HOCHSCHULE MITTWEIDA
University of Applied Sciences

MASTERARBEIT

Frau
Sina Vanessa Lipke

Konzeption und Implementierung des Antriebsstranges als eigenständige CAN-Bus-Steuereinheit in einem forensischen Demonstrator

2021

Fakultät **Angewandte Computer- und Biowissenschaften**

MASTERARBEIT

Konzeption und Implementierung des Antriebsstranges als eigenständige CAN-Bus-Steuereinheit in einem forensischen Demonstrator

Autorin:
Sina Vanessa Lipke

Studiengang:
Cybercrime/Cybersecurity

Seminargruppe:
CY19wC-M

Erstprüfer:
Prof. Dr. rer. nat. Dirk Labudde

Zweitprüfer:
M. Sc. Christian Georgi

Mittweida, August 2021

Faculty of **Applied Computer Sciences and Biosciences**

MASTER'S THESIS

Conception and implementation of the drive train as an independent CAN bus control unit in a forensic demonstrator

Author:
Sina Vanessa Lipke

Course of Study:
Cybercrime/Cybersecurity

Seminar Group:
CY19wC-M

First Examiner:
Prof. Dr. rer. nat. Dirk Labudde

Second Examiner:
M. Sc. Christian Georgi

Mittweida, August 2021

Bibliografische Angaben

Lipke, Sina Vanessa: Konzeption und Implementierung des Antriebsstranges als eigenständige CAN-Bus-Steuereinheit in einem forensischen Demonstrator, 101 Seiten, 54 Abbildungen, 12 Tabellen, Hochschule Mittweida, University of Applied Sciences, Fakultät Angewandte Computer- und Biowissenschaften

Masterarbeit, 2021

Referat

In der vorliegenden Masterarbeit wird der, im Jahr 2018 im Application Center Microcontroller entwickelte, forensische Demonstrator für das Controller Area Network (CAN) analysiert und auf Basis dessen ein Redesign konzipiert, entwickelt und getestet. Gemäß der in dieser Arbeit vermittelten Grundlagen zu aktuellen Bussystemen der Automobilindustrie und ihrer Datenübertragung, werden entsprechende CAN-Nachrichten implementiert und auf den CAN-Bus gesandt. Die Auswertung dieser Botschaften erfolgt durch die CAN-Analysesoftware BUSMASTER. Eine entsprechende Visualisierung der Daten wird durch die, für den BUSMASTER entwickelte, grafische Oberfläche realisiert.

Abstract

In this master thesis, the forensic demonstrator for the Controller Area Network (CAN) that was developed in the Application Center Microcontroller in 2018 is analyzed. On this basis, a redesign is created, developed, and tested. According to the basics of the automotive industry's current bus systems and their data transmission, appropriate CAN messages are implemented and sent to the CAN bus. The evaluation of these messages is done using the CAN analysis software BUSMASTER. A corresponding visualization of the data is realized using the graphical user interface developed for BUSMASTER.

I. Inhaltsverzeichnis

II. Abbildungsverzeichnis

III. Tabellenverzeichnis

IV. Abkürzungsverzeichnis

ACK	Acknowledge
ADC	Analog-to-Digital Converter
AP	Access Point
BLE	Bluetooth Low Energy
CAN	Controller Area Network
CAS	Car Access System
CiA	CAN in Automation
CRC	Cyclic Redundancy Check
CTR	Control
ECU	Electronic Control Unit
EN	Enable
EoF	End-of-Frame
EVB	Evaluationsboard
GND	Ground
GUI	Graphical User Interface (dt. grafische Benutzeroberfläche)
IFS	Interframe-Space
ISR	Interrupt Service Routine
MISO	Master In - Slave Out
MOSI	Master Out - Slave In
PWM	Pulsweitenmodulation
RTR	Remote Transmission Request
SCK	Serial Clock
SDI	Serial Data In
SDO	Serial Data Out
SoF	Start-of-Frame
SPI	Serial Peripheral Interface
SS	Slave-Select
TCP	Transmission Control Protocol
WLAN	Wireless Local Area Network

1 Einleitung

Die Digitalisierung ist in vollem Gange und technologische Entwicklungen wandeln sich rasant, so auch in der Automobilindustrie. Insbesondere die interne Steuerung im Fahrzeug hat sich zukunftsweisend verändert. Statt unzähligen Verkabelungen wird auf ein komplexes System aus untereinander vernetzten Steuergeräten, auch Electronic Control Units (ECUs) genannt, gesetzt. Zur Kommunikation und Steuerung dienen spezielle Bussysteme, darunter das Controller Area Network (CAN).
Diese wachsende Vernetzung offeriert zahlreiche Ansatzpunkte für unberechtigte Zugriffe Dritter auf die Fahrzeugsteuerung. Durch das Senden kompromittierter Nachrichten auf den CAN-Bus können einzelne Systeme bis hin zum gesamten Automobil gezielt attackiert werden und somit die Sicherheit im Fahrzeug gefährden.

1.1 Motivation

Das Controller Area Network als weit verbreitetstes Bussystem im Automobilbereich bietet keine Absicherung vor modifizierten Botschaften, welche auf den Bus gesendet werden. Zwar lassen sich die zur Laufzeit übertragenen Nachrichten mit speziellen Anwendungen wie der CAN-Analysesoftware BUSMASTER auslesen, jedoch ist eine Rekonstruktion der Vorgänge nicht möglich.
Die Angriffe auf das Automobil sind eine bedeutende Thematik, denn mit der Komplexität der Fahrzeugfunktionen steigen auch die Anforderungen an die Sicherheit. Folglich ergibt sich ein hoher Bedarf an Fachkräften und Forschung in diesem Bereich. Zur forensischen Analyse der Datenkommunikation wurde 2018 im Application Center Microcontroller ein CAN-Demonstrator [HAN18] entwickelt, welcher die grundlegenden Fahrfunktionen eines Automobils simuliert. Gesteuert werden kann der Demonstrator durch das Senden von CAN-Botschaften, die mit spezieller Software erzeugt und ausgewertet werden können.
Derzeit verfügt der Demonstrator über zwei Mikrocontroller, welche den CAN-Knoten und das Gateway darstellen. Das AVR-CAN-Board dient zur Steuerung der Motoren sowie der Beleuchtungselemente, das ESP32-EVB realisiert die drahtlose Komponente und beherbergt Funktionen zur Akkustandsüberwachung, Distanzmessung und Hupe.
Im Rahmen des Betriebs und der Weiterentwicklung traten jedoch verschiedene Probleme sowohl auf Seiten der Hardware als auch softwareseitig auf, welche eine Überarbeitung erfordern.

1.2 Zielstellung

Das Ziel dieser Arbeit ist die Entwicklung des Antriebsstranges als eigenständige CAN-Bus-Steuereinheit in einem forensischen Demonstrator. Die Basis bildet dabei die Analyse des, 2018 im Application Center Microcontroller entwickelten, CAN-Demonstrators hinsichtlich seiner Komponenten, der entsprechenden Funktionsweisen und einhergehenden Probleme. Das Redesign des Demonstrators soll über eine analysierbare CAN-Kommunikation verfügen. Durch mehrere Steuergeräte, die über das Controller Area Network miteinander verbunden sind, soll ein realer Aufbau simuliert werden.

2 Grundlagen

Dieses Kapitel setzt sich mit den theoretischen Grundlagen aus dem Bereich der Automobilindustrie, insbesondere der Kommunikation innerhalb des Controller Area Networks auseinander, welche die Basis dieser Arbeit darstellen. Des Weiteren werden die gegebenen Hardware- und Softwarekomponenten kurz vorgestellt.

2.1 OSI-Modell

Das ISO/OSI-Schichtenmodell, kurz OSI-Modell, ist ein Referenzmodell für Netzwerkprotokolle, welches die Kommunikation zwischen Systemen beschreibt und definiert. Das Modell besitzt sieben Schichten (Layer), welche unterschiedliche, klar voneinander abgegrenzte Aufgaben besitzen. Jede Schicht stellt dabei der darüber liegenden Schicht Dienste für die Kommunikation bereit. Die Kommunikation zwischen den Schichten erfolgt hierarchisch, mit dem darunter- bzw. darüber liegenden Layer. Der Informationsaustausch zwischen den Schichten wird über Schnittstellen definiert.
Es müssen nicht alle Schichten in einem System implementiert werden.

Im Folgenden wird jede der sieben Schichten des OSI-Modells mit ihren Aufgaben kurz erläutert. In Abbildung 2.1 sind diese Schichten zwischen dem Übertragungsmedium und der Anwendung grafisch dargestellt.

Layer 7 – Anwendungsschicht (Application Layer):
Enthält Protokolle für Anwendungsfunktionalität

Layer 6 – Darstellungsschicht (Presentation Layer):
Wandlung der Daten in unabhängige Formate.

Layer 5 – Sitzungsschicht (Session Layer):
Steuerung und Kontrolle der Sitzung zwischen zwei Kommunikationsprozessen

Layer 4 – Transportschicht (Transport Layer):
Sorgt für Ende-zu-Ende-Beziehung zwischen zwei Kommunikationsprozessen und stellt Transportdienst für höhere Anwendungsschichten bereit

Layer 3 – Vermittlungsschicht (Network Layer):
Ermöglicht Verbindungen zwischen zwei Knoten; zuständig für Wegewahl und Routing

Layer 2 – Sicherungsschicht (Data Link Layer):
Segmentierung der Daten und Sicherung mit Prüfsummen

Layer 1 – Bitübertragungsschicht (Physical Layer):
Stellt die physikalische Verbindung bereit; elektrische und mechanische Parameter werden hier festgelegt; Anpassung der Bits auf das Übertragungsmedium

[MBW10, S. 2ff]

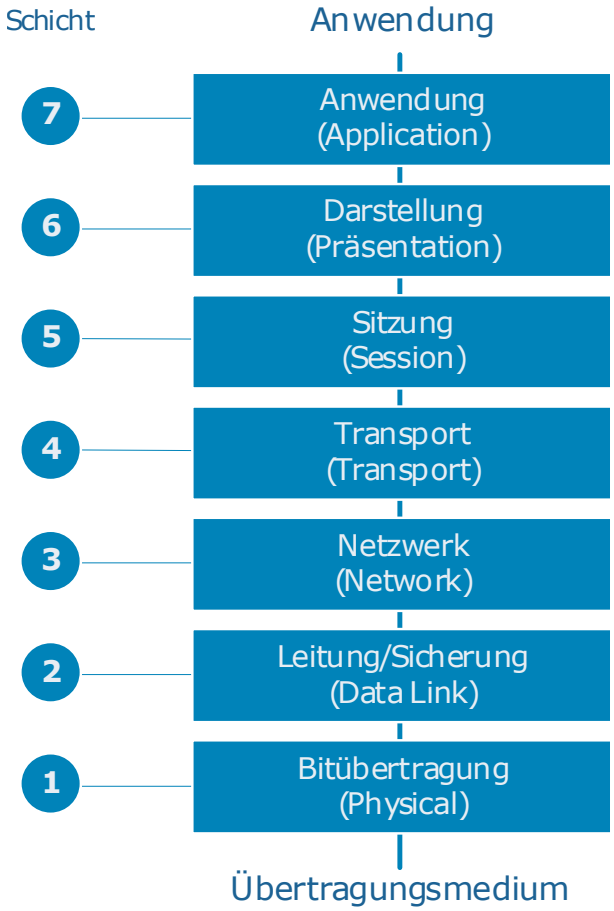


Abbildung 2.1: Aufbau des OSI-Schichtenmodells. Quelle: [THO21, S. 15]

2.2 Netzwerktopologien

Die Topologie eines Netzwerkes beschreibt die Verbindungsstruktur zwischen den einzelnen Steuergeräten eines Netzwerkes, wobei der Fokus auf einem möglichst effizienten Austausch der Nachrichten liegt. Für ein solches Datennetz gibt es verschiedene Formen des Aufbaus, sogenannte Topologien. Innerhalb jeder Topologie existieren mehrere Teilnehmer, sogenannte Slaves (in folgenden Abbildungen grau dargestellt) und teilweise ein Master-Steuergerät (in folgenden Abbildungen rot dargestellt).

2.2.1 Linientopologie

Das Hauptmerkmal der Linientopologie ist die elektrisch passive Ankopplung aller Teilnehmer an eine gemeinsame Leitung (siehe Abbildung 2.2). Die von einer ECU gesandten Nachrichten erhalten dabei alle Teilnehmer. Die Linientopologie ist die mit Abstand am häufigsten verwendete Topologie.

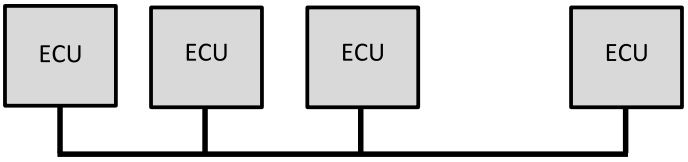


Abbildung 2.2: Schema eines Netzwerkes mit Linientopologie. Quelle: Modifiziert nach [ZIS14, S. 17]

Vorteile

Der Verkabelungsaufwand wird hierbei gering gehalten und neue Teilnehmer können ohne Unterbrechung des Betriebs hinzugefügt werden. Bei Ausfall oder gezieltem Abschalten eines Teilnehmers hat dies keine Auswirkungen auf die restlichen Steuergeräte im Netzwerk.

Nachteile

Die Buslänge sowie die Anzahl der Teilnehmer sind in dieser Topologie begrenzt. Ab einer gewissen Länge bzw. Anzahl ist daher eine Signalverstärkung vonnöten. Um Kollisionen zu vermeiden, ist eine Buszugriffsregelung erforderlich. Zudem müssen die Enden der Leitung durch einen Wellenwiderstand terminiert werden.

2.2.2 Ringtopologie

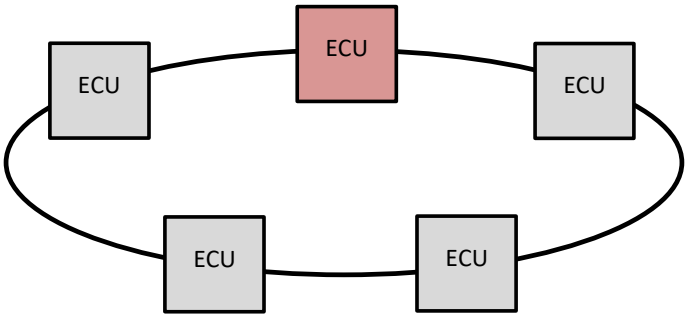


Abbildung 2.3: Schema eines Netzwerkes mit Ringtopologie. Quelle: [ZIS14, S. 18]

Die Ringtopologie (siehe Abbildung 2.3) zeichnet sich durch eine geschlossene Kette von Punkt-zu-Punkt-Verbindungen, einem sogenannten Teilstreckennetz, aus. Eine ausgesandte Nachricht durchläuft dabei so lange den Ring, bis sie den eigentlichen Empfänger erreicht hat.

Vorteile

Von Vorteil bei dieser Topologie ist, dass jeder Teilnehmer eine Signalregeneration bewirkt und Netzwerke daher sehr groß sein dürfen. Jeder Teilnehmer kann dabei über seine geografische Position identifiziert werden. Auf Grund der Punkt-zu-Punkt-Übertragung ist diese Topologie zudem für den Einsatz von Lichtwellenleitern geeignet.

Nachteile

Problematisch ist, dass bei einem Ausfall eines einzelnen Teilnehmers das gesamte System ausfällt. Es bestehen zwar Möglichkeiten zur Absicherung durch Überbrückung eines ausgefallenen Teilnehmers oder über einen redundanten Ring, diese erhöhen jedoch den Verkabelungsaufwand. Soll ein neuer Teilnehmer eingebunden werden, muss hierzu der Bus unterbrochen werden.

2.2.3 Sterntopologie

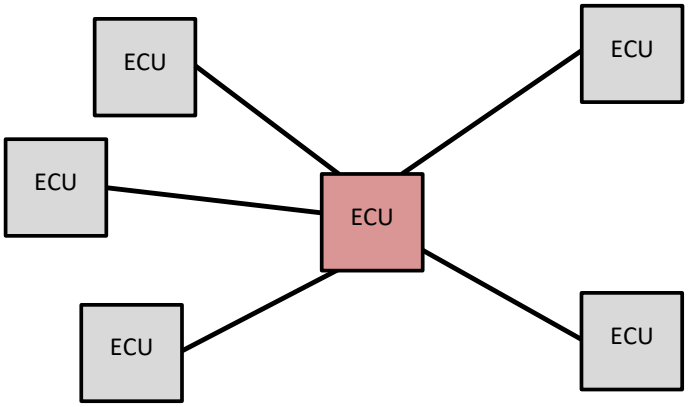


Abbildung 2.4: Schema eines Netzwerkes mit Sterntopologie. Quelle: [ZIS14, S. 18]

Abbildung 2.4 zeigt das Netzwerk einer Sterntopologie, welche alle Steuergeräte mittels einer Punkt-zu-Punkt-Verbindung mit einem zentralen Teilnehmer verbindet.

Vorteile

Im Netzwerk selbst hat jeder Teilnehmer seine eigene Verbindung, weswegen sich eine Einbindung weiterer ECUs einfach realisieren lässt. Die Topologie basiert auf einem simplen Protokoll, welches keine Zugriffsrechte erfordert.

Nachteile

Grundsätzlich ergibt sich eine große Gesamtlänge aller Verbindungen, da ein zentraler Teilnehmer so viele Schnittstellen benötigt wie sich Steuergeräte im Netzwerk befinden. Eine Kommunikation zwischen den Teilnehmern ist dabei nur über das zentrale Steuergerät realisierbar. Kommt es zum Ausfall des zentralen Teilnehmers, ist im System keine Kommunikation möglich.

[ETS02, Kap. 1.6]

2.3 Datenübertragung

Zur Datenübertragung in einem Netzwerk gibt es drei mögliche Verbindungsformen: Die Punkt-zu-Punkt-Verbindung bei der ein Steuergerät mit einem anderen kommuniziert, die Punkt-zu-Gruppe (Multicast), bei der eine ECU Daten an mehrere Teilnehmer im Netzwerk sendet und die Punkt-zu-allen (Broadcast), bei der ein Steuergerät an alle Teilnehmer sendet. Des Weiteren gibt es drei Arten der Datenübertragung. Der uni-direktionale Nachrichtenaustausch erfolgt in eine Richtung über eine Datenverbindung (Simplex), bidirektional im Wechselverkehr mit gemeinsamer Datenverbindung (Halb-Duplex) oder bidirektional mit getrenntem Datenweg für jede Richtung (Voll-Duplex). Die Verbindungsausprägung kann dabei seriell, parallel oder drahtlos vorliegen. Werden die Zeichen nacheinander übertragen, ist von einer seriellen Übertragung die Rede. Bei einer parallelen Übertragung werden die Zeichen gleichzeitig auf verschiedenen Leitungen übermittelt. Die drahtlose Übertragungsart ist grundsätzlich seriell, da es sich um eine bit-serielle Übertragung des Datenstroms handelt. Eine schematische Abbildung der seriellen und parallelen Datenübertragung ist Abbildung 2.5 zu entnehmen.

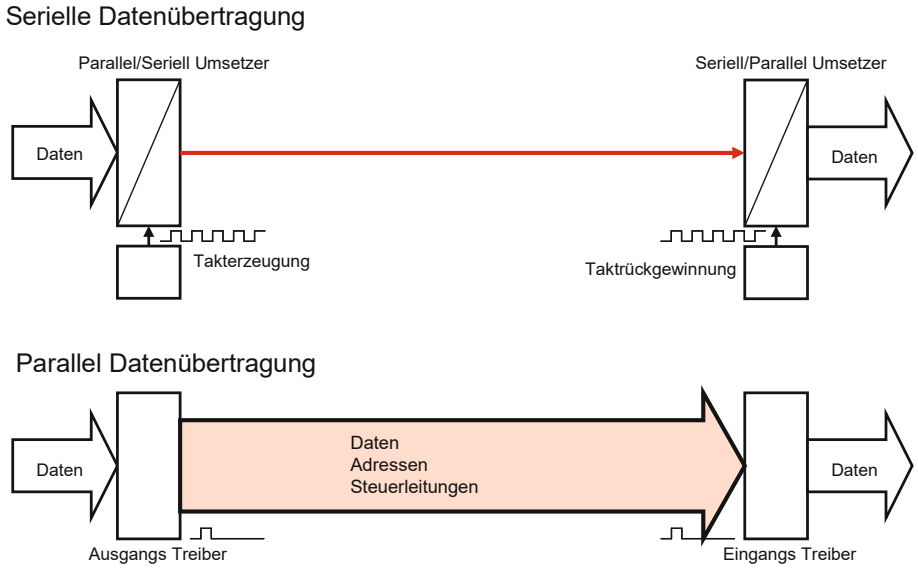


Abbildung 2.5: Serielle und parallele Datenübertragung. Quelle: [HBG14, S. 685]

Bei einer bit-seriellen Übertragung müssen Sender und Empfänger synchronisiert werden. Der Takt muss dabei aus den Synchronisierungsimpulsen vor den eigentlichen Daten aus dem Datenstrom abgeleitet werden. Nach erfolgreicher Synchronisation werden die Daten als Bitstrom bis zur Endsequenz übertragen. Zur Sicherung vor fehlerhafter Übertragung können Parity-Bits oder redundante Checksummen übertragen werden. Die parallele Datenübertragung findet insbesondere beim Austausch von sehr großen Datenmengen und hohen Geschwindigkeiten Anwendung. Der wesentliche Unterschied gegenüber der bit-seriellen Übertragung ist die Verfügung über den kompletten Datensatz, wodurch eine erheblich schnellere Abarbeitung ermöglicht werden kann. Zur Datenübertragung sind dabei die Datengruppe, Adressgruppe und Steuer-

gruppe notwendig. Zum sicheren und synchronen Einlesen der Daten ist auch hier ein Taktsignal für die Synchronisation vonnöten. Da eine parallele Übertragung bei steigender Bit-Breite auf Grund der entsprechend hohen Kosten jedoch nicht mehr sinnvoll umzusetzen ist, werden Teile der Informationen gemultiplext. Dabei werden Daten und Adressen auf denselben Leitungen übertragen und an der Empfangsseite wieder getrennt.

[HBG14, Kap. 15.1]

2.4 Bussysteme in Automobilen

Um die Vernetzung der Steuergeräte innerhalb eines Automobils zu realisieren, wurden Bussysteme zur Kommunikation entwickelt, welche die bisherige Punkt-zu-Punkt-Verbindung ersetzen sollen. Unter einem Bus wird dabei eine Leitung zwischen mehreren Steuerungskomponenten zum Datenaustausch verstanden, wobei jede ECU einen Knotenpunkt darstellt. Aufgrund der enormen Datenmenge entstand ein Netzwerk aus mehreren Systemen mit unterschiedlichen Übertragungseigenschaften. Die bekanntesten Bussysteme stellen dabei folgende dar:

- Controller Area Network (CAN)
- Local Interconnect Network (LIN)
- FlexRay
- Media Oriented Systems Transport (MOST)
- Automotive Ethernet

2.4.1 Controller Area Network

Das in den 1980er Jahren durch Bosch entwickelte Controller Area Network ist eines der am weitesten verbreiteten Bussysteme im Automobilbereich und nach der ISO 11898 international standardisiert. Mit einer Bitübertragungsrate von bis zu 1 MBit/s ermöglicht es die Datenübertragung zwischen mehreren Steuergeräten, wobei diese nach einer Linienstruktur angeordnet und gleichermaßen für das Senden und Empfangen von Nachrichten berechtigt sind. Die Anzahl der Steuergeräte ist dabei nicht durch das Protokoll begrenzt, sondern in Abhängigkeit der Leistungsfähigkeit seiner Treiberbausteine und kann durch den Einsatz von Repeatern auf 64 bis 128 Teilnehmer erweitert werden. Der signifikante Unterschied zu anderen Bussystemen liegt in seiner hohen Fehlertoleranz.

Datenübertragung im CAN

Als Übertragungsmedium kommen für das CAN Datenleitungen aus Kupfer mit verdrillten bzw. unverdrillten Adern und einer Masseleitung zum Einsatz. An dieser Datenleitung sind die Busteilnehmer, die sogenannten CAN-Knoten angeknüpft (siehe Abbildung 2.6). Ein CAN-Knoten ist dabei aus den folgenden Bestandteilen zusammengesetzt:

- Eine Software, welche Daten zur Verfügung stellt und diese verarbeitet
- Ein Controller, der die Daten der Software in CAN-Nachrichten konvertiert und auf den Bus sendet und umgekehrt die Daten der Botschaften extrahiert und an die Software leitet
- Ein Transceiver, welcher die Signale der Datenleitung aufnimmt und Informationen des CAN-Controllers auf den Bus sendet

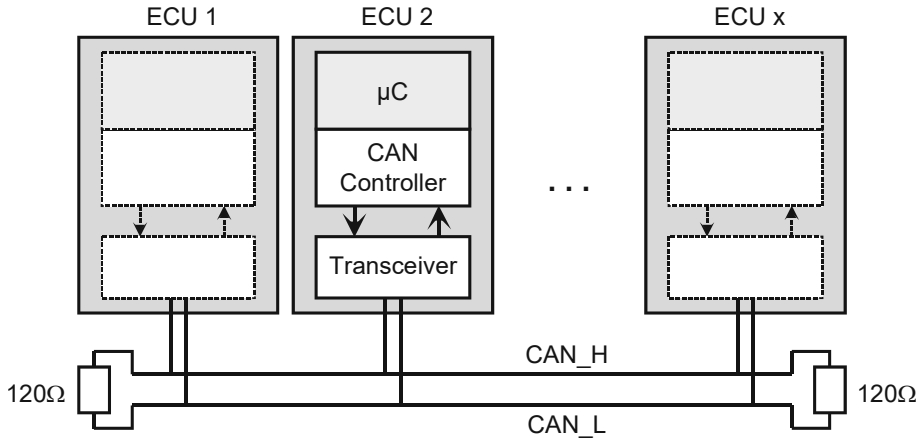


Abbildung 2.6: Aufbau des Controller Area Networks mit Komponenten eines CAN-Knotens. Quelle: [ZIS14, S. 59]

Die Datenübertragung im CAN findet seriell über Halb-Duplex statt und ist broadcast-orientiert. Es können dabei Bitraten bis 125 kBit/s im Low-Speed-Bereich sowie bis zu 1 Mbit/s im High-Speed-CAN erreicht werden. Zur Gewährleistung der Sicherheit und Verfügbarkeit wird auf eine Multimaster-Hierarchie gesetzt. Die Buszugriffssteuerung erfolgt dabei durch die bitweise Arbitrierung. Jede Nachricht im Netzwerk ist durch einen Identifier gekennzeichnet, anhand von welchem die einzelnen Teilnehmer im Netzwerk die Relevanz der Botschaft für sie überprüfen können. Durch diesen Identifier kann zudem die Priorität der Nachrichten bestimmt und ein bevorzugter Buszugriff erteilt werden. Dabei gilt: je niedriger die ID, desto höher die Wichtigkeit. Diese Eigenschaft garantiert in Störfällen die schnellstmögliche Übertragung wichtiger Informationen. Zudem können Kollisionen, welche durch einen Mehrfachzugriff unterschiedlicher Knoten entstehen können, durch die bitweise Arbitrierung der Informationen aufgelöst werden. Dafür vergleicht jeder Teilnehmer den Wert des Kommunikationsmediums mit

dem seinen. Ist sein Wert größer, beendet er seinen Sendevorgang und bereitet sich auf den Empfang einer Botschaft vor. Hierfür wird eine dominante und rezessive Bitcodierung angewandt, wobei die logische 0 dominant, die 1 rezessiv ist. Ein globales Quittierungsfeld gibt dem Sender Auskunft über den Erhalt der Botschaft durch eine oder keine ECU. Es kann hierbei keine Information über die Anzahl der Steuergeräte, welche die Botschaft richtig empfangen haben, entnommen werden, sondern nur, ob überhaupt ein Teilnehmer diese Nachricht erhalten hat. Der Sender kann somit selbst prüfen, ob er noch eine aktive Verbindung zum Bus hat. Um alle Steuergeräte im Netzwerk innerhalb einer Bitzeit synchronisieren zu können, ist die Buslänge begrenzt. Die Berechnung dieser erfolgt anhand folgender Formel:

$$\text{Buslänge} \leq 40...50\text{m} \cdot \frac{1\ \text{MBit/s}}{\text{Bitrate}}$$

Um die Synchronisation der ECUs aufrecht zu erhalten, werden sogenannte Synchronisationsflanken mittels Bitstuffing erzeugt. Hierfür überträgt der Sender nach fünf Bits gleicher Polarität (dominant oder rezessiv) einen komplementären Wert. Auf Empfängerseite wird dieses Stuff-Bit automatisch entfernt, sodass die ursprünglichen Daten wieder vorliegen.

[REI11, S. 92], [ZIS14, Kap. 3.1.2], [LAB99, Kap. 2.2, 4.1], [ETS02, Kap. 1.8.1, 2.2]

High-Speed- und Low-Speed-CAN

Die ISO-Norm unterscheidet im Controller Area Network zwischen High-Speed- und Low-Speed-CAN. Datenübertragungsraten im Bereich von 125 kBit/s bis 1 Mbit/s werden dabei als High-Speed-CAN in der ISO 11898-2 normiert und finden auf Grund der Geschwindigkeit für die Echtzeitanforderungen des Antriebsstranges Verwendung. Bei Datenraten zwischen 5 und 125 kBit/s wird Low-Speed-CAN eingesetzt. Das, in der ISO 11989-3 definierte, Netzwerk findet in Komfort- und Karosseriebereichen, wie beispielsweise der Klimaanlage oder Fensterhebern, Anwendung.
Auf Grund der endlichen Signalausbreitung muss bei steigender Datenrate mit Ausgleichvorgängen in Form von Reflexionen auf dem CAN-Bus gerechnet werden. Um dem entgegenzuwirken, müssen die Bus-Enden in High-Speed-CAN-Netzwerken mit dem Wellenwiderstand des Übertragungsmediums (hier 120 Ω) abgeschlossen werden. [REI11, S.92]

CAN-Botschaft

Innerhalb des Controller Area Networks wird zwischen den folgenden vier Nachrichtentypen unterschieden:

- **Data Frame**
  Datentelegramme werden zur Übertragung von Daten im Netzwerk genutzt.

- **Remote Frame**
  Datenanforderungstelegramme dienen der Anforderung einer bestimmten Nachricht von einem Teilnehmer im Netzwerk und sind durch eine bestimmte Codierung im Control Field (CTR) charakterisiert. In der Regel enthalten Remote Frames keine Daten

- **Error Frame**
  Fehlertelegramme signalisieren von einem Steuergerät erkannte Fehler.

- **Overload Frame**
  Ein Überlasttelegramm führt eine Verzögerung zwischen zwei Data oder Remote Frames herbei.

Abbildung 2.7 veranschaulicht den logischen Aufbau eines Standard-Datentelegramms im Controller Area Network mit den jeweiligen Feldern und zugehöriger Bitanzahl sowie dem zugehörigen Signalpegel (0 oder 1) oberhalb.

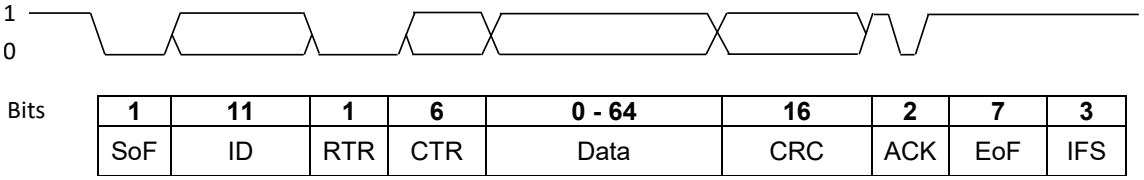


Abbildung 2.7: Aufbau des CAN-Datentelegramms eines Standard-Frames. Quelle: Eigene Darstellung

Der klassische Aufbau einer CAN-Nachricht beginnt immer mit einem Startbit, gefolgt von 11 bzw. 29 Identifier-Bits, je nachdem, ob es sich um ein Standard oder Extended Frame handelt. Die ID zusammen mit dem anschließend folgenden Remote Transmission Request (RTR) ergeben das Arbitrierungsfeld, wobei der RTR angibt, ob es sich um ein gesandtes oder angefordertes Nachrichtentelegramm handelt. Es reihen sich sechs Steuerbits an, welche die Angabe der Datenlänge der Nachricht beinhalten, die Datenbytes selbst, gefolgt von einem 16 Bit langen Cyclic Redundancy Check (CRC). Diese zyklische Redundanzprüfung dient zur Bestimmung eines Prüfwertes für die Daten, um Übertragungsfehler zu erkennen.

Die maximale Datenlänge einer CAN-Nachricht ist auf acht Byte (64 Bit) begrenzt. Es folgen zwei Quittierungsbits bevor die Nachricht mit dem End-of-Frame (7 Bits) abgeschlossen wird. Die angehängten drei Interframe-Space-Bits (IFS) kennzeichnen den minimalen Abstand bis zum nächsten möglichen Botschaftsbeginn. Die maximale Länge einer Botschaft liegt bei 117 Bits für das Standard-Frame und 136 Bits für das Extended-Frame. Aus der Anzahl der Identifier-Bits ergibt sich jeweils eine Anzahl von 2^{11} (2048) und 2^{29} (512 Millionen) verschiedenen Botschaften pro System. Eine Übersicht des Aufbaus einer CAN-Nachricht im Standard-Format ist Tabelle 2.1 zu entnehmen.
Die Begrenzung der Daten auf acht Byte ist erforderlich, um möglichst kurze Latenzzeiten für den Buszugang hoch priorisierter Nachrichten gewährleisten zu können. Eine kurze Nachrichtenlänge ist zudem in stark gestörten Bereichen von Vorteil, da mit zunehmender Blocklänge die Wahrscheinlichkeit eines Fehlers steigt und somit eine Übertragung der Daten unter schwierigen Umständen ermöglicht werden kann.

Tabelle 2.1: Aufbau eines Standard CAN-Frames

Feld	**Länge in Bit**	**Bedeutung**
Start-of-Frame (SoF)	1	Kennzeichnet den Beginn eines Telegramms
Identifier (ID)	11	Kennzeichnung der Nachricht, Prioritätsbestimmung
Remote Transmission Request (RTR)	1	Gibt an, ob es sich um ein Datentelegramm (0) oder ein Datenanforderungstelegramm (1) handelt
Control (CTR) Field	6	2 reservierte Bits + 4 Bits für die Angabe der Datenlänge des nachfolgenden Datenfeldes in Byte
Data Field	0 bis 64	Eigentliche Nutzdaten, entfällt bei Remote Frames
Cyclic Redundancy Check (CRC) Field	16	15 Bit Prüfsequenz + 1 Begrenzungsbit, enthält Fehlercode
Acknowledge (ACK) Field	2	1 Bit Bestätigungsfeld für den Erhalt der Nachricht, 1 Begrenzungsbit
End-of-Frame (EoF)	7	Kennzeichnet das Ende eines Telegramms
Interframe Space (IFS)	3	Zwischenraum zur Trennung der Telegramme

Fehlererkennung und Korrektur

Um die Integrität des Netzwerkes zu gewährleisten, bietet das CAN-Protokoll verschiedene Möglichkeiten, um Fehler bei der Übertragung zu erkennen, entsprechend zu behandeln und einzugrenzen. Durch das Error-Management können vier verschiedene Fehlertypen erkannt werden:

- Bitfehler
- Stuffing-Fehler
- Formfehler
- Acknowledgement-Fehler

Nach der Erkennung eines Fehlers wird jeder Busteilnehmer durch ein Error Frame informiert. Diese CAN-Fehlerbotschaft besteht aus einer einzigartigen Form von sechs aufeinanderfolgenden dominanten Nullen, sieben rahmenende Bits und drei Interframe-Space-Bits. Das Error Frame überschreibt alle anderen Botschaften und wird von allen Teilnehmern erkannt. Um einen Ausfall des gesamten Busses bei anhaltender Störung zu vermeiden, werden im Ausnahmefall Maßnahmen ergriffen, welche das Einwirken des störenden CAN-Knotens einschränken bzw. diesen ganz isolieren sollen.

Ein Teilnehmer kann generell drei Zustände annehmen. Den Normalzustand (Error Active) in dem ein uneingeschränktes Senden und Empfangen von Botschaften gewährleistet ist und in dem bei Eintritt eines Fehlers ein Error Flag gesetzt wird. Wurden mehrere Fehler auf dem Bus identifiziert, tritt der Error Passive Status ein. Es kann zwar weiter kommuniziert werden, im Fehlerfall wird jedoch nur noch ein Error Flag ausgesandt, sodass ein Teilnehmer in diesem Zustand den übrigen Busverkehr nicht weiter behindern kann. Bei der vollständigen Abkopplung vom Bus (Bus Off) ist das Senden und Empfangen von CAN-Nachrichten nicht mehr möglich. Der Knoten kann sich selbst als fehlerhaft erkennen und aus dem Netzwerk zurückziehen, um den Zusammenbruch des Systems zu vermeiden.

[LAB99, Kap. 2.2, 4.1], [ETS02, Kap. 1.8.1, 2.2]

CAN-Schnittstelle

Als CAN-Schnittstelle hat sich der, von der CAN in Automation (CiA) Vereinigung standardisierte, 9-polige Sub-D-Stecker nach Spezifikation CiA/DS 102 durchgesetzt. Diese Steckverbindung bietet die Möglichkeit einzelne Teilnehmer galvanisch vom Bus zu entkoppeln. Tabelle 2.2 beinhaltet die Belegung des 9-poligen D-Sub CAN-Steckers.

Tabelle 2.2: Belegung D-Sub 9-poliger CAN-Stecker. Modifiziert nach: [ENG00, S. 152]

Pin	**Signal**	**Funktion**
1	reserviert	Reserviert
2	CAN_L	CAN-Low, dominant low
3	GND	Ground (GND)
4	reserviert	Reserviert
5	(CAN_SHLD)	Optionales CAN Shield
6	(GND)	Optionales GND
7	CAN_H	CAN-High, dominant high
8	reserviert	Reserviert
9	(CAN_V+)	Optionale externe Stromversorgung

Zur einfachen Anbindung an einen Hostrechner kann ein 9-poliger D-Sub RS232C-Steckverbinder genutzt werden. Für die Übertragung von CAN-Signalen sind mindestens CAN-High, CAN-Low und Ground erforderlich.
Abbildung 2.8 zeigt die Belegung des RS232C-Steckers für CAN.

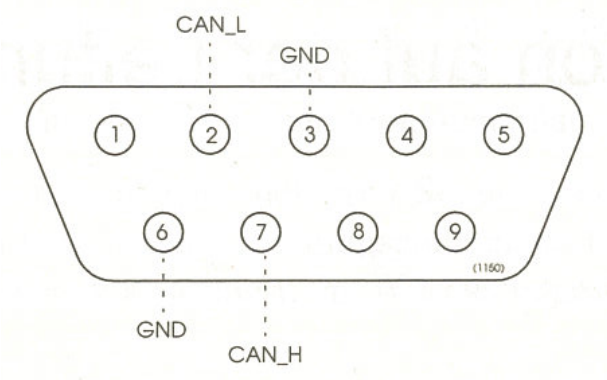


Abbildung 2.8: Skizze eines D-Sub 9-poligen RS232C-Steckverbinders für CAN. Quelle: [ENG00, S. 153]

[ENG00, S. 152f], [CIA94]

2.4.2 Weitere Bussysteme im Kfz

Neben dem Controller Area Network, als verbreitetstes Bussystem im Automobilbereich, haben sich weitere Bussysteme etabliert. Diese werden im Folgenden kurz erläutert.

Local Interconnect Network

Das Local Interconnect Network (LIN) ist ein in den 1990er Jahren vom LIN-Konsortium, einem Zusammenschluss verschiedener Kfz-Hersteller mit Motorola (heute Freescale), entwickeltes Bussystem, welches als kostengünstigere Alternative zum Low-Speed-CAN dienen soll. Mit einer Bitübertragungsrate von 1 bis 20 kBit/s findet es im Bereich der einfachen Elektronik von Tür-, Spiegel- und Sitzeinstellung weit verbreitete Anwendung. Darüber hinaus werden neue Konzepte der intelligenten Ansteuerung hochauflösender LED-Scheinwerfer mittels Kommunikation über den LIN-Bus entworfen. Das Local Interconnect Network ist in der ISO 17987 standardisiert. [ZIS14, Kap. 3.2], [LIN17]

FlexRay

FlexRay ist ein im Jahr 2000 von mehreren deutschen Fahrzeugherstellern und Zulieferern entworfenes Bussystem, welches durch höhere Bit- und Datenraten das CAN in zeitkritischen Bereichen ersetzen soll und in der ISO 17458 standardisiert ist. Das bitstrom-orientierte Übertragungsprotokoll realisiert die Kommunikation zwischen mehreren ECUs zum Austausch von Mess- und Reglersignalen sowie die Leitung elektronischer Steuersignale für beispielsweise Bremse oder Steuerung, insbesondere bei X-by-Wire-Anwendungen, im Echtzeitbetrieb und mit hoher Fehlersicherheit. [ZIS14, Kap. 3.3]

Media Oriented Systems Transport

Im Gegensatz zu den bisherigen Bussystemen, die für Steuer- und Regelaufgaben entwickelt wurden, ist der Media Oriented Systems Transport (MOST) ausschließlich für die Multimedia-Anwendungen im Automobil bestimmt. Der, Ende der Neunziger, von der Firma Oasis (heute Microchip) entwickelte Infotainmentbus realisiert dabei die Vernetzung der Bordelektronik in den Modellen der oberen und mittleren Fahrzeugklassen. Höhere Übertragungsbandbreiten von bis zu 150 MBit/s bilden den wesentlichen Unterschied zu anderen Bussystemen im Kfz-Bereich. [ZIS14, Kap. 3.4]

Automotive Ethernet

Mit zunehmendem Entwicklungsfortschritt können auch moderne Bussysteme neben dem Controller Area Network den Anforderungen nicht mehr gerecht werden. Auf Grund der hohen Kosten und dem ständigen Entwicklungsaufwand der bisher verwendeten Bussysteme hat sich das, bereits in vielen Bereichen angewandte, Ethernet, spezifiziert nach IEEE 802.3, als Alternative etabliert. Ethernet als Bussystem ist kostengünstig und vereinfacht die Integration moderner Technik, wie beispielsweise Smartphones, erheblich. Daten können in diesem Bussystem mit circa 100 MBit/s übermittelt werden. [ZIS14, Kap. 3.5]

2.5 Serial Peripheral Interface

Das Serial Peripheral Interface (SPI) ist ein synchroner, serieller Bus und ermöglicht eine schnelle Kommunikation zwischen verschiedenen Mikrocontrollern oder Peripherien. Es stellt dabei unterschiedliche Datenraten bis 20 Mbit/s zur Verfügung.
Das SPI arbeitet nach dem Master-Slave-Prinzip, wobei der Master die Datenübertragung steuert und die Slaves auswählt. Für die Adressierung wird die Out-of-Channel-Adressierung verwendet. Diese besteht aus zwei getrennten Datenleitungen sowie einer Taktleitung:

- **MOSI**
  Master Out - Slave In (auch Serial Data In (SDI) genannt): Leitung, mit welcher der Master Daten an den Slave sendet.

- **MISO**
  Master In - Slave Out (auch Serial Data Out (SDO) genannt): Leitung, mit welcher der Slave Daten an den Master sendet.

- **SCK**
  Master und Slave müssen im selben Takt lesen und schreiben, diese Eintaktung übernimmt die Serial Clock (SCK).

Durch die Trennung der Sende- und Empfangsleitung wird eine Vollduplex-Schnittstelle erzeugt. Zusätzlich zu diesen drei Leitungen erhalten einige Slaves eine Slave-Select-Leitung (SS), welche den ausgewählten Teilnehmer aktiviert. In besonderen Fällen können sich mehrere Slaves eine SS-Leitung teilen. Ein Master kann mehr als eine SS-Leitung besitzen, wodurch sich mehrere Slaves gleichzeitig ansteuern lassen. Trotzdem wird immer eine Punkt-zu-Punkt-Verbindung zwischen Master und Slave aufgebaut. Dies führt zu einer sternförmigen Topologie. Es werden jedoch auch weitere Topologien von SPI unterstützt:

- **Standalone Topologie**
  Es gibt einen Master und einen Slave.

- **Daisy-Chain-Topologie**
  Es gibt einen Master und mehrere Slaves, die in Reihe geschaltet werden.

- **Parallele Konfiguration**
  Es gibt einen Master und mehrere Slaves, die parallel verbunden sind.

[SPI14]

2.6 CAN-Demonstrator

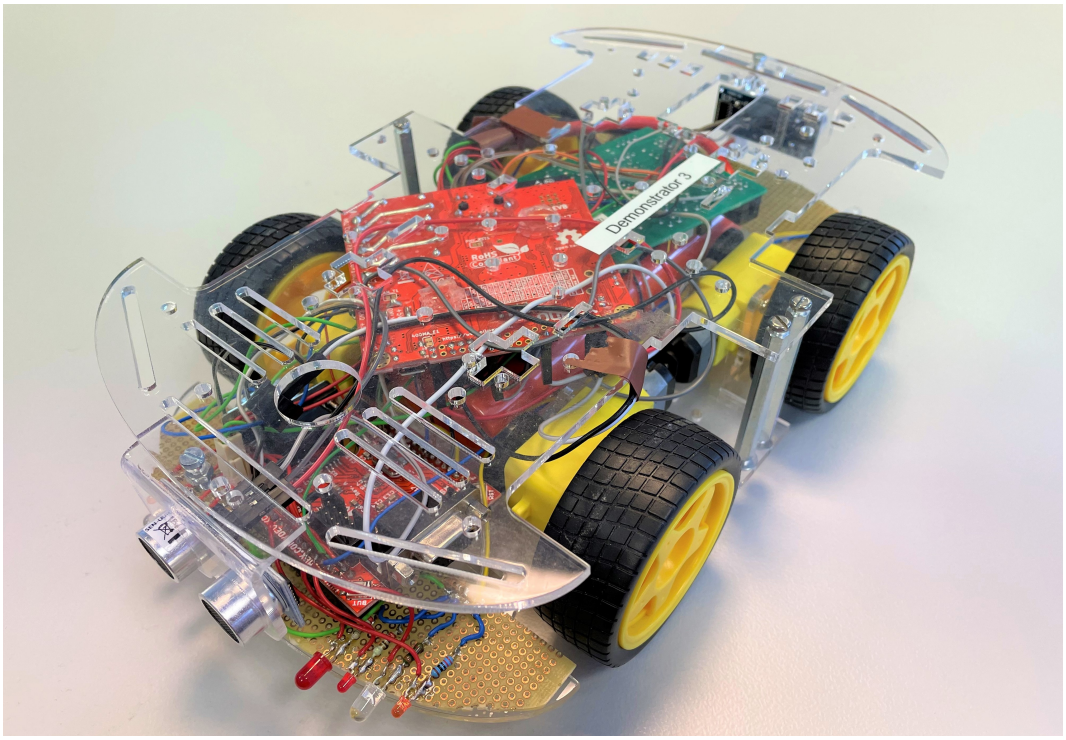


Abbildung 2.9: Version 1 des CAN-Demonstrators. Quelle: Eigene Darstellung

Der CAN-Demonstrator [HAN18] (siehe Abbildung 2.9) ist ein 2018 entwickeltes Modellauto, welches über einen CAN-Bus verfügt und für forensische Zwecke eingesetzt wird. Die Simulation grundlegender Fahrfunktionen eines Automobils, wie beispielsweise der Motorsteuerung oder diverser Beleuchtungen, können über das Senden von CAN-Botschaften gesteuert werden. Der Demonstrator verfügt derzeit über einen, auf dem Evaluationsboard (EVB) verbauten, Mikrocontroller AT90CAN128 der Firma Atmel,

welcher zur Steuerung der Motoren verwendet wird. Das AVR-CAN-Board empfängt dabei als CAN-Knoten die Nachrichten auf dem Bus.
Zur Realisierung der drahtlosen Kommunikation zwischen Demonstrator und Computer dienen zwei Mikrocontroller der Firma Espressif (ESP32), welche als Gateways fungieren und über das Wireless Local Area Network (WLAN) miteinander kommunizieren.

2.7 BUSMASTER

Der BUSMASTER [ETA17] ist eine Open-Source CAN-Analysesoftware der Firma ETAS, welche 2011 entwickelt wurde und zur Analyse des CAN-Busses sowie der Simulation einzelner Steuergeräte bis hin zu einem ganzen System dient. Das Tool unterstützt dabei eine Vielzahl an CAN-Interfaces diverser Hersteller, welche die Verbindung zwischen diesem und dem On-Board-Diagnose-Port am Automobil herstellen. Dadurch lassen sich CAN-Botschaften auf den Bus senden und es wird die Möglichkeit geboten, durch eine Signalüberwachung und das Loggen der über den Bus gesandten Daten, den Nachrichtenverkehr im gesamten Netzwerk vollständig aufzunehmen. Die übertragenen Nachrichten können farblich hinterlegt und in Form von Diagrammen grafisch dargestellt werden. Eine Nachrichtenfilterung kann sowohl auf Anwendungsebene durch einen Softwarefilter, als auch auf CAN-Controller-Ebene mittels eines Hardwarefilters geschehen. Eine Datenbank organisiert dabei die Nachrichten. Abbildung 2.10 zeigt die Benutzeroberfläche der Software.

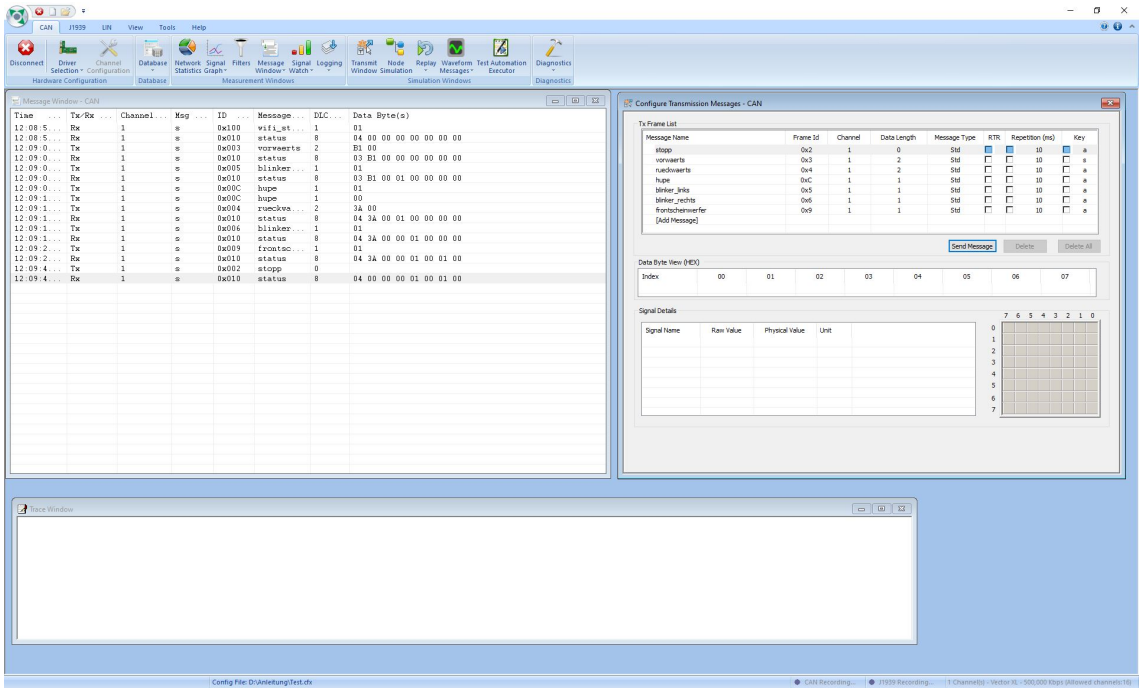


Abbildung 2.10: Benutzeroberfläche der CAN-Analysesoftware BUSMASTER der Firma ETAS. Quelle: Eigene Darstellung

2.8 V-Modell

Das V-Modell ist ein Vorgehensmodell, welches den Softwareentwicklungsprozess in Phasen organisiert. Sein Name steht in Zusammenhang mit seiner V-förmigen Darstellung, welche Abbildung 2.11 zu entnehmen ist. Der linke Teil des Modells definiert hierbei den Softwareentwicklungsprozess, der rechte die entsprechenden Tests. Jeder Entwicklungsphase wird dabei eine Testphase gegenübergestellt.

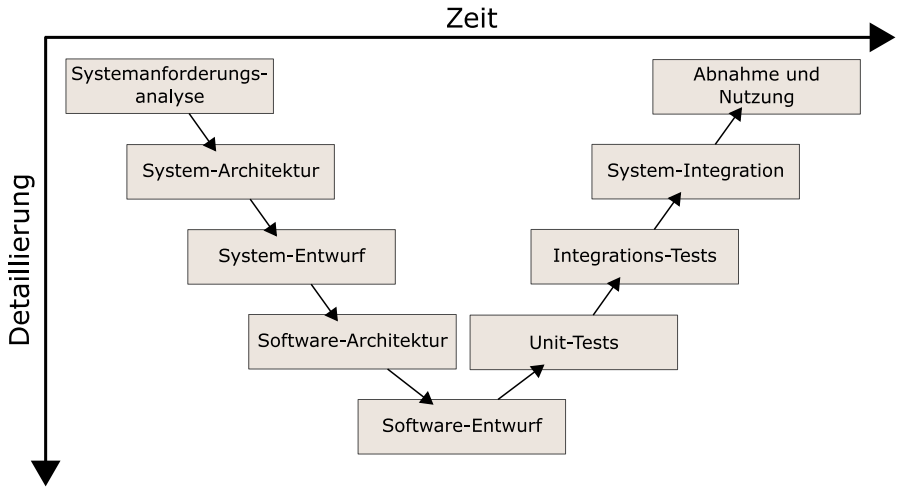


Abbildung 2.11: Stufen des V-Modells für den Softwareentwicklungsprozess. Quelle: [MIC10]

Begonnen wird auf der linken Seite mit einer fachlichen Anforderungsanalyse, welche mit jeder Stufe tiefer und detaillierter bis hin zur Spezifikation und Implementierung führt. Die Implementierung bildet dabei die Spitze des V-Modells, von der ausgehend die Spezifikationen der Entwicklungsseite den einzelnen Tests unterzogen werden. Diese werden im Folgenden kurz erläutert. Das Bindeglied beider Seiten des Modells stellen verschiedene Testfälle dar.

- **Unit-Test**
  Prüft die Programmierung der einzelnen, abgrenzbaren Module mit dem Ziel des Nachweises der technischen Lauffähigkeit.

- **Integrations-Test**
  Die einzelnen Module werden zu Komponenten zusammengefasst und integriert getestet.

- **System-Test**
  Alle Komponenten vereint, werden hier als ganzheitliches System gegen die gesamten Anforderungen getestet.

- **Abnahme-Test**
  Das System wird auf seine ursprünglichen fachlichen Anforderungen überprüft. Dieser Schritt erfolgt im Regelfall durch den Kunden.

[BPK20]

2.9 Analog-Digital-Umsetzer

Sollen analoge Eingangssignale wie beispielsweise Spannung in einen digitalen Datenstrom umgesetzt werden, kommt ein Analog-Digital-Umsetzer (engl. analog-to-digital converter, ADC) zum Einsatz. Die Umwandlung lässt sich dabei in drei Schritte unterteilen.
Der AD-Wandler teilt die Amplitude des analogen Eingangssignals in diskrete Abstände (Sampling) und weist den entstandenen Teilen einen numerischen Wert zu (Quantisierung). Den letzten Schritt bildet das Coding, bei welchem die quantisierten Werte mithilfe eines Encoders kodiert werden.
Die Auflösung des ADCs wird über die Anzahl der Bits angegeben.
[HBG14, Kap. 9.2]

2.10 Pulsweitenmodulation

Die Pulsweitenmodulation, kurz PWM, ist eine Modulationsart, bei der die elektrische Spannung zwischen zwei verschiedenen Spannungspegeln oszilliert. Die Eingangsspannung wird hierbei zyklisch unterbrochen und ein konstanter Rechteckimpuls moduliert. Das Verhältnis von Ein- und Ausschaltzeit kann dabei variieren und wird als Tastverhältnis bezeichnet (Duty-Cycle). Die untenstehende Abbildung 2.12 zeigt die Aufnahme einer PWM eines Gleichstrommotors mittels Oszilloskop.
Anwendung findet das Verfahren der Pulsweitenmodulation häufig zur stufenlosen Regulierung der Leistung von Gleichstromverbrauchern wie Leuchtdioden oder Motoren.
Da das PWM-Signal durch zwei Spannungsebenen (Low- und High-Pegel) erzeugt wird, kann dieses im Vergleich zu einem Generator mit kontinuierlichem (analogen) Signal oder einem Linearregler in einem verlustarmen Schaltbetrieb arbeiten und ist somit effizienter.
[HBG14, S. 789], [PWM20]

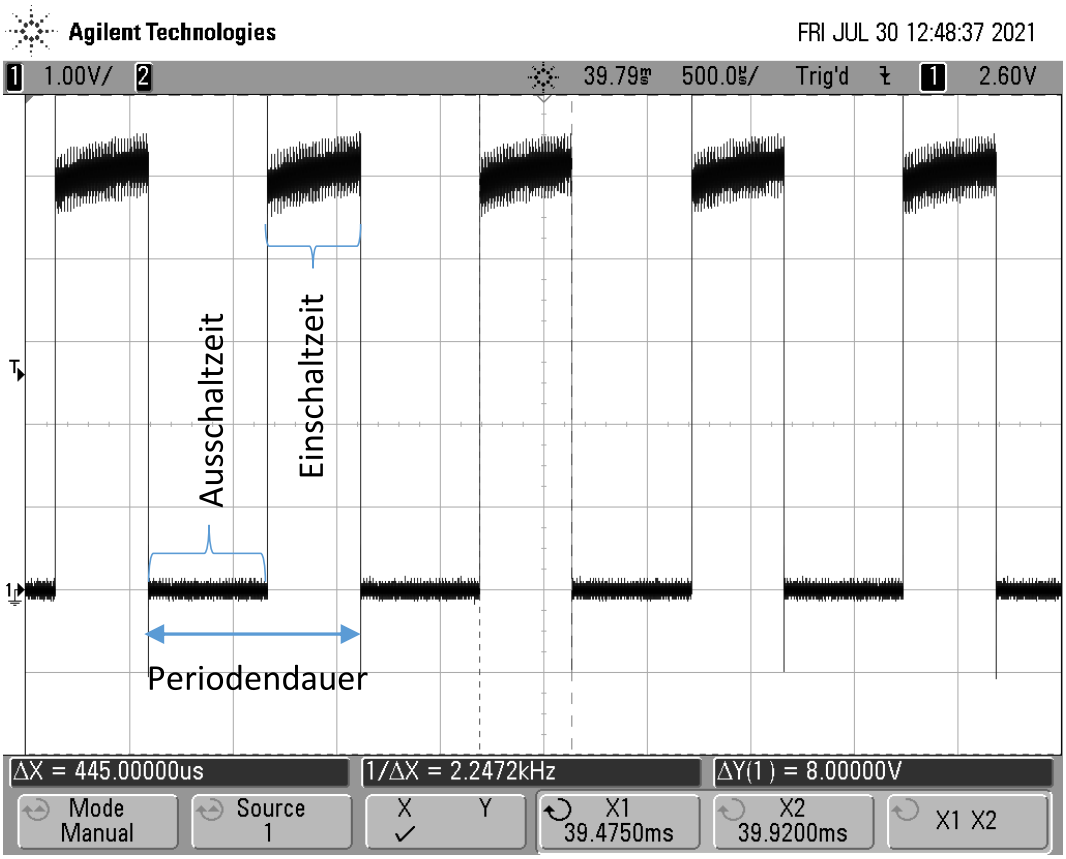


Abbildung 2.12: Aufnahme eines Ausschnittes der Pulsweitenmodulation eines Gleichstrommotors mit Oszilloskop. Quelle: Eigene Darstellung

3 Analyse

Dieses Kapitel beschäftigt sich mit der technischen Analyse der aktuellen Ist-Situation des CAN-Demonstrators sowie der Untersuchung auftretender Probleme.

Der CAN-Demonstrator wurde im Rahmen zweier Bachelorarbeiten [HAN18, LIP19] entwickelt und erweitert, welche die Grundlage dieses Kapitels bilden. Darüber hinaus wurde die Funktionalität des Demonstrators in Eigenarbeit ausgeweitet.

3.1 Hardwarekomponenten

Abbildung 3.1 zeigt die Skizze des Versuchsaufbaus des Demonstrators. Die eingezeichneten Elemente werden im Folgenden näher beleuchtet.

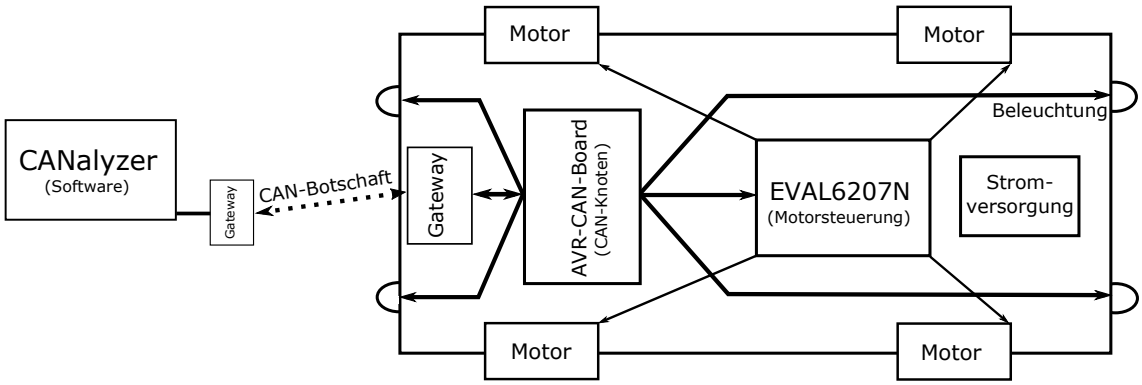


Abbildung 3.1: Skizze des Aufbaus des CAN-Demonstrators Version 1. Quelle: [HAN18, S. 21]

3.1.1 Karosserie

Das Grundgerüst des Demonstrators stellt das Robot Car Kit der Firma Joy-It [ROB20] inklusive der vier Motoren dar.

3.1.2 CAN-Knoten

Das AVR-CAN-Board [AVR10] mit dem Mikrocontroller AT90CAN128 (Atmel) von Olimex übernimmt die Steuerung des Motors und der Beleuchtung innerhalb des Demonstrators. Hierzu werden die eingehenden CAN-Nachrichten ausgewertet und weiterverarbeitet. Über eine physische Verbindung via Breadboardkabel besteht eine direkte Anbindung zu den anderen Komponenten. Das Board agiert zudem als zentraler Stromverteiler aller Bauteile.

3.1.3 Gateway

Zur Realisierung der drahtlosen Kommunikation zwischen dem CAN-Demonstrator und dem Computer dienen zwei ESP32-EVB [RIO21] der Firma Olimex, welche als zentrale Gateways fungieren. Die Übertragung zwischen den beiden Mikrocontrollern erfolgt über WLAN als Transmission Control Protocol (TCP)-Pakete. Das Gateway A im Demonstrator dient dabei als Server der Verbindung und stellt das WLAN als sogenannter Access Point (AP) zur Verfügung. Das Gateway B bildet den Client. Abbildung 3.2 zeigt den schematischen Aufbau dieser Verbindung.

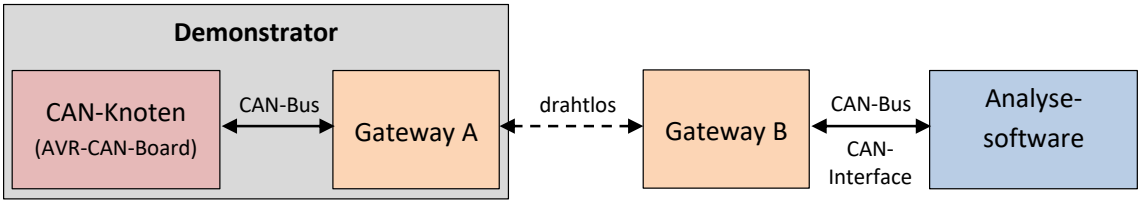


Abbildung 3.2: Schematischer Aufbau der CAN-Verbindung im Demonstrator 1. Quelle: Modifiziert nach [HAN18, S. 22]

Darüber hinaus wurde die Funktionalität im Rahmen der Weiterentwicklung auf zusätzliche Aufgaben wie die Ansteuerung einer Hupe, Distanzmessung und Akkustandsüberwachung erweitert. Diese Funktionen fallen nicht in den eigentlichen Aufgabenbereich eines Gateways, wurden jedoch aus verschiedenen hardware- und softwareseitigen Gründen an dieser Stelle und nicht im Steuergerät implementiert.
Die Stromversorgung zwischen Gateway B und dem Computer wird über ein Micro-USB-Kabel realisiert.

3.1.4 Motorsteuerung

Das Evaluationsboard EVAL6207N [EVA62] von STMicroelectronics erhält die entsprechenden Steuerungsinformationen zur Betreibung der Motoren über eine physische Verbindung durch das AVR-CAN-Board. Zur Steuerung der vier Motoren sind diese an das Board angeschlossen. Insgesamt kann der AT90CAN128 über die folgenden vier verschiedenen Zustände der Motoren bestimmen:

- Motoren sind ausgeschaltet
- Motoren stoppen
- Fahrtrichtung vorwärts
- Fahrtrichtung rückwärts

3.1.5 Beleuchtung

Zur Imitation von Scheinwerfern und Blinkern sind mehrere Leuchtdioden auf einer Euroleiterplatte am Demonstrator befestigt. Im hinteren Teil sind zwei Blinker, zwei Rückleuchten, zwei Rückscheinwerfer und zwei Bremsleuchten verbaut. Im vorderen Teil sind zwei Frontscheinwerfer implementiert. Die eingesetzten LEDs werden mit einer Betriebsspannung von 5 V durch das AVR-CAN-Board versorgt. Die Stromversorgung wird entsprechend über Vorwiderstände reguliert. Tabelle 3.1 zeigt eine Übersicht der verwendeten LEDs mit ihren jeweiligen Daten.

Tabelle 3.1: Übersicht der Beleuchtungselemente des Demonstrators V1 und deren physikalischen Eigenschaften. Quelle: Modifiziert nach [HAN18, S. 27]

Modul	**Farbe**	U_D **in V**	***I* in mA**	**Vorwiderstand**
Frontscheinwerfer	weiß	3,3	13	100 Ω
Bremsleuchten	rot	2,0	12	270 Ω
Rückleuchten	rot	2,2	13	220 Ω
Rückscheinwerfer	weiß	2,9	15	150 Ω
Blinker	gelb	2,1	12	270 Ω

Zur Befestigung der Leuchtdioden und der Vorwiderstände dienen Lochrasterplatten, auf welche diese aufgelötet sind. Die Platten wiederum sind mit Schmelzklebstoff am Robot Car Kit befestigt.

3.1.6 CAN-Bus

Der CAN-Bus überträgt die Informationen zwischen der Electronic Control Unit (AVR-CAN-Board) und dem Gateway A (ESP32-EVB). Die Verbindung von Gateway B zum Computer erfolgt über das CAN-Interface VN1610 von Vector [VEC21].

3.1.7 Stromversorgung

Die Stromversorgung im CAN-Demonstrator wird über einen Lithium-Polymer-Akkumulator mit einer Ausgangsspannung von 7,4 V, bei einer Kapazität von 30 Ah, realisiert [LIP20]. Dieser ist fest verbaut und kann über einen T-Stecker problemlos geladen werden.

Des Weiteren verfügt der CAN-Demonstrator über die folgenden Elemente und Funktionen.

3.1.8 Hupe

Zur Imitation der Hupe ist ein 2-Pin Lautsprecherstecker (SUMMER AL-60P12) [BUZ21] verbaut, welcher über zwei Breadboardkabel mit dem ESP32-EVB verbunden ist. Ein entsprechend implementierter Timer induziert bei Ablauf eine Interrupt Service Routine (ISR), welche den Programmablauf unterbricht und den Schaltzustand des Pins für die Stromzufuhr ändert. Solange gehupt werden soll läuft der Timer in einer Schleife ab und löst jedes Mal die entsprechende ISR aus. Durch den ständigen Zustandswechsel der Stromversorgung am Pin erzeugt der Magnet im Lautsprecher eine Schwingung an der Membran, welche für das menschliche Ohr als Ton wahrnehmbar ist.

3.1.9 Distanzmessung

Ähnlich eines Park-Distance-Control-Systems im Automobil ist der Demonstrator sowohl vorne als auch hinten mit einem Ultraschall-Abstandssensor (HC-SR04) [DIS17] ausgestattet. Durch das Senden eines Impulses und der Dauer der Rückantwort in Mikrosekunden, lässt sich die Distanz zu einem Objekt bestimmen.

3.1.10 Akkustandsüberwachung

Zur Überwachung des Ladezustands des Akkumulators im CAN-Demonstrator wird die anliegende Spannung über einen Analog-Digital-Converter am ESP32-EVB erfasst. Bei Erreichung eines bestimmten Schwellwerts (aktuell 6 V) wird eine Meldung ausgesandt und der Demonstrator beginnt zu hupen. Da die maximale Spannung der Batterie 8,4 V beträgt, der ESP jedoch nur eine Betriebsspannung von 3,3 V hat, wird hierfür ein Spannungsteiler genutzt. Diese Akkustandsüberwachung dient dem Schutz vor Tiefenentladung und einer möglicherweise daraus resultierenden Beschädigung des Stromversorgers.

3.2 Softwarekomponenten

Zur Steuerung des CAN-Demonstrators sind neben den genannten Hardwarekomponenten auch softwareseitige Konfigurationen nötig.

3.2.1 Steuerung des CAN-Demonstrators

Zur Ansteuerung des CAN-Demonstrators müssen CAN-Telegramme über den Bus gesandt werden. Hierfür gibt es diverse Software-Lösungen wie beispielsweise die CAN-Analysesoftware BUSMASTER der Firma ETAS oder den CANalyzer[1] von Vector. Innerhalb dieser können CAN-Botschaften definiert und in einer Datenbank abgelegt werden, um diese manuell einzeln versenden zu können.

3.2.2 CAN-Botschaften

Für die Kommunikation zwischen CAN-Analysesoftware und Demonstrator sowie weiteren CAN-fähigen Steuergeräten, wie beispielsweise dem Car Access System (CAS), werden CAN-Botschaften verwendet. Tabelle 3.2 beinhaltet alle CAN-Botschaften mit ihren Signalen, welche aktuell im Versuchsaufbau vorgesehen sind. In der Programmierung sind teilweise Botschaften angelegt, welche bisher nicht genutzt werden oder aufgrund von einer fehlenden Dokumentation nicht näher hinsichtlich ihrer Signale und Wertigkeit analysiert werden können. Diese sind im Folgenden als „unbekannt“ klassifiziert. Es handelt sich hierbei um die CAS-Nachrichten, Botschaften des Car Access Systems. Dieses Steuergerät wurde experimentell dem Versuchsaufbau hinzugefügt und so weit eingebunden, dass der Demonstrator bei erkannten CAS-Nachrichten nur auf Befehle reagiert hat, wenn der zugehörige Schlüssel eingesteckt war. Erste Versuche wurden unternommen, weshalb nicht näher beschriebene Kommandos in der unten stehenden Liste auftauchen. Da das CAS primär nichts mit dem Demonstrator zu tun hat, wird dieses daher nicht weiter betrachtet.

Tabelle 3.2: Übersicht der CAN-Botschaften des Demonstrators der Version 1. Quelle: Modifiziert nach [HAN18, S. 30]

Ziel	ID	Botschaft	Signale	Wertigkeit
Motor	0x02	Stopp		
	0x03	Fahrtrichtung vorwärts	Geschwindigkeit	0-255
			Fahrtrichtung (links/rechts)	0-2
	0x04	Fahrtrichtung rückwärts	Geschwindigkeit	0-255
			Fahrtrichtung (links/rechts)	0-2

[1] CANalyzer: `https://www.vector.com/de/de/produkte/produkte-a-z/software/canalyzer`

Tabelle 3.2: Übersicht der CAN-Botschaften des Demonstrators der Version 1. (Fortsetzung)

Ziel	**ID**	**Botschaft**	**Signale**	**Wertigkeit**
Beleuchtung	0x05	Blinker links	Status	0-1
	0x06	Blinker rechts	Status	0-1
	0x07	Warnblinker	Status	0-1
	0x08	Bremsleuchten	Status	0-1
	0x08	Frontscheinwerfer	Status	0-1
	0x08	Rückleuchten	Status	0-1
	0x08	Rückscheinwerfer	Status	0-1
Sonstiges	0x01	Car Access System	unbekannt	
	0x03*			
	0x30			
	0xBE			
	0x10	Status	Fahrtrichtung (vorwärts/rückwärts)	3, 4
			Fahrtrichtung (links/rechts)	0-2
			Geschwindigkeit	0-255
			Frontscheinwerfer	0-1
			Bremsleuchten	0-1
			Rückscheinwerfer	0-1
			Blinker links	0-1
			Blinker rechts	0-1
	0x0C	Hupe	Status	0-1
	0x0D	Akkumulator	Abfrage	1-2
	0x0E	Akkustand	Status	0-255
	0x20	Änderung der Baudrate	unbekannt	
	0x100	Wi-Fi-Status	Status	0-1
	0xC8	Watchdog	Zeit zwischen Nachrichten	0-255

* Die doppelte Aufführung der ID 0x03 liegt einem Problem im Quellcode zugrunde und wird im Folgenden beleuchtet.

3.3 Problemstellung

Dieses Kapitel beschäftigt sich mit der Untersuchung des CAN-Demonstrators auf Schwachstellen und auftretende Probleme, sowohl auf Seiten der Software als auch der Hardware.

Im Rahmen der Weiterentwicklung des CAN-Demonstrators sollen weitere technische Komponenten aus dem Automobil simuliert werden. Hierfür werden entsprechende, freie Pins an den Mikrocontrollern benötigt. Um gemäß des CAN-Aufbaus zu arbeiten und das AVR-CAN-Board nicht zu überlasten, scheint sich das Gateway für diese Implementierungen am besten zu eignen. Das ESP32-EVB verfügt physisch zwar über eine Menge freier Pins, zum Großteil sind diese jedoch intern belegt (vgl. Abbildung 3.3 und 3.4). Zur Nutzung wurden bereits erste, für den Einsatz im CAN-Demonstrator, nicht benötigte Bauteile von der Platine entfernt.

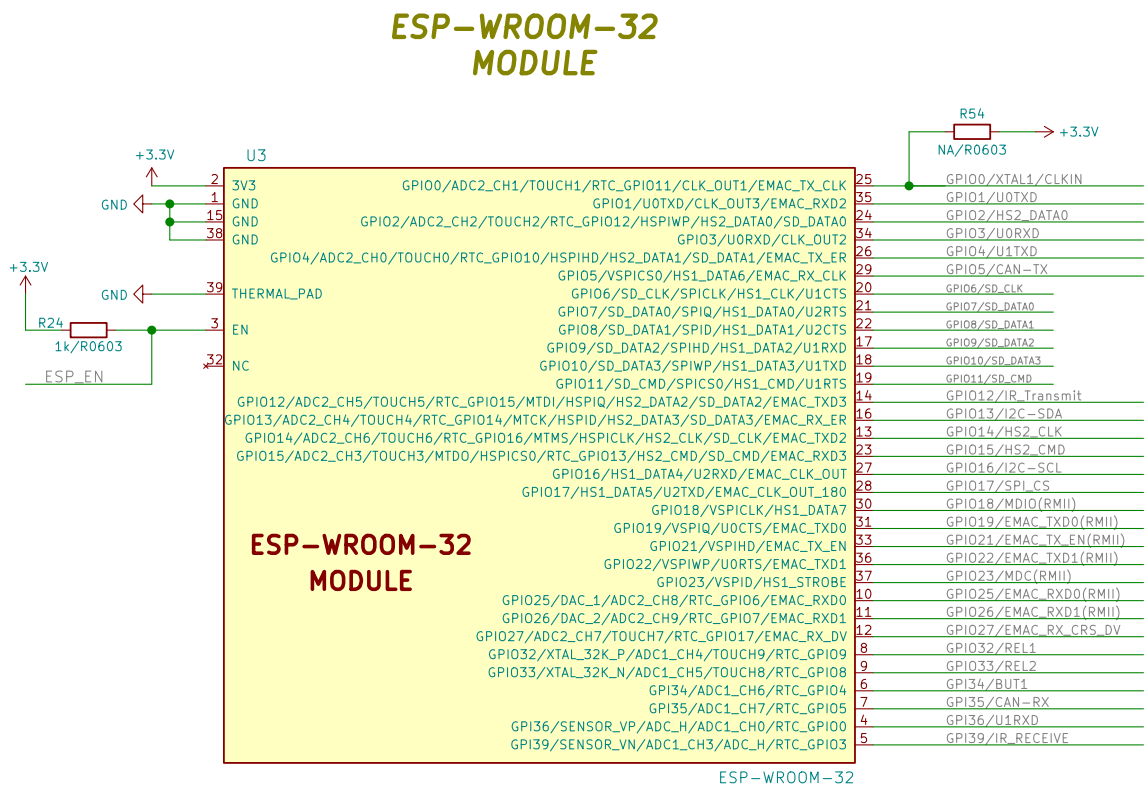


Abbildung 3.3: Pinbelegung des ESP32-EVBs (1/2). Quelle: [ESP18]

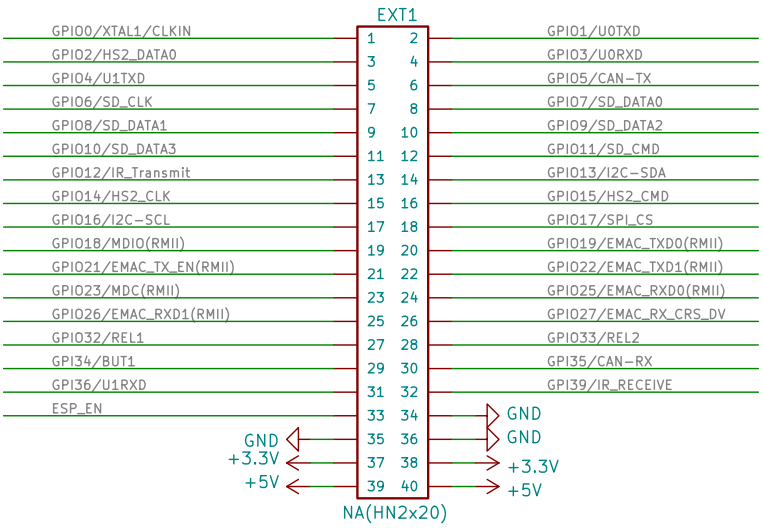


Abbildung 3.4: Pinbelegung des ESP32-EVBs (2/2). Quelle: [ESP18]

Ein großes Problem stellt zudem der stark begrenzte Platz am Demonstrator dar. Gerade die Nutzung der drei Entwicklungsplatinen mit Maßen von 75 x 75 mm beim ESP32-EVB bis 50 x 60 mm beim AVR-CAN-Board sind zu überdenken. Hinzukommen fest verbaute Komponenten wie beispielsweise Relais und LAN-Buchsen, welche durch ihre Höhe enormen Platz einnehmen.
Des Weiteren erfordert die physische Verbindung der einzelnen Komponenten eine Vielzahl an Breadboardkabeln. Diese sind aktuell nur zweckmäßig und ohne große Struktur als Verbindungsleitungen im CAN-Demonstrator verbaut, was eine unübersichtliche Verkabelung zur Folge hat. Zudem können die freiliegenden Kabel gefährlich sein. Auch das Ablösen einiger Kabel und bereits mehrfache Reparaturen sind als Problem zu nennen.

Das Controller Area Network sieht einen Aufbau mit mehreren Electronic Control Units vor, welche gezielt für die Steuerung eines Komplexes verantwortlich sind. Der aktuelle Aufbau des Demonstrators verfügt genau genommen nur über eine ECU, was nicht dem originalen CAN-Schema entspricht. Ein Mikrocontroller steuert dabei als CAN-Knoten die Motor- und Beleuchtungsfunktionen, der zweite ist neben seiner Funktion als Gateway für die drahtlose Kommunikation zusätzlich für die Hupe sowie die Distanzmessung und die Akkustandsüberwachung verantwortlich. Eine klare Funktionstrennung nach Aufgabengebiet des jeweiligen Controllers ist vonnöten.

Bei der Analyse des Quellcodes fällt zudem auf, dass es zu Doppelbelegungen der Identifier bei den CAN-Botschaften kommt. So ist die ID 0x03 beispielsweise für die Fahrtrichtung vorwärts und das Car Access System definiert.

Aktuell werden die beiden verbauten Platinen ESP32-EVB und EVAL6207N über unterschiedliche Programmierschnittstellen geflasht. Die parallele Verwendung von JTAG und Micro-USB-B ist zwar nicht problematisch, jedoch wäre eine Vereinheitlichung als Vereinfachung im Hinblick auf die Anwendung in der Lehre wünschenswert.
Neben dieser hardwareseitigen Zusammenführung wäre es für den Einsatz in der Lehre von Vorteil, nur einen Programmcode für den Demonstrator zu benötigen und mit diesem alle Steuergeräte ansprechen zu können.

4 Methoden

Dieses Kapitel beschreibt den Prozess der Konzeptionierung, der Programmierung und dem anschließenden Zusammenbau der Komponenten für das Redesign des CAN-Demonstrators, mit Schwerpunkt auf dem Antriebsstrang.

4.1 Präzisierung der Aufgabenstellung

Der CAN-Demonstrator soll im Fortlauf innerhalb der Lehre für forensische Zwecke eingesetzt werden. Hierbei sollen die Funktionalitäten gemäß derer im Automobil simuliert werden können. Die Ansteuerung soll dabei durch das Aussenden entsprechender CAN-Botschaften aus einer CAN-Analysesoftware erfolgen.

Zur Realisierung des Projekts werden folgende Hardware- und Softwarekomponenten benötigt:

- Als Grundgerüst für den CAN-Demonstrator wird eine Karosserie mit Motoren sowie Rädern benötigt.
- Zur Ansteuerung der Motoren wird ein entsprechender Motortreiber benötigt.
- Zur getreuen Nachbildung des Controller Area Networks werden mehrere Steuergeräte für die Erfüllung der einzelnen Funktionen benötigt.
- Als Einheit zwischen dem CAN-Demonstrator und der CAN-Analysesoftware soll ein zentrales Gateway fungieren.
- Für die Scheinwerfer und Blinker werden LEDs benötigt.
- Für die Realisierung der Hupe ist ein Lautsprecher vonnöten.
- Zur Distanzmessung muss ein entsprechender Sensor implementiert werden.
- Es wird eine mobile Stromversorgung benötigt.
- Zur Überprüfung ebendieser muss eine Akkustandsüberwachung implementiert werden.
- Die Steuerung des CAN-Demonstrators erfolgt durch die CAN-Analysesoftware BUSMASTER, über welche entsprechende CAN-Botschaften ausgesandt werden.
- Zur Nachrichtenübertragung von PC an Gateway dient das CAN-Interface VN1610.

4.2 Grundkonzeption

Entsprechend der bestimmten Hard- und Softwarekomponenten des vorherigen Kapitels ergeben sich für die einzelnen Komponenten unterschiedliche Aufgaben, welche im Folgenden präzisiert werden.

4.2.1 Karosserie

In der bisherigen Ausführung stellt das Karosserieset Robot Car Kit von Joy-It mit vier Gleichstrommotoren das Grundgerüst dar. Dieses soll für diese Arbeit auf Grund seiner guten Eignung übernommen werden. Das Fahrgestell ist Abbildung 4.1 zu entnehmen.

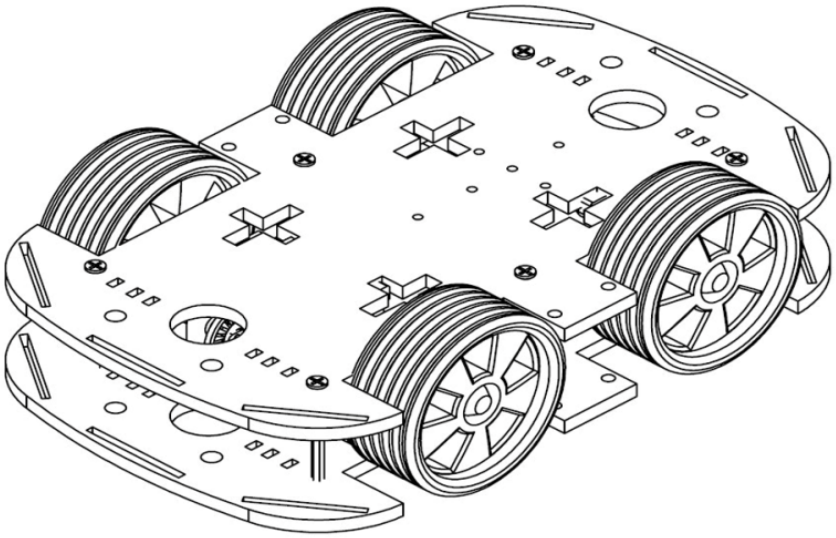


Abbildung 4.1: Robot Car Kit von Joy-It. Quelle: [ROB20]

4.2.2 Gateway

Der Demonstrator soll über ein zentrales Gateway verfügen, welches für die drahtlose Kommunikation zwischen der CAN-Analysesoftware BUSMASTER und dem CAN-Bus auf dem Demonstrator dienen soll. Hierfür muss ein entsprechender Mikrocontroller ausgewählt werden, welcher über eine Funktechnolgie verfügt und zugleich das CAN-Protokoll unterstützt.

4.2.3 Steuergeräte

Zur Steuerung der Motor- und Beleuchtungskomponenten sowie weiterer Funktionen soll der Demonstrator über mehrere Steuergeräte verfügen. Die über den CAN-Bus laufenden Nachrichten werden hier ausgewertet und weiterverarbeitet.

4.2.4 Motorsteuerung

Zur Ansteuerung der Motoren des Robot Car Kits muss ein Motortreiber ausgewählt werden, welcher die erforderlichen technischen Eigenschaften dieser erfüllt.
Diese Komponente soll Informationen für die Ansteuerung der Motoren durch das entsprechende Steuergerät erhalten. Hierfür müssen die Signale des Steuergeräts über eine physische Verbindung empfangen und umgesetzt werden. Die Motoren müssen an dieses Board angeschlossen werden.

4.2.5 Beleuchtung

Um der Beleuchtung eines Automobils möglichst nahe zu kommen, sollen die Scheinwerfer und Blinker durch mehrere Leuchtdioden realisiert werden. Entsprechend muss nach einer Befestigungsmöglichkeit für diese gesucht werden.

4.2.6 Hupe

Für die Realisierung der Hupe soll ein kleiner Lautsprecher dienen. Der bisher implementierte elektromagnetische Buzzer RND 430-00008 soll in dieser Umsetzung beibehalten werden.

4.2.7 Distanzmessung

Für eine beidseitige Distanzmessung am Demonstrator soll wie bisher der Ultraschall-Sensor HC-SR04 implementiert werden.

4.2.8 Stromversorgung

Zur uneingeschränkten, kabelungebundenen Nutzung muss der CAN-Demonstrator mit einer mobilen Stromversorgung ausgestattet werden.

4.2.9 Akkustandsüberwachung

Dieser Bestandteil soll durch ein entsprechendes Steuergerät realisiert werden. Hierfür muss dieses die an der Stromquelle anliegende Spannung kontinuierlich messen. Die bisherige Umsetzung, mit einem Analog-Digital-Wandler, würde sich eignen. Entsprechend wird ein Mikrocontroller mit ADC als Steuergerät benötigt.

4.2.10 BUSMASTER

Das Gateway und die CAN-Analysesoftware BUSMASTER sollen über das CAN-Interface VN1610 miteinander verbunden werden. Innerhalb der Datenbank der Software müssen entsprechende CAN-Botschaften definiert und implementiert werden. Für die intuitive Steuerung des CAN-Demonstrators soll die, für den BUSMASTER entwickelte grafische Benutzeroberfläche (GUI) [LIP19] dienen.

4.2.11 Überblick

Ein erstes Konzept für den Versuchsaufbau dieser Arbeit ist dem Blockschaltbild in Abbildung 4.2 zu entnehmen.

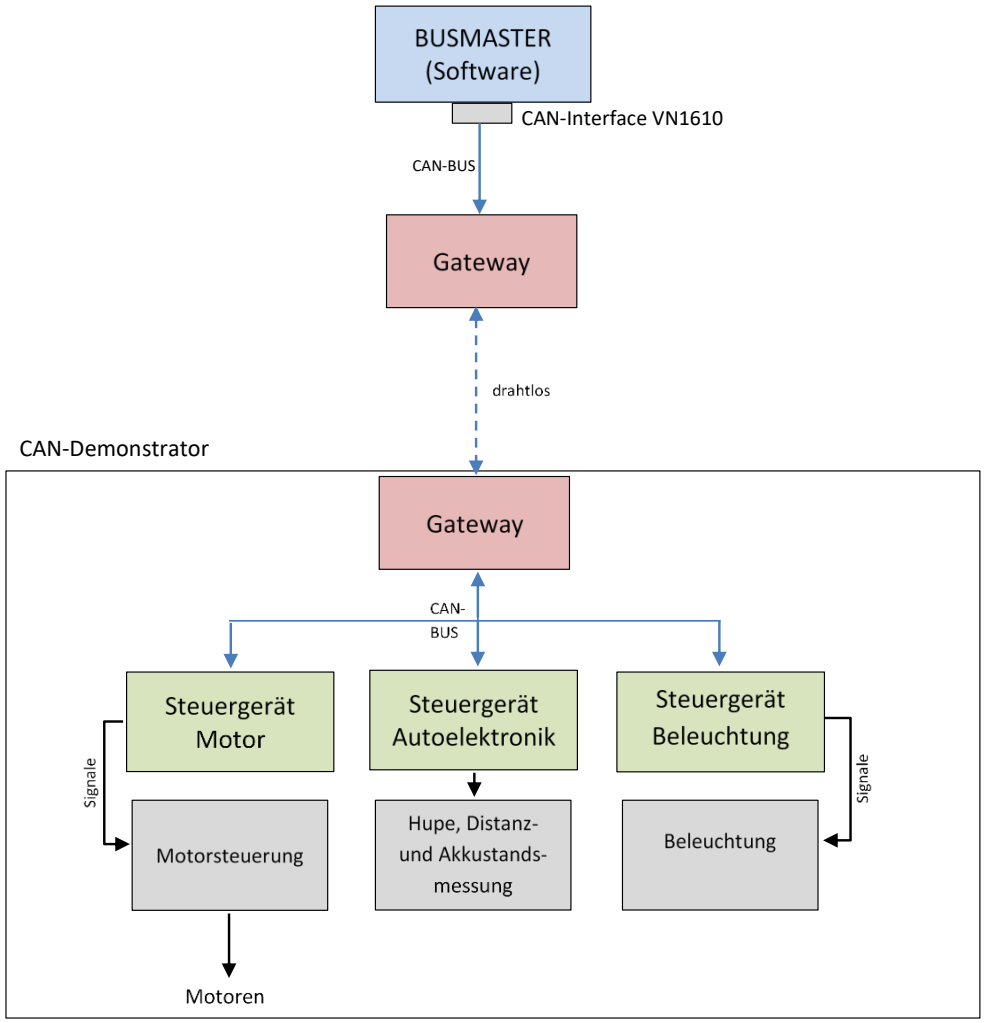


Abbildung 4.2: Blockschaltbild des Versuchsaufbaus des CAN-Demonstrators Version 2. Quelle: Eigene Darstellung

4.3 Hardwareentwurf

Nach der Festlegung der Komponenten des CAN-Demonstrators steht der Entwurf des Hardwaresystems im Vordergrund. Im Folgenden werden, entsprechend der im vorherigen Kapitel genannten, benötigten Komponenten, die Anforderungen an diese betrachtet. Im Anschluss werden mehrere Hardwarelösungen miteinander verglichen.

4.3.1 Steuergerät

Im CAN-Demonstrator sollen insgesamt drei Steuergeräte implementiert werden, welche über den CAN-Bus miteinander kommunizieren. Wie bereits zuvor beschrieben, soll die Aufteilung in ein Motorsteuergerät, ein Gerät zur Steuerung der Beleuchtungselemente und ein weiteres für Autoelektronik wie die Hupe, die Distanzmessung und die Akkustandsüberwachung erfolgen.
Die Implementierung soll die Möglichkeit der beliebigen Erweiterung um zusätzliche Steuergeräte ergeben. Anhand der Aufgaben und der in Kapitel 3.2.3 analysierten Probleme ergeben sich die im Folgenden beschriebenen Anforderungen an das Steuergerät. Von größter Wichtigkeit ist die CAN-Kompatibilität bzw. eine Möglichkeit der Umsetzung dieser. Um die Vereinheitlichung der Software und der Programmierschnittstelle realisieren zu können, sollte der Mikrocontroller Arduino-kompatibel sein. Da der Platz auf dem Robot Car Kit stark begrenzt ist, wird ein möglichst kleiner Controller gesucht, wobei dieser auf Grund der in der Lehre begrenzten Ressourcen preisgünstig sein sollte. Ein bisheriges Problem war die zu geringe Anzahl an freien Pins, welche bei der neuen Auswahl beachtet werden muss. Da eines der Steuergeräte eine Akkustandsüberwachung übernehmen soll, wird ein Analog-Digial-Wandler benötigt. Speziell für die Motor- und Beleuchtungsfunktionen ist ein pulsweitenmodulationsfähiger Pin vonnöten. Nachfolgend wird das aktuell verbaute AVR-CAN-Board mit zwei weiteren Alternativen verglichen.

AVR-CAN

Olimex Ltd. bietet mit dem AVR-CAN (siehe Abbildung 4.3) ein kompaktes Board für Entwickler an, auf welchem der Mikrocontroller AT90CAN128 der Firma Atmel verbaut ist. Die Platine ist unter anderem mit einem CAN-Bus-Treiber und zwei Sub-D Steckverbindungen, passend für das CAN-Interface VN1610, ausgestattet. Zur Programmierung des Flash-Speichers und als Debugging Interface dient eine JTAG-Schnittstelle, wobei hierfür ein extra Programmierer benötigt wird. Mit 52 Pins wird er zwar den gestellten Anforderungen gerecht, ist jedoch mit Abmessungen von 50 x 60 mm recht groß, wobei die Sub-D-Elemente seitlich darüber hinausragen. Ein Analog-Digial-Converter, welcher für eine Akkustandüberwachung benötigt wird, ist implementiert,

ebenso wie vier Timer mit PWM. Derzeit liegt der Preis für das AVR-CAN bei 16,95 €[2]. [AVR10]

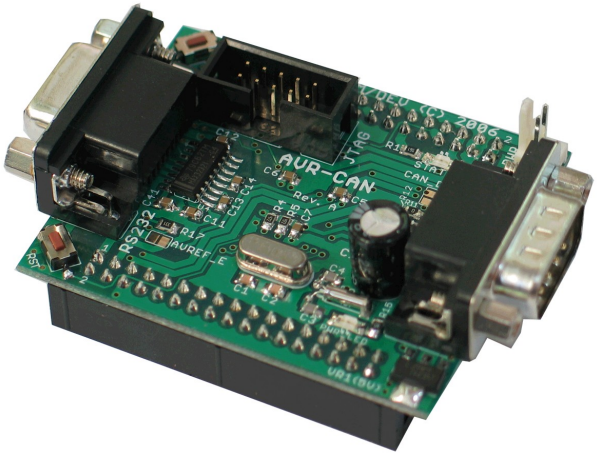


Abbildung 4.3: AVR-CAN-Board der Firma Olimex. Quelle: [AVR10]

Arduino Nano

Der Arduino Nano (siehe Abbildung 4.4) ist ein kleines, steckbrettfreundliches Board, welches auf dem 8-bit-ATMega328P basiert. Als Kommunikationsschnittstellen verfügt er über Universal Asynchronous Receiver Transmitter (UART), ein Serial Peripheral Interface (SPI) und I^2C. Insgesamt stehen 30 Pins, davon acht analoge und 22 digitale Pins, zur Verfügung, welche individuell programmiert werden können. Des Weiteren können sechs digitale I/O-Pins als PWM verwendet werden. Der ATmega328 besitzt analoge Eingänge, welche zur Spannungsmessung verwendet werden können. Für die Verbindung mit dem Computer des 45 x 18 mm kleinen Boards wird ein Mini-USB Anschluss verwendet. Der Arduino Nano ist nicht CAN-fähig, sondern benötigt zur Verwendung des CAN-Protokolls einen entsprechenden CAN-Controller sowie -Transceiver. Hierfür eignet sich das unten genannte CAN-Modul der Firma Joy-It, welches beide benötigten Komponenten vereint. Der Preis für den originalen Arduino Nano liegt derzeit bei 20 €[3]. Joy-It bietet eine etwas günstigere Alternative mit dem Arduino Nano-V3, welcher ein Arduino-kompatibler Mikrocontroller mit originalem Chip ist und für 11,99 € erworben werden kann[4]. [ARD20]

[2] Preis bei `https://www.olimex.com/Products/AVR/Development/AVR-CAN/` [Eingesehen am 27.04.2021, 14:49 Uhr]

[3] Preis bei `https://store.arduino.cc/arduino-nano` [Eingesehen am 29.06.2021, 17:29 Uhr]

[4] Preis bei `https://www.conrad.de/de/p/joy%2Dit%2Dboard%2Darduino%2Dnano%2Dv3%2Datmega328%2Dpassend%2Dfuer%2Darduino%2Dboards%2Darduino%2D1678142.html` [28.04.2021, 14:55 Uhr]

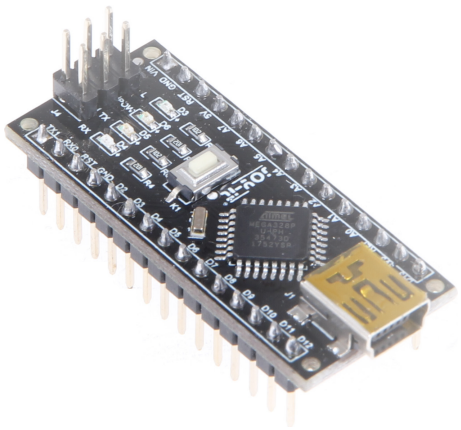


Abbildung 4.4: Arduino Nano V3 der Firma Joy-It. Quelle: [ARD20]

CAN-Modul

Das CAN-Modul der Firma Joy-It verfügt über einen MCP2525 CAN-Controller und einen MCP2562 CAN-Transceiver und stellt somit eine Komplettlösung zur Realisierung der CAN-Fähigkeit dar. Der MCP2515 (siehe Abbildung 4.5, rote Markierung) erweitert den Mikrocontroller um die CAN-Funktionalität, indem er gesendete Nutzdaten CAN-spezifisch aufbereitet und diesen somit gleichzeitig durch die Übernahme der zeitkritischen Aufgaben entlastet. Der MCP2562 (siehe Abbildung 4.5, gelbe Markierung) fungiert als Treiber. Er ist als Schnittstelle zwischen dem CAN-Controller und dem physischen Zweidraht-CAN-Bus zu betrachten. Das CAN-Modul ist Arduino-kompatibel und die Verbindung zum Mikrocontroller ist über die SPI-Schnittstelle möglich. Preislich liegt das Modul derzeit bei 9,59 €[5]. [CAN18]

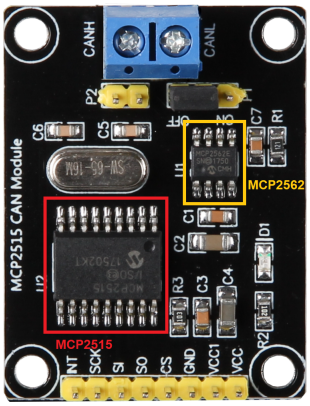


Abbildung 4.5: CAN-Modul der Firma Joy-It mit integriertem CAN-Controller MCP2515 und CAN-Transceiver MCP2562. Quelle: Modifiziert nach [CAN18]

[5] Preis bei `https://www.conrad.de/de/p/joy-it-sbc-can01-can-interface-1-st-passend%2Dfuer-entwicklungskits-arduino-banana-pi-raspberry-pi-cubieboard-1720599.html` [Eingesehen am 24.08.2021, 10:29 Uhr]

Leonardo CAN BUS Board

Eine weitere Alternative stellt das Leonardo CAN BUS Board (siehe Abbildung 4.6) der Firma HobbyTronics dar. Die auf dem Arduino Leonardo basierende Platine enthält neben dem ATmega32U4 als Mikrocontroller die nötigen Controller und Transceiver zur Unterstützung des CAN-Protokolls. Mit CAN V2.0B werden Datenraten bis 1 MB/s unter Verwendung der Standard- und erweiterten Daten- sowie Remote-Frames unterstützt. Zur direkten Nutzung der CAN-Verbindung verfügt das Board über einen D-Sub-9-Stecker und eine vier-polige Klemmenverbindung. Der Stecker sowie die Stiftleisten werden separat geliefert und können je nach Gebrauch selbst angelötet werden. Da der D-Sub-9-Stecker für diese Entwicklung nicht benötigt wird, könnte durch das Weglassen dieses Bauteils Größe eingespart werden, sodass sich die Platinenmaße auf 51 x 33 mm beschränken. Als Programmierschnittstelle steht ein Mini-USB-B-Port zur Verfügung. Einen großen Nachteil stellt die Tatsache dar, dass es zum Leonardo CAN Bus Board kein Datenblatt gibt. Es verfügt über drei Analog- und neun Digital-Pins, jedoch ist nicht ersichtlich, ob die SPI-Pins zusätzlich zur Verfügung stehen oder für die Verwendung des CANs durch die Kommunikation mit dem implementierten CAN-Controller belegt sind. Darüber hinaus liegt der Preis bei £ 24, was 28 €[6] entspricht und somit deutlich teurer als die bisher vorgestellten Lösungen ist. [LEO21]

Abbildung 4.6: Leonardo CAN Bus Board von HobbyTronics. Quelle: [LEO22]

Aus den eben vorgestellten Mikrocontrollern erweist sich der Arduino Nano V3 von Joy-It auf Grund seiner Größe, der Anzahl an freien Pins und dessen Preis als die beste Alternative.

[6] Stand: 20.08.2021

4.3.2 Gateway

Zur Realisierung der drahtlosen Kommunikation zwischen dem Demonstrator und der CAN-Analysesoftware auf dem Computer werden, äquivalent zum bisherigen Aufbau, zwei Mikrocontroller benötigt, welche als Gateways fungieren.
Als Anforderungen ergeben sich aus seiner Funktionalität die CAN-Fähigkeit sowie das Unterstützen einer Funktechnologie zur Datenübertagung. Wichtig ist zudem eine hohe Taktfrequenz, um die CAN-Botschaften unverzüglich weiterleiten zu können. Wie auch die Steuergeräte soll das Gateway möglichst Arduino-kompatibel sein, um eine Vereinheitlichung der Programmierung erzielen zu können. Auf Grund der, sich durch den Einsatz in der Lehre ergebenden, begrenzten Ressourcen soll der Mikrocontroller möglichst preisgünstig sein. Zudem ist der begrenzte Platz auf dem Robot Car Kit zu beachten. Im Folgenden werden drei Mikrocontroller miteinander verglichen.

ESP32-EVB

Bisher ist das Evaluationsboard ESP32-EVB von Olimex in Abbildung 4.7 für die drahtlose Kommunikation zwischen CAN-Demonstrator und Analysesoftware zuständig. Diese Platine mit ESP32-Controller stellt eine Komplettlösung auf Grund des implementierten CAN-Transceivers und entsprechender Schnittstelle dar. Für die Umsetzung der drahtlosen Kommunikation stehen WiFi und Bluetooth Low Energy (BLE) zur Verfügung. Bei einer Taktfrequenz von bis zu 240 MHz kann eine unverzügliche Weiterleitung der CAN-Botschaften gewährleistet werden. Ein integrierter Programmierer für Arduino und das Espressif-IoT Development Framework erleichtert die Handhabung und erfüllt die definierte Anforderung. Mit Abmessungen von 75 x 75 mm ist das Board verhältnismäßig groß und benötigt durch die darauf verbauten Relais sowie der Ethernet-Schnittstelle viel Platz in die Höhe. Aus dem Stromlaufplan des ESP32-EVB lässt sich entnehmen, dass ein Großteil der zur Verfügung stehenden Pins bereits intern belegt ist.
Erhältlich ist das ESP32-EVB derzeit für 26 €[7]. [RIO21, ESP21, ESP18]

[7] Preis bei `https://www.olimex.com/Products/IoT/ESP32/ESP32%2DEVB/open%2Dsource%2Dhardware` [Eingesehen am 12.05.2021, 11:48 Uhr]

Abbildung 4.7: ESP32-EVB von Olimex. Quelle: [RIO21]

NodeMCU-ESP32

Die in Abbildung 4.8 dargestellte NodeMCU-ESP32 der Firma Joy-It ist, mit Maßen von 48 x 11,5 mm eine kompakte Entwicklungsplatine mit ESP-WROOM32-Modul, welche sich über die Arduino IDE programmieren lässt. Bei einer Taktfrequenz von bis zu 240 MHz kann die Anforderung der schnellen Bearbeitung der CAN-Botschaften erfüllt werden. Neben 21 Pins zur Schnittstellenverbindung verfügt es über Wi-Fi und Bluetooth. Die NodeMCU-ESP32 unterstützt grundlegend das CAN-Protokoll. Der implementierte CAN-Controller ist kompatibel mit dem Standard-Frame-Format und dem Extended-Frame-Format von CAN2.0. Um CAN optimal nutzen zu können, muss die Platine um einen Transceiver erweitert werden. Hierfür eignet sich der High-Speed-CAN-Transceiver VP230 des Herstellers Texas Instruments, welcher als Schnittstelle zwischen dem CAN-Controller und dem physischen CAN-Bus fungiert. Mit einer Datenrate von 1 Mbit/s erfüllt der Chip die Geschwindigkeitsanforderungen im realen Automobil. Erhältlich ist die NodeMCU-ESP32 für 11,50 €[8], der CAN-Transceiver VP230 für 2,67 €[9] pro Stück. [NOD18]

[8] Preis bei `https://www.reichelt.de/nodemcu%2Desp32%2Dwifi%2Dund%2Dbluetooth%2Dmodul%2Ddebovjt%2Desp32%2Dp219897.html` [Eingesehen am 12.05.2021, 13:40 Uhr]

[9] Preis bei `https://www.mouser.de/ProductDetail/Texas-Instruments/SN65HVD230D?qs=QViXGNcIEAskSxPh50NSwA%3D%3D` [Eingesehen am 06.07.2021, 09:05 Uhr]

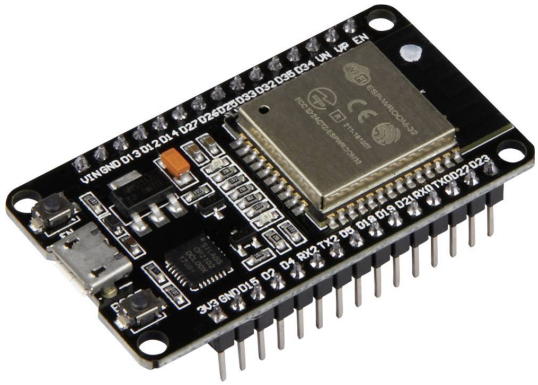


Abbildung 4.8: NodeMCU-ESP32 der Firma Joy-It. Quelle: [NOD18]

Arduino Nano 33 IOT

Als weitere Möglichkeit ergibt sich auf Grund seiner kleinen Größe von 45 x 18 mm der in Abbildung 4.9 dargestellte Arduino Nano 33 IOT. Dieses Board ist eine Weiterentwicklung des klassischen Arduino Nano, besitzt jedoch mit dem 32-Bit SAMD21 von Arm Cortex einen stromsparenden Prozessor mit einer Taktfrequenz von bis zu 48 MHz. Der Nano IOT verfügt wie die anderen beiden Entwicklungsplatinen über eine WiFi- und Bluetooth-Konnektivität. Zur Unterstützung des CAN-Protokolls wird wie beim Arduino Nano beschrieben zusätzlich ein CAN-Transceiver und -Controller benötigt. Preislich liegt diese Variante derzeit bei 22,20 €[10]. [ARD33]

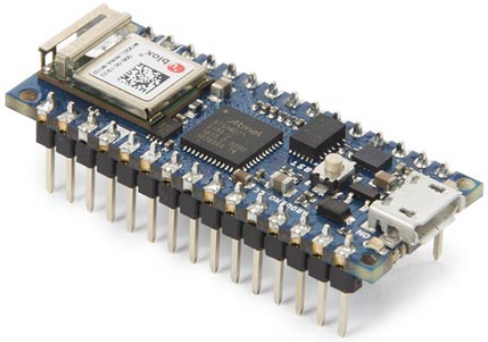


Abbildung 4.9: Arduino Nano 33 IOT von Arduino. Quelle: [ARD19]

[10] Preis bei `https://www.reichelt.de/arduino%2Dnano%2D33%2Diot%2Dsamd21g18a%2Dmit%2Dheader%2Dard%2Dnano%2D33iot%2Dh%2Dp261303.html` [Eingesehen am 11.06.2021, 8:49 Uhr]

Letztendlich ergibt sich auf Grund seines Preises und der hohen Taktfrequenz, die NodeMCU-ESP32 als die beste Alternative für diese Arbeit.

4.3.3 Motortreiber

Zur Steuerung der Motoren wird neben dem entsprechenden Steuergerät, welches die Steuerungsinformationen zur Regulierung dieser gibt, ein Motortreiber benötigt.
Dieser übersetzt die Kommandos des Mikrocontrollers in die vom Motor benötigten Stromstärken. Gleichzeitig wird der Mikrocontroller vor etwaigen Kurzschlüssen oder Überspannungen, die seitens des Motors entstehen könnten, geschützt.
Aus dem Datenblatt des Robot Car Kits [ROB20] ist zu entnehmen, dass die Motoren zur Betreibung eine elektrische Spannung von 3–9 V benötigen. Vom Hersteller werden 4,5 empfohlen, bei einer Stromstärke von 200 mA. Hinsichtlich des begrenzten Platzes auf dem Demonstrator wird nach einer möglichst kleinen Komponente gesucht.

Insgesamt wurden drei Varianten miteinander verglichen.

EVAL6207N

Bei der Evaluierungsplatine EVAL6207N (siehe Abb. 4.10) der Firma STMicroelectronics, welche bisher im CAN-Demonstrator implementiert ist, handelt es sich um einen Dual Full Bridge Driver mit PWM-Stromregler. Mit Abmessungen von 65 x 58 x 16 mm ist die Platine sehr groß und nimmt enorm viel Platz ein. Erhältlich ist das Modul derzeit für etwa 23 €[11], gilt jedoch als veraltet und wurde vom Hersteller abgekündigt. [EVA62]

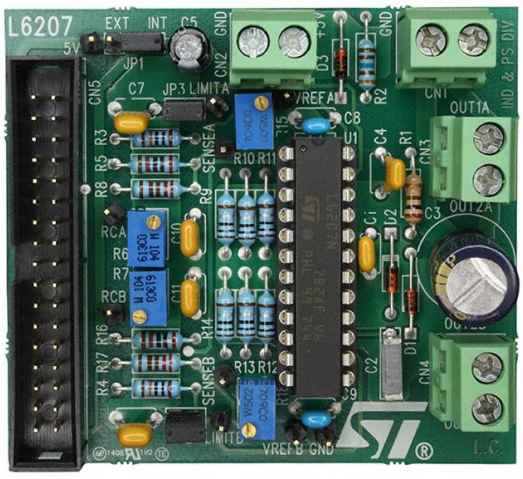


Abbildung 4.10: Motortreiber EVAL6207N der Firma STMicroelectronics. Quelle: [EVA21]

[11] Preis bei `https://www.mouser.de/ProductDetail/STMicroelectronics/EVAL6207N?qs=9iSXU9KXrZmaiw9N5nxjTg==` [Eingesehem am 12.05.2021, 12:05 Uhr]

MotoDriver2

Der MotoDriver2 in Abbildung 4.11 ist eine Erweiterungsplatine zur Ansteuerung von zwei Gleichstrommotoren, welcher eine Steuerung mit einer konstanten Spannung zwischen 5 und 35 V ermöglicht. Der darauf verbaute Treiber L298N realisiert bei einem Antriebsstrom von 2 A eine maximale Leistung von 25 Watt. Bei einer Abmessung von 43 x 43 x 27 mm ist die Platine deutlich kleiner als das bisherige Modell, hingegen durch den darauf verbauten Kühlkörper ziemlich hoch. Erhältlich ist das Motormodul von Joy-It derzeit für 5,89 €[12]. [MOT21]

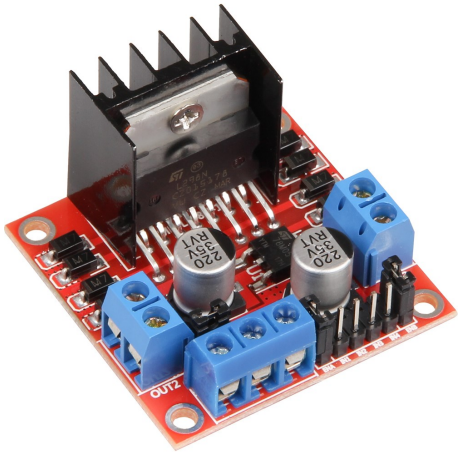


Abbildung 4.11: Motortreiber MotoDriver2 der Firma Joy-It. Quelle: [MOT21]

TB6612

Der Motortreiber TB6612 von Adafruit (siehe Abb. 4.12) ist eine Erweiterungsplatine, welche das Drehen von zwei Gleichstrommotoren mit PWM bis 1,2 A pro Kanal ermöglicht. Jeder Breakout-Chip enthält zwei volle H-Brücken mit jeweils zwei digitalen Eingängen. Das Motormodul eignet sich für Motorspannungen von 4,5 bis 13,5 V und kann wie der Arduino mit 5 V betrieben werden. Mit Maßen von 27 x 19 x 3 mm ist dies die kleinste Alternative unter den betrachteten Bauteilen. Erhältlich ist der Motortreiber TB6612 derzeit für 4,19 €[13]. [LAD20]

Folglich erweist sich der Motortreiber TB6612 auf Grund seiner kleinen Größe als die beste Lösung für diese Arbeit.

[12] Preis bei `https://www.conrad.de/de/p/joy%2Dit%2Dmotormodul%2D2%2Du%2D4%2Dphasen%2D6%2Dbis%2D12v%2D1573541.html` [Eingesehen am 06.07.2021, 9:48 Uhr]

[13] Preis bei `https://www.mouser.de/ProductDetail/Adafruit/2448?qs=GURawfaeGuCkWVI5OMiFVg%3D%3D` [Eingesehen am 12.05.2021, 14:33 Uhr]

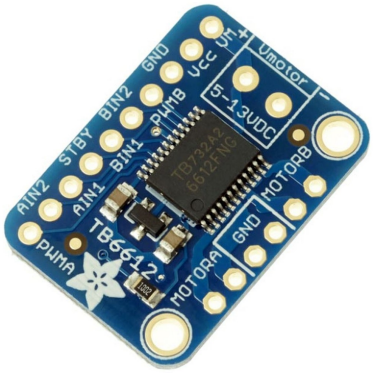


Abbildung 4.12: Motortreiber TB6612 von Adafruit. Quelle: [EVE21]

4.3.4 Stromversorgung

Als zentraler Gegenstand ist die Stromversorgung aller Komponenten zu sehen. Wie der bisherige CAN-Demonstrator soll die Version zwei auch mit einer mobilen Stromversorgung ausgestattet werden. Ein Akkumulator wird hier einer Batterie aufgrund der Nachhaltigkeit, durch die Möglichkeit des mehrfachen Aufladens, vorgezogen. Dieser soll wieder fest im Demonstrator verbaut sein. Die Kapazität des Akkumulators ergibt sich aus dem Verbrauch der verbauten Bauteile. Eine Übersicht ist Tabelle 4.1 zu entnehmen.

Tabelle 4.1: Details über die Stromversorgung der Bauteile des Demonstrators.

Bauteil	**U_{In} in V**	**I in mA**	**Quelle**
Arduino Nano (3 Stück)	7 - 12	je 19	[ARD20]
CAN-Modul (3 Stück)	5	je 5	[CAN18]
NodeMCU-ESP32	3,3 - 15	80	[NOD18, NOD21]
VP230	3 - 3,6	17	[TEX18]
TB6612	2,7 - 5,5	2,2	[LAD20, TOS07]
Motoren (4 Stück)	3 - 9	je 200	[ROB20]
LEDs (10 Stück)	5	47	Messung: Rot 6 mA (3x); Gelb 4 mA (4x); Weiß 5,5 mA (2x), 2 mA (1x)
Hupe	4 - 8	30	[BUZ21]
Distanzsensor (2 Stück)	5	je 15	[DIS17]
Gesamt		**1.078,2**	

Aus der Berechnung des Verbrauchs der Bauteile ergibt sich, dass ein Akkumulator benötigt wird, welcher mindestens eine Spannung von 5 V liefert und eine Kapazität von rund 1100 mAh aufweist. Mit dieser Leistung könnte der Demonstrator eine Stunde betrieben werden.
Um den Demonstrator in der Lehre einsetzen zu können, wäre eine Akkulaufzeit von mindestens 90 Minuten wünschenswert. Entsprechend ergibt sich hier eine Anforderung an die Kapazität von rund 1700 mAh.
Neben der soeben berechneten Leistungsfähigkeit des Akkumulators sollte dieser möglichst platzsparend im Hinblick auf den Einbau im CAN-Demonstrator sein und einen geringen Einkaufspreis besitzen. Um den Akkumulator dauerhaft im Demonstrator verbauen zu können, ist ein separater Anschluss für ein Ladegerät nötig.

LiPo 20C

Bei dem bisher im CAN-Demonstrator verbauten Modellbau-Akkupack handelt es sich um einen LiPo 20C mit 7,4 V und 3000 mAh bei einer Zellen-Zahl von zwei. Mit Abmessungen von 139 x 25 x 46 mm, bei einem Gewicht von 201 g erreicht dieser für den Verbrauch der Bauteile eine Betriebsdauer von ca. 2,7 h. Kostentechnisch liegt dieses Modell bei 29,99 €. [LIP20]

LiPo 25C

Eine etwas kleinere und günstigere Alternative ist der Lithium-Polymer-Akkupack 25C. Dieser hat einer Kapazität von 2700 mAh bei einer Spannung von 7,4 V. Für den, 105 x 35 x 18 mm großen und 135 g schweren Akkumulator ergibt sich eine Betriebsdauer von etwa 2,45 h. Dieser liegt preislich bei 20,40 €. [LIP25]

Da der LiPo 25C mit seiner Leistung die 90 Minuten Einsatz in der Lehre gut abdecken kann, wurde sich für diese kleinere und preisgünstigere Alternative entschieden.

4.3.5 Platinen

Um den Demonstrator noch umgänglicher für den Einsatz in der Lehre zu gestalten, soll die bisherige Anordnung der Mikrocontroller und Verkabelung dieser überdacht werden.

Mittelplatte

Zur Vermeidung von unübersichtlichen Verkabelungen soll der Demonstrator um eine zusätzliche Bodenplatte erweitert werden, sodass zwei getrennte Ebenen entstehen. Zwischen den unteren beiden Platten sollen dabei Komponenten verbaut werden,

welche nur in ganz seltenen Fällen zugänglich sein müssen, wie beispielsweise die Motoren, der Akku und entsprechende Kabel. Auf der mittleren Platte sollen die Mikrocontroller mit ihren CAN-Modulen befestigt werden. Diese sind somit für eine Programmierung schnell und einfach zugänglich. Die Komponenten der Mittelplatte sollen durch Steckverbindungen befestigt werden, sodass diese abnehmbar sind und sich die Verkabelung vollständig zwischen Boden- und Mittelplatte befindet. Dies würde gleichzeitig das Problem der offenliegenden Kabel minimieren.

Bodenplatine

Um weitere Kabelwege zu verkürzen oder diese ganz einzusparen, soll für die Bodenplatte eine Platine entworfen werden, auf welcher die dort zu implementierenden Komponenten wie LEDs, Distanzsensor und Hupe angebracht werden können. Die Leiterbahn würde somit innerhalb der Platine laufen und weitere freiliegende Kabel reduzieren. Auch der bisher lose liegende Spannungsteiler für die Akkustandsüberwachung soll direkt in diese Platine implementiert werden. Zur Befestigung der Platine auf der Bodenplatte könnte diese angeschraubt oder aufgeklebt werden.

4.4 Softwareentwurf

Zentraler Punkt dieser Arbeit ist der Softwareentwurf. Neben der Implementierung und Ansteuerung unterschiedlicher Mikrocontroller durch einen einheitlichen Programmcode, wird eine Funktechnologie für die drahtlose Kommunikation zwischen dem CAN-Demonstrator und der CAN-Analysesoftware auf dem Computer benötigt.

4.4.1 Softwarearchitektur

Die Nutzung unterschiedlicher Mikrocontroller bedarf auf Grund der abweichenden Unterstützung des Controller Area Networks dem Gebrauch diverser Bibliotheken. Diese bringen unterschiedliche Parameter und Variablen mit sich, wodurch eine einfache Zusammenführung der Firmware erschwert wird. Für eine einfache Anwendung ist jedoch eine einheitliche Ansteuerung essentiell. Da die Funktionsweise des CANs unabhängig der eingesetzten Hardware als identisch anzusehen ist, können die Unterschiede in der CAN-Kommunikation durch den Einsatz der modularen CAN-Bibliothek [ELI20] mit Hilfe spezifischer Funktionen ausgeglichen werden.

Zur Realisierung eines Programmcodes für alle Komponenten des CAN-Demonstrators lässt sich die Softwarearchitektur um eine zusätzliche Ebene erweitern. Das in Abbildung 4.14 dargestellte Modell der Softwarearchitekturebenen besteht aus drei Schichten. Im Folgenden wird nur der Weg der CAN-Kommunikation betrachtet.

Die Steuerungsebene beinhaltet die hardwarespezifischen Funktionen für den Empfang und das Aussenden der CAN-Frames. Hierfür kommen controllerspezifische Bibliotheken zum Einsatz. Die Aufgabe der Bearbeitungsebene ist die Konvertierung dieser noch spezifischen CAN-Botschaften in eine universelle CAN-Struktur zur Verarbeitung in der Anwendungsschicht und umgekehrt. Ihre Hauptfunktionalität besteht in der Anpassung und Aufbereitung der Daten. Die Anwendungsschicht beinhaltet den spezifischen Programmcode für die einzelnen Steuergeräte bzw. die Gateways.
Um die Ausführung der Funktionen zu gewährleisten, müssen die Schichten Zugriff auf die jeweils darüber oder darunter liegende Ebene erhalten. Den letzten Schritt nach unten bildet der Aufruf einer Übergabefunktion in die eigentliche Transportebene, auf der sich CAN, SPI und TCP befinden.

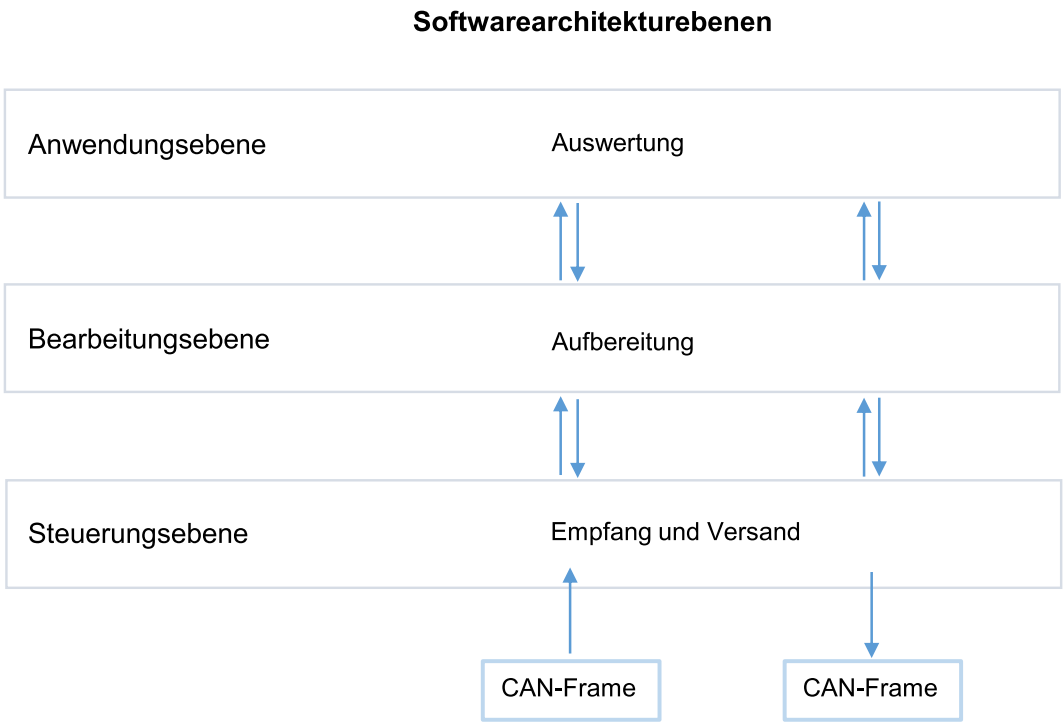


Abbildung 4.13: Modell der Softwarearchitekturebenen des Programmcodes für den CAN-Demonstrator 2. Quelle: Eigene Darstellung

4.4.2 Einordnung im OSI-Schichtenmodell

Der Aufbau eines CAN-Systems lässt sich wie in Abbildung 4.14 ersichtlich, in die drei Schichten – Bitübertragung, Sicherung und Anwendung – einordnen. Innerhalb der Bitübertragungsschicht (1) agiert der CAN-Transceiver. Die physische Ankopplung, wie beispielsweise die Pegelfestlegung, findet hier statt. Der CAN-Controller befindet sich im Aufbau des OSI-Modells in der Data Link Layer (2). Häufig ist er, wie beispielsweise bei der NodeMCU-ESP32, im Mikrocontroller integriert, kann jedoch auch als eigenständige Komponente auftreten. In das Aufgabengebiet des CAN-Controllers fallen in dieser Schicht der Buszugriff, die Prüfsummenberechnung und Nachrichtenfilterung. Der CAN-Treiber in der Application Layer (7) bildet die Schnittstelle zur Anwendung und findet sich als Softwarekomponente im Host-Controller als Treiber. Hier erfolgt die Durchführung von Feldbusaufgaben und die inhaltliche Zuordnung von Nachrichten.

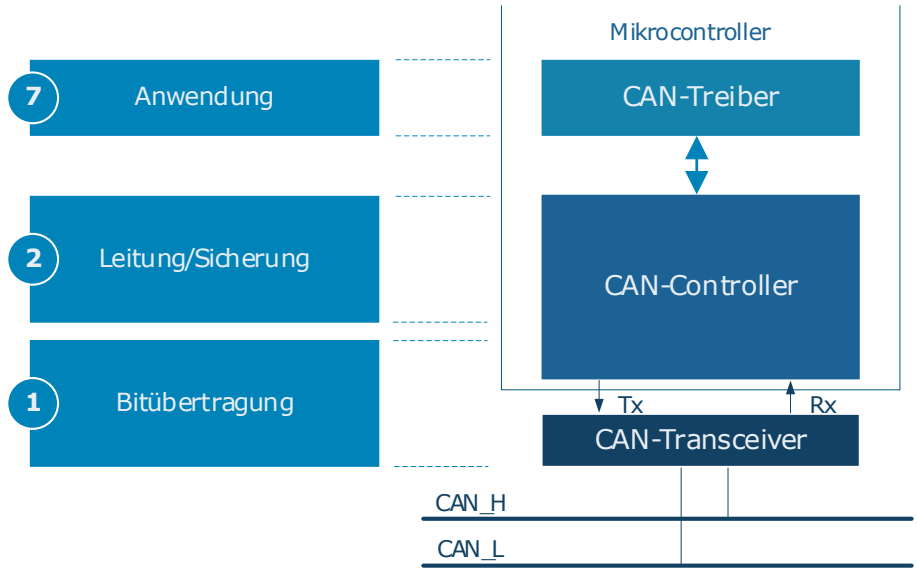


Abbildung 4.14: Einordnung des CAN-Systems im OSI-Schichtenmodell.
Quelle: [THO21, S. 28]

4.4.3 Wireless CAN

Zur Realisierung der drahtlosen CAN-Verbindung zwischen den beiden NodeMCU-ESP32, welche als Gateways fungieren, eignen sich die gängigen Übertragungsstandards Bluetooth und Wireless Local Area Network (WLAN), welche von dem ausgewählten Mikrocontroller unterstützt werden. Der CAN-Bus soll dadurch streckenweise ersetzt werden. Als Anforderungen ergeben sich aus der Funktionsaufgabe eine unmittelbare Weiterleitung der CAN-Botschaften. Hierfür muss die Datenrate des Highspeed-Busses von 1 Mbit/s gewährleistet sein. Ein niedriger Energieverbrauch zur Einsparung der Akkukapazität ist wünschenswert. Darüber hinaus soll der CAN-Demonstrator innerhalb eines Raumes zur Anwendung in der Lehre gesteuert werden können, weswegen

eine entsprechende Reichweite von 10 m unterstützt werden soll. Relevant ist zudem die Ausfallsicherheit.

Im Folgenden werden die beiden genannten Standards verglichen.

WLAN

Die NodeMCU-ESP32 unterstützt den WLAN-Standard 802.11 b/g/n bei einer Frequenz von 2,4 GHz [NOD18]. Auf Grund der Kanalbreite von 40 MHz kann IEEE 802.11n bei einer Datenrate von bis zu 150 Mbit/s eine maximale Entfernung von 100 m überbrücken und entspricht somit den Anforderungen. Die WLAN-Technologie bringt generell einen größeren Stromverbrauch mit sich. Da sich diese jedoch am Mikrocontroller nicht ausschalten lässt, ist dieses Kriterium hinfällig. Die Ausfallsicherheit ist von dem entsprechenden Netzwerkprotokoll abhängig. [ESP31, PAT80, PAT21]

Bluetooth

Die NodeMCU-ESP32 verfügt über ein integriertes Bluetooth v4.2 Low Energy [NOD18], was der energiesparenden Anforderung entgegenkommt. Dem Datenblatt ist zu entnehmen, dass Bluetooth und WiFi über ein Modul realisiert werden und dadurch nicht einzeln abgeschaltet werden können. Das oben genannte Kriterium der Energieeffizienz kann somit nicht erfüllt werden.
Wie der WLAN-Standard, sendet das in IEEE 802.15 spezifizierte BLE im Frequenzbandbereich von 2,4 GHz. Auf Grund des implementierten Klasse 1 Bluetooth-Transmitters erzielt die NodeMCU-ESP32 ebenfalls maximale Reichweite von 100 m bei einer Datenrate von 1 Mbit/s, was der Datenrate des High-Speed-CANs entspricht. Durch den zu beachtenden Payload kann die korrekte Übertragung des CAN-Frames über Bluetooth nicht garantiert werden. Bei Störungen werden die Pakete automatisch neu übertragen. [SAU13, S.258f], [PAT51], [ESP31]

Der WLAN Standard IEEE 802.11n und Bluetooth Low Energy haben einige Gemeinsamkeiten wie das Frequenzband von 2,4 GHz und die Reichweite von 100 m, jedoch erreicht WLAN eine deutlich höhere Datenrate. Da Bluetooth dieselbe Datenrate wie das High-Speed-CAN aufweist, kann eine korrekte Übertragung nicht gewährleistet werden, weswegen sich dieser Funkstandard nicht für die Realisierung dieser Arbeit eignet. Auf Grund der zusätzlichen Hinfälligkeit der Anforderung an den Stromverbrauch ergibt sich demnach WLAN als für diese Arbeit geeignete Lösung.
Als Übertragungsprotokoll wurde das Transmission Control Protocol verwendet, welches eine Punkt-zu-Punkt-Verbindung zwischen den beiden Teilnehmern ermöglicht und als verbindungsorientiertes, sicheres Protokoll gilt. TCP stellt dabei sicher, dass die Daten zuverlässig, unverändert und in der richtigen Reihenfolge zugestellt werden. [MAN18, S. 34f]

In der bestehenden Version des CAN-Demonstrators wurde diese drahtlose Verbindung bereits via WLAN und TCP umgesetzt. Grundlegend kann der bisherige Programmieransatz verwendet werden, muss jedoch an einigen Stellen überarbeitet und an das neue System angepasst werden.

4.4.4 CAN-Kommunikation

Die Realisierung der CAN-Kommunikation zwischen Steuergerät und dem CAN-Modul lässt sich über die SPI-Schnittstelle realisieren. Diese wurde bereits in vorangegangener Arbeit umgesetzt und wird entsprechend übernommen [ELI20]. Es wurde die Arduino MCP2515 CAN Interface Library [STD16] verwendet.

4.4.5 Motorsteuerung

Zur Regulierung der Motoren müssen die entsprechenden Informationen der eintreffenden CAN-Botschaft ausgewertet und der Motortreiber angesteuert werden.
Entsprechend der gewünschten Fahrtrichtung müssen die Pegel High und Low gesetzt werden. Zur Geschwindigkeitsregulierung wird eine Pulsweitenmodulation benötigt. Um zusätzlich zur Fahrtrichtung vorwärts und rückwärts den Demonstrator auch jeweils nach links oder rechts steuern zu können, wird die Geschwindigkeit auf der Seite der Richtung, in die gefahren werden soll, reduziert. Tabelle 4.2 enthält die daraus resultierenden Schaltzustände der Pins.

Tabelle 4.2: Schaltzustände der Pins für die Motoren

Motor A		**Motor B**		**Zustand**
A1	**A2**	**B1**	**B2**	
HIGH	HIGH	HIGH	HIGH	Stopp
LOW	LOW	LOW	LOW	Stopp
LOW	HIGH	LOW	HIGH	Fahrtrichtung vorwärts
HIGH	LOW	HIGH	LOW	Fahrtrichtung rückwärts

Des Weiteren soll jeder CAN-Knoten im Demonstrator als eigenständige CAN-Bus-Steuereinheit fungieren. Dies bedeutet speziell für das Motorsteuergerät, dass es selbstständig auf CAN-Nachrichten im Bus reagiert. Wird beispielsweise eine CAN-Botschaft der Distanzmessung mit kritischem Wert auf den Bus gesandt, soll das Motorsteuergerät mit einem „Hard-Stop“ der Motoren reagieren.

Die Programmierung der Motorsteuerung des bestehenden Demonstrators kann auf Grund der neu gewählten Hardware, dem Ziel der Codevereinheitlichung sowie der Funktionalität als eigenständige Einheit nicht übernommen und muss neu erarbeitet werden.

5 Implementierung

In diesem Kapitel wird die Umsetzung der zuvor im Konzeptionsteil erarbeiteten Anforderungen beschrieben. Hierbei wird auf die Realisierung der Hardware und die Besonderheiten der Programmierung eingegangen sowie Probleme in deren Umsetzung deutlich gemacht.

5.1 Hardwareentwicklung

In diesem Abschnitt wird der tatsächliche Prozess des Aufbaus des Demonstrators beschrieben. Alle bereits genannten Komponenten werden hier zusammengeführt und physisch miteinander verbunden.

5.1.1 Steuergerät

Der Demonstrator soll entsprechend dem realen CAN-Aufbau mit drei Steuergeräten, je einem für die Motorsteuerung, einem für die Beleuchtung und einem für zusätzliche Autoelektronik ausgestattet werden. Der hierfür ausgewählte Arduino Nano V3 ist mit einem ATmega328P-AU Mikrocontroller versehen. Nähere Informationen hierzu können dem Datenblatt [ARD20] entnommen werden. Da der Mikrocontroller jedoch nicht CAN-fähig ist, muss dieser entsprechend um einen CAN-Transceiver sowie -Controller erweitert werden. Hierfür eignet sich das in Kapitel 4.3.1 genannte CAN-Modul der Firma Joy-It [CAN18]. Dieses verfügt über einen MCP2515 sowie einen MCP2562 und kann über SPI mit dem Arduino Nano verbunden werden. Eine Skizze der physischen Verbindung der beiden Module ist in Abbildung 5.1 gegeben.

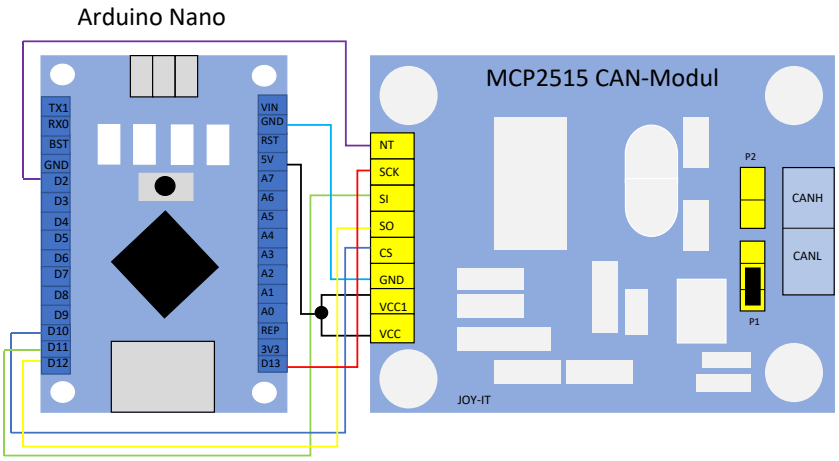


Abbildung 5.1: Skizze der physischen Verbindung zwischen Arduino Nano und CAN-Modul. Quelle: Eigene Darstellung

Da sowohl der Arduino Nano als auch das CAN-Modul mit einer Spannung von 5 V betrieben werden, kann dieses direkt vom Arduino mit Strom versorgt werden. Der Arduino selbst wird direkt mit dem Akkumulator verbunden, da dieser mit einer Eingangsspannung von 7-12 V betrieben werden kann [ARD20]. Eine Übersicht der Anschlüsse ist Tabelle 5.1 zu entnehmen.
Der für CAN-Highspeed-Netzwerke benötigte Terminalwiderstand von 120 Ω kann am Anschluss P1 des CAN-Moduls aktiviert werden.

Tabelle 5.1: Anschlussplan für die Verbindung zwischen Arduino Nano und CAN-Modul. Quelle: [JOY18, S. 1]

Arduino Nano	**CAN-Modul**
5 V	VCC
5 V	VCC1
GND	GND
D10	CS
D12	S0
D11	SI
D13	SCK
D2	NT

5.1.2 Gateway

Als zentrales Gateway im Demonstrator soll die NodeMCU-ESP32 [NOD18] fungieren. Zur optimalen Nutzung des CANs wird der ausgewählte High-Speed-CAN-Transceiver VP230 mit der NodeMCU-ESP32 verbunden.
Da der VP230 nicht sinnvoll an der NodeMCU-ESP32 befestigt werden kann und gerade für das externe Gateway eine handliche Lösung gefunden werden muss, wurde für die Anbindung des genannten Gateways an die CAN-Analysesoftware eine kompakte Platine entworfen[14], welche neben einem Steckplatz für die NodeMCU-ESP32 eine RS-232-Schnittstelle gemäß CiA/DS 102 zur Verbindung des CAN-Interfaces VN1610 besitzt. Mit dieser Lösung können alle Elemente bündig zusammengebracht werden, wodurch eine „studentensicherere“ Umsetzung auf PC-Seite als bei der Kabellösung der alten Variante entsteht.
Der VP230-CAN-Transceiver wurde entsprechend im Nachgang aufgelötet, wobei CAN-High mit Pol 7 und CAN-Low mit Pol 2 des Steckverbinders verbunden wurden. Des Weiteren sind mehrere Kondensatoren zum Schutz der Bauelemente vor Störimpulsen vorgesehen sowie ein Widerstand in Reihe von 10 kΩ zwischen dem Rs-Pin und der Masse. Dieser ermöglicht einen Flankensteuerungsmodus.

[14] Siehe Git-Projekt ACMC: `https://gitlab.hrz.tu-chemnitz.de/ACMC/Lehrsysteme/CAN-Demonstrator/hardware/esp-can-adapter.git`

Über einen Jumper kann der 120 Ω-Terminalwiderstand für Highspeed-CAN aktiviert werden. Abbildung 5.2 zeigt die Platine des externen Gateways ohne Controller.

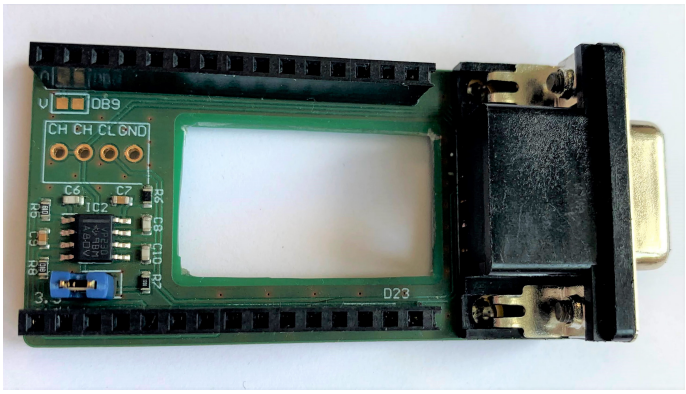


Abbildung 5.2: Platine des VP230 auf Client-Seite. Quelle: Eigene Darstellung

Für den Transceiver im Demonstrator wurde ebenfalls eine kleine Platine entworfen (siehe Abbildung 5.3), welche neben dem VP230 und den zuvor genannten Widerständen sowie Kondensatoren eine dreipolige Klemme für CAN-High, CAN-Low und Masse enthält. Abbildung 5.4 zeigt die physische Verbindung zwischen der NodeMCU-ESP32 und der Serverplatine mit VP230-Transceiver.

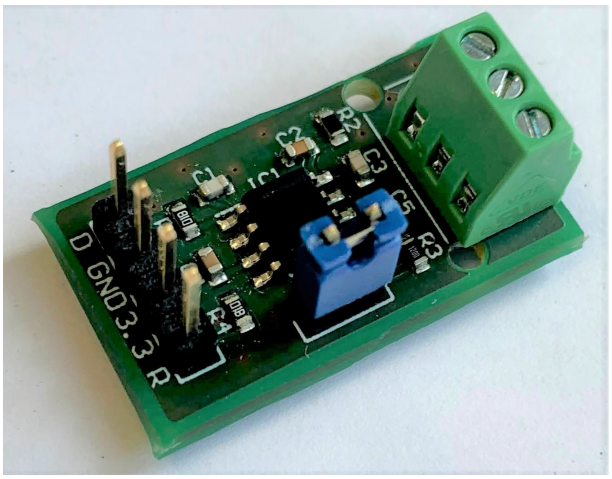


Abbildung 5.3: Platine des VP230 auf Server-Seite. Quelle: Eigene Darstellung

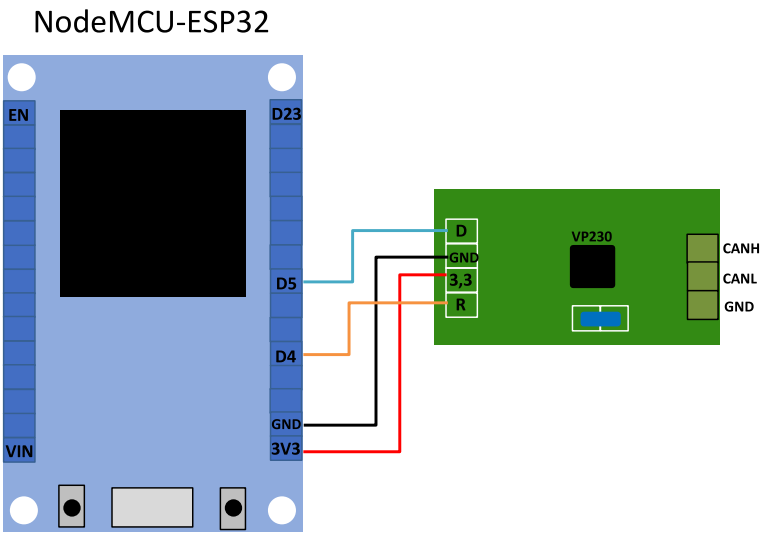


Abbildung 5.4: Skizze der physischen Verbindung zwischen der NodeMCU-ESP32 und der Server-Platine mit VP230-Transceiver. Quelle: Eigene Darstellung

5.1.3 Motorsteuerung

Zur Steuerung der Motoren wird neben dem Arduino Nano als Motorsteuergerät der Motortreiber TB6612 von Adafruit verwendet. Dieses Breakout-Board erhält die entsprechenden Steuerungsinformationen zur Betreibung der Motoren über eine physische Verbindung durch den, auf dem Arduino Nano implementierten, ATmega328P-AU. Zur Steuerung der vier Motoren sind diese an das Board angeschlossen. Um die Übertragung der Signale des Motorsteuergeräts zu realisieren, müssen die entsprechenden Pins des Arduino Nanos mit jenen des Motortreibers TB6612 verkabelt sein. Die anliegenden Pegel werden dann eingelesen und ausgewertet. Die hierfür notwendigen Pins sind:

- PWM (Pulsweitenmodulation)
- IN1 (Input 1, Steuerung der Pegel)
- IN2 (Input 2, Steuerung der Pegel)

Zur Rotation der Motoren müssen diese an den Motortreiber angeschlossen und mit zwei unterschiedlichen Spannungspotentialen angesteuert werden. Die Drehrichtung hängt dabei von den Pegeln an den Input-Pins ab. Eine Geschwindigkeitsregulierung kann mit Hilfe einer Pulsweitenmodulation erfolgen. Hierfür müssen die PWM-Kanäle angeschlossen werden.
Da das Board zwei Ausgänge für die Ansteuerung besitzt, sind die Signal- und Motorausgangspins entsprechend mit A und B gekennzeichnet. Die Stromversorgung der Motoren erfolgt direkt durch den Akkumulator. Die physische Verbindung zwischen dem Arduino Nano und dem Motortreiber ist der Abbildung 5.5 zu entnehmen.

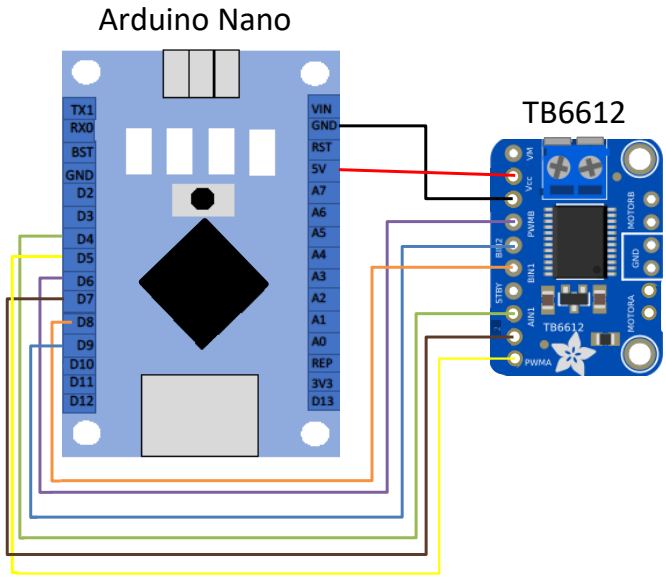


Abbildung 5.5: Skizze der physischen Verbindung zwischen dem Arduino Nano für die Motorsteuerung und dem Motortreiber TB6612. Quelle: Modifiziert nach [TTB66]

5.1.4 Weitere Funktionen

Im Rahmen einer anderen Arbeit [KRA21] wurden weitere Funktionen des CAN-Demonstrators entwickelt und implementiert. Diese werden im Folgenden kurz beschrieben.

Beleuchtung

Der in Kapitel 3.1.5 beschriebene Aufbau zur Beleuchtung im 2018 entwickelten CAN-Demonstrator wurde nicht verändert, sondern nur erweitert. Die implementierten Beleuchtungsfunktionen sind Abbildung 5.6 zu entnehmen.
Insgesamt verfügt der Demonstrator über 10 Leuchtdioden. An der Vorderseite befinden sich vier LEDs, je zwei zur Realisierung der Blinker (1, 4) und Frontscheinwerfer (2, 3), welche in der Helligkeit, je nach Funktionalität (Abblendlicht oder Fernlicht), mittels PWM reguliert werden können. Weitere vier LEDs befinden sich an der Rückseite. Je zwei für die Blinker (5, 10), zwei für das Rücklicht (6, 9) und je eine für das Rückfahrlicht (8) und die Nebelschlussleuchte (7). Äquivalent zu den Frontscheinwerfern wird das Rücklicht mittels PWM, je nach Funktion (Abblendlicht oder Bremslicht) in der Helligkeit gesteuert.
Die Stromversorgung der Leuchtdioden erfolgt durch den entsprechenden Arduino Nano und wird über Vorwiderstände reguliert. Als Befestigungslösung dienen Leiterplatten vorne und hinten.

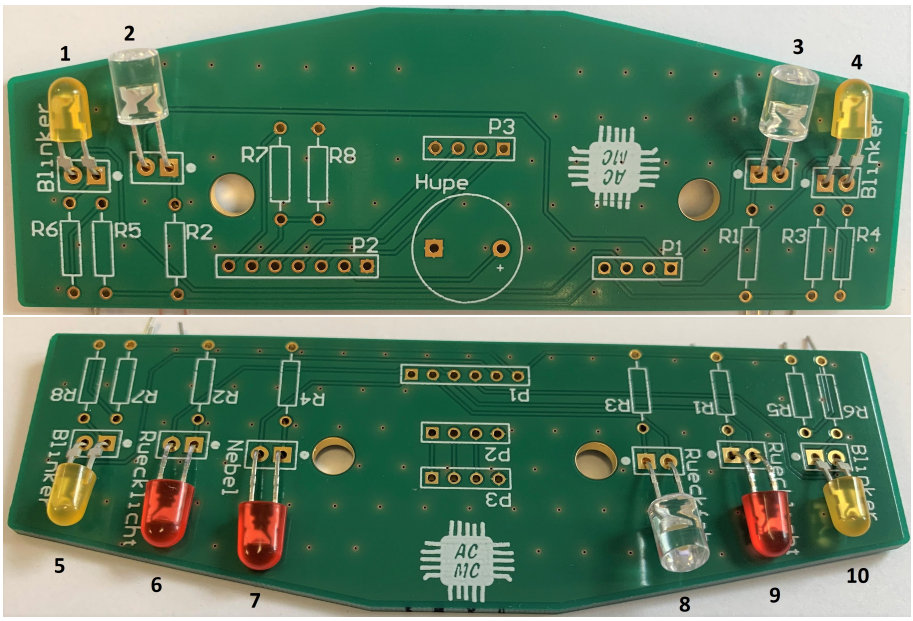


Abbildung 5.6: Übersicht über die implementierten Beleuchtungselemente. Quelle: Eigene Darstellung

Hupe

Wie bereits in Version 1 (vgl. Kapitel 3.1.8) wurde zur Imitation der Hupe der 2-Pin Lautsprecherstecker SUMMER AL-60P12 [BUZ21] implementiert. Dieser wird durch das Elektronik-Steuergerät (Arduino Nano) reguliert. Die Stromversorgung erfolgt ebenso durch den Arduino Nano.
Der Lautsprecher ist vorne auf der Platine angebracht.

Distanzmessung

Die Distanzmessung wurde sowohl in ihrer physischen Implementierung als auch softwareseitig vollständig wie im bestehenden Demonstrator übernommen. Gemessen wird jeweils nur in Fahrtrichtung. Ab einer Distanz von 100 cm zu einem Objekt werden CAN-Nachrichten, welche den Abstand beinhalten, im Intervall von einer Sekunde auf den Bus gesandt. Verringert sich die Distanz, erhöht sich das Sendeintervall. Ab einem Abstand von 50 cm zum Objekt erfolgt der Botschaftsversand aller 30 Sekunden, ab 30 cm aller 200 ms. Im Stillstand findet keine Distanzmessung statt.
Die beiden Ultraschallsensoren HC-SR04 [DIS17] wurden vorne und hinten auf die entworfenen Leiterplatten aufgelötet. Sie gehören dem Elektronik-Steuergerät an und werden durch dieses mit Strom versorgt.

Akkustandsüberwachung

Zur Überwachung des Ladezustands des Akkumulators im CAN-Demonstrator wird die anliegende Spannung über einen Analog-zu-Digital-Converter an der Elektronik-ECU (Arduino Nano) erfasst. Bei Erreichung eines bestimmten Schwellwerts wird eine Meldung ausgesandt, der Demonstrator beginnt zu hupen und stoppt. Da die maximale Spannung der Batterie 8,4 V beträgt, der Arduino Nano jedoch nur eine Betriebsspannung von 3,3 V hat, wird hierzu ein Spannungsteiler genutzt.

5.1.5 Stromversorgung

Für die ideale Stromversorgung wurde unter Einbeziehung der Verbrauchswerte der einzelnen Komponenten aus Tabelle 4.1 in Kapitel 4.3.4 und dem Augenmerk auf einer sinnvollen Anordnung der optimale Stromlaufplan entwickelt.
Direkt an den Akkumulator angeschlossene Module besitzen einen integrierten Spannungsregler. Die Spannung, mit welcher die nachgelagerten Module versorgt werden, kann Tabelle 4.1 entnommen werden.
Die Abbildung 5.7 zeigt eine grobe Skizze des Aufbaus.
Um die Stromversorgung regulieren zu können, bzw. den CAN-Demonstrator an- und ausschalten zu können, wurde ein Schalter in den Versuchsaufbau integriert.

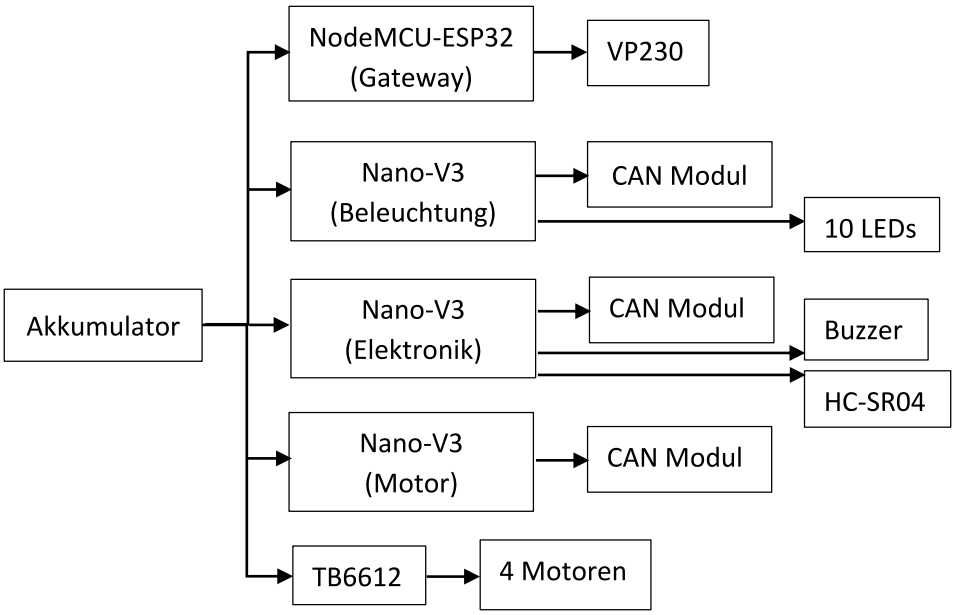


Abbildung 5.7: Übersicht der Stromversorgung im CAN-Demonstrator Version 2. Quelle: Eigene Darstellung

5.1.6 Platinen

Mittelplatte

Da die Bodenplatten des Robot Car Kits nicht einzeln erworben werden können und die vorgesehenen Bohrungen für diese Arbeit nicht hilfreich sind, wurde eine eigene Trägerplatte für die Mikrocontroller entworfen und mittels 3D-Druck gefertigt. Die Abbildung 5.8 zeigt die Unteransicht des entworfenen Modells dieser Platte[15].

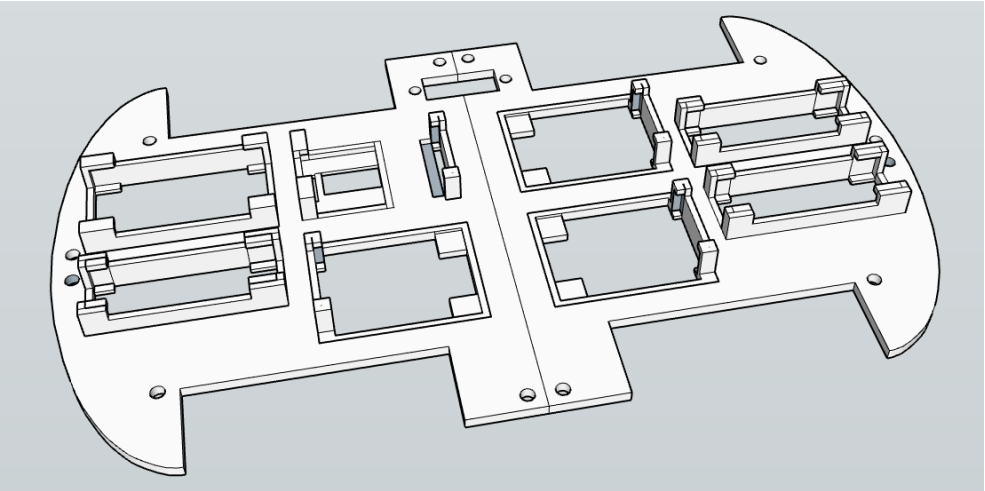


Abbildung 5.8: Unteransicht des SketchUp-Modells der Mittelplatte. Quelle: Eigene Darstellung

Die Mikrocontroller werden durch das Aufstecken auf Stiftleisten befestigt. Diese Stiftleisten wiederum sind in die nach unten ragenden Halterungen geklickt bzw. geklebt. Für den Druck wurde die Platte an der Mittellinie getrennt und in zwei separaten Teilen gefertigt. Abbildung 5.9 zeigt die gedruckte Platte mit den implementierten Komponenten.
Insgesamt entsteht durch die Verwendung der Trägerplatte eine Übersichtlichkeit bezüglich der Kommunikationswege des Controller Area Networks zwischen den einzelnen Komponenten.

[15] Siehe Git-Projekt ACMC: `https://gitlab.hrz.tu%2Dchemnitz.de/ACMC/Lehrsysteme/CAN%2DDemonstrator/hardware/can%2Ddemonstrator%2D3d%2Ddruck.git`

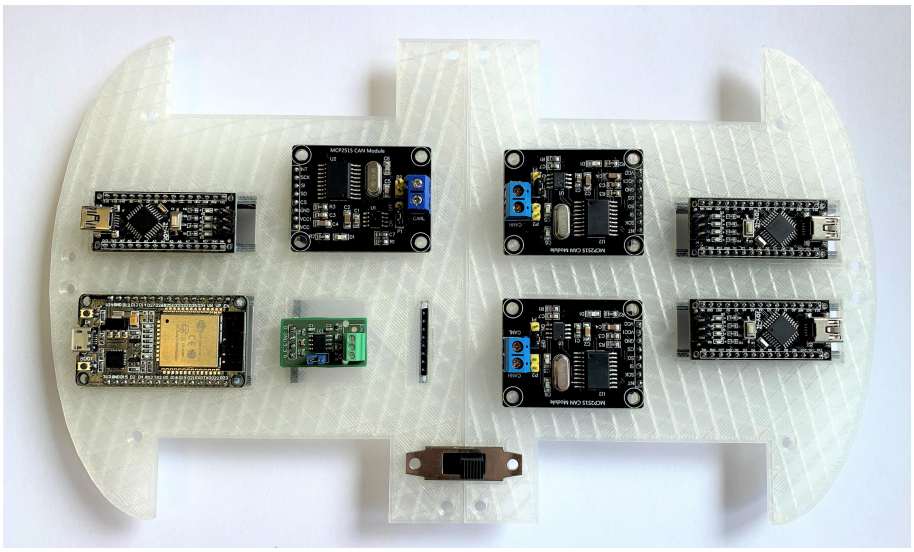


Abbildung 5.9: Gedruckte Mittelplatte mit Komponenten bestückt. Quelle: Eigene Darstellung

Bodenplatine

Zur Befestigung der LEDs, der Hupe sowie des Spannungsteilers für die Akkustands-überwachung auf der Bodenplatte wurde im Rahmen von [KRA21] jeweils eine Platine für die Front- und Heck-Seite des Demonstrators entworfen. Zur Befestigung auf der Bodenplatte dienen jeweils zwei Bohrungen. Die folgenden beiden Abbildungen 5.10 und 5.11 zeigen das Platinendesign. Die zugehörigen Schaltpläne sind Anhang A.1 und A.2 zu entnehmen.

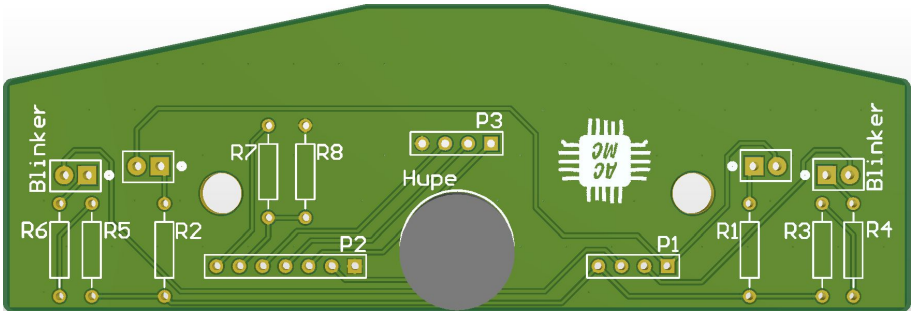


Abbildung 5.10: Platinendesign Bodenplatte vorne. Quelle: Eigene Darstellung

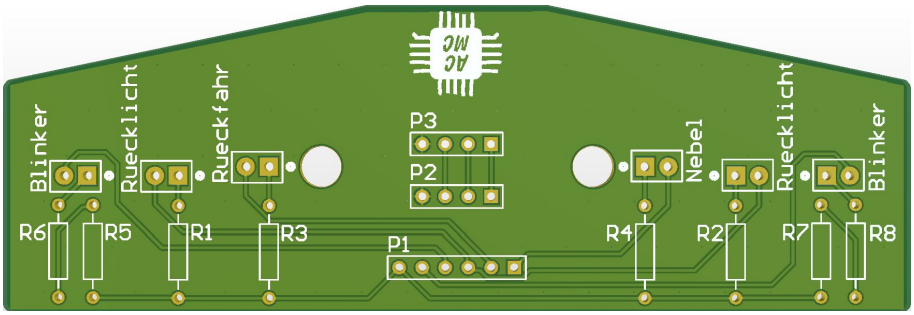


Abbildung 5.11: Platinendesign Bodenplatte hinten. Quelle: Eigene Darstellung

5.1.7 CAN-Bus

Der CAN-Bus des Demonstrators überträgt die Informationen zwischen den einzelnen Electronic Control Units (Arduino Nano) und dem Gateway (NodeMCU-ESP32) auf jenem. Für die Übermittlung der CAN-Botschaften zwischen dem externen Gateway und der CAN-Analysesoftware muss dieses über das CAN-Interface VN1610 mit dem Rechner verbunden werden.
Zur Vernetzung der Steuergeräte mit dem CAN-Bus werden die CAN-Transceiver des MCP2515-CAN-Moduls mit je zwei Kabeln für CAN-High (Pin CANH) und CAN-Low (Pin CANL) verbunden. Äquivalent werden die Pins CAN_H und CAN_L des VP230-CAN-Transceivers für das Gateway im Demonstrator angebunden. Im Unterschied zum Arduino Nano wird zusätzlich ein Widerstand von 10 kΩ zwischen dem Rs-Pin und der Masse zur Flankensteuerung benötigt.
Letztendlich werden alle CAN-High- und CAN-Low-Leitungen der Transceiver über eine Stiftleiste miteinander verbunden.

5.1.8 CAN-Demonstrator

Im letzten Schritt wurden alle zuvor beschriebenen Hardwarekomponenten auf dem Robot Car Kit implementiert. In Abbildung 5.12 ist der final umgesetzte Versuchsaufbau dargestellt. Den Abbildungen B.1 und B.2 im Anhang sind die Ansicht von oben mit den implementierten Mikrocontrollern sowie die Rückansicht zu entnehmen.

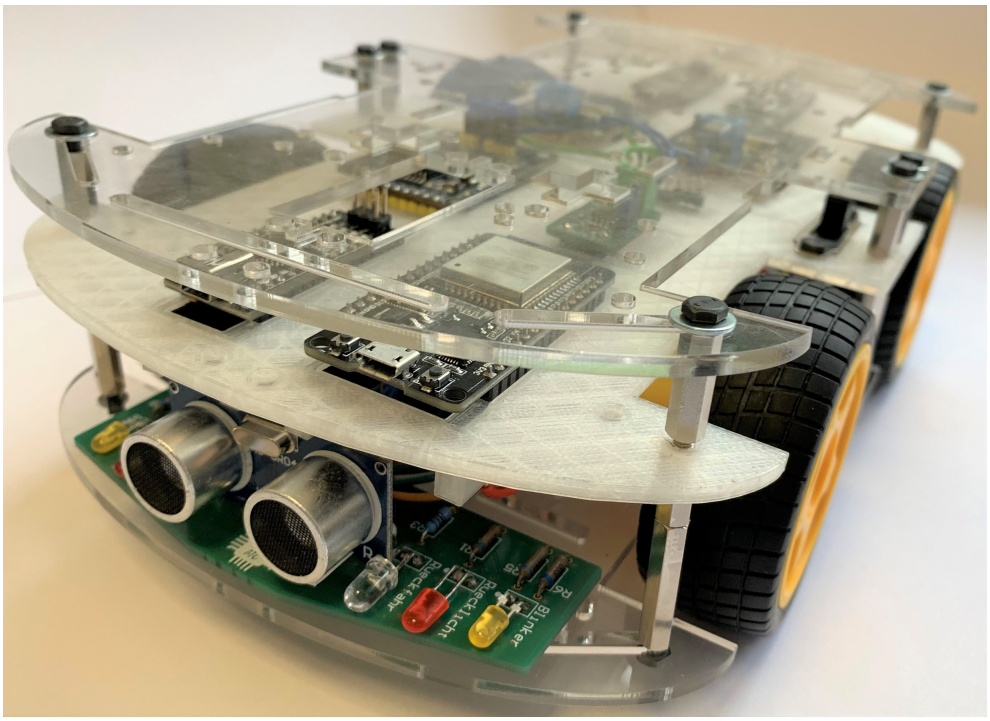


Abbildung 5.12: Finaler Aufbau des CAN-Demonstrators. Quelle: Eigene Darstellung

5.2 Softwareentwicklung

In diesem Kapitel wird die Umsetzung, der zuvor im Konzeptionsteil definierten Anforderungen an die jeweiligen Funktionalitäten der einzelnen Komponenten erläutert. Hierbei wird auf die Besonderheiten der Programmierung sowie der CAN-Kommunikation eingegangen. Als Code-Editor kommt im Folgenden Visual Studio Code Version 1.58.2 [MIC15] zusammen mit der PlatformIO IDE-Erweiterung [PLA14] zum Einsatz.

5.2.1 Definition von CAN-Botschaften

Da sich einige Implementierungen im CAN-Demonstrator grundlegend verändert haben, müssen die CAN-Botschaften unter neuer Betrachtung der Priorität dahingehend angepasst werden. Die CAN-Nachrichten benötigen einen Identifier und einen Namen. Des Weiteren kann die Anzahl der maximal zu übertragenden Bytes im Datenfeld festgelegt werden. Den CAN-Nachrichten können anschließend Signale mit Informationen über den Status einzelner Komponenten oder der derzeitigen Fahrtrichtung zugeteilt werden. Die Größe des Signals ist auf acht Bit beschränkt.

Die Priorisierung der CAN-Botschaften erfolgt anhand von deren Wichtigkeit für eine Gewährleistung der Fahrzeugsicherheit entsprechend dem realen Automobil. Ein weiteres Kriterium ist die Bedeutsamkeit der Nachricht im Demonstrator.
Von größter Bedeutung sind die Motorfunktionen, welche durch die Wahl eines niedrigen Identifiers einen bevorzugten Buszugriff erhalten. Da das Motorsteuergerät bei einer Nachricht der Distanzmessung unter 16 cm mit einem Stoppen der Motoren zur Sicherheit des Demonstrators reagiert, wurde diese Nachricht ebenfalls hoch priorisiert.
Im Straßenverkehr sind Bremslicht und Warnblinker in Gefahrensituationen zur eigenen Sicherheit essentiell und daher auch in diesem Versuchsaufbau mit niedriger ID versehen.
Tabelle 5.2 sind alle CAN-Botschaften des Demonstrators mit ihren zugehörigen Signalen zu entnehmen.

Tabelle 5.2: Übersicht der CAN-Botschaften des Versuchsaufbaus für den CAN-Demonstrator 2.

Ziel	**ID**	**Botschaft**	**Länge**	**Signale**	**Wertigkeit**
Motor	0x02	Stopp	0		
	0x03	Fahrtrichtung vorwärts	2	Geschwindigkeit	0-255
				Fahrtrichtung (links/rechts)	0-2
	0x04	Fahrtrichtung rückwärts	2	Geschwindigkeit	0-255
				Fahrtrichtung (links/rechts)	0-2
Beleuchtung	0x06	Bremslicht	1	Status	0-1
	0x07	Warnblinker	1	Status	0-1
	0x08	Blinker links	1	Status	0-1
	0x09	Blinker rechts	1	Status	0-1
	0x0A	Abblendlicht	1	Status	0-1
	0x0B	Fernlicht	1	Status	0-1
	0x0B	Rückfahrscheinwerfer	1	Status	0-1
	0x0C	Nebelschlussleuchte	1	Status	0-1
Elektrik	0x05	Distanzmessung	1	Entfernung	0-30
	0x0E	Hupe	1	Status	0-1
	0x0F	Akkustandsabfrage	1	Anforderung des Akkustandswertes	2-3
	0x10	Akkustand	2	Akkustandswert	0-510
Sonstiges	0x20	Änderung der Baudrate	2	Geschwindigkeit	100-1000

Tabelle 5.2: Übersicht der CAN-Botschaften des Versuchsaufbaus (Fortsetzung)

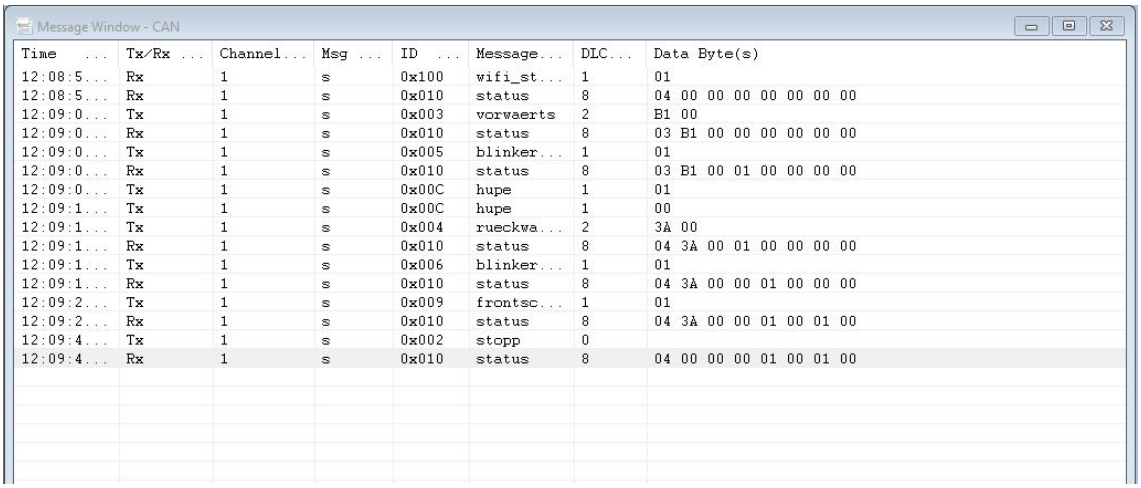
Ziel	ID	Botschaft	Länge	Signale	Wertigkeit
	0x30	Statusnachricht Motorsteuerung	3	Fahrtrichtung (vorwärts/rückwärts/gerade)	3, 4, 0
				Geschwindigkeit	0-255
				Richtung (links/rechts/gerade)	1, 2, 0
	0x100	Wi-Fi-Status	1	Status	0-1

5.2.2 Steuerung des CAN-Demonstrators

Die Steuerung des Demonstrators erfolgt durch die CAN-Analysesoftware BUSMASTER, welche CAN-Telegramme auf den Bus sendet. Der BUSMASTER bietet hierfür die Möglichkeit die vordefinierten Nachrichten aus der Datenbank einzeln manuell oder repetitiv zu versenden. Eine weitere Option ist die Nutzung des, für den BUSMASTER entwickelten, grafischen Benutzerinterfaces [LIP19]. Zur optimalen Nutzung sind Anpassungen an den neuen Versuchsaufbau nötig.

Manuelles Versenden von CAN-Botschaften

Nach der Erstellung einer Datenbank mit CAN-Botschaften im BUSMASTER, kann diese der Konfiguration hinzugefügt werden. Im *Transmit Window* können anschließend die zu übertragenden CAN-Nachrichten aus der Datenbank hinzugefügt und konfiguriert werden. In der *Data Byte View* lassen sich die einzelnen Nutzbytes für jede Nachricht editieren. Nachfolgend können die Botschaften manuell ausgesendet werden. Es besteht zudem die Möglichkeit eine Zeit einzustellen, nach der das jeweilige Frame zyklisch ausgesandt wird. Die Kommunikation zwischen Demonstrator und BUSMASTER kann im *Message Window* analysiert werden (siehe Abbildung 5.13).



Time ...	Tx/Rx ...	Channel...	Msg ...	ID ...	Message...	DLC...	Data Byte(s)
12:08:5...	Rx	1	s	0x100	wifi_st...	1	01
12:08:5...	Rx	1	s	0x010	status	8	04 00 00 00 00 00 00 00
12:09:0...	Tx	1	s	0x003	vorwaerts	2	B1 00
12:09:0...	Rx	1	s	0x010	status	8	03 B1 00 00 00 00 00 00
12:09:0...	Tx	1	s	0x005	blinker...	1	01
12:09:0...	Rx	1	s	0x010	status	8	03 B1 00 01 00 00 00 00
12:09:0...	Tx	1	s	0x00C	hupe	1	01
12:09:1...	Tx	1	s	0x00C	hupe	1	00
12:09:1...	Tx	1	s	0x004	rueckwa...	2	3À 00
12:09:1...	Rx	1	s	0x010	status	8	04 3À 00 01 00 00 00 00
12:09:1...	Tx	1	s	0x006	blinker...	1	01
12:09:1...	Rx	1	s	0x010	status	8	04 3À 00 00 01 00 00 00
12:09:2...	Tx	1	s	0x009	frontsc...	1	01
12:09:2...	Rx	1	s	0x010	status	8	04 3À 00 00 01 00 01 00
12:09:4...	Tx	1	s	0x002	stopp	0	
12:09:4...	Rx	1	s	0x010	status	8	04 00 00 00 01 00 01 00

Abbildung 5.13: Message Window der CAN-Analysesoftware BUSMASTER mit aufgezeichnetem CAN-Datenverkehr. Quelle: Eigene Darstellung

Anpassungen der GUI

Um die Steuerung des Demonstrators und die Analyse, der über den CAN-Bus laufenden Nachrichten durch das grafische Userinterface zu realisieren, sind einige Anpassungen an den neuen Versuchsaufbau vonnöten.
Durch das Hinzukommen neuer Beleuchtungsfunktionen und der Anpassung der Priorität der CAN-Botschaften, haben sich einige Identifier geändert. Diese wurden entsprechend im Code sowie der GUI angepasst. Die implementierten CAN-Nachrichten mit ihren IDs sind Tabelle 5.2: *CAN-Botschaften des Versuchsaufbaus* zu entnehmen.
Für eine optische Aufwertung getreu des Kombiinstrumentes im Automobil, wurde die GUI entsprechend um einen Hintergrund [CHR15] und Pedale zur Steuerung der Geschwindigkeit erweitert.
Zur Regulierung der Fahrtrichtung (links, rechts, gerade) wurde eine Trackbar innerhalb der Oberfläche implementiert.
Des Weiteren werden bei der Trennung der Verbindung zum BUSMASTER alle Funktionen wie die Geschwindigkeitsanzeige oder Beleuchtungselemente auf null gesetzt bzw. ausgeschaltet.
In Abbildung 5.14 ist das überarbeitete grafische Benutzerinterface zu sehen.

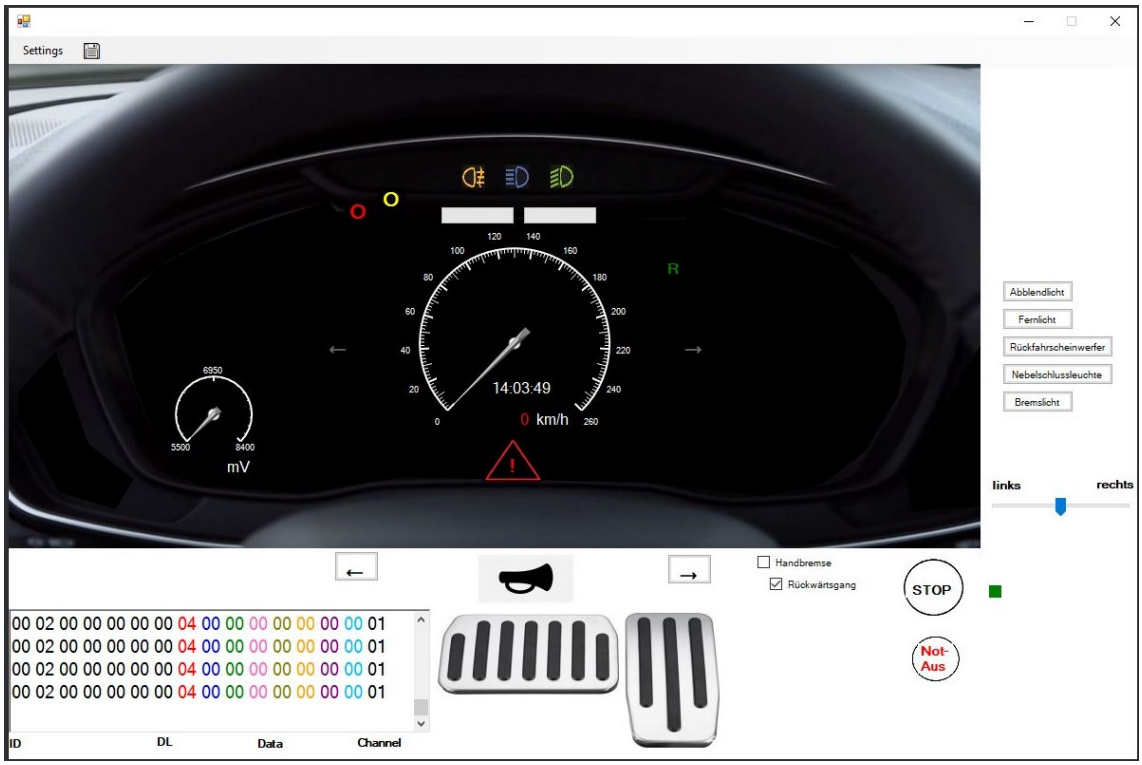


Abbildung 5.14: Bildschirmaufnahme der überarbeiteten GUI für die CAN-Analysesoftware BUSMASTER. Quelle: Eigene Darstellung

5.2.3 Wireless CAN

Für die implementierte Verbindung bedarf es eines TCP-Servers, der auf die Anfragen eines Clients antwortet. Für die Erstellung eines CAN-Frames werden folgende drei elementare Informationen benötigt:

- **Identifier**: Die ID kennzeichnet eindeutig eine Nachricht und bestimmt deren Priorität.
- **Datenlänge**: Das Feld DLC gibt Aufschluss über die Anzahl der zu übertragenden Nutzbytes.
- **Daten**: Es können maximal acht Byte Daten übertragen werden, wobei ein Nutzbyte maximal zwei Signale beinhalten kann.

Abbildung 5.15 zeigt den schematischen Ablauf der drahtlosen Kommunikation zwischen den beiden Gateways.

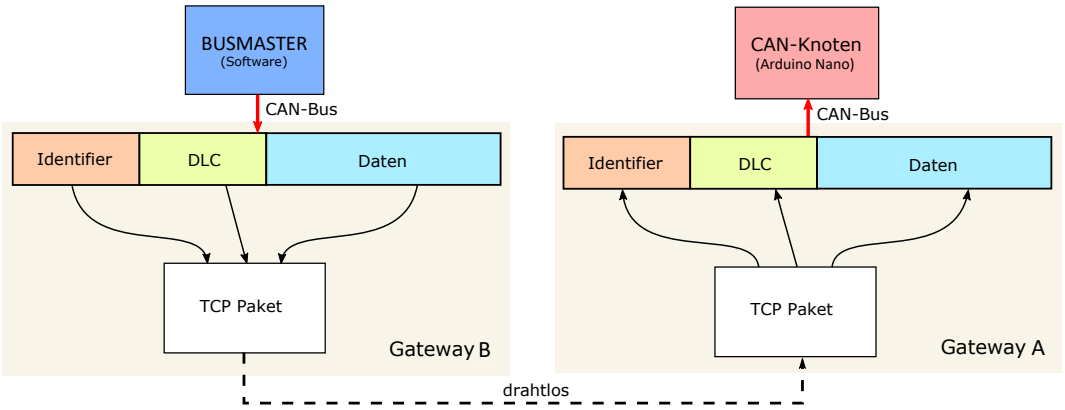


Abbildung 5.15: Schematische Darstellung der Kommunikation zwischen den Gateways des CAN-Demonstrators. Quelle: Modifiziert nach [HAN18]

Das Gateway A im Demonstrator dient dabei als Server der Verbindung und stellt das WLAN als sogenannter Access Point zur Verfügung. Das Gateway B bildet den Client, welcher sich mit dem WLAN verbinden und beim Server anmelden kann.
Wird eine CAN-Botschaft durch den BUSMASTER ausgesandt, wird diese durch das CAN-Interface an den CAN-Transceiver des Clients weitergeleitet. Dieser muss die ankommende Nachricht interpretieren und die oben genannten drei Bestandteile extrahieren. Anschließend werden ebendiese in ein TCP-Paket eingebettet und per WLAN übertragen. Hierfür reagiert der ESP32 auf die eingehende CAN-Botschaft und erzeugt einen Interrupt. Diese Interrupt Service Routine wertet das CAN-Frame aus und schreibt die Informationen in den Speicher eines globalen Objekts. Im Anschluss wird die Nachricht in einer Warteschlange abgelegt. Sobald sich eine Nachricht in der Warteschlange befindet, werden die Inhalte dieser extrahiert, in ein TCP-Paket eingefügt und versandt.
Das Gateway A im Demonstrator muss nun das TCP-Paket empfangen. Sobald der Server ein TCP-Paket erhält, wird dieses interpretiert und die darin enthaltenen Informationen in das Format eines CAN-Frames überführt und auf den CAN-Bus im Demonstrator weitergeleitet.
Analog kann das Gateway A CAN-Nachrichten von den Steuergeräten erhalten und diese als TCP-Paket an das Gateway B übertragen. Der Client stellt dann die Inhalte für den BUSMASTER zur Verfügung. Der Ablauf des Programms für das Gateway B (Client) ist Abbildung 5.16 zu entnehmen. Der Ablauf des Servers gestaltet sich analog, lediglich die CAN-Nachricht für den Verbindungsstatus wird nicht ausgesandt.

Um im Falle eines Verbindungsverlusts den Demonstrator anhalten zu können sendet das Server-Gateway eine CAN-Nachricht zum Stoppen der Motoren auf den Bus sobald die WiFi-Verbindung zum Client getrennt wurde.

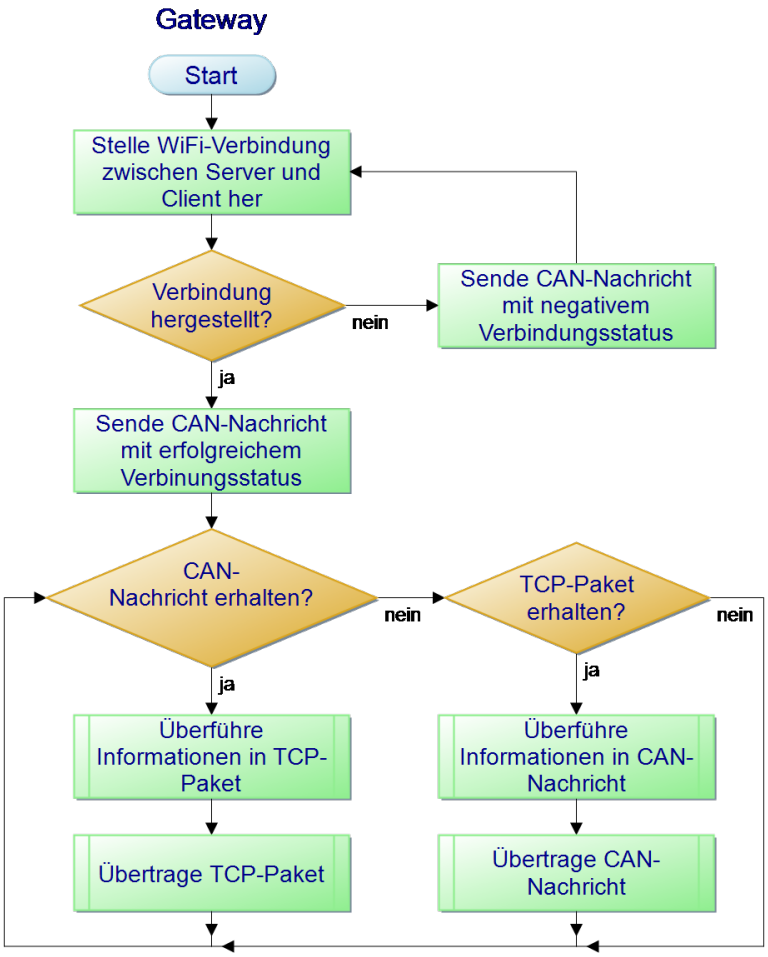


Abbildung 5.16: Programmablaufplan für den Quellcode des Gateways. Quelle: Eigene Darstellung

5.2.4 Verarbeitung der CAN-Botschaften durch die Steuergeräte

Die Steuerung des Demonstrators soll durch CAN-Botschaften erfolgen. Der Arduino Nano empfängt dazu als CAN-Knoten die Nachrichten auf dem Bus.
Zur Unterstützung des CAN-Protokolls sind die Arduino Nanos jeweils um ein CAN-Modul mit Transceiver und Controller erweitert worden. Trifft eine CAN-Botschaft beim CAN-Modul ein, werden die Signale auf dem Bus vom Transceiver in für den Controller lesbare Signale umgewandelt. Dieser bereitet die Daten auf und sendet sie über SPI an den integrierten Mikrocontroller ATMega328P-AU des Arduinos weiter. Anschließend muss der Mikrocontroller die Inhalte interpretieren und die gewünschten Aktionen ausführen. Wird eine CAN-Botschaft vom Steuergerät empfangen, wird wie beim Gateway ein Interrupt ausgelöst. Innerhalb der Interrupt Service Routine werden die Daten in einem globalen Objekt gespeichert und in einer Warteschlange abgelegt. Anschließend quittiert eine globale Variable die Auslösung des Interrupts. Im Hauptprogramm wird diese Variable geprüft und im Falle eines logischen TRUE werden die Inhalte der Nachricht ausgewertet.

Anhand des Identifiers der CAN-Botschaft kann das jeweilige Steuergerät die Relevanz der Nachricht für sich überprüfen und diese entsprechend verwerten oder verwerfen.
Die, für die Motorsteuerung bedeutsamen, CAN-Nachrichten besitzen die IDs 0x02 bis 0x05 (vgl. Tabelle 5.2: *CAN-Botschaften des Versuchsaufbaus*). Neben den Botschaften zur Steuerung der Motoren ist zusätzlich die Distanznachricht des Elektronik-Steuergerätes von Bedeutung. Die Motorsteuerung wertet den aktuellen Abstand aus und stoppt die Motoren, sobald dieser geringer als 16 cm ist.
Um den Demonstrator vor Tiefenentladung zu schützen, ist eine Akkustandsüberwachung implementiert. Das Elektroniksteuergerät sendet hierfür bei einem kritischen Akkustandswert von 6 V oder kleiner eine CAN-Nachricht auf den Bus. Die Motor-ECU wertet diese Nachrichten aus. Hat der Akkustand die Grenze erreicht, stoppt das Motorsteuergerät die Motoren. Abbildung 5.17 stellt den Ablauf für die Motoransteuerung dar.

Zur Änderung der Fahrtrichtung müssen die entsprechenden Pegel aus Tabelle 4.2 gesetzt werden. Die Regulierung der Geschwindigkeit der Motoren muss über eine Pulsweitenmodulation gelöst werden. Hierbei zählt ein im Mikrocontroller implementierter Timer bis zu einem bestimmten Wert. Ist dieser erreicht, wird der Pegel am Pin auf Low gesetzt und der Timer zählt weiter bis ein Variablenüberlauf stattfindet. Der Pegel am Pin wird nun auf High gesetzt und der Timer beginnt von vorn zu zählen.
Wie der Tabelle 4.2 zu entnehmen ist, stoppen die Motoren, wenn IN1 und IN2 des Motortreibers TB6612 dasselbe Spannungspotential haben. Unterschiedliche Spannungspotentiale führen zu einer Motorbewegung. Über den PWM-Pin kann die Geschwindigkeit reguliert werden.

Der Demonstrator soll auf jede Änderung mit einer Statusnachricht antworten. Die Implementierung einer einzigen Statusnachricht, wie in Version 1, kann durch den neuen Aufbau mit drei Steuergeräten nicht sinnvoll umgesetzt werden. Da eine Rückantwort des CAN-Demonstrators jedoch für eine forensische Analyse essentiell ist, wurde sich für eine Statusnachricht pro Steuergerät entschieden.
Die Statusnachricht des Motorsteuergeräts besitzt die ID 0x30, bei einer Datenlänge von drei Byte. Der Inhalt ist Tabelle 5.3 zu entnehmen.

Tabelle 5.3: Statusnachricht 0x30 des Motorsteuergeräts

Länge	**Signal**	**Wertigkeit**
1	Richtung (vorwärts/rückwärts/stehend)	3, 4, 0
1	Geschwindigkeit	0-255
1	Richtung (links/rechts/gerade)	1, 2, 0

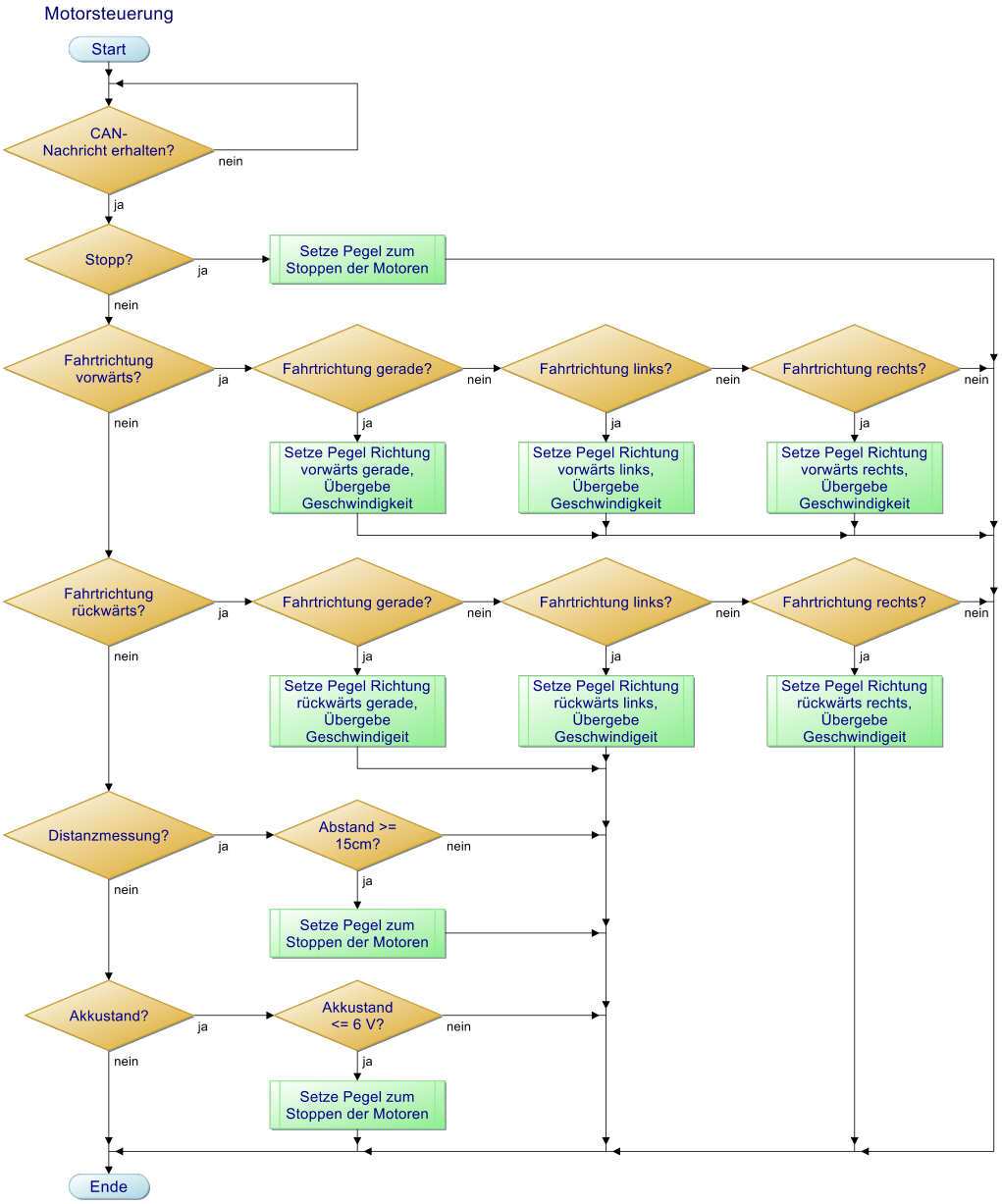


Abbildung 5.17: Programmablaufplan für den Quellcode des Motorsteuergeräts. Quelle: Eigene Darstellung

5.2.5 Projektaufbau

Die Modularisierung des Programmcodes bietet die Möglichkeit das System sehr individuell zu konfigurieren und auf unterschiedliche Mikrocontroller anwenden zu können. Die unten stehende Abbildung 5.18 gibt einen Überblick über den Projektaufbau mit seinen Programmkomponenten.

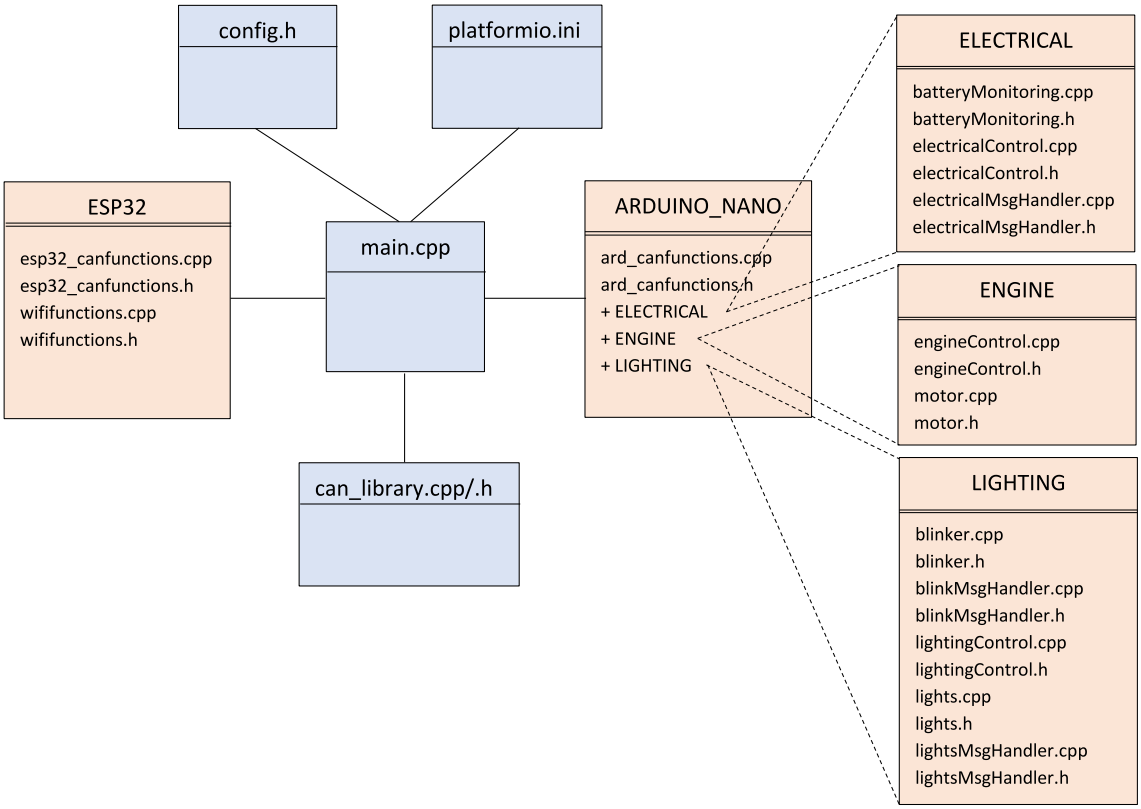


Abbildung 5.18: Programmkomponenten des Projektaufbaus. Quelle: Eigene Darstellung

Zentrales Bindeglied des Programmcodes ist die `main`-Datei. Innerhalb der Hauptschleife werden die Empfangsfunktionen für CAN und TCP aufgerufen und die Akkustandsüberwachung reguliert.
Die Projektkonfigurationsdatei `platformio.ini` enthält Einstellungen zur verwendeten Hardware. Hier sind die jeweiligen Umgebungen für die einzelnen ECUs und Gateways definiert.
Unter `config.h` finden sich zentrale Einstellungen, wie die Baudrate des CAN-Busses, die Zugangsdaten für WiFi, controllerspezifische Bibliotheken und die implementierten CAN-Botschaften.
Die `can_library`-Datei enthält die Konvertierungsfunktionen für das CAN-Frame sowie die Empfangs- und Aussendefunktionen.

Die blau hinterlegten Dateien sind jene, die für alle Controller benötigt werden.
Bei den orange gefärbten Elementen handelt es sich um controllerspezifische Ordner, welche die Dateien für diese beinhalten. Der Ordner des Arduinos enthält zudem drei weitere Unterordner für die drei Steuergeräte mit deren funktionsspezifischen Code.

5.2.6 Nutzung multipler Boards

Um sowohl die Gateways als auch alle Steuergeräte mit nur einem Programmcode steuern zu können, bedarf es einiger Einstellungen in der Projektkonfigurationsdatei `Platformio.ini`. Zunächst müssen die Boards hinzugefügt werden. Hierfür sind die in [ELI20] beschriebenen folgenden Konfigurationen von Relevanz:

- Umgebung (`env`)
- Plattform (`platform`)
- Board (`board`)
- Framework (`framework`)

Um beim Kompilierungsprozess sowie dem Flashen der Hardware einzelne Boards zu ignorieren bzw. nur das derzeit verwendete anzusteuern, kann die Arbeitsumgebung in der Symbolleiste am unteren Rand über *Switch PlatformIO Project Environment* entsprechend ausgewählt werden. Die jeweiligen externen Bibliotheken für die verschiedenen Umgebungen können mittels `lip_deps` verwaltet werden. [PLA21]

Um innerhalb der Lehre mehrere Demonstratoren gleichzeitig steuern zu können, werden für jedes Client-Server-Paar eigene Netzzugangsdaten benötigt. In der Projektkonfigurationsdatei `platformio.ini` wurde hierfür ein Flag bei den Umgebungen der Gateways (`DEMONSTRATOR_X`) angelegt. Das `X` ist entsprechend durch die Zahl des Demonstrators zu ersetzen.

6 Systemtest und Inbetriebnahme

Innerhalb dieses Kapitels wird näher auf die Inbetriebnahme des CAN-Demonstrators sowie die Durchführung der Systemtests eingegangen. Dem zu Grunde liegen vorherige Softwaretests, welche gemäß des V-Modells bereits während der Entwicklungsarbeit durchgeführt wurden.
Im Rahmen des Komponententests wurden alle einzelnen Funktionen abgeschirmt auf ihre Funktionalität geprüft. Es folgte das Testen der Zusammenarbeit voneinander abhängiger Komponenten während des Integrationstest. Die letzte Stufe dieser Arbeit bildet der Systemtest. Im Folgenden wird näher auf die Untersuchung gegen die gestellten Anforderungen des fertigstellten und zusammengebauten Demonstrators als Gesamtsystem eingegangen. Hierfür wurden vier Testszenarien mittels Testdaten durchgeführt.

6.1 Voraussetzungen

Zur erfolgreichen Inbetriebnahme des CAN-Demonstrators wird die Software BUSMASTER Version 3.2.2 mit dem zugehörigen Compiler MinGW, sowie der, für den letzten Test notwendigen Desktop-Applikation des Benutzerinterfaces, benötigt [ETA17]. Zur Verbindung zwischen dem Client-Gateway und dem Computer wird das CAN-Interface VN1610 von Vector verwendet.

6.2 Vorbereitung

Zur Vorbereitung der Testdurchführungen wurden die CAN-Botschaften mit ihren Signalen aus Tabelle 5.2 in eine Datenbank im `.dbf`-Format implementiert und der CAN-Analysesoftware BUSMASTER hinzugefügt.

6.3 Test 1: Steuerung der Motoren

Die grundlegende Funktion eines Automobils stellt das Fahren an sich dar. Der Demonstrator verfügt über die notwendigen Komponenten zur Simulation dieser.
Ziel dieses Systemtests ist es, die Motoren des Demonstrators anzusteuern und eine Drehbewegung vorwärts und rückwärts, je geradeaus, links und rechts zu bewirken. Darüber hinaus soll die Geschwindigkeit reguliert werden. Zuletzt soll ein „Hard-Stop“ des Demonstrators erreicht werden.

6.3.1 Durchführung

Für diesen Test wurden folgende, in Tabelle 6.1 dargestellte, CAN-Botschaften auf den Bus des Demonstrators gesandt.

Tabelle 6.1: Übersicht der CAN-Nachrichten des Versuchsaufbaus für Test 1: Steuerung der Motoren

ID	Daten	Signal
0x03	00 2F	vorwärts, gerade
0x03	00 5F	vorwärts, gerade
0x03	01 2F	vorwärts, links
0x03	01 5F	vorwärts, links
0x03	02 2F	vorwärts, rechts
0x03	02 5F	vorwärts, rechts
0x04	00 2F	rückwärts, gerade
0x04	00 5F	rückwärts, gerade
0x04	01 2F	rückwärts, links
0x04	01 5F	rückwärts, links
0x04	02 2F	rückwärts, rechts
0x04	02 5F	rückwärts, rechts
0x02		Stopp

Die Geschwindigkeit der Motoren wird mittels Pulsweitenmodulation reguliert. Die Signalpegel für die beiden Testversuche vorwärts mit je 2F und 5F lassen sich den Oszillogrammen in Abbildung 6.1 und 6.2 entnehmen.

6.3.2 Ergebnis

Die Motoren des Robot Car Kits können erfolgreich durch den Motortreiber TB6612 und den CAN-Knoten Arduino Nano mittels CAN-Botschaften gesteuert werden. Der Demonstrator lässt sich sowohl in die Richtungen vorwärts und rückwärts als auch links und rechts lenken. Die erfolgreiche Regulierung der Geschwindigkeit mittels PWM kann zusätzlich durch die Messung am Oszilloskop belegt werden.

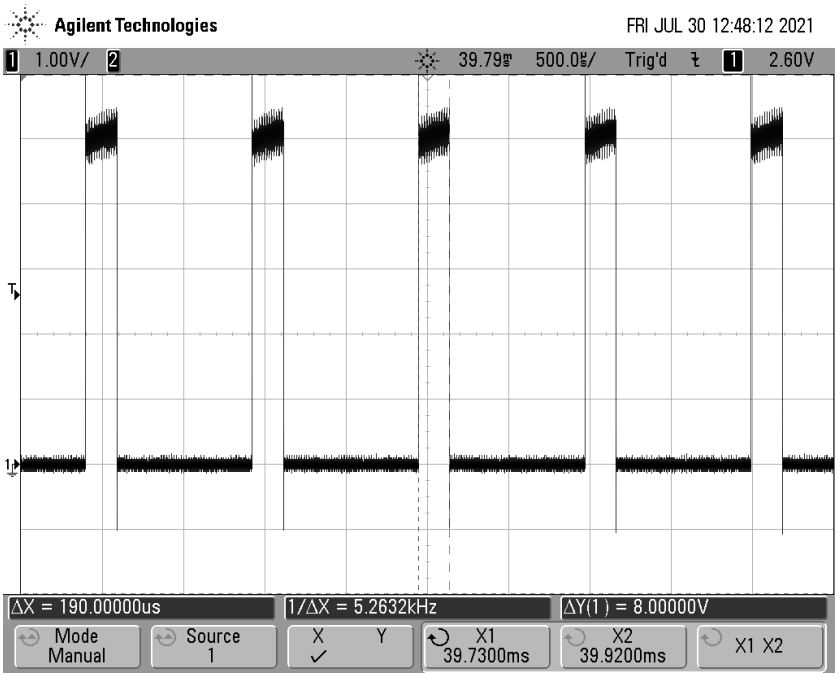


Abbildung 6.1: Oszillogramm der Pulsweitenmodulation der Gleichstrommotoren in Vorwärtsrichtung bei Ansteuerung mit Wert 2F. Quelle: Eigene Darstellung

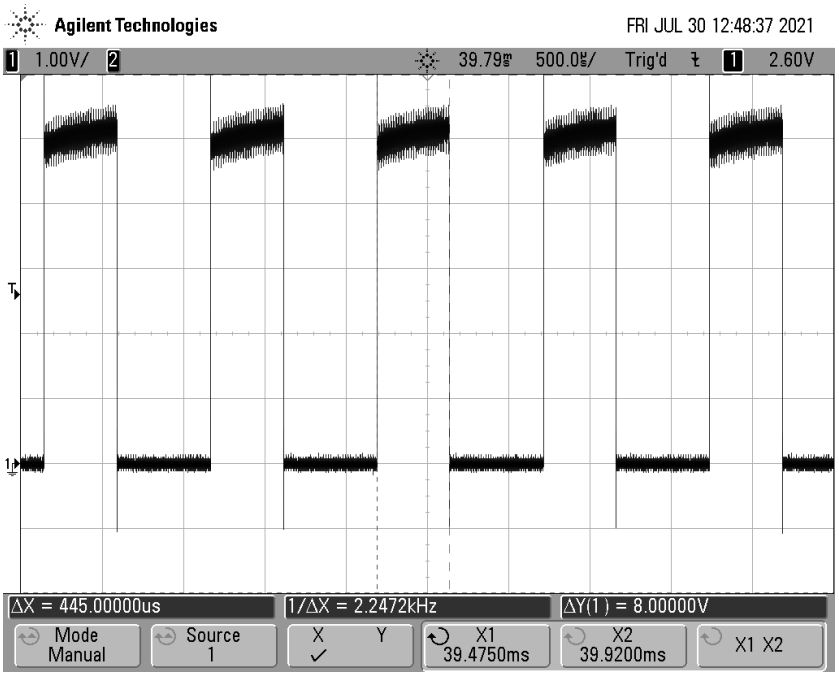


Abbildung 6.2: Oszillogramm der Pulsweitenmodulation der Gleichstrommotoren in Vorwärtsrichtung bei Ansteuerung mit Wert 5F. Quelle: Eigene Darstellung

6.4 Test 2: Reaktion auf die Distanznachricht

Zum Schutz des Demonstrators soll dieser bei einem Abstand von 15 cm zu einem Objekt anhalten. Hierfür wertet das Motorsteuergerät die Distanznachrichten (ID 0x05) der Elektronik-ECU aus und stoppt die Motorbewegung sobald der Wert kleiner oder gleich 15 beträgt.

6.4.1 Durchführung

Für diesen Test wurde der CAN-Demonstrator mit den in Tabelle 6.2 gelisteten CAN-Nachrichten bewegt und sein Bremsverhalten beobachtet. Der Demonstrator wurde in einer Entfernung von zwei Metern gegenüber einer Wand positioniert und gestartet. In jeweils drei Durchgängen pro Zustand wurde die Distanz zwischen Objekt und gestoppten Demonstrator gemessen.

Tabelle 6.2: Übersicht der CAN-Nachrichten des Versuchsaufbaus für Test 2: Reaktion auf die Distanznachricht

ID	Daten	Signal
0x03	00 2F	vorwärts, gerade
0x03	00 5F	vorwärts, gerade
0x03	00 FF	vorwärts, gerade
0x04	00 2F	rückwärts, gerade
0x04	00 5F	rückwärts, gerade
0x04	00 FF	rückwärts, gerade

6.4.2 Ergebnis

Der CAN-Demonstrator stoppt erfolgreich selbstständig vor einem Objekt. Das Motorsteuergerät ist in der Lage die Distanznachrichten auszuwerten und bei geringer Distanz zu einem Gegenstand eigenständig die Motoren zu stoppen. Die Übersicht der einzelnen Ergebnisse pro Testdurchlauf sind Tabelle 6.3 zu entnehmen.
Die Tabellenauswertung zeigt einen klaren Trend. Je höher die Geschwindigkeit, desto kürzer ist schlussendlich die Distanz zwischen stehendem Demonstrator und dem Objekt. Der CAN-Demonstrator verhält sich hier äquivalent zum realen Automobil, welches mit steigender Geschwindigkeit einen längeren Bremsweg benötigt.
Insgesamt lässt sich kein erkennbarer Unterschied zwischen der Fahrtrichtung vorwärts und rückwärts erkennen.
Es konnte somit ein wirksamer Schutz vor Zusammenstößen des Demonstrators erreicht werden.

Tabelle 6.3: Messergebnisse Test 2: Reaktion auf die Distanznachricht

ID	Daten	Distanz in cm	Durchschnitt in cm
0x03	00 2F	14	14,17
0x03	00 2F	15	
0x03	00 2F	13,5	
0x03	00 5F	11,5	11,5
0x03	00 5F	10	
0x03	00 5F	13	
0x03	00 FF	3	4,83
0x03	00 FF	2,9	
0x03	00 FF	9	
0x04	00 2F	13	13,5
0x04	00 2F	13,5	
0x04	00 2F	14	
0x04	00 5F	10,5	12,17
0x04	00 5F	13	
0x04	00 5F	13	
0x04	00 FF	6	3,5
0x04	00 FF	1,5	
0x04	00 FF	3	

6.5 Test 3: Reaktion auf die Akkustandsnachricht

Zum Schutz des Demonstrators vor Tiefenentladung wurde eine Akkustandsüberwachung im Demonstrator implementiert.
Das Motorsteuergerät wertet die CAN-Nachrichten zum Akkustand auf dem Bus aus und stoppt die Motoren, wenn der Schwellwert von 6 V erreicht wird.
Im Zuge dieses Tests soll zudem die Akkulaufzeit im Vollbetrieb bestimmt werden.

6.5.1 Durchführung

Für diesen Test wurde der Akkumulator des Demonstrators vollständig bis 8,4 V geladen. Anschließend wurde der Demonstrator mit der Höchstgeschwindigkeit und allen Beleuchtungselementen betrieben. Parallel wurde der Akkustand (ID 0x10) kontinuierlich im BUSMASTER durch Aussenden der CAN-Botschaft zur Akkustandsabfrage (ID 0x0F) überwacht. Die minimale Grenze liegt bei 6 V. Wird diese erreicht, ohne dass der Demonstrator stoppt, wird der Test an dieser Stelle beendet.

6.5.2 Ergebnis

Der Demonstrator hat bei einem Akkustandswert von 6 V selbstständig die Motoren gestoppt. Das Motorsteuergerät wertet folglich die Akkustandsnachricht der Elektronik-ECU korrekt aus und reagiert im Falle der Überschreitung des Grenzwertes.
Der unter Vollleistung (Motoren mit Höchstwert, alle Beleuchtungselemente an) betriebene Demonstrator benötigte etwa 4:15 h um einen Akkustandswert von 6 V zu erreichen. Das sind anderthalb Stunden länger als sich aus der prognostizierten Berechnung in Kapitel 4.3.4 ergibt. Damit kann der CAN-Demonstrator mehr als zwei Lehreinheiten à 90 Minuten ohne zu Laden betrieben werden.

6.6 Test 4: WiFi-Verbindungsverlust

Um im Falle eines Verlustes der WiFi-Verbindung den Demonstrator dennoch anhalten zu können, soll dieser selbst die Verbindung überwachen und im Falle einer Trennung die Motoren stoppen.
Im Code des Gateways werden je nach Zustand verschiedene Events ausgelöst. Tritt das Event der Verbindungstrennung ein, wird vom Gateway eine Botschaft zum Stoppen der Motoren auf den Bus gesandt. Im Folgenden soll die Zuverlässigkeit anhand verschiedener Szenarien getestet werden.

6.6.1 Durchführung

Für diesen Test wurden Client und Server per WLAN miteinander verbunden und letzterer über die serielle Verbindung überwacht. Es wurden folgende drei Szenarien der Verbindungstrennung durchgeführt:

- Restart am Gateway-Client per Enable (EN)-Button
- Trennung der Stromverbindung am Client
- Trennung der CAN-Verbindung am Client

6.6.2 Ergebnis

Der Server erkennt die Verbindungstrennung nicht zuverlässig. Er reagiert nur auf die Verbindungstrennung via Restart des Clients per EN-Button. Wird die WiFi-Verbindung durch das Trennen der Strom- oder CAN-Verbindung am Client unterbrochen, wird das entsprechende Event nicht ausgelöst und der Demonstrator wird nicht über den Verbindungsverlust informiert.

6.6.3 Optimierung

Zur Lösung des Problems, dass der Server die Verbindungstrennung nicht zuverlässig erkennt, kann eine eigene Überwachung in Form eines Watchdogs, welcher kontinuierlich die WiFi-Verbindung prüft, implementiert werden.
Um den Abbruch der Verbindung in einem angemessenen Zeitraum festzustellen, wird je eine Zählvariable in den Programmablauf beider Module integriert. Der Client sendet zyklisch sogenannte Watchdog-Nachrichten mit der ID 0x200 auf den CAN-Bus. Parallel überprüft der Server, ob er im vorgegebenen Zeitintervall diese Nachricht erhalten hat. Ist dem nicht so, sendet der Server eine CAN-Botschaft zum Stoppen der Motoren aus.
Da der Client den Verbindungsabbruch selbstständig detektiert, ist die Implementierung auf dieser Seite nicht vonnöten.

6.7 Test 5: Steuerung des Demonstrators durch die GUI

Zur Steuerung der zweiten Version des CAN-Demonstrators wurde die GUI an die überarbeiteten bzw. neuen Funktionen angepasst.
Das Ziel ist alle Funktionen des Demonstrators durch das grafische Userinterface steuern zu können. Hierfür müssen nach erfolgreicher Verbindung mit dem Busmaster alle Elemente der GUI betätigt werden. Parallel muss der Demonstrator sowie die GUI auf eine richtige Ausführung bzw. Darstellung beobachtet werden.

6.7.1 Durchführung

Für diesen Test wurden alle im Folgenden gelisteten Komponenten auf ihre richtige Funktionsweise, d.h. eine korrekte Ausführung im Demonstrator und entsprechend richtige Anzeige geprüft.

- Label für den Verbindungsstatus
- Uhrzeit im Kombi-Instrument
- Geschwindigkeitssteuerung der Motoren durch Pedale
- Richtungssteuerung durch Checkbox und Trackbar
- Not-Aus
- Handbremse
- Hupe
- Akkustandsmessung

- Nebelschlussleuchte
- Abblendlicht
- Rückfahrscheinwerfer
- Blinker links/rechts
- Warnblinker
- Bremslicht
- Fernlicht

6.7.2 Ergebnis

Es können alle Funktionen des CAN-Demonstrators durch die grafische Benutzeroberfläche angesteuert und korrekt dargestellt werden.
Jedoch kann innerhalb der GUI nicht zwischen einer gesendeten oder empfangenen CAN-Nachricht auf dem Bus unterschieden werden. Wird beispielsweise eine Botschaft zur Steuerung der Motoren ausgesandt, wird die Rückantwort des Demonstrators in Form einer Statusnachricht ausgewertet und innerhalb der grafischen Oberfläche visualisiert. Werden Funktionen der Steuergeräte Beleuchtung und Elektronik angesteuert, so kommt als Rückantwort die gleiche Nachricht zurück. Im Falle, dass der Demonstrator nicht reagiert und die gewünschte Funktion aktiviert, wird die Aktivierung jedoch trotzdem in der GUI dargestellt.

6.7.3 Optimierung

Zur Lösung des oben genannten Problems, müssen für die Elektronik- und Beleuchtungs-ECUs ebenfalls Statusnachrichten im Code implementiert werden. Diese Rückantwort könnte anschließend innerhalb der Oberfläche ausgewertet und entsprechend visualisiert werden. Die genannten Steuergeräte befinden sich im Rahmen von [KRA21] noch in der Entwicklung. Die GUI muss zu einem späteren Zeitpunkt an die entsprechenden Daten angepasst werden.

7 Zusammenfassung

Ziel der vorliegenden Masterarbeit war die Entwicklung des Antriebsstranges, welcher als eigenständige CAN-Bus-Steuereinheit in einem forensischen CAN-Demonstrator agieren soll. Die Basis dieser Arbeit sollte dabei die Analyse des, 2018 im Application Center Microcontroller entwickelten, forensischen CAN-Demonstrators sein, aus welchem das Redesign hervorgehen soll.

Im Rahmen dieser Arbeit konnte ein funktionsfähiger Demonstrator für den CAN-Bus entwickelt werden, welcher die grundlegenden Fahrfunktionen eines Automobils simuliert. Durch das Aussenden von CAN-Botschaften durch die CAN-Analysesoftware BUSMASTER auf den Bus kann dieser gesteuert werden. Als Motorsteuergerät dient ein Arduino Nano, welcher als CAN-Knoten die Nachrichten auf dem Bus empfängt und auswertet. Diese ECU ist in der Lage selbstständig auf CAN-Botschaften anderer Steuergeräte zu reagieren und die Motorbewegung entsprechend zu beeinflussen. Die Komponente der Eigenständigkeit konnte somit erfüllt werden.
Zur Gewährleistung der Mobilität während des Einsatzes in der Lehre wurde der Demonstrator mit einem Akkumulator ausgestattet, der eine Laufzeit von mehr als den geforderten 90 Minuten ermöglicht. Zur drahtlosen Kommunikation zwischen der CAN-Analysesoftware und dem Demonstrator dienen zwei NodeMCU-ESP32 als Gateways, eines im Demonstrator und eines am Computer, welche die CAN-Nachrichten via WiFi übertragen. Der CAN-Bus wird somit streckenweise, aber nicht vollständig ersetzt.
Zur anschaulichen Darstellung in der Lehre wurde die, für die CAN-Analysesoftware BUSMASTER, entworfene GUI entsprechend an die entwickelten Funktionen des Demonstrators angepasst. Hierfür wurde eine Statusnachricht als Rückantwort auf jede Zustandsänderung für das Motorsteuergerät implementiert. Diese enthält die Informationen über den aktuellen Zustand der Motoren, welche für die Visualisierung vonnöten sind.
Im Rahmen von abschließenden Systemtests gegen die gestellten Anforderungen, wurde ein Problem bei der Überprüfung der WiFi-Verbindung festgestellt und hierfür ein Watchdog im Gateway implementiert, der diese überwacht.
Folglich wurden alle gestellten Anforderungen, die sich aus der Zielstellung ergaben, erfüllt.

8 Ausblick

Das erarbeitete Konzept stellt eine gute Basis für zukünftige Erweiterungen und Forschungen dar.

Im Hinblick auf die Weiterentwicklung wäre ein vollständiger Verzicht auf fertige Controllerlösungen wie dem Arduino Nano oder der NodeMCU-ESP32 denkbar. Stattdessen könnte eine eigene Leiterplatte mit den benötigten Komponenten entworfen werden. Diese könnte, falls platzmäßig möglich, direkt auf der Bodenplatte angebracht werden, sodass die Verkabelungen im CAN-Demonstrator nahezu vollständig entfallen. Eine andere Variante wäre, die im Rahmen dieser Entwicklung implementierte Zwischenplatte, welche als Halterung der Entwicklungsplatinenen fungiert, durch die eigene Leiterplatte zu ersetzen.

Des Weiteren wäre es denkbar eine Applikation zur Steuerung des CAN-Demonstrators für das Smartphone zu entwickeln. Durch den Touchscreen wäre eine schnellere und intuitive Bedienung des grafischen Userinterfaces möglich und die Abhängigkeit von einem Computer nicht mehr gegeben. Der Aspekt der Mobilität des Demonstrators könnte durch die Steuerung via Smartphone deutlich erweitert werden.
Zur Realisierung müsste sich das Smartphone als Client mit dem Server im Demonstrator verbinden. Die CAN-Nachrichten der App könnten anschließend als TCP-Pakete an das Gateway des Demonstrators übermittelt werden. Innerhalb der Applikation müsste das CAN-Frame in ein TCP-Paket verpackt und ausgesandt werden. Umgekehrt muss das empfangene TCP-Paket in der Applikation extrahiert und die Daten entsprechend zur Analyse aufbereitet werden.

Der Aufbau des Demonstrators ist so gestaltet, dass dieser beliebig um zusätzliche Steuergeräte erweitert werden kann. Denkbar wären neben der Einbindung von Steuergeräten aus dem realen Automobil auch die Implementierung weiterer Bussysteme, um die Funktion des Gateways auszubauen. Eine Möglichkeit stellt der LIN-Bus dar, welcher auch durch die Analysesoftware BUSMASTER unterstützt wird.
Dieser wird in neuen Entwicklungskonzepten bereits für intelligente Scheinwerferlösungen eingesetzt.

Ein anderer vielversprechender Ansatz wäre die Implementierung von weiteren Sensoren am CAN-Demonstrator, um ein automatisiertes oder gar autonomes Fahren entwickeln zu können.

Anhang A: Schaltplan der Platinen

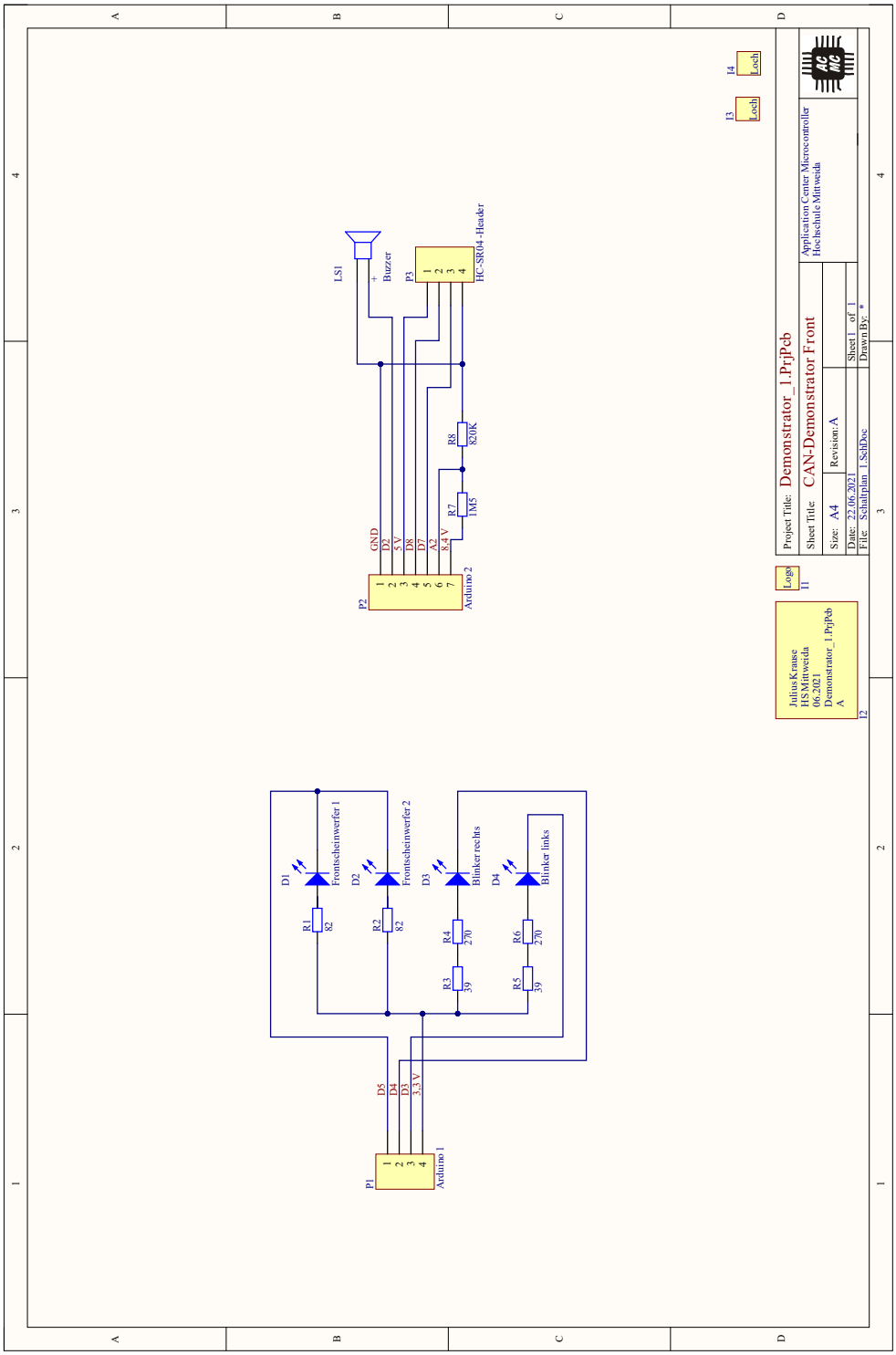


Abbildung A.1: Schaltplan der Bodenplatine vorne. Quelle: [KRA21]

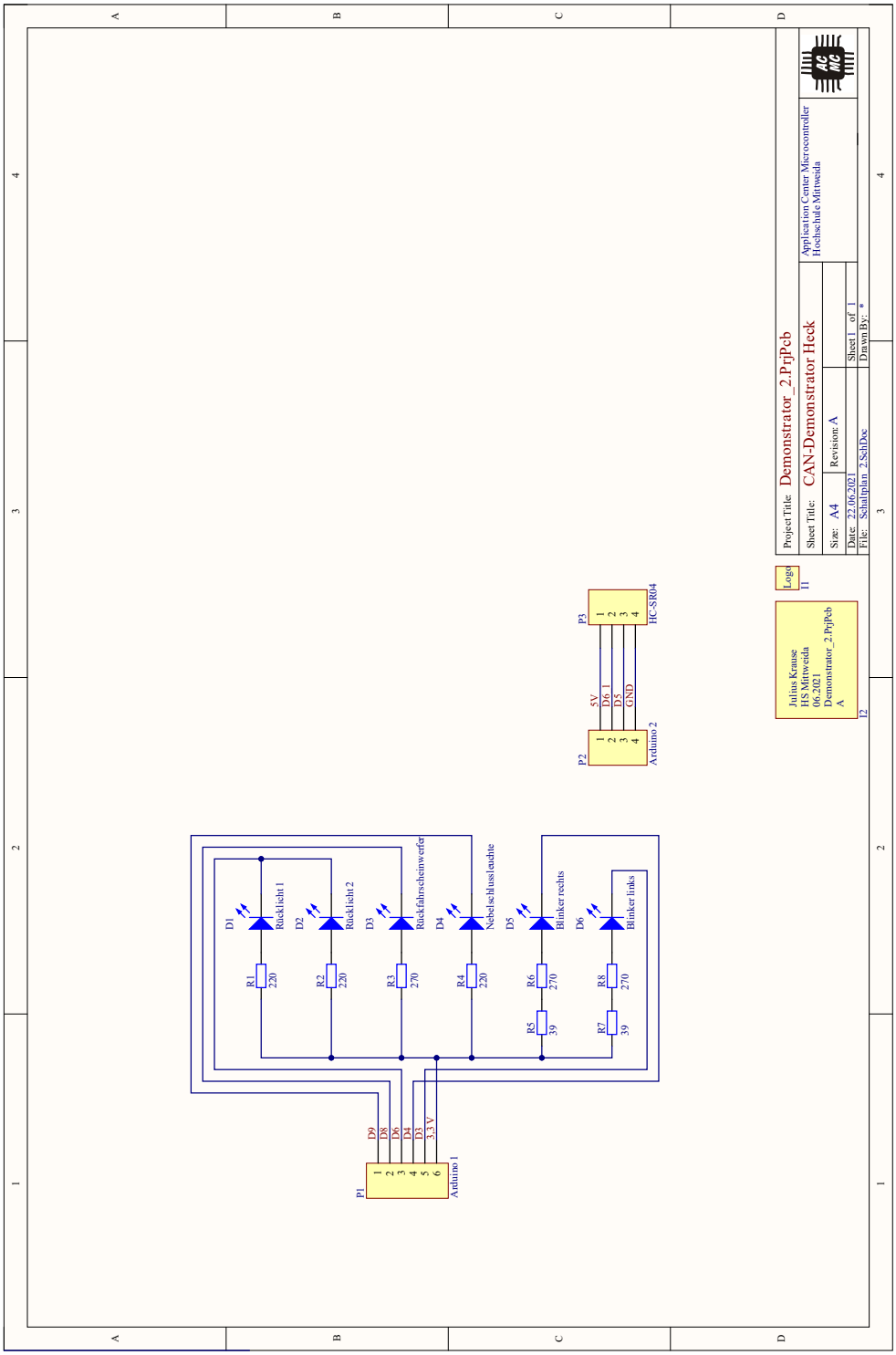


Abbildung A.2: Schaltplan der Bodenplatine hinten. Quelle: [KRA21]

Anhang B: Finaler Aufbau des CAN-Demonstrators

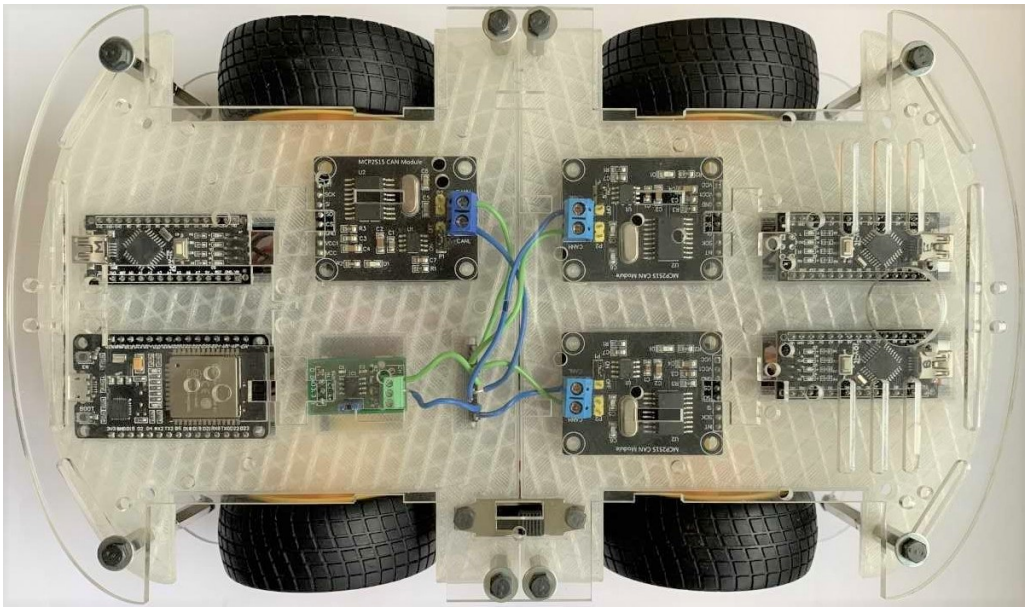


Abbildung B.1: Finaler Aufbau des CAN-Demonstrators - Aufnahme von oben. Quelle: Eigene Darstellung

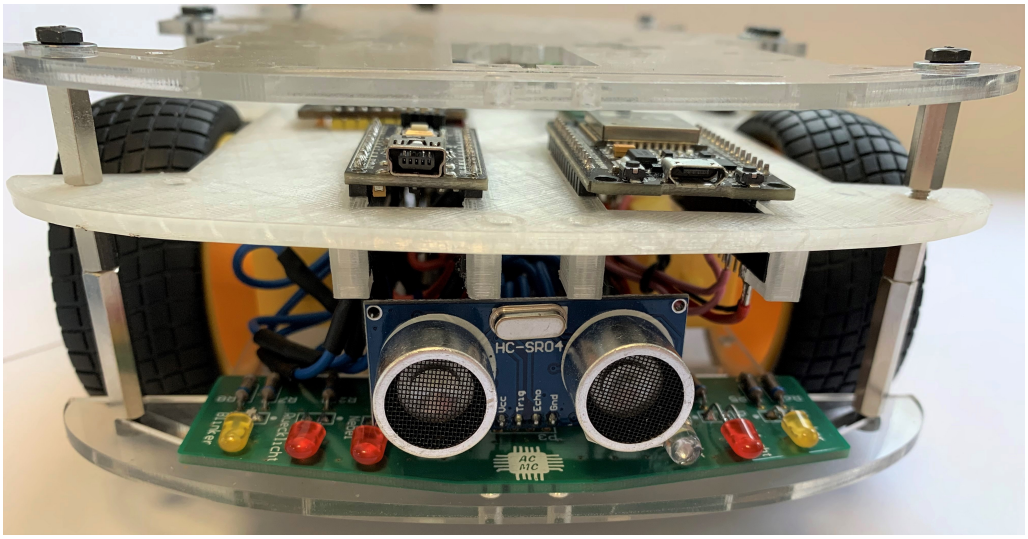


Abbildung B.2: Finaler Aufbau des CAN-Demonstrators - Aufnahme von hinten. Quelle: Eigene Darstellung

Literaturverzeichnis

[ARD33] ARDUINO: *Arduino Nano 33 IoT*. https://cdn-reichelt.de/documents/datenblatt/A300/ABX00027_ENG_TDS.pdf. Zuletzt geprüft: 06.07.2019

[ARD19] WHADDA: *ARDUINO NANO 33 IOT WITH HEADERS*. https://i1.wp.com/cdn.whadda.com/wp-content/uploads/2019/10/27104708/ard-abx00032-2.jpg?fit=600%2C450&ssl=1. Zuletzt geprüft: 06.07.2021

[ARD20] JOY-IT: *ARDUINO NANO V3: Arduino kompatibler Microcontroller mit original Chip*. https://joy-it.net/files/files/Produkte/ARD_NanoV3/ARD-NanoV3-Datenblatt-20200326.pdf. Zuletzt geprüft: 05.07.2021

[AVR10] OLIMEX LTD: *AVR-CAN development board: Users Manual*. https://www.olimex.com/Products/AVR/Development/AVR-CAN/resources/AVR-CAN.pdf. Zuletzt geprüft: 05.07.2021

[BPK20] BRANDT-POOK, Hans; KOLLMEIER, Rainer: *Softwareentwicklung kompakt und verständlich: Wie Softwaresysteme entstehen*. 3., verbesserte Auflage. Springer Vieweg, 2020. – ISBN 978–3–658–30630–4

[ETA17] ETAS GMBH: *BUSMASTER*. Version: v3.2.2. URL https://rbei-etas.github.io/busmaster/. Zuletzt geprüft: 06.07.2021

[BUZ21] DISTRELEC GROUP AG: *RND Components Electromagnetic Buzzer*. https://www.distrelec.de/Web/Downloads/_t/ds/RND%20430-00009_eng_tds.pdf. Zuletzt geprüft: 05.07.2021

[CAN18] JOY-IT: *CAN MODUL: mit MCP2515 CAN Interface & MCP 2562 Transceiver*. https://joy-it.net/files/files/Produkte/SBC-CAN01/SBC-CAN01-Datenblatt.pdf. newblock Zuletzt geprüft: 06.07.2021

[CHR15] CHRIS CHIN; EGMCARTECH: *Audi-A4-Virtual-Cockpit-1024x513*. https://www.egmcartech.com/wp-content/uploads/2015/08/2016-Audi-A4-Virtual-Cockpit-1024x513.jpg. Version: 20.08.2015 Zuletzt geprüft: 21.07.2021

[CIA94] CAN IN AUTOMATION: *CiA Draft Standard 102 Version 2.0: CAN Physical Layer for Industrial Applications*. https://www-csr.bessy.de/control/Hard/fieldbus/CAN/CiA/DS102.pdf. Zuletzt geprüft: 21.07.2021

[DIS17] JOY-IT: *Joy-IT Ultrasonic Distance Sensor*. https://cdn-reichelt.de/documents/datenblatt/A300/SEN-US01-DATASHEET.pdf. Zuletzt geprüft: 05.07.2021

[ELI20] EBERT, Nancy; LIPKE, Sina V.: *Konzeption, Implementierung und Inbetriebnahme einer modularen Softwarebibliothek zur Unterstützung des CAN-Protokolls durch Arduino-Systeme*, Hochschule Mittweida, Belegarbeit, 2020

[ENG00] ENGELS, Horst: *CAN-Bus: Feldbusse im Überblick, CAN-Bus-Protokolle, CAN-Bus-Meßtechnik, Anwendungen*. Poing : Franzis, 2000. – ISBN 3–7723–5145–x

[ESP18] OLIMEX LTD.: *ESP32-EVB*. Version: F. https://github.com/OLIMEX/ESP32-EVB/raw/master/HARDWARE/REV-F/ESP32-EVB_Rev_F.pdf. Zuletzt geprüft: 06.07.2021

[ESP21] ESPRESSIF SYSTEMS (SHANGHAI) CO., LTD: *ESP32 Series: Datasheet*. Version: 3.6. https://datasheetspdf.com/pdf-file/1097467/EspressifSystems/ESP32/1. Zuletzt geprüft: 06.07.2021

[ESP31] ESPRESSIF SYSTEMS: *ESP32-WROOM-32: Datasheet*. Version: 3.1 https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf. Zuletzt geprüft: 06.07.2021

[ETS02] ETSCHBERGER, Konrad: *Controller-Area-Network: Grundlagen, Protokolle, Bausteine, Anwendungen*. 3., aktualisierte Aufl. München : Hanser, 2002. – ISBN 3446217762

[EVA62] MOUSER ELECTRONICS, Inc.: *EVAL6207N*. https://www.mouser.de/ProductDetail/STMicroelectronics/EVAL6207N?qs=9iSXU9KXrZmaiw9N5nxjTg%3D%3D. Zuletzt geprüft: 06.07.2021

[EVA21] ARROW ELECTRONICS INC.: *EVAL6207N, Evaluation Board based on L6207 Highly Integrated, Mixed-Signal Power IC: Referenzdesign unter Verwendung des Teils L6207 von STMicroelectronics*. https://static5.arrow.com/pdfs/2016/1/4/23/1/11/339/st_/manual/eval6207n.jpg Zuletzt geprüft: 06.07.2021

[HAN18] HANFELD, Pia: *Realisierung eines forensischen Demonstrators für den CAN-Bus*, Hochschule Mittweida, Bachelorarbeit, 2018

[HBG14] HERING, Ekbert; BRESSLER, Klaus; GUTEKUNST, Jürgen: *Elektronik für Ingenieure und Naturwissenschaftler*. 6., vollst. aktualisierte und erw. Aufl. Berlin : Springer Vieweg, 2014 (Lehrbuch). – ISBN 978–3–642–05498–3

[JOY18] JOY-IT: *SBC-CAN01*. Version: 26.09.2018 https://joy-it.net/files/files/Produkte/SBC-CAN01/SBC-CAN01-Anschlussplan.pdf. Zuletzt geprüft: 21.07.2021

[KRA21] KRAUSE, Julius: *Konzeption und Implementierung eines Moduls für lichttechnische Einrichtungen mit einem CAN-Bus-Steuergerät als Komponente in einem forensischen Demonstrator*, Hochschule Mittweida, Masterarbeit, 2021

[LAD20] LADY ADA; ADAFRUIT INDUSTRIES: *Adafruit TB6612 1.2A DC/Stepper Motor Driver Breakout Board*. https://www.mouser.de/datasheet/2/737/adafruit_tb6612_h_bridge_dc_stepper_motor_driver_b-1915378.pdf. Zuletzt geprüft: 06.07.2021

[LAB99] LAWRENZ, Wolfhard; BAGSCHIK, Peter: *CAN Controller area network: Grundlagen und Praxis*. 3., bearb. Aufl. Heidelberg : Hüthig, 1999. – ISBN 3–7785–2575–1

[LEO21] HOBBYTRONICS LTD: *Leonardo CAN BUS board*. Version: 2021 https://www.hobbytronics.co.uk/leonardo-canbus. Zuletzt geprüft: 06.07.2021

[LEO22] HOBBYTRONICS LTD: *Leonardo CAN BUS board*. Version: 2021 https://www.hobbytronics.co.uk/image/cache/data/ht/leonardo-canbus-500x500.jpg. Zuletzt geprüft: 06.07.2021

[LIP20] CONRAD ELECTRONIC SE: *Conrad energy Modellbau-Akkupack (LiPo) 7.4 V 3000 mAh Zellen-Zahl: 2 20 C Stick Hardcase T-Buchse*. https://www.conrad.de/de/p/conrad%2Denergy%2Dmodellbau%2Dakkupack%2Dlipo%2D7%2D4%2Dv%2D3000%2Dmah%2Dzellen%2Dzahl%2D2%2D20%2Dc%2Dstick%2Dhardcase%2Dt%2Dbuchse%2D1414149.html. Zuletzt geprüft: 06.07.2021

[LIP25] REICHELT ELEKTRONIK GMBH & CO. KG: *RD SLP 2700 S2 Akku-Pack, Li-Polymer, 7,4 V, 2700 mAh, 25/50 C*. https://www.reichelt.de/akku%2Dpack%2Dli%2Dpolymer%2D7%2D4%2Dv%2D2700%2Dmah%2D25%2D50%2Dc%2Drd%2Dslp%2D2700%2Ds2%2Dp208382.html. Zuletzt geprüft: 06.07.2021

[LIP19] LIPKE, Sina V.: *Entwicklung eines konfigurierbaren grafischen Benutzerinterfaces für die CAN-Analysesoftware BUSMASTER*, Hochschule Mittweida, Bachelorarbeit, 2019

[LIN17] LIEBETRAU, Thomas; MARQUARDT, Matthias; ELEKTRONIKNET.DE: *Intelligente Ansteuerung hochauflösender LED-Scheinwerfer: Konzept auf Basis von Standard-MCUs*. Version: 04.12.2017 https://www.elektroniknet.de/automotive/assistenzsysteme/intelligente-ansteuerung-hochaufloesender-led-scheinwerfer.148429.html. Zuletzt geprüft: 18.08.2021

[MAN18] MANDL, Peter: *TCP und UDP Internals: Protokolle und Programmierung*. Wiesbaden : Springer Vieweg, 2018. http://dx.doi.org/10.1007/978-3-658-20149-4. http://dx.doi.org/10.1007/978-3-658-20149-4. – ISBN 978–3–658–20148–7

[MBW10] MANDL, Peter; BAKOMENKO, Andreas; WEISS, Johannes: *Grundkurs Datenkommunikation: TCP/IP-basierte Kommunikation: Grundlagen, Konzepte und Standards*. 2., überarbeitete und aktualisierte Auflage. Wiesbaden : Vieweg+Teubner Verlag / GWV Fachverlage GmbH Wiesbaden, 2010. http://dx.doi.org/10.1007/978-3-8348-9699-5. http://dx.doi.org/10.1007/978-3-8348-9699-5. – ISBN 978–3–8348–0810–3

[MIC10] MICHAEL PÄTZOLD, S. S.; WIKIPEDIA: *Stufen des V-Modells*. Version: 26. Januar 2010 https://de.wikipedia.org/wiki/Datei:V-Modell.svg. Zuletzt geprüft: 29.07.2021

[MIC15] MICROSOFT: *Visual Studio Code*. Version: 2015 https://code.visualstudio.com/. Zuletzt geprüft: 26.07.2021

[MOT21] JOY-IT: *MotoDriver2*. https://asset.conrad.com/media10/add/160267/c1/%2D/de/001573541DS01/datenblatt%2D1573541%2Djoy%2Dit%2Dmotormodul%2D2%2Du%2D4%2Dphasen%2D6%2Dbis%2D12v.pdf. Zuletzt geprüft: 06.07.2021

[NOD21] ADVANCED MONOLITHIC SYSTEMS, Inc.: *AMS1117: 1A LOW DROPOUT VOLTAGE REGULATOR*. http://www.advanced-monolithic.com/pdf/ds1117.pdf Zuletzt geprüft: 06.07.2021

[NOD18] JOY-IT: *NodeMCU ESP32: Microcontroller Entwicklungsplatine*. https://cdn-reichelt.de/documents/datenblatt/A300/SBC-NODEMCU-ESP32-DATENBLATT_V1.2.pdf. Zuletzt geprüft: 06.07.2021

[PAT51] PATRICK SCHNABEL; ELEKTRONIK-KOMPENDIUM.DE: *Bluetooth 5 / 5.1 / 5.2 - Unthinkably Connected*. https://www.elektronik-kompendium.de/sites/kom/2107121.htm. Zuletzt geprüft: 06.07.2021

[PAT80] PATRICK SCHNABEL; ELEKTRONIK-KOMPENDIUM.DE: *IEEE 802.11 / WLAN-Grundlagen*. https://www.elektronik-kompendium.de/sites/net/0610051.htm. Zuletzt geprüft: 06.07.2021

[PAT21] PATRICK SCHNABEL; ELEKTRONIK-KOMPENDIUM.DE: *Wi-Fi 5 / IEEE 802.11ac / Gigabit-WLAN*. https://www.elektronik-kompendium.de/sites/net/1602101.htm. Zuletzt geprüft: 06.07.2021

[PLA21] PLATFORMIO LABS OÜ: *Documentation*. Version: v5.2.0a9 https://docs.platformio.org/en/latest/. Zuletzt geprüft: 27.07.2021

[PLA14] PLATFORMIO LABS OÜ: *PlatformIO*. Version: 2014 https://platformio.org/. Zuletzt geprüft: 26.07.2021

[THO21] PROF. DR.-ING. JAN THOMANEK: *Steuergeräte/Software/Vernetzung – Embedded Systems II: Kapitel 1 – Vernetzte Fahrzeugsysteme*

[PWM20] PORTESCAP S.A.: *Steuerung bürstenbehafteter DC-Motoren durch Pulsweitenmodulation (PWM)*. Version: 13.10.20 https://www.pressebox.de/inaktiv/portescap%2Dsa/Steuerung%2Dbuerstenbehafteter%2DDC%2DMotoren%2Ddurch%2DPulsweitenmodulation%2DPWM/boxid/1027271. Zuletzt geprüft: 27.07.2021

[REI11] REIF, Konrad: *Bosch Autoelektrik und Autoelektronik: Bordnetze, Sensoren und elektronische Systeme ; mit 43 Tab*. 6., überarbeitete und erweiterte Auflage. Wiesbaden : Vieweg+Teubner Verlag / Springer Fachmedien Wiesbaden GmbH Wiesbaden, 2011 (Bosch Fachinformation Automobil). http://dx.doi.org/10.1007/978-3-8348-9902-6. http://dx.doi.org/10.1007/978-3-8348-9902-6. – ISBN 978–3–8348–1274–2

[RIO21] RIOT: *Olimex ESP32-EVB*. https://doc.riot-os.org/group__boards__esp32__olimex-esp32-evb.html. Zuletzt geprüft: 06.07.2021

[ROB20] JOY-IT: *ROBOT CAR KIT: Robot03*. https://joy-it.net/files/files/Produkte/Robot03/Robot03-Anleitung-20201023.pdf. Zuletzt geprüft: 05.07.2021

[SAU13] SAUTER, Martin: *Grundkurs Mobile Kommunikationssysteme: UMTS, HSPA und LTE, GSM, GPRS, Wireless LAN und Bluetooth*. 5., überarb. und erw. Aufl. 2013. Wiesbaden and s.l. : Springer Fachmedien Wiesbaden, 2013. http://dx.doi.org/10.1007/978-3-658-01461-2. http://dx.doi.org/10.1007/978-3-658-01461-2. – ISBN 978–3–658–01460–5

[STD16] SEEED TECHNOLOGY INC.; DMITRY PERESLEGIN: *Arduino MCP2515 CAN interface library*. Version: 2013, 2016 https://github.com/autowp/arduino-mcp2515. Zuletzt geprüft: 06.07.2021

[SPI14] Das SPI-Interface. In: *hf-praxis* 19 (2014), Nr. 12, 26–30. www.beam-verlag.de/app/download/16534234/hf%2Dpraxis+12%2D2014+I.pdf. Zuletzt geprüft: 29.07.2021

[TTB66] *TB6612 1.2A DC/Stepper Motor Driver*. http://wiki.t-o-f.info/uploads/Arduino/Adafruit_TB6612_1motor_bb.png. Zuletzt geprüft: 06.07.2021

[TOS07] TOSHIBA: *TB6612FNG: Driver IC for Dual DC motor*. https://www.sparkfun.com/datasheets/Robotics/TB6612FNG.pdf. Zuletzt geprüft: 19.08.2021

[EVE21] EVELTA ELECTRONICS: *TB6612 1.2A DC/Stepper Motor Driver Breakout Board*. Version: 2021 https://cdn11.bigcommerce.com/s-3fd3md1ghs/images/stencil/1280x1280/products/29289/10508/2448__50524.1571920824.jpg?c=2. Zuletzt geprüft: 06.07.2021

[VEC21] VECTOR INFORMATIK GMBH: *VN1600: Flexible Bus-Interfaces für CAN, LIN, K-Line, J1708 und IO*. Version: 2.3 https://assets.vector.com/cms/content/products/VN16xx/docs/VN16xx_FactSheet_DE.pdf. Zuletzt geprüft: 05.07.2021

[TEX18] TEXAS INSTRUMENTS: *SN65HVD23x 3.3-V CAN Bus Transceivers*. https://www.ti.com/lit/ds/symlink/sn65hvd230.pdf?ts=1625637111773&ref_url=https%253A%252F%252Fwww.google.com%252F. Zuletzt geprüft: 07.07.2021

[ZIS14] ZIMMERMANN, Werner; SCHMIDGALL, Ralf: *Bussysteme in der Fahrzeugtechnik: Protokolle, Standards und Softwarearchitektur*. 5., aktual. und erw. Aufl. Wiesbaden : Springer Vieweg, 2014 (ATZ / MTZ-Fachbuch). http://dx.doi.org/10.1007/978-3-658-02419-2. http://dx.doi.org/10.1007/978-3-658-02419-2. – ISBN 978–3–658–02418–5

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe. Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Mittweida, 24. August 2021