Bachelorarbeit

Christian Möller

Sensorik und Regelung zur Lenkung autonomer Fahrzeuge

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Christian Möller

Sensorik und Regelung zur Lenkung autonomer Fahrzeuge

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. rer. nat. Stephan Pareigis
Zweitgutachter: Prof. Dr.-Ing. Franz Korf

Abgegeben am 13. Dezember 2011

Christian Möller

Thema der Bachelorarbeit

Sensorik und Regelung zur Lenkung autonomer Fahrzeuge

Stichworte

Eingebettete Systeme, Planung, Aktor, Sensor

Kurzzusammenfassung

In dieser Arbeit wird ein Lenkwinkelgeber und -regler entwickelt, welcher auf autonomen Fahrzeugen zum Einsatz kommt. Es wird ein geeigneter Sensor ermittelt und in ein bestehendes System eingefügt. Dies umfasst die Entwicklung der Mechanik, der Elektronik und der Software.

Title of the paper

Steering sensor system and controller for autonomous vehicle

Keywords

Embedded Systems, System Design, Actor, Sensor

Abstract

The following study develops a steering-angle sensor and controller being deployed on autonomous vehicles. A suitable sensor will be determined and integrated into an already existing system. Thus, the study comprises the development of the mechanics, the electronical system and the software.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Im Rahmen des Projektes „FAUST - Fahrerlose autonome Systeme“ an der HAW Hamburg werden Fahrzeuge entwickelt, welche sich in verschieden Szenarien selbstständig steuern. Dazu gehört das Folgen einer Fahrspur, das Ausweichen an Hindernissen und das Einparken. Die entwickelten Fahrzeuge treten dabei regelmäßig in einem bundesweitem Wettkampf, dem CaroloCup, an. Hier müssen die Fahrzeuge die genannten Szenarien in möglichst kurzer Zeit erfüllen oder eine möglichst lange Strecke in einer vorgegebenen Zeit zurücklegen.

In der Vergangenheit war es nicht möglich den Ist-Zustand der Lenkung zeitnah zu ermitteln, da die Fahrzeuge nicht über die nötige Sensorik verfügen.
Eine Auswertung des Lenkeinschlag kann nur über die Kamerabilder erfolgen. Legt das Fahrzeug eine bestimmte Strecke zurück, kann die Differenz des erreichten und des berechneten Zielpunktes aus den Bildern ermittelt werden. Je kleiner die Differenz ist, desto genauer wurde der Soll-Wert erreicht. Dabei muss die Strecke ausreichend lang sein, damit eine signifikante Änderung in den Bildern erkennbar ist. Daraus resultiert, dass der Ist-Wert mit einer starken zeitlichen Verzögerung gemessen wird.
Ein weiteres Problem entsteht in Situationen, in denen keine ausreichend lange Strecke zum Messen zurückgelegt werden kann. Dazu gehört zum Beispiel das Einparken. In diesen Situationen kann lediglich darauf vertraut werden, dass die Lenkung den gewünschten Soll-Wert erreicht.

Die vorliegende Arbeit soll ein System liefern, mit dem der Ist-Zustand der Lenkung zeitnah ermittelt werden kann.

1.2 Zielsetzung

Ziel dieser Arbeit ist die Entwicklung eines Lenkwinkelgebers und eines Lenkwinkelreglers.
Es werden die elektronischen, sowie mechanischen Komponenten entwickelt und angefertigt. Softwareseitig wird die Sensorauswertung und Datenübertragung zum PC für den Lenkwinkelgeber implementiert.
Des Weiteren ist eine Lenkwinkelregelung zu implementieren. Diese soll sicherstellen, dass der vom PC vorgegebene Lenkeinschlag erreicht beziehungsweise gehalten wird.

1.3 Gliederung

Die Arbeit umfasst folgende Teile:

Kapitel Hardware behandelt den mechanischen und elektronischen Aufbau des Lenkwinkelgebers. Es setzt sich aus der Analyse des Fahrzeuges und der Herstellung des Lenkwinkelsensors zusammen.

Kapitel Software behandelt den Softwareteil des Lenkwickelgebers/-reglers. Es setzt sich aus der Analyse der bestehenden und der Implementation der neuen und der geänderten Software zusammen.

Kapitel Fazit beschreibt den Endzustand der Arbeit.

Kapitel 2

Hardware

Dieses Kapitel beschreibt die Entwicklung der Hardwarekomponenten.
Dazu wird zuerst eine Analyse der Ausgangsbasis vorgenommen. Darauf aufbauend wird die Wahl eines Sensortyps erklärt und anschließend die Implementation erläutert. Die Implementation unterteilt sich in einen mechanischen und einen elektronischen Part.

2.1 Analyse der Ausgangsbasis

Dieser Abschnitt befasst sich mit der Analyse des Fahrzeuges, auf dem die Entwicklung erfolgt.

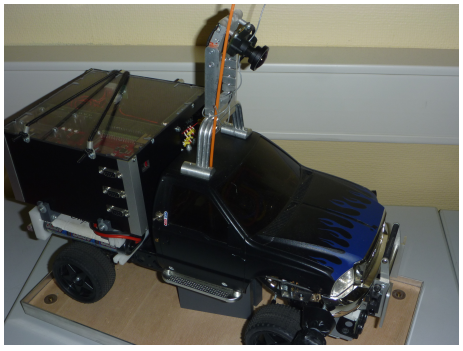

Abbildung 2.1: Fahrzeugplattform

2.1.1 Fahrzeuganalyse

Als Basis dient ein Ford F350 Modellfahrzeug der Firma Tamiya. Das Fahrzeug ist für den autonomen Betrieb stark modifiziert worden. So wurden Sensoren zur Ermittlung der Geschwindigkeit, sowie Sensoren zur Ermittlung der Abstände zu umgebenden Hindernissen verbaut.
Bei den Geschwindigkeitssensoren handelt es sich um Inkrementalgeber, welche in den Vorderrädern untergebracht sind.
Zur Abstandsmessung kommen sowohl Ultraschallsensoren, als auch Infrarotsensoren zum Einsatz.

Zusätzlich ist eine Kamera zur Fahrbahnerkennung montiert und eine komplette Beleuchtungsanlage in das Fahrzeug integriert.
Für die Verarbeitung der Sensordaten, die Ansteuerung der Aktoren und die Kopplung zwischen PC und Fahrzeug wurde eine IO-Box *(Input-Output-Box)* angefertigt.
Die Aktorik des Fahrzeuges setzt sich aus dem Lenkservo, der Fahrregler-Motor-Einheit und der Beleuchtungsanlage zusammen.
Als PC kommt ein »ASUS eeePC« zum Einsatz. Auf Diesem erfolgt die Bildverarbeitung und die Berechnung des Soll-Lenkeinschlages, sowie die Berechnung der Soll-Geschwindigkeit. Dies ist aber nicht Gegenstand der Arbeit und wird daher nur der Vollständigkeit halber erwähnt. Näheres dazu kann der Arbeit „ Systemidentifikation eines autonomen Fahrzeugs mit einer robusten, kamerabasierten Fahrspurerkennung in Echtzeit“ von Eike Jenning (Jenning, 2008) entnommen werden.

2.1.2 Analyse der Mechanik

Der Maßstab des Fahrzeuges beträgt 1:10. Neben diversen Trägern wie zum Beispiel für die Kamera oder die IO-Box sowie den Rechner, welche angefertigt wurden, sind auch massive Änderungen am Fahrwerk vorgenommen worden. Speziell im Bereich der Lenkung erfuhr das Fahrzeug große Anpassungen. So wurde der Lenkservo direkt auf die Vorderachse montiert. Im Orginal sitzt dieser am Fahrzeugrahmen und die Lenkung wird über ein umfangreiches Gestänge bewegt. Durch die Änderung der Montageposition entfällt ein Großteil dieses Gestänges, wodurch das Spiel in der Lenkung stark verringert wird. Des Weiteren kommen Präzisionskugelköpfe zum Einsatz. Dies führt zu einer weiteren Verringerung des Lenkspiels. Nachteilig ist dabei, dass durch die Komprimierung der Mechanik der Platz zur Aufnahme eines Sensors stark eingeschränkt ist. Daraus ergibt sich die erste Forderung an den Lenkwinkelsensor. Der Sensor muss verhältnismäßig klein gehalten werden, um im Bereich der Lenkung untergebracht zu werden. Zusätzlich ergibt sich aus dem geringen zur Verfügung stehenden Platz die Tatsache, dass der Lenkwinkel nicht an den Rädern gemessen werden kann. Dies wäre nur unter Verwendung präzise gefertigter filigraner Teile möglich. Dadurch erhöht sich aber die Ausfallgefahr des Sensors.

2.1.3 Analyse der IO-Box

In der IO-Box sind zwei »Hitex LPC-Sticks« verbaut. Es handelt sich dabei um Entwicklungboards mit je einem »NXP 2468 ARM7« Mikrocontroller. Diese sind über je ein »Hitex COM-Board« an ein eigens entwickeltes Peripherieboard angeschlossen. Das Peripherieboard stellt dabei die Verbindung zum Fahrzeug über mehrere Steckanschlüsse bereit. Bei der Planung des Peripherieboards wurde von einem begrenzten Umfang der Sensoren

und Aktoren ausgegangen. Daher werden lediglich die analogen bzw. digitalen Ausgänge der erwähnten Sensoren und Aktoren herausgeführt. Zusätzlich wird der I2C- und der CAN-Bus am Peripherieboard bereitgestellt. Somit ergibt sich die zweite Anforderung an den Lenkwinkelsensor. Da z.B. der Anschluss eines weiteren analogen Sensors die Entwicklung und Herstellung einer neue Platine erfordern würde, bleibt nur die Möglichkeit den Sensor über eines der beiden Bussysteme zu betreiben.

2.1.4 Analyse des Fahrzeugeinsatzes

Letztlich ist noch zu beachten, dass das Fahrzeug im Projekt „Fahrerlose autonome Syteme„, kurz FAUST, genutzt wird. An diesem Projekt arbeiten die Studierenden lediglich für ein Semester mit. Es ist daher darauf zu achten, dass die Technik verständlich und einfach zu warten bzw. zu reparieren ist. Wünschenswert ist zudem, dass die Kalibrierung der Messschaltung automatisch erfolgt. Dies verringert den Wartungsaufwand und die Fehlbedienung entscheidend.

Da das Fahrzeug über Akkus betrieben wird, ist darauf zu achten, dass der Sensor zur Lenkwinkelmessung über eine geringe elektrische Leistungsaufnahme verfügt.

Zusätzlich ist zu beachten, dass das Fahrzeug im Wetkampf eingesetzt wird. Somit dürfen die Änderungen am System nicht zu einer massiven Verlangsamung des Fahrzeuges führen.

2.1.5 Zusammenfassung der Hauptanforderungen

1. Der Sensor muss eine geringe Einbaugröße aufweisen.
2. Die Kommunikation zwischen dem Sensor und der IO-Box muss über den CAN- oder I2C-Bus erfolgen.
3. Der Sensor muss so einfach wie möglich aufgebaut sein.
4. Die Leistungsaufnahme muss gering gehalten werden.
5. Die maximale Geschwindigkeit des Fahrzeuges darf nicht massiv eingeschränkt werden.

2.2 Wahl eines Sensortyps

In diesem Abschnitt werden verschieden Sensortypen betrachtet und auf Basis der Hauptanforderungen beurteilt. Die Sensoren werden dabei in die Art ihrer Messverfahren unterteilt.

2.2.1 Widerstandsmessung

Bei der Winkelmessung per Widerstandsmessung wird ein Potentiometer als Sensor verwendet. Versehen mit einer Spannungsteilerschaltung kann ein einfaches Messsystem aufgebaut werden. Durch die Veränderung des Widerstandswertes ändert sich analog dazu auch die Spannung, welche leicht zu messen ist. Jeder Spannung kann somit ein eindeutiger Winkelwert zugeordnet werden.

Abbildung 2.2: Potentiometer

Vorteile

- Das Prinzip des Spannungsteilers ist leicht verständlich und erfordert keinen besonderen Einarbeitungsaufwand.
- Der benötigte Bauteilaufwand ist sehr gering und es können schnell verfügbare Standardkomponenten verwendet werden. Somit ist der Sensor im Bedarfsfall leicht zu reparieren bzw. zu warten. Weiterhin wirkt sich die Verwendung von Standardkomponenten positiv auf den Preis aus.
- Zur Anbindung an die IO-Box käme hier ein Analog-Digital-Wandler in Frage. Diese sind mit den entsprechenden Bussystemen erhältlich und leicht in die Schaltung zu integrieren.

Nachteile

- Eine Recherche hat ergeben, dass ein komplettes Sensorsystem nicht kommerziell verfügbar ist und somit selbst hergestellt werden müsste.

- Der Sensor ist nicht wartungsfrei, da ein Potentiometer einem mechanische Verschleiß unterliegt. Es kann z.B. passieren, dass sich das Potentiometer in einem kleinen Intervallbereich durch Schwingungen zu stark abnutzt. Dies würde zu sprunghaften Änderungen in der Spannungskurve führen. Es ist dadurch keine verlässliche Messung mehr möglich, da mehrere Winkel dieselbe Spannung erzeugen würden.

2.2.2 Opto-Elektronische Messung

Bei der opto-elektronischen Messung kämen zwei Verfahren in Frage. Zum einen die Messung über Kodierscheiben und zum anderen die Messung über Taktgeberscheiben. Beide Verfahren sind Kontaktfrei und besitzen somit keinen nennenswerten mechanischen Verschleiß.

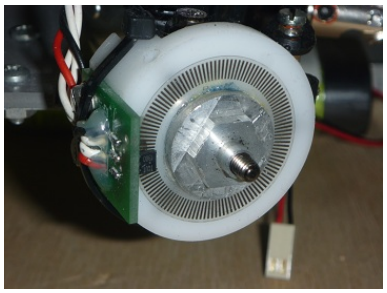

Abbildung 2.3: Inkrementalgeber

2.2.2.1 Messung per Kodierscheibe

Bei der Messung per Kodierscheibe wird das
Prinzip der Lochscheibe bzw. -karte angewendet. Dabei wird für jede Lochstreifenspalte eine Lichtschranke verwendet. Die geöffneten bzw. geschlossenen Schranken ergeben den Code, welcher einem Lenkwinkel zugeordnet werden kann.

Vorteile

- Der Sensor kann fertig bei diversen Herstellern bezogen werden.
- Das Ausgangssignal liegt ohne Wandlung in digitaler Form vor.
- Der Sensor ist wartungsfrei, da die Messung kontaktlos erfolgt.

Nachteile

- Dieser Sensortyp ist, in Bezug auf das Modellfahrzeug, nur in verhältnismäßig großen Abmessungen zu erwerben.
- Kommerzielle Varianten sind, im Vergleich zum Preis einer Lösung per Widerstandsmessung, sehr teuer.

- Eine Eigenentwicklung setzt spezielle Maschinen und Verfahren zur Herstellung voraus, um die nötige Präzision zu erreichen. Dies ist zwar möglich, aber die an der Hochschule verfügbaren Maschinen lassen nur eine relativ große Bauteilabmessung zu. Dies wirkt sich negativ auf die Gesamtgröße aus.

2.2.2.2 Messung per Taktgeberscheibe

Bei der Messung per Taktgeberscheibe wird eine einzelne Lichtschranke durch eine Streifen- oder Punktmaske unterbrochen. Jede Unterbrechung erzeugt einen zählbaren Impuls. Dieses Verfahren kommt bei den erwähnten Inkrementalgebern zur Geschwindigkeitsmessung zum Einsatz.

Vorteile

- Der Sensor kann kostengünstig bei diversen Herstellern bezogen werden.
- Der Sensor ist wartungsfrei, da die Messung kontaktlos erfolgt.
- Bei einer Eigenentwicklung ist ein einfaches Sensorsystem in einem kleinem Maßstab herstellbar. Die Taktscheibe könnte z.B. direkt auf dem »Ruderhorn« des Servos montiert werden.
- Zur Ermittlung der Winkelstellung könnte ein Zähler-IC mit Businterface zum Einsatz kommen. Eine Schaltung ist mit geringem Bauteilaufwand und unter Verwendung von Standardkomponenten leicht zu realisieren.

Nachteile

- Fertige geschirmte Lösungen fallen, im Bezug auf das Modellfahrzeug, verhältnismassig groß aus.
- Die Vergangenheit hat gezeigt, dass die Geber ohne ausreichende Abschirmung gegen einfallendes Fremdlicht sehr störanfällig sind. Zusätzlich birgt eine offene Bauform die Gefahr der Verschmutzung. Bei zunehmender Verschmutzung wird die Messung verfälscht bzw. kann im ungünstigsten Fall ausfallen. Somit entsteht ein hoher Wartungsaufwand, der wiederum durch die kleine Bauform erschwert wird. Eine Abschirmung der Messvorrichtung vergrößert die Bauform aber wieder stark und ist ohne industrielle Fertigungsverfahren, wie z.B. CNC-Fräsen, schlecht herzustellen.

2.2.3 Induktive Messung

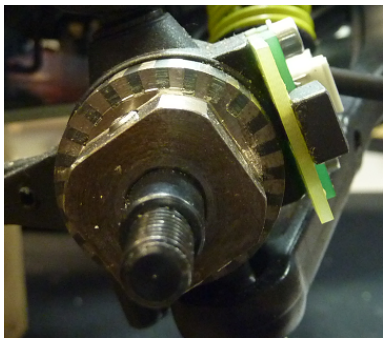

Abbildung 2.4: Hallsensor

Bei der induktiven Messung wird die Lage eines Magnetfeldes ausgewertet. Dabei wird der Hall-Effekt ausgenutzt. Bei der Messung wird durch ein „Zahnrad“ das Magnetfeld des Sensors geändert. Jeder Zahn erzeugt dabei einen zählbaren Impuls. Dieses Verfahren kommt bei der Geschwindigkeitsmessung in den intelliTrucks des FAUST-Projektes zum Einsatz. Eine Erläuterung der Induktiven Messung kann der Diplomarbeit „Design und Implementation eines Konzeptes zur Geschwindigkeitsbestimmung eines autonomen Fahrzeugs unter Verwendung eines AVR-Mikrokontrollers in Kombination mit Hall-Sensorik“ von Bachir Blal (Blal, 2007) entnommen werden.

Vorteile

- Der Sensor kann fertig bei diversen Herstellern bezogen werden.
- Der Sensor ist wartungsfrei, da die Messung kontaktlos erfolgt.
- Bei einer Eigenentwicklung wäre der Sensor mit elektronischen Standardkomponenten herstellbar. Es würden lediglich ein fertiger Hall-Geber-IC und eine Zähler-IC mit Businterface benötigt.

Nachteile

- Der Sensor besitzt eine, im Bezug auf das Modellfahrzeug, verhältnismassig große Bauform.
- Bei einer Eigenentwicklung wäre die Mechanik in der benötigten Präzision nur mit NC-Maschinen bzw. im Erodierverfahren herstellbar.
- Bei dieser Messart kann die Bewegungsrichtung nicht festgestellt werden. Diese müsste auf andere Weise ermittelt werden.

2.2.4 Entscheidung

Bei der Betrachtung der Vor- und Nachteile zeigt sich, dass Lösungen per optoelektronischer und induktiver Messung fertig aufgebaut bezogen werden können und zudem wartungsfrei sind. Leider sind all diese Lösungen, egal ob kommerziell oder eigenentwickelt, zu groß um sie in der Nähe der Lenkung unterzubringen. Dadurch würde ein Steuergestänge nötig um eine Verbindung zwischen Lenkservo und Sensor herzustellen. Hierdruch würde aber wieder Spiel in der Messeinheit entstehen, wodurch die Messung verfälscht wäre. Aus diesem Grund fiel die Wahl auf das Widerstandsmessverfahren. Es stellt die einzige Möglichkeit dar, einen kostengünstigen, leicht herzustellenden und kleinen Sensor zu entwickeln.

2.3 Implementation der Mechanik

Wie unter 2.1.2 „Analyse der Mechanik“ erwähnt, ist im Bereich der Vorderachse nur begrenzt Platz vorhanden. Im Bereich der Räder ist so wenig Platz, dass dort kein Sensor befestigt werden kann. Deshalb kommt nur eine Messung des Winkels am Lenkservo in Frage. Um das Potentiometer in der Nähe des Lenkservos zu befestigen, wurde ein geänderter Halter angefertigt. In diesem werden der Lenkservo und das Potientiometer dicht beieinander befestigt.

Am Lenkservo wurde das einarmige Ruderhorn gegen ein Doppelarmiges getauscht. Am unteren Arm wird das Lenkgestänge befestigt. Der obere Arm dient der Ansteuerung des Potentiometers. Auf der Potentiometerseite befindet sich ein modifiziertes einarmiges Ruderhorn. Die Arme auf der Seite des Servos, sowie der Seite des Potentiometers weisen die gleiche Länge auf. Somit muss kein Übersetzungsverhältnis beachtet werden. Die Verbindung beider Ruderhörner erfolgt über ein Gestänge mit Präzisionskugelköpfen. Dadurch ist die komplette Messmechanik spielfrei ausgeführt.

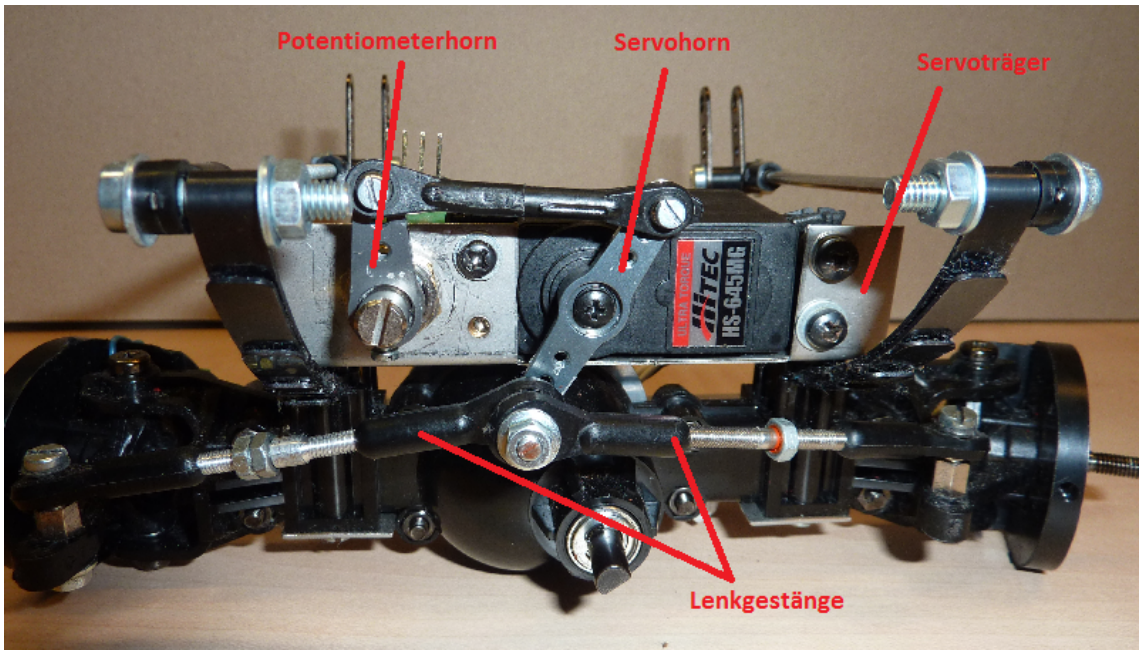

Abbildung 2.5: Komplett aufgebaute Vorderachse inkl. Sensor

Alle mechanischen Arbeiten wurden durch den Autor in der HAW-eigenen Werkstatt ausgeführt.

2.4 Test der Mechanik

Abschliessend erfolgt der Test der Mechanik. Dafür wurde eine Gradscheibe auf dem Lenkservohalter befestigt. Zur Erzeugung der PWM-Signale für den Lenkservo kommt ein Kontron 8021 Frequenzgenerator zum Einsatz. Ausgehend von der Mittelstellung der Lenkung wird der Servo so verstellt, dass die Achsschenkel ihre Anschlagspunkte erreichen.

Aus dem mechanischen Aufbau der Vorderachse ergibt sich, dass die Lenkung in einem Bereich von -30 bis +30 Grad arbeitet. Dies bedeutet, dass das Potentimeter nie seine Endanschläge erreicht.

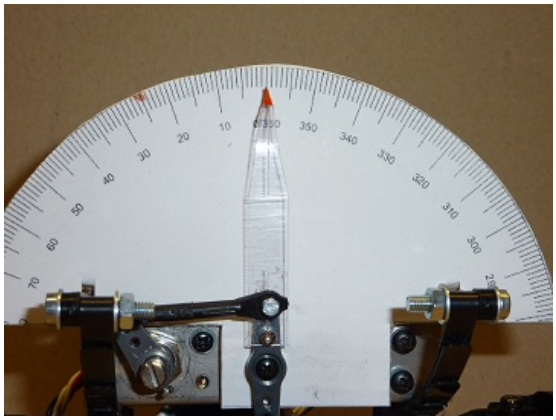

Abbildung 2.6: Lenkung in Mittelstellung

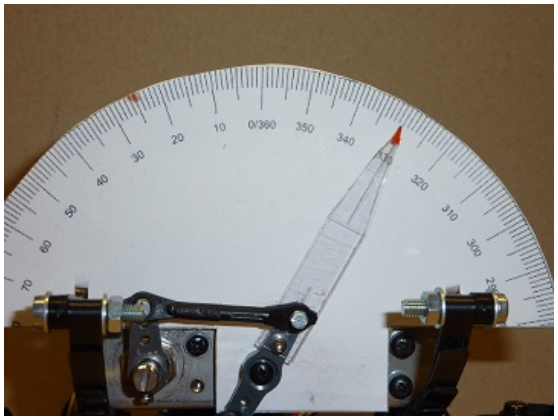

Abbildung 2.7: Lenkung in Vollauschlag

Des Weiteren zeigt die Messungen, dass eine Abstufung der Ansteuerung im Gradbereich ausreichend genau ist. Eine Verstellung der Lenkung im Minutenbereich birgt keinen erkennbaren Vorteil. Aus diesem Grund erfolgen alle weiteren Berechnungen sowie Implementationen auf einer gradgenauen Einteilung.

2.5 Implementation der Elektronik

Dieser Abschnitt teilt sich in das Design, die Entwicklung und den Test der Elektronik. Dabei wird im Bereich „Design und Entwicklung“ die Grundlage des Sensors erläutert. Des Weiteren werden die Simulation und die Berechnungen erläutert. Der Bereich „Implementation“ beschreibt den realen Aufbau der Elektronik. Es wird auf die verwendeten Komponenten eingegangen und das Layout der Platine beschrieben. Zum Schluss wird erläutert, wie die Tests der endgültigen Elektronik erfolgen und welche Ergebnisse dies liefert.

2.5.1 Design und Entwicklung

Für die Implementation des Sensors wird das Prinzip des belasteten Spannungsteilers verwendet. Ein belasteter Spannungteiler Abb. 2.8 besteht aus einem unbelastet Spannungsteiler und einem Lastwiderstand. Ein unbelasteter Spannungsteiler setzt sich aus zwei in Reihe geschalteten Widerständen zusammen. Es gelten somit die Regeln und Formeln der Reihenschaltung von Widerständen. Eine tiefergehende Behandlung kann (Altmann und Schlayer, 2001) entnommen werden. Um einen belasteten Spannungteiler

zu erhalten wird zu einem der beiden Widerstände ein Lastwiderstand RL parallel geschaltet.

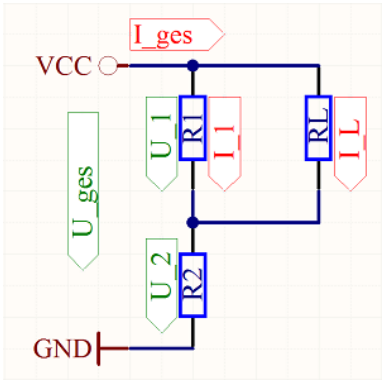

Abbildung 2.8: Spannungsteiler belastet

Wird nun statt der Widerstände R1 und R2 ein Potentiometer verwendet, erhält man einen Spannungsregler wie in Abbildung 2.9 dargestellt. Wird weiterhin statt des Lastwiderstand RL ein Analog-Digital-Converter, kurz ADC, verwendet ergibt sich daraus eine einfache Winkelmessschaltung.

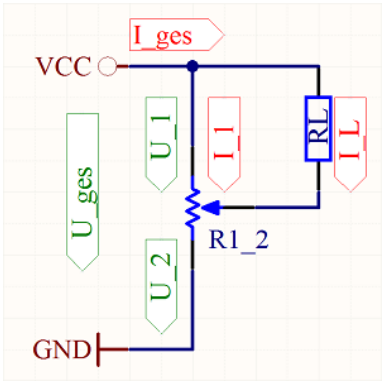

Abbildung 2.9: Schaltung mit Potentiometer

Im vorliegenden Fall kommt hierfür ein ADS1000 von Texas Instruments zum Einsatz. Dieser ADC verfügt über eine Auflösung von 12 Bit und ein I2C-Businterface. Der Widerstandswert beträgt 2,4MOhm und die Messung erfolgt bei diesem Typ differenziell. Dies bedeutet, dass die Spannung zwischen dem Messeingang Vin+ und dem Messausgang Vin- gemessen wird. Das Gegenteil dazu wäre die potenzielle Messung, bei der die

Spannung gegen Nullpotential gemessen wird. Nur durch den differenziellen Aufbau kann der ADC als Lastwiderstand verwendet werden.
Zu beachten ist, dass das Verhältnis RL:R1 klein gehalten wird, da die Kennlinie des Spannungsteilers im ungünstigsten Fall sonst nicht linear verläuft. Dies ist entscheidend, um in der Softwareimplementation unnötige Berechnungen oder Spannungstabellen zu vermeiden. Um das passende Verhältnis zu ermitteln wird die Schaltung unter PSpice simuliert und der Widerstandswert des Potentiometers empirisch ermittelt.

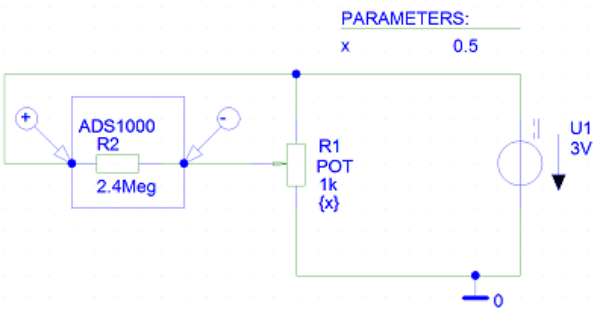

Abbildung 2.10: Simulationsschaltplan

Unter 5.2 „Simulation der Schaltung unter PSpice“ ist das zugehörige Simulationsergebnis abgebildet.
Es hat sich gezeigt, dass ein Widerstandswert von 1kOhm eine linearen Spannungsverlauf ergibt. Hier kommt ein Potentiometer Typ 148 der Firma Vishay zum Einsatz.

Wie unter 2.1.4 „Analyse des Fahrzeugeinsatzes“ erwähnt wird das Fahrzeug über Akkus versorgt. Dadurch entsteht das Problem der schwankenden Spannungsversorgung. Um die Messungen möglichst genau auszuführen wird die Messschaltung über eine Spannungsreferenz versorgt. Dies gewährleistet eine gleichbleibende Versorgungsspannung über einen großen Bereich. Hier kommt der REF3330 von Texas Instruments zum Einsatz. Diese Referenzquelle liefert 3V und bis zu 180 mA Strom.

Auf Basis der ermittelten Werte wird im folgenden die Berechnung der Widerstandswerte, der Ströme und der Leistung der Messschaltung durchgeführt.
Dabei bezeichnet der $Index_p$ die Potentiometerstellung, in der die Widerstände parallel liegen. Analog dazu bezeichnet $Index_r$ die Potentiometerstellung, in der die Widerstände in Reihe geschaltet sind. R_L bezeichnet den differenziellen Widerstand des ADC und R_{12} den Widerstand des Potentiometers.

Widerstand

$$\begin{aligned} R_{g_p} &= \frac{R_L * R_{12}}{R_L + R_{12}} \\ &= \frac{2,4M\Omega * 1k\Omega}{2,4M\Omega + 1k\Omega} \\ &\approx 999,58\Omega \end{aligned} \tag{2.1}$$

$$\begin{aligned} R_{g_r} &= R_L + R_{12} \\ &= 2,4M\Omega + 1k\Omega \\ &= 2401k\Omega \end{aligned} \tag{2.2}$$

Strom

$$\begin{aligned} I_p &= \frac{U}{R_{g_p}} \\ &= \frac{3V}{999,58\Omega} \\ &\approx 3 * 10^{-3}A \end{aligned} \tag{2.3}$$

$$\begin{aligned} I_r &= \frac{U}{R_{g_r}} \\ &= \frac{3V}{2401k\Omega} \\ &\approx 1,25 * 10^{-6}A \end{aligned} \tag{2.4}$$

elektrische Leistung

$$\begin{aligned} P_p &= \frac{U^2}{R_{g_p}} \\ &= \frac{3V * 3V}{999,58\Omega} \\ &\approx 0,009W \end{aligned} \tag{2.5}$$

$$\begin{aligned} P_r &= \frac{U^2}{R_{g_r}} \\ &= \frac{3V * 3V}{2401k\Omega} \\ &\approx 3,75 * 10^{-6}W \end{aligned} \tag{2.6}$$

Die Berechnung ergibt, dass der maximale Strom der Schaltung im Rahmen des ADC liegt.

Die Stromaufnahme der Spannungsreferenz beträgt 3,9 µA. Der ADC hat im aktiven Zustand eine Stromaufnahme von maximal 90 µA. Aus Berechnung 2.3 und den Stromaufnahmen der beiden IC's ergibt sich folgende maximale Gesamtstromaufnahme der Schaltung.

$$\begin{aligned} I_{max} &= I_p + I_{REF3330} + I_{ADC} \\ &= 3000 * 10^{-6}A + 3,9 * 10^{-6}A + 90 * 10^{-6}A \\ &= 3093,9 * 10^{-6}A \end{aligned} \tag{2.7}$$

$$\begin{aligned} I_{min} &= I_r + I_{REF3330} + I_{ADC} \\ &= 1,25 * 10^{-6}A + 3,9 * 10^{-6}A + 90 * 10^{-6}A \\ &= 95,15 * 10^{-6}A \end{aligned} \tag{2.8}$$

Zur Berechnung der Bitbreite wird die Widerstandsänderung pro Grad Lenkeinschlag zugrunde gelegt. Das Potentiometer besitzt einen effektive Rotation von $270^\circ \pm 10^\circ$. Aus dem Widerstandswert und der effektiven Rotation ergibt sich die Widerstandsänderung pro Grad, woraus sich die Spannungsänderung berechnen lässt. Hierfür wird zuerst die Spannung bei Mittelstellung und danach bei einem Grad Abweichung berechnet. Aus der Differenz der Spannungen und der Versorgungsspannung kann letztlich die benötigte Auflösung berechnet werden.

Widerstandsänderung pro Grad

$$\begin{aligned} R_{1^\circ} &= \frac{R_{12}}{Rotation} \\ &= \frac{1000\Omega}{270^\circ} \\ &= 3,703\Omega/^\circ \end{aligned} \tag{2.9}$$

Spannungsänderung pro Grad

Die Widerstandsbezeichungen in den folgenden Berechnungen beziehen sich auf Abb. 2.8.

1. Lenkung in Mittelstellung

$$\begin{aligned} R_p &= R_L||(\frac{R_{12}}{2}) \\ &= \frac{R_L * (\frac{R_{12}}{2})}{R_L + (\frac{R_{12}}{2})} \\ &= \frac{2,4M\Omega * 500\Omega}{2,4M\Omega + 500\Omega} \\ &\approx 499,8958\Omega \end{aligned} \tag{2.10}$$

$$\begin{aligned} U_m &= \frac{U_{Versorgung} * (\frac{R_{12}}{2})}{R_p + (\frac{R_{12}}{2})} \\ &= \frac{3V * 500\Omega}{499,8958\Omega + 500\Omega} \\ &\approx 1,5001562V \end{aligned} \tag{2.11}$$

2. Lenkung in Mittelstellung + 1°

$$\begin{aligned} R_p &= R_L||(R_{12} - R_{1^\circ}) \\ &= \frac{R_L * (R_{12} - R_{1^\circ})}{R_L + R_{12} - R_{1^\circ}} \\ &= \frac{2,4M\Omega * 496,297\Omega}{2,4M\Omega + 496,297\Omega} \\ &\approx 496,1943\Omega \end{aligned} \tag{2.12}$$

$$\begin{aligned} U_{m+1^\circ} &= \frac{U_{Versorgung} * (R_{12} - R_{1^\circ})}{R_p + R_{12} + R_{1^\circ}} \\ &= \frac{3V * 496,297\Omega}{496,1943\Omega + 503,703\Omega} \\ &\approx 1,4890439V \end{aligned} \tag{2.13}$$

Hieraus ergibt sich die Spannungsänderung pro Grad, mit der die benötigte Bitbreite berechnet werden kann.

$$\begin{aligned} U_{diff} &= U_m - U_{m+1^\circ} \\ &= 1,5001562V - 1,4890439V \\ &= 0,0111123V \end{aligned} \tag{2.14}$$

$$\begin{aligned} N &= \lceil \log_2(\frac{U_{Versorgung}}{U_{diff}}) \rceil \\ &= \lceil \log_2(\frac{3V}{0,0111123V}) \rceil \\ &= \lceil \log_2(269,9711131) \rceil \\ &= 9 \end{aligned} \tag{2.15}$$

Dies zeigt, dass der verwendete ADC mit einer Bitbreite von 12 bit ausreichend dimensioniert ist.

Der verwendete ADC arbeitet im Zweierkomplement. Der Wertebereich beträgt somit -2048 bis +2047. Ein Inkrement bzw Dekrement um 1 Bit ergibt wie folgend gezeigt eine Spannungsänderung von +/-0,00146484375V.

$$Output\_Code = 2048 * PGA * (\frac{V_{in+} - V_{in-}}{V_{DD}}) \tag{2.16}$$

$$\begin{aligned} V_{in+} &= \frac{Output\_Code}{2048 * PGA} * V_{DD} - V_{in-} \\ &= \frac{1}{2048 * PGA} * 3V - 0V \\ &= 0,00146484375V \end{aligned} \tag{2.17}$$

Da die Spannungsänderung pro Grad kein vielfaches der kleinsten darstellbare Spannungsänderung ist, muss eine möglichst nahekommende Gradzahl gewählt werden.

$$\begin{aligned} N &= \lceil \frac{U_{diff}}{V_{in+}} \rceil \\ &= \lceil \frac{0,0111123V}{0,00146484375V} \rceil \\ &= \lceil 7,5859968 \rceil \\ &= 8 \end{aligned} \tag{2.18}$$

Daraus lässt sich die Spannungsänderung bei 8 Bitschritten berechnen.

$$\begin{aligned} U_{diff_{ADC}} &= V_{in+} * N \\ &= 0,00146484375V * 8 \\ &= 0,01171875V \end{aligned} \tag{2.19}$$

$$\begin{aligned} U_{m+x^\circ} &= U_m - U_{diff_{ADC}} \\ &= 1,5001562V - 0,01171875V \\ &= 1,48843745V \end{aligned} \tag{2.20}$$

$$U_{m+x^\circ} = \frac{U_{Versorgung} * (\frac{R_{12}}{2} - R_{x^\circ})}{R_p + \frac{R_{12}}{2} + R_{x^\circ}} \tag{2.21}$$

Zur Berechnung der Widerstandsänderung bei 8 Bitschritten wird die Formel 2.21 nach R_{x° umgestellt.

$$\begin{aligned} R_{x^\circ} &= \frac{R_L * U_V + 2 * \frac{R_{12}}{2} * U_V - SQRT}{2(U_V + U_{m+x^\circ})} \\ &= \frac{2,4M\Omega * 3V + 2 * \frac{1k\Omega}{2} * 3V - 4.27255967346 \times 10^{10} M\Omega * V}{2 * (3V + 1,48843745V)} \\ &= 3,90505\Omega \end{aligned} \tag{2.22}$$

mit

$$\begin{aligned} SQRT &= \sqrt{a+b+c+d} \\ &= \sqrt{(21,6 + 2,14 * 10^{10} + 2,12 * 10^{10} + 2215446,043) M\Omega * V^2} \\ &= 4,27255967346 * 10^{10} M\Omega * V \end{aligned}$$

$$\begin{aligned} a &= R_L^2 * U_V^2 \\ &= 2,4M\Omega * 3V^2 \\ &= 21,6M\Omega * V^2 \end{aligned}$$

$$\begin{aligned} b &= 4 * R_L * \frac{R_{12}}{2} * U_V * U_{m+x^\circ} \\ &= 4 * 2,4M\Omega * \frac{1k\Omega}{2} * 3V * 1,48843745V \\ &= 2,143349928 * 10^{10} M\Omega * V^2 \end{aligned}$$

$$\begin{aligned} c &= 8 * R_L * \frac{R_{12}}{2} * U_{m+x^\circ}^2 \\ &= 8 * 2,4M\Omega * \frac{1k\Omega}{2} * 1,48843745V^2 \\ &= 2,126828201 * 10^{10} M\Omega * V^2 \end{aligned}$$

$$\begin{aligned} d &= 4 * (\frac{R_{12}}{2})^2 * U_{m+x^\circ}^2 \\ &= 4 * (\frac{1k\Omega}{2})^2 * 1,48843745V^2 \\ &= 2215446,043M\Omega * V^2 \end{aligned}$$

Mit 2.22 ergibt sich wie folgt eine Winkeländerung von $1,0543635^\circ$ bei 8 Bitschritten.

$$\begin{aligned} Grad_{ADC} &= \frac{Rotation}{R_{12}/R_{x^\circ}} \\ &= \frac{270^\circ}{1000\Omega/3,90505\Omega} \\ &= 1,0543635^\circ \end{aligned} \tag{2.23}$$

Die Differenz erscheint zwar gering, aber bei einem gemessenem Verstellwinkel von 30° ergibt sich ein realer Winkel von fast 32°.

2.5.2 Test der Elektronik

Für einen ersten Test wird die Elektronik auf einer Rasterplatine erstellt. Da der ADC nur im SOT23-Gehäuse erhältlich ist wird eine Trägerplatine von »Roth Elektronik« verwendet. Um das Entwicklungsfahrzeug nicht unnötig modifizieren zu müssen, erfolgt der Testaufbau „fliegend“. Dadurch sind alle Messpunkte leicht zugänglich und gleichzeitig wird die Entwicklung der Software ermöglicht, ohne das bestehende System umprogrammieren zu müssen.

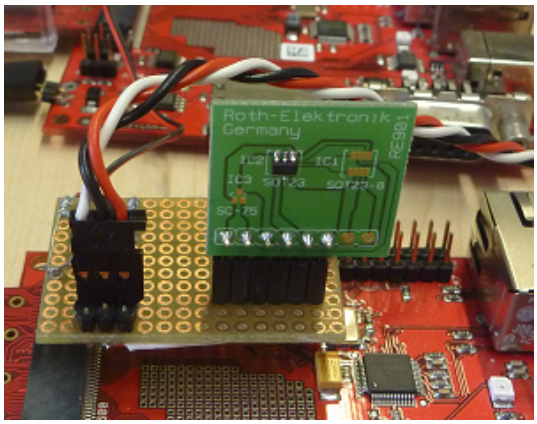

Abbildung 2.11: Testaufbau

Spannungsverlauf

In diesem Test wird geprüft, wie sich die Spannungskurve verhält. Dies sollte ein Ergebnis liefern wie es unter 2.5.1 „Design und Entwicklung“ simuliert wurde. Da wie erläutert der ADC differenziell misst, muss auch die Messung mit dem Oszilloskop differenziell erfolgen.

Daher wird zur Aufzeichnung ein SI-9000 Tastkopf verwendet. Zur Ansteuerung des Lenkservos wird die originale ARM2-Software verwendet. In der Software wird je nach Modus, in dem sich das Fahrzeug befindet das PWM-Signal für den Lenkservo erzeugt. Es existieren die 3 Modi autonomer, manueller und gedrosselt manueller Betrieb. Der Modus wird über die Fernbedienung eingestellt. Dafür misst ARM2 das Eingangssignal, welches vom Modusschalter der Fernbedienung kommt, aus. Da der Test nicht am Fahrzeug erfolgt, kommt zur Generierung des Signals ein Kontron 8021 Frequenzgenerator zum Einsatz. Die Einstellungen können dem Anhang entnommen werden. Um den Spannungsverlauf über den kompletten Bewegungsablauf aufzuzeichnen, wird die Aktor-Task des ARM2 angepasst. Das Prinzip der Tasks wird im Kapitel Software unter 3.1 „Analyse der Software“ näher erläutert. Für den Test wird die Task dahingehend geändert, das im Modus „autonomer Betrieb“ die Lenkung sich im Vollausschlag nach links befindet. Ändert sich der Modus, wird die Lenkung in den rechten Vollausschlag bewegt. Des Weiteren wird die Task für die Modus-LED abgeschaltet und der Ausgangspin für die LED als Triggerausgang verwendet. Beim Umschalten aus dem autonomen Betrieb in den manuellen wird der Ausgangspegel von low auf high gelegt. Zur Aufzeichnung der Kurve kommt ein »PicoScope 3204« 2-Kanal Oszilloskop zum Einsatz. Am Oszilloskops wird auf den Pegelwechsel getriggert und die Aufzeichnung gestartet.

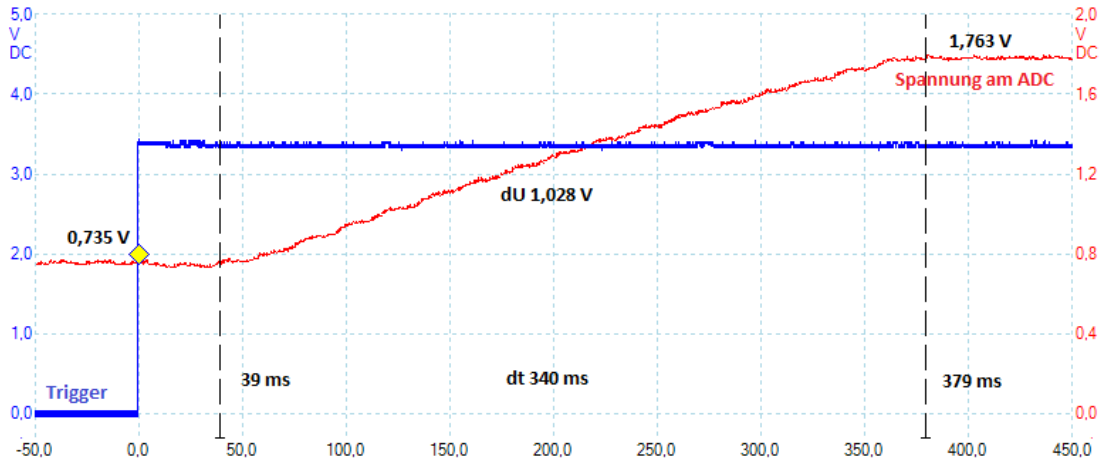

Abbildung 2.12: Spannungsverlauf

Wie im Diagramm zu erkennen ist, ändert sich Spannung wie in der Simulation linear.

Stromaufnahme

Um die Stromaufnahme zu messen, wird die Versorgungsleitung getrennt und ein Philips PM2525 Multimeter zwischen den Sensor und das COM-Board geschaltet. Die Messung

erfolgt bei ausgebautem Sensor, da wie unter 2.4 „Test der Mechanik“ erwähnt, die Endstellungen des Potentiometers andernfalls nicht erreicht werden. Als erstes wird der Strom gemessen, bei dem der ADC und das Potentiometer in Reihe geschaltet sind. Diese Messung ergibt eine Stromaufnahme von knapp 88 µA. Dieser Wert liegt unter dem in 2.8 berechneten Wert. Danach wird der Strom gemessen, bei dem der ADC und das Potentiometer parallel geschaltet sind. Hier ergibt die Messung eine Stromaufnahme von 3,139 mA. Diese ist zwar um 45,1 µA größer als unter 2.7 berechnet, was aber eine vernachlässigbare Abweichung darstellt.

Spannungsschankungen

Abschliessend wird geprüft, ob die Schaltung stabil auf Spannungsänderungen reagiert. Dafür wird der Sensor an ein Rhode&Schwarz Stromversorgungsgerät Typ NGT 20 angeschlossen. Zur Messung der Ausgangsspannung der Referenzspannungsquelle kommt wieder ein Philips PM 2525 Multimeter zum Einsatz. Laut Datenblatt arbeitet der REF3330 bis zu einer Versorgungsspannung von $V_{out} + 0,2V$ stabil. V_{out} beträgt im vorliegenden Fall 3V. Im Test wird die Spannung am Stromversorgungsgerät von 5V auf 3V gesenkt. Der Test zeigt, dass die Ausgangsspannung bis 3,1V stabil bleibt.

Somit ist gewährleistet, dass die Messung über einen großen Bereich unabhängig von der anliegenden Akkuspannung ist.

2.5.3 Implementation

Nachdem die Tests erfolgreich ausgefallen sind, wird das endgültige Layout in Protel Altium Designer erstellt. Alle Komponenten werden in SMD-Bauweise verwendetet. Dadurch kann die Platine so klein gehalten werden, dass sie mit dem Potentiometer verklebt eine komplette Sensoreinheit ergibt.

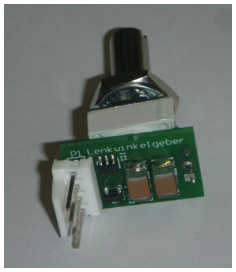

Abbildung 2.13: fertig aufgebauter Sensor

Kapitel 3

Software

In diesem Kapitel wird die Entwicklung der Software beschrieben. Das Kapitel teilt sich dabei in die Analyse der bestehenden, sowie die Implementation der neuen Software und Lösungsvorschläge der auftretenden und nicht behandelbaren Probleme.

3.1 Analyse der bestehenden Software

In diesem Abschnitt werden die bestehenden Softwareteile analysiert, welche für die vorliegende Arbeit nötig sind. Er setzt sich aus den Teilen Scheduler, Daten-Container, PC-Kommunikation, Sensor-Kommunikation, IO-Box-Kommunikation und PWM-Generierung zusammen.

Wie unter 2.1.3 „Analyse der IO-Box“ beschrieben, sind auf dem Fahrzeug 2 ARM-Mikrocontroller verbaut. Dabei deckt ARM1 die Kommunikation mit dem PC, die Verarbeitung der Sensorwerte und die Steuerung der Beleuchtung ab. Auf ARM2 erfolgt die Aufnahme und Verarbeitung der PWM-Signale des Empfängers im manuellen, sowie die Erzeugung der PWM-Signale im autonomen Betrieb. Des Weiteren erfolgt auf ARM2 die Überwachung der Akkus. Die Software ist auf beiden Mikrocontrollern modular aufgebaut. Alle Module sind als Task ausgeführt, welche über den Scheduler in fester Reihenfolge abgearbeitet werden. Jedes Modul besteht dabei aus einer Initialisierungs- und einer Updatefunktion.

3.1.1 Scheduler

Der Scheduler basiert auf einem Konzept von Michael J. Pont (Pont, 2001). Es handelt sich dabei um einen nicht-präemptiven Scheduler für harte Echtzeitanforderungen, welcher auf beiden ARM-Sticks zum Einsatz kommt. Eine Erläuterung zur Implementation des Schedulers kann der Arbeit „Ein modulares Sensor-Aktor-System für mobile Robotik“ von Sebastian Eickhoff (Eickhoff, 2011) entnommen werden.

Der verwendete Scheduler wird durch einen Timer gesteuert. Jeder Timerablauf erzeugt einen Interrupt, im folgenden Tick genannt.

Wichtig für die vorliegende Arbeit ist die Initialisierung der einzelnen Tasks. Beim Systemstart werden in `main()` die Initialisierungen der Tasks aufgerufen und die Tasks anschließend dem Scheduler bekannt gemacht. In der Funktion `SCH_Add_Task()` wird für jede Task eine Struktur `sTask` angelegt, in der ein Handle auf die Updatefunktion, die Anzahl an Ticks bis zum ersten Start (Delay) und die Anzahl der Ticks in der die Task zyklisch ausgeführt werden soll, gespeichert wird. Diese Strukturen werden in dem Feld `SCH_tasks_G[]` gehalten. In der `SCH_Update()`-Funktion, welche durch den Timerinterrupt aufgerufen wird, wird geprüft, ob die Zeit bis zum ersten Start abgelaufen ist. Falls nicht, wird das Delay dekrementiert. Ist die Delay-Zeit abgelaufen, werden die Updatefunktionen der Tasks in der Reihenfolge, in der sie dem Scheduler bekannt gemacht wurden abgearbeitet. Bei der vorliegenden Scheduler-Implementation ist keine Möglichkeit vorgesehen, zu prüfen ob eine Task erfolgreich gestartet wurde und arbeitet. Somit ist nicht gewährleistet, dass Abhängigkeiten zwischen den Tasks erfüllt sind. Um eine automatischen Setup der Lenkregelung zu implementieren ist es aber nötig, dass die Tasks zumindest in einer kontrollierten Reihenfolge starten und dies prüfbar ist. Da alle Berechnungen der Regelungen und des Gebers von einer wohl definierten Lenkstellung ausgehen, zum Beispiel Lenkung in Mittelstellung, ist es nötig, dass die Sensorik abgefragt werden kann um den Spannungswert der Lenkstellung zu speichern. Deshalb muss sichergestellt sein, dass das Modul `mod_SensCom` arbeitet wenn die Initialisierung der Lenkungsmodule erfolgt.

3.1.2 Modul `mod_dataContainer`

Das Modul `mod_dataContainer` stellt eine global zugängliche Daten-Struktur zur Verfügung. In dieser werden alle Werte gespeichert, die bei der Kommunikation zwischen den einzelnen Task bzw. den ARM-Mikrocontrollern benötigt werden. Jedes ARM-System verfügt über einen eigenen Daten-Container. Im folgenden werden die Variablen erläutert, welche für den Lenkwinkelregler bzw den Lenkwinkelgeber nötig sind.

ARM 1	
Variable	*Bedeutung*
`out_steerangle`	dient zur Speicherung des Ist-Lenkwinkels zur Übertragung an den PC
`in_steerangle`	dient zur Speicherung des vom PC berechneten Soll-Lenkwinkels zur Übertragung an ARM2
ARM 2	
Variable	*Bedeutung*
`steer_in`	dient zur Speicherung des vom PC berechneten Soll-Lenkwinkels, welcher über ARM1 an ARM2 übertragen wird

3.1.3 Modul `mod_PcCom`

Die Kommunikation mit dem PC erfolgt über die USB-Schnittstelle. Die USB-Implementation stammt von der Firma Hitex und wird hier nicht weiter erläutert. Darauf setzt die PC-Kommunikationstask auf. Sie empfängt zum einen die Steuerbefehl für die Geschwindigkeit, sowie den Lenkeinschlag vom PC und zum anderen sendet sie die Sensorwerte und die fahrdynamischen Zustände an den PC. Hier wurde schon die Übertragung des Ist-Lenkwinkels vorgesehen, wodurch die Task nicht angepasst werden muss. Die PC-Kommunikation ist wie erwähnt auf ARM1 implementiert. Daher werden alle vom PC bereitgestellten Soll-Werte wie Geschwindigkeit und Lenkeinschlag für das Senden an ARM2 im Daten-Container zwischengespeichert.

3.1.4 Modul `mod_SensCom`

Die Sensor-Kommunikations-Task verwendet das I2C-Bus-Protokoll. Über dieses Bussystem ist auch der Analog-Digital-Wandler des Lenkwinkelsensors an die IO-Box angeschlossen. Die I2C-Kommunikation wird über einen linearen Zustandsautomaten realisiert. Dadurch wird jeder Kommunikationsteilnehmer in einer gleichbleibenden Reihenfolge bedient. Im ersten Durchlauf des Zustandsautomaten erfolgt die Initialisierung der Busteilnehmer. Jeder weitere Durchlauf dient anschließend der Abfrage der Sensorwerte. Diese werden im Daten-Container gespeichert. Hier muss die Konfiguration und die Abfrage des Lenkwinkelsensors hinzugefügt werden.

3.1.5 Modul `mod_BoxCom`

Die Kommunikation zwischen den beiden ARM-Sticks erfolgt über den CAN-Bus. Hier wird unter anderem der Soll-Lenkwinkel und die Soll-Geschwindigkeit von ARM1 an ARM2 übertragen. Es muss hier die Übertragung des Ist-Wertes der Lenkung hinzugefügt werden. Des Weiteren findet hier die Übertragung der Spannungsmessungen von ARM2 zu ARM1 statt.

Der PC liefert für den Lenkeinschlag einen Wert zwischen -100 und +100. Aus diesem Wert wird auf ARM2 in der IO-Box-Kommunikationstask das PWM-Signal berechnet. Diese Berechnung muss später durch den Regler erfolgen, da der PC dann lediglich den Soll-Wert in Grad liefert.

3.1.6 Modul `mod_Actuator`

In der Aktuator-Task werden die ARM-internen Timer programmiert. Diese Timer generieren die PWM-Signale für den Lenkservo und den Fahrregler. Dies geschieht wie erwähnt auf ARM2. Da der Lenkeinschlag sich nicht ändern darf, bis das automatische Setup der Lenkwinkelregelung, beziehungsweise des Lenkwinkelgebers abgeschlossen ist, muss hier das Setzen der PWM-Werte kontrolliert werden.

3.2 Implementation

Dieser Abschnitt beschreibt die Softwareänderungen und -neuerungen, die zur Implementation des Gebers und der Regelung nötig sind. Zuerst erfolgt die Beschreibung der Änderungen an der bestehenden Software, um die Arbeit des Gebers bzw. des Reglers zu ermöglichen. Es erfolgt dabei eine Unterteilung in die Abschnitte Scheduler, Daten-Container, Sensor-Kommunikation, IO-Box-Kommunikation und PWM-Generierung. Anschließend werden die Neuerungen beschrieben. Diese sind das Modul des Lenkwinkelgebers und das Modul des Lenkwinkelreglers. Die Kommunikation mit dem PC wird hier nicht behandelt, da im Module PC-Kommunikation (siehe 3.1.3) keine Änderungen nötig sind.

3.2.1 Scheduler

Für das automatische Setup muss sichergestellt sein, dass die Kommunikation zu den Sensoren vor der Initialisierung des Lenkwinkelgeber- bzw. Lenkwinkelreglermodules

funktioniert. Alle Berechnungen des Lenkwinkelgebers und der Lenkwinkeregelung erfolgen auf dem Spannungswert der bei Lenkmittelstellung anliegt und in der Initialisierung gespeichert wird. Dieser kann aber erst ermittelt werden, wenn das Modul `mod_SensCom` bereits arbeitet. Deshalb wird der Aufruf der Initialisierungsfunktion der einzelnen Tasks aus der Main-Funktion in den Scheduler verschoben.

Der Scheduler wird um die folgenden Variablen erweitert.
In der Struktur `sTask` wird ein Zeiger auf die Initialisierungsfunktion der Task hinzugefügt. Weiterhin wird die Bitmaske `runBefore` zur Struktur `sTask` hinzugefügt. Jedes Bit steht dabei für den Index im Array `SCH_tasks_G[]`, in welchem die Strukturen der einzelnen Task gespeichert sind. Zusätzlich wird die Struktur `sTask` um die Variable `initFinished` erweitert. Diese zeigt an, ob die Initialisierung der dazugehörigen Task abgeschlossen ist. Zum Schluss wird noch die globale Bitmaske `tasksAlreadyRuns` benötigt, welche zum Anzeigen der Tasks dient, die mindestens einmal ihre Updatefunktion ausgeführt haben. Erst nach einem erfolgreichen Durchlauf der Updatefunktion wird die entsprechende Task als vollständig initialisiert angesehen.

Wird nun eine Task zum Scheduler hinzugefügt, wird zusätzlich eine Liste von Zeigern der Updatefunktionen übergeben, welche mindestens einmal ausgeführt worden sein müssen, bevor die eigene Initialisierung erfolgen kann. Diese Liste wird mit den in `SCH_tasks_G[]` gespeicherten Tasks verglichen und die Bitmaske `runBefore` der neuen Task erzeugt. Ist es nötig, dass eine Task vor der neuen laufen muss, wird das entsprechende Bit in der Bitmaske gesetzt.

In der Update-Funktion des Schedulers wird nun beim Taskwechsel überprüft, ob die Bitmaske `runBefore` der nächsten auszuführenden Task und `tasksAlreadyRuns` gleich sind. Ist dies der Fall, wird über `initFinished` geprüft, ob die Initialisierung der Task bereits durchgeführt wurde. Falls die Task noch nicht erfolgreich initialisiert wurde, wird die Initialisierung der Task aufgerufen. Ist diese erfolgreich, wird die Variable `initFinished` auf `true` gesetzt. Schlägt die Initialisierung hingegen fehl, wird die Variable `initFinished` auf `false` gesetzt. Somit wird die Initialisierung im nächsten Scheduler-Zyklus erneut aufgerufen. Dies kann zum Beispiel passieren, wenn die Sensoren noch nicht Einwandfrei arbeiten.

Ist die Task erfolgreich initialisiert worden, wird das Delay mit jedem Aufruf des Schedulerupdates dekrementiert. Erreicht das Delay der Task den Wert 0, so wird sie in den Updatezustand versetzt und kann durch den Dispatcher ausgeführt werden. Dieser setzt bei der Ausführung der Task das entsprechende Bit in der Maske `tasksAlreadyRuns`. Dadurch wird angezeigt, dass von dieser Task abhängige Tasks nun initialisiert werden können.

Dieses Verhalten ist zwar vorerst nur auf ARM1 nötig, aber die Änderung erfolgt auf beiden ARM-Sticks, um das Systemgrundgerüst identisch zu halten.

3.2.2 Daten-Container

In den Daten-Containern müssen die Strukturen für den Spannungswert des ADC erweitert werden. Es wird dafür die Variable `steerVoltage` auf beiden ARM-Sticks hinzugefügt. In dieser wird die zuletzt gemessene Spannung gespeichert. Die Initialisierung erfolgt mit dem Wert 0. Dies ist der für die verwendete Schaltung minimalste Wert des ADC, welcher wie unter 2.4 „Test der Mechanik“ beschrieben nicht erreicht wird. Dadurch kann ermittelt werden, ob die Kommunikation zum ADC funktioniert. Bei einer Spannung grösser 0 wird angenommen, dass die Kommunikation funktioniert. Auf ARM1 wird zusätzlich die Variable `init_LWG` hinzugefügt. Diese wird als boolesche Variable verwendet, und wird zur Steuerung der Datenübertragung zu ARM2 verwendet. Ist diese `true`, kann der aktuell gemessene Wert des Sensors in der IO-Box-Kommunikation gesendet werden.

3.2.3 Sensor-Kommunikation

Um die Kommunikation mit dem Lenkwinkelsensor zu ermöglichen, wird der Zustandsautomat der I2C-Kommunikation auf ARM1 um die Zustände `CONFIG_LWG` und `READ_LWG` erweitert. Der Zustand `CONFIG_LWG` wird beim Systemstart durchlaufen und der ADC konfiguriert. Zur Konfiguration wird das folgende Byte an den ADC übertragen.

Bit	Bezeichnung	Wert	Bedeutung
7	`ST/BSY`	`1`	muss im „continuous conversion mode“ auf 1 gesetzt werden
4	`SC`	`0`	der ADC arbeitet im „continuous conversion mode“
1 / 0	`PGA`	`0 / 0`	Programmable Gain Amplifier auf 1 einstellen
2,3,5,6		`0`	müsssen immer 0 sein

Eine nähere Erklärung kann dem Datenblatt des ADC entnommen werden.

Der Zustand `READ_LWG` liest 2 Byte aus dem Output-Register des ADC und setzt anschliessend den Folgezustand auf `GETPROTOCOL_READ_USVL_INITI`. Im Folgezustand

erfolgt das Kopieren des I2C-Empfangspuffers mit dem Spannungswert in den Daten-Container. Danach steht der Wert allen Tasks auf ARM1 und zur Übertragung an ARM2 zur Verfügung.

3.2.4 IO-Box-Kommunikation

Auf ARM1 wird die Funktion `task_BoxCom_Update()` geändert. Ursprünglich wurde die Gangwahl des verbauten Getriebes durch den PC gesteuert. Das Getriebe ist mittlerweile fest auf den 3. Gang eingestellt, da die Vergangenheit gezeigt hat, dass das Schalten bei Strassenfahrt unnötig ist. Die Übertragung des Ganges wird entfernt und durch das Übertragen der Spannung ersetzt. Dadurch bleibt die zu übertragende Datenmenge gleich.

Beim Schreiben des Sendepuffers wird vorab geprüft, ob die Variable `init_LWG` den Wert `true` enthält. Solange dies nicht der Fall ist, wird der minimal mögliche positive Wert des ADC übertragen. Dadurch wird ARM2 signalisiert, dass die Initialisierung des Lenkwinkelgebers noch nicht abgeschlossen ist und die Räder weiterhin in Geradeausstellung verbleiben müssen. Dies wird unter 3.2.6 „Implementation Lenkwinkelgeber“ näher erläutert.

Auf ARM2 wird die Funktion `isr_CAN` angepasst. Die Berechnung der PWM-Signale wird entfernt und die Funktion dahingehend geändert, dass die übertragenen Werte lediglich im Daten-Container gespeichert werden.

3.2.5 PWM-Generierung

In der Aktuator-Task auf ARM2 wird das Setzten der PWM-Signale in Abhängigkeit des Fahrzeugmodus gesteuert. Solange Lenkwinkelregler und Lenkwinkelgeber nicht einwandfrei arbeiten, müssen sich die Räder in Geradeausstellung befinden.

Im autonomen Betrieb wird der vom Lenkwinkelregler berechnet PWM-Wert im Timer gesetzt. Der Lenkwinkelregler berechnet aber erst nach einer erfolgreichen Initialisierung diesen Wert. Solange der Regler nicht arbeitet, wird der Initialwert für das PWM-Signal des Datencontainers nicht geändert. Somit bleiben die Räder bis zum Ende der Initialisierung in Geradeausstellung.

Im manuellen Modus wird der Wert für das PWM-Signal von der Capture-Task geliefert. Diese Task misst die Signale, welche von der Fernbedienung erzeugt werden. Da beim Umschalten in den autonomen Modus kein erneutes Setup erfolgt, muss geprüft werden, ob der Lenkwinkelgeber und der Lenkwinkelregler erfolgreich initialisiert wurden. Dafür

wird geprüft, ob der von ARM1 übertragene Spannungswert 0 Volt beträgt. Falls ja, wird davon ausgegangen, dass die Initialisierung des Lenkwinkelgebers bzw. des Lenkwinkelreglers noch nicht abgeschlossen ist und der Timer erhält den Wert der die Lenkung in Mittelstellung hält. Andernfalls wird das von der Capture-Task gemessene PWM-Signal an den Timer übergeben.

3.2.6 Lenkwinkelgebermodul

Das Lenkwinkelgebermodul hat die Aufgabe, den Ist-Zustand der Lenkung zu ermitteln und an den PC zu übermitteln. Als zu übertragende Einheit wird der Winkel in Grad gewählt.

Die Implementation erfolgt auf ARM1, da sowohl die Kommunikation über den I2C-Bus zum Sensor, als auch die Kommunikation zum PC hier stattfindet.

Wie jedes Modul besteht auch das Lenkwinkelgebermodul aus einem Initialisierungs- und einem Updateteil, welche im folgenden getrennt erläutert werden.

Initialisierung

Über ein Handle auf die im Daten-Container gehaltene Datenstruktur erhält das Lenkwinkelgebermodul Zugriff auf den Spannungswert des ADC. Dieser Zeiger wird in der Initialisierungsphase über `getDataContainer()` geholt und in `pDataContainer` gespeichert.

Für das automatische Setup ist es nötig, dass die Lenkung beim Systemstart in Geradeausstellung gebracht und bis zum Abschluss der Initialisierung dort gehalten wird.
Wie unter 2.3 „Implementation der Mechanik“ beschrieben, wird mechanisch bedingt nicht der komplette Verstellbereich des Potentiometers verwendet. Daher kann davon ausgegangen werden, dass im normalen Betrieb die Endstellungen des Potentiometers nicht erreicht werden. Dies kann zur Überprüfung der Kommunikation mit dem Sensor verwendet werden. Wie unter 3.2.2 „Modul Daten-Container“ beschrieben wird in der Daten-Struktur die Variable für die Spannung mit 0 initialisiert. Während der Lenkwinkelgeberinitialisierung wird geprüft, ob der aktuelle gespeicherte Wert gleich 0 ist. Ist dies der Fall, so wird die Initialisierung abgebrochen, da davon ausgegangen wird, dass die Kommunikation zum Sensor noch nicht einwandfrei funktioniert. Der Scheduler muss die Initialisierung dann zu einem späteren Zeitpunkt erneut aufrufen.
Wenn die Prüfung ergibt, dass der aktuelle Wert ungleich 0 ist, wird davon ausgegangen,

dass die Kommunikation zum ADC funktioniert und das sich die Räder in Geradeausstellung befinden. Der zu diesem Zeitpunkt gemessene Wert wird für die späteren Berechnungen als Referenzspannung in der Variable `middleVoltage` gespeichert. Zusätzlich wird die Struktur-Variable `pDataContainer->init_LWG` auf `true` gesetzt, wodurch die Datenübertragung zu ARM2 freigegeben wird. Siehe dazu 3.2.4 „IO-Box-Kommunikation“.

Update

Die Funktion `task_LWGeber_Update()` wird zyklisch alle 50ms durch den Scheduler aufgerufen.

Zur Berechnung des Lenkwinkels wird im ersten Schritt die Differenz der zuletzt gemessenen und der in `middleVoltage` gespeicherten Spannung berechnet.
Diese Differenz wird modulo durch `binGrad` geteilt. Darin ist der Wert gespeichert um den sich die Spannung bei $1,0543635°$ Lenkwinkeländerung ändert. Somit ergibt die Modulodivision die Abweichung der Lenkung zur Mittelstellung.
Auf die Abweichung werden $90°$ aufaddiert und in der Struktur-Variablen `pDataContainer->out_steerangle` gespeichert. Ist die Abweichung grösser als $90°$ bedeutet dies einen Lenkeinschlag nach rechts und analog dazu ein kleinerer Wert einen Lenkeinschlag nach links.
Dieser Wert wird durch die PC-Kommunikationstask an den PC übertragen.

3.2.7 Lenkwinkelreglermodul

Der Regler soll sicherstellen, dass der vom PC berechnete Sollwert erreicht und gehalten wird. Dies bedeutet, dass der Regler ein Führungs- und ein Störverhalten aufweisen muss.(Busch, 2005, S.157)
Der Regler wird auf ARM2 implementiert, da hier die Signalerzeugung für den Lenkservo erfolgt. Wie unter 3.2.3 „Sensor-Kommunikation“ beschrieben erfolgt die Abfrage des Lenkwinkelsensors auf ARM1. Somit ergibt sich die abgebildete Kommunikationskette.

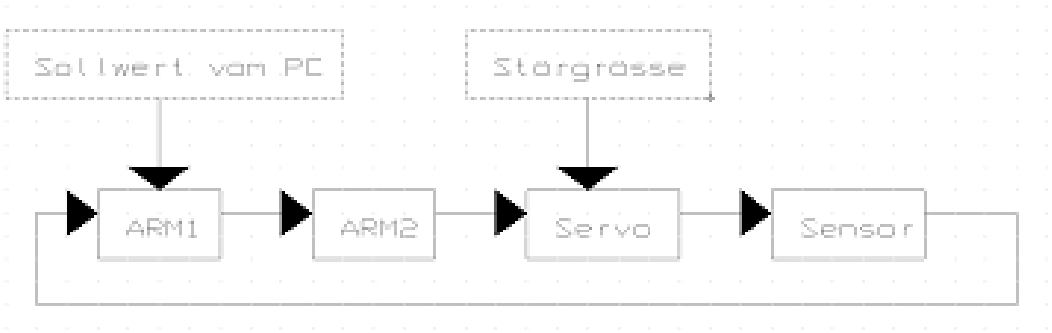

Abbildung 3.1: Kommunikationskette

Die Beurteilung des Zeitverhaltens eines Regelkreises erfolgt über die Aufnahme der Sprungantwort.

> Hierzu wird die Eingangsgröße des Regelkreisgliedes sprunghaft von null auf einen Endwert $\widehat{x}_3$ verändert. (Busch, 2005, S. 31)

Dies wird unter 2.5.2 „Test der Elektronik“ bereits getan, wodurch die Aufzeichnung des Spannungsverlaufes als Sprungantwort verwendet werden kann.
Aus ihr ist zu erkennen, dass der Regelkreis ein PT1-Verhalten aufweist. Man spricht von einem PT1-Verhalten, wenn der Regelkreis ein Verzögerungsglied enthält und die Regelgröße einen stationären Endwert erreicht.(Unbehauen, 2005, S.91) Im vorliegenden Fall weist die Aufzeichnung eine Totzeit (Verzögerung) von 39 ms auf. Ein PT1-Verhalten erweist sich in so fern als vorteilhaft, da zur Berechnung der Regelparameter die Einstellregeln nach Takahashi (Lutz und Wendt, 2007) verwendet werden können. Ein entscheidendes Kriterium ist hierbei die Totzeit des Regelkreises, da über diese die Koeffizenten des Reglers berechnet werden.

Da bei der Aufzeichnung der Sprungantwort lediglich die Totzeit zwischen der Aktuator-Task auf ARM2 und dem Potentiometer gemessen wird ist noch eine Messung der Laufzeit zwischen ARM1 und ARM2 nötig. Diese Zeit, die benötigt wird um den Messwert von ARM1 an das Lewnkwinkelregelmodul auf ARM2 zu übermitteln, muss zu der in der Sprungantwort aufgezeichneten Totzeit addiert werden. Die Messung wird im folgenden erklärt. Um nicht die Initialisierungszeit zu Messen muss sichergestellt sein, dass das komplette Mikrocontrollersystem initialisiert wurde. Auf Grund des Systemaufbaus wird das Lenkwinkelregelmodul als letztes initialisiert. Um dies auf ARM1 zu signalisieren wird die Übertragung der Spannungswerte der Akkus verwendet. Dafür wird die Task zur Messung der Akkuspannung deaktiviert. Nachdem die Initialisierung des Regler abgeschlossen ist wird die Variable für die Spannung des Fahrakkus auf 12V gesetzt. Auf ARM1 wird in der Sensortask geprüft ob diese Spannung gesetzt ist. Falls ja wird ein Triggersignal ausgelöst, wodurch die Messung beginnt. Gleichzeitig wird der Wert der Spannung des Lenkwinkel-

sensors mit 0xFFFF überschrieben. In der Aktuatortask auf ARM2 wird die übertragene Spannung auf diesen Wert geprüft und ebenfalls ein Triggersignal erzeugt, wenn die Prüfung positiv ausfällt.

Bei der ersten Messung ergibt sich eine Laufzeit von 110 ms.

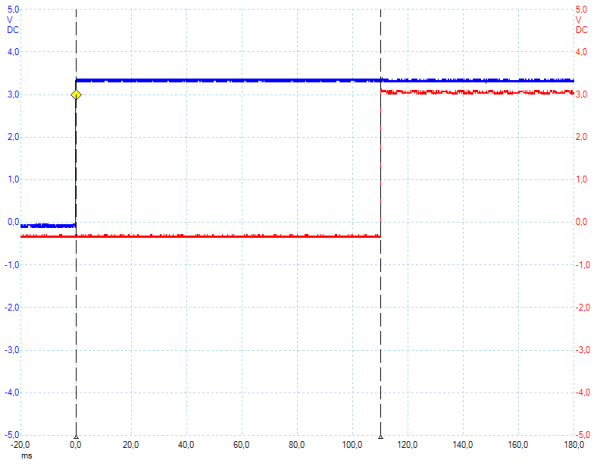

Abbildung 3.2: 110ms Laufzeit

Mit der Totzeit aus der Sprungantwort ergibt sich somit eine Gesamttotzeit von 149 ms. Diese lange Laufzeit ist auf die Systemeigenschaften zurückzuführen. Da alle Tasks zyklisch abgearbeitet werden vergeht bei ungünstiger Anordnung der Tasks zu viel Zeit zwischen dem Lesen des Sensors und dem Übertragen der gemessenen Spannung an ARM2. Daher wird die Anordnung der Tasks geändert, sodass die geringstmögliche Zeit zwischen Sensor-Task und IO-Box-Task vergeht. Dadurch kann die Laufzeit massiv gesenkt werden. Eine neue Messung ergibt eine Laufzeit von 4ms. Zur Sicherheit wird die Messung mehrfach wiederholt. Dabei zeigt sich, dass die Laufzeit stark schwankt. Die längste gemessene Laufzeit beträgt 14ms. Dies ergibt sich aus der Tatsache, dass beide ARM-Sticks nicht synchron zueinander laufen. Startet ein Stick langsamer bzw schneller als der andere, verschiebt sich somit die Zeit zwischen den IO-Box-Tasks auf ARM1 und ARM2.

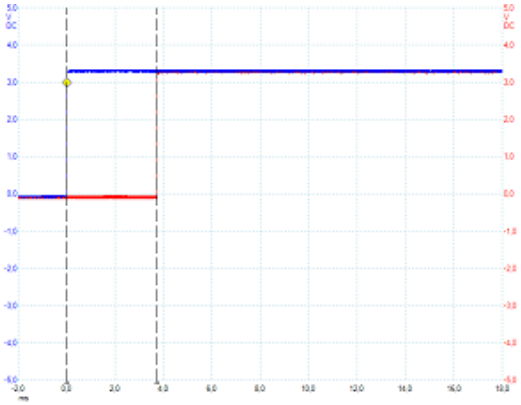

Abbildung 3.3: 4ms Laufzeit

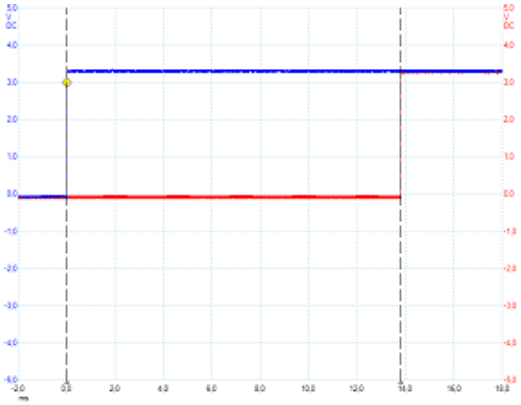

Abbildung 3.4: 14ms Laufzeit

Aus den gemessenen Zeiten ergibt sich somit eine Gesamttotzeit von 53 ms.

Da dies lediglich eine Momentaufnaheme der Totzeit ist, kann nicht davon ausgegangen werden, dass es sich um die längste mögliche Totzeit handelt. Aus diesem Grund erfolgt eine Berechnung auf Basis der Softwarestruktur und den Zeiten der einzelnen Tasks.

Zu diesem Zweck müssen im ersten Schritt die Abarbeitungszeiten jeder einzelnen Tasks ermittelt werden. Des Weiteren sind verschiedene Messungen zwischen den Tasks auf jedem einzelnen ARM-System nötig. Dabei ist zu beachten, dass die Messungen ohne angeschlossene Sensoren erfolgt. Dies ist wichtig, da dadurch zum Beispiel keine Interrupts durch die Inkrementalgeber ausgelöst werden und somit die Rechenzeiten der Tasks nicht unterbrochen werden. Somit wird das System nicht im Volllastbetrieb betrachtet, was aber ausreichend ist.

Die Ergebnisse sind in folgenden Tabellen aufgeführt.

ARM 1			
Task		*Interval*	*längste gemessene Dauer*
1	mod_Lights	50ms	16 µs
2	mod_IrRange	50ms	14 µs
3	mod_Distance	25ms	14 µs
4	mod_PcCom	25ms	250 µs
5	mod_SensCom	50ms	28 µs
6	mod_LWG	50ms	18 µs
7	mod_BoxCom	10ms	35 µs

ARM 1			
Task		*Interval*	*längste gemessene Dauer*
1	mod_Actuators	10ms	4 µs
1	mod_ModeLED	10ms	2 µs
1	mod_Voltages	10ms	5 µs
1	mod_RcCapture	10ms	5 µs
1	mod_BoxCom	10ms	11 µs
1	mod_LWRegler	10ms	0 µs

Zusätzlich wird auf beiden ARM-Systemen die Gesamtdauer aller Tasks ausgemessen, wenn diese nacheinander abgearbeitet werden. Dies geschieht auf ARM1 alle 50ms und auf ARM2 alle 10ms. Auf ARM1 dauert dies insgesamt 10ms und auf ARM2 2ms. Hierbei ist zu beachten, dass sich die Gesamtdauer aller Tasks aus den Zeiten der einzelnen Task, den Zeiten der Taskwechsel und auftretenden Interrupts zusammensetzt. Aus den gemessenen Werten wird folgende Grafik erstellt. Die Darstellung enthält keine mechanischen Verzögerungen und soll nur das Verhalten zwischen dem Lesen des Sensors und dem Erzeugen des PWM-Signals verdeutlichen.

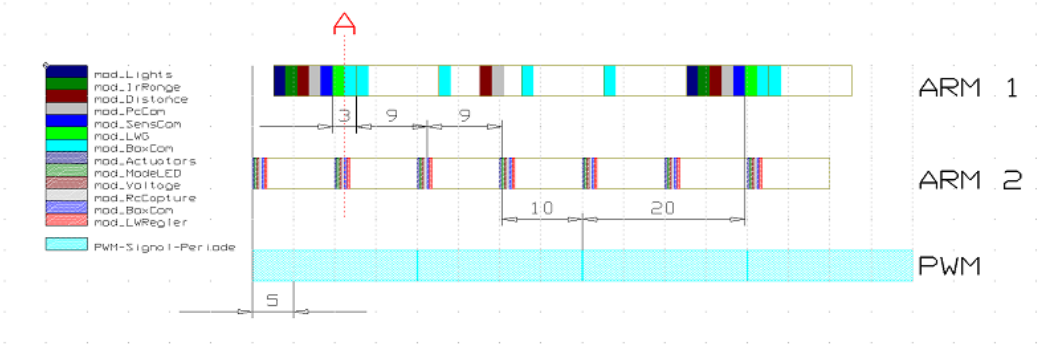

Abbildung 3.5: Timing (grössere Abbildung 5.2)

Wie zu erkennen ist, tritt ein Extremfall ein, wenn ARM1 später startet als ARM2. Dadurch sind die BoxCom-Tasks zeitlich so versetzt, dass auf ARM2 das Speichern der CAN-Nachrichten gerade abgeschlossen ist, wenn ARM1 die neuen Werte sendet. Dies passiert an der mit A markierten Stelle in Grafik 3.5. Der Einfachheit halber wurde hier ignoriert, dass zwischen der Übergabe der Werte in der BoxCom-Task an den CAN-Controller auf ARM1 und dem tatsächlichen Senden zusätzliche Zeit benötigt wird. Analog dazu wurde die Zeit zwischen dem Empfangen der CAN-Nachricht und dem Speichern durch die IO-Box-Task auf ARM2 ignoriert. Würde dies beachtet werden ergebe sich ein noch geringerer zeitlicher Versatz des Systemstarts zwischen ARM1 und ARM2 als in der Grafik

dargestellt. Aus dem Verhalten ergibt sich, dass die neuen Werte erst ungefähr 9ms nach dem Empfangen auf ARM2 in den Datencontainer geschrieben werden.

Durch die Reihenfolge der Tasks auf ARM2, ist das Setzen der Aktorwerte im aktuellen Zyklus bereits abgeschlossen. Somit vergehen weitere 9ms bis die im Lenkwinkelregler berechneten Werte durch die Aktortask im PWM-Timer gesetzt werden.

Tritt nun der Fall ein, dass zu diesem Zeitpunkt gerade eine PWM-Periode gestartet wurde, so vergehen weitere 20ms bis der neue Sollwert an den Lenkwinkelservo übertragen wird. Zusätzlich ist noch die Zeit zu beachten, welche zwischen dem Auslesen des Sensors und der Übertragung durch die BoxCom-Task auf ARM1 vergeht. Diese Zeit wurde mit 3ms gemessen.

Aus den gemessenen und den aus der Software-Betrachtung abgeleiteten Werten ergibt sich eine Totzeit von 51ms. Dies entspricht annähernd der gemessenen Zeit. Die Differenz von 2ms ergibt sich da in die Betrachtung die mechanische Verzögerung und die Rechenzeiten das ADC nicht eingeflossen sind. Aus diesem Grund erfolgt die weitere Berechnung auf Basis der gemessenen Totzeit von 53ms.

Regelkreise mit langer Totzeit lassen sich schwierig regeln. Wird der neue Sollwert zu früh gesetzt kann es vorkommen, dass während der Totzeit die Sollstellung erreicht wird und der Regler somit übersteuert bzw untersteuert. Dadurch kann das System in Schwingungen geraten.

Um eine grobe Abschätzung der Regelperiode zu erhalten wird zu Testzwecken ein einfacher Regler implementiert. Dabei wird in jedem Regelzyklus das PWM-Signal um 1 µs inkrementiert bis der Spannungswert des Lenkwinkelservos 1 Grad erreicht. Es zeigt sich, dass die Lenkung bei einer Periodendauer unter 100ms übersteuert.

Für eine Lenkwinkelregelung ist eine zeitnahe Reaktion nötig um auf kurzmöglichster Strecke auf Störgrößen und Sollwertänderungen zu reagieren. Deshalb wird aus den vorhergehenden Betrachtungen die Strecke berechnet, welche das Fahrzeug während der Totzeit zurücklegt.

Es wird die maximale erreichbare Fahrzeuggeschwindigkeit von 25km/h angesetzt. Dabei erfolgt die Berechnung einmal auf Basis der gemessen Totzeit von 53ms und zum anderen auf der gemessenen Peridodendauer von 100ms.

$$\begin{aligned} s_{53ms} &= v_{max} * t_{tot} \\ &= 25km/h * 53ms \\ &= 0,694cm/ms * 53ms \\ &= 36,81cm \end{aligned} \tag{3.1}$$

$$\begin{aligned} s_{100ms} &= v_{max} * t_{tot} \\ &= 25km/h * 100ms \\ &= 0,694cm/ms * 100ms \\ &= 69,4cm \end{aligned} \tag{3.2}$$

Diese Strecken sind verständlicherweise für eine Lenkwinkelregelung viel zu lang.

Daher erfolgt eine Berechnung der Geschwindigkeit, die das Fahrzeug maximal fahren darf, um auf einer 1cm langen Strecke auf auftretende Störgrössen zu reagieren. Eine Streckenlänge von 1cm wurde hier gewählt, da gerade die Lenkregelung extrem zeitnah reagieren muss.

$$\begin{aligned} v_{max} &= \frac{s}{t_{tot}} \\ &= \frac{1cm}{53ms} \\ &= 0,002cm/ms \\ &= 0,72km/h \end{aligned} \tag{3.3}$$

Hieraus lässt sich erkennen, dass eine wirkungsvolle Lenkwinkelregelung unter den gegeben Umständen nicht zu realisieren ist. Des Weiteren wird hiermit die Hauptforderung 5. unter 2.1.5 verletzt. Ganz besonders weil sich die Fahrzeuge im Wettkampfeinsatz schnellstmöglich bewegen müssen.
Daher wird lediglich die Task zur Lenkwinkelregelung ohne die nötigen Koeffizienten als Skelett implementiert und so konfiguriert, dass die Task nicht ausgeführt wird. Somit kann nach einer nötigen Anpassung bzw Veränderung des Systems der Regler einfach zugeschaltet werden.

Unter 3.3“Vorschläge zur Systemanpassung“ wird näher auf die nötigen Veränderungen eingegangen.

Aus den gemessenen Werten und der Periodendauer auf ARM1 ist ein weiteres Problem zu erkennen. Die IO-Box-Kommunikationstask wird alle 10ms ausgeführt und alle 50ms ein kompletter Taskzyklus welcher 10ms dauert. In den vorliegenden Messungen wird wie bereits erwähnt kein Interrupt durch die Inkrementalgeber ausgelöst. Betrachtet man das System unter Volllast so verlängert sich der Zyklus durch die auftretende Interruptverarbeitung über die 10ms hinaus. Somit ist die Echtzeitfähigkeit nicht mehr gewährleistet.

3.3 Vorschläge zur Problemlösung

Dieses Kapitel befasst sich mit den während der Entwicklung entdeckten Problemen des Systems. Dazu gehören sowohl Probleme, welche einen stabilen Systemlauf verhindern. Aber auch Probleme, welche zeitkritische Anwendungen wie eine Regelung nahezu unmöglich machen gehören dazu. Es werden nochmals die 4 prägnantesten Probleme an kleinen Beispielen erläutert und mögliche Lösungen aufgezeigt.

3.3.1 Zeitliche Verzögerung der Kommunikation

Problem

Durch die starre Aufteilung der Systeme in einzelne Tasks kommt es bei der Übertragung einzelner Parameter zur zeitlichen Verzögerung. Als Beispiel wird im folgenden der Ablauf der Übertragung der Aktorwerte vom PC zu ARM2 dargestellt.

Die Kommunikation erfolgt wie bereits erwähnt über das USB- und das CAN-Protokoll. Der Ablauf läßt sich dabei in die folgenden Schritte unterteilen:

1. Die USB-Interrupt-Service-Routine speichert die vom PC empfangenen Werte in einem Byte-Array ab.
2. Aus diesem Array liest das Module `mod_PcCom` die einzelnen Werte und speichert diese im Datencontainer ab.
3. Das Modul `mod_BoxCom` liest den Datencontainer und übergibt die Werte an das CAN-Bus-Interface.
4. Das CAN-Bus-Interface sendet die Werte an ARM2.

Wie zu erkennen ist, erfolgen hier mehrere unnötige Speicher- und Lesezugriffe von Werten, welche nur von ARM2 benötigt werden. Das Verhalten ist ebenfalls auf die I2C-

Kommunikation übertragbar. Auch hier erfolgt erst ein Zwischenspeichern der Messwerte im Datencontainer um diese später an ARM2 zu übertragen.

Lösungsvorschlag

Zur Lösung des Problems sollte das System dahingehend geändert werden, dass zum Beispiel die Sollwerte für die Servomotoren direkt aus der USB-ISR an das CAN-Interface übergeben werden. Dabei ist natürlich auf einen ausreichenden Schutz des CAN-Interfaces zu achten, da bei dieser Lösung das Schreiben von mehreren Funktionen aus geschehen kann und eventuell zu sendende Werte überschrieben werden. Dies bedarf einer eingehenderen Prüfung. Zusätzlich ist zu prüfen, ob das Speichern der Sensorwert in den Datencontainer über eine eigene Task nötig ist, oder nicht alle Werte aus den diversen ISR's direkt gespeichert werden können.

3.3.2 Slotverschobene Abarbeitung von Soll-/Istwertänderungen

Problem

In dieser Arbeit ist die vorab entwickelte Einstellung der Taskperioden und -anordnung übernommen worden. Es zeigt sich aber, dass durch die gewählte Aufteilung das Echtzeitverhalten des Systems beeinträchtigt ist. Die Verarbeitung der Tasks erfolgt in so genannten Zeitslots. Die Länge eines Slots wird dabei durch die kürzeste Periode aller Tasks bestimmt. Bei der vorherrschenden Aufteilung sind alle Taskperioden direkt oder indirekt ein Vielfaches der kürzesten Periode. Somit tritt früher oder später der Fall ein, dass alle Tasks in einem Zeitslot laufen. Ist nun die Gesamtdauer aller Tasks länger als die kleinste Periode, reagiert das System nicht mehr in der vorgegebenen Zeit.
Ein weiteres Problem besteht in der Anordnung der Tasks. Wie in Abbildung 3.5 auf ARM1 erkennbar ist, wird die BoxCom-Task alle 10ms am Anfang des Zeitslots ausgeführt. Alle 50ms wird die Task aber erst am Ende des Slots abgearbeitet. Dieses Verhalten entsteht aus der Tatsache, dass die Anordnung für diese Arbeit geändert wurde, um ein möglichst zeitnahes Senden der aktuellen Messwerte zu ermöglichen. Würde die BoxCom-Task als erste Task ausgeführt werden, würde ein vergleichbares Verhalten wie auf ARM2 auftreten. Durch die dort gewählte Reihenfolge werden die von ARM1 empfangenen Werte erst im nächsten Slot verarbeitet. Auf ARM1 würde dies bedeuten, dass die gemessenen Werte erst im nächsten Slot übertragen werden.

Lösungsvorschlag

Es musste das komplette System unter Beachtung beider Scheduler ausgemessen und angepasst werden. Dabei ist zu prüfen, in welchen Perioden die Task minimal und maximal ausgeführt werden dürfen, damit das System seine Echtzeitfähigkeit behält. Es wäre zu Prüfen, ob eine Verteilung der Perioden auf Primzahlen verhindern würde, dass zu viel Tasks im selben Slot ausgeführt werden.

Des Weiteren ist die Reihenfolge der Tasks zu prüfen und anzupassen. Dabei ist in Betracht zu ziehen, welche Werte zeitkritisch übertragen werden müssen und welche eventuell verzögert übertragen werden können. Es ist also eine einfache indirekte Priorisierung zu implementieren.

3.3.3 Zeitliche Verzögerung durch Systemkonzept

Problem

Wie unter 3.1 „Analyse der bestehenden Software“ erklärt, wird von ARM1 die Kommunikation mit den Sensoren über das I2C-Protokoll übernommen. Das Erzeugen der Stellwerte der Servomotoren wird aber durch ARM2 vorgenommen. Durch diese Aufteilung entsteht eine zeitliche Verzögerung bis die aktuellen Werte in ARM2 vorliegen. Dies ist aber gerade bei Regelungen schwierig, da das System unnötig verspätet auf Störeinflüsse reagiert und die Änderung der Sollgröße verspätet gemessen werden kann.

Lösungsvorschlag

Im Fall der Messungen über den I2C-Bus ist zu Überlegen, ob ein Multimastersystem Abhilfe schaffen kann. Da beide ARM-Mikrocontroller am I2C-Bus angeschlossen sind kann ARM2 bei Bedarf selbst die benötigten Sensorwerte abfragen und somit zeitnah verarbeiten.

3.3.4 Fehlende Komponentensynchronisation

Problem

Durch die fehlende Synchronisierung kommt es zu zeitlichen Schwankungen. Es ist somit nicht ohne weiteres möglich zu bestimmen, zu welchem Zeitpunkt eines der Teilsysteme

welche Funktion im Gesamtsystem ausführt. Diese Schwankungen können teilweise auf die elektrotechnische Eigenschaften des Systems zurückgeführt werden. Hier sind zum Beispiel kapazitive Bauteile zu beachten, bei denen eine Verzögerung durch unterschiedliche Ausgangszustände möglich ist. Dies ist aber wohl als marginal zu betrachten.

Entscheidender ist die Initialisierung der Einzelsysteme, da zum Beispiel die Initialisierung der USB-Schnittstelle den Start des ARM1-Systems verzögern kann. Somit führt ARM2 seine in Zusammenhang mit ARM1 stehenden Funktionen, wie BoxCom zu früh aus. Dies ist auch aus Abbildung 3.5 zu entnehmen.

Lösungsvorschlag

Um zumindest die beiden ARM-Systeme zu synchronisieren wäre ein einfaches Handshake denkbar. Dafür sind pro System je ein Eingangs- und ein Ausgangs-Pin nötig, welche zur Signalisierung der Zustande verwendet werden. Im folgenden soll hier ein beispielhafter Ablauf aufgezeigt werden.

1. Zu Zeitpunkt X führt ARM1 sein Modul `mod_BoxCom` aus und setzt seinen Ausgangspin auf 1
2. ARM1 wartet auf einen Highpegel auf seinem Synchronisationseingangspin
3. Wenn ARM2 sein Modul `mod_BoxCom` ausführt setzt er ebenfalls seinen Ausgangspin auf 1
4. ARM2 wartet auf einen Highpegel am Eingangspin
5. ist der Eingangspin 1, so setzt ARM2 seinen Ausgangspin auf 0 und arbeitet seine Tasks weiter ab
6. Wenn auf ARM1 am Eingangspin high anliegt, wird der Ausgang auf 0 gesetzt und ARM1 arbeitet seine Tasks ab

Die Eingangspins sollten hierbei als Interrupteingange verwendet werden, um unnötiges Pollen zu vermeiden.

Zusätzlich sollte die Initialisierung der PWM-Timer betrachtet werden, damit möglichst zeitnah auf eine Änderung der Sollwerte reagiert werden kann.

Kapitel 4

Fazit

In der vorliegenden Arbeit konnte ein Lenkwinkelgebersystem implementiert werden. Dadurch ist es nun möglich den gerade vorherrschenden Lenkeinschlag zu messen ohne das Fahrzeug bewegen müssen. Bei der Regelung wurden diverse Probleme festgestellt, welche eine befriedigende Implementierung unmöglich machen. Es wurden Lösungswege aufgezeigt um die erkannten Probleme zu beseitigen und ein stabiles Gesamtsystem zu erhalten, welches auch zeitkritische Anwendungen verarbeiten kann.

Abbildungsverzeichnis

Literaturverzeichnis

[Altmann und Schlayer 2001] ALTMANN, Siegfried ; SCHLAYER, Detlef: *Lehr- und Übungsbuch Elektrotechnik*. Fachbuchverlag Leipzig, 2001 (ISBN 3446215093)

[Blal 2007] BLAL, Bachir: *Design und Implementation eines Konzeptes zur Geschwindigkeitsbestimmung eines autonomen Fahrzeugs unter Verwendung eines AVR-Mikrokontrollers in Kombination mit Hall-Sensorik*, HAW Hamburg, Diplomarbeit, 2007

[Busch 2005] BUSCH, Peter: *Elementare Regelungstechnik Allgemeingültige Darstellung ohne höhere Mathematik*. Vogel Buchverlag, 2005 (ISBN 3834330469)

[Eickhoff 2011] EICKHOFF, Sebastian: *Ein modulares Sensor-Aktor-System für mobile Robotik*, HAW Hamburg, Diplomarbeit, 2011

[Jenning 2008] JENNING, Eike: *Systemidentifikation eines autonomen Fahrzeugs mit einer robusten, kamerabasierten Fahrspurerkennung in Echtzeit*, HAW Hamburg, Diplomarbeit, 2008

[Lutz und Wendt 2007] LUTZ, Holger ; WENDT, Wolfgang: *Taschenbuch der Regelungstechnik*. Verlag Harri Deutsch, 2007 (ISBN 9783817118076)

[Pont 2001] PONT, Michael J.: *Patterns for Time-Triggered Embedded Systems: Building Reliable Applications with the 8051 Family of Microcontrollers (with CD-ROM)*. Addison-Wesley Longman, 2001 (ISBN 9780201331387)

[Unbehauen 2005] UNBEHAUEN, Heinz: *Regelungstechnik I. Klassische Verfahren zur Analyse und Synthese linearer kontinuierlicher Regelsysteme, Fuzzy-Regelsysteme*. Vieweg Verlag, 2005

Kapitel 5

Anhang

5.1 Inhalt der CD

- Bachelorthesis als PDF Dokument
- Schlussstand der Software

5.2 Vergrößerte Abbildungen

Abbildung 5.1: Simualtionsergebnis

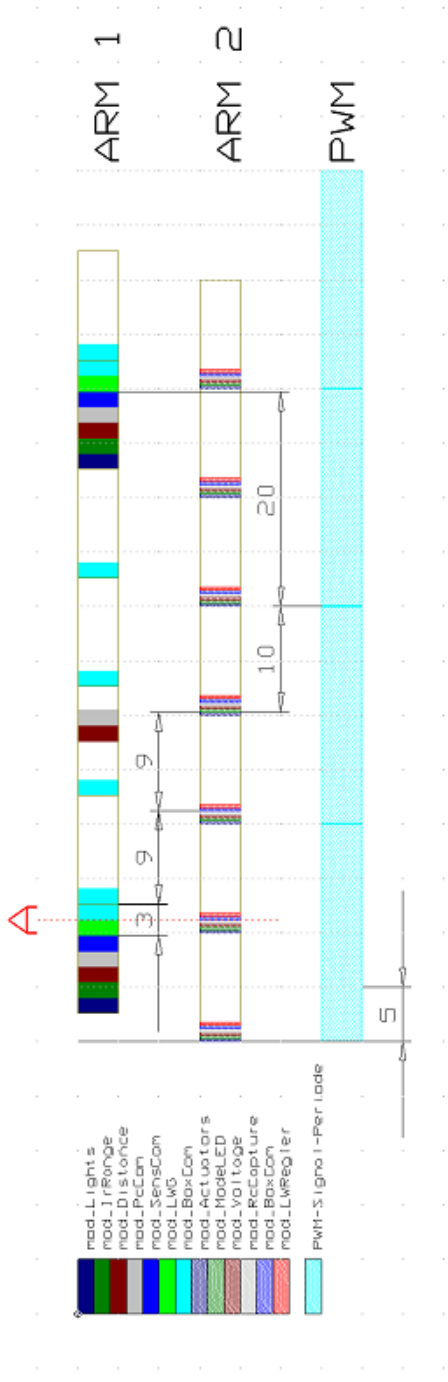

Abbildung 5.2: Timing

5.3 Einstellungen des Kontron 8021 Frequenzgenerators zur PWM-Signal-Erzeugung

Attribut	**Wert**
Frequenz	50 Hz
Amplitude	5,0 V
Offset	0,0 V
WID	$1,5 * 10^{-3}$
Output	Square positive
Mode	PULSE

5.4 Einstellung des PicoScope 3204 Oszilloskops zur Aufnahme des CAN-Busses

Attribut	**Wert**	
Kanal	A	
Eingangsbereich	5V	
Kupplung	DC	
Erfassungszeit	2ms/div	
Trigger	Wiederholen	
Triggerkanal	A	
Flanke	fallend	
Schwellwert	2V	
Werkzeuge	Serial Decoding	
	Protocols	CAN High
	Threshold	3V
	Baud Rate	250 kBaud
	Display	„In View“ und „In Window“

5.5 Einstellung des PicoScope 3204 Oszilloskops zur Aufnahme des I2C-Busses

Attribut	**Wert**	
Kanal	A	
Eingangsbereich	10V	
Kupplung	DC	
Erfassungszeit	20us/div	
Kanal	B	
Eingangsbereich	10V	
Kupplung	DC	
Erfassungszeit	20us/div	
Trigger	Wiederholen	
Triggerkanal	A	
Flanke	fallend	
Schwellwert	2V	
Werkzeuge	Serial Decoding	
	Protocols	I2C
	Clock Channel	B
	Clock Threshold	2V
	Data Threshold	2V
	Settings	8 bit Read/Write Address
	Display	„In View“ und „In Window“

5.6 Einstellung des PicoScope 3204 Oszilloskops zur Aufnahme der Spannungskurve

Attribut	Wert
Kanal	A
Eingangsbereich	10V
Kupplung	DC
Kanal	B
Eingangsbereich	2V
Sonde	x10
Kupplung	DC
Erfassungszeit	50ms/div
Trigger	Einzeln
Triggerkanal	B
Flanke	steigend
Schwellwert	1V

5.7 Code-Änderungen auf ARM2 zur Aufnahme der Spannungsmessung

5.7.1 Datei main.c

```
1  /**
2   * \file main.c
3   * \brief Main()
4   * \author Sebastian Eickhoff
5   * \anchor a_main
6   * \changed Christian Moeller
7   */
8
9  // MODULES
10 #include "mod_Voltages.h"
11 #include "mod_Actuators.h"
12 #include "mod_RcCapture.h"
13 #include "mod_ModeLED.h"
14 #include "mod_BoxCom.h"
15 #include "mod_Tempomat.h"
16
17 // SYSTEM
```

```
18 #include "main.h"
19 #include "config.h"
20 #include "init.h"
21 #include "Sch51.h"
22
23 int main(void) {
24
25     // initialize ports & functions
26     init();
27
28     // initialize scheduler
29     SCH_Init();
30
31     // initialize tasks
32     task_Actuators_Init();
33     task_ModeLED_Init();
34     task_Voltages_Init();
35     task_RcCapture_Init();
36     task_BoxCom_Init();
37     task_Tempomat_Init();
38
39     // add tasks: funtion_pointer, delay (ms), period (ms)
40     SCH_Add_Task(task_Actuators_Update, BOOT_DELAY_MSEC, 10);
41 // Modus-LED-Task abschalten
42 // Pin dient als Triggersignal zur Spannungsmessung
43 //  SCH_Add_Task(task_ModeLED_Update, BOOT_DELAY_MSEC, 10);
44     SCH_Add_Task(task_Voltages_Update, BOOT_DELAY_MSEC, 10);
45     SCH_Add_Task(task_RcCapture_Update, BOOT_DELAY_MSEC, 10);
46     SCH_Add_Task(task_BoxCom_Update, BOOT_DELAY_MSEC, 10);
47 //  SCH_Add_Task(task_Tempomat_Update, BOOT_DELAY_MSEC, 250);
48
49     // start the scheduler
50     SCH_Start();
51
52     while(1) {
53         // execute the added tasks
54         SCH_Dispatch_Tasks();
55     }
56
57     // never end up here!
58     return 0;
59 }
```

5.7.2 Datei mod_Actuators.c

```
1 /**
2  * \file mod_Actuators.c
```

```
3    *  \author Sebastian Eickhoff
4    *  \changed Christian Moeller
5    */
6
7   // MODULES
8   #include "mod_DataContainer.h"
9
10  // SYSTEM
11  #include "lpc24xx.h"
12  #include "config.h"
13  #include "IO.h"
14
15
16  data_t* pDataContainer; //!< pointer to global data container
17
18
19  void task_Actuators_Init(void) {
20
21      //---------------------------------------------------
22      // SW-INIT
23      //---------------------------------------------------
24      pDataContainer = getDataContainer();
25
26
27      //---------------------------------------------------
28      // HW-INIT
29      //---------------------------------------------------
30
31      // Setup PWM-1
32      // Prescaler: 17+1 @ 18MHz PCLK = 1MHz = 1us/CLK
33      PWM1_PR = PRESCALE_1_MHZ;
34      // match-values:
35      // MR-0: pulse-length (CLKs)
36      // MR-X: positiv pulse-duration (CLKs). PWM_NEUTRAL = 1500
37      PWM1_MR0 = PWM_PERIOD_LENGTH;
38      PWM1_MR1 = PWM_NEUTRAL;
39      PWM1_MR2 = PWM_NEUTRAL;
40      PWM1_MR3 = PWM_NEUTRAL;
41
42      // enable pwm output 1.1 & 1.2 & 1.3
43      PWM1_PCR = BIT(9) | BIT(10) | BIT(11);
44      // reset and hold
45      PWM1_TCR = BIT(1);
46
47      // enable all latches
48      PWM1_LER = 0xFF;
49      // start timer & pwm
```

```
50      PWM1_TCR = BIT(0) | BIT(3);
51
52      IOClear(0,18);
53  }
54
55  void task_Actuators_Update(void) {
56
57      switch(pDataContainer->drive_mode) {
58          case DRIVE_MODE_INIT:
59          #if 0    // Abgeschaltet zur Messung der Spannungskurve
60              PWM1_MR1 = PWM_NEUTRAL;
61              PWM1_MR2 = PWM_NEUTRAL;
62              PWM1_MR3 = PWM_NEUTRAL;
63          #endif
64              IOClear(0,18);        //Triggersignal auf 0 setzen
65              PWM1_MR1 = 1000;      //Lenkung auf Volleinschlag rechts
66              PWM1_MR2 = 1000;
67              PWM1_MR3 = 1000;
68              break;
69
70          case DRIVE_MODE_AUTONOMIC:
71          #if 0    // Abgeschaltet zur Messung der Spannungskurve
72              PWM1_MR1 = pDataContainer->motor_auto;
73              // PWM1_MR1 = pDataContainer->motor_tempomat;
74              PWM1_MR2 = pDataContainer->steer_auto;
75              PWM1_MR3 = pDataContainer->gear_auto;
76          #endif
77              IOSet(0,18);          //Triggersignal auf 1 setzen
78              PWM1_MR1 = 2000;      //Lenkung auf Volleinschlag links
79              PWM1_MR2 = 2000;
80              PWM1_MR3 = 2000;
81              break;
82
83          case DRIVE_MODE_MANUAL_F:
84          #if 0    // Abgeschaltet zur Messung der Spannungskurve
85              PWM1_MR1 = pDataContainer->motor_rc;
86              PWM1_MR2 = pDataContainer->steer_rc;
87              PWM1_MR3 = pDataContainer->gear_rc;
88          #endif
89              IOClear(0,18);        //Triggersignal auf 0 setzen
90              PWM1_MR1 = 1000;      //Lenkung auf Volleinschlag rechts
91              PWM1_MR2 = 1000;
92              PWM1_MR3 = 1000;
93              break;
94
95          case DRIVE_MODE_MANUAL_S:
96          #if 0    // Abgeschaltet zur Messung der Spannungskurve
```

```
97              if(pDataContainer->motor_rc < 1675) {
98                  PWM1_MR1 = pDataContainer->motor_rc;
99              } else {
100                 PWM1_MR1 = 1675;
101             }
102             PWM1_MR2 = pDataContainer->steer_rc;
103             PWM1_MR3 = pDataContainer->gear_rc;
104         #endif
105             IOClear(0,18);        //Triggersignal auf 0 setzen
106             PWM1_MR1 = 1000;      //Lenkung auf Volleinschlag rechts
107             PWM1_MR2 = 1000;
108             PWM1_MR3 = 1000;
109             break;
110
111         default:
112         #if 0    // Abgeschaltet zur Messung der Spannungskurve
113             PWM1_MR1 = PWM_NEUTRAL;
114             PWM1_MR2 = PWM_NEUTRAL;
115             PWM1_MR3 = PWM_NEUTRAL;
116         #endif
117             IOClear(0,18);        //Triggersignal auf 0 setzen
118             PWM1_MR1 = 1000;      //Lenkung auf Volleinschlag rechts
119             PWM1_MR2 = 1000;
120             PWM1_MR3 = 1000;
121             break;
122     }
123
124     // enable new pwm-values
125     PWM1_LER |= BIT(1) | BIT(2) | BIT(3);
126 }
```

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 13. Dezember 2011	
Ort, Datum	Unterschrift