Für ein Modellfahrzeug sind die Räder das Wirkelement des elektrischen Antriebs. An ihnen wird die rotatorische Bewegung des Motors in die translatorische Bewegung des Fahrzeugs umgesetzt.

2.2 Eingrenzung der Auswahl

Für die Auswahl eines elektrischen Antriebs für ein autonomes Modellfahrzeug sollten zunächst einige Eingrenzungen gemacht werden, um eine sinnvolle Menge an möglichen Bauteilen für den elektrisch Antrieb aufzustellen.

Zunächst lässt sich die Auswahl des elektrischen Antriebs auf die Auswahl des Elektromotors und eines passenden Reglers beschränken. Getriebe und Räder sind bereits an dem Chassis X-Ray T3 spec. 2012 des neuen Modellfahrzeugs verbaut. Je nach Eigenschaften des ausgewählten Motors bedarf es möglicherweise einer Anpassung des Getriebes, um ein optimiertes Übersetzungsverhältnis zwischen Motor- und Raddrehzahl herzustellen. Wie in Abschnitt 2.1.2 erwähnt, sind aufgrund der Energieversorgung durch einen gewöhnlichen Modellbauakku lediglich Gleichstrommotoren sinnvoll. Neben der Einschränkung auf Gleichstrommotoren ist bereits im Vorfeld eine Eingrenzung auf die Elektromotoren und Regler eines Herstellers sinnvoll. Eine Auswahl ohne Einschränkung auf einen Hersteller wäre aufgrund der großen Anzahl an Herstellern mit z.T. erheblichen Qualitätsunterschieden wesentlich aufwendiger. Dabei ist die Qualität der Komponenten – insbesondere die Qualität des Reglers – entscheidend für die Regelbarkeit und Langlebigkeit des Antriebs. Die Wahl fiel dabei letztlich auf den Hersteller Faulhaber[1], mit dessen Elektromotoren und Reglern das Team der FU-Fighters[2] mit den Small-Size Robotern im Jahr 2004 sehr gute Erfahrungen gemacht hat [3].

Trotz der Vorauswahl eines Herstellers und der Einschränkung auf Gleichstrommotoren stehen sich allerdings immer noch zwei verschiedene Konstruktionsprinzipien gegenüber.

Das erste Prinzip ist der klassische Gleichstrommotor. Bei dieser ältesten Konstruktionsweise des Elektromotors bildet ein Dauermagnet den Stator (auch Ständer) und eine je nach Konstruktionsweise variable Anzahl von Spulen den Rotor (auch Läufer). Die elektrische Umpolung der Spulen bzw. Ankerwicklungen des Rotors findet am Kommutator durch Schleifkontakte – die sog. Bürsten – statt. Hier besteht zu jedem Zeitpunkt eine Reibung, die den Wirkungsgrad des Motors verschlechtert und auch zu stetigem Verschleiß der Bürsten führt, insbesondere bei hohen Drehzahlen und hoher Temperatur [14, S. 14].

Das zweite Prinzip ist der bürstenlose oder auch elektrisch kommutierte Gleichstrommotor (Abb. 2.3). Bei diesem befindet sich der Dauermagnet in der Rolle des Rotors und die Spulen in der Rolle des Stators. Durch Hall-Sensoren, mit denen die Ausprägung des magnetischen Feldes gemessen wird und dadurch die Stellung des Rotors bekannt ist, können die Spulen für jede Rotorstellung nach den Gesetzmäßigkeiten der Lorentzkraft optimal beschaltet werden [22, S. 62 f.].

Durch diese elektronische Beschaltung werden keinerlei Bürsten mehr benötigt und somit das Reibungsdrehmoment erheblich reduziert. Da es sich bei bürstenlosen Gleichstrommotoren damit um das langlebigere, geräuschärmere und effizientere Konstruktionsprinzip handelt,

[1] http://www.faulhaber.com/
[2] http://robocup.mi.fu-berlin.de/

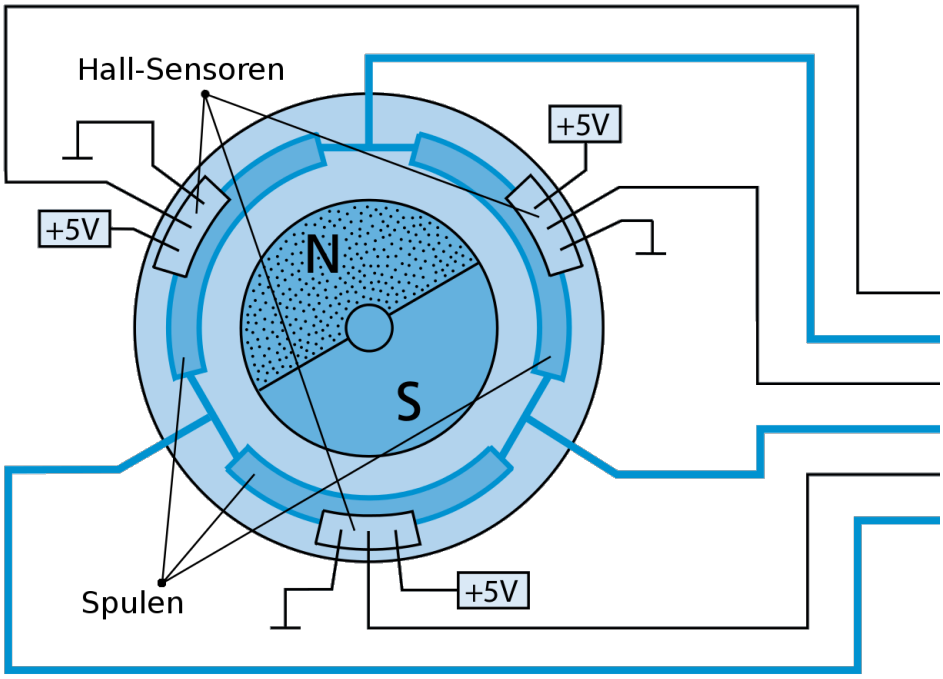


Abbildung 2.3: Schema eines bürstenlosen Gleichstrommotors
Quelle: [8] (leicht verändert)

werden für die Auswahl nur bürstenlose Elektromotoren betrachtet [14, S. 17 f.]. Das FAUST-Projekt der HAW Hamburg verwendet ebenfalls einen bürstenlosen Elektromotor, der über einen gewöhnlichen Fahrtensteller mittels *Pulsweitenmodulation* geregelt wird [23].

Die Auswahl eines Reglers ist insbesondere von der Wahl des Motors abhängig. Zwar bietet FAULHABER verschiedene Regler für bürstenlose Elektromotoren an, jedoch fällt die Wahl für einen Regler mit der Wahl für einen Motor (Abschnitt 2.5.1 und 2.5.2).

2.3 Erhebung der Anforderungen

Für die korrekte Auswahl eines Elektromotors ist die Erhebung der Anforderungen an die Längsdynamik des Modellfahrzeugs und damit die geforderte Leistung des Elektromotors von entscheidender Bedeutung.

Wichtige Größen der Längsdynamik sind die gewünschte Höchstgeschwindigkeit und die gewünschte Beschleunigung. Aus diesen Größen lassen sich die Fahrwiderstände und die Drehzahl berechnen, die der Elektromotor zu überwinden bzw. zu leisten hat (Abschnitt 2.4). Neben der Leistungsanforderung stellen die Außenabmessung und die Befestigung des Elektromotors ebenso wichtige Kriterien dar, da der Platz in einem Modellfahrzeug im Maßstab 1:10 nur begrenzt zur Verfügung steht. Insbesondere ist die Befestigung des Elektromotors an die vorgegebene Halterung des Chassis gebunden, sofern man großen Aufwand und Kosten

[3] 4 Zellen Lithium-Polymer-Akku

Fahrzeuglängsdynamik	
Eigenschaft / Anforderung	*Wert*
Gewicht des Fahrzeugs (inkl. Karosserie und Aufbau)	ca. 2 kg
Radius der Räder	31 mm
Anzahl Zähne Motorzahnrad	22
Anzahl Zähne großes Hauptzahnrad	84
Anzahl Zähne kleines Hauptzahnrad	20
Anzahl Zähne Hinterachszahnrad	38
zur Verfügung stehende Akkuspannung	$14,8$ V[3]
Höchstgeschwindigkeit (V_{max})	4 $\frac{m}{s}$
Beschleunigung auf V_{max}	4 $\frac{m}{s^2}$

Abmessungen	
Abstand der Befestigungsbohrung des Motors (180 ° versetzt)	19-30 mm
min. Länge der Motorwelle	12 mm
max. Durchmesser des Motors	38 mm
max. Länge des Motors	65 mm

Tabelle 2.1: Eigenschaften des Modellfahrzeugs und Anforderungen an den Antrieb

für die Fertigung individueller Teile vermeiden will. Tabelle 2.1 zeigt einige Eigenschaften des Modellfahrzeugs und Anforderungen an den Antrieb.

Eine Höchstgeschwindigkeit von 4 $^m/_s$ (= $14,4$ $^{km}/_h$) bei einem Modellfahrzeug im Maßstab 1:10 mag auf den ersten Blick sehr hoch wirken, da ein realer PKW bei einer Geschwindigkeit von 40 $^m/_s$ (= 144 $^{km}/_h$) nicht in der Lage wäre, in Verkehrsführungen, wie sie in der Stadt vorkommen, autonom zu fahren. Ein geplantes Szenario für das Modellfahrzeug ist jedoch die Vorhersage langer Geraden, bspw. durch eigene aufgezeichnete Karten mittels Inertialsensoren, auf denen die Höchstgeschwindigkeit erreicht werden soll. Sobald Kurven oder Hindernisse durch die *Computer Vision* erkannt werden oder auf Grundlage einer Karte bekannt sind, wird die Aufgabe darin bestehen, die Geschwindigkeit zu drosseln.

2.4 Berechnung der benötigten Motoreigenschaften

2.4.1 Berechnung der Drehzahl

Bei der Berechnung der Drehzahl ist die Geschwindigkeit des Fahrzeugs maßgebend. Über die Raddrehzahl, die sich aus dem Quotienten von Geschwindigkeit und Radumfang ergibt,

sowie über das Übersetzungsverhältnis des Getriebes, das bei diesem Modellfahrzeug starr verkoppelt ist, lässt sich die Motordrehzahl für jede beliebige Geschwindigkeit berechnen.

Die folgende Berechnung dient zum einen der Feststellung, ob die benötigte Motordrehzahl für die gewünschte Höchstgeschwindigkeit das empfohlene Maximum bei Dauerbelastungen nicht überschreitet, da ansonsten die Lebensdauer des Elektromotors stark darunter leiden würde. Zum anderen lässt sich mit der Berechnung feststellen, welche Spannung nötig wäre, um den Motor auf die gewünschte Drehzahl bei Höchstgeschwindigkeit zu beschleunigen. Darüber hinaus bedarf es der Drehzahl zur Berechnung der benötigten Leistung im Nennlastpunkt.

Nachfolgend wird die schrittweise Berechnung der benötigten Motordrehzahl bei Höchstgeschwindigkeit aufgeführt.

1. Drehzahl am Rad / an der Hinterachse:

$$
\begin{aligned}
\text{Radradius} \quad & 31\ mm \\
\text{Radumfang} \quad & 2 \cdot \pi \cdot 31\ mm = 194,78\ mm \\
\text{Raddrehzahl} \quad & \frac{4\ \frac{m}{s}}{194,78\ mm} = \frac{4000\ \frac{mm}{s}}{194,78\ mm} = 20,54\ \frac{U}{s} \\
& 20,54\ \frac{U}{s} \cdot 60\ s = \underline{\underline{1\,232,4\ \frac{U}{min}}}
\end{aligned}
$$

2. Drehzahl am Hauptzahnrad:

$$
\begin{aligned}
\text{Anzahl Zähne Hinterachszahnrad} \quad & 38 \\
\text{Anzahl Zähne Hauptzahnrad (klein)} \quad & 20 \\
\text{Übersetzungsverhältnis} \quad & \frac{38}{20} = 1,9 \\
\text{Hauptzahnrad-Drehzahl} \quad & \text{Raddrehzahl} \cdot 1,9 \\
& = 1\,232,4\ \tfrac{U}{min} \cdot 1,9 = \underline{\underline{2\,341,56\ \frac{U}{min}}}
\end{aligned}
$$

3. Drehzahl am Motor:

$$
\begin{aligned}
\text{Anzahl Zähne Hauptzahnrad (groß)} \quad & 84 \\
\text{Anzahl Zähne Motorzahnrad} \quad & 22 \\
\text{Übersetzungsverhältnis} \quad & \frac{84}{22} = 3,81 \\
\text{Motordrehzahl} \quad & \text{Hauptzahnrad-Drehzahl} \cdot 3,81 \\
& = 2\,341,56\ \tfrac{U}{min} \cdot 3,81 = \underline{\underline{8\,921,34\ \frac{U}{min}}}
\end{aligned}
$$

Um die gewünschte Höchstgeschwindigkeit von 4 $^{m}/_{s}$ zu erreichen, wird bei einem Übersetzungsverhältnis von $3,81 \cdot 1,9 = 7,24$ zwischen Hinterachszahnrad und Motorzahnrad eine Drehzahl von $8\,921,34\ ^{U}/_{min}$ am Motor benötigt.

2.4.2 Berechnung der Leistung

Für die Ermittlung der maximal benötigten Leistung bedarf es der Berechnung der verschiedenen Fahrwiderstände bei der Beschleunigung auf Höchstgeschwindigkeit und während konstanter Fahrt mit Höchstgeschwindigkeit. Aus diesen Fahrwiderständen lässt sich das benötigte Drehmoment am Rad berechnen, welches wiederum unter Berücksichtigung von Übersetzung und Wirkungsgrad des Getriebes das am Motor erforderliche Moment ergibt. Die benötigte Leistung lässt sich sodann aus dem Drehmoment am Motor und der Drehzahl bei Höchstgeschwindigkeit berechnen.

2.4.2.1 Fahrwiderstand

Der Fahrwiderstand setzt sich aus mehreren Komponenten zusammen. Diese Komponenten sind der Beschleunigungswiderstand, der Rollwiderstand, der Luftwiderstand und der Steigungswiderstand [18, S. 324 ff.].

Beschleunigungswiderstand
Der Beschleunigungswiderstand existiert aufgrund der Trägheit von Massen und entsteht durch die Änderung der Geschwindigkeit – also der Beschleunigung – des Fahrzeugs.

Der Beschleunigungswiderstand ist wie folgt definiert: [18, S. 328]

$$F_B = k_m \cdot m \cdot a$$

Formelzeichen	*Bezeichnung*	*Einheit*
F_B	Beschleunigungswiderstand	N
k_m	Massenfaktor	-
m	Masse	kg
a	Beschleunigung	$\frac{m}{s^2}$

Um den Beschleunigungswiderstand berechnen zu können, wird zunächst die gewünschte Beschleunigung festgelegt. Die Beschleunigung lässt sich aus der Zeit ableiten, innerhalb derer das Fahrzeug auf eine bestimmte Geschwindigkeit gebracht werden soll. Für die Beschleunigung des Modellfahrzeugs wird das Erreichen der Höchstgeschwindigkeit von 4 $^m/_s$ angenommen. Um diese Höchstgeschwindigkeit zu erreichen, wird eine Zeit von einer Sekunde angenommen. Diese ist zwar nicht maßstabsgetreu, denn kein realer PKW beschleunigt von 0 auf 144 $^{km}/_h$ in einer Sekunde, entspricht aber dennoch den Werten eines gewöhnlichen Modellfahrzeugs. Die Beschleunigung von 0 auf 4 $^m/_s$ beträgt somit:

$$a = \frac{4\ \frac{m}{s}}{1\ s} = 4\ \underline{\underline{\frac{m}{s^2}}}$$

Der Massenfaktor stellt die Trägheiten der rotierenden Teile als dimensionslose Zahl dar und erfasst damit die scheinbare Vergrößerung der Fahrzeugmasse [18, S. 328]. Rotierende Teile mit einem erwähnenswerten Radius und Gewicht sind lediglich die vier Räder mit einem

Gewicht von je 24 g. Bei dem Modellfahrzeug, dessen Gesamtgewicht sich auf ca. 2 kg beläuft, ist der Massenfaktor somit zu vernachlässigen und lässt sich daher als 1 annehmen.

$$\begin{aligned} F_B &= k_m \cdot m \cdot a \\ &= 1 \cdot 2\ kg \cdot 4\ \frac{m}{s^2} \\ &= \underline{\underline{8\ N}} \end{aligned}$$

Der Beschleunigungswiderstand beträgt 8 N.

Rollwiderstand

Der Rollwiderstand (auch Rollreibung) entsteht durch die Arbeit, die sich aus der Walkarbeit am Rad und der Arbeit für die Verformung der Straße zusammensetzt.

Der Rollwiderstand ist wie folgt definiert: [18, S. 324]

$$F_R = f_r \cdot m \cdot g \cdot \cos\alpha$$

Formelzeichen	*Bezeichnung*	*Einheit*
F_R	Rollwiderstand	N
f_r	Rollwiderstandsbeiwert	-
m	Masse	kg
g	Erdbeschleunigung ($= 9,81$)	$\frac{m}{s^2}$
α	Steigungswinkel	°

Der Rollwiderstandsbeiwert (auch Rollwiderstandskoeffizient) gibt die Paarung von Fahrbahn und Rad als dimensionslose Zahl wieder. Die Paarung „PKW-Reifen auf Asphalt“ hat bspw. einen Rollwiderstandsbeiwert von 0,013 [18, S. 325]. Durch die Cosinus-Funktion für den Steigungswinkel α wird abgebildet, dass der Rollwiderstand bei Geradeauslauf am größten ist, da die Gewichtskraft hierbei senkrecht auf die Fahrbahn wirkt.

Da keine Daten für den Rollwiderstandsbeiwert von Modellfahrzeugen in der Literatur existieren und eine exakte Bestimmung des Rollwiderstandsbeiwerts durch messtechnisches Verfahren – bspw. Ausrollversuche – im Rahmen dieser Arbeit nicht sinnvoll durchgeführt werden kann, muss eine Annahme bezüglich des Rollwiderstandsbeiwerts getroffen werden. Hierfür sollten die genannten einzelnen Faktoren des Rollwiderstandsbeiwerts betrachtet werden. Für die Walkarbeit stellt die Verformung eines Reifens einen Anhaltspunkt dar. Je geringer die Verformung, desto geringer ist die Walkarbeit [18, S. 324].

Bei den aktuell am Modellfahrzeug verwendeten Reifen handelt es sich um welche aus dem professionellen Modellbau mit einer sehr geringen Flankenhöhe von 3 mm, so dass von einer sehr geringen Verformung und somit Walkarbeit ausgegangen werden kann. Dies bestätigt auch der optische Eindruck. Die Verformung der Reifen unter der Belastung des Chassis inklusive Aufbauten ist für das menschliche Auge, anders als bei PKW-Reifen, denen man die Verformung unter Belastung deutlich ansieht, nicht feststellbar.

Die Verformung der Fahrbahn ist ein weiterer Faktor für die Annahme des Rollwiderstandsbeiwerts. Hierbei soll von der Fahrbahn unter Wettbewerbsbedingungen ausgegangen werden, die aus einem gummiartigem Tanzbelag besteht, so dass aufgrund des verhältnismäßig weichen Materials von einer größeren Verformung als bei einer Betonfahrbahn ausgegangen werden muss.

Durch die geringere Walkarbeit des Modellbaureifens im Vergleich zu einem Autoreifen auf der einen Seite und der größeren Verformung der Fahrbahn auf der anderen Seite, erscheint die Annahme des Rollwiderstandsbeiwerts mit 0,02 als ein realistischer Maximalwert.

$$\begin{aligned} F_R &= m \cdot g \cdot f_r \cdot \cos 0\ ^\circ \\ &= 2\ kg \cdot 9,81\ \frac{m}{s^2} \cdot 0,02 \cdot 1 \\ &= \underline{\underline{0,4\ N}} \end{aligned}$$

Als Rollwiderstand wird daher ein Wert von näherungsweise 0,4 N angesetzt.

Luftwiderstand

Der Luftwiderstand entsteht durch den Druck der umströmenden Luft auf die Stirnfläche des Fahrzeugs und der Reibung dieser Strömung am Fahrzeug.

Der Luftwiderstand ist wie folgt definiert: [18, S. 325]

$$F_L = \frac{1}{2} \cdot \rho_{Luft} \cdot c_w \cdot A \cdot v^2$$

Formelzeichen	*Bezeichnung*	*Einheit*
F_L	Luftwiderstand	N
ρ_{Luft}	Luftdruck	$\frac{kg}{m^3}$ $(1,2\ \frac{kg}{m^3}$ bei NN)
c_w	Luftwiderstandsbeiwert	-
A	Stirnfläche	m^2
v	Geschwindigkeit	$\frac{m}{s}$

Der Luftwiderstandsbeiwert (auch Luftwiderstandskoeffizient oder c_w-Wert) ist ebenfalls eine dimensionslose Zahl und stellt gewissermaßen einen Wert für die Windschnittigkeit eines Fahrzeugs dar, die umso besser ist, je kleiner der Luftwiderstandsbeiwert ist [18, S. 354].

Wie beim Rollwiderstandsbeiwert existieren auch hierfür keine Daten in der Literatur. Auch hier könnte ein exakter Wert durch eine Messung im Windkanal bestimmt werden, die allerdings im Rahmen dieser Arbeit wiederum nicht praktikabel ist, so dass auch hier eine Annahme getroffen werden muss. Dabei ist es sinnvoll, zunächst die Karosserie des Modellfahrzeugs ohne den Aufbau, an dem die Kamera befestigt ist, und anschließend den Aufbau alleine zu betrachten. Die Körper können so anteilig mit ihrer Stirnfläche und dem jeweils angenommen c_w-Wert zum Gesamtwert beitragen.

Für die Annahme des c_w-Wertes der Karosserie gibt es zwei Anhaltspunkte. Zum einen kann man die Form der Karosserie grob einer konvexen Halbkugel annähern. Für eine solche

Form existiert ein c_w-Wert, der mit $0,34$ angegeben wird [25]. Zum anderen handelt es sich bei der Karosserie des Modellfahrzeugs um ein Modell eines realen Fahrzeugs, nämlich des Audi TT Coupé (TT 8J Modellreihe), daher lässt sich auch der c_w-Wert des tatsächlichen Fahrzeugs als Näherung annehmen. Audi gibt den c_w-Wert des Audi TT 8J mit 0,3 an [6]. Da es sich bei der Modellkarosserie lediglich um eine einfache Plastikform handelt, sind die aerodynamischen Feinheiten des tatsächlichen Fahrzeugs nicht nachgebildet. So führt z.B. die vollständige Umströmung der Modellkarosserie im Bereich des Motors im Gegensatz zur Durchströmung des Motorraums beim realen Fahrzeug zu einer positiven Beeinflussung des c_w-Wertes. Andererseits bewirkt die Öffnung im Dach der Modellkarosserie für den Aufbau der Kamera eine negative Beeinflussung des c_w-Wertes.

Betrachtet man die Abweichung vom realen Fahrzeug, so lässt sich der c_w-Wert einer konvexen Halbkugel von 0,34 als obere Grenze annäheren und stellt somit einen realistischen Wert für den c_w-Wert des Modellfahrzeugs dar.

Der c_w-Wert des Kameraaufbaus kann als eine senkrecht zur Strömung stehende Platte angenommen werden. Der c_w-Wert eines solchen Körpers wird mit $1,2$ angegeben [25].

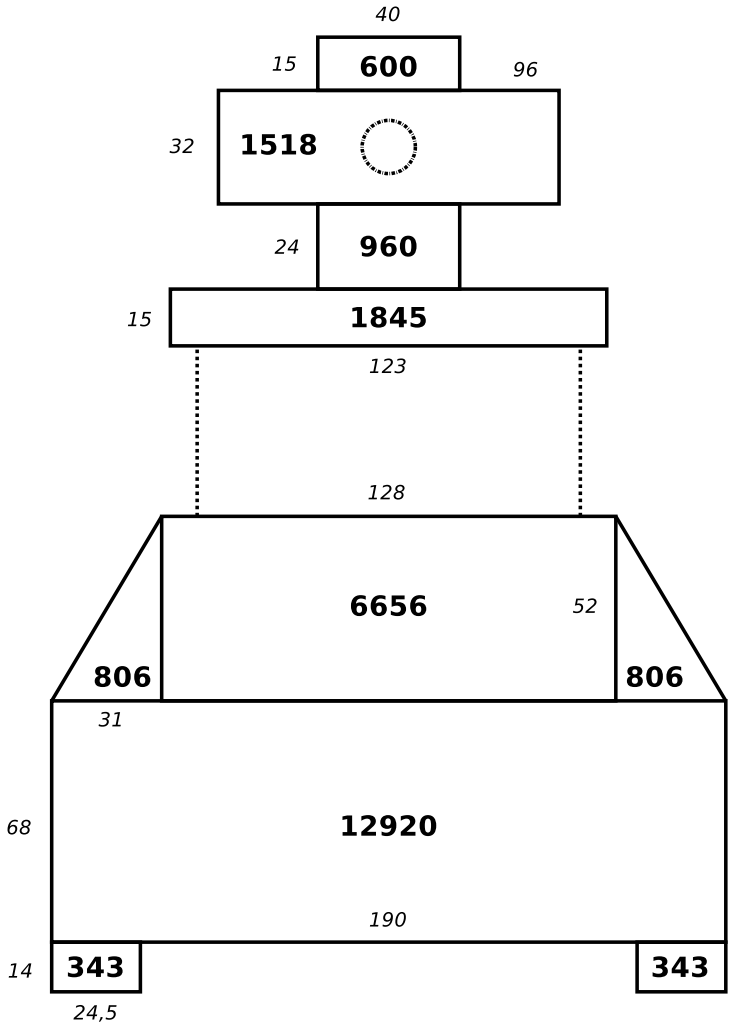


Abbildung 2.4: Fahrzeugstirnfläche anhand einer Skizze des Fahrzeugquerschnitts

In Abbildung 2.4 ist die Stirnfläche anhand einer Skizze des Fahrzeugquerschnitts dargestellt. Dabei ist der Kameraaufbau mit der nach vorne ausgerichteten Kamera abgebildet, da sich zum Zeitpunkt dieser Arbeit die Konstruktion des Aufbaus für die zukünftige *Omnicam* noch in der Entwicklung befand. Die Stirnfläche des Modellfahrzeugs inklusive des Aufbaus ließ sich messtechnisch ermitteln, indem die äußersten Punkte der Karosserie und des Aufbaus in horizontaler und vertikaler Ausrichtung gemessen wurden. Dabei beträgt die Stirnfläche der Karosserie des Fahrzeugs 21 874 mm^2, die Stirnfläche des Kameraaufbaus 4 923 mm^2. Daraus ergibt sich eine gesamte Stirnfläche von 26 797 mm^2.

$$\begin{aligned} F_L &= \frac{1}{2} \cdot \rho_{Luft} \cdot c_w \cdot A \cdot v^2 \\ &= \frac{1}{2} \cdot \rho_{Luft} \cdot (c_{w_{Auto}} \cdot A_{_{Auto}} + c_{w_{Aufbau}} \cdot A_{_{Aufbau}}) \cdot v^2 \\ &= \frac{1}{2} \cdot 1,2 \ \frac{kg}{m^3} \cdot (0,34 \cdot 0,021874 \ m^2 + 1,2 \cdot 0,004923 \ m^2) \cdot 16 \ \frac{m}{s} \\ &= \underline{\underline{0,13 \ N}} \end{aligned}$$

Der Luftwiderstand beträgt somit näherungsweise 0,13 N.

Steigungswiderstand
Im Rahmen des Carolo-Cups finden keine Fahrten an Steigungen statt. Deshalb wird dieser Fahrwiderstand nicht in die Berechnung der benötigten Leistung einbezogen.

Gesamtfahrwiderstand
Mit der Addition der einzelnen Fahrwiderstände erhält man den gesamten Fahrwiderstand, den das Modellfahrzeug überwinden muss.

$$\begin{aligned} F_G &= F_B + F_R + F_L + F_S \\ &= 8 \ N + 0,4 \ N + 0,13 \ N + 0 \ N \\ &= \underline{\underline{8,53 \ N}} \end{aligned}$$

Die Räder müssen demnach eine Kraft von maximal 8, 53 N auf die Fahrbahn übertragen.

Es ist deutlich zu erkennen, dass der Beschleunigungswiderstand mit 8 N den weitaus größten Anteil am Gesamtfahrwiderstand einnimmt. Eine halb so starke Beschleunigung würde daher den Gesamtfahrwiderstand annäherend halbieren, was für die Auswahl des Motors unter Umständen von Bedeutung sein kann, denn die angenommene Beschleunigung von 4 m/s^2 ist keine kritische Anforderung.

2.4.2.2 Drehmoment

Die Antriebskraft wird von den Rädern durch eine Drehbewegung – somit in Form von Drehmoment – an die Fahrbahn übertragen und in Fortbewegung umgesetzt. Mit Hilfe des Radius der Räder und der aufzubringenden Kraft gegen den Fahrwiderstand lässt sich das benötigte Drehmoment am Rad berechnen.

Das Drehmoment ist wie folgt definiert: [18, S. 34]

$$M = r \cdot F$$

Formelzeichen	*Bezeichnung*	*Einheit*
M	Drehmoment	Nm
r	Länge des Hebels	m
F	Kraft	N

$$\begin{aligned} M_{Rad} &= r_{Rad} \cdot F_G \\ &= 31\ mm \cdot 8,53\ N = 0,031\ m \cdot 8,53\ N \\ &= 0,264\ Nm = \underline{\underline{264\ mNm}} \end{aligned}$$

Daraus folgt, dass die Räder mit einem Drehmoment von 264 mNm angetrieben werden müssen.

Durch das Getriebe mit dem Übersetzungsverhältnis von 7,24 wird das benötigte Drehmoment am Motor gewandelt. Zu beachten ist hierbei, dass pro Getriebestufe ein Verlust von etwa 6 % durch Reibung auftritt [22, S. 328].

In unserem Modellfahrzeug gibt es zwei Getriebestufen:

$$\begin{aligned} \text{Motorzahnrad} &\leftrightarrow \text{Hauptzahnrad} \\ \text{Hauptzahnrad} &\leftrightarrow \text{Hinterachszahnrad} \end{aligned}$$

Bei einem Verlust von 6 % pro Stufe liegt der Wirkungsgrad bei:

$$0,94 \ \cdot \ 0,94 = 0,8836$$

$$\begin{aligned} M_{Motor} &= \frac{M_{Rad}}{\text{Übersetzungsverh.} \cdot \text{Wirkungsgrad}} \\ &= \frac{264\ mNm}{7,24 \cdot 0,8836} \\ &= \underline{\underline{41,27\ mNm}} \end{aligned}$$

Der Motor benötigt somit ein maximales Drehmoment von $41,27\ mNm$.

2.4.2.3 Leistung

Aus dem in Abschnitt 2.4.2.2 berechneten Drehmoment und der in Abschnitt 2.4.1 berechneten Drehzahl lässt sich nun die benötigte Leistung ermitteln. Die Leistung ist das entscheidende Kriterium, das der Elektromotor im Hinblick auf die geforderten Fahreigenschaften erfüllen muss.

Leistung ist definiert als Energie pro Zeit bzw. Arbeit pro Zeit. Aus diesem Zusammenhang ergibt sich, dass das Produkt aus Kraft und Geschwindigkeit eine Leistung ist. Bei einer rotatorischen Bewegung besteht dieser Zusammenhang zwischen Drehmoment und Winkelgeschwindigkeit.

Definition der Leistung über das Drehmoment und die Winkelgeschwindigkeit: [18, S. 34]

$$P = M \cdot \omega = M \cdot \frac{n}{60} \cdot 2 \cdot \pi$$

Formelzeichen	*Bezeichnung*	*Einheit*
P	Leistung	W
M	Drehmoment	Nm
ω	Winkelgeschwindigkeit	$\frac{rad}{s}$
n	Drehzahl	$\frac{U}{min}$

$$\begin{aligned} P &= M_{Motor} \cdot \frac{n_{Motor}}{60} \cdot 2 \cdot \pi \\ &= 41,27\ mNm \cdot \frac{8\,921,34\ \frac{U}{min}}{60\ s} \cdot 2 \cdot \pi \\ &= 0,041\,27\ Nm \cdot 148,69\ \frac{U}{s} \cdot 2 \cdot \pi \\ &= \underline{\underline{38,56\ W}} \end{aligned}$$

Das Ergebnis zeigt, dass der Elektromotor eine maximale Abgabeleistung von $38,56\ W$ benötigt, um den geforderten Fahreigenschaften gerecht zu werden.

Da sich Leistung auch aus Kraft und Geschwindigkeit ergibt, ließe sich die maximal benötigte Abgabeleistung des Elektromotors unter Berücksichtigung des Wirkungsgrads des Getriebes auch direkt aus dem Fahrwiderstand und der Höchstgeschwindigkeit berechnen. Das maximal benötigte Drehmoment ist allerdings auch eine wichtige Größe zur Auswahl des richtigen Motors, weswegen der oben aufgeführte Weg gewählt wurde.

2.5 Ergebnisse und Auswahl

2.5.1 Elektromotor

Anhand der berechneten Anforderungen an den Motor lässt sich nun aus dem Portfolio der bürstenlosen Gleichstrommotoren von Faulhaber der Motor, der mit seinen Eckdaten am dichtesten an den Berechnungen liegt, auswählen. Mit dem Blick ins Datenblatt der einzelnen Motoren lassen sich die entscheidenden Eckdaten herausfinden.

Eine wichtige Größe ist hierbei die Drehzahlkonstante, da die Drehzahl linear von der anliegenden Spannung abhängig ist. Zu beachten ist hierbei, dass die Drehzahl bei Höchstgeschwindigkeit mit der Spannung der verwendeten Akkus (vierzellige Lithium-Polymer-Akkus

mit $14,8\ V$ Nennspannung) zu erreichen sein muss. Die maximale Abgabeleistung lässt sich direkt ablesen und mit der geforderten Leistung vergleichen. Das Drehmoment eines Elektromotors lässt sich nicht pauschalisieren, sondern ist von der anliegenden Stromstärke abhängig und wird daher nicht im Datenblatt angegeben. Lediglich das Anhaltemoment, das aussagt, bei welchem anliegenden Moment der Motor aus Nenndrehzahl stehen bleibt, wird konkret angegeben. FAULHABER empfiehlt für das Lastdrehmoment weniger als die Hälfte des Anhaltemoments. Kurzzeitig lassen sich auch höhere Momente abrufen, halten diese jedoch an, führen sie aufgrund der großen Ströme zu erhöhtem Verschleiß durch die thermische Belastung des Motors. Nicht zu vernachlässigen war die Anforderung an die Bohrungen zur Befestigung des Motor im neuen Chassis (Abschnitt 2.3).

Die genaue Betrachtung ergab, dass die berechneten Eigenschaften mit den Eckdaten folgenden Motors die größte Übereinstimmung hervorbrachten: *3056 012 B K1155*.

Tabelle 2.2 zeigt eine Gegenüberstellung der Anforderungen bzw. der berechneten Werte mit den Eckdaten des gewählten Motors. Es ist deutlich zu erkennen, dass der gewählte Motor *3056 012 B* alle Kriterien bis auf die der Abgabeleistung erfüllt. Die Abgabeleistung des Motors erreicht erst bei $22\,000\ {}^{U}/_{min}$ das Maximum [8] und erfüllt somit nicht die Anforderungen. Auswirken wird sich dies allerdings nur auf die maximal erreichbare Beschleunigung, denn diese hat den mit Abstand größten Anteil am Gesamtfahrwiderstand. Sie unterliegt jedoch keiner kritischen Anforderung, wodurch dieser Motor für die gestellten Anforderungen dennoch gut geeignet ist.

Eigenschaft	*Berechnung*	*Motor* 3056 012 B
Drehzahlkonstante	-	$750\ \frac{\frac{U}{min}}{V}$
Spannung bei V_{max}	max. $14,8\ V$	$\frac{8\,921,34\ \frac{U}{min}}{750\ \frac{\frac{U}{min}}{V}} = 11,9\ V$
Anhaltemoment	-	$95\ mNm$
Lastdrehmoment	$41,27\ mNm$	$\frac{95\ mNm}{41,72\ mNm} > 2$
Abgabeleistung	$38,56\ W$	$48\ W$ bei $22\,000\ \frac{U}{min}$
Abstand der Befestigungsbohrung des Motors (180 ° versetzt)	19-30 mm	19 mm
min. Länge der Motorwelle	12 mm	14 mm
max. Durchmesser des Motors	38 mm	30 mm
max. Länge des Motors	65 mm	56 mm

Tabelle 2.2: Gegenüberstellung der berechneten Eigenschaften und der Eckdaten des gewählten Motors — *Quelle:* [8]

Bei der Modellbezeichenung bedeuten dabei *3056* die Außenabmessung von 30 mm Durchmesser und 56 mm Länge, *012* die Nennspannung von 12 V und *B*, dass es sich um eine

bürstenlose Kommutierung handelt. *K1155* bezeichnet lediglich die angepasste Ausführung des Motors für den gewählten Regler [13, 8] (Abschnitt 2.5.2).

2.5.2 Regler

Wie in Abschnitt 2.2 erwähnt, ist die Auswahl eines Reglers stark an die Auswahl des Motors gekoppelt. Im Portfolio von FAULHABER stehen hier zwei unterschiedliche Modellgruppen zur Auswahl. Zu der einen Modellgruppe gehören die *Speed Controller*. Diese funktionieren ähnlich wie übliche Modellbauregler und können die Drehzahl eines Motors regeln und dabei die Ströme auf ein Maximum begrenzen. Angesteuert werden sie über Pulsweitenmodulation [13]. Die andere Modellgruppe sind die *Motion Controller*. Die Motion Controller ermöglichen nicht nur Rampenbeschleunigung sowie -bremsung und sogar Schrittmotorbetrieb, sie lassen sich vorzugsweise mit einem ASCII[4]-Befehlssatz über eine RS-232[5] Schnittstelle oder CAN steuern. Darüber hinaus gibt es die Möglichkeit sämtliche Betriebsparameter im laufenden Betrieb abzufragen und die Auswertung der Hall-Sensoren des Motors zu Zwecken der Odometrie zu nutzen. Hierbei werden Impulse über eine Schnittstelle ausgeliefert, die mittels des Mikrocontrollers ausgewertet und somit zur Erfassung der zurückgelegten Strecke genutzt werden können [10]. Ein zentraler, externer Drehgeber wird durch die Verwendung eines Motion Controllers unnötig. Über den ASCII-Befehlssatz lassen sich alle Funktionen des Reglers steuern. Dazu gehört bspw. die Drehzahlvorgabe durch eine direkte Angabe in Umdrehung pro Minute, was gegenüber Pulsweitenmodulation ein deutliches Plus an Präzision ist und insbesondere keiner Implementierung eigener Regelungslogik bzw. Regelkreise bedarf.

Der Umfang der Funktionalitäten sowie die vielseitige und dennoch einfache Handhabung des Motion Controllers bietet gegenüber Speed Controllern einen deutlichen Vorteil. Für den Betrieb mit dem Motor *BL 3056 012 B* empfiehlt FAULHABER die Verwendung der Motion Controller *MCBL 3006 S RS* oder *MCBL 3003 P RS*, wobei beide in den Eckdaten genau übereinstimmen. Der *3003* wird lediglich ohne eigenes Gehäuse verschickt und verfügt daher über keine eigenen Kühlkörper, was den maximal zulässigen Dauerstrom auf 3 A [13] begrenzt. Da sich der Einbau des Reglers ohne eigenes Gehäuse in das Modellfahrzeug schwierig gestalten würde, bietet sich eine Lösung mit eigenem Gehäuse an.

Die Wahl fiel deshalb auf folgenden Motion Controller: *MCBL 3006 S RS*

Dabei bedeute *MCBL*, dass es sich um einen Motion Controller für bürstenlose Motoren handelt. *3006* bedeutet, dass die maximale Betriebsspannung 30 V und der maximal zulässige Dauerstrom bei 6 A liegt. *S* bedeutet, dass die Anschlüsse mit Schraubklemmen befestigt werden und *RS*, dass der Motion Controller über eine RS-232 Schnittstelle zur Kommunikation verfügt [11, S. 8]. Für den Einsatz im autonomen Modellfahrzeug ist keine CAN-Anbindung nötig, da es weiter keine Geräte gibt, die CAN nutzen. Die RS-232 Schnittstelle stellt somit für diesen Einsatz eine optimale Lösung dar.

[4] American Standard Code for Information Interchange
[5] Standard für serielle Schnittstellen

3 Einbau, Verschaltung und Konfiguration

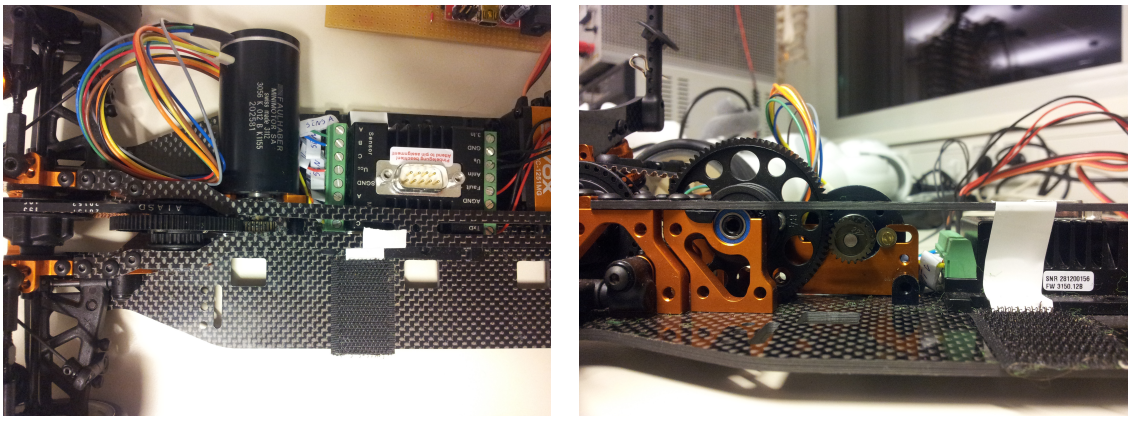
3.1 Einbau der Komponenten

Der Regler ließ sich ohne weitere Vorbereitung im Chassis unterbringen.

Der Einbau des Elektromotors war mit leichten Komplikationen behaftet, die allerdings bereits vorhersehbar waren. Bei dem Elektromotor *MCBL 3006 S RS* handelt es sich nämlich (wie bei allen Modellen von FAULHABER) nicht um einen Motor, der in erster Linie für den Einsatz in Modellfahrzeugen konzipiert ist. Merkmale dafür sind der im Modellbau unübliche Wellendurchmesser von 4 mm und die Form des Gehäuses auf der Seite des nutzbaren Wellenendes. Hier befindet sich eine Erhebung von $1,4$ mm mit einem Radius von 13 mm, die die Befestigung des Motors an der Halterung des Chassis zunächst erschwerte. Um der Erhebung an der Halterung Platz zu verschaffen, wurde der Ausschnitt der Halterung für die Motorwelle durch Feilen um 1-2 mm vergrößert. Um übliche Motorzahnräder aus dem Modellbau nutzen zu können, wurde ein 22-zähniges Motorzahnrad (Abschnitt 2.3) mit einem (für Modellbau typischen) Lochdurchmesser von $3,17$ mm mit Hilfe einer Ständerbohrmaschine auf einen Lochdurchmesser von 4 mm aufgebohrt. Mit dieser Maßnahme ließ sich sodann das Motorzahnrad auf der Welle des Motors befestigen.

Die vier Befestigungsbohrungen des Motors an der Seite des nutzbaren Wellenendes liegen um 90 ° versetzt auf einem Kreis mit 19 mm Durchmesser. Die Schlitze der Motorhalterung am Chassis sind für Befestigungsbohrungen von 19-30 mm ausgelegt, womit sich die Befestigungsmöglichkeit des Motors am Limit befindet und keinerlei Bewegungsspiel für die Justierung des Zahnflankenspiels zwischen Motorzahnrad und Hauptzahnrad vorhanden ist. Für das gewählte 22-zähnige Motorzahnrad stellte sich der daraus resultierende feste Abstand zum Hauptzahnrad als zu groß dar, so dass kein befriedigendes Zahnflankenspiel möglich war. Um letzteres zu verbessern, wurde der Schlitz, der das Bewegungsspiel einschränkte, ebenfalls (wiederum durch Feilen) um 2-3 mm vergrößert.

Abbildung 3.1 zeigt den eigebauten Motor und Regler in zwei Perspektiven im neuen Chassis.



(a) Motor und Regler von oben (b) Motor und Regler von der Seite

Abbildung 3.1: Der eingebaute Motor und Regler im neuen Chassis

3.2 Verschaltung der Komponenten

3.2.1 Verschaltung von Regler und Elektromotor

Die Verschaltung von Regler und Elektromotor erwies sich als problemlos. Genaue Anleitungen zu der Verschaltung der einzelnen Litzen des Motors an den Anschlüssen des Reglers sind in dem Datenblatt des Motors und in der Betriebsanleitung des Reglers zu finden [8, 11]. Die Verwendung aller acht vorhandenen Litzen des Motors ist dabei obligatorisch. Drei der Litzen bilden die Signalleitungen der Hall-Sensoren und drei weitere Litzen die Schaltleitungen für die Spulen. Die zwei verbleibenden Litzen werden als Spannungsversorgung (Vcc) für die Hall-Sensoren und als Erdung (GND) verwendet [11].

3.2.2 Verschaltung von Regler und Mikrocontroller

Für die Verschaltung von Regler und Motorboard mussten zunächst einige Vorbereitungen getroffen werden. Für die Steuerung des neuen Antriebs wurde ein weiteres Motorboard verwendet, das allerdings nur für die Entwicklung und das Testen der Kommunikationsschnittstellen genutzt werden sollte. Somit wurden auch nur die Anschlüsse der seriellen RS-232 Schnittstelle und des Fehlerausgangs des Reglers benötigt. Es war daher sinnvoll, eine neue kleine Steckplatine zu löten, an der nur die benötigten Anschlüsse mit Steckverbindungen versehen werden mussten. Die Anschlüsse für den Empfänger der Funkfernbedienung und des Lenkservos wurden an diesem Motorboard später ebenfalls mit Steckern versehen, um die gesamte Antriebsplattform im Funkfernbedienungsmodus zu testen.

Für die Kommunikation über die RS-232 Schnittstelle musste ein ungenutzter UART[1] des Mikrocontrollers gewählt werden, dessen zwei Leitungen (receive und transmit) nicht bereits durch andere Funktionen belegt waren, denn alle übrigen Komponenten auf der neuen Platt-

[1] Universal Asynchronous Receiver Transmitter

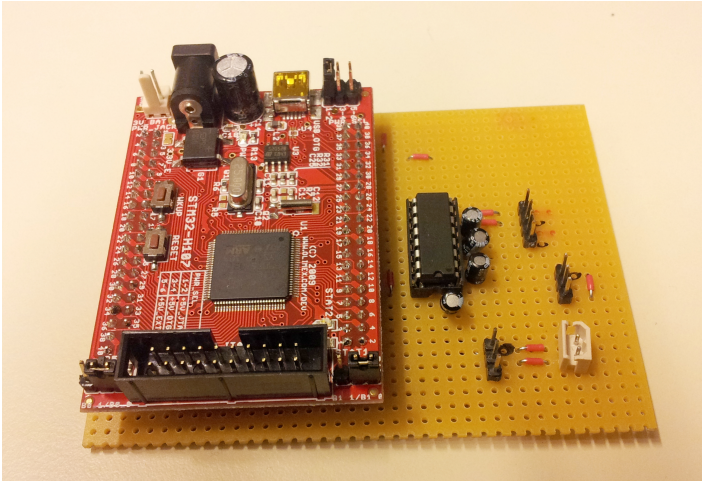


Abbildung 3.2: Das Motorboard für die Entwicklung der Antriebslogik

form sollten ihre bisherigen Anschlüsse am Motorboard und damit auch die softwareseitige Implementierung beibehalten.

Für den Fehlerausgang des Reglers, der für die Ausgabe der Impulse des internen Drehgebers verwendet wird, konnte ein Anschluss an eine der freigewordenen Leitungen gelötet werden, an die der zuvor genutzte externe Drehgeber angeschlossen war. Bei den Impulsen des internen Drehgebers handelt es sich nicht wie beim alten Drehgeber um einen 2-kanaligen Gray-Code, aus dem die Drehrichtung des Antriebs ermittelt werden konnte, sondern lediglich um ein 1-kanaliges Rechtecksignal mit einer steigenden Flanke für jeden Impuls, so dass nur noch eine Leitung benötigt wird. Die Drehrichtung kann bei dieser Konfiguration softwareseitig ausgewertet werden, da immer bekannt ist, welcher Befehl dem Regler zuletzt gesendet wurde und somit Geschwindigkeit und Drehrichtung ebenfalls bekannt sind.

Die einzigen freien Anschlüsse, an denen die Leitungen einer UART herausgereicht werden, gehören zur UART4. Der Anschluss der Sendeleitung (TxD) befindet sich dabei an Pin *EXT2-15*, der Anschluss der Empfangsleitung (RxD) an Pin *EXT2-14*. Für den Anschluss des Fehlerausgangs wird der Pin *EXT1-8* verwendet, an dem der Kanal 1 (*CH1*) des Timers *TIM1* herausgereicht wird [17]. Mit dem Timer *TIM1* werden so die Impulse durch den Drehgeber gezählt (Abschnitt 4.3).

3.2.2.1 Signalpegelanpassung

Der RS-232 Standard verwendet Signalpegel von +3 bis +15 V zur Darstellung einer logischen „0“ und −3 bis −15 V zur Darstellung einer logischen „1“ [19, S. 204 ff.] (üblich sind Pegel von −12 und +12 V). Um zum einen die Empfangsleitung der UART4 des Mikrocontrollers, die nur 5 V tolerant ist [21, S. 29], zu schützen und zum anderen beim Senden immer ein eindeutiges, zuverlässiges Signal mit −12 und +12 V zu erzeugen, wird ein Pegelwandler benötigt. Zwischen dem Mikrocontroller und dem Pegelwandler wird mit TTL[2] kompatiblen Signalen kommuniziert, die direkt vom Mikrocontroller gesendet und verstanden werden kön-

[2] Transistor-Transistor-Logik

nen. Bei TTL-Signalen wird allerdings eine logische „1“ durch einen Signalpegel von $2 \ldots 5\ V$ und eine logische „0“ durch einen Signalpegel von $0 \ldots 0,8\ V$ repräsentiert [19, S. 61] (umgekehrt zu den Signalpegeln des RS-232 Standards). Um für die ausgehenden Signale einen Pegel von -12 bis $+12\ V$ zu erzeugen, wird eine sog. Ladungspumpe benötigt. Schutzwiderstände und -dioden schützen wiederum den Eingangskanal vor zu hohen Signalpegeln [19, S. 210].

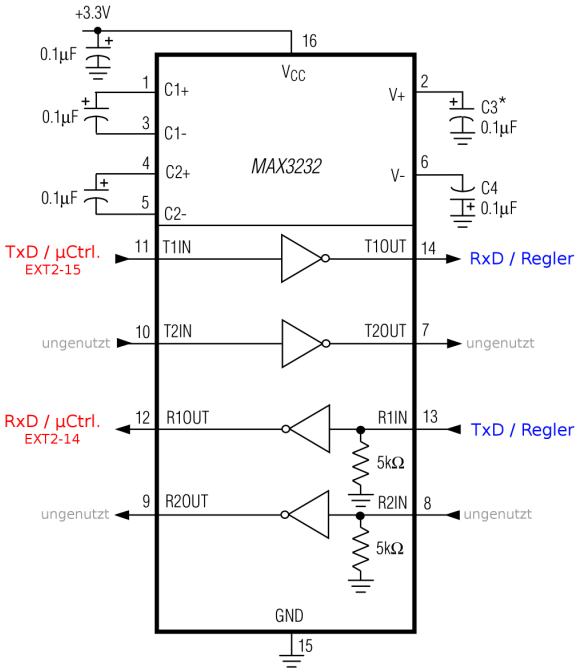


Abbildung 3.3: Schematisches Schaltbild des MAX3232, nach [15, S. 12]

Abbildung 3.3 zeigt die Verschaltung des von uns eingesetzten Pegelwandlers MAX3232 anhand eines schematischen Schaltbilds. Für die korrekte Funktion der Ladungspumpe kommt der MAX3232 mit nur fünf $0,1\ \mu F$ Elektrolytkondensatoren und einer Eingangsspannung von $3,3\ V$ aus. Diese Eingangsspannung kann direkt vom Mikrocontroller geliefert werden, so dass hier kein Anschluss an eine externe Spannungsquelle nötig ist und sich daher der MAX3232 als sehr praktikabel erwies. Bei der Verkabelung der Anschlüsse des MAX3232 und der Anschlüsse des Reglers galt es zu beachten, dass sich die Leitungen kreuzen müssen. Das heißt, dass die Sendeleitung des Reglers an die Empfangsleitung des MAX3232 angeschlossen werden muss und umgekehrt [11]. *TxD* des Reglers wurde daher an *R1IN* des MAX3232 und *RxD* des Reglers an *T1IN* des MAX3232 angeschlossen. Die Verkabelung der Ausgänge des MAX3232 an die Eingänge des Mikrocontrollers war dann entsprechend ungekreuzt (*R1OUT* an *RxD* des Mikrocontrollers, *T1IN* an *TxD* des Mikrocontrollers) [15].

Der Signalpegel des Fehlerausgangs ist direkt TTL-kompatibel und kann direkt an den Mikrocontroller angeschlossen werden. Dies war dem Handbuch des Reglers zwar nicht explizit zu entnehmen, konnte jedoch mit Hilfe eines Oszilloskops gemessen und bestätigt werden.

3.3 Konfiguration des Reglers

3.3.1 Hilfsmittel zur Konfiguration des Reglers

Alle Konfigurationsmöglichkeiten des Reglers ließen sich über die Software MOTION MANAGER, die direkt von FAULHABER angeboten wird und kostenlos von deren Homepage[3] bezogen werden kann, ausschöpfen. Mit Hilfe dieser Software konnten sämtliche Parameter über eine graphische Oberfläche konfiguriert und anschließend in den Flashspeicher des Reglers geschrieben werden, so dass auch nach dem Trennen der Spannungsversorgung die Konfigurationen erhalten blieben und somit bei jeder Verwendung korrekt gesetzt waren. Viele diese Parameter sind Grundeinstellungen des Antriebs, die bei der ersten Verwendung mit der Software MOTION MANAGER mit Hilfe eines Assistenten ermittelt werden. Eine detaillierte Anleitung zur Verwendung der Software findet sich im Handbuch [12]. Über den ASCII-Befehlssatz können jedoch auch fast alle Parameter im laufenden Betrieb durch den Mikrocontroller über die RS-232 Schnittstelle geändert werden.

3.3.2 Konfiguration der Kommunikationsparameter

Bei der Konfiguration des Reglers stand die Konfiguration der Baudrate (auch Symbolrate) an erster Stelle. Nur wenn Mikrocontroller und Regler auf die gleiche Baudrate konfiguriert sind, können die Geräte über eine RS-232 Schnittstelle kommunizieren. Die Baudrate ist die Schritt- oder Übertragungsgeschwindigkeit, mit der die Symbole übertragen werden bzw. auf der Empfangsseite erwartet werden. Bei der Kommunikation über RS-232 werden zwei Signalpegel verwendet (Abschnitt 3.2.2.1), so dass pro Schritt ein Symbol mit einem Bit übertragen werden kann. Bei RS-232 ist daher die Bitrate gleich der Baudrate [7, S. 147].

Um eine feingranulare Geschwindigkeitsregelung, d. h. eine möglichst hochfrequente Änderung der Zielgeschwindigkeit durch Befehle an den Regler zuzulassen, sollte die für den Regler höchste einstellbare Baudrate von 115 200 $^{baud}/_{s}$ gewählt werden. Der Regler und die UART4 des Mikrocontrollers wurden somit auf diese Baudrate konfiguriert. Neben der Geschwindigkeit ist auch die Syntax eines Datenrahmens zu beachten. Die Anzahl der Datenbits pro Datenrahmen muss ebenso wie die Fragen, ob Paritätsbit verwendet wird und ob das Stopbit aus einem oder zwei Bits bestehen soll, festgelegt werden. Ein Startbit schreibt der Standard vor und wird immer verwendet. Eine typische Konfiguration, die auch bei der Kommunikation zwischen Regler und Mikrocontroller zum Einsatz kommt, ist die Verwendung von acht Datenbits, also einem Byte, keinem Paritätsbit und einem Stopbit, kurz auch 8N1 bezeichnet. Die Verwendung von acht Datenbits enthält zwar ein Bit mehr als ein ASCII-Zeichen, das nach dem Standard nur sieben Bit groß ist [19, S. 379], ist aber eine Vereinfachung für das Rechnen in Byte und wird so auch für die Kommunikation mit der Einheit für die Bildverarbeitung und die Fahrtenplanung eingesetzt.

[3] http://www.faulhaber.com/MotionManager/

3.3.2.1 Berechnung der möglichen Datenrate

Pro Zeichen werden ein Startbit, acht Datenbits und ein Stopbit gesendet, was in der Addition $1 + 8 + 1 = 10$ Bits bedeutet. Daraus ergibt sich jedoch auch, dass nicht die volle Bitrate von 115 200 $^{bit}/_{s}$ für die Nutzdaten zur Verfügung steht. Das Verhältnis von Nutzdaten zu Bitrate ergibt sich aus folgender Formel:

$$115\,200\ \frac{bit}{s} \cdot \frac{8}{10} = 92\,160\ \frac{bit}{s}$$

Es können dadurch 92 160 $^{bit}/_{s}$ an Nutzdaten tatsächlich versendet werden.

Ein vollständiger Geschwindigkeitsbefehl, der an den Regler gesendet wird, besteht maximal aus sieben Zeichen. Zur Erinnerung: Die angenommene Höchstgeschwindigkeit erfordert eine Drehzahl von 8 921, 34 $^{U}/_{min}$ (Abschnitt 2.4.1) und ist damit vierstellig (Nachkommastellen können nicht übertragen werden).

1	2	3	4	5	6	7
V	[-]	X	X	X	X	\r

Das 'V' bedeutet velocity *und signalisiert dem Regler, dass nun eine Drehzahlangabe in Form der nachfolgenden Zahl gesendet wird. Das Vorzeichen '[-]' ermöglicht, dass auch eine negative Drehzahl, also eine Drehung des Motors in die entgegengesetzte Richtung, gesendet werden kann. Ein 'X' steht dabei für eine Ziffer einer maximal vierstelligen Zahl. '\r' ist ein Steuerzeichen und bedeutet* carriage return*. Dem Regler wird damit signalisiert, dass der Befehl vollständig ist [10] (Abschnitt 4.2.4).*

Es müssen pro Befehl somit maximal sieben Byte ($7\ byte \cdot 8\ \frac{bit}{byte} = 56\ bit$) gesendet werden.

$$\frac{92\,160\ \frac{bit}{s}}{56\ \frac{bit}{\text{Befehl}}} = 1\,645\ \frac{\text{Befehle}}{s}$$

Maximal 1 645 siebenstellige Geschwindigkeitsbefehle können pro Sekunde an den Regler gesendet werden. Bei kürzeren Befehlen können es auch mehr Befehle pro Sekunde sein. Da die Regelung auf eine Drehzahl allerdings nicht beliebig schnell erfolgt, könnten 1 645 Befehle pro Sekunde vom Regler jedoch nicht umgesetzt werden. Eine derart hohe Rate der Befehle würde zu einer erhöhten Belastung der Leistungselektronik führen, die für die Lebensdauer des Reglers aufgrund der daraus entstehenden thermischen Belastung nicht förderlich wäre.

3.3.3 Konfiguration der Parameter

Neben der Übertragungsgeschwindigkeit für die Kommunikation müssen auch einige Antriebs- und Betriebsparameter des Reglers für seine Aufgabe im Modellfahrzeug definiert und in den Flashspeicher des Reglers geschrieben werden. Tabelle 3.1 zeigt eine Aufstellung der wichtigsten Parameter mit kurzer Beschreibung.

Parameter	*Beschreibung*
`EN`	Regler aktivieren
`ENCOUT`	Fehlerausgang als Impulsausgang konfigurieren
`LPN32`	32 Impulse pro Motorumdrehung
`AC150`	Beschleunigungsrampe mit $150\ \frac{\frac{U}{s}}{s} \cdot 60\ s = 9000\ \frac{\frac{U}{min}}{s}$ ($\approx 4\ \frac{m}{s^2}$)
`DEC150`	Bremsrampe mit $150\ \frac{\frac{U}{s}}{s}$
`SP8921`	Maximale Drehzahl bei 8 921 $\frac{U}{s}$ ($= 4\ \frac{m}{s}$)

Tabelle 3.1: Aufstellung der wichtigsten Parameter zur Konfiguration des Reglers

4 Implementierung der Kommunikationsschnittstelle

4.1 Architektur

4.1.1 Framework

Auf dem Mikrocontroller wird das vom Berlin United Racing Team und dem Team der Berlin United FUmanoids[1] eigens entwickelte Framework *flawless* eingesetzt. Das Framework stellt u. a. ein Modul zur einfachen Initialisierung des Systems bzw. der übrigen Module (*system initializer*), eine Nachrichtenpumpe (*message pump*) für den Datenaustausch zwischen den Modulen und ein Protokoll (*generic flawless protocol*) für die Kommunikation mit der Einheit für die Bildverarbeitung und die Fahrtenplanung (in diesem Fall ein IGEPv2[2] *Single-Board* Computer) zur Verfügung.

4.1.1.1 Nachrichtenpumpe

Unter allen Modulen des Frameworks stellt sich die Nachrichtenpumpe als das zentrale Modul dar, da sich innerhalb der Nachrichtenpumpe die Betriebsschleife (*main loop*) des Mikrocontrollers befindet. Sämtliche Funktionsaufrufe seitens des Mikrocontrollers beruhen demnach auf Nachrichten. Die Nachrichten sind durch IDs in verschiedene Kategorien unterteilt, damit den einzelnen Modulen nur die für sie relevanten Nachrichten zugestellt werden.

Jedes Modul registriert sich bei seiner Initialisierung auf eine (oder auch mehrere) Nachrichtenkategorie und bekommt Nachrichten dieser Kategorie zyklisch durch die Nachrichtenpumpe zugestellt, falls eine entsprechende Nachricht vorhanden ist. Technisch gesehen wird aus der Nachrichtenpumpe heraus eine bei der Registrierung angegebene Adresse im Speicher aufgerufen, an der die *Callback*-Funktion des Moduls gespeichert ist. Bei dem Aufruf der Callback-Funktion wird wiederum die Adresse des Speicherorts der Nachricht an die Funktion übergeben. Innerhalb der Funktion wird aus der Speicheradresse die Nachricht gelesen und ausgewertet. Entsprechend der Auswertung ruft das Modul aus der Callback-Funktion heraus daraufhin die entsprechenden internen Funktionen auf.

[1] http://www.fumanoids.de/
[2] http://www.isee.biz/index.php/products/processor-boards/igepv2-board

4.1.2 Interface zur Antriebsregelung

Alle Funktionen zur Regelung und Auswertung des Antriebs sind durch das Interface *speed.h* (Listing 4.1) von der konkreten Implementierung der jeweiligen Hardware abgekapselt. Sowohl der alte als auch der neue Antrieb nutzen dieses Interface, um sich einfach in die Software-Architektur einfügen zu können, wenngleich die konkreten Implementierungen völlig unterschiedlich voneinander sind.

Listing 4.1: speed.h - Interface zur Antriebsregelung und -auswertung

```
1  typedef int32_t speed_distanceMm_t;
2  typedef int32_t speed_MMpS_t;
3  typedef int32_t speed_RPM_t;
4
5  int32_t speed_getTicks();
6  speed_distanceMm_t speed_getDistanceMm();
7
8  void speed_setSpeed(speed_MMpS_t speed);
9  speed_MMpS_t speed_getLastMmPerSec();
10 bool speed_isDrivingForward();
```

4.2 Informationen vom Mikrocontroller zum Regler

4.2.1 Konfiguration des UART

Um Informationen versenden zu können, muss ein UART konfiguriert werden (Listing 4.2). Die Konfiguration ist dabei sehr hardwarenah und erfordert das Setzen von einzelnen Bits in verschiedene Register. Mit Hilfe der Bibliothek (*library*) LIBOPENCM3[3], die eine Vielzahl von Definitionen und Funktionen für STM32-Boards zur Verfügung stellt, ist es möglich, eine vereinfachte Sicht auf die Konfiguration zu erreichen.

Wichtig bei der Konfiguration von Hardwarekomponenten des Mikrocontrollers ist die Aktivierung des Systemtakts an den entsprechenden Leitungen (*ports* und *pins*). Sobald die Leitungen mit dem Systemtakt versorgt sind, lassen sich die einzelnen Pins zum Senden (PC10) und Empfangen (PC11) konfigurieren, bevor die entscheidenden Parameter des UART gesetzt werden. Diese wurden in Abschnitt 3.3.2 erläutert. Um das Empfangen von Daten zu ermöglichen, muss zunächst für den verwendeten UART der *nested vectored interrupt controller* – ein boardeigenes System zur Verwaltung der Interruptvektortabelle – aktiviert werden. Der UART wird darüber hinaus so konfiguriert, dass nur empfangene Pakete einen Interrupt auslösen und bspw. nicht jedes erfolgreiche Senden [20, S. 183, 765]. Genutzt wird das Empfangen von Informationen des Reglers über den UART aktuell jedoch nicht, da es für den Betrieb nicht weiter erforderlich ist, könnte aber sehr wohl Aufschluss über einige

[3] http://www.libopencm3.org/

zusätzliche Parameter geben (Abschnitt 4.4). Abschließend wird die UART aktiviert und steht somit für die folgenden Aufgaben bereit.

Listing 4.2: USART.c - Konfiguration des UART

```
1  void init_transport_faulhaber(uint32_t baudrate)
2  {
3      usart_disable(TRANSPORT_DEVICE_FAULHABER);
4      // Enable IO port C clock
5      RCC_APB2ENR |= RCC_APB2ENR_IOPCEN;
6      // Enable alternate function clock
7      RCC_APB2ENR |= RCC_APB2ENR_AFIOEN;
8      // Enable UART4 clock
9      RCC_APB1ENR |= RCC_APB1ENR_USART4EN;
10
11     // Setup GPIO pin UART_4_TX on GPIO port C for transmit
12     gpio_set_mode(GPIOC, GPIO_MODE_OUTPUT_50_MHZ,
13                           GPIO_CNF_OUTPUT_ALTFN_PUSHPULL, GPIO10);
14
15     // Setup GPIO pin UART_4_RX on GPIO port C for receive
16     gpio_set_mode(GPIOC, GPIO_MODE_INPUT,
17                           GPIO_CNF_INPUT_FLOAT, GPIO11);
18
19     /* Setup UART parameters. */
20     usart_set_baudrate(TRANSPORT_DEVICE_FAULHABER, baudrate,
           rcc_ppre1_frequency);
21     usart_set_databits(TRANSPORT_DEVICE_FAULHABER, 8);
22     usart_set_stopbits(TRANSPORT_DEVICE_FAULHABER, USART_STOPBITS_1);
23     usart_set_mode(TRANSPORT_DEVICE_FAULHABER, USART_MODE_TX_RX);
24     usart_set_parity(TRANSPORT_DEVICE_FAULHABER, USART_PARITY_NONE);
25     usart_set_flow_control(TRANSPORT_DEVICE_FAULHABER, USART_FLOWCONTROL_NONE)
           ;
26
27     // Enable UART4 interrupt
28     nvic_enable_irq(NVIC_USART4_IRQ);
29
30     // Enable UART4 Receive interrupt
31     USART_CR1(TRANSPORT_DEVICE_FAULHABER) |= USART_CR1_RXNEIE;
32
33     usart_enable(TRANSPORT_DEVICE_FAULHABER);
34 }
```

4.2.2 Konstanten und Initialisierung des Reglers

Alle Funktionen, Variablen und Konstanten für die Informationen in Richtung Regler, das sind Befehle zum Setzen von Parametern und insbesondere die Befehle zur Drehzahlvorgabe, sind in *engineFaulhaber.c* implementiert.

Um die Implementierung der Funktion speed_setSpeed(speed_MMpS_t speed), in der aus einer Geschwindigkeitsvorgabe die entsprechende Motordrehzahl berechnet wird, unabhängig von der Getriebeübersetzung zu gestalten, sind die Anzahl der Zähne der Zahnräder als Konstanten definiert (Listing 4.3). Sollte sich in Zukunft an den verwendeten Zahnrädern etwas ändern, wird es ein Leichtes sein, die Definitionen zu ändern und damit die Drehzahlberechnung anzupassen. Des Weiteren gibt es noch die Definition einer maximalen Geschwindigkeit, die nicht überschritten werden kann. Diese wird zunächst auf 2 $^{m}/_{s} = 2000\ ^{mm}/_{s}$ festgelegt. Sobald die Zuverlässigkeit der *Computer Vision* ausreichend getestet wurde, kann die maximale Geschwindigkeit bis zur angenommene Höchstgeschwindigkeit von 4 $^{m}/_{s} = 4000\ ^{mm}/_{s}$ erhöht werden.

Für die Auswertung der zurückgelegten Strecke werden die Impulse pro Umdrehung, die mit dem gesetzten Parameter im Regler äquivalent sein müssen, und die sich (unter Berücksichtigung des Radumfangs und der Getriebeübersetzung) daraus ergebenden Mikrometer, die pro Impuls zurückgelegt werden, definiert. Bei Testläufen, in denen verschiedene Entfernungen entlang eines Maßbands gefahren wurden, wurden Abweichung in Höhe von ca. 2,5 % gemessen (bei vorgegebener Fahrt von 100 *cm* kam das Fahrzeug bereits nach 97,5 *cm* zum Stehen). Eine Abweichung in dieser Höhe ist allerdings für berechnete Parameter sehr gut und ließ sich durch eine zusätzliche Kalibrierung korrigieren. Die erforderliche Korrektur wurde durch mehrere Fahrten ermittelt und in Form eines Subtrahenden der Mikrometer pro Impuls ebenfalls definiert. Es ist jedoch zu beachten, dass eine solche Kalibrierung stark von der Umgebung abhängig ist. So ist bspw. der Schlupf der Räder, der bei der Abweichung vermutlich den größten Faktor ausmacht, auf allen Fahrbahnbelägen unterschiedlich und sollte daher für jeden Fahrbahnbelag neu kalibriert werden.

Listing 4.3: faulhaber.h - Definition der Konstanten

```
1  /* Maximum Speed to get all Ticks (max 4000 per sec) */
2  #define ENGINE_FAULHABER_MAX_SPEED_MM_PER_SEC 2000
3
4  /* Set measures of following elements to calculate motor rpm */
5  #define TIRE_RADIUS_MM 31.0f
6  #define FAULHABER_GEAR_TEETH 22.0f
7  #define MAIN_GEAR_BIG_TEETH 84.0f
8  #define MAIN_GEAR_SMALL_TEETH 20.0f
9  #define AXIS_GEAR_TEETH 38.0f
10
11 #define TIRE_CIRCUMFERENCE (2.0f * M_PI * TIRE_RADIUS_MM)
12 #define GEAR_RATIO ((MAIN_GEAR_BIG_TEETH / FAULHABER_GEAR_TEETH) *
13                     (AXIS_GEAR_TEETH / MAIN_GEAR_SMALL_TEETH))
14
15 /* Needs to be equal with controller parameter */
16 #define TICKS_PER_ROUND 32
17 /* Tick correction, due to tire slip. Calibrate for every road surface. */
18 #define TICK_CORRECTION 25
19 #define MICROMETER_PER_TICK (((TIRE_CIRCUMFERENCE * 1000.0f) /
20                              (GEAR_RATIO * TICKS_PER_ROUND)) - TICK_CORRECTION)
```

Bei der Initialisierung des Moduls *engineFaulhaber* werden einige Befehle zum Setzen der wichtigsten Parameter des Reglers gesendet (Listing 4.4). Auch wenn alle diese Parameter bei der ersten Konfiguration des Reglers in den Flashspeicher geschrieben wurden, sollten alle kritischen Parameter (erneut) auf die benötigten Werte gesetzt werden, um Fehlkonfigurationen auszuschließen. Eine Erläuterung zu den Befehlen ist in Abschnitt 3.3.3 zu finden.

Listing 4.4: engineFaulhaber.c - Initialisierung des Reglers

```
1  static void engine_init(void)
2  {
3      sendCommandToFaulhaber("EN");
4      sendCommandToFaulhaber("V0");
5      sendCommandToFaulhaber("ENCOUT");
6      sendCommandToFaulhaber("LPN32");
7      sendCommandToFaulhaber("AC150");
8      sendCommandToFaulhaber("DEC150");
9      sendCommandToFaulhaber("SP8921");
10 }
```

4.2.3 Erstellung der Befehle zur Drehzahlregelung

Die Funktion speed_setSpeed(speed_MMpS_t speed) (Listing 4.5) ist die wichtigste Funktion zum Regeln des Antriebs. Die Funktion bekommt die gewünschte Geschwindigkeit als Parameter übergeben und prüft zunächst, ob die Geschwindigkeit im Rahmen der definierten Höchstgeschwindigkeit liegt (Listing 4.3). Anschließend wird die benötigte Motordrehzahl aus der Geschwindigkeit berechnet. Da nur Ganzzahlwerte an den Motor geschickt werden können, werden die Nachkommastellen, die bei den Divisionen während der Berechnung entstehen, durch die automatische Typumwandlung (*cast*) auf den Ganzzahltyp abgeschnitten. Zur Minimierung der daraus resultierenden Ungenauigkeiten geschieht dies nur einmal pro Umrechnung der Geschwindigkeit in die Drehzahl.

Um nun dem Regler einen Befehl zum Anpassen der Drehzahl senden zu können, muss zunächst ein solcher Befehl aus dem Schlüsselsymbol 'V' (*velocity*) [10] und der berechneten Drehzahl erstellt werden. Dabei ist zu beachten, dass der Regler ASCII-Zeichen erwartet und die berechnete Drehzahl von einem Zahldatentyp (*int32*) in eine Zeichenkette (*string* bzw. *char array*) umgewandelt werden muss. Dies geschieht durch die Funktion itoa(**int** integer, **char** *buffer), die den übergebenen *integer* in den *buffer* als ASCII-Zeichen schreibt. Sobald dies erledigt ist, steht der Befehl zum Versenden bereit und wird der Funktion sendCommandToFaulhaber(**char** *string) übergeben.

Sobald der Befehl gesendet wurde, werden die Fahrtrichtung und damit auch die Zählrichtung des Timers *TIM1* [20, S. 310] für die Auswertung der Impulse des Drehgebers aus der gerade gesendeten Drehzahl ermittelt und gespeichert. Dafür werden auch hier Funktionen der LIBOPENCM3-Bibliothek verwendet. Des Weiteren wird die aktuelle Geschwindigkeit als

Nachricht gepostet, damit andere Module – sofern sie auf eine solche Nachricht registriert sind – über die aktuelle Geschwindigkeit informiert werden. Die Funktionen speed_isDrivingForward () und speed_getLastMmPerSec() des Interfaces *speed.h* geben über ein *return*-Statement die Werte der entsprechenden Variablen zurück (Listing 4.5).

Listing 4.5: engineFaulhaber.c - Drehzahlregelung

```
1  void speed_setSpeed(speed_MMpS_t speed)
2  {
3      /* Check if speed is inside max speed range */
4      speed_MMpS_t sendSpeed =
5              MAX(speed, -ENGINE_FAULHABER_MAX_SPEED_MM_PER_SEC);
6      sendSpeed =
7              MIN(sendSpeed, ENGINE_FAULHABER_MAX_SPEED_MM_PER_SEC);
8
9
10     /* Calculate speed with tire circumference and gear ratio
11      * We need rounds per minute (= 60 seconds) and negative values
12      * due to Faulhaber assembly direction for driving forward.
13      * We need to do it before rounding so we do it here. */
14     speed_RPM_t sendRpm = (sendSpeed / TIRE_CIRCUMFERENCE) *
15                               GEAR_RATIO * -60;
16
17     /* Build string for Faulhaber communication using itoa
18      * and put a 'V' vor VELOCITY at the beginning */
19     g_sendRpmBuffer[0] = 'V';
20     itoa(sendRpm,&g_sendRpmBuffer[1]);
21
22     /* Send out sendRpmBuffer to Faulhaber controller */
23     sendCommandToFaulhaber(g_sendRpmBuffer);
24
25     /* If the rpm is negative the car is driving forward.
26      * Change timer direction accordingly. */
27     if (sendRpm < 0)
28     {
29         g_drivingForward = true;
30         timer_disable_counter(TIM1);
31         TIM1_CR1 = TIM_CR1_DIR_UP;
32         timer_enable_counter(TIM1);
33     } else {
34         g_drivingForward = false;
35         timer_disable_counter(TIM1);
36         TIM1_CR1 = TIM_CR1_DIR_DOWN;
37         timer_enable_counter(TIM1);
38     }
39
40     /* Save the sent speed as current speed */
41     g_currentSpeed = speed;
42
43     /* The requested speed can now be posted as current speed */
44     (void)msgPump_postMessage(MSG_ID_CURRENT_SPEED, &speed);
```

```
45 }
46
47
48 bool speed_isDrivingForward()
49 {
50     return g_drivingForward;
51 }
52
53
54 speed_MMpS_t speed_getLastMmPerSec()
55 {
56     return g_currentSpeed;
57 }
```

4.2.4 Versenden der Befehle

Die Funktion sendCommandToFaulhaber(**char** *string) hängt der übergebenen Zeichenkette das Steuerzeichen '\r' (*carriage return*) an, womit dem Regler signalisiert wird, wann ein Befehl zu Ende ist und ausgeführt werden kann [10] (Listing 4.6). Dies geschieht hier, damit der Programmierer – bspw. beim Definieren der Befehle in der Initialisierung (Listing 4.4) – dieses Steuerzeichen nicht mit angeben muss. Schlussendlich wird die Zeichenkette an die Funktion transport_send_string(**const** transport_device_values_t bus, **const char** *string) übergeben, die die fertige Zeichenkette bzw. den fertigen Befehl Zeichen für Zeichen überträgt.

Listing 4.6: engineFaulhaber.c - Vorbereiten zum Senden

```
1  void sendCommandToFaulhaber(char *string)
2  {
3      /* Faulhaber controller expects carriage return
4       * at the end of any command.
5       * So we always add a carriage return to terminate command. */
6      int stringSize = strlen(string);
7      // Add 2, 1 for carriage return (\r) and 1 for terminating null
8      char sendString[stringSize+2];
9      int i;
10     // Copy original string to sendString buffer
11     for (i=0;i<stringSize;i++)
12     {
13         sendString[i] = string[i];
14     }
15     // Add \r and null
16     sendString[i] = '\r';
17     sendString[++i] = 0;
18
19     /* Finally send command */
20     transport_send_string(TRANSPORT_DEVICE_FAULHABER, sendString);
21 }
```

4.3 Informationen vom Regler zum Mikrocontroller

4.3.1 Initialisierung zum Erfassen der Impulse

Wie in Abschnitt 3.2.2 erläutert, handelt es sich bei den Impulsen um ein einkanaliges Signal, für das entsprechend ein Pin an einem Timer zum Zählen der Impulse konfiguriert werden muss (Listing 4.7). Zunächst muss allerdings auch hier der Systemtakt an der entsprechenden Leitung aktiviert werden. Anschließend wird der Pin (PE09) als Eingang von Timer *TIM1* an Kanal 1 (*CH1*) konfiguriert. Dabei muss zunächst der Pin auf Input gesetzt und der Kanal 1 auf diesen Pin gelegt (*mapped*) werden [20, S. 165]. Als nächstes ist der Timer so zu konfigurieren, dass er von einem externen Signal getaktet und vorher aktiviert (*triggered*) wird. Sobald der erste Impuls – auf Signalebene bedeutet das eine steigende Flanke – eintrifft, werden fortan alle Impulse bzw. steigenden Flanken gezählt [20, S. 283 ff.]. Überläufe bzw. Unterläufe des Zählregisters werden durch einen Interrupt behandelt, was die Aktivierung von Interrupts des Timers im *nested vectored interrupt controller* und zusätlich das Einschalten des entsprechenden Interrupttyps (*update interrupt*) erfordert [20, S. 315 f.]. Sobald der Timer vollständig konfiguriert ist, wird er „scharf geschaltet“ und beginnt zu zählen, sobald der erste Impuls eintrifft.

Listing 4.7: rotaryFaulhaber.c - Initialisierung zum Zählen der Impulse

```
1  static void rotary_init()
2  {
3      // Enable IO port E clock
4      rcc_peripheral_enable_clock(&RCC_APB2ENR, RCC_APB2ENR_IOPEEN);
5      // Enable alternate function clock
6      rcc_peripheral_enable_clock(&RCC_APB2ENR, RCC_APB2ENR_AFIOEN);
7      // Enable TIM1 clock
8      rcc_peripheral_enable_clock(&RCC_APB2ENR, RCC_APB2ENR_TIM1EN);
9
10     /* Configure Pin */
11     // Setup GPIO pin TIM1_CH1 on GPIO port E as PULL UP
12     gpio_set_mode(GPIOE, GPIO_MODE_INPUT,
13                   GPIO_CNF_INPUT_PULL_UPDOWN, GPIO9);
14     // Activate pull up
15     GPIOE_ODR |= GPIO9;
16     // Remap tim1 channel 1 and 3 input to PIOE9 and PIOE11
17     AFIO_MAPR |= AFIO_MAPR_TIM1_REMAP_FULL_REMAP;
18
19     /* Configure Timer */
20     // Disable preload
21     timer_disable_preload(TIM1);
22     // Enable TIM1 interrupt
23     nvic_enable_irq(NVIC_TIM1_UP_IRQ);
24     /* Set CC1 as input, mapped to TI1.
25      * libopencm3 bug... TIM_CCMR1_CC1S_IN_TI1 is defined as 0x02. */
26     TIM1_CCMR1 = 0x1;
27     // Filter the input to reduce noise
```

```
28      TIM1_CCMR1 |= TIM_CCMR1_IC1PSC_8;
29      /* Configure External Clock Mode with rising edges on TI1.
30       * TI1 also triggers Counter.*/
31      TIM1_SMCR |= TIM_SMCR_SMS_ECM1;
32      TIM1_SMCR |= TIM_SMCR_TS_IT1FP1;
33      // Enable capture on TI1
34      TIM1_CCER |= TIM_CCER_CC1E;
35      // Enable update interrupt
36      TIM1_DIER |= TIM_DIER_UIE;
37      /* Set counter to middle at beginning.
38       * We can go forawrd or backward at beginning. */
39      TIM1_CNT = TIMER_CENTER_POSITION;
40
41      /* Finally enable Counter */
42      TIM1_CR1 |= TIM_CR1_CEN;
43  }
```

4.3.2 Verhalten bei Über-/Unterlauf

Da der Timer über ein 16-Bit-Zählregister (*TIM1_CNT*) verfügt, kann es zum Über- bzw. Unterlauf kommen, d. h. die Werte 65 535 über- bzw. 0 unterschritten werden [20, S. 327 f.]. In einem solchen Fall wird durch einen Interrupt die Funktion speed_CounterOverUnderflow() aufgerufen, in der die bisher gezählten Impulse in eine 32-Bit-Variable gespeichert und das Zählregister des Timers zurückgesetzt werden (Listing 4.8). Um von der Fahrtrichtung unabhängig bis zum ersten Über-/Unterlauf die gleiche Anzahl an Impulsen zu erhalten, wird der Wert des Zählregisters bei der Initialisierung und jedem Über-/Unterlauf auf die Mittelposition von 32 768 gesetzt. Ein Über-/Unterlauf tritt daher (bei aktueller Konfiguration) immer nach $26,67\ m$ Fahrt in eine Richtung auf. Bei der aktuellen Implementierung wird das Zählregister bei Vorwärtsfahrt inkrementiert, bei Rückwärtsfahrt dekrementiert. Das hat zur Konsequenz, dass nach gleich weiter Fahrstrecke in beide Richtungen der Timer in der Ausgangsposition steht und es daher den Anschein hat, als sei keine Strecke zurückgelegt worden.

Listing 4.8: rotaryFaulhaber.c - Interruptroutine bei Über-/Unterlauf

```
1  void speed_CounterOverUnderflow()
2  {
3      timer_disable_counter(TIM1);
4      if(speed_isDrivingForward())
5      {
6          g_tickCounter += TIMER_CENTER_POSITION;
7      }
8      else
9      {
10         g_tickCounter -= TIMER_CENTER_POSITION;
11     }
```

```
12      TIM1_CNT = TIMER_CENTER_POSITION;
13      timer_enable_counter(TIM1);
14  }
```

4.3.3 Auswertung der Impulse

Letztendlich werden mit der Funktion speed_getDistanceMm() die Impulse zur Odometrie ausgewertet, aus denen sich die gefahrene Strecke (im Saldo) ergibt (Listing 4.9). Dazu werden die gezählten Impulse mit Hilfe der Funktion speed_getTicks() abgefragt und mit der Anzahl der Impulse pro Mikrometer multipliziert. Um eine Angabe in *mm* zu erhalten wird noch durch den Faktor 1000 dividiert.

Aufgerufen wird die Funktion speed_getDistanceMm() innerhalb des gleichnamigen RPC vom IGEPv2, der die Information über die zurückgelegte Strecke für die Fahrtenplanung verwendet. In Zukunft wird es auch möglich sein, diese Informationen auf Seiten des Mikrocontrollers in Kombination mit Inertialsensoren zur Selbstlokalisierung zu verwenden.

Listing 4.9: rotaryFaulhaber.c - Auswertung der gezählten Impulse

```
1  int32_t speed_getTicks()
2  {
3      tmp = g_tickCounter;
4      tmp += (TIM1_CNT-TIMER_CENTER_POSITION);
5      return tmp;
6  }
7
8
9  speed_distanceMm_t speed_getDistanceMm()
10 {
11     return (speed_distanceMm_t) ((MICROMETER_PER_TICK * speed_getTicks()) /
           1000.0f);
12 }
```

4.4 Ausblick auf weitere Möglichkeiten

Die Informationen, die den Mikrocontroller vom Regler erreichen, sind bisher (lediglich) die Impulse des integrierten Drehgebers und keine Informationen höherer Ebene über den UART. Mit der Auswertung der Impulse zu Zwecken der Odometrie lässt sich der Antrieb jedoch in allen Belangen zuverlässig regeln, so dass weitere Informationen zunächst nicht benötigt werden.

Aus den Informationen höherer Ebene, bei denen es sich um Antworten (im ASCII-Format) auf vorausgegangene Anfragen handelt, könnte insbesondere ein Überwachung des Antriebs

realisiert werden. So wäre es bspw. möglich, die aktuelle Istdrehzahl, die Iststromaufnahme oder die Temperatur des Reglers abzufragen [10]. Aus der aktuellen Stromaufnahme ließe sich der Stromverbrauch berechnen und mit Hilfe der aktuellen Drehzahl könnte man die Abweichung von der Solldrehzahl feststellen und korrigierend eingreifen. Um dies zu ermöglichen, müsste ein Modul implementiert werden, das jeder Anfrage die richtige Antwort zuordnet. Realisieren ließe sich dies bspw. durch eine Warteschlange, da die Antworten in der Reihenfolge der Anfrage eintreffen würden.

Weitere Möglichkeiten liegen in den Betriebsarten des Motors. So ließe sich im laufenden Betrieb in den Schrittmotorbetrieb wechseln sowie die Schrittweite einstellen [10]. Fahrmanöver, die eine äußerst exakte Regelung benötigen, könnten somit auch problemlos gemeistert werden. Die vorgenommenen festen Einstellungen zur Rampengenerierung oder die Anzahl der Impulse pro Umdrehung könnten je nach Fahrsituation auch dynamisch angepasst und damit optimiert werden. Um solche Funktionalitäten zu implementieren, müssten entsprechende Vorgaben vom IGEPv2 an den Mikrocontroller gesendet oder Funktionen implementiert werden, die die aktuellen Vorgaben des IGEPv2 analysieren und entsprechend handeln, bspw. die Beschleunigungsrate erhöhen oder verringeren.

Sicher ist, dass dieser Antrieb über ein großes Spektrum an Funktionen verfügt wie kaum ein anderer und daher für alle Zwecke gerüstet ist.

5 Zusammenfassung und Fazit

Das Ziel dieser Arbeit war die Bereitstellung einer Antriebsplattform durch die Integration eines elektrischen Antriebs, der allen Regelungsansprüchen für das autonome Fahren jederzeit gerecht wird. Dazu gehörte insbesondere das Fahren einer in $^{mm}/_{s}$ vorgegebenen Geschwindigkeit.

Mit der ausführlichen Berechnung der Fahrzeuglängsdynamik und den daraus resultierenden benötigten Eigenschaften des Elektromotors konnte für die gestellten Anforderungen an die Längsdynamik des Modellfahrzeugs ein sehr adäquater Elektromotor des Herstellers FAULHABER ausgewählt werden. Wichtig für die exakte Regelung dieses Antriebs ist der leistungsstarke Regler (ebenfalls vom Hersteller FAULHABER), der es ermöglicht, den Motor durch die konkrete Vorgabe einer Drehzahl zu regeln. Die Vorgaben werden in Form von ASCII-Befehlen vom Mikrocontroller über eine RS-232 Schnittstelle an den Regler gesendet. Darüber hinaus dient der Regler als neuer zentraler Drehgeber der Odometrie, indem der Fehlerausgang als Impulsausgang genutzt wird und die gelieferten Impulse vom Mikrocontroller ausgewertet werden.

Dank der umfassenden Datenblätter und Handbücher stellten der Einbau und die Verschaltung der Komponenten keine großen Probleme dar. Auch die Konfiguration war insbesondere durch die mitgelieferte Software MOTION MANAGER ein Leichtes. Wichtige Parameter waren bei der Konfiguration die Baudrate für die Kommunikation, die Betriebsart des Motors und die Verwendung des Fehlerausgangs als Impulsausgang für den Drehgeber.

Die Implementierung seitens des Mikrocontrollers beschränkte sich darauf, die Kommunikation über eine RS-232 Schnittstelle und die Auswertung der Impulse zu implementieren. Dafür war lediglich die Konfiguration einer UART und eines Timers nötig. Das Verfahren, den Antrieb über ASCII-Befehle zu regeln, erwies sich als sehr schlank und fehlerunanfällig. Ebenso stellte sich die Auswertung der Impulse dar, die für jeden eintreffenden Impuls das Zählregister des Timers (je nach Fahrtrichtung) inkrementieren oder dekrementieren und somit gezählt werden, so dass sich die zurückgelegte Strecke leicht errechnen lässt.

Die Ergebnisse der Integration des neuen elektrischen Antriebs stellten sich bei Tests als beeindruckend dar und erfüllen alle Erwartungen. Aufgrund der hohen Qualität des Elektromotors und des Reglers ist es nun möglich, jede beliebige Geschwindigkeit (bis zur definierten Höchstgeschwindigkeit von 4 $^{m}/_{s}$) zu erreichen und konstant zu halten. Auch die Odometrie erwies sich als sehr zuverlässig und exakt. Für die Fahrmanöver – insbesondere für das Einparken am Seitenstreifen – stehen nun keine Hindernisse in Form mangelnder Regelbarkeit des elektrischen Antriebs mehr im Weg.

Durch die starke Abkapselung des gewünschten Verhaltens durch Befehle von der tatsächlichen Regelung steht auch für eventuelle neue Fahrmanöver eine zuverlässige Antriebsplattform zur Verfügung. Mit der zentralen Odometrie wird es in Zukunft unter Zuhilfenahme von Inertialsensoren möglich sein, aus der gefahrenen Strecke eine Karte zu erstellen und diese dann für die Navigation zu nutzen. Auch Regler und Elektromotor selbst weisen noch ein großes Potential an bislang ungenutzten, für zukünftige Aufgaben jedoch möglicherweise sinnvollen Funktionen auf, wie etwa die Auswertung verschiedener Parameter des Reglers (z.B. aktueller Iststrom zur Aufzeichnung des Stromverbrauchs) und die Möglichkeit des Schrittmotorbetriebs.

Als Fazit bleibt festzustellen, dass dieser elektrische Antrieb auf lange Sicht alle diesbezüglichen Anforderungen an ein autonomes Modellfahrzeug erfüllen können wird, ohne dafür eigene, mitunter fehleranfällige Leistungselektronik implementieren zu müssen.

Abbildungsverzeichnis

1.1 MadeInGermany von AutoNOMOS 2
1.2 Erstes Modellfahrzeug des Berlin United Racing Teams 3
1.3 Verwendeter Mikrocontroller von Olimex 4
1.4 Logo des Berlin United Racing Teams 5

2.1 Komponenten des elektrischen Antriebs 7
2.2 Regler als Komponente des el. Antriebs im Modellbau 8
2.3 Schema eines bürstenlosen Gleichstrommotors 10
2.4 Skizze des Fahrzeugquerschnitt 16

3.1 Eingebauter Motor und Regler 24
3.2 Motorboard zur Antriebsentwicklung 25
3.3 Schematisches Schaltbild des MAX3232 26

Tabellenverzeichnis

2.1 Eigenschaften und Anforderungen an den Antrieb 11
2.2 Gegenüberstellung der berechneten Eigenschaften und der Auswahl 20

3.1 Aufstellung der wichtigsten Parameter des Reglers 29

Quellcodeverzeichnis

4.1 **speed.h** - Interface zur Antriebsregelung und -auswertung 32
4.2 **USART.c** - Konfiguration des UART 33
4.3 **faulhaber.h** - Definition der Konstanten 34
4.4 **engineFaulhaber.c** - Initialisierung des Reglers 35
4.5 **engineFaulhaber.c** - Drehzahlregelung 36
4.6 **engineFaulhaber.c** - Vorbereiten zum Senden 37
4.7 **rotaryFaulhaber.c** - Initialisierung zum Zählen der Impulse 38
4.8 **rotaryFaulhaber.c** - Interruptroutine bei Über-/Unterlauf 39
4.9 **rotaryFaulhaber.c** - Auswertung der gezählten Impulse 40

Literaturverzeichnis

[1] *Carolinchen IV / π*. http://www.team-cdlc.de/?q=node/46, Abruf: 16.11.2012. – Fahrzeugbeschreibung des Team CDLC — Technische Universität Braunschweig

[2] *Computers and sensors aid car in autonomous Berlin drive*. http://www.theengineer.co.uk/sectors/automotive/news/computers-and-sensors-aid-car-in-autonomous-berlin-drive/1010234.article, Abruf: 16.11.2012

[3] *DC-Motoren als Leistungssportler*. http://www.faulhaber.com/sprache2/n218514/n.html, Abruf: 16.11.2012

[4] *Frequently Asked Questions about the AutoNOMOS Project*. http://autonomos-labs.de/faq, Abruf: 16.11.2012

[5] *„Carolo-Cup“ Regelwerk 2012*. 17.06.2011

[6] Audi AG: *Technische Daten Audi TT Coupé 2.0 TFSI*. Juli 2006

[7] Brinkschulte, Uwe ; Ungerer, Theo: *Mikrocontroller und Mikroprozessoren*. 2. überarb. Aufl. Berlin : Springer Verlag, 2007. – ISBN 978–3–540–4680–1

[8] Dr. Fritz Faulhaber GmbH & Co. KG: *Bürstenlose DC-Servomotoren Serie 3056 ... B*. Edition 2012–2013

[9] Dr. Fritz Faulhaber GmbH & Co. KG: *Motion Controller V2.5, 4-Quadranten PWM mit RS232 oder CAN-Schnittstelle*. Edition 2012–2013

[10] Dr. Fritz Faulhaber GmbH & Co. KG: *Motion Control Serie MCBL 300x RS [...] Kommunikations- / Funktionshandbuch*. 2. Version. Schönaich, Dezember 2011

[11] Dr. Fritz Faulhaber GmbH & Co. KG: *Motion Controller Serie MCBL 3002 / 03 / 06 [...] Gerätehandbuch*. 1. Aufl. Schönaich, November 2011

[12] Dr. Fritz Faulhaber GmbH & Co. KG: *PC-Software FAULHABER Motion Manager 4 Bedienungsanleitung*. 6. Aufl. Softwarestand 4.6. Schönaich, 2011

[13] Dr. Fritz Faulhaber GmbH & Co. KG: *Technische Informationen*. 4. Aufl. Schönaich, 2012–2013

[14] Garbrecht, Friedrich W.: *Auswahl von Elektromotoren - leicht gemacht - Der Weg von der Anwendungsanalyse zum richtig dimensionierten Elektromotor*. 1. Aufl. Berlin, Offenbach : VDE Verlag, 2008. – ISBN 978–3–800–72911–1

[15] Maxim Integrated Products Inc.: *MAX3222/MAX3232/MAX3237/MAX3241 3.0V to 5.5V, Low-Power, up to 1Mbps, True RS-232 Transceivers Using Four 0.1µF External Capacitors*. Rev. 7, 2007

[16] Nowacki, Filip: *Konzeptentwicklung eines Sicherheits- und Navigationsmoduls basierend auf einer geometrischen Umgebungskartierung für autonom fahrende Systeme*. – FAUST-Projekt — Hochschule für Angewandte Wissenschaften Hamburg

[17] Olimex Ltd.: *STM32-H107 development board Users Manual*. Rev. C, June 2011

[18] Reif, Konrad ; Dietsche, Karl-Heinz ; GmbH, Robert B.: *Kraftfahrtechnisches Taschenbuch*. 27., überarb. und erw. Aufl. Wiesbaden : Vieweg+Teubner Verlag, 2011. – ISBN 978–3–834–81440–1

[19] Schürmann, Bernd: *Rechnerverbindungsstrukturen - Bussysteme und Netzwerke*. 1. Aufl. Wiesbaden : Vieweg Verlag, 1997. – ISBN 978–3–528–05562–2

[20] STMicroelectronics: *RM0008 Reference manual – [...] STM32F107xx advanced ARM-based 32-bit MCUs*. Rev. 11, 2010

[21] STMicroelectronics: *STM32F107xx*. Rev. 6, 2011

[22] Stölting, Hans-Dieter ; Kallenbach, Eberhard: *Handbuch Elektrische Kleinantriebe*. 3., neu bearb. und erw. Aufl. München, Wien : Hanser Verlag, 2006. – ISBN 978–3–446–40019–1

[23] Timoschenko, Alexander: *Implementierung einer Geschwindigkeitsregelung als Prozessor-Element auf einer SoC-Plattform für ein autonomes Fahrzeug*. November 2011. – FAUST-Projekt — Hochschule für Angewandte Wissenschaften Hamburg

[24] Wikipedia: *Lithium-Polymer-Akkumulator*. http://de.wikipedia.org/wiki/Lithium-Polymer-Akkumulator, Abruf: 16.11.2012

[25] Wikipedia: *Strömungswiderstandskoeffizient*. http://de.wikipedia.org/wiki/Str%C3%B6mungswiderstandskoeffizient, Abruf: 16.11.2012