Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Vitalij Stepanov

Mikrocontroller und CAN-basierte verteilte Regelung einer Steer-by-Wire Lenkung mit harten Echtzeitanforderungen

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Vitalij Stepanov

Mikrocontroller und CAN-basierte verteilte Regelung einer Steer-by-Wire Lenkung mit harten Echtzeitanforderungen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

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

Betreuender Prüfer: Prof. Dr. Franz Korf
Zweitgutachter: Prof. Dr. Andreas Meisel

Abgegeben am 22. August 2011

Vitalij Stepanov

Thema der Bachelorarbeit

Mikrocontroller und CAN-basierte verteilte Regelung einer Steer-by-Wire Lenkung mit harten Echtzeitanforderungen

Stichworte

Steer-by-Wire, Echtzeit, Regelung, Lenkung, Force-Feedback, Mikrocontroller, CAN, CANopen, Ethernet, TTEthernet

Kurzzusammenfassung

Der Gegenstand dieser Bachelorarbeit ist die Entwicklung einer Mikrocontroller-basierten Steer-by-Wire Lenkung. Die Vorteile wie dynamische Anpassung des Lenkungsverhaltens und geschwindigkeitsabhängige Lenkung werden untersucht und demonstriert. Mit einem Schalter lässt sich zwischen einer gewöhnlichen Lenkung für den Stadtverkehr und einer sehr direkten Sportlenkung umschalten. Eine externe Schnittstelle erlaubt die Lenkung wunschgerecht anpassen und damit jedes beliebige Fahrzeug emulieren. Mit dem Force Feedback wird die Verbindung zu der Fahrbahn hergestellt. Eine sehr flexible Architektur erlaubt den Einsatz der Regelung sowohl innerhalb eines CAN Busses als auch in einem Real-time Ethernet Netzwerk.

Title of the paper

Microcontroller and CAN-based disrtibuted Control of Steer-by-Wire steering with hard Real-time requirements

Keywords

Steer-by-Wire, Real-time, Control, Steering, Force-Feedback, Microcontroller, CAN, CANopen, Ethernet, TTEthernet

Abstract

The subject of this Bachelor thesis is the development of a microcontroller based Steer-by-Wire steering. The advantages like dynamic adaptation of the steering behaviour or steering dependent on speed will be examined and demonstrated. By the operating a switch can be choosen between a usual steering and very direct sports steering behaviour. With an external interface can emulate any vehicle steering behaviour. The Force feedback produces the connection with the road. A very adaptable architecture permits the application of the control within CAN-bus or in Real-time Ethernet network.

Inhaltsverzeichnis

Kapitel 1

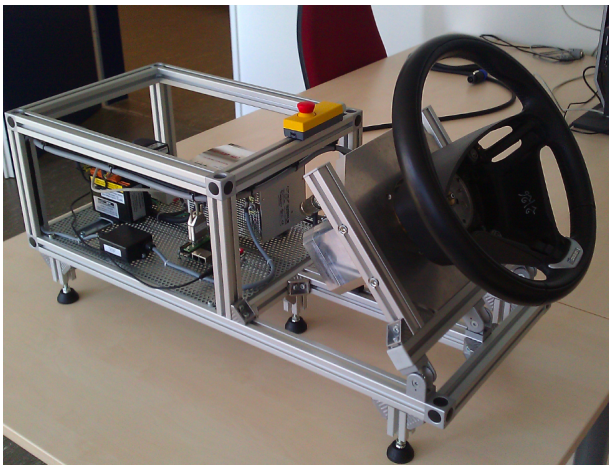
Einführung

Die modernen Autos sind ohne Bussysteme nicht mehr denkbar. Sie machen die Elektronik übersichtlicher, mindern die Kabelbaumlänge, erlauben eine schnellere Diagnose und den Zugriff auf jede Komponente von einer zentralen Stelle. Die Anzahl der Buskomponenten wächst stetig und damit auch die Menge der zu übertragenen Daten. Diese Daten stehen in starker Konkurrenz zueinander. Würden beispielsweise die Scheibenwischer die Daten der Einspritzregelung stören, würde der Motor ausgehen, wenn man die Scheibenwischer betätigt. Auch die sicherheitsrelevanten Daten wie die der Antriebsschlupfregelung (ASR), des elektronischen Stabilitätsprogramms (ESP) oder X-by-Wire[1] stellen besondere Anforderungen an die Rechtzeitigkeit der Datenübertragung. Daher werden die Nachrichten priorisiert übertragen. Da die Bandbreite der Busse limitiert ist, führen die ständig wachsenden Ansprüche der modernen Technologien die Busse an die Grenzen des Möglichen. Eine Variante zur Lösung dieses Problems ist Real-time Ethernet. Der bietet eine Bandbreite bis zu 1 Gbit/s und ist echtzeitfähig. In Rahmen des Projekts, mit dem sich das Communication over Real-time Ethernet (CoRE) Team befasst, wird die Zusammenarbeit des Real-time Ethernet Protokolls mit den im Automotivebereich etablierten Bussen wie CAN und CANopen untersucht und analysiert. Dadurch könnten bereits entwickelte Komponenten ohne weiteres in die neuen Übertragungsprotokolle eingebunden werden. Der Vorteil von Real-time Ethernet ist die sehr hohe Bandbreite, die nicht nur den Echtzeitanforderungen genügt, sondern erlaubt auch sehr große Datenmengen wie beispielsweise Multimedia zu übertragen, ohne Echtzeitfähigkeit zu beeinflussen.

Zielsetzung und Abgrenzung Im Rahmen dieser Arbeit soll eine Mikrocontroller basierte Steer-by-Wire Lösung entwickelt werden (vgl. Abschnitt 2.1). Dabei sollen die CAN bzw. CANopen basierte Komponenten wie der Lenkradmotor, der Motor der Lenkgetriebe sowie der Kraftaufnehmer in Betrieb genommen und zu einem Regelkreis geschlossen werden.

[1] X-by-Wire ist ein System, bei dem die mechanische Verbindung mit den Aktoren und Sensoren durch eine elektrische Leitung ersetzt wird

(a) Rad



(b) Lenkrad

Abbildung 1.1: Die Übersicht des Gesamtsystems

Der Controller soll die Position des Lenkrades in regelmäßigen Zeitabständen auslesen sowie den daraus resultierenden Lenkwinkel berechnen und einstellen. Auch die Querlenkkraft, die auf das Rad einwirkt, soll ermittelt und auf das Lenkrad zurückgeführt werden. Man spricht dabei von Force Feedback (vgl. Abschnitt 2.1.3). Um den Aufgabenbereich besser überblicken zu können, wird die Arbeit gegliedert und schrittweise durchgeführt. Zunächst soll die Funktionalität der Regelung nur innerhalb der CAN Topologie realisiert werden. Im zweiten Schritt werden die CAN Endgeräte, die sich in unterschiedlichen Bussen befinden, mittels einer Bridge über Real-time Ethernet zusammengeschlossen. Dafür werden die in der Abbildung 1.1 dargestellte zwei räumlich getrennten Systeme zu einem gemeinsamen System verbunden. Es soll weiterhin möglich werden, die Echtzeitverletzung zu emulieren, um damit auf die Notwendigkeit von Real-time Ethernet aufmerksam zu machen.

In diesem Zusammenhang sollen die Vorteile der Steer-by-Wire Lösungen greifbar dargestellt werden. Zu den größten Vorteilen zählt die Möglichkeit, das Lenkverhalten dynamisch zu verändern. Um auch das wahrnehmbar darzustellen, werden folgende Teilziele gesetzt:

- Es sollen unterschiedliche Modi wie z. B. normales Auto und Rennwagen über einen Schalter einstellbar sein.
- Die externe Schnittstelle zur Veränderung des Lenkverhaltens über das Infotainment Center soll ein beliebiges Fahrzeugverhalten wie z. B. Porsche oder Ferrari ermöglichen.
- Das Lenkradverhalten soll geschwindigkeitsabhängig realisiert werden und die Geschwindigkeit über ein Potentiometer einstellbar sein.

- Das entwickelte Produkt ist sicherheitsrelevant und soll daher sehr robust und ausfallsicher realisiert werden.

Da der Fokus des Projektes die echtzeitfähige Kommunikation ist, werden Aspekte wie Sicherheit durch Redundanz außen vor gelassen.

Struktur der Arbeit Das Kapitel 2 beschreibt die Grundlagen, die für das bessere Verstehen der Arbeit erforderlich sind. Dazu gehört eine kurze Einführung in die Lenksysteme bzw. Steer-by-Wire, CAN, CANopen sowie die Einführung in die Regelungstechnik. In Kapitel 3 werden die Besonderheiten der Hardware und das verwendete Betriebssystem erläutert. Kapitel 4 enthält die Beschreibung der Softwarearchitektur. In Kapitel 5 werden die Umsetzung, die Berechnung der Schedulingstabelle und die externe Schnittstelle zur Einstellung des Lenkverhaltens beschrieben. Die gestellten Anforderungen werden in Kapitel 6 mit den Ergebnissen der Umsetzung verglichen und analysiert. Kapitel 7 enthält die Zusammenfassung und die Ideen zur Verbesserung bzw. Erweiterung.

Kapitel 2

Grundlagen

In diesem Kapitel werden die für diese Arbeit relevanten Grundlagen vermittelt, die für das Verständnis der Architektur erforderlich sind. Im Abschnitt 2.1 werden die Grundlagen der Lenkung bzw. Steer-by-Wire[2] dargestellt, sowie deren Vorteile und Sicherheitsanforderungen diskutiert. Abschnitt 2.2 enthält die Beschreibung des CAN Protokolls und befasst sich mit der Problematik der Arbitrierung sowie der Priorisierung der Nachrichten. Weiterhin vermittelt der Abschnitt 2.3 die Erweiterung des CAN Protokolls zu CANopen, erklärt die Zugriffsmöglichkeiten und befasst sich mit der Notwendigkeit von Objekten. Die Grundlagen der Regelungstechnik und die Aspekte der Echtzeitanforderungen werden im Abschnitt 2.4 aufgezeigt.

2.1 Steer-by-Wire

Immer öfter setzen sich Systeme durch, in denen die mechanischen Verbindungen durch elektrische Leitungen ersetzt werden. Man fasst sie unter dem Oberbegriff „X-by-Wire“ zusammen. Es gibt eine Reihe ähnlicher Systemen wie z.B. „Drive-by-Wire“ oder auch „Fly-by-Wire“. In der Luftfahrtindustrie sind derartige Systeme sehr komplex und fortgeschritten. Sie erlauben nicht nur die Übertragung der Befehle an Aktoren, sondern können vollständig autonom handeln. Ein Beispiel dafür ist der Autopilot im Flugzeug, der schon eine breite Anerkennung und Akzeptanz verdient hat.
In einem herkömmlichen Fahrzeug wird der Winkel des Lenkrades durch eine komplizierte kinematische Verbindung an die Räder übertragen und damit die Lenkung des Fahrzeuges ermöglicht. Diese Verbindung besteht aus einer Lenksäule, die mittels eines Lenkgetriebes die radiale Bewegung in eine lineare Bewegung umsetzt und mithilfe der Spurstange den Lenkwinkel einstellt (vgl. Leiter u. a. (2008)) . Bei einer Steer-by-Wire Lenkung wird die

[2] Steer-by-Wire ist ein Lenksystem, bei dem die mechanische Verbindung mit den Aktoren und Sensoren durch eine elektrische Leitung ersetzt wird

Lenksäule zwischen dem Lenkrad und der Lenkzahnstange mit den Winkelsensoren und Motoren ersetzt. Der einfache Steer-by-Wire Aufbau ist in der Abbildung 2.1 dargestellt. Die

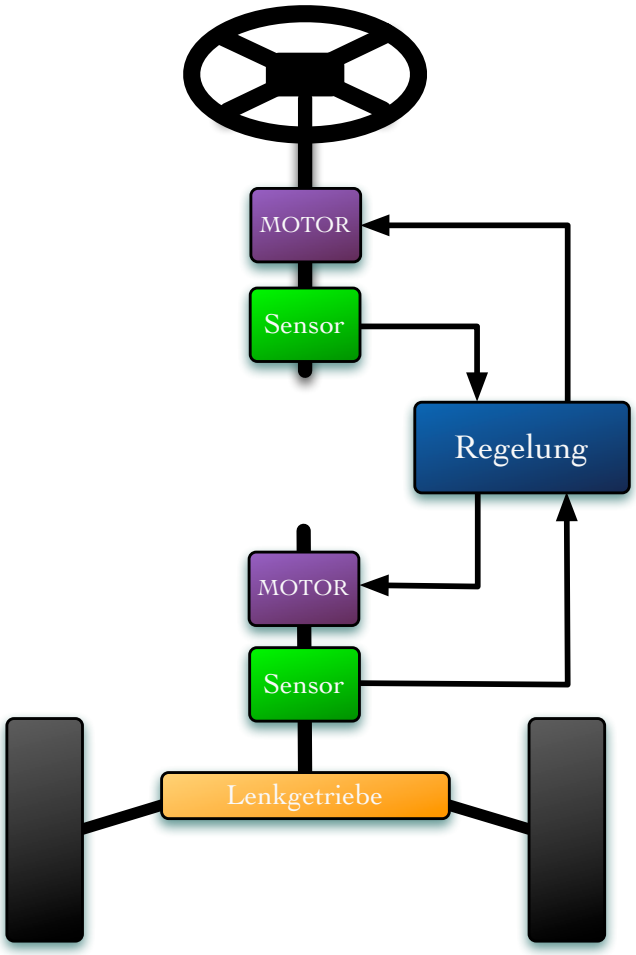


Abbildung 2.1: Prinzipieller Aufbau der Steer-by-Wire Lenkung
(nach Quelle: Institut für Arbeitswissenschaft Darmstadt, 2011)

Verbindung von Sensoren und Aktoren wird über direkte Leitungen oder Bus realisiert. Die übergeordnete Regelung ermittelt mithilfe eines Encoders den Winkel des Lenkrades und berechnet daraus den Winkel des Rades. Weiterhin erteilt er den Befehl an den Motor, den berechneten Sollwinkel einzustellen. Weicht der Istwinkel von dem Sollwinkel ab, so wird die entgegengesetzte Kraft auf das Lenkrad übertragen und damit die direkte Verbindung mit der Fahrbahn realisiert. Da sich ein auf dem Motor montiertes Lenkrad unbegrenzt drehen kann, müssen die Endanschläge, die den Lauf des Lenkrades begrenzen, entweder mechanisch oder mittels Software bzw. Regelkreis realisiert werden.

2.1.1 Beurteilung des Systems

Ein auf diese Weise entkoppeltes System bietet eine sehr hohe Flexibilität bezüglich der Platzierung des Lenkrades, weil die elektrischen Leitungen viel einfacher realisierbar sind als die mechanischen Verbindungen. Um beispielsweise eine bessere Manövrierfähigkeit des Gabelstaplers zu erreichen, befindet sich das lenkende Rad im hinteren Teil des Fahrzeuges. Daher ist die Strecke, die man mechanisch überbrücken muss, nicht nur länger, sondern auch umständlicher zu realisieren. Die Verwendung von Steer-by-Wire vereinfacht die Realisierung der Lenkung in diesem Fall wesentlich und gehört daher schon seit einiger Zeit zum Standard. Es ist ein bedeutender Schritt auf dem Weg zu den autonomen fahrerlosen Systemen. Durch das Abschalten der Verbindung zwischen dem Lenkrad und der Lenksäule lässt sich sehr einfach ein Diebstahlschutz realisieren. Das Fahrzeug ist dadurch nicht mehr lenkbar und für die Diebe nicht mehr attraktiv. Allerdings ist zu bedenken, dass der Ausfall des Diebstahlschutzes auch ein Kappen der Verbindung bewirken kann und daher als Sicherheitsrisiko anzusehen ist. Ein weiterer wesentlicher Aspekt ist die Möglichkeit, sowohl die Kraftübersetzung als auch das Lenkverhalten dynamisch anzupassen. Dadurch lässt sich die Lenkung von einem beliebigen Fahrzeug auf Knopfdruck nachahmen. Die wegfallende Lenksäule erhöht nicht nur den Komfort für den Fahrer, sondern verringert das Verletzungsrisiko bei schweren Unfällen.
Zu den negativen Aspekten zählt die Tatsache, dass beim Ausfall des Steer-by-Wire Systems das Fahrzeug nicht mehr lenkbar wäre. Um dieses Problem in den Griff zu bekommen muss die Sicherheit durch Redundanz[3] gewährleistet sein. Auf diese Problematik wird ausführlich im Abschnitt 2.1.4 eingegangen. Manchmal ist es sinnvoll, die Vibrationen des Motors am Lenkrad zu spüren, um eventuell die Fehlerursachen einfacher diagnostizieren zu können. Durch die fehlende Verbindung zum Motor werden sie aber nicht mehr wahrnehmbar.

2.1.2 Dynamisches Lenkverhalten

Zu den größten Vorteilen von Steer-by-Wire Systemen gehört die Möglichkeit einer dynamischen Anpassung der Lenkung. Dabei handelt es sich nicht nur um die Übersetzung, sondern auch um die Endanschläge des Lenkrades. Dies erlaubt eine Lenkung zu entwickeln, die bestimmte Verhaltenseigenschaften entsprechend der Situation sofort abrufbar macht. Zu diesen Eigenschaften gehört eine sportliche direkte Lenkung wie beim Rennauto oder die Lenkung im Stadtverkehr, wobei der Fahrer entlastet wird. Bei hohen Geschwindigkeiten kann eine hektische Lenkweise zum Überschlag des Fahrzeuges führen. Aus diesem Grund kann eine nichtlineare Übersetzung in Abhängigkeit von der Geschwindigkeit eingesetzt werden. Auch das Parken ist eine spezielle Situation, wobei die nichtlineare Übersetzung sinnvoll genutzt werden könnte. Auf der Abbildung 2.2 sind alle diese Situationen grafisch dar-

[3] Mit Redundanz wird die mehrfache Ausführung der sicherheitrelevanten Ressourcen beschrieben. Damit wird ein störungsfreier Betrieb ermöglicht

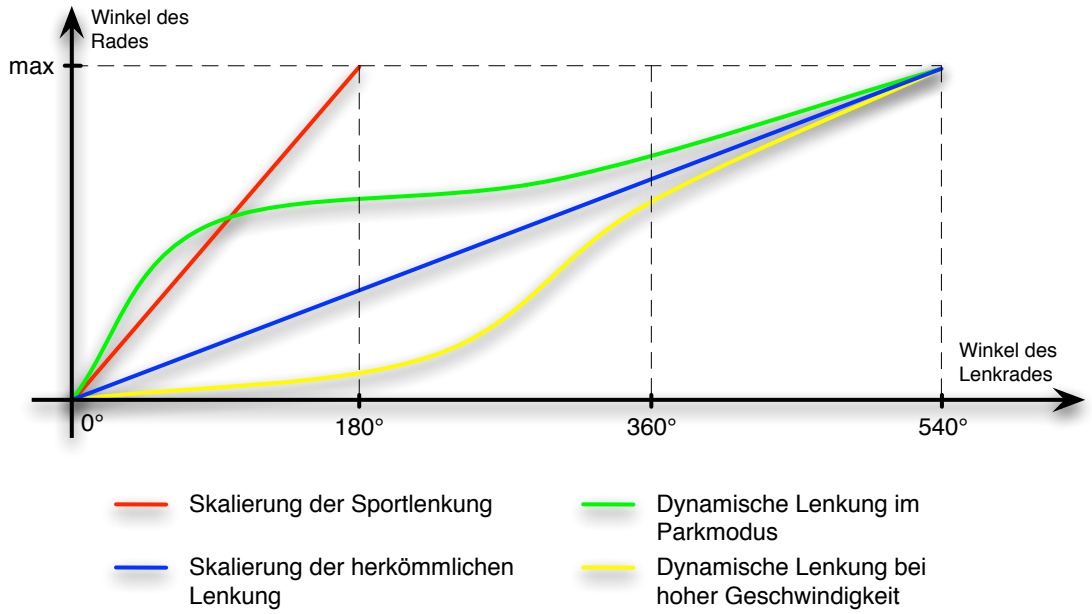


Abbildung 2.2: Grafische Darstellung von möglichen Steer-by-Wire Lenkungsverhalten und deren Skalierung auf den Lenkinterval

gestellt. Eine ideale lineare Übersetzung mechanisch zu entwickeln ist eine Wissenschaft für sich. Insbesondere bei nichtlinearen Übersetzungen ist die Engineerkunst gefragt. Dieses Problem vereinfacht sich wesentlich mit Steer-by-Wire und erlaubt, die Vorteile der Modi abhängig von der Situation oder per Knopfdruck sinnvoll auszunutzen.

2.1.3 Force Feedback

Durch den Ersatz einer mechanischen Verbindung durch elektrische Leitungen werden die Ereignisse der Fahrbahn wie Steine oder Straßenschäden für den Fahrer nicht mehr erkennbar. Um diese Verbindung herzustellen, soll die Querbeschleunigung bzw. die Querkraft mittels eines Kraftaufnehmers gemessen und auf das Lenkrad übertragen werden, so dass es vom Fahrer wahrgenommen werden kann. Eine solche Lösung wird in dem Patent von Discenzo (2000) beschrieben. Im Gegensatz zu den mechanischen Lösungen kann der Umfang der Rückmeldung dynamisch angepasst werden. So lässt sich beispielsweise die Kraft komplett ausschalten oder erhöhen. Dadurch kann die Kraftübertragung flexibel angepasst werden und es erhöht sich der Komfort für den Fahrer.
Eine weitere Vorgehensweise benötigt keinen Kraftaufnehmer und besteht darin, dass man in einem übergeordneten Steuergerät die Sollposition des Rades mit der Istposition vergleicht, um daraus Schlüsse über die Einwirkung der Gegenkraft zu ziehen. Der Nachteil dabei ist,

dass nur festgestellt werden kann, dass die Kraft in eine bestimmte Richtung gewirkt hat, aber nicht die Kraftstärke. Anhand der Differenz zwischen Soll- und Istposition des Rades kann die Sollposition des Lenkrades bestimmt und eingestellt werden.

2.1.4 Fehlertoleranz in verteilten Systemen

Bei den Systemen mit sicherheitsrelevanten Aufgaben ist die Fehlertoleranz von besonderer Bedeutung, denn der Ausfall des Systems kann gravierende Folgen mit sich bringen. Andrew S. Tanenbaum (2003) bringt die Fehlertoleranz mit weiteren Eigenschaften in Verbindung. So besagt beispielsweise die Zuverlässigkeit, dass das System während eines bestimmten Intervalls ohne Unterbrechung funktioniert. Die Zuverlässigkeit eines Systems kann man mit Mean Time Between Failures (MTBF) ausdrucken. Je höher der Wert ist, umso zuverlässiger ist das System. Unter einem weiteren wichtigen Sicherheitsaspekt verbirgt sich die Fähigkeit, bei einem vorübergehenden Ausfall dafür zu sorgen, dass nichts katastrophales passiert. Besonders für Fahrzeuge, die gemäß Straßenverkehrs-Zulassungs-Ordnung (StVZO) für den Verkehr zugelassen werden, hat ein Ausfall der Lenkung nicht nur Auswirkungen für die Insassen, sondern auch für die anderen Verkehrsteilnehmer sowie die Fußgänger. Daher schreibt das Bundesministerium der Justiz (2011) im § 38 der StVZO vor, dass: „Bei Versagen der Lenkhilfe muss die Lenkbarkeit des Fahrzeuges erhalten bleiben“. Da ein Steer-by-Wire System auch als Lenkhilfe anzusehen ist und keine mechanische Verbindung existiert, muss dieser Aussage besondere Beachtung geschenkt werden.

Sicherheit durch Redundanz

Um die Lenkbarkeit auch bei Ausfällen zu gewährleisten, werden in sehr kritischen Systemen die ausfallkritischen Teile mehrfach (redundant) platziert. Die Patente von Hommel (2001) und Ewbank (2003) beschreiben derartige Ansätze. Zu den ausfallkritischen Teilen gehören nicht nur Aktoren und Sensoren, sondern ebenso die Übertragungsmedien und die Spannungsversorgung. Jede Strecke enthält ein separates Steuergerät, der die Funktionalität der Lenkung realisiert. Ein solcher Ansatz ist in der Abbildung 2.3 übersichtlich dargestellt. Das Steuergerät überwacht die Zustände von Aktoren und Sensoren, um einen Ausfall feststellen zu können. Dafür wird eine Logik benötigt, die beide Steuergeräte überwacht und den ausgefallenen Regelkreis abschaltet. Sollte eine Regelkreis eine Störung aufweisen, so wird der Fahrer mittels eines akustischen oder visuellen Signals gebeten, die Werkstatt aufzusuchen und die Störung beseitigen zu lassen. Um reproduzierbare Fehler auf der Softwareebene zu vermeiden, sollen die Algorithmen unterschiedlich implementiert werden. Man spricht dabei von Diversität. Werden die Algorithmen einfach kopiert, so bedeutet das, dass wenn der erste Algorithmus einen Fehler aufweist, so betrifft dies auch den zweiten. Dies führ zu dem Ausfall von beiden Regelkreisen und stört damit die Funktionalität der Lenkung.

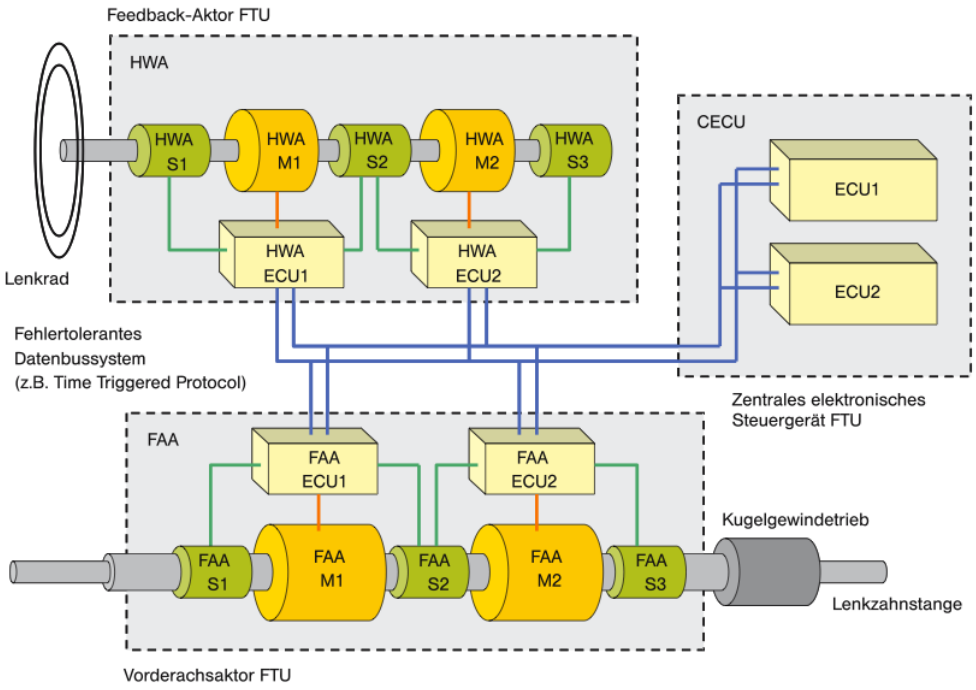


Abbildung 2.3: Redundante Steer-by-Wire Lenkung
(Quelle: Wallentowitz und Reif, 2006)

2.2 CAN

Unter Controller Area Network (CAN) verbirgt sich ein Kommunikationsprotokoll, das im Automotivebereich weit verbreitet ist und sich seit Jahrzehnten durchgesetzt hat. Das Protokoll ist von der International Organisation for Standardization (2003) in ISO 11898 1-3 spezifiziert und verfügt über eine die Übertragungsrate bis zu 1 Mbit/s. Das CAN Protokoll implementiert Schicht 1 und 2 des OSI Referenzmodells und realisiert damit eine zuverlässige Übertragung. Es handelt sich um ein ereignisgesteuerts Kommunikationsprotokoll. Die Nachrichten, die auf dem Bus landen, können von allen Teilnehmer gelesen werden. Jeder Teilnehmer entscheidet für sich, ob die Nachricht für ihn relevant ist oder nicht. Außer den Nutzdaten enthält das CAN Telegramm weitere Bits, die den Anfang bzw. das Ende des Frame markieren, sowie 15 Bits lange Cyclic Redundancy Check (CRC) und weitere Bits, die zwecks Übersichtlichkeit an dieser Stelle nicht beschrieben werden. Eine ausführliche Beschreibung kann man aus Zimmermann und Schmidgall (2008) oder Zeltwanger (2001) entnommen werden. Die Framelänge bei einem 11 Bit Identifier beträgt max. 111 Bit. Um die Synchronisation bei gleichpoligen Bits durchführen zu können, wird Bit Stuffing angewendet. Dafür wird nach fünf gleichpoligen Bits ein komplementäres Bit eingefügt. Daher kann die Länge

des Frames abhängig von der Payload[4] bis zu 19 Bit länger werden. Aus der nominalen Bitzeit von 1 μs resultiert die Gesamtübertragungszeit eines Telegramms von 130 μs. Um Reflektionen bei der Übertragung zu vermeiden, sollen beiden Enden des Busses mit den Abschlusswiderständen von 120 Ω terminiert werden.

2.2.1 Nachrichtenaufbau

Jede CAN Nachricht wird durch den Identifier eindeutig und kann bis zu 8 Byte Nutzdaten enthalten. Dabei wird nicht das Gerät gekennzeichnet, sondern die Nachricht selbst. Anhand des Identifiers kann entschieden werden, um welche Art der Nachricht es sich handelt. Weiterhin beinhaltet die Nachricht die Länge der enthaltenen Daten im Wertebereich $n \in 0 \ldots 8$. Die Abbildung 2.4 beschreibt den allgemeinen Aufbau eines CAN Telegramms. Der Identi-

Abbildung 2.4: Allgemeine Bescheibung einer CAN Nachricht

fier ist in Blau dargestellt. Das gelbe Kästchen enthält die Anzahl der Nutzdaten, die mittels nachfolgendem grauen Kästchen dargestellt werden. Das erste Byte enthält dabei den Least Significant Byte (LSB) und das letzte den Most Significant Byte (MSB). Dies bedeutet, dass die Byteorder des CAN Protokolls Big Endian ist. Eine Nachricht kann bis zu 8 unterschiedliche Parameter enthalten. Ebenfalls sind die Parameter, die aus mehreren Bytes bestehen, zulässig. Dabei werden die LSB und MSB für jeden Parameter separat betrachtet und müssen entsprechend interpretiert werden. Abhängig von der Byteorder des Zielsystems sollen möglicherweise die Bytes gedreht werden. Die Abbildung 2.5 enthält ein CAN Telegramm

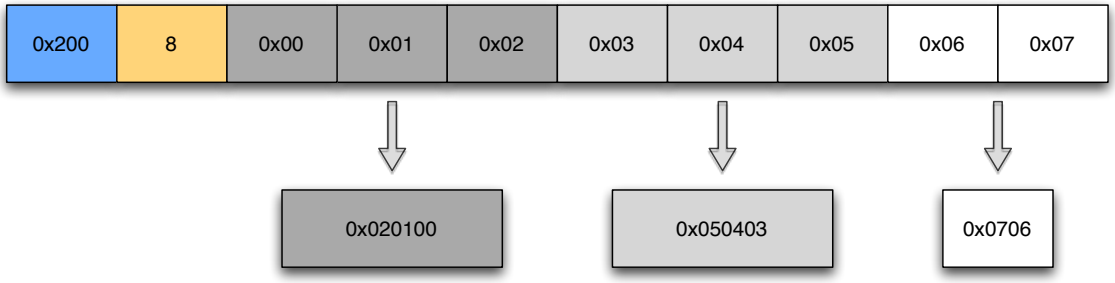


Abbildung 2.5: Interpretation von zusammenhängenden Daten

[4] Payload bezeichnet die Nutzdaten, die übertagen werden sollen

mit der Länge von 8 Byte. Die Daten werden logisch auf 3 Parametern aufgeteilt. Der erste und der zweite Parameter bestehen jeweils aus 3 Bytes, der dritte aus 2 Bytes. Bei diesem Beispiel handelt es sich beim Zielsystem um Little Endian, daher sollen die Bytes innerhalb einer logischen Trennung geswappt werden.

2.2.2 Arbitrierung und Priorisierung der Nachrichten

Das CAN Protokoll nutzt das Carrier Sense Multiple Access / Collision Resolution (CSMA/-CR) Übertragungsverfahren. Das heißt, dass man in der Lage ist nicht nur die Kollisionen zu erkennen, sondern auch die zerstörungsfreie Arbitrierung[5] durchzuführen. Um die Kollisionen in einem Multimaster System erkennen zu können, wird zwischen dem dominanten und rezessiven Pegel unterschieden. Der dominante Pegel entspricht einem logischen Wert '0' und der rezessive einer '1'. Konkurrieren zwei Master um den Bus, so zerstört ein dominanter Pegel den rezessiven und damit erlangt die kleinere ID den Buszugriff. Daraus folgt, dass die kleinere ID immer die höchste Priorität hat. Merkt anderseits der Sender, dass die Daten auf dem Bus nicht mit seinen Solldaten übereinstimmen, so stellt er fest, dass er den Buszugriff verloren hat und entscheidet sich zurückzutreten, um die Daten der hoch priorisierten Nachricht nicht zu zerstören.

2.3 CANopen

Bei dem CANopen Protokoll handelt es sich um einer Erweiterung des CAN Protokolls. Dieses ist von CAN in Automation (CiA) (2011) international standardisiert und speziell für die eingebetteten Systeme entwickelt worden. Die Implementierung des CANopen Protokolls reicht bis zur Anwendungsschicht des OSI Referenzmodells und schreibt daher jedem der Datenbytes eine Bedeutung vor. Man unterscheidet zwischen einer quittierten Service Daten Objekt (SDO) und der schnelleren nichtquittierten Prozess Daten Objekt (PDO) Kommunikation. Die jeweiligen Vorteile werden in den Abschnitten 2.3.3 Kommunikation und 2.3.2 vorgestellt. Ein wesentlicher Unterschied zum CAN Protokoll ist, dass man nicht nur zwischen den Nachrichtenidentifier unterscheidet, sondern noch einen Mechanismus zur Unterscheidung der Knotenadresse zur Verfügung hat.

2.3.1 CANopen Identifier

Eine CAN Nachricht wird als Communication Object (COB) bezeichnet. Um die Adressierung der verschiedenen Geräte zu gewährleisten, werden der Geräte-ID die kleinsten 4 Bit zugeteilt. Die Adresse 0 ist für das Network Managment (NMT) reserviert, daher können

[5] Die Arbitrierung ist ein Verfahren zum Lösen der Zugriffskonflikten beim Zugriff mehreren Nutzer auf eine gemensame Ressource

Objekt	COB-ID
NMT	0x00
SYNC	0x80
Tx-PDO1	0x180 + Knotenadresse
Rx-PDO1	0x200 + Knotenadresse
Tx-SDO	0x580 + Knotenadresse
Rx-SDO	0x600 + Knotenadresse

Tabelle 2.1: Die von CANopen reservierte COB-IDs und deren Semantik.

nur bis zu 15 Geräte unterscheiden werden. Natürlich muss die Geräte-ID innerhalb eines Busses eindeutig sein. Die Communication Object Identifier (COB-ID) setzt sich aus der Summe von COB und der Geräte-ID zusammen. In der Tabelle 2.1 sind die Communications Objekte (COB) aufgelistet, die für diese Arbeit von Bedeutung sind. Auch hier werden die Nachrichten entsprechend der im Abschnitt 2.2.2 vorgestellten Arbitrierung bzw. dem Priorisierungsverfahren gehandhabt. Das bedeutet, dass auch an dieser Stelle die Nachrichten mit der niedrigsten COB-ID die höchste Priorität besitzen.

2.3.2 SDO Zugriff

Bei einem SDO Zugriff wird jede Nachricht immer von dem Empfänger quittiert. In einigen Fällen ist das sehr vorteilhaft, anderseits bringt es eine Busbelastung mit sich. In dem folgenden Beispiel wird der SDO Zugriff dargestellt. Die Geräte-ID des Zielknotens ist 1. In der Abbildung 2.7 ist eine ausgehende Nachricht aufgezeichnet, die an das entsprechende Gerät gesendet werden soll. Mittels eines Commands (CMD) wird die Art des Zugriffes

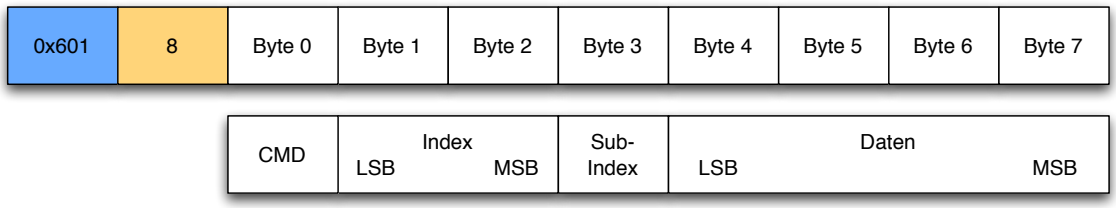


Abbildung 2.6: Aufteilung einer CANopen Rx Nachricht in Parameter

festgelegt. Bei einem Lesezugriff muss die Länge der Daten nicht spezifiziert werden. Man erhält sie in der Antwortnachricht an derselben Stelle. Bei einem schreibenden Zugriff legt die kleinste hexadezimale Zahl des CMD die Länge der Daten fest. Nachfolgend werden alle möglichen Zugriffsarten für die Tx Nachrichten aufgelistet:

```
0x40 Lesezugriff
0x23 Schreiben 4 Bytes
0x2B Schreiben 2 Bytes
0x2F Schreiben 1 Byte
```

In dem Index-Segment wird die Nummer eines Objekts angegeben, das angesprochen bzw. gelesen werden soll. Die Objekte sind standardisiert und werden von CAN in Automation e. V. (CiA) geleitet. Außerdem enthält die Nachricht einen Subindex, der als Parameter eines Objekts anzusehen ist. Die Anzahl von verfügbaren Subindexen ist hersteller- bzw. implementierungsspezifisch der Dokumentation des Gerätes zu entnehmen. Die Abbildung 2.7 enthält die Antwortnachricht, mit der der Empfänger den Erhalt der Nachricht quittiert. Im

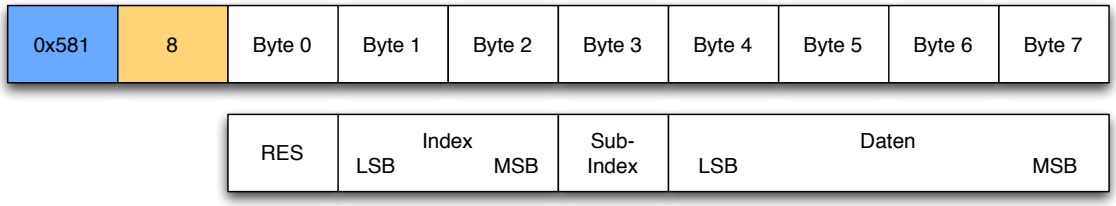


Abbildung 2.7: Aufteilung einer CANopen Tx Nachricht in Parameter

Gegensatz zu der Anfrage enthält das erste Byte der Payload der Antwort den Response (RES), indem die angeforderte Operation bestätigt oder als fehlerhaft markiert wird. War der Zugriff fehlerhaft, so enthält das Datensegment eine 4 Byte lange CANopen konforme Fehlernummer. Die möglichen RES Werte werden nachfolgend aufgelistet.

```
0x60 Daten erfolgreich geschrieben
0x80 Fehler, Datensegment enthält eine Fehlernummer
0x43 4 Bytes empfangen
0x4B 2 Bytes empfangen
0x4F 1 Byte empfangen
```

Bei einer lesenden Anfrage enthält auch hier die niedrigste hexadezimale Zahl der RES die Anzahl der versendeten Daten. Mit der zweiten Zahl wird der lesende Zugriff kenntlich gemacht. Damit man die Möglichkeit hat, die Herkunft der Daten verifizieren zu können enthält auch die Antwortnachricht die Objekt-ID und den Subindex. Damit kann verhindert werden, dass die Daten versehentlich anders interpretiert werden.

2.3.3 PDO Zugriff

Anders als bei SDO hat der PDO Zugriff keine Bestätigung. Dadurch wird nicht nur der Bus entlastet, sondern die Reaktionszeit des Systems wird ebenfalls kürzer. Außerdem lassen

sich die Ereignisse, auf denen die Auswertung bzw. das Versenden der Daten ausgeführt werden kann, unterschiedlich einstellen. Man kann die PDO so einstellen, dass auf die Veränderung des Wertes reagiert wird. Auf die zweite Art, das Versenden auf die SYNC Nachricht, wird im Abschnitt 2.3.5 eingegangen. Ein weiterer, wesentlicher Vorteil von PDO ist die Möglichkeit, mittels nur einer Nachricht mehrere Parameter gleichzeitig zu setzen bzw. abzurufen und in einer Nachricht versenden zu können. Dabei entfällt der im SDO vorhandene Overhead wie Zugriffsart, Objekt-ID und Subindex. Dadurch lässt sich das zu übertragene Datenvolumen effizienter verwenden. Damit die PDO genutzt werden können, muss dem Endgerät mitgeteilt werden welche Parameter die Quelle bzw. das Ziel sind. Diesen Mechanismus nennt man PDO Mapping.

2.3.4 PDO-Mapping

Sowohl für die Tx (Objekt 0x1800) als auch für Rx (Objekt 0x1400) sollen die Kommunikationsparameter eingestellt werden. Dazu gehören der Identifier (Subindex 1), die Art der Reaktion (Subindex 2) und die Inhibit-Zeit (Subindex 3). Mittels Inhibit-Zeit wird der minimale Abstand zwischen dem Absenden der PDO festgelegt. Eine PDO Nachricht kann 1 bis 8 unterschiedliche Parameter enthalten. In den Objekten 0x1600 für Rx und 0x1A00 für Tx werden die Parameter für PDO1 eingetragen. Der Subindex 0 legt die Anzahl der Parameter fest, in den nachfolgenden Subindexen werden die jeweiligen Parameter gemappt. Dafür werden der Objektindex, Subindex und die Länge des Parameters in Bits zu einem 4 Byte langen Wert zusammengefasst und im Datensegment übertragen. Beispielweise beim Mappen eines 16 Bit (hexadezimal 0x10 Bit) langen Objekts 0x6040, Subindex 1 resultiert ein Wert 0x6040 01 10. Mittels einer NMT Nachricht wird das Gerät aus dem „Preoperational Mode“ in den „Operational Mode“ versetzt und damit PDO aktiviert. Die NMT Nachricht enthält 2 Datenbytes. Im ersten Byte wird das Kommando zum Starten bzw. zum Stoppen angegeben, das zweite Byte enthält die Adresse des Knotens, der angesprochen wird. Nachdem der Knoten aktiviert wurde, werden die Nachrichten erwartet bzw. entsprechend der Konfiguration versendet. Die dabei entstehenden Nachrichten werden je nach Konfiguration ähnlich der Abbildung 2.5 interpretiert.

2.3.5 Synchronisierte Datenzugriffe

Sehr oft ist man in verteilten Systemen gezwungen, mehrere Endgeräte zum selben Zeitpunkt zu parametrieren bzw. Daten holen zu müssen. Dabei fällt eine iterative Variante aus, weil durch die nacheinander folgenden Nachrichten Verzögerungen entstehen. Für die synchronisierten Zugriffe sieht das CANopen Protokoll einen Konzept vor. Mittels einer SYNC Nachricht, die alle Knoten gleichzeitig erreicht, wird die Übernahme bzw. das Versenden der Daten angeregt. Der Zeitabstand zwischen zwei SYNC Nachrichten entspricht einer Periode.

Üblicherweise werden in der ersten Hälfte der Periode die Antworten der gemappten Parameter erwartet. In der zweiten Hälfte werden die Parameter versendet, die mit der SYNC Nachricht übernommen werden sollen. Die SYNC Nachricht hat den Identifier 0x80 und damit eine sehr hohe Priorität. Weiterhin enthält sie keine Daten und ist daher nur 47 Bit lang.

2.4 Regelungstechnik

Bereits seit längerer Zeit gehören die Regelprozesse zu unserem alltäglichen Leben. Sie sind so gut integriert, dass sie nicht mehr als solche erkannt werden. Beispielweise stellt eine Heizungsanlage einen Regelungsprozess dar, der die vorgegebene, gewünschte Temperatur mit der tatsächlich, gemessenen vergleicht. Anhand der Abweichung wird entschieden, wie weit das Ventil des Heizungskörpers geöffnet werden soll, um die gewünschte Temperatur schnell wie möglich zu erreichen und konstant zu halten. Ein weiteres Beispiel mit dem selben Prinzip ist die Klimaautomatik, bei der ebenfalls die Temperatur erfasst wird, mit dem Unterschied, dass die Kaltluftzufuhr geregelt wird. Auch eine sehr einfache Sache, die wir täglich benutzen, ist der Spülkasten der Toilette, indem die Wasserzufuhr mechanisch geregelt wird. Vor allem in der Industrie und Automatisierungstechnik ist die Regelungstechnik von besonderer Bedeutung. So wird beispielsweise in der Lebensmittelindustrie die Menge der Komponenten vollständig automatisch geregelt und überwacht.

2.4.1 Regelkreis

Bei einem Regelkreis handelt es sich um einen technischen Prozess, der mittels regelmäßiger Beobachtung der Eingangsgröße die physikalischen Ausgangsgrößen beeinflusst. Das charakteristische Merkmal eines Regelkreises ist die negative Rückführung der Regelgröße, die den Vergleich des Istwertes mit dem Sollwert ermöglicht. Die Differenz bildet die Regelabweichung $e(t)$. Aus der damit bestimmten Regelabweichung $e(t)$ berechnet der Regler die Stellgröße $u(t)$. Aus der Einwirkung der Störgröße auf die Stellgröße resultiert der Istwert. Der in der Abbildung 2.8 dargestellte Regelkreis ist von der Zeit abhängig und stellt damit ein kontinuierliches System dar. Es gibt eine Reihe von Parametern, die erlauben das Verhalten des Regelkreises zu analysieren. Die Totzeit[6] charakterisiert die Verzögerung zwischen der Führungsgröße und der Stellgröße. Manchmal werden auch die Begriffe Laufzeit oder Transportzeit verwendet. Eine lange Totzeit beeinflusst das Regelverhalten des Regelkreises negativ. Daher soll darauf geachtet werden, die Totzeit kurz wie möglich zu halten. Eine Sehr ausführliche Beschreibung ist im Buch von Mann u. a. (2009) vorhanden.

[6] Der Begriff Totzeit T_T steht für die Zeit, die von der Änderung am Eingang bis zur Änderung am Ausgang vergeht

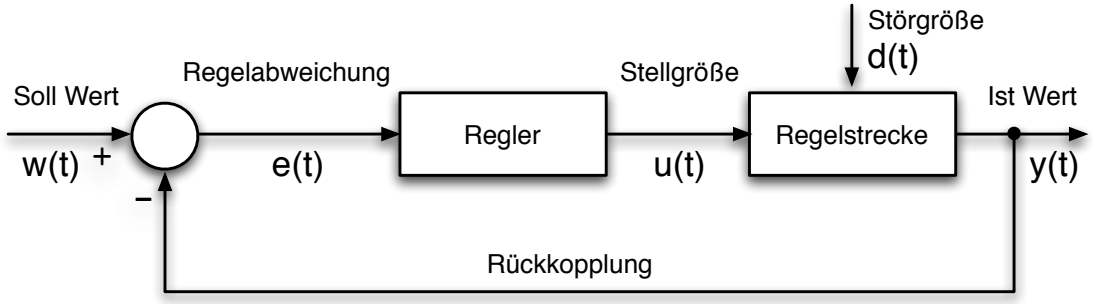


Abbildung 2.8: Grafische Darstellung eines einfachen Regelkreises (nach Quelle: Mann u. a. (2009))

2.4.2 Fahrer als Regler

Von Geburt an wendet der Mensch die Prinzipien der Regelungstechnik unbewusst an. Durch Vergleich und Beobachtung entscheidet der Mensch, ob er das vorgegebene Ziel erreicht hat und versucht gegebenenfalls darauf zu reagieren. So erlernt er im Laufe des Lebens von anfänglich einfachen Abläufen wie die Hand hochheben oder strecken bis zu komplexen Abläufen wie das Laufen oder Schreiben. Auch Fahrrad fahren ist ein sehr komplexer Regelkreis, bei dem die menschlichen Wahrnehmungen, wie das Gleichgewicht oder räumliche Orientierung, in mechanische Bewegungen, wie das Lenken oder die Gewichtsverlagerung, umgesetzt werden.

2.4.3 Echtzeitanforderungen in der Regelungstechnik

Unter dem Begriff Echtzeit versteht man, dass die Dauer eines Vorgangs bis zur Deadline[7] abgeschlossen und daher vorhersagbar ist. Die Geschwindigkeit ist dabei unerheblich, viel wichtiger ist die Rechtzeitigkeit. Bei den Echtzeitsystemen unterscheidet man die weiche Echtzeitanforderung[8] und harte Echtzeitanforderung[9] (vgl. Tanenbaum (2009)). Das System mit dem weichen Echtzeitanforderungen kann durch nicht berechenbare Vorgänge wie Speicherauslagerung, Caching, oder Hardwarenterrupts verzögert werden und damit die Deadline überschreiten. Bei der Verletzung der Deadline hat es keine Konsequenzen, daher kann die Anwendung weiterlaufen. Im Gegensatz zu der weichen Echtzeitanforderungen muss das System mit dem harten Echtzeitanforderungen die Vorhersagbarkeit des Prozess-

[7] Deadline ist ein bestimmter Zeitpunkt, zu dem eine Aufgabe erledigt werden muss. Die Verletzung der Deadlin hat gravierende Auswirkungen auf das System

[8] Weiche Echtzeitanforderung bedeutet, dass die Dauer eines Vorgangs die angegebene Obergrenze üblicherweise nicht überschreitet

[9] Harte Echtzeitanforderung bedeutet dass anhand von Hardwarespezifikationen und Modellrechnungen eine beweisbare obere Grenze für die Dauer eines Vorgangs angegeben werden kann

verhaltens bei Volllast und in anderen Ausnahmesituationen garantieren. Daher kann das nicht Einhalten der Deadline gravierende Folgen haben.

Zum Beispiel verbraucht ein Prozess mehr Zeit, als ihm zugeteilt wurde, so wird er von einem anderen Prozess unterbrochen, unabhängig davon, ob er den Vorgang bzw. die Berechnung abgeschlossen hat. So rechnet der nachfolgende Prozes mit den ungültigen Werten weiter.

Die Regelung gehört zu den kritischen Anwendungen, weil sie in definierten Zeitintervallen die Daten der Sensoren verarbeitet und die Aktoren mit den errechneten Daten versorgt. Kommt eine Verletzung der Deadline zustande, so wird beispielsweise die Berechnung der Daten nicht abgeschlossen, ein Prozesswechsel findet statt und der Aktor wird mit den ungültigen Daten beliefert. Dies kann zu einem Absturz oder im schlimmsten Fall zur mechanischen Beschädigung des Systems führen.

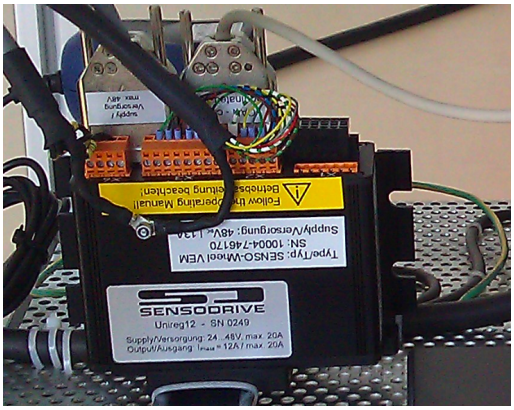
Kapitel 3

Hardware

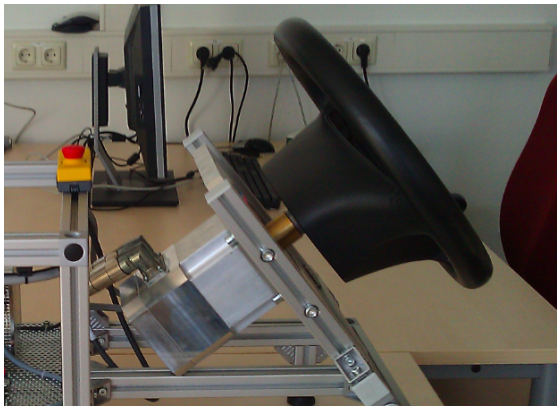
Dieses Kapitel beschreibt die in vorliegenden Arbeit verwendete Hardware. Dazu wird auf die einzelnen Endgeräte eingegangen und die grundlegende Funktionalität erklärt. Weiterhin wird der Mikrocontroller erläutert, der als Zielsystem für die Entwicklung der Steer-by-Wire Lenkung zum Einsatz kommt und die Endgeräte des gesamten Systems steuert.

3.1 SENSO-Wheel Lenkrad

Bei diesem Endgerät handelt es sich um ein Lenkrad, das die Funktion einer Fahrzeuglenkung nachahmt. Die in der Abbildung 3.1 (a) dargestellte Regelung realisiert bereits das grundlegende Lenkungsverhalten wie die Endanschläge bzw. Dämpfung, Reibung und Federsteifigkeit. Die Regelung steuert die Bewegungen des Motors, auf dem das Lenkrad montiert ist (siehe Abbildung 3.1 (b)). Mittels des CAN Protokolls kann die Regelung parametriert



(a) Regelung des Lenkrades



(b) Motor und darauf befestigte Lenkrad

Abbildung 3.1: Die Komponenten des Lenkrades

bzw. der Winkel des Lenkrades ausgelesen werden. Dieser Abschnitt soll nur die grundlegende Funktionalität aufzeigen, die zum Verständnis der Funktionsweise beiträgt. Die detaillierte Beschreibung des Gerätes ist den Ausführungen der SensoDrive GmbH (2010) zu entnehmen.

3.1.1 CAN Watchdog

Die Steuerung des Lenkrades verfügt über einen CAN Watchdog, der das Lenkrad in den Fehlerzustand überführt, wenn innerhalb von 50 ms keine Nachricht empfangen wurde. Deren Art ist nicht festgelegt. Jede empfangene Nachricht setzt den Watchdog zurück. Sollte der Watchdog auslaufen, so geht der im Abschnitt 3.1.3 beschriebene Zustandsautomat in den Fehlerzustand.

3.1.2 Nachrichten

Es handelt sich um einer reinen CAN Kommunikation. Die Interpretation der Parameter wird entsprechend der Abbildung 2.5 realisiert. Mit der Nachricht 0x200 wird die Regelung des Lenkrades gesteuert. Das erste Byte enthält das Steuerwort, mit dem der Zustandsautomat aus dem Abschnitt 3.1.3 gesteuert wird. Das LSB Nibble[10] enthält den Zustand des Automaten, mit dem MSB Nibble wird der Modus des Lenkrades spezifiziert. Innerhalb dieser Bachelorarbeit werden der Normalmodus (0x1) und der Referenzfahrt Modus (0x4) verwendet. Mit den Bytes 3 und 4 wird die Position der Endanschläge gesetzt. Byte 7 legt die prozentuale Kraft innerhalb der Endanschlägen fest. Mit dem Byte 8 wird die prozentuale Kraft außerhalb der Endanschläge gesetzt. Die Nachricht 0x210 enthält in den Bytes 1 und 2 die Bestätigung des Status der Regelung. Befindet sich der Zustandsautomat im Fehlerzustand, dann enthalten die Bytes 3 und 4 den Fehlercode. Mit der Nachricht 0x201 wird das Verhalten der Lenkung parametriert. Bytes 1 und 2 setzen den Sollmoment, damit ist das Force Feedback realisiert. Die Reibung wird mit den Bytes 3 und 4 eingestellt. Dämpfung befindet sich in den Bytes 5 und 6. In den Bytes 7 und 8 ist die Federsteifigkeit untergebracht. Nachricht 0x211 liefert die Position (1,2,3,4) die Geschwindigkeit(5,6) der Drehung und den Istmoment(7,8). Die Regelung berechnet aus den Parametern den Motormoment nach der Formel:

Motormoment = Sollmoment + Reibung + Dämpfung * Geschwindigkeit + Federsteifigkeit * Position

Daran wird die Abhängigkeit der Dämpfung von der Geschwindigkeit und der Federsteifigkeit von der Position deutlich.

[10] Ein Nibble ist eine 4 Bit große Datenmenge, sie wird meistens mit hexadezimalen Zahlen dargestellt

3.1.3 Zustandsautomat

Die Regelung des Lenkrades wird von dem in der Abbildung 3.2 dargestellten Zustandsautomaten kontrolliert. Durch das Steuerwort der Nachricht 0x200 werden die Zustandstransitionen ausgeführt und der Modus angegeben (vgl. Abschnitt 3.1.2). Der Zustandsautomat ist in

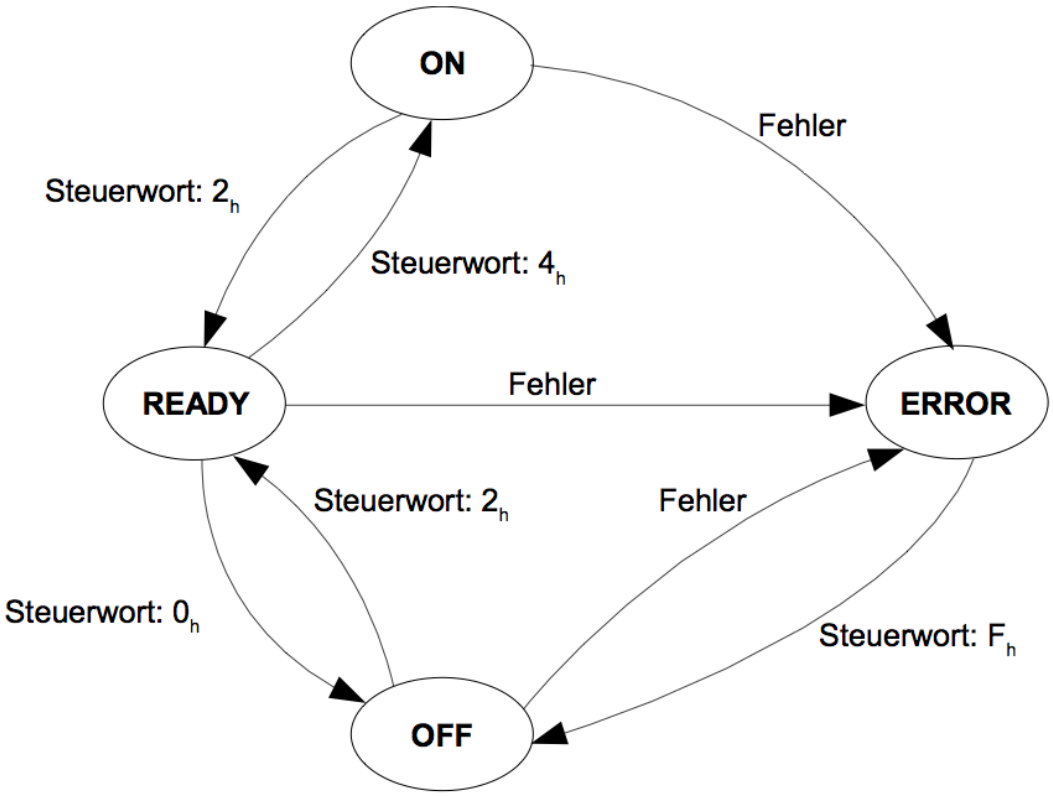


Abbildung 3.2: Zustandsautomat der Lenkrad-Regelung
(Quelle: SensoDrive GmbH (2010))

jedem Modus gleich und befindet sich nach dem Einschalten oder nach dem Moduswechsel im Zustand *OFF*. Um die Regelung zu aktivieren muss der Zustandsautomat in den Zustand *ON* geführt werden. Mit dem Steuerwort 0x2 wird der Zustand *READY* erreicht. Erst nach der Bestätigung der Zustandstransition in der Nachricht 0x210 kann zum Zustand *ON* gewechselt werden. Anschließend soll das Steuerwort 0x4 gesendet und damit der Automat in den Zustand *ON* gebracht werden. Aus den Zuständen *OFF*, *READY* und *ON* beim Auftreten eines Fehlers wird in den Zustand *ERROR* gewechselt. Mit dem Steuerwort 0xF wird der Fehler quittiert und der Zustand *OFF* erreicht.

3.1.4 Referenzfahrt

Nach dem Einschalten übernimmt die Regelung die Einschaltposition als die Position Null. Mit der Referenzfahrt kann das Lenkrad in eine richtige Initialposition gebracht werden. Das Lenkrad verfügt über einen Index Sensor, nachdem die Referenzierung sucht. Die Referenzfahrt dreht das Lenkrad im Uhrzeigersinn bis der Index-Sensor erreicht ist. Mit dem Offset kann nach dem Referenzieren eine beliebeige Position eingefahren werden. Dies ist dann nötig, wenn die gewünschte Nullposition nicht mit der Position des Index-Sensors übereinstimmt. Die Nachricht 0x200 enthält in den Bytes 5 und 6 das Positionsoffset, das angibt, um wie viel Grad das Lenkrad zurückgedreht werden soll. Diese Position wird als die neue Nullposition übernommen. Wenn das zweite Byte der Nachricht 0x210 den Wert 0x68 aufweist, so ist die Referenzfahrt zu Ende. Danach kann die Regelung in den Normalmodus wechseln und den Motor einschalten.

3.1.5 Analoger Eingang

Um das geschwindigkeitsabhängige Lenkverhalten zu realisieren, benötigt man einen analogen Eingang. Die Regelung des Lenkrades verfügt über diesen. Ein Potentiometer wird die Geschwindigkeit des Fahrzeuges simulieren. Das Potentiometer braucht eine Spannungsversorgung von 4,8 Volt. Diese kann dem Encoder entnommen werden. Damit der Potentiometer nicht zu große Ströme auf sich zieht und damit Spannungsausbrüche an der Regelung verursacht, muss das Potentiometer mindestens 1 kΩ groß sein. Andernfalls kann die Regelung beschädigt werden. In der Abbildung 3.3 ist der Schaltplan zum Anschluss eines

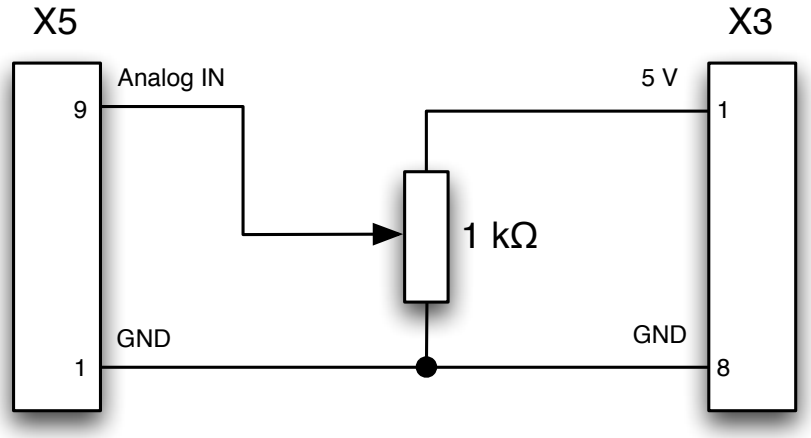
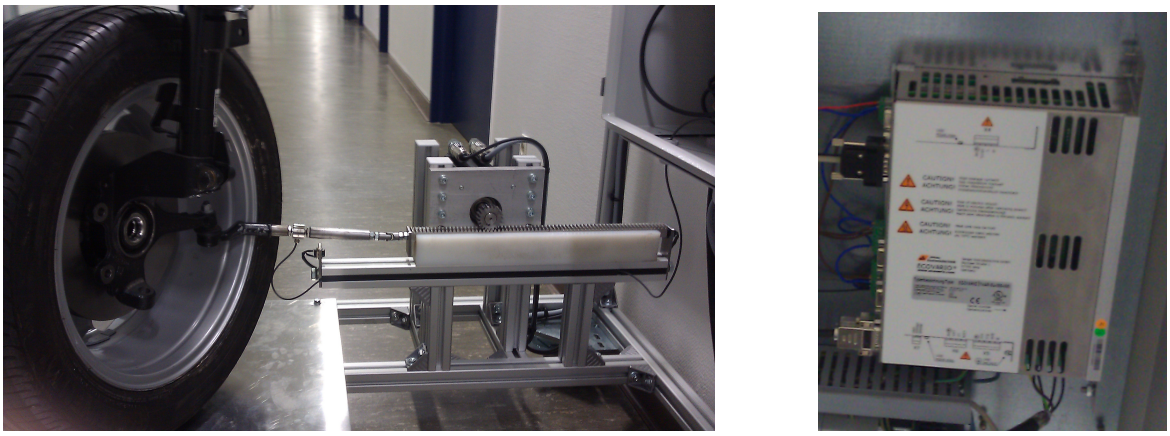


Abbildung 3.3: Schaltplan zum Anschluß eines Potentiometers zur Simulation der Geschwindigkeit

Potentiometers dargestellt. Die Masse des Encoders X3 Pin 8 muss mit der Masse des AD-Wandlers X5 Pin 1 verbunden werden. Die 5 Volt Versorgungsspannung werden dem X3 Pin 1 entnommen. Die äußeren Pins des Potentiometers werden mit der Spannungsversorgung verbunden. Der Pin mit dem veränderbaren Widerstand wird am X5 Pin 9 angeschlossen. Mit der Nachricht 0x20A wird die Messung angefordert. Die Nachricht 0x21A enthält in den Bytes 3 und 4 den Messwert von dem analogen Eingang.

3.2 ECOVARIO 214AR-BJ Verstärker mit Antrieb

Für die Realisierung der Querlenkung wird die radiale Kraft des Motors mit dem Zahnstangenantrieb in die lineare Kraft umgesetzt (siehe Abbildung 3.4 (a)). Die Kraft wird an den Querlenker übertragen, der das Rad zum Ausschlag bringt. Der in der Abbildung 3.4 (b) dar-



(a) Der Motor und die Mechanische Verbindung zum Rad

(b) Regelung des Motors

Abbildung 3.4: Die Realisierung der Querlenkung und entsprechender Verstärker

gestellte Verstärker übernimmt die Regelung des Motors. Die Regelung verfügt über einen Sollwertgenerator, der dafür sorgt, dass die durch die CAN Nachricht übermittelte absolute Position mit der angegebenen Geschwindigkeit angefahren wird.

3.2.1 Nachrichten

Der Verstärker wird mit dem CANopen Protokoll (vgl. Abschnitt 2.3) parametriert bzw. gesteuert. Die Parameter der Regelung werden mittels SDO Kommunikation eingestellt. Die GeräteID legt die Priorität der Nachricht fest. Damit die Nachrichten des Verstärkers mit den Nachrichten unter allen Endgeräten eindeutig bleiben, wird die GeräteID auf 2 eingestellt. Eine genauere Beschreibung der CANopen Objekte ist dem Objektverzeichnis der Jenaer Antriebstechnik GmbH (2009) zu entnehmen.

3.2.2 Referenzfahrt

Die Position des Rades nach dem Einschalten ist nicht bekannt. Um die Position zu bestimmen, verfügt der Verstärker über mehrere Referenzierungsmöglichkeiten, die sich in Präzision bzw. der Art der Suche unterscheiden. Die Endschalter schützen die bewegten Teile vor dem Erreichen der unerlaubten Position. Lichtschranken eignen sich sehr gut als Endschalter, weil sie sehr genau sind und mechanisch nicht beschädigt werden können. Die in der Abbildung 3.5 dargestellte äußeren Lichtschranken haben die Funktion der Endschalter. Mit

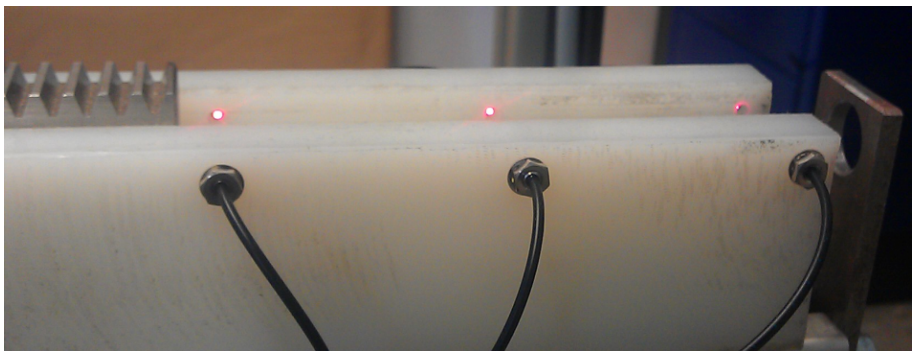


Abbildung 3.5: Lichtschranken

der mittleren Schranke wird der Wertebereich in zwei Unterbereiche geteilt. So kann beim Referenzieren sofort die Richtung zum Nullpunkt festgestellt werden.

3.3 DF30CAN digiCLIP CAN Messverstärker mit dem Kraftaufnehmer C9B

Um das Force Feedback zu realisieren, muss die Querkraft, die auf das Rad einwirkt, bestimmt werden. Dafür wird in der Abbildung 3.6 (a) dargestellte Messverstärker mit dem Kraftsensor in der Abbildung 3.6 (b) verbunden. Der Messverstärker kann mit unterschiedlichen Sensoren genutzt werden. Daher muss er entsprechend dem angeschlossenen Sensor eingestellt werden.

3.3.1 Nachrichten

Der Verstärker unterstützt CANopen Protokoll (vgl. Abschnitt 2.3). Mittels SDO Zugriffe wird der Messverstärker parametriert bzw. der Messwert ausgelesen. Um die Eindeutigkeit der Nachrichten zu gewährleisten wird die GeräteID auf 3 eingestellt. Die Beschreibung der CANopen Objekte ist dem Benutzerhandbuch der HBM GmbH zu entnehmen.

(a) Messverstärker. (Quelle: HBM GmbH)

(b) Kraftsensor

Abbildung 3.6: Die Hardware des Kraftaufnehmers

3.3.2 Skalierung

Damit der Messverstärker die elektrischen Messwerte in die entsprechenden physikalischen Werte umwandeln kann, müssen die Parameter des Sensors eingestellt werden. Der Kraftaufnehmer C9B besitzt enie elektrische Spanne zwischen 0 $\frac{mV}{V}$ und 1 $\frac{mV}{V}$. Die physikalische Spannbreite liegt zwischen 0 N und 5000 N. Nach der richtigen Einstellung liefert der Messverstärker den Messwert in Newton (N).

3.4 Entwicklungsboard Hilscher NXHX 500 ETM

Als Zielsystem steht das Entwicklungsboard NXHX 500 ETM zur Verfügung. Dieses basiert auf dem 32 Bit Prozessor ARM926EJ-S mit 200 MHz Real-time clock und verfügt über zwei 10/100 MBit/s Ethernet Kanäle, Dual Port Memory sowie USB Interface. Der Prozessor besitzt vier interruptfähige Hardwaretimer, die für eigene Zwecke genutzt werden.
Die ausführliche Beschreibung ist der Dokunmentatin Hilscher GmbH zu entnehmen. Weiterhin hat das Board drei Kanäle, die für den Anschluss der optionalen Peripheriegeräte zur Verfügung stehen.

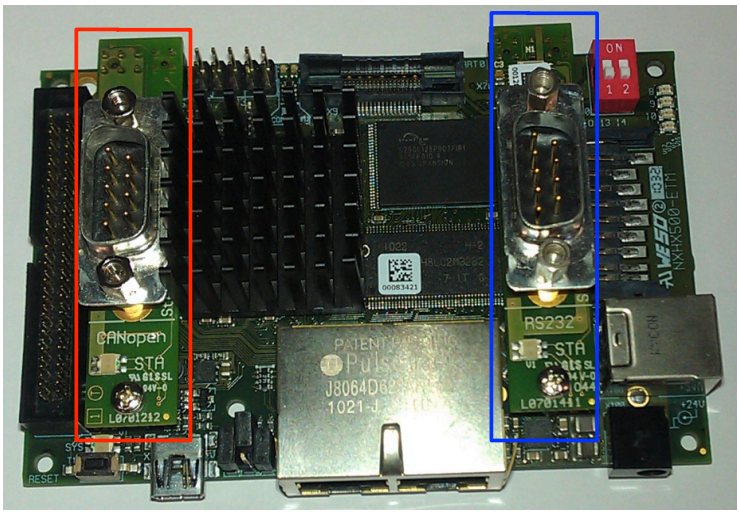


Abbildung 3.7: Microcontroller NXHX500ETM

3.4.1 Peripherie

Das Entwicklungsboard verfügt über Reihe von Peripheriegeräten, die sowohl in die Platine integriert als auch optional einsteckbar sind. Nachfolgend werden die für diese Arbeit relevanten Peripheriegeräte beschrieben. Den Zugriff auf die Peripherie ermöglicht die Hardware Abstraction Layer (HAL).

GPIO

Das NXHX 500 ETM weist 16 General Purpose Input/Output (GPIO) auf, die teilweise mit den anderen Geräten gemeinsam verwendet werden. Die GPIO 8-10 sind mit den LEDs festverdrahtet. Zwei verfügbare Switch-Schalter sind ebenfalls an den GPIO 13 und 14 angeschlossen. Es ist darauf zu achten, dass die Hardwareschalter die GPIOs mit dem RS-232 Modul teilen und beim eingesteckten Modul nicht verfügbar sind. Die GPIOs 9, 11, 13, 14 haben die Ausführung an den Pins und können auch als Ein- bzw. Ausgänge genutzt werden.

CAN Modul NXHX-CO

Für die CAN bzw. CANopen Kommunikation wird das Modul NXHX-CO verwendet. Das Modul wird in den Channel 2 des Entwicklungsboards eingesteckt. In der Abbildung 3.7 ist es mit dem roten Viereck umrandet. Den Zugriff auf die Hardware übernimmt der vom Hersteller gelieferte CAN HAL. Das Versenden der Nachricht wird von der Hardware ausgeführt und verbraucht daher die CPU Rechenzeit nicht.

RS-232 Modul NXHX-RS

Das RS-232 Modul wird in den dafür vorgesehenen Slot eingesteckt. In der Abbildung 3.7 ist es mit dem blauen Viereck umrandet. Um die textuellen Ausgaben am Display darzustellen, muss der Terminal mittels des Nullmodem-Kabels mit dem RS-232 Modul verbunden werden. Dafür kann auch ein herkömmlicher Computer mit der entsprechenden Software verwendet werden. Folgende Parameter müssen eingestellt werden:

```
Baudrate:       115200
Data Bits:      8
Parity:         none
Stop Bit:       1
Flow Control:   Hardware
```

Nachdem der im Abschnitt 5.2.6 beschriebene Debugmodus aktiviert ist, erscheinen die Ausgaben auf dem Display.

3.4.2 Entwicklungsumgebung

Als Entwicklungssprache ist C am besten geeignet, weil sie sehr effizient, schnell und hardwarenahe ist. Zum Kompilieren wird die Yagarto Toolchain verwendet. Die Toolchain beinhaltet alle nötigen Werkzeuge, um eine Anwendung für die ARM 9 Architektur zu erstellen. Die Erstellung von Sourcecode und das Kompilieren ist mit Eclipse CDT oder HiTOP möglich. Der Upload der Software über USB Kabel bzw. Debugging ist nur mittels HiTOP möglich.

3.4.3 Betriebssystem

Das von dem Hersteller gelieferte Betriebssystem „rcX“ ist für die Entwicklung des Real-time Ethernet (RTE) Stacks nicht geeignet, weil die Sourcecodes nicht zur Verfügung stehen. Die nötige Modifikation des Ehternet Stacks ist daher nicht möglich. Deswegen entwickelte Kai Müller (2011) ein Minibetriebssystem, das die echtzeitfähige Schedulingalgorithmen implementiert und der Anwendung zur Verfügung stellt. Vereinfacht werde ich diese Entwicklung im weiteren Verlauf der Bachelorarbeit als Betriebssystem bezeichnen. Weiterhin bietet das Betriebssystem den RTE Stack an, der sich mit der Synchronisierung der Zeit innerhalb des Netzwerkes befasst. Das Modul *Terminal* ermöglicht die textuelle Ausgabe über die RS-232 Schnittstelle. Im Betriebssystem können weiterhin die verfügbaren Module der HAL gbenutzt werden.

Scheduling

Der Scheduling[11] eines Echtzeitbetriebssystems unterscheidet sich von dem herkömmlichen in den verwendeten Schedulingalgorithmen. Das Echtzeitbetriebssystem stellt sehr hohe Ansprüche an die Einhaltung der Genauigkeit bei der Zuteilung der Rechenzeit. Die Nichteinhaltung der Deadline kann katastrophale Folgen haben. In dem oben beschriebenen Betriebssystem existiert keine Verwaltung des Kontextwechsels. Dieser wird mit dem Aufruf der Funktion realisiert und ähnelt damit dem Interrupt Handling bzw. der ISR. Der Scheduler verwendet den Fast Interrupt Request (FIQ) und hat damit die höchste Priorität. Die Schedulingtabelle befindet sich in der Datei *config.c* und enthält Zeitpunkte, zu denen die die Funktionen aufgerufen werden. Zu dem in der Schedulingtabelle eingegebenen Zeitpunkt wird die entsprechen Funktion aufgerufen und damit die Zuteilung der Rechenzeit realisiert. Kommt der nächste Zeitpunkt, so wird der laufende Kontext verlassen, unabhängig davon, ob die Berechnungen abgeschlossen sind. Die Prioritäten für weitere Aufgaben werden mit den Prioritäten des Interrupts gesteuert und im nachfolgenden Abschnitt beschrieben.

Interrupts

Alle Ereignisse des Mikrocontrollers besitzen eine Priorität, die mit der Priorität ID des entsprechenden Interrupts festgelegt wird. Die kleinere ID verfügt über eine höhere Priorität. Die Prioritäten werden in der Datei *interrupt.h* festgelegt. Nachfolgend werden die für diese Arbeit relevanten Prioritäten werden aufgelistet:

```
VIC_INTERRUPT_VECTOR_ETHERNET_RECEIVE0     1
VIC_INTERRUPT_VECTOR_SCHEDULER_SENDTT      6
VIC_INTERRUPT_VECTOR_SCHEDULER_TASK        8
VIC_INTERRUPT_VECTOR_CAN_RECV             12
VIC_INTERRUPT_VECTOR_GPIO                 13
```

Das Versenden bzw. das Empfangen der RTE Nachrichten hat eine höhere Priorität als die Prozesse und kann den Verlauf des Programms kritisch beeinflussen. Daher muss beim Erstellen des Schedulings besonders präzise vorgegangen werden.

3.5 ^{TTE}Development Switch 100 Mbit/s

Der im Abschnitt 4.1 beschriebene Switch stellt besondere Anforderungen an die Netzwerkkommunikation. Er muss über ein gemeinsames Ethernet Medium sowohl nicht kritische als auch zeitkritische Daten übertragen. Diese hohe Anforderung erfüllt der von TTTech entwickelte Real-time Ethernent Switch. Mit dem in der Abbildung 3.8 dargestellten 8 Port Switch werden die Nachrichten in drei Nachrichtenklassen aufgeteilt. Die Klasse Time-Triggered

[11] Unter Scheduling verbirgt sich die Erteilung dem Prozess nach einem festgelegten Zeitplan den Zugriff auf die Ressource

Abbildung 3.8: Real-time Ethernet Switch

(TT) beschreibt zeitkritische Nachrichten, die zu einem bestimmten Zeitpunkt empfangen bzw. versendet werden. Die zweite Klasse beinhaltet die Rate-Constrained (RC), die ebenfalls zeitkritisch sind, aber eine geringere Priorität als TT Nachrichten haben und daher in den freien Slots versendet werden. Bei mehreren RC Sendern konkurrieren sie um die gemeinsame Bandbreite. Damit das Verhalten des Netzwerkes vorhersagbar bleibt, besitzt die RC Klasse eine Bandbreitenbegrenzung. Diese wird durch den minimalen Abstand zwischen zwei gleichen Nachrichten festgelegt. Die Best-Effort (BE) Klasse entspricht dem herkömmlichen Ethernet. Bevor der Switch verwendet werden kann, muss er entsprechend der im Abschnitt 5.3.4 dargestellten Schedulungtabelle konfiguriert werden. Dafür müssen für jeden Port des Switches alle ankommenden und rausgehenden TT und RC Nachrichten registriert werden. Kennt der Switch die Nachricht nicht oder sie kommt vom falschen Port, so wird sie verworfen. Für den Ethernet Traffic (BE) kann sowohl eine herkömmliche lernfähige oder eine statische Routingtabelle eingestellt werden. Die genaue Beschreibung der Parametrierung ist der Betriebsanleitung des Switch zu entnehmen (vgl. TTTech GmbH).

Kapitel 4

Architektur und Design

In diesem Kapitel werden die Anforderungen des Projektes analysiert und daraus eine Architektur abgeleitet. Weiterhin werden im Abschnitt 4.1 die Entscheidungen bezüglich des Architekturdesigns dargestellt und diskutiert. Der Abschnitt 4.2 reduziert zwecks Übersichtlichkeit die Aufgabenstellung auf die Grundfunktionalität. Im Abschnitt 4.3 wird auf die Problematik der Komponentenverteilung im Netzwerk eingegangen. Abschnitt 4.4 beinhaltet die Beschreibung des konzeptionellen Ansatzes und stellt diesen mittels Modellierungstechniken übersichtlich dar. Abschließend wird das entwickelte Design analysiert und auf die Abdeckung der Anforderungen überprüft.

4.1 Architektur

Bevor man mit dem Architekturdesign beginnen kann, wird zunächst die grundlegende Funktionalität der Regelung untersucht. In der Ausgangslage werden die vorhandenen Endgeräte räumlich in zwei CAN Busse aufgeteilt. In einem Bus werden der Motor und der Kraftaufnehmer platziert, in dem anderen befindet sich das Lenkrad. Da sowohl der Motor als auch der Kraftaufnehmer CANopen Protokoll implementieren, kann von einem reinen CANopen Bus gesprochen werden. Die Aufgabe des Controllers besteht dabei darin, in regelmäßigen Abständen den Winkel des Lenkrades per CAN auszulesen und daraus die Position des Motors zu berechnen und einzustellen. Mittels einer Real-time Kommunikation sollen diese beiden Bussysteme und der Controller zu einem gemeinsamen Netzwerk zusammengeschlossen werden. Dafür ist eine Bridge (vgl. Abbildung 4.1) notwendig, die die CAN Nachrichten in Real-time Ethernet Frames verpackt und an den Controller verschickt. Die Bridge und der Controller ähneln sich funktional. Beide tunneln die CAN Nachrichten mittels Realtime Ethernet Kommunikation. Dies bedeutet, dass der Controller nicht nur das Tunneling-Protokoll der Bridge beherrscht, sondern auch die komplette Bridge nachahmen muss. Daher stellt sich die Frage ob man die Funktionalität der Bridge wiederverwenden kann. Diese Funktionalität wäre innerhalb eines Mikrocontrollers leicht realisierbar, indem der CAN-Bus

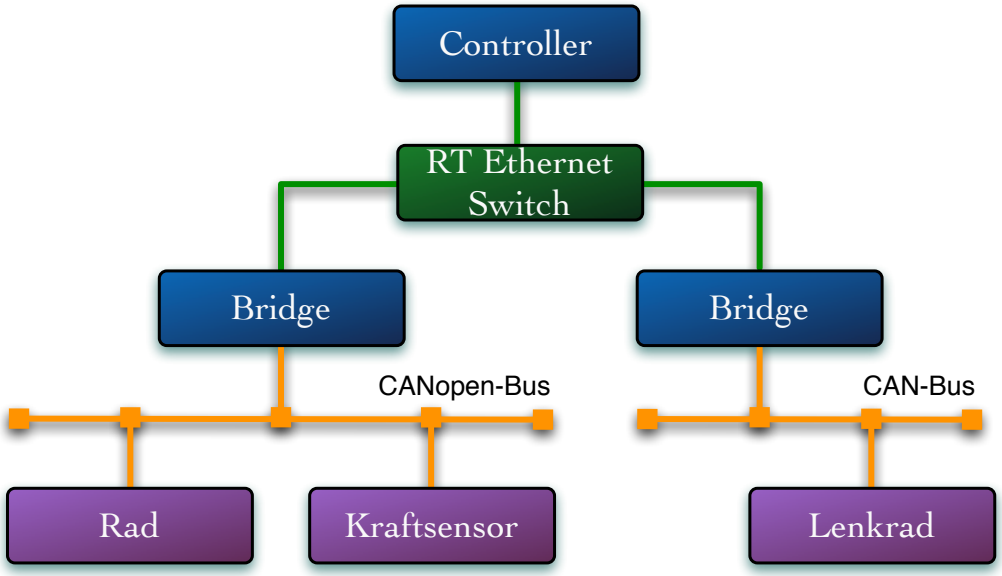


Abbildung 4.1: Konzeptioneller Aufbau einer Steer-by-Wire Netzwerkarchitektur

virtuell realisiert wird. Der Einsatz der Bridge ermöglicht nicht nur ihre Wiederverwendbarkeit, sondern reduziert mögliche Fehlerquellen und erlaubt den Einsatz des Controllers sowohl über Real-time Ethernet als auch nur über CAN bzw. CANopen. Die Realisierung der Bridge mitsamt des virtuellen CAN Interfaces wird in eine eigenständige Abschlussarbeit verlagert, mit der sich Jan Kamieth (2011) beschäftigt. Der Real-time Ethernet Stack wird von Kai Müller (2011) implementiert. Ein weiterer positiver Aspekt dieser Designentscheidung ist die erhöhte Flexibilität bezüglich des Einsatzes und der absoluten Unabhängigkeit von der Bridge, was sich bei der Fehlersuche besonders auszahlt.

4.2 Vereinfachte Architektur

Bei sehr komplexen und unübersichtlichen Aufgaben ist es von besonderer Bedeutung, dass das Gesamtziel in kleinere, beherrschbare Teilaufgaben unterteilt wird. Während der Softwareentwicklung der Grundfunktionalität hat die räumliche Entfernung zwischen dem Controller und den Aktoren bzw. Sensoren keine kritische Auswirkung. Daher wird der Controller zuerst nur innerhalb eines CAN-basierten Systems realisiert. Die Abbildung 4.2 stellt eine vereinfachte Architektur der Abbildung 4.1 dar. Alle Knoten sind am gleichen Bussystem angeschlossen. Dadurch werden die Vorgänge, die in einer vollständigen Architektur parallel ablaufen sequentiell ausgeführt, was zur Auslastung der Bandbreite führen kann. Die dabei festgelegte Teilaufgabe begrenzt sich auf die Realisierung der Funktionalität der Len-

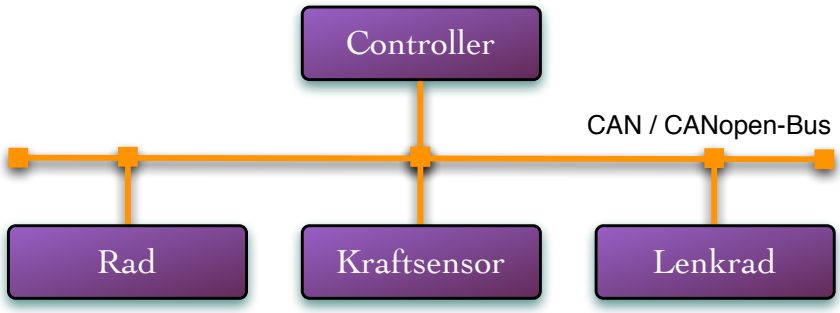


Abbildung 4.2: Vereinfachte Netzwerkarchitektur

kung bzw. des Force Feedbacks. Nachdem der Controller entwickelt und getestet ist, kann er in die vollständige Architektur integriert werden. Allerdings muss dabei beachtet werden, dass die in diesem Ansatz auftretende Busauslastung auf zwei Bussysteme verteilt wird und dadurch einen parallelen Ablauf ermöglicht. Hinzu kommt die Verzögerung des Real-time Netzwerkes. Diese beeinflusst das Zeitverhalten des Controllers.

4.3 Verteilter Regelkreis

Sowohl das Lenkrad als auch der Servoverstärker des Motors besitzen eine separate Regelung, die teilweise eigenständig die Aufgaben übernimmt. Die Regelung des Lenkrades realisiert die grundlegende Funktionalität der Endanschläge sowie des Lenkverhaltens (vgl. Abschnitt 3.1). Der im Abschnitt 3.2 beschriebene Servoverstärker erlaubt die Positionierung und das Setzen der Positionierungsgeschwindigkeit. Diese Funktionen sind nicht in einem gemeinsamen Controller, sondern als separate funktionale Blöcke realisiert. Da sie zu einem gemeinsamen über ein Netzwerk verteiltes System zusammengeschlossen werden, spricht man von einer verteilten Regelung. Die Funktionalität der Steer-by-Wire Lenkung setzt sich aus zwei Pfaden zusammen. Der erste Pfad hat die Aufgabe, den Winkel des Lenkrades auszulesen, auf den Wertebereich der Zahnstange zu skalieren und auf die berechnete Position zu setzen. Erst durch die Rückkopplung über den Force Feedback Pfad und den Fahrer wird dies zu einem Regelkreis. Der Kraftsensor wird in regelmäßigen Abständen aufgefordert den Messwert an den Controller zu senden. Aus dem Messwert der Querkraft, die auf die Querstange einwirkt, wird das Drehmoment berechnet und anhand dessen das Lenkrad eingestellt. Dieser Regelkreis unterscheidet sich von dem im Abschnitt 2.4.1 vorgestellten, da hier die Veränderung der Querkraft direkt auf den von dem Fahrer vorgegebenen Sollwert einwirkt.

4.4 Softwaredesign

Im Gegensatz zu den herkömmlichen PC's verfügen die Mikrocontroller über begrenzte Ressourcen und Rechenleistung. Daher wird bei dem Softwaredesign für einen Mikrocontroller besonders auf die Effizienz geachtet. Oftmals ist ein komplexes Design mit Overhead verbunden, der die Ressourcen des Mikrocontrollers stark belasten kann. Daher werden die Schwerpunkte auf die Effizienz, Erweiterbarkeit und die Wartbarkeit gelegt. Die Anzahl der Module und Funktionsaufrufe soll auf ein Minimum reduziert werden, da die verschachtelte Funktionsaufrufe unnötig Rechenzeit und Speicher verbrauchen. Da diese Arbeit auf den bereits entwickelten Komponenten aufsetzt, muss das Softwaredesign entsprechend entworfen werden. Weiterhin sollen Kundenanforderungen analysiert und daraus resultierende Anwendungsfälle festgelegt werden.

4.4.1 Anwendungsszenarien

Zum Festlegen der Anwendungsszenarien werden die gesetzten Ziele und Anforderungen analysiert. Das Use-Case Diagramm 4.3 stellt typische Lenkungsvorgänge eines Fahrers dar. In der Abbildung sind fünf Anwendungsszenarien erkennbar, die nachfolgend beschrieben werden.

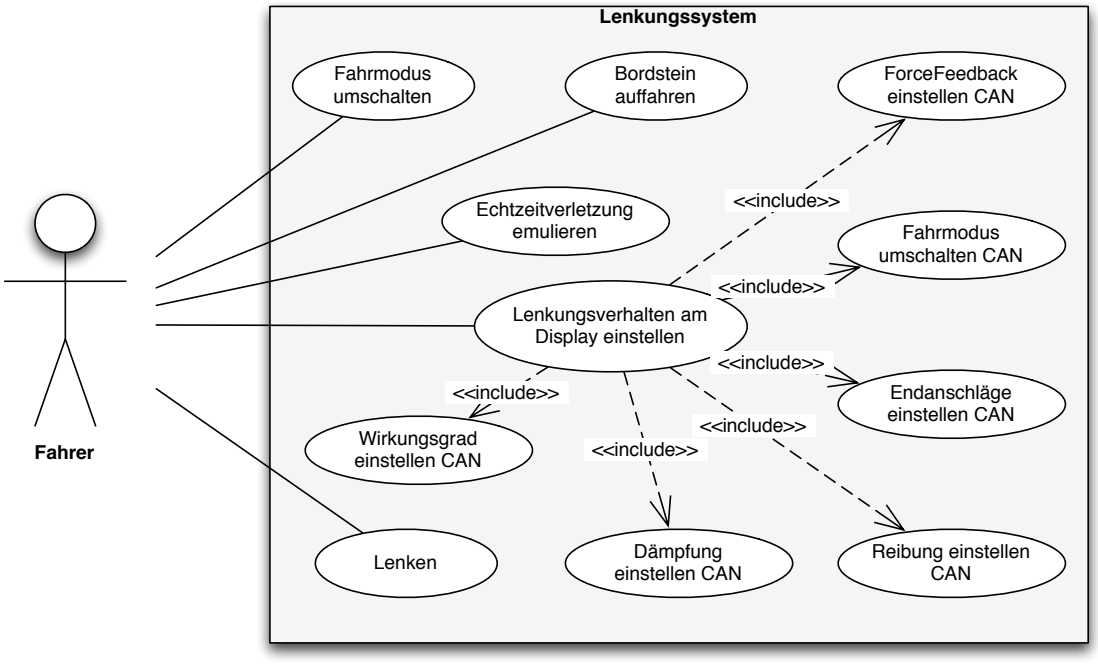


Abbildung 4.3: Anwendungsfälle des Produktes

- In dem Fall „Fahrmodus auswählen“ wird mittels eines Schalters zwischen dem Sportmodus und dem Cruisemodus umgeschaltet.
- Sollte der Fahrer auf den „Bordstein auffahren“ und damit eine Querkraft erzeugen, so wird diese an das Lenkrad weitergereicht.
- Der Usecase „Lenken“ entspricht dem gewöhnlichen Fahren.
- Im Fall „Echtzeitverletzung emulieren“ soll die Notwendigkeit der harten Echtzeitanforderungen greifbar gemacht werden.
- Mit dem Anwendungsfall „Lenkungsverhalten am Display einstellen“ werden mittels eines Infotainment Centers per CAN die Parameter der Lenkung wie Force Feedback, Endanschläge, Reibung, Dämpfung, Wirkungsgrad und der Fahrmodus eingestellt.

4.4.2 Verteilte Kommunikation

Mittels eines Sequenzdiagramms können parallel ablaufende Vorgänge übersichtlicher dargestellt werden. Das in der Abbildung 4.4 veranschaulichte Diagramm zeigt den prinzipiellen Aufbau der verteilten Kommunikation zwischen den einzelnen Komponenten in der zeitli-

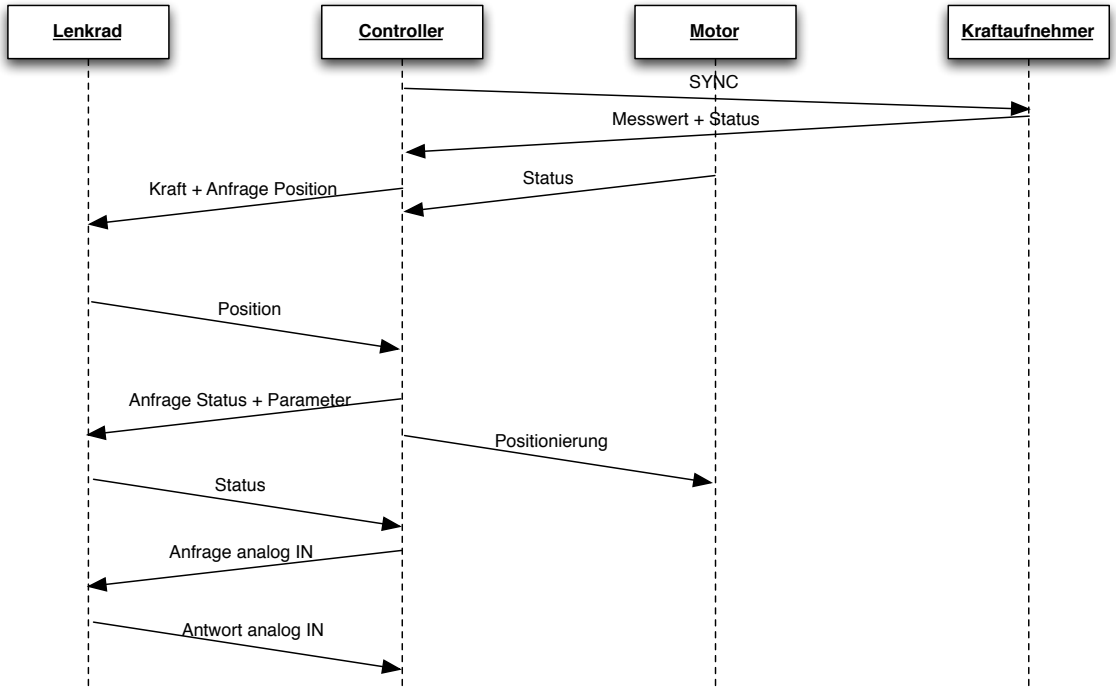


Abbildung 4.4: Prinzipieller Aufbau der verteilten Kommunikation

chen Abfolge auf. Für beide Architekturen ist die dargestellte Kommunikation gleich. Die aufgeführte Folge von Nachrichten wird zyklisch wiederholt und stellt damit eine vollständige Periode[12] dar. Der Controller spielt dabei die übergeordnete Rolle. Mittels der ersten Nachricht, die der Controller an das Lenkrad richtet, wird die Kraft des Kraftaufnehmers gesetzt. Der dafür benötigte Messwert des Kraftaufnehmers wird mittels SYNC Nachricht angefordert (vgl. Abschnitt 2.3.5). Aus der erhaltenen Position des Rades wird in die Position des Motors berechnet und an den Motor übermittelt. Weiterhin erhält der Controller die Statussignale von dem untergeordneten Knoten und reagiert gegebenenfalls auf diese.

4.4.3 Module und Softwarearchitektur

Jedes der Endgeräte benötigt eine übergeordnete Steuerungseinheit, die die Daten bzw. den Status in regelmäßigen Abständen ausliest. Dadurch ergeben sich die Module für den Motor, den Kraftaufnehmer und das Lenkrad, die parallel ihre Aufgaben verrichten. Die Aufgabe der Module besthet darin, die entsprechende Hardware in den Betriebsmodus zu führen, die Daten bzw. den Statuszustand in regelmäßigen Abständen anzufordern und gegebenenfalls auf Fehler zu reagieren. Die Kommunikation zwischen den Endgeräten erfolgt mittels eines CAN Moduls, der die CAN Nachrichten versendet bzw. entgegennimmt. Das Modul Control-

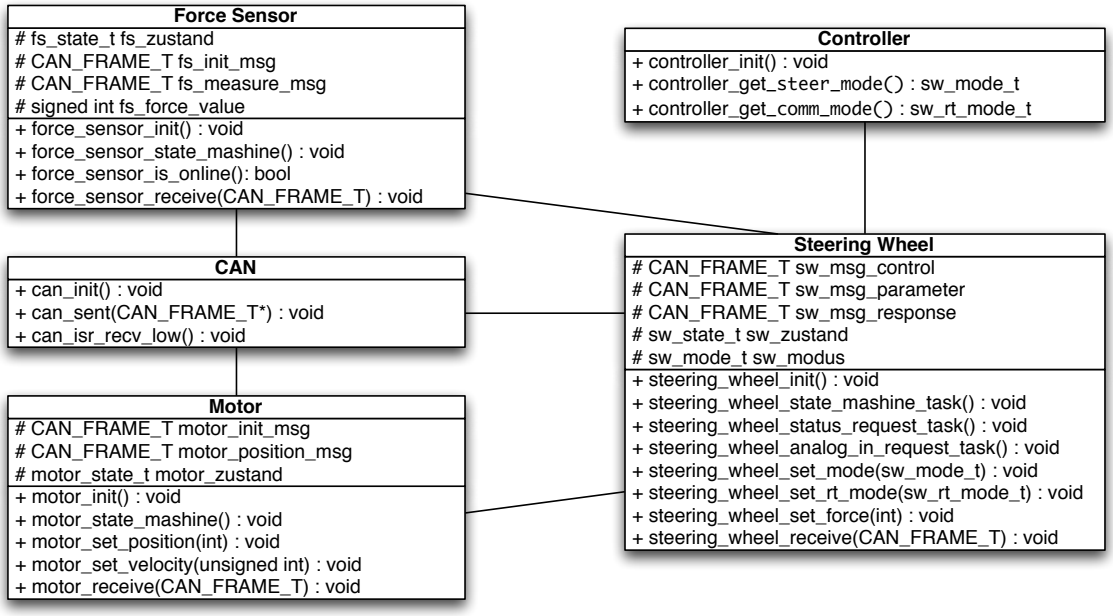


Abbildung 4.5: Klassendiagramm der Softwarearchitektur

[12] Als Periode bezeichnet man ein Zeitintervall zwischen zwei, sich in gleichen Abständen wiederholenden Ereignissen

ler hat die Aufgabe alle Module zu initialisieren. Weiterhin stellt es die Funktionen zum Zugriff auf die Hardwareschalter zur Verfügung. Das in der Abbildung 4.5 dargestellte Klassendiagramm enthält die Übersicht der festgelegten Module, sowie die Funktionen und Parameter. Um umfangreiche Architekturveränderungen während der Umsetzungsphase zu vermeiden, wird die Softwarearchitektur auf die Erfüllung der Kundenanforderungen überprüft. Dafür werden alle möglichen Anwendungsszenarien durchgespielt und die Schlüssigkeit der Vorgänge getestet.

- Die Position des Schalters „Fahrmodus auswählen“ wird mit der Funktion `controller_get_steer_mode()` abgerufen und als Parameter an die Funktion `steering_wheel_set_mode(sw_mode_t)` übergeben um das Lenkungsverhalten anzupassen.
- Mittels der Funktion `controller_get_comm_mode()` wird festgestellt, ob die Echtzeitverletzung emuliert werden soll oder nicht.
- Sollte auf das Lenkrad eine Kraft eingewirkt haben, so kann sie mit dem Aufruf der Funktion `steering_wheel_set_torque(int)` auf das Lenkrad gesetzt und damit Force Feedback realisiert werden.
- Verändert der Fahrer den Winkel des Lenkrades, so wird daraus die Position des Motors berechnet und als Parameter an die Funktion `motor_set_position(int)` übergeben.
- Die Anwendung des Infotainment Centers schickt CAN Nachrichten an das Lenkradmodul. Anhand der ID stellt das Modul fest, ob es sich um die Parameter der Lenkung handelt und stellt diese ein.

Diese Überlegungen zeigen, dass die entworfene Softwarearchitektur den gesetzten Zielen und den Kundenanforderungen gerecht wird. Die Realisierung wird im nachfolgenden Kapitel beschrieben.

Kapitel 5

Umsetzung

Dieses Kapitel beschreibt die Realisierung der Software bzw. das Zusammenspiel mit weiteren Bachelorarbeiten im Projektumfeld. Weiterhin enthält das Kapitel die Beschreibung des externen CAN Interfaces zur Anpassung des Lenkungsverhaltens.

5.1 Arbeiten im Projektumfeld

Das Gesamtziel des Projektes setzt sich aus mehreren Bachelorarbeiten zusammen, die in diesem Abschnitt beschrieben werden. Insgesamt sind es drei Bachelorarbeiten, die mit Steer-by-Wire eng verbunden sind und teilweise aufeinander aufbauend. So setzt beispielsweise die im Abschnitt 4.1 vorgestellte Bridge auf der Implementierung von Real-time Ethernet auf. Auch das Infotainment Center kann erst nach der Definition der Schnittstelle zur Anpassung des Fahrverhaltens entwickelt werden. Die Konzeption bzw. Entwicklung der Steer-by-Wire Lenkung ist von der Bridge und dem Betriebssystem abhängig.

5.1.1 Realtime Ethernet Implementierung

Im Zuge des Communication over Real-time Ethernet (CoRE)-Projekts entwickelt Kai Müller (2011) nach der TTEthernetspezifikation von TTTech den Microcontroller-basierten Real-time Ethernet Stack. Die Echtzeitkommunikation setzt eine global synchronisierte Zeit voraus. Um die Zeitabweichungen zwischen den einzelnen Knoten zu beheben, werden entsprechend dem TTEthernet Protokoll Sync Nachrichten ausgetauscht und damit die globale Zeit eingestellt. Damit laufende Prozesse und das Versenden der Nachrichten planmäßig abgearbeitet werden, sind spezielle Schedulingalgorithmen nötig. Das Ziel dieser Bachelorarbeit ist ebenfalls die dafür benötigten Algorithmen zu implementieren bzw. als Echtzeitbetriebssystem für die Anwendung zur Verfügung zu stellen.

5.1.2 Entwuf einer CAN-RTE - Bridge

Bei dieser Arbeit handelt es sich um eine Lösung, die CAN-basierte Netzwerke mittels Real-time Ethernet verbindet. Die von Jan Kamieth (2011) entwickelte Bridge übernimmt die Aufgabe die ereignisgesteuerte CAN Kommunikation auf die zeitgesteuerte Real-time Ethernet Kommunikation umzusetzen. Dafür wird der gesamte Traffic in Real-time Ethernet Kanäle aufgeteilt. Man unterscheidet TT, RC und BE Kanäle. Außerdem verwaltet Bridge Routen für alle CAN Nachrichtentypen. Weiterhin stellt sie eine virtuelle CAN-Schnittstelle zur Verfügung, die einen vielfältigen Einsatz erlaubt. Ein beispielhafter Einsatz dieser Architektur ist die Backbone-Infrastruktur. Dafür werden mehrere Netzwerke mithilfe eines sehr leistungsfähigen Netzwerkes verbunden.

5.1.3 Infotainment Center

Grundidee dieser Bachelorarbeit ist die Visualisierung der Möglichkeiten, die mit dem Real-time Ethernet gegeben sind. So kann man über ein gemeinsames Medium sowohl zeitkritische Kommunikation als auch multimediale Anwendungen mit hohen Bandbreitenanforderungen betreiben, ohne dass sie sich gegenseitig beeinflussen. Zu den zeitkritischen Anwendungen im Auto zählt beispielsweise die Kommunikation zwischen den Steuergeräten. Die multimedialen Anwendungen in Bezug auf die Zukunft sind beispielsweise die Rückfahrtkamera, Kameras als Ersatz der Außenspiegel oder auch Musik bzw. Videos, die an der Tankstelle gekauft und sofort heruntergeladen werden können. Über ein Touchscreen sollen die Parameter der Lenkung über Schiebebalken angepasst werden. Weiterhin wäre denkbar, dass durch die Auswahl eines Fahrzeuges aus der Liste die entsprechenden Eigenschaften der Lenkung eingestellt werden.

5.2 Programmaufbau und Beschreibung der Module

Das im Abschnitt 3.4.3 vorgestelltes Echtzeitbetriebssystem besitzt keine gewöhnliche Threadverwaltung. Die Funktion, die die Rolle des Threads übernimmt, wird jedes Mal aufgerufen und nicht an der unterbrochenen Stelle fortgesetzt. Ein herkömmlicher Threadaufbau ist nur mit sehr viel Aufwand realisierbar. Zudem enthält das Betriebssystem weder Synchronisationsverfahren, wie Semaphoren oder Mutexe, noch die Möglichkeit, den Programmablauf anzuhalten. Daher bietet sich ein Zustandsautomat an, der den Zustand des Programms speichert, um beim nächsten Aufruf wieder an der unterbrochenen Stelle fortzufahren. Abhängig von dem Zustand führt der Automat die erforderliche Aktion durch und bei Bedarf wird sie wiederholt. Die im Abschnitt 4.4.3 diskutierte Softwarearchitektur besteht aus einem Hauptprogramm und drei Threads. Die Rolle des Hauptprogramms übernimmt das Modul Controller. Auf seine Funktionalität wird im folgenden Abschnitt eingegangen. Die Threads

haben die Aufgabe mit den CAN bzw. CANopen Geräten zu kommunizieren. Zu den parallel ablaufenden Programmteilen gehören die Module *Steernig Wheel*, *Force Sensor* und *Motor*. In der Konfigurationsdatei des Scheduler *config.c* werden die Dauer der Periode und die Zeiteinträge zu jedem Thread angegeben. Anhand dieser zum Anfang der Periode relativen Zeiteinträge weiß der Scheduler, wann die Funktion, die den Thread beinhaltet, aufgerufen werden soll. Der Zustandsautomat speichert den Verlauf des Threads ab und setzt beim nächsten Aufruf an der unterbrochenen Stelle fort. Threads haben die Aufgaben, die ankommenden Daten zu verarbeiten, eine CAN Nachricht zusammenzusetzen und zu versenden. Die ankommenden CAN Nachrichten sind ereignisgesteuert und können zu jedem Zeitpunkt eintreffen. Aufgrund der Bestätigung des erfolgreich durchgeführten Vorgangs wird eine Zustandstransition des Automaten durchgeführt. Somit wird der Übergang vom ereignisgesteuerten Empfangen zum zeitgesteuerten Senden realisiert. Da eine CAN Nachricht aufgrund der Synchronisierung von Real-time Ethernet (vgl. Abschnitt 5.3) verloren gehen kann, wird die Nachricht in jedem Zyklus erneut versendet bis eine Bestätigung empfangen wird. Dadurch erreicht man die vorausgesetzte Robustheit der Zustandsautomaten und vermeidet Deadlocks.

5.2.1 Modul „Controller“

Das Modul Controller ist mit der `main()` vergleichbar. Seine Aufgabe ist es, die verwendeten Module zu initialisieren bzw. Threads zu erzeugen. Weiterhin enthält das Modul die Funktionen zum Zugriff auf die Hardwareschalter. Damit innerhalb eines Mikrocontrollers die Konfiguration der Hardware an einer Stelle erfolgt, wird GPIO und damit beide Schalter in der `main()` initialisiert. Beim Hochfahren des Betriebssystems wird die Initialisierung des *Controllers* einmalig aufgerufen und damit die Anwendung gestartet.

Initialisierung

Mit dem Aufruf von `void controller_init(void)` werden die Module *Steernig Wheel*, *Force Sensor* und *Motor* initialisiert. Nach der Initialisierung verhalten sich die Threads wie unabhängige Programmteile. Ab diesem Zeitpunkt führen sie die Initialisierung der zugehörigen Endgeräte durch, um die Bereitschaft der Hardware zu realisieren. Auf die Funktionalität der einzelnen Bestandteile wird in den entsprechenden Modulbeschreibungen eingegangen.

Zugriff auf die Hardwareschalter

Die Interrupt-gesteuerte Nutzung von GPIO ist nicht möglich, da die Interrupts entweder bei steigender oder bei fallender Flanke ausgelöst werden (vgl. Abschnitt 3.4.1). Weiterhin sind die Schalter nicht prellsicher und müssen entprellt werden. Bei einer softwarebasierten

Entprellung wird der Zustand der Schalter eine bestimmte Zeit beobachtet. Erst wenn innerhalb dieser Zeit keine Zustandsveränderungen festgestellt wurden, wird der Wert übernommen. Diese Lösung ist mit dem gewählten Ansatz nicht realisierbar. Sollte der Zustand des Schalters einmal pro Periode ausgelesen werden, so wird der Wert auf die Gesamtperiode verlängert und damit der ähnliche Effekt erzielt. Durch das Verschieben des Schalters 1 (siehe Abbildung 3.7) wird zwischen dem *Sport*- und *Cruisemodus* umgeschaltet. Mit Aufrufen der Funktion `controller_get_steer_mode()` kann der Zustand des Schalters abgefragt werden. Als Rückgabewert wird der Modus zurückgegeben, der eingestellt werden soll. Mittels Schalter 2 wird eingestellt, ob die Emulation der Verletzung von Echtzeitkommunikation durchgeführt werden soll. Durch den Aufruf von `controller_get_comm_mode()` kann die Position des Schalter abgefragt werden. Abhängig von dem Rückgabewert werden entweder alle Nachrichten versendet oder nur ein Teil davon. Auf diese Weise wird der Eindruck vermittelt, dass die Nachrichten verloren gegangen sind.

5.2.2 Modul „CAN“

Die Aufgabe dieses Moduls besteht darin, die Verbindung mit dem im Abschnitt 3.4.1 vorgestellten CANopen Hardwaremodul herzustellen und damit die Kommunikation mit den CAN Endgeräten zu ermöglichen. Die ankommenden Nachrichten sind ereignisgesteuert und können zu jedem Zeitpunkt eintreffen. Wird eine Nachricht empfangen, so generiert CAN-HAL den Interrupt, der vom Modul ausgewertet wird. Zudem enthält CAN Modul die Definition von CANopen-konformen Konstanten, die für die Parametrierung bzw. die Zusammensetzung von Nachrichten genutzt werden. Bevor die Hardware zum Einsatz kommen kann, muss die Initialisierung durchgeführt werden.

Initialisierung

Durch den Aufruf von `void can_init(void)` wird die CAN-Hardware initialisiert. Damit alle CAN Geräte untereinander kommunizieren können, müssen sie die eingestellte Übertragungsgeschwindigkeit unterstützen. Daher werden alle Geräte auf eine gemeinsame Geschwindigkeit von 1 Mbit/s eingestellt. Weiterhin wird die Priorität des Interrupts eingestellt (vgl. Abschnitt 3.4.3). Nachdem der CAN auf Cannel 2 eingestellt wurde (vgl. Abschnitt 3.4.1), wird der Interrupt aktiviert und kann benutzt werden.

Vermittlung der empfangenen Nachrichten zu den Modulen

Da das Empfangen von Nachrichten durch das Auslösen von Interrupt mitgeteilt wird, muss in der Interrupt Service Routine (ISR) untersucht werden, wie mit der Nachricht zu verfahren ist. Die Tabelle 5.1 enthält eine Übersicht der Nachrichten, die vom CAN Modul empfangen werden können. Anhand des Identifiers wird entschieden, an welches Modul die Nachricht

ID	Protokoll	Modul	Beschreibung
0x182	CANopen	Motor	PDO Status des Motors
0x281			NMT Antwort
0x582			SDO Antwort
0x183		Force Sensor	PDO Status des Kraftaufnehmers
0x583			SDO Antwort
0x604		Steering Wheel	SDO Parameter der CAN Schnittstelle Anfrage
0x210	CAN		Status des Lenkrades Antwort
0x211			Position des Lenkrades Antwort
0x21A			Geschwindigkeit des Fahrzeugs Antwort

Tabelle 5.1: Liste der ankommenden Nachrichten und deren Zielmodule

übermittelt werden soll. Jedes Modul hat eine Funktion, die die CAN Nachricht erwartet und gegebenenfalls verarbeitet. Durch den Aufruf der Funktion des zugehörigen Moduls wird die Nachricht an den Empfänger weitergereicht.

Versenden der Nachrichten

Um eine CAN Nachricht zu versenden, wird der Zeiger auf die Struktur, die einen CAN Frame enthält, an die Funktion `can_send(CAN_FRAME_T*tFrame)` übergeben. Die Funktion leitet das an den CAN HAL weiter, der die Nachricht anschließend versendet. Das Versenden der CAN Nachricht geschieht parallel und verbrauch nicht die CPU Rechenzeit. Die Tabelle 5.2 enthält die Nahrichten und deren Quellmodule, die vom Controller versendet werden.

ID	Protokoll	Modul	Beschreibung
0x0	CANopen	Motor	NMT Nachricht, schaltet PDO ein
0x252			Position, Geschwindigkeit setzen
0x602			SDO Anfrage
0x80		Force Sensor	SYNC Nachricht
0x603			SDO Anfrage
0x584		Steering Wheel	SDO Parameter der CAN Schnittstelle Antwort
0x200	CAN		Status des Lenkrades Anfrage
0x201			Position des Lenkrades Anfrage
0x20A			Geschwindigkeit des Fahrzeugs Anfrage

Tabelle 5.2: Liste der ausgehenden Nachrichten und deren Quellenmodule

5.2.3 Modul „Steering Wheel“

Dieses Modul übernimmt die Aufgabe der Kommunikation mit dem im Abschnitt 3.1 vorgestellten Lenkrad. Das Modul besteht aus drei Threads. Der erste wird durch die Funktion `steering_wheel_state_mashine_task(void *arg)` repräsentiert und enthält den Zustandsautomat, der den Ablauf des Lenkrades verwaltet. Seine Funktionalität wird nachfolgend ausführlich beschrieben. Der zweite Thread hat die Aufgabe, den Wert des analogen Eingangs anzufordern und befindet sich in der Funktion `steering_wheel_analog_in_request_task(void *arg)`. Mit dem Messwert wird die Geschwindigkeit des Fahrzeuges emuliert und damit die Abhängigkeit des Lenkungsverhaltens von der Geschwindigkeit des Fahrzeuges nachgeahmt (siehe Abschnitt 5.4). Der dritte Thread befindet sich in der Funktion `steering_wheel_status_request_task(void *arg)` und dient der regelmäßigen Statusabfrage des Lenkrades. Außerdem enthält die Anfrage die Steuerungsparameter wie Betriebsmodus bzw. Zustand, Endanschläge, und das Drehmoment. Anhand des in der Antwort enthaltenen Status kann der Verlauf der Hardware beobachtet werden. Sollte ein Fehler auftreten, kann darauf reagiert und das Endgerät wieder in den Betriebszustand gebracht werden.

Parametrierung

Das Modul hat eine Reihe von konstanten Parametern, die man vor dem Kompilieren verändern und damit die Eigenschaften des Controllers anpassen kann. Mittels der in der Datei *steering_wheel.h* definierten Konstanten können *default* Parameter für die Lenkungsmodi *CRUISE*, *SPORT CUSTOM* und *DEMO* eingestellt werden. Weiterhin enthält die Datei Konstanten zum Zugriff auf das externe Interface, das im Abschnitt 5.5 beschrieben wird.

Initialisierung

Threads sind Programmteile, die parallel zur Initialisierung ablaufen können. Daher wäre es falsch, wenn der Thread vor der abgeschlossenen Initialisierung auf nicht initialisierte Variablen zugreifen würde. Deswegen hat der Thread den Zustand *Offline*, der den Zugriff auf die Variablen verbietet. Erst durch den Aufruf von `steering_wheel_init()` werden die Variablen initialisiert und erst dann wird der Thread freigegeben.

Zustandsautomat

Der Zustandsautomat hat die Aufgabe, beim erstmaligen Einschalten das Lenkrad zu referenzieren und in den Betriebsmodus zu bringen. Dies steuert der im Abschnitt 3.1.3 beschriebene Zustandsautomat des Endgerätes. Eine Besonderheit dieses Zustandsautomaten ist, dass der Befehl zum Verändern des Zustandes nicht den Abschluss der Transition bestätigt,

sondern lediglich das Empfangen der Nachricht. Damit erhält die Bestätigung nicht den aktuellen Zustand, sondern den Zustand zum Zeitpunkt des Versendens. Durch diesen Einsatz erreicht die Regelung des Endgerätes eine sehr schnelle Reaktionszeit mit der Konsequenz, dass der übergeordnete Zustandsautomat auf die Veränderung des Zustands pollen muss. Erst nach dem Eintreten des Ereignisses wird die Zustandstransition ausgeführt. In der

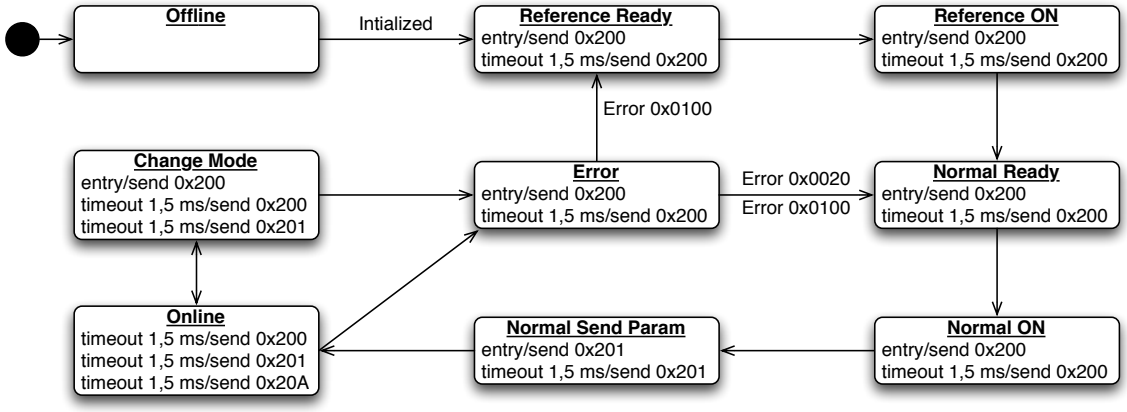


Abbildung 5.1: Zustandsautomat des Moduls Steering Wheel

Abbildung 5.1 ist der Zustandsautomat des Moduls Lenkrad schematisch dargestellt. Jeder Zustand enthält die Information über die Nachrichten ID, die in dem Zustand versendet wird. Im Zustand *Offline* wird gewartet bis die Initialisierung abgeschlossen ist. Am Ende der Initialisierung wird die Zustandstransition zum *Reference Ready* durchgeführt und damit die Hardware in den entsprechenden Zustand überführt (vgl. Abschnitt 3.1.3). Nachdem der Zustand von der Hardware bestätigt wird, wechselt der Automat in den Zustand *Reference ON* und die Referenzfahrt beginnt. Das Lenkrad dreht sich auf der Suche nach dem Index im Uhrzeigersinn. Nachdem der Index gefunden wurde, bewegt sich das Lenkrad zu der angegebenen Nullposition zurück. Erst wenn diese Position erreicht ist, nimmt der Zustandsautomat den Zustand *Normal Ready* an, um den Motor auf das Einschalten vorzubereiten. Zudem werden die Endanschläge und der Wirkungsgrad eingestellt. Im Zustand *Normal ON* wird der Motor des Lenkrades eingeschaltet. Der Zustand *Normal Send Parameter* stellt die Parameter der Lenkung wie Reibung, Dämpfung, und Federsteifigkeit ein. Danach befindet sich der Automat im Zustand *Online* und liest regelmäßig den Winkel des Lenkrades aus (Nachricht ID 0x201). Aus dem Winkel wird die Position des Lenkrades berechnet und an die Funktion `motor_set_position(int position)` des Motor übergeben.

$$\alpha_{Lenkrad} = \frac{Position_{Lenkrad} * 360^{\circ}}{Inkremente\ pro\ Drehung} \tag{5.1}$$

$$Position_{Motor} = \frac{\alpha_{Lenkrad} * MOTOR\_ENDSTOP}{\beta_{Lenkrad\ endstop}} \tag{5.2}$$

Zudem enthält die Antwort die Geschwindigkeit der Verstellung des Lenkrades. Aus der Geschwindigkeit und der Position der Endanschläge wird die Positionierungsgeschwindigkeit des Motors ermittelt und an die Funktion `motor_set_velocity(int velocity)` übergeben.

$$\upsilon_{Motor} = \frac{MOTOR\_ENDSTOP}{sw\_endstop} * 360^{\circ} * \mid \upsilon_{Lenkrad} \frac{Umdr.}{Min} \mid * 60s \tag{5.3}$$

Der zweite und der dritte Thread sind nur im Zustand *Online* aktiv. Im zweiten Thread, der sich in der Funktion `steering_wheel_analog_in_request_task(void *arg)` befindet, wird mit der Nachricht ID 0x20A der Wert des analogen Eingangs angefordert. Abhängig von dem erhaltenen Wert der Geschwindigkeit wird die Federsteifigkeit der Lenkung eingestellt (vgl. Abschnitt 5.4). Auf diese Weise erreicht man, dass bei sehr hoher Geschwindigkeit das Lenkrad stärker zur Nullposition geführt wird. Der dritte Thread ist in der Funktion `steering_wheel_status_request_task(void *arg)` platziert und versendet an das Endgerät einmal pro Periode die Statusanfrage (Nachricht ID 0x200). Die Antwort enthält den Status der Hardware bzw. den Fehlercode. Sollte sich der Zustandsautomat der Hardware im Fehlerzustand befinden, wird in den Zustand *Error* gewechselt, um den Fehler zu korrigieren. Die Modi der Lenkung unterscheiden sich in der Position der Endanschläge. Beim Umschalten aus dem Modus *Cruise* in den Modus *Sport* kann vorkommen, dass sich das Lenkrad ausserhalb der neuen Endanschlägen befindet. Das könnte dazu führen, dass der Motor maximale Kraft entwickelt und die Benutzer verletzt. Dies lässt die Regelung des Lenkrades nicht zu und nimmt den Fehlerzustand an. Um das zu vermeiden, wird der Zustandsautomat in den Zustand *Change Mode* versetzt, der das Lenkrad in die erlaubte Position bringt und die Endanschläge stufenweise versetzt. Nachdem der gewünsche Modus eingestellt ist, kehrt der Zustandsautomat in den Zustand *Online* zurück.

Fehlerbehandlung

Befindet sich der Zustandsautomat im Fehlerzustand, so wird versucht, den Fehler zu quittieren. Anhand der Fehlernummer wird über die weitere Vorgehensweise entschieden. Es gibt nur drei Arten von Fehlern, die innerhalb der Software korrigiert werden können.

`0x0020` CAN Bus Fehler

`0x0080` CAN-Watchdog Zeitüberschreitung

`0x0100` Unerlaubte Position (|Position| > 14440°)

Beim Quittieren von CAN Bus Fehler und CAN-Watchdog Zeitüberschreitung wird in den Zustand *Normal Ready* gewechselt und damit der Motor wieder eingeschaltet. Bei der unerlaubten Position muss wieder referenziert werden, daher ist der nächste Zustand *Reference Ready*.

Parametrierung über Schalter

Die Positionen der Schalter werden im *Online* Zustand ein Mal pro Periode ausgelesen (vgl. Abschnitt 5.2.1) und entsprechend das Verhalten der Lenkung angepasst. Dabei ist zu beachten, dass die GPIOs, die für die Schalter verwendet werden, mit dem RS232 Modul gemeinsam genutzt werden. Daher funktionieren die Schalter beim eingesteckten RS232 Modul nicht.
Mit dem Schalter 1 werden die *Cruise*- oder *Sportmodi* eingestellt. Der mit der Funktion `controller_get_steer_mode()` ausgelesene Modus wird als Parameter an die Funktion `steering_wheel_set_mode(sw_mode_t mode)` übergeben. Der *Sportmodus* hat eine kleinere Distanz zwischen den Endanschlägen, daher kann beim Umschalten von *Cruise* zum *Sport* die Position des Lenkrades außerhalb der Endanschläge liegen. Ein weiteres Problem, das mit dem Umschalten der Modi verbunden ist, ist die Skalierung der Position des Rades, die anhand der Position der Endanschläge berechnet wird. Verändert sich die Position der Endanschläge deutlich, so wird das Rad stärker bzw. schwächer ausgelenkt.
Der Schalter 2 erlaubt, die Verletzung der Echtzeitkommunikation zu emulieren. Eine TT Nachricht kann zur Laufzeit nicht in eine RC oder BE Nachricht umgewandelt werden. Dafür müssen alle Knoten und der Switch umprogrammiert werden. Die Verletzung der Echtzeitkommunikation lässt sich dadurch emulieren, dass das Versenden der Nachrichten abhängig von der Schalterposition teilweise ausgelassen wird. Die mittels der Funktion `controller_get_comm_mode()` erhaltene Kommunikationsart wird an `steering_wheel_set_rt_mode(sw_rt_mode_t)` übergeben und damit der entsprechende Modus eingestellt.

5.2.4 Modul „Force Sensor“

Dieses Modul übernimmt die Aufgabe der Kommunikation mit dem im Abschnitt 3.3 vorgestellten CAN-Messverstärker. Das Modul besteht aus einem Thread, der den Zustandsautomat enthält. Die Aufgabe des Zustandsautomaten ist die Überwachung der richtigen Reihenfolge der Ausführung, sowie die Parametrierung bzw. Skalierung der Hardware. Im Modul wird die Querkraft, die auf das Rad einwirkt, gemessen und an dem Lenkrad eingestellt. So wird die Verbindung des Lenkrades mit der Straße hergestellt und damit der Force Feedback Pfad realisiert.

Parametrierung

Die in der Datei *force_sensor.h* enthaltenen Konstanten beschreiben die einzelnen Parameter des Moduls. Die Konstante `CANOPEN_ADDR_FORCE_SENSOR` enthält die Geräte-ID und muss mit der in der Hardware eingestellten Geräte-ID übereinstimmen. Damit der an dem CAN-Messverstärker angeschlossene Kraftsensor die richtigen Werte liefert, müssen mittels vier Punkten die Spannen der Skalierung beschrieben werden.

`FS_VALUE_SCALING_ZERO` = 0 $\frac{mV}{V}$

`FS_VALUE_SCALING_ZERO_PHYS` = 0 N

`FS_VALUE_SCALING_MAX` = 1 $\frac{mV}{V}$

`FS_VALUE_SCALING_MAX_PHYS` = 5000 N

Mit den richtig eingestellten Parametern liefert der Messverstärker den vorzeichenbehafteten ganzzahligen Wert der Kraft in Newton.

Initialisierung

Threads sind Programmteile, die parallel zur Initialisierung ablaufen können. Daher wäre falsch, wenn der Thread vor der abgeschlossenen Initialisierung auf nicht initialisierte Variablen zugreifen würde. Deswegen hat der Thread den Zustand *Offline*, der den Zugriff auf die Variablen verbietet. Erst durch den Aufruf von `force_sensor_init()` werden die Variablen initialisiert und im Anschluss daran der Thread des Zustandsautomaten freigegeben.

Zustandsautomat

Der in der Funktion `force_sensor_state_mashine_task(void *arg)` platzierte Zustandsautomat hat die Aufgabe den Threadablauf zu steuern und gegebenenfalls auf die auftretenden Fehler zu reagieren. Weiterhin repräsentiert der Zustandsautomat das Gedächtnis des Moduls. Alle Parameter werden jedes Mal sequenziell gesetzt, was die Anzahl der Zustände spürbar erhöht. Das hat den Vorteil, dass die Software auch nach dem Austausch der Hardware oder der Wiederherstellung der Werkseinstellungen weiterhin funktioniert.
Die Abbildung 5.2 stellt den Zustandsautomat des CAN Messverstärkers dar. Jeder Zustand enthält die Information über die Nachricht, die er versendet, und das CANopen konforme Objekt bzw. den Subindex, der für das Setzen der Parameter zuständig ist. Am Anfang ist der Zustandsautomat im Zustand *Offline*. Erst wenn die Initialisierung der Variablen abgeschlossen ist, geht er in den Zustand *Scaling Zero* und fängt an, die Hardware zu parametrieren. Die Zustände *Scaling Zero*, *Scaling Zero Phys*, *Scaling MAX*, *Scaling MAX Phys* setzen die entsprechenden in der Headerdatei definierten Parameter des Sensors. Dadurch

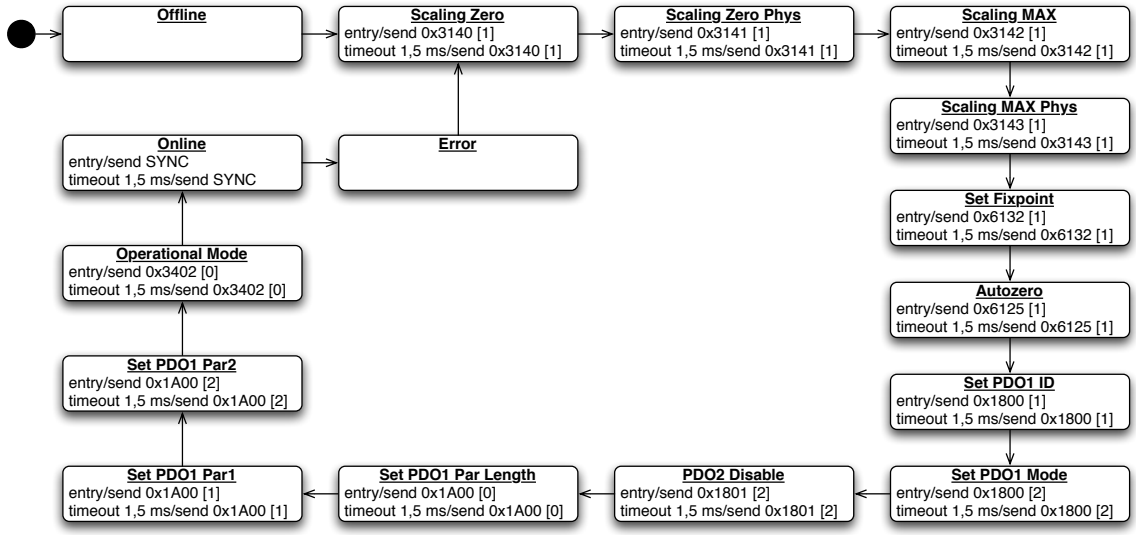


Abbildung 5.2: Zustandsautomat des Moduls Force Sensor

weiß der Messverstärker, wie er die elektrische Messung des Sensors in die physikalischen Werte umsetzen soll. Im Zustand *Set Fixpoint* wird die Position des Dezimalpunktes auf 0 gesetzt, weil die Nachkommastellen für den Fahrer keine spürbaren Werte darstellen. Für die richtige Justierung wird im Zustand *Autozero* der Messwert im unbelasteten Zustand als Referenz-Nullpunkt übernommen. Die Bewegung des Motors kann diese Justierung verfälschen. Daher muss darauf geachtet werden, dass der Motor in diesem Zustand still steht. Aus diesem Grund wird der Motor erst dann initialisiert, wenn der Kraftaufnehmer *online* ist. Durch den Aufruf der Funktion `steering_wheel_is_online()` wird die Bereitschaft der Hardware festgestellt. Der nächste Zustand *Set PDO1 ID* setzt einen CANopen konformen Identifier (Nachrichten ID 0x183), mit dem die Nachricht versendet werden wird. Im Zustand *Set PDO1 Mode* wird das Ereignis gesetzt, bei dem die Nachricht mit dem Messwert und dem Status versendet wird. Als solches Ereignis muss jede SYNC Nachricht gelten. Im Messverstärker ist standardmäßig der PDO2 aktiviert und belastet unnötig den Bus. Daher schaltet der Zustand *PDO2 Disable* den PDO2 aus. Der Zustand *Set PDO1 Par Length* legt die Anzahl der zu mappenden Parameter fest. In dem *Set PDO1 Par1* wird der Status auf die ersten 4 Byte der Nachricht eingestellt. Mit dem Zustand *Set PDO1 Par2* wird der *int32* Messwert auf die letzten 4 Byte der Nachricht gemappt. Die resultierende Größe der Nachricht 0x183 ist daher 8 Byte. Damit die PDO Kommunikation aktiviert wird, muss der *Operational Mode* gesetzt werden. Danach befindet sich der Kraftaufnehmer im Zustand *Online*. In diesem Zustand wird die SYNC Nachricht versendet, die den CAN Messverstärker und den Motor zum Versenden der gemappten Werte auffordert. Da der Messverstärker vor dem Motor initialisiert wird, wird die SYNC Nachricht vom Modul Force Sensor versendet. Anhand des

Status wird der Zustand der Hardware überprüft. Ist der Zustand fehlerfrei, wird der erhaltene Messwert als Parameter an die Funktion `steering_wheel_set_force(int force)` übergeben. Die Funktion berechnet aus der Kraft das Drehmoment und stellt dieses am Lenkrad ein. Dafür wird die Kraft mit dem Radius des Lenkrades in Meter multipliziert. Sollte ein Fehler auftreten, so wechselt der Zustandsautomat in den Zustand *Error*.

Fehlertoleranz

Befindet sich der Zustandsautomat im Fehlerzustand, so wird versucht den Fehler zu beseitigen. Anhand des Status wird über die weitere Vorgehensweise entschieden. Es gibt nur zwei Arten von Fehlern, die innerhalb der Software korrigiert werden können.

- *Messwert ungültig*
- *Skalierfehler*

Im Fall eines Skalierfehlers handelt es sich um den Fehler, der nur innerhalb der Initialisierung vorkommen konnte. Daher wird die Initialisierung wiederholt. Ist der Messwert ungültig, so wird an das Lenkrad als Messwert die 0 übergeben. Bei den restlichen Fehlern handelt es sich um schwerwiegende Fehler, die mit der Software nicht beseitigt werden können. Daher bleibt das Modul bis zur Beseitigung im Fehlerzustand.

5.2.5 Modul „Motor“

Dieses Modul übernimmt die Aufgabe der Kommunikation mit dem im Abschnitt 3.2 vorgestellten Servoverstärker. Das Modul besteht aus einem in der Funktion `motor_state_mashine_task(void *arg)` platzierten Thread, der den Zustandsautomat beinhaltet. Die Aufgabe des Zustandsautomaten ist die Überwachung der richtigen Reihenfolge der Ausführung sowie die Parametrierung der Hardware. Das Modul hat die Aufgabe, die berechnete Position bzw. die Positionierungsgeschwindigkeit des Motors an den Servoverstärker zu versenden und damit einzustellen.

Parametrierung

Die in der Datei *motor.h* enthaltenen Konstanten beschreiben die einzelnen Parameter des Moduls. Die Konstante `CANOPEN_ADDR_MOTOR` enthält die Geräte-ID und muss mit der in der Hardware eingestellten Geräte-ID übereinstimmen. Mit der Konstante `MOTOR_VALUE_HOMING_MODE` wird der Referenzmodus festgelegt. Bei dem *default* Modus 21 handelt es sich um die Referenzierung mit negativem Referenzschalter ohne Nullimpulsauswertung (Jenaer Antriebstechnik GmbH (2009)).

Initialisierung

Threads sind Programmteile, die parallel zur Initialisierung ablaufen können. Daher wäre falsch, wenn der Thread vor der abgeschlossenen Initialisierung auf nicht initialisierte Variablen zugreifen würde. Deswegen hat der Thread den Zustand *Offline*, der den Zugriff auf die Variablen verbietet. Erst durch den Aufruf von `motor_init()` werden die Variablen initialisiert und erst dann wird der Thread des Zustandsautomaten freigegeben.

Zustandsautomat

Der in der Funktion `motor_state_mashine_task(void *arg)` platzierte Zustandsautomat hat die Aufgabe, den Threadablauf zu steuern und gegebenenfalls auf die auftretenden Fehler zu reagieren. Weiterhin repräsentiert der Zustandsautomat das Gedächtnis des Moduls. Alle Parameter werden jedes Mal sequenziell gesetzt, was die Anzahl der Zustände spürbar erhöht. Das hat den Vorteil, dass die Software auch nach dem Austausch der Hardware oder der Wiederherstellung der Werkseinstellungen weiterhin funktioniert. Die Abbil-

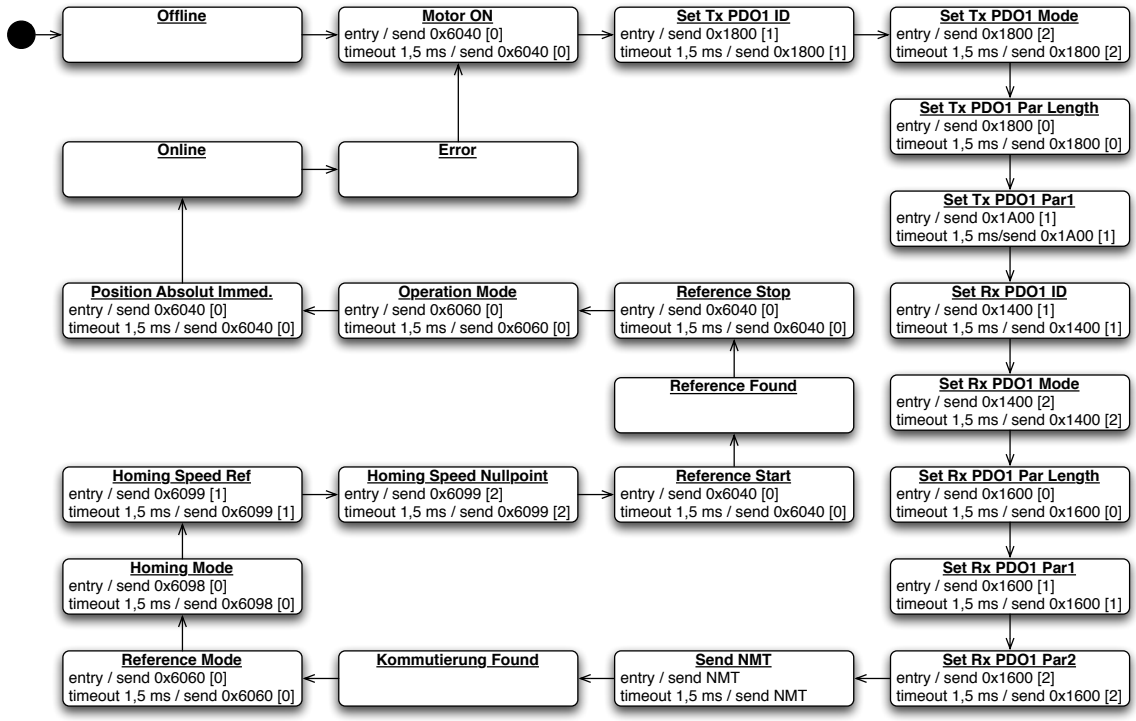


Abbildung 5.3: Zustandsautomat des Moduls Motor

dung 5.2 stellt ein Zustandsautomat des CAN Messverstärkers dar. Jeder Zustand enthält die Information über die Nachricht, die er versendet, und das CANopen konforme Objekt

bzw. den Subindex, der für das Setzen des Parameters zuständig ist. Am Anfang ist der Zustandsautomat *offline*. Nachdem die Variablen initialisiert wurden, geht der Automat in den Zustand *Motor ON*. Die mechanische Bewegung des Motors kann die Initialisierung des Kraftaufnehmers verfälschen. Daher wird die Nachricht zum Einschalten des Motors erst dann versendet, wenn der Kraftaufnehmer *online* ist. Nachdem die Bestätigung erhalten wurde, nimmt der Zustandsautomat den Zustand *Set Tx PDO1 ID* an und der PDO Mapping von Tx Parameter beginnt. Im Zustand *Set Tx PDO1 ID* wird ein CANopen konformer Identifier gesetzt (Nachricht ID 0x182), mit dem die Nachricht versendet wird. Im *Set Tx PDO1 Mode* wird das Ereignis gesetzt, bei dem die Nachricht mit dem Messwert und dem Status versendet wird. In diesem Fall wird die auf jede SYNC Nachricht eingestellt. Der Zustand *Set Tx PDO1 Par Length* legt die Anzahl der zu mappenden Parameter fest. In dem *Set Tx PDO1 Par1* wird der Status auf die ersten 4 Byte der Nachricht gemappt. Die resultierende Größe der Nachricht 0x182 ist daher 4 Byte. Nachfolgend wird die Nachricht gemappt, mit der die Position des Motors bzw. die Geschwindigkeit der Positionierung gesetzt wird. Der Zustand *Set Rx PDO1 ID* setzt den 0x252 als Identifier, der mit der PDO1 assoziiert werden soll. Das Setzen der Position gehört zum kritischen Pfad. Um die Verzögerungen zu verringern, wird im Zustand *Set Rx PDO1 Mode* die Übernahme der ankommenden Daten sofort nach der Veränderung eingestellt. So verkürzt sich die Totzeit im Vergleich mit dem synchronisierten Ansatz um die Länge der SYNC Nachricht und den Abstand zwischen dem Setzen der Position und dem Versenden der SYNC Nachricht. Der Zustand *Set Rx PDO1 Par Length* legt die Anzahl der gemappten Parameter fest. Im Zustand *Set Rx PDO1 Par1* wird die Position in den ersten 4 Byte der Nachricht gemappt. Der Zustand *Set Rx PDO1 Par2* mappt die Positionierungsgeschwindigkeit auf die restlichen 4 Byte der Nachricht. Daher beträgt die Größe der Nachricht 8 Byte. Nachdem Rx PDO Mapping erfolgreich durchgeführt wurde, muss das Endgerät in den Operational Mode umgeschaltet werden. Dafür wird im Zustand *Send NMT* die NMT Nachricht versendet und damit ist das Endgerät bereit die PDOs zu versenden bzw. zu empfangen. Im Zustand *Kommutierung Found* wird gewartet, bis die Kommutierung gefunden wurde. Erst wenn das entsprechende Bit im Statusregister gesetzt wurde, wird der nächste Zustand angenommen. Der Zustand *Reference Mode* schaltet die Hardware in den Modus zur Referenzfahrt ein. Die Art der im Zustand *Homing Mode* gesetzten Referenzfahrt ist von der Hardware abhängig und im Abschnitt 3.2.2 beschrieben. Die Zustände *Homing Speed Ref* und *Homing Speed Nullpoint* setzen die Geschwindigkeit der Referenzfahrt. Im Zustand *Reference Start* wird die Referenzfahrt gestartet. Nach dem erfolgreichen Start geht der Zustandsautomat in den Zustand *Reference Found* und wartet, bis die Referenzfahrt zu Ende ist. Erst wenn das entsprechende Bit im Statusregister gesetzt wurde, wird die nächste Zustandstransition ausgeführt. Der Zustand *Reference Stop* stoppt die Referenz. Damit die Positionierung erfolgen kann, muss die Hardware in den *Operation Mode* umgeschaltet werden. Weiterhin wird im Zustand *Position Absolut Immediately* sofortiges Positionieren eingestellt. Danach befindet sich das Modul im *Online* Zustand und ist bereit die Position an die Hardware zu versenden. Die mit den Funktionen `motor_set_position(int position)`

und `motor_set_velocity(unsigned int velocity)` gesetzten Parameter werden in die Nachricht 0x252 verpackt und versendet. Sollte ein Fehler auftreten, so wechselt der Zustandsautomat in den Zustand *Error*.

Fehlertoleranz

Befindet sich der Zustandsautomat im Fehlerzustand, so wird versucht, den Fehler zu beseitigen. Anhand des Status wird über die weitere Vorgehensweise entschieden. Es gibt nur zwei Arten von Fehlern, die innerhalb der Software korrigiert werden können.

- Fault
- Schleppfehler

Fault bezeichnet einen allgemeinen Fehler. Der Automat bleibt im Fehlerzustand bis der Fehler beseitigt wurde. Beim Schleppfehler handelt es sich um den Fall, bei dem die Kraft des Motors kleiner als die Gegenkraft war und infolgedessen die Positionsabweichung einen bestimmten Wert überschritten hat. Da es sich um mechanisch bewegte Teile handelt, schaltet sich der Motor automatisch aus. In einer realen Situation muss der Motor wieder eingeschaltet werden, daher enthält der Zustandsautomat die Zustandstransition aus dem *Error* in den Zustand *Motor ON*. Bei einem Demonstrator handelt es sich um einen Vorführgerät. Ursache für den Schleppfehler kann daher zum Beispiel ein eingeklemmter Finger sein. Aus diesem Grund ist das Einschalten des Motors sehr gefährlich und der Zustandsautomat bleibt im Zustand *Error*. Durch einen Neustart kann das System wieder in den Betriebszustand gebracht werden.

5.2.6 Debug

Um die Wartbarkeit der Software zu erhöhen existiert ein Debug Modus. Mit der Definition der nachfolgend beschriebenen Konstanten können die textuelle Ausgaben aktiviert werden. Dadurch kann die Funktionalität der Zustandsmaschinen und der einzelnen Parameter überprüft werden. Dabei soll beachtet werden, dass einige Module parallel ablaufen und die Ausgaben sich vermischen können. Daher ist es hilfreich, die Debug Modi einzeln zu aktivieren.

- Mit der Definition der Konstante `__DEBUG_SW_STATEMASHINE__` werden die Ausgaben des Lenkrad-Zustandsautomaten eingeschaltet.
- Die Konstante `__DEBUG_SW_ANGLE__` gibt den empfangenen Lenkradwinkel an der Konsole aus. Eine Ausgabe findet nur statt, wenn sich der Zustandsautomat des Lenkrades im Zustand *Online* befindet. Die Werte werden periodisch ausgegeben.

- Um den Zustandsautomat des Kraftaufnehmers zu testen muss die Konstante __DEBUG_FS_STATEMASHINE__ definiert werden.
- Mittels der Konstante __DEBUG_FS_MEASURE__ wird der Messwert des Kraftaufnehmers am Terminal ausgegeben. Die Ausgabe der Werte wird erst dann ausgeführt, wenn sich der Zustandsautomat des Kraftaufnehmers im Zustand *Online* befindet.
- Die Definition der Konstante __DEBUG_MOTOR_STATEMASHINE__ schaltet den Debug-Modus für den Motor Zustandsautomat ein.
- Für die Überprüfung der Berechnung der Motorposition muss die Konstante __DEBUG_MOTOR_POSITION__ definiert werden. Die Position enthält die Anzahl der Inkremente vom Nullpunkt.
- Für die Überprüfung der Berechnung der Positionierungsgeschwindigkeit muss die Konstante __DEBUG_MOTOR_POS_VELOCITY__ definiert werden. Die Geschwindigkeit wird in Inkrementen pro Sekunde angegeben.

Die textuellen Ausgaben erfolgen über die RS232 Schnittstelle am Terminal. Die Parameter zur Einstellung sind im Abschnitt 3.4.1 beschrieben. Es ist darauf zu achten, dass das RS232 Modul die GPIOs der Schalter ebenfalls nutzt und beim eingesteckten Modul die Schalter nicht funktionieren. Zudem verbrauchen die textuellen Ausgaben CPU Rechenzeit und können daher das Verlauf des Programms um mehr als 350 μs verzögern.

5.3 Scheduling

In einem präemptiven Multitasking-Betriebssystem steuert der Scheduler den Zugriff auf die gemeinsamen Ressourcen. Zu den Ressourcen zählt auch die Rechenzeit, um die die Prozesse untereinander konkurrieren. Die Bearbeitungszeit eines Threads setzt sich aus der Laufzeit des Threads und der Zeit, die für den Kontextwechsel benötigt wird, zusammen. In Echtzeitsystemen hat die Verletzung der Deadline gravierende Auswirkungen (vgl. Abschnitt 2.4.3). Ist die dem Thread zugeteilte Rechenzeit überschritten, so wird der Vorgang unterbrochen, auch wenn die Berechnungen noch nicht abgeschlossen sind. Weiterhin brauchen die Prozesse, die den Zugriff auf die Hardware wie zum Beispiel CAN bzw. Real-time Ethernet Stack steuern, ebenfalls Rechenzeit und müssen in Betracht gezogen werden. In einem Regelprozess darf Scheduling nicht allzu pessimistisch durchgeführt werden, denn dadurch erhöht sich die Totzeit des Systems (siehe Abschnitt 2.4.1). In den folgenden Abschnitten werden die einzelnen Messungen vorgestellt und daraus eine Schedulingtabelle abgeleitet.

5.3.1 Laufzeiten der Nachrichten

Bei der Echtzeitkommunikation teilen zeitkritische Nachrichten und nicht zeitkritische Nachrichten ein gemeinsames Übertragungsmedium. Für Scheduling sind zeitkritische Nachrichten von besonderer Bedeutung, weil sie zu einem bestimmten Zeitpunkt versendet werden sollen. In der Abbildung 5.4 werden alle Nachrichten abhängig von der Anwendung in die Real-time Ethernet Nachrichtenklassen aufgeteilt. Die SDO Kommunikation, die für die In-

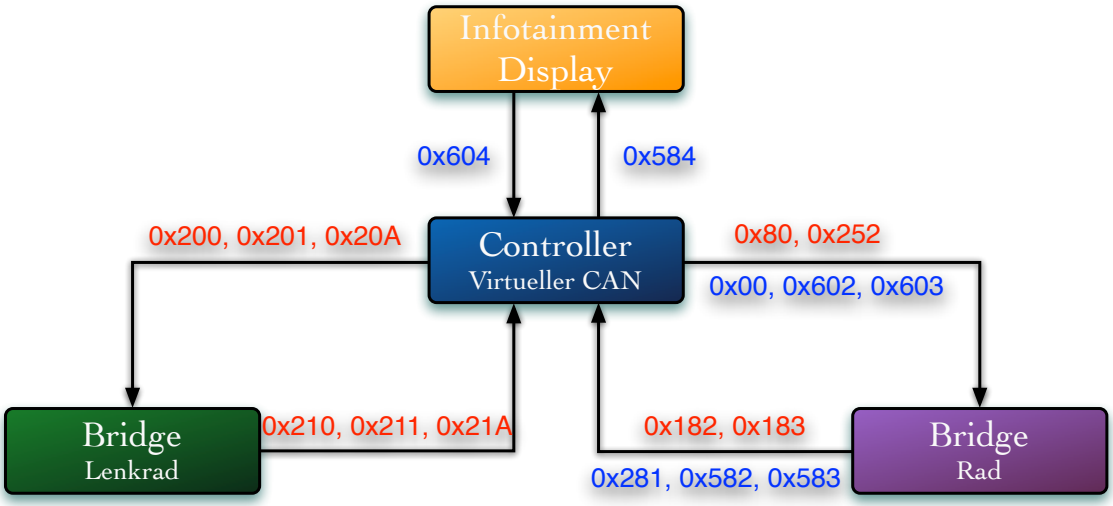


Abbildung 5.4: Die Richtung der CAN Kommunikation zwischen den Knoten

itialisierung der Hardware genutzt wird, ist nicht an einen bestimmten Zeitpunkt gebunden und gehört daher zu der RC Klasse. Die CAN Identifier der RC Kommunikation sind blau dargestellt. Wenn die Initialisierung abgeschlossen ist, befinden sich die Endgeräte im Betriebszustand und senden in regelmäßigen Abständen die Werte bzw. den Status des Gerätes. Diese Nachrichten sind zeitkritisch, gehören damit zu der TT Klasse und werden mit roter Farbe markiert. Um die Zugehörigkeit der Antwort zu der Anfrage übersichtlicher darzustellen, werden alle Nachrichten in der Tabelle 5.3 zusammengefasst. Weiterhin enthält die Tabelle entsprechende Laufzeiten der Nachrichten sowie die minimale und die maximale Reaktionszeit der Endgeräte. Bei manchen Nachrichtenpaaren weicht von Periode zu Periode die minimale Reaktionszeit von der maximalen ab. Daher soll die Schedulingtabelle so gestaltet werden, dass die aktuelle Nachricht beim Zugriff bereits empfangen ist. Wird zum Beispiel versucht die Antwortnachricht abzuholen, bevor sie empfangen wurde, so wird der alte Inhalt abgeholt. Der neue Wert wird erst in der nächsten Periode verfügbar, was die Totzeit der Regelung im schlimmsten Fall um eine bis mehrere Perioden verlängert.

Tx ID	Größe Byte	Zeit μs	Rx ID	Größe Byte	Zeit μs	Reaktionszeit μs		RTE Klasse
						Min	Max	
0x0	0	57	0x281	0	52	176	393	RC
0x80	0	41	0x182	4	93	180	260	TT
0x80	0	41	0x183	8	121	250	300	
0x200	8	115	0x210	8	117	20	60	
0x201	8	113	0x211	8	116	20	60	
0x20A	2	74	0x21A	8	114	20	60	
0x252	8	115	-	-	-	-	-	
0x602	8	115	0x582	8	114	179	385	RC
0x603	8	115	0x583	8	115	343	3803	
0x584	8	111	0x604	8	117	20	100	

Tabelle 5.3: Laufzeiten und die Reaktionszeiten der Nachrichtenpaaren

5.3.2 Laufzeit der Prozesse

Zu der Laufzeit der Prozesse gehört nicht nur die Rechenzeit im Thread, sondern auch die Zeit, die für Kontextwechsel benötigt wird. Laut der Messung von Kai Müller (2011) beträgt die Zeit des Kontextwechsels $4,6\mu s$. Der Scheduler Startet den Kontextwechsel früher, damit das Programm genau zum angegebenen Zeitpunkt gestartet wird. Der Rücksprung nimmt $1,5\mu$s in Anspruch. Die Summe von diesen beiden Werten legt den minimalen Abstand zwischen zwei Prozessen bzw. Ereignissen fest. Die Tabelle 5.4 enthält die Liste der Threads

Thread Name	Thread Max μs	Ges. Zeit μs
Force sensor Statemashine	16,07	22,17
Lenkrad Statemashine	22,34	28,44
Lenkrad Status Task	13,36	19,36
Analog IN Task	12,82	18,92
Motor Statemashine	14,19	20,29

Tabelle 5.4: Laufzeiten der Threads

mit den entsprechenden Threadlaufzeiten. Die Gesamtzeit, die die CPU für den Thread braucht, ist eine Summe von Threadlaufzeit, Funktionsaufrufzeit und Rücksprungzeit. Damit die Schedulingstabelle vollständig entworfen werden kann, werden noch die Messungen des Versenden über CAN bzw. virtuellen CAN benötigt. Laut der Messung von Jan Kamieth (2011) beträgt die Dauer des Versendens über CAN $30-45\mu$s und über Real-time Ethernet $25-30\mu$s.

5.3.3 Schedulingtabelle für CAN Kommunikation

Die in den vorherigen Abschnitten beschriebenen Laufzeiten bilden die Grundlage für die Berechnung des Zeitverhaltens für die im Abschnitt 4.2 vorgestellte Architektur. Die Endgeräte und der Controller befinden sich in einem gemeinsamen CAN Netzwerk, daher soll darauf geachtet werden, dass die Nachrichten möglichst optimal die Bandbreite ausnutzen. Der in der Abbildung 5.5 dargestellter Zeitplan beschreibt das Verhalten des Controllers und

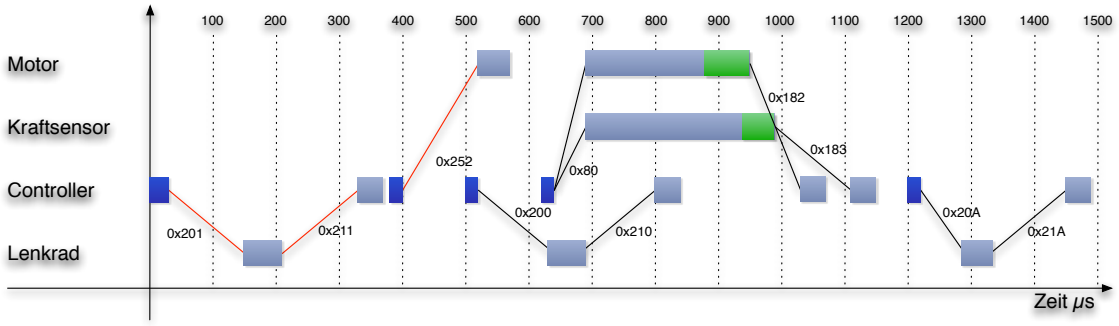


Abbildung 5.5: Scheduling der CAN basierten Kommunikation

der Endgeräte in der zeitlichen Abfolge. Die Gesamtperiode des Zeitplans beträgt 1,5 ms. Das Ziel der Regelung ist die Einstellung der Auslenkung des Rades. Da die Auslenkung von der Position des Lenkrades abhängig ist, nennen wir diesen Pfad kritisch. Der kritische Pfad ist mit den roten Linien markiert. Mit den grauen Balken wird die Bearbeitungszeit bzw. Rechenzeit dargestellt. Die Bearbeitungszeit der Nachrichten 0x182 und 0x183 variiert sehr stark. Die Differenz zwischen der minimalen und der maximalen Abarbeitungszeit wird grün dargestellt. Die Threads des Controllers werden mit den blauen Kästchen visualisiert. Beim Zeitpunkt 0 μs wird der Thread `steering_wheel_state_mashine_task()` aktiviert, der die Anfrage der Position (Nachricht 0x201) an das Lenkrad versendet. Der Winkel des Lenkrades (Nachricht 0x211) wird empfangen, abgearbeitet und zum Zeitpunkt 380 μs im Thread `motor_state_mashine_task()` an das Motor versendet. Zum Zeitpunkt 500 μs fordert der Thread `steering_wheel_status_request_task()` den Status des Lenkrades an. Um die Bandbreite besser zu nutzen, versendet der Thread `force_sensor_state_mashine_task()` beim Zeitpunkt 620 μs parallel zu der Reaktionszeit des Lenkrades die SYNC Nachricht, die den Status bzw. den Messwert vom Motor und des Kraftaufnehmers anfordert. Die Nachricht 0x183 enthält die Messung der Force Feedback Kraft und wird im nächsten Zyklus mit der Nachricht 0x201 gesetzt. Die Kollision der Nachrichten 0x182 und 0x183 wird auf der Sicherungsschicht der CAN Implementierung aufgelöst (siehe Abschnitt 2.2.2) und findet erst nach dem Versenden der Nachricht 0x252 statt. Beim Zeitpunkt 1200 μs wird der Thread `steering_wheel_analog_in_request_task()` gestartet, der mit der Nachricht 0x20A den analogen Eingang abfragt. Der empfangene Wert

emuliert die Geschwindigkeit des Fahrzeugs (vgl. Abschnitt 5.4). Die Totzeit der Regelung ist die Zeit, die für die Abarbeitung des kritischen Pfades benötigt wird und beträgt 520 μs.

5.3.4 Schedulingtabelle für Real-time Ethernet Kommunikation

Die in den vorherigen Abschnitten beschriebenen Laufzeiten bilden die Grundlage für die Berechnung des Zeitverhaltens für die im Abschnitt 4.1 vorgestellte Architektur. Die Schedulingtabelle für die Real-time Ethernet basierte Architektur unterscheidet sich von der CAN Netzwerk Architektur in mehreren wesentlichen Aspekten. Die Real-time Ethernet Implementierung reserviert die ersten 100 μs der Periode für die Synchronisierung der globalen Zeit. Die Synchronisierung hat aus der Sicht des Schedulers die höchste Priorität und würde jeden Thread verdrängen. In der Abbildung 5.6 ist diese Phase gelb dargestellt. Weiterhin enthält die Architektur zusätzliche Komponenten, die die Nachrichten untereinander austauschen bzw. übermitteln. Die CAN Endgeräte werden in zwei räumlich getrennte Busse aufgeteilt. Daher kann die CAN Kommunikation parallel ablaufen und trotz der ausfallenden Synchronisationsphase bleibt die Regelperiode von 1,5 ms erhalten. Der erste CAN Bus und die zugehörigen Endgeräte sind oberhalb der *Rad-Bridge* platziert und schließen diese mit ein. Der zweite CAN Bus befindet sich unterhalb der *Lenkrad-Bridge* und schließen die *Lenkrad-Bridge* mit ein. Zudem ist ein Real-time Ethernet Switch enthalten, der die Bridges und den Controller verbindet. Auch hier stellen die

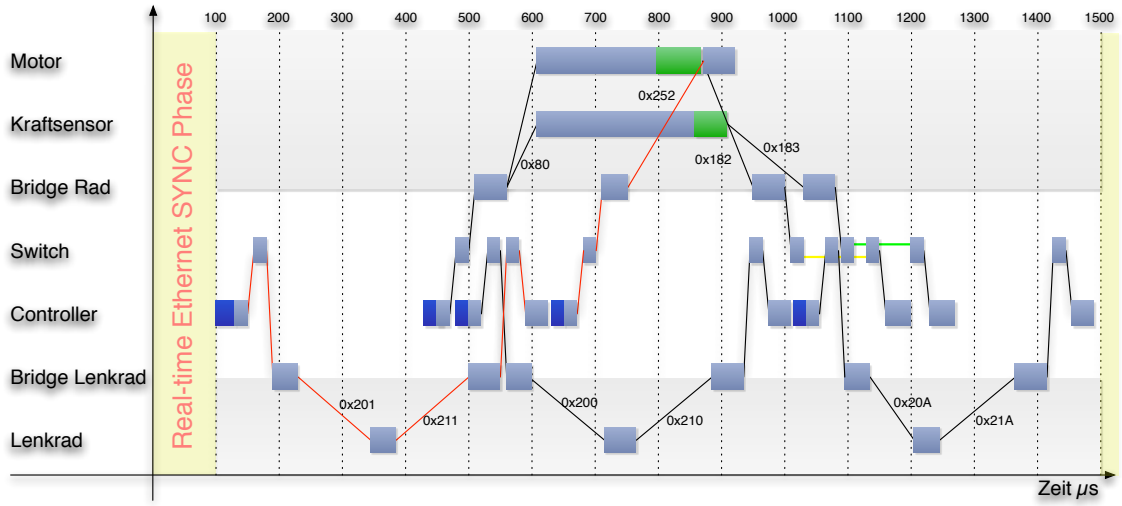


Abbildung 5.6: Scheduling der Real-time Ethernet basierten Kommunikation

grauen Balken die Bearbeitungszeit bzw. Rechenzeit dar. Im Vergleich zu der Abbildung 5.5 sind hier zusätzlich der Real-time Ethernet Switch und die Bridges am kritischen Pfad beteiligt. Die Threads des Controllers werden mit den blauen Kästchen dargestellt. Beim

Zeitpunkt 100 μs wird der Thread `steering_wheel_state_mashine_task()` aktiviert, der die Anfrage der Position (Nachricht 0x201) an das Lenkrad versendet. Der Thread `force_sensor_state_mashine_task()` versendet beim Zeitpunkt 430 μs an den Kraftsensor und den Motor die SYNC Nachricht. Am Zeitpunkt 480 μs fordert der Thread `steering_wheel_status_request_task()` den Status des Lenkrades an. Da der Real-time Ethernet Switch und die Bridges Full-duplex sind, kollidieren die Nachrichten bei 550 μs nicht. Der Winkel des Lenkrades (Nachricht 0x211) wird empfangen, abgearbeitet und zum Zeitpunkt 630 μs im Thread `motor_state_mashine_task()` an den Motor versendet. Beim Zeitpunkt 1020 μs wird der Thread `steering_wheel_analog_in_request_task()` gestartet, der mit der Nachricht 0x20A den analogen Eingang abfragt. Die Kollision zwischen der Nachricht 0x182 und der Nachricht 0x183 wird auf der Sicherungsschicht der CAN Implementierung aufgelöst (siehe Abschnitt 2.2.2). Die Nachricht 0x182 wird in der Switch zwischengespeichert und zum Zeitpunkt 1150 μs an den Controller versendet. Der Controller empfängt den Messwert der Force Feedback-Kraft bei 1160 μs und wird im nächsten Zyklus mit der Nachricht 0x201 an das Lenkrad einstellen. Auch die Nachricht 0x183 wird zwischengespeichert und erst bei 1220 μs den Switch verlassen. Durch den zusätzlichen Real-time Ethernet Kommunikationsaufwand verlängert sich der kritische Pfad und damit auch die Totzeit der Regelung auf 870 μs.

5.3.5 Einstellung des Systems

Der Scheduler erwartet in der Datei *config.c* die Schedulingtabelle, die das Verhalten des Geräts beschreiben. Damit man nicht für jede Bridge und den Controller ein eigenständiges Projekt benötigt, werden die Schedulingtabellen mittels der Precompiler directiven generiert. So hat man alle Daten in einem Projekt, was für Übersichtlichkeit und eine bessere Wartbarkeit sorgt. In der Datei *config_demostrator_main.h* wird festgelegt, welche Rolle die Implementierung annehmen soll. Das hat den Vorteil, dass man mit einer kleinen Veränderung an einer zentralen Stelle das Verhalten des Systems anpasst und somit die manuelle Veränderung der Schedulingtabelle vermeiden wird.

config.c

Abhängig von den in der Datei *config_demostrator_main.h* definierten Konstanten werden entsprechende Schedulingtabellen in die *config.c* eingefügt und damit eine gültige Konfiguration des Schedulers gewährleistet.

config_demostrator_main.h

In dieser Datei wird die Rolle der Implementierung durch die Definition der entsprechenden Konstante festgelegt. Nachfolgend werden die verfügbaren Rollen beschrieben.

- `CONTROLLER` Bei der Definition dieser Konstante befindet sich die Implementierung in Rolle des Controllers. Mit der Konstante `CAN_VIRTUAL` wird die Art der CAN Kommunikation festgelegt. Ist die Konstante definiert, so wird die Schedulingtabelle aus dem Abschnitt 5.3.4 eingebunden. In diesem Fall dient der Controller im Real-time Ethernent Netzwerk als SYNC-Master. Er versendet in der SYNC-Phase die SYNC Nachrichten um das Netzwerk auf die gemensame Zeit zu bringen. Wenn die Konstante nicht definiert ist, handelt es sich um einen CAN basierten Controller und es wird die im Abschnitt 5.3.3 beschriebene Schedulingtabelle genutzt. Beide Tabellen befinden sich in der Datei *config_controller.h*.

- `BRIDGE_WHEEL` Die Konfiguration entspricht der in der Abbildung 5.6 dargestellten *Bridge Lenkrad*. Die Schedulingtabelle befindet sich in *config_wheel.h*

- `BRIDGE_STEER` Die Konfiguration entspricht der in der Abbildung 5.6 dargestellten *Bridge Rad*. Die Schedulingstabelle befindet sich in *config_steer.h*

Es darf nur eine Rolle zur Zeit festgelegt werden, weil jede Konstante eine Schedulingtabelle einfügt. Bei mehreren Einträgen werden Tabellen mit den gleichen Namen eingefügt, was der Compiler als Fehler erkennt.

5.4 Geschwindigkeitsabhängiges Lenkungsverhalten

In einem herkömmlichen Fahrzeug beeinflusst die Fahrgeschwindigkeit das Lenkverhalten. Bei sehr hoher Geschwindigkeit wird das Lenkrad beim Loslassen in die Mitte zurückgeführt. Das sogenannte Rücklaufverhalten ist von der Bauweise der Lenkung abhängig. Da eine lineare Lenkungskurve angestrebt wird, nehmen wir das Rücklaufverhalten linear an. Um dieses Verhalten visuell darzustellen muss die Geschwindigkeit des Fahrzeuges ermittelt werden. Da die tatsächliche Geschwindigkeit nicht verfügbar ist, muss eine andere Lösung gefunden werden. Dafür wird an den im Abschnitt 3.1.5 beschriebenen analogen Eingang des Lenkrades ein Potentiometer angeschlossen, mit dem im Laufe der Zeit eine sich verändernde Geschwindigkeit emuliert wird. Der eingestellte Wert wird mittels einer CAN Nachricht ausgelesen und an den Wertebereich der Federsteifigkeit skaliert. Mit der Nachricht 0x20A wird der Messwert der Geschwindigkeit angefordert. Die Antwortnachricht 0x21A enthält den Messwert des analogen Eingangs, der sich im Wertebereich zwischen 0 und 0x3FF befindet. Abhängig von dem Modus wird der Wert der Federsteifigkeit ermittelt und an das Lenkrad übermittelt. Im Modus *Demo* ist diese Funktionalität nicht verfügbar, da dieser nur für den Zweck der Veranschaulichung des Demonstrators entwickelt wurde und mit der Geschwindigkeit nichts zu tun hat. Die Steilheit der Federsteifigkeit ist von der eingestellten Reibung und Dämpfung des Modus abhängig (siehe Abschnitt 5.5.1). Ist die Dämpfung zu klein eingestellt, so kommt es zu Überschwingungen um den Mittelpunkt. Die Einstellungen

für die Modi *Cruise*, *Sport* und die *default* Einstellungen für *Custom* befinden sich in der Datei *steering_wheel.h*.

5.5 CAN Inteface zur Parametrierung des Controllers

Für die Verbesserung der Usability-Eigenschaften des Produktes wird die Schnittstelle entwickelt, die zusätzlich zu den fest programmierten Modi wie *Sport* und *Cruise* einen Modus *Custom* zur Verfügung stellt, indem das Lenkverhalten bzw. Force Feedback beliebig angepasst werden kann. Mittels der externen Schnittstelle hat man die Möglichkeit, den Modus des Lenkrades umzuschalten, sowie eine eigene Konfiguration einzustellen. Diese Konfiguration erlaubt die Lenkung eines beliebigen Fahrzeuges nachzuahmen (siehe Abschnitt 5.1.3). Die Schnittstelle implementiert das CANopen Protokoll (vgl. Abschnitt 2.3). Mit dem im ersten Byte der Nachricht angegebenen Zugriffscode kann man zwischen Lese- bzw. Schreibzugriff unterscheiden. Die Geräte-ID der externen Schnittstelle ist 4, die Parametrierung erfolgt mittels SDO Zugriff (vgl. Abschnitt 2.3.2). Die resultierende SDO Tx Adresse nach dem CANopen Protokoll entspricht 0x604. Mit der ID 0x584 wird die SDO Antwort versendet.

5.5.1 Parametrieren der Lenkung

Alle Parameter der Lenkung sind über das CANopen-konforme Objekt 0x6060 erreichbar. Mit dem Subindex werden die einzelnen Parameter angesprochen. Die Tabelle 5.5 enthält die Übersicht der Parameter, die zur Parametrierung des Lenkungsverhaltens genutzt werden. Nachfolgend wir auf jeden Parameter der Liste einzeln eingegangen. Die in der Tabelle zusammengefassten Parameter besitzen unterschiedliche Datentypen. Der Datentyp gibt die Größe des Parameter in Bits und die Art der Daten an. So entspricht der Datentyp *u8* dem vorzeichenlosen 1 Byte großen Parameter. Das Beispiel beschreibt den lesenden bzw. schreibenden Zugriff auf den Subindex 0 und stellt damit Modus *Custom* ein.

Objekt	Subindex	Datentyp	Wertebereich	Beschreibung
0x6060	0	u8	0, 1, 2, 3	Umschalten von Modi
	1	u8	0 - 8	Force Feedback
	2	u16	180° - 540°	Endanschläge
	3	u8	0 - 100%	Wirkungsgrad
	4	u16	0 - 5000	Reibung
	5	u16	0 - 500	Dämpfung

Tabelle 5.5: Die Übersicht der Objekte für die Parametrierung des Controllers

```
Lesen        604 8 40 60 60 00 xx xx xx xx
Antwort      584 8 4f 60 60 00 01 xx xx xx

Schreiben    604 8 2f 60 60 00 02 xx xx xx
Antwort      584 8 60 60 60 00 02 xx xx xx
```

Beim lesenden Zugriff muss darauf geachtet werden, dass der erste Byte der Antwort den CANopen-konformen Datentyp enthält, der ausgewertet werden muss (vgl. Abschnitt 2.3.2).

Umschalten des Modus

Mit dem Subindex 0 wird zwischen vier verfügbaren Modi umgeschaltet. Jeder der Modi hat eine bestimmte Funktion, die nachfolgend erklärt wird.

0 *Cruise*. In diesem Modus entspricht die Lenkung einem gewöhnlichen Fahrzeug. Die einfache Lenkung ist perfekt für den Stadtverkehr geeignet. Die Endanschläge befinden sich bei 540°.

1 *Sport*. Dieser Modus emuliert die sehr direkte Lenkung eines Rennwagens mit Endanschlägen bei 180°. Ein sehr hoher Wirkungsgrad der Lenkung erfordert viel Kraft beim Lenken.

2 *Custom*. Der Modus erlaubt die Parameter der Lenkung beliebig anzupassen. Die Default-Werte entsprechen dem *Cruiseodus*. Weitere Parameter werden mit den Subindexen 1 - 5 eingestellt.

3 *Demo*. Beim Einschalten diesen Modus bewegt sich das Lenkrad bzw. das Rad abwechselnd erst in eine Richtung, beim Erreichen der Endposition in die andere Richtung. So soll bei der Vorführung die Aufmerksamkeit der vorbeigehenden Zuschauer auf den Demonstrator gelenkt werden.

Force Feedback

Der Subindex 1 lässt im Modus *Custom* die Wirkung der Force Feedback Kraft in den definierten Stufen anpassen. Dies erlaubt einerseits durch die Minderung der Kraft den Komfort beim Fahren, anderseits durch die Erhöhung der Kraft eine bessere Verbindung zur Straße zu erreichen. Die Tabelle 5.6 enthält die Skalierung der Kraft abhängig vom Wert des Parameters. Beim *default* Wert 4 wird die Kraft, die auf das Rad einwirkt, 1:1 in den Drehmoment umgerechnet. Kleinere Werte verringern die Kraft, größere erhöhen sie entsprechend der Tabelle.

Force Feedback Parameter	0	1	2	3	4	5	6	7	8
Drehmoment Multiplikator	$\frac{1}{5}$	$\frac{1}{4}$	$\frac{1}{3}$	$\frac{1}{2}$	1	2	3	4	5

Tabelle 5.6: Skalierung der Force Feedback Kraft abhängig vom Parameter

Endanschläge

Mittels Subindex 2 wird der Winkel des Lenkrades eingestellt, bei dem in beiden Richtungen von der Nullposition eine spürbare Endgrenze realisiert wird. Der Parameter ist eine vorzeichenlose ganzzahlige Zahl 2 Byte lang, mit dem Wertebereich von 180° bis 540°.

Wirkungsgrad

Mit dem Subindex 3 wird der Wirkungsgrad eingestellt, der beschreibt, wie effektiv die Kraft vom Rad an das Lenkrad übertragen wird. Der Wirkungsgrad wird in Prozent eingegeben und hat den Wertebereich von 0 bis 100%. Bei 100% wird die Kraft 1:1 übertragen. Bei höheren Werten erhöht sich der Lenkkraftaufwand.

Reibung

Unter Subindex 4 verbirgt sich die Reibung, die die Hemmung der Bewegung zwischen der Fahrbahn und dem Rad beschreibt. Zudem hängt die Reibung von der Lenkübersetzung, Lenkelastizität und Achsenkinematik ab. Ist die Reibung zu groß, wird die Lenkung zu stumpf und verhindert den Kontakt zur Fahrbahn. Sonst sind Vibrationen und Drehschwingungen spürbar. Eine richtig eingestellte Reibung erhöht die Kundenzufriedenheit.

Dämpfung

Der Subindex 5 stellt die Dämpfung ein, die die kontinuierliche Minderung der Bewegung oder Schwingung beschreibt. Wenn einem gedämpften, schwingenden System keine neue Energie zugeführt wird, verebbt seine Schwingung, die Amplitude nimmt ab. Somit wird die Überschwingung verhindert.

5.5.2 Fehlerbehandlung

Sollte der Zugriff fehlerhaft sein, dann enthält die SDO Antwort im ersten Byte der Nachricht die Konstante 0x80. Anhand des in Bytes 5-8 enthaltenen CANopen-konformen Fehlercodes kann der Fehlergrund festgestellt werden. Die Tabelle 5.7 enthält die Liste der möglichen Fehler, die nachfolgend beschrieben werden.

Fehler Nr.	Define aus *can.h*	Beschreibung
0x00000206	CANOPEN_ERROR_OBJECT_DOESNT_EXIST	Objekt existiert nicht
0x11000906	CANOPEN_ERROR_SUBINDEX_DOESNT_EXIST	Subindex existiert nicht
0x00000008	CANOPEN_ERROR_COMMON	Allgemeiner Fehler
0x12000706	CANOPEN_ERROR_DATA_TYPE_TOO_BIG	Datenlänge zu gross
0x13000706	CANOPEN_ERROR_DATA_TYPE_TOO_SMALL	Datenlänge zu klein

Tabelle 5.7: Liste der Fehler mit den entsprechenden Fehlernummern

- `CANOPEN_ERROR_OBJECT_DOESNT_EXIST` tritt dann auf, wenn an den Controller ein nicht implementiertes Objekt übergeben wurde.
- `CANOPEN_ERROR_SUBINDEX_DOESNT_EXIST` wird von dem Controller gemeldet, wenn der Zugriff auf ein nicht existierenden Subindex ausgeführt wurde.
- `CANOPEN_ERROR_COMMON` der allgemeine Fehler tritt dann auf, wenn die Grenze des Wertebereiches verletzt wurde oder die Nachrichtengröße weniger als 4 Byte lang ist und damit keine Daten enthalten kann.
- `CANOPEN_ERROR_DATA_TYPE_TOO_BIG` dieser Fehler kommt dann vor, wenn die angegebene Länge der Daten größer ist, als die Länge der Variablen.
- `CANOPEN_ERROR_DATA_TYPE_TOO_SMALL` dieser Fehler tritt dann auf, wenn die angegebene Länge der Daten kleiner ist, als die Länge der Variablen.

Kapitel 6

Qualitätsicherung

In diesem Kapitel wird das entwickelte Produkt auf die Erfüllung der gestellten Qualitätsanforderungen überpüft. Zudem wird die Korrektheit gegen Anforderungen sowie deren vollständige Umsetzung untersucht. Weiterhin werden die Aspekte Zuverlässigkeit, Wartbarkeit sowie Benutzerfreundlichkeit analysiert.

6.1 Verifikation

Innerhalb der Verifikation werden Fehler lokalisiert und behoben. Die Fehler müssen so früh wie möglich entdeckt werden, weil die Fehlersuche im Gesamtsystem viel umständlicher ist als in den kleineren Teilen. Zudem ist das Debuggen von Echtzeitanwendungen, besonders in eingebetteten Systemen sehr umständlich. Um den verdeckten Fehlern auf die Spur zu kommen, werden alle Module dem Modultest[13] unterzogen. Mit den im Abschnitt 5.2.6 beschriebenen Debug Parametern wurden das spezifizierte Verhalten der Zustandsautomaten und die Berechnungen der Parameter überprüft. Diese Tests lassen sich jederzeit wiederholen, was die Wartbarkeit der Software wesentlich erhöht. Bei den Tests wurden sowohl erlaubte als auch unerlaubte Eingaben durchgeführt. Die Modultests zeigen die Anwesenheit der Fehler und nicht deren Abwesenheit (vgl. International Software Testing Qualification Board (2005)). Daher müssen die dynamischen Tests mit anderen Methoden abgesichert werden. Dafür wurden zusätzlich Codereviews durchgeführt, um die logischen Fehler zu entdecken. Nachdem einzelne Module auf die Erfüllung der Funktionalität verifiziert worden sind, müssen diese im Zusammenspiel mit den restlichen Programmteilen getestet werden. Dafür werden die zu einem Gesamtsystem zusammengefügte Module einem Integrationstest[14] unterzogen. Des Weiteren wurden Schnittstellentests durchgeführt, um die Kommuni-

[13] Der Modultest ist der Softwaretest, der die Funktionalität des Moduls sicherstellt. Ziel des Modultests ist es, frühzeitig Programmfehler zu entdecken

[14] Bei einem Integrationstest werden die aufeinander abgestimmten Komponenten in dem Gesamtsystem getestet und auf die Erfüllung der Anforderungen überprüft

kation zwischen den Komponenten zu überprüfen. Da die CAN Schnittstelle zur Anpassung des Lenkverhaltens die Grundlage für die nachfolgende Bachelorarbeit darstellt, muss sie besonders grünldich getestet werden. Dafür wurden CANopen-konforme Objekte bzw. Subindizes sowohl mit gültigen als auch mit ungültigen Werten überprüft. Weiterhin wurden Parameter in den Gültigkeitsbereichen, aber auch in den Grenzwertbereichen und außerhalb der gültigen Werte ausführlich getestet.

6.2 Validierung

Bei der Validierung muss die Erfüllung der Kundenanforderungen nachgewiesen werden. Dafür muss das System in beiden Architekturen getestet werden. Die Eignungsprüfung erfolgt auf der Grundlage des im Abschnitt 4.4.1 aufgestellten Anforderungsprofils. Zuers wurde das System innerhalb des CAN netwerkes getestet. Alle Anwendungsszenarien wurden ausführlich durchgeführt und dabei auf die Benutzerfreundlichkeit untersucht. Um die Nutzererwartungen zu prüfen, wurden die Testpersonen sowie die Mitentwickler gebeten, die Funktionalität des Produktes zu beurteilen. Dabei wurden die Anmerkungen und Verbesserungsvorschläge aufgenommen und anschließend umgesetzt. Weiterhin wurden die Aspekte Ausfallsicherheit und Benutzbarkeit untersucht. Zudem wurden Reviews mit den Teammitgliedern zur Aufdeckung von Unklarheiten und irrtümlichen Annahmen durchgeführt. Während der Validierung wurde festgestellt, dass der Motor des Lenkrades außerhalb der Endanschlägen nicht genügend Kraft entwickelt. Mit kleinen Anstrengungen ließ sich der Endanschlag überwinden. Daher wurde ein anderer, leistungsfähigerer Motor bestellt und eingebaut. Die Software wurde entsprechend angepasst und getestet. Beim Test des Gesamtsystems ist eine Anomalie fesgestellt worden, dass bei einer bestimmten Konstellation die Regelung des Lenkrades die Statusnachricht mit dem falschen Inhalt versendet hat. Anstatt des Zustandes der Regelung bzw. des Statuses versendete diese die Position bzw. die Geschwindigkeit. Die Statusnachricht wird ausgewertet um die geforderte Robustheit zu erreichen. Bei einem falschen Inhalt der Nachrich führt der Zustandsautomat des Moduls *Steering Wheel* falsche Transition aus und landet unnötigerweise in der Referenzierung. Die Nachricht ist auch mittels proprietärer Software CANAlyzer auf dem CAN-Bus sichtbar. Mit dieser Software ist es nicht möglich, den Fehler zu reproduzieren und damit ein Programmfehler auszuschliessen. Die Ursache der Anomalie muss noch untersucht werden. Die Real-time Ethernet basierte Architektur besteht aus drei aufeinander aufbauenden Bachelorarbeiten und befindet sich noch in der Integrationsphase, daher konnte nicht ausfühlich getestet werden. Da die Funktionalität der Steer-by-Wire Lenkung in einem CAN Bussystem erfolgreich getestet wurde, kann davon ausgegangen werden, dass bei einem richtig eingestellten Scheduling die Lenkung einwandfrei funktioniert.

Kapitel 7

Zusammenfassung und Fazit

Nachdem die Realisierung der Steer-by-Wire Lenkung abgeschlossen ist, beurteilt dieses Kapitel die erreichen Ergebnisse.

7.1 Zusammenfassung

In Zusammenarbeit von Studenten aus den Studiengängen Maschinenbau, Mechatronik und Informatik ist ein Steer-by-Wire System entstanden, das die Fähigkeiten der Real-time Ethernet untersucht und anschaulich darstellt. Zudem wurden die Vorteile der Steer-by-Wire Technologie analysiert und dem Endbenutzer demonstriert. Innerhalb dieser Arbeit wurden die Anforderungen des zu entwickelnden Produktes untersucht und daraus eine sehr flexible Architektur entworfen. Aufgrund der Bridgearchitektur kann die Regelung sowohl in einem CAN Netzwerk als auch innerhalb des Real-time Ethernet Netzwerkes betrieben werden.

Damit diese Arbeit nicht nur für einen engen Kreis der Fachleute verständlich ist, vermittelt Kapitel 2 grundlegendes Wissen, das zum Verstehen der Arbeit erforderlich ist. Es enthält eine kurze Erläuterung von CAN, CANopen Protokolle sowie die Einführung in die Regelungstechnik. Weiterhin werden die Vorteile von Steer-by-Wire gegenüber den herkömmlichen Lenksystemen dargestellt.
Das Kapitel 3 stellt das verwendete Hardware vor. Dabei werden die die Regelungen des Lenkrades und des Motors mit der Lenkgetriebe beschrieben. Weiterhin wird der Messverstärker des Kraftsensors erläutert, der zur Realisierung von Force Feedback erforderlich ist. Zudem wird der verwendete Mikrocontroller und das Betriebssystem beschrieben. Anschliessend wird der TTTech Switch vorgestellt und seine Besonderheiten erklärt.
Im Kapitel 4 werden die Anforderungen an die Architektur gestellt. Aufgrund der geforderten Flexibilität der Anwendung werden zwei Architekturen vorgestellt, die je nach Bedarf verwendet werden können. Zudem werden die Anwendungsfälle festgelegt und die Softwarearchitektur konzipiert.

Das Kapitel 5 beschreibt die Umsetzung der Softwarearchitektur, sowie die Zustandsautomaten die Initialisierung der Hardware bzw. Ausfallsicherheit gewährleisten. Zudem werden die Abläufe in der zeitlichen Abfolge zu den Schedulingtabellen zusammengefasst und grafisch dargestellt. Weiterhin enthält das Kapitel die Beschreibung der CANopen konformen Schnittstelle zur Einstellung des Lenkverhaltens mittels des Infotainment Centers.
Die gestellten Anforderungen werden in Kapitel 6 mit den Ergebnissen der Umsetzung verglichen und analysiert. Dabei werden die Verifikation bzw. Validierung durchgeführt.

7.2 Fazit

Die für diese Arbeit gestellten Anforderungen wurden vollständig erfüllt. Der Demonstrator vermittelt die Möglichkeiten von Steer-by-Wire, zeigt Vorteile und Nachteile. Die Lenkung ist sehr flüssig und präzise. Das Rad bewegt sich ohne Verzögerung. Für den Einsatz in einem echten Fahrzeug ist es jedoch nicht geeignet, weil die benötigte Ausfallsicherheit erst durch Redundanz erreichbar ist.

Es hat sich herausgestellt, dass beim Umschalten zwischen den Modi sich die Auslenkung des Rades verändert. Der Wertebereich der Endanschläge liegt zwischen 180° und 540°. Die Abweichung nahe dem Nullpunkt ist nicht sehr groß. Würde beispielsweise das Lenkrad im *Cruisemodus* bei 180° stehen, so entspricht es einem drittel des maximalen Radeinschlages. In einem *Sportmodus* ist es ein maximaler Einschchlag, was einer Verdreifachung entspricht. Beim Umschalten in der Kurve würde die Regelung den Einschlag des Rades verändern, was zu den unerwünschten Effekten führen könnte. Eine mögliche Lösung dieses Problems wäre, beim Umschalten den Einschlag des Rades zu halten. Daraus folgt, dass das Lenkrad positioniert werden muss. Die Kraft, die auf das Lenkrad eingestellt wird, könnte dem Fahrer die Hände verdrehen oder sogar verletzen. Andererseits muss der Fahrer zu jedem Zeitpunkt die Möglichkeit haben das Fahrzeug zu lenken. Dafür muss die Kraft, mit der der Fahrer entgegenwirkt, überwacht werden um entscheiden zu können, ob der Fahrer lenkt. Eine einfachere Lösung ist, während der Fahrt das Umschalten nur in einem bestimmten Einschlagintervall zu erlauben.

Ein weiterer Aspekt ist die dynamische Anpassung der Lenkungskurve. Beim Parken ist es wichtig mit kürzeren Bewegungen auf die auftretenden Situationen zu reagieren. Bei hohen Geschwindigkeiten ist es anders, heftige Bewegungen können das Auto zum Überschlag bringen. Mit der Steer-by-Wire Lenkung kann eine beliebige Lenkkurve dynamisch einstellen. Daher soll untersucht werden ob die in der Abbildung 2.2 dargestellte Lenkungskurven für den Mensch vorteilhaft sind, weil ein nichtlineares Verhalten sehr schwer nachvollziehbar und daher gewöhnungsbedürftig ist.

Danksagung

Ein besonderer Dank gilt meinen beiden Korrektoren, Prof. Dr.-Ing. Franz Korf, der mich durch seine hilfreichen Anregungen und seine Geduld immer wieder unterstützt hat, sowie Prof. Dr.-Ing. Andreas Meisel.
Nicht zuletzt möchte ich mich bei meiner Familie bedanken, die mir während dieses Studiums sehr viel Geduld entgegengebracht und mich die ganze Zeit moralisch unterstützt hat.
Weiterhin bedanke ich mich beim Core Team für die Möglichkeit, an so einem tollen Projekt teilzunehmen, und bei allen Kollegen, die mir immer mit gutem Rat zur Seite standen.
Bei meinen Freunden möchte ich mich für die vielen Stunden Korrekturlesen sowie die interessanten Beiträge und Änderungsvorschläge bedanken.

Literaturverzeichnis

[Bundesministerium der Justiz 2011] BUNDESMINISTERIUM DER JUSTIZ: *§38 Lenkeinrichtung*. 2011. – URL http://www.gesetze-im-internet.de/stvzo/__38.html. – Zugriffsdatum: 2011-05-21

[CAN in Automation (CiA) 2011] CAN IN AUTOMATION (CIA): *Canopen*. 2011. – URL http://www.can-cia.org/index.php?id=canopen. – Zugriffsdatum: 2011-05-06

[Communication over Real-time Ethernet (CoRE)] COMMUNICATION OVER REAL-TIME ETHERNET (CORE): *Communication over Real-time Ethernet*. – URL http://www.informatik.haw-hamburg.de/core.html

[Discenzo 2000] DISCENZO, Frederick M.: *US Patent: Steer by wire system with feedback*. 2000. – URL http://www.google.de/patents?hl=de&lr=&vid=USPAT6097286&id=CR0EAAAAEBAJ&oi=fnd&dq=6097286&printsec=abstract

[Ewbank 2003] EWBANK: *US Patent: Redundant Steer-by-Wire System*. 2003. – URL http://www.google.de/patents?id=41EOAAAAEBAJ&pg=PA2&dq=6208923&hl=en&ei=V4fuTdHoMsmSOp3IvJII&sa=X&oi=book_result&ct=result&resnum=4&ved=0CC0Q6AEwAw

[HBM GmbH] HBM GmbH (Veranst.): *DF30CAN digiCLIP Betriebsanleitung*. A1748-2.0 de. – URL http://www.hbm.de

[Hilscher GmbH] Hilscher GmbH (Veranst.): *NXHX500 ETM Betriebsanleitung*. 1.0

[Hommel 2001] HOMMEL: *US Patent: Fault tolerant electromechanical steer-by-wire steering actuator*. 2001. – URL http://www.google.de/patents?hl=de&lr=&vid=USPAT6208923&id=O6oGAAAAEBAJ&oi=fnd&dq=steer-by-wire&printsec=abstract

[Institut für Arbeitswissenschaft Darmstadt 2011] INSTITUT FÜR ARBEITSWISSENSCHAFT DARMSTADT: *Prinzipdarstellung eines Steer-by-Wire-Systems* . April 2011. –

URL http://www.arbeitswissenschaft.de/website/projects/foi_automot/ergonomisch_152/de/de_ergonomisch_univer_1.php. – Zugriffsdatum: 2011-05-18

[International Organisation for Standardization 2003] INTERNATIONAL ORGANISATION FOR STANDARDIZATION: *ISO 11898 1-3*. 2003. – URL http://www.iso.org. – Zugriffsdatum: 2011-05-06

[International Software Testing Qualification Board 2005] INTERNATIONAL SOFTWARE TESTING QUALIFICATION BOARD: *Certified Tester, Foundation Level Syllabus*. 2005. – URL http://www4.in.tum.de/lehre/vorlesungen/sw_test/ss06/ISTQB_Lehrplaene/ISTQB_D.pdf

[Jan Kamieth 2011] JAN KAMIETH: Entwurf und Implementation einer Mikrocontroller-basierten Bridge zur Kopplung von CAN Bussen über Time Triggered Realtime Ethernet / Hochschule für Angewandte Wissenschaften Hamburg (HAW Hamburg). 2011. – Forschungsbericht

[Jenaer Antriebstechnik GmbH 2009] Jenaer Antriebstechnik GmbH (Veranst.): *Ecovario Objektverzeichnis*. 11 2009. – URL http://www.jat-gmbh.de/download/ecosoftware/Handbuch_Objektverzeichnis_ECO_de_091130.pdf

[Kai Müller 2011] KAI MÜLLER: Time-Triggered Ethernet für eingebettete Systeme: Design, Umsetzung und Validierung einer echtzeitfähigen Netzwerkstack-Architektur / HAW Hamburg. 2011. – Forschungsbericht

[Leiter u. a. 2008] LEITER, Ralf ; MISSBACH, Stefen ; WALDEN, Michael: *Fahrwerk: Fahrwerk, Lenkung, Reifen und Räder*. Vogel Buchverlag, 2008. – ISBN 978-3-8343-3001-7

[Mann u. a. 2009] MANN, Heinz ; SCHIFFELGEN, Horst ; FRORIEP, Rainer: *Einführung in die Regelungstechnik : analoge und digitale Regelung*. Pearson Studium, 2009. – ISBN 978-3-446-41765-6

[SensoDrive GmbH 2010] SENSODRIVE GMBH: *SENSOWheel Betriebsanleitung v4p0*, 06 2010

[Tanenbaum 2003] TANENBAUM, Andrew S.: *Verteilte Systemen Grundlagen und Paradigmen*. Pearson Studium, Oktober 2003. – ISBN 978-3-8273-7057-4

[Tanenbaum 2009] TANENBAUM, Andrew S.: *Moderne Betriebssysteme - 3., aktualisierte Auflage*. Pearson Studium, Oktober 2009. – ISBN 978-3-8273-7342-7

[TTTech GmbH] TTTech GmbH (Veranst.): *Betriebsanleitung*. 1.0

[Wallentowitz und Reif 2006] WALLENTOWITZ, Henning ; REIF, Konrad: *Handbuch Kraftfahrzeugelektronik*. Springer Verlag, 2006. – ISBN 978-3-528-03971-X

[Zeltwanger 2001] ZELTWANGER, Holger: *CANopen*. VDE Verlag, 2001. – ISBN 978-3-8007-2448-0

[Zimmermann und Schmidgall 2008] ZIMMERMANN, Werner ; SCHMIDGALL, Ralf: *Bussysteme in der Fahrzeugtechnik - 3. Auflage*. Vieweg + Teubner, September 2008. – ISBN 978-3-8348-0447-1

Glossar

Arbitrierung
: Die Arbitrierung ist ein Verfahren zum Lösen der Zugriffskonflikten beim Zugriff mehreren Nutzer auf eine gemensame Ressource.

Deadline
: Deadline ist ein bestimmter Zeitpunkt, zu dem eine Aufgabe erledigt werden muss. Die Verletzung der Deadlin hat gravierende Auswirkungen auf das System.

harte Echtzeitanforderung
: Harte Echtzeitanforderung bedeutet dass anhand von Hardwarespezifikationen und Modellrechnungen eine beweisbare obere Grenze für die Dauer eines Vorgangs angegeben werden kann.

Integrationstest
: Bei einem Integrationstest werden die aufeinander abgestimmten Komponenten in dem Gesamtsystem getestet und auf die Erfüllung der Anforderungen überprüft.

Modultest
: Der Modultest ist der Softwaretest, der die Funktionalität des Moduls sicherstellt. Ziel des Modultests ist es, frühzeitig Programmfehler zu entdecken.

Nibble
: Ein Nibble ist eine 4 Bit große Datenmenge, sie wird meistens mit hexadezimalen Zahlen dargestellt.

Payload
: Payload bezeichnet die Nutzdaten, die übertagen werden sollen.

Periode
: Als Periode bezeichnet man ein Zeitintervall zwischen zwei, sich in gleichen Abständen wiederholenden Ereignissen.

Redundanz

Mit Redundanz wird die mehrfache Ausführung der sicherheitrelevanten Ressourcen beschrieben. Damit wird ein störungsfreier Betrieb ermöglicht.

Scheduling

Unter Scheduling verbirgt sich die Erteilung dem Prozess nach einem festgelegten Zeitplan den Zugriff auf die Ressource.

Steer-by-Wire

Steer-by-Wire ist ein Lenksystem, bei dem die mechanische Verbindung mit den Aktoren und Sensoren durch eine elektrische Leitung ersetzt wird.

Totzeit

Der Begriff Totzeit T_T steht für die Zeit, die von der Änderung am Eingang bis zur Änderung am Ausgang vergeht.

weiche Echtzeitanforderung

Weiche Echtzeitanforderung bedeutet, dass die Dauer eines Vorgangs die angegebene Obergrenze üblicherweise nicht überschreitet.

X-by-Wire

X-by-Wire ist ein System, bei dem die mechanische Verbindung mit den Aktoren und Sensoren durch eine elektrische Leitung ersetzt wird.

Abkürzungsverzeichnis

ASR	Antriebsschlupfregelung
BE	Best-Effort
CAN	Controller Area Network
CiA	CAN in Automation e. V.
COB	Communication Object
COB-ID	Communication Object Identifier
CoRE	Communication over Real-time Ethernet
CRC	Cyclic Redundancy Check
CSMA/CR	Carrier Sense Multiple Access / Collision Resolution
ESP	Elektronisches Stabilitätsprogramm
FIQ	Fast Interrupt Request
GPIO	General Purpose Input/Output
HAL	Hardware Abstraction Layer
HAW Hamburg	Hochschule für Angewandte Wissenschaften Hamburg
ISR	Interrupt Service Routine
LSB	Least Significant Byte
MSB	Most Significant Byte
MTBF	Mean Time Between Failures

NMT	Network Managment
PDO	Prozess Daten Objekt
RC	Rate-Constrained
RTE	Real-time Ethernet
SDO	Service Daten Objekt
StVZO	Straßenverkehrs-Zulassungs-Ordnung
TT	Time-Triggered

Abbildungsverzeichnis

Sachregister

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) bzw.§24(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 22. August 2011 Vitalij Stepanov