



Hochschule Karlsruhe – Technik und Wirtschaft
Fakultät für Maschinenbau und Mechatronik

Automatisierung einer experimentellen BUS-Analyse

Wintersemester 2016/2017

Bachelor-Thesis (B. Eng.)

von

Julian Amann

geboren am 10.12.1993

in Saarlouis

Matrikelnummer.: 43142

Betreuer der Hochschule Karlsruhe

Prof. Dr. rer. nat Peter Neugebauer

Betreuer am Institut für Energieeffiziente Mobilität

Marcel Rumez M. Sc.

Dauer der Arbeit

1. Dezember 2016 - 9. März 2017



Fakultät für Maschinenbau und Mechatronik

Bachelor-Thesis:	Julian Amann Fahrzeugtechnologie
Arbeitsplatz:	Hochschule Karlsruhe – Technik und Wirtschaft Fakultät für Maschinenbau und Mechatronik 76133 Karlsruhe
Betreuer am Arbeitsplatz:	Rumez, Marcel M. Sc.
Betreuer Dozent:	Prof. Dr. rer. nat. Peter Neugebauer
Datum der Ausgabe: 1.12.2016	Abgabetermin: 28.02.2017 <i>31.03.2017</i> Skrida
Kurzthema / Subject:	

Automatisierung einer experimentellen CAN-Bus-Analyse Automation of an experimental CAN bus analysis

Aufgabenstellung:

Zur Auswertung des Datenverkehrs eines KFZ ist es notwendig, die Zuordnung technischer Sachverhalte wie Messwerte, Stellglied-Ansteuerungen oder Sensorwerten zu den entsprechenden Bus-Botschaften zu kennen. Dies erfolgt üblicherweise in der Kommunikations-Matrix (K-Matrix), die aber durch den OEM nicht immer zur Verfügung gestellt wird. In dieser Arbeit soll die Analyse eines Fahrzeug-CAN-Bus-Systems automatisiert werden, um die Botschaften der Daten-Bytes schnell und zuverlässig lesbar zu machen. Im Ergebnis soll ein automatisiertes Erstellen der K-Matrix zumindest teilweise möglich sein.

Dazu soll zunächst eine experimentelle Referenz geschaffen werden: verschiedene Betriebszustände des Fahrzeugs (und deren Änderung) sollen mit dem zeitgleichen Busverkehr verglichen werden. Die Auswertung erfolgt manuell für einige ausgewählte Signale. Im Anschluss muss eine geeignete Datenverwaltung gewählt und durchgeführt werden, um eine Basis für die automatisierte Verwaltung und Analyse der Daten zu schaffen. Um eine allgemein anwendbare Analyse zu gewährleisten, muss die Erhebung und Analyse unabhängig von der Bus-Schnittstelle möglich sein.

Im Einzelnen sind die folgenden Punkte zu bearbeiten:

- Einarbeitung in die Theorie von CAN-Systemen und der Datenauswertung
- Vorbereitung und Durchführung der experimentellen Datenerhebung
- Aufzeichnung und manuelle Analyse der gewonnenen Daten anhand einzelner Botschaften
- Erarbeiten eines Algorithmus zur automatisierten Datenanalyse
- Anwendung der automatisierten Analyse auf weitere Daten
- Dokumentation und Präsentation der Ergebnisse

Vorsitzender des
Prüfungsausschusses

Prof. Dr.-Ing. Norbert Skricka

Betreuer Dozent

Prof. Dr. rer. nat. Peter Neugebauer

Zusammenfassung

Ein Beispiel für ein Netzwerk aus Steuergeräten, die in Fahrzeugen untereinander kommunizieren, ist der CAN-BUS. Die Mikrocomputer versenden Botschaften mit Informationen, die aus Sensoren gewonnen wurden, oder aufgrund einer Aktion des Fahrzeugführers weitergeleitet werden müssen. Aufgrund der Anzahl an Botschaften sind geregelte Kommunikationsstrukturen unter den Steuergeräten unabdingbar. Da für diese Strukturen allerdings kein Standard definiert ist, sind diese Regeln nur dem Hersteller bekannt. Aus diesem Grund ist es dem Benutzer nicht möglich herauszufinden welche Informationen in den Botschaften stecken. Diese Arbeit beschäftigt sich damit, diese Botschaften zu entschlüsseln, indem den gesendeten Signalen reale Ereignisse zugeordnet werden. Diese Zuordnungen werden in tabellarischer Form aufgelistet und als Kommunikationsmatrix bezeichnet.

Nachdem einige Signale manuell zugeordnet wurden, folgt die Entwicklung eines Algorithmus, der die Verbindungen automatisiert erfassen soll. Dazu wird eine Versuchsdurchführung vorgeschlagen, die die notwendigen Datensätze an realen Fahrzeugen erzeugt. Für die weitere Analyse werden die Daten in Matlab eingelesen, wo in einem KDD-Prozess die entsprechenden Signale ermittelt werden. Unterschieden wird dabei, ob der User bereits Vorwissen zu den Kommunikationsstrukturen des Busses hat und deshalb eine gezielte Suche durchführen will, oder ob er eine allgemeine Suche ohne Vorwissen starten muss. Sobald die richtigen Signale gefunden sind, wird eine Datenbasis in Excel durch diese erweitert. Die gewonnenen Informationen dienen als Grundlage für zukünftige Projekte, in denen es immer wieder nötig ist, die Korrelation zwischen Signalen und realen Ereignissen zu kennen. Außerdem wird eine Unabhängigkeit vom Hersteller geschaffen.

Abstract

One example for a network of control-units that communicate among each other in vehicles is the CAN-BUS. The microcomputers send messages containing information obtained from sensors or information that must be transferred due to a driver's action. Because of the number of messages, regulated communication-structures between the control-units are needed. Since there is no standard, those rules are only known by the manufacturer. For this reason, it is impossible for the user to find out what information the message contains. This thesis aims to decipher those messages by correlating the sent signals with real events. Those correlations are listed in table form and are called matrix of communication.

After this has been done manually for chosen signals, the development of an algorithm, that aims to find the correlations automatically, follows. Therefore, an experimental procedure, which generates the required data sets from real vehicles, is proposed. For further analysis, the data sets are imported to Matlab which identifies the corresponding signals by using a KDD-process. A distinction is made according to whether the user has previous knowledge of the communication-structures of the BUS or whether he has to search in general. Once the correct signals are found, a data-base in Excel will be created and extended.

The derived information provide a basis for future projects that frequently need the correlations between signals and real events. Furthermore, an independence of the manufacturer is created.

Erklärung

Hiermit versichere ich wahrheitsgemäß, dass diese Arbeit selbstständig angefertigt und alle benutzten Hilfsmittel vollständig und genau angegeben wurden.

Quellen, die im Wortlaut oder ihrem Sinn nach entnommen wurden, sind durch genaue Angabe deren Herkunft kenntlich gemacht und vermerkt.

Karlsruhe, den 9. März 2017

Julian Amann

Inhaltsverzeichnis

Abbildungsverzeichnis	IX
Tabellenverzeichnis	XI
1. Einleitung	1
2. Grundlagen	3
2.1. Reverse Engineering	3
2.2. Bussysteme	4
2.3. Datenverarbeitung	18
2.4. Hilfsmittel	21
3. Entwicklung der Methodik	23
3.1. Ausgangssituation	23
3.2. Verwendete Kommunikationsmatrix	24
3.3. Manuelle Analyse	25
3.4. Automatisierung der Analyse	27
3.5. Korrektes Interpretieren der gefundenen Signale	32
4. Implementierung	35
4.1. Datenverarbeitung ohne Vorwissen	35
4.1.1. Selection - Selektion	35
4.1.2. Preprocessing - Vorverarbeitung	36
4.1.3. Transformation	37
4.1.4. Data-Mining - Datenergründung	38
4.1.5. Interpretation/Evaluation	42
4.2. Datenverarbeitung mit Vorwissen	43

5. Ergebnisbetrachtung	45
5.1. Ergebnisse	45
5.2. Verifizierung der Ergebnisse	47
6. Fazit und Ausblick	49
A. Anhang	55
A.1. Auszug aus dem Datenblatt des NXP TJA 1051/3	55
A.2. Programmablaufplan der Unterfunktion „Daten mit CANoe auf- zeichnen“	56
A.3. Programmablaufplan der Funktion „identifizieren“	57
A.4. Ausgewählte Signale der K-Matrix des BMW i3	66
A.5. CD	69

Nomenklatur

ACK	Acknowledgement
BUS	Binary Unit System
CAN	Controller Area Network
CANH	CAN-High-Leitung
CANL	CAN-Low-Leitung
CAPL	Communication Access Programming Language
CRC	Cyclic Redundancy Check
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
DEL	Delimiter
DLC	Data Length Code
ECU	Electronic Control Unit
EMV	Elektromagnetische Verträglichkeit
EOF	End Of Frame
HsKA	Hochschule Karlsruhe - Technik und Wirtschaft
ID	Identifier
IDE	Identifier Extension
IEEM	Institut für Energieeffiziente Mobilität
ISO	International Organization for Standardization
KDD	Knowledge Discovery in Databases
Kfz	Kraftfahrzeug

Inhaltsverzeichnis

KNN	Künstliche Neuronale Netze
K-Matrix	Kommunikationsmatrix
l_{BUS}	Buslänge
LIN	Local Interconnect Network
LSB	Least Significant Bit
μ C	Mikrocontroller
MOST	Media Oriented Systems Transport
MSB	Most Significant Bit
NRZ	Non Return to Zero
OBD	On Board Diagnose
OEM	Original Equipment Manufacturer
OSI	Open System Interconnection
Pap	Programmablaufplan
Pkw	Personenkraftwagen
RE	Reverse Engineering
RTR	Remote Transmission Request
SAE	Society of Automotive Engineers
SOF	Start Of Frame
USB	Universal Serial Bus

Abbildungsverzeichnis

2.1. Darstellung der CANH und CANL mit Abschlusswiderständen am High-Speed-CAN	10
2.2. Spannungsspeigel bei Low- und High-Speed-CAN	11
2.3. Wired-And-Schaltung	11
2.4. CAN-Data-Frame	12
2.5. Überblick Knowledge Discovery in Databases	20
3.1. Ausschnitt des Trace-Fensters des Body-CANs des BMW i3	26
3.2. Versuchsaufbau zur Datenerhebung	26
3.3. Programmablaufplan zur Ermittlung von Signalen	28
3.4. Ausschnitt einer von CANoe erzeugten xml-Datei	29
3.5. Workspace nach korrekter Variablendeclaration	30
3.6. Ergebnistabelle für das Beispiel „Warnblinker anschalten“	31
4.1. Klartextausgabe zur einfacheren Interpretation für den User	42

Tabellenverzeichnis

2.1. OSI-Schichtenmodell	6
2.2. SAE-Klassifikation von Bussystemen	8
2.3. Wichtige CAN-Standards	8
2.4. Funktionen der einzelnen Felder im Data-Frame	13
2.5. Bitweise Arbitrierung	14
2.6. K-Matrix, die nur Sender und Empfänger beinhaltet	16
2.7. Beispielhafte K-Matrix, die auch Informationen zu Botschaften und Signalen beinhaltet	17
3.1. Beispielhaft verwendete Kommunikationsmatrix	24
3.2. Anordnung des Trace-Fensters in CANoe	25
3.3. Auswahl vermuteter IDs nach manueller Analyse	27
4.1. Benötigte Form der Datenmatrix	38
4.2. Eingabeparameter für die Funktion „identifizieren_mit_vorwissen“ . .	43

1. Einleitung

Ein gutes Unternehmen zeichnet sich unter anderem durch eine strukturierte Kommunikation aus. Wenn eine Abteilung etwas bereits berechnet hat, ist es nicht nötig, dass eine zweite Abteilung die gleiche Berechnung noch einmal durchführen muss. Viel sinnvoller ist es, die Ergebnisse an alle Abteilungen weiterzugeben, für die sie relevant sind. Stellt man sich nun vor, man hätte ein Netzwerk für Informationen, an das alle Abteilungen angeschlossen sind, könnte jeder seine Ergebnisse an jeden weitergeben. Allerdings wäre man vermutlich schnell damit überfordert, herauszufiltern, welche Botschaften wichtig für die eigene Abteilung sind und welche nicht. Außerdem könnte es passieren, dass sehr wichtige Nachrichten in der schieren Menge verloren gehen. Somit wird schnell die Notwendigkeit klar, dass gewisse Regeln für das Weiterleiten von Informationen vorgegeben werden müssen. Eine Möglichkeit wäre zum Beispiel, jeder Nachricht eine Nummer zuzuordnen. Für jede Nummer könnte definiert werden, wie wichtig die Nachricht ist, die dahinter steckt und für welche Abteilung sie relevant ist.

Ersetzt man nun die Abteilungen durch Steuergeräte und bewegt sich im Kontext von Fahrzeugen, würde man ein solches Netzwerk des Informationsaustauschs als BUS bezeichnen. Durch steigende Anforderungen an Komfort, Sicherheit und Spaß ist es nötig, dass Autos immer mehr Daten von ihrer Umwelt und den Fahrzeugführern aufnehmen. Dazu dienen Bedienelemente im Fahrerraum und Sensoren, deren Signale in Steuergeräten weiterverarbeitet werden. Auch hier wird versucht, so effizient wie möglich zu arbeiten. Ist einem Steuergerät ein Signal bereits bekannt, gibt es dieses an alle Steuergeräte weiter, für die es interessant ist. Doch wie in unterschiedlichen Unternehmen haben unterschiedliche Hersteller auch unterschiedliche Strukturen für die Kommunikation der Steuergeräte. Um jedoch herauszufinden, welche Informationen dafür sorgen, dass genau das vom Fahrer gewünscht Ereignis eintritt, ist es essentiell zu wissen, welche Botschaften zum Zeitpunkt der durchgeführten Aktion auf dem BUS versendet werden. Könnte

1. Einleitung

man diese Botschaften herausfinden, wäre es möglich, dem Fahrzeug Parameter zu entlocken, die wiederum neue Berechnungen möglich machen könnten.

Die vorliegende Arbeit beschäftigt sich mit ebendieser dieser Thematik: Dem automatisierten Herausfinden von Botschaften, die als Reaktion auf ein bestimmtes Verhalten des Fahrers gesendet werden. Dazu wird zunächst eine experimentelle Referenz geschaffen, indem ein CAN-BUS händisch auf Zusammenhänge zwischen den Zuständen des Busses und den vorgegebenen Ereignissen untersucht wird. Mit Hilfe der zuvor beschriebenen Grundlagen wird dann eine Methodik entwickelt, mit der diese manuelle Analyse automatisiert wird. Danach wird die Implementierung der entwickelten Funktionen erläutert und die Ergebnisse und deren Verifizierung geschildert. Zum Schluss wird ein Fazit gezogen und ein Ausblick gegeben.

2. Grundlagen

Die folgenden Abschnitte geben einen Überblick über eine Auswahl der Grundlagen, die zum Entwickeln der Methodik nötig sind. Dieser beinhaltet eine Zusammenfassung zum Reverse Engineering, dem KDD-Prozess und zu Bussystemen mit Hauptaugenmerk auf dem CAN-BUS.

2.1. Reverse Engineering

Möchte man ein Produkt fertigen, erstellt man einen Plan und kann aufgrund dessen die einzelnen Arbeitsschritte durchführen und dabei die richtigen Maße einhalten. Dreht man dieses Vorgehen um, also erstellt man einen Plan aufgrund eines Produktes, spricht man vom sog. *Reverse Engineering*. Dieses „(...)“ bezeichnet die Nachkonstruktion eines bereits bestehenden Produktes. Durch Zerlegung des Produktes kann auf die Funktions-, Design- und Fertigungsprinzipien sowie die Wertschöpfungsstruktur geschlossen werden.“ [1]

Im Rahmen dieser Thesis wird versucht ein Produkt (nämlich die Kommunikationsmatrix von Fahrzeugen) nachzukonstruieren. Da diese in der Regel nur dem **Original Equipment Manufacturer (OEM)** zur Verfügung steht und sie somit durch Rückschlüsse basierend auf dem zugrundeliegenden BUS-Verkehr erstellt werden muss, wird dieser Vorgang als Reverse-Engineering-Methode eingestuft.

Das Verfahren stammt ursprünglich aus dem Maschinenbau, wo man zum Beispiel ein Bauteil nach der Fertigung digitalisiert und somit einen Soll-Ist-Vergleich von der digitalen Planversion und dem entsprechenden „echten“ Modell erhält.

In der Softwaretechnik versteht man unter Reverse Engineering „(...)“ den Prozess, die einem fertigen (und meist schlecht dokumentierten) Softwaresystem zugrundeliegenden Ideen und Konzepte aufzuspüren und zu dokumentieren. Der Entwick-

2. Grundlagen

lungsprozess wird gewissermaßen rückwärts durchlaufen.“ [2] Es beschreibt also den Vorgang, durch die Analyse von Daten die Funktionalität eines Programms (oder eben die Strukturen einer Kommunikationsmatrix) zu ermitteln.

„Zweck des Reverse Engineering kann zum einen die Analyse von Wettbewerbsprodukten sein, aber auch das Erkennen von Differenzierungsmöglichkeiten.“ [1] Das bedeutet also, dass Unternehmen, die mit Reverse Engineering arbeiten, sowohl einen Vorteil haben was die eigene Fertigung betrifft, als auch im Wissen über die Methoden der direkten Wettbewerber. Dies setzen viele OEMs mit Industriespionage gleich, da sich fremdes geistiges Eigentum angeeignet wird und es ist damit auch der Grund, aus dem viele Hersteller in ihren Lizenzbedingungen verbieten ihre Produkte einem Reverse-Engineering-Prozess zu unterziehen.

Solche Verbote sind allerdings oftmals nicht grenzenlos gültig, denn der Nutzer eines Produktes hat generell das Recht zur Überprüfung der Anwendungssicherheit seiner Software. Zusätzlich greift im Rahmen der Forschung Art. 5 Abs. 3 GG: „Kunst und Wissenschaft, Forschung und Lehre sind frei. Die Freiheit der Lehre entbindet nicht von der Treue zur Verfassung.“ Diese *Freiheit der Forschung* berechtigt grundsätzlich dazu, das Reverse Engineering anzuwenden, solange es nicht dazu führt, dass ein Produkt im Nachhinein kommerziell genutzt wird. Somit ist die vorliegende Arbeit, die ausschließlich im Rahmen der Forschung durchgeführt wird, immer mit dem Gesetz vereinbar und beinhaltet keine rechtlich bedenklichen Methoden.

2.2. Bussysteme

Da Kunden immer höhere Ansprüche an Sicherheit, Komfort und Entertainment haben und diese Eigenschaften maßgeblich von der Elektronik beeinflusst werden, findet ein Großteil der Innovationen im Automobil auf ebendieser Ebene statt. Daraus resultiert ein immenser Innovationsdruck für Automobilhersteller, die jeweils bestrebt sind, die Wünsche als erster und am besten zu erfüllen. So ist es heute nicht unüblich, mehr als 80 Steuergeräte (engl. **Electronic Control Unit (ECU)**) in einem **Personenkraftwagen (Pkw)** zu finden. Zu Beginn wurden diese konventionell vernetzt: jedes Steuergerät, das eine Information eines anderen Steuergeräts benötigte, wurde mit diesem verbunden. Mit steigender Zahl der ECUs folgte daraus ein enormer Verkabelungsaufwand, der sich sowohl in der Masse als auch in

den Kosten des Fahrzeugs niederschlug. Um diesen Problemen entgegenzuwirken brauchte man bald eine neue Lösung, die die Signale schnell und sicher im System verteilen konnte: „der bitserielle Austausch von Daten über einen von einer Anzahl elektronischer Steuergeräte gemeinsam genutzten Kommunikationskanal (**Binary Unit System (BUS)**).“ [3] Das bedeutet, dass nun nicht mehr jedes Steuergerät untereinander verkabelt wird, sondern, dass es eine Leitung gibt, auf der alle Signale nacheinander übertragen werden und auf die jede ECU zugreifen kann. So spart man nicht nur Kosten und Masse, da der Kabelbaum drastisch schrumpft, man garantiert auch einen zuverlässigen Datenaustausch und hat den Vorteil der Mehrfachnutzung der Daten: die Information eines Sensors, der zuvor vielleicht zwei mal verbaut werden musste, kann nun auf den BUS gegeben werden, wodurch jedes Steuergerät mit dieser Information arbeiten kann.

Mit steigender Beliebtheit der Bussysteme entwickelten sich immer mehr verschiedene Kommunikationsprotokolle. Auf diese Weise wird allerdings die Kommunikation über unterschiedliche technische Systeme hinweg unmöglich gemacht. Daraus folgt die Notwendigkeit eines einheitlichen Systems, das diese Kommunikation ermöglicht und somit herstellerübergreifende Innovationen möglich macht. Eine Lösung bot 1984 die **International Organization for Standardization (ISO)**, die den gesamten Datentransfer eines Systems in einzelne Schichten (Layer) unterteilte und in ihrer ISO/IEC 7498-1 als **Open System Interconnection (OSI)** standardisierte [4]. Dieses Kommunikationsmodell unterteilt die komplexen Strukturen der bitseriellen Kommunikation in sieben Schichten (Siebenschichtenmodell) und definiert die Kommunikation zwischen diesen Schichten. Die sieben Schichten sind in Tabelle 2.1 dargestellt.

2. Grundlagen

Tabelle 2.1.: Zusammenfassung des OSI-Schichtenmodells aufgeteilt in Layer, Schicht und Funktionen [4, 5, 6].

	Layer	Schicht	Funktionen
7	Application	Anwendung	Zugriff auf das Kommunikationssystem, Entkopplung, Anwendung von Kommunikation
6	Presentation	Darstellung	Semantik, Datenkompression, Verschlüsselung, Übersetzer verschiedener Datenformate
5	Session	Sitzungssteuerung	Unterhalten längerer Sitzungen, Definition von Synchronisationspunkten
4	Transport	Datentransport	Verbindungsaufl- und abbau, Segmentierung, Sequenzierung, Assemblierung
3	Network	Vermittlung	Routing (Wahl des Übertragungsweges zwischen zwei Busknoten)
2	Data Link	Datensicherung	Botschaftsaufbau, Buszugriff, Flusskontrolle, Fehlersicherung
1	Physical	Bitübertragung	Physikalische Busankopplung, Stecker, Übertragungsmedium, Leitungscodierung

Das OSI-Schichtenmodell sorgt damit für klare Hierarchien, wodurch sich eine sichere Kommunikation zwischen den Steuergeräten (oder auch „Knoten“) ergibt, selbst wenn sie von verschiedenen Herstellern kommen [5]. Allerdings sind für die Anwendung im **Kraftfahrzeug** (Kfz) nicht alle Schichten notwendig. Wichtig sind vor allem die beiden untersten Layer „Physical“ und „Data Link“. Der Physical-Layer sorgt bei der seriellen Bitkommunikation für die physikalische Übertragung der Signale. Dies geschieht für gewöhnlich durch einen sogenannten Transceiver¹, dessen Aufgabe es beim Senden ist, die empfangenen Daten in die korrekten Spannungspiegel umzusetzen. Beim Empfangen wandelt er die Spannungspiegel dagegen in für den Controller verwertbare Signale um. Die Spannungspiegel sind dabei im Protokoll der ersten Schicht definiert. Die Aufgabe der Datensicherungsschicht übernimmt in der Regel ein Kommunikationscontroller, der über das Protokoll der

¹ von engl. Transmitter (Sender) und Receiver (Empfänger)

zweiten Schicht die Adressierung, die Nachrichtenbildung (Framing) und sowohl die Fehlererkennung als auch die Fehlerbehebung abwickelt. Somit reduziert sich das Siebenschichtenmodell auf ein Zweischichtenmodell, was den Umgang mit den Protokollen stark vereinfacht. Alles Weitere, was nicht eindeutig einer Schicht zugeordnet werden kann, wird unter der Anwendungsschicht verbucht. Daraus ergibt sich also ein Dreischichtenmodell, das alles Notwendige für die Kommunikation zwischen den ECUs beinhaltet.

Neben dem wohl wichtigsten Bussystem, dem **Controller Area Network (CAN)** (vgl. Unterabschnitt 2.2), haben sich noch weitere Systeme etabliert, die hier kurz beschrieben werden sollen. Diese haben sich vor allem daraus entwickelt, dass man einerseits Systeme benötigte, die so günstig wie möglich und andererseits so schnell wie möglich waren (um sicherheitskritische Signale wie zum Beispiel Bremssignale weiterzuleiten). Daraus ergibt sich eine Scherung, die im Normalfall nicht von einem einzigen System zu stemmen ist. Ein Vertreter der Busse, die in sicherheitskritischen Gebieten verwendet werden, ist der sog. *Flexray*. Dieser „herstellerübergreifende, deterministische und fehlertolerante Kommunikationsstandard“ erreicht eine Datenrate² von bis zu 10 MBd [7]. Im Gegensatz dazu existiert das sog. **Local Interconnect Network (LIN)**. Diese kostengünstige Variante wird mittlerweile gerne in der Komfortelektronik als Kommunikationsschnittstelle zwischen Sensoren und Aktoren verbaut. Da es allerdings als Übertragungsmedium nur eine Eindrahtleitung verwendet, ist das Netzwerk relativ störanfällig, was eine Verwendung in sicherheitsrelevanten Bereichen unmöglich macht. Im Multimediacbereich hat sich darüber hinaus der sog. **Media Oriented Systems Transport (MOST)** durchgesetzt, der Lichtwellenleiter als Übertragungsmedium verwendet und dadurch Bitraten von bis zu 150 MBd erreicht [8].

Die **Society of Automotive Engineers (SAE)** führte aufgrund der hohen Anzahl verschiedener Bussysteme 2002 eine Klassifizierung ein, die vor allem die Geschwindigkeit und die Kosten pro Knoten beschreibt. Dazu werden die Systeme in die Klassen *A* bis *C* sowie in eine Klasse für *Multimedia* unterteilt. Diese sind in Tabelle 2.2 dargestellt.

²wird in Baud [Bd] angegeben, mit $1Bd = 1\frac{Bit}{s}$

2. Grundlagen

Tabelle 2.2.: SAE-Klassifikation von Bussystemen [8, 9].

Klasse	Preis/Knoten	Bitrate	Typische Vertreter	Typische Anwendung
A	$\approx 4 \$$	$<25 kBd$	LIN	Subbussysteme
B	$\approx 5 \$$	$<125 kBd$	CAN (Low-Speed)	Karosserieelektronik
C	$\approx 10 \$$	$<1000 kBd$	CAN (High-Speed)	Antriebssysteme
Multimedia	$> 10 \$$	$<10 MBd$	MOST	Multimedia

Controller Area Network

Seit 1994 gilt bei Bussystemen ein offener Industriestandard: CAN. Dieses Bussystem wurde Mitte der 80er-Jahre von der *Robert Bosch GmbH* präsentiert, zum ersten Mal in Zusammenarbeit mit der *Intel Corporation* implementiert und ist mittlerweile in der *ISO 11898* standardisiert. Die wichtigsten Standards für CAN sind in folgender Tabelle aufgeführt:

Tabelle 2.3.: Wichtige genormte CAN-Standards und wofür sie benötigt werden [10].

ISO 11898 - 1	Data Link Layer
ISO 11898 - 2,5,6	Physical Layer für High-Speed-CAN
ISO 11898 - 3	...für Low-Speed-CAN
ISO 11898 - 4	Erweiterung des Data Link Layers für zeitgesteuerte Kommunikation

CAN zeichnet sich durch seine hohe Übertragungsgeschwindigkeit von $125 kBd$ (Low-Speed-CAN) bzw. bis zu $1 MBd$ (High-Speed-CAN) aus³. Dabei kommt der High-Speed-CAN vor allem in echtzeitrelevanten Netzwerken wie z.B. dem Fahrwerk zur Anwendung, während man den Low-Speed-CAN gerne in Komfortanwendungen zum Einsatz kommen lässt. Da CAN außerdem eine sehr geringe Fehleranfälligkeit aufweist, wird er neben der Automobilindustrie auch in industriellen Anwendungen benutzt. Daraus ergibt sich eine hohe Stückzahl an CAN-Controllern, die wiederum dazu führt, dass der Preis für die zugehörigen Steuergeräte weiter sinkt. In einem Umfeld, in dem Entscheidungen auch an Cent-

³im Anwendungsfall sind allerdings aufgrund von Störungen nur ca. $500 kBd$ üblich [6]

Beträgen festgemacht werden müssen, ist dies eine weitere Stärke des Netzwerks. Die Knoten sind in einem CAN-Netzwerk in der Regel in einer *Linientopologie* angeordnet und folgen dem *Multi-Master-Prinzip*. Das bedeutet, dass alle Knoten gleichberechtigt sind und somit jeder Knoten ereignisgesteuert auf den BUS zugreifen darf [10, 6].

CAN-Protokoll: Physical Layer

Der *Physical Layer* regelt die Bitübertragung des Netzwerks, d.h. die Ankopplung an den BUS, die Codierung des Bitstroms, das Timing und die Synchronisierung (vgl. Tabelle 2.1). Wichtig dafür sind Bauteile wie der Mikrocontroller (μ C), der CAN-Controller, der CAN-Transceiver sowie die **CAN-High-Leitung** (CANH) und die **CAN-Low-Leitung** (CANL). Die Schnittstelle kann man grob in zwei Teile gliedern: die Kommunikationshardware und die Kommunikationssoftware [11]. Die Basis für die Kommunikationssoftware bildet der μ C, in dem höhere Kommunikationsdienste hinterlegt sind [11]. Die grundlegenden Funktionen sind hingegen in den restlichen Bauteilen implementiert. Der CAN-Controller wickelt das Protokoll ab, während der Transceiver die physikalische Verbindung zu CANH und CANL darstellt. Hierbei handelt es sich um das Übertragungsmedium in Form von zwei verdrillten Leitungen. Es wird also einmal ein höherer und ein niedrigerer Spannungspegel übertragen. Das Nutzen dieser sog. *Twisted-Pair-Leitung* hat den Vorteil, dass in EMV-kritischen⁴ Situationen beide Leitungen gleich beeinflusst werden. Bildet man im Nachgang die Differenz $\Delta = U_{High} - U_{Low}$ bleibt diese unbeeinflusst. Der prinzipielle Aufbau eines High-Speed-CAN-Netzwerks ist in Abbildung 2.1 dargestellt.

Die beiden Widerstände von 120Ω dienen dem Abschluss der Leitung und werden deshalb als *Abschlusswiderstände* bezeichnet. Sie sind nötig, um die Reflexionen am Ende der Leitung zu unterdrücken und entsprechen dem Wellenwiderstand der Zweidrahtleitung [6].

Der *High-Speed* und der *Low-Speed-CAN* unterscheiden sich in einigen Punkten. Zunächst ist die Buslänge beim *High-Speed-CAN* wesentlich beschränkter. Da die

⁴Elektromagnetische Verträglichkeit (EMV)

2. Grundlagen

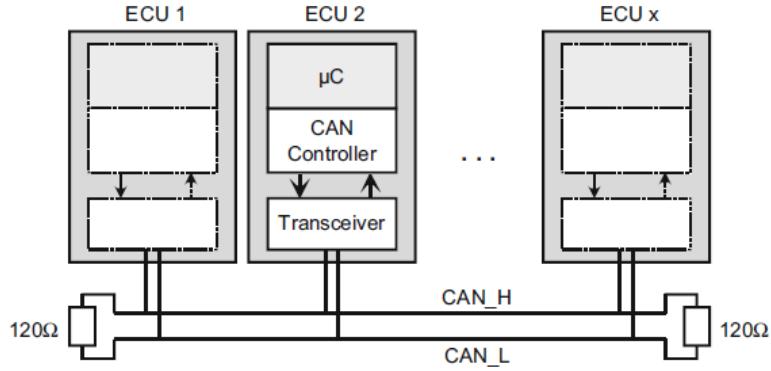


Abbildung 2.1.: Die CAN-Knoten sind in μ Cs, Controller und Transceiver unterteilt und mit dem CANH und CANL verbunden, um das Differenzsignal auswerten zu können. Der Abschlusswiderstand unterdrückt Reflexionen. Abbildung gilt für High-Speed-CAN [10].

Signale schneller eintreffen, ist die Bearbeitungszeit für die Steuergeräte kürzer. Daraus folgt eine Faustformel für die Buslänge, in der erkennbar ist, dass die potentielle Länge des Busses kleiner wird, wenn die Baudrate, die im Nenner steht, zunimmt:

$$l_{BUS} \leq (40...50)m \cdot \frac{1MBd}{Baudrate}$$

Darüber hinaus haben die Spannungspiegel beim *Low-Speed-CAN* deutlich mehr Zeit sich aufzubauen, wodurch wesentlich höhere Pegel als beim *High-Speed-CAN* entstehen. Diese sind in Abbildung 2.2 dargestellt.

Beim Low-Speed-CAN ergibt sich dort eine Differenzspannung von 5 V für den rezessiven Pegel und eine Differenzspannung von 2 V für den dominanten Pegel. Der High-Speed-CAN dagegen wertet ein ΔU von 0 V als rezessive 1 und ein ΔU von 2 V als dominante 0 [13]. Die 0 wird grundsätzlich in Binärcodes als *dominant* bezeichnet, da sie die *recessive 1* überschreibt. Dies spielt zum Beispiel bei der bitweisen Arbitrierung eine große Rolle (vgl. Tabelle 2.5). Die Hierarchie der einzelnen Pegel wird in der Praxis durch eine sog. *Wired-And*-Verdrahtung erreicht. Diese ist in Abbildung 2.3 dargestellt. Liegt beispielsweise am Signaleingang *S1* eine 1 wird diese im Negationsblock zu einer 0 umgewandelt. Somit sperrt der

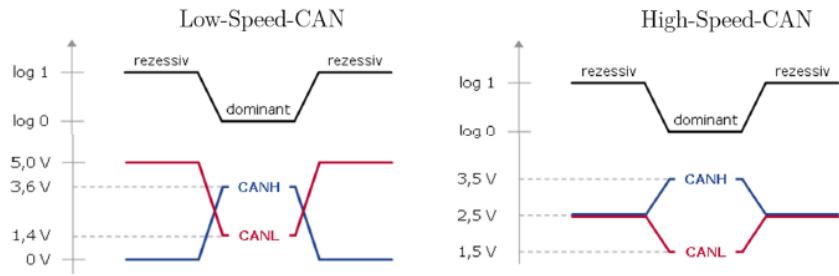


Abbildung 2.2.: Die Spannungspegel bei Low- und High-Speed-CAN unterscheiden sich stark aufgrund der vorhandenen Zeit die Endzustände zu erreichen [12].

Transistor und der BUS wird ebenfalls auf 1 gehoben. Liegt dagegen eine 0 an S1 an, wird diese zu einer 1 negiert. Dadurch schaltet der Transistor durch und zieht den gesamten BUS auf ein bestimmtes Potential, das von dem Widerstand zwischen dem BUS und der Masse abhängig ist. In diesem Fall ist es irrelevant welches Signal an den restlichen Signaleingängen anliegt, da sich dieses Potential (also die logische 0) immer durchsetzt.

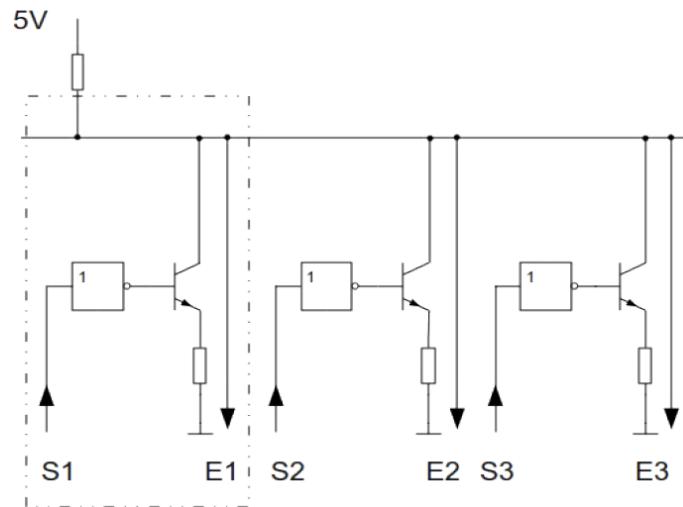


Abbildung 2.3.: Wired-And-Schaltung: Liegt eine Null an einem der Signaleingänge schaltet der zugehörige Transistor durch und zieht den gesamten BUS auf Null, unabhängig was an den anderen Signaleingängen anliegt [6].

2. Grundlagen

CAN-Protokoll: Data-Link Layer

Neben dem *Physical Layer* spielt im CAN-Netzwerk auch der *Data-Link Layer* eine große Rolle. Er regelt den Zugriff und den Aufbau eines sog. *CAN-Frames*. CAN-Frames werden in verschiedene Kategorien unterteilt:

- Data-Frame
- Remote-Frame
- Error-Frame

Im Folgenden wird vor allem auf den *Data-Frame* eingegangen.

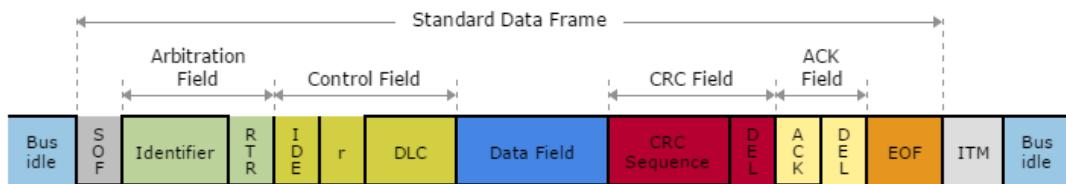


Abbildung 2.4.: Gliederung des Standard-CAN-Data-Frames [14].

Eine Übersicht zu den Funktionen der einzelnen Felder ist in Tabelle 2.4 gegeben. Sendet ein Knoten eine Botschaft, erhält jeder an den BUS angeschlossene Knoten diesen Frame. Allerdings sind die Nutzdaten nicht für jeden Knoten interessant. Diese Tatsache macht eine sog. *Akzeptanzfilterung* notwendig. Wie in Tabelle 2.4 beschrieben, kennzeichnet der **Identifier (ID)** eine Botschaft und ist damit inhalts- und nicht knotenbezogen. Anhand dieses IDs entscheidet die CAN-Schnittstelle, ob die Botschaft, die hinter dem ID steht, für den Knoten relevant oder nicht relevant ist (die Relevanz wird unter anderem in der **Kommunikationsmatrix** (K-Matrix) geregelt, vgl. Unterabschnitt 2.6).

Tabelle 2.4.: Funktionen der einzelnen Felder im Data-Frame [15, 16].

Feld	Name	Länge	Funktion
kein Feld	SOF	1 Bit	Der <i>Start of Frame</i> kennzeichnet den Beginn einer Botschaft. Außerdem dient er der Netzwerksynchronisation, indem er immer dominant übertragen wird
Arbitration Field	Identifier	11 Bits	Er legt die Priorität der Botschaft fest und kennzeichnet für welche Knoten die Botschaft wichtig ist
	RTR	1 Bit	Der <i>Remote Transmission Request</i> kennzeichnet den Frametyp. Im Fall des Data-Frames wird das Bit dominant übertragen
Control Field	IDE	1 Bit	Die <i>Identifier Extension</i> legt fest, ob es sich um das Standard-(11 Bit) oder Extended-Format(29 Bit) des IDs handelt. Im Standard-Format ist es dominant
	r	1 Bit	Reserviert für potentielle Erweiterungen
	DLC	4 Bits	Der <i>Data Length Code</i> gibt an, wie viele Nutzbytes folgen
Data Field	Data Field	0-64 Bits	Enthält die tatsächlichen Nutzdaten und damit die Botschaft
CRC-Field	CRC	15 Bits	Der <i>Cyclic Redundancy Check</i> überprüft die Botschaft durch Bilden der Prüfsumme auf Richtigkeit
	DEL	1 Bit	Rezessiver <i>Delimiter</i> , der auf die CRC folgt
ACK-Field	ACK	1 Bit	War die Prüfsumme korrekt, wird das <i>Acknowledgement</i> -Bit auf dominant gesetzt
	DEL	1 Bit	Rezessiver <i>Delimiter</i> , der auf das ACK-Bit folgt
kein Feld	EOF	7 Bit	Der <i>End of Frame</i> beendet die Botschaft mit sieben rezessiven Bits

Darüber hinaus legt der ID die Priorität der Botschaft fest. Wollen mehrere Knoten gleichzeitig sendend auf den BUS zugreifen (dies ist häufig der Fall, da der CAN-BUS ereignisgesteuert ist), wird die Botschaft mit der höchsten Priorität - das bedeutet mit dem niedrigsten ID - gesendet. Um dies zu garantieren, bietet das **Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA)**-Verfahren eine gute Lösung. Bei der *bitweisen Arbitrierung* vergleichen die Knoten ihre IDs mit dem aktuellen Wert des Busses. Ist der Zustand des Busses dominant, obwohl

2. Grundlagen

ein rezessives Bit gesendet wurde, wird der Arbitrierungsversuch eingestellt, da der ID, gegen den man „antrat“, einen niedrigeren Wert und damit eine höhere Priorität hat. Die gesamte Arbitrierung beruht wiederum auf dem Prinzip der Wired-And-Verdrahtung (vgl. Abbildung 2.3). Tabelle 2.5 zeigt beispielhaft einen Arbitrierungsvorgang.

Tabelle 2.5.: Bitweise Arbitrierung: Die dick markierten Pegel werden gesendet. S1 stellt den Arbitrierungsversuch nach zwei Bits ein, da er ein rezessives Bit sendet aber ein Dominantes auf dem BUS liegt. S2 hört nach drei Bits auf. S3 setzt sich durch [6].

S1	0	1	0	1	0	1	0	1	...
S2	0	0	1	1	0	0	1	1	...
S3	0	0	0	0	1	1	1	1	...
BUS	0	0	0	0	1	1	1	1	...

Neben der Zugriffsregelung ist auch die Datensicherung im Data-Link Layer beschrieben. Neben der in Unterabschnitt 2.2 beschriebenen Twisted-Pair-Leitung kommen auch das sog. *Bitstuffing*, der **Cyclic Redundancy Check** (CRC) und das **Acknowledgement** (ACK)-Bit zur Anwendung. Das Bitstuffing ist notwendig, da im CAN-BUS eine **Non Return to Zero** (NRZ)-Codierung benutzt wird. Bei dieser Codierung wird, im Gegensatz zur sog. *Manchester Codierung*, kein Komplementär-Bit zwischen zwei gleichartigen Bits eingefügt. Daraus entstehen beim längeren Übertragen von gleichen Signalen Synchronisierungsprobleme. Deshalb wird beim Bitstuffing immer nach fünf homogenen Bits ein Komplementär-Bit eingefügt. So kann sich der BUS über die Pegeländerung selbst nachsynchronisieren. Es ist also möglich, dass sich ein Frame nur durch Stuffbits um bis zu 29 Bits auf bis zu 132 Bits verlängert. Dazu kommen der CRC und das ACK-Bit. Bei dem CRC wird auf Basis der *ISO 11898-1* über den gesamten Frame⁵ eine sog. *Checksumme* errechnet. In der Norm ist ein Generatorpolynom definiert, das zu keinem Rest führt, wenn man eine korrekte Checksumme durch das Polynom dividiert. Erhält der Empfänger bei der Division doch einen Rest, muss er das ACK-Bit negativ quittieren, indem er es auf *logisch 1* setzt. Erkennt der Empfänger keinen Fehler, setzt er das ACK-Bit auf *logisch 0*.

⁵ohne Stuffbits

Wurde im Laufe eines Frames ein Fehler erkannt, wird von dem Knoten, der den Fehler entdeckt hat, bewusst gegen die Bitstuffing-Regel verstößen, indem er sechs dominante Bits sendet. Dies führt dazu, dass die restlichen Knoten das Senden einstellen. Dieser Verstoß wird als *aktives Error Flag* bezeichnet und ist ein Teil eines sog. *Error-Frames*. Im Anschluss an das aktive Error Flag kann ein sekundäres Error Flag folgen, welches dem ersten Sender des Error Flags signalisiert, dass er den Fehler zuerst bemerkt hat. Dies wird erkannt, indem sich sein rezessiver Pegel auf dem BUS nicht gegen das dominante sekundäre Error Flag durchsetzen kann. Wird bei einem Knoten mehrfach der gleiche Fehler detektiert, ist es möglich, diesem Knoten das Senden zeitweise zu untersagen bzw. ihn sogar komplett von der BUS-Kommunikation auszuschließen. Dieser Ausschluss wird betrieben, da man davon ausgehen kann, dass der Knoten defekt ist. Im Anschluss an die Error Flags folgt der *Error Delimiter*, der aus acht rezessiven Bits besteht und den Frame abschließt.

Ein weniger häufig im Fahrzeug vorkommender Frame ist der sog. *Remote-Frame*, mit dem man die Nutzdaten von beliebigen CAN-Knoten anfragen kann. Da der CAN-BUS allerdings ereignisgesteuert ist und daher mit dem Senden der Botschaften nicht warten muss, bis ein Remote-Frame diese angefragt hat, treten die Remote-Frames deutlich seltener auf als Data-Frames. Wird trotzdem ein Remote-Frame gesendet, versucht der BUS sofort mit dem geforderten ID zu antworten, jedoch unterliegt das Senden dieses Data-Frames wiederum der bitweisen Arbitrierung. Der Aufbau eines Remote-Frames ist baugleich zu den Data-Frames mit dem Unterschied, dass das Data-Field fehlt. Das wichtigste Unterscheidungsmerkmal liegt hier im RTR-Bit: Während dieses beim Data-Frame dominant übertragen wird, liegt beim Remote-Frame ein rezessiver Zustand vor (vgl. Tabelle 2.4) [10, 6].

Die Kommunikationsmatrix

Da die K-Matrix in der vorliegenden Arbeit eine entscheidende Rolle spielt, wird an dieser Stelle gesondert auf sie eingegangen.

Es liegen viele verschiedene Definitionen der K-Matrix vor. Einerseits wird sie häufig als eine Matrix beschrieben, in der geregelt ist, welche ECU welche Botschaft sendet und wiederum für welchen Knoten diese Botschaft interessant ist.

2. Grundlagen

Tabelle 2.6.: Beispielhafte K-Matrix, die beschreibt welcher Knoten welchen ID sendet und welcher Knoten welchen ID als Empfänger akzeptiert. Ein „/“ kennzeichnet, dass der Knoten die Nachricht weder sendet, noch annimmt [6].

ID	Knoten A	Knoten B	Knoten C	Knoten D
0x12	Sender	Empfänger	/	/
0x34	/	Sender	Empfänger	Empfänger
0x52	/	Empfänger	/	Sender
...

In diesem Fall ist es uninteressant, was in den einzelnen Botschaften steht, also welche Signale sie transportieren. Lediglich die IDs werden festgelegt, um zwischen hoch- und niederprioren Nachrichten unterscheiden zu können. Diese Struktur der K-Matrix ist in Tabelle 2.6 dargestellt. Der „0x...“-Operator vor den Zahlen bedeutet (wie auch ein nachgestelltes „h“), dass es sich hierbei um Zahlen im *Hexadezimal*-Format handelt, dessen Verwendung üblich für die Beschreibung von Anwendungen ist, die explizit mit einzelnen Bits arbeiten. So erspart man sich das Aufschreiben der wesentlich längeren Zahlen im *Binär*-Format, das jede *Dezimal*-Zahl (also die „normale“ Darstellung von Zahlen) nur durch Nullen und Einsen beschreibt.

Hardwareseitig wird die Relevanz einer Botschaft für einen Knoten folgendermaßen geregelt: Sobald der Frame empfangen wurde, wird er auf Korrektheit geprüft. Nachdem diese bestätigt wurde, wird die *Akzeptanzprüfung* durchgeführt, bei der der ID mit den Strukturen der K-Matrix verglichen wird. Stimmt der ID mit den IDs überein, die für diesen μ C als relevant angegeben sind, wird die Nachricht in den Empfangspuffer geschrieben. Stimmen die IDs nicht überein, wird die Botschaft verworfen und die Nutzdaten erreichen den Empfangspuffer nicht.

Da sich diese Arbeit allerdings mit den Botschaften und Signalen der BUS-Teilnehmer beschäftigt, bietet diese Definition der K-Matrix keinen zielführenden Ausgangspunkt. In anderen Beschreibungen wird sie um zusätzliche Informationen erweitert. Darunter fallen zum Beispiel welches Steuergerät welche Botschaft unter welchen Bedingungen und mit welcher Zykluszeit sendet und welche Signale in einer Botschaft enthalten sind. Damit ergibt sich eher eine Struktur wie in Tabelle 2.7 dargestellt.

Tabelle 2.7.: Beispielhafte K-Matrix, die um Informationen zu Zuständen, Botschaften, Signalen und weiteren Informationen zum Frame erweitert wurde. Als Ausgangszustand wird der Standard-Wert bei ruhigem BUS bezeichnet.

ID	0x1A2				
Sender	Knoten A				
Empfänger	Knoten C				
DLC	8				
Zykluszeit	0,1 s				
Ausgangszustand	A4h	00h	00h	A0h	...
Signale	Zähler	Warnblinker an/aus	/	Checksumme	...
		Blinker links an/aus			
		Blinker rechts an/aus			

Zu beachten ist hier, dass auf einem Byte mehrere Signale liegen können. Das Beispiel *Warnblinker an/aus* kann prinzipiell durch nur ein Bit dargestellt werden (beispielweise 01h bzw. 0000 0001 für *Warnblinker an* und 00h bzw. 0000 0000 für *Warnblinker aus*; in diesem Fall wird ausschließlich das nullte Bit verändert). Somit sind noch sieben Bits frei, in die man weitere Signale integrieren kann. Eine durchaus gängige Option ist es, alle Blinkerzustände auf ein Byte zu legen. Man könnte zum Beispiel 00h und 01h für den Zustand des Warnblinkers reservieren, 02h und 03h für den linken Blinker, 04h und 05h für den rechten Blinker usw., sodass man nur ein Byte benötigen würde, um die Zustände aller Blinker abzubilden. Andererseits gibt es auch Signale, die mehr Platz brauchen. Denkt man beispielsweise an den Fensterheber, so braucht man Signale für *hoch*, *runter*, *Fensterheber links*, *Fensterheber rechts*, usw., welche man auch auf einem Byte anbringen könnte. Darüber hinaus werden aber auch Signale benötigt, die einen Soll-Wert für den Drehwinkel des treibenden Motors definieren. Da man für eine entsprechende Regelung auch noch einen Ist-Wert benötigt, muss auch dieser in einem Signal mitgeteilt werden. Da es sich hier nicht bloß um zwei Zustände wie „an“ und „aus“ handelt, muss für diese Information unter Umständen ein ganzes Byte besetzt werden. Zum Beispiel kann aus der Position des treibenden Elektromotors die Stellung des Fensters bestimmt werden. Um die aktuelle Position des Motors darstellen zu können, müssen mehrere Umdrehungen auf acht Bits, also

2. Grundlagen

256 Werten, abgebildet werden. Für solche Signale gibt es in der K-Matrix auch einen „Schlüssel“, also einen zugehörigen Umrechnungsfaktor, mit dem man den Wert des Signals in den realen Winkel überführen kann.

2.3. Datenverarbeitung

Auf einer CAN-BUS-Leitung werden Signale mit bis zu 500 kBd gesendet. Dadurch entstehen sehr große Datenmengen, die hohe Ansprüche an Systeme haben, die versuchen Wissen aus ihnen zu extrahieren. Eine Lösung für dieses Problem bietet das sog. *Data-Mining*.

Die Fortschritte in der Mikrotechnik sind rasant und ebnen den Weg für immer bessere Speichermedien. Als Resultat wachsen die vorhandenen Datenmengen exponentiell, wodurch es dem Menschen fast unmöglich ist, den Überblick über die sich in den Datenbanken befindenden Informationen zu behalten. Das Ergebnis ist die Notwendigkeit einer automatisierten Datenverarbeitung, die schnell und sicher brauchbare Informationen aus unübersichtlichen Datenbanken zieht. Aus den englischen Begriffen Data (Daten) und Mining (graben) hat sich schnell ein treffender Begriff für den Prozess der Wissensgewinnung aus großen Datenbeständen entwickelt: das Data-Mining. Heute wird der Begriff vor allem in den Medien oft mit dem **Knowledge Discovery in Databases** (KDD) gleichgesetzt. Tatsächlich ist das Data-Mining aber nur ein Schritt des KDD-Prozesses: „ein mehrere Stufen umfassender Prozess, in dem Wissen aus gesammelten Daten gelernt beziehungsweise extrahiert wird.“ [17] Der KDD-Prozess wurde oft unterschiedlich definiert. Die am häufigsten zitierte Definition lautet: „Der KDD-Prozess ist ein nichttrivialer Prozess zur Identifikation gültiger, neuartiger, potentiell nützlicher und verständlicher Muster in Daten.“ [18]

Den Autoren dieser Definition zufolge unterteilt sich der KDD-Prozess in folgende Schritte:

1. Selection - Selektion
2. Preprocessing - Vorverarbeitung
3. Transformation
4. Data-Mining - Datenergründung
5. Interpretation/Evaluation

Bei der *Selection* geht es darum eine Teilmenge des gesamten Datenbestands zu bilden. Diese Teilmenge ist die erste grobe Gliederung und sollte möglichst so gewählt sein, dass keine relevanten Daten verloren gehen. Darauf folgt das *Preprocessing*. Dabei werden alle noch übrigen Daten zusammengeführt, indem die Datentypen angeglichen und somit Kompatibilitätsprobleme behoben werden. Im nächsten Schritt werden fehlerhafte Daten gelöscht und der Bestand von inkonsistenten Datenbeständen befreit. Danach geht es bei der *Transformation* darum, die Daten für das Data-Mining vorzubereiten. Das bedeutet sie werden so angeordnet, dass das Data-Mining schnell und effizient mit ihnen arbeiten kann. All diese Schritte können mit enormem Aufwand verbunden sein. Die eigentliche Analyse, also die tatsächliche Wissensgenerierung, findet allerdings im *Data-Mining* statt. Dieses ist wiederum in verschiedene Verfahren gegliedert. Die Basis bilden statistische Anwendungen, da das Data-Mining seine Ursprünge in der Statistik hat. Dank leistungsfähigerer Prozessoren sind heute Methoden weit verbreitet, die **Künstliche Neuronale Netze (KNN)** verwenden. Doch nicht jede Fragestellung ist mit solchen Methoden beantwortbar. Daher fallen auch Algorithmen unter das Data-Mining, die eigens für die vorhandene Problemstellung entwickelt wurden und nicht unbedingt auf statistischen Methoden oder KNN beruhen müssen.

Beim Data-Mining wird in erster Linie nach Mustern gesucht. Dabei unterscheidet man zwischen Gruppen (*Cluster*), Verbindungsmustern (*Link*), zeitlichen Mustern (*Sequences*), Abweichungen, Formeln und Gesetzmäßigkeiten. Des Weiteren wird zwischen *beschreibender* und *vorhersagender Analyse* unterschieden. Während sich die beschreibende Analyse mit Gruppen beschäftigt, interessiert sich die vorhersagende Analyse vor allem für zeitliche Muster, Regeln, Abhängigkeiten, Formeln

2. Grundlagen

und Gesetzmäßigkeiten [17].

An dieser Stelle wird explizit auf die Sequenzanalyse eingegangen, da vor allem diese in der vorliegenden Arbeit eine Rolle spielt. Der Sequenzanalyse liegt der Wunsch der Vorhersage zugrunde. Man betrachtet zeitliche Zusammenhänge und versucht durch solche zeitlichen Muster häufig wiederkehrende Abfolgen in den Daten vorherzusagen. Die Sequenzanalyse beschäftigt sich also mit der Frage, was in der Zukunft passieren wird. Dies wird gerne für Prognosen in der Wirtschaft genutzt, wo es zum Beispiel interessant ist, wie sich ein Währungskurs entwickelt [18]. Aber auch ein gegensätzlicher Verlauf lässt sich beurteilen. Tritt zum Beispiel ein Ereignis ein, das nichts mit dem erwarteten Ereignis zu tun hat, kann dies auch erkannt werden. Auf diese Arbeit angewandt, bedeutet das, dass man nach Ereignissen Ausschau halten muss, die andersartig sind, als die in der Vergangenheit. Zum Schluss folgt der letzte Schritt: die *Interpretation/Evaluation*. Hierbei müssen die gefundenen Daten vom Benutzer interpretiert und somit in Wissen umgewandelt werden. Diese Aufgabe kann ihm der Computer, der lediglich die nötigen Informationen zusammenfügen kann, nicht abnehmen. Die Interpretation stellt den wichtigsten Schritt dar und wird immer leichter, je gewissenhafter die vorherigen Schritte durchgeführt wurden.

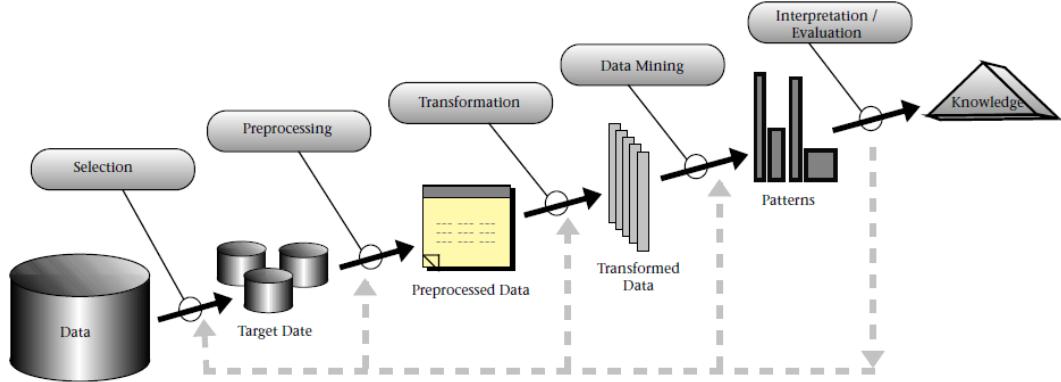


Abbildung 2.5.: Ein Überblick über die Schritte im Knowledge-Discovery-in-Databases-Prozesses, um aus einem Datenbestand Wissen zu generieren. Der Prozess startet links mit allen Daten (*Data*) und endet rechts mit dem gewonnenen Wissen (*Knowledge*) [18].

2.4. Hilfsmittel

In diesem Abschnitt sollen kurz die benutzten Hilfsmittel beschrieben werden. Außerdem wird erläutert, warum gerade diese benutzt wurden.

BUS-Interface: CANcase VN1630

Das *CANcase VN1630* der Firma *Vector Informatik GmbH* wird am Institut für Energieeffiziente Mobilität (IEEM) gerne als Verbindung zwischen BUS und Laptop genutzt. Neben zwei Schnittstellen für CAN und LIN bietet es die Möglichkeit der Geschwindigkeitsregulation, wodurch es möglich ist, auf Busse mit verschiedenen Übertragungsraten zuzugreifen. Darüber hinaus verfügt das Gerät über einen Anschluss zur externen Spannungsversorgung. Das CANcase wird über einen USB 2.0-Anschluss an den Rechner angeschlossen. Außerdem verfügt es über eine eigene Aufzeichnungsfunktion. Da es von der selben Firma wie die verwendete Messsoftware stammt, sind Hard- und Software optimal aufeinander abgestimmt. Das CANcase VN1630 ist eine relativ kostengünstige Lösung mit der flexibel gearbeitet werden kann. Es erfüllt alle Anforderungen, die für diese Arbeit gestellt wurden. Aus diesen Gründen wurde es für alle Messungen genutzt.

Messsoftware: CANoe

Ebenfalls von der Firma Vector kommt das Softwaretool *CANoe*. Es dient in erster Linie der Entwicklung und dem Testen von Steuergeräten und ist in der Lage den gesamten Busverkehr übersichtlich darzustellen. Dank der integrierten Programmiersprache **Communication Access Programming Language (CAPL)** kann es den kompletten Verkehr oder gezielt einzelne Botschaften analysieren. Daneben gibt es die Funktion, selbst Signale auf den BUS zu senden. Die wichtigste Funktion für die vorliegende Arbeit ist aber das Aufzeichnen ganzer Datenströme und das Exportieren in verschiedene Formate. Verwendet wurde die Version 8.2.98 (SP4). CANoe ist ein viel verwendetes Softwaretool, das in Kombination mit dem bereits beschriebenen CANcase eine einfache Lösung darstellt, alle für diese Arbeit benötigten Daten zu erfassen. Aus diesen Gründen wurde CANoe als Messsoftware benutzt.

2. Grundlagen

Datenformate umwandeln: Excel

Die allseits bekannte Berechnungssoftware *Excel* der *Microsoft Corporation* ist in der Lage auch große Datensätze in Matrizenform zu speichern und somit auch umzuwandeln. Die Version *2016* bietet eine komfortable Umwandlung von Daten, indem sie es ermöglicht Zellen zu trennen, einzelne Zeilen und/oder Spalten zu löschen und die bereinigten Daten in einem anderen Format zu speichern.

Aus diesen Gründen wurde es als Zwischenschritt zur Datenaufbereitung verwendet.

Datenverarbeitung: Matlab

Matlab der Firma *The Mathworks, Inc.* ist eine beliebte Software zur Berechnung von Matrizen. Sie bietet zahlreiche Toolboxen und eine C-basierte Programmiersprache. Die Möglichkeit, eigene Funktionen zu schreiben, bot die Grundlage der Datenverarbeitung für diese Thesis. Außerdem ist es möglich, den erzeugten Code mit Matlab-eigenen Befehlen als *C/C++*-Code zu exportieren und so auch ohne Matlab-Lizenz leserlich zu machen. Verwendet wurde Version *R2016a*. Die benötigte Lizenz wird ebenfalls von der HsKA bereitgestellt.

Als gängige und allseits bekannte Software mit vielen Funktionen, die die Arbeit vereinfachen, war Matlab eine gute Lösung zur Bearbeitung der Aufgabenstellung.

3. Entwicklung der Methodik

Das nachfolgende Kapitel stellt die Maßnahmen dar, die nötig waren, um die gestellte Aufgabe zu bewältigen. Dazu wird zunächst die Ausgangssituation geschildert. Danach wird die verwendete K-Matrix vorgestellt und es werden die einzelnen durchgeführten Schritte erläutert und erklärt, warum diese notwendig waren. Auf den Algorithmus zur Auswertung der Daten wird in Kapitel 4 eingegangen.

3.1. Ausgangssituation

Als Untersuchungsobjekte für die vorliegende Arbeit dienten die Fahrzeuge des IEEM. Dieses bietet einen *Mercedes CL550* und einen *BMW i3*, was den Vorteil hat, dass die K-Matrix unabhängig von verschiedenen Standards herausgefunden werden kann. An beiden Pkw waren bereits Buszugriffe vorhanden. Dazu wurde beim BMW i3 die Rückbank entfernt und ein CAN-Knoten ausgebaut. Nachdem die Transceiver freigelegt wurden, wurde bestimmt, um welche Art Transceiver es sich hier handelt. Das Datenblatt der *TJA 1051/3*-Transceiver der Firma *NXP Semiconductors N.V.* ist öffentlich im Internet verfügbar (vgl. Anhang A.1). Darin war ersichtlich, auf welchen Pins die Leitungen für CANH bzw. CANL lagen. Auf diese Pins wurden Verbindungen gelötet und so die beiden Pegel durch einen *D-SUB DE-9*-Stecker abgegriffen (vgl. auch [19]). Diese Schnittstelle ist kompatibel mit dem CANcase und sorgt somit dafür, dass die Signale im weiteren Verlauf für den Rechner verwertbar gemacht werden.

Die vorliegenden Bussignale richtig zu deuten, ist heutzutage unabdingbar. Dazu gibt es sog. *Diagnosegeräte*, die an die **On Board Diagnose (OBD)**-Buchse des Fahrzeugs angeschlossen werden und so zum Beispiel den Fehlerspeicher auslesen können. Allerdings sind diese oft herstellerspezifisch und teuer in der Anschaffung. Dafür bieten sie eine schnelle und unkomplizierte Lösung für die OEM-eigenen

3. Entwicklung der Methodik

Fahrzeuge. Damit steht fest, dass dies keine universelle Lösung darstellt.

3.2. Verwendete Kommunikationsmatrix

Wie in Unterabschnitt 2.6 aufgezeigt, gibt es für die K-Matrix einige unterschiedliche Definitionen. Diese beinhalten oftmals Informationen darüber, wer der Sender ist und wer die Empfänger einer Botschaft sind. Diese Spezifikationen zu ermitteln ist allerdings nur möglich, indem man sich jeweils an die Knoten des Busses anschließt. Die vorliegende Arbeit hingegen, möchte eine reine Softwarelösung für die Ermittlung der K-Matrix bieten. Aus diesem Grund wird darauf verzichtet, die Sender und Empfänger der einzelnen Botschaften herauszufinden. Viel mehr wird sich auf die gesendeten Signale und weitere Informationen zu den IDs konzentriert. Darunter fallen Eigenschaften wie der DLC, die Zykluszeit, die Bytes und Bits, die sich ändern, die Wertänderungen der einzelnen Bytes und etwaige weitere Eigenschaften. Somit ergibt sich für die hier verwendete K-Matrix eine Struktur wie in Tabelle 3.1 abgebildet.

Tabelle 3.1.: Beispielhaft verwendete Kommunikationsmatrix: Das Signal mit dem ID 322, bei dem sich der Wert von Byte drei von null auf 24 erhöht (entspricht einer Veränderung von Bit drei und vier), wird mit dem DLC acht mit einer Zykluszeit von $0,1\text{ s}$ gesendet und sorgt dafür, dass der Warnblinker angeschaltet wird.

Funktion	Warnblinker anschalten
ID	322
Byte	3
Bit(s)	3,4
Wertänderung von	0
nach	24
Zykluszeit	$0,1\text{ s}$
DLC	8

3.3. Manuelle Analyse

Der erste Schritt zu einer automatisierten Analyse ist eine manuelle Analyse. Diese ist essentiell, wenn auch sehr aufwendig, um herauszufinden, wie sich das System verhält.

Wie in Abschnitt 3.1 aufgezeigt, wurde die Basis zur Auswertung von Signalen bereits gelegt, indem die BUS-Schnittstelle hergestellt wurde. Somit muss für eine erste manuelle Analyse nur noch das CANcase zwischen BUS und Laptop (mit Hilfe einer **Universal Serial Bus (USB)**-Verbindung) geschaltet werden. Daraufhin zeigte eine schnelle Auswertung, dass der sog. *Body-CAN* mit 500 kBd Übertragungsrate arbeitet. Der Body-CAN eignet sich sehr gut als Test-CAN, da auf diesem Signale liegen, die man leicht aus dem Innenraum betätigen kann. Dazu gehören zum Beispiel der Warnblinker, die Handbremse, der Fensterheber und Weitere. An dieser Stelle ist zu bemerken, dass dieses Vorwissen zwar vorhanden war, aber für die Bearbeitung des Themas nicht relevant ist. Der Body-CAN wurde verwendet, da diese Schnittstelle bereits existierte und hat sich im Nachhinein als gute Wahl herausgestellt.

Nachdem die entsprechenden Einstellungen getätigt wurden, wurde die CANoe-Oberfläche den Bedürfnissen angepasst. Wichtig war hierbei vor allem das sog. *Trace-Fenster*, in dem alle Botschaften des angezapften Busses angezeigt werden. In Tabelle 3.2 sind die benutzten Anzeigeelemente kurz erläutert.

Tabelle 3.2.: Anordnung des Trace-Fensters in CANoe.

Time	Zeit ab Beginn der Messung
ID	Identifier der Botschaft (vgl. Tabelle 2.4)
DLC	Data Length Code der Botschaft (vgl. Tabelle 2.4)
Data	Die Datenbytes der Botschaft
Counter	Zählt, wie oft der ID bereits gesendet wurde

Im Anschluss konnten die ersten manuellen Versuche getätigt werden. Diese wurden zunächst relativ willkürlich durchgeführt, um ein Gefühl für das System zu bekommen. Bei laufender Messsoftware wurden verschiedene Signale auf den BUS gesendet, indem beispielsweise im Fahrzeug der Blinker oder der Fensterheber betätigt wurden. Dabei wurde das Trace-Fenster von CANoe beobachtet und versucht herauszufinden, welche Botschaften sich änderten. Hat man eine sich

3. Entwicklung der Methodik

ändernde Botschaft entdeckt, wurde versucht die gleichen Signale noch einmal zu senden, um zu verifizieren, dass man den richtigen ID gefunden hat. So wurde versucht, einige Signale händisch zu identifizieren. Eine große Hilfe war dabei die Funktion von CANoe, dass Botschaften, die entweder immer das Gleiche senden oder das Senden vorübergehend eingestellt haben, grau hinterlegt werden. Dies ist in Abbildung 3.1 ersichtlich.

Time	ID	DLC	Data	Coun...
14.342511	165	8	180 241 255 247 127 0 0 193	717
14.268261	192	2	248 32	72
14.342753	217	8	248 241 0 16 0 240 127 192	717
12.695387	236	8	255 255 255 255 8 15 252 255	3

Abbildung 3.1.: Ausschnitt des Trace-Fensters des Body-CANs des BMW i3: Der ID 236 wurde nach 12,695387 s nach Messungsbeginn zuletzt gesendet, der ID 165, dessen letzte sechs Bytes sich längere Zeit nicht mehr ändern, hat einen DLC von acht und wurde seit Messstart bereits 717 mal gesendet.

Auf dieser Basis konnten erste Vermutungen getroffen werden, welcher ID für welches Signal relevant ist. Abbildung 3.2 zeigt den entsprechenden Versuchsaufbau, Tabelle 3.3 die ersten Vermutungen. Die Darstellung der IDs (und aller anderen Eigenschaften) im Dezimal-Format ist zwar eher unüblich, wurde aber dennoch gewählt, da die Verarbeitung mit Matlab dadurch mit weniger Schwierigkeiten verbunden war.

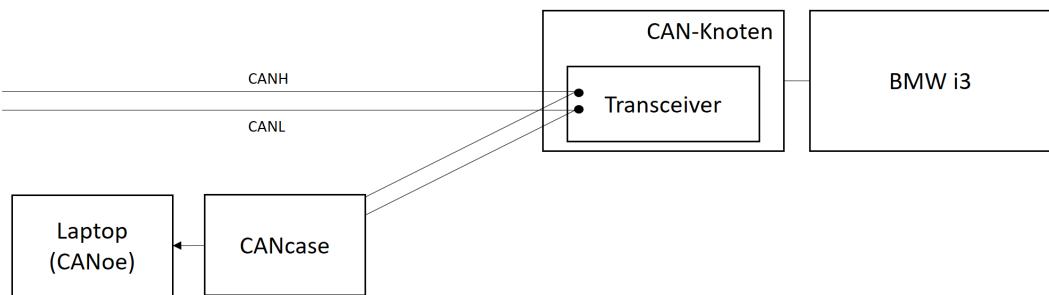


Abbildung 3.2.: Versuchsaufbau zur Datenerhebung: Das CANH- und CANL-Signal wird am Transceiver des CAN-Knotens vom CANcase abgegriffen und durch eine USB-Verbindung zum Laptop weitergeleitet.

Tabelle 3.3.: Auswahl vermuteter IDs nach manueller Analyse.

Aktion	Vermuteter ID
Warnblinker an/aus	502
Handbremse ziehen/lösen	732
Fensterheber rechts betätigen	952
Fensterheber links betätigen	950

Bei den vermuteten IDs fehlen natürlich noch einige Informationen wie zum Beispiel die Bytes und Bits, die sich ändern. Das herauszufinden ist aber mit wesentlich mehr Aufwand verbunden und kann nicht einfach im Trace-Fenster abgelesen werden, da es durch Latenzzeiten zu nicht dargestellten Informationen kommen kann, obwohl sie trotzdem intern verarbeitet werden. Dazu kommt, dass die Wechsel so schnell durchgeführt werden, dass sie mit bloßem Auge nicht verfolgbar sind. Der DLC eines IDs dagegen kann dank CANoe relativ einfach bestimmt werden, da dieser in einer eigenen Spalte angezeigt wird und für den jeweiligen ID in aller Regel konstant bleibt.

3.4. Automatisierung der Analyse

Im nächsten Schritt wurde die manuelle Analyse automatisiert. Die Idee dahinter ist die gleiche wie bei der manuellen Analyse: Ändert sich eine Botschaft im selben Moment, in dem ein Schalter im Fahrzeug benutzt wird, ist die Wahrscheinlichkeit hoch, dass ein Zusammenhang besteht. Natürlich steigt die Wahrscheinlichkeit, den richtigen ID zu finden, stark an, wenn man weitere Bewertungskriterien einführt. Diese werden in Unterabschnitt 4.1.4 genauer spezifiziert.

Im Folgenden soll erklärt werden, welche Schritte nötig sind, um eine automatisierte Auswertung der BUS-Daten zu erhalten. Der zugehörige Programmablaufplan ist in Abbildung 3.3 gegeben.

3. Entwicklung der Methodik

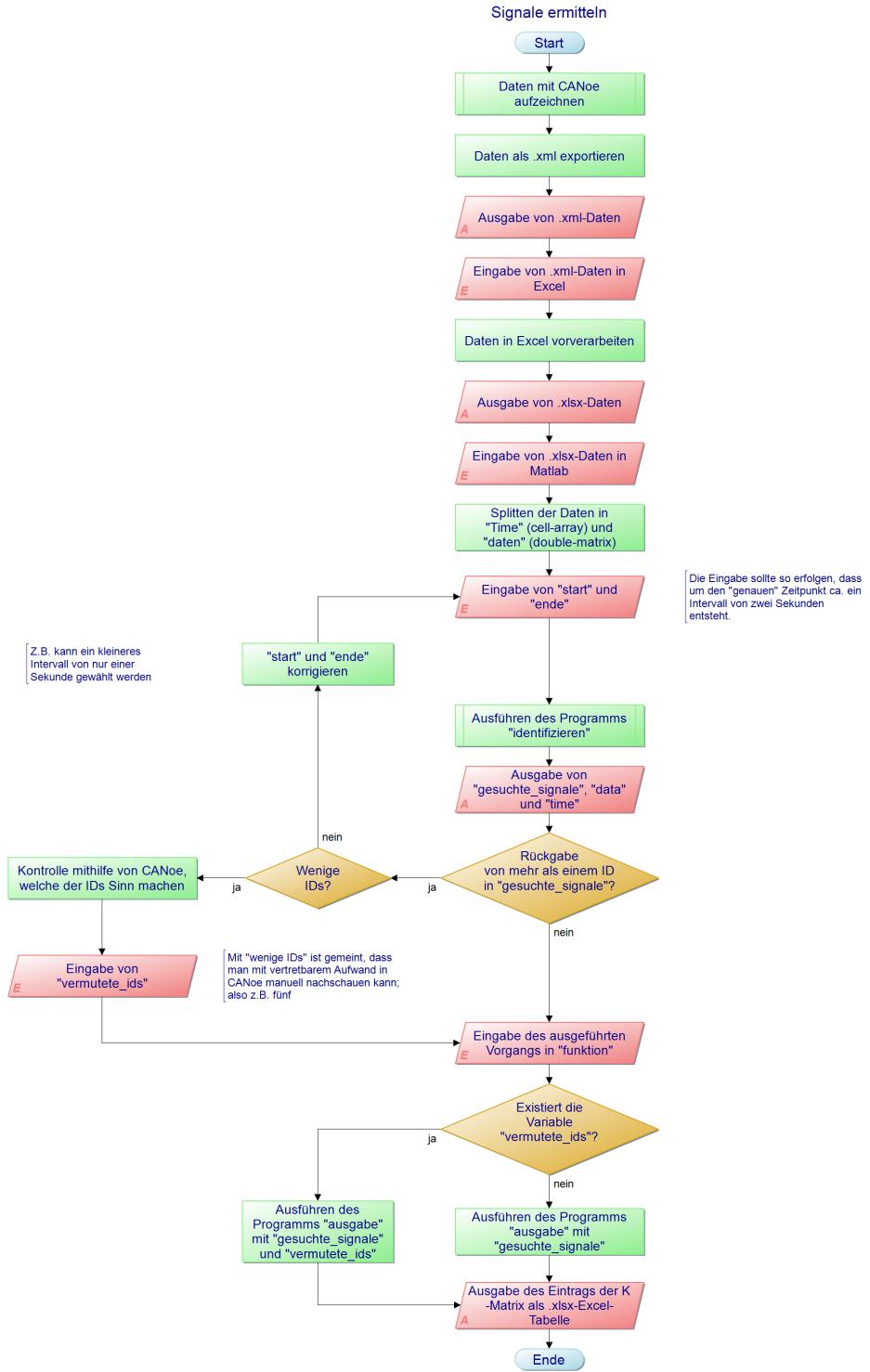


Abbildung 3.3.: Programmablaufplan zur Ermittlung von Signalen mit nur einem Datensatz.

Im ersten Schritt ist es natürlich notwendig die Daten aufzuzeichnen. Dazu sollte der BUS möglichst „ruhig“ sein, es sollte also wenig *Traffic* (von engl. *Verkehr*; also die Summe der Nachrichten, die auf dem BUS liegen) vorhanden sein. Nachdem alles korrekt verbunden und eingestellt wurde, kann man die Messung beginnen. Nun sollte man ca. 15 Sekunden warten, damit der CAN „hochfahren“ kann. Das bedeutet, dass die ECUs aus dem Schlafmodus reaktiviert werden und der größte Teil der Nachrichten, die bei jedem Benutzen des Busses gesendet werden, mindestens einmal auf den BUS gelegt wurde. Dies entscheidet im späteren Verlauf maßgeblich über Erfolg und Misserfolg der Analyse. Sind 15 Sekunden vergangen, muss die geplante Aktion ausgeführt werden. Hier soll das Beispiel des Warnblinkers wieder aufgegriffen werden. Wird die Aktion also als „Warnblinker anschalten“ definiert, muss dieser vorab natürlich ausgeschaltet sein. Nach den 15 Sekunden wird nun einmal der Knopf für den Warnblinker gedrückt. Im Anschluss werden weitere 15 Sekunden gewartet, sodass die Messung insgesamt ca. 30 Sekunden dauert.

Die gewonnenen Daten können dann über das Trace-Fenster von CANoe exportiert werden. Dafür bietet sich das *xml*-Format an, da es eine übersichtliche Datenmatrix erzeugt, die gut weiterverarbeitet werden kann. Ein solcher Ausschnitt ist in Abbildung 3.4 dargestellt.

Time	ID	DLC	Data	Counter
0.003406	788	3	17 1 255	1
0.004327	165	8	232 254 255 247 127 0 0 193	1
0.004781	217	8	164 254 0 16 0 240 127 192	1

Abbildung 3.4.: Ausschnitt einer von CANoe erzeugten xml-Datei: Alle Botschaften, die während der Aufzeichnung gesendet wurden, werden hier in chronologischer Reihenfolge dargestellt.

Danach müssen sie noch etwas modifiziert werden. Dies geschieht, indem sie in Excel importiert und dort bearbeitet werden. Dort wird die richtige Reihenfolge der Messdaten eingestellt und es wird dafür gesorgt, dass jedes Datenbyte in eine eigene Zelle geschrieben wird (vgl. Unterabschnitt 4.1.3). Nur so ist ein sinnvoller Import in Matlab möglich.

Nachdem der Datensatz im *xlsx*-Format gespeichert wurde, wird er in Matlab

3. Entwicklung der Methodik

importiert. Dabei wird er außerdem in einen *cell*-Array¹ namens „Time“ und eine *double*-Matrix² namens „daten“ auseinandergezogen. Die Namen können dabei frei gewählt werden, aber die nachfolgenden Funktionen arbeiten mit den vorgeschlagenen Bezeichnungen, weshalb diese Namensgebung sinnvoll ist. Das Matlab-interne Format für *Workspaces*³ und *Scripts*⁴ wird als *mat* bezeichnet.

Dann ist die manuelle Eingabe der Variablen „start“ und „ende“ erforderlich. Dabei handelt es sich um den Start- und Endzeitpunkt des Zeitintervalls, in dem der Warnblinker betätigt wurde. Da er nach ca. 15 s gedrückt wurde, wird „start“ in diesem Beispiel als 14 und „ende“ als 16 deklariert. Wichtig ist dabei, dass das Zeitintervall nicht zu groß ist. Allerdings kann auch ein Endzeitpunkt, der zu nah an dem realen Zeitpunkt der Betätigung liegt, zu schlechteren Ergebnissen führen, weil es möglich ist, dass aufgrund von beispielsweise Latenzzeiten noch gar kein entsprechendes Signal gesendet wurde. Das zweisekündige Intervall zwischen 14 und 16 erfüllt beide Voraussetzungen. Nach korrekter Eingabe findet man die Variablen im Workspace wieder, wie in Abbildung 3.5 ersichtlich.

Workspace	
Name	Value
daten	14345x11 double
ende	16
start	14
Time	14345x1 cell

Abbildung 3.5.: Workspace nach korrekter Variablen-deklaration: Die Matrix „daten“ hat immer elf Spalten und genau so viele Zeilen wie der Vektor „Time“.

Die komplette Auswertung erfolgt dann automatisch, sobald die Funktion „identifizieren“ aufgerufen wurde. Die wichtigsten Teile des Algorithmus werden in Kapitel 4 erläutert. Hier soll nur erwähnt werden, dass die Funktion als Rückgabe die Variable „gesuchte_signale“ definiert.

Ist die Auswertung abgeschlossen, muss kontrolliert werden, wie viele IDs als Rückgabe definiert wurden. Wurde mehr als ein ID gefunden, muss die Frage gestellt werden, ob das Ergebnis verwertbar ist. Das heißt, es muss kontrolliert

¹ein Datentyp von Matlab, der benutzt wird, wenn ein Vektor verschiedene Datentypen enthält

²ein Datentyp, der auch Kommazahlen darstellen kann

³von engl. *Arbeitsspeicher*; alle Variablen, die zu einem bestimmten Zeitpunkt definiert sind

⁴von engl. *Skripte*; eine Folge von Befehlen

werden, wie viele IDs ermittelt wurden. Ist die Anzahl zu groß, um händisch mit Hilfe von CANoe ausgewertet zu werden, sollte der Start- und Endzeitpunkt neu definiert werden. Dabei sollte zunächst versucht werden, „start“ möglichst nah vor dem tatsächlichen Startzeitpunkt zu definieren. Bringt dies noch immer nicht den erhofften Erfolg, kann auch das Ende des Intervalls näher an den realen Zeitpunkt gerückt werden. Sind es nur wenige IDs, die zurückgegeben wurden, ist der Richtige schnell im Trace-Fenster von CANoe gefunden, indem man eine neue Messung startet und sich nur diese IDs anzeigen lässt. Führt all dies zu keinem vernünftigen Ergebnis, kann auch ein neuer Datensatz zu genaueren Angaben führen. Wurden jedoch passende IDs unter den gefundenen IDs entdeckt, werden diese in dem Vektor „vermutete.ids“ definiert. Ist die Lösung hingegen eindeutig, muss dieser Vektor nicht definiert werden.

Die letzte manuelle Definition ist die Bezeichnung des ausgeführten Vorgangs. In den *char-Array*⁵ „funktion“ würde man in diesem Beispiel also „Warnblinker anschalten“ schreiben.

Im letzten Schritt muss nur noch die Ausgabe durchgeführt werden. Dabei kann man die dafür benötigte Funktion „ausgabe“ mit oder ohne die Variable „vermutete_ids“ ausführen. Wird sie ohne diese Variable ausgeführt, schreibt sie alle Daten von „gesuchte_signale“ in die angegebene Excel-Tabelle, andernfalls nur die Signale der vermuteten IDs. Die Matrix „gesuchte_signale“ und der Array „funktion“ werden hingegen bei jedem Aufruf benötigt. Die Ergebnistabelle für dieses Beispiel ist in Abbildung 3.6 dargestellt.

Funktion	ID	Byte	Bit	Von	Nach	Zykluszeit	DLC	Zaehler?
Warnblinker anschalten	322	3		0	24		0,1	8 0

Abbildung 3.6.: Ergebnistabelle für das Beispiel „Warnblinker anschalten“ (vgl. Tabelle 3.1): Die Funktion „ausgabe“ schreibt das gefundene Signal automatisch in die vorher benannte Excel-Tabelle und ergänzt diese, falls es sich nicht um den ersten Eintrag handelt.

⁵ein Datentyp, der auch Buchstaben darstellen kann

3.5. Korrektes Interpretieren der gefundenen Signale

Obwohl das Programm dem Nutzer die meiste Arbeit abnimmt, liegt die richtige Interpretation am Ende in seiner Verantwortung. Ein Beispiel zur Erklärung ist das flexible Dach eines Cabrios. Zum Schutz der Insassen sind für ein Cabrio sog. *sichere Zustände* definiert. Diese sind gegeben, wenn das Dach entweder vollständig geschlossen und verankert, oder wenn das Dach vollständig zurückgefahren und der Überrollsitzschutz ausgefahren ist. Alle Zustände dazwischen sind bei einem Unfall mit einem erhöhten Verletzungsrisiko für Fahrer und Beifahrer verbunden. Es ist klar, dass diese verschiedenen Zustände auch auf dem BUS abgebildet sein müssen. Ist man also mit geschlossenem Dach unterwegs, liegt ein sicherer Zustand vor, der beispielsweise mit einem Wert von 0 in einer Botschaft dargestellt wird. Betätigt man nun den Knopf für das Öffnen des Daches, muss sich dieser Zustand ändern. Da das Dach nun aber nicht sofort vollständig zurückgefahren und der Überrollbügel ausgefahren ist, liegt eine gewisse Zeit kein sicherer Zustand vor. Würde man den zweiten sicheren Zustand, in dem das Dach geöffnet ist, mit einer 2 auf dem selben Byte, wie zuvor definiert, müsste noch der nicht sichere Zustand zwischen 0 und 2 abgebildet werden. Ein naheliegender Wert wäre die 1 auf dem selben Byte.

Führt man nun also die Aktion „Dach öffnen“ aus und lässt diese von den Funktionen analysieren, bekommt man als Rückgabewert, dass sich der Wert des entsprechenden Bytes beim Öffnen des Daches von 0 auf 1 ändert. Wie eben beschrieben, ist dies keine Garantie dafür, dass der Wert für „vollständig zurückgefahren und Überrollbügel ausgefahren“ 1 ist. Für ein besseres Ergebnis sollte also immer noch versucht werden, ein komplementäres Ereignis zu erzeugen. In diesem Fall wäre dies das Schließen des Daches. Somit bekommt man für das Öffnen einen Wechsel von 0 auf 1 und für das Schließen einen Wechsel von 2 auf 1. Damit wäre klar, dass die beiden Sicherer Zustände mit 0 und 2 definiert sind und der unsichere Zustand dazwischen mit einer 1 abgebildet wird.

Des Weiteren ist für manche Signale ein tiefer gehender Verständnis der BUS-Kommunikation sehr hilfreich. Dazu soll eine elektrische Sitzverstellung als Beispiel dienen. Möchte man den Sitz nach vorne verstellen, betätigt man den dafür

vorgesehenen Knopf. Intern wird dadurch im Grunde der Wunsch geäußert, dass der Sitz nach vorne fahren soll. Dieser Wunsch muss in einem Soll-Wert versendet werden. Damit der Sitz quasi stufenlos verstellt werden kann, ist eine möglichst hohe Auflösung dieses Wertes notwendig. Das bedeutet, dass der Soll-Wert beispielsweise ein ganzes Byte besetzt, um 256 verschiedene Positionen abbilden zu können. Um den Sitz auf die richtige Position regeln zu können, ist auch ein Ist-Wert nötig, für den die gleiche Auflösung wie für den Soll-Wert gewählt werden sollte, da der Sitz natürlich nicht sofort an der geforderten Stelle ist. Darüber hinaus kann es wichtig sein, auch noch abzubilden, ob der Sitz sich schnell bewegen soll, um größere Distanzen schneller zurückzulegen, oder ob er sich langsam bewegen soll. Dies macht Sinn, um die gewünschte Position möglichst genau zu erreichen, da man diese bei zu hoher Geschwindigkeit verfeheln könnte. Dazu müsste dann noch ein drittes Byte belegt werden. Es ist also sehr gut möglich, dass für nur einen Vorgang mehrere Bytes verändert werden.

Zu beachten ist auch, dass die Algorithmen zunächst nicht zwischen zyklisch und einfach versendeten Botschaften unterscheiden. Da allerdings für jede Botschaft eine Zykluszeit errechnet wird, muss überlegt werden, ob es sich um die eine oder andere Art von Botschaften handelt. Zu bedenken ist, dass die Zykluszeiten normalerweise nicht den höheren Sekundenbereich erreichen und dass es sich bei solchen Rückgabewerten nicht unbedingt um eine zyklische Botschaft handelt. Allerdings gibt es dafür keine Standards, wodurch der OEM durchaus eine solche Zykluszeit definieren könnte. Deshalb wird sie im Code trotzdem ausgewertet.

4. Implementierung

Das folgende Kapitel beschäftigt sich mit der Implementierung der benötigten Funktionen. Das Hauptaugenmerk liegt dabei auf den Funktionen „identifizieren“ und „identifizieren_mit_vorwissen“ welche den Kern der Analyse darstellen. Des Weiteren werden Teile von ausgewählten Unterfunktionen erklärt.

4.1. Datenverarbeitung ohne Vorwissen

Die Funktion „identifizieren“ kann als *main*-Funktion¹ verstanden werden. Sie ruft alle Unterfunktionen auf und bietet selbst nur wenig Logik. Ein vollständiger Programmablaufplan (Pap) befindet sich in Anhang A.3.

Wie in Abschnitt 2.3 beschrieben, teilt sich die Datenverarbeitung in fünf Schritte auf: Selection, Preprocessing, Transformation, Data-Mining und Interpretation/Evaluation. Diese Gliederung wurde bei der Wissensgenerierung aus den Datensätzen befolgt und dient deshalb auch als Unterteilung dieses Kapitels.

4.1.1. Selection - Selektion

Die erhobenen Datensätze werden im Laufe der Codeabwicklung mehr als einmal selektiert. Die erste Möglichkeit dazu besteht bereits beim Export, wo es möglich ist, nur bestimmte IDs zu exportieren bzw. bestimmte IDs nicht zu berücksichtigen. Dies ist vor allem dann interessant, wenn schon Vorwissen existiert, durch das manche IDs als nicht relevant eingestuft werden können (vgl. Abschnitt 4.2). Da hier allerdings davon ausgegangen wird, dass keinerlei spezielle Kenntnisse zu diesem CAN vorliegen, werden immer alle IDs exportiert.

¹ von engl. *hauptsächlich, wichtigste*

4. Implementierung

Die erste Selektion, die immer stattfindet, erfolgt in der Funktion „data_einzeln_berechnen“. Sie sorgt dafür, dass alle doppelten Botschaften aus dem Datensatz entfernt werden. Zudem werden alle vorhandenen IDs ausgelesen und es wird bestimmt, wann die einzelnen Zustände zum ersten mal auftauchen. Da hierbei jedoch potentiell interessante Daten verloren gehen, wird der reduzierte Datensatz neben den originalen Daten in einer neuen Variablen gespeichert. Die aus dem reduzierten Datensatz bestimmten vorhandenen IDs spielen im Nachfolgenden noch eine wichtige Rolle. Mit diesen wird nämlich nun im Originaldatensatz bestimmt, wie oft jeder Zustand inklusive ID vorkommt. Kommt ein Zustand nur genau einmal vor, bedeutet das, dass der ID eine sich nie ändernde Botschaft sendet. Über die Gesamtheit des Datensatzes heißt dies eindeutig, dass dieser ID nichts mit dem manuell gesendeten Signal zu tun, da hierbei ein deutlich erkennbarer Flankenwechsel auftreten muss. Somit können alle Zustände gelöscht werden, die sich nicht verändern. Dies geschieht in der Funktion „konstante_botschaften_loeschen“, nachdem die entsprechenden IDs aus dem Array mit den vorhandenen IDs gelöscht wurden und als „dezimierte_ids“ an die Funktion übergeben wurden. Außerdem löscht sie die zugehörigen Zeitschritte im Zeitvektor, sodass dieser keine Werte mehr enthält, die zu einer konstanten Botschaft gehören. So wird darüber hinaus garantiert, dass der Zeitvektor die selbe Länge hat, wie der dezimierte Datensatz. Die signifikanteste Selektion findet jedoch später statt, sobald die wahrscheinlichsten IDs berechnet wurden (vgl. Unterabschnitt 4.1.4). In der Funktion „einzel_ids_berechnen“ wird der gesamte Datensatz auf die Zustände reduziert, die mit den IDs gesendet wurden, die in den Variablen „wahrscheinlichste_ids“ stehen. Dies wird außerdem für die zweit- und dritt-wahrscheinlichsten IDs und für jede errechnete Auswahl (die Berechnung der hierfür nötigen Variablen wird im ersten Teil des Data-Mining in Unterabschnitt 4.1.4 beschrieben) durchgeführt. So entstehen stark dezimierte Datenmengen, die im zweiten Teil des Data-Mining schnell analysiert werden können.

4.1.2. Preprocessing - Vorverarbeitung

Für rechnerinterne Prozesse ist es entscheidend, alle Daten in einem einheitlichen Datentyp vorliegen zu haben. Genau dafür sorgt das Preprocessing.

Wie in Abschnitt 3.4 beschrieben, werden die Daten immer als $n \times 11$ -Matrix

ingelesen. Für einen besseren Überblick wird diese zunächst transponiert². Der Zeitvektor bedarf allerdings tiefgreifender Veränderungen, da er beim Exportieren aus Excel in einer unbrauchbaren Form in Matlab importiert wird. Dies ist das Resultat der Tatsache, dass Excel die Zahlen ≥ 1 nicht als 1.000001, sondern als 1000001 ausgibt. Die Zahlen $\in [0, 1[$ ³ dagegen werden als *characters*, zum Beispiel als '0.999999' dargestellt. Der dafür von Matlab benutzte Datentyp „cell“ ist nicht berechenbar, da er für Vektoren mit wechselnden Datentypen verwendet kann und somit gar nicht in der Lage ist, verwertbare Informationen zu speichern. Er ist also nur eine „Abbildung“ der im richtigen Format enthaltenen Informationen. Um brauchbare Matrizen zu erstellen, bietet Matlab für diesen Fall verschiedene interne Funktionen: *str2double*, um den im *cell* enthaltenen *string* in einen *double* umzuwandeln (dies ist im Fall '0.999999' nötig) und *cell2mat*, durch die der *cell* ebenfalls in einen *double* umgewandelt wird (dies ist im Fall 1000001 nötig). Bei der letzteren Operation ist es im Nachgang nur noch wichtig, den erhaltenen Wert durch 1000000.0 zu dividieren⁴, um den gewünschten Wert zu erhalten. Für die Entscheidung, welche Umwandlung verwendet werden muss, bietet Matlab eine weitere Funktion: *iscellstr*. Sie gibt eine 1 zurück, wenn in der *cell*-Variablen ein *string* geschrieben steht und kontrolliert innerhalb einer *for*-Schleife die gesamte Länge des eingegebenen Zeitvektors. So wird automatisch der Umbruch von '0.999999' auf 1000000 detektiert und entsprechend bearbeitet. Als Rückgabe wird eine neue Variable namens „time“ erstellt, die alle Zeitpunkte im richtigen Format beinhaltet.

4.1.3. Transformation

Die Transformation fand in der vorliegenden Arbeit gleich zu Anfang statt. Sie beinhaltet alle Schritte, die dafür sorgen, den verarbeitenden Programmen die richtigen Datenformate zu liefern. Wie in Abschnitt 3.4 geschildert, ist dies das *mat*-Format, mit dem Matlab in der Regel arbeitet. Wichtig hierbei ist, dass die importierten Daten aus Excel eine bestimmte Form haben. Diese ist in Tabelle 4.1

²Spiegelung an der Hauptdiagonalen

³da es sich hier um Elemente, die die Zeit beschreiben, handelt, müssen die Werte < 0 nicht betrachtet werden

⁴da es sich bereits um eine *double*-Variable handelt, kann diese ohne Weiteres auch nur mit einer *double*-Variablen verrechnet werden; also z.B. 1.0 statt 1

4. Implementierung

dargestellt.

Tabelle 4.1.: Benötigte Form der Datenmatrix: in den elf Spalten müssen immer die gleichen Elemente stehen; n ist variabel und entspricht der Anzahl der Botschaften, die im aufgezeichneten Zeitbereich gesendet wurden.

	Spalte 1	Spalte 2	Spalte 3	Spalten 4 - 11
Zeile 1	ID_1	DLC_1	$Counter_1$	$(\text{Datenbyte } 1 - 8)_1$
Zeile 2	ID_2	DLC_2	$Counter_2$	$(\text{Datenbyte } 1 - 8)_2$
...
Zeile $n - 1$	ID_{n-1}	DLC_{n-1}	$Counter_{n-1}$	$(\text{Datenbyte } 1 - 8)_{n-1}$
Zeile n	ID_n	DLC_n	$Counter_n$	$(\text{Datenbyte } 1 - 8)_n$

Die Form wird durch Excel-interne Operationen erreicht. In Abbildung 3.4 ist ersichtlich, dass die Datenbytes alle zusammen in einer Zelle gespeichert werden. In Excel ist es möglich, die beinhaltenden Zellen aufzusplitten und so jedem Datenbyte eine eigene Zelle zuzuweisen. Danach muss lediglich noch die Spalte für den Counter in Spalte drei verschoben werden, um die endgültige Form zu erreichen. Zu beachten ist beim Matlab-Import, dass auch Error-Flags auftauchen können. Diese belegen im Datensatz eine gesamten Zeile mit den Worten *Error active* oder *Error passive*. Da sich diese Mitteilung in der Datenmatrix befindet und dementsprechend als *double* importiert werden soll, kann in der beinhaltenden Zeile keine verwertbare Information festgestellt werden. Das Gleiche gilt auch für leere Zellen, die auftauchen können, wenn der DLC kleiner als acht ist und dadurch nicht alle Spalten von vier bis elf gefüllt sind. Für diese Fälle ist es notwendig, die Importoptionen so anzupassen, dass die nicht formgerechten Zellen mit einer null gefüllt werden. Eine solche Einstellung entfällt beim Zeitvektor aufgrund des *cell*-Datentyps und dadurch, dass es ein Spaltenvektor ist, in dem es keine „Lücken“ gibt.

4.1.4. Data-Mining - Datenergründung

In den vorangegangenen Abschnitten wurde ein großer Aufwand beschrieben, der nur dazu dient, die Daten so vorzubereiten, dass sie rechnerintern gut verarbeitet werden können. Wissen konnte dabei jedoch noch nicht generiert werden. Es

wurden lediglich einige Erkenntnisse darüber gewonnen, dass die gesuchten Informationen an manchen Stellen nicht gefunden werden können. Der sprichwörtliche Heuhaufen wurde also verkleinert. Nun beginnt die tatsächliche Analyse.

Das Programm geht dabei ähnlich vor wie bei der manuellen Analyse (vgl. Abschnitt 3.3) beschrieben. Im ersten Teil der Wissensgenerierung wird die Funktion „auswahl_berechnen“ aufgerufen. Die Funktion trifft verschiedene Auswahlen von IDs, die in ein bestimmtes Raster fallen und deshalb etwas mit dem erzeugten Signal zu tun haben. Dazu betrachtet sie den BUS und bemerkt spezielle Veränderungen, die sie in drei unterschiedlichen Variablen speichert. Ausgegangen wird hierbei von dem bereits dezimierten Datensatz. Aus diesem wird jeder Zustand auf bestimmte Eigenschaften überprüft und so erkannt, ob er weiterverarbeitet werden soll, oder nicht.

Die erste Auswahl

Die erste Auswahl ist die kleinste und wichtigste Auswahl. In ihr werden alle IDs gespeichert, deren Botschaften sich im vorgegebenen Zeitintervall zwischen „start“ und „ende“ zum ersten Mal seit längerer Zeit ändern. Sie müssen also vorab bereits mit einer konstanten Botschaft auf dem BUS gelegen haben und sich davon ausgehend im angegebenen Zeitbereich zum ersten Mal geändert haben. Man kann bei diesen IDs mit hoher Wahrscheinlichkeit davon ausgehen, dass sie sich nur aufgrund des eingegebenen Signals ändern, da sie vorab schon lange Zeit konstant auf dem BUS gesendet haben. Besonders für Ereignisse, die sich durch „an“ und „aus“ beschreiben lassen, ist diese Auswahl das wichtigste Indiz.

Die zweite Auswahl

Die zweite Auswahl speichert alle IDs ab, die zum ersten Mal überhaupt im vorgegebenen Zeitintervall auftauchen. Dies bedeutet, sie dürfen vorher noch gar nicht auf dem BUS gesendet haben. Die zweite Auswahl ist damit eine wichtige Ergänzung zur ersten, welche die neuen IDs übersieht. Hieraus ergibt sich auch die Notwendigkeit beim Aufzeichnen der Daten eine gewisse Zeit warten zu müssen. Wäre der BUS noch am „aufwachen“, würden in dieser Auswahl sehr viele IDs stehen, da sie gerade zum ersten Mal gesendet werden. Damit könnte die Auswahl kaum zur Reduktion des noch vorhandenen Datensatzes beitragen und wäre somit

4. Implementierung

nutzlos.

Die größte Zielgruppe der zweiten Auswahl sind Botschaften, die Ereignisse mit geringerer Priorität beschreiben. Diese müssen nicht unbedingt zyklisch auf den BUS gesendet werden, sondern können bei Bedarf einmalig gesendet werden.

Die dritte Auswahl

In der dritten Auswahl werden alle IDs gesichert, die im Intervall von „start“ bis „ende“ eine völlig neue Botschaft senden (der *counter* wird dabei natürlich nicht beachtet). Sie stellt die größte Auswahl dar und dient weniger dazu, einen Datensatz mit möglichen Kandidaten zu erzeugen, als weitere IDs auszuschließen.

Außer bei der zweiten Auswahl muss immer eine Veränderung festgestellt werden, um einen ID in eine Auswahl speichern zu dürfen. Deshalb werden auch jeweils der vorausgehende und der aktuelle Zustand in verschiedenen Variablen gespeichert, um später feststellen zu können, welche Wertänderung zur Aufnahme in die Auswahl geführt hat. Für die zweite Auswahl wird als vorausgehender Zustand der Nullvektor angegeben.

Darüber hinaus wird in der Funktion „auswahl_berechnen“ auch noch berechnet, wie oft sich jede Botschaft und die zugehörigen Bytes ändern. Das Ergebnis wird in der Matrix „wie_oft“ gespeichert und wird für die weitere Auswertung benötigt.

Aus den ermittelten Auswahlen wird im nächsten Schritt ein Ranking erstellt. In der Variablen „wahrscheinlichste_ids“ werden die IDs gespeichert, die am häufigsten in den Auswahlen vorkommen. Diese IDs haben Priorität gegenüber den Restlichen und werden bei der folgenden Auswertung bevorzugt behandelt.

Zunächst werden nun, wie in Unterabschnitt 4.1.1 beschrieben, kleine Datensätze erstellt, die nur noch die wahrscheinlichsten IDs und deren Botschaften enthalten. Hier ist anzumerken, dass die nachfolgend beschriebenen Operationen auch für alle anderen Auswahlen durchgeführt werden. Da aber davon ausgegangen wird, dass die richtigen IDs in der Variablen „wahrscheinlichste_ids“ stehen, wird diese Variable hier beispielhaft beschrieben. Danach wird mit Hilfe von „wie_oft“ ermittelt, welche Bytes der Botschaften eines ID sich nie ändern. Diese fallen logischerweise durch das Raster und sind keine potentiellen Signalträger. Der nächste wichtige

Schritt ist, zu bestimmen, ob es sich bei den Bytes um Zähler handelt oder nicht. Der Botschafts-Counter, der zählt wie oft ein ID gesendet wurde, wird dabei nicht betrachtet. Zähler können im Frame auch dann auftauchen, wenn zum Beispiel gezählt werden muss, wie lange ein bestimmter Zustand beständig ist und sind für die weitere Betrachtung irrelevant. Steht also ein Zähler in einem Byte, so wird es nicht weiter betrachtet. Die Information darüber wird trotzdem festgehalten und später auch ausgegeben. Hinter der Bestimmung, ob es sich um einen Zähler handelt, steckt zunächst eine simple Logik: verändert sich der Wert des Bytes bei jedem Senden um einen konstanten Wert, ist es ein Zähler. Das Problem dabei ist, dass ein Byte nur $2^8 = 256$ Werte darstellen kann. Deshalb folgt auf die einfache Logik noch ein Mechanismus zur Auswertung, welcher jede konstante Änderung mit dem Inkrement einer Hilfsvariablen bewertet. Ist der Wert nicht konstant, wird der Wert der Variablen dagegen um eins reduziert. Übersteigt der Wert der Hilfsvariablen danach einen bestimmten Punkt, wird das Byte als Zähler markiert. Auf diese Weise wird verhindert, dass ein Zähler mit Überlauf, der zum Beispiel von 256 wieder auf 0 springt und sich somit nicht konstant ändert, nicht als Zähler interpretiert wird.

Mittlerweile weiß das Programm also, welche IDs mit der höchsten Wahrscheinlichkeit das gesuchte Signal enthalten, welche IDs auf keinen Fall in Frage kommen und welche Bytes es sein könnten. Nun ist es noch notwendig, die Bytes weiter zu entschlüsseln und somit auf die Bit-bzw. Signal-Ebene vorzudringen. Dafür müssen zunächst jeweils die beiden Botschaften, die zur Aufnahme in eine Auswahl geführt haben, in einen Binär-Wert umgewandelt werden. Dazu wurde die Funktion „binaertransformation“ geschrieben. An diesem Punkt der Funktion wurden bereits alle Informationen ermittelt, die nötig sind, um die finale Auswertung durchzuführen.

In der Funktion „auswertung_berechnen“ wird nun alles vereint, um die gesuchten Signale zu ermitteln. Dazu werden die Botschaften der verbleibenden IDs weiter analysiert. Es wird kontrolliert, welche Bits sich genau ändern. Dies wird abgeglichen mit der Information, in welchen Bytes ein Zähler steht. Da der Datensatz an dieser Stelle bereits so stark dezimiert wurde, werden die Bytes nur noch zusätzlich mit der Variablen „haeufigkeiten“ abgestimmt. Aufgrund der Änderungen der Bits konnte nun bestimmt werden, welche die vermutlichen Signale sind, die hinter der

4. Implementierung

manuellen Eingabe im Fahrzeug stecken.

Diese werden als „gesuchte_signale“ ausgegeben und beinhalten an diesem Punkt Informationen über den ID, die Nummer des Bytes (von null bis sieben), die Nummer des Bits (ebenfalls von null bis sieben), den Ausgangswert des Bytes und den Endwert des Bytes.

In den folgenden Code-Zeilen werden noch Spalten mit den zugehörigen Zykluszeiten, dem DLC und der Information, ob in einem Byte ein Zähler steht oder nicht, angehängt. Somit ergibt sich als Ergebnis eine Matrix mit acht Spalten, in der alle Informationen stehen, die nötig sind, um die gefundenen Signale zu definieren.

4.1.5. Interpretation/Evaluation

Der letzte Punkt im KDD-Prozess und damit der letzte Schritt der Implementierung, ist die Interpretation bzw. Evaluation der Ergebnisse. Um es dem User einfacher zu machen, werden die Ergebnisse der vorangehenden Analyse dazu als Klartext wie in Abbildung 4.1 ausgegeben.

```
Die Analyse ergab, dass folgende/r ID/s am wahrscheinlichsten ist/sind:  
Das gesuchte Signal steckt vermutlich in  
Byte 0, Bit 0 des ID 502.  
(Wert von Byte 0 wechselt von 128 zu 177)
```

Abbildung 4.1.: Klartextausgabe zur einfacheren Interpretation für den User

Ist das Ergebnis eindeutig, d.h. es wurde genau ein ID gefunden, kann ohne Weiteres die letzte Funktion mit dem Namen „ausgabe“ aufgerufen werden. Diese verlangt als Eingabeparameter die Variablen „gesuchte_signale“ und „Funktion“. Als Funktion wird der ausgeführte Vorgang definiert, wie zum Beispiel „Fensterheber links runter“ oder „Warnblinker an“. Erhält die Funktion nur diese beiden Variablen, schreibt sie automatisch alles Nötige in die vorab definierte Excel-Tabelle.

Bekommt man dagegen mehr als einen ID zurück, ist es zuerst notwendig zu verifizieren, welche die Richtigen sind (vgl. Abschnitt 3.4). Danach definiert man lediglich einen Vektor, in dem die IDs stehen, die bei der Kontrolle erkannt wurden. Diesen Vektor nennt man „vermutete_ids“ und übergibt ihn an dritter Stelle der Funktion „ausgabe“. Auf diese Weise schreibt die Funktion nur die Signale in die Excel-Tabelle, die vorher definiert wurden.

Mit diesem Schritt ist die Implementierung durchlaufen und Wissen wurde in Form

einer Excel-Tabelle erzeugt. Sollte es jedoch aus unbekannten Gründen dazu kommen, dass kein logisches Ergebnis ausgegeben wird, gibt die Auswertelogik der Funktion Tipps zurück, wie man das Ergebnis positiv beeinflussen kann. So wird es dem User einfacher gemacht, den Grund zu finden, aus dem kein verwertbarer Rückgabewert resultierte.

4.2. Datenverarbeitung mit Vorwissen

Die Funktion „identifizieren“ kann als Universallösung betrachtet werden, die nur einen Datensatz und einen einigermaßen genauen Zeitpunkt benötigt, um ein Signal zu detektieren. Oftmals gibt es aber auch Aufgabenstellungen, bei denen bereits Vorwissen vorhanden ist. Dazu wurde die Funktion „identifizieren_mit_vorwissen“ konzipiert.

Arbeitet man beispielsweise in der Umgebung des Motors, kann man davon ausgehen, dass betroffene Signale eher im niedrigen Bereich der IDs liegen, da sie eine dementsprechend höhere Priorität besitzen. So kann man den aufgenommenen Datensatz bereits manuell im Voraus stark reduzieren, wodurch die restliche Bearbeitung natürlich stark vereinfacht wird. Das Selbe gilt zum Beispiel für das Netzwerkmanagement.

Tabelle 4.2.: Eingabeparameter für die Funktion „identifizieren_mit_vorwissen“, mit der Zeilenanzahl n .

<i>daten</i>	Der Datensatz, als $n \times 11$ -Matrix
<i>Time</i>	Der Spaltenvektor mit allen Zeitschritten (<i>cell</i>)
<i>start</i>	Der Startzeitpunkt für das relevante Intervall
<i>ende</i>	Der Endzeitpunkt für das relevante Intervall
<i>wichtige_ids</i>	Zeilenvektor mit allen zu berücksichtigenden IDs
<i>unwichtige_ids</i>	Zeilenvektor mit allen nicht zu berücksichtigenden IDs
<i>kleinster_id</i>	Untere Grenze der zu berücksichtigenden IDs
<i>groesster_id</i>	Obere Grenze der zu berücksichtigenden IDs

Darüber hinaus kann es sein, dass man einem Signal bereits einen ID zuordnen kann. Sucht man nun ein ähnliches Signal, ist es gut möglich, dass es auf dem

4. Implementierung

selben ID liegt. Möchte man dies überprüfen, kann man der Funktion auch genau diesen (oder eben auch mehrere) ID übergeben und der Algorithmus konzentriert sich nur auf diesen/diese ID/s. Weiß man dagegen bereits, dass ein ID auf keinen Fall das gesuchte Signal enthält, kann man diesen auch außen vor lassen, indem man dem Programm im Voraus mitteilt, dass es diesen ID ignorieren soll.

All dies kann in die Funktion „`identifizieren_mit_vorwissen`“ integriert werden, indem man die in Tabelle 4.2 aufgeführten Variablen richtig definiert.

Beim Aufruf der Funktion ist zu beachten, dass die Variablen `daten`, `Time`, `start` und `ende` definiert sein müssen, da das Programm ansonsten nicht funktioniert. Die restlichen Parameter können optional eingeführt werden. Möchte man nicht alle, oder gar keine verzichtbaren Eingaben machen, müssen diese auf null gesetzt werden. Die Funktion läuft dann nach dem gleichen Schema ab wie die Funktion „`identifizieren`“. Da hierbei in manchen Fällen bereits der ID bekannt ist, kann man in der main-Funktion den Aufruf von „`wann_data_einzeln`“ aktivieren. Diese Funktion ermittelt die genauen Sendezeitpunkte der gegebenen IDs. ein solches Vorgehen kann zu besseren Ergebnissen führen, da das relevante Zeitfenster so kleiner definiert werden kann.

Daraus ergeben sich viele weitere Fragestellungen, die mit Hilfe der entwickelten Funktionen bearbeitet werden können. Sehr komplexe Zusammenhänge können herausgefunden werden, indem man durch die Datenverarbeitung ohne Vorwissen bereits eine Auswahl an IDs eingrenzen kann. Je mehr Datensätze dazu verwendet werden, desto exakter wird die Vorauswahl. Das so ermittelte Vorwissen kann dann in die Funktion „`identifizieren_mit_vorwissen`“ eingelesen werden. So kann aus einer Kombination der Funktionen sehr viel tiefer gehendes Wissen generiert werden.

5. Ergebnisbetrachtung

Das Ziel der vorliegenden Arbeit ist, den vom Fahrer erzeugten Ereignissen ein BUS-Signal zuzuordnen, um eine Verbindung zwischen realem BUS-Verkehr und Aktionen des Fahrzeugs herzustellen, da diese Informationen nur dem OEM vorliegen. In diesem Kapitel werden die Ergebnisse, die während des Bearbeitungszeitraums gefunden wurden und deren Verifizierung beleuchtet.

5.1. Ergebnisse

Methodik zur Signalfindung

Die Hauptaufgabe ist das Finden einer Methodik zum automatischen Herausfinden der gesendeten BUS-Signale eines Fahrzeugs.

Die gefundene Vorgehensweise ist als Datenverarbeitungsprozess zu verstehen. Die einzige manuelle Arbeit ist dabei das Generieren eines entsprechenden Datensatzes. Wie in Abschnitt 3.3 beschrieben, bietet CANoe dazu eine gute Lösung. Sobald die Daten vorhanden sind, gibt es die zwei Funktionen „identifizieren“ und „identifizieren_mit_vorwissen“, die, je nach Informationsstand des Users, in der Lage sind die korrekten Signale zu finden. Danach muss nur noch die Funktion „ausgabe“ ausgeführt werden, um die gefundenen Signale (entweder alle oder nur die in „vermutete_ids“) in die vorab definierte Excel-Tabelle einzutragen.

Der Grad der Automatisierung ist damit sehr hoch, sobald die nötigen Daten vorliegen.

5. Ergebnisbetrachtung

Unabhängigkeit vom Wissensstand

Die entwickelten Funktionen bieten Lösungen für verschiedene Fragestellungen. Möchte man einfache Signale finden, erledigt dies die Funktion „identifizieren“ schnell und zuverlässig. Hat man dagegen bereits Vorwissen, das die IDs betrifft, kann man dieses nutzen und dadurch Filter definieren, die die Ausgabe des Algorithmus noch genauer machen. Wendet man eine Kombination der beiden Funktionen an, können auch Signale in sehr „rauen“ BUS-Umgebungen¹ analysiert werden.

Unabhängigkeit von der Schnittstelle

Als KDD-Prozess ist die Analyse nur von der Richtigkeit des gegebenen Datensatzes abhängig. Dieser muss die einzelnen Zeitschritte, den ID, den DLC, den Botschafts-Counter und alle Datenbytes in Dezimalform enthalten (vgl. Tabelle 4.1). Wie er erzeugt wurde spielt keine Rolle. Somit ist die Methode völlig unabhängig von der gegebenen Schnittstelle. Einzig eine Verbindung zum BUS muss bestehen.

Unabhängigkeit vom Hersteller

Die Funktionen wurden mit Hilfe der Fahrzeuge des IEEM entwickelt - einem Mercedes-Benz CL550 und einem BMW i3. Damit wurde sichergestellt, dass die Analyse nicht nur bei einem Hersteller funktioniert. Da der Praxistest nur am BMW vorgenommen werden konnte, wurde die Verifizierung (vgl. Abschnitt 5.2) an einem von Mercedes-Benz verwendeten Kombiinstrument vorgenommen. So wurde eine Unabhängigkeit vom OEM bewiesen.

Ausgewählte Signale

Nach einer manuellen Analyse des BUS-Verkehrs wurden zunächst einige Vermutungen zu den Verbindungen zwischen ausgewählten Vorgängen und ihren BUS-Signalen aufgestellt (vgl. Tabelle 3.3). Diese konnten mit Hilfe der Code-Abwicklung bestätigt werden. Daraus ergibt sich die Grundlage für die K-Matrix des BMW

¹also eine BUS-Umgebung, in der sehr viel Traffic herrscht

i3, die man, je nach Bedarf, stetig erweitern kann. Die Auswahl ist in Anhang A.4 ersichtlich und beinhaltet ausschließlich Signale des Body-CANs.

Zu sehen ist, dass nur wenige Signale ihre Informationen tatsächlich in nur einem Byte übertragen. Dazu gehört zum Beispiel der Zustand der Handbremse, der auf dem zweiten Byte des ID 732 übertragen wird. Dabei gibt es einen Zwischenzustand, der anzeigt, dass die Handbremse gerade weder ganz gelöst noch gezogen ist. Dass die Handbremse angezogen ist, wird mit der Zahl 11 angegeben, dass sie offen ist, mit der 2. Die Fensterheber dagegen übertragen einen Soll- und einen Ist-Wert und die Information, ob sich das Fenster gerade hebt oder senkt. Dazu werden drei Bytes versendet.

Es wurden also ohne Probleme verschiedenartige Signale identifiziert.

5.2. Verifizierung der Ergebnisse

Bei Aufgaben, in denen ein Programm dem Menschen in gewisser Weise überlegen ist, spielt das Vertrauen in das Programm eine sehr große Rolle. Da es jedoch nicht möglich ist, jedes Ergebnis händisch zu überprüfen und darüber hinaus eine solche Überprüfung natürlich auch einer gewissen Fehlertoleranz unterliegt, muss eine vernünftige Vertrauensbasis geschaffen werden. Die Idee ist dabei sehr einfach: die gefundenen Signale werden auf den BUS gesendet und es wird beobachtet, ob das erwartete Ereignis eintritt. Auf diese Weise ergibt sich der Beweis, dass die ermittelten IDs und Botschaften korrekt sind. Leider ist es aus verschiedenen Gründen nicht erlaubt, die Signale an das reale Fahrzeug zu senden. Deshalb musste eine Alternative gefunden werden, die einen gleichwertigen Beweis liefern kann. Eine Lösung bot dazu der Versuchsaufbau einer Laborveranstaltungen der HsKA mit dem Titel *Automotive E/E-Systeme*. Dieser Aufbau besteht aus einem Kombiinstrument einer S-Klasse von Mercedes-Benz, einer entsprechenden Spannungsversorgung von 13,8 V, einem CANcase VN1630 und den zugehörigen Verkabelungen. Viel wichtiger für die Verifizierung ist allerdings die Tatsache, dass für diese Veranstaltung reale Signale des Kombiinstruments ermittelt wurden.

Mit den bekannten Signalen ergibt sich eine eindeutige Vorgehensweise zur Lieferung des Beweises der Funktionstüchtigkeit. Diese Vorgehensweise wurde für verschiedene Signale angewandt und soll nun an einem Beispiel erläutert werden: Als Erstes wird das bekannte Signal, das den linken Blinker einschaltet, in CANoe

5. Ergebnisbetrachtung

definiert und in eine Botschaft integriert. In diesem Fall gehört das Signal zur Botschaft des ID mit der Nummer $0x29 = 41_d$. Der ID hat einen DLC von drei und beinhaltet darüber hinaus noch weitere Signale wie zum Beispiel das Ein- und Ausschalten des Warnblinkers und des rechten Blinkers. Der linke Blinker wird eingeschaltet, indem der Wert von Byte null auf 64 geändert wird. Da der Anfangswert des Bytes null ist, entspricht dies dem Setzen einer eins auf dem sechsten Bit. Die zyklische Sendezzeit wird hier mit 100 ms definiert [20].

Nachdem das Signal definiert wurde, wurde der Laptop mit dem CANcase verbunden und die Messung, wie in Abschnitt 3.4 beschrieben, durchgeführt. Der Unterschied war allerdings, dass das Signal diesmal nicht erzeugt wurde, indem ein Knopf am Kombiinstrument betätigt wurde, sondern indem das vordefinierte Signal für den linken Blinker durch einen interaktiven Generatorblock über CANoe an das CANcase übermittelt wurde. Daraufhin wurde der linke Blinker des Kombiinstruments aktiviert, wodurch bewiesen war, dass das Signal richtig definiert wurde.

Dieser gesamte Versuch wurde wiederum aufgezeichnet und im Anschluss mit den in Kapitel 4 dargestellten Funktionen analysiert. Der Algorithmus hatte also die Aufgabe, das bekannte Signal zu finden. Gibt er dementsprechend den vorher definierten ID und dessen Botschaft als Ausgabewert zurück, ist der Beweis geliefert, dass die Methodik funktioniert.

Tatsächlich lieferte die Analyse das richtige Signal als Rückgabe. Es kann also davon ausgegangen werden, dass die ermittelten Zustände der realen K-Matrix des BMW i3 korrekt sind.

Diese Verifizierung wurde mit Hilfe der Funktion „identifizieren“ durchgeführt. Im nächsten Schritt wurde der Array „wichtige_ids“ mit einer Länge von eins und einem Wert von 41 definiert, um auch die Funktion „identifizieren_mit_vorwissen“ zu testen. Diese lieferte mit und auch ohne weitere Einschränkungen durch die Variablen „kleinster_id“ und „groesster_id“ das selbe Ergebnis und kann somit ebenfalls als funktionstüchtig bezeichnet werden.

Da der Mercedes-Benz CL550 des IEEM für die Auswertung leider nicht mehr zur Verfügung stand, dient diese Verifizierung auch als Beweis dafür, dass die gesamte Auswertung nicht nur für den BMW i3, sondern auch für ein von Mercedes-Benz verwendetes Kombiinstrument funktioniert und damit völlig unabhängig vom OEM ist.

6. Fazit und Ausblick

Die hier entwickelten Funktionen zum Finden von CAN-BUS-Signalen funktionierten im Rahmen dieser Arbeit nahezu fehlerfrei. Allerdings wurde zur besseren Wiederholbarkeit und aus Gründen der Erreichbarkeit fast nur am Body-CAN des BMW i3 gearbeitet. Außerdem wurde dieser aus versicherungstechnischen Gründen während der Arbeit nie bewegt. Das Auswerten von Signalen, die während der Fahrt gesendet werden, ist mit einem erhöhten Aufwand verbunden. Da der Algorithmus sich vor allem auf Veränderungen von Botschaften konzentriert, wäre es nötig, dafür zu sorgen, dass die Zustände der Botschaften völlig konstant gehalten werden, bis die geplante Aktion durchgeführt wurde. Für eine solche Analyse sollten vorab möglichst viele Informationen zu den Botschaften generiert werden, um die Datenauswertung optimal vorbereiten zu können. Dies ist zwar mit großem Aufwand verbunden, ergibt aber ein tiefgreifendes Wissen über hochpriore Botschaften, welche Informationen zu (Elektro-) Motoren und Fahrmanövern beinhalten. Mit diesen Informationen sind Rückschlüsse auf andere wichtige Fahrzeugparameter möglich, die man ohne die Kenntnis über die Informationen der Botschaften nicht erlangen könnte.

Dafür kann man sich für viele Anwendungen, die auf dem Kennen von Signalen basieren, den Aufwand von finanziellen Mitteln sparen, da es nicht unbedingt notwendig ist, das Diagnosegerät des Herstellers zu kaufen. Lediglich der Anschluss an den BUS muss gelegt werden. Danach kann die Auswertung mit wenig Mühe durchgeführt werden. Die Abhängigkeit vom OEM entfällt. Allerdings wird für die richtige Interpretation der gefundenen Informationen eine gute Kenntnis des CAN-Busses vorausgesetzt.

Da eine vergleichbare Lösung nicht bekannt ist, eröffnen sich neue Möglichkeiten für die Arbeit mit CAN. So könnten sich zukünftige Arbeiten damit beschäftigen, die K-Matrix des BMW i3 zu vervollständigen, indem die erzeugte Excel-Tabelle als Datenbasis in Matlab eingebunden wird. Dadurch wäre es möglich, einen Al-

6. Fazit und Ausblick

gorithmus zu erschaffen, der sein Vorwissen selbst bezieht. Natürlich ist dies nicht auf den BMW i3 beschränkt, sondern bei jeder BUS-Übertragung, die die nötigen Informationen sendet, möglich.

Sollen bestimmte Anwendungsfälle bearbeitet werden, können die Funktionen auch ohne Matlab beliebig verändert und erweitert werden, wenn vorher aus Matlab ein C-Code generiert wird. Die bisher entwickelten Funktionen bieten dafür eine solide Basis.

Interessant wäre auch eine Verknüpfung des Algorithmus mit CANoe. Könnte man den genauen Zeitpunkt festhalten, indem das Signal des Fahrers gesendet wurde, könnte das nötige Zeitfenster ebenfalls automatisch erzeugt werden. Das Zeitfenster könnte so deutlich kleiner gehalten werden. Sind Latenzzeiten und Übertragungsraten genau bekannt, könnte man sich sogar fast auf einen einzigen Zeitpunkt des entsprechenden Zeitvektors begrenzen. Dies würde in Kombination mit der Tatsache, dass das Preprocessing deutlich vereinfacht würde, den Grad der Automatisierung weiter erhöhen.

Literatur

- [1] Katrin Alisch, Ute Arentzen und Eggert Winter. *Gabler Wirtschaftslexikon*. 16. Auflage. Wiesbaden und s.l.: Gabler Verlag, 2004. ISBN: 978-3-663-01439-3. DOI: 10.1007/978-3-663-01439-3. URL: <http://dx.doi.org/10.1007/978-3-663-01439-3> (geprüft am 10.01.2017).
- [2] Jun.-Prof. Dr. Holger Giese. “Softwaretechnik-Praktikum: 5. Vorlesung”. Diss. Paderborn: Universität Paderborn, 2006. (Geprüft am 07.02.2017).
- [3] *Vector E-Learning: Motivation*. Stuttgart, 2017. URL: https://elearning.vector.com/index.php?seite=vl_sbs_introduction_de&root=376493&wbt_ls_kapitel_id=480361&wbt_ls_seite_id=480362&d=yes (geprüft am 11.01.2017).
- [4] International Organization for Standardization. *ISO/IEC 7498-1: Information Technology - Open Systems Interconnection - Basic Reference Model: The Basic Model*. Genf, 1994. URL: <https://www.ecma-international.org/activities/Communications/TG11/s020269e.pdf> (geprüft am 11.01.2017).
- [5] Vector Informatik GmbH. *Vector E-Learning: ISO/OSI-Modell*. Stuttgart, 2017. URL: https://elearning.vector.com/index.php?seite=vl_sbs_introduction_de&root=376493&wbt_ls_kapitel_id=4%E2%80%8B80361&wbt_ls_seite_id=500546&d=yes (geprüft am 11.01.2017).
- [6] Prof. Dr. rer. nat. Peter Neugebauer. “Fahrzeugelektronik 2: Bussysteme”. Skript. Karlsruhe: Hochschule Karlsruhe - Technik und Wirtschaft, 2010-2014. (Geprüft am 15.12.2016).
- [7] Vector Informatik GmbH. *Vector E-Learning: Einführung*. Stuttgart, 2017. URL: <http://www.vector-elearning.com/> (geprüft am 09.01.2017).
- [8] Daniel Schüller. “Bussysteme im Automobil”. Ausarbeitung. Landau: Universität Koblenz-Landau, 20.01.2005. (Geprüft am 09.02.2017).

Literatur

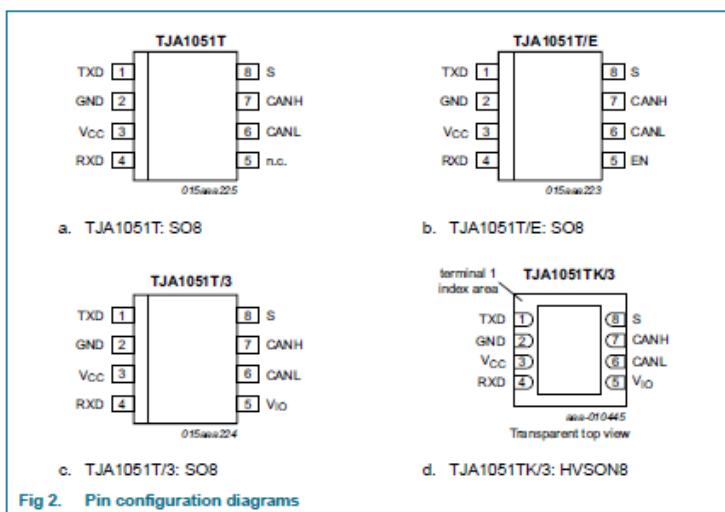
- [9] Jonas Hummes. “Übersicht und Entwicklung von automotiven Bussystemen”. Ausarbeitung. Landau: Universität Koblenz-Landau, 24.07.2009. (Geprüft am 09.02.2017).
- [10] Werner Zimmermann und Ralf Schmidgall. *Bussysteme in der Fahrzeugtechnik: Protokolle, Standards und Softwarearchitektur*. 5., aktual. und erw. Aufl. ATZ / MTZ-Fachbuch. Wiesbaden: Springer Vieweg, 2014. ISBN: 978-3-658-02418-5. DOI: 10.1007/978-3-658-02419-2. URL: <http://dx.doi.org/10.1007/978-3-658-02419-2> (geprüft am 10.01.2017).
- [11] Vector Informatik GmbH. *Vector E-Learning: CAN-Netzwerk*. Stuttgart, 2017. URL: https://elearning.vector.com/index.php?&wbt_ls_seite_id=500582&root=376493&seite=vl_sbs_introduction_de (geprüft am 11.01.2017).
- [12] Christof Kary. “Ermittlung der Rekuperationsenergie und der Leistungsaufnahme von Nebenverbrauchern in einem Elektroauto”. Bachelorthesis. Karlsruhe: Hochschule Karlsruhe - Technik und Wirtschaft, 12.09.2016. (Geprüft am 11.01.2017).
- [13] International Organization for Standardization. *Road vehicles – Controller area network (CAN)*. Genf, 12/2015. URL: http://www.iso.org/iso/catalogue_detail.htm?csnumber=63648 (geprüft am 31.01.2017).
- [14] Vector Informatik GmbH. *Vector E-Learning: Frametypen*. Hrsg. von Vector Informatik GmbH. Stuttgart, 2017. URL: https://elearning.vector.com/index.php?&wbt_ls_seite_id=451394&root=376493&seite=vl_can_introduction_de (geprüft am 31.01.2017).
- [15] ME-Messsysteme. *CAN Bus Grundlagen: Aufbau einer CAN Nachricht*. Hrsg. von ME-Messsysteme. Henningsdorf, 2017. URL: <https://www.me-systeme.de/de/support/grundlagen/can-bus-grundlagen> (geprüft am 31.01.2017).
- [16] Prof. Dr.-Ing. Klemens Gintner. “Sensorik: Signalübertragung in der Sensorik”. Skript. Stuttgart: AKAD University, 2016. (Geprüft am 11.01.2017).
- [17] Sabine Queckbörner. “Was ist Data Mining: Business Intelligence ii - Data Mining & Knowledge Discovery”. Seminar. Kaiserslautern: Technische Universität Kaiserslautern, 23.1.2004. URL: <http://wwwlgis.informatik.uni-kl.de/archiv/wwwdvs.informatik.uni-kl.de/courses/seminar/WS0304/ausarbeitung1.pdf> (geprüft am 08.02.2017).

- [18] Usama Fayyad, Gregory Piatetsky-Shapiro und Padhraic Smyth. *From Data Mining to Knowledge Discovery in Databases*. Hrsg. von American Association for Artificial Intelligence. 1996. URL: <http://www.csd.uwo.ca/faculty/ling/cs435/fayyad.pdf> (geprüft am 08.02.2017).
- [19] Kevin Rassner und Murat Öztürk. “CL500 CAN-Bus”. Projektarbeit. Karlsruhe: Hochschule Karlsruhe - Technik und Wirtschaft, 13.12.2012. (Geprüft am 12.01.2017).
- [20] Marcel Rumez. “Automotive E/E-Systeme”. Laborveranstaltung. Karlsruhe: Hochschule Karlsruhe - Technik und Wirtschaft, 10.16. (Geprüft am 26.02.2017).

A. Anhang

A.1. Auszug aus dem Datenblatt des NXP TJA 1051/3

6.1 Pinning



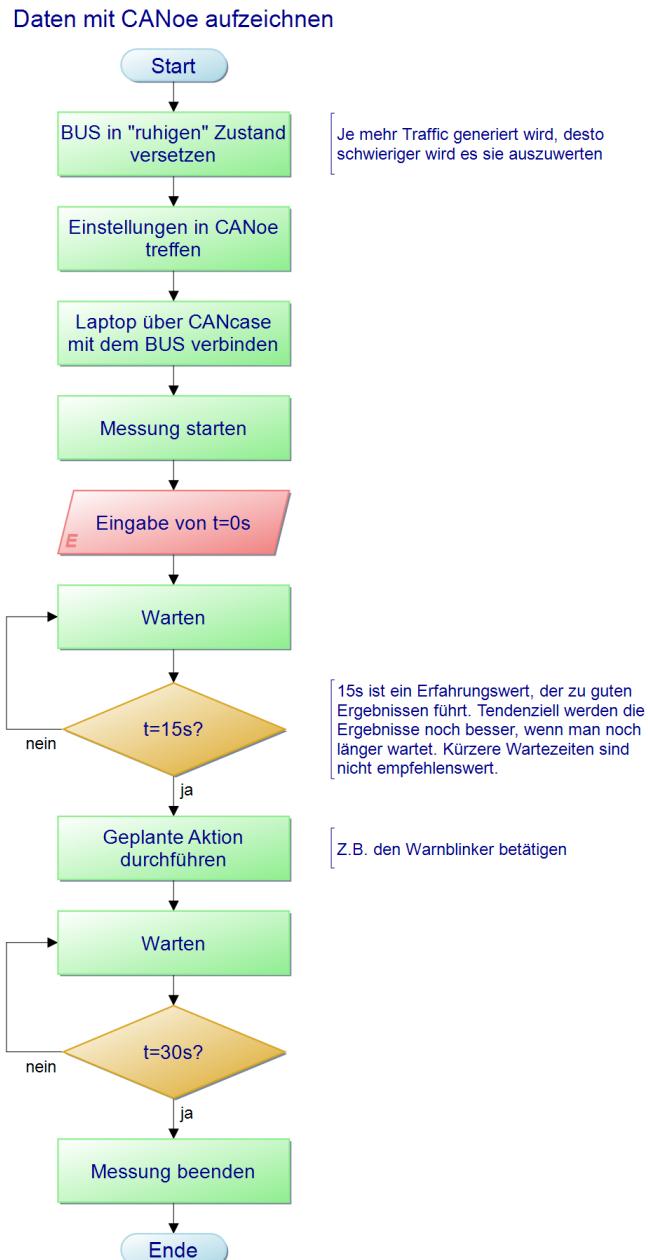
6.2 Pin description

Table 3. Pin description

Symbol	Pin	Description
TXD	1	transmit data input
GND ^[1]	2	ground
V _{cc}	3	supply voltage
RXD	4	receive data output; reads out data from the bus lines
n.c.	5	not connected; in TJA1051T version
EN	5	enable control input; TJA1051T/E only
V _{io}	5	supply voltage for I/O level adapter; TJA1051T/3 and TJA1051TK/3 only
CANL	6	LOW-level CAN bus line
CANH	7	HIGH-level CAN bus line
S	8	Silent mode control input

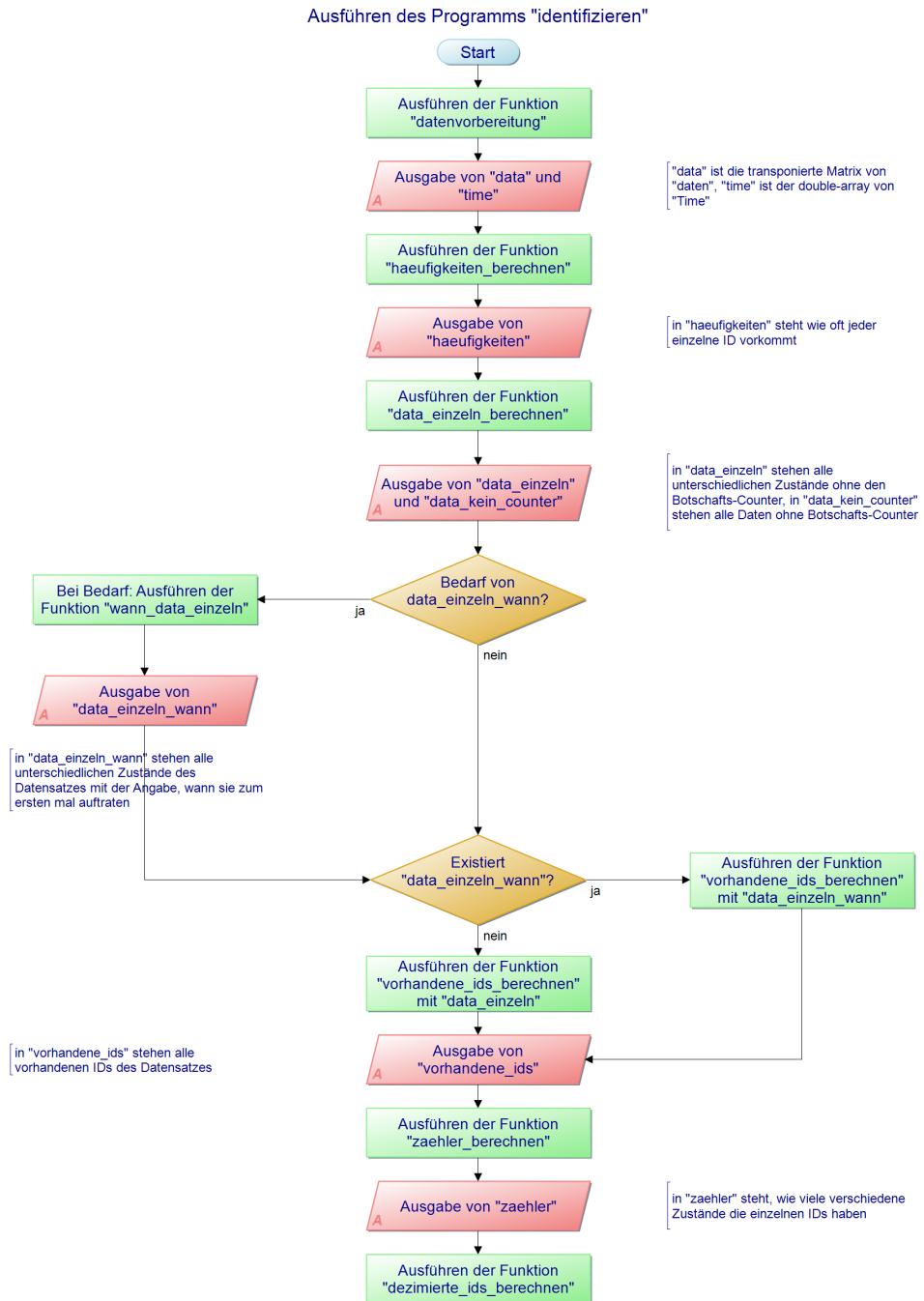
[1] HVSON8 package die supply ground is connected to both the GND pin and the exposed center pad. The GND pin must be soldered to board ground. For enhanced thermal and electrical performance, it is recommended that the exposed center pad also be soldered to board ground.

A.2. Programmablaufplan der Unterfunktion „Daten mit CANoe aufzeichnen“

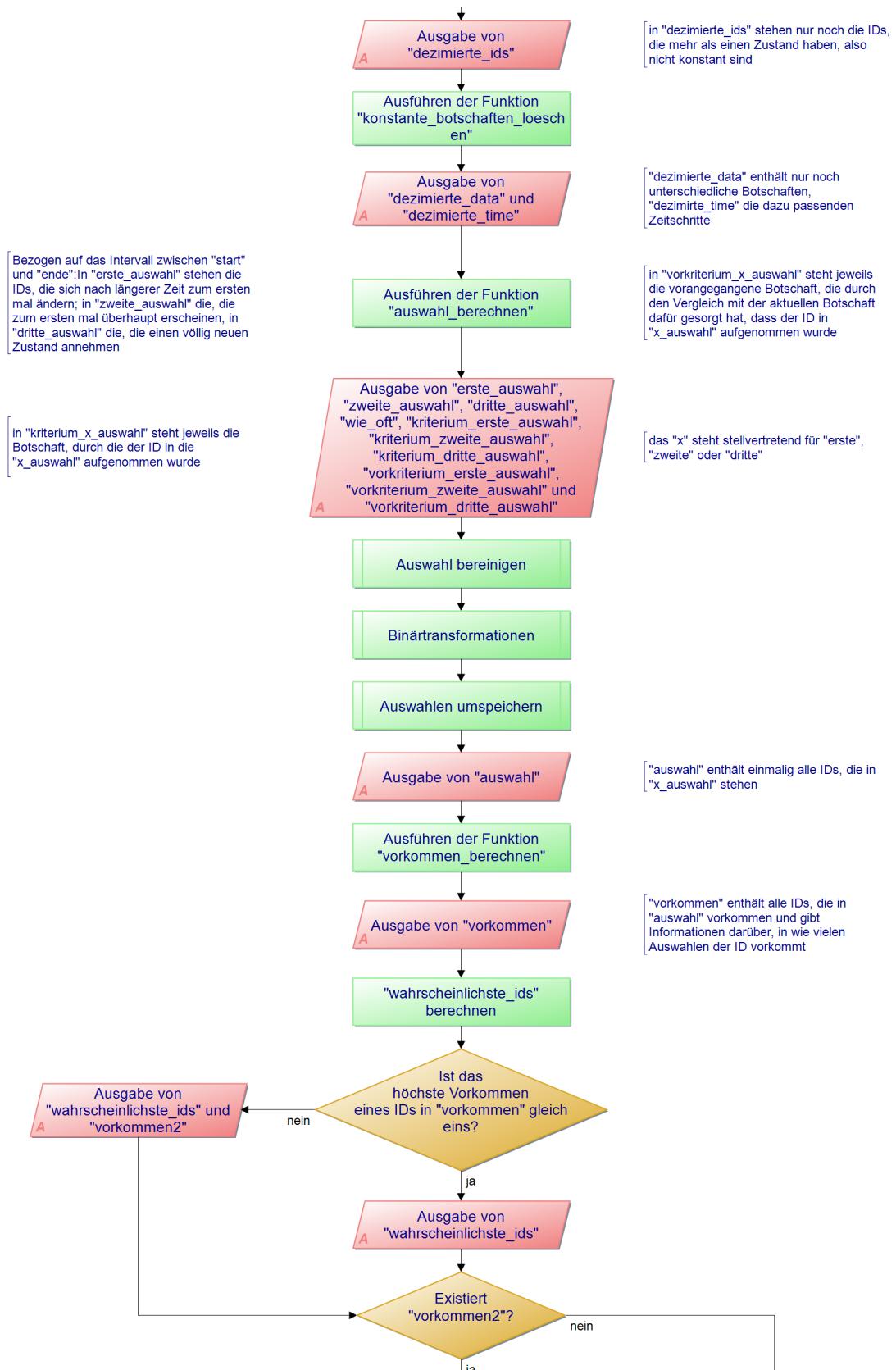


A.3. Programmablaufplan der Funktion „identifizieren“

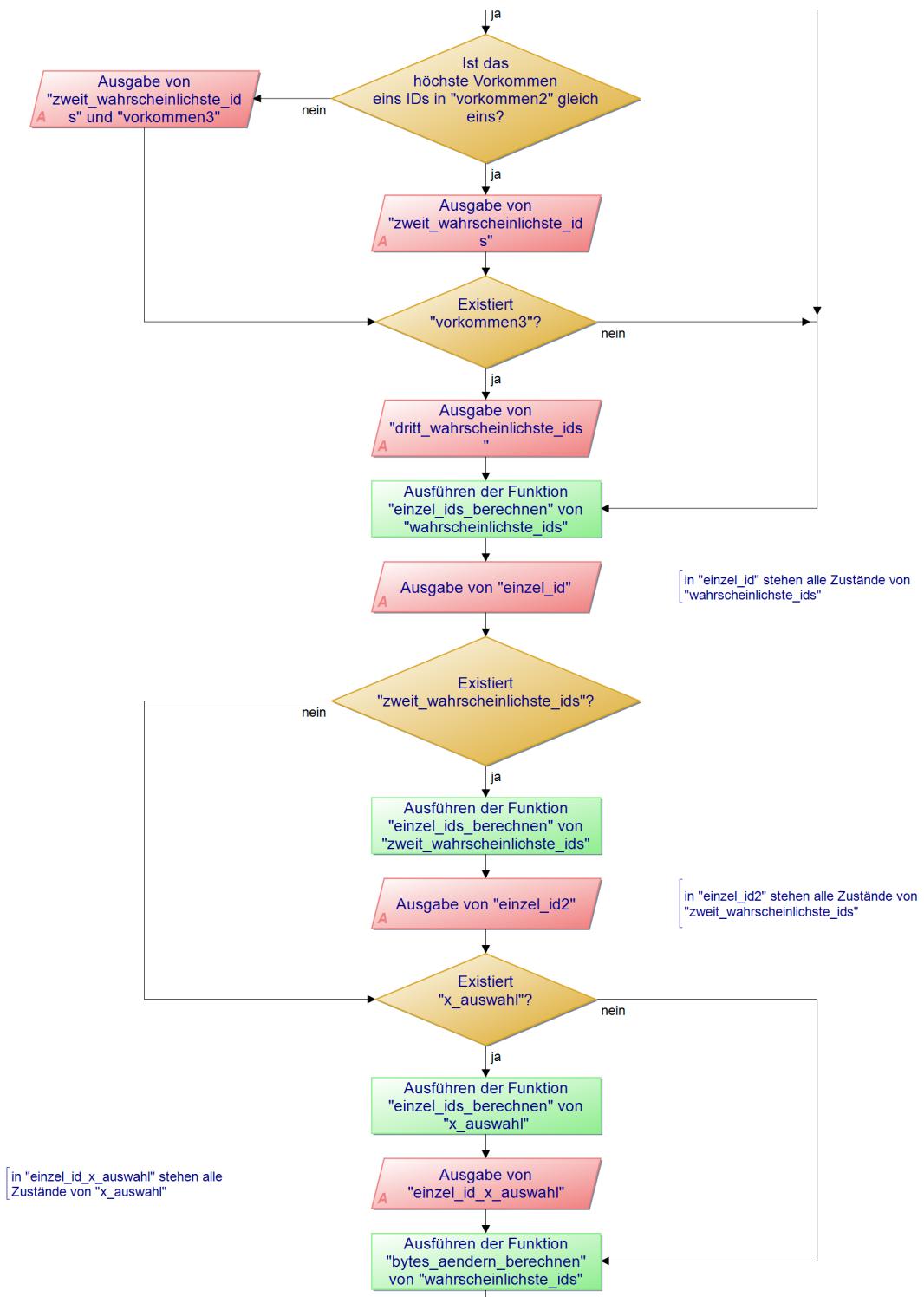
A.3. Programmablaufplan der Funktion „identifizieren“



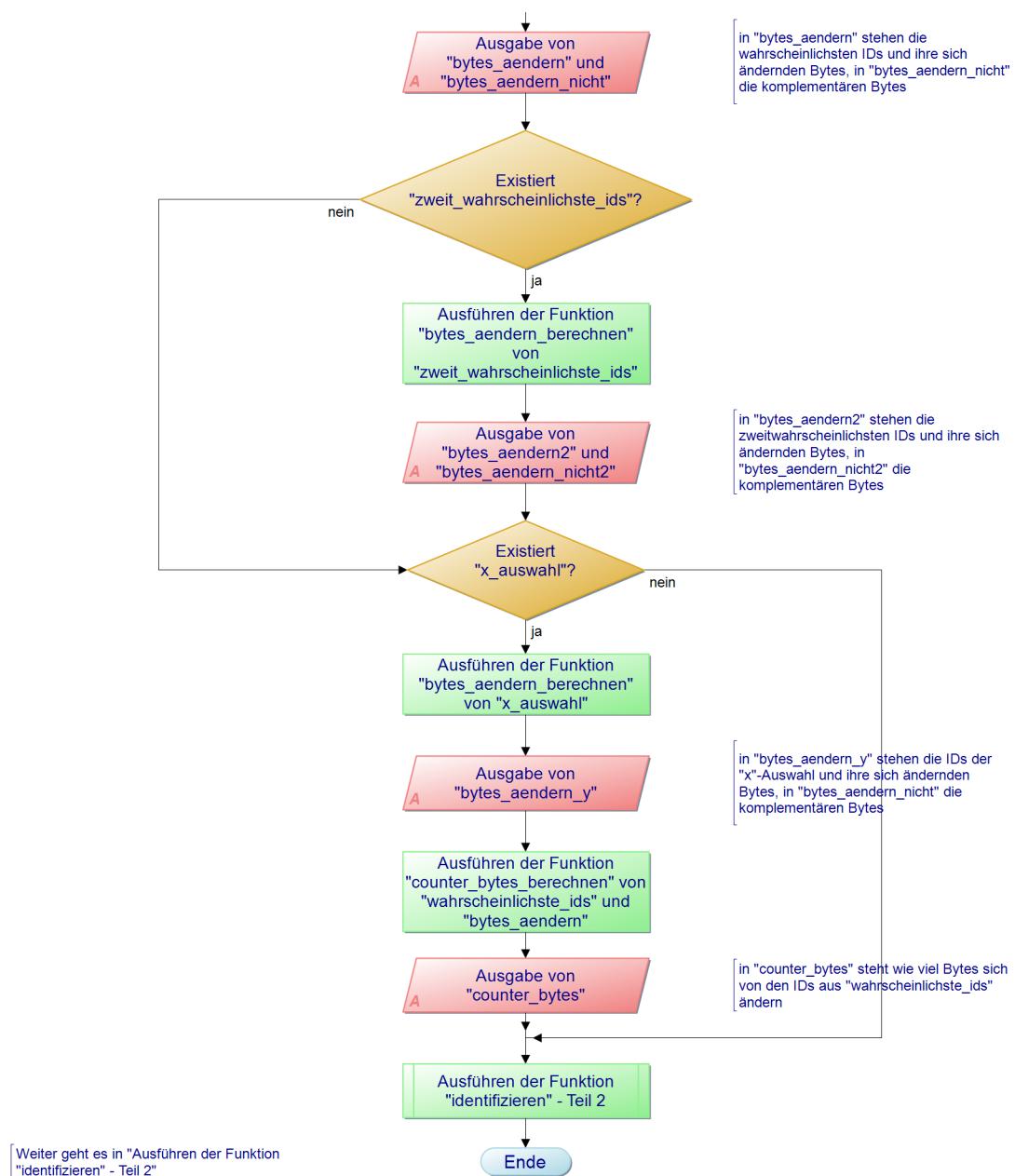
A. Anhang



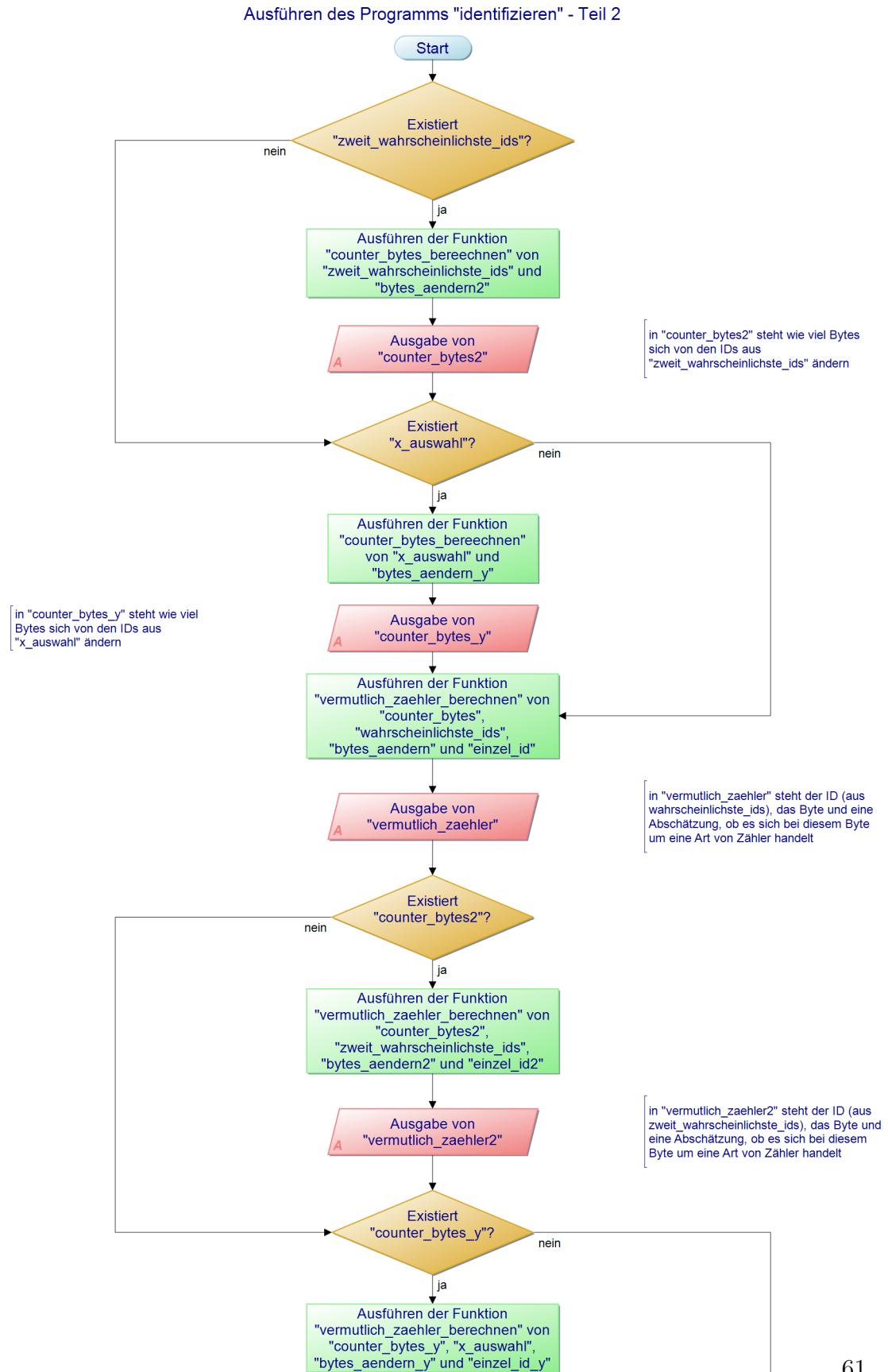
A.3. Programmablaufplan der Funktion „identifizieren“



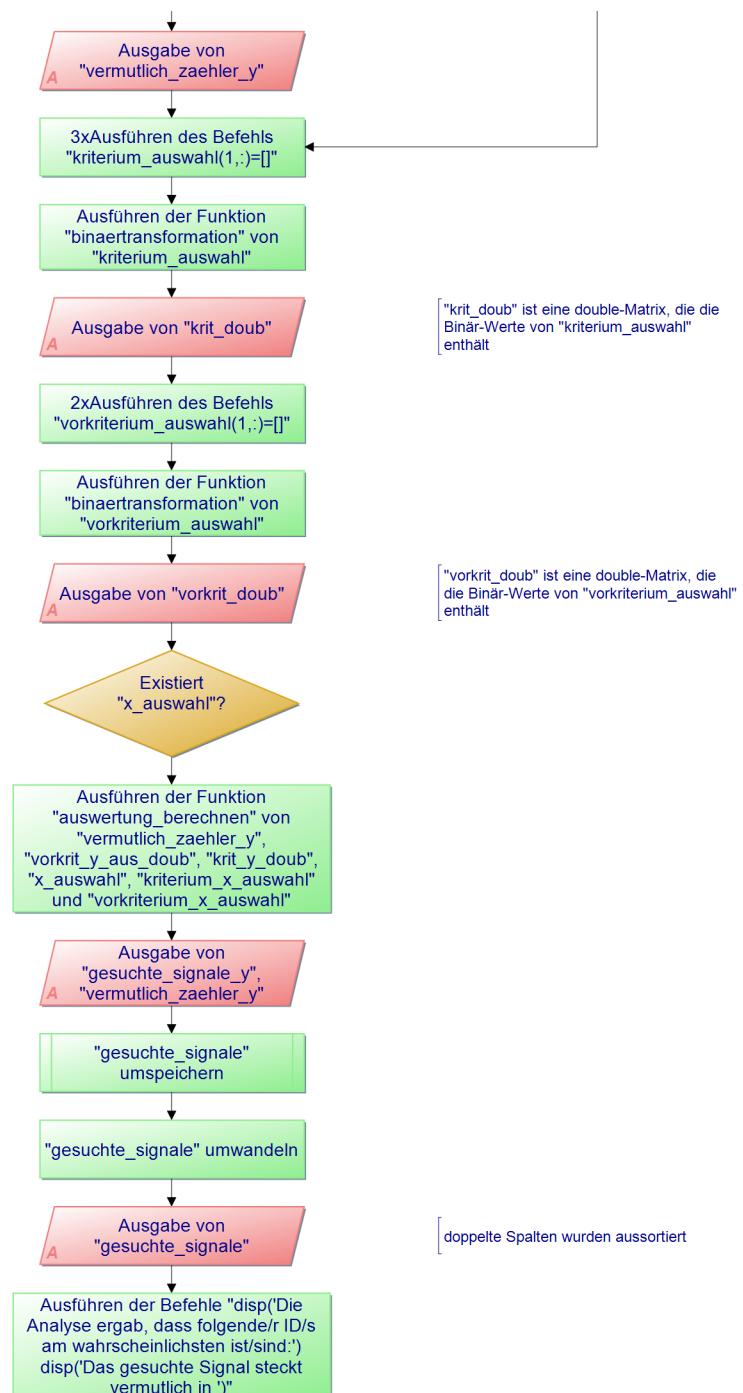
A. Anhang



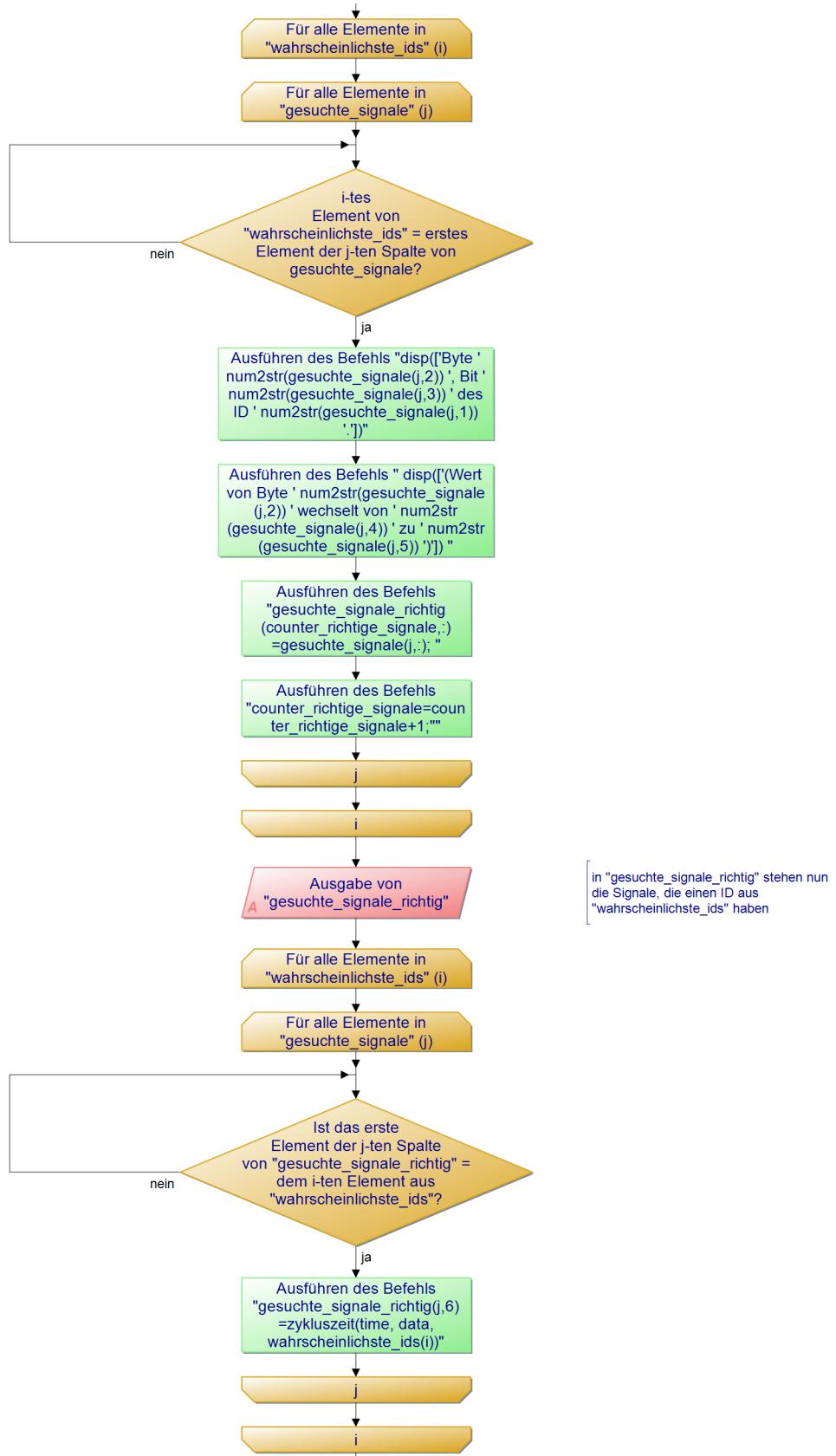
A.3. Programmablaufplan der Funktion „identifizieren“



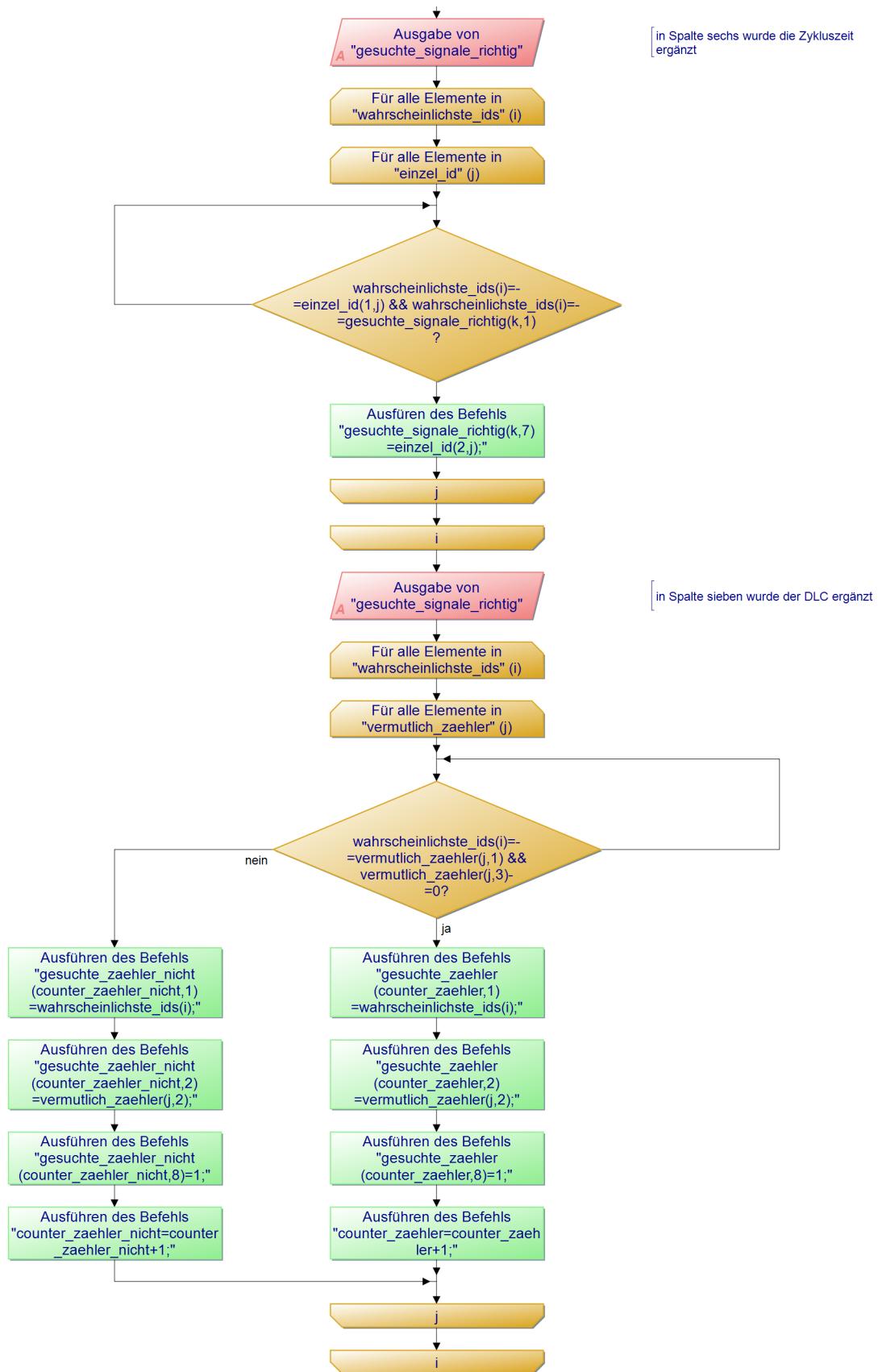
A. Anhang



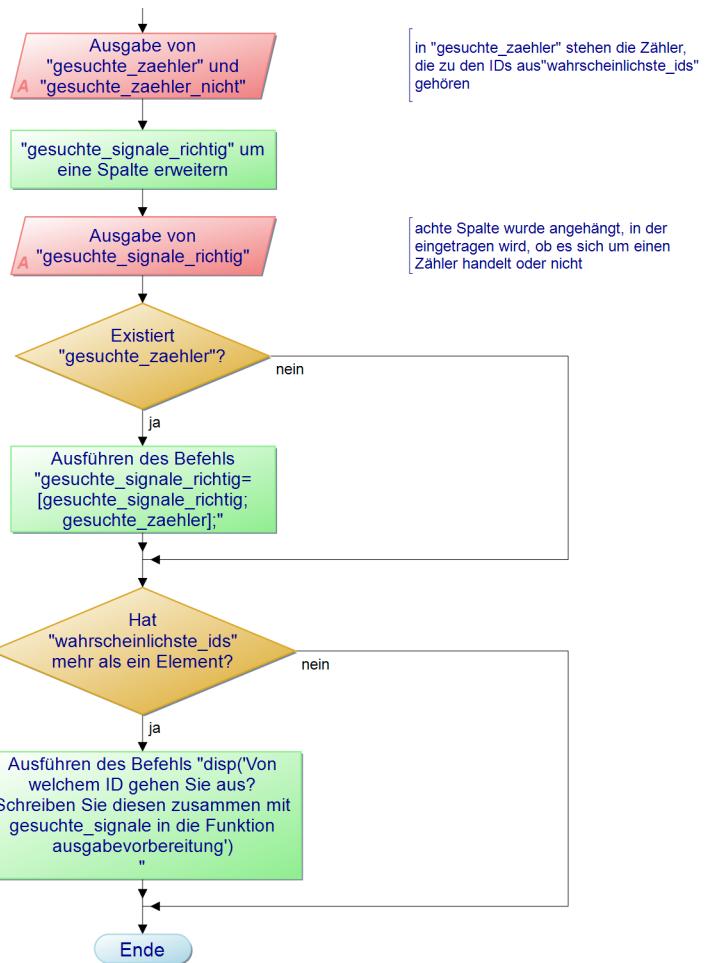
A.3. Programmablaufplan der Funktion „identifizieren“



A. Anhang



A.3. Programmablaufplan der Funktion „identifizieren“



A.4. Ausgewählte Signale der K-Matrix des BMW i3

Funktion	ID	Byte	Bit	Startwert	Endwert	Zykluszeit [s]	DLC	Zähler?
Warnblinker anschalten	502	0		128	177	2	2	nein
			0					
			4					
			5					
		1		240	242	2	2	nein
			1					
	950	0		0	4	0,1	3	nein
			2					
			1	0	232	0,1		
			0					
			2					
		1		3			3	nein
			4					
			5					
			6					
			7					
	2	2		0	232	0,1	3	nein
			3					
			5					
			6					
			7					
Handbremse lösen	732	2		11	10	1	3	nein
Handbremse ziehen	732	2		2	10	1		
			3					

A.4. Ausgewählte Signale der K-Matrix des BMW i3

Funktion	ID	Byte	Bit	Startwert	Endwert	Zykluszeit [s]	DLC	Zähler?
Fensterheber rechts runter	952	0	0	4	0,1	3	nein	
		1	0	232	0,1	3	nein	
Zentralver- riegelung mit Schlüssel schließen	764	0	5	129	33	/	7	nein

A.5. CD

