Untersuchungen zur Anomalieerkennung in automotive Steuergeräten durch verteilte Observer mit Fokus auf die Plausibilisierung von Kommunikationssignalen

Zur Erlangung des akademischen Grades eines
DOKTOR-INGENIEURS
von der KIT-Fakultät für
Elektrotechnik und Informationstechnik
des Karlsruher Instituts für Technologie (KIT)

genehmigte
DISSERTATION
von

M. Eng. Marc Weber

geb. in: Reutlingen

Tag der mündlichen Prüfung:	7. März 2019
Hauptreferent:	Prof. Dr.-Ing. Eric Sax
Korreferent:	Univ.-Prof. Dr. Wolfgang Pree

To my family

Kurzfassung

Die zwei herausragenden automobilen Trends Konnektivität und hochautomatisiertes Fahren bieten viele Chancen, aber vor allem in ihrer Kombination auch Gefahren. Einerseits wird das Fahrzeug immer mehr mit seiner Außenwelt vernetzt, wodurch die Angriffsfläche für unautorisierten Zugriff deutlich steigt. Andererseits erhalten Steuergeräte die Kontrolle über sicherheitsrelevante Funktionen. Um das Risiko und die potentiellen Folgen eines erfolgreichen Angriffs möglichst gering zu halten, sollte eine Absicherung auf mehreren Ebenen erfolgen.

Der Fokus dieser Arbeit liegt auf der innersten Absicherungsebene und dabei speziell auf der Überwachung von Fahrezug-interner Kommunikation. Hierfür empfehlen Wissenschaft und Industrie unter anderem den Einsatz von Intrusion Detection/Intrusion Prevention Systemen. Das erarbeitete Konzept greift diesen Vorschlag auf und berücksichtigt bei der Detaillierung die Steuergeräte-spezifischen Randbedingungen, wie beispielsweise die vergleichsweise statische Fahrzeugvernetzung und die limitierten Ressourcen. Dadurch entsteht ein hybrider Ansatz, bestehend aus klassischen Überwachungsregeln und selbstlernenden Algorithmen. Dieser ist nicht nur für die Fahrzeug-interne Kommunikation geeignet, sondern gleichermaßen für den Steuergeräte-internen Informationsaustausch, die Interaktion zwischen Applikations- und Basissoftware sowie die Überwachung von Laufzeit- und Speichereigenschaften. Das übergeordnete Ziel ist eine ganzheitliche Steuergeräte-Überwachung und damit eine verbesserte Absicherung im Sinne der Security. Abweichungen vom Sollverhalten - sogenannte Anomalien - werden jedoch unabhängig von deren Ursache erkannt, sei es ein mutwilliger Angriff oder eine Fehlfunktion. Daher kann dieser Ansatz

auch zur Verbesserung der Safety beitragen, speziell wenn Applikationen und Algorithmen abzusichern sind, die sich während des Lebenszyklus eines Fahrzeugs verändern oder weiterentwickeln.
Im zweiten Teil der Arbeit steht die Plausibilisierung von einzelnen Kommunikationssignalen im Vordergrund. Da deren möglicher Verlauf nicht formal beschrieben ist, kommen hierfür selbstlernende Verfahren zum Einsatz. Neben der Analyse und der Auswahl von grundsätzlich geeigneten Algorithmen ist die Leistungsbewertung eine zentrale Herausforderung. Die zu erkennenden Anomalien sind vielfältig und in der Regel sind nur Referenzdaten des Normalverhaltens in ausreichender Menge vorhanden. Aus diesem Grund werden unterschiedliche Anomalie-Typen definiert, welche die Anomaliesynthese in Normaldaten strukturieren und somit eine Evaluierung anhand der Erkennungsrate erlauben. Die Evaluierungsergebnisse zeigen, dass eine Signalplausibilisierung mittels künstlichen neuronalen Netzen (Autoencoder) vielversprechend ist. Zum Abschluss betrachtet die vorliegende Arbeit daher die Herausforderungen bei deren Realisierung auf automotive Steuergeräten und liefert entsprechende Kennzahlen für die benötigte Laufzeit und den Speicherverbrauch.

Abstract

Connectivity and autonomous driving are two major automotive trends. They have the potential to increase safety and comfort but also yield new dangerous scenarios, especially in their combination. On the one hand, vehicles get increasingly connected with each other and external infrastructure, thus offering more attack surface for hackers. On the other hand, electronic control units (ECU) take over safety critical functions like longitudinal and lateral control of the vehicle. To ensure security, different mechanisms should be applied to form a multi-level defense.

The thesis at hand focusses the innermost defense barrier, especially the observation of in-vehicle communication, for which different researchers and associated industry recommend the use of intrusion detection and intrusion prevention systems. This suggestion is refined and the Automotive Observer concept is proposed considering automotive specifics like the relatively static system design as well as the limited resources on embedded devices. This results in a hybrid approach consisting of static rules followed by machine learning based algorithms. The concept is not only suited for the observation of in-vehicle communication but also takes ECU-internal communication, the interaction between application and basic software as well as runtime and memory consumption into account. The high-level goal is an improved reaction on cyber attacks. However, deviations from normal behavior are identified by means of anomaly detection and independent from the root cause, which might be a cyber attack or a functional misbehavior. Therefore, the proposed concept can also help to enhance functional safety, for example by safeguarding self-learning or updatable applications.

The second part of the thesis deals with the plausibility of single communi-

cation signals. While some of their properties are specified in a semi-formal manner, the possible temporal behavior is not given during development time. For this reason, self-learning mechanisms are applied. Besides the selection of a suited algorithm, the necessary performance evaluation is a major challenge. Potential anomalies are manifold and usually there is no anomalous reference data available. To overcome this situation, multiple anomaly types are defined, which allow for structured synthesization of anomaly instances and therefore enable an evaluation based on the true and false positive rate. The results show that signal plausibility checking via Autoencoders - a specific type of artificial neural networks - is promising. At the end, the thesis at hand discusses challenges when implementing Autoencoders on automotive ECUs and gives some corresponding key figures on runtime and memory consumption.

Danksagung

Ich bedanke mich ganz herzlich bei Prof. Dr.-Ing. Eric Sax für die Betreuung meines Promotionsvorhabens am Institut für Technik der Informationsverarbeitung (ITIV), speziell für die stets ausführlichen und sehr hilfreichen Rückmeldungen. Vielen Dank auch an Univ.-Prof. Dr. Wolfgang Pree für die Übernahme des Korreferats und an die weitere Prüfungskommission, bestehend aus Prof. Dr. rer. nat. Cornelius Neumann, Prof. Dr.-Ing. Martin Doppelbauer und Prof. Dr. rer. nat. Bernhard Holzapfel.
Besonderer Dank geht an die Vector Informatik GmbH in Person von Dr. Thomas Beck, Dr. Helmut Schelling, Dr. Günther Heling und Jochen Rein für die Ermöglichung und die Unterstützung meines Promotionsvorhabens. Des Weiteren bedanke ich mich bei Dr. Bastian Zimmer und seinem Team für die hilfreichen Gespräche und die Rücksicht auf meine zeitlich begrenzte Anwesenheit in den letzten Jahren.
Vielen Dank an die Kollegen des ITIV, allen voran Felix Pistorius und Daniel Grimm, für fruchtbare Diskussionen und die gute Arbeitsatmosphäre.
Außerdem bedanke ich mich bei allen Studierenden - im Speziellen bei Simon Klug, Simon Leiner, Jonas Maas und Georg Wolf - die mit ihren Abschlussarbeiten einen wertvollen Beitrag geleistet haben.
An dieser Stelle vielen Dank an meine Eltern, Christa und Claus Weber, und an meine Schwester, Annika Weber, unter anderem für den bedingungslosen Rückhalt. Das letzte und gleichzeitig größte Dankeschön geht an Alexandra Lechner. Ohne ihr Verständnis und ihre Unterstützung wäre mein Promotionsvorhaben in dieser Form nicht möglich gewesen.

Reutlingen, im März 2019 *Marc Weber*

Inhaltsverzeichnis

Abkürzungsverzeichnis

ABS	Antiblockiersystem
ACK	Acknowledge
Adaptive	AUTOSAR Adaptive Platform
API	Application Programming Interface
App	Applikation
ARXML	AUTOSAR Extensible Markup Language
ASIL	Automotive Safety Integrity Level
ATLAS	Automatically Tuned Linear Algebra Software
AUC	Area Under Curve
Auto-ISAC	Automotive Information Sharing and Analysis Center
AUTOSAR	Automotive Open Systems Architecture
C/S	Client/Server
CAL	Cybersecurity Assurance Level
CALLOBS	SysCall Observer
CAN	Controller Area Network
CAN FD	CAN with Flexible Data Rate
CAN-ID	CAN-Identifier
CAN-TP	CAN-Transport Protocol
CAN_H	CAN-High
CAN_L	CAN-Low
CANDRV	CAN-Treiber
CANIF	CAN-Interface
CANOBS	CAN Observer
CC	Common Criteria
CC	Computing Cluster

CDD	Complex Driver
CIDS	Clock-based Intrusion Detection System
Classic	AUTOSAR Classic Platform
CNN	Convolutional Neural Network
COM	Communication Modul
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CSMA/CR	Carrier Sense Multiple Access/Collision Resolution
CSMS	Cybersecurity Management System
CSV	Comma-separated values
DBC	Database for CAN
DEL	Delimiter
DEM	Diagnostic Event Manager
DLC	Data Length Code
DPI	Deep Packet Inspection
DSRC	Dedicated Short Range Communication
DTLS	Datagram Transport Layer Security
E.U.	European Union
E/E	Elektrik/Elektronik
E2E	End-to-End
ECU	Electronic Control Unit
EOF	End of Frame
ESP	Elektronisches Stabilitätsprogramm
ETHDRV	Ethernet Treiber
ETHIF	Ethernet Interface
ETHOBS	Ethernet Observer
EU	European Union
EVITA	E-safety Vehicle Intrusion Protected Applications
FANN	Fast Artificial Neural Network Library
FN	False Negative
FOTA	Firmware Over-the-Air

FP	False Positive
FPGA	Field Programmable Gate Array
FPR	False Positive Rate
FPU	Floating Point Unit
FROBS	FlexRay Observer
GCC	GNU Compiler Collection
GENET	Generated Networks
HARA	Hazard Analysis and Risk Assessment
HIDS/HIPS	Host-based Intrusion Detection/Prevention System
HiPAC	High Performance Packet Classification
HS-Trees	Half-Space-Trees
HSM	Hardware Security Module
HTA	Hardware Trust Anchor
I-PDU	Interaction Layer Protocol Data Unit
I/O	Input/Output
ID	Identifier
IDE	Identifier Extension
IDS	Intrusion Detection System
iForest	Isolation Forest
IPS	Intrusion Prevention System
IPsec	Internet Protocol Security
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
ISAC	Information Sharing and Analysis Center
ISO	International Organization for Standardization
ISR1	Interrupt-Service-Routine der Kategorie eins
ISR2	Interrupt-Service-Routine der Kategorie zwei
IT	Informationstechnik
ITIV	Institut für Technik der Informationsverarbeitung
ITM	Intermission
ITU	International Telecommunication Union

JSON	JavaScript Object Notation
KDD	Knowledge Discovery in Databases
KI	Künstliche Intelligenz
KIT	Karlsruher Institut für Technologie
LDCOM	Large Data Communication Modul
LIN	Local Interconnect Network
LINOBS	LIN Observer
LODA	Lightweight On-line Detector of Anomalies
LSTM	Long Short-Term Memory
MAC	Media Access Control
MAC	Message Authentication Code
MISRA	Motor Industry Software Reliability Association
MMU	Memory Management Unit
MOST	Media Oriented Systems Transport
MPU	Memory Protection Unit
MSE	Mean Squared Error
NAB	Numenta Anomaly Benchmark
NHTSA	National Highway Traffic Safety Administration
NIDS/NIPS	Network-based Intrusion Detection/Prevention System
NNEF	Neural Network Exchange Format
NvM	Non-volatile RAM Manager
OBD	On-Board-Diagnose
OBD-II	On-Board Diagnostics-II
OCSVM	One-Class Support Vector Machine
OF	Outlier Factor
ONNX	Open Neural Network Exchange Format
OS	Operating System
OSEK	Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug
OSI	Open Systems Interconnection
OTA	Over-the-Air

P-Port	Provider-Port
PDU-ID	Protocol Data Unit-Identifier
PDUR	PDU Router
PLAN	Piecewise Linear Approximation of a Nonlinear Function
POSIX	Portable Operating System Interface
PRC	Precision
QM	Quality Management
QoS	Quality-of-Service
R-Port	Require-Port
RAM	Random Access Memory
ReLU	Rectifier Linear Unit
RMSE	Root Mean Square Error
RNN	Replicator Neural Network
ROC	Receiver Operating Characteristics
ROM	Read-only Memory
RPC	Remote Procedure Call
RTE	Runtime Environment
RTM	Runtime Measurement
RTR	Remote Tranmission Request
S/R	Sender/Receiver
SAE	Society of Automotive Engineers
SBC	System-Basis-Chip
SC	Scalability Class
SCHM	Schedule Manager
SDL	Security Development Lifecycle
SecOC	Secure Onboard Communication
SEIS	Sicherheit in Eingebetteten IP-basierten Systemen
SEM	Security Event Memory
SESAMO	Security and Safety Modelling
SEVECOM	Secure Vehicle Communication
SGD	Stochastic Gradient Decent

SIGOBS	Signal Observer
SNNS	Stuttgart Neural Network Simulator
SOAD	Socket Adaptor
SOF	Start of Frame
SOME/IP	Service-oriented Middleware over IP
SOTA	Software Over The Air
SPI	Stateful Packet Inspection
SVDD	Support Vector Domain/Data Description
SVM	Support Vector Machine
SW-C	AUTOSAR Software-Komponente
SYSOBS	System Observer
T4	Text Template Transformation Toolkit
TARA	Threat Analysis and Risk Assessment
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TN	True Negative
TP	True Positive
TPR	True Positive Rate
TSN	Time-Sensitive Networking
TTCAN	Time-Triggered CAN
U.S.	United States of America
UDP	User Datagram Protocol
UDS	Unified Diagnostic Services
UN	United Nations
USB	Universal Serial Bus
V2I	Vehicle-to-Infrastructure
V2V	Vehicle-to-Vehicle
VC121	Vector Controller 121
VDX	Vehicle Distributed Executive
VLAN	Virtual Local Area Network
VZ	Vorzeichen

WLAN	Wireless Local Area Network

1 Einleitung

1.1 Motivation

Neben der Elektrifizierung prägen aktuell zwei weitere Themen die Entwicklung in der Automobilindustrie: Hochautomatisiertes beziehungsweise autonomes Fahren und Konnektivität. Im September 2017 prognostizierte Strategy& ca. 80 Millionen Fahrzeuge mit Automatisierungslevel 4 oder 5[1] bis 2030 sowie 470 Millionen vernetzte Fahrzeuge bis 2025 [ps17]. Beide Zahlen beziehen sich auf einen Markt bestehend aus den Vereinigten Staaten von Amerika (U.S.), der Europäischen Union (E.U.) und China.

Automatisiertes Fahren bedeutet auf technischer Ebene, dass Fahrzeuge mit einer Vielzahl von Sensoren und Aktoren ausgestattet werden. Damit können elektronische Steuereinheiten (Steuergeräte oder Electronic Control Units - ECUs) einerseits die Umgebung erfassen und andererseits Längs- und Querführung übernehmen. Dies erfordert leistungsstarke Hardware wie Prozessoren und Grafikeinheiten sowie komplexe Algorithmen, beispielsweise zur Bildverarbeitung.

Unter Konnektivität versteht man im automobilen Umfeld die Vernetzung des Fahrzeugs mit seiner Außenwelt. Diese spielt in mehreren Bereichen eine entscheidende Rolle. Viele der heutigen und zukünftigen Fahrzeugkäufer haben den Anspruch, ihr Smartphone nahtlos in ein Fahrzeug integrieren zu können, um dort zu telefonieren, die Lieblingsmusik aus ihrer Media-

[1] Automatisierungslevel 4 entspricht nach SAE J3016 einer Hochautomatisierung und Level 5 der Vollautomatisierung [SAE18]. Hochautomatisierung bedeutet, dass ein Fahrzeug in definierten Anwendungsfällen komplett eigenständig agiert. Bei einer Vollautomatisierung existiert keine Einschränkung auf bestimmte Anwendungsfälle. Damit kann ein Fahrzeug jede Fahraufgabe in jeder Situation eigenständig meistern und benötigt somit keinen Fahrer mehr.

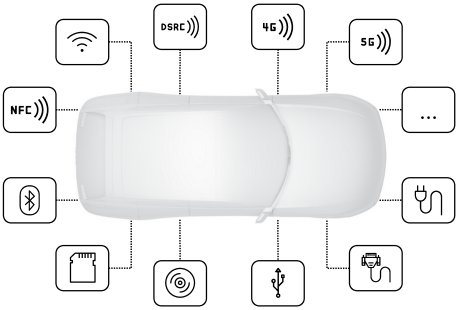

Bild 1.1: Kommunikationsschnittstellen eines Fahrzeugs[2]

thek zu hören oder weitere Applikationen (Apps) zu verwenden. Konnektivität beschreibt allerdings auch die Kommunikation von Fahrzeugen mit Infrastruktur wie Ampelanlagen und Parkhäusern sowie die Kommunikation zwischen Fahrzeugen. Dieser Teil spielt eine zentrale Rolle für die anstehenden Entwicklungen in den Bereichen Fahrzeugsicherheit und automatisiertes Fahren. Ein Beispiel hierfür ist der sogenannte eCall, welcher bei einem Unfall automatisch einen Notruf absetzt. Bei Personenkraftwagen und leichten Nutzfahrzeugen ist das System seit dem 31. März 2018 Voraussetzung für die europäische Typzulassung [Cou15]. Damit sind alle neuen Fahrzeuge automatisch zu einem gewissen Grad vernetzt. Abbildung 1.1 veranschaulicht mögliche Schnittstellen eines modernen und zukünftigen Fahrzeugs zur Kommunikation mit externen Gegenstellen.

Durch die steigende Vernetzung mit der Außenwelt und den damit verbundenen Luftschnittstellen nimmt allerdings gleichzeitig auch die Gefahr von Cyberangriffen, d.h. dem Manipulieren des elektronischen Systems, deutlich zu. Es entsteht das Risiko eines Angriffs auf eine gesamte Fahrzeugflotte, da physischer Zugriff auf Fahrzeuge nicht mehr notwendig ist. Einer der bis heute bekanntesten automobilen Cyberangriffe ist der soge-

[2] Quelle: Vector Informatik GmbH

nannte *Jeep Hack* von Charlie Miller und Chris Valasek [MV15]. Die beiden Wissenschaftler entdeckten mehrere Sicherheitslücken im Elektrik/Elektronik (E/E)-System eines zu der Zeit aktuellen Fahrzeugs und nutzten diese systematisch aus. Am Ende gelang die Beeinflussung von sicherheitsrelevanten Funktionen wie Bremse und Lenkung mittels handelsüblichem Computer über eine Mobilfunk-Verbindung. In einer früher veröffentlichten Arbeit unterteilen Charlie Miller und Chris Valasek das Vorgehen bei einem entfernten automobilen Cyberangriff in drei Stufen [MV14]:

1. Entfernter Zugriff auf ein Steuergerät
2. Zugriff auf ein sicherheitsrelevantes, Fahrzeug-internes Netzwerk, beispielsweise durch das Überbrücken eines Gateways
3. Auslösen eines (sicherheitskritischen) Fehlverhaltens

Dieses Beispiel zeigt, dass speziell die Kombination aus automatisiertem Fahren und Konnektivität hohe Risiken birgt. Einerseits haben Steuergeräte die Möglichkeit über die vorhandenen Sensoren und Aktoren aktiv in das Fahrverhalten einzugreifen oder die Fahrzeugführung komplett zu übernehmen. Andererseits vergrößert sich die Angriffsfläche durch die zusätzlichen (Luft-)Schnittstellen maßgeblich. Des Weiteren steigt die Menge an Software und damit auch die Anzahl an Sicherheitslücken in Fahrzeugen kontinuierlich an. 2015 wurde von ca. 100 Millionen Zeilen Software-Code in einem durchschnittlichen Oberklassefahrzeug ausgegangen [MDQ15]. Zum Vergleich: Facebook hatte im selben Zeitraum etwas über 60 Millionen Codezeilen [MDQ15]. NXP[3] geht von durchschnittlich ca. 300 Millionen Codezeilen im Jahr 2020 aus [Mco17].

Dass die Gefahr von sicherheitskritischen Cyberangriffen real ist, zeigen nicht nur Miller und Valasek. In den vergangenen Jahren wurden bereits einige, hauptsächlich wissenschaftlich motivierte Angriffe erfolgreich ausgeführt und dokumentiert. Bild 1.2 zeigt eine kurze Historie, wobei nur

[3] NXP ist einer der führenden Chiphersteller im automotive Bereich.

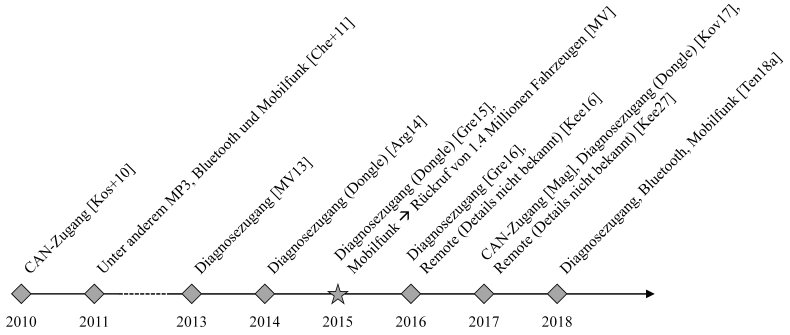

Bild 1.2: Historie bekannter, sicherheitskritischer Cyberangriffe auf Fahrzeuge

bekannte Vorfälle aufgelistet sind. Koscher et al. beeinflussten bereits 2010 unterschiedliche Komponenten im Fahrzeug wie beispielsweise die Bremsen [Kos+10]. Ihre Arbeit blieb damals weitgehend unbeachtet, da physischer Zugriff auf das Fahrzeug beziehungsweise die Fahrzeug-internen Netzwerke notwendig war. Aus diesem Grund erweiterten sie ihre Forschung auf drahtlose Schnittstellen und es gelang ihnen unter anderem über MP3, Bluetooth und Mobilfunk Nachrichten in das Fahrzeug-interne Netzwerk einzuspeisen[4], mit potentiell sicherheitskritischen Folgen [Che+11]. Ein ähnliches Vorgehen wählten Miller und Valasek. Zunächst verwendeten sie 2013 den lokalen Zugang über die Fahrzeugdiagnose-Schnittstelle, um beispielsweise eine Vollbremsung auszulösen [MV13]. 2015 folgte der bereits erwähnte Jeep Hack, welcher die Angriffssicherheit von Fahrzeugen in den Fokus der Öffentlichkeit rückte [MV15]. Gleichzeitig resultierte dieser Vorfall in der ersten automobilen Rückrufaktion aufgrund eines Cyberangriffs. Betroffen waren 1.4 Millionen Fahrzeuge. Auch bei vielen Fahrzeugherstellern fand daraufhin ein Umdenken statt. Parallel zu Miller und Valasek sowie in den darauf folgenden Jahren, nutzen Wissenschaftler die Fahrzeugdiagnose-Schnittstelle, um sicherheitskritisches Fehlverhalten

[4] Angriffe über Luftschnittstellen werden auch als *Remote*-Hacks bezeichnet.

auszulösen [Gre16]. Zunächst erscheint hierfür lokaler Fahrzeugzugriff erforderlich. Allerdings existieren Zubehörteile - sogenannte Dongles, die auf Basis der Fahrzeugdiagnose Zusatzdienste anbieten, wie beispielsweise ein elektronisches Fahrtenbuch. Einige dieser Dongles sind mit drahtlosen Schnittstellen ausgestattet, welche wiederum kompromittiert werden können und damit Angriffe aus der Ferne ermöglichen [Arg14] [Gre15] [Kov17]. In den Jahren von 2016 bis 2018 gelangen Keen Security Lab drei erfolgreiche Angriffe auf moderne Fahrzeuge über eine Luftschnittstelle [Kee16] [Kee27] [Ten18a]. Eine weitere Angriffsmöglichkeit zeigte Maggi 2017 indem er Spezifika der Fahrzeug-internen Vernetzungstechnologie ausnutzte, um dediziert einzelne Steuergeräte von der Kommunikation auszuschließen [Mag17].

Bei allen erwähnten Angriffen, hatten die Forscher am Ende Zugriff auf die Fahrzeug-interne Kommunikation, welche in diesen Fällen mittels Controller Area Network (CAN) erfolgte. Das sicherheitskritische Fehlverhalten resultierte stets durch das Versenden (oder Unterdrücken) von entsprechenden Botschaften. Eine ausführliche Einführung in das Thema Fahrzeugvernetzung und CAN folgt in den Abschnitten 2.1.3 und 2.1.4. Außerdem waren Fahrzeuge unterschiedlicher Marken betroffen. Es handelt sich somit nicht um das Problem eines einzelnen Herstellers sondern um ein Thema, das allgemeingültig zu adressieren ist.

Die fehlerfreie Funktion eines Fahrzeugs - und damit die Sicherheit der Insassen - ist zukünftig nur zu gewährleisten, wenn das Elektroniksystem gegen Cyberangriffe geschützt ist. Einem Angreifer soll beziehungsweise muss das Erreichen jeder einzelnen Angriffsstufe so schwer wie möglich gemacht werden. Dieses sogenannte *Defense-in-Depth* Paradigma wird von Wissenschaft und Industrie gleichermaßen empfohlen [MV14] [ZP20] [EM16], wobei sich die einzelnen Schichten und Schutzmaßnahmen unterscheiden können. Abbildung 1.3 zeigt einen Defense-in-Depth Ansatz, angelehnt an [MV14] und [EM16].

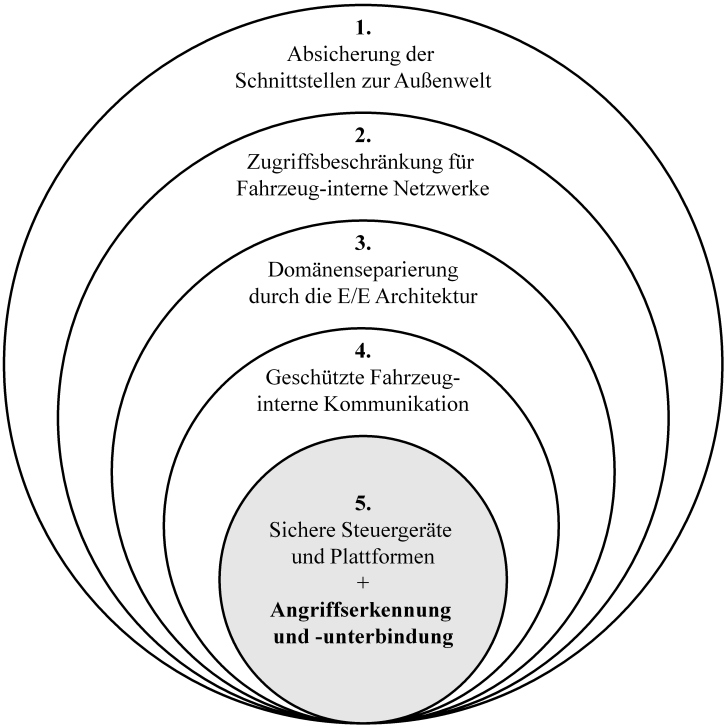

Bild 1.3: Mögliche Security-Maßnahmen

1. Um den entfernten Zugriff auf Steuergeräte zu erschweren (erste Angriffs-Stufe), sind die informationstechnischen Schnittstellen zur Außenwelt abzusichern. Dabei ist es wichtig, nicht benötigte Protokolle und Funktionen abzuschalten, um so die Angriffsfläche auf ein Minimum zu verringern. Die Absicherung verbleibender Schnittstellen kann mittels sicherer Protokolle wie Transport Layer Security (TLS) und entsprechenden kryptographischen Algorithmen erfolgen.

2. Sollte ein Angreifer die erste Stufe überwinden und Zugriff auf ein Steuergerät erhalten, ist zu verhindern, dass dieses unerlaubte Nach-

richten in Fahrzeug-internen Netzwerken versendet. Ein Infotainment-System benötigt keinen Zugriff auf die Bremsen eines Fahrzeugs und sollte daher auch keine entsprechenden Nachrichten versenden können.

3. Die Auslegung des elektrischen und elektronischen Systems (siehe *E/E-Architektur* in Abschnitt 2.1.3) spielt ebenfalls eine entscheidende Rolle. Moderne Fahrzeugplattformen setzen auf eine Trennung der unterschiedlichen Funktionsbereiche (Domänen). Dabei sorgen physikalisch getrennte Netzwerke für eine Separierung der Domänen. Infotainment-Steuergeräte können damit zum Beispiel nicht direkt mit Steuergeräten des Antriebsstrangs kommunizieren. Ist ein Informationsaustausch unbedingt notwendig, regeln sogenannte Gateways den Datenfluss. Der On-Board-Diagnose (OBD) Zugang sollte über einen privaten CAN-Bus realisiert werden, um einfaches Mitlesen und Manipulationen der regulären Kommunikation zu verhindern. Somit erschweren Schicht zwei und drei gemeinsam die zweite, zuvor erwähnte Angriffsstufe.

4. Hat ein Angreifer Zugriff auf ein sicherheitsrelevantes, Fahrzeug-internes Netzwerk erlangt, verhindert Schicht vier ein einfaches Manipulieren von ausgetauschten Daten. Eine Möglichkeit besteht in der kryptografischen Absicherung kritischer Nachrichten. Dabei sind die eigentlichen Nutzdaten in der Regel nicht verschlüsselt. Stattdessen wird unter der Verwendung eines geheimen Schlüssels ein Message Authentication Code (MAC) berechnet und der Nachricht angehängt. Dieser stellt die Authentizität und Integrität sicher, siehe Abschnitt 3.2.1.

5. Schicht vier ist nur intakt, solange die verwendeten Schlüssel geheim blieben. Um dies zu gewährleisten, ist eine sichere Schlüsselablage auf den Steuergeräten beziehungsweise in spezieller Hard-

ware notwendig. Des Weiteren enthält Schicht fünf weitere Security-Maßnahmen zur Absicherung von Steuergerätefunktionalität. Dazu zählen unter anderem Firewalls und Intrusion Detection Systeme (IDS, siehe Abschnitt 3.2.2) zur Erkennung von Fehlverhalten und Manipulationen, wie sie beispielsweise verschiedene Wissenschaftler für den CAN-Bus empfehlen [MV14] [HKD11]. Schicht vier und fünf erschweren somit das Auslösen von (sicherheitskritischem) Fehlverhalten (dritte Angriffs-Stufe) oder versuchen zumindest dessen Auftreten zu erkennen.

Firewalls sind Teil der Absicherungsmaßnahmen von Schicht eins und zwei (Bild 1.3). Zusätzlich sorgen sie in Schicht fünf dafür, dass sich Steuergeräte bis zu einem gewissen Grad eigenständig schützen können, auch wenn alle anderen Security-Barrieren vom Angreifer bereits überwunden wurden. Ergänzend dazu bieten IDS in Schicht fünf die Möglichkeit Cyberangriffe beziehungweise Auffälligkeiten zu erkennen und Daten darüber zu sammeln. Diese Daten helfen bei der Weiterentwicklung und Absicherung des E/E-Systems und können im Extremfall zu sofortigen Gegenmaßnahmen führen. Daher sind diese Systeme, neben den anderen vorgestellten Security-Maßnahmen, für moderne und zukünftige Fahrzeuge von besonderem Interesse.
Beim autonomen Fahren spielt neben Technologiesprüngen in den Bereichen Sensorik, Vernetzung und Datenverarbeitung auch das Thema Absicherung eine entscheidende Rolle. Hier gibt es auf zwei Ebenen Paradigmenwechsel: Einerseits sind pass/fail-Kriterien für komplexe Fahrerassistenzsysteme oftmals nicht mehr ausreichend. Wann ist beispielsweise ein autonomes Einparkmanöver erfolgreich abgeschlossen [Sax16]? Welcher Abstand zum Bordstein, zum Hinter- bzw. Vordermann und welcher Winkel zum Bordstein wird noch als akzeptabel angesehen? Auf der anderen Seite steigt die Anzahl der potentiellen Testfälle dramatisch an, mit weitreichenden Folgen für die Absicherung von Fahrzeugfunktionen. Ein autonomes

Einparkmanöver kann in fast beliebig vielen Ausgangssituationen starten, auch wenn nur wenige Parameter betrachtet werden, wie Länge und Breite der Parklücke und der seitliche Abstand zu den bereits parkenden Fahrzeugen [Sax16]. Selbst in diesem vermeintlich einfachen Beispiel ist ein Testen aller möglichen Fälle praktisch nicht mehr darstellbar. Ein autonomes Fahrzeug wird also im Laufe seines Lebenszyklus mit sehr vielen Situationen konfrontiert, die zur Entwicklungszeit nicht explizit getestet wurden. Eine weitere, noch zu lösende Aufgabe in diesem Bereich ist die Absicherung von adaptiven Applikationen und Algorithmen, die zur Laufzeit lernen und sich selbstständig anpassen. Eine vollständige Absicherung der Funktionen und Algorithmen für das autonome Fahren zur Entwicklungszeit ist momentan nur schwer vorstellbar und stellt Fahrzeughersteller sowie Zulieferer vor große Herausforderungen. Ein möglicher Baustein zur Lösung dieses Problems ist die Überwachung der Fahrzeugelektronik zur Laufzeit. Wird eine ungewöhnliche Reaktion eines Steuergeräts oder des gesamten Fahrzeugs erkannt, kann beispielsweise eine extreme Beschleunigung verhindert oder das Fahrzeug in einen sicheren Zustand überführt werden, um die Auswirkungen des Fehlverhaltens zu minimieren.

Beide aufgezeigten Problemstellungen haben eine Gemeinsamkeit: Ein Steuergerät weicht vom erwarteten Sollverhalten ab. Im Fall eines Cyberangriffs werden beispielsweise CAN-Botschaften versendet, die normalerweise nicht oder nur in einer deutlich geringeren Frequenz auftreten oder es wird manipulierter Code auf einem Steuergerät ausgeführt. Beide Abweichungen vom Sollverhalten sind auf das Absicherungs-Problem übertragbar. Sollte ein Algorithmus fehlerhaft reagieren, wäre dies möglicherweise an ungewöhnlichem Laufzeitverhalten beobachtbar. Daher beschäftigt sich diese Arbeit mit der Überwachung von charakteristischen Eigenschaften eines Steuergeräts in Echtzeit. Im Fokus steht zunächst das Protokollieren der Abweichungen vom Sollverhalten. Auf Basis dieser Daten lassen sich zukünftig gegebenenfalls aktive Gegenmaßnahmen ableiten. Ziel ist es, die Gesamtsicherheit von Steuergeräten in den Ausprägungen Security und

Safety zu verbessern und einen Beitrag zum sicheren (autonomen) Fahren zu leisten.

1.2 Struktur der Arbeit

Die vorliegende Arbeit gliedert sich in neun Kapitel. Direkt im Anschluss an diesen Absatz, erläutert Kapitel zwei alle notwendigen technischen und begrifflichen Grundlagen, die zum Verständnis der Arbeit erforderlich sind. Danach analysiert Kapitel drei den aktuellen Stand der Technik in Wissenschaft und Industrie bezüglich Safety und Security in modernen automobilen E/E-Architekturen. Das vierte Kapitel stellt zunächst die grundlegende Idee eines Automotive Observers vor und geht nachfolgend auf die einzelnen Bestandteile des Konzepts ein. Es folgt eine detaillierte Betrachtung von selbstlernenden Algorithmen zur Plausibilisierung von Signalverläufen. Die resultierenden Herausforderungen bei der Anwendung dieser Algorithmen auf Realdaten diskutiert Kapitel fünf. Nach einer Leistungsbewertung in Kapitel sechs beschreibt Kapitel sieben die prototypische Realisierung innerhalb eines AUTOSAR-basierten Steuergeräts. Kapitel acht gibt eine abschließende Zusammenfassung bevor Kapitel neun die Arbeit mit einem Ausblick auf zukünftige Forschungsfragen abrundet.

2 Technische und begriffliche Grundlagen

Dieses Kapitel gibt eine kurze Einführung in alle, für die vorliegende Arbeit relevanten Themengebiete. Damit sind die nachfolgenden Inhalte auch für fachfremde Leser verständlich. Wird mehr Detailwissen benötigt, verweisen die einzelnen Unterkapitel auf entsprechende Literatur.

2.1 Fahrzeugelektronik

Die Geschichte des modernen Automobils beginnt 1885 mit dem Benz Patent-Motorwagen, Typ 1 von Carl Benz, dem ersten motorbetriebenen und alltagstauglichen Dreirad. Das dazugehörige Patent DRP 37435 von 1886 wird als Geburtsurkunde des Automobils angesehen [Daia] [Daib]. Quasi zeitgleich entwickelte Gottlieb Daimler den Reitwagen, das erste von einem Benzinmotor angetriebene Motorrad.

Bild 2.1: Carl Benz[1]

In den folgenden Jahrzehnten fand eine rasante Entwicklung statt und das Automobil wurde mit immer mehr elektrischen Komponenten ausgestattet, wie beispielsweise der Beleuchtungsanlage. Bild 2.2 zeigt das gesamte elektrische Bordnetz eines Volkswagen Typ 1 aus den 1950er Jahren (Käfer).

[1] Quelle: https://www.daimler.com/konzern/tradition/gruender-wegbereiter/carl-benz.html
[2] Quelle: http://www.volkswagen-classic.de/pdf/reparaturleitfaeden/leitfaden_typ_1_1952_57/repleitfaden_52_57_kapitel_E.pdf

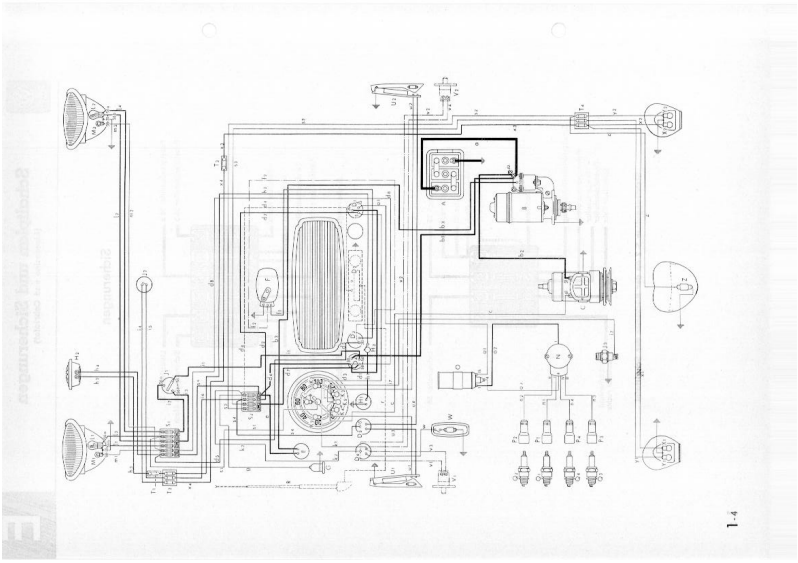

Bild 2.2: Stromlaufplan des Volkswagen Typ 1[2]

2.1.1 Steuergeräte

Mit der Benzineinspritzung D-Jetronic von Bosch hielt 1967 die Elektronik Einzug ins Automobil. Der Hauptgrund waren die damals eingeführten, strengen Abgasauflagen im US-Bundesstaat Kalifornien, deren Erfüllung einen geringen Benzinverbrauch voraussetzte [Rob]. Die Steuer- bzw. Regelungsaufgaben übernimmt dabei das sogenannte Steuergerät.

Peter Scholz definiert den Begriff Steuergerät wie folgt [Sch05]: „Ein Steuergerät (engl.: Electronic Control Unit, kurz: ECU) ist die physikalische Umsetzung eines eingebetteten Systems. Es stellt damit die Kontrolleinheit eines mechatronischen Systems dar. In mechatronischen Systemen bilden Steuergeräte und Sensorik/Aktorik oft eine Einheit.“

Im gleichen Buch beschreibt er ein eingebettetes System als „ein in ein umgebendes technisches System eingebettetes und mit diesem in Wechselwirkung stehendes Computersystem."
Heute enthalten fast alle Steuergeräte als zentrales Element einen Mikrocontroller oder Mikroprozessor, auf dem anwendungsspezifische Applikationssoftware ausgeführt wird, welche Steuerungs-, Regelungs-, Überwachungs- und Datenverarbeitungsaufgaben ausführt [Sch05].
Weitere automobile Entwicklungen auf Elektronik- und damit ECU-Basis folgten in den Jahren danach, wie beispielsweise das Antiblockiersystem (ABS). ABS gab es bereits in den frühen 1970er Jahren. Das 1978 von Bosch entwickelte und mit Halbleiter-Technik realisierte System war allerdings das erste, welches eine separate Regelung aller vier Räder erlaubte [Sch13].

2.1.2 Vernetzung

Die elektronischen Systeme und deren Steuergeräte waren zu Beginn isoliert oder tauschten Daten über Punkt-zu-Punkt Verbindungen aus. Die Komplexität der Verkabelung, des sogenannten Kabelbaums, nahm aufgrund der Punkt-zu-Punkt Verbindungen mit steigendem Datenaustausch exponentiell zu, was erhöhte Kosten, erhöhtes Gewicht und erhöhten Kabelverlegeaufwand verursachte. Dies führte schließlich zur Entwicklung des *Controller Area Network* (CAN) durch die Robert Bosch GmbH in den 1980er Jahren. CAN war der erste Kommunikationsbus für den Datenaustausch zwischen automotive Steuergeräten und wurde 1991 zum ersten Mal in Fahrzeugen eingesetzt. CAN gibt es mittlerweile in verschiedenen Ausprägungen mit unterstützten Datenraten bis 1 Mbit/s [ZS14].
Durch die stetig steigenden und immer vielfältigeren Anforderungen an die Kommunikation innerhalb des Fahrzeugs, entwickelte die Automobilindustrie im Laufe der Zeit unterschiedliche Bussysteme und Netzwerktechno-

logien. Nach CAN folgte Ende der 1990er Jahre das *Local Interconnect Network* (LIN) als einfaches und kostengünstiges Bussystem für die Anbindung von simpler Sensorik und Aktorik, wie beispielsweise einem Schaltermodul. Dementsprechend niedrig sind auch die Anforderungen an die Bandbreite, welche bei 20 kbit/s begrenzt ist [ZS14].
Anfang der 2000er hatten verschiedene Fahrzeughersteller das Ziel sogenannte X-by-Wire Systeme einzuführen. Diese Systeme verzichten auf die mechanische Kopplung zwischen Sensorik und Aktorik. Stattdessen regelt ein Steuergerät die Aktorik basierend auf verschiedenen Sensorwerten. Ein prominentes Beispiel hierfür ist Steer-by-Wire, das ohne mechanische Kopplung von Lenkrad und Lenkgetriebe auskommt. Aus diesen Anforderungen entstand das redundant ausgelegte und zeitgesteuerte (time-triggered) Bussystem *FlexRay*, welches zudem eine erhöhte maximale Datenrate von zweimal 10 Mbit/s bietet [ZS14].
Für die Ende der 1990er und Anfang der 2000er aufkommenden Multimedia Anwendungen bieten alle bisher erwähnten Bussysteme nicht genügend Bandbreite. Daher schauten sich die Automobilhersteller nach Alternativen um und wurden mit *Media Oriented Systems Transport* (MOST) fündig. MOST ist speziell für die Audio-/Video-Übertragung ausgelegt und basiert auf einer Ringtopologie. Dabei ist jedes kommunizierende Steuergerät physikalisch mit einem Vorgänger und einem Nachfolger verbunden, siehe Bild 2.4 (rechts-oben). Heute bietet die Technologie eine maximale Bandbreite von 150 Mbit/s bei der Verwendung von optischer Übertragungstechnik, was eine Realisierung vergleichsweise teuer macht.
Die neueste Netzwerktechnik innerhalb des Fahrzeugs ist *Ethernet*. Durch die immer größer werdende Menge an Software und den damit verbundenen, längeren (Re-)Programmierzeiten[3] von Steuergeräten, beispielsweise am Ende der Produktionslinie, wurde Ende der 2000er Ethernet als Fahr-

[3] Die *Programmierung* eines Steuergeräts bezieht sich in diesem Fall nicht auf die Erstellung des Quellcodes sondern auf das Herunterladen der Software in den Speicher des Mikrocontrollers/-prozessors (engl.: Flash-Download).

zeugzugang von einigen Automobilherstellern eingeführt. Im Vergleich zum bis dato eingesetzten CAN-Bus konnte man durch Ethernet und der unterstützten Bandbreite von 100 Mbit/s eine dramatische Reduzierung der (Re-)Programmierzeiten erreichen. Anfang der 2010er Jahre fand Ethernet auch Einzug in die Fahrzeug-interne Vernetzung und nicht nur als Fahrzeugzugang. Maßgebliche Treiber für diese Entwicklung sind Anforderungen aus den Infotainment- und Fahrerassistenzbereichen. Im Infotainmentbereich verspricht Ethernet eine deutliche Kostenreduktion im Vergleich zu MOST. Dies ist vor allem in der automotive-spezifischen Bitübertragungsschicht 100BASE-T1 begründet. Sie erlaubt eine Datenübertragungsrate von 100 Mbit/s full-duplex über ein ungeschirmtes, verdrilltes Adernpaar [IEEa]. Ein weiterer Anwendungsfall ist die Anbindung neuartiger Sensoren für teil- und vollautomatisiertes Fahren, wie Radare, Kameras und Laser-Scanner. Diese Sensoren erzeugen eine große Menge an Daten, die zur Verarbeitung an die entsprechenden Steuergeräte übertragen werden müssen. Hierfür kommt ebenfalls 100BASE-T1 zum Einsatz oder die entsprechende Gigabit-Variante 1000BASE-T1 [IEEb] - weiterhin unter Verwendung eines einzelnen Adernpaars. Momentan wird bereits über Datenraten größer 1 Gbit/s [IEEc] und über 10 Mbit/s [IEEd] diskutiert. Letztere hätte bei entsprechend geringen Kosten das Potential CAN (teilweise) abzulösen.

2.1.3 Elektrik/Elektronik-Architektur

Als CAN für die Kommunikation zwischen Steuergeräten eingeführt wurde, bestand die gesamte Elektrik/Elektronik (E/E)-Architektur eines Fahrzeugs aus einem Bus, über den alle Steuergeräte miteinander verbunden waren [ZS14]. Durch das Einbringen neuer E/E-Komponenten in jeder Fahrzeuggeneration sowie durch die vermehrte Vernetzung von Steuergeräten und Funktionen stieg der Kommunikationsbedarf rapide an. Ein CAN-Bus war daher schnell ausgelastet und ein zweiter wurde verlegt. Heutige E/E-Architekturen sind heterogen und bestehen aus einer Vielzahl von Bussyste-

men und Netzwerken unterschiedlicher Technologien und Topologien. Um das E/E-System besser zu strukturieren, ist ein Teilnetz in der Regel einem bestimmten Bereich - einer sogenannten Domäne - zugeordnet. Dadurch gibt es beispielsweise einen CAN-Bus für Motor und Getriebe (Powertrain), einen für den Fahrzeug-Innenraum (Comfort), einen FlexRay für Fahrwerkskomponenten (Chassis) und einen MOST-Ring für Multimediaanwendungen (Infotainment). Einzelne Steuergeräte können auch an mehrere Kommunikationsbusse und -netzwerke angeschlossen sein, je nach benötigten Daten und realisierter Funktion.
Werner Zimmermann und Ralf Schmidgal fassen den Sachverhalt in ihrem Buch zusammen und geben dabei implizit eine Definition für den Begriff der E/E-Architektur [ZS14]:

„Das Konzept für die Aufteilung der Fahrzeugfunktionen auf verschiedene Steuergeräte, deren Vernetzung, die Optimierung der Einbauorte sowie die Verteilung und Steuerung der notwendigen elektrischen Energie im Fahrzeug, die sogenannte Elektrik/Elektronik-(E/E)-Architektur, ist heute eine der komplexesten Aufgaben bei der Entwicklung einer Fahrzeugfamilie."

Gleichzeitig ist die E/E-Architektur ein Wegbereiter für die Elektronik- und inzwischen hauptsächlich Software-basierten Innovationen im Fahrzeug der letzten Jahre. Bild 2.3 zeigt das exponentielle Wachstum an diesen Funktionen.

Zentrales Gateway

Fahrzeughersteller stießen mit dem zuvor beschriebenen, heterogenen Netzwerk ebenfalls an Grenzen. Mit der Zeit wurden immer mehr Daten zwischen den unterschiedlichen Domänen ausgetauscht. Zum Beispiel zeigt ein zentraler Bildschirm dem Fahrer Innenraumdaten an und das elektronische

[4] Quelle: Vector Informatik GmbH

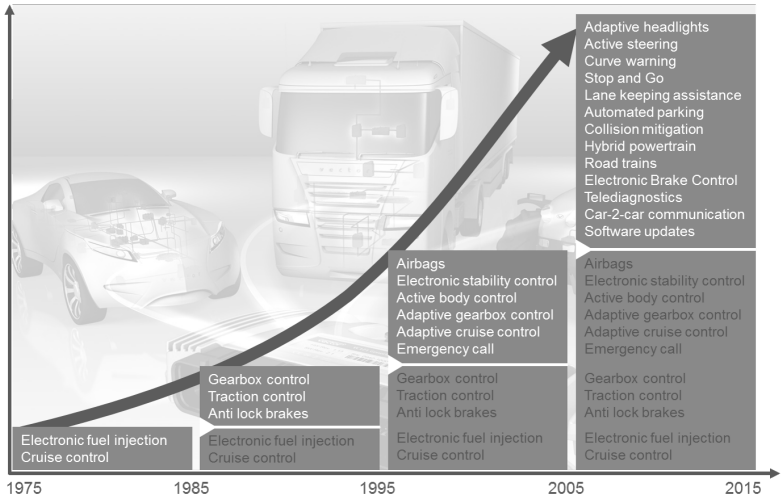

Bild 2.3: Exponentielles Wachstum an E/E-Funktionen[4]

Stabilitätsprogramm (ESP) sendet Daten an Motor und Getriebe. Um Kommunikation zwischen Domänen realisieren zu können, wurden verschiedene Steuergeräte an mehrere Bussysteme angeschlossen. Diese speziellen Steuergeräte arbeiten als Übersetzer zwischen verschiedenen Bussystemen und werden als Gateways bezeichnet. Mit der Zunahme an Gateways steigt allerdings auch die Komplexität der E/E-Architektur. Die vergleichsweise unstrukturierte Kopplung verschiedener Bussysteme ist auf Dauer nur schwer beherrschbar. Besonders herausfordernd ist dabei der Umgang mit unterschiedlichen Fahrzeugvarianten und -ausstattungen. Das Hinzufügen oder Entfernen eines Steuergeräts aufgrund einer gewählten Fahrzeugkonfiguration kann Auswirkungen auf große Teile der E/E-Architektur haben. Zur Reduktion der Komplexität und Steigerung der Beherrschbarkeit setzen daher viele Fahrzeughersteller heute auf E/E-Architekturen mit zentralem Gateway. Das zentrale Gateway ist als spezielles Steuergerät an alle im Fahrzeug verbauten Netzwerke angeschlossen, mit Ausnahme von

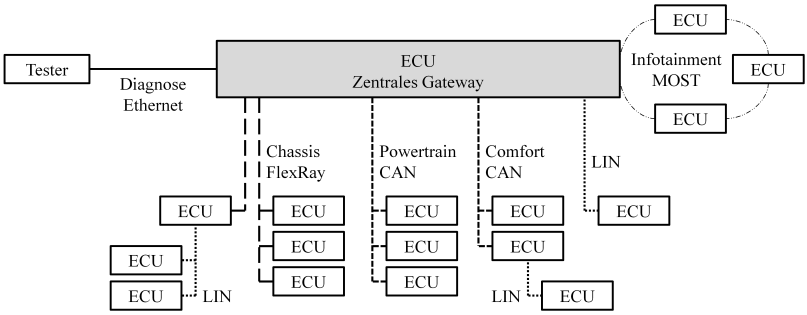

Bild 2.4: E/E-Architektur mit zentralem Gateway

privaten Verbindungen und beispielsweise LIN-Busen zur reinen Sensor- und Aktor-Ansteuerung. Jegliche Kommunikation zwischen den Netzwerken findet über das zentrale Gateway statt. Damit sind die unterschiedlichen Domänen auch in der E/E-Architektur deutlich voneinander getrennt. Zudem ist normalerweise der Diagnosezugang über das zentrale Gateway geregelt, um unter anderem den direkten Zugriff auf Fahrzeug-interne Kommunikation zu verhindern oder zu erschweren. An diesem Beispiel wird deutlich, dass bereits die E/E-Architektur selbst maßgeblich zur Angriffssicherheit beiträgt (siehe Bild 1.3). In Bild 2.4 ist eine E/E-Architektur mit zentralem Gateway beispielhaft und vereinfacht dargestellt. Das zentrale Gateway regelt hier den Datenaustausch zwischen den Domänen Chassis, Powertrain, Comfort und Infotainment sowie die Anbindung eines Testers zur Fahrzeugdiagnose.

Domänenarchitektur

Unter anderem führt die Einführung und stetige Weiterentwicklung von Fahrerassistenzfunktionen zu einem weiteren, starken Anstieg der notwendigen Inter-Domänen-Kommunikation. Dadurch wird in der zuvor diskutierten E/E-Architektur das zentrale Gateway selbst zum Problem. In einem aktuellen Fahrzeug eines Premium-Herstellers verbindet dieses Steuergerät bei-

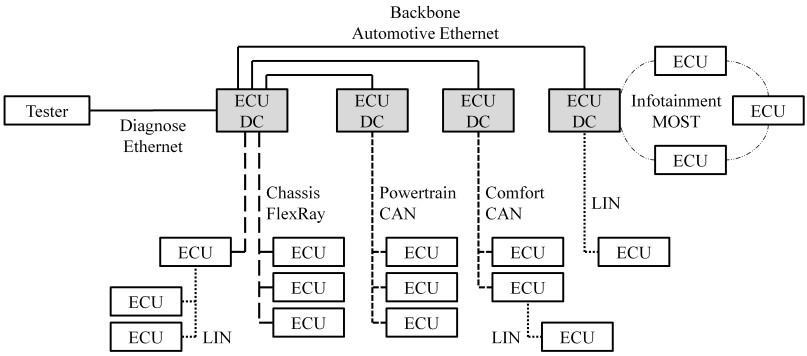

Bild 2.5: E/E-Architektur mit vier Domänencontrollern

spielsweise bis zu zwei FlexRay-, sechs CAN- und zehn LIN-Busse sowie den Ethernet-basierten Diagnosezugang. Dies stellt heutige, dafür typischerweise verwendete Mikrocontroller vor große Herausforderungen, sowohl im Bereich der Rechenleistung als auch in Sachen Speicherbedarf[5]. Daher gehen viele Hersteller in der nächsten Fahrzeuggeneration zu einer sogenannten Domänenarchitektur über. Dabei erhält jede Domäne ein rechenstarkes Steuergerät, das zentrale Funktionen übernimmt. Dazu zählt die Anbindung der gekapselten Domäne an ein Hochgeschwindigkeitsnetzwerk, das alle Domänen miteinander verbindet - der sogenannte Backbone. Da für diesen Backbone meistens Automotive Ethernet zum Einsatz kommt, trägt die Entwicklung und Standardisierung dieser Netzwerktechnologie maßgeblich zur Verbreitung der Domänenarchitektur bei beziehungsweise ermöglichte in vielen Fällen erst ihre Verwirklichung. Bild 2.5 zeigt eine beispielhafte Domänenarchitektur. Ausgehend von Bild 2.4 wurde in diesem Beispiel das zentrale Gateway durch vier Domänencontroller ersetzt, die über Automotive Ethernet miteinander verbunden sind.

[5] Ein typischer Gateway Mikrocontroller ist der MPC5748G von Freescale/NXP mit 160 MHz und 756 kB RAM.

In einer speziellen Form dieser Architektur sind die einzelnen Domänen nicht den bisher diskutierten Funktionsbereichen wie Powertrain, Chassis usw. zugeordnet. Stattdessen wird das Fahrzeug in positionsabhängige Domänen aufgeteilt, wie beispielsweise vorne-links, vorne-rechts, hinten links und hinten-rechts. Mit diesem Ansatz lässt sich der Verkabelungsaufwand minimieren. In jeder sogenannten Zone ist ein leistungsstarker Domänenrechner verortet an den die umliegenden, kleineren Steuergeräte beziehungsweise Sensoren und Aktoren angeschlossen sind. Ein Nachteil der Zonen-Architektur ist die nicht vorhandene physikalische Trennung zwischen den Funktionsbereichen. Um zu verhindern, dass zum Beispiel Infotainment-Kommunikation die Steuerung des Antriebstrangs stört, ist die Trennung auf andere Art sicherzustellen. Bei Verwendung von Ethernet, bieten sich hier Virtual Local Area Networks (VLANs) an. VLANs sind logisch getrennte Netzwerke auf Basis einer gemeinsamen physikalischen Verkabelung. Zusätzlich kann auf zeitgesteuerte Kommunikation zurückgegriffen werden, wie sie momentan von der Time-Sensitive Networking (TSN) Arbeitsgruppe der IEEE standardisiert wird. Steuergeräte-seitig, speziell in den Domänenrechnern, ist ebenfalls auf eine strikte Trennung der Funktionsbereiche zu achten, sodass diese sich nicht gegenseitig negativ beeinflussen können. Neben Hardware-Maßnahmen wie der Verwendung von mehreren Prozessoren oder Prozessor-Kernen innerhalb eines Steuergeräts, lässt sich eine logische Trennung von Ausführungsbereichen in Software realisieren, siehe Abschnitt 2.1.5.

Zentralrechner

Die Domänenarchitektur ist ein erster Schritt in Richtung Funktionszentralisierung. In zukünftigen E/E-Architekturen wird dieser Trend vermutlich weiter zunehmen. Dafür gibt es unterschiedliche Gründe. Zum einen versuchen Fahrzeughersteller damit die Anzahl der verbauten Steuergeräte zu reduzieren, um Kosten zu sparen und die Komplexität der Vernetzung

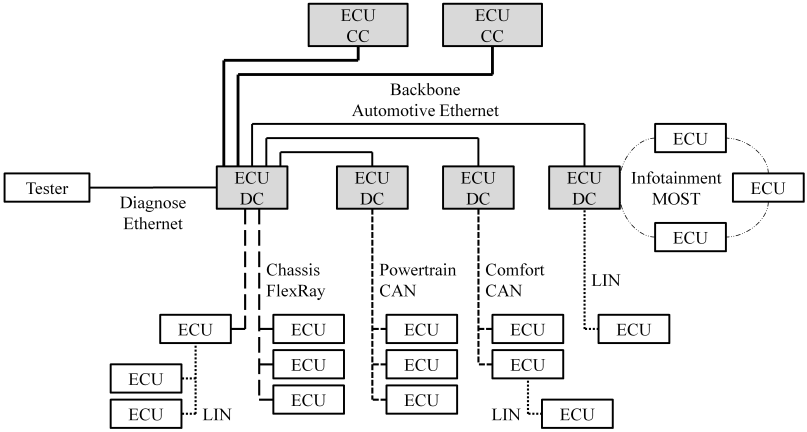

Bild 2.6: E/E-Architektur mit zwei Zentralrechnern

zu verringern. Zum anderen werden für hochautomatisierte Fahrfunktionen neue E/E-Architekturen benötigt, welche die benötigte Rechnenleistung zur Verfügung stellen und zusätzlich ein beherrschbares Systemdesign ermöglichen. Außerdem haben viele Fahrzeughersteller die Anforderung in Zukunft bestimmte Applikationen, darunter auch Fahrfunktionen, via Software Over The Air (SOTA) im Feld nachinstallieren zu können. Die erwähnten Ziele und Herausforderungen führen nach aktuellem Industrieverständnis zu einer E/E-Architektur mit sogenannten Zentralrechnern, auch Rechencluster oder Computing Cluster (CC) genannt. Diese Zentralrechner werden breitbandig z.B. via Gigabit-Ethernet an den Backbone angeschlossen und bieten im Vergleich zu klassischen automotive Steuergeräten eine sehr große Rechenleistung. Bild 2.6 erweitert die Domänenarchitektur aus Bild 2.5 um zwei Zentralrechner. Die dargestellte E/E-Architektur kann in vier Ebenen unterteilt und von oben nach unten gelesen werden, wobei die Abstraktion von oben nach unten abnimmt. Dies lässt sich anhand einer fiktiven Funktion für hochautomatisiertes Fahren erläutern. Auf Ebene 1 übernehmen Zentralrechner die längerfristige beziehungsweise strategische Fahrpla-

nung, unter Berücksichtigung aller zur Verfügung stehender Informationen. Dazu zählen neben den im Fahrzeug verbauten Sensoren wie Radar, LIDAR und Kameras beispielsweise auch Kartendaten, Car2X-Kommunikation und Informationen aus Mobilfunk-Verbindungen. Einige Hersteller sehen unter anderem für die strategische Fahrplanung die Cloud beziehungsweise ein Backend-System als Teil der E/E-Architektur eines Fahrzeugs. In diesem Fall könnte das Backend-System oberhalb der Zentralrechner als Ebene 0 in die gezeigte E/E-Architektur mit aufgenommen werden. Die Domänencontroller bilden Ebene 2 und sind für die konkrete Manöverplanung zuständig, wie zum Beispiel ein anstehender Spurwechsel. Klassische automotive Steuergeräte mit mittlerer Rechenleistung realisieren auf Ebene 3 die Ansteuerung und Regelung der einzelnen Fahrzeugkomponenten wie Motor, Getriebe, Fahrwerk und Bremsen. Auf unterster Ebene befinden sich die intelligenten Aktoren und Sensoren, welche mit den Steuergeräten kommunizieren.

Wie bereits erwähnt, spielen die Zentralrechner auch für nachladbare Software eine entscheidende Rolle. Heutige Steuergeräte (vgl. Ebene 3) sind sehr stark auf einen vordefinierten Anwendungsfall zugeschnitten. Dies betrifft sowohl die Auslegung der Hardware als auch die verwendete Software. Ein Motorsteuergerät kann beispielsweise zur Laufzeit keine zusätzlichen Funktionen wie die Getriebesteuerung übernehmen. Selbst mit einer kompletten Software-Aktualisierung wäre dies aufgrund der eingeschränkten Ressourcen und der fehlenden Anbindung von Sensorik sowie Aktorik nicht möglich. Sollen in Zukunft neue Fahrzeugfunktionen vom Endanwender im Feld nachinstallierbar sein, müssen diese auf einer höheren Abstraktionsebene realisiert werden. Die Zentralrechner sind dafür während der Entwicklungszeit entsprechend flexibel auszulegen. Einerseits muss die verwendete Hardware rechenstark und gegebenenfalls sogar erweiterbar sein, um Reserven für zukünftige Funktionen aufzuweisen. Andererseits muss auch die eingesetzte Software-Architektur das Nachinstallieren von neuen und zuvor unbekannten Funktionen erlauben.

Ein weiterer zentraler Punkt zukünftiger E/E-Architekturen ist die redundante Auslegung von Funktionen im Hinblick auf autonomes Fahren. Dazu zählt eine redundante Berechnung als auch eine redundante Kommunikation, beispielsweise mittels Ringtopologien.

2.1.4 Controller Area Network

Um eine bessere Einordnung von unterschiedlichen Netzwerktechnologien und deren Schichten zu ermöglichen hat sich das Open Systems Interconnection (OSI)-Referenzmodell etabliert. Das Modell wurde 1983 von der International Telecommunication Union (ITU) zum ersten Mal veröffentlicht [Int94a] [Inta]. 1984 folgte die Veröffentlichung als Standard der International Organization for Standardization (ISO) [Int94b]. Das OSI-Referenzmodell gliedert eine Netzwerktechnologie in sieben Schichten, wie Bild 2.7 zeigt. Mit ansteigender Schichtnummer nimmt dabei die Abstraktion zu. Im Folgenden sind die Hauptaufgaben der jeweiligen OSI-Schichten kurz zusammengefasst [Inta].

7	Anwendungsschicht (Application Layer)
6	Darstellungsschicht (Presentation Layer)
5	Sitzungsschicht (Session Layer)
4	Transportschicht (Transport Layer)
3	Vermittlungsschicht (Network Layer)
2	Sicherungsschicht (Data Link Layer)
1	Bitübertragungsschicht (Physical Layer)

Bild 2.7: OSI-Referenzmodell

1. Auf der untersten Ebene befindet sich die *Bitübertragungsschicht*. Dieser Schicht sind alle Mechanismen zugeordnet, welche dem Aufbau, Aufrechterhalten und Abbau des physikalischen Übertragungskanals dienen. Außerdem legt sie fest wie Bits, beispielsweise elektrisch oder optisch, übertragen werden.

2. Die darauf aufbauende *Sicherungsschicht* bietet die Möglichkeit Daten in einem bestimmten Format, einem sogenannten Frame, zwischen zwei Netzwerkteilnehmern auszutauschen. Außerdem regelt diese Schicht den Zugriff auf den Übertragungskanal und ist für die Erkennung von Fehlern in der Bitübertragungsschicht verantwortlich. Für die Fehlererkennung enthält ein Frame meistens ein Cyclic Redundancy Check.

3. Die *Vermittlungsschicht* stellt auf Ebene drei die Unabhängigkeit von Routingbeziehungen und damit von der darunter liegenden Netzwerktopologie her. Sie erlaubt eine Ende-zu-Ende Kommunikation aus Netzwerkteilnehmer-Sicht.

4. Daten von höheren Schichten werden durch die *Transportschicht* mit einer definierten Zuverlässigkeit verbindungslos oder verbindungsorientiert übertragen. Zusätzlich findet gegebenenfalls eine Segmentierung und Assemblierung der Daten sowie eine Flusskontrolle statt.

5. Der geordnete Datenaustausch, vergleichbar mit einer Transaktion, wird in der *Sitzungsschicht* realisiert. Ein Beispiel hierfür sind entfernte Funktionsaufrufe, sogenannte Remote Procedure Calls (RPC). Dabei sind Antworten eindeutig der passenden Anfrage zuzuordnen.

6. Die Aufgabe der *Darstellungsschicht* ist eine einheitliche Repräsentation von übertragenen Applikationsdaten, beispielsweise unabhängig von Betriebssystem und Byte-Order der Netzwerkteilnehmer.

7. Auf oberster Ebene im OSI-Modell stellt die *Anwendungsschicht* den Applikationen Funktionen und Dienste zur Verfügung. Die Applikationen selbst sind nicht Teil dieser Schicht.

Die Standardisierung von CAN durch die ISO deckt die OSI-Schichten eins und zwei ab. Auf der Bitübertragungsschicht gibt es zwei Ausprägungen: Den High Speed CAN (ISO 11898-2) [Int16a] sowie den Low Speed CAN (ISO 11898-3) [Int06]. Physikalisch wird die Differenzspannung zwischen den zwei Leitungen CAN-High (CAN_H) und CAN-Low (CAN_L) zur Bitübertragung verwendet. Im Falle von High Speed CAN entspricht eine Differenzspannung zwischen CAN_H und CAN_L von 2V einer logischen „0“. Bei einer logischen „1“ haben beide Leitungen das gleiche Potential. Dies entspricht dem rezessiven Pegel, der auch anliegt, sofern keine Kommunikation stattfindet. Das als Linien-Bus ausgelegte Kommunikationssystem muss an beiden Enden mit 120 Ohm terminiert werden, was dem Wellenwiderstand der verwendeten, verdrillten Zweidrahtleitung entspricht.

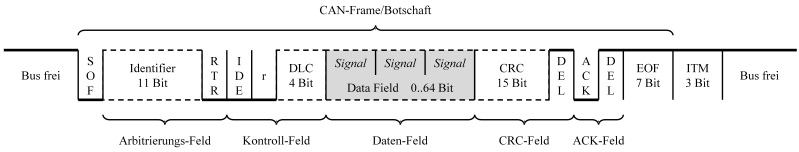

Bild 2.8: CAN 2.0A-Frame[6]/Botschaft auf OSI-Schicht zwei

Bild 2.8 zeigt den Aufbau eines Standard CAN-Frames auf OSI-Schicht zwei nach ISO 11898-1 [Int15].
Der Start eines Frames (*Start of Frame* (SOF)) wird durch eine dominante „0“ angezeigt.
Es folgt bei CAN 2.0A der 11 Bit lange *Identifier* (CAN-ID). Dieser erfüllt gleich mehrere Aufgaben. CAN verwendet eine Botschaftsadressierung, wodurch jeder CAN-ID eine bestimmte Nutzdatenstruktur zugeord-

[6] Quelle: Vector Informatik GmbH und in Anlehnung an [VWeb+18a]

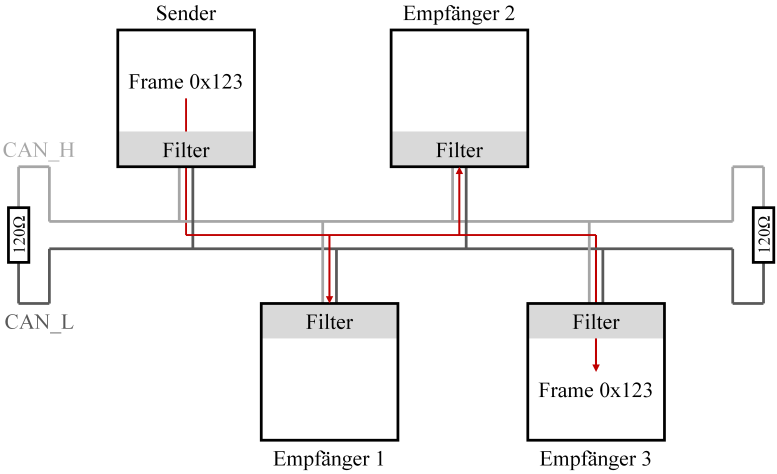

Bild 2.9: CAN-Bus

net ist. Ein Empfänger kann somit anhand der CAN-ID die Nutzdaten interpretieren. Da ein klassischer Bus zum Einsatz kommt, empfängt jeder angeschlossene Knoten alle CAN-Frames (Broadcast-Kommunikation) und muss die für ihn relevanten herausfiltern. Die Filterung basiert ebenfalls auf der CAN-ID und ist oftmals in Hardware realisiert. Damit regelt ein Knoten, welche Frames weiter prozessiert und welche zu einem sehr frühen Zeitpunkt verworfen werden. Ein Beispiel dafür ist in Bild 2.9 dargestellt. Der Frame mit der CAN-ID 0x123 wird potentiell von drei Knoten empfangen. Allerdings ist nur bei Empfänger drei der Filter so konfiguriert, dass der Frame diesen passiert. Eine weitere Aufgabe ist die Priorisierung. Je kleiner die CAN-ID, d.h. je früher eine logische „0“ übertragen wird, desto höher die Priorität des Frames. Während der Übertragung der CAN-ID, der sogenannten Arbitrierungsphase, liest jeder Knoten den aktuellen Buspegel zurück und überprüft, ob der gelesene Pegel mit dem gerade von ihm gesendeten Pegel übereinstimmt. Falls nicht, stellt er das Senden ein. Da sich par-

allel sendende Knoten unauffällig zurückziehen, kann die Übertragung des höchstprioren Frames regulär fortgesetzt werden. Das Buszugriffsverfahren ist damit Carrier Sense Multiple Access/Collision Resolution (CSMA/CR).
Nach der CAN-ID kommt das *Remote Transmission Request* (RTR)-Bit. Mit diesem Bit kann ein Knoten das Versenden des Frames mit der gleichen CAN-ID veranlassen. Ein RTR Frame enthält selbst keine Daten. Dieser Mechanismus hat im automotive Umfeld keine Relevanz.
Das *Identifier Extension* (IDE) Bit zeigt an, ob die 11 Bit CAN-ID um weitere 18 Bit ergänzt wird, welche gegebenenfalls direkt auf das IDE Bit folgen (CAN 2.0B). Im Falle der erweiterten Adressierung mit 29 Bit CAN-IDs ist das ursprüngliche, bereits diskutierte RTR Bit rezessiv zu senden. Nach den zusätzlichen 18 Bit folgt das dann gültige RTR, gefolgt von zwei reservierten Bits. Anderenfalls, d.h. bei Standard 11 Bit CAN-IDs, folgt auf das IDE ein reserviertes Bit.
Der anschließende *Data Length Code* (DLC) gibt die Länge der übertragenen Nutzdaten in Bytes an.
Das *Data Field* enthält null bis acht Byte Nutzdaten, die wiederum in *Signale* unterteilt sind. Ein Signal repräsentiert eine dedizierte zu übertragende Information und besteht bei CAN aus minimal einem und maximal 64 Bit. Beispiele hierfür sind Fahrzeuggeschwindigkeit, Motordrehzahl, Temperatur aber auch reine Statusinformationen wie die aktuelle Gangwahl oder eine Schalterstellung. Die konkrete Instanz innerhalb eines Frames ist ein Signalwert.
Um Übertragungsfehler zu erkennen, enthält ein CAN-Frame eine Prüfsumme, den *Cyclic Redundancy Check* (CRC). Das Generatorpolynom gibt der Standard vor. Abgeschlossen wird das CRC-Feld mit einem rezessiven *Delimiter* (DEL)-Bit.
Wurde ein CAN-Frame bis zu diesem Punkt erfolgreich übertragen, sendet der Empfänger im *Acknowledge* (ACK)-Slot einen dominaten Pegel. Der eigentliche Sender des Frames, setzt dieses Bit rezessiv. Quittiert ein Empfänger den Frame positiv durch ein dominantes ACK-Bit, kann dennoch nicht

davon ausgegangen werden, dass der Frame von allen Konten erfolgreich empfangen wurde. Das ACK-Feld wird wiederum mit einem DEL-Bit abgeschlossen.
Das Ende des Frames (*End of Frame* (EOF)) ist durch sieben rezessive Bits gekennzeichnet.
Bevor ein neuer Frame starten kann, sind weitere drei rezessive Bits als *Intermission* (ITM) einzuhalten.
Zwei Besonderheiten des CAN-Bus sind noch zu erwähnen:

1. Zur Bittaktsynchronisation der Empfänger auf den Sender müssen unter Umständen sogenannte Stuff-Bits künstlich eingefügt werden. Diese erzwingen einen Pegelwechsel nach fünf gleichen Bits und werden von den Empfängern in Hardware wieder herausgefiltert bevor der Frame weiter verarbeitet wird.

2. Sollte ein Knoten einen Fehler bei der Übertragung eines CAN-Frames feststellen, sendet dieser einen sogenannten Error-Frame, bestehend aus sechs dominanten Bits und überschreibt damit die eigentlichen Daten. Da sechs dominante Bits einen Verstoß gegen das Bit-Stuffing bedeuten, erkennen alle Knoten die fehlerhafte Übertragung und verwerfen den Frame. Damit wird die Datenkonsistenz sichergestellt. Um zu verhindern, dass eine fehlerhafte CAN-Zelle (der Teil des Mikrocontrollers, der die CAN-Schnittstelle auf Schicht zwei in Hardware umsetzt) den gesamten Bus lahm legt, beispielsweise durch ungerechtfertigtes Senden von Error-Frames, überwacht sich jede CAN-Zelle selbst und nimmt sich gegebenenfalls eigenständig vom Bus. Dieser Mechanismus beruht auf separaten Fehlerzählern für Sende- und Empfangsrichtung.

Bosch spezifizierte 2012 eine Erweiterung des CAN-Bus, die es erlaubt bis zu 64 Nutzdatenbytes zu übertragen. Gleichzeitig kann die Taktung im Control Field, Data Filed und Check Field auf über 1 Mbit/s erhöht wer-

den. Diese Erweiterung ist unter dem Namen CAN with Flexible Data Rate (CAN FD) [Lin12] bekannt und fließt in die ISO 11898 Standard-Reihe mit ein. Eine Übersicht und weitere Details zu CAN (FD) geben Werner Zimmermann und Ralf Schmidgal [ZS14].
Wie bereits erwähnt, bezieht sich die Standardisierung des CAN-Bus auf die OSI-Schichten eins und zwei. Als Vermittlungs- und Transportschicht existiert das CAN Transport Protocol (CAN-TP), welches hauptsächlich bei der Fahrzeugdiagnose Anwendung findet und durch die ISO 15765-2 standardisiert ist [Int16b]. Im Rahmen von *Unified Diagnostic Services* (UDS) standardisiert die ISO auch große Teile der OSI-Schichten fünf bis sieben für den Anwendungsfall der Fahrzeugdiagnose[7] [Int13a] [Int13b] [Int12a].
Der applikative Datenaustausch zwischen Steuergeräten findet üblicherweise direkt über die Signale im Datenfeld von CAN-Frames statt. In diesem Fall existieren keine standardisierten Protokolle für die OSI-Schichten drei bis sieben. Die Aufgabe der Darstellungsschicht übernimmt implizit die sogenannte *Kommunikationsmatrix*, in welcher ein Fahrzeughersteller unter anderem den exakten Aufbau der Nutzdaten sowie die darin enthaltenen Signale semi-formal beschreibt. Damit ist die systemweite Interpretierbarkeit sichergestellt. Die Anwendungsschicht wird durch eine einheitliche Schnittstelle zwischen Applikations- und Basissoftware realisiert, siehe nachfolgende Abschnitte. Sollten für einzelne Anwendungsfälle zusätzliche Kommunikationsmechanismen notwendig sein, wie beispielsweise eine Session-Verwaltung, sind diese applikativ umzusetzen.

2.1.5 Steuergeräte-Software

Für Steuergeräte auf verschiedenen E/E-Architekturebenen gibt es unterschiedliche Anforderungen an die eingesetzte Software. Intelligente Sensoren und Aktoren sowie andere Kleinststeuergeräte auf unterster Ebene haben oftmals nur eine dedizierte Aufgabe. Die dafür notwendige Software imple-

[7] Für abgasrelevante Systeme gelten gesonderte Anforderungen und Standards.

mentiert neben der eigentlichen Logik beispielsweise einen LIN-Slave, um Sensordaten weiterzuleiten beziehungsweise neue Aktordaten zu empfangen. Da sowohl Umfang als auch Komplexität meistens gering sind, kommt bei diesen Steuergeräten oft kein vollständiges Betriebssystem zum Einsatz sondern die Software läuft direkt auf dem Mikrocontroller („bare-metal“). Für klassische automotive Steuergeräte mit 16 oder 32 Bit Mikrocontroller auf Ebene 3 wird üblicherweise ein Echtzeitbetriebssystem verwendet. Dieses erlaubt das Ausführen mehrerer Tasks und abstrahiert in Teilen die darunter liegende Hardware.

1993 gründeten große Teile der deutschen Automobilindustrie das Standardisierungsgremium *Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug* (OSEK). Das Gremium schloss sich 1994 mit dem französischen Pendant *Vehicle Distributed Executive* (VDX) zusammen, sodass die offizielle Abkürzung OSEK/VDX entstand. Eines der bedeutendsten Arbeitsergebnisse ist die Spezifikation des gleichnamigen Betriebssystems (OSEK/VDX-OS), welche 2005 auch als ISO 17356-3 [Int05a] veröffentlicht wurde. Dieses Betriebssystem wird statisch konfiguriert. Alle Tasks, Ressourcen usw. sind vor dem Kompiliervorgang festzulegen und können zur Laufzeit nicht verändert werden. Daher ist unter anderem ein dynamisches Hinzufügen von Tasks nicht möglich.

OSEK/VDX bildet die Grundlage der Standardisierung einer Software-Architektur für automotive Steuergeräte. Diese Arbeit wird seit Anfang der 2000er Jahr von der *Automotive Open Systems Architecture* (AUTOSAR) Partnershaft mit internationalem Fokus weitergeführt. AUTOSAR spezifiziert und standardisiert dabei nicht nur das Betriebssystem und weitere Software-Module sondern eine komplette Software-Architektur und ein entsprechendes Vorgehensmodell. Die Partnerschaft umfasst momentan neun Core Partner (BMW Group, Bosch, Continental, Daimler, Ford, General Motors, PSA Group, Toyota und Volkswagen Group), 50 Premium Partner, 33 Development Partner, 113 Associate Partner und 20 Attendees [AUT]. Bis 2017 lag der Fokus der AUTOSAR Partnerschaft auf dem direk-

ten Nachfolger des OSEK/VDX Systems, der AUTOSAR Classic Platform (*kurz:* AUTOSAR Classic). Dieser Standard deckt einen Großteil der heute in Fahrzeugen verbauten Steuergeräte ab. Ausnahmen bildeten in der Vergangenheit nur die angesprochenen Kleinststeuergeräte und Steuergeräte aus der Infotainment-Domäne, welche mit Audio- und Videoverarbeitung sowie der Anbindung von Drittanbieter-Geräten, wie zum Beispiel Mobiltelefonen, eine Sonderrolle in der E/E-Architektur eines Fahrzeugs einnahmen. Aufgrund der hohen Rechenanforderungen kommen in diesem Bereich bereits seit einigen Jahren leistungsstarke (Mikro-)Prozessoren zum Einsatz, die unter anderem Grafikbeschleuniger enthalten können. Die Unterstützung dieser Hardware, sowie die geforderte Flexibilität der Software, ist mit AUTOSAR Classic nur schwer und in manchen Fällen nicht zu erreichen. Daher werden für diese Steuergeräte andere Software-Architekturen und Betriebssysteme benötigt, die beispielsweise auf den *Portable Operating System Interface* (POSIX)-Standards basieren [IEE17]. POSIX definiert eine Programmierschnittstelle (engl.: Application Programming Interface (API)) zwischen Applikationssoftware und Betriebssystem und basiert auf UNIX-Definitionen. In aktuellen und zukünftigen E/E-Architekturen haben Steuergeräte für erweiterte Fahrerassistenzfunktionen, Domänencontroller und Zentralrechner neben den sicherheitsrelevanten auch Infotainment-ähnliche Anforderungen im Bezug auf Rechenleistung und Flexibilität. Deshalb werden auch hier immer öfters POSIX-basierte Systeme verwendet. Aus diesem Grund entschloss sich die AUTOSAR Partnerschaft eine zweite Softwareplattform zu definieren, die speziell für Hochleistungssteuergeräte ausgelegt ist - die POSIX-basierte AUTOSAR Adaptive Platform (*kurz:* AUTOSAR Adaptive). Der Fokus liegt dabei auf der Software-Architektur sowie dem Vorgehensmodell und lässt bei der Implementierung mehr Freiheiten als beispielsweise AUTOSAR Classic. Ein weiterer Paradigmenwechsel betrifft die Programmiersprache und die verwendeten Sprachkonstrukte. Bisher wurde die Software für automotive Steuergeräte typischerweise in ANSI C funktionsorientiert implementiert. Dies spiegelt sich auch in den OSEK

und AUTOSAR Classic Standards wieder. Zusätzlich schränkt der *Motor Industry Software Reliability Association* (MISRA)-C Standard die erlaubten Sprachkonstrukte ein. Unter anderem soll keine dynamische Speicherverwaltung und keine Pointer-Arithmetik verwendet werden. Im Gegensatz dazu setzt AUTOSAR Adaptive auf die Programmiersprache C++ und objektorientierte Programmierung. Trotz aller Unterschiede ist die Interoperabilität der beiden AUTOSAR Plattformen gewährleistet.

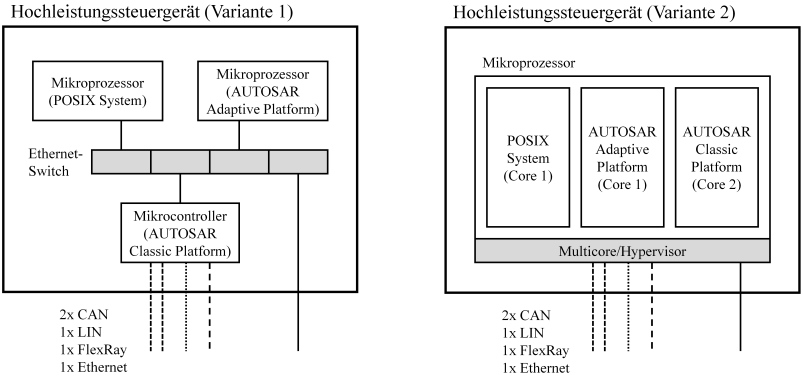

Bild 2.10: Aufbau eines Hochleistungssteuergeräts
Variante 1: Mehrprozessor/-controller System
Variante 2: Multicore/Hypervisor

Die vorliegende Arbeit fokussiert sich auf AUTOSAR Classic, da diese in den meisten der aktuell und - nach aktuellem Wissenstand auch - zukünftig verbauten Steuergeräte zum Einsatz kommt, selbst innerhalb von Hochleistungssteuergeräten. Dies klingt im ersten Moment widersprüchlich zum vorherigen Absatz, lässt sich aber durch die Hardwarearchitektur erklären. Typischerweise enthält ein Hochleistungssteuergerät einen oder mehrere Mikroprozessoren, auf denen ein POSIX-System ausgeführt wird. Parallel dazu ist in der Regel ein klassischer Mikrocontroller verbaut, der beispielsweise sicherheitsrelevante Aufgaben erfüllt und für die Fahrzeug-interne

Kommunikation zuständig ist. In den meisten Fällen findet hier AUTOSAR Classic Anwendung. Alternativ bietet sich ein entsprechend leistungsstarker Multicore-Prozessor und/oder ein Hypervisor für die Realisierung eines solchen Steuergeräts an. In diesem Fall können die einzelnen Softwareplattformen auf unterschiedlichen Kernen beziehungsweise in unterschiedlichen Partitionen ausgeführt werden. Bild 2.10 zeigt zwei mögliche Realisierungsvarianten eines Hochleistungssteuergeräts mit POSIX und AUTOSAR Classic Anteilen.

AUTOSAR Classic Platform

Die erste Version der AUTOSAR Classic Spezifikationen wurde im Juni 2005 veröffentlicht. Seither verfolgt die Partnerschaft Ziele wie Skalierbarkeit, Flexibilität, Wiederverwendbarkeit und Wartbarkeit von Steuergeräte-Software sowie die damit verbundene Kostenreduzierung. Architektonisch und funktional entwickelte sich die Softwareplattform in den letzten Jahren deutlich weiter. Die folgenden Beschreibungen und Erklärungen basieren - sofern nicht explizit anders vermerkt - auf dem vierten Hauptrelease, für welches im Dezember 2017 die Spezifikationen in der Version 4.3.1 freigegeben wurden. Dieser Abschnitt vermittelt ausschließlich die Grundlagen. Bei Bedarf werden weitere Details im Verlauf der Arbeit erklärt.

Zuerst wird näher auf die Software-Architektur eingegangen, die aus drei Hauptebenen besteht: Auf Ebene 1 befindet sich die *Basissoftware*, welche grundlegende Dienste zur Verfügung stellt. Dazu zählen unter anderem das Betriebssystem und die Kommunikationsstacks zum Versenden und Empfangen von Nachrichten. Die *Applikationssoftware* auf Ebene 3 implementiert die eigentliche Steuergeräte-Funktionalität wie beispielsweise einen Regelalgorithmus. Zwischen Basissoftware und Applikationssoftware liegt die sogenannte *Runtime Environment* (RTE) als Kernelement der Software-Architektur. Als Laufzeitumgebung abstrahiert sie jegliche Interaktion der Applikationssoftware und sorgt dafür, dass diese unabhängig von ande-

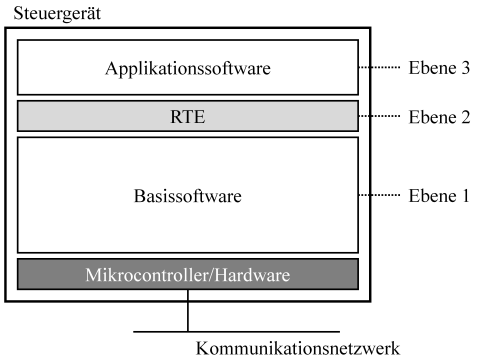

Bild 2.11: AUTOSAR Classic Software-Architektur

rer Applikations- beziehungsweise Basissoftware entwickelt werden kann. Auch die einfache Wiederverwendung von Software wird damit ermöglicht. Bild 2.11 zeigt eine schematische Darstellung der AUTOSAR Classic Software-Architektur. Die einzelnen Bestandteile sind im Folgenden näher beschrieben.

Basissoftware: Die Basissoftware ist modular aufgebaut und besteht insgesamt aus ca. 100 Modulen, die separat spezifiziert sind. Abhängig vom konkreten Steuergerät und dessen Anforderungen wird eine Untermenge davon verwendet. Zusätzlich unterteilt AUTOSAR Classic die Basissoftware in vier Blöcke, denen die einzelnen Module zugeordnet sind. In Bild 2.12 ist diese Strukturierung dargestellt.
Der erste Block ist die *Mikrocontrollerabstraktion*. Darin sind die verschiedenen Hardwaretreiber für die Mikrocontroller-interne Peripherie enthalten, wie beispielsweise für die Kommunikationscontroller. Darüber liegt die *Steuergeräteabstraktion* mit den Hardwaretreibern für die Mikrocontroller-externe Peripherie. Dazu zählen unter anderem Treiber für die Kommunikationstransceiver. Deren Implementierung ist unabhängig vom eingesetzten

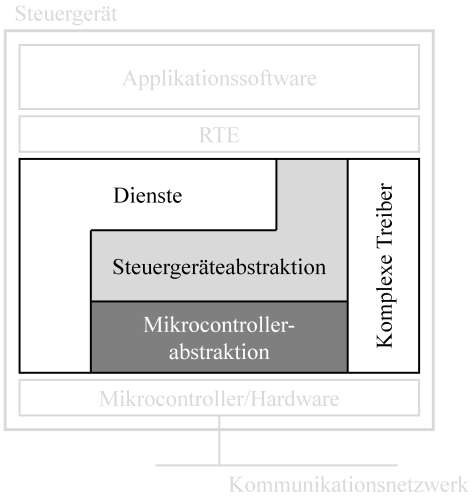

Bild 2.12: AUTOSAR Classic Basissoftware-Architektur

Mikrocontroller jedoch abhängig von den externen Peripheriebausteinen. Zusätzlich enthält die Steuergeräteabstraktion die sogenannten Interface-Module, welche ein Mikrocontroller- und Steuergeräte-unabhängiges API besitzen. In einem weiteren Block sind alle Module zu finden, die der Applikations- und Basissoftware *Dienste* zur Verfügung stellen. Dienste können sowohl rein Software-basiert sein als auch das Verwenden von hardwarebezogenen Funktionalitäten weiter abstrahieren. Die Implementierung ist hardwareunabhängig, mit Ausnahme des Betriebssystems, das ebenfalls diesem Block zugeordnet ist. Jegliche Interaktion zwischen Applikations- und Basissoftware findet in der Regel über die Dienste-Module statt. Einzig der Zugriff auf Input/Output (I/O) Hardware wird direkt über die Steuergeräteabstraktion realisiert. Die Strukturierung der AUTOSAR Classic Basissoftware vereinfacht die Wiederverwendung und erlaubt einen Steuergeräte-spezifischen Zuschnitt. Die hohe Anzahl an Modulen und Programmierschnittstellen verursacht allerdings an manchen Stellen zusätzliche Pro-

grammlaufzeit. Um dennoch besonders zeitkritische Anwendungen realisieren zu können und die Verwendung von Spezial- oder Übernahmesoftware zu ermöglichen, spezifiziert AUTOSAR Classic die *komplexen Treiber* (engl.: Complex Driver (CDD)). Dieser Block enthält selbst keine Module sondern ist ein Platzhalter für beliebige, proprietäre Basissoftware, die unabhängig von der bisher diskutierten Architektur integriert werden kann.
Neben dem Betriebssystem enthält die Basissoftware Module für folgende Funktionsbereiche:

- *Communication*: Dieser Teil abstrahiert die Fahrzeug-interne Kommunikation und ist ein Kernbaustein der Basissoftware. Eine Applikation benötigt damit kein Wissen über die zu Grunde liegende Netzwerktechnologie, um darüber zu kommunizieren. AUTOSAR Classic unterstützt nativ CAN, LIN, FlexRay und Ethernet.

- *Crypto*: Die Module dieses Funktionsbereichs erlauben einen standardisierten Zugriff auf kryptografische Funktionen, die sowohl in Software als auch hardwarebeschleunigt ausgeführt werden können.

- *Input/Output*: Über die I/O-Module kann Applikationssoftware Sensoren, Aktoren und Steuergeräteperipherie ansteuern.

- *Memory*: Das Speichern und Auslesen von nicht-flüchtigen Daten übernehmen die Memory-Module. Unterstützt wird Mikrocontroller-interner und extern angeschlossener Speicher.

- *Off-board Communication*: Drahtlose Kommunikation mit Fahrzeug-externen Partnern wird durch diesen Teil der Basissoftware abstrahiert, ähnlich zum Funktionsbereich Communication. Momentan spezifiziert AUTOSAR Classic Module für den Vehicle-to-X Anwendungsfall.

- *System*: Dieser Teil der Basissoftware enthält alle übergreifenden Systemfunktionen, die für den Betrieb eines Steuergeräts notwendig sind.

Dazu zählen das Betriebssystem, Diagnose- und Fehlerspeicherfunktionalität, Statusmanagement, Systemüberwachung und weitere Bibliotheksfunktionen.

Applikationssoftware: Die Applikationssoftware ist wie die Basissoftware modular aufgebaut und besteht aus sogenannten AUTOSAR Software-Komponenten (SW-C). SW-Cs sind Funktionsblöcke die abstrakt über Ports miteinander interagieren können. Ein Port kann Informationen zur Verfügung stellen (P-Ports - von provide), Informationen benötigen (R-Ports - von require) oder die Kombination aus beidem darstellen (PR-Port). Welche Informationen ausgetauscht werden und auf welche Weise dies passiert ist in einer Schnittstellenbeschreibung festgehalten (Port-Interface). Es gibt jeweils verschiedene Typen von SW-Cs und Port-Interfaces. Da diese für die vorliegende Arbeit eine untergeordnete Rolle spielen, wird nicht näher darauf eingegangen.

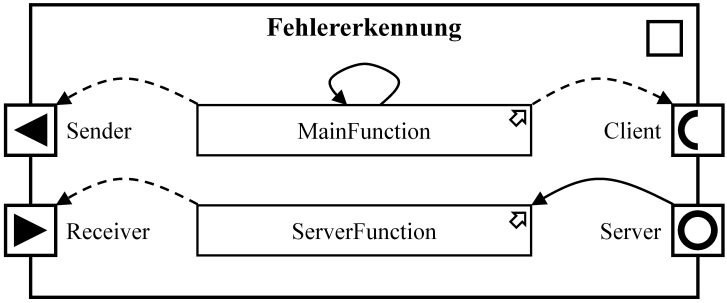

Bild 2.13: AUTOSAR Classic Applikationssoftware-Komponente

Beispielhaft zeigt Bild 2.13 eine Applikationssoftware-Komponente *Fehlererkennung* mit vier Ports. Die Ports auf der linken Seite beziehen sich auf eine datenorientierte Sender/Receiver-Schnittstelle (S/R) und jene auf der rechten Seite auf eine funktionsorientierte Client/Server-Schnittstelle (C/S). Bei *Sender* und *Server* handelt es sich um P-Ports. *Receiver* und *Client* sind

entsprechend R-Ports.
Bisher wurde auf die SW-Cs und deren Interaktion über Ports eingegangen. Ein weiterer wesentlicher Bestandteil der Applikationssoftware betrifft die Ausführung des Programmcodes. AUTOSAR Classic definiert hierfür die sogenannten *Runnable Entities* (kurz: *Runnables*). Ein Runnable ist ein Teil des Programmcodes, dessen Ausführung der Scheduler separat anstößt. In der Implementierung entspricht dies einer speziellen C-Funktion. Die Aktivierung kann auf verschiedene Arten erfolgen, beispielsweise zyklisch, sobald Daten an einem Receiver-Port zur Verfügung stehen oder mit dem Aufruf einer Server-Funktion durch einen Client. Innerhalb eines Runnables kann wiederum auf die Ports einer SW-C zugegriffen werden. Zwei Beispiele hierfür sind in Bild 2.13 dargestellt. Das Runnable *MainFunction* wird zyklisch aktiviert, erkennbar am durchgezogenen Pfeil auf sich selbst. Gestrichelte Pfeile markieren die Ports, auf die das Runnable zur Laufzeit zugreift. Im Gegensatz zu *MainFunction* aktiviert der Scheduler *ServerFunction* nur beim Aufruf der entsprechenden C/S-Funktion, wieder zu erkennen am durchgezogenen Pfeil. Auf die übliche Beschriftung der Pfeile mit dem dazugehörigen API wurde in Bild 2.13 aus Übersichtlichkeitsgründen verzichtet.

Runtime Environment: Die RTE ist das zentrale Element der Software-Architektur, um einige Ziele von AUTOSAR Classic zu erreichen. Sie entkoppelt SW-Cs sowohl von der Basissoftware als auch von anderen SW-Cs und fungiert als Steuergeräte-Middleware. Zusätzlich kümmert sich die RTE um das Scheduling der einzelnen Runnables. Damit lassen sich SW-Cs größtenteils Steuergeräte-unabhängig entwickeln, von einem Steuergerät auf ein anderes verschieben und einfach in unterschiedlichen Projekten wiederverwenden.

Jegliche Interaktion findet über die RTE statt, wie Bild 2.14 anhand eines Beispiels verdeutlicht. In diesem Fall existieren drei SW-Cs, die untereinander verbunden sind. Exemplarisch wird im Folgenden der Datenaustausch

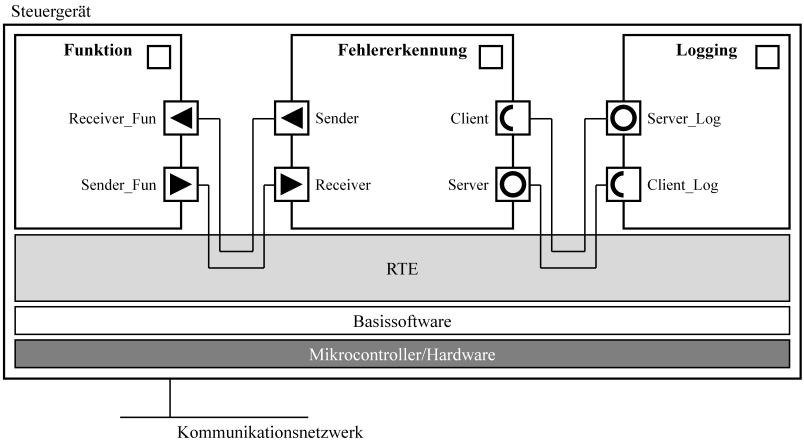

Bild 2.14: Interaktion zwischen SW-Cs über die RTE

zwischen einer *Funktion* und der *Fehlererkennung* betrachtet. Anstatt die Daten direkt - beispielsweise über ein API - an Port *Receiver* zu senden, werden diese durch den Port *Sender_Fun* der RTE übergeben, die sich dann um deren richtige Zustellung kümmert. Sollte eine weitere SW-C an den gleichen Daten interessiert sein, ist dies über die RTE ohne Probleme möglich. An der Implementierung der *Funktion* ändert sich dabei nichts. Es spielt ebenso keine Rolle, ob eine SW-C auf dem gleichen oder auf einem anderen Steuergerät die Daten benötigt. Im letzteren Fall abstrahiert die RTE das Versenden der Daten über ein entsprechendes Bussystem beziehungsweise Netzwerk. Auch eine Kombination aus beiden Fällen ist möglich. Der Übersichtlichkeit halber sind die Runnables in Bild 2.14 nicht dargestellt.

Methodik: Der Abschnitt zur AUTOSAR Classic Platform betrachtete bisher die Architektur der Steuergeräte-Software und deren Bestandteile. Wie oben erwähnt definiert AUTOSAR Classic zusätzlich ein dazugehöriges Vorgehensmodell, auf welches dieser Absatz genauer eingeht. Grund-

sätzlich wird ein Top-Down Ansatz verfolgt, der auf Gesamtsystemebene startet. Ein Fahrzeughersteller definiert dabei, gegebenenfalls zusammen mit seinen Zulieferern, die gesamte Softwarefunktionalität und strukturiert diese in einzelne Funktionsblöcke, die SW-Cs. Die Spezifikation jeder SW-C basiert auf dem AUTOSAR Extensible Markup Language (ARXML)-Format und umfasst mindestens die Definition der Komponente an sich, deren Ports und die dazugehörigen Port-Interfaces. Auch eine detailliertere Modellierung inkl. des internen Aufbaus ist möglich. Das resultierende ARXML-Dokument ist die *Software Component Description*. Da zu diesem frühen Zeitpunkt der Entwicklung noch keine Steuergeräte existieren, sind die SW-Cs über den sogenannten *Virtual Functional Bus* miteinander verbunden. Die gesamte Softwareapplikation kann somit größtenteils unabhängig von der E/E-Architektur eines Fahrzeugs modelliert werden. Die Definition der Steuergeräte und Fahrzeugnetzwerke erfolgt im nächsten Entwicklungsschritt und wird in den *ECU Descriptions* und in der *System Constraint Description* festgehalten, ebenfalls ARXML-Dokumente. Zusammen mit den Software Component Descriptions bilden diese Dokumente die *System Description*. Darin beschreibt ein Fahrzeughersteller im Extremfall die gesamte E/E-Architektur, die Kommunikation in den Fahrzeug-internen Netzwerken, die Struktur und Schnittstellen der Applikationssoftware sowie deren Aufteilung auf die einzelnen Steuergeräte. Ältere Formate zur Beschreibung der Kommunikation in Fahrzeugnetzwerken, wie beispielsweise das Database for CAN (DBC)-Format, werden somit nicht mehr benötigt.

Die Informationen der System Description sind für die Entwicklung der einzelnen Steuergeräte essentiell. Ein Fahrzeughersteller möchte einem Zulieferer jedoch nicht die Daten der gesamten E/E-Architektur und Applikationssoftware zur Verfügung stellen. Daher extrahiert er in einem weiteren Prozessschritt die für ein spezifisches Steuergerät relevanten Informationen und speichert diese in einem separaten Artefakt, dem *ECU Extract of System Description*. Mit dem Erstellen und der Übergabe dieses ARXML-Dokuments (ggf. auch mehrere Dokumente) an einen Zulieferer endet die

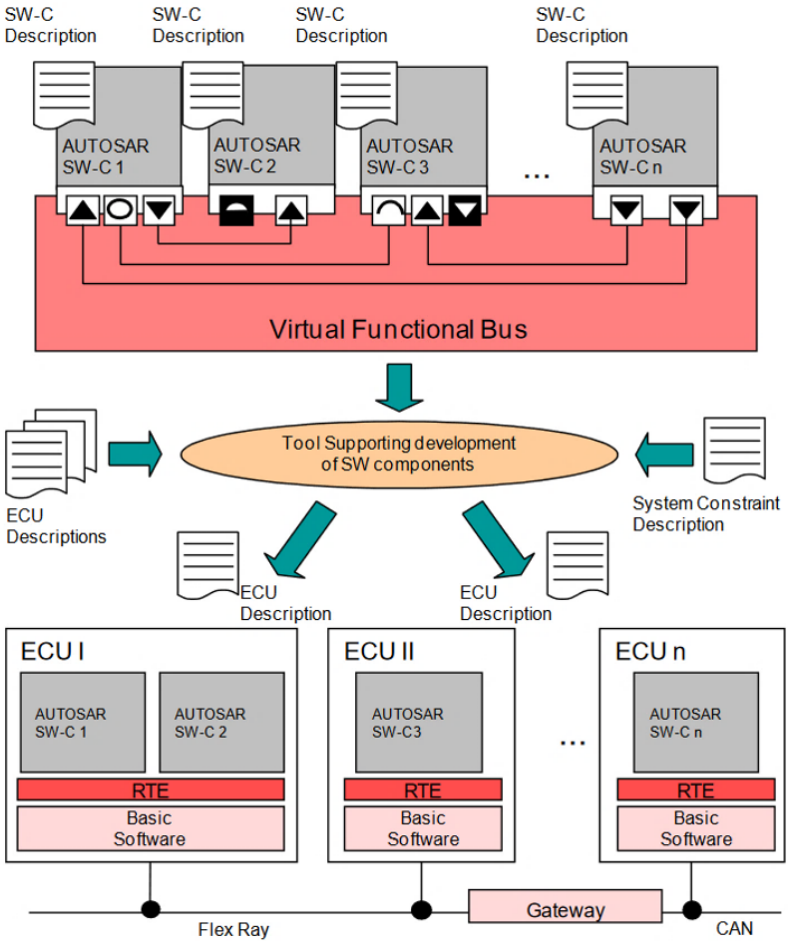

Bild 2.15: Vorgehensmodell von AUTOSAR Classic [AUT17a]

Gesamtsystemsicht. Das aus [AUT17a] entnommene Bild 2.15 zeigt das beschriebene Vorgehensmodell und die dazugehörigen ARXML-Artefakte.
Die nachfolgende Entwicklung der Software für ein Steuergerät hat verschiedene Aspekte. Einerseits definiert AUTOSAR Classic für jedes Basissoftware-Modul eine Vielzahl von Konfigurationsparametern, die Hersteller- und Steuergeräte-spezifisch zu setzen sind. Dabei hilft der ECU Extract of System Description. Mit dessen Hilfe können viele Einstellungen automatisiert vorgenommen werden. Die restlichen Parameter sind vom Zulieferer festzulegen. Nachdem die Konfiguration erfolgt und in der *ECU Configuration Description* abgespeichert ist, erzeugt ein Generator die fehlenden Teile der Basissoftware. Einige Module bestehen hauptsächlich aus statischem Programmcode und es wird lediglich eine Header-Datei generiert, welche die Werte der Konfigurationsparameter enthält. Andere Module werden fast vollständig dynamisch erzeugt. Dies gilt auch für die RTE, die zusammen mit der restlichen Basissoftware die Steuergeräte-spezifische Implementierung des Virtual Functional Bus darstellt. Der zweite Aspekt der Entwicklung von AUTOSAR Classic-basierter Steuergeräte-Software ist die strukturelle Verfeinerung und Implementierung der Applikation. Im ECU Extract of System Description ist typischerweise nur eine grobe Strukturierung in SW-Cs vorgegeben. Wie bei den meisten Prozessschritten im Vorgehensmodell ist der Zulieferer auch hierbei auf Toolunterstützung angewiesen.

2.2 Künstliche Intelligenz und maschinelles Lernen

Die Themengebiete künstliche Intelligenz und maschinelles Lernen haben viele Begriffe geprägt, die teilweise in der Literatur nicht einheitlich verwendet werden. Daher folgen eine kurze Einführung sowie die Begriffsdefinitionen, wie sie im Rahmen dieser Arbeit gültig sind.

2.2.1 Künstliche Intelligenz

Bereits 1950 diskutierte Alan M. Turing die Frage „Can machines think?“ [Tur50]. Er beschreibt unter anderem einen Test, der als eine der ersten informellen Definitionen von künstlicher Intelligenz (KI) angesehen wird - ohne dabei den Begriff selbst zu verwenden. Bild 2.16 zeigt eine schematische Darstellung des Turing-Tests.

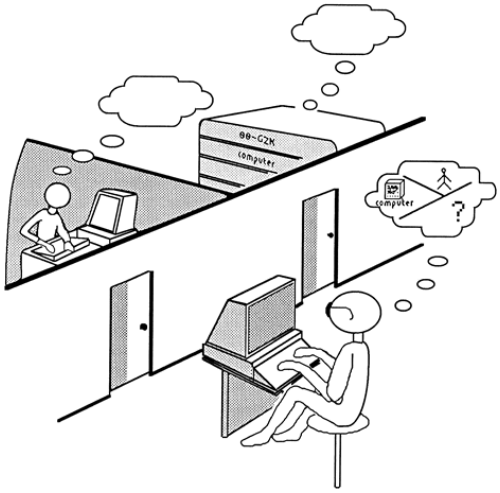

Bild 2.16: Turing-Test [Cop93]

Definition des Turing-Tests aus [Blo09]: „Im Zuge dieses Tests führt ein menschlicher Fragesteller über eine Tastatur und einen Bildschirm ohne Sicht- und Hörkontakt mit zwei ihm unbekannten Gesprächspartnern eine Unterhaltung. Der eine Gesprächspartner ist ein Mensch, der andere eine Maschine. Beide versuchen, den Fragesteller davon zu überzeugen, dass sie denkende Menschen sind. Wenn der Fragesteller nach der intensiven Befragung nicht klar sagen kann, welcher von beiden die Maschine ist, hat die Maschine den Turing-Test bestanden.“

In dem von J. McCarthy et al. gestellten Antrag für ein Forschungsprojekt [**McCarthy.04.04.1996**] taucht im Jahr 1955 zum ersten Mal der Begriff *Artificial Intelligence* auf. Die darauf folgende *Dartmouth Conference* ein Jahr später gilt als Gründungsveranstaltung für das Forschungsgebiet der künstlichen Intelligenz. 2007 definiert J. McCarthy künstliche Intelligenz folgendermaßen [McC07]:

„It is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable."

Man unterscheidet zwei Arten von künstlicher Intelligenz [Sea80]:
Eine Maschine mit *starker* künstlicher Intelligenz hat die Fähigkeit alle Aufgaben zu lösen, die menschliche Intelligenz erfordern. Es wird bis heute kontrovers darüber diskutiert, ob eine starke künstliche Intelligenz überhaupt möglich ist.
Im Gegensatz dazu beschränkt sich eine *schwache* künstliche Intelligenz auf einzelne Bereiche der menschlichen Intelligenz, bezogen auf bestimmte Anwendungsgebiete. Diese Art künstlicher Intelligenz ist heute schon in vielen Bereichen im Einsatz.
Die Begrifflichkeiten im Umfeld der künstlichen Intelligenz werden meistens über *Intelligenz* oder *menschliche Intelligenz* definiert. Die philosophische Frage, was Intelligenz beziehungsweise menschliche Intelligenz ist, wird in dieser Arbeit nicht weiter betrachtet.

2.2.2 Maschinelles Lernen

In den Anfängen der Computerwissenschaft wurde davon ausgegangen, dass ein Computer nur Aufgaben erfüllen kann, für die er explizit programmiert wurde. Dies wiederlegte Arthur Samuel 1956 mit einem lernenden Programm, welches seine Fähigkeiten beim Dame-Spiel selbstständig verbessert [IBM03]. 1959 definiert er implizit maschinelles Lernen [Sam59]:

„Programming computers to learn from experience should eventually eliminate the need for much of this detailed programming effort.“

1997 gibt Tom M. Mitchell eine formellere Definition. Er beschreibt maschinelles Lernen als die Eigenschaft einer Maschine, beziehungsweise eines Computerprogramms, sich selbst aus Erfahrung anzupassen, um bessere Ergebnisse zu erzielen [Mit97]:

„A computer program is said to learn from experience E with respect to some class of tasks T and performance P, if its performance at tasks in T, as measured by P, improves with experience E.“

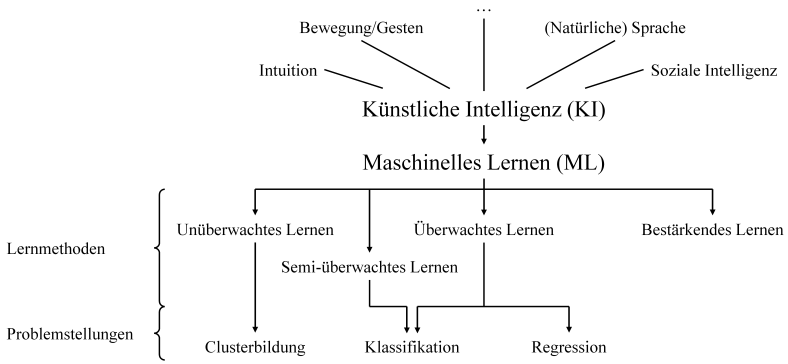

Bild 2.17: Übersicht zu künstlicher Intelligenz und maschinellem Lernen

Maschinelles Lernen ist ein grundlegender Baustein für künstliche Intelligenz und ist heute ein sehr vielseitiges Themengebiet mit einem einheitlichen Grundprinzip: Ein Computerprogramm *lernt* anhand von vorhandenen Datensätzen indem es seine internen Parameter gezielt anpasst. Dieser Vorgang nennt sich *Training* oder Trainingsphase. Das in den Trainingsdaten enthaltene Wissen soll extrahiert und in der nachfolgenden *Inferenz* (auch Testphase genannt) auf neue Datensätze angewendet werden. Aufgrund der Vielseitigkeit wird eine detailliertere Strukturierung vorgenommen. Bild 2.17 gibt einen ersten Überblick und zeigt, dass sich maschinelles Lernen in verschiedene Lernmethoden gliedern lässt, welche wiederum bei unterschiedlichen Problemstellungen Anwendung finden. Die folgenden Abschnitte gehen näher darauf ein.

Lernmethoden

Auf der ersten Ebene lässt sich maschinelles Lernen anhand des verwendeten Lernverfahrens gliedern. Dabei ist zu unterscheiden, welche Daten dem Computerprogramm zur Verfügung stehen, um seine internen Parameter anzupassen und die Ergebnisse zu verbessern.
Überwachtes Lernen (engl.: supervised learning) benötigt Trainingsdaten, die neben den Eingangswerten auch die dazugehörigen, erwarteten Ausgangswerte beinhalten. In diesem Zusammenhang spricht man von annotierten Daten (engl.: labeled data). Der Lernvorgang selbst besteht aus drei Schritten: Als erstes wird der zu trainierende Algorithmus mit einem Satz (engl.: *batch*) an Eingangswerten ausgeführt. Nachfolgend sind die berechneten Ausgangswerte mit den erwarteten zu vergleichen. Im letzten Schritt sind die Parameter des Algorithmus so anzupassen, dass sich im nächsten Durchlauf die berechneten Ausgangswerte dem erwarteten Ergebnis weiter annähern. Dieser Lernvorgang ermöglicht eine automatische Optimierung und Erfolgsbewertung. Das Bereitstellen von annotierten Datensätzen ist allerdings oftmals schwierig oder sogar unmöglich.

Im Gegensatz zum überwachten Lernen verwendet man beim *unüberwachten Lernen* (engl.: unsupervised learning) keine annotierten Trainingsdaten. In diesem Fall stehen nur Eingangswerte zur Verfügung und das Computerprogramm versucht eigenständig Zusammenhänge - sogenannte Muster - zu erkennen. Unüberwachtes Lernen hat den großen Vorteil, dass keine annotierten Daten bereitgestellt werden müssen. Nachteilig ist die erschwerte Ergebnis-Interpretation beziehungsweise -Bewertung, da die erwarteten Ausgangswerte im Voraus nicht bekannt sind.
Bei *semi-überwachtem Lernen* (engl.: semi-supervised learning) ist nur für einen Teil der Eingangswerte das erwartete Ergebnis bekannt. Im Fall von Ausreißer- oder Anomalieerkennung (siehe Unterkapitel 2.3) bezieht sich semi-überwachtes Lernen darauf, dass nur für den *Normalfall* annotierte Trainingsdaten vorhanden sind [CBK09].
Das letzte Lernverfahren ist *bestärkendes Lernen* (engl.: reinforcement learning), welches auf einem Belohnungsprinzip beruht. Ein Algorithmus soll dabei den Weg zu einem bestimmten Ziel eigenständig erlernen. Im Training stehen dafür zunächst nur die Eingangswerte zur Verfügung. Zu bestimmten Zeitpunkten wird dem Computerprogramm jedoch mitgeteilt, ob die generierten Ausgangswerte gut oder schlecht waren. Dies geschieht mittels positiver oder negativer Belohnung. Ein typisches Beispiel ist das Trainieren von Computerspielen. Dabei wird nicht jeder einzelne Spielzug bewertet (nicht zu jedem Spielzug ist das erwartete Ergebnis gegeben), sondern das Computerprogramm bekommt am Ende eine positive Belohnung, wenn das Spiel gewonnen wurde, anderenfalls eine negative.
Eine weitere Unterscheidung ergibt sich durch den Zeitpunkt der Parameteranpassung. Dazu ist zunächst die Einführung des Begriffs *Epoche* (engl.: epoch) notwendig. Eine Epoche beschreibt, dass der zu trainierende Algorithmus genau einmal mit jedem *Eingangsvektor* (engl.: input vector oder sample) des Trainingsdatensatzes ausgeführt wurde. Ein Eingangsvektor setzt sich üblicherweise wiederum aus mehreren Werten, den sogenannten *Merkmalen* (engl.: features), zusammen. In der Bildverarbeitung entspricht

im Regelfall ein Bild einem Eingangsvektor und beispielsweise jedes einzelne Pixel einem Merkmal.
Beim *Batch-Training* werden die Algorithmenparameter jeweils nach einer vollständigen Epoche angepasst. Im Gegensatz dazu, geschieht die Anpassung beim *Mini-Batch-Training* nach einer zu definierenden Batch-Größe. Dabei besteht ein Batch aus einem Subset der Trainingsdaten und die Batch-Größe gibt die Anzahl der enthaltenen Eingangsvektoren an. Im Extremfall, d.h. bei einer Batch-Größe von eins, sind die Parameter nach jedem einzelnen Eingangsvektor anzupassen. Diesen Fall bezeichnet man auch als *Stochastic Gradient Decent* (SGD).
Für alle Lernmethoden gilt, dass während des Trainings die *Parameter* eines Algorithmus bestimmt werden. Ergänzend dazu besitzen viele Algorithmen des maschinellen Lernens auch *Hyperparameter*, die vorab festzulegen sind. Für die in Abschnitt 2.2.3 vorgestellten künstlichen neuronalen Netze sind beispielsweise die Schichten- und die jeweilige Neuronenanzahl zu definierende Hyperparameter. Das Trainingsergebnis sind die Kantengewichte (Parameter).

Problemstellungen

Nachdem im vorherigen Absatz auf die unterschiedlichen Lernverfahren eingegangen wurde, sind nachfolgend die drei wichtigsten Problemstellungen beschreiben, bei welchen maschinelles Lernen bereits erfolgreich eingesetzt wird.
Bei der *Klassifikation* ist die Aufgabe den gegebenen Eingangswerten einen bestimmten diskreten Ausgangswert zuzuordnen. Damit findet eine Klassifizierung der Eingangswerte statt. Entsprechende Aufgaben werden meistens durch überwachte Lernverfahren adressiert. Mittels annotierter Trainingsdaten wird ein Klassifikator gelernt, der die Eingangswerte möglichst präzise in die vorgegebenen Klassen einordnet. Ist das Training erfolgreich abgeschlossen, klassifiziert das Computerprogramm in der Testphase auch neue,

bis dahin unbekannte Datensätze richtig. Ein bekanntes Beispiel ist die Objekterkennung in Bilddaten und die Zuordnung eines gefundenen Objekts zu einer bestimmten Klasse. So müssen in modernen Fahrzeugen Objekte in Echtzeit erkannt und klassifiziert werden, beispielsweise in Fahrzeuge, Verkehrsschilder, Fußgänger und Hindernisse. Die Objekterkennung wird dafür mittels ausgesuchter, annotierter Datensätze trainiert.

Clusterbildung hat das Ziel nicht annotierte Trainingsdaten zu gruppieren. Im Unterschied zur Klassifikation ist damit die Zuordnung der Trainingsdaten zu den gesuchten Klassen nicht vorhanden. Die Cluster, in welche die Eingangswerte gruppiert werden, sind zu Beginn unbekannt. Durch die nicht annotierten Trainingsdaten ist Clusterbildung eine Problemstellung für unüberwachtes Lernen und ein wichtiger Bestandteil bei der Wissensentdeckung in großen Datenmengen. Ein fiktives Beispiel ist die Ermittlung eines bis dato unbekannten Zusammenhangs zwischen erhöhtem Treibstoffverbrauch, Regen und einer Temperatur von unter 4 Grad Celsius. Dieser Zusammenhang wäre dann aus einem entsprechend gebildeten Cluster ersichtlich.

Bei den beiden bisher vorgestellten Problemstellungen ist das Ziel die Zuordnung von Eingangswerten zu einem diskreten Ausgangswert - einer Klasse oder einem Cluster. Im Gegensatz dazu wird bei der *Regression* eine Funktion approximiert, um beliebigen Eingangswerten einen kontinuierlichen Ausgangswert zuordnen zu können. Für die Approximation einer Funktion sind üblicherweise Stützstellen mit den dazugehörigen Stützwerten notwendig. Daher ist die Regression eine Aufgabe für überwachtes Lernen. Liegt beispielsweise das Wissen vor, dass Niederschlag, Temperatur und Treibstoffverbrauch zusammenhängen, kann durch eine Regression der veränderte Treibstoffverbrauch in Abhängigkeit von Niederschlag und Temperatur vorhergesagt werden.

Die Problemstellungen können einzeln oder in Kombination auftreten, je nach zur Verfügung stehender Datensätze. Im obigen Beispiel zu Clusterbildung wird ein Zusammenhang entdeckt. Das Beispiel zur Regression

nutzt dieses neue Wissen, um eine Vorhersage zu treffen, die bisher nicht möglich war. Die beschriebenen Problemstellungen spielen daher auch im Rahmen der Wissensentdeckung in Datenbanken (*Knowledge Discovery in Databases* (KDD)) eine große Rolle [FPS96].

2.2.3 Künstliche neuronale Netze

Künstliche neuronale Netze sind eine Technik der Informationsverarbeitung in Computersystemen. Als Vorbild dienen biologische neuronale Netze wie das menschliche Gehirn und das Rückenmark. Die Vorbildfunktion bezieht sich sowohl auf den Aufbau und die Struktur der Netze als auch auf die zu bewältigenden Aufgaben wie beispielsweise Muster-, Bild- und Spracherkennung sowie das selbstständige Lernen und Bewältigen von Aufgaben. Biologische neuronale Netze bestehen aus vielen Nervenzellen, sogenannten Neuronen. Diese setzen sich wiederum aus Synapsen, Dendriten, Zellkern (Soma) und Axon zusammen [Kri05].
Ihren Ursprung haben künstliche neuronale Netze bereits in den 40er Jahren. 1943 veröffentlichten Warren S. McCulloch und Walter Pitts ihr vereinfachtes Modell eines Neurons, um damit logische Berechnungen im Gehirn darstellen und nachvollziehen zu können [MP43]. Im vorgestellten Modell arbeiten die Neuronen nach dem alles-oder-nichts Prinzip und können nur binäre Ein- und Ausgangssignale verarbeiten. Wie beim biologischen Vorbild gibt es anregende und hemmende Eingänge. Ist ein hemmender Eingang aktiv, ist das Neuron blockiert, was dem Binärwert „0“ entspricht. Ist kein hemmender Eingang aktiv, werden die anregenden Eingänge aufsummiert. Überschreitet die Summe eine festgelegte Schwelle, feuert das Neuron (Binärwert „1“). Damit lassen sich die logischen Funktionen UND, ODER und NICHT abbilden.

Einschichtiges Perceptron

Einen wichtigen Meilenstein setzt Frank Rosenblatt 1958. Sein Perceptron [Ros58] ist bis heute die Grundlage moderner künstlicher neuronaler Netze. Der Hauptunterschied zum Modell von McCulloch und Pitts sind die gewichteten Eingangssignale (später: Merkmale) und die darauf angepasste und angewendete Lernregel von Donald O. Hebb [Heb49], welche die Stärkung von wichtigen Neuronenverbindungen beschreibt.

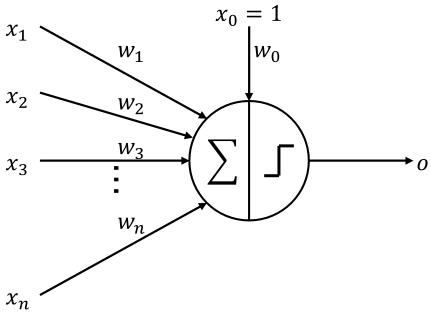

Bild 2.18: Einschichtiges Perceptron mit einem Neuron

Das Perceptron funktioniert wie sein biologisches Vorbild und ist in Bild 2.18 dargestellt. Die Eingangssignale x_1 bis x_n und ein Offset x_0 werden über gewichtete (w_0 bis w_n) Verbindungen, den sogenannten Kanten, dem Kern zugeführt. Dort findet die Summation der gewichteten Eingangssignale und die anschließende Berechnung des Ausgangssignals o über eine Aktivierungsfunktion φ statt. Dabei lassen sich die einzelnen Bestandteile einer biologischen Nervenzelle wiederfinden. Die Synapsen zusammen mit den Dendriten bilden die Eingangssignale und die gewichteten Kanten. Der Zellkern entspricht der Kombination aus Summation und Berechnung des Ausgangssignals. Das Axon ist die Weiterleitung des Ausgangssignals.
Das ursprüngliche Perceptron besitzt als Aktivierungsfunktion eine Sprung-

funktion (siehe Bild 2.18), welche den Wertebereich des Ausgangssignals auf „0“ und „1“ einschränkt. Damit eignet es sich für binäre Klassifikationsaufgaben. Das Ausgangssignal ergibt sich aus folgender Formel:

$$o = \varphi(\sum_{i=0}^{n} x_i w_i) \tag{2.1}$$

Mit dem Perceptron können zwei Mengen an Eingangssignalkombinationen im n-dimensionalen Raum binär klassifiziert werden, solange die Mengen linear separierbar sind. Ein Beispiel einer einfachen binären Klassifikationsaufgabe ist das Abbilden der logischen ODER-Funktion. Dies ist mit einem Perceptron, bestehend aus einem Neuron mit zwei Eingangssignalen (x_1 und x_2), realisierbar. Die vier möglichen Kombinationen der Eingangssignale können linear voneinander separiert und der entsprechenden Klasse, d.h. dem entsprechenden Wert des Ausgangssignals, zugeordnet werden. Bild 2.19 zeigt diese lineare Separierbarkeit.

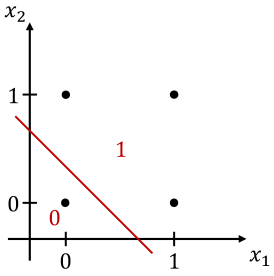

Bild 2.19: Linear separierbare ODER-Abbildung

Durch das Verwenden mehrerer Neuronen entsteht eine Struktur wie in Bild 2.20. Jedes Neuron ist mit jedem Eingangssignal verbunden und erzeugt ein eigenständiges Ausgangssignal. Dabei kann der Eindruck entstehen, dass es sich um ein zweischichtiges Perceptron handelt. Dies ist jedoch nicht der Fall, da es sich bei den Kreisen in der ersten Spalte, auch *Schicht* (engl.: layer) genannt, lediglich um Verteilpunkte für die Eingangssignale

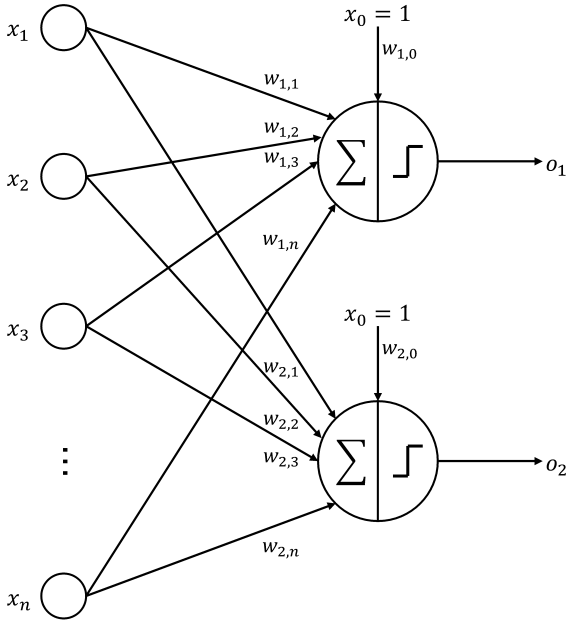

Bild 2.20: Einschichtiges Perceptron mit zwei Neuronen

handelt. Dort findet keine Summation und keine Berechnung des Ausgangssignals statt, wie es bei vollwertigen Neuronen der Fall ist. Es handelt sich also weiterhin um ein einschichtiges Perceptron, das lediglich in seiner Darstellung zwei Schichten aufweist: Die Eingangsschicht mit den Verteilpunkten, die auch Eingangsneuronen genannt werden, und die Ausgangsschicht mit den eigentlichen Neuronen. Bei den Kantengewichten repräsentiert der erste tiefgestellte Index j die Nummer des Neurons, welchem das Eingangssignal über diese Kante zugeführt wird. Der zweite Index i referenziert das dazugehörige Eingangssignal.

Die verallgemeinerte Formel für den j-ten Ausgangswert ergibt sich wie folgt:

$$o_j = \varphi(\sum_{i=0}^{n} x_i w_{j,i}) \tag{2.2}$$

Die Adaption des Perceptrons an eine bestimmte Aufgabenstellung erfolgt bei gegebener Netzstruktur durch das Anpassen der Kantengewichte $w_{j,i}$ (vgl.: Parameter), die zu Beginn mit Zufallswerten initialisiert sind. Zunächst wird ein Eingangsvektor des Trainingsdatensatzes dem Netz zugeführt. Dieses berechnet mit den aktuellen Kantengewichten die Ausgangssignale. Nach der Ermittlung der Differenz zwischen berechneten und vorgegebenen Ausgangssignalen (vgl.: überwachtes Lernen, Abschnitt 2.2.2), sind die Kantengewichte anzupassen (vgl.: SGD, Abschnitt 2.2.2). Stefan Selle beschreibt die Anpassungsvorschrift wie folgt [Sel98]:

- „Wenn die Ausgabe Eins (aktiv) ist und Eins sein soll oder wenn sie Null (inaktiv) ist und Null sein soll, dann werden die Gewichtungen nicht verändert."
- „Wenn die Ausgabe Null ist, aber Eins sein sollte, werden die Gewichtungen für alle aktiven Eingabeverknüpfungen erhöht."
- „Wenn die Ausgabe Eins ist, aber Null sein sollte, werden die Gewichtungen für alle aktiven Eingabeverknüpfungen verringert."

$$\Delta w_{j,i} = \eta \delta_j x_i \qquad \text{mit} \qquad \delta_j = t_j - o_j \tag{2.3}$$

Formel 2.3 beschreibt die Anpassung der Kantengewichte formaler. η ist die sogenannte *Lernrate* (engl.: learning rate), die festlegt, in welchem Umfang ein Kantengewicht pro Lernschritt angepasst wird. t_j ist der in den Trainingsdaten vorgegebene Ausgangssignalwert.

Rosenblatt lieferte 1962 den Beweis, dass alle vom Perceptron darstellbaren Geraden, Ebenen beziehungsweise Hyperebenen zur linearen Separierung der Eingangssignalkombinationen mit dem angewandten Lernalgorithmus in endlicher Zeit bestimmt werden können [Ros62]. Der große

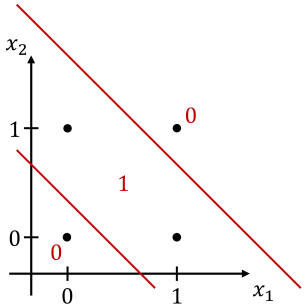

Bild 2.21: Nicht linear separierbare XOR-Abbildung

Durchbruch gelang damals dennoch nicht. Verantwortlich dafür war die Einschränkung auf linear separierbare Probleme. Marvin Lee Minsky und Seymour Papert lieferten 1969 in ihrem Buch den mathematischen Beweis, dass das Perceptron bestimmte Probleme nicht lösen kann [MP72]. Dazu zählt die Abbildung eines logischen XOR. Bild 2.21 zeigt die nicht linear separierbare Problemstellung.

Die Veröffentlichung von Minsky und Papert trug maßgeblich dazu bei, dass die Forschungsaktivitäten im Bereich der künstlichen neuronalen Netze in den folgenden Jahren nahezu eingestellt wurden [Hil95]. Dies betraf unter anderem auch Karl Steinbuch, den Gründer des heutigen Instituts für Technik der Informationsverarbeitung am Karlsruher Institut für Technologie, siehe Anhang A.1. Stattdessen verschob sich der Forschungsschwerpunkt auf die symbolische künstliche Intelligenz, die sich mit der symbolischen Darstellung und Verarbeitung von Wissen und Informationen beschäftigt [Uth01].

Mehrschichtiges Perceptron

Im Unterschied zum einschichtigen Perceptron wird beim mehrschichtigen Perceptron mindestens eine zusätzliche Schicht zwischen Eingangs- und Ausgangsschicht verwendet. Diese Zwischenschichten werden als *ver-*

steckte Schichten (engl.: hidden layer) bezeichnet. Durch die zusätzlichen Schichten können mehr Probleme gelöst werden, beispielsweise die Klassifikation von nicht linear separierbaren Eingangssignalkombinationen wie es bei der Abbildung eines XOR der Fall ist. In Bild 2.22 ist ein mehrschichtiges Perceptron dargestellt, welches das XOR-Problem löst. Die Neuronen in der versteckten Schicht abstrahieren die beiden Trenngeraden aus Bild 2.21. Das erste Neuron der versteckten Schicht bildet dabei beispielsweise ein logisches x_1^1 ODER x_2^1, das zweite ein logisches NICHT (x_1^1 UND x_2^1). Die Verknüpfung der beiden Zwischenergebnisse, d.h. ein logisches x_1^2 UND x_2^2, übernimmt das Neuron in der Ausgangsschicht. Daraus ergibt sich für das Ausgangssignal o_1^2 Formel 2.4. Der hochgestellte Index zeigt dabei die Schicht im neuronalen Netz an, wobei die Eingangsschicht nicht mitgezählt wird.

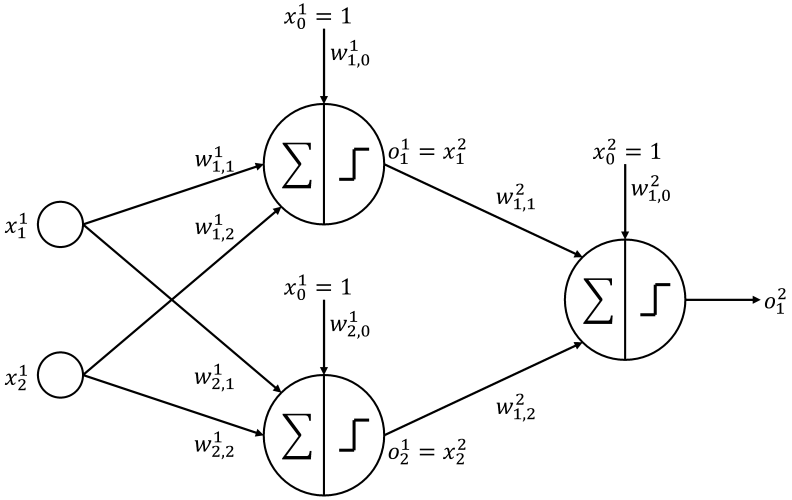

Bild 2.22: Mehrschichtiges Perceptron für die Abbildung eines logischen XOR[8]

[8] Adaptiert von [VWeb+18b]

$$o_1^2 = (x_1^1 \text{ ODER } x_2^1) \text{ UND NICHT } (x_1^1 \text{ UND } x_2^1) \tag{2.4}$$

In ihrer kritischen Perceptron-Betrachtung diskutieren Minsky und Papert 1969 auch bereits mehrschichtige Anordnungen von Neuronen [MP72]. Ihrer Meinung nach ist das Erweitern des Perceptrons nutzlos. Den Hauptgrund dafür sehen sie in der bis dato fehlenden Lernregel für mehrschichtige Anordnungen. 1986 widerlegten David E. Rumelhart, Geoffrey E. Hinton und Ronald J. Williams die pessimistische Annahme von Minsky und Papert, indem sie den sogenannten *Backpropagation-Algorithmus* auf mehrschichtige Perceptrons anwendeten und zeigten, dass dadurch abstrakte und nützliche Repräsentationen der Eingangssignale in den versteckten Schichten entstehen [RHW86]. Damit können auch nicht linear lösbare Klassifizierungs- und Regressionsprobleme trainiert und bewältigt werden. Die Idee den Backpropagation-Algorithmus auf künstliche neuronale Netze anzuwenden publizierte Paul Werbos 1974 [Wer74]. Die praktische Anwendung erfolgte 1982 [Wer82]. Doch erst die Veröffentlichung von Rumelhart et al. brachte aufgrund der vorgestellten internen Repräsentationen den Durchbruch.

Beim Backpropagation-Algorithmus für mehrschichtige Perceptrons ist die Fehlerfunktion E ein Gütemaß dafür, wie gut die vom künstlichen neuronalen Netz berechneten mit den vorgegebenen Ausgangswerten übereinstimmen. Bei einer Regressionsaufgabe gibt sie beispielsweise Auskunft darüber, wie gut eine mathematische Funktion approximiert wurde. Als Fehlerfunktion wird der über alle n Ausgangsneuronen aufsummierte quadratische Fehler verwendet. In Formel 2.5 ist t_j der vorgegebene und o_j der berechnete Ausgangswert des j-ten Neurons [RM86]. Zur Vereinfachung der Ableitung wird der Faktor $\frac{1}{2}$ vorangestellt.

$$E = \frac{1}{2} \sum_{j=1}^{n} (t_j - o_j)^2 \tag{2.5}$$

Der Ablauf des Trainings ist identisch zur Gewichtsanpassung bei einem einschichtigen Perceptron, d.h. ein Eingangsvektor wird angelegt, der Ausgangsvektor und der Approximationsfehler berechnet, gefolgt von der entsprechenden Anpassung der Kantengewichte. Die Gewichtsanpassung kann auch erst nach dem Anlegen mehrerer Eingangsvektoren erfolgen (siehe Batch-/Mini-Batch-Training, Abschnitt 2.2.2). Die Adaption der Kantengewichte selbst basiert auf dem Gradientenabstiegsverfahren, welches iterativ versucht einen Punkt auf einer mehrdimensionalen Funktion in das Minimum zu bringen. Dabei wird der Punkt nach jeder Iteration in jeder Dimension ein Stück in Abstiegsrichtung, d.h. in negativer Gradientenrichtung, verschoben. Der Punkt liegt somit in der nächsten Iteration ein Stück näher am Minimum. Bezogen auf mehrschichtige Perceptrons, sollen die betroffenen Kantengewichte so angepasst werden, dass die Fehlerfunktion E minimal wird. Die Veränderung des Kantengewichts $w_{j,i}$ berechnet sich über die partielle Ableitung der Fehlerfunktion E, multipliziert mit der Lernrate η, wie in Formel 2.6 dargestellt [RM86]. x_i ist der Ausgangswert des i-ten Neurons der vorherigen Schicht oder der i-te Wert im Eingangsvektor.

$$w_{j,i,neu} = w_{j,i} + \Delta w_{j,i} \qquad \text{mit} \qquad \Delta w_{j,i} = -\eta \frac{\partial E}{\partial w_{j,i}} = \eta \delta_j x_i \quad (2.6)$$

Die folgenden Formeln zeigen den Unterschied in der Berechnung von δ_j, abhängig davon, ob sich das j-te Neuron in der Ausgangsschicht (Formel 2.7) oder in einer versteckten Schicht (Formel 2.8) befindet [RM86]. n entspricht dabei der Anzahl der Eingangssignale des j-ten Neurons. Wird eine versteckte Schicht betrachtet, ist m die Anzahl der Neuronen in der nächsten Schicht, welche den Ausgangswert des j-ten Neurons als Eingangssignal erhalten. Bei einem vollständig verbundenen Netz entspricht m der Anzahl der Neuronen in der nächsten Schicht.

$$\delta_j = \varphi'(\sum_{i=1}^{n} x_i w_{j,i})(t_j - o_j) \qquad \text{für Neuronen in der Ausgangsschicht} \tag{2.7}$$

$$\delta_j = \varphi'(\sum_{i=1}^{n} x_i w_{j,i}) \sum_{k=1}^{m} \delta_k w_{k,j} \qquad \text{für Neuronen in einer versteckten Schicht} \tag{2.8}$$

Da für die Berechnung der Gewichtsanpassung die Ableitung der Aktivierungsfunktion φ notwendig ist, bieten sich hier Funktionen an, die einfach differenzierbar sind. Auch für die Fehlerfunktion E sind heute verschiedene Varianten im Einsatz.

Trotz des spätestens seit Rumelhart et al. bekannten Backpropagation-Algorithmus dauerte es bis in die späten 2000er Jahre, ehe das mehrlagige Perceptron und die entwickelten, speziellen Ausprägungen davon in der Breite eingesetzt wurden. Dies lag vor allem an der zuvor fehlenden Rechenleistung, um große Netze in kurzer Zeit trainieren zu können und an den fehlenden Trainingsdaten. In beiden Bereichen wurden in den letzten Jahren enorme Fortschritte gemacht. Bei der Rechenleistung ist dies im speziellen auf die Multi-Core Technologie zurückzuführen, die für die Berechnung von neuronalen Netzen sehr gut geeignet ist, da jedes Neuron einer Schicht auf einem separaten Core gerechnet werden kann.

Ein genereller Nachteil des Gradientenabstiegsverfahrens ist, dass es in einem lokalen Minimum konvergieren kann. Das Auffinden des globalen Minimums ist nicht garantiert. Trotzdem wird heute bei (semi-)überwachtem Lernen der Backpropagation Algorithmus - oder eine optimierte Variante davon - in den meisten Fällen eingesetzt.

Bei allen bisher vorgestellten künstlichen neuronalen Netzen handelt es sich um feed-forward Netze, da innerhalb des Netzes keine Rückkopplungen zu finden sind. Rekurrente Netze bauen auf dem gleichen Neuronenmodell

wie das Perceptron auf, enthalten allerings interne Rückkopplungen. Diese Klasse und spezielle Ausprägungen von feed-forward Netzen, bei denen beispielsweise Neuronen in der n-ten Schicht nicht mit allen Neuronen der $(n-1)$-ten Schicht verbunden sind, werden hier nicht weiter betrachtet, da sie in der vorliegenden Arbeit keine Anwendung finden.

2.3 Anomalieerkennung

Der Begriff Anomalie steht für eine Beobachtung, die ein von der Normalität abweichendes Verhalten zeigt, wobei die Normalität im Kontext der Beobachtung zu definieren ist. Eine oft zitierte Definition einer Anomalie gibt Douglas M. Hawkins 1980 [Haw80]:

„An outlier is an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism."

Als Synonym für abweichendes Verhalten wird im Folgenden *Fehlverhalten* verwendet. Um im Laufe der Arbeit verschiedene Fehlverhalten und deren Erkennung besser einordnen zu können, wird der Begriff Anomalie verfeinert. Chandola et al. verwenden dafür drei Anomalie-Klassen [CBK09]. Diese Unterteilung wird in der vorliegenden Arbeit übernommen.
Die erste und einfachste Anomalie-Klasse ist die *Punktanomalie*. Ist eine Beobachtung innerhalb eines Datensatzes, ohne Berücksichtigung weiterer Informationen, als Anomalie erkennbar, dann handelt es sich um eine Punktanomalie. Wenn beispielsweise innerhalb einer CAN-Bus Aufzeichnung in einem bestimmten Frame die Wertekombination 1. Gang, 1500 rpm und 200 km/h übertragen wurde, liegt eine Punktanomalie vor (vgl. [VWeb+18a]). Nachfolgend die Definition nach Chandola et al. [CBK09]:

„If an individual data instance can be considered as anomalous with respect to the rest of data, then the instance is termed as a point anomaly."

Im Gegensatz zu einer Punktanomalie ist eine *kontextuelle Anomalie* nur unter Berücksichtigung zusätzlicher Informationen als solche zu erkennen. Dies definieren Chandola et al. [CBK09] folgendermaßen:

„If a data instance is anomalous in a specific context (but not otherwise), then it is termed as a contextual anomaly (also referred to as conditional anomaly [Son+07])."

Ist in der oben genannten CAN-Bus Aufzeichnung die Beobachtung 5. Gang, 3000 rpm und 100 km/h enthalten, ist dies zunächst keine Punktanomalie, da die Werte plausibel erscheinen. Existiert hingegen ein positionsbezogener Kontext, der anzeigt, dass der Frame in einer 30er-Zone aufgezeichnet wurde, dann handelt es sich um eine kontextuelle Anomalie. Nur mit dem kontextuellen Wissen lässt sich die Anomalie erkennen. Anhand der reinen Beobachtung, ist dies nicht möglich.
Die dritte Anomalie-Klasse ist die *kollektive Anomalie*. Eine kollektive Anomalie beschreibt zusammenhängende Beobachtungen, die gemeinsam ein Fehlverhalten gegenüber dem Datensatz aufweisen. Chandola et al. [CBK09] definieren eine kollektive Anomalie wie folgt:

„If a collection of related data instances is anomalous with respect to the entire data set, it is termed as a collective anomaly."

Im Beispiel der CAN-Bus Aufzeichnung wäre eine kollektive Anomalie eine Reihe von Beobachtungen, die anzeigen dass der Fahrer über einen sehr langen Zeitraum im 1. Gang mit 10 km/h konstanter Geschwindigkeit fährt. Eine Punktanomalie kann nicht erkannt werden, da die einzelnen Beobachtungen valide sind. Auch mit positionsbezogenem Kontext ist keine Anomalie zu erkennen, da beispielsweise bei Stau auch auf der Autobahn 10 km/h plausibel sind. Treten jedoch keinerlei Schwankungen der Ge-

schwindigkeit über einen langen Zeitraum auf (Tempomat ist bei 10 km/h deaktiviert), stellen die zusammenhängenden Beobachtungen eine kollektive Anomalie dar.
Der Vorgang, anhand von Beobachtungen ein von der Normalität abweichendes Verhalten festzustellen, wird als Anomalieerkennung bezeichnet [CBK09]:

„Anomaly detection refers to the problem of finding patterns in data that do not conform to expected behavior."

Zur Anomalieerkennung eignet sich eine Vielzahl von Verfahren, die Konzepte aus unterschiedlichen Forschungsgebieten aufgreifen. Dazu zählen unter anderem Algorithmen des maschinellen Lernens und statistische Auswertungen. Die Auswahl eines passenden Verfahrens ist abhängig vom Anwendungsgebiet und der damit verbundenen Problemstellung. Für deren Beschreibung verwenden Chandola et al. die vier unten genannten Eigenschaften [CBK09]. Bild 2.23 fasst die unterschiedlichen Einflussfaktoren zusammen und ist [CBK09] entnommen.

1. Die Art der zu untersuchenden Daten (engl.: Nature of Data), zum Beispiel Bilder, Text, Sprache oder Zeitreihen
2. Das Vorhandensein von annotierten Trainingsdaten (engl.: Labels)
3. Die gesuchte Anomalie-Klasse oder die gesuchten Anomalie-Klassen (engl.: Anomaly Type), zum Beispiel kollektive Anomalien
4. Die Art der Ausgabe (engl.: Output), zum Beispiel eine binäre Klassifikation oder eine Anomalie-Wahrscheinlichkeit

Wie zuvor erwähnt, ist Anomalieerkennung ein Anwendungsfall für maschinelles Lernen. Die entsprechende Einordnung erfolgt in Bild 2.24 als Erweiterung der in Bild 2.17 gezeigten Übersicht.

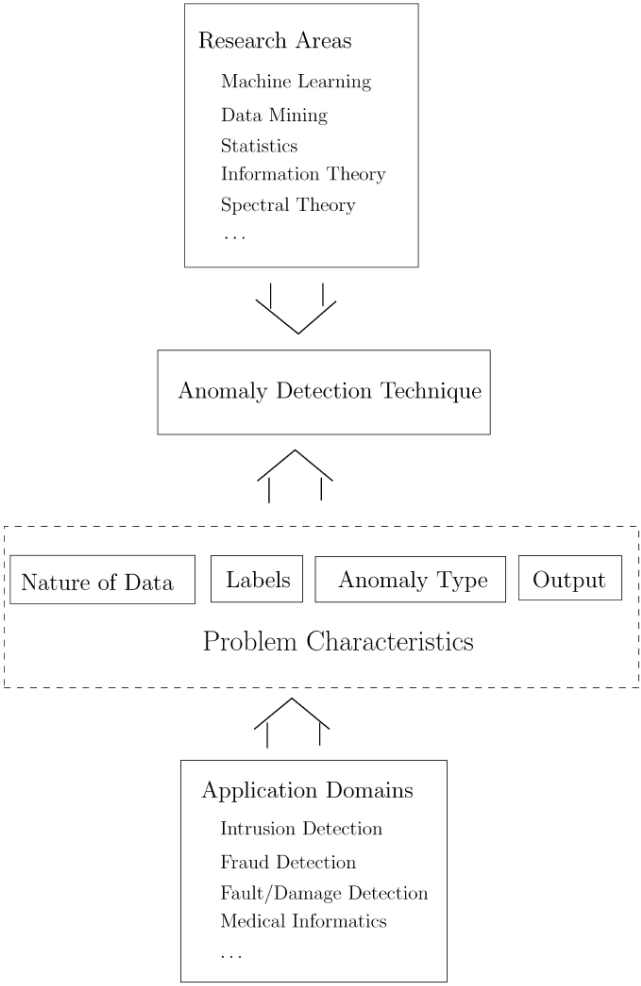

Bild 2.23: Einflussfaktoren bei der Auswahl eines Anomalieerkennungsalgorithmus[CBK09]

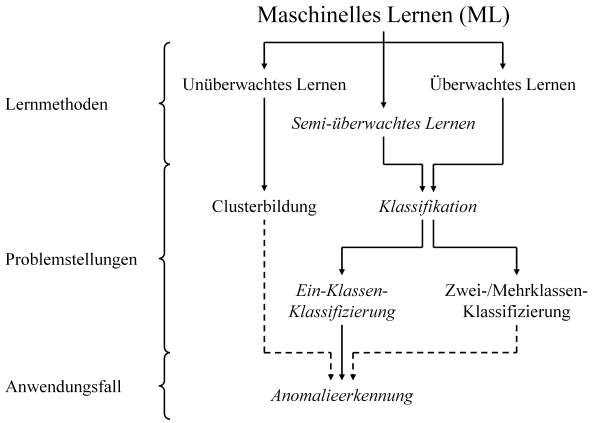

Bild 2.24: Anomalieerkennung als Anwendungsfall des maschinellen Lernens

Innerhalb der Problemstellung Klassifikation lässt sich nochmals zwischen einer *Ein-Klassen-Klassifizierung* (engl.: One-Class Classification) und einer *Zwei-/Mehrklassen-Klassifizierung* unterscheiden, wobei die Anomalieerkennung in der Regel ein Anwendungsfall der Ein-Klassen-Klassifizierung darstellt. Die einzige bekannte Klasse repräsentiert dabei die *normalen* Daten[9]. Anormale Datenpunkte liegen außerhalb und werden nicht weiter unterteilt beziehungsweise klassifiziert.
Im Gegensatz dazu sind bei einer klassischen Zwei-/Mehrklassen-Klassifizierung alle Klassen vorgegeben. Das folgende Beispiel verdeutlicht den Unterschied: Soll eine Bilderkennung Autos von anderen Verkehrsteilnehmern unterscheiden, existiert bei einer Ein-Klassen-Klassifizierung lediglich die Klasse *Auto* - und implizit die Klasse *kein Auto*. Eine Mehrklassen-Klassifizierung würde zwischen *Auto*, *Bus*, *Fahrrad* usw. differenzieren.
Die folgenden Abschnitte erläutern und diskutieren drei ausgesuchte Verfahren, welche im Rahmen dieser Arbeit näher betrachtet wurden. Alle drei

[9] Es eignen sich auch Verfahren der Clusterbildung und der Zwei-/Mehrklassen-Klassifizierung zur Anomalieerkennung. Diese spielen für die vorliegende Arbeit jedoch keine Rolle.

realisieren eine Ein-Klassen-Klassifizierung. Daher sind die relevanten Begriffe in Bild 2.24 kursiv geschrieben.

2.3.1 Neuronale Netze und Autoencoder

Neuronale Netze kommen bei verschiedenen Aufgabenstellungen erfolgreich zum Einsatz. Ein prominentes Beispiel ist die Bildklassifizierung beziehungsweise Objekterkennung mittels Convolutional Neural Networks (CNN). Diese Netze lernen in der Regel - zumindest teilweise - überwacht. Sogenannte Autoencoder werden hingegen mit Daten einer Klasse trainiert und lernen damit semi-überwacht. Anwendung finden sie unter anderem bei der Datenkomprimierung, der Dimensionsreduktion und in der Anomalieerkennung. Die folgende Beschreibung der Funktionsweise ist angelehnt an [VWeb+18b].
Autoencoder basieren auf dem mehrschichtigen Perceptron, welches in Abschnitt 2.2.3 vorgestellt wurde, und sind somit feed-forward Netze ohne interne Rückkopplungen. Sie versuchen die Eingangsvektoren am Ausgang zu reproduzieren. Werden in den verdeckten Schichten weniger Neuronen eingesetzt als am Ein- und Ausgang (diese Eigenschaft wird auch *Bottleneck* genannt), liegen dort die Eingangsdaten in einer komprimierten Form vor. Diese Idee wurde bereits 1985 von Ackley et al. [AHS85] formuliert und untersucht. Rumelhart et al. griffen das Problem 1986 auf [RM86] und Cottrell et al. [CMZ89] entwickelten das Konzept weiter. Damals lag der Fokus auf der beschriebenen Datenkomprimierung, speziell im Hinblick auf das Übertragen von Informationen. Erste Autoencoder bildeten beispielsweise acht binäre und exklusiv aktive Eingänge auf drei versteckte Neuronen ab und stellten sie auf Ausgangsseite wieder her. Ein mögliches Lernergebnis ist hierbei die binäre Repräsentation der Eingänge in der versteckten Schicht.

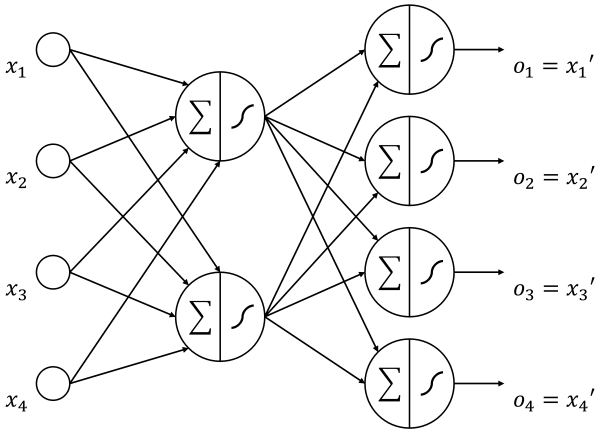

Bild 2.25: Exemplarischer Autoencoder[10]

Da Autoencoder versuchen die Eingangswerte x_i am Ausgang zu reproduzieren (o_i), müssen in der Ausgangsschicht gleich viele Neuronen vorhanden sein wie in der Eingangsschicht. In Bild 2.25 ist ein exemplarischer Autoencoder dargestellt. Er besitzt vier Eingangsneuronen (links), zwei Neuronen in der versteckten Schicht (Mitte) und wiederum vier in der Ausgangsschicht (rechts). Die Eingangsneuronen sind kleiner gezeichnet, da diese lediglich Verteilungspunkte für die Eingangswerte darstellen. Sie besitzen keine Summation und keine Aktivierungsfunktion. Die Neuronen der versteckten sowie der Ausgangsschicht verwenden in diesem Beispiel eine sigmoidale Aktivierung ($sig(x) = \frac{1}{1+e^{-x}}$). Die Kantengewichte $w_{j,i}$ sowie der jeweilige Offset sind aus Gründen der Übersichtlichkeit nicht eingezeichnet.
Für den Anwendungsfall der Anomalieerkennung führten Hawkins et al. [Haw+02] [Wil+02] diese Art neuronaler Netze als Replicator Neural Net-

[10] Adaptiert von [VWeb+18b]

works (RNN) ein[11]. Allerdings hat sich der Begriff Autoencoder in der Literatur durchgesetzt. Außerdem besteht bei RNN die Verwechslungsgefahr mit rekurrenten neuronalen Netzen, welche ebenfalls mit RNN abgekürzt werden. Daher wird im Rahmen dieser Arbeit der Begriff Autoencoder verwendet.

Das Ziel ist, einen Autoencoder so zu trainieren, dass er bei der anschließenden Inferenz *normale* Eingangsdaten möglichst genau und *anormale* mit größer Abweichung reproduziert. Während des Trainings, repräsentieren die Eingangsvektoren gleichzeitig die erwartete Ausgabe. Nachdem ein Autoencoder den Ausgangsvektor berechnet hat, wird dieser mit dem dazugehörigen Eingangsvektor verglichen und die Kantengewichte so angepasst, dass die Differenz beim nächsten Versuch geringer ausfällt. Damit lernt der Autoencoder die Trainingsdaten - und bei erfolgreicher Generalisierung auch dazu ähnliche Eingangsvektoren - immer besser zu reproduzieren. Außerdem gilt die Annahme, dass davon abweichende Eingangsdaten einen größeren Reproduktionsfehler verursachen. Bestehen die Trainingsdaten ausschließlich oder zu größten Teilen aus normalen Eingangsvektoren, lassen sich damit Anomalien erkennen.

Da keine explizite Annotation vorliegt und implizit davon ausgegangen wird, dass nur normale Trainingsdaten vorhanden sind, handelt es sich um semi-überwachtes Lernen (siehe Abschnitt 2.2.2).

Die Klassifizierung von Eingangsvektoren während der Inferenz erfolgt anhand des Reproduktionsfehlers. Dieser wird von Hawkins et al. als *Outlier Factor* (OF) bezeichnet und ist als die mittlere quadratische Abweichung über alle n Merkmale definiert [Haw+02], siehe Formel 2.9. x_i ist dabei der i-te Wert (Merkmal) des Eingangsvektors und o_i der dazugehörige, repro-

[11] In der Literatur werden die beiden Begriffe nicht ganz einheitlich verwendet. Während beispielsweise Dau et al. [DCS14] Autoencoder und RNN als Synonyme betrachten, beziehen unter anderem Tóth et al. [TG04] den Begriff RNN auf die besondere, treppenförmige Aktivierungsfunktion in der mittleren versteckten Schicht.

duzierte Wert. Liegt der OF oberhalb eines definierten Schwellwerts OF_S, handelt es sich um eine Anomalie, anderenfalls um normale Eingangsdaten.

$$OF = \frac{1}{n}\sum_{i=1}^{n}(x_i - o_i)^2 \tag{2.9}$$

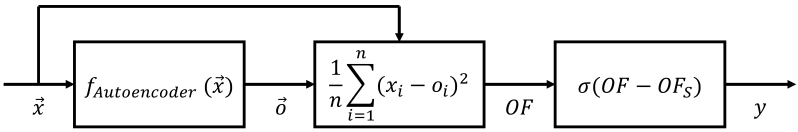

Bild 2.26: Anomalieerkennung mittels Autoencoder [VWeb+18b]

Bild 2.26 fasst den Inferenz-Ablauf zur Anomalieerkennung mittels Autoencoder zusammen. Der linke Block stellt die erlernte Reproduktionsfunktion $f_{Autoencoder}$ in Abhängigkeit des Eingangsvektors $\vec{x}$ dar. Nach der Abbildung auf $\vec{o}$ findet die Berechnung des OF im mittleren Block statt. Als letzter Schritt erfolgt die Anwendung des Schwellwerts OF_S, dargestellt über die Sprungfunktion σ. Der binäre Ausgangswert y ist 0 bei einem normalen Eingangsvektor und 1 bei einer erkannten Anomalie.

Zahlreiche Wissenschaftler optimierten den Autoencoder-Ansatz für unterschiedliche Anwendungsfälle und Szenarien. Beispiele dafür sind Stacked/Deep Autoencoder [HS06], Sparse Autoencoder [Ng], Denoising Autoencoder, Contractive Autoencoder [Rif+11], Convolutional Autoencoder, Variational Autoencoder [Pu+16] und Kombinationen davon [Vin+10] [Mas+11]. Viele dieser Varianten wurden mit Fokus auf die Bildverarbeitung entwickelt, können aber auch bei der Anomalieerkennung hilfreich sein.

Zur Verarbeitung von kontinuierlichen Datenströmen und Zeitreihen eignen sich besonders rekurrente Netzarchitekturen (engl. Recurrent Neural Networks) [LBE15]. Diese enthalten Rückkopplungen, welche die Speicherung

von internen Zuständen erlauben. Eine spezielle und aktuell weit verbreitete Form ist das sogenannte Long Short-Term Memory (LSTM) [HS97], welches vor allem in den Bereichen der Spracherkennung und Textverarbeitung große Erfolge aufweisen kann. Google verwendet LSTMs unter anderem für die Spracherkennung und zur automatischen Umwandlung von Sprache in Text [Bea15] [SSB14], zur Übersetzung [Wu+16] sowie in seiner Nachrichten App *Allo* [Kha16]. Amazon nutzt sie für den Sprachassistenten Alexa [Met18] und Microsoft verwendet diese Netze ebenfalls zur Spracherkennung [Xio+17]. Auch die Kombination aus rekurrenten Netzen beziehungsweise LSTMs und Autoencodern (Recurrent Autoencoder beziehungsweise LSTM Autoencoder) wird bereits erforscht [Ami+17] [Hsu17].
Der Einsatz von LSTMs zur Anomalieerkennung in Zeitreihen ist ebenfalls möglich, wie einige Arbeiten dazu zeigen [Mal+15] [Mal+16] [Bon+17] [Sin17] [Du+17]. Zum gleichen Zweck verwenden Marchi et al. [Mar+15], Wolpher [Wol18a] und Grover [Gro18] wiederum eine Kombination aus LSTM und Autoencoder.

2.3.2 Ensemble-basierte Algorithmen und LODA

In den letzten Jahren wurden speziell für die Anomalieerkennung entwickelte Algorithmen vorgestellt, die auf einem homogenen Ensemble basieren. Dabei werden viele *schwache* Klassifikatoren des gleichen Typs trainiert, welche in Summe aber vielversprechende Ergebnisse liefern. Diese Algorithmen bieten zusätzlich die Möglichkeit zur parallelen Berechnung einzelner Klassifikatoren. Ein bekannter und inzwischen beliebter Algorithmus dieser Art ist der Isolation Forest (iForest). Er wurde 2008 von Liu et al. präsentiert [LTZ08] und danach weiter erforscht [LTZ12]. Bei einer empirischen Analyse der Autoren war iForest anderen Algorithmen bezüglich Klassifizierungsleistung und benötigter Rechenzeit überlegen [LTZ12]. Sein Einsatz zur Anomalieerkennung in Datenströmen wurde von Ding und Fei untersucht [DF13] und Sun et al. verwenden ihn zur Erkennung von un-

natürlichem Benutzerverhalten [Sun+16]. Hofmockel et al. ermitteln mittels iForest Anomalien in Sensordaten, die in Fahrzeugen gesammelt und in einem Backend-System ausgewertet werden [HS18]. Der Grundgedanke des Algorithmus besteht darin, dass Anomalien einfacher zu separieren sind als normale Datenpunkte. Während des Trainings selektiert iForest zunächst ein zufälliges Merkmal. Danach unterteilt er ein Subset der Trainingsdaten anhand dieses Merkmals wiederum zufällig und solange, bis alle Datenpunkte des Subsets isoliert sind. Dadurch entsteht ein binärer Entscheidungsbaum. Dieses Vorgehen wiederholt sich x-mal (x ist ein festzulegender Hyperparameter). Es entstehen x Entscheidungsbäume, wobei jeder Baum auf einem anderen, zufälligen Merkmal basiert. Bei der Inferenz traversiert ein neuer Datenpunkt alle Entscheidungsbäume. Die dabei zurückgelegte, durchschnittliche Pfadlänge bildet die Grundlage der Anomaliebewertung. Es gilt die Annahme, dass anormale Datenpunkte einfacher zu separieren sind und daher die durchschnittliche Pfadlänge kürzer ausfällt.
Eine Optimierung von iForest sind die sogenannten Half-Space-Trees (HS-Trees) [TTL11]. Nach der Merkmalsauswahl werden die Trainingsdaten nur bis zu einer vordefinierten Baumtiefe unterteilt. Jedes Blatt eines Baums repräsentiert daher nicht einen einzelnen Datenpunkt sondern eine bestimmte Anzahl davon. Diese Anzahl wird am Blatt annotiert und stellt die *Masse* des verbleibenden Zweigs dar. Während der Inferenz traversiert ein neuer Datenpunkt wiederum alle erstellten Entscheidungsbäume. Grundlage der Anomaliebewertung ist jetzt nicht mehr die durchschnittliche Pfadlänge sondern die durchschnittlich erreichte Masse. Diese Optimierung ermöglicht beziehungsweise vereinfacht das Trainieren parallel zur Inferenz (engl.: online learning), da die einzelnen Massen zur Laufzeit angepasst werden können.
Mit Lightweight On-line Detector of Anomalies (LODA) stellte Pevný 2016 einen weiteren Ensemble-basierten Algorithmus zur Anomalieerkennung vor [Pev16]. Im Falle von LODA dienen Histogramme von zufälligen Projektionen des Eingangsvektors als Bewertungsgrundlage. Die einzelnen

Auftrittswahrscheinlichkeiten werden anschließend in einem gemeinsamen Klassifikator zusammengefasst. Nachfolgend ist die Funktionsweise von LODA detaillierter beschrieben, angelehnt an [VWeb+18a].
Wie bereits erwähnt, erzeugt LODA zufällige Projektionen der Eingangswerte. Dies geschieht indem ein Eingangsvektor $\vec{x}$ nacheinander mit mehreren Projektionsvektoren $\vec{w}_i$ gleicher Dimension d multipliziert wird. Das Ergebnis ist jeweils ein skalarer Wert z_i, wobei der Index i für die i-te Projektion steht, siehe Formel 2.10.

$$z_i = \vec{x}^T \vec{w}_i \tag{2.10}$$

Zu Beginn generiert LODA die dünn besetzten Projektionsvektoren. Dabei wählt der Algorithmus jeweils $d^{-\frac{1}{2}} \cdot 100\%$ der Elemente eines Projektionsvektors zufällig aus und belegt diese wiederum mit zufälligen Werten entsprechend der Normalverteilung $\mathcal{N}(0,1)$. Die restlichen Elemente werden auf null gesetzt. Pevný begründet die Verwendung der Normalverteilung für nicht-null Elemente mit dem Johnson–Lindenstrauss Lemma [Pev16]. Dieses besagt, dass damit der L_2-Abstand (euklidische Distanz) zwischen Datenpunkten im Projektionsraum näherungsweise dem im Originalraum entspricht [JL84]. Durch die dünne Besetzung entstehen zudem vielfältige Unterräume, welche der Algorithmus zunächst separat bewertet. Die Anzahl der nicht-null Elemente bestimmt die Dimension der Unterräume und wurde nach Li gewählt [Pev16]. Li zeigte, dass bei zufälligen Projektionen der L_2-Abstand beibehalten werden kann, auch wenn nur $d^{-\frac{1}{2}} \cdot 100\%$ der Elemente belegt sind [Li07].
In der Trainingsphase projiziert LODA jeden Eingangsvektor nach Formel 2.10 in die verschiedenen Unterräume. Danach folgt die Einsortierung der skalaren Ergebnisse z_i in die dazugehörigen Histogramme h_i. Der jeweilige Wertebereich ist dabei in b gleich breite Intervalle (engl.: bins) unterteilt [Pev16]. Für die Berechnung der Histogramm-spezifischen Intervallanzahl gibt es unterschiedliche Ansätze, unter anderem Sturges Formel [Stu26], die

Regel nach Scott für normalverteilte Daten [Sco79] sowie die Regel nach Freedman und Diaconis [FD81]. Häufig kommt Sturges Formel zum Einsatz, die in Formel 2.11 dargestellt ist und in der n die Anzahl an Datenpunkten repräsentiert.

$$b = 1 + \log_2 n \tag{2.11}$$

Diese Formel ist für große Datensätze zu konservativ. LODA verwendet stattdessen das Vorgehen nach Birgé und Rozenholc [BR06] [Dav+09], welches die größte, benachteiligte Wahrscheinlichkeit L_n maximiert. Formel 2.12 zeigt die dazugehörige Berechnungsvorschrift für L_n, in der b und n wiederum für die Intervallanzahl beziehungsweise die Anzahl an Datenpunkten stehen. n_i ist die Anzahl der Datenpunkte, welche in den i-ten Intervall fallen. Der rechte Teil von Formel 2.12 sorgt für die Benachteiligung von zu vielen Intervallen.

$$L_n(b) = \sum_{i=1}^{b} n_i \log\left(\frac{bn_i}{n}\right) \underbrace{-(b - 1 + (\log b)^{2,5})}_{\substack{\text{Benachteiligung für eine} \\ \text{zu große Intervallanzahl } b}} \tag{2.12}$$

Am Ende des Trainings existieren k befüllte Histogramme. Um bei der Inferenz einen neuen Eingangsvektor $\vec{x}$ zu klassifizieren, ermittelt LODA zunächst die jeweilige Auftrittswahrscheinlichkeit $\hat{p}_i$ des projizierten Datenpunkts z_i im Histogramm h_i. Dazu wird die Anzahl der Datenpunkte, die im gleichen Intervall liegen wie z_i, durch die Gesamtanzahl an Datenpunkten im Histogramm (n) dividiert. Dieses Vorgehen wiederholt sich für alle Projektionen. Die finale Anomaliebewertung berechnet sich nach Formel 2.13.

$$f(\vec{x}) = -\frac{1}{k}\sum_{i=1}^{k} \log \hat{p}_i(\vec{x}^T \vec{w}_i) = -\log\left(\prod_{i=1}^{k} \hat{p}_i(\vec{x}^T \vec{w}_i)\right)^{\frac{1}{k}} \tag{2.13}$$

Der Ausgabewert ist damit proportional zur negativen, logarithmischen Auftrittswahrscheinlichkeit eines Eingangsvektors, d.h. je unwahrscheinlicher ein Eingangsvektor, desto größer die zugewiesene Anomaliebewertung [Pev16]. Die endgültige Klassifizierung in normal beziehungsweise anormal erfolgt anhand eines Schwellwerts.
LODA bestimmt die Anzahl der Projektionen (k) ebenfalls automatisch. Dazu berechnet der Algorithmus die durchschnittliche Änderung der Anomliebewertung nach dem Hinzufügen einer Projektion, siehe Formel 2.14. Die Größenordnung von $\hat{\sigma}_k$ ist abhängig vom Anwendungsfall und wird daher auf $\hat{\sigma}_1$ normiert. Letztendlich verwendet LODA die Anzahl an Projektionen, für die Formel 2.15 gilt, wobei der Schwellwert τ auf 0,01 gesetzt ist.

$$\hat{\sigma}_k = \frac{1}{n} \sum_{i=1}^{n} |f_{k+1}(\vec{x}_i) - f_k(\vec{x}_i)| \tag{2.14}$$

$$\arg\min_k \frac{\hat{\sigma}_k}{\hat{\sigma}_1} \geq \tau \tag{2.15}$$

2.3.3 One-Class Support Vector Machine

Ein weiterer Algorithmus zur Anomalieerkennung ist die One-Class Support Vector Machine (OCSVM). Da dieser im Rahmen der vorliegenden Arbeit hauptsächlich zum Vergleich mit den anderen Verfahren herangezogen wird, beschränkt sich die nachfolgende Beschreibung auf die prinzipielle Funktionsweise. Detailliertere Informationen inklusive der mathematischen Formeln sind in weiterführender Literatur zu finden [Sch+00] [VGWS18].
Ähnlich zu LODA wurde auch die OCSVM speziell zur Anomalieerkennung entworfen und im Jahr 2000 von Schölkopf et al. vorgestellt [Sch+00]. Als Basis dient die klassische Support Vector Machine (SVM) von Vapnik [Vap95], welche zwei Punktewolken im n-dimensionalen Raum durch eine lineare Hyperebene trennt. Die Hyperebene wird dabei so berechnet, dass der Abstand zu den nächstgelegenen Punkten - den sogenannten *Stützvektoren* (engl.: Support Vectors) - maximal ist. Die Stützvektoren sind gleich-

zeitig das Ergebnis des Trainingsvorgangs. Allerdings unterscheidet sich der Trainingsablauf für eine SVM (und für LODA) vom Training eines neuronalen Netzes. Während sich die Hyperebene über ein Optimierungsproblem deterministisch berechnen lässt, findet bei neuronalen Netzen ein iterativer Prozess Anwendung, der aufgrund von zufälligen Initialwerten üblicherweise nicht-reproduzierbare Ergebnisse liefert. Zusätzlich muss das Training neuronaler Netze an geeigneter Stelle abgebrochen werden. Bei der Inferenz berechnet eine SVM die Lage neuer Datenpunkte relativ zur Hyperebene und entscheidet auf Basis dessen über die Klassenzugehörigkeit. Der linke Teil von Bild 2.27 zeigt die prinzipielle Funktionsweise einer SVM anhand zweidimensionaler Trainingsdaten mit den Merkmalen x_1 und x_2. Im Beispiel trennt die berechnete Hyperebene normale von anormalen Daten. Punkte mit Pfeil sind Stützvektoren. Liegen neue Datenpunkte links der Hyperebene, klassifiziert die SVM diese als normal, anderenfalls als anormal.

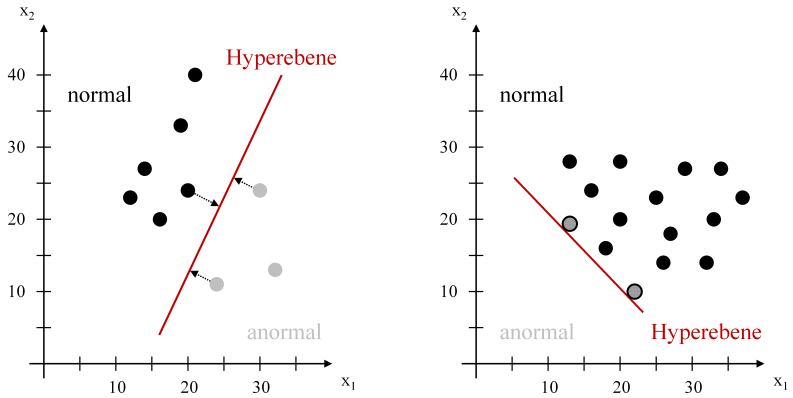

Bild 2.27: Unterschied zwischen SVM (links) und OCSVM (rechts)

Die klassische SVM verwendet überwachtes Lernen und benötigt damit annotierte Trainingsdaten, im obigen Beispiel Punkte der beiden Klassen normal und anormal. Im Gegensatz dazu arbeitet eine OCSVM semi-

überwacht. Sie wird lediglich mit Eingangswerten einer Klasse trainiert, im Bereich der Anomalieerkennung sind dies üblicherweise *normale* Daten. Anstatt Trainingsdaten zweier Klassen zu separieren, trennt die berechnete Hyperebene einer OCSVM alle Datenpunkte mit maximalem Abstand vom Ursprung. Dieses Verhalten ist im rechten Teil von Bild 2.27 dargestellt. Die schraffiert gefüllten Datenpunkte repräsentieren in diesem Fall die Stützvektoren. Alle neuen Datenpunkte, die in den Bereich hoher Dichte fallen (im Beispiel rechts der Hyperebene) werden der normalen Klasse zugeordnet. Punkte im Bereich niedriger Dichte (im Beispiel links der Hyperebene und näher am Ursprung) klassifiziert die OCSVM als Anomalie.

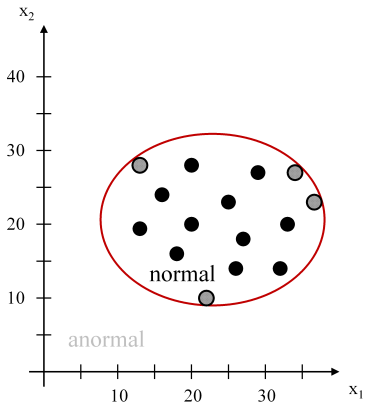

Bild 2.28: Nichtlineare Klassifizierung mittels OCSVM und Kernel-Trick

Sowohl SVM als auch OCSVM berechnen lineare Hyperebenen. Eine lineare Separierbarkeit der Eingangsdaten ist für die meisten realen Anwendungsfälle jedoch nicht gegeben. Abhilfe schafft der sogenannte *Kernel-Trick*, mit dessen Hilfe die Datenpunkte in einen neuen Merkmalsraum höherer (theoretisch unendlicher) Dimension überführt werden. Eine lineare Hyperebene in diesem Raum resultiert in einem nicht-linearen Klassifikator im originalen Merkmalsraum, beispielhaft dargestellt in Bild 2.28.

Die für den Kernel-Trick vielfach eingesetzte radiale Basisfunktion ist parametrierbar, sodass für die darin enthaltene Variable σ ein passender Wert zu finden ist. Zudem erlaubt die OCSVM mittels sogenannten Schlupfvariablen kleine Verletzungen des Klassifikators. Über den Hyperparameter ν lässt sich der maximale Anteil von Datenpunkten definieren, welche die zu berechnende Trennung verletzen. Gleichzeitig bestimmt dieser Wert eine anteilsmäßige Untergrenze von Stützvektoren.
Parallel zur OCSVM schlug Tax 1999 mit der Support Vector Domain/Data Description (SVDD) eine weitere Form der Stützvektor-basierten Anomalieerkennung vor [TD99] [Tax01], die später unter anderem von Theissler aufgegriffen wurde [The14]. Anstatt die Datenpunkte mittels Hyperebene vom Ursprung zu trennen, ist das Ziel der SVDD eine umschließende Kugel mit minimalem Volumen zu berechnen. Wird für eine OCSVM die radiale Basisfunktion als Kernel verwendet, verhalten sich beide Verfahren äquivalent und unterscheiden sich lediglich anhand unterschiedlicher Parametrierungsmöglichkeiten.

2.3.4 Anomaliebewertung

Zum Vergleich der Leistungsfähigkeit unterschiedlicher Algorithmen, sind Metriken notwendig. Bei Klassifizierungsaufgaben ist in der Regel eine eindeutige Zuordnung eines Ergebnisses zu *richtig* oder *falsch* möglich. In der Bildverarbeitung wird ein Bild richtig klassifiziert - z.B. als Fußgänger - oder falsch - z.B. ein Fußgänger als Fahrzeug. Eine populäre Methode der entsprechenden Ergebnisaufbereitung ist die sogenannte *Confusion Matrix*, welche die prozentuale Aufteilung zwischen *True Positives* (TP), *False Positives* (FP), *True Negatives* (TN) und *False Negatives* (FN) darstellt. TPs sind Datenpunkte, die in Bezug auf die gesuchte Klasse richtig einsortiert wurden. Wird das Bild eines Fußgängers als Fußgänger klassifiziert, handelt es sich dementsprechend um ein TP. Im Gegensatz dazu sind FPs Datenpunkte, welche fälschlicherweise dieser Klasse zugeordnet werden - zum Bei-

spiel die Klassifizierung von Fahrzeugbildern als Fußgänger. Analog dazu handelt es sich bei TNs um richtig klassifizierte Datenpunkte einer anderen Klasse und FNs sind Datenpunkte der gesuchten Klasse, die ein Algorithmus nicht als solche erkennt. Diese vier Basis-Metriken sind auch bei der Anomalieerkennung hilfreich, wobei die gesuchten Datenpunkte in diesem Fall die Anomalien sind. Damit sind gefundene Anomalien TPs. Die gesamte Confusion Matrix ist in Tabelle 2.1 dargestellt.

		Originale Klasse	
		Anomalie	*Normal*
Klassifizierungs-ergebnis	*Anomalie*	True Positive (TP)	False Positive (FP)
	Normal	False Negative (FN)	True Negative (TN)

Tabelle 2.1: Confusion Matrix bezogen auf Anomalieerkennung [VWeb+]

Die *True Positive Rate* (TPR) und die *False Positive Rate* (FPR) sind von den Basis-Metriken abgeleitete Kennzahlen. Formel 2.16 zeigt die Berechnungsvorschrift der TPR, welche die Erkennungsrate der vorhandenen Anomalien repräsentiert. In der Literatur wird die TPR auch als Sensitivity, Hit Rate oder Recall bezeichnet [Faw06]. Die Berechnung der FPR, auch bekannt unter Falschalarmrate (engl.: False Alarm Rate) [Faw06], erfolgt nach Formel 2.17. Als weitere wichtige Kennzahl stellt die Präzision (engl.: Precision (PRC)) das Verhältnis von erkannten zu gemeldeten Anomalien dar, siehe Formel 2.18.

$$TPR = \frac{TP}{TP+FN} \tag{2.16}$$

$$FPR = \frac{FP}{FP+TN} \tag{2.17}$$

$$PRC = \frac{TP}{TP+FP} \tag{2.18}$$

Sofern es sich um eine binäre Klassifikation handelt (wie bei der Anomalieerkennung), enthalten die vier Basis-Metriken alle Leistungsinformationen [Bal+00]. Zusätzlich existieren Metriken, die versuchen das Gesamtergebnis in einem skalaren Wert abzubilden. Ein bekanntes Beispiel hierfür ist der *F-Score* aus Formel 2.19 [van79] [Chi92] beziehungsweise der speziellere *F_1-Score*, welcher mit $\beta = 1$ das harmonische Mittel aus PRC und TPR bildet. Andere Werte von β entsprechen einer höheren Gewichtung von PRC oder TPR [Sas07].

$$F_\beta - Score = \frac{(\beta^2 + 1) \cdot PRC \cdot TPR}{\beta^2 \cdot PRC + TPR} \tag{2.19}$$

$$\text{mit } \beta = 1 : F_1 - Score = \frac{2 \cdot PRC \cdot TPR}{PRC + TPR} = \frac{2 \cdot TP}{2 \cdot TP + FN + FP}$$

Im automotive Umfeld ist neben der Erkennungsrate die Falschalarmrate besonders interessant. Aufgrund der Praxisrelevanz und der guten Interpretierbarkeit werden im Rahmen dieser Arbeit hauptsächlich TPR und FPR als Leistungsmetriken verwendet, wobei eine hohe TPR bei möglichst niedriger FPR das Ziel ist. Eine perfekte Klassifizierung entspricht demnach 100% TPR bei 0% FPR.

Receiver Operating Characteristics

Viele Anomalieerkennungsalgorithmen berechnen einen kontinuierlichen Wert. Dieser repräsentiert den Grad der Anomalität eines Datenpunkts beziehungsweise eine Anomaliewahrscheinlichkeit. Die endgültige Klassifizierung findet in der Regel durch die Anwendung eines Schwellwerts statt, wie beispielsweise für Autoencoder (Abschnitt 2.3.1) und LODA (Abschnitt 2.3.2) beschrieben. Die Receiver Operating Characteristics (ROC)-

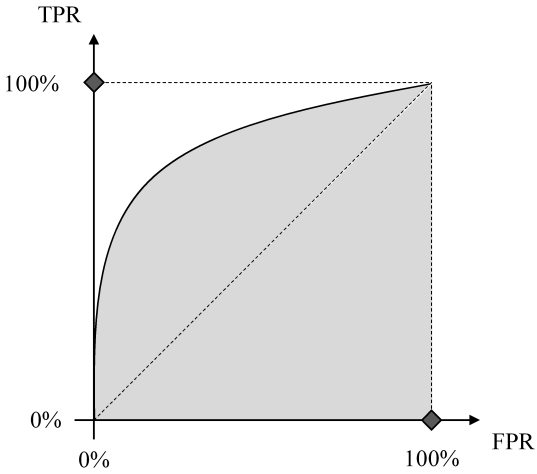

Bild 2.29: Beispielhafte ROC-Kurve

Kurve stellt die TPR über der entsprechenden FPR in Abhängigkeit dieses Schwellwerts dar und ist monoton steigend [Agg17], siehe Bild 2.29. Mit zunehmender Anzahl von anormal klassifizierten Datenpunkten, steigt neben der Erkennungs- auch die Falschalarmrate. Die Diagonale zwischen (0|0) und (100|100) ist dabei die erwartete Leistungsfähigkeit einer zufälligen Entscheidung. Je weiter die ROC-Kurve auf der linken Seite dieser Diagonalen liegt, desto größer ist die Fläche unter der Kurve (engl.: Area Under Curve (AUC)) und desto besser ist der untersuchte Algorithmus. Die ROC AUC ist damit eine weitere Metrik, welche im Rahmen der vorliegenden Arbeit jedoch keine Anwendung findet. Bereiche mit hoher FPR sind für die Praxis uninteressant und sollen daher nicht in die Bewertung einfließen.

3 Sicherheit in automotive Steuergeräte-Software

In der deutschen Sprache ist der Begriff Sicherheit alleine doppeldeutig. Es muss zwischen funktionaler Sicherheit (Safety) und Angriffssicherheit (Security) unterschieden werden. Getrieben vom steigenden Einfluss der Elektrik und Elektronik fokussierte die Automobilindustrie in der Vergangenheit klar die funktionale Sicherheit. Ausfälle oder Fehlfunktionen von E/E-Komponenten dürfen die Fahrzeuginsassen nicht gefährden. Dies gilt insbesondere für Funktionen und Komponenten, die das Fahrverhalten aktiv beeinflussen können. Eine Fehlfunktion des ESP oder des verwendeten Kommunikationssystems darf beispielsweise kein grundloses Blockieren eines Rades verursachen. Die entwickelten Absicherungsmaßnahmen adressieren dabei die potentiellen Fehlfunktionen im E/E-System. Eine Erhöhung der Angriffssicherheit spielte dabei eine untergeordnete Rolle. Bis vor wenigen Jahren war ein Fahrzeug ein in sich geschlossenes System. Für das beabsichtigte Herbeiführen eines Fehlverhaltens war physischer Zugriff auf das Fahrzeug notwendig, um beispielsweise falsche CAN-Botschaften in das Fahrzeugnetzwerk einzuspeisen. Mit physischem Zugriff könnte ein Angreifer allerdings auch eine Bremsleitung durchtrennen. Daher sahen viele Fahrzeughersteller lange nicht die Notwendigkeit das E/E-System vor gezielten Manipulationen zu schützen. Ausnahme war das Aktualisieren von Steuergeräte-Software. Hier wurden frühzeitig Mechanismen für sicheres Re-Programmieren eingeführt. Dazu zählt eine rudimentäre Absicherung der Fahrzeugdiagnose sowie die Signierung der eigentlichen Steuergeräte-Software. Durch die zunehmende Vernetzung von Fahrzeugen mit der Au-

ßenwelt und die dazugehörigen Luftschnittstellen ist die bisherige Argumentation nicht mehr gültig. Hacker und Wissenschaftler zeigten öffentlichkeitswirksam, dass physischer Zugriff nicht weiter notwendig ist, um das E/E-System und das Verhalten eines Fahrzeugs zu manipulieren [Che+11] [MV15]. Dadurch ändert sich momentan die Einstellung von Fahrzeugherstellern bezüglich Angriffssicherheit grundlegend und Security ist inzwischen ein zentraler Punkt in der E/E-Entwicklung.
Zunächst folgt eine Betrachtung von Steuergeräte-Software in Bezug auf die Gewährleistung funktionaler Sicherheit. Im Anschluss gibt dieses Kapitel einen Überblick bezüglich der momentan in Steuergeräten eingesetzten Mechanismen, welche das E/E-System vor gezielten Manipulationen schützen. Dabei liegt am Ende der Fokus speziell auf Firewalls und Intrusion Detection Systemen.

3.1 Safety-Mechanismen

Safety ist seit vielen Jahren wichtiger Bestandteil in der Entwicklung von automotive Steuergeräten. Im November 2011 wurde dazu der ISO 26262 Standard *Road vehicles – Functional safety* veröffentlicht (Teile 1-9, Teil 10 folgte im August 2012) [Intb] und seit 2016 arbeitet die Industrie an einer aktualisierten und erweiterten Ausgabe. Der Standard definiert die *Automotive Safety Integrity Level* (ASIL) A bis D, welche die automotive-spezifische Anpassung der *Safety Integrity Level* aus IEC 61508 [Int10] darstellen. Grundlage der ASIL-Klassifizierung bildet eine Gefahren- und Risikoanalyse (Hazard Analysis and Risk Assessment (HARA)). Nach der Festlegung des zu betrachtenden Systems, wie beispielsweise einer Fahrerassistenzfunktion, müssen zunächst die Gefahren ermittelt werden, die durch eine potentielle Fehlfunktion auftreten können. Im nächsten Schritt ist jede identifizierte Gefahr bezogen auf unterschiedliche Fahrsituationen zu bewerten. Das Risiko ergibt sich aus der Schwere der Auswirkung (Severity), der Auftrittshäufigkeit der betrachteten Fahrsituation (Exposure) und

der entsprechenden Beherrschbarkeit durch einen durchschnittlichen Fahrer (Controllability). Mit Hilfe von vorgegebenen Tabellen findet die Einordnung in ASIL-Level statt [Wal10]. ASIL D resultiert dabei in den höchsten Sicherheitszielen und damit in den höchsten funktionalen Sicherheitsanforderungen. Als nicht sicherheitskritisch eingestufte Gefahren erhalten das *Quality Management* (QM)-Level. Wie die funktionalen Sicherheitsanforderungen erfüllt werden, zum Beispiel von welchem Steuergerät (in Hardware oder in Software), ist an dieser Stelle noch nicht definiert. Diese Entscheidung erfolgt in einem späteren Schritt bei der entsprechenden Zuordnung der abgeleiteten, technischen Sicherheitsanforderungen. Neben der Spezifikation von prozessrelevanten Aspekten und organisatorischen Voraussetzungen enthält die ISO 26262 auch Vorschläge und Empfehlungen für Methoden, die bei der Erfüllung von Sicherheitsanforderungen helfen. Für Steuergeräte-Software haben sich im Rahmen von AUTOSAR Classic und im Laufe der Zeit einige Methoden etabliert.
Die zum Schutz der Fahrzeug-internen Kommunikation verwendete Ende-zu-Ende Absicherung ist nachfolgend beschrieben. Alle weiteren Mechanismen, welche die Überwachung der Laufzeitumgebung sowie der Steuergeräte-internen Abläufe betreffen, sind in Anhang A.2 zu finden. Obwohl die Mechanismen anhand von AUTOSAR Classic erklärt werden, lassen sich die Prinzipien auf andere Software-Architekturen übertragen.

3.1.1 End-to-End Protection

Die Ende-zu-Ende Absicherung (engl.: End-to-End (E2E) Protection) schützt ausgetauschte Applikationsdaten innerhalb von Kommunikationsnachrichten. Ein Sender berechnet dazu zusätzliche Kontrollinformationen und verschickt diese gemeinsam mit den Applikationsdaten. Auf Empfängerseite findet die gleiche Berechnung statt. Stimmt das Resultat mit den empfangenen Kontrollinformationen überein, ist davon auszugehen, dass die Applikationsdaten unverfälscht beim Empfänger angekommen sind. Die

Berechnung der Kontrollinformationen findet auf Senderseite üblicherweise möglichst früh statt, beispielsweise in der Applikationssoftware selbst oder mittels sogenanntem Transformer in der RTE [AUT17b] - sofern die Steuergeräte-Software auf AUTOSAR Classic basiert. Analog dazu überprüft der Empfänger die Kontrollinformationen möglichst spät. Die Applikationssoftware selbst ist nicht Teil des OSI-Modells und liegt oberhalb der Anwendungsschicht, vergleiche Bild 2.7. Daher wird mittels E2E Protection nicht nur das eigentliche Kommunikationsmedium wie CAN oder Ethernet abgesichert, sondern auch die an der Kommunikation beteiligte Basissoftware und damit alle sieben Schichten des OSI-Modells. Das Prinzip ist in Bild 3.1 veranschaulicht. Schwarze Blitze kennzeichnen potentielle Fehler im Kommunikationspfad, die mittels E2E Protection erkennbar sind. Der Mechanismus ermöglicht es ASIL Applikationen Daten über unsichere Basissoftware und Kommunikationsmedien auszutauschen. Dies gilt auch für den Steuergeräte-internen Austausch von Applikationsdaten, wobei hierfür eine sichere Middleware beziehungsweise RTE aus Performanzgründen zu bevorzugen ist.

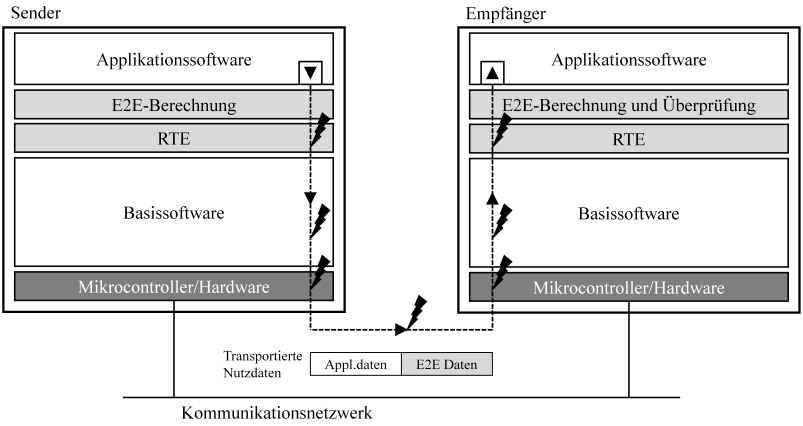

Bild 3.1: E2E Protection in einem AUTOSAR Classic-basierten Steuergerät

Üblicherweise setzen sich die Kontrollinformationen aus einem CRC und einem Sequenzzähler zusammen, wobei potentiell noch weitere Informationen Anwendung finden können. Verschiedene Kommunikationsmedien und -paradigmen benötigen unterschiedliche Kontrollinformationen, um die Integrität der Applikationsdaten zu gewährleisten. Zum Beispiel genügt bei acht Byte langen CAN-Nutzdaten normalerweise ein acht Bit CRC. Um längere Nutzdaten in Ethernet-Netzwerken auf ähnlichem Niveau abzusichern, sind hingegen 16 oder 32 Bit CRC notwendig. Auch die Position der Kontrollinformationen innerhalb der Kommunikationsnachricht kann variieren. Daher spezifiziert AUTOSAR E2E-Profile mit unterschiedlichem Fokus [AUT17c]. Diese definieren alle notwendigen Parameter, wie beispielsweise das Polynom zur Berechnung der CRC sowie die Art und Position der verwendeten Kontrollinformationen. Da alle Parameter öffentlich verfügbar sind und keine kryptografischen Verfahren Anwendung finden, bietet die E2E Protection keinerlei Schutz gegen mutwillige Manipulationen.

3.2 Security-Mechanismen

Im Gegensatz zu Safety, stand (Cyber-)Security lange Zeit nicht im Fokus vieler Fahrzeughersteller. Das E/E-System von Fahrzeugen war abgeschlossen und von außen nicht erreichbar, es sei denn man hatte physischen Zugriff. Diese Abgeschlossenheit ist aufgrund der steigenden Konnektivitätsanforderungen nicht mehr gewährleistet. Einerseits möchten Fahrzeugbesitzer ihre mobilen Endgeräte mit dem Infotainment-System koppeln und andererseits benötigen neue Sicherheitstechnologien Kommunikation mit Infrastruktur und anderen Verkehrsteilnehmern. Des Weiteren ist es das Ziel vieler Fahrzeughersteller Steuergeräte-Software im Feld aktualisieren zu können, unabhängig von einem Werkstattaufenthalt. Neben der Behebung von sicherheitskritischen Fehlern können damit auch Funktionserweiterungen angeboten werden, was neue Geschäftsmodelle ermöglicht. Die Software-Aktualisierung über Luftschnittstellen ist auch unter den Namen

Over-the-Air (OTA) Updates und *Software Over-the-Air* (SOTA) bekannt. Neben der steigenden Konnektivität nimmt die Verbreitung von erweiterten Fahrerassistenzfunktionen stark zu. Damit sind Steuergeräte vermehrt in der Lage aktiv in das Fahrverhalten einzugreifen. Diese Kombination aus entferntem Zugriff und steigendem Einfluss von Softwarefunktionen birgt große Risiken. Nachdem diese Risiken öffentlichkeitswirksam von Sicherheitsforschern aufgezeigt wurden [Che+11] [MV15] (siehe *Jeep Hack* in Unterkapitel 1.1), ist Security inzwischen ein bedeutender Aspekt bei der Entwicklung neuer automotive E/E-Systeme.

Momentan gibt es noch keinen internationalen Standard für automotive Cybersecurity, vergleichbar mit der ISO 26262 für Safety. Für klassische informationstechnische Systeme hat sich die ISO/IEC 15408 - auch bekannt als *Common Criteria for Information Technology Security Evaluation*, kurz: Common Criteria (CC) - als Standard zur Evaluation der Cybersicherheit durchgesetzt [Intc]. Darüber hinaus standardisiert die ISO/IEC 27001 Anforderungen an ein Informationssicherheits-Managementsystem, von der Einrichtung bis hin zur kontinuierlichen Verbesserung [Int13c].

Im Januar 2016 veröffentlichte die Society of Automotive Engineers (SAE) den automotive-spezifischen Standard J3061 *Cybersecurity Guidebook for Cyber-Physical Vehicle Systems* [SAE16]. Er empfiehlt die Anwendung eines Cybersecurity Prozesses über den gesamten Entwicklungs- und Lebenszyklus eines Fahrzeugs, definiert Richtlinien dafür und schlägt passende Werkzeuge sowie Methoden vor. Die National Highway Traffic Safety Administration (NHTSA) veröffentlichte im Oktober 2016 die *Cybersecurity Best Practices for Modern Vehicles* [Nat16]. Neben prozessrelevanten und organisatorischen Themen sind darin auch prinzipielle technische Aspekte und Lösungen beschrieben. Mit dem Ziel eines globalen Informationsaustauschs in Sachen automotive Cybersecurity gründete sich im Juli 2015 das *Automotive Information Sharing and Analysis Center* (Auto-ISAC) nach dem Vorbild der bereits bestehenden ISACs für andere Industriezweige. Auto-ISAC versteht sich als Plattform über die auch neu entdeckte Bedro-

hungen und Schwachstellen herstellerübergreifend ausgetauscht werden. Ein zusätzliches Arbeitsergebnis sind die *Automotive Cybersecurity Best Practices*, die einen ähnlichen Fokus haben wie der SAE J3061 Standard und das NHTSA Dokument. Für Nichtmitglieder ist jedoch nur eine Executive Summary verfügbar [Aut]. Inzwischen arbeiten ISO und SAE gemeinsam an einem internationalen Standard, dem ISO/SAE 21434 *Road vehicles – Cybersecurity Engineering*. Dieser soll das Pendant zur ISO 26262 für automotive Cybersecurity darstellen und damit keine spezifischen technischen Lösungen enthalten. Geplanter Veröffentlichungstermin ist Juni 2020. Die vorherigen Arbeiten der SAE fließen in den neuen Standard mit ein. Es ist davon auszugehen, dass - angelehnt an die ISO 26262 - eine Bedrohungs- und Risikoanalyse (Threat Analysis and Risk Assessment (TARA)) auf Basis von festzulegenden Werten (Assets) die Grundlage des weiteren Vorgehens bildet. Zudem ist zur Einordnung der untersuchten Bedrohungen die Einführung eines Cybersecurity Assurance Levels (CAL) in Diskussion, analog zu ASIL A-D. Eine Möglichkeit der dafür notwendigen Risikobewertung wird im *E-safety Vehicle Intrusion Protected Applications* (EVITA)-Projekt vorgestellt [EVI12]. Diese Methode bestimmt ein Risiko-Level anhand der Angriffswahrscheinlichkeit sowie der potentiellen Auswirkungen und hilft somit unter anderem entsprechende Gegenmaßnahmen zu priorisieren [Rud+09]. Bei Bedrohung der funktionalen Sicherheit fließt auch die Beherrschbarkeit im Fehler-/Angriffsfall mit in die Bewertung ein. Parallel zur Erstellung der ISO/SAE 21434 gibt es auf europäischer Ebene weitere Aktivitäten. Im Rahmen der United Nations Economic Commission for Europe (UNECE) erarbeitete die *UN Task Force on Cyber Security and OTA Issues* einen Gesetzesvorschlag, der unter anderem ein Cybersecurity Management System (CSMS) als Voraussetzung für die europäische Typzulassung von Fahrzeugen vorsieht [Tas18]. Das CSMS soll dabei den gesamten Lebenszyklus abdecken, von der Entwicklung, über die Produktion bis zur Wartung. Zusätzlich existieren in der Europäischen Union (engl.: European Union, EU) Bestrebungen, ein Framework für die Cybersecurity-

Zertifizierung von Endverbraucher-Produkten - und damit auch Fahrzeugen - einzuführen. Dies ist Bestandteil des *EU Cybersecurity Act*, welchem Mitte Dezember 2018 zugestimmt wurde [Eur18].
Im Gegensatz zur Automobilindustrie haben sich in der klassischen Informationstechnik (IT) bereits Cybersecurity-Prozesse etabliert. Eine Einführung dazu ist in Anhang A.3.1 zu finden.
Vor beziehungsweise während der Prozess-Standardisierung wurden bereits einige konkrete Security-Mechanismen für Steuergeräte-Software definiert und umgesetzt. Im Hinblick auf die vorhandenen Möglichkeiten und die empfohlenen Umfänge steht die Automobilindustrie in diesem Bereich allerdings noch am Anfang. Ein implizit enthaltener Security-Mechanismus ist die statische Systemauslegung. Ohne Software-Update können zur Laufzeit beispielsweise keine neuen Software-Komponenten oder Betriebssystem-Tasks erzeugt werden. Auch die versendeten und empfangenen Kommunikationsnachrichten sind fest vorgegeben. Unbekannte Nachrichten verwirft der Kommunikationsstack automatisch. Zusätzlich hilft die Einschränkung auf statische Speicherverwaltung durch Programmierrichtlinien wie MISRA, siehe Abschnitt 2.1.5. Ergänzend dazu sind weitere Maßnahmen erforderlich, um einen umfangreichen Schutz zu gewährleisten.
Als Grundlage für die meisten Security-Mechanismen definiert AUTOSAR Classic einen Software-Stack, der kryptografische Funktionen bereitstellt, siehe Anhang A.3.2. Der einzige, aktuell standardisierte Anwendungsfall für diese Funktionen ist die im nächsten Abschnitt beschriebene Kommunikationsabsicherung. Weitere Security-Mechanismen realisieren Fahrzeughersteller und Zulieferer momentan als Teil der Applikationssoftware oder als Complex Driver. Beispiele hierfür sind das sichere Aufstarten (engl.: *Secure Boot*) und die sichere Software-Aktualisierung (engl.: *Secure (Software) Update*), wie in Anhang A.3 beschrieben. Außerdem wurden zusätzliche Anwendungsfälle im Rahmen von AUTOSAR Classic Version 4.4 standardisiert. Dazu zählen eine Schlüsselverwaltung sowie ein sicheres

Logging-Verfahren. Die Veröffentlichung des Standards erfolgte im Oktober 2018.

3.2.1 Secure Onboard Communication

Um die Fahrzeug-interne Kommunikation zwischen Steuergeräten gegen mutwillige Manipulation und Angriffe (siehe schwarzer Blitz in Bild 3.2) zu schützen, ist die in Abschnitt 3.1.1 vorgestellte E2E Protection nicht ausreichend. Daher wurde die sogenannte *Secure Onboard Communication* (SecOC) spezifiziert, welche Kommunikationsnachrichten durch das Hinzufügen eines *Message Authentication Codes* (MAC) authentifiziert. Der MAC berechnet sich aus den Applikationsdaten, einem eindeutigen Identifier (ID), einem geheimen Schlüssel und einem Zähler, welcher die Neuheit der Nachricht sicherstellt. Sollte die MAC-Überprüfung auf Empfängerseite fehlschlagen, wird die Nachricht verworfen und gegebenenfalls die Applikation informiert. SecOC arbeitet ressourcensparend und wurde für den Einsatz auf CAN-, FlexRay- und Ethernet-Netzwerken konzipiert. Die Verschlüsselung der Applikationsdaten ist nicht vorgesehen. Es ist davon auszugehen, dass für die meisten Anwendungsfälle eine reine Authentifizierung ausreicht. Die Spezifikation von SecOC erfolgte im Rahmen von AUTOSAR Classic, wobei der Mechanismus auch in andere Software-Architekturen integriert werden kann. AUTOSAR Classic definiert das gleichnamige Basissoftware-Modul auf Serviceebene [AUT17d]. Es ist in Bild 3.2 explizit dargestellt und übernimmt die Erweiterung der Nutzdaten um Zähler und MAC. Die Berechnung des MACs selbst erfolgt über den in Anhang A.3.2 beschriebenen Crypto-Stack. Dieser kann gleichzeitig auch die sichere Speicherung des dafür notwendigen, symmetrischen Schlüssels gewährleisten. Bild 3.2 zeigt die Ähnlichkeit der Funktionsweise von SecOC mit der E2E Protection. Unterschiedlich sind jedoch die übertragenen Zusatzinformationen sowie deren Berechnung.

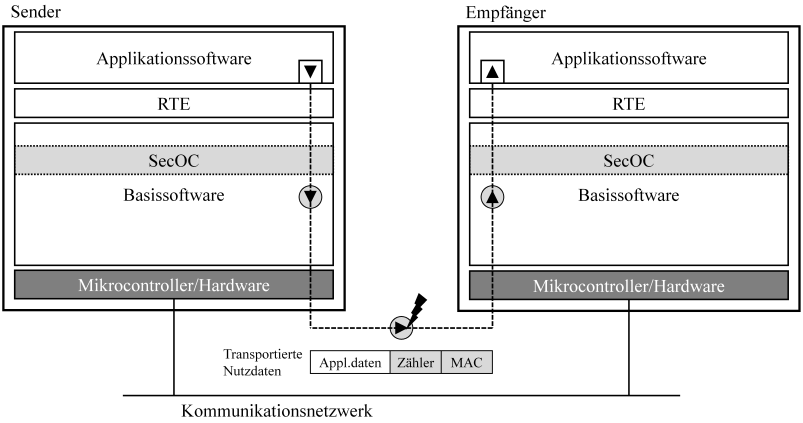

Bild 3.2: SecOC in einem AUTOSAR Classic-basierten Steuergerät

Für Ethernet-basierte Netzwerke gibt es weitere, in der IT-Welt etablierte Protokolle, die eine Authentifizierung und Verschlüsselung von Kommunikation erlauben. Prominente Beispiele hierfür sind IEEE 802.1AE (Media Access Control (MAC) Security, kurz: *MACsec*) [IEE06], Internet Protocol Security (*IPsec*) [Int05b], Transport Layer Security (*TLS*) [Int08] [Int18] und Datagram TLS (*DTLS*) [Int12b]. MACsec arbeitet paketorientiert auf OSI-Schicht zwei. Dabei werden zusätzliche Protokollfelder (engl.: tag) im Ethernet-Frame nach Ziel- und Quell-MAC-Adresse eingefügt. Sollte die Hardware Quality-of-Service (QoS) Mechanismen auf Basis von IEEE 802.1Q unterstützen [IEE18], muss diese gegebenenfalls angepasst werden, um auch MACsec authentifizierte oder verschlüsselte Pakete entsprechend verarbeiten zu können. Auf OSI-Schicht drei arbeitet IPsec, das initial für das Internet Protocol Version 6 (IPv6) spezifiziert und später für IPv4 portiert wurde. Erste Fahrzeughersteller setzen auf IPsec zur Authentifizierung von ausgesuchter Ethernet-Kommunikation. Vorteilhaft bei IPsec ist, dass die Absicherung unabhängig vom Transportprotokoll stattfindet. Im Gegensatz dazu arbeitet TLS ausschließlich über dem Transmission Control

Protocol (TCP) [Int81]. TLS ist im automotive Umfeld bereits im Einsatz, beispielsweise für die sichere Kommunikation von Hybrid- und Elektrofahrzeugen mit Ladensäulen nach ISO/IEC 15118 [Int14]. Zusätzlich sind erste Anwendungen von TLS für eine sichere Fahrzeugdiagnose in Diskussion. DTLS stellt das Pendant von TLS für das User Datagram Protocol (UDP) [Int80] dar, wobei noch kein Anwendungsfall in Fahrzeugen bekannt ist.

3.2.2 Firewalls und Intrusion Detection/Prevention Systeme

Im klassischen IT-Umfeld kommen zur Absicherung von Netzwerken und Kommunikationsteilnehmern Firewalls und Intrusion Detection/Prevention Systeme erfolgreich zum Einsatz. Eine Firewall ist ein spezielles Programm, das den Empfang und den Versand unerwünschter Kommunikationsnachrichten unterbindet, und ist mittlerweile fester Bestandteil der gängigen IT-Betriebssysteme. Es verhindert damit unbefugte Zugriffe auf lokale Rechner und Infrastruktur über ein ungesichertes Netzwerk, wie beispielsweise das Internet.

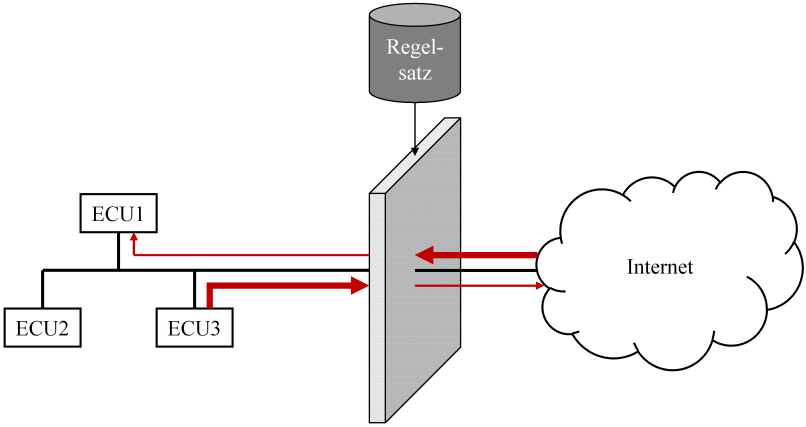

Bild 3.3: Prinzipielle Funktionsweise einer Firewall

Da eine Firewall auch ausgehende Nachrichten blockieren kann, stellt sie einen Empfangs- und Sendefilter dar, wie Bild 3.3 verdeutlicht. Die Dicke der Pfeile kennzeichnet dabei die Menge der ankommenden und abgehenden Nachrichten. Welche Pakete die Firewall passieren und welche blockiert beziehungsweise verworfen werden, ist in einem Regelsatz festzulegen.
Eine mögliche Kategorisierung bezieht sich auf die zur Filterung verwendeten Informationen. Im einfachsten Fall betrachtet eine Firewall die Informationen eines einzelnen Pakets unabhängig von vorheriger Kommunikation. Ist dies der Fall, spricht man von einem *Paketfilter*. Sobald mehrere Pakete die Filterentscheidung beeinflussen, handelt es sich um einen *statusbehafteten Paketfilter* (engl.: Stateful Packet Inspection (SPI)). Dieser kann beispielsweise in einem Request/Response Protokoll ein valides Antwortpaket aufgrund der fehlenden Anfrage blockieren. Ein normaler Paketfilter ohne Wissen über den Zusammenhang von Request und Response würde die valide Antwort passieren lassen. Beide Filtertechniken verwenden ausschließlich Protokollinformationen in den dazugehörigen Regelsätzen und arbeiten normalerweise auf den OSI-Schichten zwei bis vier. Im Falle von Standard CAN-Botschaften können sich die Regeln also auf Informationen aus dem Arbitrierungs- und Kontrollfeld beziehen, nicht aber auf die Applikationswerte im Datenfeld. Botschaften mit ungültiger Checksumme verwirft bereits die CAN-Hardware. Die dritte Kategorie Firewall implementiert eine sogenannte *Deep Packet Inspection* (DPI). Damit ist sie in der Lage auch Applikationsdaten auszuwerten und basierend darauf ein Paket gegebenenfalls zu verwerfen.
Bei der Definition von Firewall-Regeln gibt es den sogenannten *Black-List* und *White-List* Ansatz. Eine Black-List besteht aus Regeln, bei deren Erfüllung das entsprechende Paket zu verwerfen ist. Konträr dazu enthält eine White-List Regeln, welche Pakete explizit akzeptieren. Beide Ansätze können in einer konkreten Implementierung auch kombiniert werden. Zusätzlich sind weitere Aktionen bei der Erfüllung einer Regel möglich, wie beispielsweise das Aufrufen einer speziellen Applikation.

Mit ihrer Eigenschaft aktiv in das Kommunikationsverhalten eingreifen zu können (d.h. Pakete zu blockieren) gehören Firewalls zu den Intrusion Prevention Systemen (IPS). Im Gegensatz dazu beschränken sich Intrusion Detection Systeme (IDS) auf das Erkennen und Protokollieren von unerwünschter Kommunikation oder auffälligem Systemverhalten. Die Entscheidung über mögliche Reaktionen oder Gegenmaßnahmen wird nachgelagert getroffen, beispielsweise durch einen Sicherheitsbeauftragten, der die Protokolle entsprechend zu interpretieren hat. Implementierungsseitig gibt es zwei unterschiedliche Ansätze:

Signatur-basierte IDS verwenden einen Katalog von bekannten Angriffen und Patterns, um unerwünschtes Systemverhalten zu identifizieren. Dies bietet den Vorteil einer sehr geringen Falschalarmrate. Sollte ein bekanntes Pattern an Nachrichten auftauchen, handelt es sich mit großer Wahrscheinlichkeit um einen Angriff. Auf der anderen Seite entstehen immer neue Attacken und bestehende entwickeln sich weiter. Selbst bei regelmäßiger Aktualisierung erkennen Signatur-basierte IDS nur bekannte und im Katalog vorhandene Angriffe. Das Vorgehen entspricht dem oben angesprochenen Black-List Ansatz.

Eine gegensätzliche Idee verfolgen *Anomalie-basierte IDS*. Diese Systeme benötigen im Vorfeld eine möglichst genaue Definition des Normalverhaltens. Im laufenden Betrieb wird jegliche Abweichung davon als möglicher Angriff eingestuft, vergleichbar zu einer White-List. Damit können Anomalie-basierte IDS auch im Voraus nicht bekannte Attacken erkennen. Ist die Definition des Normalverhaltens jedoch nicht möglich oder fehlerhaft, beispielsweise aufgrund einer sehr hohen Dynamik im zu überwachenden System, kann es zu hohen Falschalarmraten kommen. Viele Anomalie-basierte Systeme nutzen Algorithmen des maschinellen Lernens, um das Normalverhalten aus den Trainingsdaten abzuleiten und anschließend dagegen zu prüfen.

Beziehen sich IDS und IPS auf die Überwachung der Kommunikation, wie in diesem Abschnitt beschrieben, spricht man von *Netzwerk-basierten*

Systemen (NIDS/NIPS). Daneben existieren auch *Host-basierte* IDS/IPS (HIDS/HIPS), welche einen Computer (Host) überwachen. Diese Systeme verwenden zur Erkennung von Unregelmäßigkeiten oftmals Daten der Benutzerschnittstelle, wie beispielsweise die Anzahl an fehlgeschlagenen Anmeldeversuchen. Darüber hinaus kann das Betriebssystem wertvolle Informationen liefern. Dazu zählen unter anderem die Aufrufsequenz und Aufrufhäufigkeit von speziellen Funktionen (z.B. System Calls) sowie der Ressourcenbedarf von Applikationen.
Alle möglichen Kombinationen aus Überwachungsfokus (Netzwerk/Host) und Funktionsweise (Signatur/Anomalie) finden in IT-Systemen bereits Anwendung. In Fahrzeugnetzwerken und auf Steuergeräten sind Firewalls und IDS/IPS aktuell noch die Ausnahme. Da sich moderne automotive E/E-Architekturen mit der Verwendung von Ethernet, den offenen Schnittstellen zur Außenwelt und dem Einsatz von höheren Betriebssystemen wie Linux den klassischen IT-Systemen annähern, rücken diese Absicherungssysteme zunehmend in den Fokus der Automobilindustrie. Verschiedene Wissenschaftler und Unternehmen untersuchen und empfehlen Netzwerk-basierte IDS für beziehungsweise in Fahrzeugen [HKD11] [Stu+13] [MJ14] [vB16]. Auch in der Standardisierung von Steuergeräte-Software spielten Firewalls und IDS/IPS bisher eine untergeordnete Rolle. Das Thema gewinnt allerdings stets an Bedeutung und für das kommende AUTOSAR Classic Release 4.5 wird an einer entsprechenden Spezifikation gearbeitet.
Unabhängig von der Standardisierung bieten unterschiedliche Unternehmen proprietäre Firewall- und IDS/IPS-Lösungen an [Arg] [esc] [Oku18] [Gua] [Saf18], welche in der Regel aus zwei Teilen bestehen:

1. Ein Anomalie-basierter, Fahrzeug-interner Teil, der Auffälligkeiten zur Laufzeit erkennt, loggt und an ein Backend-System meldet. Im Großteil der aktuell verfügbaren Lösungen arbeitet dieser Teil auf Basis von festzulegenden Regeln und ist auf den Netzwerk-Anteil beschränkt. Die hierfür notwendigen Regeln werden (teilweise) aus vor-

handenen Spezifikationsdokumenten abgeleitet. Erste Unternehmen entwickeln Überwachungssysteme, die auf lernenden Algorithmen in Steuergeräten beruhen und dabei auch Host-basierte Anteile betrachten [Aur18] [cog18] [Saf18]. Manche davon fokussieren zwar nicht explizit den Anwendungsfall der Intrusion Detection, können aber gegebenenfalls einen wertvollen Beitrag dazu leisten.

2. Ein Backend-System, das die gemeldeten Anomalien der gesamten Fahrzeugflotte analysiert, darstellt und gegebenenfalls eine Aktualisierung der Sicherheitseinstellungen/-mechanismen vornehmen kann (via OTA/SOTA). Für die Analyse kommen unter anderem Big-Data Verfahren und maschinelles Lernen zum Einsatz.

Einen etwas tieferen Einblick geben Patente und Patentanmeldungen von entsprechenden Unternehmen, wie zum Beispiel [GBL16], [BGL15], [All+18] und [Moe+14]. Detaillierte Informationen zu den Fahrzeug- und Backend-seitig eingesetzten Methoden und Algorithmen sind jedoch nur in eingeschränktem Umfang öffentlich verfügbar, da es sich hierbei um Kernkompetenzen der jeweiligen Unternehmen handelt.

Aktueller Stand der Forschung

Aufgrund der hohen Relevanz beschäftigen sich seit einigen Jahren viele Forschungsprojekte mit automotive Cybersecurity. Einen Überblick gibt der SAE J3061 Standard in Anhang H [SAE]. Dort sind die relevanten Projekte zwischen 2004 und der Veröffentlichung des Standards (2016) aufgelistet, ohne Anspruch auf Vollständigkeit. Ein Schwerpunkt war und ist die Erforschung der Security im Rahmen von Vehicle-to-Vehicle (V2V) und Vehicle-to-Infrastructure (V2I) Kommunikation, da durch diese Technologien das Fahrzeug eine zusätzliche sicherheitskritische Luftschnittstelle erhält. In Bild 1.1 sind V2V und V2I unter dem Punkt Dedicated Short Range Communication (DSRC) zusammengefasst. Des Weiteren beschäftigen sich

viele Konsortien mit der Definition eines automotive Cybersecurity Engineering Prozesses oder einzelnen Bestandteilen daraus, wie beispielsweise dem Testen von Security. Hardware-Erweiterungen und sichere Ausführungsumgebungen basierend auf Virtualisierung und der Isolation von einzelnen Prozessen oder Applikationen sind weitere Forschungsschwerpunkte. Ein Projekt speziell für automotive Firewalling und IDS/IPS ist zum aktuellen Zeitpunkt keines bekannt.

Das *Secure Vehicle Communication* (SEVECOM, 2006-2010) Projekt sieht Fahrzeug-interne Kommunikation als einen Bestandteil sicherer V2V und V2I Anwendungen [06]. Daher wurde Fahrzeug-interne Intrusion Detection berücksichtigt aber als Randthema nicht vollständig adressiert. Im Rahmen von *Sicherheit in Eingebetteten IP-basierten Systemen* (SEIS, 2009-2012) untersuchten die Teilnehmer unter anderem eine Sicherheits-Middleware, wobei hier der Fokus eher auf der Middleware selbst als auf den einzelnen potentiell enthaltenen Mechanismen - wie zum Beispiel einem IDS - lag [Gla+10]. Im *Security and Safety Modelling* (SESAMO, 2012-2015) Projekt wurden Bausteine für die Modellierung von Safety und Security in eingebetteten Systemen definiert [SES13]. Darunter finden sich Access control/Traffic filtering, Run-Time Monitoring, Plausibility Checks, Logging, Security Audit und Information Flow Control. Ersterer umfasst das Firewall-Konzept. Alle weiteren sind potentielle Bausteine eines IDS/IPS.

Im Rahmen der angesprochenen Forschungsprojekte als auch unabhängig davon, entstanden in den vergangenen Jahren zahlreiche wissenschaftliche Veröffentlichungen zum Thema Angriffserkennung und Angriffssicherheit für Fahrzeug-interne Netzwerke. Auf Publikationen mit großer Relevanz für die vorliegende Arbeit wird im Folgenden näher eingegangen.

2008 diskutierten Larson et al. einen Ansatz zur spezifikationsbasierten Angriffserkennung in Fahrzeug-internen Netzwerken anhand von CAN und

CANopen[1] [LNJ08]. Dabei handelt es sich um ein Anomalie-basiertes IDS, für welches das Normalverhalten anhand vorliegender Spezifikationen definiert wird. Ein Fahrzeugnetzwerk ist eine sehr statische Überwachungsumgebung, verglichen mit der klassischen IT-Welt. Steuergeräte, deren Kommunikationsverbindungen untereinander sowie die berechneten Funktionen werden in der Entwicklungsphase festgelegt und ändern sich während des Betriebs nicht oder nur in geringem Umfang. Zudem sind nicht nur die Kommunikationsprotokolle, sondern auch die ausgetauschten Nachrichten vom Fahrzeughersteller fest vorgegeben. Obwohl Larson et al. sich unter anderem auf CANopen stützen, welches in der Praxis für automotive Netzwerke keine Rolle spielt, sprechen die statische E/E-Architektur und die Definition der ausgetauschten Nachrichten durch den Fahrzeughersteller (nicht durch CANopen sondern durch die Kommunikationsmatrix) für den Einsatz eines Anomalie-basierten IDS. Zudem erwähnen die Autoren am Ende ihrer Veröffentlichung, dass ein rein spezifikationsbasiertes System nicht alle Attacken erkennt, sondern dass zusätzlich erweiterte Anomalieerkennungsmechanismen notwendig sind, wie beispielsweise das Erstellen und Auswerten statistischer Profile.

Einen Schritt weiter gehen 2010 Müter et al. [MGF10] in ihrer Veröffentlichung „A Structured Approach to Anomaly Detection for In-Vehicle Networks“. Darin diskutieren sie spezifische Herausforderungen bei der Realisierung eines automotive IDS und argumentieren dabei ebenfalls für den Einsatz von Anomalie-basierten Systemen. Zudem wird kurz auf die verschiedenen Datenquellen, die Positionierung eines IDS innerhalb der E/E-Architektur, die benötigte Echtzeitfähigkeit und die Herausforderung einer geeigneten Reaktion bei erkanntem Fehlverhalten eingegangen. Im weiteren Verlauf fokussieren Müter et al. auf ein Netzwerk-basiertes IDS für das sie sogenannte Anomalieerkennungssensoren definieren. Jeder Sensor

[1] CANopen ist seit 2002 eine europäische Norm [DIN02] und spezifiziert sowohl Kommunikationsmechanismen als auch Gerätefunktionalitäten für den Datenaustausch über einen CAN-Bus. Anwendung findet der Standard heute hauptsächlich in der Industrieautomation.

überwacht einen spezifischen Aspekt der Fahrzeug-internen Kommunikation und basiert auf eindeutigen Informationen, sodass keine Falschalarme resultieren. Die dafür notwendigen Informationsquellen sind bereits vorhandene Dokumente wie die Kommunikationsmatrix oder zusätzliche Spezifikationen. So ist sichergestellt, dass alle erkannten Anomalien auch tatsächlich ein Fehlverhalten darstellen. Allerdings gibt es auch mit diesem Ansatz keine Garantie, dass ein System alle Anomalien erkennt. Nachfolgend wird auf die acht, von Müter et al. definierten Anomalieerkennungssensoren, näher eingegangen [MGF10]. Im Gegensatz zur originalen Veröffentlichung werden die einzelnen Sensoren in dieser Arbeit mit N-1 bis N-8 abgekürzt („N“ steht für Netzwerk).

- **Formality Sensor (N-1)**: Dieser Sensor überwacht die formale Korrektheit von Kommunikationsnachrichten. Dazu zählt beispielsweise das Einhalten der festgelegten Längen einzelner Header- und Datenfelder sowie eine korrekt berechnete Checksumme. Grundlage ist die CAN-Protokollspezifikation, welche unter anderem das Botschaftsformat (CAN-Frame) auf OSI-Schicht zwei definiert (siehe Abschnitt 2.1.4). Die Überprüfung der CRC Checksumme realisieren moderne CAN-Controller bereits in Hardware. Korrupte Botschaften werden dabei direkt verworfen.

- **Location Sensor (N-2)**: Die Kommunikationsmatrix legt unter anderem fest, welche Nachrichten in einem Bussystem oder Netzwerk erlaubt sind. Im Falle von CAN ist damit klar geregelt, welche IDs auftreten dürfen. Der Location Sensor erkennt das Auftreten nicht erlaubter Nachrichten, obwohl diese formal korrekt sein können.

- **Range Sensor (N-3)**: Zu jedem übertragenen Signal, wie beispielsweise der Fahrzeuggeschwindigkeit, ist normalerweise der gültige Wertebereich in der Kommunikationsmatrix vorgegeben. Wird zum Beispiel die aktuelle Motordrehzahl in einem 16 Bit unsigned Integer

übertragen, ist zusätzlich festgelegt, dass ausschließlich Werte im Bereich zwischen null und 6000 Umdrehungen pro Minute vorkommen dürfen. Das Einhalten dieser Randbedingungen überprüft ein Range Sensor.

- **Frequency Sensor (N-4)**: Viele Nachrichten werden in automotive Netzwerken zyklisch übertragen, unter anderem aus Sicherheitsgründen. Die Zykluszeiten sind in der Kommunikationsmatrix vorgegeben und daher Gegenstand der Überwachung mittels Frequency Sensoren. CAN ist jedoch ein ereignisgesteuertes Bussystem ohne feste Zeit-Slots[2]. Dadurch lassen sich Botschaften auch spontan versenden, beispielsweise beim Betätigen eines Schalters. Um sporadische Kommunikation ebenfalls zeitlich überwachen zu können, sind zusätzliche Informationen notwendig. Eine Möglichkeit besteht darin, die Kommunikationsmatrix um entsprechende Attribute zu erweitern. Voraussetzung wäre eine Anpassung des Standards (DBC/AUTOSAR). Zudem ist es selbst für Fahrzeughersteller herausfordernd zum Beispiel eine maximale und minimale Frequenz für Ereignis-basierte Botschaften zu definieren.

- **Correlation Sensor (N-5)**: Die E/E-Architektur von modernen Fahrzeugen enthält mehrere (Sub-)Netzwerke. Gateways sorgen dafür, dass wichtige Daten überall zur Verfügung stehen, indem sie Nachrichten entsprechend weiterleiten. Sollten sich hierbei Unregelmäßigkeiten zeigen, da ein Angreifer beispielsweise Nachrichten im Zielnetzwerk einspeist - nicht aber im eigentlichen Quellnetzwerk, erkennt dies der Correlation Sensor. Grundlage der Überwachung ist wiederum die Kommunikationsmatrix, sofern darin alle beteiligten Bussysteme beschrieben sind.

[2] Die Erweiterung Time-Triggered CAN (TTCAN) wird nicht betrachtet, da diese bei Fahrzeug-internen Netzwerken keine Anwendung findet.

- **Protocol Sensor (N-6)**: Vergleichbar mit dem Formality Sensor detektiert der Protocol Sensor Fehlverhalten in Protokollen, dieses Mal allerdings oberhalb von OSI Schicht zwei. Die dafür notwendigen Informationen sind nicht in der CAN-Spezifikation enthalten. Für das standardisierte CAN-TP können die dazugehörigen Spezifikation und die Kommunikationsmatrix als Überwachungsgrundlage dienen. Erstere definiert die Botschaftsformate sowie das erwartete Verhalten und zweitere legt die Botschaften fest, welche die Transportprotokoll-Kommunikation realisieren. Hauptanwendungsfall ist hierbei die Fahrzeugdiagnose. Des Weiteren gibt es Hersteller-spezifische Protokolle, wie beispielsweise einfache Request/Response Verfahren. Die daran beteiligten Botschaften sowie das erwartete Verhalten sind jedoch nicht in der standardisierten Kommunikationsmatrix vermerkt. Sowohl bei der Fahrzeugdiagnose als auch bei proprietären Protokollen kann eine formal korrekte Response einen Angriff beziehungsweise ein Fehlverhalten darstellen, wenn zuvor kein passender Request abgesetzt wurde.

- **Plausibility Sensor (N-7)**: Viele Kommunikationssignale unterliegen Randbedingungen, welche deren zeitlichen Verlauf einschränken. So ist es physikalisch unmöglich, dass sich die Fahrzeuggeschwindigkeit innerhalb von 100 ms um 100 km/h verändert. Die Überwachung dieser Randbedingungen ist Aufgabe der Plausibility Sensoren, wobei keine Informationen zum gültigen Signalverlauf in der Kommunikationsmatrix enthalten sind. Eine Anreicherung mit entsprechenden Werten ist theoretisch zwar möglich, stößt aber auf die gleichen Herausforderungen wie beim Frequency Sensor für sporadische Botschaften beschrieben. Hinzu kommt, dass Fahrzeughersteller ein Signal oftmals baureihenübergreifend definieren, die validen Verläufe aber unterschiedlich ausfallen können. Ein bestimmter Verlauf des Geschwindigkeitssignals kann für ein Fahrzeug mit 500 PS valide sein

während er bei einer Motorleistung von 70 PS physikalisch unmöglich ist.

- **Consistency Sensor (N-8)**: In modernen Fahrzeugen tauschen Steuergeräte tausende Signale untereinander aus. Zwischen manchen davon existiert eine Korrelation auf logischer Ebene, wie beispielsweise zwischen Fahrzeuggeschwindigkeit, Motordrehzahl, eingelegtem Gang und Kupplungsstatus. Sollte sich die Geschwindigkeit unabhängig von Motordrehzahl und Gang bei geschlossener Kupplung ändern, liegt eine Anomalie vor. Die Erkennung solcher Abweichungen in der Korrelation mehrerer Datenquellen erfolgt mittels Consistency Sensoren. Die standardisierte Kommunikationsmatrix enthält diesbezüglich keine Informationen.

Criterion / **Detection Sensor**	Specification-Based	Number of Messages	Number of Bus Systems	Different Message Types	Payload-Inspection	Semantic-Based
Formality	*true*	1	1	*n.a.*	*false*	*false*
Location	*true*	1	1	*n.a.*	*false*	*false*
Range	*true*	1	1	*n.a.*	*true*	*false*
Frequency	***false***	*n*	1	*false*	*false*	*false*
Correlation	*true*	*n*	*n*	*true*	*false*	*false*
Protocol	***false***	*n*	**1**	*true*	*false*	*false*
Plausibility	*false*	*n*	1	*false*	*true*	*true*
Consistency	*false*	*n*	*n*	*true*	*true*	*true*

Tabelle 3.1: Klassifizierung der Anomalieerkennungssensoren von Müter et al. [MGF10] mit drei abweichenden Bewertungen (fett)

Nach der Definition der Anomalieerkennungssensoren klassifizieren Müter et al. diese anhand verschiedener Kriterien. Tabelle 3.1 ist ihrer Veröf-

fentlichung entnommen [MGF10] und zeigt die entsprechende Einordnung. Aufgrund der obigen Diskussion der einzelnen Sensoren, ergeben sich für die vorliegende Arbeit drei abweichende Bewertungen. Diese sind in Tabelle 3.1 bereits angepasst und fett dargestellt.

Das Kriterium *Specification-Based* zeigt an, ob die Überwachung rein auf Basis von Spezifikationen erfolgt. Müter et al. verweisen für CAN hauptsächlich auf die Kommunikationsmatrix, wobei auch andere Spezifikationen als Informationsquelle dienen können. Im Rahmen dieser Arbeit werden nur standardisierte Spezifikationsdokumente und -formate betrachtet. Daher ergibt sich für die Frequency und Protocol Sensoren eine abweichende Bewertung. Die Kriterien *Number of Messages* und *Number of Bus Systems* zeigen an, ob eine Nachricht zur Anomalieerkennung ausreichend ist beziehungsweise ob Nachrichten von unterschiedlichen (Sub-)Netzwerken zu untersuchen sind. Der Protocol Sensor benötigt nach obiger Betrachtung nur Zugriff auf ein Netzwerk statt auf mehrere, wie von Müter et al. ursprünglich definiert. *Different Message Types* bezieht sich auf die Verwendung unterschiedlicher Nachrichten zur Anomalieerkennung. Bei CAN entspricht *true* der Untersuchung von Botschaften mit verschiedenen CAN-IDs. Das Kriterium *Payload Inspection* zeigt an, ob die Nutzdaten einer Nachricht vom entsprechenden Sensor untersucht werden. Ist *Semantic-Based* ebenfalls *true*, ist zudem Wissen über den semantischen Inhalt der Nutzdaten notwendig.

Am Ende ihrer Veröffentlichung präsentieren Müter et al. noch eine Möglichkeit die Kritikalität von erkannten Anomalien zu bewerten, basierend auf der Gewichtung von einzelnen Sensoren und der Betrachtung der Überwachungsergebnisse über einen gewissen Zeitraum.

In ihrer Masterarbeit stellen Salman und Bresch 2017 ein IDS für CAN-basierte Fahrzeug-interne Netzwerke vor [SB17]. Sie referenzieren unter anderem auf Müter et al. [MGF10] und orientieren sich im weiteren Verlauf der Arbeit an den definierten Anomalieerkennungssensoren. Im Gegensatz zur vorliegenden Arbeit, unterscheiden Salman und Bresch den spezifika-

tionsbasierten Ansatz und Anomalieerkennungsmechanismen. Für ersteren werden die erforderlichen Regeln aus der Kommunikationsmatrix abgeleitet und die entstehende White-List ordnen die Autoren der Signatur-basierten Erkennung zu.

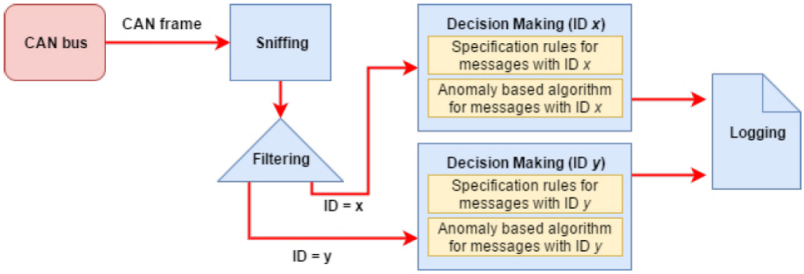

Bild 3.4: IDS-Architektur von Salman und Bresch [SB17]

In ihrem rein Software-basierten System kommen beide Verfahren parallel zum Einsatz. Die vorgeschlagene IDS-Architektur ist in Bild 3.4 dargestellt. Da die Erkennung jeweils spezifisch für eine CAN-ID arbeitet, ist es damit nicht möglich Sensoren zu realisieren, welche unterschiedliche Botschaften beobachten. Dies betrifft konkret Correlation, Protocol und Consistency Sensoren.

Salman und Bresch implementieren Formality, Location und Range Sensoren nach dem spezifikationsbasierten Ansatz. Zusätzlich schlagen sie vor, Formality sowie Range Sensoren auf einem Gateway zu integrieren und für Location Sensoren einen separaten Netzwerkknoten zu verwenden. Frequency und Plausibility Sensoren sind mittels Anomalieerkennung realisiert, wobei auf deren Verortung nicht näher eingegangen wird. Die Autoren argumentieren für die Anomalie-basierte Erkennung von Frequenzabweichungen bei periodischen Botschaften, obwohl die Zykluszeit in der Kommunikationsmatrix vorgegeben ist. Untersuchungen zeigten, dass es hier auch im Normalbetrieb zu erheblichen Schwankungen kommen kann. Die vorgeschlagene Lösung ist ein Entprellmechanismus, der erst einen

Alarm generiert, wenn mehrere aufeinander folgende Botschaften eine Abweichung in der Zykluszeit aufweisen. Der implementierte Plausibility Sensor überwacht die Fahrzeuggeschwindigkeit und arbeitet mit einer maximal zulässigen Geschwindigkeitsänderung zwischen zwei Botschaften.
Die bisher vorgestellten Arbeiten fokussieren sich hauptsächlich auf die Strukturierung und das Design von automotive Intrusion Detection Systemen. Parallel dazu erforschen Wissenschaft und Industrie die Anwendung von selbstlernenden Algorithmen zur Anomalieerkennung in CAN-Kommunikation als Teil eines entsprechenden IDS. Taylor et al. [TLJ16] trainieren ein LSTM-Netz je CAN-ID und nutzen das Bitmuster der Nutzdaten als Eingangsvektor. Der Ausgangsvektor entspricht der Vorhersage der Nutzdaten für die nächste CAN-Botschaft mit gleicher ID. Basierend auf dem Unterschied zwischen den prognostizierten und den tatsächlich empfangenen Nutzdaten wird die Anomaliebewertung vorgenommen. Dieser Ansatz ist generisch und erfordert kein Wissen über die Zusammensetzung der Nutzdaten. Die Autoren evaluieren ihren Ansatz anhand von fünf erzeugten Anomalie-Szenarien. Bei 0% FPR erreichen sie im Durchschnitt (über alle Anomalie-Szenarien und über alle untersuchten CAN-IDs) eine TPR von 44,5%. Passen sie den Schwellwert so an, dass 100% der Anomalien als solche erkannt werden, steigt die FPR auf durchschnittlich 12,8%.
Die Interpretation von Nutzdaten spielt auch bei Batmaz und Pete eine Rolle [BP]. Ihr vorgestelltes Anomalieerkennungssystem für CAN enthält einen Vorverarbeitungsblock, welcher eine Selektion basierend auf der CAN-ID vornimmt und den Inhalt extrahiert. Nachgelagert folgen zwei LSTM-Netze. Das erste erzeugt eine spezifische Fehlerbewertung auf Basis derer das zweite Netz die Wahrscheinlichkeit eines Angriffs berechnet. Die Autoren erreichen damit eine TPR von 87% und eine FPR von 6%.
Suda et al. verwenden neben den Nutzdaten-Bytes auch die CAN-ID und den Abstand zwischen zwei Botschaften mit gleicher ID als Eingangsdaten für ein LSTM-Netz [SNH18]. Damit ist ihr Ansatz in der Lage sowohl Botschafts- als auch Signal-basierte Angriffe zu erkennen. Allerdings ver-

wenden Suda et al. überwachtes Lernen und benötigen daher auch anormale Trainingsdaten. Des Weiteren enthält ihre Veröffentlichung keine detaillierte Leistungsevaluierung für alle betrachteten Angriffe (Manipulation von CAN-ID und Nutzdaten sowie das Fluten des Busses, siehe auch Abschnitt 3.2.3).

Regulierung

In den U.S. wurde im Juli 2015 ein initialer Vorschlag für eine Gesetzesänderung vorgebracht, die unter anderem den Schutz vor Hackerangriffen, den Schutz gesammelter Fahrzeugdaten und das Erkennen, Melden und Reagieren auf Hackerangriffe vorschreibt - der sogenannte *Security and Privacy in Your (SPY) Car Act of 2015* [Uni15]. Ein überarbeiteter Entwurf liegt seit März 2017 vor [Uni17a]. Sollte diese Gesetzesänderung verabschiedet werden, müssen spätestens zwei Jahre nach Inkrafttreten nahezu alle, für den U.S. Markt bestimmten Fahrzeuge, die erwähnten Mechanismen enthalten, darunter auch ein IDS/IPS. Parallel dazu liegt dem Repräsentantenhaus ein Gesetzesentwurf vor, der die Erforschung und Untersuchung der dafür notwendigen Technologien zum Ziel hat [Uni17b]. Die Leitung des Vorhabens soll laut Entwurf bei der NHTSA liegen.
Im Anhang von SAE J3061 [SAE16] sind mögliche Security-Maßnahmen aufgelistet. Im Gegensatz zum Gesetzesentwurf ist die SAE bezüglich Machbarkeit und Akzeptanz von IDS/IPS in Fahrzeugen eher skeptisch eingestellt und sieht noch Forschungsbedarf. Die vorliegende Arbeit untersucht Machbarkeit und Wirksamkeit von IDS in automotive Steuergeräten und soll somit zur Akzeptanz der Technologie auf Seiten der Standardisierung beitragen.
Der Gesetzesvorschlag der UN Task Force on Cyber Security and OTA Issues fordert das Vorhandensein eines CSMS für die europäische Typzulassung von Fahrzeugen, wie zu Beginn des Unterkapitels beschrieben. Dieses muss einen Prozess für das Überwachen und das Erkennen von Angriffen sowie

für die nachgelagerte Reaktion enthalten, siehe Annex A in [Tas18]. Ein Ansatz zur Erfüllung der ersten beiden Punkte ist die Verwendung eines Fahrzeug-seitigen IDS.

3.2.3 Kommunikationsbasierte Cyber-Attacken

Das aktuelle Kapitel behandelte bisher Safety- und Security-Maßnahmen für automotive Steuergeräte. Im Hinblick auf Security ist zusätzlich die Betrachtung von Angriffen notwendig. Das resultierende Angreifermodell hilft bei der späteren Einordnung und Auswahl von Gegenmaßnahmen. Nachfolgend sind daher die möglichen Angriffe auf die Fahrzeug-interne Kommunikation am Beispiel von CAN dargestellt. In Bezug auf die unterschiedlichen Stufen einer Attacke und der damit verbundenen Gegenmaßnahmen (vgl. Bild 1.3), wird davon ausgegangen, dass ein Angreifer Zugriff auf ein Fahrzeug-internes Netzwerk erlangt hat. Entweder er besitzt physischen Zugang oder er hat die ersten beiden Barrieren der vorgestellten Defense-in-Depth-Strategie bereits überwunden, siehe Unterkapitel 1.1.

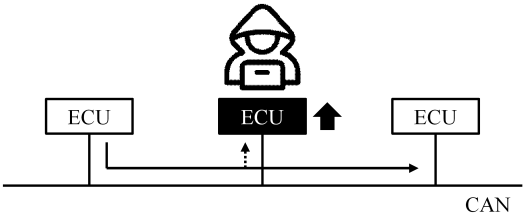

Bild 3.5: Eavesdropping-Attacke[3]

Da es sich bei einem CAN-Bus um ein Broadcast-Medium handelt, kann ein zusätzlich installierter oder ein kompromittierter Busteilnehmer alle Botschaften empfangen und sich damit unberechtigten Zugriff auf Infor-

[3] Hacker-Icon von Freepik (www.freepik.com/www.flaticon.com)

mationen verschaffen. Bild 3.5 zeigt diese einfache Form einer Attacke, welche auch als *Eavesdropping* bezeichnet wird.

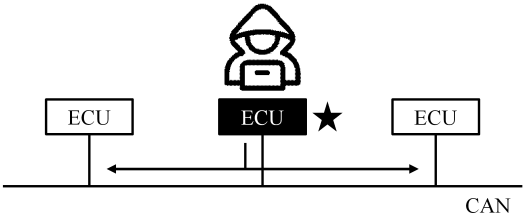

Bild 3.6: Spoofing-Attacke[3]

In einer nächsten Stufe, kann ein kompromittiertes Steuergerät nicht nur Botschaften mitlesen sondern diese auch willkürlich erzeugen und versenden (*Fabrication Attack* [CS16]), siehe Bild 3.6. Da bei CAN auf OSI-Schicht zwei keine Authentifizierung stattfindet, kann ein Empfänger zunächst nicht feststellen, ob eine Botschaft vom legitimen Sender stammt. Bei ansonsten gültigen CAN-IDs handelt es sich damit gleichzeitig um *Spoofing*. Eine potentielle Gegenmaßnahme hierfür ist die in Abschnitt 3.2.1 vorgestellte SecOC, welche auf einer höheren Software-Schicht arbeitet.
Bei gleichzeitigem Lese- und Schreibzugriff sind außerdem *Replay*-Attacken möglich [Ji+17]. Dabei zeichnet ein Angreifer gültige Botschaften auf und versendet diese zu einem späteren Zeitpunkt nochmals. Abhilfe schafft ein entsprechender Sequenzzähler (wie zum Beispiel in SecOC).
Hat ein Kommunikationsteilnehmer schreibenden Zugriff auf den Bus, lässt sich ohne großen Aufwand und bedingt durch die Arbeitsweise von CAN ein *Denial-of-Service* realisieren. Bei der entsprechenden *Flooding*-Attacke erzeugt ein Teilnehmer hochpriore CAN-Botschaften (zum Beispiel mit ID 0) in einer so hohen Frequenz, dass der Bus vollständig ausgelastet ist.

Etwas aufwändiger ist die Realisierung eines selektiven Denial-of-Service, welcher sich nur über bestimmte Botschaften oder Steuergeräte erstreckt. Maggi demonstrierte 2017, dass durch das selektive Kippen von Bits inner-

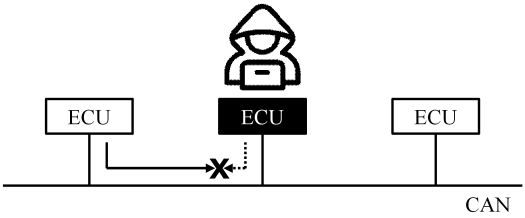

Bild 3.7: Selektiver Denial-of-Service/Drop-Attacke[3]

halb von CAN-Botschaften einzelne Steuergeräte selektiv vom Bus getrennt werden können [Mag17]. Dabei missbraucht er den Fehlerzähler, welcher das Fluten des Busses durch fehlerhafte Teilnehmer verhindern soll, siehe Abschnitt 2.1.4. Um einzelne Botschaften zu verwerfen (*Drop*-Attacke), genügt das Unterbrechen beziehungsweise Überschreiben einer Übertragung mit einem Error-Frame, wie in Bild 3.7 dargestellt.
Das Unterbinden des Botschaftsversands auf einem kompromittierten Steuergerät ist ein weiteres Beispiel für Denial-of-Service (*Suspension*-Attacke [CS16]). In Kombination mit Spoofing entsteht eine sogenannte *Masquerade*-Attacke [CS16]. In diesem Fall unterbindet der Angreifer den Versand der originalen Botschaft und sendet stattdessen eine selbst erzeugte mit gleicher ID. Dadurch wird sowohl die Unterdrückung als auch die Erzeugung von Botschaften - und somit die gesamte Attacke - verschleiert.

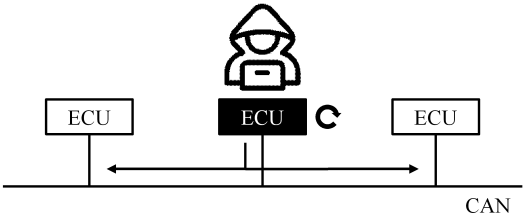

Bild 3.8: Tampering-Attacke auf dem originalen Sender[3]

Mit mehr Systemwissen und Zugriffsmöglichkeiten kann ein Angreifer auch den originalen Inhalt einer Botschaft manipulieren (*Tampering*-Attacke), beispielsweise direkt auf einem kompromittierten Sender, siehe Bild 3.8[4].

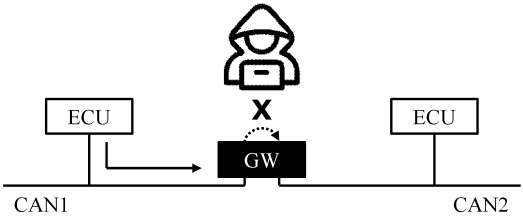

Bild 3.9: Blockieren der Botschaftsweiterleitung[3]

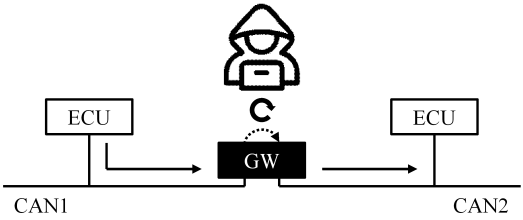

Bild 3.10: Manipulation von Botschaften während des Weiterleitens[3]

Bisher lag der Fokus der diskutierten Attacken auf regulären Busteilnehmern (Endknoten). Für Angreifer sind allerdings Gateway-Steuergeräte besonders interessant. Damit lassen sich die erwähnten Attacken potentiell auf allen angeschlossenen Bussystemen realisieren. Zusätzlich ermöglichen Gateways zwei weitere Angriffsmöglichkeiten beim Weiterleiten von

[4] Das von Cho et al. vorgestellte Clock-based IDS (CIDS) nutzt den Takt-Versatz zwischen Steuergeräten und bestimmt basierend darauf die Sender von periodischen Botschaften [CS16]. Das Vorgehen ermöglicht die Detektion von Fabrication-, Suspension- und Masquerade-Attacken. Tampering im originalen Sender ist durch das CIDS hingegen nicht erkennbar.

Botschaften. Zum einen lässt sich das Weiterleiten blockieren, was einer Suspension- oder Drop-Attacke auf dem Zielbus entspricht. Als sogenannter *Man-in-the-Middle* ist ein Gateway auch in der Lage Nachrichten während des Weiterleitens zu verändern. Dies ist eine andere Form von Tampering. Beide Attacken sind in Bild 3.9 und Bild 3.10 dargestellt.

Die vorgestellten Attacken wurden anhand von CAN erläutert. Für andere, Fahrzeug-interne Kommunikationssysteme sind die gleichen oder ähnliche Angriffe ebenfalls möglich - sofern keine Gegenmaßnahmen getroffen werden. Im Fall von Ethernet müssen Angreifer jedoch beachten, dass die Kommunikation Empfänger-adressiert erfolgt und Switches die Nachrichten nur an entsprechende Punkt-zu-Punkt Verbindungen weiterleiten. Dadurch wird unter anderem das Mitlesen aller Nachrichten im Netzwerk erschwert. Hat ein Angreifer Zugriff auf den Switch, sind alle Nachrichten dieses Sub-Netzwerks sichtbar und es ergeben sich die diskutierten Angriffsmöglichkeiten. Daher sind Switches beziehungsweise die entsprechenden Steuergeräte in einem Ethernet-Netzwerk besonders zu schützen.

4 Der Automotive Observer

Das E/E-System hat sich über die letzten Jahrzehnte zu einem der wichtigsten Teilsysteme eines Fahrzeugs entwickelt. Es trägt maßgeblich zu einem Großteil der Sicherheitsfunktionen bei, angefangen bei der Fahrzeugbeleuchtung, über Airbags bis hin zu elektronischen Fahrassistenten. Daher ist die funktionale Sicherheit seit langem integraler Bestandteil der E/E-Entwicklung. Die Betrachtung der Angriffssicherheit war hingegen lange nicht im Fokus vieler Hersteller, da das Fahrzeug als abgeschlossenes System betrachtet wurde. Eine gezielte Manipulation des E/E-Systems erforderte lange Zeit physischen Zugriff auf einzelne Komponenten. Durch die zunehmende Vernetzung von Fahrzeugen mit ihrer Umgebung per Luftschnittstelle, ist diese Prämisse nicht mehr gültig und die Angriffssicherheit ist inzwischen ein weiterer zentraler Punkt bei der E/E-Entwicklung.
In Kapitel 3 wurden die Mechanismen der funktionalen Sicherheit und der Angriffssicherheit separat voneinander betrachtet, wobei eine starke Kopplung zwischen den beiden Themen besteht. Kann die Angriffssicherheit nicht gewährleistet werden, ist auch die funktionale Sicherheit nicht mehr gegeben. Dies trifft besonders auf Fahrzeuge mit Assistenzsystemen zu, die aktiv das Fahrverhalten beeinflussen oder sogar Fahraufgaben übernehmen. Außerdem haben sowohl Safety als auch Security das Ziel ein Fehlverhalten einzelner Komponenten - und damit in letzter Konsequenz des Fahrzeugs - zu verhindern. Lediglich die betrachteten Ursachen unterscheiden sich. Daraus ergibt sich das folgende Ziel der vorliegenden Arbeit:

Das übergeordnete Ziel ist eine ursachenunabhängige Erkennung von Fehlverhalten im E/E-System eines Fahrzeugs in Echtzeit. Der Mechanismus,

*welcher diese Fehlererkennung realisiert wird im Folgenden **Automotive Observer (kurz: Observer)** genannt und soll auf einem oder mehreren Steuergeräten ausgeführt werden.*

4.1 Grundlegender Ansatz

Im Bereich der IT-Security erkennen IDS unerwünschtes Systemverhalten entweder Signatur- oder Anomalie-basiert, siehe Abschnitt 3.2.2. Bis vor wenigen Jahren war der Einsatz von Signatur-basierten IDS in Fahrzeugen nicht vorstellbar. Ein Hauptgrund war die Notwendigkeit von regelmäßigen Aktualisierungen der Angriffssignaturen [MGF10]. Diese Einschränkung existiert heute nicht mehr. Zukünftige Fahrzeuge bieten die Möglichkeit von Software-Aktualisierungen über eine Mobilfunkverbindung oder Wireless Local Area Network (WLAN) mittels OTA/SOTA. Ein weiterer wichtiger Punkt für Signatur-basierte Systeme ist der Hersteller-übergreifende Austausch von Sicherheitslücken und Angriffssignaturen. Dieser kann über sogenannte Information Sharing and Analysis Center (ISAC) erfolgen. Ein entsprechendes ISAC für die Automobilindustrie wurde 2015 gegründet, siehe Unterkapitel 3.2. Unter der Voraussetzung, dass der Informationsaustausch funktioniert, ist damit auch die zweite Voraussetzung für ein Signatur-basiertes IDS gegeben. Ein Nachteil dieses Verfahrens bleibt jedoch bestehen: Der Fokus auf bekannte Schwachstellen und Angriffe.
Im Gegensatz zu Signatur-basierten Systemen können Anomalie-basierte IDS auch unbekannte Angriffe detektieren. Allerdings besteht bei diesem Ansatz die Herausforderung, das Normalverhalten möglichst genau zu definieren. Hier hilft die statische Auslegung von automotive E/E-Architekturen verglichen zu klassischen IT-Systemen. Für Fahrzeug-interne Netzwerke existiert in der Regel eine Systembeschreibung, in der Steuergeräte und ausgetauschte Nachrichten semi-formal definiert sind. Wird das AUTOSAR Austauschformat verwendet, kann der Fahrzeughersteller darin sogar die zu

implementierenden Software-Komponenten und deren Schnittstellen spezifizieren. Dies ist ein großer Unterschied zur weit dynamischeren, klassischen IT und erleichtert die Definition des Normalverhaltens maßgeblich. Mit der Einführung von Ethernet und Service-orientierter Kommunikation wird auch der Datenaustausch in Fahrzeug-internen Netzwerken dynamischer [Web13] [Web15] [JWH17]. Gleiches gilt für die Software von leistungsstarken Steuergeräten. Unter Verwendung von AUTOSAR Adaptive ist die Installation von zusätzlichen Applikationen zur Laufzeit möglich. Damit nähern sich automotive Netzwerke und Steuergeräte klassischen IT-Systemen an. Ein weiterer Nachteil von Anomalie-basierten IDS ist üblicherweise eine erhöhte Falschalarmrate. Dies ist bei einer Anwendung im Fahrzeug zu vermeiden, unabhängig von einer potentiellen Reaktion. Zu viele Falschalarme würden die Akzeptanz eines solchen Systems reduzieren. Selbst wenn keine aktiven Gegenmaßnahmen getroffen werden, führen viele Falschalarme zu einer potentiell nicht mehr überschaubaren Menge an Daten, die im Nachgang zu analysieren wären.
Aus drei Gründen wurde zu einem frühen Zeitpunkt der vorliegenden Arbeit entschieden, den Observer dennoch Anomalie-basiert zu konzipieren:

1. Der Fokus auf bekannte Angriffe bei Signatur-basierten IDS ist im automotive Umfeld unattraktiv, auch wenn Mechanismen für drahtlose Software-Aktualisierungen und den notwendigen Informationsaustausch vorhanden sind. Ein anderweitig entdeckter Angriff[1] muss zunächst analysiert, die Informationen verteilt, eine entsprechende Software-Aktualisierung entwickelt und anschließend ausgerollt werden. Für ein reines IDS ist dies gegebenenfalls noch ausreichend. Das sofortige Einleiten von Gegenmaßnahmen, auch bei bis dato unbekannten Angriffen, ist hingegen nicht möglich.

[1] Würden alle Systeme Signatur-basiert arbeiten, wäre es schwierig überhaupt neue Angriffe zu entdecken.

2. Die statische Auslegung von automotive E/E-Architekturen ist ein Vorteil, der sich bei der Definition des Normalverhaltens nutzen lässt. Je mehr Informationen über ein System a priori vorhanden sind, desto besser lassen sich üblicherweise Anomalieerkennungsverfahren anwenden. Auch wenn Netzwerke und Steuergeräte in Zukunft dynamischer aufgebaut sind, bleibt das System deutlich statischer als IT-Infrastruktur und Computer. Sollten die erforschten Anomalieerkennungsverfahren für Ethernet-Netzwerke und AUTOSAR Adaptive Steuergeräte nicht mehr anwendbar sein (aufgrund der größeren Dynamik), betrifft dies nach heutigem Diskussionsstand nur einen Teil der E/E-Architektur, siehe Abschnitt 2.1.3. Außerdem besteht bei Signatur-basierten Systemen eine ähnlich herausfordernde Aufgabe, da sich ein Angriff im Laufe der Zeit weiterentwickeln kann und sich damit unter Umständen Varianten mit (leicht) unterschiedlichen Signaturen ausprägen.

3. Wie am Anfang dieses Kapitels beschrieben, soll der Observer Security und Safety gleichermaßen adressieren. Da mittels Anomalieerkennung Abweichungen vom Normalverhalten ursachenunabhängig erkannt werden, ist diese Anforderung implizit erfüllt. Eine Übertragung des Signatur-basierten Ansatzes auf Safety-Aspekte würde hingegen die Spezifikation aller möglichen Fehlverhalten erfordern.

4.1.1 Einordnung im Bereich Security

Unter dem Gesichtspunkt der Security, ist der Observer ein Anomalie-basiertes IDS, das Cyber-Angriffe anhand von Abweichungen zum Normalverhalten innerhalb eines Steuergeräts in Echtzeit erkennt. Dabei kombiniert er Netzwerk- und Host-basierte Überwachungsmechanismen. Systeme aus der Büro- und Internetwelt können aufgrund der unterschiedlichen Randbedingungen größtenteils nicht direkt übernommen werden. Dazu zählen die statische Netzwerkauslegung, Fahrzeug-spezifische Bussysteme wie CAN,

unterschiedliche Kommunikationsparadigmen (Signal- und Serviceorientierung), unterschiedliche Software-Architekturen und Betriebssysteme sowie die geforderte Echtzeitfähigkeit in Kombination mit limitierten Speicher- und Rechenressourcen.

4.1.2 Einordnung im Bereich Safety

Das E/E-System eines Fahrzeugs muss unter allen Umständen die funktionale Sicherheit gewährleisten. Dafür kommen die in Unterkapitel 3.1 und Anhang A.2 beschrieben Mechanismen bereits flächendeckend in Steuergeräten zum Einsatz. Im Vergleich zur Security fällt auf, dass viele Safety-Mechanismen den Hostanteil adressieren. Der Netzwerkanteil ist durch die End-to-End Protection abgedeckt. Im Blickwinkel der Safety ist der Observer daher ein System-Monitor, der einige existierende Mechanismen vereinigt und ergänzt, um das korrekte Verhalten eines Steuergeräts und der entsprechenden Kommunikationspartner zu überwachen.

4.2 Aufgaben und Anforderungen

Das Erkennen von Fehlverhalten, unabhängig von dessen Ursache, ist die Hauptaufgabe des Observers. Wie in Unterkapitel 4.1 beschrieben, ist dies für Security und Safety gleichbedeutend. Daher wird bei der Aufgabenbeschreibung nicht mehr zwischen den beiden Bereichen unterschieden.

Die angestrebte, ganzheitliche Überwachung eines Steuergeräts beinhaltet mehrere Aspekte, wie in Bild 4.1 dargestellt. Ein Teil ist die Betrachtung der *Kommunikation*, allerdings ohne Wissen über die funktionalen Aufgaben der kommunizierenden Steuergeräte beziehungsweise Applikationen. Dies erleichtert den Entwicklungsprozess. Eine Abstraktionsebene tiefer ist zwischen Steuergeräte-interner und -externer Kommunikation zu unterscheiden. Externe Kommunikation beschreibt den Datenaustausch über Fahrzeug-interne Netzwerke wie CAN, LIN, FlexRay oder Ethernet. Interne Kommunikation beschreibt den Datenaustausch zwischen

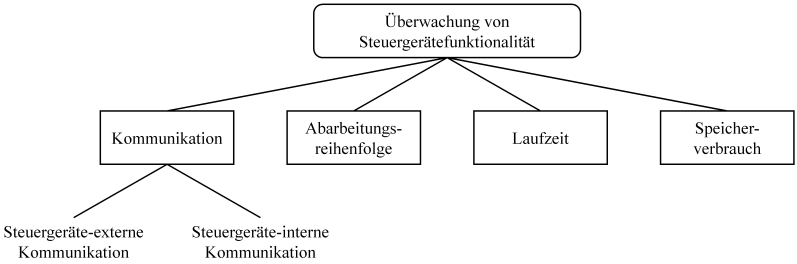

Bild 4.1: Zu überwachende Aspekte der Steuergerätefunktionalität

Applikationen/SW-Cs, die auf dem gleichen Steuergerät realisiert sind. Die hierbei ausgetauschten Daten können auch klassische Ein- und Ausgangssignale eines Steuergeräts repräsentieren, beispielsweise die Werte von direkt angeschlossenen Sensoren und Aktoren. Die Überwachung der Steuergeräte-externen Kommunikation entspricht dem Netzwerk-basierten Anteil des Observers. Der Steuergeräte-interne Datenaustausch ist ein Host-basierter Aspekt.

Ein weiterer, Host-basierter Anteil ist die Analyse der *Abarbeitungsreihenfolge*, von Prozess- beziehungsweise Taskebene bis hin zu einzelnen Funktionsaufrufen. Auf Funktionsebene bietet speziell die Interaktion von Applikations- und Basissoftware Überwachungspotential. In höheren Betriebssystemen handelt es sich um die Sequenz von aufgerufenen System-Calls. In einem AUTOSAR Classic-basierten Steuergerät sind es die Aufrufe von Basissoftware- und SW-C-Schnittstellen über die RTE.

Im Kontext der Abarbeitungsreihenfolge kann auch die *Laufzeit* einzelner Elemente betrachtet werden. Zudem gibt die reine Prozessorauslastung bereits Hinweise auf den Zustand eines Steuergeräts.

Neben der Laufzeit ist auch der *Speicherverbrauch* von Prozessen und Funktionen zu überwachen. Zusätzlich zur Memory Protection (siehe Anhang A.2.1) lässt sich der Stackverbrauch feiner granular untersuchen, um Unregelmäßigkeiten in der Abarbeitung (frühzeitig) zu erkennen.

Im Security-Umfeld wird zwischen Intrusion Detection und Intrusion Prevention Systemen unterschieden. Die Aufgaben eines IDS sind die Erkennung einer Cyber-Attacke und das Melden oder Protokollieren dieser. Im Gegensatz dazu ist ein IPS in der Lage aktiv in das überwachte System einzugreifen, um beispielsweise eine Cyber-Attacke zu unterbinden oder deren Auswirkung einzuschränken. Der Ablauf und die Auswirkung eines Eingriffs sind anwendungsspezifisch und nicht standardisiert. Bei Safety-Mechanismen ist die Reaktion auf ein erkanntes Fehlverhalten meistens exakt spezifiziert. Die Reaktionen haben das Ziel einen unsicheren Zustand zu vermeiden beziehungsweise den sicheren Zustand wieder herzustellen. Sie reichen vom Verwerfen einer korrupten Kommunikationsnachricht bis hin zu einem Steuergeräte-Neustart.
In Sachen Fehlerbehandlung soll sich der vorgestellte Observer wie ein IDS verhalten. Eine erkannte Abweichung vom Normalverhalten ist inklusive Zusatzinformationen zum Steuergerätezustand zu protokollieren. Ein aktives Eingreifen in die Steuergerätefunktionalität ist im ersten Schritt nicht erwünscht. Die Auswirkungen von Gegenmaßnahmen, wie blockierte Kommunikationsnachrichten, Netzwerkteilnehmer oder Applikationen, können positiv wie negativ sein und stehen nicht im Fokus dieser Arbeit.
Um die Design-Entscheidungen in Unterkapitel 4.3 besser nachvollziehen zu können, sind nachfolgend alle Anforderungen aufgelistet.

[Req1] Der Observer muss Auffälligkeiten in Fahrzeug-interner Kommunikation erkennen und dabei verschiedene Netzwerktechnologien unterstützen, wie beispielsweise CAN und Ethernet (vgl. Netzwerkbasiertes IDS).

[Req2] Der Observer soll Auffälligkeiten bei der Ausführung von Steuergeräte-Software erkennen. Dies beinhaltet die Überwachung von Steuergeräte-interner Kommunikation, Abarbeitungsreihenfolgen, Laufzeiten und Speicherverbrauch (vgl. Host-basiertes IDS).

[Req3] Der Observer muss nach dem Prinzip der Anomalieerkennung (White-List-Ansatz) arbeiten.

[Req4] Die Erkennung von Auffälligkeiten soll direkt im Fahrzeug und unabhängig von einer dauerhaften Verbindung zu externer Infrastruktur (wie beispielsweise einem Backend) erfolgen.

[Req5] Die Erkennung von Auffälligkeiten muss in Echtzeit auf einem oder mehreren Steuergeräten erfolgen, um zukünftig sofortige Gegenmaßnahmen einleiten zu können.

[Req6] Der Observer soll erkannte Auffälligkeiten und die dazugehörigen Informationen zur späteren Analyse sicher abspeichern.

[Req7] Der Observer soll komplett in Software realisierbar sein, um Hardware-Abhängigkeiten zu vermeiden.

[Req8] Die Implementierung des Observers muss einen möglichst geringen Ressourcenbedarf - Laufzeit und Speicherverbrauch - aufweisen, sodass eine Realisierung auf aktuellen 32 Bit automotive Mikrocontrollern[2] möglich ist.

[Req9] Der Observer soll modular aufgebaut sein, sodass einzelne Bestandteile entfallen können, falls diese nicht benötigt werden oder falls es bei der Implementierung auf einem Steuergerät zu einem Ressourcenengpass kommt. Gleichzeitig soll dies die Umsetzung einzelner Bestandteile in (spezieller) Hardware ermöglichen.

[Req10] Der Observer soll erweiterbar sein, um die Überwachung weiterer Netzwerktechnologien oder Steuergeräteeigenschaften möglichst einfach realisieren zu können.

[2] Beispiele für aktuelle 32 Bit automotive Mikrocontroller sind NXP MPC57xx, Renesas RH850 und Infineon Aurix TC3xx.

[Req11] Für einzelne Überwachungsaufgaben gibt es existierende und etablierte Mechanismen aus dem Security- und Safety-Bereich. Diese kann der Observer wiederverwenden.

4.3 Systemarchitektur

Der grundlegende Aufbau des Observers wurde in [VWeb+18a] initial vorgestellt. Da die Überwachung auf dem Prinzip der Anomalieerkennung basiert, ist eine präzise Definition des Normalverhaltens notwendig. Hierfür ist die statische Auslegung einer automotive E/E-Architektur vorteilhaft. Im heute üblichen Entwicklungsprozess entstehen beim Fahrzeughersteller Artefakte, welche die Fahrzeug-interne Kommunikation und im Falle von AUTOSAR Classic auch die genutzten Applikationsschnittstellen semi-formal beschreiben, siehe Abschnitt 2.1.5. Es wird davon ausgegangen, dass diese Artefakte - die Kommunikationsmatrix oder die AUTOSAR System Description - als Spezifikation des Normalverhaltens zur Verfügung stehen. Basierend darauf und auf den standardisierten Protokollspezifikationen lassen sich Überwachungsregeln ableiten, nachfolgend als *statische Checks* bezeichnet. Diese Definition deckt sich mit dem Anwendungskriterium „Specification-Based“ (AC-1), das Müter et al. für ihre Anomalieerkennungssensoren einführen [MGF10], wobei Hersteller-spezifische und informelle Spezifikationen für den Observer nicht berücksichtigt werden. Für manche Überwachungsmöglichkeiten beziehungsweise Sensoren sind Informationen notwendig, die in den aktuellen Entwicklungsartefakten nicht enthalten sind. Die Definition von minimalem und maximalem Signalwert reicht beispielsweise nicht aus, um einen Signalverlauf plausibilisieren zu können. Dazu müssten zusätzliche Informationen wie der maximale und minimale Gradient vorhanden sein. Eine Ergänzung der Artefakte um die benötigten Werte wäre technisch möglich, würde aber größeren prozesstechnischen und organisatorischen Aufwand bedeuten. Sollen zum Beispiel zwei zusätzliche Parameter den maximalen und minimalen Gradienten des

Geschwindigkeitssignals spezifizieren (maximale Beschleunigung und Verzögerung), müsste dies in Abhängigkeit der Motorleistung geschehen. Die Folge wäre eine erhöhte Varianz der Artefakte, was deren bereits komplexe Erzeugung nochmals deutlich aufwändiger machen würde. Außerdem ist fraglich, ob ein Fahrzeughersteller die zusätzlichen Werte spezifizieren kann. Möglicherweise existiert das dafür benötigte Wissen nur bei einem Zulieferer oder die Werte lassen sich nur schwer a priori festlegen. Daher verwendet der Observer für die Überwachung aktuell nicht einheitlich spezifizierter Eigenschaften Verfahren und Algorithmen des maschinellen Lernens. Die in der Trainingsphase erzeugten Modelle repräsentieren das Normalverhalten und werden während der Laufzeit zur Anomalieerkennung verwendet. Diese Art der Überwachung wird mittels *lernenden Checks* realisiert.

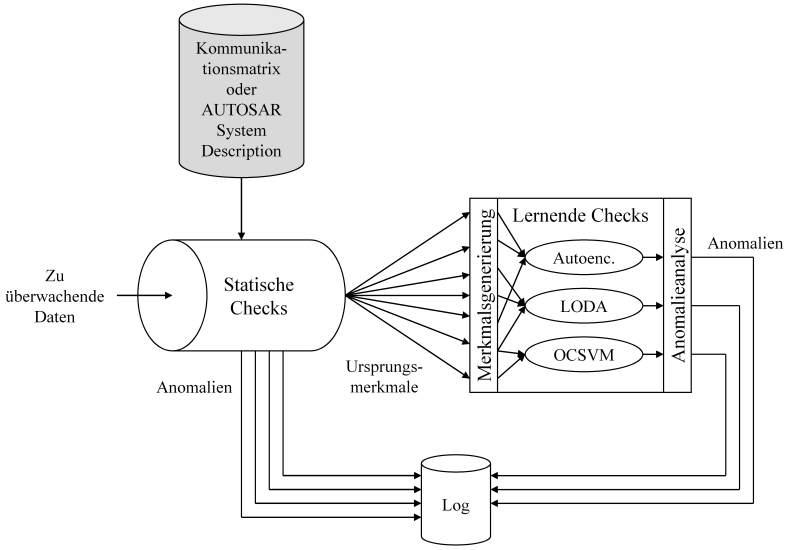

Bild 4.2: Aufteilung in statische und lernende Checks[3]

[3] Angelehnt an [VWeb+18a]

Bild 4.2 zeigt die Aufteilung in statische und lernende Checks innerhalb des Observers. Salman und Bresch argumentieren ebenfalls für diese Aufteilung (spezifikationsbasierte Überwachung und Anomalieerkennung, siehe Abschnitt 3.2.2) [SB17]. In ihrem Konzept arbeiten alle Checks auf einer Ebene und werden nacheinander ausgeführt. Im Gegensatz dazu besteht der Observer aus einem zweistufigen System. Die erste Stufe sind die statischen Checks, welche komplett aus der Kommunikationsmatrix oder der AUTOSAR System Description abgeleitet sind. Sie trägt erkannte Anomalien direkt in ein Fehlerspeicher oder eine *Log*-Datei ein. Zusätzlich leiten statische Checks relevante Daten an die zweite Stufe weiter. Damit extrahiert die erste Stufe bereits Informationen für die nachfolgenden lernenden Checks, die *Ursprungsmerkmale* in Bild 4.2. Die implementierten Algorithmen des maschinellen Lernens benötigen gegebenenfalls weitere, davon abgeleitete Merkmale. Soll beispielsweise der Geschwindigkeitsverlauf plausibilisiert werden, ist eine Zeitreihe des entsprechenden Signalwerts nötig. Deren Generierung und Aktualisierung übernimmt der Block *Merkmalsgenerierung*. An diesem Beispiel wird gleichzeitig der Vorteil des zweistufigen Konzepts für die spätere Implementierung deutlich. Um die Fahrzeuggeschwindigkeit auf deren maximalen und minimalen Wert zu prüfen, müssen die statischen Checks das entsprechende Signal zuerst aus dem Daten-Feld einer Kommunikationsnachricht extrahieren, siehe Bild 2.8. Dies ist für die Plausibilisierung ebenfalls erforderlich. Das Weiterleiten des Signalwerts als Ursprungsmerkmal spart Rechenzeit, da eine erneute Extraktion in der zweiten Stufe überflüssig ist. Des Weiteren werden alle Merkmale - sowohl die Ursprungsmerkmale als auch die davon abgeleiteten - einmalig an zentraler Stelle abgelegt. Die nachfolgenden lernenden Checks können sich beliebig daraus bedienen. Somit lässt sich der Speicherbedarf für die Merkmalsverwaltung minimieren. In Bild 4.2 teilen sich zum Beispiel zwei Algorithmen (LODA und OCSVM) das gleiche Merkmal. Merkmalsgenerierung und -verwaltung sind damit möglichst effizient gestaltet.
Das Konzept erlaubt zusätzlich die parallele Verwendung unterschiedlicher

Algorithmen, wie Autoencoder, LODA und OCSVM (siehe Bild 4.2). Das Ergebnis eines Algorithmus ist entweder ein binärer Wert (Anomalie: ja/nein) oder die Anomalie-Wahrscheinlichkeit, auch bekannt als Anomaly-Score. Die finale Anomaliebewertung geschieht innerhalb des Blocks *Anomalieanalyse*. Liefert der Algorithmus selbst bereits eine finale Entscheidung, schreibt die Anomalieanalyse erkannte Anomalien in den Fehlerspeicher oder in eine Log-Datei. Liegt ein Anomaly-Score vor, wird beispielsweise anhand eines Schwellwerts entschieden, ob eine Anomalie abzuspeichern ist. Um die Erkennungsraten beziehungsweise die Konfidenz zu steigern, können mehrere Algorithmen die gleichen Merkmale auf Anomalien prüfen [The17]. Ist dies der Fall, spricht man von einem inhomogenen Ensemble. Das Prinzip entspricht dem des homogenen Ensembles (siehe Abschnitt 2.3.2), mit dem Unterschied, dass ein inhomogenes Ensemble aus unterschiedlichen Algorithmen besteht. Auch dieser Ansatz wird mit dem vorgestellten Konzept unterstützt. In der Anomalieanalyse lassen sich die Ergebnisse mehrerer Algorithmen zusammenfassen, gewichten und weiter verarbeiten, um eine finale Bewertung zu erhalten. Insgesamt erlaubt das vorgestellte Konzept die flexible und effiziente Instanziierung unterschiedlicher Anomalieerkennungsalgorithmen und -verfahren in einer einheitlichen Systemarchitektur.

4.3.1 Entwicklungsprozess

Bisher wurde vorwiegend auf die statischen und lernenden Checks zur Anomalieerkennung auf dem Steuergerät sowie das dazugehörige Logging eingegangen. Der folgende Abschnitt fokussiert sich auf die Erstellung der Checks sowie den damit verbundenen Prozess und basiert auf [VWeb+18b] und [VWeb+]. Wie bereits in Bild 4.2 angedeutet, beginnt ein effizienter Erstellungsprozess mit einer semi-formalen Beschreibung der Kommunikation und von Steuergeräte-Interna mittels Kommunikationsmatrix oder AUTOSAR System Description. Sofern diese Beschreibungen vollständig sind,

lassen sich daraus per Definition alle statischen Checks ableiten. Im Optimalfall wird direkt Steuergerätecode generiert. Aus mehreren Gründen ist es dennoch hilfreich einen Zwischenschritt einzuführen:

1. Im Laufe einer Steuergeräte-Entwicklung verändert sich auch die Kommunikationsmatrix beziehungsweise die Systembeschreibung. Frühe Versionen sind häufig unvollständig oder fehlerhaft. Um dennoch einen funktionsfähigen Observer erstellen zu können, ist es notwendig die abgeleiteten statischen Checks vor der Codegenerierung zu überprüfen und gegebenenfalls manuell anzupassen.

2. Durch die begrenzten Ressourcen in einem Steuergerät ist es in der Regel nicht möglich alle potentiellen Überwachungen zu realisieren. In diesem Fall sind die wichtigsten zu selektieren.

3. Die Beschreibungsdateien enthalten nicht nur notwendige Informationen für die statischen Checks sondern auch nützliche Informationen für die lernenden Checks. So geht aus der Kommunikationsmatrix beispielsweise hervor, welche Signale zur Plausibilisierung vorhanden sind.

Aus den zuvor genannten Gründen wurde das Konzept um eine Konfigurationssoftware erweitert, den *Anomaly Detection Configurator*. Der Anwender führt dieses Tool auf seinem Desktopcomputer aus und liest damit zu Beginn die Kommunikationsmatrix beziehungsweise die AUTOSAR System Description ein. Anstatt direkt Steuergerätecode zu generieren, leitet das Tool zunächst die Überwachungsregeln ab und stellt diese in einem menschenlesbaren Format dar. Der Anwender hat daraufhin die Möglichkeit eventuell falsch abgeleitete Regeln zu korrigieren, zusätzliche Regeln einzubringen oder Regeln zu löschen. Außerdem kann er auf Basis der eingelesenen Informationen festlegen, welche lernenden Checks implementiert werden. Den finalen Regelsatz speichert der Anomaly Detection Configurator in der *Anomaly Detection Configuration*. Diese enthält alle Informa-

tionen, welche für die Generierung der statischen Checks notwendig sind. Im Gegensatz dazu enthält sie für die lernenden Checks zunächst nur die vorab festzulegenden Hyperparameter sowie die Algorithmenparameter in untrainierter Form. Nachdem die Parameter durch entsprechendes Training ermittelt wurden - bei neuronalen Netzen sind es die Kantengewichte - ist die Anomaly Detection Configuration entsprechend zu aktualisieren. Danach wird auf Basis dieser generischen Beschreibung der Quellcode für die Zielplattform generiert, zum Beispiel für AUTOSAR Classic oder AUTOSAR Adaptive.

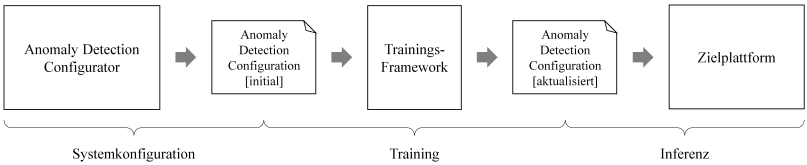

Bild 4.3: Arbeitsablauf von der Systemkonfiguration bis zur Inferenz[4]

Der gesamte Arbeitsablauf unterteilt sich damit in die drei Schritte Systemkonfiguration, Training und Inferenz. Eine Zusammenfassung ist in Bild 4.3 dargestellt. Der Trainingsschritt entfällt bei statischen Checks.

Bild 4.4 basiert auf [VWeb+18b] und zeigt, dass statische Checks spezifisch für das jeweilige Überwachungsgebiet sind. Im dargestellten Beispiel sind es die Kommunikation via CAN und Ethernet sowie die Überwachung von Betriebssysteminformationen wie Laufzeit und Speicherverbrauch. Das Framework für die lernenden Checks ist hingegen allgemeingültig. Alle implementierten Algorithmen bedienen sich aus der zentralen Merkmalsablage und erzeugen einen Ausgabewert, der nachfolgend noch bewertet wird.
Das in Bild 4.4 ebenfalls neu hinzugekommene *Anomaly Logging* abstrahiert das persistente Speichern der erkannten Anomalien. Damit ist das

[4] Adaptiert von [VWeb+]
[5] Adaptiert von [VWeb+18b]

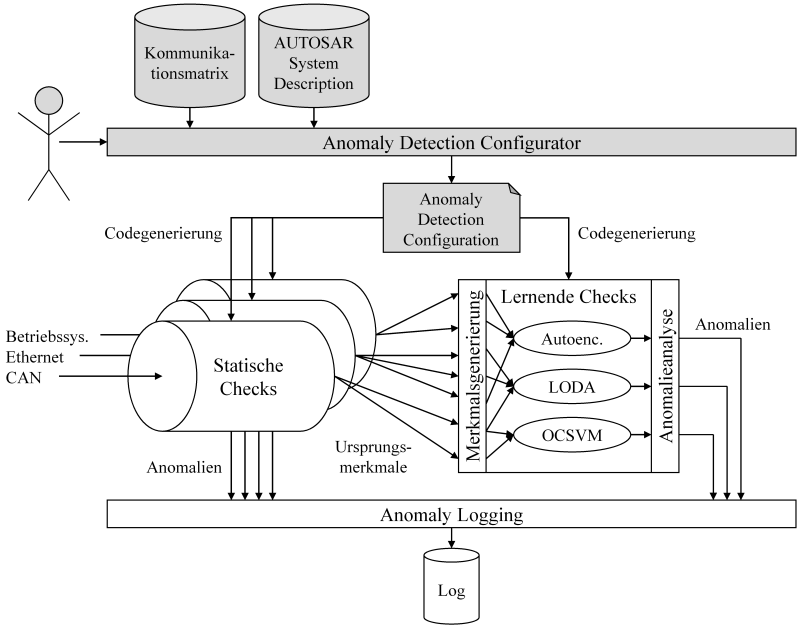

Bild 4.4: Gesamtkonzept für den Automotive Observer[5]

Verwenden von unterschiedlichen Logging-Mechanismen möglich. In AUTOSAR Classic erfolgt das Ablegen von Fehlerspeichereinträgen über den Diagnostic Event Manager (DEM), der seit Release 4.4 auch das sichere Ablegen von Security-relevanten Informationen unterstützt. Der DEM ist damit der geeignete Logging-Mechanismus für erkannte Anomalien unter AUTOSAR Classic. Ein vergleichbarer Mechanismus ist in AUTOSAR Adaptive noch nicht standardisiert. Ein Ansatz ist die Verwendung eines Daemons, der Anomalien in eine Log-Datei schreibt und diese kryptografisch absichert. Soll das persistente Speichern nicht nur auf dem Steuergerät selbst erfolgen, realisiert das Anomaly Logging auch die Anbindung an ein Backend-System, auf welchem die erkannten Anomalien gesammelt und weiter verarbeitet beziehungsweise ausgewertet werden können.

4.3.2 Erfüllung der aufgestellten Anforderungen

Der gewählte Systemaufbau erfüllt beziehungsweise unterstützt die meisten der in Unterkapitel 4.2 aufgestellten Anforderungen. Die Unterstützung von [Req1], [Req2] und [Req10] sowie die Erfüllung von [Req9] erfolgt durch den modularen Aufbau und die generische Gestaltung der lernenden Checks. Die tatsächliche Erkennung von Anomalien sowie die Verwendung der passenden Verfahren ([Req3] und [Req11]) obliegt den implementierten Algorithmen. Mit der möglichst ressourcenschonenden Auslegung, beispielsweise durch die statischen Checks und die Wiederverwendung von Zwischenergebnissen (Ursprungsmerkmale) in den lernenden Checks, unterstützt die Systemarchitektur [Req4], [Req5] und [Req8]. Ebenfalls abgedeckt ist [Req6], wobei durch die zusätzliche Entkopplung mittels Anomaly Logging auch eine Anbindung an externe Infrastruktur möglich ist. [Req7] is uneingeschränkt erfüllt, da keinerlei Hardwareunterstützung vorausgesetzt wird.

4.4 Anomalieerkennung in Fahrzeug-interner Kommunikation

Die meisten Fahrzeug-Hacks sind mehrstufig aufgebaut, wie das Beispiel von Miller und Valasek [MV15] sowie die Defense-in-Depth-Strategie zeigen. Eine Stufe basiert üblicherweise auf dem Einspeisen ungewollter Kommunikationsnachrichten in ein Fahrzeug-internes Netzwerk, um beispielsweise gültige Nachrichten zu überschreiben, einen Mikrocontroller zu reprogrammieren, ein Steuergerät in Überlast zu treiben und damit einen Reset auszulösen oder einen Denial-of-Service Angriff zu realisieren. Die Auswirkungen eines resultierenden Fehlverhaltens können gravierend ausfallen. Daher wird im Folgenden die Anomalieerkennung in Fahrzeug-internen Netzwerken näher betrachtet. Dabei liegt der Fokus im ersten Schritt auf CAN. Obwohl Ethernet in aktuelle Fahrzeuggenerationen Einzug hält und dabei andere Netzwerktechnologien wie MOST und teilweise FlexRay aus

dem Fahrzeug verdrängt, ist CAN vor allem in sicherheitskritischen Bereichen wie dem Antriebsstrang weiterhin das dominierende Bussystem. Auch wenn sich Ethernet weiter im Fahrzeug verbreitet, ist momentan davon auszugehen, dass CAN in Zukunft dennoch eine wichtige Rolle einnimmt. Aus Kostengründen ist eine vollständige Ablösung von CAN durch den aktuellen Automotive Ethernet Standard 100BASE-T1 (siehe Abschnitt 2.1.2) nicht absehbar[6].

4.4.1 Observer vs. Anomalieerkennungssensoren

Abschnitt 3.2.2 beschreibt die Strukturierung von potentiellen Auffälligkeiten im Kontext eines Fahrzeug-internen und Netzwerk-basierten IDS nach Müter et al. [MGF10].

Die ersten vier Spalten von Tabelle 4.1 fassen die bisherige Diskussion zusammen. Wird in der Spalte *Informationsquellen* auf die standardisierte Kommunikationsmatrix verwiesen, ist zusätzlich die erforderliche Variante angegeben. Je nach Informationsgehalt, ist zwischen System-, Bussystem- und ECU-Ebene zu unterscheiden. Die meisten Informationen sind enthalten, wenn eine Systembeschreibung vorliegt. In dieser Form sind alle Netzwerke und deren Nachrichten in der Kommunikationsmatrix spezifiziert. Eine Ebene darunter existiert eine Netzwerk-spezifische Beschreibung mit allen dazugehörigen Nachrichten. Die restriktivste Form bezieht sich ausschließlich auf ein Steuergerät und enthält nur die dafür relevanten Informationen. Diese Variante wird vom Fahrzeughersteller üblicherweise an seine Zulieferer verteilt, um möglichst wenig Systemwissen preiszugeben[7].
Alle Sensoren, denen ausschließlich standardisierte Informationsquellen zu

[6] Die aktuell in der Spezifikation befindliche und kostengünstigere Version mit einer Bandbreite von 10 Mbit/s könnte dies ändern. Allerdings wird es selbst dann einige Jahre dauern, bis alle Steuergeräte auf die neue Netzwerktechnologie umgestellt sind.

[7] Sowohl das DBC- als auch das ARXML-Format können alle Varianten darstellen. Für das DBC-Format sind die ersten beiden gängig. Eine Gesamtsystembeschreibung erfordert in diesem Fall Hersteller-spezifische Erweiterungen. AUTOSAR unterscheidet standardmäßig zwischen der *System Description* mit System- oder Netzwerkbezug und dem *ECU Extract of System Description*, welcher auf ein Steuergerät zugeschnitten ist.

Nr.	Sensor	Spezifikation	Informationsquellen	Realisierung
N-1	Formality	Standardisiert	CAN-Spezifikation	Statische Checks
N-2	Location	Standardisiert	Kommunikationsmatrix (System- oder Bussystemebene)	Statsiche Checks
N-3	Range	Standardisiert	Kommunikationsmatrix (System-, Bussystem- oder ECU-Ebene)	Statische Checks
N-4	Frequency	Periodische Botschaften: Standardisiert	Kommunikationsmatrix (System-, Bussystem- oder ECU-Ebene)	Statische Checks
		Sporadische Botschaften: Nicht herstellerübergreifend standardisiert	Kommunikationsmatrix + Trainingsdaten	Lernende Checks
N-5	Correlation	Standardisiert	Kommunikationsmatrix (Systemebene)	Statische Checks
N-6	Protocol	CAN-TP/Diagnose: Standardisiert	CAN-TP- und UDS-Spezifikationen + Kommunikationsmatrix (System-, Bussystem- oder ECU-Ebene)	Statische Checks
		Proprietäre Protokolle: Nicht herstellerübergreifend standardisiert	Kommunikationsmatrix + Trainingsdaten	Lernende Checks
N-7	Plausibility	Nicht herstellerübergreifend standardisiert	Kommunikationsmatrix + Trainingsdaten	Lernende Checks
N-8	Consistency	Nicht herstellerübergreifend standardisiert	Kommunikationsmatrix + Trainingsdaten	Lernende Checks

Tabelle 4.1: Anomalieerkennungssensoren nach Müter et al. [MGF10] und deren Realisierung mittels statischen und lernenden Checks

Grunde liegen, werden im Observer-Konzept mittels statischen Checks realisiert, siehe Spalte *Realisierung* in Tabelle 4.1. Es wird dabei davon ausgegangen, dass die jeweils notwendige Variante der Kommunikationsmatrix während der Entwicklungszeit zur Verfügung steht. Die Überwachung der „Frequenz“ von sporadischen Botschaften und von Hersteller-spezifischen Protokollen durch statische Checks ist aufgrund der in Abschnitt 3.2.2 beschriebenen Randbedingungen Stand heute nicht möglich. Um die aktuell schon komplexe Erstellung der Kommunikationsmatrix nicht zusätzlich zu erschweren, setzt die vorliegende Arbeit für diese Sensoren auf lernende Checks. Gleiches gilt für die Realisierung der Plausibility und Consistency Sensoren, da die notwendigen Informationen weder in standardisierten Spezifikationen noch in der Kommunikationsmatrix enthalten sind.

4.4.2 Priorisierung der lernenden Checks

Statische Checks sind eine Transformation der Spezifikationen in ausführbare Überwachungsregeln und es sind keine weiteren Untersuchungen bezüglich der Realisierbarkeit notwendig. Die Herausforderung besteht hier in einer effizienten Implementierung für Steuergeräte. Die lernenden Checks haben diese Herausforderung ebenfalls. Außerdem kommen weitere hinzu, wie beispielsweise die Auswahl geeigneter Algorithmen und deren Parametrierung. Daher untersucht die vorliegende Arbeit nachfolgend die lernenden Checks, speziell im Hinblick auf eine Realisierung mit ressourcenbeschränkter Steuergerätehardware. Aus folgenden Gründen liegt der Fokus dabei auf dem Plausibility Sensor für physikalische Signale:

1. Sicherheitskritische Botschaften werden im Fahrzeug üblicherweise periodisch übertragen. Daher erhält der Frequency Sensor für sporadische Kommunikation eine niedrigere Priorität bei der Umsetzungsreihenfolge.

2. Aufgrund der Vielfältigkeit der Protokolle auf höheren OSI-Schichten und der Abhängigkeit vom Fahrzeughersteller, wird der Protocol Sensor im Rahmen dieser Arbeit nicht näher betrachtet. Außerdem können Verletzungen von höheren Protokollen teilweise auch von anderen Sensoren detektiert werden. Ein Beispiel ist das zu häufige Versenden von Request-Botschaften, welches der Frequency Sensor erkennen kann.

3. Für die Realisierung von Consistency Sensoren spielt die Korrelation von Signalen eine entscheidende Rolle und Wissen über deren semantische Bedeutung ist vorteilhaft. Ein beispielhafter lernender Check könnte Fahrzeuggeschwindigkeit, Motordrehzahl, Gang und Kupplungsstatus auf ihre Konsistenz zueinander prüfen. Dazu müsste allerdings zuerst bekannt sein, welche Kommunikationssignale die entsprechenden Werte enthalten. Ein erfahrener Anwender

könnte die Signale zwar im Anomaly Detection Configurator markieren, ein geplantes, manuelles Eingreifen wäre jedoch aufwändig und damit unvorteilhaft. Außerdem sind die Kommunikationsmatrix beziehungsweise die darin enthaltenen Signalnamen oftmals kryptisch. Eine Möglichkeit die Selektion zu automatisieren ist die Berechnung der Korrelationsmatrix anhand von Trainingsdaten, gefolgt von der Selektion der am stärksten korrelierenden Signale.
Des Weiteren sind korrelierende Signale potentiell in Botschaften enthalten, die auf unterschiedlichen Netzwerken übertragen werden. Im Gegensatz zu Plausibility Sensoren, lassen sich Consistency Sensoren damit gegebenenfalls nur auf Gateways oder Steuergeräten realisieren, welche mit mehreren Netzwerken verbunden sind. Dies ist auch aus Tabelle 3.1 [MGF10] ersichtlich (*Number of Bus Systems* = *n*).

4. Die Plausibilität eines Signals kann sowohl entscheidende Hinweise auf einen Hackerangriff als auch auf ein funktionales Fehlverhalten liefern. Manipuliert ein Angreifer zum Beispiel das Geschwindigkeitssignal, sodass ein physikalisch unmöglicher Verlauf entsteht (Signalsprünge etc.), sollte ein passender Plausibility Sensor dies erkennen. Andererseits würde ein Sensorausfall während der Fahrt ebenfalls einen nicht plausiblen Sprung im Signalverlauf auslösen, sofern keine Redundanzen im System vorhanden sind.
Analog zum Consistency Sensor besteht hierbei die Herausforderung der Selektion von zu überwachenden Signalen. Ein erster Ansatz ist die Berechnung der Varianz und der Änderungsrate von Signalwerten. Damit lassen sich physikalische Signale anhand von vorhandenen Trainingsdaten größtenteils automatisiert identifizieren.

4.4.3 Signalplausibilisierung in Bezug auf das Angreifermodell

Dieser Abschnitt nimmt Bezug auf die in Abschnitt 3.2.3 eingeführten Angriffsmöglichkeiten und klärt deren Zusammenhang mit dem Observer be-

ziehungsweise mit der Signalplausibilisierung.
Eavesdropping lässt sich bei CAN nur unter Berücksichtigung physikalischer Übertragungseigenschaften erkennen, beispielsweise durch eine geänderte Leistungsaufnahme [Mag17]. Voraussetzung dafür ist ein zusätzlich angeschlossener Busteilnehmer. Ein kompromittiertes Steuergerät, das unberechtigt CAN-Botschaften empfängt, lässt sich auf Basis von Bus- und Kommunikationseigenschaften nicht detektieren. Eavesdropping bei CAN steht nicht im Fokus des aktuellen Observer-Konzepts.
Die Erzeugung von zusätzlicher Kommunikation im Falle von Fabrication-, Spoofing-, Replay- und Flooding-Attacken ist für periodische und für nicht spezifizierte Botschaften durch statische Checks erkennbar. Bei Verwendung einer existierenden, Event-basierten CAN-ID, ist ein erweiterter Frequency Sensor notwendig (siehe Abschnitt 4.4.1).
Selektiver Denial-of-Service durch das Zerstören von Botschaften oder durch das Unterbinden des Sendevorgangs ist für periodische Kommunikation anhand geänderter Zykluszeiten ebenfalls statisch erkennbar. Das Erkennen von zerstörten, Event-basierten Botschaften erfordert wiederum erweiterte Maßnahmen wie zum Beispiel lernende Checks. Zudem können die Fehlerzähler der CAN-Hardware als Informationsquelle dienen.
Die Signalplausibilisierung untersucht Botschaften auf fehlerhaften Inhalt und adressiert somit Masquerade- beziehungsweise Tampering-Attacken sowie die Manipulation auf dem originalen Sendesteuergerät oder während der Weiterleitung in einem Gateway (Man-in-the-Middle). Das betrachtete Angreifermodell geht deshalb davon aus, dass unberechtigter Zugriff auf ein Fahrzeug-internes Bussystem oder ein im Fahrzeug verbautes Steuergerät (Endpunkt oder Gateway) besteht und im letzteren Fall dessen Software manipuliert werden kann. Eine entsprechende Manipulation ist nicht nur mit lokalem Zugang sondern auch über Luftschnittstellen möglich. Dies zeigten Miller und Valasek bereits 2015 [MV15]. Zukünftig sind Software-Aktualisierungen über eine Luftschnittstelle die Regel und müssen entsprechend abgesichert sein, um Missbrauch zu verhindern.

4.5 Lernende Checks zur Signalplausibilisierung

Für die Anomalieerkennung in Signalverläufen beziehungsweise in Daten mit zeitlichem Verlauf, eignen sich prinzipiell viele verschiedene Algorithmen des maschinellen Lernens. Unter Berücksichtigung der folgenden Randbedingungen, lässt sich die Auswahl deutlich eingrenzen:

- Es stehen lediglich normale Trainingsdaten zur Verfügung, die nicht weiter unterteilt sind. Der ausgewählte Algorithmus muss daher in der Lage sein, ein Modell für das Normalverhalten semi-überwacht zu erlernen und abweichendes Verhalten als Anomalie zu erkennen (siehe Ein-Klassen-Klassifizierung in Unterkapitel 2.3). Dadurch scheiden klassische Klassifikations-Algorithmen aus, welche Trainingsdaten aller möglichen Klassen benötigen und überwacht lernen. Selbst wenn annotierte Anomaliedaten vorhanden wären, würde ein damit trainierter Klassifikator wahrscheinlich nur die bereits bekannten Anomalien erkennen. Dies widerspricht dem Grundgedanken der Anomalieerkennung, alle möglichen Abweichungen vom Normalverhalten zu detektieren [Req3].

- Da die Inferenz in *Echtzeit* erfolgen muss [Req5], dürfen die Algorithmen nicht auf abgeschlossene Datensätze angewiesen sein.

- Neben der prinzipiellen Funktionsweise des Algorithmus ist der *Ressourcenverbrauch* entscheidend für eine mögliche Anwendung als lernender Check [Req8]. Steuergeräte sind kostensensible Bauteile und daher stark ressourcenbegrenzt. Zusätzlich zum Kostendruck tragen der zur Verfügung stehende Bauraum innerhalb eines Fahrzeugs und die teilweise vorherrschenden Umgebungsbedingungen dazu bei, dass die Rechenleistung eines durchschnittlichen Steuergeräts weit unterhalb eines modernen Mobiltelefons liegt. Ein automotive Mikrocontroller der oberen Mittelklasse ist mit 100-300 MHz getaktet,

hat mehrere hundert Kilobyte bis wenige Megabyte Random Access Memory (RAM) und einige Megabyte Read-only Memory (ROM). Ausnahmen sind die in Abschnitt 2.1.3 vorgestellten Domänen- und Zentralrechner, sowie Steuergeräte im Infotainment und Fahrerassistenzbereich. Hier kommen deutlich leistungsfähigere Systeme zum Einsatz. Der Observer soll auf den meisten Steuergeräten integrierbar sein. Daher wird besonderer Wert darauf gelegt, dass die implementierten Algorithmen wenig Speicher (vor allem RAM) benötigen. Ebenso ist auf eine möglichst kurze Berechnungszeit zu achten. Diese Randbedingungen gelten nur für die Inferenz auf dem Steuergerät. Da das Training zunächst auf einem Arbeitsplatzrechner oder Server-System stattfindet, kann dieser Vorgang speicher- und rechenintensiv ausfallen. Für fertig trainierte Algorithmen kann der Steuergeräte-Quellcode auf Basis der Anomaly Detection Configuration generiert werden.

Die Ressourcenbeschränkungen schließen nachbarschaftsbasierte Clustering-Verfahren aus, welche zur Bewertung eines neuen Eingangsvektors dessen Abstand zu allen gespeicherten Datenpunkten berechnen. Zum einen besitzen Steuergeräte zu wenig Speicherplatz, um alle Trainingsdaten für die Inferenz vorzuhalten, zum anderen ist die große Anzahl an Abstandsberechnungen rechenintensiv. Clustering-Verfahren, die mit Cluster-Zentren arbeiten, weisen diese Nachteile nicht auf, wurden im Rahmen der vorliegenden Arbeit jedoch nicht näher betrachtet.

- Obwohl das Training aktuell auf einem Arbeitsplatzrechner oder auf einem Server stattfindet, sollen die implementierten Algorithmen *Online Learning* unterstützen. Dies ist momentan aufgrund der Ressourcenbeschränkungen zwar noch nicht umsetzbar, könnte in Zukunft aber notwendig sein. Ein mögliches Szenario ist die Absicherung von selbstlernenden Algorithmen, wie sie voraussichtlich beim hochau-

tomatisierten Fahren zum Einsatz kommen werden. Entwickeln sich Algorithmen und damit potentiell auch die von ihnen versendeten Signale zur Laufzeit weiter, muss sich die Überwachung gleichermaßen eigenständig anpassen. Zusätzlich kann eine fortlaufende Anpassung auch bei Alterungseffekten nützlich sein. Allerdings ist dann ebenfalls zu berücksichtigen, dass fortgeschrittene Angreifer die selbstlernende Eigenschaft der Erkennungsmechanismen ausnutzen können, um ein System langsam und unentdeckt zu kompromittieren.

4.5.1 Bewertung der One-Class Support Vector Machine

Einer der bekanntesten Algorithmen zur Anomalieerkennung ist die in Abschnitt 2.3.3 beschriebene One-Class Support Vector Machine. Ihre Funktionsweise erfüllt die Voraussetzung der One-Class Classification. Den Separator im Training zu berechnen ist zwar rechenintensiv, die Inferenz - d.h. die Klassifizierung von neuen Datenpunkten - ist hingegen vergleichsweise einfach. Es genügt die Position des neuen Datenpunkts relativ zum Separator zu berechnen, um eine Klassifizierung in normal oder anormal vorzunehmen. Obwohl dafür einige komplexere mathematische Operationen notwendig sind, wird davon ausgegangen, dass dies in Echtzeit auf einem Steuergerät möglich ist. Damit sind auch die Randbedingungen bezüglich Echtzeitfähigkeit und Ressourcenverbrauch erfüllt. Im Gegensatz dazu ist Online Learning nicht darstellbar. Der Separator berechnet sich mathematisch eindeutig auf Basis aller Trainingsdatenpunkte sowie der gewählten Hyperparameter σ und ν. Eine Anpassung zur Laufzeit ist nicht möglich, da ansonsten alle Datenpunkte auf einem Steuergerät zur Verfügung stehen müssten. Zusätzlich würde die Berechnung eines neuen Separators viel Rechenleistung beziehungsweise Zeit beanspruchen. Ein weiterer negativer Punkt ist die Bestimmung von σ und ν, deren Werte maßgeblichen Einfluss auf das Klassifikationsergebnis haben und schwer zu ermitteln sind.

Aus den genannten Gründen findet die OCSVM im Rahmen des Observers keine Anwendung.

4.5.2 Bewertung von Ensemble-basierten Algorithmen

LODA ist ein Ensemble-basierter Anomalieerkennungsalgorithmus und arbeitet nach dem One-Class Classification Prinzip, siehe Abschnitt 2.3.2. Bei seiner Entwicklung stand eine geringe Laufzeit- und Speicherkomplexität im Vordergrund. Zudem weist LODA einige weitere Vorteile auf:

- Die Laufzeit ist deterministisch und wächst sub-linear bezogen auf die Merkmalsanzahl.
- Wie HS-Trees unterstützt LODA Online Learning.
- Bei LODA besteht das Trainingsergebnis aus den erstellten Histogrammen. Für die Bestimmung der Histogramm- und Klassenanzahl kommen entsprechende Optimierungsalgorithmen zum Einsatz [Pev16] [BR06]. Der zur finalen Klassifizierung notwendige Schwellwert lässt sich ebenfalls automatisch ermitteln, siehe Abschnitt 5.6.1. Damit ist die Größe des Eingangsvektors der einzige Hyperparameter, der vor Trainingsbeginn festzulegen ist.
- Beim Auftreten einer Anomalie können Rückschlüsse auf das oder die verursachenden Merkmal(e) gezogen werden.
- Im Gegensatz zu vielen anderen Algorithmen des maschinellen Lernens, kann LODA mit Eingangswerten umgehen, die außerhalb des normalisierten Bereichs liegen. Dies kann beispielsweise vorteilhaft sein, wenn sich der zulässige Wertebereich im Laufe der Lebenszeit eines Fahrzeugs ändert.

In seiner Veröffentlichung erwähnt Pevný, dass LODA speziell für Datenströme entworfen ist und für Beispiele aus diesem Anwendungsgebiet im

Durchschnitt sieben bis acht Mal schneller arbeitet als HS-Trees - bei näherungsweise der gleichen Detektionsleistung [Pev16]. Durch die genannten Vorteile ist LODA der vielversprechendste Ensemble-basierte Algorithmus. Er erfüllt alle Randbedingungen des vorliegenden Anwendungsfalls und wird daher weiter untersucht.

4.5.3 Bewertung von neuronalen Netzen

Rekurrente Netze inklusive LSTMs sind durch ihre Gedächtniseigenschaft prädestiniert für die Analyse von Zeitreihen, wie der erfolgreiche Einsatz in verschiedenen Anwendungen zeigt, siehe Abschnitt 2.3.1. In Abschnitt 3.2.2 wurden drei LSTM-basierte Ansätze zur Anomalieerkennung in CAN-Kommunikation vorgestellt. Die von den ersten beiden Konzepten erreichte Kombination aus TPR und FPR wird für die angestrebte Anwendung im Rahmen des Observers allerdings als nicht ausreichend eingestuft. Das Ziel ist eine FPR von 0% bei einer möglichst hohen TPR, wobei die von Taylor et al. erreichten 44,5% als zu niedrig angesehen werden [TLJ16]. Der Ansatz von Suda et al. scheidet aufgrund des überwachten Lernens aus [SNH18] (keine One-Class Classification, siehe auch [Req3]). Außerdem sind LSTM-Netze im Vergleich zu klassischen Autoencodern deutlich komplexer aufgebaut. Sie bestehen in der Regel aus mehreren individuellen Gewichtsmatrizen und nichtlinearen Funktionen. In Anbetracht der Ressourcenbeschränkung auf automotive Steuergeräten, setzt die vorliegende Arbeit - neben LODA - auf klassische Autoencoder zur Signalplausibilisierung. Das in der Kommunikationsmatrix enthaltene Wissen (zum Beispiel über die transportierten Signale) soll dabei den Wegfall des komplexeren Modells kompensieren und den Einsatz von einfacheren Netzen unterstützen beziehungsweise ermöglichen. Der prinzipielle Aufbau von Autoencodern sowie deren Funktionsweise sind in Abschnitt 2.3.1 beschrieben. Sie realisieren eine One-Class Classification und erfüllen auch die restlichen Randbedingungen aus Unterkapitel 4.5. Die Inferenz besteht aus einer Vorwärtsrechnung und

setzt sich aus mehreren Multiplikationen und Additionen, der Berechnung der Aktivierungsfunktionen sowie der abschließenden Fehlerbewertung zusammen. Daher ist die Berechnung kleiner Netze auf Steuergeräten in Echtzeit möglich. Eine ausführliche Diskussion der Hyperparameter und deren Einfluss auf die Anwendbarkeit folgt in Abschnitt 5.6.2 und in Unterkapitel 7.2. Das Training ist mathematisch aufwändiger und kann längere Zeit in Anspruch nehmen, weshalb dafür ein Arbeitsplatzrechner verwendet wird. Die Funktionsweise an sich erlaubt Online Learning, d.h. Autoencoder lassen sich in Zukunft - entsprechend leistungsstarke Hardware vorausgesetzt - auf dem Zielsystem trainieren. Auch eine Kombination der Ansätze ist möglich: Nach der initialen Parametrierung auf einem Arbeitsplatzrechner lässt sich das Training auf dem Zielsystem zur Laufzeit fortsetzen, um auf veränderte Umgebungsbedingungen zu reagieren.

4.5.4 Sliding Window zur Zeitreihenbildung

Weder LODA noch klassische Autoencoder speichern interne Zustände während der Inferenz. Daher können sie den Bezug aufeinander folgender Eingangsvektoren nicht herstellen und der zeitliche Verlauf eines Signals muss im Eingangsvektor selbst enthalten sein. Dessen Bildung erfolgt nach dem Sliding Window Prinzip. In Bezug auf die von Chandola et al. definierten Anomalie-Klassen (siehe Unterkapitel 2.3) wird damit eine kollektive Anomalie in eine Punktanomalie überführt.
Unter der Annahme, dass jeweils eine Algorithmusinstanz für die Plausibilisierung eines einzelnen Signals zuständig ist, entspricht die Fenstergröße des Sliding Windows der Merkmalsanzahl. Bild 4.5 ist [VWeb+18b] entnommen und zeigt den prinzipiellen Aufbau mit einer Fenstergröße von vier. Die einzelnen Elemente beziehungsweise Merkmale sind beispielhaft mit den Eingangsneuronen eines Autoencoders verbunden.

Jeden empfangenen oder gesendeten Signalwert übergeben die statischen Checks der Merkmalsgenerierung (siehe Bild 4.4). Diese führt zunächst ei-

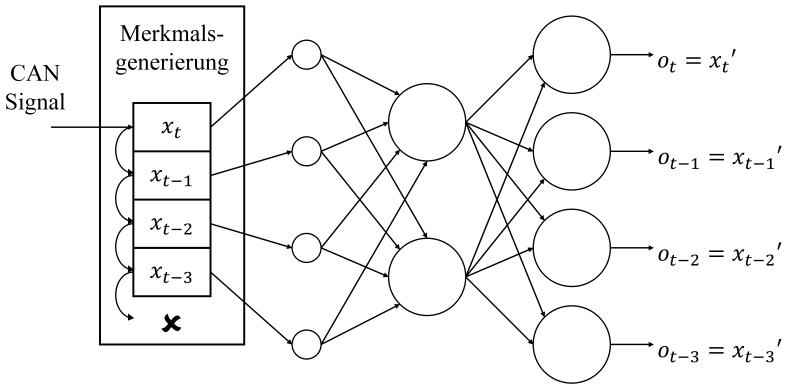

Bild 4.5: Sliding Window zur Zeitreihenbildung [VWeb+18b]

ne Normalisierung auf einen Wertebereich zwischen 0 und 1 durch. Viele Algorithmen des maschinellen Lernens benötigen normalisierte Eingangsmerkmale für deren erfolgreiche Anwendung. Danach erstellt und verwaltet die Merkmalsgenerierung eine Zeitreihe je betrachtetes Signal. In Bild 4.5 entspricht x_t dem aktuellen und x_{t-1} bis x_{t-3} den vergangenen Werten. Bei einer Aktualisierung wird x_{t-3} verworfen, x_t bis x_{t-2} jeweils um ein Element verschoben und der neue, normalisierte Signalwert als x_t gesetzt [VWeb+18b]. Sobald dieser Vorgang abgeschlossen ist, kann der nachgelagerte Algorithmus mit dem entsprechenden Eingangsvektor berechnet werden.

5 Signalplausibilisierung mit Realdaten

Das in Kapitel 4 vorgestellte Konzept wird weiter verfolgt und detailliert. Dieses Kapitel legt dabei den Fokus auf die Anwendung der ausgewählten, selbstlernenden Algorithmen zur Signalplausibilisierung und damit auf die lernenden Checks des Observers, siehe Bild 5.1.

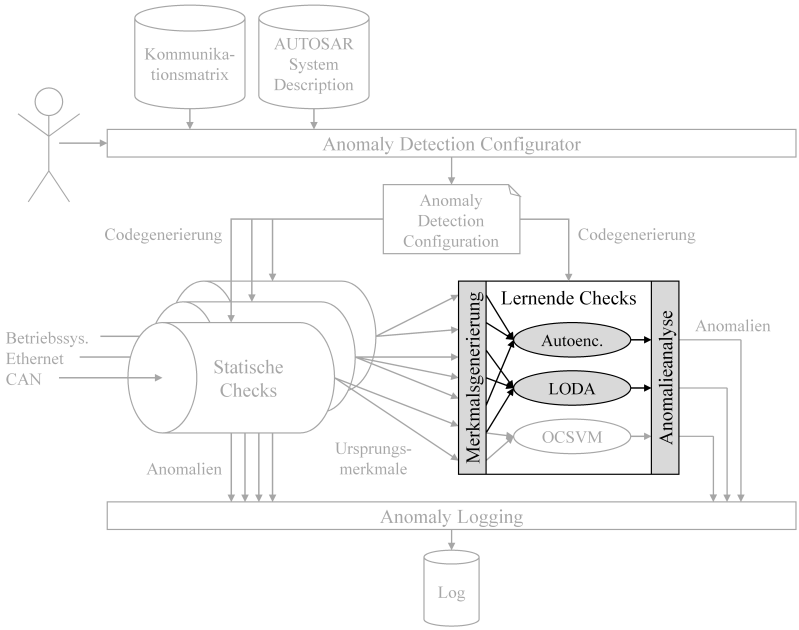

Bild 5.1: Fokus auf die Signalplausibilisierung mittels lernenden Checks

Ergänzend zu Bild 5.1 zeigt Bild 5.2 das Ablaufdiagramm zum definierten Vorgehen. Die Nummerierung der einzelnen Schritte entspricht den folgenden Unterkapiteln. Da Kapitel 5 und Kapitel 6 eng miteinander verbunden sind, enthält Bild 5.2 auch bereits die in Kapitel 6 behandelte Evaluierung. Das prinzipielle Vorgehen wurde in [VWeb+] vorgestellt und im weiteren Verlauf der vorliegenden Arbeit verfeinert. Daher sind große Teile dieses Kapitels [VWeb+] entnommen beziehungsweise daran angelehnt.

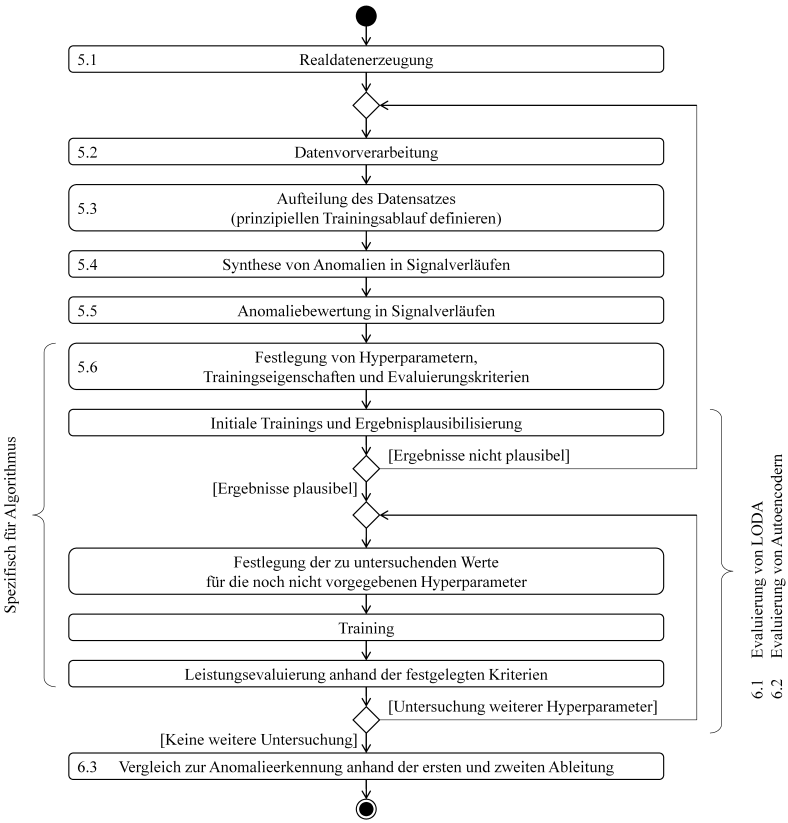

Bild 5.2: Vorgehen bei der Implementierung und Evaluierung der Signalplausibilisierung

5.1 Realdatenerzeugung

Die Evaluierung der untersuchten Algorithmen soll auf Basis von realen Fahrzeugdaten erfolgen. Für deren Erzeugung wurden zunächst folgende zwei Vorgehensweisen betrachtet:

1. *Aufzeichnen von CAN-Kommunikation*: Die erste Möglichkeit ist das Mithören und Aufzeichnen der Kommunikationsbotschaften, welche Steuergeräte über Fahrzeug-interne CAN-Busse versenden. Da der Observer auf einem Steuergerät ebenfalls auf Basis von empfangenen und gesendeten CAN-Botschaften arbeitet, sind die so erzeugten Daten nahe an der Realität. Sie bilden beispielsweise die Periode zwischen einzelnen Botschaften und Signalauflösungen exakt ab. Die Untersuchungen würden damit das reale Anwendungsszenario widerspiegeln. Aus wissenschaftlicher Sicht ist dieses Vorgehen die gewünschte Art der Datenerzeugung. Auf technischer Seite existieren allerdings Randbedingungen. Zum einen muss der Zugang zu Fahrzeug-internen CAN-Bussen gewährleistet sein. Aus Diagnosegründen besitzen zwar alle Fahrzeuge einen CAN-Zugang, bei modernen E/E-Architekturen ist dieser aber aus Sicherheitsgründen durch ein Gateway isoliert, siehe Abschnitt 2.1.3. Im Regelbetrieb sind dadurch keine Fahrzeug-internen Botschaften auf dem sogenannten Diagnose-CAN sichtbar. Zum Mithören und Aufzeichnen muss daher ein separater, mechanischer und elektrischer Zugang geschaffen werden. Zum anderen ist die Interpretation der aufgezeichneten CAN-Botschaften eine Herausforderung, selbst wenn ein solcher Zugang existiert. Die Dekodierung der übertragenen Nutzdaten in einzelne Signale (siehe Daten-Feld in Bild 2.8) ist nur mit Hilfe der Kommunikationsmatrix oder durch intensives Reverse-Engineering möglich.

2. *Datenaufzeichnung mittels OBD-II*: Wie bereits erwähnt, besitzt jedes Fahrzeug einen CAN-Zugang für Diagnoseanwendungen. Dieser ist

im On-Board Diagnostics-II (OBD-II) Standard spezifiziert[1]. OBD-II erfordert die Überwachung von abgasrelevanten Systemkomponenten zur Laufzeit und das Abspeichern von erkannten Fehlfunktionen zur späteren Auswertung [Cal16]. Zusätzlich soll ein externes Gerät in der Lage sein, den abgasrelevanten Fahrzeugstatus auszulesen. Hierfür wurde ein standardisierter Steckverbinder unter gleichem Namen eingeführt. Seit einigen Jahren existieren sogenannte OBD-II Dongles, die Endanwendern den Zugriff auf diese Schnittstelle erlauben - beispielsweise über eine Smartphone-App[2]. Darüber ist es unter anderem möglich, abgasrelevante Signale, wie beispielsweise Fahrzeuggeschwindigkeit, Motordrehzahl und verschiedene Temperaturen, im Request/Response-Verfahren abzufragen. Für diese Art der Datenerzeugung sind keine Fahrzeugeingriffe notwendig. Aufgrund des standardisierten Formats der auslesbaren Signale, wird die Kommunikationsmatrix ebenfalls nicht benötigt. Nachteilig ist hingegen, dass das Format der so erzeugten beziehungsweise aufgezeichneten Daten nicht dem der Fahrzeug-internen Kommunikation entspricht. Speziell die Periodendauer zwischen zwei Signalaktualisierungen ist deutlich länger und kann bei OBD-II erheblich schwanken, während zyklische Botschaften auf Fahrzeug-internen Bussystemen meistens mit konstanter Periode von einigen Millisekunden versendet werden. Die Schwankungen sind abhängig von der aktuellen Auslastung des Gateways - falls vorhanden, der betroffenen Bussysteme und des Steuergeräts, welches die Anfrage prozessiert.

[1] OBD-II wurde zuerst in Kalifornien eingeführt und ist dort seit 1996 für Benzinfahrzeuge gültig. 1997 folgten die Dieselfahrzeuge. In Europa ist das OBD-II Pendant seit 2001 für Benzin- und seit 2004 für Dieselfahrzeuge vorgeschrieben.

[2] OBD-II Dongles stellen inzwischen ein großes Sicherheitsrisiko dar, da sie zusätzliche Angriffsfläche für Hacker bieten [Arg14] [Gre15] [Kov17] und Security meistens nicht im Fokus bei deren Entwicklung stand. Das Sicherheitsrisiko steigt nochmals, wenn die OBD-II Buchse direkten Zugriff auf Fahrzeug-interne Bussysteme erlaubt.

Fahrzeughersteller halten die Kommunikationsmatrix streng geheim, sodass diese zur Dekodierung von CAN-Botschaften nicht zur Verfügung stand. Deshalb wurden Fahrzeugsignale über die OBD-II Schnittstelle aufgezeichnet. Zum Einsatz kam dabei das OBD-II Dongle *KIWI 3* von PLX Devices und die Smartphone App *OBD Auto Doctor* von Creosys auf unterschiedlichen iOS-basierten Geräten. Mit der Smartphone App wurden die folgenden zehn Sensorsignale zyklisch abgefragt und je Fahrt in einer Komma-separierten (engl.: comma-separated values (CSV)) Datei abgespeichert[3]: Temperatur der Kühlflüssigkeit, Ansaugdruck, Motordrehzahl, Fahrzeuggeschwindigkeit, Ansaugtemperatur, Luftdurchsatz, Drosselklappenstellung, Umgebungstemperatur sowie Gaspedalstellung „D“ und „E“[4].

Time	Engine Coolant Temperature [°C]	Intake Manifold Absolute Pressure [kPa]	Engine RPM [RPM]	Vehicle Speed Sensor [km/h]
16:54:58.117	11			
16:54:58.207	11	102		
16:54:58.297	11	102	1041	
16:54:58.387	11	102	1041	0
16:54:58.448	11	102	1041	0
16:54:58.537	11	102	1041	0
16:54:58.598	11	102	1041	0
16:54:58.688	11	102	1041	0
16:54:58.779	11	102	1041	0
16:54:58.867	11	102	1041	0
16:54:58.957	11	102	1041	0
16:54:59.047	11	102	1041	0
16:54:59.137	11	102	1042	0
16:54:59.198	11	102	1042	0

Bild 5.3: Aufgezeichnete Sensorwerte mittels OBD-II [VWeb+]

Bild 5.3 zeigt einen Ausschnitt aus einer gespeicherten CSV-Datei. Die zehn konfigurierten Sensorsignale werden im Round-Robin-Verfahren abgefragt, wobei in Bild 5.3 nur die ersten vier dargestellt sind. Für jeden aktualisierten Wert enthält die Datei eine separate Zeile, d.h. ein Sensorwert wird jede zehnte Zeile aktualisiert (siehe Markierungen in Bild 5.3). Aufgrund der diskutierten Abhängigkeit von Bus- und Steuergeräteauslastung,

[3] Die initiale Abspeicherung erfolgt in XLSX-Dateien, welche zur einfacheren Verarbeitung ins CSV-Format konvertiert wurden.
[4] Gaspedalstellung „D“ und „E“ sind zwei separate Signale und repräsentieren eine redundante Datenerfassung

benötigt eine Werteabfrage zwischen 60 und 90 ms. Damit liegt die Aktualisierungsperiode eines Signals zwischen 600 und 900 ms.

5.2 Datenvorverarbeitung

Damit die mittels OBD-II erzeugten Daten zum Training sowie für die Validierung und den Test von selbstlernenden Algorithmen (siehe Unterkapitel 5.3) verwendet werden können, muss zunächst eine drei-stufige Vorverarbeitung stattfinden, siehe Bild 5.4.

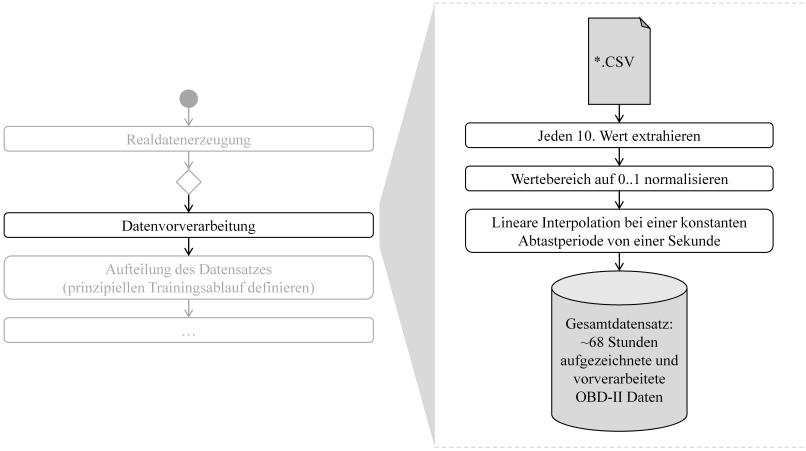

Bild 5.4: Vorverarbeitungschritte für die OBD-II Daten

Als erstes sind die relevanten Werte aus einer CSV-Spalte zu extrahieren. Für das Drehzahlsignal stehen die aktualisierten Werte zum Beispiel in den Zeilen vier, 14, 24, usw. Danach folgt die Normalisierung auf einen Wertebereich zwischen null und eins. Minimaler und maximaler Signalwert sind in der Regel in der Kommunikationsmatrix enthalten. Da diese nicht zur Verfügung steht, findet die Normalisierung anhand aufgezeichneter Signalwerte statt. Der aufgezeichnete Maximalwert entspricht dem normalisierten Wert eins, der aufgezeichnete Minimalwert dem normalisierten Wert null.

Im letzten Schritt wird ein äquidistanter Abstand zwischen den einzelnen Signalwerten von einer Sekunde mittels linearer Interpolation hergestellt. Dies ist notwendig, da der zeitliche Abstand zwischen zwei Datenpunkten dem Algorithmus nicht als Merkmal übergeben wird. Im verwendeten Sliding Window Ansatz besteht der Eingangsvektor lediglich aus einer Folge von Signalwerten, siehe Bild 4.5. Theoretisch könnte man die zeitlichen Abstände als zusätzliche Merkmale zur Verfügung stellen, was allerdings die Größe des Eingangsvektors verdoppeln und damit auch den Ressourcenverbrauch deutlich erhöhen würde.

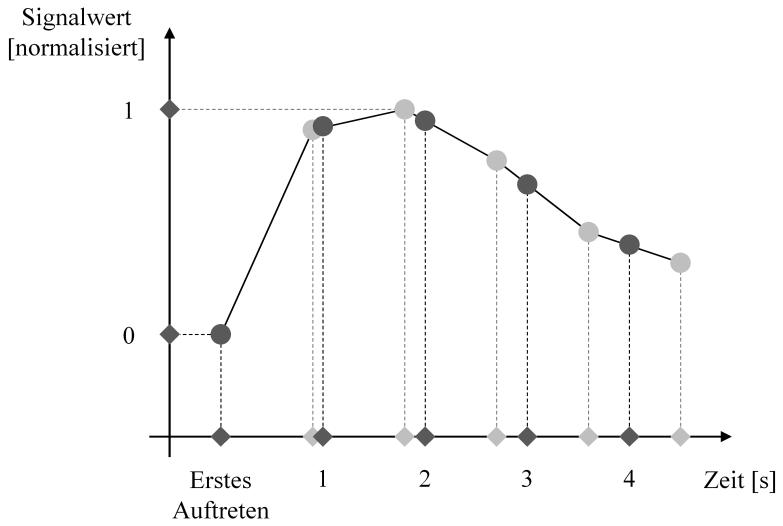

Bild 5.5: Vorverarbeitung der aufgezeichneten Signalwerte [VWeb+]

Bild 5.5 zeigt die Normalisierung des Wertebereichs sowie die lineare Interpolation zur Anpassung der Abtastrate anhand eines beispielhaften Signalverlaufs. Die hellen Punkte repräsentieren die aufgezeichneten, die dunklen die interpolierten und normalisierten Signalwerte.

5.3 Aufteilung des Datensatzes

Insgesamt wurden ca. 80 Fahrten mit einer Gesamtfahrzeit von ca. 68 Stunden in einem Seat Leon Kombi 5F 2.0 TDI (150 PS) aufgezeichnet. Dies entspricht ca. 245.000 normalisierten Datenpunkten. Am Steuer saß dabei immer der gleiche Fahrer. Der gesamte Datensatz steht öffentlich zur Verfügung [VWeb18] und enthält ca. 60% Autobahn-, 15% Überland-Abschnitte sowie 25% städtische Anteile. Eine spezielle Fahrt enthält einen Abschnitt auf verschneiter Fahrbahn mit durchdrehenden Rädern. Die dazugehörige Datei ist mit einem entsprechenden Namen gekennzeichnet und repräsentiert kein Normalverhalten. Alle anderen Aufzeichnungen werden als *Normaldaten* interpretiert.

Für eine aussagekräftige Bewertung von trainierten Algorithmen sind disjunkte Trainings- und Testdaten notwendig. Die Evaluierung hat auf Basis von neuen, dem Algorithmus unbekannten Daten zu erfolgen. Findet die Bewertung anhand von bekannten Trainingsdaten statt, ist keine Aussage über die Generalisierungsfähigkeit des Algorithmus möglich und geschönte Ergebnisse wären die Folge. Außerdem wäre eine Überanpassung - d.h. ein Auswendiglernen der Trainingsdaten - mit diesem Vorgehen nicht erkennbar.

Durch die folgenden drei Eigenschaften von vielen selbstlernenden Algorithmen werden häufig mehrere Algorithmus-Instanzen für die gleiche Problemstellung trainiert und nachfolgend die *leistungsstärkste*[5] ausgewählt. Bild 5.6 gibt eine grafische Zusammenfassung.

1. Viele Algorithmen besitzen Hyperparameter, die vor Trainingsbeginn festzulegen sind. Erste Richtwerte lassen sich über Erfahrungswerte oder Machbarkeitsstudien bestimmen. Auf Basis einer Leistungsbewertung kann allerdings eine entsprechende Anpassung beziehungsweise Feinabstimmung erfolgen.

[5] Auf die notwendigen Bewertungskriterien zur Bestimmung des leistungsstärksten Algorithmus wird in Unterkapitel 5.5 und in Kapitel 6 näher eingegangen.

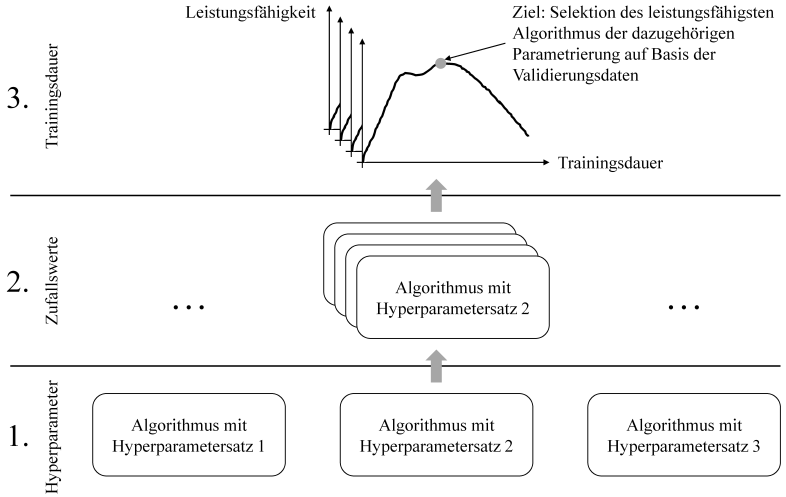

Bild 5.6: Auswahl des leistungsstärksten Algorithmus in Bezug auf die Hyperparameter, die zufälligen Anteile und die Trainingsdauer

2. Viele Algorithmen basieren (zumindest teilweise) auf Zufallswerten. Bei LODA sind es beispielsweise die zufälligen Projektionsvektoren und bei Autoencodern - sowie bei neuronalen Netzen allgemein - sind es die zufällig initialisierten Kantengewichte. Daher resultiert das Training mehrerer Instanzen des gleichen Algorithmus üblicherweise in unterschiedlichen Ergebnissen. Um diesen Effekt zu berücksichtigen, können mehrere Instanzen unter Verwendung der gleichen Hyperparameter trainiert und nachfolgend die leistungsstärkste daraus ausgewählt werden.

3. Während des Trainings einer Algorithmus-Instanz ist zu entscheiden, wann die höchste allgemeine Leistungsfähigkeit erreicht ist. Dauert das Training zu lange, besteht eine erhöhte Wahrscheinlichkeit der Überanpassung und die Generalisierungsfähigkeit nimmt gegebenenfalls wieder ab.

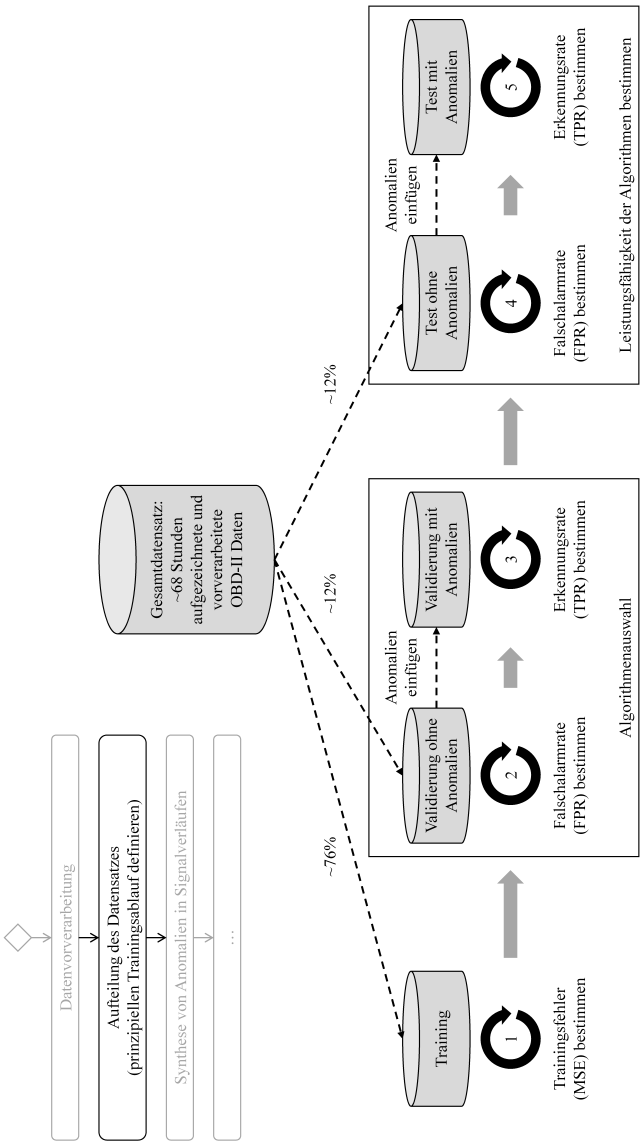

Bild 5.7: Aufteilung des Gesamtdatensatzes in fünf Teile [VWeb+]

Wird die Auswahl anhand der zuvor diskutierten Testdaten getroffen, ist lediglich sichergestellt, dass es sich um den leistungsstärksten Algorithmus in Bezug auf diese Testdaten handelt. Möglicherweise ist die Leistungsfähigkeit auf anderen, unbekannten Daten geringer. Daher erfolgt eine Aufteilung des Gesamtdatensatzes in Trainings-, Validierungs- und Testdaten. Die Verwendung der Trainingsdaten bleibt unverändert. Auf Basis der Validierungsdaten findet eine erste Bewertung der Leistungsfähigkeit und die darauf aufbauende Selektion des Algorithmus statt. Unabhängig davon wird die finale Leistungsfähigkeit anhand der noch ungesehenen Testdaten ermittelt. Somit hat die Auswahl keinen direkten Einfluss auf das Testergebnis.
In Bild 5.7 ist die Aufteilung in Trainings-, Validierungs- und Testdaten dargestellt, wobei es von den letzten beiden Teildatensätzen jeweils zwei Varianten gibt. Der Teildatensatz *Training* besteht aus ca. 76% aller Aufzeichnungen und enthält implizit ausschließlich Normaldaten. Die Validierungsdaten setzen sich aus ca. 12% der Aufzeichnungen zusammen. Für die Variante *Validierung ohne Anomalien* bleiben die aufgezeichneten Daten unverändert. Im Gegensatz dazu wurden die (gleichen) Daten in der Variante *Validierung mit Anomalien* mit synthetischen Anomalien angereichert. Eine ausführliche Diskussion bezüglich der Anomalien folgt im nächsten Unterkapitel. Analog zu den Validierungsdaten gibt es die Testdaten (ebenfalls ca. 12%) in den Varianten *Test ohne Anomalien* und *Test mit Anomalien*. Bild 5.7 enthält bereits die im Rahmen der vorliegenden Arbeit verwendeten Bewertungskriterien (Mean Squared Error (MSE), FPR und TPR). Auf diese wird in Unterkapitel 5.5 näher eingegangen.

5.4 Synthese von Anomalien in Signalverläufen

Bevor eine Bewertung der Leistungsfähigkeit von Algorithmen erfolgen kann, müssen zunächst Signalverläufe mit Anomalien angereichert werden. Dazu wurden 13 Anomalie-Typen definiert, welche nachfolgend erklärt und in Bild 5.8 exemplarisch dargestellt sind. Sieben davon wurden bereits in

[VWeb+] vorgestellt und nachträglich um sechs weitere ergänzt. Da jeder mögliche Signalverlauf das Ergebnis eines Angriffs oder eines Software-Fehlers sein kann, orientieren sich die Definitionen hauptsächlich an potentiellen Hardware-Fehlern im E/E-System. Einen guten Überblick dazu gibt der Standard ISO 26262-5:2011 [Int11a], welcher die funktionale Sicherheit im Rahmen der automotive Hardware-Entwicklung betrachtet. In dessen Annex D sind die zu betrachtenden Fehlerfälle für unterschiedliche Komponenten des E/E-Systems aufgelistet. Für den vorliegenden Anwendungsfall sind besonders Fehler von Sensoren relevant, da deren Messgrößen oftmals als Kommunikationssignale zur Verfügung stehen. Allerdings gibt es auch Fehler in anderen E/E-Komponenten - wie beispielsweise der Spannungsversorgung, die Auswirkung auf Sensorwerte und Kommunikationssignale haben können. Eine genauere Betrachtung erfolgt im Rahmen der Typ-Definitionen.

Jeder Anomalie-Typ besitzt Konfigurationsparameter. Im Rahmen dieser festzulegenden Grenzen, entstehen bei der Synthetisierung zufällige Anomalien. Die ermittelten Ergebnisse haben damit eine allgemeinere Aussagekraft. Wird hingegen die selbe Anomalie mehrfach in einen Signalverlauf eingefügt, nimmt die allgemeine Aussagekraft ab.

Bei der Parametrierung der einzelnen Typen ist darauf zu achten, dass erkennbare Anomalien entstehen. Sind die Einstellungen zu konservativ, resultiert die Synthetisierung potentiell in alternativen Signalverläufen, die weiterhin gültig sind. Im Gegensatz dazu erzeugt eine zu progressive Einstellung gegebenenfalls Anomalien, deren Erkennung trivial ist. In beiden Fällen wäre das Evaluationsergebnis verzerrt. Aus diesem Grund wurden die Trainingsdaten auf charakteristische Eigenschaften untersucht, die bei der Parametrierung der jeweiligen Anomalie-Typen helfen. Die Konfiguration ist so gewählt, dass auch Anomalien an der Grenze zum Normalverhalten entstehen. Dabei muss der gültige Wertebereich zwischen null und eins stets beibehalten werden. Außerdem bleibt der erste und letzte Datenpunkt

im anormalen Signalausschnitt unverändert, um eine einheitliche Aus- und Bewertung zu ermöglichen.

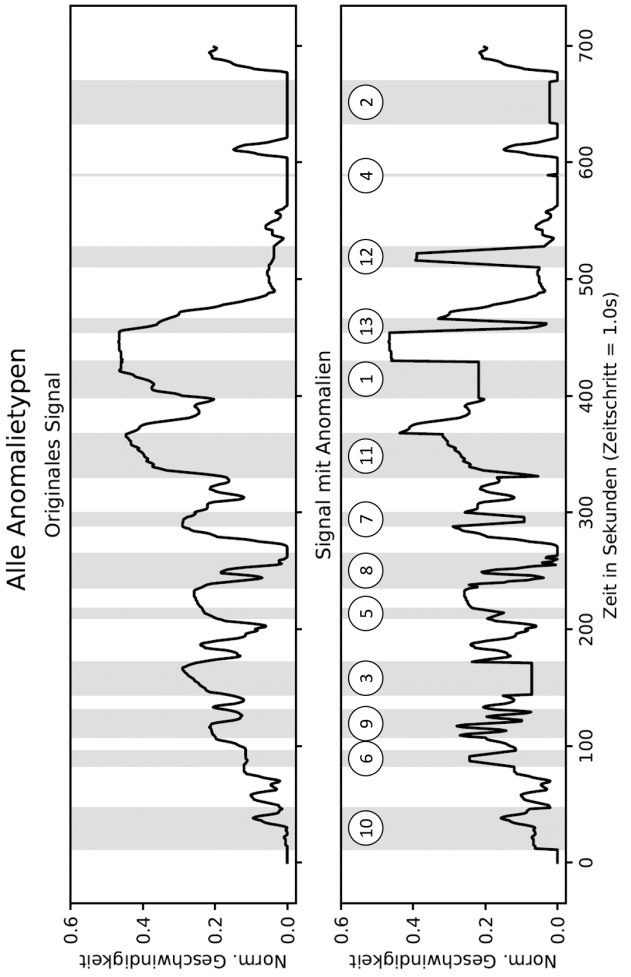

Bild 5.8: Ein Beispiel für alle 13 Anomalie-Typen

1. Eine Anomalie des Typs *Plateau* friert einen Signalwert für eine konfigurierbare Dauer ein. Auslöser kann beispielsweise ein festsitzender Sensor sein. Innerhalb der ISO 26262-5:2011 entspricht das beschriebene Verhalten dem Sensor-Fehler „Stuck in range“ [Int11a]. In der vorliegenden Arbeit dauert eine Anomalie dieses Typs zwischen zwölf und 40 Signalaktualisierungen.

2. Eine *Positive Step Plateau* Anomalie besteht aus einem positiven Wertesprung gefolgt von einem konstanten Signalwert. Im vorliegenden Fall beträgt die Dauer wiederum zwischen zwölf und 40 Signalaktualisierungen. Der Wertesprung ist abhängig von den Trainingsdaten und liegt zwischen dem 0,7- und 2-Fachen der maximalen Signaländerung, auf welche ein nahezu konstanter Wert folgt. Im Extremfall springt das Signal auf den minimalen oder maximalen Wert, beispielsweise durch einen Kurzschluss der Sensorleitung gegen Masse oder Versorgungsspannung. Der ISO 26262-5:2011 Standard listet diese Fehlerfälle als „Short Circuit to Ground“ und „Short Circuit to Vbat“ für die Verkabelung beziehungsweise Steckverbinder [Int11a].

3. *Negative Step Plateau* ist das Pendant zu Positive Step Plateau mit dem Unterschied, dass eine Anomalie mit einem negativen Wertesprung beginnt. Die Parametrierung ist analog.

4. Eine *Positive Peak* Anomalie hat die Form eines Dreiecks. Die *Spitzigkeit*, berechnet durch die Steigung auf der linken minus der Steigung auf der rechten Dreiecksseite, und die Dauer sind konfigurierbar. Für diese Arbeit wurde die maximale Spitzigkeit der Trainingsdaten ermittelt. Erzeugte Anomalien erhalten wiederum das 0,7- bis 2-Fache dieses Werts und haben eine Dauer von drei bis elf Aktualisierungen. Eine Dauer von drei entspricht dabei einem einzelnen Ausreißer, da Start- und Endpunkt unverändert bleiben. Dieses Verhalten kann durch eine kurzzeitige Störung verursacht werden, wie zum Bei-

spiel eine Spitze in der Versorgungsspannung (vgl. ISO 26262-5:2011 Fehlerfall „Power spikes“ in der entsprechenden Kategorie [Int11a]).

5. Analog zum Positive Peak Typ gibt es *Negative Peak* Anomalien. Definition und Parametrierung unterscheiden sich lediglich im Vorzeichen der Spitzigkeit.

6. Anomalien des Typs *Positive Ramp Plateau* lassen sich in drei Phasen unterteilen. Im ersten Drittel steigt der Signalwert linear an. Im zweiten bleibt der Wert konstant bevor er sich im letzten Drittel wieder linear an den originalen Signalwert am Ende der Anomalie annähert. Welchen Wertebereich die Rampe zu Beginn durchläuft ist einstellbar. Im vorliegenden Fall werden Anomalien mit der 0,7- bis 2-fachen Steigung generiert, bezogen auf den entsprechenden Maximalwert der Trainingsdaten (maximale durchschnittliche Steigung, ermittelt über die entsprechende Dauer und gefolgt von einem nahezu konstanten Signalwert). Die Dauer ist auf zwölf bis 24 Signalaktualisierungen konfiguriert. Dieses Fehlverhalten kann beispielsweise durch eine gestörte Rückkopplung in einem Regelkreis entstehen.

7. Der Anomalie-Typ *Negative Ramp Plateau* orientiert sich am Positive Ramp Plateau. Einziger Unterschied ist der lineare Abfall im ersten Drittel. Die Erzeugung und die damit zusammenhängende Parametrierung ist bis auf das Vorzeichen der Steigung identisch. Auch die möglichen Ursachen decken sich mit denen des Anomalie-Typs Positive Ramp Plateau.

8. Im Fall einer *Noise* Anomalie wird der originale Signalverlauf mit einem weißen Rauschen überlagert. Die Konfiguration der Amplitude orientiert sich wieder an den vorhandenen Trainingsdaten. Dazu wurde die maximale durchschnittliche Signaländerung, nach Vorzeichenwechseln der Steigung und über die potentielle Dauer einer Anomalie bestimmt. Die Amplitude der erzeugten Anomalien liegt zwischen

dem 0,7- und 5-Fachen des ermittelten Werts und die Dauer beträgt zwischen zwölf und 40 Signalaktualisierungen. Auslöser von Signalrauschen kann beispielsweise ein defekter Sensor oder eine unzureichende Abschirmung beziehungsweise Filterung sein. Des Weiteren kann eine fehlerhafte Sensor-Anbindung (zum Beispiel ein offener Signaleingang am Mikrocontroller) zu einem Rauschen des dazugehörigen Signals führen. Der letztere Fehlerfall ist im ISO Standard als „Open Circuit„ in der Kategorie für Verkabelung und Steckverbinder aufgeführt [Int11a].

9. Eine spezielle Form des Rauschens ist die Überlagerung des originalen Signalverlaufs mit einer Schwingung (*Sine* Anomalien). Innerhalb der ISO 26262-5:2011 ist der Fehlerfall „Oscillation“ für mehrere E/E-Komponenten definiert. Dazu zählen Sensoren, Versorgungsspannung sowie digitale und analoge Ein-/Ausgänge [Int11a]. Die Anzahl der überlagerten Sinus-Schwingungen (n) berechnet sich über $n = D/10 + 1$, wobei die Division in Integer-Logik erfolgt. D steht für die Dauer der Anomalie und liegt zwischen zwölf (zwei Schwingungen) und 40 (fünf Schwingungen) Signalaktualisierungen. Zur Konfiguration der Amplitude wurden die einzelnen Signalausschnitte der Trainingsdaten mit einer diskreten Fourier-Transformation untersucht und dabei der Maximalwert im relevanten Frequenzbereich ermittelt. Das 0,7- bis 3-Fache dieses Werts wird als Amplitude für die erzeugten Anomalien verwendet.

10. Den plötzlichen, positiven Versatz eines Signalwerts sollen Anomalien des Typs *Positive Step Offset* nachbilden. Dabei wird über die konfigurierte Dauer von zwölf bis 40 Aktualisierungen ein konstanter Wert addiert. Der Offset entspricht dem 0,7- bis 2-Fachen der maximalen positiven Signaländerung innerhalb der Trainingsdaten. Damit besteht eine Ähnlichkeit zu den Positive Step Plateau Anomalien. Allerdings bleibt der originale Signalverlauf größtenteils erhalten. Le-

diglich zu Beginn und am Ende der Anomalie entstehen unnatürliche Wertesprünge. Die ISO 26262-5:2011 listet den Fehlerfall „Offsets“ für Sensoren [Int11a].

11. *Negative Step Offset* Anomalien entsprechen in ihrer Definition und Konfiguration größtenteils dem Positive Step Offset Typ. Einziger Unterschied ist das Vorzeichen des Offsets.

12. Die Typen *Positive Ramp Offset* und Positive Ramp Plateau besitzen die gleiche Aufteilung in drei Phasen. Auch die festgelegte Dauer von zwölf bis 24 Signalaktualisierungen ist identisch. Anstatt das originale Signal zu überschreiben, werden die Rampen und der Offset im Fall einer Positive Ramp Offset Anomalie addiert. Das dadurch erzeugte Fehlverhalten ist subtiler. Die Rampen besitzen die 0,7- bis 2-fache Steigung, bezogen auf den entsprechenden Maximalwert der Trainingsdaten. Der ISO Standard verweist in diesem Fall auf einen „Drift“-Fehler in Bezug auf die Spannungsversorgung sowie für digitale und analoge Ein-/Ausgänge [Int11a]. Es ist jedoch denkbar, dass diese Art von Fehler auch bei Sensoren auftritt.

13. Wiederum analog zu Positive Ramp Offset wurde der Anomalie-Typ *Negative Ramp Offset* definiert und parametriert. Lediglich das Vorzeichen der Rampensteigung ist in diesem Fall negativ.

Einige der diskutierten Fehlerfälle können statisch vorhanden sein oder dynamisch zur Laufzeit auftreten. Die Detektion anhand eines Kommunikationssignals adressiert das dynamische Auftreten. Für die Erkennung von statischen Fehlern existieren bereits etablierte Mechanismen und Maßnahmen, die bei der Qualitätssicherung zum Einsatz kommen.
Zu den typischen, zu betrachtenden Sensor-Fehlern zählt nach ISO 26262-5:2011 auch ein Verlassen des gültigen Wertebereichs (vgl. „Out-of-range“) [Int11a]. Dies lässt sich allerdings durch statische Checks erkennen (N-3:

Range Sensor), weshalb hierfür kein separater Anomalie-Typ definiert wurde.

Element	Analyzed failure modes for 99% diagnostic coverage	Anomalie-Typ bezogen auf potentielle Auswirkungen des Fehlerfalls auf ein Kommunikationssignal
Harnesses including splice and connectors	Open Circuit	Noise
	Contact Resistance	
	Short Circuit to Ground (d.c Coupled)	Negative Step Plateau
	Short Circuit to Vbat	Positive Step Plateau
	Short Circuit between neighbouring pins	
	Resistive drift between pins	
Sensors including signal switches	*No generic fault model available. Detailed analysis necessary. Typical failure modes to be covered include*	
	Out-of-range	(statische Checks: N-3)
	Offsets	Negative/Positive Step Offset
	Stuck in range	Plateau
	Oscillations	Sine
Power supply	Drift and oscillation	Negative/Positive Ramp Offset, (Negative/Positive Ramp Plateau,) Sine
	Under and over Voltage	
	Power spikes	Negative/Positive Peak

Tabelle 5.1: Zuordnung von Anomalie-Typen zu den Fehlerfällen der ISO 26262-5:2011 [Int11a]

Tabelle 5.1 fasst die Zuordnung von Anomalie-Typen zusammen, wobei die ersten beiden Spalten der ISO 26262-5:2011, Tabelle D.1 entnommen sind [Int11a]. Die verbleibenden Fehlerfälle haben potentiell einen diffusen Einfluss auf einen Signalverlauf. Daher ist die Zuordnung eines diskreten Anomalie-Typs nicht möglich.
Neben Sensoren betrachtet die ISO 26262-5:2011 auch digitale und analoge Ein-/Ausgänge von Mikrocontrollern beziehungsweise Mikroprozessoren. Diese werden in der vorliegenden Arbeit als simple Sensoren/Aktoren angesehen und haben damit gegebenenfalls Einfluss auf Kommunikationssignale. Die zugeordneten Fehlerfälle sind „stuck-at“, „stuck-open“, offene und hochohmige Ausgänge, Kurzschlüsse zwischen Signalleitungen, Drift und Oszillation. Daraus ergeben sich keine weiteren Anomalie-Typen. Stuck-at, Drift und Oszillation sind bereits abgedeckt. Stuck-open sowie offene und hochohmige Ausgänge sind teilweise durch Noise Anomalien adressiert. Wie zuvor erwähnt, wurden für diffuse Auswirkungen, die beispielsweise durch einen Kurzschluss zweier Signalleitungen auftreten können, keine separaten Typen definiert.

5.5 Anomaliebewertung in Signalverläufen

Abschnitt 2.3.4 diskutiert die Bewertung von Anomalieerkennungsmechanismen sowie die dazugehörigen Basis-Metriken TP, FP, TN und FN. Bei Signalverläufen und speziell bei der Verwendung eines Sliding Window Ansatzes ist zusätzlich zu definieren, wann eine Anomalie als TP markiert wird. Dabei spielen zwei Faktoren eine Rolle: Zum einen erstreckt sich eine Anomalie über eine gewisse Dauer und zum anderen sind die anormalen Datenpunkte mehrfach in einem Eingangsvektor enthalten. Bild 5.9 zeigt ein entsprechendes Beispiel mit einer Negative Peak Anomalie zwischen dem zweiten und vierten Signalwert. Wie in Unterkapitel 5.4 beschrieben, bleibt der erste und letzte Datenpunkt im Anomalieausschnitt unverändert. Damit ist in Bild 5.9 lediglich der dritte Wert modifiziert (schwarz hinterlegt). Im

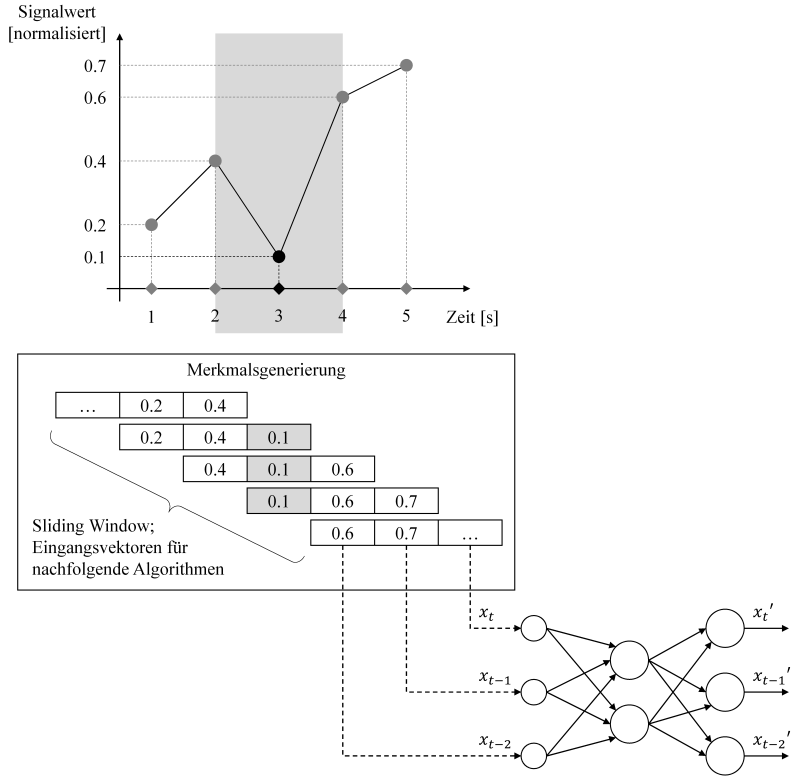

Bild 5.9: Anomaliebewertung in Signalverläufen [VWeb+]

gegebenen Beispiel besteht das Sliding Window und damit der Eingangsvektor der nachfolgenden Algorithmen aus drei Elementen. Im mittleren Teil von Bild 5.9 sind die dazugehörigen Eingangsvektoren dargestellt, die nacheinander den Algorithmen vorgelegt werden. Entsprechend des Sliding Windows taucht der anormale Wert dreimal auf. Daher stellt sich die Frage, ob eine Klassifizierung aller betroffenen Eingangsvektoren - oder einer Mehrzahl davon - als anormal notwendig ist, um eine Anomalie als TP zu markieren. Ziel dieser Arbeit ist das Erkennen von Angriffen oder

Fehlverhalten. Aus diesem Grund gilt eine Anomalie als detektiert, sobald ein Eingangsvektor mit mindestens einem anormalen Datenpunkt entsprechend klassifiziert wird. Analog zum TP entspricht ein fälschlicherweise anormal klassifizierter Eingangsvektor direkt einem FP. Eine Zusammenfassung mehrerer, aufeinender folgender FP findet nicht statt.

5.6 Festlegung von Hyperparametern, Trainingseigenschaften und Evaluierungskriterien

5.6.1 Vorgaben für die Implementierung von LODA

Der einzige Hyperparameter von LODA - die Größe des Sliding Windows beziehungsweise des Eingangsvektors - lässt sich nicht pauschal vorab bestimmen. Daher folgt eine entsprechende empirische Untersuchung in Unterkapitel 6.1.

Aufgrund der Arbeitsweise von LODA dauert das Training genau eine Epoche (d.h. alle Trainingsdaten werden dem Algorithmus genau einmal vorgelegt, siehe Abschnitt 2.2.2) und es müssen dafür keine zusätzlichen Eigenschaften festgelegt werden. Nach dem einmaligen Prozessieren aller Trainingsdaten sind die benötigten Histogramme erstellt. Der Schwellwert für die finale Klassifizierung entspricht der höchsten Anomaliebewertung für einen Eingangsvektor aus dem Trainingsdatensatz (nach Abschluss des Trainings). Damit ist sichergestellt, dass diese Daten keinen Falschalarm erzeugen. Da die notwendigen Projektionen auf dem Zufallsprinzip basieren, erfolgt die Auswahl der leistungsstärksten LODA-Instanz anhand der Validierungsdaten. Die in Abschnitt 2.3.4 eingeführte False Positive Rate (FPR) berechnet sich anhand der unveränderten, die True Positive Rate (TPR) anhand der mit Anomalien angereicherten Daten, siehe Bild 5.7. Unter Verwendung der Testdaten führt das gleiche Vorgehen zur finalen Beurteilung der Leistungsfähigkeit.

Abschnitt 4.3.1 beschreibt den Entwicklungsprozess für die lernenden Checks des Observers auf Basis der Anomaly Detection Configuration. Für LODA

wird allerdings kein Trainings-Framework benötigt beziehungsweise verwendet, siehe Bild 4.3. Die Implementierung erfolgte nativ in Python 3.

5.6.2 Vorgaben für die Implementierung von Autoencodern

Für Autoencoder wurde zunächst eine Machbarkeitsstudie anhand von simulierten Daten durchgeführt. Neben einer prinzipiell erfolgreichen Anwendung bei der Signalplausibilisierung stand dabei die erforderliche Netzgröße im Vordergrund der Untersuchung. Die ermittelten Richtwerte geben Aufschluss über die Anwendbarkeit in automotive Steuergeräten. Die Ergebnisse sind in Anhang A.4 zu finden.
Im Gegensatz zu LODA besitzen Autoencoder mehrere Hyperparameter, die vor Trainingsbeginn festzulegen und nachfolgend aufgelistet sind. Die ersten drei davon wurden bereits im Rahmen der Machbarkeitsstudie initial betrachtet und sind Bestandteil der empirischen Untersuchung und Evaluierung auf Basis von OBD-II Daten, siehe Unterkapitel 6.2. Die Hyperparameter vier bis sieben werden einmalig vorgegeben und bleiben für den weiteren Verlauf der Arbeit konstant.

1. *Anzahl der Neuronen in der Ein- und Ausgangsschicht*: Bedingt durch die Arbeitsweise ist die Anzahl der Ein- und Ausgangsneuronen bei Autoencodern immer identisch. Sie entspricht der Anzahl an Merkmalen, die zu reproduzieren sind.

2. *Anzahl der versteckten Schichten*: Ein einfacher Autoencoder besitzt eine versteckte Schicht. Die Verwendung von mehreren versteckten Schichten ist jedoch ebenfalls möglich. Die Gesamttopologie besteht in diesem Fall aus ineinander geschachtelten, einfachen Autoencodern, wobei die versteckte Schicht des äußeren gleichzeitig die Ein- und Ausgangsschicht des jeweils inneren Autoencoders darstellt.

3. *Anzahl der Neuronen in den versteckten Schichten*: Die Neuronenanzahl in den versteckten Schichten kann größer oder kleiner sein als in

der Ein- und Ausgangsschicht. Weniger Neuronen zwingen den Autoencoder eine Abstraktion beziehungsweise Kompression der Eingangsdaten zu finden.

4. *Aktivierungsfunktion aller Neuronen*: Es gibt eine Vielzahl unterschiedlicher Aktivierungsfunktionen. Unter den gegebenen Randbedingungen ist eine möglichst einfache Funktion vorteilhaft, da die Berechnung komplexer nicht-linearer Zusammenhänge aufwändig ist. Siehe dazu auch die Diskussionen in Unterkapitel 7.2. Im Gegensatz zur Machbarkeitsstudie werden daher nachfolgend ausschließlich Neuronen mit der sogenannten *Rectifier* Aktivierungsfunktion verwendet. Diese ist als $f(x) = max(0,x)$ definiert und in Bild 5.10 dargestellt. Die dazugehörigen Neuronen sind auch als Rectifier Linear Units (ReLU) bekannt.

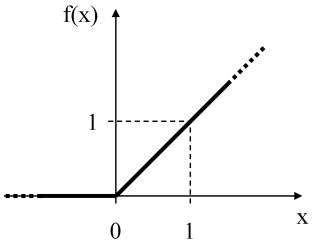

Bild 5.10: Die Rectifier Aktivierungsfunktion

Der Rectifier ist aufgrund seiner Einfachheit und Robustheit die am meisten verwendete Aktivierungsfunktion [RZL17]. Des Weiteren trugen ReLUs in den vergangenen Jahren maßgeblich zum Erfolg von tiefen neuronalen Netzen bei [He+15]. He et al. verweisen dabei auf unterschiedliche Anwendungsgebiete, in denen mit ReLUs verbesserte Resultate erzielt wurden [GBB11] [MHN13] [Zei+13] [NH10] [KSH12]. Nach einer Untersuchung verschiedener Aktivie-

rungsfunktionen empfehlen Ramachandran et al. Swish. Diese ist als $f(x) = x \cdot sig(\beta x)$ definiert und lieferte in ihren Evaluierungen bessere oder identische Ergebnisse im Vergleich zum Rectifier. Im Hinblick auf eine Realisierung in automotive Steuergeräten, werden im Rahmen dieser Arbeit ReLUs bevorzugt. Die Berechnung der Rectifier Aktivierungsfunktion enthält im Vergleich zu Swish keine komplexere mathematische Funktion. Dies wirkt sich positiv auf die benötigte Laufzeit aus. Ein Performanzvergleich zwischen Sigmoid-Neuronen und ReLUs ist in Unterkapitel 7.2 zu finden.

5. *Anzahl der Verbindungen zwischen den Neuronen*: In der vorliegenden Arbeit kommen ausschließlich vollständig verbundene Netze zum Einsatz (engl.: fully-connected). Damit ist jedes Neuron einer Schicht mit allen Neuronen der nachfolgenden Schicht verbunden.

6. *Initialisierung der Kantengewichte*: Für ReLU-Netze empfiehlt die Literatur häufig das Initialisierungsschema nach He et al. [He+15], welches im untersuchten Fall jedoch zu instabilen Trainingsergebnissen führte und daher keine Anwendung findet. Stattdessen erhalten die Kantengewichte zu Beginn zufällige Werte entsprechend einer Normalverteilung mit Mittelwert null und Standardabweichung 0,1[6]. Um betragsmäßig große Gewichte beziehungsweise Ausreißer zu vermeiden, werden Werte, die weiter als die doppelte Standardabweichung vom Mittelwert entfernt sind, verworfen und erneut zufällig bestimmt. Dadurch entsteht eine abgeschnittene Normalverteilung für die initialen Kantengewichte (engl.: truncated normal distribution) [God18].

[6] Für die Machbarkeitsstudie wurden die initialen Kantengewichte zufällig aus dem Wertebereich zwischen -0,05 und 0,05 gewählt.

7. *Fehlermaß*: Das Fehlermaß bestimmt den Reproduktionsfehler eines Autoencoders. Hierfür wird der Outlier Factor[7] nach Hawkins [Haw+02] verwendet, siehe auch Abschnitt 2.3.1.

Neben den Hyperparametern eines Autoencoders, sind auch für das Training selbst einige Eigenschaften zu definieren. Im Rahmen der vorliegenden Arbeit wurde hierfür das Vorgehen nach Bild 5.11 gewählt. Es folgt eine Beschreibung der einzelnen Schritte beziehungsweise der festzulegenden Eigenschaften.

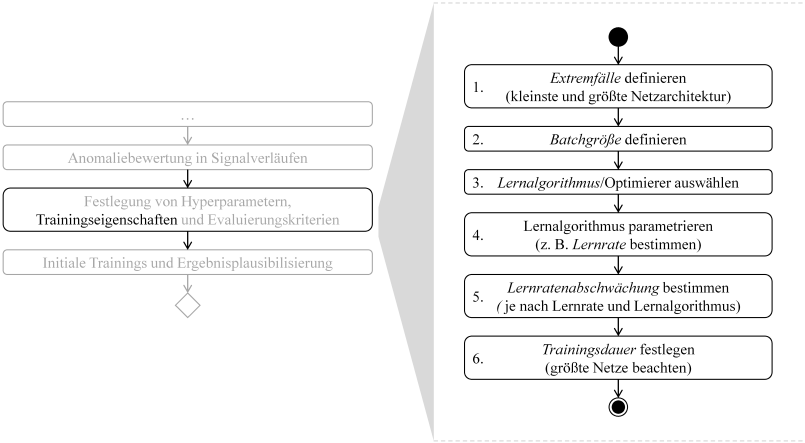

Bild 5.11: Vorgehen zur Ermittlung der Trainingseigenschaften

1. *Extremfälle*: Zur Bestimmung einiger Eigenschaften ist es notwendig, vorab die (minimale und) maximale Größe der zu trainierenden Autoencoder festzulegen. Die kleinsten betrachteten Netze bestehen aus zwei Eingangs-, einem versteckten und zwei Ausgangsneuronen. Die größten untersuchten Autoencoder - d.h. jene mit den meisten Kan-

[7] Der Outlier Factor entspricht der mittleren quadratischen Abweichung (engl.: Mean Squared Error, MSE) über alle Merkmale.

tengewichten - besitzen 32 Ein-/Ausgangsneuronen und 96 versteckte Neuronen.

2. *Batchgröße*: Die Batchgröße bestimmt die Anzahl an Eingangsvektoren innerhalb eines Batches (siehe 2.2.2). Sie liegt zwischen einem Eingangsvektor (SGD) und einer kompletten Epoche (Batch-Training). Ist die Prozessierung eines Batches abgeschlossen, passt der Lernalgorithmus die Kantengewichte entsprechend an. Für die Evaluierung in Unterkapitel 6.2 beinhaltet ein Batch 256 Eingangsvektoren. Damit handelt es sich um Mini-Batch-Training, siehe Abschnitt 2.2.2.

3. *Lernalgorithmus*: In Abschnitt 2.2.3 wurde der Backpropagation-Algorithmus vorgestellt, welcher auf dem Gradientenabstiegsverfahren basiert. Zwischenzeitlich existieren mehrere weiterentwickelte Verfahren. Im Folgenden kommt daher der Optimierer *Adam* [KB14] zum Einsatz, welcher weiterhin auf dem Gradientenabstiegsverfahren beruht. Allerdings bestimmt Adam die Änderungsrate für jedes Kantengewicht individuell und beachtet dabei auch die jeweils letzten Gewichtsanpassungen, vergleichbar zu einer Massenträgheit. Der Optimierer besitzt selbst wiederum Parameter. Mit Ausnahme der Lernrate (siehe nächster Punkt) werden hierfür die von Kingma und Ba vorgeschlagenen Werte verwendet [KB14][8]. Diese resultierten bei einer empirischen Untersuchung auf Basis der OBD-II Daten in stabilen Trainingsergebnissen.

4. *Lernrate*: Eine wichtige Eigenschaft des Trainings ist die Lernrate η. Sie bestimmt, wie stark die Kantengewichte nach der Verarbeitung eines Batches angepasst werden, siehe 2.2.3. Der Optimierer Adam verwendet standardmäßig eine Lernrate von 0,001. Speziell bei großen Netzen (zum Beispiel 32 Ein-/Ausgangsneuronen und 96 versteckte

[8] beta1 = 0,9, beta2 = 0,999, epsilon = 1e-8

Neuronen) wurden damit allerdings keine stabilen Trainingsergebnisse erzielt. Ein Großteil der trainierten Netze wies eine TPR von unter 5% auf. Durch die Erhöhung der Lernrate auf 0,003 verbesserten sich die Ergebnisse maßgeblich, siehe Unterkapitel 6.2. Um bei dieser vergleichsweise hohen Lernrate dennoch ein lokales oder das globale Minimum präzise zu finden, wird zusätzlich eine Lernratenabschwächung verwendet, wie im nächsten Punkt beschrieben.

5. *Lernratenabschwächung*: Während des Trainings ist es möglich, die Lernrate nach einer definierten Anzahl von prozessierten Batches zu verringern. Dies verhindert sprunghaftes Verhalten um ein lokales (oder das globale) Minimum und unterstützt das Konvergieren des Trainings. Aufgrund der hohen initialen Lernrate von 0,003 wird diese bei der folgenden Evaluierung nach jeweils 100.000 Batches um 33,3% verringert.

6. *Trainingsdauer*: Sofern keine anderen Abbruchkriterien definiert sind und eintreten, bestimmt die Trainingsdauer, wie lange zu trainieren ist. Eine Möglichkeit ist die Angabe einer Epochenanzahl (Anzahl an Iterationen über den gesamten Trainingsdatensatz), eine andere die Angabe der zu prozessierenden Batches. Die maximale Trainingsdauer ist im Folgenden auf 1.000.000 Batches festgelegt. Bei den verwendeten OBD-II Daten entspricht dies ca. 1450 Epochen. Der Wert wurde so gewählt, dass die größten untersuchten Autoencoder während des Trainingsvorgangs konvergieren. Kleinere Netze konvergieren in der Regel deutlich schneller, daher ist eine Betrachtung der größten Netze ausreichend.

Das Trainingsergebnis sind die ermittelten Kantengewichte zwischen den einzelnen Neuronen. Bevor eine Evaluierung erfolgen kann, ist zusätzlich der Schwellwert OF_S zu bestimmen. Analog zum Vorgehen bei LODA entspricht der gesetzte Wert dem maximalen Reproduktionsfehler, welcher bei

gegebener Parametrierung für einen Eingangsvektor der Trainingsdaten entsteht.

Es existieren mehrere Frameworks für die Implementierung von künstlichen neuronalen Netzen. Zu den bekanntesten gehören TensorFlow™ [Aba+15], PyTorch [17] [PyT], Caffe [Jia+14] und Scikit-learn [Ped+11]. Aufgrund der breiten Anwenderbasis sowie der integrierten Visualisierungs- und Auswertemöglichkeiten, wird im Rahmen dieser Arbeit TensorFlow™ in Kombination mit der Programmiersprache Python 3 eingesetzt. TensorBoard ist integrierter Bestandteil von TensorFlow™ und ermöglicht eine komfortable, Browser-basierte Analyse der Trainings-, Validierungs- und Testergebnisse.

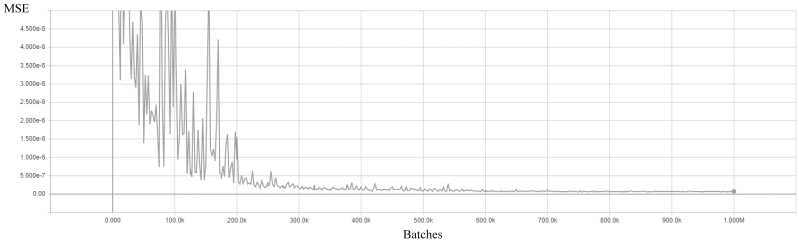

Bild 5.12: Durchschnittlicher Reproduktionsfehler eines 32-96-32 Autoencoders über der Trainingsdauer

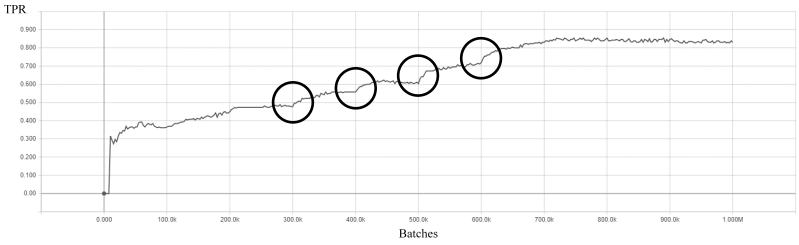

Bild 5.13: Beispielhafter Verlauf der TPR während des Trainings

Bild 5.12 und Bild 5.13 zeigen eine beispielhafte TensorBoard Auswertung für das Training eines 32-96-32 Autoencoders. In Bild 5.12 ist der durchschnittliche Reproduktionsfehler eines Eingangsvektors des aktuellen Batches über die Trainingsdauer dargestellt. Der kontinuierlich abnehmende Fehler ist eine erste Indikation für erfolgreiches Training.
Die Auswertung der TPR über die Trainingsdauer ist in Bild 5.13 zu sehen. Grundlage bilden die mit Anomalien angereicherten Validierungsdaten. Dieses Beispiel verdeutlicht auch die ausgewählten Trainingseigenschaften. Zu Beginn des Trainings steigt die TPR - auch durch die hohe initiale Lernrate - steil an, flacht dann aber etwas ab. Wie zuvor erwähnt wird die Lernrate alle 100.000 Batches um 33,3% verringert. Der Effekt ist besonders im Bereich zwischen 300.000 und 600.000 Batches zu erkennen. Nach jeder Lernratenabschwächung, steigt die TPR nochmals an. Die entsprechenden Stellen sind in Bild 5.13 mit Kreisen markiert. Ab 700.000 Batches bleibt die TPR nahezu konstant. Das Training konvergiert bei ungefähr 85% TPR. Da es sich hierbei um das größte untersuchte Netz handelt, wird davon ausgegangen, das alle untersuchten Autoencoder binnen einer Million Trainingsbatches konvergieren (siehe Festlegung der Trainingsdauer).
Anhand des in Bild 5.15 dargestellten Trainingsverlaufs eines 8-8-8 Autoencoders ist zu erkennen, dass die höchste TPR (auf den Validierungsdaten mit Anomalien) nicht notwendigerweise am Ende des Trainings erreicht wird. Den Bestwert erzielt diese Algorithmus-Instanz nach ca. 260.000 Batches (siehe Stern in Bild 5.15). Am Ende des Trainings ist die TPR deutlich geringer, obwohl der dazugehörige MSE weiter abnimmt, siehe Bild 5.14. Der Autoencoder lernt auch Eingangsdaten zu reproduzieren die anormale Signalwerte enthalten. Dies entspricht einer Überanpassung. Im Extremfall lernt er die perfekte Identitätsfunktion und es wäre keine Anomalieerkennung mehr möglich. Daher ist es notwendig während des Trainings die leistungsstärkste Autoencoder-Parametrierung - d.h. die entsprechenden Kantengewichte - zu ermitteln. Dazu wurden die folgenden vier Metriken definiert, die nach jeweils 2500 prozessierten Batches berechnet werden:

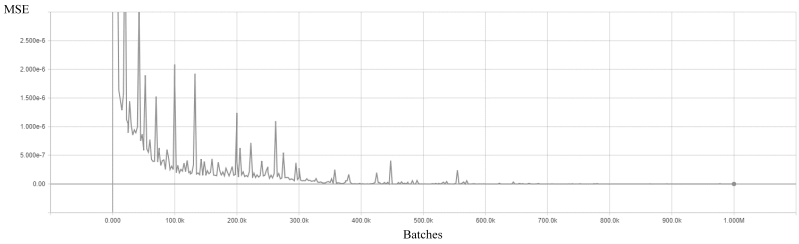

Bild 5.14: Durchschnittlicher Reproduktionsfehler eines 8-8-8 Autoencoders über der Trainingsdauer

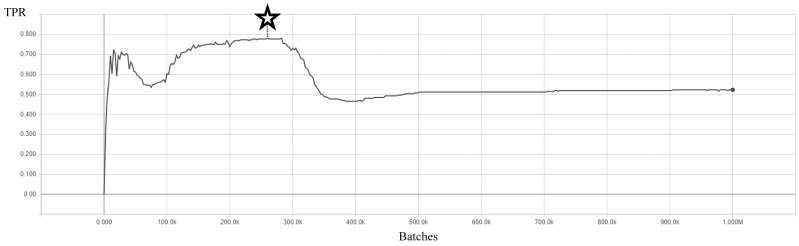

Bild 5.15: TPR eines 8-8-8 Autoencoders über der Trainingsdauer

1. *Best of loss*: Verweist auf die Kantengewichte des Autoencoders, welche zum geringsten durchschnittlichen Reproduktionsfehler führen, basierend auf dem Validierungsdatensatz ohne Anomalien.

2. *Best of max loss*: Diese Kantengewichte führen zum geringsten maximalen Reproduktionsfehler unter Verwendung der Validierungsdaten ohne Anomalien.

3. *Best of TPR*: Mit diesen Kantengewichten erreicht der Autoencoder die beste TPR basierend auf dem Validierungsdatensatz mit Anomalien.

4. *Best of TPR at zero FP*: Der Parametersatz, welcher in der höchsten TPR resultiert und gleichzeitig keine Falschalarme auslöst. Die

Ermittlung der Falschalarme basiert auf den Validierungsdaten ohne Anomalien. Zur Berechnung der TPR wird das Pendant mit Anomalien verwendet.

Sollte eine der vier Metriken ihren (zwischenzeitlichen) Bestwert erreichen, wird die dazugehörige Parametrierung gespeichert. Inklusive der Kantengewichte nach dem Prozessieren aller 1.000.000 Batches, liegen damit fünf Parametersätze je Trainingslauf vor.

6 Leistungsevaluierung basierend auf FPR und TPR

Die durchgeführte Evaluierung und die daraus resultierenden Ergebnisse werden im Rahmen dieses Kapitels diskutiert. Untersucht wurde die Anomalieerkennung in Signalverläufen mittels LODA und Autoencoder auf Basis der aufgezeichneten OBD-II Daten. Neben der Ermittlung von TPR und FPR liegt ein besonderes Augenmerk auf der Untersuchung unterschiedlicher Hyperparameter und der damit verbundenen Anwendbarkeit in automotive Steuergeräten.

Unterkapitel 5.3 beschreibt das generelle Vorgehen bei der Leistungsbewertung anhand der Datensatzaufteilung in Training, Validierung und Test. Bild 5.7 gibt eine grafische Zusammenfassung. Bevor die Evaluierung erfolgen kann, sind zunächst die Validierungs- und Testdaten mit Anomalien anzureichern. Dazu wurden auf beiden Teildatensätzen jeweils 260 zufällige Anomalien erzeugt, 20 für jeden Anomalie-Typ (siehe Unterkapitel 5.4).

Der benötigte Schwellwert zur Anomalieklassifizierung - sowohl bei LODA als auch bei Autoencodern - wird anhand der Trainingsdaten bestimmt, wie in Abschnitt 5.6.1 und 5.6.2 beschrieben.

Die Bewertung der einzelnen Algorithmen erfolgt anhand deren FPR und TPR. Dabei wird die FPR auf Basis der Validierungs- und Testdaten ohne Anomalien ermittelt. Dies ist notwendig, da erzeugte Anomalien ansonsten Falschalarme überdecken können. Ein verzerrtes Ergebnis wäre die Folge. Die mit Anomalien angereicherten Teildatensätze kommen für die Bestimmung der dazugehörigen TPR zum Einsatz. Die Auswahl der jeweils leistungsstärksten Algorithmus-Instanz basiert auf den Validierungsdaten,

wobei folgende Definition gilt:

*Für die vorliegende Arbeit ist der **leistungsstärkste Algorithmus** jener, welcher bei der niedrigsten FPR die höchste TPR aufweist.*

Dieses Kriterium wurde aufgrund der besonderen Bedeutung der FPR für den vorliegenden Anwendungsfall gewählt. Zu viele Falschalarme erzeugen im besten Fall Mehraufwand in einer nachgelagerten Untersuchung, zum Beispiel auf Backend-Seite, siehe Abschnitt 3.2.2. Bei einer Notifizierung des Fahrers verringern Falschalarme die Akzeptanz des Systems und bei aktiven Gegenmaßnahmen sind die Folgen gegebenenfalls noch gravierender, beispielsweise durch die fälschliche Abschaltung von (Teil-)Systemen.
Wie in Unterkapitel 5.3 beschrieben, umfasst die Algorithmenauswahl drei Ebenen. Auf erster Ebene existieren üblicherweise Hyperparameter, für welche die passenden Werte zu ermitteln sind. Dieser Punkt wird durch die empirische Untersuchung adressiert. Auf zweiter Ebene ist der Effekt von Zufallszahlen zu berücksichtigen. Im Fall von LODA sind es die zufälligen Projektionsvektoren, bei Autoencodern die zufällige Initialisierung der Kantengewichte. Daher wurde jeder untersuchte Algorithmus beziehungsweise jede untersuchte Netztopologie zehn Mal mit identischen Hyperparametern trainiert. Die Vergleiche im weiteren Verlauf des Kapitels beziehen sich dabei immer auf die jeweils leistungsstärksten Instanzen, unter Verwendung der obigen Definition. Die letzte Ebene repräsentiert die Auswahl der Parametrierung während eines Trainingslaufs. Bedingt durch die Arbeitsweise des Algorithmus, findet die Bewertung für LODA nur einmal am Ende des Trainings statt. Im Gegensatz dazu wird beim Training von Autoencodern die FPR und TPR in regelmäßigen Abständen ermittelt und die dazugehörige Netzparametrierung bei Bedarf abgespeichert, siehe Abschnitt 5.6.2. Eine grafische Zusammenfassung des Vorgehens gibt Bild 5.6.
Aufgrund seiner besonderen Bedeutung im E/E-System eines Fahrzeugs bildet das Geschwindigkeitssignal der aufgezeichneten OBD-II Daten die

Grundlage der nachfolgenden Evaluierung. Außerdem ist die Fahrzeuggeschwindigkeit und deren zeitlicher Verlauf ohne fach-spezifisches Wissen vorstell- und interpretierbar.

6.1 Evaluierung von LODA

LODA besitzt nur einen Hyperparameter (siehe Abschnitt 5.6.1): Die Größe des Eingangsvektors, d.h. die betrachtete Sequenzlänge. Deshalb erfolgte eine empirische Untersuchung der Leistungsfähigkeit in Abhängigkeit dieses Hyperparameters. Unter Berücksichtigung der Definition des leistungsstärksten Algorithmus, stand zunächst die FPR im Vordergrund. Für jede untersuchte Eingangsvektorgröße wurden zehn LODA-Instanzen trainiert und nachfolgend die FPR auf Basis der Validierungsdaten ohne Anomalien ermittelt. Tabelle 6.1 fasst die Ergebnisse zusammen, wobei die Einträge jeweils die Instanz mit der geringsten FPR repräsentieren.

Sequenzlänge	4	8	16	**32**
FPR [%]	1,3	0,9	0,7	**0,3**

Tabelle 6.1: Geringste FPR für LODA in Abhängigkeit der Sequenzlänge

Sequenzlänge	4	**8**	16	32
TPR [%]	33,5	**52,3**	41,2	31,2

Tabelle 6.2: Höchste TPR für LODA in Abhängigkeit der Sequenzlänge

Die Validierungsdaten umfassen ungefähr 27.000 Datenpunkte/Eingangsvektoren und damit ca. 7,5 Stunden Fahrt (bei einer Abtastrate von einer Sekunde). Eine FPR von 0,3% entspricht in diesem Fall ca. 80 Falschalarmen auf dem gesamten Teildatensatz beziehungsweise einem Falschalarm alle fünf bis sechs Minuten. Dies ist aus zuvor genannten Gründen für den angestrebten Anwendungsfall nicht akzeptabel. Es ist jedoch auffällig, dass

die FPR mit steigender Sequenzlänge stetig abnimmt. Daher erscheint die Untersuchung noch größerer Sequenzlängen sinnvoll. Zunächst wurde allerdings für die bereits vorliegenden Konfigurationen die jeweils höchste TPR ermittelt, unabhängig von der LODA-Instanz mit der niedrigsten FPR. Tabelle 6.2 veranschaulicht, dass die erreichte TPR ihren Maximalwert bei einer Sequenzlänge von acht erreicht und danach wieder abnimmt. Unter der Annahme, dass sich dieser Trend für noch größere Sequenzlängen fortsetzt, ist eine weitere Evaluierung von LODA nicht zielführend, zumal die Betrachtung von FPR und TPR bisher unabhängig voneinander erfolgt ist. Die LODA-Instanz mit der niedrigsten FPR wird wahrscheinlich nicht gleichzeitig die höchste TPR aufweisen. Aus diesem Grund wurde LODA nicht weiter evaluiert und der Fokus stattdessen auf die Untersuchung von Autoencodern gelegt.

6.2 Evaluierung von Autoencodern

Die Machbarkeitsstudie bezüglich der Anwendung von Autoencodern zur Anomalieerkennung in Signalverläufen lieferte erste Richtwerte bezüglich der benötigten Netztopologie, siehe Anhang A.4. Diese Untersuchung wird im Folgenden anhand der OBD-II Daten fortgesetzt und erweitert. Im Vordergrund stehen die drei Hyperparameter, welche die Neuronenanzahl bestimmen (siehe Abschnitt 5.6.2), d.h. die Anzahl der Neuronen in der Ein- und Ausgangsschicht, die Anzahl der versteckten Schichten sowie die Anzahl der Neuronen in den versteckten Schichten. Für die restlichen Hyperparameter und Trainingseigenschaften gelten die in Abschnitt 5.6.2 festgelegten Werte.
Wie in Anhang A.4, gilt auch in diesem Kapitel die folgende Definition zur Beschreibung einer Netztopologie: $X - Y - ... - Y - X$
Jede Zahl steht für die Anzahl der Neuronen in der entsprechenden Schicht. X ist damit die Anzahl an Ein- beziehungsweise Ausgangsneuronen, Y die Neuronenanzahl in der ersten beziehungsweise letzten versteckten Schicht

usw.

Vor einer zeitlich aufwendigen empirischen Untersuchung, ist es wertvoll, stichprobenartig Algorithmen zu trainieren und die erzielten Ergebnisse auf Plausibilität zu prüfen. So lassen sich einerseits prinzipielle Implementierungsprobleme und ungeeignete Verfahren frühzeitig erkennen, siehe Unterkapitel 6.1. Andererseits hilft dieses Vorgehen bei der Detektion von Unregelmäßigkeiten im verwendeten Datensatz. Die Ergebnisplausibilisierung für die initial trainierten Autoencoder erfolgte anhand der dazugehörigen Reproduktionsfehler. Der maximale Reproduktionsfehler für die Trainingsdaten war um mehr als zwei Größenordnungen höher als für die Validierungs- und Testdaten. Obwohl der insgesamt größte Fehler unter Verwendung der Trainingsdaten entstehen soll - anderenfalls würden bei Validierungs- und Testdaten Falschalarme erzeugt werden - lagen die ermittelten Werte zu weit auseinander. Aufgrund des verhältnismäßig hohen Schwellwerts[1] lag die TPR auf einem nicht zufriedenstellenden Wert um die 30%.

Eine genauere Analyse zeigte die in Bild 6.1 dargestellten Unregelmäßigkeiten in den Trainingsdaten, welche einen maximalen Reproduktionsfehler von ca. $1,5 \cdot 10^{-4}$ erzeugen. Zum Vergleich: Der maximale Reproduktionsfehler für die Validierungs- und Testdaten beträgt unter Verwendung des selben Autoencoders $2,6 \cdot 10^{-7}$ beziehungsweise $3,1 \cdot 10^{-7}$.

1. Die Fahrzeuggeschwindigkeit weist in einer aufgezeichneten Datei einen unnatürlichen Verlauf auf. Am Anfang und am Ende des in Bild 6.1 (links) gekennzeichneten Bereichs besitzt das Signal einen vergleichsweise hohen positiven beziehungsweise negativen Gradienten. Zusätzlich sind große Teile des markierten Verlaufs stückweise linear. Dies ist unter Berücksichtigung einer Fahrzeugbewegung und im Ver-

[1] Der verwendete Schwellwert entspricht dem maximalen Reproduktionsfehler für die Trainingsdaten nach Abschluss des Trainings.

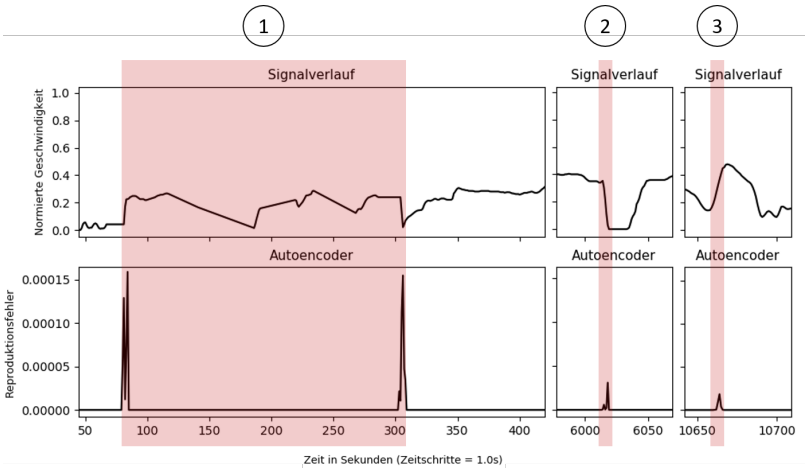

Bild 6.1: Unregelmäßigkeiten in den Trainingsdaten

gleich zu den restlichen aufgezeichneten Daten unnatürlich und lässt auf Probleme bei der Datenakquise oder -aufzeichnung schließen.

2. Die mittlere Markierung in Bild 6.1 kennzeichnet eine starke Bremsung - nachfolgend als Vollbremsung bezeichnet - von ca. 80 km/h bis zum Stillstand binnen fünf Sekunden. Große Verzögerungen sind häufiger in den Trainingsdaten enthalten, wobei die aufgezeichnete Vollbremsung eine Ausnahme darstellt.

3. Auf der rechten Seite von Bild 6.1 ist eine starke Beschleunigung über fünf Sekunden zu sehen, von ca. 45 km/h bis ca. 95 km/h. Der vergleichsweise hohe Gradient über mehrere Sekunden ist ebenfalls ein Sonderfall in den aufgezeichneten Daten.

Sowohl die Vollbremsung als auch die anhaltend starke Beschleunigung sind Fahrsituationen, die im normalen Straßenverkehr auftreten. Die initial trainierten Autoencoder erzeugen in diesen Fällen dennoch hohe Reproduktionsfehler und kennzeichnen sie damit als Anomalien. Aufgrund des sel-

tenen Auftretens ist die entsprechende Klassifizierung nachvollziehbar und zeigt eine grundlegende Herausforderung bei der Anomalieerkennung: Die Definition des Normalverhaltens. Für lernende Algorithmen sollten die Trainingsdaten daher das gesamte erlaubte Normalverhalten - d.h. alle *normalen* Fahrsituationen - in ähnlichem Umfang abbilden. Anderenfalls steigt in der Regel die Wahrscheinlichkeit von Falschalarmen bei normalem, aber in den Trainingsdaten unterrepräsentiertem Verhalten. Da die beiden beschriebenen Fahrsituationen in den Trainingsdaten enthalten sind und auf deren Basis der Schwellwert zur Anomalieklassifizierung bestimmt wird, nimmt im vorliegenden Fall die TPR ab. Es werden lediglich Anomalien erkannt, welche einen noch höheren Reproduktionsfehler verursachen als die Vollbremsung und die anhaltend starke Beschleunigung.
Aus den beschriebenen Gründen wurden alle drei aufgezeichneten Dateien, welche die in Bild 6.1 gezeigten Signalverläufe enthalten, im Datensatz entsprechend gekennzeichnet und aus den Trainingsdaten entfernt. Zusätzlich wird dies bei der finalen Bewertung berücksichtigt.

Der untere Teil von Bild 5.2 zeigt ein iteratives Vorgehen, welches in Bild 6.2 detailliert ist. Im ersten Durchlauf (linke Seite von Bild 6.2) liegt der Fokus auf Netztopologien mit einer versteckten Schicht. Die Anzahl der Ein-/Ausgangsneuronen sowie der versteckten Neuronen ist Gegenstand der Untersuchung, wobei das größte betrachtete Netz bereits bei der Bestimmung der Trainingsparameter festgelegt wurde. Es besitzt 32 Ein-/Ausgangs- und 96 versteckte Neuronen. Es wird davon ausgegangen, dass größere Autoencoder aufgrund des Ressourcenverbrauchs (RAM/ROM und Laufzeit) nicht für den breiten Einsatz auf aktuellen automotive Steuergeräten geeignet sind. Die dazugehörige Laufzeit- und Speicheranalyse folgt in Unterkapitel 7.2. Die verbleibenden, kleineren Netztopologien wurden basierend auf der Anzahl von Ein-/Ausgangsneuronen gruppiert. Diese Gruppierung bildet wiederum kleinere Iterationen innerhalb des ersten Durchlaufs zur Untersuchung von Topologien mit einer versteckten Schicht, siehe rechter Teil von Bild 6.2.

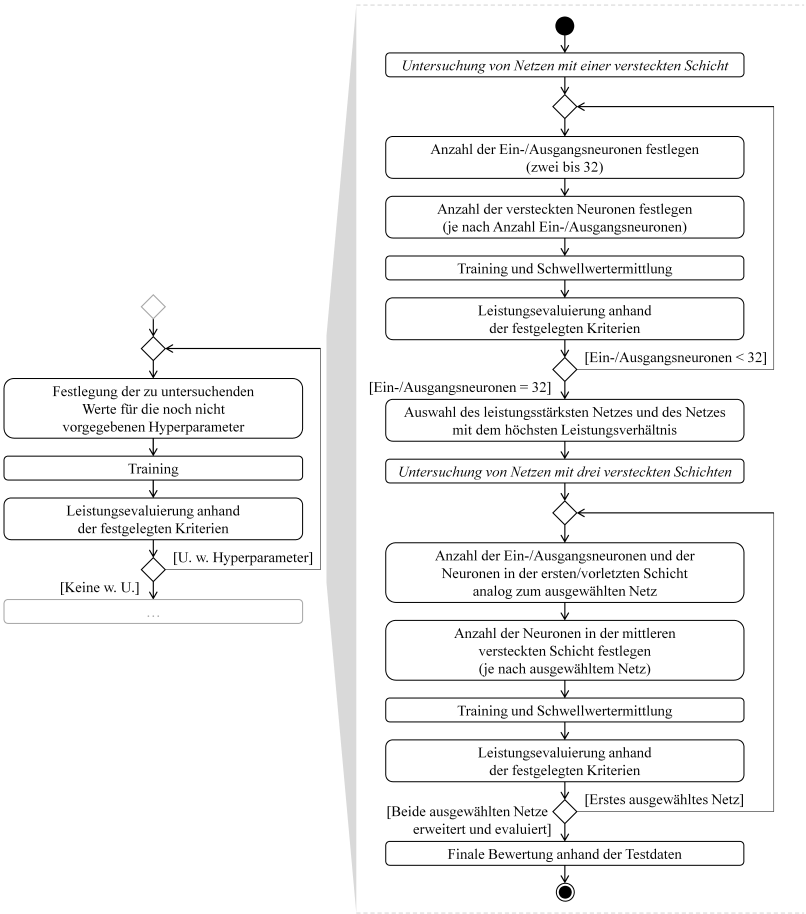

Bild 6.2: Vorgehen bei der Autoencoder-Evaluierung

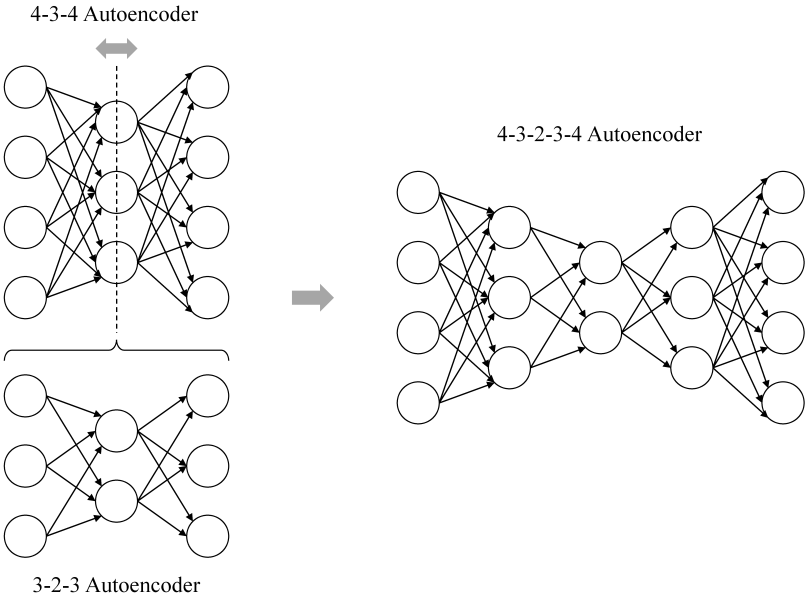

Bild 6.3: Autoencoder mit mehreren versteckten Schichten

Nach der Untersuchung aller festgelegten Topologien mit einer versteckten Schicht, wird das leistungsstärkste Netz (nach obiger Definition) ausgewählt und um zwei weitere versteckte Schichten erweitert. Dabei entsteht ein Stacked/Deep Autoencoder. Das Prinzip ist in Bild 6.3 verdeutlicht. Für das gezeigte Beispiel wäre es möglich den 4-3-4 und den 3-2-3 Autoencoder separat voneinander zu trainieren [HS06] [Vin+10]. Dies findet im Rahmen der vorliegenden Arbeit jedoch nicht statt. Autoencoder mit drei versteckten Schichten werden als ein Netz betrachtet und dementsprechend trainiert.
Ein weiteres Ziel ist die Bestimmung des leistungstärksten Autoencoders in Abhängigkeit der Netzgröße - und damit der benötigten Rechenleistung. Hierfür wurde als Metrik das *Leistungsverhältnis LV* nach Formel 6.1 definiert.

$$LV = \frac{\frac{TPR_{0,net}}{TPR_{0,max}}}{\frac{num_{connections,net}}{num_{connections,max}}} \qquad \text{wenn } \frac{TPR_{0,net}}{TPR_{0,max}} >= 0,75 \tag{6.1}$$

$TPR_{0,max}$ ist dabei die höchste TPR bei gleichzeitig null Falschalarmen[2], in Bezug auf alle untersuchten Autoencoder mit einer versteckten Schicht. $num_{connections,max}$ ist die dazugehörige Anzahl der Neuronen-Verbindungen bzw. Kantengewichte der entsprechenden Topologie[3]. Analog dazu sind $TPR_{0,net}$ und $num_{connections,net}$ die Werte für den zu vergleichenden Autoencoder. Damit gilt $LV = 1$ für das leistungsstärkste Netz. Sollte eine Topologie bei deutlich weniger Neuronen-Verbindungen eine TPR nahe am absoluten Maximum erzielen, ist $LV > 1$. Je größer LV, desto besser ist das Verhältnis zwischen Leistungsfähigkeit und benötigter Rechenleistung. Bei der Berechnung werden allerdings nur Netze berücksichtigt, deren TPR mindestens 75% von $TPR_{0,max}$ beträgt. Die Topologie des Autoencoders mit dem höchsten Leistungsverhältnis wird ebenfalls selektiert und im zweiten Durchlauf (linke Seite von Bild 6.2) mit zwei zusätzlichen versteckten Schichten betrachtet. Die aufgestellte Randbedingung verhindert dabei die Auswahl eines sehr kleinen Netzes mit geringer TPR.

Die Analyse startet mit den kleinstmöglichen Topologien, welche einen Signalverlauf als Eingangsvektor erhalten. Tabelle 6.3 zeigt die Ergebnisse für Netze mit zwei Ein- und Ausgangsneuronen. Die leistungsstärksten Instanzen der evaluierten Topologien generieren unter Verwendung der Validierungsdaten keine Falschalarme. Daher ist die jeweils höchste TPR bei null Falschalarmen dargestellt. Wie zu Beginn dieses Kapitels beschrieben, wurden je zehn Autoencoder-Instanzen mit gleichen Hyperparametern trainiert.

[2] Aufgrund der Machbarkeitsstudie wurde die Annahme getroffen, dass einige der trainierten Autoencoder auf Basis der Validierungsdaten keine Falschalarme erzeugen.

[3] Jede Neuronen-Verbindung bzw. jedes Kantengewicht entspricht einer Multiplikation bei der Inferenz des Autoencoders.

Topologie	2-1-2	2-2-2	2-4-2	2-6-2	2-8-2	2-10-2	**2-12-2**	2-14-2	2-16-2
TPR (bei FPR = 0) [%]	28,5	33,5	30,4	52,7	64,2	38,5	**64,6**	61,2	60,4

Tabelle 6.3: Evaluierungsergebnisse für Autoencoder mit 2-x-2 Topologie

Anomalie-Typ	Plateau	Pos. Step Plateau	Neg. Step Plateau	Pos. Peak	Neg. Peak	Pos. Ramp Plateau	Neg. Ramp Plateau
TPR [abs.]	3/20	13/20	11/20	8/20	3/20	14/20	18/20
Anomalie-Typ	Noise	Sine	Pos. Step Offset	Neg. Step Offset	Pos. Ramp Offset	Neg. Ramp Offset	*Summe*
TPR [abs.]	9/20	18/20	16/20	19/20	18/20	18/20	*168/260*

Tabelle 6.4: Erkannte Anomalien durch den leistungsstärksten 2-12-2 Autoencoder

Die Ergebnisse in Tabelle 6.3 zeigen, dass die TPR bei den drei kleinsten Netzen deutlich unter 50% liegt. Hier scheint die Anzahl an versteckten Neuronen nicht ausreichend zu sein, um eine passende interne Darstellung zu finden. Im Gegensatz dazu erkennt ein Autoencoder mit lediglich zwei Ein-/Ausgangs- und zwölf versteckten Neuronen bereits 64,6% aller erzeugten Anomalien und löst dabei gleichzeitig keinen Falschalarm aus.
Tabelle 6.4 enthält die komplette Aufschlüsselung der erkannten Anomalien für den leistungsstärksten 2-12-2 Autoencoder mit 64,6% TPR. Die Aufschlüsselung nach Anomalie-Typen zeigt, dass speziell Plateau, Positive Peak, Negative Peak und Noise Anomalien vergleichsweise schlecht erkannt werden. Dies lässt sich durch deren Form erklären. Beispielsweise ist das charakteristische Merkmal eines (Negative) Peaks die Spitze, d.h. der Vorzeichenwechsel der ersten Ableitung, welche bei lediglich zwei Eingangsneuronen nie vollständig im Eingangsvektor abgebildet wird. Ähnlich verhält es sich beim Übergang ("Knick") in ein Plateau. Auch Noise Anomalien sind durch häufige Vorzeichenwechsel in der ersten Ableitung gekennzeichnet und damit nur eingeschränkt in einem Eingangsvektor mit zwei Werten enthalten.

Topologie	3-1-3	3-2-3	3-3-3	3-6-3	3-9-3	**3-12-3**	3-15-3	3-18-3	3-21-3
TPR (bei FPR = 0) [%]	15,4	53,8	65,0	64,2	78,8	**85,0**	83,1	72,7	83,5

Tabelle 6.5: Evaluierungsergebnisse für Autoencoder mit 3-x-3 Topologie

Anomalie-Typ	Plateau	Pos. Step Plateau	Neg. Step Plateau	Pos. Peak	Neg. Peak	Pos. Ramp Plateau	Neg. Ramp Plateau
TPR [abs.]	10/20	18/20	16/20	12/20	17/20	16/20	19/20
Anomalie-Typ	Noise	Sine	Pos. Step Offset	Neg. Step Offset	Pos. Ramp Offset	Neg. Ramp Offset	*Summe*
TPR [abs.]	17/20	19/20	20/20	20/20	18/20	19/20	*221/260*

Tabelle 6.6: Erkannte Anomalien durch den leistungsstärksten 3-12-3 Autoencoder

Aufgrund der erwähnten Anomalie-Charakteristika, die erst ab einer Sequenzlänge von drei im Eingangsvektor enthalten sind, erreichen 3-x-3 Topologien deutlich höhere TPRs, siehe Tabelle 6.5. Die enthaltenen Werte repräsentieren wiederum Netze, die auf Basis der Validierungsdaten keine Falschalarme auslösen. 85% TPR sind bereits nahe am erreichten Maximum für Topologien mit einer versteckten Schicht. Die detaillierte Analyse des leistungsstärksten 3-12-3 Autoencoders in Tabelle 6.6 bestätigt die gestiegene TPR für die entsprechenden Anomalie-Typen. Auffällig ist, dass auch in diesem Fall die höchste TPR mit einer großen versteckten Schicht erzielt wird. Ein ähnliches Bild ergibt sich für die Topologien mit vier, sechs, acht, zwölf und 16 Ein-/Ausgangsneuronen. Eine versteckte Schicht mit mehr Neuronen als die Ein- beziehungsweise Ausgangsschicht ist zunächst konträr zum Grundgedanken des Autoencoders, eine komprimierte Darstellung des Eingangsvektors zu finden und diesen damit zu abstrahieren. Im vorliegenden Anwendungsfall und für vergleichsweise kleine Topologien scheint eine große versteckte Schicht jedoch vorteilhaft zu sein, um eine gute interne Darstellung der möglichen Signalverläufe zu finden. Durch das fehlende „Bottleneck“ (Verjüngung in der versteckten Schicht), besteht die Gefahr der Überanpassung, siehe Abschnitt 5.6.2 und Bild 5.15. Die kontinuierliche Überwachung des Trainingsverlaufs sowie das Abspeichern von Para-

metrierungen (vergleichbar mit Zwischenständen) anhand definierter Metriken erlaubt dennoch das erfolgreiche Verwenden solcher Netze.

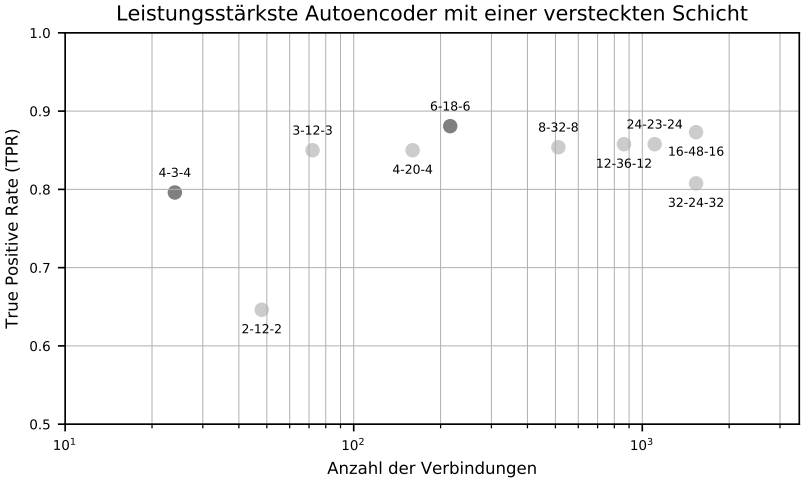

Bild 6.4: Zusammenfassung der Evaluierungsergebnisse für Autoencoder mit einer versteckten Schicht

Bild 6.4 fasst die Evaluierungsergebnisse für Autoencoder mit einer versteckten Schicht zusammen. Dabei ist jeweils der Autoencoder einer Gruppe als Datenpunkt visualisiert, welcher die höchste TPR (y-Achse) bei keinem Falschalarm erreicht. Wie zuvor erwähnt wurden die Ergebnisse auf Basis der Validierungsdaten ohne und mit Anomalien ermittelt. Auf der x-Achse ist die Anzahl der Verbindungen zwischen Neuronen in logarithmischer Skalierung aufgetragen. Die benötigte Rechenzeit für eine Inferenz skaliert unter anderem mit der Verbindungsanzahl beziehungsweise mit den notwendigen Multiplikationen. Eine detailliertere Laufzeitanalyse folgt in Unterkapitel 7.2.

Die höchste, bisher erreichte TPR beträgt 88,1% und wird von einem 6-18-6 Autoencoder erreicht, wobei andere Topologien ähnlich gute Ergebnisse liefern. Tabelle 6.7 zeigt die dazugehörigen Ergebnisse je Anomalie-Typ.

Anomalie-Typ	Plateau	Pos. Step Plateau	Neg. Step Plateau	Pos. Peak	Neg. Peak	Pos. Ramp Plateau	Neg. Ramp Plateau
TPR [abs.]	12/20	18/20	17/20	14/20	19/20	17/20	19/20
Anomalie-Typ	Noise	Sine	Pos. Step Offset	Neg. Step Offset	Pos. Ramp Offset	Neg. Ramp Offset	*Summe*
TPR [abs.]	16/20	20/20	20/20	20/20	18/20	19/20	*229/260*

Tabelle 6.7: Erkannte Anomalien durch den leistungsstärksten 6-18-6 Autoencoder

Autoencoder mit 24 beziehungsweise 32 Ein-/Ausgangsneuronen erreichen die höchste TPR mit einer klassischen „Bottleneck“-Topologie (24-23-24 und 32-24-32), liegen mit 85,8% und 80,8% allerdings etwas unterhalb des Maximums. Alle detaillierten Ergebnisse sind in Anhang A.5 zu finden.

Anomalie-Typ	Plateau	Pos. Step Plateau	Neg. Step Plateau	Pos. Peak	Neg. Peak	Pos. Ramp Plateau	Neg. Ramp Plateau
TPR [abs.]	12/20	18/20	17/20	14/20	6/20	13/20	18/20
Anomalie-Typ	Noise	Sine	Pos. Step Offset	Neg. Step Offset	Pos. Ramp Offset	Neg. Ramp Offset	*Summe*
TPR [abs.]	16/20	18/20	20/20	20/20	17/20	18/20	*207/260*

Tabelle 6.8: Erkannte Anomalien durch den leistungsstärksten 4-3-4 Autoencoder

Bild 6.4 enthält einen Datenpunkt, auf den bisher nicht näher eingegangen wurde. Der eingezeichnete 4-3-4 Autoencoder hat eine TPR von 79,6%. Er ist damit nicht der leistungsstärkste seiner Gruppe, erreicht aber das höchste Leistungsverhältnis LV, siehe Formel 6.1. Tabelle 6.8 zeigt die detaillierte Aufschlüsselung nach Anomalie-Typ. Im Vergleich zum insgesamt leistungsstärksten 6-18-6 Netz, ist die TPR um 8,5% verringert. Andererseits besitzt diese Topologie lediglich 24 $((4\cdot3)+(3\cdot4))$ statt 216 Kantengewichte, was sich in einer deutlich kürzeren Ausführungszeit bemerkbar macht, siehe Unterkapitel 7.2.
Entsprechend des zuvor definierten Vorgehens (siehe Bild 6.2), werden die 6-18-6 und die 4-3-4 Topologie nachfolgend um zwei versteckte Schichten erweitert und erneut untersucht. Ein-/Ausgangsschicht sowie die erste und letzte versteckte Schicht erhalten die Neuronenanzahl der ausgewählten To-

pologien. Die mittlere versteckte Schicht ist wiederum variabel gehalten und Gegenstand der empirischen Untersuchung.

Topologie	6-18-6-18-6	6-18-12-18-6	6-18-17-18-6	6-18-18-18-6	6-18-36-18-6	6-18-54-18-6	6-18-72-18-6
TPR (bei FPR = 0) [%]	83,5	88,5	87,3	86,9	88,8	**89,2**	88,5

Tabelle 6.9: Evaluierungsergebnisse für Autoencoder mit 6-18-x-18-6 Topologie

Anomalie-Typ	Plateau	Pos. Step Plateau	Neg. Step Plateau	Pos. Peak	Neg. Peak	Pos. Ramp Plateau	Neg. Ramp Plateau
TPR [abs.]	12/20	19(16)/20	18(17)/20	15(20)/20	18(19)/20	15(16)/20	20(18)/20
Anomalie-Typ	Noise	Sine	Pos. Step Offset	Neg. Step Offset	Pos. Ramp Offset	Neg. Ramp Offset	*Summe*
TPR [abs.]	19/20	20/20	20(19)/20	20/20	17/20	19/20	*232/260*

Tabelle 6.10: Erkannte Anomalien durch den leistungsstärksten 6-18-54-18-6 Autoencoder

Topologie	4-3-1-3-4	4-3-2-3-4	4-3-3-3-4	4-3-6-3-4	4-3-9-3-4	**4-3-12-3-4**	4-3-15-3-4
TPR (bei FPR = 0) [%]	14,6	56,2	69,2	79,6	79,2	**80,0**	79,2

Tabelle 6.11: Evaluierungsergebnisse für Autoencoder mit 4-3-x-3-4 Topologie

Anomalie-Typ	Plateau	Pos. Step Plateau	Neg. Step Plateau	Pos. Peak	Neg. Peak	Pos. Ramp Plateau	Neg. Ramp Plateau
TPR [abs.]	12/20	18/20	17/20	14/20	6/20	14/20	18/20
Anomalie-Typ	Noise	Sine	Pos. Step Offset	Neg. Step Offset	Pos. Ramp Offset	Neg. Ramp Offset	*Summe*
TPR [abs.]	16/20	18/20	20/20	20/20	17/20	18/20	*208/260*

Tabelle 6.12: Erkannte Anomalien durch den leistungsstärksten 4-3-12-3-4 Autoencoder

Tabelle 6.9 und 6.11 zeigen die Evaluierungsergebnisse für die untersuchten Topologien mit drei versteckten Schichten. In beiden Fällen steigt die maximal erreichte TPR im Vergleich zu den Netzen mit einer versteckten Schicht leicht an. Die detaillierten Ergebnisse der Autoencoder mit der

höchsten TPR sind in der jeweils darunter platzierten Tabelle 6.10 und 6.12 dargestellt. Unter Verwendung der 6-18-54-18-6 Topologie erreichen zwei trainierte Autoencoder die maximale TPR von 89,2%. Da sich die TPR unterschiedlich zusammen setzt, enthält Tabelle 6.10 die detaillierten Ergebnisse beider Netze. Sofern sich die Werte unterscheiden, stehen die des zweiten Netzes in Klammern. In Summe erkennen beide Instanzen drei Anomalien mehr als das 6-18-6 Pendant. Bei der 4-3-12-3-4 Topologie ist es eine erkannte Positive Ramp Plateau Anomalie mehr.

Eine Zusammenfassung der bisherigen Evaluierungen gibt Tabelle 6.13. Darin sind die erkannten Anomalien der jeweils leistungsstärksten Autoencoder aller untersuchten Topologie-Gruppen dargestellt. Die höchste TPR bei null Falschalarmen erreichen die 6-18-54-18-6 Netze. Diese ist mit 89,2% allerdings nur 1,1% höher als die des 6-18-6 Autoencoders. Gleichzeitig erhöht sich durch die hohe Anzahl an Kantengewichten die benötigte Rechenleistung für eine Inferenz. Sollte die Rechenleistung kein limitierender Faktor sein, kann die TPR durch ein inhomogenes Ensemble weiter erhöht werden. Für die untersuchten Daten wird die höchste TPR erreicht, wenn der 6-18-6, beide 6-18-54-18-6 und der 24-23-24 Autoencoder parallel arbeiten und jede Instanz direkt Anomalien meldet (kein Mehrheitsentscheid). Dieser Fall ist in der letzten Zeile von Tabelle 6.13 dargestellt. Das Ergebnis für das Ensemble setzt sich aus den jeweils höchsten Erkennungsraten für die einzelnen Anomalie-Typen zusammen. Unter der Annahme, dass sich die erkannten Anomalien je Autoencoder nicht vollständig überlappen, fällt die mögliche TPR noch höher aus[4]. Da alle gelisteten Instanzen keine Falschalarme auslösen, entsteht auf Basis der untersuchten Daten diesbezüglich kein negativer Seiteneffekt durch das Ensemble. Generell erhöht sich jedoch die Wahrscheinlichkeit eines Falschalarms. Um diese wieder zu reduzieren, kann ein Mehrheitsentscheid eingeführt werden - was allerdings

[4] Die Analyse der erkannten Anomalien erfolgte auf Basis der einzelnen Typen. Eine Aufschlüsselung beziehungsweise ein Vergleich auf Basis einzelner Anomalie-Instanzen stand nicht im Fokus der vorliegenden Arbeit.

	Plateau [x/20]	Pos. Step Plateau [x/20]	Neg. Step Plateau [x/20]	Pos. Peak [x/20]	Neg. Peak [x/20]	Pos. Ramp Plateau [x/20]	Neg. Ramp Plateau [x/20]	Noise [x/20]	Sine [x/20]	Pos. Step Offset [x/20]	Neg. Step Offset [x/20]	Pos. Ramp Offset [x/20]	Neg. Ramp Offset [x/20]	Summe [x/260]
2-12-2	3	13	11	8	3	14	18	9	18	16	19	18	18	168
3-12-3	10	18	16	12	17	16	19	17	19	20	20	18	19	221
4-3-4	12	18	17	14	6	13	18	16	18	20	20	17	18	207
4-3-12-3-4	12	18	17	14	6	14	18	16	18	20	20	17	18	208
4-20-4	11	18	16	12	18	16	19	16	19	20	20	17	19	221
6-18-6	12	18	17	14	19	17	19	16	20	20	20	18	19	229
6-18-54-18-6	12	19	18	15	18	15	20	19	20	20	20	17	19	232
6-18-54-18-6	12	16	17	20	19	16	18	19	20	19	20	17	19	232
8-32-8	11	16	16	14	18	17	19	16	20	20	20	17	18	222
12-36-12	10	18	17	14	16	17	20	15	20	19	20	17	20	223
16-48-16	10	16	17	16	16	16	20	19	20	19	20	18	20	227
24-23-24	10	17	18	15	9	16	20	20	20	20	20	18	20	223
32-24-32	9	16	16	13	5	16	20	19	20	18	20	18	20	210
Ensemble	12	19	18	20	19	17	20	20	20	20	20	18	20	243

Tabelle 6.13: Erkannte Anomalien durch die jeweils leistungsstärksten Autoencoder einer Gruppe

erneut die TPR beeinflusst. D.h. eine Anomalie wird zum Beispiel nur gemeldet, wenn mindestens zwei der vier Autoencoder eine solche erkennen. Bei limitierten Ressourcen ist im vorliegenden Anwendungsfall die Verwendung des 6-18-6 Autoencoders eine mögliche Alternative, ohne große Abstriche bei der TPR hinnehmen zu müssen. Sollten noch weniger Ressourcen vorhanden sein, steht das 4-3-4 Netz mit einer TPR von 79,6% zur Verfügung. Auch unter Verwendung des 6-18-54-18-6 Autoencoders als neue Referenz, besitzt dieses Netz weiterhin das höchste Leistungsverhältnis.

Anhand von Tabelle 6.13 ist zu erkennen, dass der Anomalie-Typ *Plateau* allgemein am schlechtesten detektiert wurde. Dies lässt sich durch dessen Definition erklären. Das Einfrieren eines Werts lässt sich nur dann erkennen, wenn zuvor eine hohe Dynamik im Signalverlauf vorhanden war. Eine Plateau-Anomalie während einer konstanten Autobahnfahrt mit Tempomat ist nicht oder nur sehr schwer zu erkennen. Im Gegensatz dazu ist die Erkennungsrate bei *Sine*, *Positive Step Offset* und *Negative Step Offset* Anomalien überdurchschnittlich hoch. Hier könnte in Zukunft die Parametrierung des Anomalie-Typs angepasst werden, um die Grenzen der einzelnen Topologien besser evaluieren zu können.

Die bisherige Evaluierung der unterschiedlichen Topologien basiert auf den Validierungsdaten. Um eine finale Bewertung abgeben zu können, wurden die selektierten Autoencoder nochmals mit den bisher nicht verwendeten Testdaten untersucht. Die detaillierten Ergebnisse sind in den Tabellen 6.14, 6.15 und 6.16 dargestellt. Für die 4-3-4 Topologie ergibt sich unter Verwendung der Testdaten eine TPR von 78,5% (Validierungsdaten: 79,6%), bei weiterhin keinem Falschalarm. Bei den Erkennungsraten einzelner Anomalie-Typen gibt es Unterschiede. Basierend auf den Testdaten werden mehr Anomalien der Typen *Plateau*, *Positive Step Plateau*, *Negative Step Plateau* und *Negative Ramp Offset* erkannt. Auf der anderen Seite fällt die Erkennungsrate von *Positive Peak*, *Negative Ramp Plateau*, *Noise*, *Sine* und *Positive Ramp Offset* geringer aus. In Bild 6.14 sind die Abweichun-

gen entsprechend markiert (grün: Höhere Erkennungsrate; rot: Geringere Erkennungsrate im Vergleich zu den Validierungsdaten).

Anomalie-Typ	Plateau	Pos. Step Plateau	Neg. Step Plateau	Pos. Peak	Neg. Peak	Pos. Ramp Plateau	Neg. Ramp Plateau
TPR [abs.]	14/20	19/20	20/20	11/20	6/20	13/20	16/20
Anomalie-Typ	Noise	Sine	Pos. Step Offset	Neg. Step Offset	Pos. Ramp Offset	Neg. Ramp Offset	*Summe*
TPR [abs.]	15/20	14/20	20/20	20/20	16/20	20/20	*204/260*

Tabelle 6.14: Erkennungsraten des leistungsstärksten 4-3-4 Autoencoders auf den Testdaten

Anomalie-Typ	Plateau	Pos. Step Plateau	Neg. Step Plateau	Pos. Peak	Neg. Peak	Pos. Ramp Plateau	Neg. Ramp Plateau
TPR [abs.]	14/20	19/20	19/20	10/20	12/20	14/20	19/20
Anomalie-Typ	Noise	Sine	Pos. Step Offset	Neg. Step Offset	Pos. Ramp Offset	Neg. Ramp Offset	*Summe*
TPR [abs.]	14/20	16/20	20/20	20/20	18/20	20/20	*215/260*

Tabelle 6.15: Erkennungsraten des leistungsstärksten 6-18-6 Autoencoders auf den Testdaten

Anomalie-Typ	Plateau	Pos. Step Plateau	Neg. Step Plateau	Pos. Peak	Neg. Peak	Pos. Ramp Plateau	Neg. Ramp Plateau
TPR [abs.]	15(14)/20	19(18)/20	19/20	14/20	11(16)/20	14/20	18(17)/20
Anomalie-Typ	Noise	Sine	Pos. Step Offset	Neg. Step Offset	Pos. Ramp Offset	Neg. Ramp Offset	*Summe*
TPR [abs.]	18/20	19/20	19(18)/20	20/20	17/20	20/20	*223(224)/260*

Tabelle 6.16: Erkennungsraten des leistungsstärksten 6-18-54-18-6 Autoencoders auf den Testdaten

Eine etwas höhere Abweichung ergibt sich beim 6-18-6 Autoencoder, welcher auf den Testdaten eine TPR von 82,7% (Validierungsdaten: 88,1%) bei keinem Falschalarm erreicht. Analog dazu verhalten sich die beiden Autoencoder mit 6-18-54-18-6 Topologie, siehe Tabelle 6.16 (beide besitzen auf den Validierungsdaten eine TPR von 89,2% bei null Falschalarmen).

Beide weisen unter Verwendung der Testdaten eine etwas verringerte TPR auf, 85,8% beziehungsweise 86,2%. Da die Algorithmenauswahl anhand der Validierungsdaten erfolgt, dürfen die Ergebnisse der Testdaten darauf keinen Einfluss haben. Aus diesem Grund wird eine weitere Metrik benötigt, die bei gleicher FPR und TPR eine Instanz auswählt - beispielsweise der geringere, durchschnittliche Reproduktionsfehler der Validierungsdaten ohne Anomalien. Nach dieser Metrik würde der 6-18-54-18-6 Autoencoder ausgewählt, welcher auf den Testdaten eine TPR von 85,8% erreicht und dabei keinen Falschalarm auslöst. Die Instanz mit 86,2% meldet hingegen zwei Falschalarme[5].

Da die Erkennungsraten einzelner Anomalie-Typen bei allen drei Topologien in die gleiche beziehungsweise in eine ähnliche Richtung abweichen (siehe farbliche Markierungen), lässt sich dieses Verhalten zum Großteil auf die unterschiedlichen Teildatensätze und die darin zufällig synthetisierten Anomalien zurückführen.

Positiv zu bewerten sind in allen Fällen die ähnlichen Ergebnisse zwischen Validierungs- und Testdaten. Dies zeigt eine gute Generalisierungsfähigkeit der trainierten Netze. Die zwei FP des zweiten 6-18-54-18-6 Autoencoders zeigen jedoch das Risiko einer geringen FPR im Realbetrieb, selbst wenn die Validierung ohne Falschalarm abgeschlossen wurde. Im angesprochenen Fall wäre es umgerechnet ein Falschalarm ungefähr alle vier Fahrstunden[6].

Des Weiteren ist die zu Beginn des Unterkapitels diskutierte Anpassung der Trainingsdaten zu berücksichtigen. Wären die aufgezeichnete Vollbremsung und die anhaltend starke Beschleunigung in den Validierungs- oder Testdaten enthalten, würden diese Fahrsituationen zusätzliche Falschalarme erzeugen. Zumindest zu Beginn ist die gleichmäßige Abdeckung aller Fahrsituationen in den Trainingsdaten unwahrscheinlich und es ist mit einer FPR

[5] Auch wenn einzelne Autoencoder aller betrachteten Topologien keine Falschalarme auf Validierungs- und/oder Testdaten aufweisen, zeigte sich während der empirischen Untersuchung, dass größere Netze eher Falschalarme erzeugen.

[6] Zwei Falschalarme bei 27.819 Testdatenpunkten und einer Abtastrate von einer Sekunde.

größer 0% zu rechnen. Da noch keine aktiven Gegenmaßnahen eingeleitet werden, wird davon ausgegangen, dass eine geringe Anzahl von Falschalarmen - beispielsweise ein Falschalarm pro Monat[7] - für ein initiales Ausrollen der Überwachung akzeptabel ist. Allerdings ist der damit erzeugte Nachbearbeitungsaufwand im Backend nicht zu vernachlässigen, weshalb weiterhin das Ziel von null Falschalarmen gilt. Wie bereits angedeutet ist eine potentielle Maßnahme das Aufzeichnen und Verwenden von mehr Daten, welche nach Möglichkeit alle normalen Fahrsituationen gleichermaßen abbilden. Dabei ist zu definieren, was als *normale* Fahrsituation angesehen wird. Ein Beispiel hierfür ist die in Unterkapitel 5.3 erwähnte Aufzeichnung durchdrehender Räder auf glatter Fahrbahn. Der dazugehörige Signalverlauf erzeugte den höchsten Reproduktionsfehler, wurde aber gleich zu Beginn als anormales Verhalten angesehen und die Datei dementsprechend aussortiert. Bei einem vergleichsweise kleinen Datensatz - wie die verwendeten OBD-II Daten - besteht die Gefahr, dass im Realbetrieb normale aber dennoch unbekannte Fahrsituationen auftreten und daher als Anomalie klassifiziert werden. Unter der Annahme, dass dies in der Realität immer der Fall ist, lösen mehr Daten das angesprochene Problem nicht vollständig, verringern aber dessen Auftrittswahrscheinlichkeit. Eine ergänzende Maßnahme zur Reduzierung der FPR ist das Einfügen eines Sicherheitsabstands beim verwendeten Schwellwert. Dieser könnte beispielsweise auf das 1,1- oder 1,2-Fache des maximalen Reproduktionsfehlers von Trainingsdaten gesetzt werden. Damit würde allerdings auch eine geringere TPR einhergehen, siehe die Diskussion am Anfang des Unterkapitels.

Ein direkter Vergleich der Leistungsfähigkeit mit den in Abschnitt 3.2.2 vorgestellten, LSTM-basierten Ansätzen zur Anomalieerkennung in CAN-Kommunikation ist nicht möglich. Die Autoren der entsprechenden Arbeiten verwenden sowohl andere Datensätze als auch unterschiedliche Ano-

[7] Ein Falschalarm pro Monat entspricht einer FPR von $\approx 1{,}9 \cdot 10^{-7}$, bei einer Zykluszeit der entsprechenden Kommunikationsnachricht von 20 ms und einer Fahrzeugnutzung von einer Stunde am Tag.

malien beziehungsweise Attacken als Grundlage für die jeweilige Evaluierung. Im Gegensatz zur vorliegenden Arbeit nutzen diese Ansätze außerdem Eingangsmerkmale auf Botschaftsebene - zum Beispiel die gesamten 64 Nutzdaten-Bits und nicht die daraus extrahierten Signalwerte. Deshalb, sowie aufgrund von teilweise unbekannten Hyperparametern und Trainingseigenschaften, ist ein unverändertes Anwenden der vorgestellten Ansätze auf die gesammelten OBD-II Daten ebenfalls nicht möglich[8].

6.3 Vergleich mit erster und zweiter Ableitung

Bei Betrachtung der Anomalie-Typen und deren Definitionen (siehe Unterkapitel 5.4) is zu erkennen, dass einige der erzeugten Anomalien einen betragsmäßig großen Wert der ersten und/oder zweiten Ableitung des Signals verursachen. Daher stellt sich die Frage, wie gut eine Anomalieerkennung rein auf Basis der ersten und/oder zweiten Ableitung des Signals abschneidet. Aufgrund des geringeren Ressourcenverbrauchs[9] - verglichen mit LODA und Autoencodern - wäre bei vergleichbaren Ergebnissen die Erkennung anhand der Ableitung(en) zu bevorzugen.

Des Weiteren reagiert auch ein Autoencoder sensibel auf große Änderungen im Signalverlauf und in dessen Steigung, wie das Beispiel in Bild 6.5 verdeutlicht. Im oberen Drittel ist ein originaler, darunter der modifizierte Signalverlauf dargestellt. Die hellgrau hinterlegten Bereiche markieren eingefügte Anomalien. Im unteren Drittel ist der Reproduktionsfehler inklusive des verwendeten Schwellwerts zu sehen, wobei detektierte Anomalien dunkelgrau hinterlegt sind. Bei genauerer Betrachtung fällt auf, dass

[8] Für einen aussagekräftigen Vergleich unterschiedlicher Verfahren ist ein standardisierter Datensatz, inklusive annotierter Anomalien, sowie eine einheitliche Auswertemethodik notwendig. Im Hinblick auf die Anomalieerkennung in Datenströmen beziehungsweise Zeitreihen existiert mit dem Numenta Anomaly Benchmark (NAB) ein entsprechender Ansatz [LA15] [Ahm+17]. In zukünftigen Arbeiten kann der vorgestellte, Autoencoder-basierte Ansatz mit dem NAB evaluiert und in dessen Rahmen mit anderen Algorithmen verglichen werden.

[9] Es müssten lediglich die drei letzten Signalwerte gespeichert werden, um anhand der Wertedifferenzen die erste und zweite Ableitung zu berechnen. Komplexere mathematische Operationen sind nicht notwendig, ebenso keine Multiplikationen.

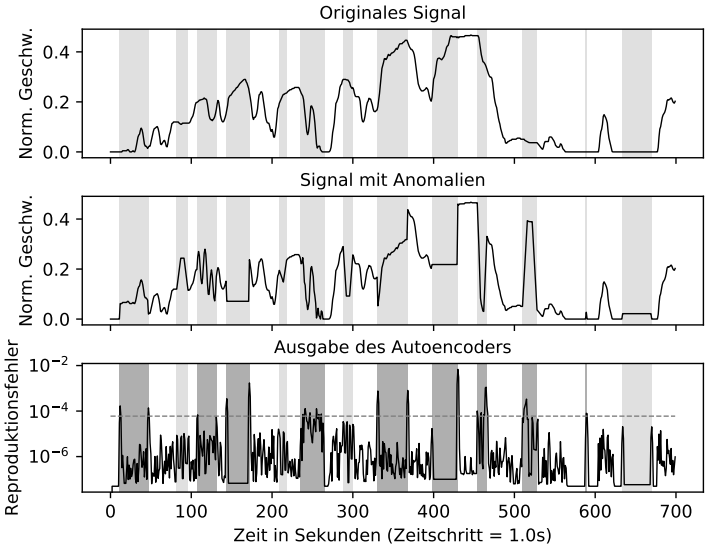

Bild 6.5: Reproduktionsfehler eines beispielhaften Autoencoders

der Reproduktionsfehler in Abschnitten mit betragsmäßig großer Signalsteigung oder großer Änderung der Steigung besonders hoch ausfällt. Deshalb ist auch aus diesem Gesichtspunkt ein Vergleich zur Anomalieerkennung mittels Schwellwerten für die erste und zweite Ableitung interessant.
Die Schwellwertbildung erfolgte nach dem gleichen Prinzip wie für LODA und Autoencoder. Zunächst wurden die Maximalwerte der ersten und zweiten Ableitung anhand der Trainingsdaten ermittelt. Diese dienen bei der Inferenz (d.h. bei Validierung und Test) als Schwellwert für die Klassifizierung, wobei ein betragsmäßig größerer Wert als Anomalie eingestuft wird. Hierbei ist es wichtig die positiven und negativen Werte getrennt voneinander zu betrachten, wie an folgendem Beispiel deutlich wird: Ein Fahrzeug kann in der Regel deutlich stärker verzögern als beschleunigen. Ein einzelner Maximalwert auf Basis des Betrags wäre üblicherweise nur für die

Verzögerung gültig und würde für die Beschleunigung unrealistische Werte zulassen.

Im Allgemeinen wurden die Anomalie-Typen so parametriert, dass auch Anomalien entstehen, die basierend auf den Ableitungen nicht zu erkennen sind, gegebenenfalls aber mit fortschrittlicheren Algorithmen, siehe Unterkapitel 5.4. Bei der Ergebnisinterpretation ist allerdings zu beachten, dass für zwei Anomalie-Typen die minimale Erkennungsrate bereits durch deren Parametrierung vorgegeben ist. Die Positive und Negative Step Offset Anomalien starten mit einem zufälligen Wertesprung im Bereich des 0,7- bis 2-Fachen des Maximalwerts der Trainingsdaten. Davon lassen sich durchschnittlich ca. 75% über die erste Ableitung detektieren, abhängig von der zufälligen Erzeugung. Für die Negative Step Offset Anomalien sind es im vorliegenden Fall sogar fast 100%. Der Schwellwert für die maximale negative Signaländerung liegt bei 0,086 im normalisierten Wertebereich. Damit werden Anomalien generiert, die einen Wertesprung zwischen -0,06 und -0,173 aufweisen. Am Ende der Anomalie findet der gleiche Wertesprung in positive Richtung statt. Der Schwellwert für die maximale positive Signaländerung liegt jedoch bei 0,064. D.h. alle Negative Step Offset Anomalien mit einem Werteabfall größer 0,064 werden durch den Rücksprung am Ende detektiert. Dies spiegelt sich auch in den Ergebnissen wieder, siehe Tabelle 6.17.

Anomalie-Typ	Plateau	Pos. Step Plateau	Neg. Step Plateau	Pos. Peak	Neg. Peak	Pos. Ramp Plateau	Neg. Ramp Plateau
TPR [abs.]	5/20	6/20	9/20	0/20	1/20	0/20	9/20
Anomalie-Typ	Noise	Sine	Pos. Step Offset	Neg. Step Offset	Pos. Ramp Offset	Neg. Ramp Offset	*Summe*
TPR [abs.]	6/20	16/20	13/20	19/20	10/20	13/20	*107/260*

Tabelle 6.17: Erkannte Anomalien durch Schwellwerte für die erste Ableitung

Die anderen Anomalie-Typen sind bezüglich der Ableitungen subtiler und deren Erkennung ist abhängig von unterschiedlichen Faktoren, wie beispielsweise dem originalen Signalverlauf unmittelbar vor und nach der Ano-

malie. Die Ergebnisse für die Validierungsdaten sind in den Tabellen 6.17 und 6.18 aufgelistet.

Anomalie-Typ	Plateau	Pos. Step Plateau	Neg. Step Plateau	Pos. Peak	Neg. Peak	Pos. Ramp Plateau	Neg. Ramp Plateau
TPR [abs.]	6/20	11/20	13/20	13/20	8/20	6/20	12/20
Anomalie-Typ	Noise	Sine	Pos. Step Offset	Neg. Step Offset	Pos. Ramp Offset	Neg. Ramp Offset	*Summe*
TPR [abs.]	10/20	13/20	16/20	20/20	11/20	14/20	*153/260*

Tabelle 6.18: Erkannte Anomalien durch Schwellwerte für die zweite Ableitung

Anomalie-Typ	Plateau	Pos. Step Plateau	Neg. Step Plateau	Pos. Peak	Neg. Peak	Pos. Ramp Plateau	Neg. Ramp Plateau
TPR [abs.]	6/20	11/20	13/20	13/20	8/20	6/20	12/20
Anomalie-Typ	Noise	Sine	Pos. Step Offset	Neg. Step Offset	Pos. Ramp Offset	Neg. Ramp Offset	*Summe*
TPR [abs.]	10/20	16/20	16/20	20/20	12/20	14/20	*157/260*

Tabelle 6.19: Erkannte Anomalien durch Schwellwerte für die erste und zweite Ableitung (Validierungsdaten)

In Tabelle 6.19 sind die Klassifizierungsergebnisse basierend auf der ersten und zweiten Ableitung zusammengefasst. Insgesamt werden 157 der 260 erzeugten Anomalien detektiert, was einer TPR von 60,4% entspricht. Positiv ist die FPR von 0%, welche jedoch nur gewährleistet ist, wenn alle Extremsituationen in den Trainingsdaten enthalten oder anderweitig definiert sind.

Anomalie-Typ	Plateau	Pos. Step Plateau	Neg. Step Plateau	Pos. Peak	Neg. Peak	Pos. Ramp Plateau	Neg. Ramp Plateau
TPR [abs.]	9/20	11/20	17/20	9/20	8/20	4/20	12/20
Anomalie-Typ	Noise	Sine	Pos. Step Offset	Neg. Step Offset	Pos. Ramp Offset	Neg. Ramp Offset	*Summe*
TPR [abs.]	11/20	11/20	17/20	20/20	14/20	19/20	*162/260*

Tabelle 6.20: Erkannte Anomalien durch Schwellwerte für die erste und zweite Ableitung (Testdaten)

Die Resultate zeigen, dass eine Schwellwert-basierte Anomalieerkennung auf Basis der ersten und zweiten Ableitung bis zu einem gewissen Grad möglich ist. TPR und FPR sind besser als die mit LODA erzielten Ergebnisse. Im Vergleich zum leistungsstärksten Autoencoder (6-18-54-18-6), ist die TPR allerdings um 28,8% geringer. Auch die vergleichsweise kleine 4-3-4 Topologie besitzt eine um 19,2% höhere TPR. Die FPR beträgt in allen Fällen 0%. Daher sind Autoencoder für den untersuchten Anwendungsfall weiterhin die bevorzugte Variante der Anomalieerkennung. Steht nur minimal Rechenleistung zur Verfügung, kann eine Überwachung der ersten und zweiten Ableitung dennoch einen nicht zu vernachlässigenden Mehrwert darstellen - zumal die Anwendung auf den Testdaten vergleichbare Ergebnisse lieferte (0% FPR), siehe Tabelle 6.20.

7 Realisierung in AUTOSAR Classic Steuergeräten

Im Folgenden werden verschiedene Aspekte bei der Realisierung des Observers in einem AUTOSAR Classic-basierten Steuergerät betrachtet. Nach der Vorstellung einer möglichen Software-Architektur, wird auf die prototypische Implementierung von Autoencodern zur Signalplausibilisierung im Rahmen einer Software-Komponente eingegangen. Der Fokus liegt in diesem Kapitel ausschließlich auf AUTOSAR Classic, wodurch der Zusatz *Classic* nachfolgend entfällt.

7.1 Integration in die AUTOSAR Software-Architektur

Das modulare Design des Observers soll sich nach Möglichkeit auch in dessen Implementierung wiederspiegeln (siehe [Req9] in Unterkapitel 4.2). Theoretisch könnte der Observer intern zwar modular aufgebaut sein aber dennoch als ein komplexer Treiber in einen AUTOSAR Software-Stack integriert werden, siehe Abschnitt 2.1.5 und Bild 2.12. Dies wäre jedoch aus mehreren Gründen nachteilig:

- Die diskutierte Modularität würde verloren gehen - zumindest aus Sicht des AUTOSAR Software-Stacks.
- Welche Teile des Observers im CDD implementiert sind, wäre lediglich auf Quellcode- und nicht auf Architekturebene sichtbar.
- Um eine zukünftige Standardisierung des Observers im Rahmen von AUTOSAR zu ermöglichen, ist eine modulare Software-Architektur

vorteilhaft. Die AUTOSAR Basissoftware selbst verfolgt ebenfalls einen modularen Ansatz, siehe 2.1.5, und alle Arbeitsabläufe sowie Werkzeuge sind darauf ausgelegt.

- Die Aufteilung in mehrere Module erhöht die Wartbarkeit und unterstützt gleichzeitig eine verteilte Entwicklung.

Aufgrund der genannten Punkte besteht die Software-Architektur des Observers aus mehreren Modulen, welche nachfolgend näher erläutert sind. Da jedes Modul seinerseits den Spezifikations- und administrativen Aufwand erhöht, wurde auf eine passende Granularität geachtet.

7.1.1 Überwachung der Kommunikation

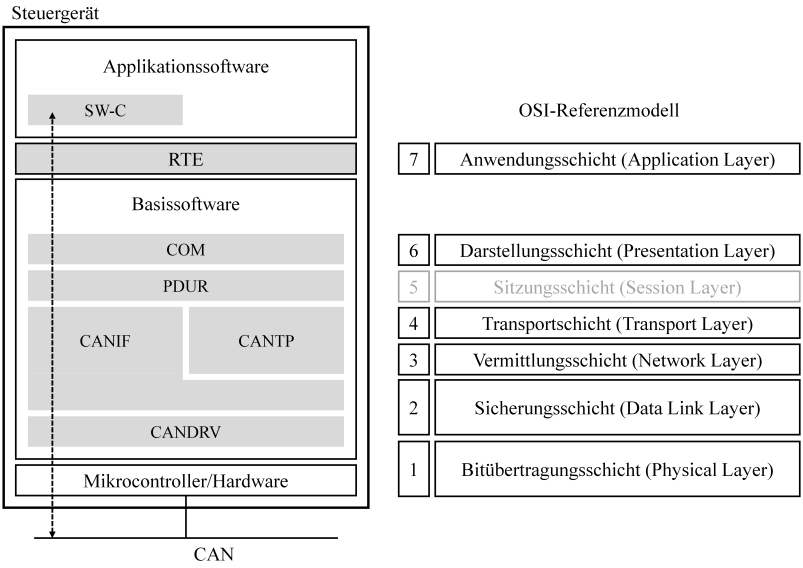

Bild 7.1: AUTOSAR CAN-Stack

Der linke Teil von Bild 7.1 zeigt die AUTOSAR Software-Architektur bezogen auf CAN-Kommunikation. Zum Vergleich ist rechts das OSI-Referenzmodell dargestellt, wobei Schicht fünf im abgebildeten Software-Stack nicht vorhanden ist. Auf unterster Softwareebene kümmert sich der CAN-Treiber (CANDRV) um die Ansteuerung der CAN-Hardware des Mikrocontrollers. Das darüber liegende CAN-Interface (CANIF) leitet die Nutzdaten einer empfangenen CAN-Botschaft als Interaction Layer Protocol Data Unit (I-PDU) mit einem eindeutigen Identifier (PDU-ID) an das PDU Router (PDUR) Modul weiter. Im Sendefall wird eine zu übertragende I-PDU in eine CAN-Botschaft verpackt und dem CANDRV übergeben. Der PDUR ist für das Stack-interne Verteilen von I-PDUs verantwortlich. Dies gilt sowohl in vertikaler als auch in horizontaler Richtung. Beim vertikalen Weiterleiten verbindet der PDUR das Communication Modul (COM) mit den Kommunikationsstacks für CAN, LIN, FlexRay und Ethernet. Im Fall eines Gateways kann der PDUR I-PDUs auch direkt zwischen Kommunikationsstacks (horizontal) weiterleiten, ohne Zutun der Applikationssoftware. Muss beispielsweise eine I-PDU von einem CAN-Netzwerk auf ein anderes geroutet werden, übergibt der PDUR die vom CANIF empfangene I-PDU direkt wieder dem CANIF. Anhand der internen PDU-ID entscheidet das CANIF welcher CANDRV die CAN-Botschaft zu versenden hat.
Bisher wurden die CAN-Nutzdaten als ein Datenblock betrachtet (I-PDU). Das Extrahieren beziehungsweise Verpacken der einzelnen Signale ist Aufgabe des COM. Dieses Modul bildet gleichzeitig die Schnittstelle der Basissoftware zur RTE. Die darüber liegenden SW-Cs empfangen und versenden somit ausschließlich abstrakte Signale, ohne Wissen über I-PDUs oder Kommunikationsnachrichten.
Anhand der vorgestellten Funktionsweise des AUTOSAR Software-Stacks wird eine erste Aufteilung der Observer-Implementierung vorgenommen. Die für Botschafts-basierte Checks notwendigen Informationen stehen teilweise nur im CANDRV oder im CANIF zur Verfügung, wie beispielsweise das Quellnetzwerk. Das CANIF wird aus Steuergeräte-Sicht konfigu-

riert und verarbeitet alle erlaubten CAN-Botschaften unabhängig vom Bussystem. Empfängt ein Steuergerät eine Botschaft zum Beispiel auf dem Powertrain-CAN, obwohl diese normalerweise nur vom Body-CAN zu empfangen ist, leitet das CANIF dennoch die passende I-PDU an den PDUR weiter. Aus Security beziehungsweise Observer-Sicht ist dies eine wichtige Information und lässt gegebenenfalls auf einen Angriff schließen. Des Weiteren verwerfen moderne CAN-Controller formal invalide Pakete bereits in der Hardware, sodass diese in der Software nicht sichtbar sind. Stattdessen erlaubt die Hardware in der Regel das Auslesen entsprechender Fehlerzähler. Die dafür notwendigen Schnittstellen bietet wiederum nur der CAN-DRV. Module oberhalb des CANIF dürfen diese Schnittstellen aus Architektursicht nicht verwenden. Aus den genannten Gründen werden alle Checks, welche auf Basis von Botschaften arbeiten, in einer unteren Softwareebene durchgeführt. Konkret handelt es sich um die Formality, Location, Frequency, Correlation und Protocol Sensoren (siehe *Payload-Inspection* in Tabelle 3.1).

Ein weiteres Argument für die Implementierung der Botschafts-basierten Checks in einer hardwarenahen Softwareschicht ist ein besserer Schutz des Steuergeräts. Sollten zukünftig Gegenmaßnahmen ergriffen werden, wie das Blockieren von Botschaften, ist ein frühes Eingreifen vorteilhaft. Damit lässt sich zum einen das Ausnutzen potentieller Fehler in der Basissoftware verhindern. Zum anderen benötigt das Prozessieren jeder fehlerhaften Botschaft Rechenzeit. Diese lässt sich durch den gewählten Ansatz minimieren. Wenn ein Angreifer mit einer Denial-of-Service Attacke eine Überlastsituation erzeugen möchte, würde ein Steuergerät in kurzer Zeit sehr viele Botschaften empfangen. Verwirft man diese beispielsweise erst auf Applikationsebene, wäre die dafür benötigte Rechenzeit beträchtlich, da alle Basissoftware-Module die Botschaften regulär verarbeiten. Durch das Blockieren in einer hardwarenahen Softwareschicht lässt sich dieser Rechenaufwand minimieren und es erhöht sich die Chance, dass die eigentliche Steuergeräte-Funktionalität erhalten bleibt.

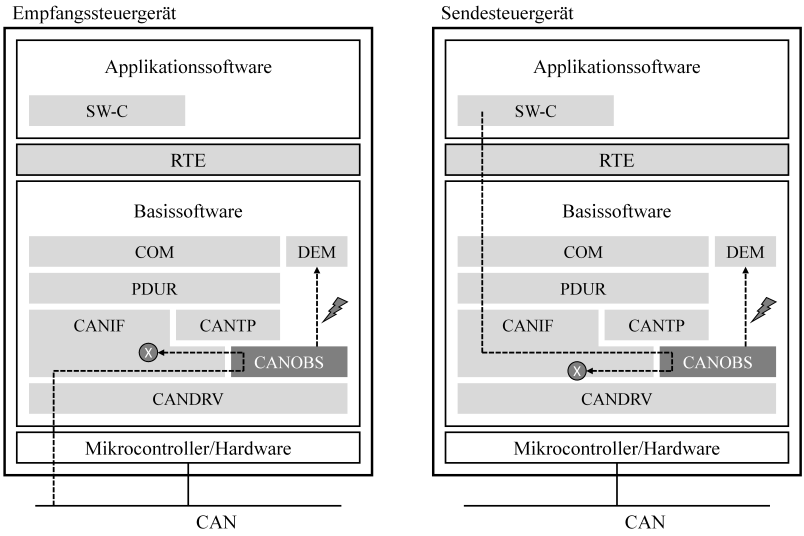

Bild 7.2: AUTOSAR CAN-Stack mit Observer-Anteilen

Zur Minimierung des Realisierungsaufwands wird eine Hardware-unabhängige Implementierung bevorzugt. Jeder CANDRV ist spezifisch für einen CAN-Controller und stellt ein standardisiertes API zur Verfügung. Um nicht jeden einzelnen Treiber anpassen zu müssen, bietet sich ein neues Software-Modul direkt oberhalb des Treibers an. Dort existiert momentan nur das CANIF. Unter Berücksichtigung der Rückwärtskompatibilität und um weiterhin Software-Stacks ohne Observer zu ermöglichen, werden die Schnittstellen zwischen CANDRV und CANIF nicht angepasst. Stattdessen befindet sich das CAN-spezifische Observer-Modul (CANOBS) parallel zum CANIF. Die resultierende Software-Architektur ist in Bild 7.2 dargestellt.

Im Empfangsfall ruft das CANIF zuerst das CANOBS auf, welches die entsprechenden Checks implementiert und entdeckte Anomalien dem DEM meldet, siehe linke Seite von Bild 7.2. Dieses Modul verwaltet in AUTO-

SAR den klassischen Fehlerspeicher. Gemeldete Anomalien lassen sich somit über die reguläre Fahrzeugdiagnose (Unified Diagnostic Services, UDS) auslesen. Im Rahmen des Konzepts *Security Extensions* für das AUTOSAR Classic Release 4.4 wurde der DEM um ein Security Event Memory (SEM) erweitert, welches in Kombination mit dem Non-volatile RAM Manager (NvM) das sichere Ablegen von Security-relevanten Events erlaubt. Damit sind die gespeicherten Anomalien zusätzlich gegen nachträgliche Manipulation geschützt. Sollten Gegenmaßnahmen erlaubt sein, kann das CANOBS Modul dem CANIF mitteilen, ob die Botschaft weiter zu prozessieren oder zu verwerfen ist (weißes X in Bild 7.2).
Die rechte Seite von Bild 7.2 zeigt den Sendefall. Kurz vor dem Übergeben einer CAN-Botschaft an den entsprechenden Treiber prüft der Observer, ob ein Versand erlaubt ist. Analog zum Empfangsfall werden zunächst erkannte Anomalien nur geloggt. Eine Besonderheit ist im Sendefall zu beachten: Mit der Übergabe einer Botschaft an den Treiber ist deren Versand noch nicht sichergestellt. Es existieren Möglichkeiten, um das Versenden gewollt oder ungewollt zu unterbinden. Der tatsächliche Versand wird über eine sogenannte Transmission Confirmation (TX-Confirmation) bestätigt. Diese sollte dem CANOBS ebenfalls zugeführt werden. Damit sind genauere Analysen möglich, beispielsweise über die Botschaftsfrequenz, was potentielle Falschalarme verhindern kann.
Für die Überwachung der Steuergeräte-externen Kommunikation wurde beispielhaft die Integration des Observers in den CAN-Stack diskutiert. Das Prinzip lässt sich auf die LIN-, FlexRay- und Ethernet-Stacks übertragen. Damit ergibt sich je ein Netzwerk-spezifisches Observer-Modul (CANOBS, LINOBS, FROBS und ETHOBS), welches die Checks auf Nachrichtenebene implementiert. Neben den erwähnten Vorteilen spricht auch die Verarbeitung der unterschiedlichen Nachrichtentypen und deren Eigenschaften für diese Aufteilung. Während das CANOBS CAN-ID und DLC auszuwerten hat, benötigt das ETHOBS beispielsweise Informationen über das VLAN sowie über MAC- und IP-Adressen. Die Wiederverwendung von Quellcode

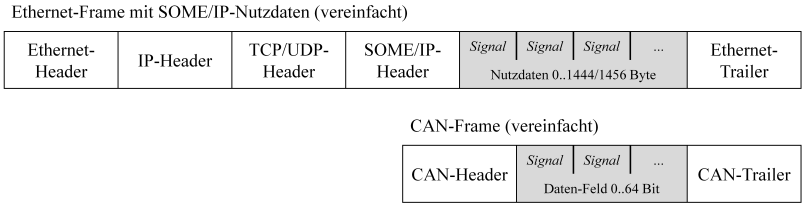

Bild 7.3: Vereinfachte Darstellung eines Ethernet- und eines CAN-Frames

ist damit begrenzt.
Bisher wurden die Signal-basierten Überwachungen mittels Range, Plausibility und Consistency Sensoren (siehe *Payload-Inspection* in Tabelle 3.1), nicht berücksichtigt. Deren Implementierung könnte ebenfalls in den Netzwerk-spezifischen Observer-Modulen erfolgen, was allerdings zwei Nachteile mit sich bringen würde:

1. Im Gegensatz zu Nachrichten sind die darin enthaltenen Signale Netzwerk-unabhängig. Ein Signal repräsentiert eine spezifische Information, welche der Applikation zur Verfügung steht, zum Beispiel die Fahrzeuggeschwindigkeit. Es kann Bestandteil mehrerer Nachrichten sein und in unterschiedlichen Netzwerken versendet werden[1]. Diese Eigenschaft erlaubt eine effiziente, zentrale Implementierung der entsprechenden Checks. Anderenfalls müssten alle Netzwerk-spezifischen Observer-Module gleichartige Überwachungen realisieren. Zusätzlich wäre ein weiteres Modul für die Steuergeräte-interne Kommunikation notwendig.

2. Dieses Design würde einen erhöhten Ressourcenverbrauch verursachen; nicht nur aufgrund der möglichen Mehrfachimplementierung von Checks, sondern auch durch die erforderliche Signalextraktion.

[1] In AUTOSAR verwenden die Software-Komponenten der Applikation ausschließlich Signale. Das COM-Modul verpackt diese bei Bedarf in I-PDUs, welche der PDUR an die entsprechenden Kommunikationsstacks weiterleitet.

In den Netzwerk-spezifischen Observer-Modulen müssten alle Kommunikationsprotokolle - zumindest rudimentär - verarbeitet werden, um die Signale aus den Nutzdaten extrahieren zu können. Eine zweite Implementierung der Kommunikationsstacks wäre die Folge. Dies ist besonders bei Ethernet, unter Verwendung von TCP oder der im automotive Umfeld beliebten Service-oriented Middleware over IP (SOME/IP), kaum darstellbar. Der obere Teil von Bild 7.3 zeigt den vereinfachten Aufbau eines Ethernet-Frames mit SOME/IP-Nutzdaten. Im unteren Teil von Bild 7.3 ist die vereinfachte Darstellung eines CAN-Frames zu sehen. Hier müssen die Signale entsprechend der Kommunikationsmatrix aus den Nutzdaten extrahiert werden. Dies ist ebenfalls eine rechenintensive Aufgabe. Außerdem ist bereits das COM Modul exakt für diesen Anwendungsfall konzipiert und aufgrund der limitierten Ressourcen auf einem Steuergerät sollte keine Aufgabe doppelt ausgeführt werden, siehe auch [Req11] in Unterkapitel 4.2.

Unter Berücksichtigung der beiden zuvor genannten Punkte erfolgt die Implementierung der Observer-Anteile zur Signalüberwachung (SIGOBS) oberhalb des COM Moduls. Alternativ ist auch eine Realisierung nach dem gleichen Prinzip wie bei den Netzwerk-spezifischen Modulen möglich (siehe Interaktion CANIF und CANOBS). Um dasselbe SIGOBS-Modul auch für Steuergeräte-interne Kommunikation verwenden zu können, ist eine Umsetzung innerhalb oder auf gleicher Ebene mit der RTE die bevorzugte Lösung. Dies wäre sowohl für die Basissoftware als auch für die Applikationssoftware transparent, wie die linke Seite von Bild 7.4 zeigt. Auch ein späteres Blockieren von Signalwerten ist realisierbar. Der Quellcode der RTE wird jedoch fast vollständig aus der Steuergerätekonfiguration generiert. Daher erfolgt die Umsetzung der Signal-basierten Checks für die nachfolgende Untersuchung innerhalb einer SW-C, welche sich zwischen

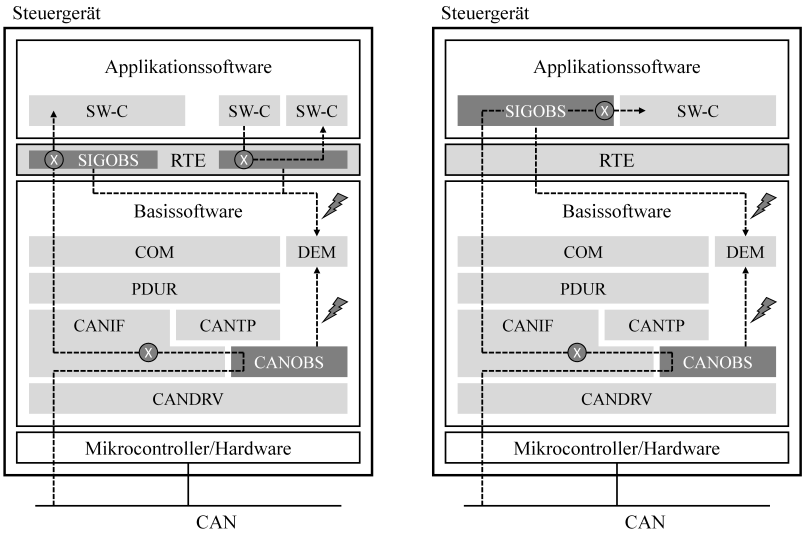

Bild 7.4: Signal-basierte Überwachung in der RTE beziehungsweise als SW-C

der eigentlichen Applikationssoftware und der RTE befindet. Dieses Vorgehen zeigt die rechte Seite von Bild 7.4.

Überwachung der Steuergeräte-internen Kommunikation

Die AUTOSAR Software-Architektur ermöglicht die Überwachung der Kommunikation zwischen SW-Cs innerhalb der RTE, siehe linke Seite von Bild 7.4. Da hierbei keine Nachrichten-basierten Checks vorgelagert sind (siehe CANOBS in Bild 7.4), bieten einige der entsprechenden Sensoren in diesem Fall auch auf Signalebene einen Mehrwert.

- AUTOSAR erlaubt das Versenden und Empfangen von dynamisch langen Arrays auf Signalebene. Die zur Laufzeit verwendete Anzahl von Array-Elementen kann ein angepasster Formality Sensor prüfen,

vergleichbar mit der Länge der Nutzdaten einer Kommunikationsnachricht.

- Die RTE generiert SW-C-spezifische Schnittstellen, welche das Schreiben und Lesen fremder Signale in der Regel verhindern. Um API-Hijacking, d.h. das Verwenden von APIs fremder SW-Cs, zu entdecken, kann dennoch ein angepasster Location Sensor zum Einsatz kommen.

- Mit einem entsprechenden Frequency-Sensor lässt sich das zu häufige oder zu seltene Lesen beziehungsweise Schreiben von Signalwerten erkennen.

- Das Versenden von Signalen in unterschiedlichen Nachrichten sowie Signal-Routings sind in der RTE und in der Applikation üblicherweise nicht sichtbar. Diese Funktionen realisiert die Basissoftware eigenständig. Daher enthält der SIGOBS keinen (angepassten) Correlation Sensor.

- Der Protocol Sensor stellt wiederum eine Ergänzung der Signal-basierten Checks dar, indem er Kommunikationsmuster zwischen einzelnen SW-Cs überwacht. Da die Sequenzen vorab nicht (semi-)formal spezifiziert werden, bieten sich hierfür lernende Checks an. In einer generalisierten Form handelt es sich eher um einen neuen *Sequence Sensor*. Dieser kann auch auf Basis von Nachrichten arbeiten und dort - speziell für die Überwachung von sporadischer Kommunikation - einen Mehrwert bieten.

7.1.2 Überwachung des Steuergeräte-internen Verhaltens

Zum Steuergeräte-internen Verhalten zählen die Abarbeitungsreihenfolge, die Laufzeit und der Speicherverbrauch von Tasks, Runnables und Funktionen sowie die Interaktion von Applikations- und Basissoftware über Kon-

trollschnittstellen[2]. Für die Überwachung der genannten Punkte benötigt der Observer Zugriff auf Informationen des Betriebssystems und der RTE. Das Betriebssystem verwaltet die einzelnen Tasks sowie die dazugehörigen Speicherbereiche (zum Beispiel die Stacks). Auf der anderen Seite ist der sogenannte Schedule Manager (SCHM) Teil der RTE. Er implementiert die einzelnen Tasks und ist für das Scheduling von Funktionen und Runnables zuständig. Sowohl Betriebssystem als auch RTE bieten ein API, um Scheduling-Informationen zu erhalten. Dazu zählen beispielsweise die *Post-* und *PreTask-Hooks*. Diese können verwendet werden, um sich über einen Scheduling-Vorgang notifizieren zu lassen. Weitere Schnittstellen und die exakte Anbindung des Observers an das Betriebssystem beziehungsweise die RTE stehen nicht im Fokus der vorliegenden Arbeit.
Architektonisch ergeben sich zwei weitere Observer-Module: Als Teil der Basissoftware überwacht das SYSOBS-Modul das Scheduling (Abarbeitungsreihenfolge) sowie Laufzeit und Speicherverbrauch auf unterschiedlichen Abstraktionsebenen (Tasks, Runnables und Funktionen). Es verwendet die entsprechenden Schnittstellen zu Betriebssystem und RTE. Da jegliche Interaktion der SW-Cs untereinander und mit der Basissoftware über die RTE stattfindet, ist die Überwachung der Kontrollschnittstellen in ein separates CALLOBS-Modul ausgelagert. Es befindet sich innerhalb der RTE, analog zum SIGOBS. Relevant sind hierbei der Frequency- und der in Abschnitt 7.1.1 diskutierte Sequence Sensor.

7.1.3 Software-Architektur des Observers

Eine Zusammenfassung der eingeführten Observer-Module und des resultierenden AUTOSAR Software-Stacks zeigt Bild 7.5. Aus Übersichtlichkeitsgründen sind lediglich die Kommunikationsstacks für CAN und

[2] Kontrollschnittstellen sind APIs, über welche keine Signalwerte versendet oder empfangen werden. Stattdessen bieten sie Zugriff auf Dienste der Basissoftware, wie beispielsweise das Schreiben und Lesen von nicht-flüchtigem Speicher, oder dienen der Ablaufsteuerung.

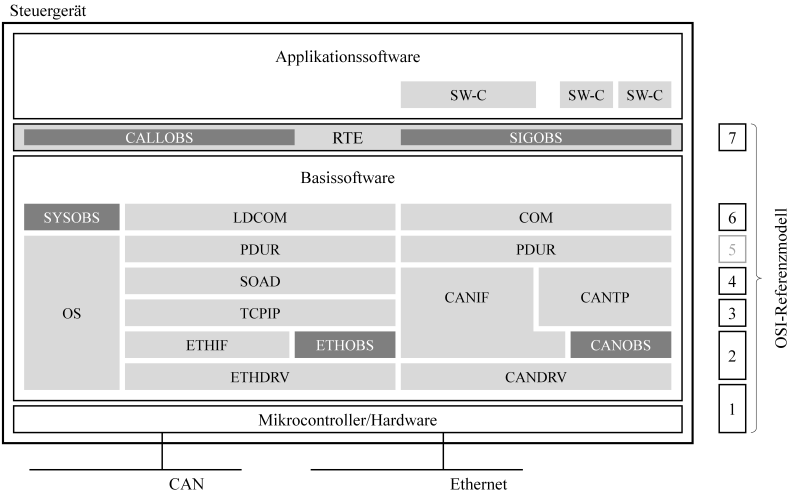

Bild 7.5: Resultierende Software-Architektur mit Observer

Ethernet [3] dargestellt. Ebenso fehlt die jeweilige Anbindung der Observer-Module an den DEM. In Bezug auf die Kommunikationsstacks sind auf der rechten Seite nochmals die dazugehörigen Schichten des OSI-Referenzmodells abgebildet.

Jeder Kommunikationsstack erhält ein entsprechendes Observer-Modul in der ersten Hardware-unabhängigen Softwareschicht, dargestellt als CAN-

[3] Der Ethernet Treiber (ETHDRV) ist für die Ansteuerung der entsprechenden Hardware-Peripherie verantwortlich. Das darüber liegende Ethernet Interface (ETHIF) verwaltet unter anderem VLANs und stellt entsprechende Schnittstellen für den Versand und Empfang von Ethernet Nutzdaten zur Verfügung. Anders als bei CAN enthalten die Nutzdaten weitere Protokolle, angefangen bei IPv4/IPv6 bis hin zu TCP und UDP. Diese Protokolle verarbeitet der TCP/IP-Stack beziehungsweise das TCPIP Modul. Wie für den CAN-Stack erläutert, verwenden die höheren Softwareschichten I-PDUs mit eindeutigen Kennungen zur Stack-internen Weitergabe von Daten. Ein klassischer TCP/IP-Stack arbeitet hingegen mit sogenannten Sockets als Kommunikationsendpunkte. Das Zusammenführen dieser beiden Paradigmen ist Aufgabe des Socket Adaptor (SOAD) Moduls. Dieses verwendet wiederum I-PDUs an den bekannten Schnittstellen zum PDUR. Da Ethernet in der Regel mit großen Nutzdatenmengen arbeitet - im Vergleich zu maximal acht Byte langen I-PDUs bei CAN - und die Kommunikation oftmals mittels SOME/IP Service-orientiert stattfindet, wurde das Large Data Communication (LDCOM) Modul eingeführt, welches eine vereinfachte Signalschnittstelle zur RTE realisiert.

OBS und ETHOBS. In der RTE finden sich die Module für die Signal- und Kontrollfunktions-Überwachung (SIGOBS und CALLOBS). Das SYSOBS Modul implementiert die Checks bezüglich Scheduling, Laufzeit und Speicherverbrauch.
Alle Module implementieren das Konzept nach Bild 4.4. Da die statischen Checks abhängig von den jeweiligen Eingangsdaten sind, ist deren Aufteilung auf die verschiedenen Module intuitiv möglich. Die lernenden Checks werden je nach Überwachungsgegenstand ebenfalls entsprechend aufgeteilt. Wie oben bereits beschrieben geschieht das Anomalie-Logging über den DEM.

7.2 Signalplausibilisierung in eingebetteten Systemen

Wie in Abschnitt 4.4.2 beschrieben fokussiert sich diese Arbeit auf die lernenden Checks zur Kommunikationsüberwachung und im Speziellen auf den Plausibility Sensor. Dazu wurden LODA sowie Autoencoder näher untersucht und anhand eines Realdatensatzes evaluiert, siehe Kapitel 5 und 6. Aufgrund der Evaluierungsergebnisse folgt im nächsten Schritt die Implementierung von Autoencodern zur Signalplausibilisierung in eingebetteten Umgebungen.

7.2.1 Auswahl des Codegenerators

Grundlage für die Implementierung ist die angereicherte Anomaly Detection Configuration, wie in Bild 4.3 dargestellt. Dort ist die vollständige Parametrierung der zu implementierenden Autoencoder enthalten. Ziel ist die vollständige Generierung des ANSI C-Quellcodes für die Inferenz auf einem Steuergerät, mit speziellem Fokus auf die benötigte Rechenzeit und den benötigten Speicherplatz (RAM und ROM). Außerdem soll der Code konform zum MISRA C Standard sein. Dieser definiert Programmierrichtlinien für C-Code in sicherheitskritischen Systemen und ist in der Version MISRA C:2012 [13] für AUTOSAR Classic Basissoftware vorgeschrieben

[AUT17e]. Direktive 4.12 untersagt beispielsweise die Verwendung von dynamischer Speicherverwaltung. Haupteinsatzgebiet sind Systeme mit funktionalen Sicherheitsanforderungen. Da für diese Systeme die Angriffssicherheit immer wichtiger wird, wurden drei Ergänzungen veröffentlicht. Amendment 1 führt zusätzliche Richtlinien ein [16]. Die beiden anderen vergleichen MISRA C mit den existierenden Secure Coding Richtlinien *C Secure* und *CERT C* [18a] [18b].

	Programmier-sprache	Fixkomma	ReLU	MISRA C
TensorFlow™	C++/Python	✓	✓	×
FANN	C	✓	×	×
SNNS	C	×	×	(✓)

Tabelle 7.1: Vergleich existierender Frameworks für die Inferenz von neuronalen Netzen

Im Rahmen der vorliegenden Arbeit wurden folgende, bereits existierende Frameworks für die Inferenz von neuronalen Netzen auf embedded Systemen untersucht, siehe auch Tabelle 7.1: TensorFlow™ (inklusive TensorFlow Lite), Fast Artificial Neural Network Library (FANN) [Nis31] und der Stuttgart Neural Network Simulator (SNNS) [Zel+94].
TensorFlow Lite ist eine spezielle Variante von TensorFlow™. Die C++-Bibliothek ist bezüglich Codegröße und Ausführungsgeschwindigkeit für den Einsatz auf embedded Systemen optimiert, wobei die Zielplattformen hauptsächlich Android und iOS sind. Inzwischen existiert auch eine Variante für den Raspberry Pi, welche zu Beginn dieser Arbeit noch nicht zur Verfügung stand. Deshalb basieren die nachfolgenden Untersuchungen und Ergebnisse auf der klassischen C++/Python-Implementierung von TensorFlow™. FANN ist ebenfalls eine Bibliothek und enthält zahlreiche embedded Optimierungen. Unter anderem ist FANN in C geschrieben und unterstützt Fixkommaarithmetik. Die bisher vorgestellten Frameworks ba-

sieren auf dem dynamischen Laden von neuronalen Netzen zur Laufzeit. Im Gegensatz dazu ist SNNS ein Simulator für neuronale Netze, der es erlaubt, statischen C-Code für die Inferenz zu generieren.
Die geforderte MISRA C Konformität kann nur mit SNNS erreicht werden, da TensorFlow™ und FANN Netze dynamisch laden. Auf der anderen Seite fehlt in SNNS aktuell die Unterstützung von Fixkommaarithmetik und ReLUs. Außerdem bietet das Framework noch Optimierungspotential bezüglich der Ausführungsgeschwindigkeit. Der notwendige Anpassungsaufwand wird als hoch eingestuft, da der Codegenerator selbst auch in C geschrieben ist. Daher fiel die Entscheidung, eine neue Lösung namens Generated Networks (GENET) zu entwickeln.

7.2.2 Funktionsweise von GENET

Im Kontext des Observer-Konzepts hat GENET die Aufgabe den Quellcode für die Inferenz von Autoencodern zu generieren, siehe Bild 7.6. Das Austauschformat für trainierte Autoencoder wurde als Teil der Anomaly Detection Configuration selbst definiert und nutzt JavaScript Object Notation (JSON) [Int17]. Es umfasst die Festlegung der einzelnen Schichten, der verwendeten Aktivierungsfunktionen (je Schicht) sowie der entsprechenden Kantengewichte. Ein Beispiel ist in Anhang A.6 zu finden. Für den Austausch trainierter neuronaler Netze existieren inzwischen auch von der Industrie spezifizierte Formate. Zwei Beispiele hierfür sind das Open Neural Network Exchange Format (ONNX) [Fac17] und das neu veröffentlichte Neural Network Exchange Format (NNEF) [Khr18]. Diese können ebenfalls alle benötigten Informationen abbilden, sodass eine Konvertierung des eigens definierten JSON-Formats möglich ist.
Neben der Anomaly Detection Configuration benötigt GENET implementierungsspezifische Parameter, um einen für die Zielplattform optimierten Quellcode zu generieren:

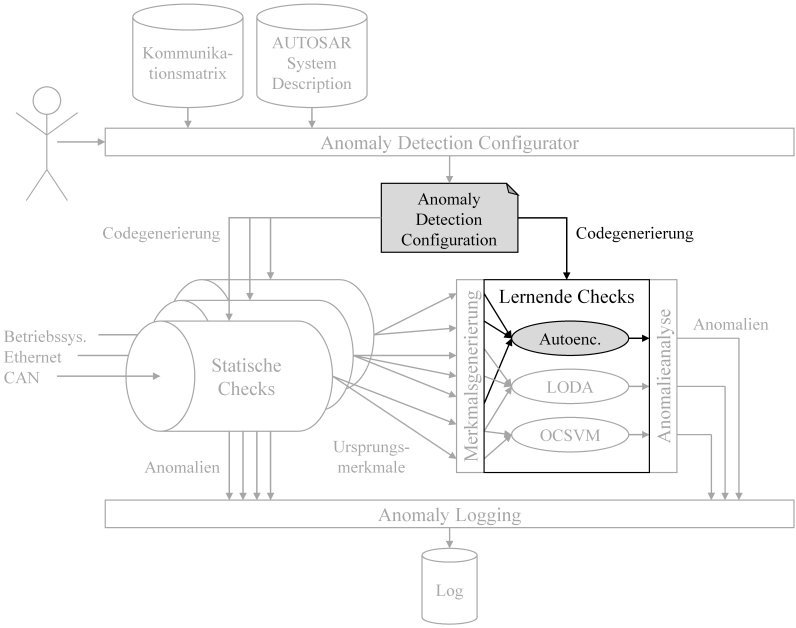

Bild 7.6: Einordnung von GENET in das Observer-Konzept

- *Gleit-/Fixkommaarithmetik*: Dieser Parameter bestimmt, ob der C-Code Fließkomma-Datentypen verwendet. Sollte die eingesetzte Hardware keine Fließkommaeinheit (engl. Floating Point Unit (FPU)) besitzen, empfiehlt sich die Verwendung von Fixkommaarithmetik. Selbst wenn eine FPU zur Verfügung steht, kann der Einsatz von Fixkommaarithmetik einen Performanzgewinn mit sich bringen. Siehe dazu die Auswertungen in Abschnitt 7.2.3.

- *Zielplattform*: Zur Optimierung des Quellcodes müssen die Endianness, die native Wortgröße und die Größe des L1-Datencaches der Zielplattform bekannt sein. Für eine reibungslose Integration in die Entwicklungsumgebung des Gesamtsystems, verwendet GENET zu-

sätzliche Parameter, wie beispielsweise die Namen benötigter, Plattformspezifischer Header-Dateien aus der Standardbibliothek.

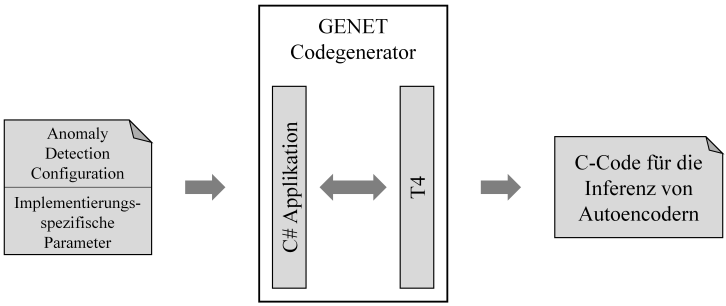

Bild 7.7: C-Code Generierung mit GENET

Der Codegenerator selbst besteht aus Text-Templates und einer dazugehörigen Applikationslogik. Für die Text-Templates kommt das Text Template Transformation Toolkit (T4) von Microsoft zum Einsatz. Die Applikationslogik ist in C# programmiert. Bild 7.7 zeigt den prinzipiellen Arbeitsablauf mit GENET. Der generierte C-Code ist auf optimale Laufzeitperformanz ausgelegt und verwendet unter anderem eine optimierte Matrixmultiplikation.

7.2.3 Laufzeitevaluierung auf einem Raspberry Pi 3

Für die Beurteilung der Einsatzmöglichkeiten von Autoencodern zur Signalplausibilisierung erfolgte eine Laufzeitmessung auf Basis von zwei unterschiedlichen Hardwareplattformen. Die erste Plattform ist ein Raspberry Pi 3 Model B und steht stellvertretend für leistungsfähige automotive Steuergeräte, wie beispielsweise Domänencontroller. Der eingesetzte Prozessor ist ein Broadcom BCM2837 mit vier 64 Bit ARM Cortex-A53 Kernen, die jeweils mit 1,2 GHz getaktet sind. Eine FPU ist in dieser Leistungsklasse ebenfalls Standard. Diese Prozessorarchitektur ist von ARM für automo-

tive Anwendungen vorgesehen [Tur]. Als Betriebssystem wurde Raspbian Lite Version 9 verwendet. Damit ist auch ein Vergleich mit den alternativ untersuchten Frameworks möglich, welche nicht auf C-Code basieren. Des Weiteren gelten die nachfolgenden Randbedingungen:

- Die Inferenz findet jeweils auf einem einzelnen CPU-Kern statt.
- Die Messdauer beträgt jeweils mindestens 7,5 Sekunden in denen mehrere Inferenzen berechnet werden. Das Resultat ist die durchschnittliche Berechnungszeit einer Inferenz.
- Die Messung der CPU-Zeit erfolgt über die C-Funktion `clock()` der Standardbibliothek [Int11b] beziehungsweise bei TensorFlow™ mit der `clock()` Funktion des Python-Moduls `time` [Pyt].
- Es kommen die Frameworks in folgenden Versionen zum Einsatz: TensorFlow™ Version 1.5.0, FANN Commit vom 29. November 2015 und SNNS Version 4.3.
- Alle Frameworks verwenden 32 Bit Fließkommaarithmetik.
- Alle Frameworks verwenden 64 Bit Fixkommaarithmetik, sofern unterstützt.
- Der Quellcode für FANN, SNNS und GENET wurde mit GCC Version 6.3.0 und maximaler Geschwindigkeitsoptimierung (`-o3`) kompiliert. Die Kompilierung von TensorFlow™ erfolgte auf einem Debian 9 System mit Linaro 6.3.1, ebenfalls mit maximaler Geschwindigkeitsoptimierung [Lon].
- ReLUs werden in FANN und SNNS nicht unterstützt. Im von SNNS generierten Code wurden die Sigmoid-Aktivierungsfunktionen manuell durch Rectifier ersetzt.

- Bei Verwendung von Fixkommaarithmetik in Kombination mit Sigmoid-Neuronen nutzt GENET die stückweise lineare Approximation (engl. Piecewise Linear Approximation of a Nonlinear Function (PLAN)) nach Amin et al. für die Aktivierungsfunktionen [ACH97][4]. FANN nutzt hierfür eine eigene stückweise lineare Approximation.

- Die konkreten Werte der Kantengewichte spielen bei der Laufzeitmessung keine Rolle.

8-7-8 Autoencoder-Topologie

Der erste Vergleich basiert auf einem Autoencoder mit einer 8-7-8 Topologie[5]. Damit umfasst eine Inferenz die Berechnung von 112 Multiplikationen (Übergang von der Eingangs- zur versteckten Schicht: $8 \cdot 7$ Kantengewichte/Multiplikationen; Übergang von der versteckten zur Ausgangsschicht: $7 \cdot 8$ Kantengewichte/Multiplikationen) und die Berechnung von 15 Aktivierungsfunktionen. Die Bildung des Reproduktionsfehlers und die nachgelagerte Anomaliebewertung ist nicht Teil der Messung. Die jeweils erste Hälfte von Bild 7.8 und der dazugehörigen Tabelle 7.2 zeigt die ermittelten Ergebnisse, einmal unter Verwendung von ReLUs (sofern vom Framework unterstützt) und einmal für Neuronen mit sigmoidaler Aktivierungsfunktion.
GENET schneidet durchweg besser ab als SNNS. Im Vergleich zu den anderen Frameworks ist lediglich TensorFlow™ bei sigmoidalen Aktivierungsfunktionen schneller.

Wie erwartet benötigt ein Autoencoder gleicher Topologie weniger Ausführungszeit, wenn ReLUs statt Neuronen mit sigmoidaler Aktivierungs-

[4] In einer Vorabuntersuchung unterschiedlicher Sigmoid-Approximationen, weißt die PLAN-Approximation den geringsten Root Mean Square Error (RMSE) auf.
[5] Der Vergleich auf Basis des Raspberry Pi 3 wurde in einer frühen Phase der vorliegenden Arbeit durchgeführt. Zu diesem Zeitpunkt stand die Topologie des leistungsstärksten Autoencoders und die Topologie mit dem größten Leistungsverhältnis noch nicht fest. Daher basiert der Vergleich auf zwei Netzen mittlerer Größe. Die Aussagekraft der Ergebnisse bleibt dadurch unverändert.

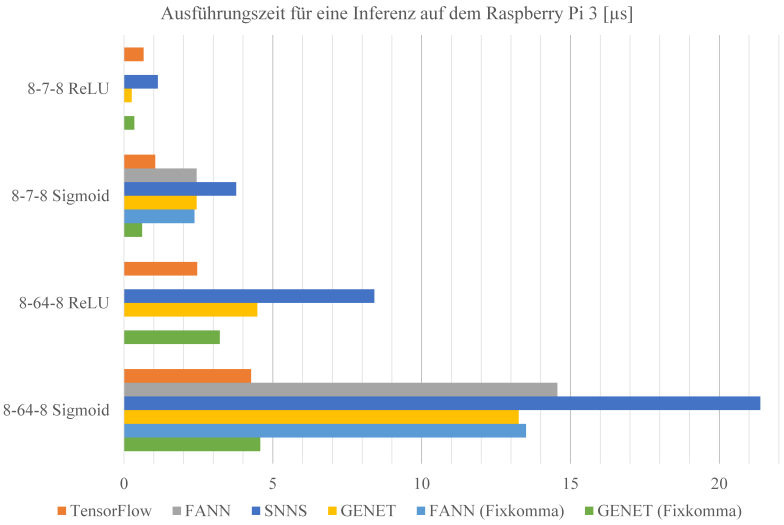

Bild 7.8: Ausführungszeiten für eine Inferenz auf dem Raspberry Pi 3

Ausführungszeit [µs]	TensorFlow™	FANN	SNNS	GENET	FANN (Fixkomma)	GENET (Fixkomma)
8-7-8 ReLU	0,66	n/a	1,14	0,26	n/a	0,35
8-7-8 Sigmoid	1,05	2,44	3,77	2,44	2,37	0,61
8-64-8 ReLU	2,46	n/a	8,41	4,48	n/a	3,22
8-64-8 Sigmoid	4,27	14,56	21,38	13,26	13,51	4,58

Tabelle 7.2: Ausführungszeiten für eine Inferenz auf dem Raspberry Pi 3

funktion verwendet werden. Auffällig ist der große Unterschied bei GENET unter Verwendung von Fließkommaarithmetik. Da die restlichen Hyperparameter gleich bleiben, ist der Unterschied einzig in der Aktivierungsfunktion begründet, welche somit auch den Hauptteil der Gesamtlaufzeit beansprucht. Findet die Berechnung in Fixkommaarithmetik statt, ist der Unterschied deutlich geringer. Dies liegt an der verwendeten PLAN-Approximation, welche die Sigmoidfunktion stückweise linear abbildet. Die vergleichsweise kurze Ausführungszeit des Autoencoders mit sigmoidalen

Aktivierungsfunktionen unter TensorFlow™ (Fließkommaarithmetik) lässt darauf schließen, dass hier bereits ähnliche Optimierungen vorhanden sind. Der geringe Laufzeitunterschied zwischen den beiden FANN Messungen ist einer aufwändigen Approximation der Sigmoidfunktion geschuldet.

8-64-8 Autoencoder-Topologie

Im zweiten Schritt wurde die versteckte Schicht auf 64 Neuronen vergrößert. Dadurch steigt die Anzahl an Multiplikationen für eine Inferenz von 112 auf 1024 ($8 \cdot 64 + 64 \cdot 8$) und die Anzahl an Aktivierungsfunktionen von 15 auf 72. Die Bildung des Reproduktionsfehlers ist weiterhin außen vor. In der zweiten Hälfte von Bild 7.8 und Tabelle 7.2 sind die ermittelten Ergebnisse dargestellt, wiederum für die beiden unterschiedlichen Aktivierungsfunktionen.
Unter Verwendung von Fixkommaarithmetik ist die Anzahl an Multiplikationen der entscheidende Laufzeitfaktor. Dies spiegelt sich auch in den Messergebnissen für GENET wieder. Bei der 9,14-fachen Multiplikationsanzahl und 4,8-mal mehr Aktivierungsfunktionen steigt die Laufzeit bei ReLUs um den Faktor 9,2. Die Berechnung der Sigmoid-Approximation ist aufwändiger und fällt daher mehr ins Gewicht. In diesem Fall steigt die Laufzeit um Faktor 7,5.
Unter Verwendung von Fließkommaarithmetik in Kombination mit ReLUs haben weitere Faktoren einen Einfluss auf die gemessene Laufzeit, sodass diese um das 17,2-Fache zunimmt. Eine genauere Analyse des überproportionalen Anstiegs wurde im Rahmen der vorliegenden Arbeit nicht durchgeführt. Aufgrund des Anteils der Sigmoidfunktionen an der Gesamtlaufzeit, skaliert die Fließkomma-Implementierung unter GENET wieder erwartungsgemäß. Bei der 4,8-fachen Anzahl an Aktivierungsfunktionen dauert eine Inferenz des entsprechenden Autoencoders um Faktor 5,4 länger. Der überproportionale Anstieg erklärt sich durch die 9,14-fache Anzahl an Multiplikationen.

Die Zunahme der Laufzeit, im Vergleich zur 8-7-8 Topologie, ist bei TensorFlow™ deutlich geringer als zunächst erwartet. Die Inferenz des 8-64-8 Autoencoders dauert lediglich um Faktor 3,76 (ReLUs), beziehungsweise um Faktor 4,07 (Sigmoid) länger. Dies lässt sich vor allem darauf zurückführen, dass TensorFlow™ intern die Automatically Tuned Linear Algebra Software (ATLAS) benutzt [Ten18b]. ATLAS ist eine Bibliothek und enthält für bestimmte Hardware-Plattformen optimierte Funktionen der linearen Algebra [WD98] [ATL]. Für den vorliegenden Anwendungsfall sind die Vektorisierungsmöglichkeiten moderner Prozessoren besonders vorteilhaft. Dabei werden mehrere skalare Operationen in einer Vektoroperation zusammengefasst und berechnet. Dadurch lassen sich beispielsweise Vektor- und Matrixmultiplikationen - wie sie bei der Berechnung von neuronalen Netzen notwendig sind - deutlich beschleunigen.

Bewertung GENET

In allen untersuchten Fällen ist GENET deutlich schneller als der SNNS, welcher als einzige Alternative zur Verfügung steht (siehe Abschnitt 7.2.1). Des Weiteren sind die ermittelten Laufzeiten bei Fließkommaarithmetik auf dem Niveau von FANN. Bei den entsprechenden Fixkommaberechnungen unterbietet GENET FANN um Faktor 3,9 beziehungsweise 2,9. In drei von vier Fällen hat TensorFlow™ die kürzesten Laufzeiten. Einzig bei der Inferenz des 8-7-8 Autoencoders mit ReLUs arbeitet GENET schneller.
Der dynamische, Bibliothek-basierte Ansatz von TensorFlow™ ist für den gegebenen Anwendungsfall ungeeignet. Jedoch zeigen die Messergebnisse, dass es bei GENET noch Potential zur weiteren Laufzeitoptimierung gibt.

7.2.4 Laufzeitevaluierung auf einem VC121

Da die Mehrheit der aktuell verbauten Steuergeräte deutlich weniger Rechenleistung besitzen als der bisher verwendete Raspberry Pi 3, wird im nächsten Schritt GENET auf einem Standard automotive Mikrocontroller

evaluiert. Zum Einsatz kommt ein SPC56EC64 der Firma ST Microelectronics mit Power Architektur®. Dieser 32 Bit Dual-Core Mikrocontroller ist unter anderem im Vector Controller 121 (VC121[6]) verbaut und dort mit 120 MHz getaktet. Analog zur vorherigen Untersuchung findet die Signalplausibilisierung mittels GENET auf einem Kern statt. Zusätzlich steht eine FPU zur Verfügung, welche allerdings für die Messungen deaktiviert wurde, um die Einsatzmöglichkeiten auf noch schwächeren Plattformen zu prüfen. Als Laufzeitumgebung dient MICROSAR 4 Release 19, die AUTOSAR Classic Implementierung der Vector Informatik GmbH.

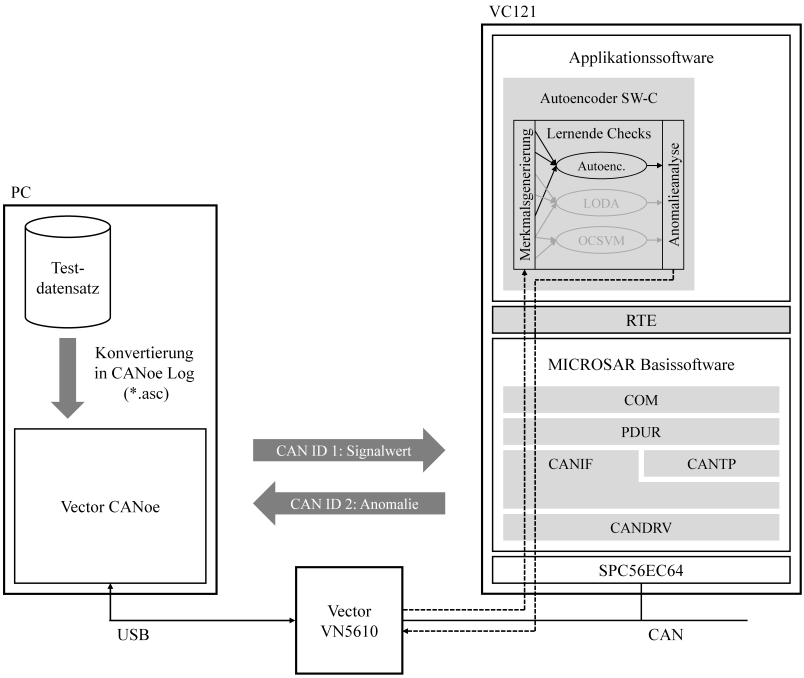

Bild 7.9: Messaufbau mit VC121

[6] Das VC121 ist ein Universalsteuergerät, welches speziell für den Einsatz in Prototypen und in Kleinserien entwickelt wurde.

Bild 7.9 zeigt den gesamten Messaufbau. Zunächst konvertiert ein Python-Skript den vorverarbeiteten Testdatensatz (mit Anomalien) in eine Log-Datei im CANoe[7] ASC-Format. Dies ermöglicht ein Abspielen der aufgezeichneten Signalwerte über CAN. Hierfür wurde die CAN-Botschaft mit der ID 1 definiert, welche das Geschwindigkeitssignal - beziehungsweise je einen Wert davon - in den Nutzdaten transportiert. Da CANoe auf einem normalen Bürorechner unter Windows ausgeführt wird und dieser in der Regel keine CAN-Schnittstelle besitzt, ist ein sogenanntes Netzwerk-Interface notwendig. Das eingesetzte VN5610 der Vector Informatik GmbH ist über Universal Serial Bus (USB) mit dem Bürorechner verbunden und bietet fahrzeugseitig zwei CAN und zwei Automotive Ethernet Schnittstellen. Über das VN5610 versendet CANoe die CAN-Botschaften mit der ID 1 auf dem angeschlossenen Bus. Das VC121 beziehungsweise die MICROSAR Basissoftware wurde entsprechend als Empfänger dieser CAN-ID und des Geschwindigkeitssignals konfiguriert. Damit steht die benötigte Empfangsschnittstelle (Sender-Receiver Interface) den SW-Cs der Applikationssoftware zur Verfügung. Der mit GENET erzeugte Code wurde in eine SW-C integriert und mit dem entsprechenden Receiver-Port verbunden (siehe dazu auch die Diskussion in Abschnitt 7.1.1). Des Weiteren ist dieser Port so konfiguriert, dass beim Empfang eines neuen Signalwerts das Runnable der SW-C getriggert wird, in welchem der Autoencoder implementiert ist. Die Bildung des Sliding Windows findet ebenfalls innerhalb dieses Runnables statt. Damit implementiert es die Merkmalsgenerierung, den lernenden Check (Autoencoder) und die Anomalieanalyse (siehe folgender Absatz) für die Plausibilisierung des Geschwindigkeitssignals (vgl. Bild 7.9). Die zeitlichen Abstände zwischen den einzelnen Signalwerten beziehungsweise CAN-Botschaften spielen keine Rolle, solange Trainings- und Testdaten auf der gleichen Abtastrate basieren. Ein Signal, das mit einem Hertz abgetastet ist (vgl. Unterkapitel 5.2), kann somit auch im zehn oder 100 ms Raster

[7] CANoe ist das Simulationswerkzeug für automotive Netzwerke der Vector Informatik GmbH.

wiedergegeben und plausibilisiert werden.
Zusätzlich wurde eine CAN-Botschaft mit der ID 2 definiert. Nach jedem Empfang und Prozessieren eines Signalwerts, überträgt damit das VC121 den Reproduktionsfehler sowie die binäre Anomalieklassifizierung zurück an CANoe. Dazu wurde die Basissoftware wiederum entsprechend konfiguriert und die Autoencoder SW-C mit den passenden Ports verbunden. Dies dient ausschließlich der einfacheren Auswertung und wäre in einem realen System meistens überflüssig. Stattdessen würde die SW-C bei einer erkannten Anomalie einen Fehlerspeichereintrag über den DEM setzen. Eine daran anschließende Diskussion folgt in Unterkapitel 7.3.
Die Laufzeitmessung an sich übernimmt das in MICROSAR integrierte Runtime Measurement (RTM). Bei entsprechender Konfiguration ermittelt dieses Basissoftware-Modul die reine CPU-Zeit zwischen zwei Messpunkten. Das Starten und Stoppen der Messung erfolgt über CANoe. Dafür - und für die Übertragung der Messergebnisse - wird ein separater Automotive Ethernet Kanal zwischen VC121 und CANoe/VN5610 verwendet. Im Vergleich zu den Messungen in Abschnitt 7.2.3, inkludieren die nachfolgenden Werte das Bilden der Zeitreihe, die Berechnung des Reproduktionsfehlers sowie die abschließende Anomaliebewertung mittels Schwellwert.

Fixkommaperformanz

Da die FPU der VC121 deaktiviert ist, steht die Fixkommaperformanz im Vordergrund. Außerdem werden, analog zu Unterkapitel 6.2, ausschließlich Autoencoder mit Rectifier Aktivierungsfunktion analysiert. Bild 7.10 und Tabelle 7.3 fassen die Messergebnisse zusammen. Dargestellt ist die durchschnittliche Ausführungszeit der kompletten Bewertung, welche beim Empfang eines neuen Signalwerts innerhalb des entsprechenden Runnables stattfindet. Untersucht wurde die jeweils leistungsstärkste Autoencoder-Topologie mit einer (6-18-6) und mit drei versteckten Schichten (6-18-54-18-6) sowie das Netz mit dem höchsten Leistungsverhältnis (4-3-4). Zum

Vergleich der VC121 mit dem Raspberry Pi 3 wurden zusätzlich die in Abschnitt 7.2.3 verwendeten 8-7-8 und 8-64-8 Topologien betrachtet. Des Weiteren ist die Berechnungsdauer in nativer (32 Bit) und doppelter Wortgröße (64 Bit) gegenübergestellt.

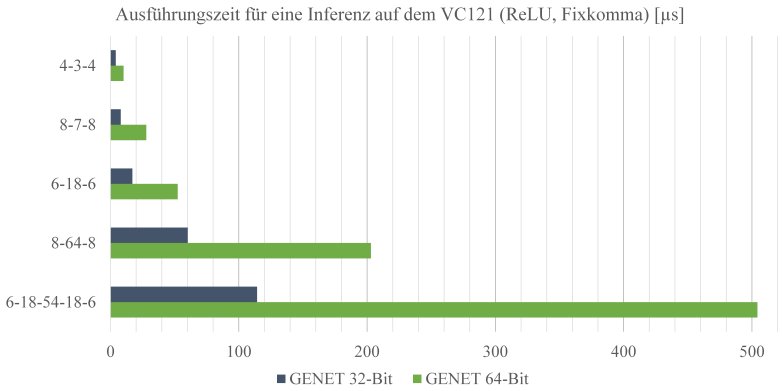

Bild 7.10: Ausführungszeiten für eine Inferenz auf dem VC121

Ausführungszeit [µs]	4-3-4	8-7-8	6-18-6	8-64-8	6-18-54-18-6
GENET 32-Bit	4,13	7,99	17,09	60,22	114,3
GENET 64-Bit	10,23	27,94	52,44	202,98	504,26

Tabelle 7.3: Ausführungszeiten für eine Inferenz auf dem VC121

Da der SPC56EC64 mit 120 MHz Taktfrequenz arbeitet, ist bei nativer Wortgröße eine mindestens zehnmal längere Ausführungszeit zu erwarten, im Vergleich zum Raspberry Pi 3. Für eine 8-7-8 Topologie mit ReLUs dauert eine Inferenz auf der VC121 ca. acht Mikrosekunden und ist damit um Faktor 22,8 langsamer. Bei der 8-64-8 Topologie beträgt der Faktor 18,7. Der überproportionale Anstieg ist einerseits durch die Zeitreihenbildung und die Anomaliebewertung zu erklären, welche in den VC121-Messwerten enthalten sind. Beide Schritte haben bei Topologien mit vergleichsweise we-

nig Neuronen/-verbindungen einen höheren Anteil an der Gesamtlaufzeit. Andererseits wirken sich Prozessorspezifika, wie beispielsweise die Cache-Architektur, auf die Gesamtlaufzeit aus.
Die nativen Wortgrößen unterscheiden sich zwischen VC121 und Raspberry Pi 3. Daher wurde eine zweite Messreihe mit doppelter Wortgröße (64 Bit) durchgeführt. Additionen mit doppelter Wortgröße dauern ungefähr doppelt so lange wie bei nativer Wortgröße, bei Multiplikationen erhöht sich die Laufzeit ungefähr um Faktor vier. Dies bestätigen auch die ermittelten Messergebnisse. Bei doppelter Wortgröße steigt die Berechnungsdauer je nach Topologie um Faktor 2,5 bis 4,4.

Fixkommagenauigkeit

Aufgrund der gezeigten Messergebnisse ist die Berechnung bei nativer Wortgröße zu favorisieren. Im nächsten Schritt ist zu klären, ob die damit erzielbare Genauigkeit ausreichend ist. Mit TensorFlow™ in Fließkommaarithmetik trainierte und evaluierte Autoencoder stellen dabei die Referenz dar. Nach Abschluss des Trainings und der Selektion anhand der Validierungsdaten steht die angereicherte Anomaly Detection Configuration zur Verfügung auf Basis derer GENET C-Code generiert. Zusätzlich werden für den Testdatensatz mit Anomalien der Reproduktionsfehler zu jedem Eingangsvektor sowie der zuvor ermittelte Schwellwert OF_S exportiert.
Es folgt die Inferenz über den Testdatensatz mit Anomalien mittels GENET C-Code, wobei wiederum der Reproduktionsfehler zu jedem Eingangsvektor abgespeichert wird. Aus Laufzeitgründen findet diese Evaluierung in einer separaten Desktopanwendung statt. Der generierte C-Code, welcher normalerweise Teil der AUTOSAR SW-C ist, bleibt dabei unverändert. Abschließend können die mit TensorFlow™ und GENET berechneten Reproduktionsfehler miteinander verglichen werden. Durch den bekannten Schwellwert ist außerdem eine Bewertung der GENET Implementierung anhand der TPR und FPR möglich.

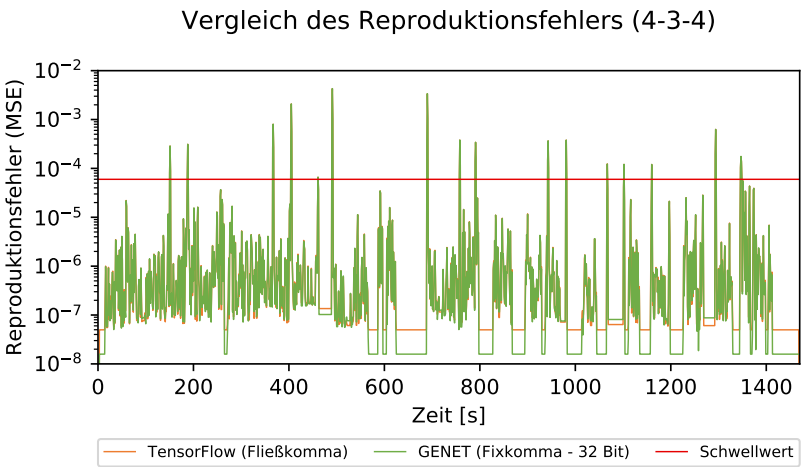

Bild 7.11: Vergleich des Reproduktionsfehlers von GENET und TensorFlow™ (4-3-4 Autoencoder)

Bild 7.11 zeigt einen Ausschnitt des Vergleichs der Reproduktionsfehler für den 4-3-4 Autoencoder mit dem höchsten Leistungsverhältnis. In diesem Fall arbeitet der GENET C-Code mit 32 Bit Fixkommaarithmetik. Eine Worst-Case-Betrachtung anhand der Autoencoder Parametrierung ergab einen notwendigen, ganzzahligen Anteil von drei Bit. Abzüglich des Vorzeichen-Bits (VZ) stehen damit noch 28 Bit für den gebrochenen Anteil zur Verfügung.

$$x = a \cdot b = a^{(F)} \cdot b^{(F)} \cdot 2^{-f} \tag{7.1}$$

Eine Multiplikation zweier Zahlen a und b in ihrer Fixkommadarstellung $a^{(F)}$ und $b^{(F)}$ erfolgt nach Formel 7.1. Die Variable f steht für die Bitanzahl des gebrochenen Anteils. Um bei den erforderlichen Multiplikationen einen Überlauf zu vermeiden, ist daher die Hälfte der 28 noch zur Verfügung stehenden Bits freizuhalten. Insgesamt ergibt sich daraus die Aufteilung der

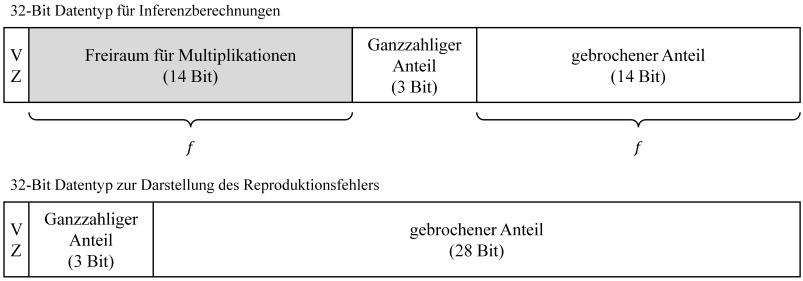

Bild 7.12: Datentyp zur sicheren Multiplikation bei 32 Bit Fixkommaarithmetik

32 Bit nach Bild 7.12[8] (oben). Der entsprechende Datentyp wird für die Inferenz-Berechnungen verwendet. Einzige Ausnahme ist der Reproduktionsfehler, welcher das Endergebnis darstellt. Hier ist der besprochene Freiraum nicht mehr notwenig und es steht die volle Genauigkeit zur Verfügung, im besprochenen Beispiel 28 Bit.

Der Datentyp mit 14 Bit Freiraum hat eine Zahlenauflösung von 2^{-14} ($\approx 6 \cdot 10^{-5}$). 32 Bit Fließkommatypen besitzen im relevanten Zahlenraum (siehe drei Bit ganzzahliger Anteil in Bild 7.12) mindestens eine Auflösung von 2^{-20} ($\approx 10^{-6}$). Durch diesen Unterschied ergeben sich bei der Fixkommaarithmetik Quantisierungsfehler. Diese sind in Bild 7.11 zu erkennen, besonders bei Zahlenwerten der Größenordnung 10^{-7}. Grün dargestellt ist der vom GENET C-Code berechnete Reproduktionsfehler, orange die Referenz aus TensorFlow™. Im aktuell untersuchten Fall sind die Quantisierungsfehler unproblematisch, da der Schwellwert zwischen 10^{-4} und 10^{-5} liegt. Es ergibt sich kein Unterschied zwischen der Anomaliebewertung (TPR und FPR) mittels GENET (32 Bit Fixkomma) und der TensorFlow™ Referenz (Fließkomma).

[8] Eine Alternative wäre das Verwenden des nächst größeren Datentyps (64 Bit) für die Zwischenergebnisspeicherung. Dies wäre wiederum mit einer erhöhten Laufzeit verbunden, weshalb die Reservierung von Freiraum bevorzugt wird.

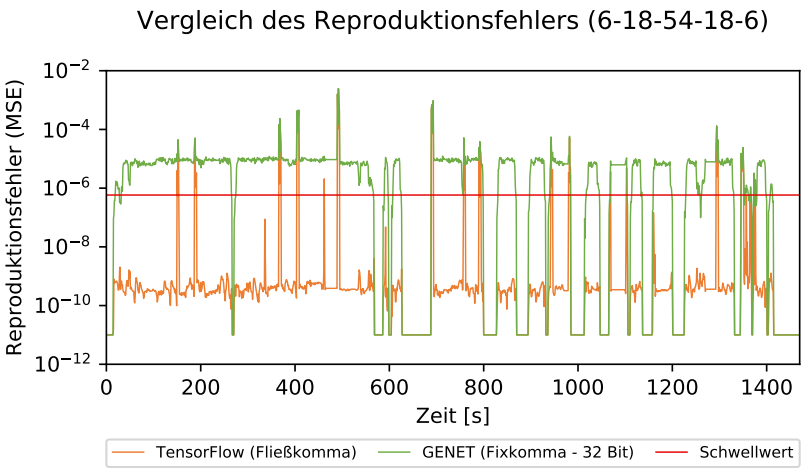

Bild 7.13: Vergleich zwischen GENET und TensorFlow™ (6-18-54-18-6 Autoencoder)

Problematisch ist der Quantisierungsfehler bei größeren Netztopologien, wie Bild 7.13 für den leistungsstärksten Autoencoder (6-18-54-18-6) zeigt. Der Reproduktionsfehler des GENET C-Codes ist im gleichen Testdatenausschnitt fast ausschließlich größer als der Schwellwert, welcher für größere Netztopologien tendenziell einen kleineren Wert annimmt und im vorliegenden Fall zwischen 10^{-6} und 10^{-7} liegt. Eine performante Anomalieerkennung ist damit nicht mehr möglich.

Eine Lösung des Problems ist die Verwendung des 64 Bit Fixkommatyps, welcher mit Vorzeichenbit, 28 Bit Freiraum, sieben Bit ganzzahligem und 28 Bit gebrochenem Anteil arbeitet. Die entsprechende Zahlenauflösung von 2^{-28} ($\approx 4 \cdot 10^{-9}$) resultiert in deckungsgleichen Reproduktionsfehlerkurven von GENET und TensorFlow™. Die zweite Möglichkeit besteht in der Verwendung von Fließkommadatentypen. Auch in diesem Fall produziert der von GENET generierte C-Code die gleichen Ergebnisse wie TensorFlow™.

Fließkommaperformanz

Wie zuvor diskutiert, erfordern größere Netze eine erhöhte Genauigkeit bei der Berechnung und damit entweder 64 Bit Fixkomma- oder Fließkommadatentypen. Fließkomma-Berechnungen ohne FPU benötigen allerdings deutlich mehr Rechenzeit. Eine Inferenz des untersuchten 4-3-4 Autoencoders mit ReLUs (inklusive Zeitreihenbildung und Anomaliebewertung) benötigt auf der VC121 durchschnittlich 111 µs, bei der 6-18-54-18-6 Topologie sind es 5745 µs. Damit ist die Berechnung der Netze im Vergleich zu 64 Bit Fixkommaarithmetik ungefähr um Faktor elf langsamer. Für Prozessoren ohne FPU sollten daher ausschließlich Fixkommadatentypen zum Einsatz kommen.

Zusätzliche Belastung des Prozessors

Zum Abschluss der Laufzeitevaluierungen auf Basis des VC121 wird nachfolgend auf die zusätzliche Prozessorlast eingegangen, welche die Signalplausibilisierung mittels Autoencoder in einem fiktiven Anwendungsfall erzeugt. Es gilt weiterhin die Annahme, dass die Inferenz auf einem Prozessorkern stattfindet.
Für den Fall, dass alle 10 ms ein aktualisierter Signalwert vorliegt[9], sind 100 Inferenzen pro Sekunde zu berechnen. Bild 7.14 und Tabelle 7.4 stellen die zusätzliche Prozessorlast pro plausibilisiertes Signal dar und fassen damit gleichzeitig die diskutierten Ausführungszeiten zusammen.

Stehen 10% Laufzeit auf einem Prozessorkern zur Verfügung, lassen sich damit unter Verwendung des 4-3-4 Autoencoders (32 Bit Fixkommaarithmetik) 250 Signale plausibilisieren[10]. Dabei wird davon ausgegangen, dass die Signale bereits vom Steuergerät empfangen und verarbeitet werden. An-

[9] 10 ms ist eine weit verbreitete Zykluszeit für periodische Botschaften in sicherheitskritischen Fahrzeugnetzwerken.

[10] Bei einem Raspberry Pi 3 sind es mit einem 8-7-8 Netz (64 Bit Fixkommaarithmetik) bereits ca. 2850. Unter der Annahme, dass die Laufzeiten ähnlich skalieren wie bei der VC121, lassen sich unter Verwendung einer 4-3-4 Topologie 7800 Signale plausibilisieren.

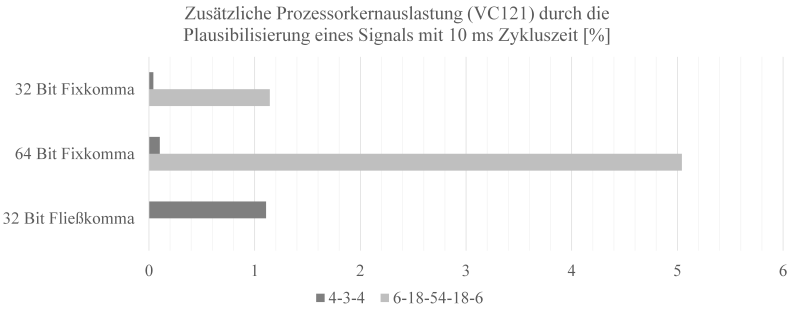

Bild 7.14: Prozessorlast durch die Signalplausibilisierung mittels Autoencoder auf dem VC121

zusätzliche Prozessorlast [%]	4-3-4	6-18-54-18-6
32 Bit Fixkomma	0,04	1,14
64 Bit Fixkomma	0,10	5,04
32 Bit Fließkomma	1,11	57,45

Tabelle 7.4: Prozessorlast durch die Signalplausibilisierung mittels Autoencoder auf dem VC121

sonsten verursacht die Prozessierung in der Basissoftware weiteren Mehraufwand. Speziell bei leistungsschwächeren Plattformen wie der VC121 ist eine Plausibilisierung aller Signale nicht möglich. Stattdessen sind die wichtigsten zu fokussieren, beispielsweise physikalische Sensor- und Aktorwerte.

Da die Auswahl von Steuergeräte-Prozessoren kostengetrieben ist, sind diese in der Regel stark ausgelastet. Umso wichtiger ist eine effiziente Berechnung der Signalplausibilisierung. Daher sollten nach Möglichkeit kleine Autoencoder-Topologien zum Einsatz kommen. Diese benötigen nicht nur wengier mathematische Operationen, sondern erlauben auch die Verwendung von 32 Bit Fixkommaarithmetik und bieten so einen doppelten Laufzeitvorteil.

7.2.5 Speicherverbrauch auf einem VC121

Neben der Ausführungszeit ist ein geringer Speicherverbrauch Voraussetzung für die Anwendbarkeit der Autoencoder auf realen Steuergeräten. Daher wurde sowohl für das leistungsstärkste Netz als auch für das Netz mit dem höchsten Leistungsverhältnis der Verbrauch von RAM und ROM ermittelt. Die Ergebnisse sind in Tabelle 7.5 dargestellt. Es ist jedoch zu beachten, dass die Werte stark vom eingesetzten Kompiler und den verwendeten Kompiler-Optionen abhängig sind. Im Rahmen der vorliegenden Arbeit kam die GNU Werkzeugkette[11] von SysProgs zum Einsatz, welche für die eingesetzte Power Architektur® angepasst ist [Sys]. Zusätzlich wurde unter anderem die Kompiler-Option *-Ofast* zur Laufzeitoptimierung verwendet.

Speicherverbrauch [Byte]	RAM	ROM
4-3-4 (Fixkomma, 32 Bit)	20	1112
4-3-4 (Fixkomma, 64 Bit)	40	2656
4-3-4 (Fließkomma, 32 Bit)	20	1896
6-18-54-18-6 (Fixkomma, 32 Bit)	28	14024
6-18-54-18-6 (Fixkomma, 64 Bit)	56	36436
6-18-54-18-6 (Fließkomma, 32 Bit)	28	17456

Tabelle 7.5: Speicherbedarf einer Autoencoder-Instanz auf dem VC121

Der von GENET generierte C-Code ist auf kurze Ausführungszeiten ausgelegt. Des Weiteren wird ein geringer RAM-Verbrauch angestrebt, zu Lasten des benötigten ROM. Dies spiegelt sich auch in den Ergebnissen wieder, siehe Tabelle 7.5. Der Hauptprozessor des SPC56EC64 besitzt 1,5 MB integriertes ROM (Code Flash) und 192 kB RAM. Aus diesem Gesichtspunkt ist der RAM-Verbrauch der implementierten Autoencoder ausreichend gering. Bei der Realisierung von 250 Instanzen mit einer 4-3-4 Topologie (siehe vorheriger Abschnitt), werden 5 kB RAM und damit 2,6% des verfügbaren Speichers benötigt. Im Vergleich dazu ist der ROM-Verbrauch mit 278 kB

[11] GNU Werkzeugkette basierend auf der GNU Compiler Collection (GCC) Version 4.9.0.

beziehungsweise 18,5% höher. Dieser steigt für 6-18-54-18-6 Autoencoder nochmals deutlich an, zumal aufgrund der Fixkomma-Genauigkeit die 64 Bit Variante notwendig ist. Hiervon benötigen bereits 41 Instanzen das gesamte verfügbare ROM. Ob die aktuelle Implementierung mit dem Fokus auf Laufzeit und RAM die optimale Lösung darstellt, ist für jeden Anwendungsfall und je nach verfügbaren Ressourcen individuell zu entscheiden. Eine Reduzierung des ROM-Verbrauchs ist möglich, geht allerdings zu Lasten der Laufzeit und/oder des benötigten RAMs.

7.2.6 Versuchsfahrzeug-Demonstrator

Nach der detaillierten Betrachtung von Laufzeit und Speicherverbrauch der mittels OBD-II Daten trainierten Autoencoder, wurde abschließend ein Demonstrator auf Basis eines realen Versuchsfahrzeugs aufgebaut. Die linke Seite von Bild 7.15 fasst die vorbereitenden Schritte zusammen. Auf der rechten Seite ist der resultierende Versuchsaufbau als Erweiterung beziehungsweise Anpassung von Bild 7.9 abgebildet. Die bisher verwendeten (vorverarbeiteten) OBD-II Daten besitzten eine Abtastrate von einer Sekunde. Im Versuchsfahrzeug wird die CAN-Botschaft, welche das Geschwindigkeitssignal enthält, allerdings mit einer Zykluszeit von 20 ms übertragen. Daher war zunächst ein neuer Datensatz zu erzeugen. Da das Versuchsfahrzeug keine Straßenzulassung besitzt, wurde auf die Simulationssoftware CarMaker der IPG Automotive GmbH [IPG] zurückgegriffen. Aufgrund der geänderten Abtastrate, sind auch die bisher trainierten und evaluierten Netze nicht anwendbar und es wird eine neue Autoencoder-Instanz benötigt. Bis einschließlich der *Synthese von Anomalien* ist das Vorgehen analog zu Bild 5.2. Die in Unterkapitel 5.5 besprochene Anomaliebewertung ändert sich nicht und ist daher in Bild 7.15 nicht separat aufgeführt. Deutlich vereinfacht ist hingegen die Netz-Auswahl. Für den Demonstrator gab es keinen Anspruch, den für diesen Fall leistungsstärksten Autoencoder oder jenen mit dem höchsten Leistungsverhältnis zu ermitteln. Auf Basis der Erkenntnisse

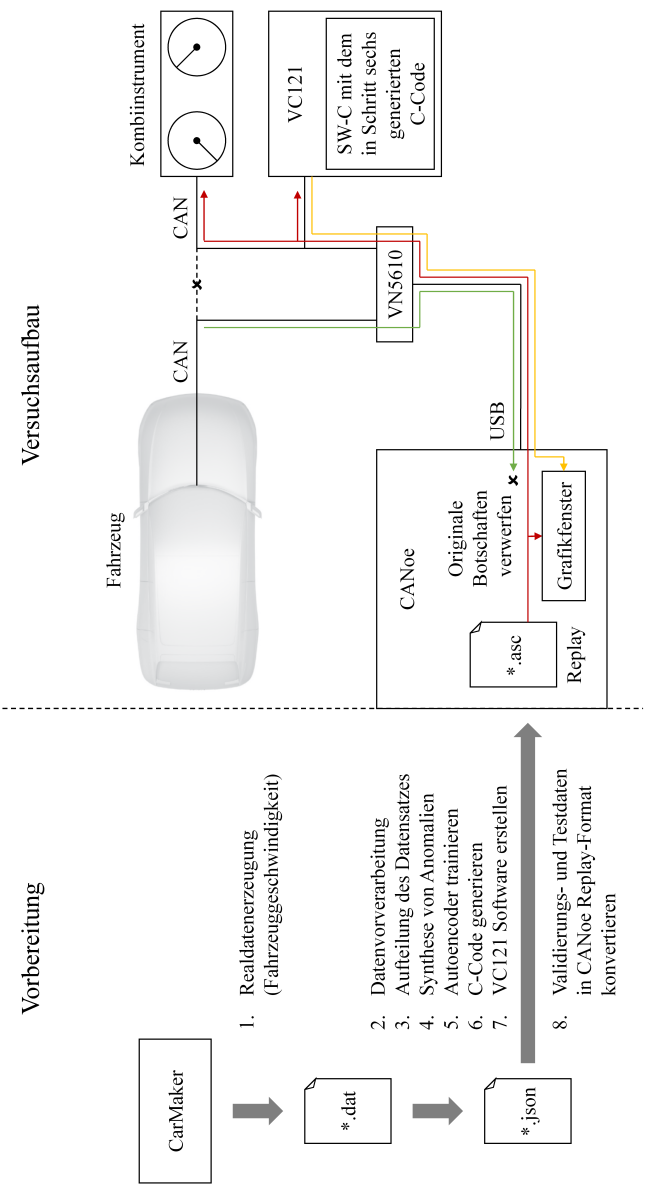

Bild 7.15: Vorbereitung und Aufbau des Versuchsfahrzeug-Demonstrators[12]

aus Unterkapitel 6.2 wurden stichprobenartig Netze mit drei bis acht Ein-/Ausgangsneuronen und mit einer versteckten Schicht trainiert. Der dabei leistungsfähigste Autoencoder besitzt eine 3-12-3 Topologie. Die Generierung des entsprechenden C-Codes erfolgte wiederum mittels GENET und in 64 Bit Fixkommaarithmetik[13].

Zu Demonstrationszwecken wurde im Versuchsfahrzeug die CAN-Verbindung zum Kombiinstrument aufgetrennt und beide Enden an ein VN5610 Netzwerk-Interface angeschlossen. Dieses ist über USB mit einem Notebook verbunden, auf welchem die Netzwerk-Analyse und -Simulationssoftware CANoe ausgeführt wird. In CANoe werden die beiden CAN-Verbindungen als separate Busse betrachtet: Ein Bus exklusiv für das Kombiinstrument und ein zweiter für das verbleibende Fahrzeugnetzwerk. Um den regulären Betrieb zu gewährleisten, leitet ein virtuelles Gateway die empfangenen Botschaften auf den jeweils anderen Bus weiter. Ausgenommen davon sind die Botschaften, welche Fahrzeuggeschwindigkeit und Motordrehzahl transportieren. Diese verwirft das virtuelle Gateway, siehe grüner Pfeil in Bild 7.15. Stattdessen gibt CANoe, unter Verwendung der passenden CAN-IDs, die mittels CarMaker aufgezeichneten Daten wieder. Dargestellt ist das *Replay* durch rote Pfeile. Während der Vorbereitung wurden hierfür die mit Anomalien angereicherten Validierungs- und Testdaten in das entsprechende CANoe-Format konvertiert. Sowohl das Kombiinstrument als auch die Steuergeräte im verbleibenden Fahrzeugnetzwerk empfangen somit alle Botschaften im richtigen Zyklus, wobei das virtuelle Gateway als Man-in-the-Middle agiert und dabei Fahrzeuggeschwindigkeit sowie Motordrehzahl manipuliert.

Das VC121 mit entsprechend konfigurierter Basissoftware ist am gleichen CAN-Bus angeschlossen wie das Kombiinstrument, siehe rechte Seite von Bild 7.15. Es implementiert den 3-12-3 Autoencoder zur Plausibilisierung des Geschwindigkeitssignals innerhalb einer SW-C. Zusätzlich versendet es

[13] Die Verwendung von 32 Bit Fixkommaarithmetik führte zu dem in Abschnitt 7.2.4 diskutierten Verhalten.

den berechneten Reproduktionsfehler sowie die Anomaliebewertung in separaten CAN-Botschaften. Diese werden wiederum von CANoe empfangen und zur Ergebnisdarstellung verwendet.

Bild 7.16: Signalplausibilisierung im Versuchsfahrzeug

Bild 7.16 zeigt ein Foto der Signalplausibilisierung im Versuchsfahrzeug. Auf dem Bildschirm des Notebooks ist das Grafikfenster von CANoe zu sehen. Die orange/untere Kurve repräsentiert die eingespeiste Fahrzeuggeschwindigkeit, dunkelgrün ist die dazugehörige Motordrehzahl dargestellt. Beide Werte werden im Kombiinstrument entsprechend angezeigt, auch wenn sich das Fahrzeug nicht bewegt. Ungefähr in der Mitte des Bildschirms weist die Fahrzeuggeschwindigkeit eine *Negative Ramp Offset* Anomalie auf, welche korrekt detektiert wurde - zu erkennen an den vertikalen Linien, welche jeweils den Übergang zwischen den Bewertungsklassen

normal (unterer Fensterrand) und *anormal* (oberer Fensterrand) markieren. Der Aufbau des Demonstrators zeigt, dass eine Signalplausibilisierung mittels Autoencoder auch unter realen Randbedingungen (zum Beispiel bei einer Zykluszeit von 20 ms) und mit vergleichsweise leistungsschwacher Steuergerätehardware in Echtzeit möglich ist. Außerdem führte das Vorgehen nach Bild 5.2 erneut zu einem erfolgreichen Ergebnis, auch wenn in diesem Fall auf eine detaillierte Untersuchung unterschiedlicher Netz-Topologien und die anschließende Leistungsevaluierung verzichtet wurde.

7.3 Der Observer in einer E/E-Architektur

Nach der Diskussion zur Umsetzung der Signalplausibilisierung auf realen Steuergeräten betrachtet dieses Unterkapitel die Instanziierung des Observers in einer E/E-Architektur. Ziel ist eine optimale Abdeckung der Überwachung bei möglichst wenig zusätzlichem Aufwand und Ressourcenverbrauch.
Eine Option ist die Einbringung eines zusätzlichen und dedizierten Steuergeräts, welches ausschließlich Überwachungsaufgaben übernimmt. Dies hätte den Vorteil, dass die entdeckten Anomalien an einer zentralen Stelle gespeichert sind und vereinfacht so das Zusammenspiel mit externen Gegenstellen, zum Beispiel mit einem Backend. Ein weiterer Vorteil ist, dass sich dieser Ansatz transparent für andere Steuergeräte umsetzen lässt, sofern lediglich die Fahrzeug-interne Kommunikation zu überwachen ist. Nachteilig sind die zusätzlichen Entwicklungs- und Produktionskosten für ein separates Steuergerät sowie die folgenden technischen Punkte:

1. Ein Observer-Steuergerät müsste an alle zu überwachenden Bussysteme und Netzwerke angeschlossen sein. Dies kann, je nach E/E-Architektur und Fokus der Überwachung, erheblichen Aufwand bei der Verkabelung bedeuten.

2. Das Observer-Steuergerät müsste entsprechend leistungsstark sein, um jegliche Fahrzeug-interne Kommunikation (Nachrichten und Signale) zu überwachen. Einerseits übertragen moderne Fahrzeuge eine vier- bis fünfstellige Anzahl an Signalen in ihren internen Netzwerken. Allein der Empfang aller Nachrichten und die Extraktion aller Signale ist rechenaufwändig und speicherintensiv. Da die Signale nicht bereits anderweitig von der Applikationssoftware verwendet werden, existieren keine Synergieeffekte für die Prozessierung in der Basissoftware. Hinzu kommt die Signalplausibilisierung - unabhängig von den implementierten Algorithmen zur Anomalieerkennung. Andererseits sind die Sensoren N-1 bis N-6 und N-8 ebenfalls umzusetzen, siehe Abschnitt 3.2.2. Die dafür eingesetzten Mechanismen benötigen weitere Laufzeit und weiteren Speicherplatz.

3. Ein separates Steuergerät ist zunächst nicht in der Lage Host-basierte Checks durchzuführen. Damit entfällt die Überwachung von Abarbeitungsreihenfolgen, Laufzeit, Speicherverbrauch und Steuergeräteinterner Kommunikation. Es sei denn, andere Steuergeräte stellen zusätzliche Informationen über deren interne Vorgänge zur Verfügung. Der Austausch dieser Informationen über die existierenden Fahrzeuginternen Netzwerke wird jedoch als unrealistisch angesehen beziehungsweise ist nur in sehr eingeschränktem Umfang möglich. Die meisten vorhandenen Netzwerke, im Speziellen die CAN-Busse, sind bereits gut ausgelastet und nach aktueller Einschätzung müsste jedes zu überwachende Steuergerät eine große Menge an zusätzlichen Informationen übertragen. Dies würde sowohl die Netzwerke als auch die Steuergeräte unverhältnismäßig belasten. Lediglich für einzelne oder spezielle Überwachungen ist eine solche zentralisierte Strategie denkbar.

Alternativ zu einem separaten Steuergerät könnte der Observer auf einem vorhandenen zentral instanziiert werden. Hierfür würde sich beispielsweise

ein zentrales Gateway anbieten, sofern es in der E/E-Architektur existiert. Dieser Ansatz würde die zusätzlichen Kosten verringern, hätte aber ebenfalls die zuvor beschriebenen, technischen Nachteile. Zusätzlich geht momentan der Trend zu Domänen- und Zentralrechner-Architekturen, wie in Abschnitt 2.1.3 beschrieben. In diesen E/E-Architekturen wurde das zentrale Gateway bewusst durch mehrere Domänencontroller ersetzt.

7.3.1 Verteilter Observer

Aufgrund der diskutierten Nachteile einer zentralisierten Lösung wird eine verteilte Instanziierung des Observers bevorzugt, welche im Folgenden näher beschrieben ist. Die dazu passende Übersicht gibt Bild 7.17 auf Basis der in Abschnitt 2.1.3 vorgestellten Domänen-Architektur.

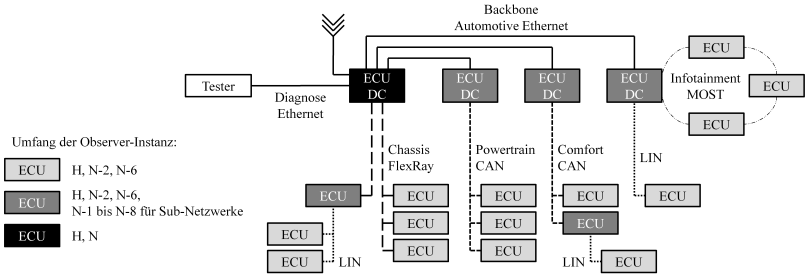

Bild 7.17: Verteilung des Observers in einer E/E-Architektur

Aus den zuvor genannten Gründen, sind die Host-basierten Überwachungen direkt auf dem jeweiligen Steuergerät auszuführen (gekennzeichnet durch ein *H* in der Legende von Bild 7.17). Die Umsetzung der Netzwerk-basierten Überwachungen ist differenzierter zu betrachten. Sollte ein Angreifer physischen Zugang zu einem Fahrzeug-internen Netzwerk besitzen, kann er als Man-in-the-Middle die Kommunikation zu einem isolierten Steuergerät manipulieren, siehe Abschnitt 7.2.6. Um diese Art von Angriff erkennen zu können, muss die Implementierung aller Netzwerk-basierten

Anomalieerkennungssensoren - mit Ausnahme von N-5 (siehe unten) - in entsprechendem Umfang auf allen Steuergeräten erfolgen. Dies wird aktuell als unrealistisch angesehen. Daher folgt ein Vorschlag zur Verteilung der Sensoren auf unterschiedliche Steuergeräte-Klassen, der jedoch akzeptiert, dass Angriffe mit physischem Fahrzeugzugang gegebenenfalls nicht erkannt werden können.

Unter Berücksichtigung der genannten Randbedingung genügt die einmalige Ausführung von N-1, N-3, N-4, N-7 und N-8 (siehe Abschnitt 3.2.2) auf einem Steuergerät, welches die gesamte Kommunikation in einem Netzwerk beobachten kann. Für klassische Bussysteme wie CAN ist die Voraussetzung implizit bei jedem angeschlossenen Teilnehmer erfüllt, wobei die Instanziierung auf einem leistungsstarken Steuergerät favorisiert wird - beispielsweise auf einem Domänencontroller. Im Gegensatz dazu leiten Ethernet-Switches Nachrichten nur an die adressierten Empfänger weiter. Mit modernen Automotive Ethernet-Switches ist die Überwachung des gesamten Datenaustauschs dennoch an einer zentralen Stelle möglich. Inzwischen haben einige Bausteine einen frei programmierbaren Mikroprozessor integriert, der vollen Zugriff auf alle Switch-Ports und damit alle ausgetauschten Nachrichten hat. Damit ist eine Instanziierung der entsprechenden Observer-Anteile direkt auf dem Switch denkbar und die bevorzugte Lösung. Eine zweite Möglichkeit ist das Verwenden eines sogenannten Uplink-Ports. Bei entsprechender Konfiguration leitet dieser Port alle Datenpakete an den daran angeschlossenen Host-Controller weiter, welcher die Überwachungsaufgaben übernimmt.

Sensor N-5 ist per Definition an Steuergeräte gebunden, welche Zugriff auf mehrere Netzwerke haben und damit die Korrelation von weitergeleiteten Nachrichten überwachen können.

Einen Mehrwert bietet die Instanziierung von N-2 auf allen Kommunikationsteilnehmern. Während generell unzulässige Nachrichten ebenfalls zentral entdeckt werden können, ist dies für prinzipiell zulässige nicht immer möglich. Bei CAN ist beispielsweise der Absender einer Botschaft unbe-

kannt. Da allerdings jedes Steuergerät weiß, welche Botschaften es empfängt und versendet, kann der legitime Sender einer CAN-ID einen potentiellen Angriff erkennen, wenn er „seine“ CAN-ID unerwarteterweise empfängt.
Die Überwachung von Hersteller-spezifischen Protokollen mittels N-6 ist unter Umständen nur auf den daran teilnehmenden Steuergeräten möglich. In diesem Fall ist der entsprechende Sensor auf den jeweiligen Endknoten (hellgraue Steuergeräte in Bild 7.17) zu instanziieren.
Aus obiger Diskussion ergibt sich folgender Vorschlag für die Verteilung der Sensoren zur Netzwerk-basierten Überwachung: N-2 ist auf allen Steuergeräten zu instanziieren. Gleiches gilt für N-6, sofern Hersteller-spezifische Protokolle vorhanden sind und sich diese nicht zentralisiert prüfen lassen. Gateways und Domänencontroller haben die Aufgabe ihre Sub-Netzwerke ganzheitlich zu überwachen, was die Implementierung von N-1 bis N-8 bedeutet. Für das hierarchisch höhere Netzwerk ist wiederum die Implementierung von N-2 und gegebenenfalls N-6 ausreichend. Diese Klasse von Steuergeräten ist in Bild 7.17 in einem mittleren Grauton dargestellt. Des Weiteren existiert eine zentrale Instanz des Observers, welche neben dem Hostanteil auch alle angeschlossenen Netzwerke ganzheitlich überwacht - der globale Master-Observer. In Bild 7.17 ist es der dunkelste Domänencontroller, welcher den Ethernet-Switch des Backbones enthält und im Beispiel außerdem die Fahrzeug-externe Kommunikation realisiert.

Kommunikation zwischen den Observer-Instanzen

Jede Observer-Instanz kann erkannte Anomalien lokal im Fehlerspeicher ablegen. Bei einer Implementierung für AUTOSAR Classic erfolgt dies über den DEM, siehe Abschnitt 7.1.1. Um einen Gesamtüberblick zu erhalten und für ein vereinfachtes Melden der erkannten Anomalien, beispielsweise an ein Backend-System, ist zusätzlich eine zentrale Ablage im Master-Observer hilfreich. Des Weiteren kann der Master-Observer eine Konso-

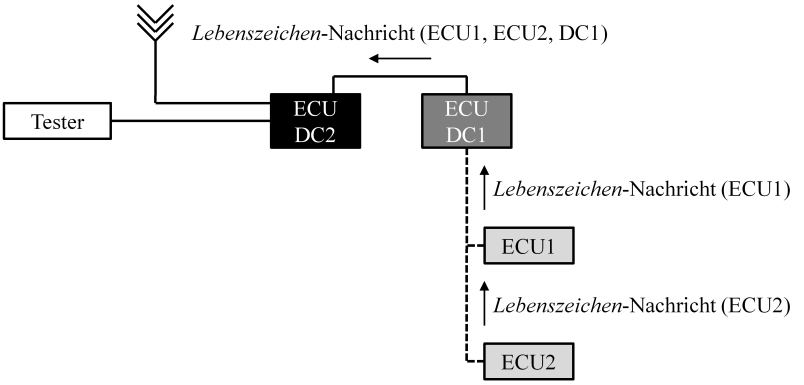

Bild 7.18: Verbauschutz durch verteilte Observer

lidierung von redundant erkannten Anomalien vornehmen und somit die Fahrzeugdiagnose im Hinblick auf das Auslesen des DEM beziehungsweise des SEM vereinfachen.

Für die Realisierung der zentralen Ablage müssen alle Observer-Instanzen Informationen über die erkannten Anomalien sicher zum Master übertragen. Hierfür bietet AUTOSAR mit SecOC bereits eine passende Lösung, siehe Abschnitt 3.2.1.
Unter der Annahme, dass auf jedem Steuergerät ein Observer instanziiert ist, lässt sich damit gleichzeitig ein Verbauschutz realisieren. Meldet sich jede Instanz in zyklischen Abständen beim nächst höheren Observer in der Hierarchie, erhält der globale Master einen Überblick bezüglich aller aktuell aktiven Steuergeräte. Bild 7.18 detailliert einen Ausschnitt aus Bild 7.17. *ECU1* und *ECU2* senden je eine *Lebenszeichen*-Nachricht an den dazugehörigen Netzwerk-Master, im Beispiel *ECU DC1*. Dieser leitet die empfangenen Informationen in seiner eigenen Lebenszeichen-Nachricht an den globalen Master (*ECU DC2*) weiter, welcher damit den Status aller im Fahrzeug verbauten Steuergeräte kennt. Sollte das Lebenszeichen eines Steuer-

geräts unerwartet ausbleiben[14], speichern die betroffenen Master-Observer eine Anomalie ab. Zusätzlich kann der Netzwerk-Master seine Überwachungsregeln für N-2 anpassen, um die erlaubten Nachrichten entsprechend einzuschränken. Ist ein Steuergerät nicht aktiv (Lebenszeichen-Nachricht bleibt aus) und es werden trotzdem dessen Applikations-Nachrichten empfangen, ist ebenfalls eine Anomalie zu speichern. Das beschriebene Szenario deutet auf eine Attacke hin, bei der ein Angreifer ein vorhandenes Steuergerät außer Kraft setzt (zum Beispiel mittels selektivem Denial-of-Service, siehe Bild 3.7) und versucht, dessen Verhalten zu imitieren, dabei aber nur ausgesuchte Nachrichten nachbildet - exklusive des geschützten Lebenszeichens.

7.3.2 Fahrzeug-externe Kommunikation

Für die Überwachung benötigt der Observer keine Konnektivität und keine Kommunikation mit Fahrzeug-externen Gegenstellen (siehe [Req4] in Unterkapitel 4.2). Zur weiteren und Fahrzeug-übergreifenden Analyse sind erkannte Anomalien jedoch an ein Backend-System zu übertragen. Dies sollte so schnell wie möglich erfolgen, um kurzfristige Gegenmaßnahmen[15] zu ermöglichen. Daher ist die reine Speicherung von Anomalien bis zum nächsten Werkstattaufenthalt nicht ausreichend. Stattdessen sind Anomalien direkt nach dem Erkennen per Luftschnittstelle zu melden.
Im Backend-System können zum einen Auswertungen stattfinden, für welche die Rechenleistung im Fahrzeug nicht ausreichend ist. Zum anderen stehen dort nicht nur Informationen bezogen auf ein Fahrzeug, sondern über die gesamte Fahrzeugflotte zur Verfügung. Dieses Wissen ist notwendig, um unter anderem ortsbezogene Angriffswellen, Modell-spezifische Schwach-

[14] Bei Teilnetzbetrieb ist das gezielte Abschalten von einzelnen Steuergeräten gewollt. Die benötigte Information liegt durch das Netzwerkmanagement im dazugehörigen Netzwerk-Master ebenfalls vor und kann damit berücksichtigt werden.

[15] Die angesprochenen, kurzfristigen Gegenmaßnahmen werden in diesem Fall durch das Backend-System beziehungsweise durch entsprechende Mitarbeiter eingeleitet und nicht durch den Observer selbst.

stellen, Modell-spezifisches Fehlverhalten oder sogar Bauteil-bezogene Probleme zu identifizieren. Basierend darauf lassen sich Entscheidungen über Gegenmaßnahmen treffen. Welche Daten zur Nachbereitung und zur weiteren Analyse ans Backend übertragen werden müssen, ist ein separates Forschungsfeld. Hier spielen Faktoren wie der zur Verfügung stehende Speicherplatz im Fahrzeug sowie die Bandbreite der Luftschnittstellen entscheidende Rollen.
Neben dem Melden der erkannten Anomalien ist die Fahrzeug-externe Kommunikation auch für Observer-Updates nutzbar. Die als SOTA oder Firmware Over-the-Air (FOTA) bekannte Technologie ermöglicht das Aktualisieren von Steuergeräte-Software unabhängig von einem Werkstattaufenthalt. Am Beispiel von Tesla [Tes] ist zu erkennen, dass SOTA nicht nur für Fehlerkorrekturen, sondern auch für funktionale Erweiterungen beziehungsweise Aktualisierungen verwendet wird. Inzwischen arbeiten alle namhaften Fahrzeughersteller und Zulieferer an entsprechenden Lösungen. Das Aktualisieren des Observers selbst kann aus unterschiedlichen Gründen notwendig sein:

1. Durch funktionale Software-Updates können sich Steuergeräte-Interna wie Abarbeitungsreihenfolge, Laufzeit, Speicherverbrauch sowie der Kontroll- und Datenfluss signifikant verändern. Überwacht ein Observer mindestens eines davon, ist auch dieser zu aktualisieren.

2. Betrifft ein funktionales Software-Update neben Steuergeräte-Interna zusätzlich die Fahrzeug-interne Kommunikation - da beispielsweise neue Nachrichten versendet werden - sind die betroffenen Observer-Instanzen zu aktualisieren, je nach Verteilung und Implementierung der Überwachungssensoren.

3. Werden von den lernenden Checks zur Laufzeit zu viele Falschalarme an das Backend gemeldet, ist dies durch ein Update zu beheben. Die Entwicklung und das Training der Algorithmen kann kontinuier-

lich erfolgen, auch während eine erste Version bereits in Steuergeräten verbaut ist. Ist ein neuer, verbesserter Parametersatz gefunden - oder sogar ein neuer Algorithmus - lässt sich dieser mittels SOTA auf die im Einsatz befindlichen Fahrzeuge und Steuergeräte übertragen.

4. Ist zukünftig eine Firewall Teil des Observers (siehe Unterkapitel 9.7), stellt das Update des Regelsatzes eine mögliche Gegenmaßnahme bei erkannten Angriffen dar.

8 Zusammenfassung

Die zunehmende Vernetzung von Fahrzeugen mit Unterhaltungselektronik, anderen Fahrzeugen und externer Infrastruktur über kabelgebundene und vor allem kabellose Schnittstellen, ermöglicht neue Angriffsvektoren. Zusätzlich können Steuergeräte das Fahrverhalten vermehrt aktiv beeinflussen, wodurch bei einem entsprechenden, erfolgreichen Angriff die Sicherheit der Fahrzeuginsassen nicht mehr gewährleistet ist. Daher sind geschichtete Gegenmaßnahmen innerhalb einer E/E-Architektur anzuwenden. Ein Teil der sogenannten Defense-in-Depth Strategie ist ein Intrusion Detection/Prevention System, welches potentielle Angriffe auf einem Steuergerät in Echtzeit erkennt und gegebenenfalls unterbindet. In diesem Kontext definiert die vorliegende Arbeit ein Konzept zur Überwachung von Steuergeräte-Funktionalität, der *Automotive Observer*. Anstatt spezifischer Angriffe erkennt der Observer Anomalien, d.h. Abweichungen vom Normalverhalten. Er ist damit nicht eingeschränkt auf den Anwendungsfall der Angriffserkennung, sondern ermöglicht auch die Detektion von funktionalen Fehlern.

Die Architektur des Observers sowie das prinzipielle Vorgehen bei dessen Realisierung ist in Bild 8.1 dargestellt. Basierend auf vorhandenen, semiformalen Entwicklungsartefakten (zum Beispiel die Kommunikationsmatrix beziehungsweise die AUTOSAR System Description) entsteht der Quellcode zur Überwachung des Steuergeräts. Dieser ist zweigeteilt: Statische Checks kommen für fest vorgegebene Eigenschaften zum Einsatz. Lernende Checks implementieren hingegen Algorithmen des maschinellen Lernens, um Eigenschaften zu überwachen, welche während der Entwicklungszeit nicht explizit vorgegeben werden (können). Beide Verfahren werden sequentiell angewendet und speichern erkannte Anomalien zentral ab.

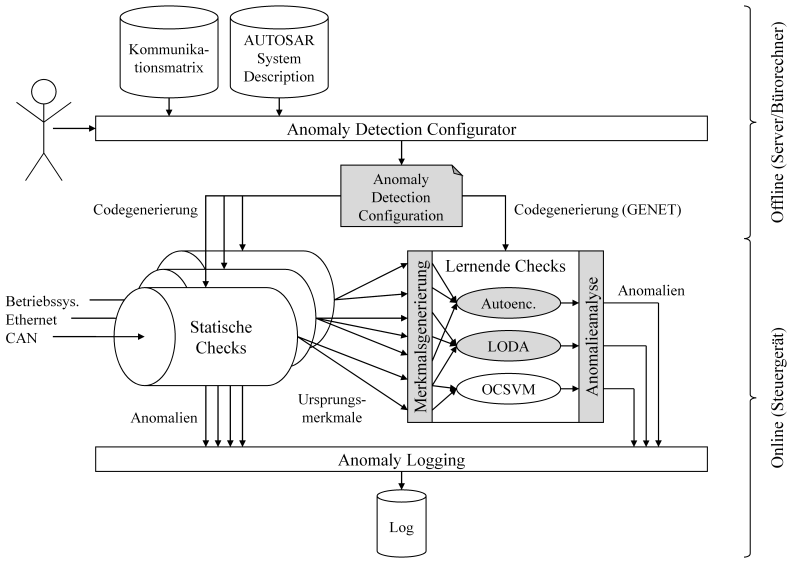

Bild 8.1: Automotive Observer (fokussierte Bereiche sind grau hinterlegt)

Das Observer-Konzept umfasst Netzwerk- und Host-basierte Aspekte, wobei auf erstere näher eingegangen wird. Aus den dafür vorhandenen Überwachungsmöglichkeiten fokussiert der zweite Teil dieser Arbeit die Plausibilisierung von Kommunikationssignalen. Dabei ist der zeitliche Verlauf eines Signals zu prüfen, welcher nicht (semi-)formal festgelegt ist. Für den notwendigen lernenden Check wird ein Ansatz mittels Autoencoder - eine Form von künstlichen neuronalen Netzen - gewählt. Die Algorithmen-Auswahl erfolgte anhand definierter Kriterien und vor dem Hintergrund der beschränkten Ressourcen auf embedded Hardware. Eine Bestimmung aller benötigten Hyperparameter ist vorab jedoch nicht möglich. Daher wurden unterschiedliche Netz-Topologien empirisch untersucht, nach einem zuvor festgelegten Vorgehen. Grundlage hierfür ist das Geschwindigkeitssignal von eigens aufgezeichneten OBD-II Daten. Diese Daten enthalten keine

realen Anomalien, weshalb zunächst 13 parametrierbare Anomalie-Typen definiert und anschließend je 20 Instanzen in den Validierungs- und in den Testdaten synthetisiert wurden. Unter Verwendung der mit Anomalien angereicherten Testdaten erreicht der leistungsstärkste Autoencoder eine TPR von 85,8% bei gleichzeitig keinem Falschalarm. Im Hinblick auf die Ressourcenbeschränkung wurde zusätzlich der Autoencoder mit dem höchsten Leistungsverhältnis ermittelt, eine eigens festgelegte Metrik, welche die Anzahl der notwendigen Multiplikationen und die damit verbundene Laufzeit berücksichtigt. Das so bestimmte Netz besitzt eine TPR von 78,5% bei ebenfalls keinem Falschalarm.

Die gewählte Parametrierung der Anomalie-Typen resultiert in Anomalie-Instanzen, die sich teilweise nicht oder nur marginal vom Normalverhalten unterscheiden. Unter Berücksichtigung aller Anomalie-Typen, ergibt sich dadurch eine erwartete TPR von ca. 80%. Die erreichte TPR liegt für den leistungsstärksten Autoencoder somit über den Erwartungen, für die Topologie mit dem höchsten Leistungsverhältnis etwas darunter. Positiv bewertet wird die erreichte FPR von 0%. Es ist allerdings zu beachten, dass die Definition des Normalverhaltens implizit über die verwendeten Trainingsdaten erfolgt. Die sich dabei ergebenden Herausforderungen sind anwendungsspezifisch zu adressieren. Für den praktischen Einsatz ist eine FPR nahe null notwendig, da ansonsten zusätzliche Aufwände auf Backend-Seite entstehen oder in Zukunft sogar unnötige Gegenmaßnahmen eingeleitet werden. Bei einem akzeptierten Falschalarm pro Monat und einer durchschnittlichen Nutzungsdauer eines Fahrzeugs von einer Stunde am Tag, liegt die benötigte FPR für die Plausibilisierung eines Signals mit 20 ms Zykluszeit bei ca. $1{,}9 \cdot 10^{-7}$.

Der Ansatz mittels Autoencoder (inklusive des angewendeten Vorgehens) liefert vielversprechende Ergebnisse, wobei sich die präsentierten Werte für TPR und FPR auf das Geschwindigkeitssignal der aufgezeichneten OBD-II Daten beziehen und sich nicht implizit auf andere Signale, Daten, Fahrzeuge oder Fahrer übertragen lassen. Für den praktischen Einsatz ist die angedeu-

tete Generalisierung erforderlich und daher in nachfolgenden Arbeiten zu untersuchen, siehe Unterkapitel 9.1.

Der letzte Teil der vorliegenden Arbeit untersucht die Realisierung des Observers und im Speziellen die Signalplausibilisierung mittels Autoencoder auf einem AUTOSAR Classic-basierten Steuergerät. Ein entwickelter Generator erzeugt den dafür notwendigen C-Code für die Inferenz auf Basis der Anomaly Detection Configuration (siehe Bild 8.1), welche nach erfolgreichem Training alle dafür notwendigen Parameter enthält. Auf der eingesetzten Steuergeräte-Hardware (120 MHz Power Architektur®) benötigt eine Inferenz des leistungsstärksten Autoencoders 504,26 µs, während jener mit dem höchsten Leistungsverhältnis eine Berechnungsdauer von 4,13 µs aufweist. Letzterer ist mit einem Bedarf von 20 Byte RAM und 1112 Byte ROM auch auf Steuergeräten mit vergleichsweise wenig Speicherressourcen realisierbar. Des Weiteren zeigt ein Versuchsfahrzeug-Demonstrator die Umsetzung der Signalplausibilisierung unter realen Randbedingungen sowie die Übertragbarkeit des angewendeten Vorgehens.
Der Quellcode für die Inferenz von Autoencodern ist auf eine kurze Laufzeit sowie geringen RAM-Verbrauch optimiert und ermöglicht damit auch deren Einsatz auf leistungsschwächeren Steuergeräten. Der Generator ist jedoch nicht auf Autoencoder beschränkt, sondern erzeugt C-Code für die Inferenz von beliebigen, vollständig-verbundenen, feed-forward Perceptrons. Damit eröffnen sich auch Einsatzmöglichkeiten abseits des Observers.

Die vorliegende Arbeit leistet einen Beitrag zur Steigerung der Angriffssicherheit moderner und zukünftiger Fahrzeuge. Das vorgestellte Observer-Konzept ist in der Lage, Mechanismen für die Überwachung verschiedener Steuergeräte-Funktionen und -Eigenschaften effizient zu vereinigen und so unterschiedliche Angriffe zu erkennen. Die fokussierte Signalplausibilisierung ist unabhängig vom Fahrzeugnetzwerk und auch bei Steuergeräteinterner Kommunikation anwendbar. Ziel ist die Erkennung von Anomalien

im zeitlichen Verlauf eines Kommunikationssignals, wie sie bei Masquerade-, Tampering-, Man-in-the-Middle-Attacken und bei der Manipulation von Daten im originalen Sender auftreten können. Schafft es ein Angreifer allerdings einen validen Signalverlauf zu erzeugen, beispielsweise unter Berücksichtigung physikalischer Randbedingungen, ist die Attacke mittels Signalplausibilisierung nicht erkennbar. Das Beispiel zeigt den kontinuierlichen Wettlauf zwischen Angreifer und Schutz- beziehungsweise Erkennungsmaßnahmen. Daher verringern die untersuchten Mechanismen die Wahrscheinlichkeit unentdeckter Attacken, schließen sie aber nicht aus.

Bisher wurde davon ausgegangen, dass ein kompromittiertes Netzwerk beziehungsweise Steuergerät Voraussetzung für die Manipulation von Kommunikationssignalen ist. Eine andere mögliche Ursache sind Angriffe auf die im Fahrzeug verbaute Umfeld-Sensorik, zum Beispiel das sogenannte *Sensor-Jamming*. Hierbei werden Sensoren durch äußere Einflüsse gezielt gestört oder übersteuert. Dies beinhaltet das Blenden von Kameras und Laser-Scannern mit starken Lichtquellen sowie vergleichbare Angriffe gegen Radar- und Ultraschall-Sensoren. Wird beispielsweise der Abstand zum vorausfahrenden Fahrzeug mittels Kamera oder Radar ermittelt und als Kommunikationssignal übertragen, erzeugt Sensor-Jamming voraussichtlich ebenfalls unnatürliche Werte-Verläufe. Die Signalplausibilisierung kann hier zur Absicherung beitragen, zumal das Erzeugen von manipulierten aber dennoch validen Signalverläufen in diesem Fall nochmals herausfordernder ist.

Da die Detektion von Anomalien unabhängig von deren Ursache erfolgt, bietet der Observer neben einer Angriffserkennung auch einen Mehrwert bei der funktionalen Absicherung von Steuergeräte-Funktionen und angeschlossenen Sensoren/Aktoren. Unerwartetes Systemverhalten kann unterschiedliche Ursachen haben, zum Beispiel das versehentliche Überschreiben von Speicherbereichen oder unerwartete zeitliche Abhängigkeiten (engl.: Race Conditions). Das frühzeitige Erkennen des entstehenden Fehlverhaltens erlaubt Rückschlüsse auf die Ursache und trägt in der Regel zur Lösung des

Problems bei.
Ein weiterer Anwendungsfall für den Observer ist die vorausschauende Diagnose (engl.: Predictive Maintenance). Verschiedene Bauteile und Sensoren altern und verschleißen im Laufe der Zeit. Deutet sich ein drohender Bauteilausfall in einem dazugehörigen Kommunikationssignal an, zum Beispiel aufgrund eines größeren Rauschens oder durch Spannungs- beziehungsweise Stromspitzen, eignet sich die Signalplausibilisierung zur frühzeitigen Erkennung. Im Optimalfall werden eine Fahrzeugpanne und die damit verbundenen Unannehmlichkeiten für den Fahrer vermieden.
Das Observer-Konzept adressiert automotive Steuergeräte. Die Architektur sowie die untersuchten Algorithmen zur Signalplausibilisierung sind jedoch auf andere Anwendungsgebiete übertragbar und können auch dort einen Beitrag zur Angriffssicherheit sowie zur funktionalen Absicherung leisten.

9 Zukünftige Forschungsfelder

Als Teil der Zusammenfassung zeigt Bild 8.1 die gesamte Architektur des Observers. Die von der vorliegenden Arbeit fokussierten Bestandteile sind grau hinterlegt. Schwerpunkt sind die lernenden Checks, im Speziellen die Signalplausibilisierung mittels Autoencoder. Sowohl in diesem Bereich als auch in Bezug auf die nicht näher betrachteten Observer-Bestandteile existieren Anknüpfungspunkte für zukünftige Arbeiten. Ausgenommen von der nachfolgenden Diskussion sind der Anomaly Detection Configurator und die Anomaly Detection Configuration. In beiden Fällen liegt der Schwerpunkt auf deren Realisierung. Konzeptionelle Herausforderungen, die von zukünftigen Forschungsarbeiten adressiert werden müssen, sind aktuell keine bekannt.

9.1 Generalisierung der Signalplausibilisierung mittels Autoencoder

Die in Unterkapitel 6.2 gezeigten und diskutierten Ergebnisse für die Signalplausibilisierung mittels Autoencoder sind vielversprechend. Außerdem ist eine Realisierung des Verfahrens auf ressourcenbeschränkter Steuergeräte-Hardware möglich, siehe Unterkapitel 7.2. Dennoch gibt es weiteren Forschungsbedarf.
Die Untersuchungen im Rahmen dieser Arbeit beziehen sich auf das Geschwindigkeitssignal der OBD-II Daten. Zukünftig ist zu prüfen, ob sich für andere Signale vergleichbare Ergebnisse erzielen lassen. Speziell die Betrachtung von weiteren, kontinuierlichen Signalen mit unterschiedlichen Charakteristika ist dabei ein Anknüpfungspunkt. Wie gut funktioniert die

Erkennung bei vergleichsweise dynamischen beziehungsweise statischen Signalen, zum Beispiel bei Motordrehzahl und Außentemperatur? Separat dazu sind Status- und Binärsignale zu betrachten.
Neben den zu plausibilisierenden Signalen ist auch der Einfluss der dazugehörigen Eingangsdaten zu analysieren. Welche Auswirkung hat zum Beispiel die Abtastrate und kann durch eine Unter- oder Überabtastung die Leistungsfähigkeit verbessert beziehungsweise der Ressourcenverbrauch gesenkt werden? Unter Verwendung von realen Sensor-Werten ist zudem zu untersuchen, ob eine weitere Vorverarbeitung - wie beispielsweise eine Tiefpassfilterung - notwendig ist.
Ein wichtiger und praxisrelevanter Punkt ist die Generalisierungsfähigkeit der trainierten Autoencoder. Im besten Fall ist nur eine trainierte Instanz je Signal und Fahrzeughersteller notwendig. Für manche Signale ist dies eine besondere Herausforderung. Betrachtet man zum Beispiel die Plausibilisierung der Fahrzeuggeschwindigkeit, unterscheiden sich valide Signalverläufe teilweise deutlich zwischen unterschiedlichen Modellen und Motorisierungsvarianten. Ein Kleinwagen mit 75 PS beschleunigt und verzögert anders als ein Sportwagen mit 500 PS. Daher kann es sein, dass ein Autoencoder, der mit Daten des 75 PS Fahrzeugs trainiert wurde und im 500 PS Fahrzeug eingesetzt wird, eine hohe Falschalarmrate aufweist. Daraus ergeben sich Fragen, die in Zukunft zu adressieren sind: Kann ein Autoencoder so trainiert werden, dass er über verschiedene Fahrzeugkonfigurationen, Fahrzeuge oder Baureihen generalisiert? Falls ja, welche Auswirkungen hat dies auf die Erkennungsrate und lässt sich gegebenenfalls ein guter Kompromiss finden? Fahrzeug-spezifische Instanzen erscheinen momentan unrealistisch. Auf der einen Seite gibt es viele Konfigurationen, die teilweise nicht explizit getestet werden, und es ist nicht möglich alle Fahrzeug-spezifischen Datensätze während der Entwicklungszeit zu erzeugen. Andererseits ist ein Training der Algorithmen im Fahrzeug aufgrund der Ressourcenbeschränkung aktuell ebenfalls nicht darstellbar. Dies ändert sich möglicherweise in Zukunft, wobei dann auch das Training selbst abzu-

sichern ist, um zu verhindern, dass ein Angreifer aktiv und bewusst Einfluss darauf nimmt.
Zusätzlich zur gewünschten Generalisierung über mehrere Fahrzeuge und Baureihen muss ein trainierter Algorithmus unterschiedliche Fahrer abstrahieren - zumindest solange im Fahrzeug nicht (weiter-)trainiert wird. Am attraktivsten erscheint daher der zuvor diskutierte Kompromiss aus Generalisierung, hoher Erkennungs- und möglichst niedriger Falschalarmrate.

9.2 Vervollständigung der lernenden Checks zur Netzwerk-basierten Überwachung

N-4 (Frequency Sensor), N-6 (Protocol Sensor), N-7 (Plausibility Sensor) und N-8 (Consistency Sensor) lassen sich nicht oder zumindest nicht vollständig aus standardisierten Spezifikationsdokumenten ableiten, siehe Abschnitt 4.4.1. Aufgrund der Priorisierung fokussierte sich diese Arbeit auf den Plausibility Sensor. In Zukunft sind auch die anderen drei Sensoren genauer zu betrachten, um eine vollständige Abdeckung der Überwachung zu erreichen:

- N-4: Falls eine Nachricht zyklisch versendet wird, ist dies üblicherweise in der Kommunikationsmatrix festgehalten und deren Periode lässt sich mittels statischen Checks überwachen. Anders verhält es sich bei sporadischen Nachrichten, welche keine Zykluszeit besitzen. Ein Punkt zukünftiger Forschungsarbeiten sollte daher die Untersuchung von Mechanismen sein, welche sporadische Kommunikation als normal beziehungsweise anormal klassifizieren können. Eventuell hilft das Einführen einer minimalen und maximalen Zykluszeit oder eines Mindestabstands zwischen zwei Nachrichten.
  Ein anderer Ansatz könnte auf dem Vorgehen von Marchetti et al. basieren [MS17]. Dabei wird während einer Trainingsphase anhand von CAN-Aufzeichnungen eine binäre Übergangsmatrix angelernt, die beschreibt, welche Botschaften aufeinander folgen können. Unter

der Annahme, dass die Kommunikation einer Regelmäßigkeit unterliegt, ist eine unbekannte Botschaftsfolge als Anomalie zu klassifizieren. Positiv ist der vergleichsweise geringe Speicherverbrauch für die Übergangsmatrix sowie der geringe Rechenaufwand bei der Inferenz. Eine mögliche Erweiterung des Ansatzes im Hinblick auf sporadische Kommunikation ist das Verwenden von Übergangswahrscheinlichkeiten (Botschaft X folgt mit $x\%$ Wahrscheinlichkeit auf Botschaft Y, wobei $x > 0$) anstatt der binären Matrix. Damit lässt sich die Gesamtwahrscheinlichkeit für eine Botschaftsfolge berechnen und basierend darauf kann die finale Klassifizierung mittels Schwellwert erfolgen.

- N-6: In der Kommunikationsmatrix ist ebenfalls vermerkt, welche Nachrichten der Fahrzeugdiagnose zugeordnet sind. Deshalb lassen sich hierfür statische Checks ableiten. Informationen über OEM-spezifische beziehungsweise applikative Protokolle wie Request/Response-Verfahren sind nicht enthalten. Eine zukünftige Untersuchung könnte klären, ob es möglich ist, OEM-Spezifika ohne großen Mehraufwand für die Fahrzeughersteller in die Kommunikationsmatrix mit aufzunehmen. Noch hilfreicher wären Verfahren, die Kommunikationsstrukturen beziehungsweise -paradigmen automatisch erkennen und überwachen.

- N-8: Großes Potential bieten Konsistenz-basierte Checks. Ist die Konsistenz zwischen semantisch vergleichbaren Informationen aus unabhängigen Quellen nicht mehr gegeben, lassen sich gegebenenfalls Rückschlüsse auf einen Hackerangriff oder auf ein funktionales Fehlverhalten ziehen. Ein Beispiel hierfür ist die Kombination aus Motordrehzahl, Übersetzung (Gang), Fahrzeuggeschwindigkeit und Kupplungsstatus. Diese vier Signale stehen in Verbindung zueinander. Nimmt die Fahrzeuggeschwindigkeit ab und die Motordrehzahl gleichzeitig zu (Kupplung nicht betätigt), liegt eine Anomalie vor. In zukünftigen Arbeiten gibt es einerseits die Frage zu klären, ob sich

semantische Zusammenhänge automatisiert erkennen lassen. Ein Ansatz ist das Berechnen einer Korrelationsmatrix und die anschließende Überwachung von Signalen, die eine starke Korrelation zueinander aufweisen.

Auf der anderen Seite stellt sich die Frage nach passenden Algorithmen für den Consistency Sensor. Eine Möglichkeit ist die Anpassung beziehungsweise die Erweiterung der vorgestellten Signalplausibilisierung mittels Autoencoder. Anstatt den Eingangsvektor aus der Zeitreihe eines Signals zu bilden, könnte dieser auch aus den Zeitreihen von zwei oder mehr Signalen bestehen. Eine initiale Untersuchung dazu im Rahmen der durchgeführten Machbarkeitsstudie war vielversprechend, siehe Anhang A.4. Ein erster Test mit realen OBD-II Daten konnte die Ergebnisse leider nicht bestätigen. Dies lässt sich potentiell auf mehrere Ursachen zurückführen: Zum einen ist die zusätzlich verwendete Motordrehzahl deutlich dynamischer als die Geschwindigkeit. Eine Abtastperiode von einer Sekunde - wie bei den verwendeten OBD-II Daten - ist dafür möglicherweise zu groß. Zum anderen steht in den aufgezeichneten Daten der aktuelle Gang und der Kupplungsstatus nicht zur Verfügung. Daher lohnt es sich diesen Ansatz mit realen CAN-Daten in zukünftigen Arbeiten weiter zu verfolgen. Falls einfache Autoencoder die benötigte Leistungsfähigkeit nicht erreichen, können komplexere Netzarchitekturen wie beispielsweise Convolutional Autoencoder analysiert werden. Diese Art von Convolutional Neural Networks (CNN) überträgt das Autoencoder-Prinzip auf 2D-Eingangsdaten und findet hauptsächlich in der Bildverarbeitung Anwendung. Es ist jedoch eine Übertragung auf den vorliegenden Anwendungsfall möglich, indem jede Zeitreihe eines Signals als Bildzeile interpretiert wird, angelehnt an Lee et al. [Lee+18]. Das resultierende Graustufenbild, dessen Größe sich durch die Anzahl der zusammenhängenden Signale (Anzahl der Zeilen) sowie durch das berücksichtigte Zeitfenster (Anzahl der Spalten)

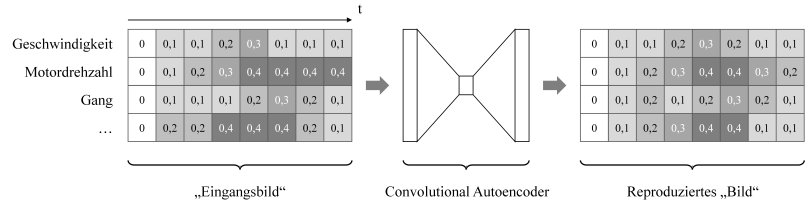

Bild 9.1: Reproduktion eines Graustufenbilds mittels Convolutional Autoencoder

ergibt, lässt sich mit Convolutional Autoencodern untersuchen. Bild 9.1 veranschaulicht das Konzept. Analog zur Plausibilisierung eines einzelnen Signals, entsteht ein neuer Eingangsvektor - in diesem Fall ein neues Graustufenbild - nach dem Sliding-Window Prinzip, sobald aktualisierte Werte der betrachteten Signale vorliegen. Die Unterscheidung zwischen normalen Daten und Anomalien erfolgt wiederum auf Basis des Reproduktionsfehlers und eines dazugehörigen Schwellwerts.

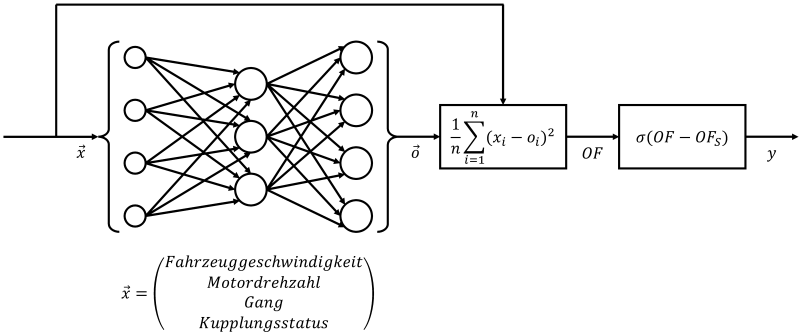

Bild 9.2: Minimierter Ansatz zur Überwachung der Konsistenz zwischen Signalen

Sollten aufgrund der Ergebnisse Convolutional Autoencoder die favorisierte Methode darstellen, ist deren erhöhter Ressourcenverbrauch und die damit verbundene Anwendbarkeit in automotive Steuerge-

räten zu untersuchen. Benötigt eine embedded Realisierung zu viele Ressourcen, könnte der Eingangsvektor auf den jeweils aktuellen Signalwert reduziert werden. In diesem Fall wäre ein normaler Autoencoder ausreichend, wie Bild 9.2 beispielhaft zeigt.

9.3 Umsetzung von statischen Checks

Statische Checks standen nicht im Fokus der vorliegenden Arbeit, siehe Bild 8.1. Da sie vollständig aus vorhandenen Spezifikationsdokumenten abgeleitet werden, ist deren Realisierbarkeit implizit gegeben. Es besteht jedoch die Herausforderung einer effizienten Implementierung, was sich anhand von CAN-Kommunikation verdeutlichen lässt.

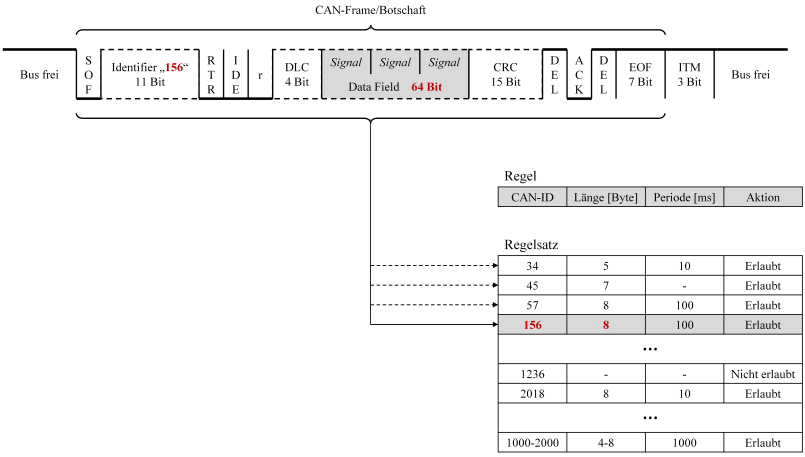

Bild 9.3: Beispiel: Statische Checks für CAN

Ein leistungsfähiges Steuergerät empfängt mehrere hundert unterschiedliche CAN-Botschaften, viele davon mit einer Zykluszeit zwischen 10 ms und 1 s. Der statische Location Sensor N-2 soll dabei überprüfen, ob eine CAN-ID auf dem entsprechenden Bussystem vorkommen darf. Dies geschieht üblicherweise anhand von definierten *Regeln*. Eine Regel umfasst

Werte für die zu prüfenden Eigenschaften - zum Beispiel die CAN-ID - sowie eine Aktion, welche bei deren Übereinstimmung mit einer empfangenen Botschaft auszuführen ist. Im einfachsten Fall erfolgt die Auswahl der zutreffenden Regel durch eine lineare Suche, wie in Bild 9.3 dargestellt. Bei 500 unterschiedlichen Botschaften, mit einer durchschnittlichen Zykluszeit von 100 ms, sind damit 5000 Suchvorgänge pro Sekunde durchzuführen, was eine deutlich steigende Prozessorlast verursacht. Hinzu kommt die analoge Überwachung der gesendeten Botschaften.

Das obige Beispiel zeigt die Notwendigkeit eines effizienten Suchalgorithmus. Am schnellsten wäre eine direkte Indizierung der zutreffenden Regel über die CAN-ID. Dieser Ansatz benötigt allerdings ein Array (siehe Regelsatz in Bild 9.3) mit 2^{11} (Standard CAN) oder 2^{29} (Extended CAN) Einträgen und ist zumindest für Extended CAN aufgrund des benötigten Speichers nicht darstellbar. Daher wird hier ein effizienter Kompromiss aus Laufzeit und Speicherverbrauch benötigt.

Zu berücksichtigen ist außerdem, dass Regeln üblicherweise nicht ausschließlich für N-2 definiert werden, sondern beispielsweise gleichzeitig N-1 (Formality Sensor) und N-4 (Frequency Sensor) abdecken. In Bild 9.3 sind daher neben der CAN-ID auch die Nutzdatenlänge und die Zykluszeit zu prüfende Eigenschaften.

Aus Komfort- und Effizienzgründen erscheint es sinnvoll, eine Regel für mehrere Botschaften zu definieren. Um dies zu ermöglichen ist es notwendig, für einzelne Eigenschaften Bereiche statt feste Werte vorzugeben. Ein Beispiel hierfür ist die unterste Regel in Bild 9.3. Diese gilt für alle Botschaften im CAN-ID-Bereich zwischen 1000 und 2000. Dadurch ergibt sich eine weitere Herausforderung: Regeln können überlappende Gültigkeitsbereiche aufweisen (im Beispiel für CAN-ID 1236). Der Suchvorgang kann nicht beim ersten Treffer abgebrochen werden und es sind zusätzlich Prioritäten zu vergeben. Außerdem war beim bisherigen Ansatz nur die Definition von erlaubten Botschaften notwendig (White-List). Durch die Regelüberlappung ist auch das Festlegen von explizit unerwünschten Botschaften

sinnvoll.
Aus den diskutierten Gründen ist die Untersuchung von Implementierungs- und Optimierungsmöglichkeiten für statische Checks ebenfalls ein Anknüpfungspunkt für zukünftige Arbeiten. Einen vielversprechenden Ansatz für die Laufzeitoptimierung liefert Heinz [Hei28]. Der vorgestellte *High Performance Packet Classification* (HiPAC) Algorithmus wurde mit Fokus auf die Paketfilterung unter Linux entworfen, eignet sich potentiell aber auch für die Optimierung der statischen Checks.

9.4 Unterstützung von Ethernet und Service-orientierter Kommunikation

Signal-basierte Überwachungen (N-3, N-7 und N-8) sind unabhängig von der darunterliegenden Netzwerktechnologie. Dies gilt nicht für Nachrichten-basierte Checks (N-1, N-2 und N-4 bis N-6). Hier besteht eine Abhängigkeit zum Nachrichtenformat. Während bei CAN einzelne Botschaften mittels CAN-ID identifiziert werden, kommen bei Ethernet MAC- und IP-Adressen sowie Portnummern zum Einsatz. Dennoch sind die von Müter et al. definierten Sensoren auf Nachrichten- und Signalebene allgemeingültig [MGF10].
Mit der Einführung von Ethernet hielt gleichzeitig ein neues Kommunikationsparadigma Einzug in die automotive E/E-Architektur: Service-orientierte Kommunikation. Dabei bieten einzelne Steuergeräte Services an und verwenden beziehungsweise abonnieren sich auf die Services anderer Steuergeräte. Obwohl hierfür bereits Lösungen aus dem IT-Umfeld existieren, wurde aufgrund der automotive Eigenheiten SOME/IP spezifiziert, welches sich in zwei Hauptbestandteile gliedert: Die Service-Discovery sowie die Übertragung von dynamisch serialisierten Daten. Dadurch gestaltet sich die gesamte Kommunikation über Ethernet deutlich dynamischer als bisher über CAN und es ergeben sich zusätzliche Überwachungsmöglichkeiten auf Serviceebene. Prinzipiell ist deren Einordnung unter dem Protocol Sensor

(N-6) möglich, was allerdings der Tragweite der neuen Kommunikationsart nicht gerecht wird. Daher sollten sich zukünftige Forschungsarbeiten explizit mit der Überwachung von Service-orientierter Kommunikation im automotive Umfeld befassen. Es wird angenommen, dass sich neue Überwachungssensoren definieren lassen, welche die Strukturierung von Müter et al. ergänzen.

9.5 Host-basierte Überwachung

Ein weiteres Feld für zukünftige Arbeiten ist die Untersuchung der Host-basierten Überwachungsanteile in Bezug auf Abarbeitungsreihenfolge, Laufzeit und Speicherverbrauch. Auch hierfür gelten die allgemeinen Randbedingungen der Erkennung in Echtzeit sowie die Ressourcenbeschränkung. Wahrscheinlich ist es für die Host-basierten Anteile sinnvoll zwischen den klassischen automotive Steuergeräten mit AUTOSAR Classic Laufzeitumgebung und leistungsstarken Steuergeräten mit Unix-basierten Betriebssystemen und/oder AUTOSAR Adaptive zu unterscheiden. AUTOSAR Classic Systeme sind in der Regel statischer ausgelegt als Unix-basierte Systeme und verhalten sich dadurch deterministischer. Im Extremfall kann der Fahrer eines Fahrzeugs die Software eines Unix-basierten Steuergeräts - wie beispielsweise der Head-Unit - sogar selbst beeinflussen, indem er neue Apps installiert. Aus diesem Grund eignen sich möglicherweise verschiedene Algorithmen für die Host-basierte Überwachung der unterschiedlichen Software-Plattformen. Dies gilt es in Zukunft näher zu untersuchen und zu evaluieren, wobei für AUTOSAR Classic Steuergeräte auf jeden Fall die statische Konfiguration berücksichtigt werden sollte.

9.6 Unterstützung von spezieller Hardware

Neben einer reinen Software-Lösung für statische Checks, lassen sich einzelne Teile davon auch in spezielle Hardware auslagern, um so Ressour-

cen des Mikrocontrollers zu sparen. Ein Beispiel ist der neu am Markt erhältliche NXP TJA115x CAN/CAN FD Transceiver [Ada16] [NXP18]. Nachdem die zu sendenden CAN-IDs in Hardware konfiguriert wurden, garantiert der Baustein, dass kein anderer Busteilnehmer eine gültige Botschaft mit gleicher CAN-ID senden kann. Sollte eine Botschaft mit gleicher CAN-ID empfangen werden, zerstört der Transceiver diese noch während der Übertragung mit einem aktiven Error-Flag. Ähnlich verhält es sich in Senderichtung. Der Baustein versendet lediglich Botschaften mit einer ihm bekannten CAN-ID. Zusätzlich realisiert er eine Bandbreitenbegrenzung und verhindert somit Bus-Flooding. Damit kann die Hardware große Teile des Location Sensors übernehmen und die Realisierung des Frequency Sensors unterstützen.

Zusätzlich zu Transceiver-Bausteinen, liefern auch die zumeist integrierten Kommunikationscontroller wertvolle Informationen, die Rückschlüsse auf Fehlverhalten oder Angriffe erlauben. Im Fall von CAN sind es unter anderem die Fehlerzähler in Sende- und Empfangsrichtung. Diese werden durch fehlerhafte Botschaften (Error-Flag) entsprechend erhöht und bei fehlerfreier Kommunikation verringert. Im Normalbetrieb sollten die Zähler einen sehr niedrigen Stand aufweisen. Eine kürzlich veröffentlichte Denial-of-Service Attacke nutzt deren Funktion, um spezifische Steuergeräte von der Kommunikation auszuschließen [Mag17]. Durch regelmäßiges Auslesen der Zähler sind solche Angriffe ebenfalls zu detektieren.

Im Kontext der lernenden Checks beziehungsweise der untersuchten Autoencoder ist eine Auslagerung der Inferenz und/oder des Trainings auf spezielle Hardware interessant. Erste Chiphersteller arbeiten an der Integration von entsprechenden Beschleunigern in automotive Mikrocontroller, sodass zukünftig eine Alternative zu Field Programmable Gate Array (FPGA)-basieren Ansätzen zur Verfügung steht.

Die optimale Ausnutzung von Hardware-Eigenschaften und deren reibungslose Integration in das Automotive Observer-Konzept könnte Gegenstand zukünftiger Forschungsarbeiten sein. Außerdem sind die erwähnten, auto-

matisierten Gegenmaßnahmen zu beleuchten, wie im nächsten Unterkapitel ausführlicher diskutiert wird.

9.7 Backend-Anbindung und das Einleiten von Gegenmaßnahmen

Die letzten diskutierten Anknüpfungspunkte betreffen das Speichern von erkannten Anomalien, die Kommunikation des Observers mit externer Infrastruktur und das Ergreifen von Gegenmaßnahmen, siehe Anomaly Logging in Bild 8.1.

Nachdem eine Anomalie erkannt wurde, sind die dazugehörigen Informationen abzuspeichern und an ein Backend-System zu übertragen. Damit erhält der Fahrzeughersteller einen Gesamtüberblick über seine Fahrzeugflotte und kann über weitere Reaktionen entscheiden. Daraus ergeben sich Fragestellungen für weiterführende Forschungsarbeiten: Wie lässt sich eine Anomalie präzise und formal beschreiben? Welche Informationen sind in diesem Zusammenhang zwingend abzuspeichern und zu übertragen? Welche Informationen sind für zusätzliche Analysen hilfreich?

Im Backend kann durch weiterführende menschliche oder automatisierte Analyse über Gegenmaßnahmen - wie beispielsweise ein Software-Update - entschieden werden, wobei die Behandlung der gemeldeten Anomalien ein separater Forschungszweig ist, auf den an dieser Stelle nicht näher eingegangen wird. Da der Observer die Erkennung von Anomalien direkt in Steuergeräten realisiert, ist ein nächster Schritt das automatische Ergreifen von Gegenmaßnahmen im Fahrzeug. Dies wäre sowohl für Safety- als auch für Security-Aspekte ein großer Gewinn. Dabei ist jedoch sicherzustellen, dass keine unerwünschten, negativen Seiteneffekte auftreten. Wird zum Beispiel eine Nachricht, die eine automatische Notbremsung auslösen würde, ungerechtfertigt blockiert, hätte dies unter Umständen fatale Folgen.

Da die untersuchten lernenden Checks auf Wahrscheinlichkeiten beruhen,

sind risikobehaftete Gegenmaßnahmen aktuell nicht erwünscht. Weitergehende Forschungstätigkeiten könnten an diesem Punkt ansetzen. Bevor hart in die Funktionalität eines Fahrzeugs eingegriffen wird, könnte die Interaktion mit dem Fahrer eine erste Reaktion auf erkannte Anomalien darstellen. Hoppe et al. diskutieren eine dreistufige Rückmeldung an den Fahrer, je nach Kritikalität der erkannten Anomalie [HKD08] [HKD09]. Die drei Stufen gliedern sich in optische, akustische und haptische Rückmeldung. Allerdings ergeben sich auch hier zunächst weitere Forschungsfragen. Wie lässt sich beispielsweise die Kritikalität einer Anomalie bewerten? Außerdem ist die Interaktion mit dem Fahrer kritisch zu hinterfragen. Ist ein Fahrer in der Lage angemessen zu reagieren oder wird er durch Warnmeldungen lediglich verwirrt und abgelenkt? Bei zu vielen Falschalarmen besteht außerdem die Gefahr, dass ein Fahrer die Meldungen ignoriert oder sogar gänzlich abschaltet.

Zumindest die Diskussion bezüglich der Anomalie-Wahrscheinlichkeit ist für statische Checks obsolet. Erkannte Anomalien stellen (bei richtiger Konfiguration und Implementierung) ein Fehlverhalten oder einen Angriff dar. Die Frage der angemessenen Reaktion stellt sich aber auch hier. Für invalide Kommunikationsnachrichten ist eine realisierbare Gegenmaßnahme die Implementierung einer Firewall. Dazu wäre lediglich eine kleine Anpassung der Spalte *Aktion* in Bild 9.3 notwendig. Anstatt *erlaubt* und *nicht erlaubt* könnte die Aktion der Firewall mit *accept* beziehungsweise *drop* hinterlegt werden. Der Eintrag in den Fehlerspeicher sollte zusätzlich erfolgen.

A Anhang

A.1 Karl Steinbuch und die Lernmatrix

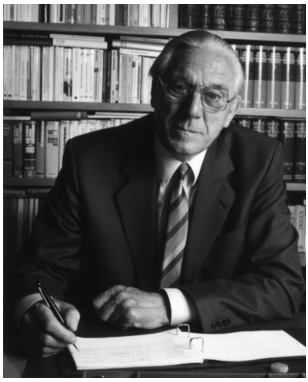

Bild A.1: Karl Steinbuch[1]

Von der ausbleibenden Finanzierung für die Erforschung von künstlichen neuronalen Netzen in den 70er Jahren war auch Karl Steinbuch betroffen, der mit seiner Lernmatrix zu den Pionieren in diesem Bereich zählt. Karl Steinbuch, geboren 1917 in Stuttgart-Cannstatt und gestorben 2005 in Ettlingen, ist außerdem Namensgeber der *Informatik* [Ste57] und Gründer des Instituts für Nachrichtenverarbeitung an der Universität Karlsruhe - dem heutigen Institut für Technik der Informationsverarbeitung (ITIV) am Karlsruher Institut für Technologie (KIT). 1961 veröffentlichte er das Konzept der Lernmatrix [Ste61], welche ein künstliches neuronales Netz darstellt. Bild A.2 zeigt die dazugehörige Zeichnung aus [Ste61].

Die Spalten in Bild A.2 repräsentieren die Eingangssignale (Eigenschaften), welche in diesem Fall komplementär vorliegen. Das „Auge“ ist ein optischer Sensor, der schwarze und weiße Pixel in die elektrischen Eingangssignale umwandelt. Jede Zeile stellt ein mögliches Ausgangssignal (Bedeutung) dar. An den Kreuzungspunkten von Spalten und Zeilen können Verbindungen (Reflexe) entstehen, die einem elektrischen Widerstand ent-

[1] Quelle: http://www.etit.kit.edu/1076.php

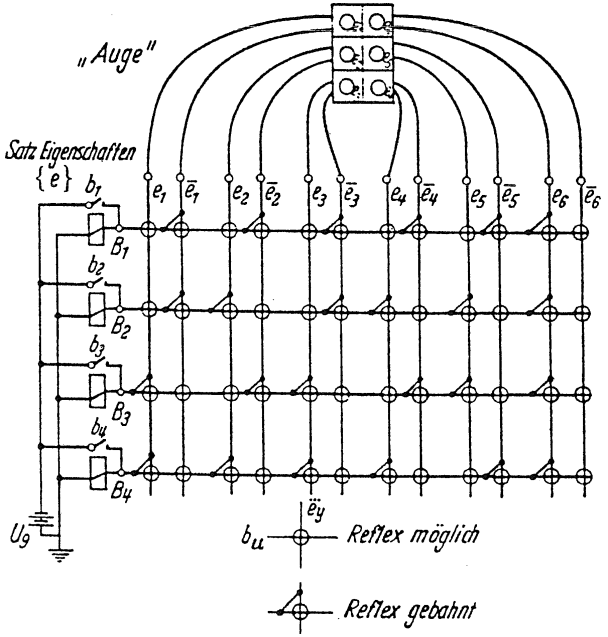

Bild A.2: Die Lernmatrix [Ste61]

sprechen. Das Besondere daran ist, dass diese Widerstände eine lineare aber auch eine nicht-lineare Kennlinie aufweisen können. Über die Zeilen der Lernmatrix werden die einzelnen Ströme aufsummiert, welche sich durch die Eingangssignale und die Widerstände an den Verbindungsstellen ergeben. Der Relais-Mechanismus auf der linken Seite von Bild A.2 wählt die Zeile mit dem größten Strom aus und selektiert so das bedeutendste Ausgangssignal.

Überführt man einen Teil der Lernmatrix in die heute übliche Darstellung für neuronale Netze, entsteht Bild A.3 (linke Seite), welches [Hil95] entnommen ist. Die Summation als Kern des Neurons entspricht einer Zeile der Lernmatrix. Die potentiell nicht-linearen Widerstände an den Kreu-

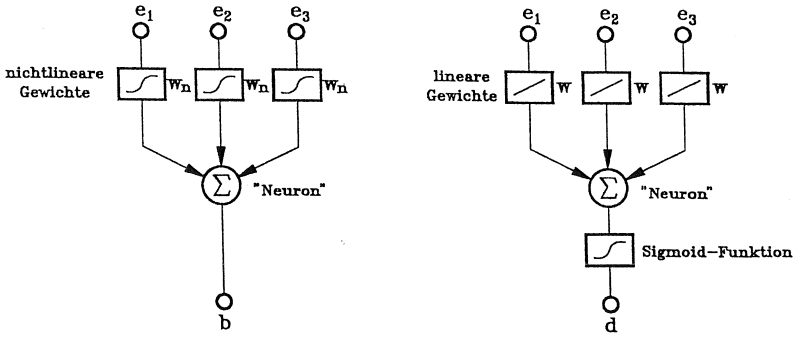

Bild A.3: Vergleich Lernmatrix und Perceptron [Hil95]

zungspunkten lassen sich als entsprechende Kantengewichte darstellen und ersetzen die linearen Kantengewichte sowie die Aktivierungsfunktion des Perceptrons. Wichtig ist, dass sich die nicht-linearen Kantengewichte individuell einstellen lassen und damit eine größere Variabilität entsteht [Hil95]. Die rechte Seite von Bild A.3 zeigt das Perceptron zum Vergleich, wobei hier bereits eine nicht-lineare Aktivierungsfunktion, die Sigmoidfunktion, verwendet wird.
Erwähnenswert ist auch, dass Steinbuch sich bereits zu Beginn Gedanken über eine mögliche Anordnung von Lernmatrizen in Schichten gemacht hat [Ste61]. Diese Struktur ist wiederum vergleichbar mit dem mehrschichtigen Perceptron, das in Abschnitt 2.2.3 beschrieben ist. Weiterführende Ideen bezüglich der Lernmatrix beschreibt er in seinem Buch [Ste63].
Steinbuch stellte die Forschung auf dem Gebiet der künstlichen neuronalen Netze aufgrund der fehlenden Finanzierung ein und die Lernmatrix geriet in den folgenden Jahren in Vergessenheit.

A.2 Safety-Mechanismen für automotive Steuergeräte

Dieser Teil des Anhangs ergänzt die Diskussion der Safety-Mechanismen für automotive Steuergeräte aus Unterkapitel 3.1. Während in Unterkapitel

3.1 auf die Absicherung der Fahrzeug-internen Kommunikation eingegangen wird, sind hier Mechanismen für die Überwachung von Steuergeräte-internen Abläufen und Eigenschaften beschrieben (vgl. Host-basierte Überwachung).

A.2.1 Betriebssystem

AUTOSAR Classic erweitert den Funktionsumfang des OSEK/VDX Betriebssystems (engl.: Operating System (OS)) und führt dafür die sogenannten *Scalability Classes* (SC) ein [AUT17f]. SC1 ist die kleinste Ausbaustufe und ergänzt unter anderem *Schedule Tables* sowie ein *Stack Monitoring*, siehe Anhang A.2.1. Auf die *Timing Protection* (SC2) und die *Memory Protection* (SC3) wird nachfolgend näher eingegangen. SC4 umfasst alle genannten Erweiterungen, wie Bild A.4 verdeutlicht.
Der Fokus liegt auf AUTOSAR Classic, weshalb der Zusatz *Classic* entfällt.

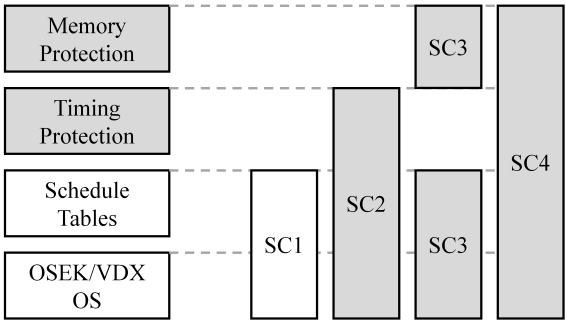

Bild A.4: Erweiterungen im AUTOSAR Betriebssystem[2]

Zum besseren Verständnis der nachfolgenden Sicherheitsmechanismen zeigt Bild A.5 das grundlegende Zustandsdiagramm eines AUTOSAR OS-Tasks, inklusive der entsprechenden Übergänge in Anlehnung an [AUT17f]. Alle Tasks starten im Zustand *Suspended*. Nach seiner Aktivierung ist ein

[2] Quelle: Vector Informatik GmbH

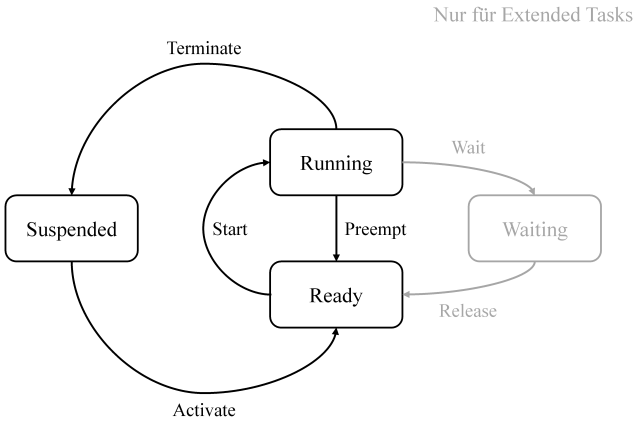

Bild A.5: Taskzustände im AUTOSAR OS[3]

Task *Ready* und wartet bis ihm der Scheduler Central Processing Unit (CPU)-Laufzeit zuweist. Sobald dies geschehen ist, nimmt er den Zustand *Running* ein. Die Ausführung dauert so lange an, bis der Task beendet ist (Übergang in *Suspended*), er unterbrochen wird (Übergang in *Ready*) oder auf einen Wartepunkt läuft (Übergang in *Waiting*). Wartepunkte sind jedoch nur bei Extended Tasks erlaubt.

In den folgenden zwei Abschnitten werden Timing und Memory Protection näher erläutert. Beide Sicherheitsmechanismen beziehen sich auf Tasks und Interrupt-Service-Routinen der Kategorie zwei (ISR2). Das OS verwaltet ISR2 während eine Interrupt-Service-Routine der Kategorie eins (ISR1) unabhängig vom OS ausgeführt wird, d.h. es findet unter anderem kein Kontextwechsel statt. Daraus ergibt sich eine Einschränkung der erwähnten Sicherheitsmechanismen: Die Überwachung auf Sub-Task- oder C-Funktionsebene sowie von ISR1 ist damit nicht möglich.

[3] In Anlehnung an [AUT17f]

Timing Protection

Die Timing Protection erlaubt es dem Anwender zeitliche Eigenschaften von Tasks und ISR2 zu überwachen. Für die zeitlich fehlerfreie Ausführung eines Tasks oder einer ISR2, muss genügend CPU-Zeit zur Verfügung stehen und entsprechend zugewiesen sein. Außerdem darf keine zu lange Unterbrechung auftreten, zum Beispiel aufgrund einer nicht verfügbaren Ressource. Die Timing Protection überwacht daher drei Messwerte [AUT17f]: Die reine Ausführungszeit, den Abstand zwischen zwei aufeinander folgenden Übergängen in den *Ready*-Zustand beziehungsweise ISR2 Aktivierungen sowie die Dauer von Interrupt- und Ressourcensperren. Maximale Ausführungs- und Sperrzeit sowie minimaler Aktivierungs-/Releaseabstand sind während der Entwicklungszeit zu bestimmen und im OS statisch zu konfigurieren. Sollte zur Laufzeit eine Verletzung auftreten, kann der Anwender im aufgerufenen *ProtectionHook* über das weitere Vorgehen entscheiden. Die Möglichkeiten reichen von der Wiederaufnahme des normalen Betriebs bis zum Herunterfahren des gesamten Steuergeräts. Eine Einschränkung der Überwachung auf bestimmte Tasks und ISR2 ist möglich und in den meisten Fällen sinnvoll.

Memory Protection und OS-Applikationen

Ein OS mit Memory Protection überwacht in Kombination mit der *Memory Protection Unit* (MPU) des Mikrocontrollers Speicherzugriffe zur Laufzeit, wobei zwischen Stack, Daten und Code unterschieden wird [AUT17f]. Das OS ändert bei einem Kontextwechsel (Task oder ISR2) die Konfiguration der MPU, sodass der Zugriff auf fremde Speicherbereiche möglichst stark eingeschränkt ist. Die MPU erkennt und blockiert fehlerhafte Schreib- und gegebenenfalls auch Leseoperationen. Das OS informiert die Applikation beziehungsweise den Anwender bei einer Zugriffsverletzung durch den Aufruf der *ProtectionHook*. Sollte hardwareseitig keine MPU zur Verfügung stehen, ist die beschriebene Memory Protection nicht einsetzbar. Für diesen

Fall bietet ein AUTOSAR OS die Möglichkeit den Stack Software-basiert zu überwachen. Damit werden Stackfehler bei einem Kontextwechsel erkannt.

Voraussetzung für die Verwendung der Memory Protection ist eine Aufteilung der Steuergeräte-Software in OS-Applikationen, die jeweils einen eigenen Ausführungsraum bilden [AUT17f]. Dabei ist jedes vom OS verwaltete Objekt - dazu zählen Tasks (inklusive der dazugehörigen Events), ISR2, Alarme, Schedule Tables und Counter - genau einer OS-Applikation zuzuweisen. Innerhalb einer OS-Applikation sind alle Objekte untereinander sichtbar und können sich gegenseitig verwenden. Der Zugriff auf externe Objekte ist konfigurierbar. Des Weiteren ist zwischen *trusted* und *non-trusted* OS-Applikationen zu unterscheiden. Erstere haben uneingeschränkten Zugriff auf Speicher und OS. Timing- und Memory Protection können für diese OS-Applikationen deaktiviert werden. Im Umkehrschluss wird erwartet, dass es hier zu keinen Laufzeitproblemen kommt. *Non-trusted* OS-Applikationen besitzen nur eingeschränkten Zugriff auf Speicher und OS-Funktionen. Das OS selbst ist *trusted*. Wenn OS-Applikationen mit unterschiedlichen ASIL-Anforderungen Anwendung finden, muss das OS den sicheren Kontextwechsel gewährleisten und dem höchsten eingesetzten ASIL entsprechen.

OS-Applikationen verfolgen das Prinzip der geschützten Ausführungsräume, welches sich auch bei nicht-AUTOSAR Systemen wiederfindet. In höheren Betriebssystemen bildet der Prozess einen abgeschlossenen Ausführungsraum und damit eine sogenannte *Error Containment Region*, im weiteren Sinne vergleichbar mit einer AUTOSAR OS-Applikation. POSIX-Systeme kommen auf leistungsstärkeren Prozessoren zum Einsatz, die üblicherweise eine Memory Management Unit (MMU) enthalten. MMUs erweitern den Funktionsumfang von MPUs zum Beispiel um virtuelle Speicheradressierung.

Service Protection

Neben der Timing- und Memory Protection schützt sich das Betriebssystem selbst vor inkonsistenten Zuständen, indem es die Verwendung verschiedener Funktionen einschränkt [AUT17f]. So können Teile des API nur zu bestimmten Zeitpunkten verwendet werden. Anderenfalls wird ein Fehler zurückgegeben.

A.2.2 Watchdog

Sind die Sicherheitsmaßnahmen des Betriebssystems nicht ausreichend, da beispielsweise eine Überwachung auf SW-C oder C-Funktionsebene benötigt wird, ist die Verwendung von *Watchdogs* üblich. Ein Watchdog besteht in der Regel aus einem Software- und einem Hardware-Anteil. Der Software-Anteil überwacht die korrekte Abarbeitung von Programmteilen und triggert die Watchdog-Hardware in regelmäßigen Abständen. Im Fehlerfall existieren mehrere Reaktionsmöglichkeiten, vom Notifizieren der Applikationssoftware bis zum Reset des Mikrocontrollers oder des Steuergeräts. Letzteres realisiert die Watchdog-Hardware, falls die zuvor erwähnten Trigger für eine bestimmte Zeit ausbleiben[4]. Die dafür notwendige Schaltung kann im Mikrocontroller selbst oder in einem extern angeschlossenen System-Basis-Chip (SBC) beziehungsweise Watchdog-Chip verortet sein.

AUTOSAR definiert einen Software-Stack, bestehend aus *Watchdog Manager*, *Watchdog Interface* und *Watchdog Treiber*. Der Treiber übernimmt die Ansteuerung der Hardware und das Interface kümmert sich um die Abstraktion des Steuergeräte-Aufbaus in Bezug auf Watchdogs. Damit ist die Implementierung des Watchdog Managers auf Serviceebene hardwareunabhängig [AUT17g]. Zur besseren Strukturierung der Überwachung werden sogenannte *Supervised Entities* verwendet. Diese stellen eine logische Überwa-

[4] Sofern von der Hardware unterstützt, ist ein Software-basierter Reset des Mikrocontrollers ebenfalls möglich.

chungseinheit dar und sind unabhängig von anderen Software-Konstrukten wie Basissoftware-Modul, SW-C und Runnable. Jede Supervised Entity hat einen eigenen Überwachungsstatus und enthält mindestens einen *Checkpoint*. Checkpoints sind wichtige Punkte im Programmablauf deren Erreichen dem Watchdog Manager über ein entsprechendes API zu melden ist. Auf Basis dieser Informationen findet die Überwachung im Watchdog Manager statt. Das Modul stellt hierfür drei unterschiedliche Mechanismen breit, die im Folgenden näher beschrieben sind.

Alive Supervision

Per Alive Supervision überwacht der Watchdog Manager die periodische Ausführung von Supervised Entities [AUT17g]. Ein zu häufiges oder zu seltenes Abarbeiten stellt ein Fehlverhalten dar. Für diese Art der Überwachung ist im einfachsten Fall nur ein Checkpoint innerhalb einer Supervised Entity notwendig. Es können allerdings auch mehrere unabhängige Checkpoints konfiguriert werden.

Deadline Supervision

Für nicht-periodische Supervised Entities ist die Alive Supervision unbrauchbar. Hier kommt stattdessen die Deadline Supervision zum Einsatz. Damit überwacht der Watchdog Manager, dass zwischen dem Erreichen zweier Checkpoints einer Supervised Entity nicht zu viel oder zu wenig Zeit vergeht [AUT17g]. Daher sind für jede Deadline Supervision zwei aufeinanderfolgende Checkpoints zu konfigurieren.

Logical Supervision

Mit der Logical Supervision gibt es einen dritten Watchdog Manager Mechanismus [AUT17g]. Dieser überwacht den logisch korrekten Programmablauf. Die mögliche Reihenfolge der konfigurierten Checkpoints kann als

```
CP0  i = 0;
CP1  while (i < n) {
         if (a[i] < b[i])
CP2          a[i] = b[i];
         else
CP3          a[i] = 0;
         i++;
CP4  }
```

Bild A.6: Code-Beispiel mit Watchdog Manager Checkpoints[5]

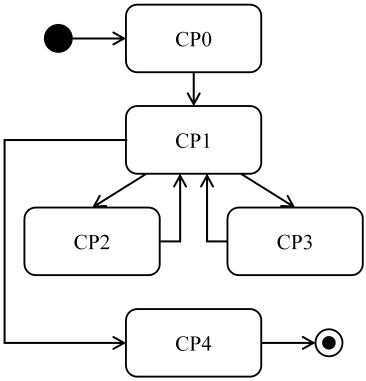

Bild A.7: Checkpoint-Graph zu A.6[5]

Graph dargestellt werden. Knoten repräsentieren Checkpoints und Kanten die entsprechenden Übergänge. Der Watchdog Manager erkennt zur Laufzeit das Auftreten von nicht konfigurierten Checkpoint-Sequenzen. Das Timing spielt hierbei keine Rolle. Bild A.6 zeigt ein Code-Beispiel aus der AUTOSAR Watchdog Manager Spezifikation [AUT17g]. Auf der linken Seite sind die definierten Checkpoints angegeben (abweichend zur AUTOSAR Spezifikation). Der dazugehörige Graph ist in Bild A.7 zu sehen. Im Gegensatz zu Alive und Deadline Supervision erlaubt dieser Mechanismus auch eine Supervised Entity übergreifende Anwendung.

A.2.3 Sichere Initialisierung

Eine fehlerfreie Initialisierung des Steuergeräts beziehungsweise von dessen Software ist Voraussetzung für einen reibungslosen Betrieb. Für Initialisierungsfunktionen gelten daher gegebenenfalls Sicherheitsanforderungen nach ASIL A-D [Intb]. Diese betreffen jedoch ausschließlich den Entwicklungsprozess und erfordern in der Regel keine zusätzlichen technischen Maßnahmen.

[5] Adaptiert von [AUT17g]

A.2.4 Sichere Verwaltung von nicht-flüchtigem Speicher

Das sichere Abspeichern und Auslesen von nicht-flüchtigem Speicher ist ein weiterer Safety-Mechanismus, der heute bereits im Einsatz ist. Dabei wird der Speicher als unsichere Datensenke beziehungsweise -quelle betrachtet, ähnlich zu den unsicheren Bussystemen und Netzwerken bei der Fahrzeug-internen Kommunikation. Bei sicherheitskritischen Nachrichten kommt hier eine End-to-End Protection mit CRC und Sequenzzähler zum Einsatz (siehe Abschnitt 3.1.1). Vergleichbar dazu erhält jeder Speicherblock eine separate CRC, um beim Lesen der Daten eine Integritätsprüfung durchführen zu können. Einen Sequenzzähler sieht die Basissoftware hingegen nicht vor. Bei Bedarf kann dieser applikativ umgesetzt werden.

A.3 Security-Mechanismen für automotive Steuergeräte

A.3.1 Security-Prozesse

Im IT-Bereich ist unter anderem der *Security Development Lifecycle* (SDL) von Microsoft [Mica] anerkannt und findet in seiner originalen oder in einer angepassten Form Anwendung. Zur Identifikation und Klassifizierung von Bedrohungen enthält der SDL das von Garms et al. (ebenfalls Microsoft) vorgestellte STRIDE-Modell [Sho] [Sho08]. Es wurde 2001 veröffentlicht [HL01] und basiert auf sechs grundlegenden Absichten von Angreifern:

- **S**poofing: Jemanden oder etwas imitieren
- **T**ampering: Manipulation von Code oder Daten
- **R**epudiation: Abstreiten der Verantwortlichkeit für etwas
- **I**nformation Disclosure: Geheime oder private Daten veröffentlichen
- **D**enial of Service: Dienste außer Kraft setzen oder stören
- **E**levation of Privilege: Sich unbefugten Zugriff verschaffen

Eine Bewertung der erkannten Bedrohungen kann beispielsweise anhand der Kriterien des DREAD-Modells erfolgen [HL03], welches allerdings in der Praxis inzwischen umstritten ist. Es enthält teils subjektive Kriterien und kann damit zu unterschiedlichen Bewertungen einer Bedrohung führen. Die Klassifizierungsmerkmale können dennoch einen Gedankenanstoß geben und sind daher nachfolgend aufgeführt.

- **D**amage Potential: Das Schadenspotential
- **R**eproducibility: Reproduzierbarkeit/Wiederholung des Angriffs
- **E**xploitability: Aufwand, um eine Schwachstelle auszunutzen
- **A**ffected Users: Mögliche Anzahl betroffener Anwender
- **D**iscoverability: Wahrscheinlichkeit, dass eine Sicherheitslücke bekannt wird

Während das STRIDE-Modell im Wesentlichen aus Sicht des Angreifers aufgestellt ist, kann eine Betrachtung der Angriffssicherheit auch aus Sicht der informationstechnischen Grundwerte erfolgen, auch bekannt als primäre Schutzziele. Diese setzen sich aus Vertraulichkeit (engl.: confidentiality), Integrität (engl.: integrity) und Verfügbarkeit (engl.: availability) zusammen [Bun12] und bilden das sogenannte *CIA*-Dreieck, siehe Abbildung A.8.

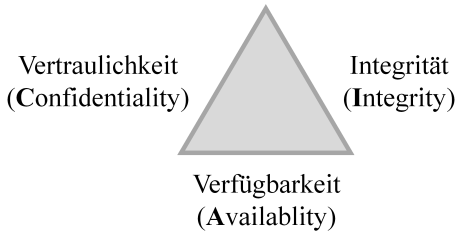

Bild A.8: CIA-Dreieck

Der Umgang mit beziehungsweise die Darstellung von weiteren Schutzzielen, insbesondere der Authentifizierung (engl.: authentication), Autorisierung (engl.: authorization) und Nichtabstreitbarkeit (engl.: non-repudiation), unterscheidet sich in der Literatur. Bei einer hierarchielosen Betrachtung [Micb] lässt sich jedem Schutzziel eine Bedrohung des STRIDE-Modells zuordnen:

- Authentication: Spoofing
- Integrity: Tampering
- Non-Repudiation: Repudiation
- Confidentiality: Information Disclosure
- Availability: Denial of Service
- Authorization: Elevation of Privilege

Alternativ kann argumentiert werden, dass die weiteren Schutzziele von den drei Grundwerten abgeleitet sind [SW00]. Spoofing, Repudiation und Elevation of Privilege bedrohen damit potentiell mehrere Grundwerte.

A.3.2 Software-Stack für kryptografische Funktionen

Innerhalb von AUTOSAR Classic stellt der Crypto-Stack grundlegende Funktionen für viele Security-Mechanismen bereit. Dazu zählen zum Beispiel das Ablegen von Schlüsselmaterial und Zertifikaten, das Berechnen von kryptografischen Algorithmen und die Generierung von Zufallszahlen. Der Crypto-Stack besteht aus *Crypto Service Manager* [AUT17h], *Crypto Interface* und *Crypto Treiber*. Der Treiber ist für die Ansteuerung der Crypto-Hardware zuständig, die im Mikrocontroller integriert oder extern angeschlossen sein kann. Ein Beispiel ist das im Rahmen von EVITA spezifizierte *Hardware Security Module* (HSM), in seinen Ausprägungen *light*,

medium und *full* [EVI12]. Sollte keine dedizierte Hardware zur Verfügung stehen, kann das Treiber-Modul die entsprechende Ablage oder Berechnung auch auf dem Mikroprozessor selbst durchführen, was allerdings ein Sicherheitsrisiko mit sich bringt. Daher wird in den meisten Fällen ein sogenannter *Hardware Trust Anchor* (HTA), zum Beispiel in Form eines HSMs, bevorzugt. Das Interface-Modul abstrahiert wiederum die dazugehörige Steuergeräte-Architektur. Über den Crypto Service Manager kann die restliche Basis- sowie die Applikationssoftware auf die Funktionen des Crypto-Stacks zugreifen.

A.3.3 Sicheres Aufstarten

Ein Security-Mechanismus, der bereits häufig Anwendung findet, ist das sichere Aufstarten (engl.: Secure Boot). Damit wird das Ausführen von manipulierter Steuergerätesoftware verhindert. Bei diesem Verfahren startet zunächst ein sogenanntes *Boot Manager* Programm im sicheren Teil der Hardware, zum Beispiel innerhalb eines HSMs. Dessen Validität wurde zuvor vom *Boot-ROM* des Mikrocontrollers/-prozessors sichergestellt. Der Boot Manager überprüft daraufhin die Integrität und Authentizität der zu startenden Applikation beziehungsweise des zu startenden Bootloaders. Dafür wurde beim (Re-)Programmieren des Steuergeräts neben der eigentlichen Software auch ein dazugehöriger MAC und die passenden kryptografischen Schlüssel im sicheren Teil der Hardware abgelegt. Bei jedem Startvorgang berechnet der Boot Manager den MAC über die auszuführende Software und vergleicht das Ergebnis mit dem abgespeicherten Wert. Die Startsequenz wird nur fortgesetzt wenn die beiden MACs übereinstimmen.

A.3.4 Sicheres Software-Update

Die sichere Aktualisierung von Steuergerätesoftware ist eng mit dem sicheren Aufstarten verbunden. Bei einer entsprechenden Diagnoseanfrage wechselt das Steuergerät vom regulären Applikationsmodus in den Bootloader.

Dieser empfängt die aktualisierte Software. Je nach Anwendungsfall und Sicherheitsstufe findet die Übertragung verschlüsselt statt. Nach der erfolgreichen Prüfung von Integrität und Authentizität anhand eines MAC oder einer digitalen Signatur ersetzt der Bootloader den alten Softwarestand im Flash-Speicher. Wie bereits in Anhang A.3.3 erwähnt, wird zusätzlich die übertragene MAC im sicheren Teil der Hardware abgelegt.

A.3.5 Sichere Diagnose

Eine der wichtigsten Security-Maßnahmen und Voraussetzung für ein sicheres Software-Update ist die sichere Fahrzeugdiagnose. Hierfür existieren unterschiedliche Sicherheitsstufen, welche ein Diagnosetester freischalten kann. Die darin verfügbaren Diagnosedienste legt der Fahrzeughersteller individuell fest. Der Wechsel in eine höhere Sicherheitsstufe findet über ein *Seed & Key*-Verfahren statt (UDS-Service *0x27: SecurityAccess*). Dabei fordert der Diagnosetester einen *Seed* vom Steuergerät an. Auf dessen Basis wird ein Schlüssel (engl.: *Key*) berechnet und zurückgesendet. Da das Steuergerät die Berechnungsvorschrift kennt, kann es den empfangenen mit dem erwarteten Schlüssel vergleichen und gegebenenfalls die gewünschte Sicherheitsstufe freischalten. Dieses Verfahren gilt in der Wissenschaft und in der Industrie jedoch als unsicher und nicht mehr zeitgemäß. Im Rahmen der Aktualisierung des ISO 14229-1 Standards wird daher ein neuer UDS-Service (*0x29: Authentication*) spezifiziert, der die Tester-Authentifizierung anhand eines digitalen Zertifikats erlaubt. Zusätzlich ist damit eine feingranulare Beschränkung der Zugriffsrechte möglich. Die AUTOSAR Classic Platform unterstützt in Version 4.4 bereits den neuen UDS-Service, verweist aber auf die nicht öffentlich verfügbare Vorabversion der ISO 14229-1 [AUT18].

A.4 Machbarkeitsstudie für Autoencoder

Dieser Teil des Anhangs fasst die in Abschnitt 5.6.2 erwähnte Machbarkeitsstudie und deren Ergebnisse zusammen, wobei große Teile [VWeb+18b] entnommen sind. Das Ziel ist die Klärung der Fragestellung, ob eine Plausibilisierung von kontinuierlichen Signalen mittels Autoencodern prinzipiell möglich ist. Zusätzlich sollen erste Erfahrungen bezüglich der erforderlichen Netzgröße gesammelt werden.

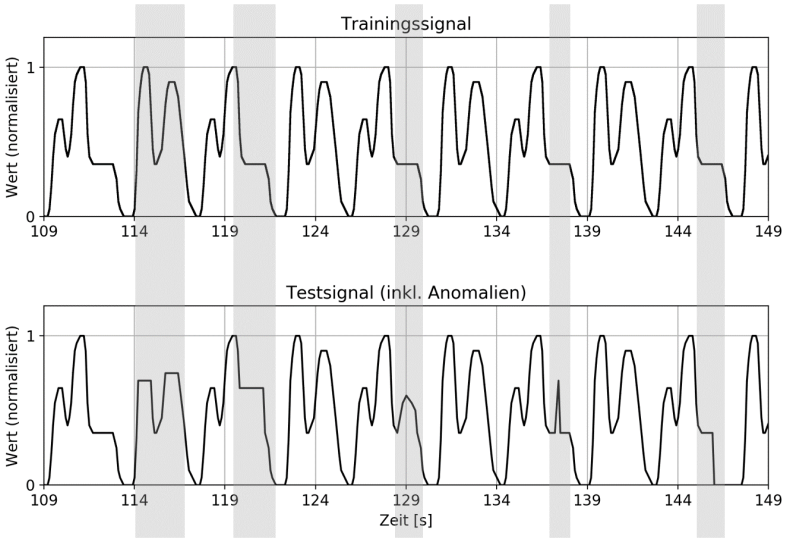

Bild A.9: Trainingssignal (oben) und Testsignal (unten) für die Machbarkeitsstudie [VWeb+18b]

Das simulierte Trainingssignal ist einem möglichen Geschwindigkeitsverlauf eines Fahrzeugs nachempfunden und hat eine Dauer von ca. 8,5 Sekunden. Im generierten Trainingsdatensatz, welcher 10.000 Signalwerte enthält, wird es periodisch wiederholt. Bild A.9 zeigt im oberen Teil einen entsprechenden Ausschnitt. Der Testdatensatz basiert auf den Trainingsdaten und wurde an unterschiedlichen Stellen manipuliert. Dies ist im unteren

Teil von Bild A.9 dargestellt. Graue Balken markieren die veränderten Bereiche. Die dritte Änderung bei Sekunde 129 ist absichtlich so gewählt, dass ein anderer, potentiell normaler Signalverlauf entsteht. An dieser Stelle sollen die untersuchten Netze keine Anomalie melden. Auch reale Trainingsdaten können nicht alle möglichen Fahrprofile enthalten. Eingesetzte Algorithmen müssen Anomalien anhand abstrakter Eigenschaften erkennen und nicht durch Auswendiglernen der Trainingsdaten. Daher soll der alternative Signalverlauf einen ersten Hinweis auf die geforderte Generalisierungsfähigkeit geben.
Mit den diskutierten Daten wurden Netze mit einer unterschiedlichen Anzahl von Ein- und Ausgangsneuronen, versteckten Schichten und Neuronen in den versteckten Schichten trainiert und getestet. Um Netze einfach beschreiben zu können, gilt die folgende Nomenklatur: $X_1 - X_2 - ... - X_n$. X_1 steht für die Anzahl von Neuronen in der Eingangsschicht, X_2 für die in der ersten versteckten Schicht usw. Des Weiteren kommen Neuronen mit sigmoidaler Aktivierungsfunktion ($f(x) = sig(x) = \frac{1}{1+e^{-x}}$) zum Einsatz.
Da die initialen Kantengewichte zufällig festgelegt werden (im Bereich von -0,05 bis 0,05), ist die Vergleichbarkeit einzelner Netze schwierig. Um diesen Effekt abzuschwächen, wurde jede Netztopologie zehn mal trainiert. Der Vergleich findet danach immer zwischen den jeweils besten Netzen statt.

A.4.1 Anzahl der Ein- und Ausgangsneuronen

Zunächst wurde der Einfluss einer unterschiedlichen Anzahl von Ein- und Ausgangsneuronen untersucht. Das kleinste betrachtete Netz hat eine 4-4-4, das größte eine 64-64-64 Topologie. Alle Netze haben eine versteckte Schicht mit gleich vielen Neuronen wie in der Ein- beziehungsweise Ausgangsschicht. Bild A.10 zeigt das Resultat. Nach Trainings- und Testsignal folgen die Ergebnisse der untersuchten Netztopologien.

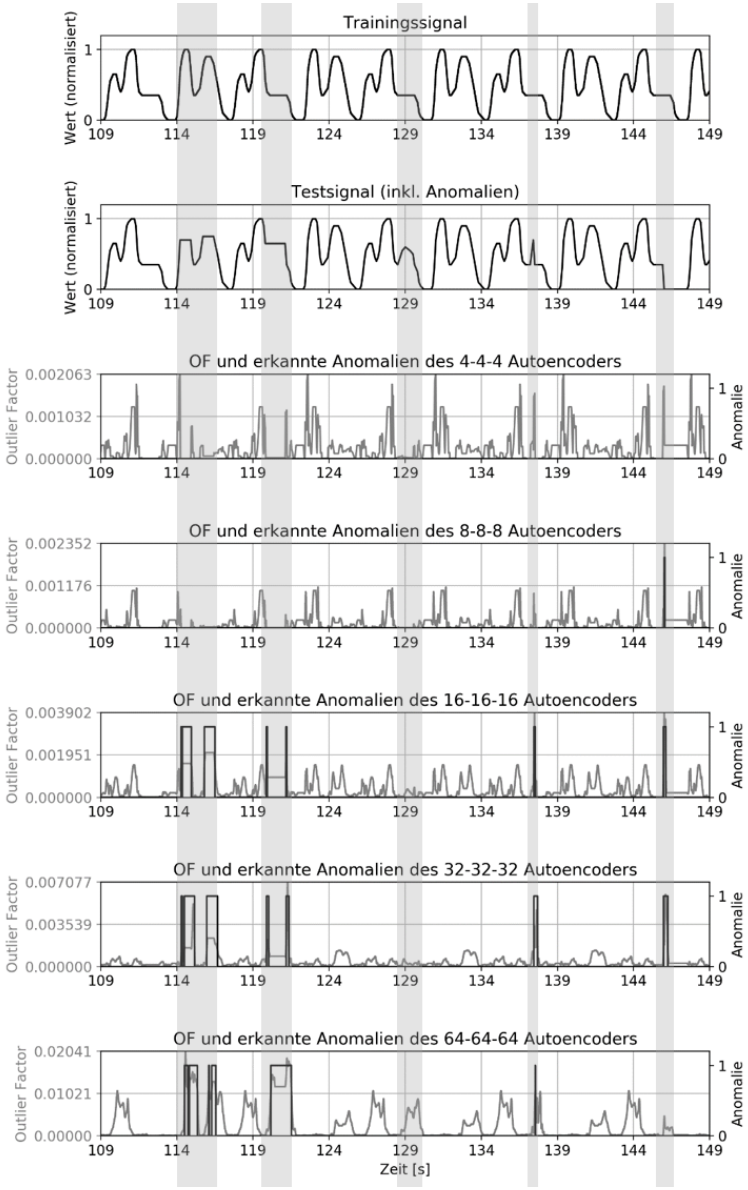

Bild A.10: Variation der Ein-/Ausgangsneuronenanzahl [VWeb+18b]

Der graue Verlauf entspricht dem Outlier Factor, d.h. dem Reproduktionsfehler, und der schwarze Verlauf zeigt die erkannten Anomalien nach Anwendung des Schwellwerts. Der Schwellwert entspricht dem maximalen Outlier Factor, den das fertig trainierte Netz bei Inferenz der Trainingsdaten erzeugt. Damit wird garantiert, dass das Trainingssignal selbst keine Anomalien hervorruft.
Das beste Ergebnis liefern die Netze mit 16-16-16 und 32-32-32 Topologie. Sie erkennen alle Anomalien, wobei sie den alternativen Signalverlauf nicht als solche markieren und damit keinen Falschalarm auslösen. Kleine Netze (4-4-4 und 8-8-8) erkennen bis auf eine Ausnahme keine Anomalien. Das größte Netz erkennt die letzte eingefügte Anomalie ebenfalls nicht. Interessant zu beobachten ist der Outlier Factor. Je größer die Netztopologie, desto ruhiger ist im Allgemeinen sein Verlauf. Zu beachten ist auch der mit größeren Netzen zunehmende Wertebereich.

A.4.2 Anzahl der Neuronen in der versteckten Schicht

Ausgehend von der 16-16-16 Topologie, welche in der ersten Untersuchung das beste Ergebnis in Bezug auf die Netzgröße lieferte, wird im zweiten Schritt die Anzahl der Neuronen in der versteckten Schicht verändert. Sowohl ein Flaschenhals (16-15-16) als auch eine Aufspreizung (16-17-16) ergeben keine nennenswerten Unterschiede, siehe Bild A.11.

A.4.3 Anzahl der versteckten Schichten und Neuronen

Aufbauend auf der vorherigen Untersuchung, werden in diesem Schritt Autoencoder mit drei versteckten Schichten verwendet. Bild A.12 zeigt, dass diese Netze sensibler reagieren und das exakte Auftreten einer Anomalie besser bestimmen können. Für die erste Anomalie erkennen beispielsweise zwei der erweiterten Netze drei von vier unnatürlichen Änderungen im Signalverlauf. Dies geht allerdings zu Lasten des Speicherverbrauchs und des Berechnungsaufwands.

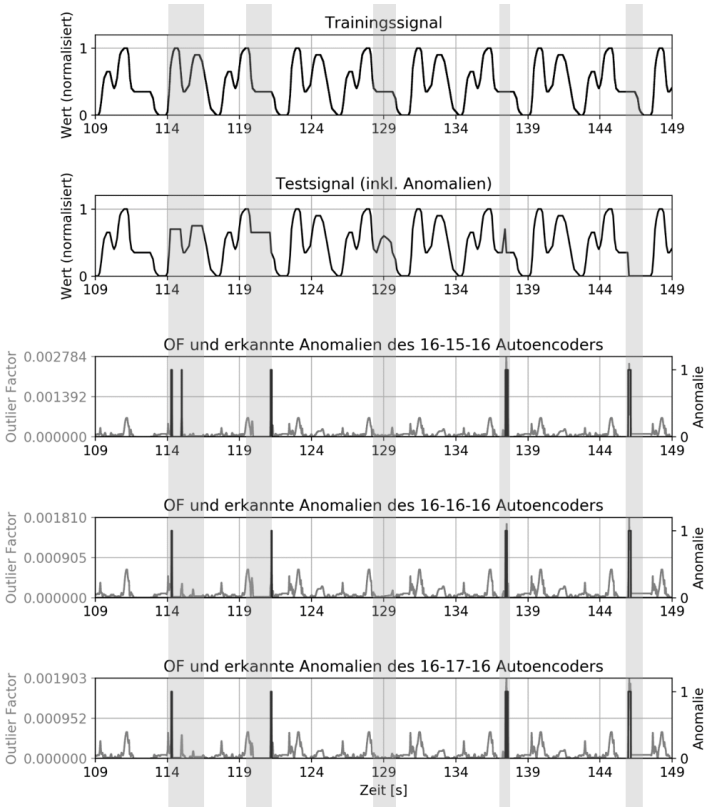

Bild A.11: Variation der versteckten Schicht [VWeb+18b]

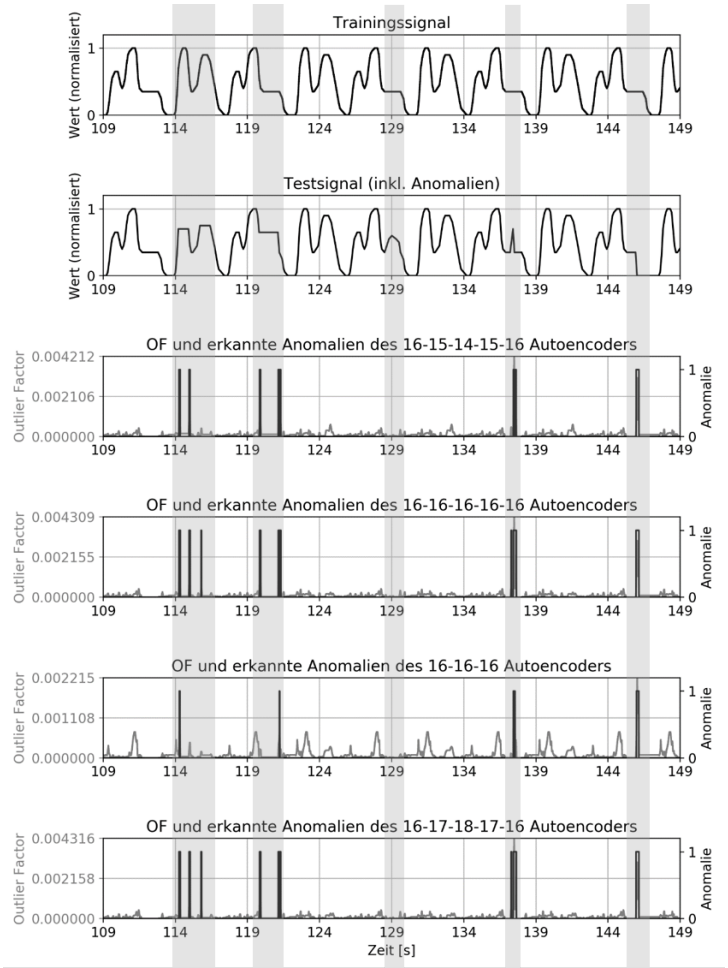

Bild A.12: Variation der Schichtenanzahl [VWeb+18b]

A.4.4 Erste und zweite Ableitung als zusätzliche Eingangsmerkmale

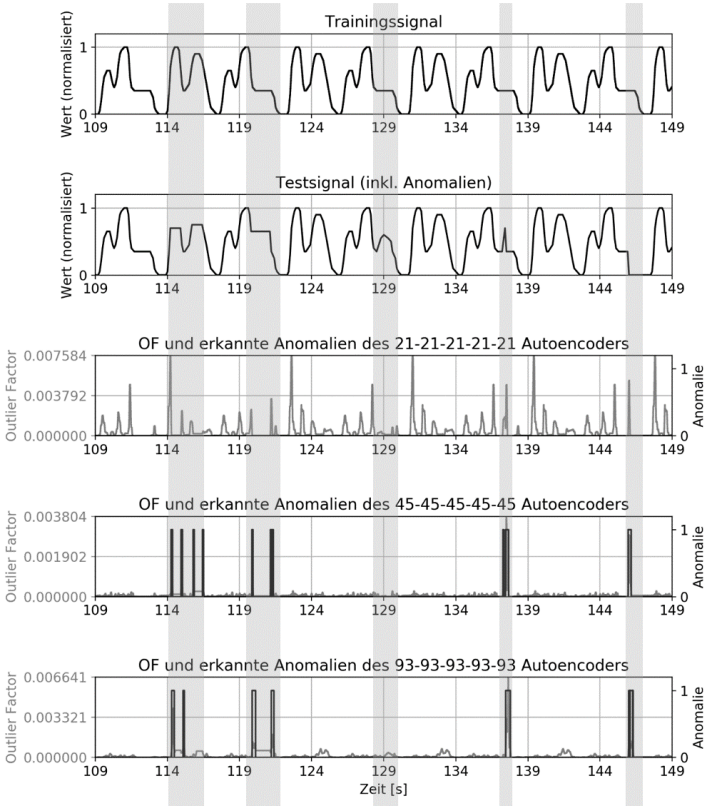

Bild A.13: Erste und zweite Ableitung als zusätzliche Eingangsmerkmale [VWeb+18b]

Bisher bestand der Eingangsvektor ausschließlich aus einer Folge von normalisierten Signalwerten. Die damit erzielten Ergebnisse lassen vermuten, dass die untersuchten Autoencoder vor allem abrupte Änderungen im Signalverlauf als Anomalie erkennen. Daher wurde der Eingangsvektor um

Merkmale zweiter Ordnung ergänzt und enthält neben den (normalisierten) Signalwerten auch deren erste und zweite Ableitung. Der Eingangsvektor eines 45-45-45 Netz setzt sich damit aus den 16 letzten Signalwerten sowie den daraus berechneten 15 ersten und 14 zweiten Ableitungen zusammen. Die entsprechend notwendige Vorverarbeitung findet wiederum in der Merkmalsgenerierung statt. Da die minimalen und maximalen Gradienten erster und zweiter Ordnung nicht in der Kommunikationsmatrix definiert sind, werden diese Werte aus den Trainingsdaten ermittelt, um damit die Normalisierung durchzuführen. Das beste Ergebnis liefert der untersuchte 45-45-45-45-45 Autoencoder. Er detektiert präzise alle ungewöhnlichen Stellen im Signalverlauf, wobei die dritte Änderung - wie gewünscht - nicht als Anomalie erkannt wird. Bild A.13 zeigt die vollständige Auswertung. Durch die hohe Neuronenanzahl steigt auch der Speicherverbrauch und die benötigte Rechenleistung deutlich an. Für ein 16-16-16 Netz sind 512 Kantengewichte zu speichern und entsprechend viele Multiplikationen je Inferenz zu berechnen $((16 \cdot 16) \cdot 2)$. Beim diskutierten 45-45-45-45-45 Netz sind es dagegen 8100 Gewichte beziehungsweise Berechnungen $((45 \cdot 45) \cdot 4)$.

A.4.5 Korrelation von Signalverläufen

Als letzten Punkt untersucht die Machbarkeitsstudie die Verwendung von korrelierenden Signalen zur Anomalieerkennung. Obwohl der Fokus aktuell auf der Plausibilisierung liegt, dient dies als Ausblick auf eine mögliche Realisierung des Anomalieerkennungssensors N-8 (Consistency) mittels Autoencoder. Wenn sich ein Signalverlauf durch einen Angriff oder eine Fehlfunktion verändert, dies aber keine Auswirkung auf davon abhängige Signale hat, müsste sich die Korrelation erkennbar verändern. Deshalb besteht der Eingangsvektor in diesem Schritt aus den normalisierten Werten von zwei abhängigen Signalen. Für ein 32-32-32 Netz werden dementsprechend die 16 letzten Werte des ersten, gefolgt von den 16 letzten Werten des zweiten Signals als Merkmale verwendet. Während des Trainings hat

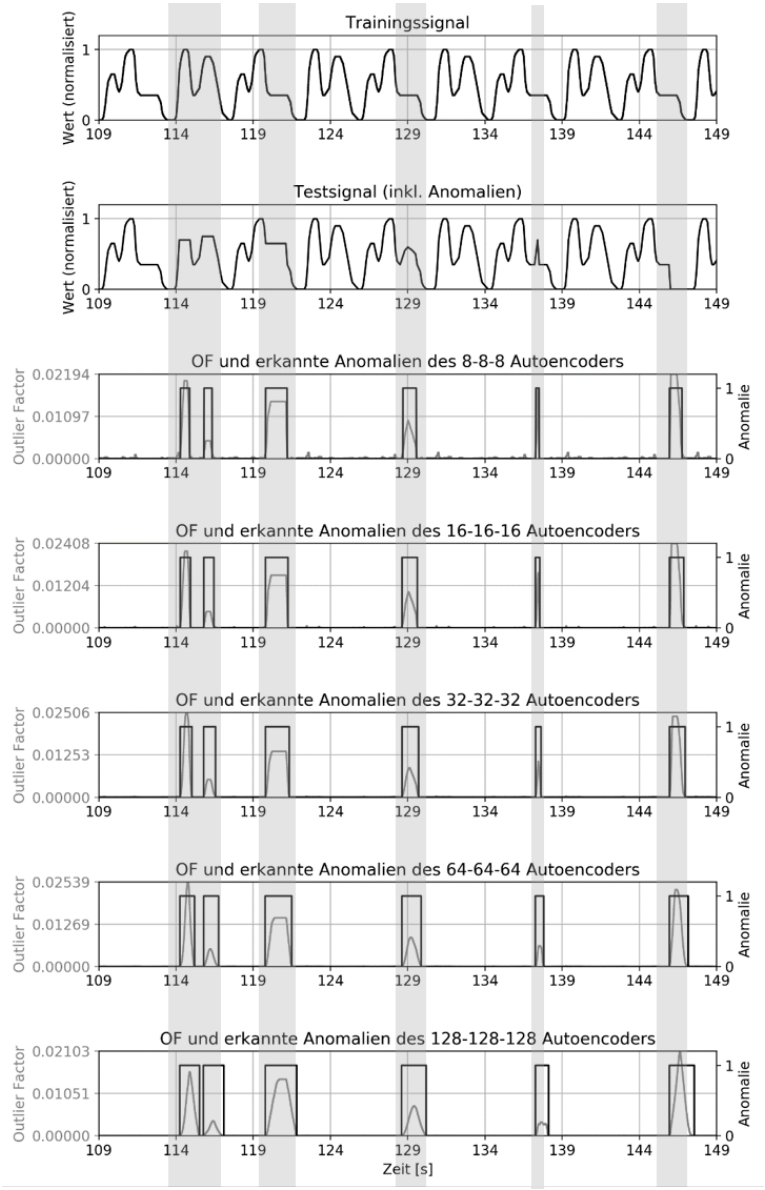

Bild A.14: Korrelation von Signalverläufen [VWeb+18b]

das erste Signal weiterhin den in Bild A.9 dargestellten Verlauf. Das zweite Signal ist davon abgeleitet und wurde gespiegelt sowie skaliert[6]. Bei der Inferenz wird das erste Signal mit den bereits gezeigten Anomalien angereichert (siehe Bild A.9) während das zweite Signal unverändert bleibt. Das vielversprechende Ergebnis ist in Bild A.14 dargestellt. Aus Übersichtlichkeitsgründen ist das zweite, unveränderte Signal nicht abgebildet.
Alle Anomalien werden erfolgreich erkannt, selbst von den kleinsten Netzen mit geringem Ressourcenverbrauch (8-8-8). In diesem Fall stellt auch der alternative Verlauf (dritte Veränderung) eine Anomalie dar, da sich das korrelierende Signal nicht entsprechend verhält. Zu beachten ist zudem der große Ausschlag des Outlier Factors während einer Anomalie, vergleichbar mit einem hohen Signal-Rauschabstand. Dies lässt auf eine sichere Erkennung schließen. Die letzte Auffälligkeit ist die Dauer der erkannten Anomalien. Je größer der betrachtete Signalausschnitt (größere Netze), desto länger die Dauer der gemeldeten Anomalien, da der Korrelationsfehler länger im betrachteten Ausschnitt enthalten ist.

A.4.6 Zusammenfassung der Machbarkeitsstudie

Die Machbarkeitsstudie hat gezeigt, dass die Plausibilisierung von Signalverläufen (Anomalieerkennungssensor N-7) mittels Sliding Window Ansatz und klassischen Autoencodern möglich ist. Bereits Netze mit 16-16-16 Topologie liefern gute Ergebnisse, siehe Bild A.10. Autoencoder mit mehr versteckten Schichten reagierten sensibler auf anormale Änderungen im Signalverlauf. Ein ähnliches Bild ergibt sich bei der Verwendung von erster und zweiter Ableitung als zusätzliche Merkmale im Eingangsvektor. Beides geht auf Kosten des Ressourcenverbrauchs, welcher bei einer Anwendung in automotive Steuergeräten eine entscheidende Rolle spielt. Im ersten Schritt ist es ausreichend eine Anomalie zu erkennen. Den genauen Zeitpunkt sowie die exakte Dauer spielen zunächst eine untergeordnete Rolle.

[6] Aufgrund der Normalisierung der Eingangsdaten hat die Skalierung keinen Effekt.

Daher erscheinen einfache Netztopologien vielversprechend für den praktischen Einsatz. Auf Basis simulierter Daten und wenigen synthetisierten Anomalien lässt sich jedoch noch keine allgemeingültige Aussage treffen. Obwohl diese unter dem Gesichtspunkt eines möglichen Geschwindigkeitssignals erstellt wurden, ist die Tragfähigkeit des Konzepts tiefergehend zu untersuchen.
Gleiches gilt für die Anomalieerkennung anhand korrelierender Signale. Hier sind die simulierten Daten noch weniger repräsentativ, da eine perfekte Korrelation in Realdaten selten vorkommt. Neben Signal- beziehungsweise Sensorrauschen, existieren außerdem Zusammenhänge über mehr als zwei Signale. Zum Beispiel besteht ein Bezug zwischen Fahrzeuggeschwindigkeit, Motordrehzahl, Gang und Kupplungsstatus. Dies sind Herausforderungen, die anhand von Realdaten weiter zu untersuchen sind.

A.5 Detaillierte Evaluierungsergebnisse

Als Ergänzung zu Unterkapitel 6.2 sind nachfolgend die Evaluierungsergebnisse auf Basis der Validierungsdaten für die untersuchten Autoencoder-Topologien aufgelistet. Die Diagramme zeigen die jeweils leistungsstärksten Autoencoder (höchste TPR bei keinem Falschalarm) einer Topologie, zusammengefasst in Gruppen mit der gleichen Anzahl an Ein- und Ausgangsneuronen. Zum jeweils leistungsstärksten Autoencoder einer Gruppe folgt eine detaillierte Aufschlüsselung der erkannten Anomalien in Tabellenform.

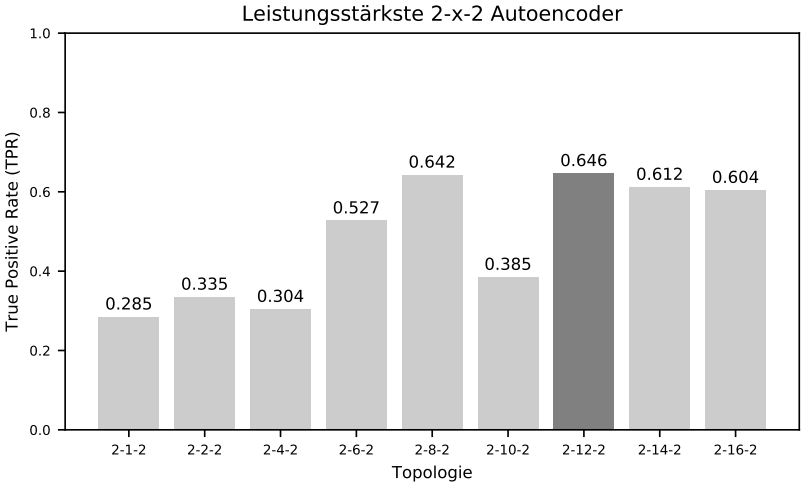

Bild A.15: Evaluierungsergebnisse für Autoencoder mit 2-x-2 Topologie

Anomalie-Typ	Plateau	Pos. Step Plateau	Neg. Step Plateau	Pos. Peak	Neg. Peak	Pos. Ramp Plateau	Neg. Ramp Plateau
TPR [abs.]	3/20	13/20	11/20	8/20	3/20	14/20	18/20
Anomalie-Typ	Noise	Sine	Pos. Step Offset	Neg. Step Offset	Pos. Ramp Offset	Neg. Ramp Offset	*Summe*
TPR [abs.]	9/20	18/20	16/20	19/20	18/20	18/20	*168/260*

Tabelle A.1: Erkannte Anomalien durch den leistungsstärksten 2-12-2 Autoencoder

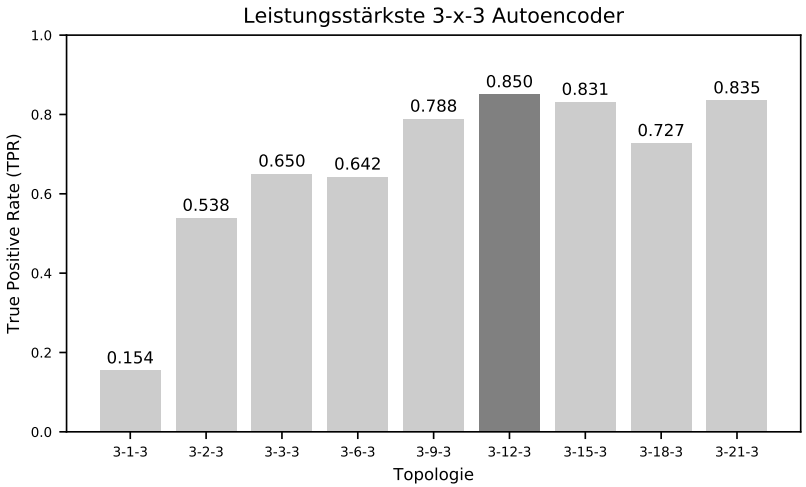

Bild A.16: Evaluierungsergebnisse für Autoencoder mit 3-x-3 Topologie

Anomalie-Typ	Plateau	Pos. Step Plateau	Neg. Step Plateau	Pos. Peak	Neg. Peak	Pos. Ramp Plateau	Neg. Ramp Plateau
TPR [abs.]	10/20	18/20	16/20	12/20	17/20	16/20	19/20
Anomalie-Typ	Noise	Sine	Pos. Step Offset	Neg. Step Offset	Pos. Ramp Offset	Neg. Ramp Offset	*Summe*
TPR [abs.]	17/20	19/20	20/20	20/20	18/20	19/20	*221/260*

Tabelle A.2: Erkannte Anomalien durch den leistungsstärksten 3-12-3 Autoencoder

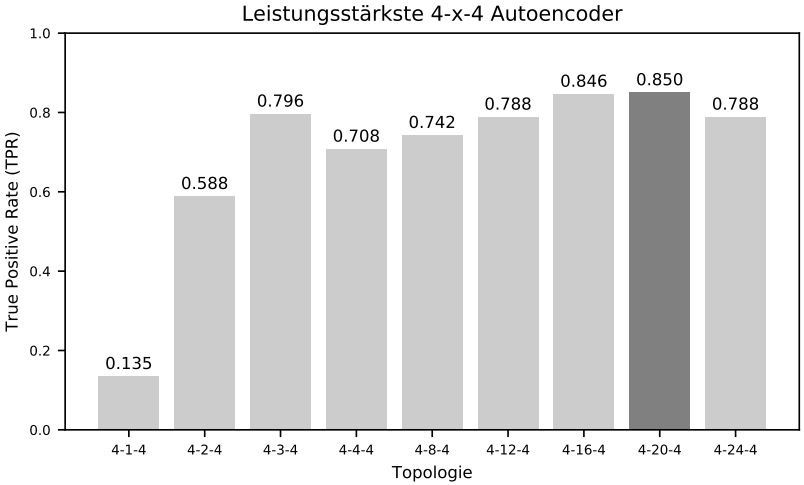

Bild A.17: Evaluierungsergebnisse für Autoencoder mit 4-x-4 Topologie

Anomalie-Typ	Plateau	Pos. Step Plateau	Neg. Step Plateau	Pos. Peak	Neg. Peak	Pos. Ramp Plateau	Neg. Ramp Plateau
TPR [abs.]	11/20	18/20	16/20	12/20	18/20	16/20	19/20
Anomalie-Typ	Noise	Sine	Pos. Step Offset	Neg. Step Offset	Pos. Ramp Offset	Neg. Ramp Offset	*Summe*
TPR [abs.]	16/20	19/20	20/20	20/20	17/20	19/20	*221/260*

Tabelle A.3: Erkannte Anomalien durch den leistungsstärksten 4-20-4 Autoencoder

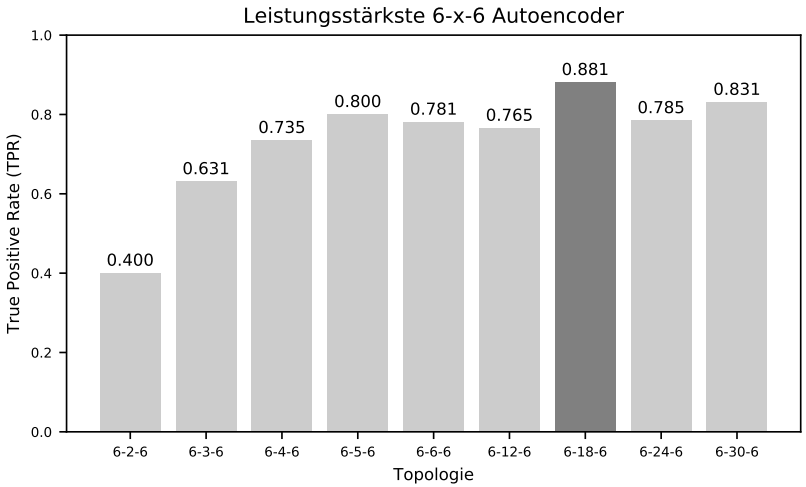

Bild A.18: Evaluierungsergebnisse für Autoencoder mit 6-x-6 Topologie

Anomalie-Typ	Plateau	Pos. Step Plateau	Neg. Step Plateau	Pos. Peak	Neg. Peak	Pos. Ramp Plateau	Neg. Ramp Plateau
TPR [abs.]	12/20	18/20	17/20	14/20	19/20	17/20	19/20
Anomalie-Typ	Noise	Sine	Pos. Step Offset	Neg. Step Offset	Pos. Ramp Offset	Neg. Ramp Offset	*Summe*
TPR [abs.]	16/20	20/20	20/20	20/20	18/20	19/20	*229/260*

Tabelle A.4: Erkannte Anomalien durch den leistungsstärksten 6-18-6 Autoencoder

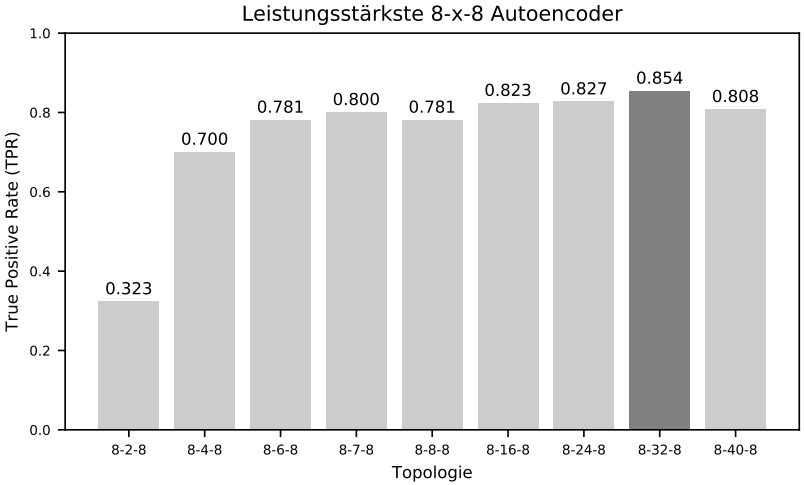

Bild A.19: Evaluierungsergebnisse für Autoencoder mit 8-x-8 Topologie

Anomalie-Typ	Plateau	Pos. Step Plateau	Neg. Step Plateau	Pos. Peak	Neg. Peak	Pos. Ramp Plateau	Neg. Ramp Plateau
TPR [abs.]	11/20	16/20	16/20	14/20	18/20	17/20	19/20
Anomalie-Typ	Noise	Sine	Pos. Step Offset	Neg. Step Offset	Pos. Ramp Offset	Neg. Ramp Offset	*Summe*
TPR [abs.]	16/20	20/20	20/20	20/20	17/20	18/20	*222/260*

Tabelle A.5: Erkannte Anomalien durch den leistungsstärksten 8-32-8 Autoencoder

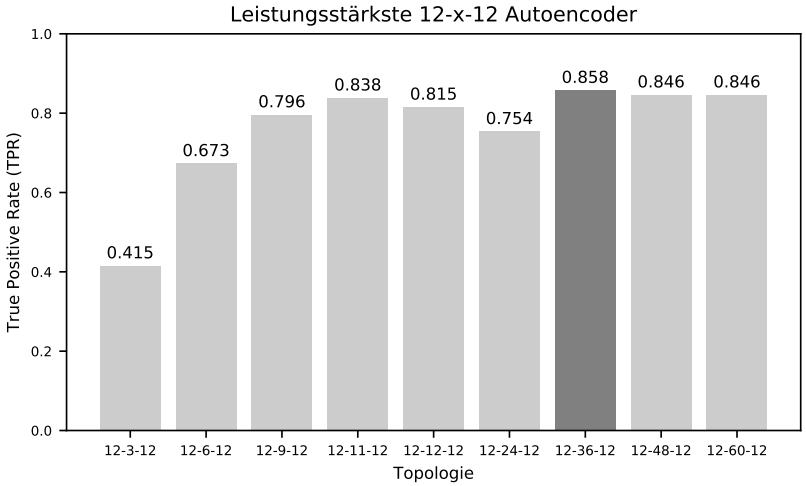

Bild A.20: Evaluierungsergebnisse für Autoencoder mit 12-x-12 Topologie

Anomalie-Typ	Plateau	Pos. Step Plateau	Neg. Step Plateau	Pos. Peak	Neg. Peak	Pos. Ramp Plateau	Neg. Ramp Plateau
TPR [abs.]	10/20	18/20	17/20	14/20	16/20	17/20	20/20
Anomalie-Typ	Noise	Sine	Pos. Step Offset	Neg. Step Offset	Pos. Ramp Offset	Neg. Ramp Offset	*Summe*
TPR [abs.]	15/20	20/20	19/20	20/20	17/20	20/20	*223/260*

Tabelle A.6: Erkannte Anomalien durch den leistungsstärksten 12-36-12 Autoencoder

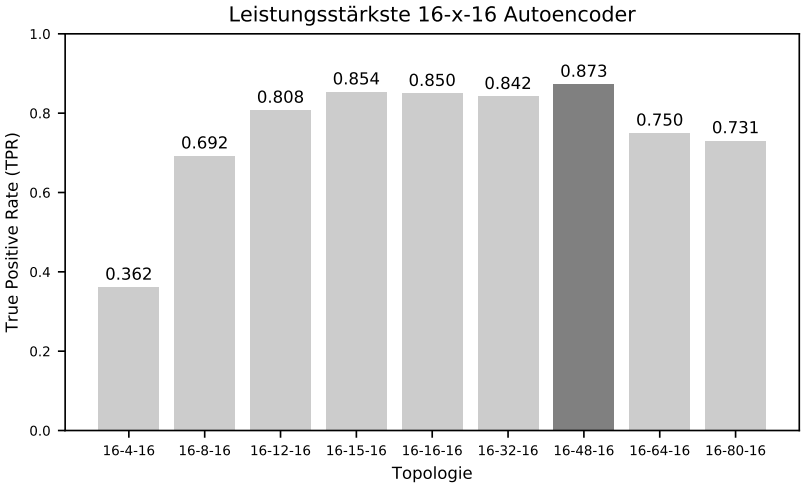

Bild A.21: Evaluierungsergebnisse für Autoencoder mit 16-x-16 Topologie

Anomalie-Typ	Plateau	Pos. Step Plateau	Neg. Step Plateau	Pos. Peak	Neg. Peak	Pos. Ramp Plateau	Neg. Ramp Plateau
TPR [abs.]	10/20	16/20	17/20	16/20	16/20	16/20	20/20
Anomalie-Typ	Noise	Sine	Pos. Step Offset	Neg. Step Offset	Pos. Ramp Offset	Neg. Ramp Offset	*Summe*
TPR [abs.]	19/20	20/20	19/20	20/20	18/20	20/20	*227/260*

Tabelle A.7: Erkannte Anomalien durch den leistungsstärksten 16-48-16 Autoencoder

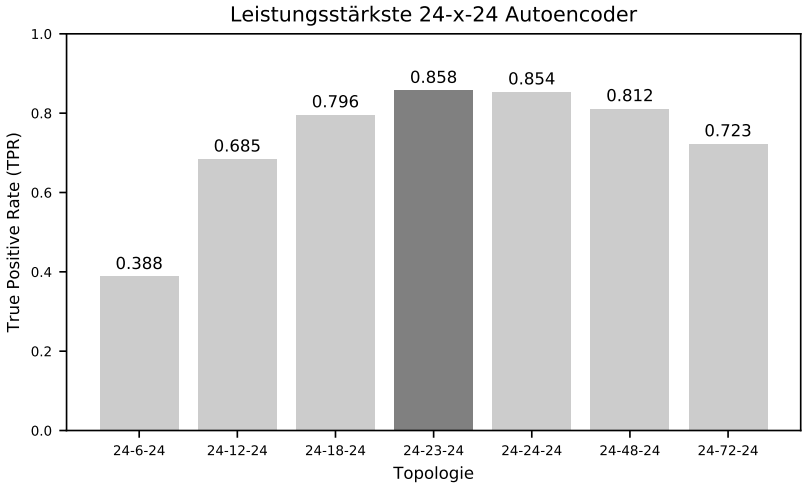

Bild A.22: Evaluierungsergebnisse für Autoencoder mit 24-x-24 Topologie

Anomalie-Typ	Plateau	Pos. Step Plateau	Neg. Step Plateau	Pos. Peak	Neg. Peak	Pos. Ramp Plateau	Neg. Ramp Plateau
TPR [abs.]	10/20	17/20	18/20	15/20	9/20	16/20	20/20
Anomalie-Typ	Noise	Sine	Pos. Step Offset	Neg. Step Offset	Pos. Ramp Offset	Neg. Ramp Offset	*Summe*
TPR [abs.]	20/20	20/20	20/20	20/20	18/20	20/20	*223/260*

Tabelle A.8: Erkannte Anomalien durch den leistungsstärksten 24-23-24 Autoencoder

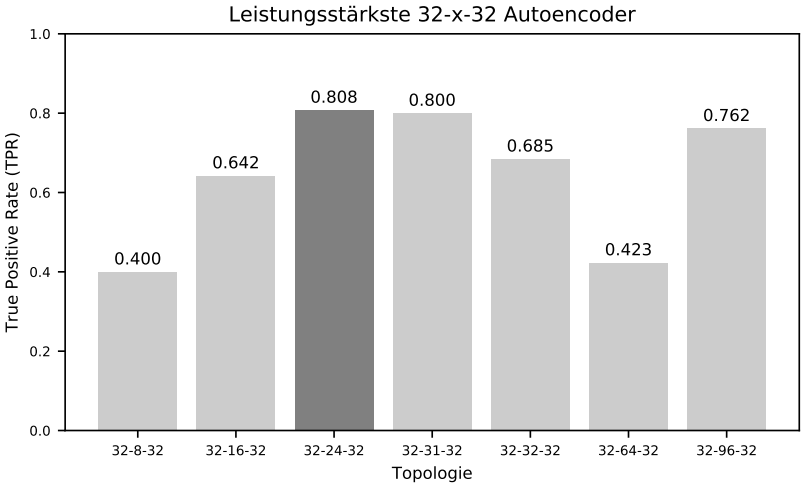

Bild A.23: Evaluierungsergebnisse für Autoencoder mit 32-x-32 Topologie

Anomalie-Typ	Plateau	Pos. Step Plateau	Neg. Step Plateau	Pos. Peak	Neg. Peak	Pos. Ramp Plateau	Neg. Ramp Plateau
TPR [abs.]	9/20	16/20	16/20	13/20	5/20	16/20	20/20
Anomalie-Typ	Noise	Sine	Pos. Step Offset	Neg. Step Offset	Pos. Ramp Offset	Neg. Ramp Offset	*Summe*
TPR [abs.]	19/20	20/20	18/20	20/20	18/20	20/20	*210/260*

Tabelle A.9: Erkannte Anomalien durch den leistungsstärksten 32-24-32 Autoencoder

A.6 Austauschformat für trainierte Autoencoder

Als Teil der Anomaly Detection Configuration wurde ein Austauschformat für trainierte Autoencoder spezifiziert. Es basiert auf JSON und unterstützt beliebige, vollständig-verbundene, feed-forward Perceptrons.

```
{
    "spec_version": "5",
    "feedforward": {
        "layers": [
            [
                "relu",
                [
                    0.0002830855664797127,
                    0.7319703102111816,
                    0.7320597171783447
                ]
            ],
            [
                "relu",
                [
                    0.0007007931126281619,
                    0.6831615567207336
                ],
                [
                    -0.001036616275086999,
                    0.6831640601158142
                ]
            ]
        ]
    }
}
```

Bild A.24: Austauschformat für trainierte Autoencoder; Beispiel für eine 2-1-2 Topologie

Bild A.24 zeigt beispielhaft eine Definition für einen 2-1-2 Autoencoder. Das Wurzelobjekt besitzt zwei Elemente: Einen String für die Version der Spezifikation (*spec_version*) und ein weiteres Objekt (*feedforward*). Letzteres enthält ein Array (*layers*), wobei jedes Element an sich wiederum ein

Array[7] ist und jeweils eine Schicht des modellierten neuronalen Netzes repräsentiert. Die Definition jeder Schicht beginnt mit einem String für die verwendete Aktivierungsfunktion (im Beispiel „relu“) gefolgt von (potentiell) mehreren Arrays, wovon jedes ein Neuron abbildet und die Gewichte für dessen Eingänge beinhaltet. Die versteckte Schicht des in Bild A.24 dargestellten 2-1-2 Autoencoders besitzt ein Neuron mit drei Eingängen: Der Offset und die zwei Eingangssignale. Analog dazu ist die Ausgangsschicht mit zwei Neuronen modelliert.

[7] Arrays in JSON können Elemente unterschiedlichen Typs beinhalten.

Abbildungsverzeichnis

Tabellenverzeichnis

Formelverzeichnis

Literaturverzeichnis

[SAE18] SAE International, Hrsg. *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*. 15. Juni 2018. URL: https://www.sae.org/standards/content/j3016_201806/.

[ps17] pwc und strategy&, Hrsg. *The 2017 Strategy& Digital Auto Report. Fast and furious: Why making money in the "roboconomy" is getting harder*. September 2017. URL: https://www.strategyand.pwc.com/media/file/2017-Strategyand-Digital-Auto-Report.pdf (besucht am 04. 03. 2018).

[Cou15] Council of the European Union , European Parliament. *Regulation (EU) 2015/758 of the European Parliament and of the Council of 29 April 2015 concerning type-approval requirements for the deployment of the eCall in-vehicle system based on the 112 service and amending Directive 2007/46/EC*. (EU) 2015/758. 29. Apr. 2015.

[MV15] Charlie Miller und Chris Valasek. *Remote exploitation of an unaltered passenger vehicle*. DEF CON 23, August 10, 2015.

[MV14] Charlie Miller und Chris Valasek. *A survey of remote automotive attack surfaces*. DEF CON 22, 2014.

[MDQ15] David McCandless, Pearl Doughty-White und Miriam Quick. *Codebases. Millions of lines of code*. Hrsg. von information is beautiful. 2015. URL: https://informationisbeautiful.net/visualizations/million-lines-of-code/.

[Mco17] Ross Mcouat. *Cars are made of code*. Hrsg. von NXP. NXP. 2017. URL: https://blog.nxp.com/automotive/cars-are-made-of-code (besucht am 24. 09. 2018).

[Kos+10] Karl Koscher u. a. „Experimental security analysis of a modern automobile“. In: *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE. 2010, S. 447–462.

[Che+11] Stephen Checkoway u. a. „Comprehensive Experimental Analyses of Automotive Attack Surfaces“. In: *USENIX Security Symposium*. San Francisco. 2011.

[MV13] Charlie Miller und Chris Valasek. *Adventures in automotive networks and control units*. DEF CON 21, 2013.

[Gre16] Andy Greenberg. *Hackers Hijack a Big Rig Truck's Accelerator and Brakes*. Hrsg. von wired. 2016. URL: https://www.wired.com/2016/08/researchers-hack-big-rig-truck-hijack-accelerator-brakes/ (besucht am 24. 09. 2018).

[Arg14] Argus Cyber Security. *A remote attack on an aftermarket telematics service*. 2014. URL: https://argus-sec.com/remote-attack-aftermarket-telematics-service/ (besucht am 15. 05. 2017).

[Gre15] Andy Greenberg. *Hackers Cut a Corvette's Brakes via a Common Car Gadget*. Hrsg. von wired. 2015. URL: https://www.wired.com/2015/08/hackers-cut-corvettes-brakes-via-common-car-gadget/ (besucht am 24. 09. 2018).

[Kov17] Alexei Kovelman. *A Remote Attack on the Bosch Drivelog Connector Dongle*. Hrsg. von Argus Cyber Security. 2017. URL: https://argus-sec.com/remote-attack-bosch-drivelog-connector-dongle/ (besucht am 15. 05. 2017).

[Kee16] Keen Security Lab of Tencent. *Car Hacking Research: Remote Attack Tesla Motors*. 2016. URL: https://keenlab.tencent.com/en/2016/09/19/Keen-Security-Lab-of-Tencent-Car-Hacking-Research-Remote-Attack-to-Tesla-Cars/ (besucht am 24. 09. 2018).

[Kee27] Keen Security Lab of Tencent. *New Car Hacking Research: 2017, Remote Attack Tesla Motors Again*. 2017-07-27. URL: https://keenlab.tencent.com/en/2017/07/27/New-Car-Hacking-Research-2017-Remote-Attack-Tesla-Motors-Again/ (besucht am 24. 09. 2018).

[Ten18a] Tencent Keen Security Lab. *New Vehicle Security Research by KeenLab: Experimental Security Assessment of BMW Cars*. 2018. URL: https://keenlab.tencent.com/en/2018/05/22/New-CarHacking-Research-by-KeenLab-Experimental-Security-Assessment-of-BMW-Cars/ (besucht am 24. 09. 2018).

[Mag17] Federico Maggi. *A Vulnerability in Modern Automotive Standards and How We Exploited It*. Technical Brief. Hrsg. von TrendLabs. August 2017. URL: https://documents.trendmicro.com/assets/A-Vulnerability-in-Modern-Automotive-Standards-and-How-We-Exploited-It.pdf.

[ZP20] Michael Ziehensack und Roman Pallierer. *Secure Automotive Ethernet for automated driving. Multi-level security architecture*. EB TechPaper. Hrsg. von Elektrobit Automotive GmbH. 2016-04-20.

[EM16] Christof Ebert und Eduard Metzker. „Risiko-orientierte Methodik. Praxis-Erfahrungen zur Anwendung von Cyber Security“. In: *Elektronik automotive* (Sonderausgabe Software 2016 2016).

[HKD11] Tobias Hoppe, Stefan Kiltz und Jana Dittmann. „Security threats to automotive CAN networks—Practical examples and selected short-term countermeasures“. In: *Reliability Engineering & System Safety* 96 (1 2011), S. 11–25. ISSN: 09518320. DOI: 10.1016/j.ress.2010.06.026.

[Sax16] Eric Sax. *Testing 2020. "Just when I knew all the answers they changed the questions"*. Unter Mitarb. von Vector Informatik GmbH. Stuttgart, 3. Mai 2016.

[Daia] Daimler AG. *Carl Benz. Gründer und Wegbereiter*. URL: https://www.daimler.com/konzern/tradition/gruender-wegbereiter/carl-benz.html.

[Daib] Daimler AG. *Unternehmensgeschichte. Benz Patent-Motorwagen: Das erste Automobil (1885–1886)*. URL:

https://www.daimler.com/konzern/tradition/geschichte/1885-1886.html.

[Rob] Robert Bosch GmbH. *Unternehmensgeschichte*. URL: http://www.bosch.de/de/de/our_company_1/history_1/history.html.

[Sch05] Peter Scholz. *Softwareentwicklung eingebetteter Systeme. Grundlagen, Modellierung, Qualitätssicherung*. ger. Xpert.press. Berlin: Springer, 2005. 232 S. ISBN: 9783540234050. URL: http://site.ebrary.com/lib/alltitles/docDetail.action?docID=10182931.

[Sch13] Florian Schäffer. *OBD. Fahrzeugdiagnose in der Praxis*. Elektronik. Haar: Franzis Verlag, 2013. Online-Ressource. ISBN: 3645270264.

[ZS14] Werner Zimmermann und Ralf Schmidgall. *Bussysteme in der Fahrzeugtechnik. Protokolle, Standards und Softwarearchitektur*. ger. 5., aktual. und erw. Aufl. ATZ / MTZ-Fachbuch. Schmidgall, Ralf (VerfasserIn). Wiesbaden: Springer Vieweg, 2014. 507 S. DOI: 10.1007/978-3-658-02419-2.

[IEEa] IEEE Standards Association, Hrsg. *IEEE Standard for Ethernet Amendment 1: Physical Layer Specifications and Management Parameters for 100 Mb/s Operation over a Single Balanced Twisted Pair Cable (100BASE-T1)*. New York, NY, USA: IEEE. DOI: 10.1109/IEESTD.2016.7433918.

[IEEb] IEEE Standards Association, Hrsg. *IEEE Standard for Ethernet Amendment 4: Physical Layer Specifications and Management Parameters for 1 Gb/s Operation*

over a Single Twisted-Pair Copper Cable. New York, NY, USA: IEEE. DOI: 10.1109/IEEESTD.2016.7564011.

[IEEc] IEEE Standards Association. *P802.3ch - Standard for Ethernet Physical Layer Specifications and Management Parameters for Greater Than 1 Gb/s Automotive Ethernet*. URL: https://standards.ieee.org/project/802_3ch.html (besucht am 17. 10. 2018).

[IEEd] IEEE Standards Association. *P802.3cg - Standard for Ethernet Amendment. Physical Layer Specifications and Management Parameters for 10 Mb/s Operation and Associated Power Delivery over a Single Balanced Pair of Conductors*. URL: https://standards.ieee.org/project/802_3cg.html (besucht am 17. 10. 2018).

[Int94a] International Telecommunication Union. *ITU-T Recommendations. ITU-T X.200 (07/1994)*. 1994. URL: http://www.itu.int/ITU-T/recommendations/rec.aspx?rec=2820.

[Inta] International Telecommunication Union, Hrsg. *Information technology - Open Systems Interconnection - Basic reference model: The basic model*. ITU-T Recommendation. Version 07/94.

[Int94b] International Organization for Standardization, Hrsg. *Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*. 1. Nov. 1994. URL: https://www.iso.org/standard/20269.html.

[Int16a] International Organization for Standardization, Hrsg. *Road vehicles – Controller area network (CAN) – Part 2: High-speed medium access unit*. 1. Dez. 2016. URL: https://www.iso.org/standard/67244.html.

[Int06] International Organization for Standardization, Hrsg. *Road vehicles – Controller area network (CAN) – Part 3: Low-speed, fault-tolerant, medium-dependent interface*. 1. Juni 2006. URL: https://www.iso.org/standard/36055.html.

[Int15] International Organization for Standardization, Hrsg. *Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signalling*. 1. Dez. 2015. URL: https://www.iso.org/standard/63648.html.

[Lin12] Thomas Lindenkreuz. *CAN FD - CAN with Flexible Data Rate*. Stuttgart, 2012.

[Int16b] International Organization for Standardization, Hrsg. *Road vehicles – Diagnostic communication over Controller Area Network (DoCAN) – Part 2: Transport protocol and network layer services*. 1. Apr. 2016. URL: https://www.iso.org/standard/66574.html.

[Int13a] International Organization for Standardization, Hrsg. *Road vehicles – Unified diagnostic services (UDS) – Part 1: Specification and requirements*. 1. März 2013. URL: https://www.iso.org/standard/55283.html.

[Int13b] International Organization for Standardization, Hrsg. *Road vehicles – Unified diagnostic services (UDS) –*

Part 2: Session layer services. 1. Feb. 2013. URL: https://www.iso.org/standard/45763.html.

[Int12a] International Organization for Standardization, Hrsg. *Road vehicles – Unified diagnostic services (UDS) – Part 3: Unified diagnostic services on CAN implementation (UDSonCAN)*. 1. Dez. 2012. URL: https://www.iso.org/standard/55284.html.

[Int05a] International Organization for Standardization, Hrsg. *Road vehicles – Open interface for embedded automotive applications – Part 3: OSEK/VDX Operating System (OS)*. 1. Nov. 2005. URL: https://www.iso.org/standard/40079.html.

[AUT] AUTOSAR Web Team. *AUTOSAR Introduction*. December 2017. Hrsg. von AUTOSAR. URL: https://www.autosar.org/fileadmin/HOW_TO_JOIN/AUTOSAR_Introduction.pdf (besucht am 05. 01. 2018).

[IEE17] IEEE Standards Association, Hrsg. *IEEE Standard for Information Technology: Portable Operating System Interface (POSIX(R)) Base Specifications, Issue 7*. 6. Dez. 2017. URL: https://standards.ieee.org/standard/1003_1-2017.html.

[AUT17a] AUTOSAR, Hrsg. *Virtual Functional Bus*. Version AUTOSAR CP Release 4.3.1. 8. Dez. 2017.

[Tur50] A. M. Turing. „Computing Machinery and Intelligence“. In: *Mind* LIX (236 1950), S. 433–460. ISSN: 0026-4423. DOI: 10.1093/mind/LIX.236.433.

[Cop93] J. Copeland. *Artificial Intelligence. A Philosophical Introduction*. Wiley, 1993. ISBN: 9780631183853. URL:

https://books.google.de/books?id=PVDNf9MUygIC.

[Blo09] Marco Block-Berlitz. *Vorlesung: "Künstliche Intelligenz"*. Freie Universität Berlin, 2009.

[McC07] John McCarthy. *Basic Questions*. 12.11.2007. URL: http://www-formal.stanford.edu/jmc/whatisai/node1.html (besucht am 05. 03. 2017).

[Sea80] John R. Searle. „Minds, brains, and programs“. In: *Behavioral and Brain Sciences* 3 (03 1980), S. 417. DOI: 10.1017/S0140525X00005756.

[IBM03] IBM. *IBM Archives: 1956*. en-US. 23.01.2003. URL: http://www-03.ibm.com/ibm/history/history/year_1956.html (besucht am 26. 06. 2017).

[Sam59] Arthur L. Samuel. „Some Studies in Machine Learning Using the Game of Checkers“. In: *IBM Journal of Research and Development* 3 (3 1959), S. 210–229. DOI: 10.1147/rd.33.0210.

[Mit97] Tom M. Mitchell. *Machine Learning*. McGraw-Hill series in computer science. New York: McGraw-Hill, 1997. xvii, 414.

[CBK09] Varun Chandola, Arindam Banerjee und Vipin Kumar. „Anomaly detection: A survey“. In: *ACM Computing Surveys* 41 (3 2009), S. 1–58. DOI: 10.1145/1541880.1541882.

[FPS96] Usama Fayyad, Gregory Piatetsky-Shapiro und Padhraic Smyth. „From Data Mining to Knowledge Discovery in Databases“. In: *AI Magazine* 17 (3 1996), S. 37–54.

[Kri05] David Kriesel. *Ein kleiner Überblick über Neuronale Netze*. 2005. URL: http://www.dkriesel.com/science/neural_networks (besucht am 18. 04. 2017).

[MP43] Warren S. McCulloch und Walter Pitts. „A logical calculus of the ideas immanent in nervous activity“. In: *The Bulletin of Mathematical Biophysics* 5 (4 1943), S. 115–133. DOI: 10.1007/BF02478259.

[Ros58] Frank Rosenblatt. „The perceptron. A probabilistic model for information storage and organization in the brain“. In: *Psychological Review* 65 (6 1958), S. 386–408. ISSN: 0033-295X. DOI: 10.1037/h0042519.

[Heb49] Donald Olding Hebb. *The Organization of Behavior. A Neuropsychological Theory*. New York und London: John Wiley & Sons, Inc. und Chapman & Hall, Limited, 1949. 365 S.

[Sel98] Stefan Selle. „Einsatz Künstlicher Neuronaler Netze auf dem Aktienmarkt“. Wirtschaftswissenschaftliche Fakultät. Diplomarbeit im Studiengang Volkswirtschaftslehre. Heidelberg: Ruprecht-Karls-Universität, 1998. 125 S.

[Ros62] Frank Rosenblatt. *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms*. Washington: Spartan Books, 1962.

[MP72] Marvin L. Minsky und Seymour A. Papert. *Perceptrons. An introduction to computational geometry*. eng. 2. print. with corr. Papert, Seymour A. (VerfasserIn). Cambridge/Mass.: The MIT Press, 1972. 258 S. ISBN: 0262630222.

[Hil95] Wolfgang Hilberg. „Karl Steinbuch, ein zu Unrecht vergessener Pionier der künstlichen neuronalen Systeme“. Mitteilung aus dem Institut für Datentechnik der Technischen Hochschule Darmstadt. In: *Frequenz* (49 1995), S. 28–37.

[Uth01] Anne-Dore Uthe. *Symbolische KI. Lexikon der Kartographie und Geomatik*. Hrsg. von Spektrum Akademischer Verlag. 2001. URL: http://www.spektrum.de/lexikon/kartographie-geomatik/symbolische-ki/4780.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton und Ronald J. Williams. „Learning Internal Representations by Error Propagation“. In: *Parallel distributed processing. Explorations in the microstructure of cognition*. Hrsg. von David E. Rumelhart und James L. McClelland. A Bradford book. Cambridge, Mass.: MIT Press, 1986, S. 318–362. ISBN: 0262631121.

[Wer74] Paul Werbos. „Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences“. PhD thesis. Cambridge, Mass.: Harvard University, 1974. 454 S.

[Wer82] Paul J. Werbos. „Applications of advances in nonlinear sensitivity analysis“. In: *System Modeling and Optimization. Proceedings of the 10th IFIP Conference New York City, USA, August 31 - September 4, 1981*. Hrsg. von R. F. Drenick und F. Kozin. Bd. 38. Lecture Notes in Control and Information Sciences 38. Berlin und Heidelberg: Springer, 1982, S. 762–770. ISBN: 3-540-11691-5. DOI: 10.1007/BFb0006203.

[RM86] David E. Rumelhart und James L. McClelland, Hrsg. *Parallel distributed processing. Explorations in the microstructure of cognition*. eng. A Bradford book. Cambridge, Mass.: MIT Press, 1986. ISBN: 0262631121.

[Haw80] D. M. Hawkins. *Identification of Outliers*. eng. Monographs on Applied Probability and Statistics. Dordrecht: Springer, 1980. 188 S. ISBN: 978-94-015-3994-4. DOI: 10.1007/978-94-015-3994-4.

[Son+07] Xiuyao Song u. a. „Conditional Anomaly Detection“. In: *IEEE Transactions on Knowledge and Data Engineering* 19 (5 2007), S. 631–645. DOI: 10.1109/TKDE.2007.1009.

[AHS85] David H. Ackley, Geoffrey E. Hinton und Terrence J. Sejnowski. „A Learning Algorithm for Boltzmann Machines“. In: *Cognitive Science* 9 (1 1985), S. 147–169. ISSN: 03640213. DOI: 10.1016/S0364-0213(85)80012-4.

[CMZ89] Garrison W. Cottrell, Paul Munro und David Zipser. „Image Compression by Back Propagation: An Example of Extensional Programming“. In: *Models of Cognition. A Review of Cognitive Science*. Volume One. Hrsg. von Noel E. Sharkey. Norwood, NJ: Ablex, 1989, S. 208–240. ISBN: 0-89391-528-9.

[Haw+02] Simon Hawkins u. a. „Outlier Detection Using Replicator Neural Networks“. In: *Data Warehousing and Knowledge Discovery. 4th International Conference, DaWaK 2002 Aix-en-Provence, France, September 4-6, 2002 Proceedings*. Hrsg. von Yahiko Kambayashi, Masatoshi Arikawa und Werner Winiwarter. Lecture

Notes in Computer Science 2454. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg, 2002.

[Wil+02] Graham Williams u. a. „A Comparative Study of RNN for Outlier Detection in Data Mining“. In: *Proceedings / 2002 IEEE International Conference on Data Mining, ICDM 2002. 9 - 12 December 2002, Maebashi City, Japan*. Hrsg. von Vipin Kumar. Los Alamitos, Calif.: IEEE Computer Society, 2002, S. 709–712.

[DCS14] Hoang Anh Dau, Vic Ciesielski und Andy Song. „Anomaly Detection Using Replicator Neural Networks Trained on Examples of One Class“. In: *Simulated evolution and learning. 10th International Conference, SEAL 2014, Dunedin, New Zealand, December 15-18, 2014 ; proceedings*. Hrsg. von Grant Dick. Bd. 8886. Lecture Notes in Computer Science 8886. Cham: Springer, 2014, S. 311–322. ISBN: 978-3-319-13562-5. DOI: 10.1007/978-3-319-13563-2_27.

[TG04] László Tóth und Gábor Gosztolya. „Replicator Neural Networks for Outlier Modeling in Segmental Speech Recognition“. In: *Advances in Neural Networks - ISNN 2004. Proceedings, Part I.* International Symposium on Neural Networks (Dalian, China). Hrsg. von Fu-Liang Yin, Jun Wang und Chengan Guo. Bd. 3173. Lecture Notes in Computer Science 3173. Berlin und Heidelberg: Springer, 2004, S. 996–1001. ISBN: 978-3-540-22841-7. DOI: 10.1007/978-3-540-28647-9_164.

[HS06] G. E. Hinton und R. R. Salakhutdinov. „Reducing the Dimensionality of Data with Neural Networks“. In:

Science 313 (5786 2006), S. 504–507. DOI: 10.1126/science.1127647.

[Ng] Andrew Ng. „Sparse Autoencoder“. CS294A Lecture notes.

[Rif+11] Salah Rifai u. a. „Contractive Auto-Encoders: Explicit Invariance During Feature Extraction“. In: *Proceedings of the Twenty-Eighth International Conference on Machine Learning*. 28th International Conference on Machine Learning (Bellevue, WA, USA). Hrsg. von Lise Getoor. International Machine Learning Society u. a. Madison, 2011, S. 833–840. ISBN: 978-1-4503-0619-5.

[Pu+16] Yunchen Pu u. a. „Variational Autoencoder for Deep Learning of Images, Labels and Captions“. In: *Advances in Neural Information Processing Systems 29*. Neural Information Processing Systems 2016 (Barcelona, Spain). Hrsg. von D. D. Lee u. a. Curran Associates, Inc, 2016, S. 2352–2360. URL: http://papers.nips.cc/paper/6528-variational-autoencoder-for-deep-learning-of-images-labels-and-captions.pdf.

[Vin+10] Pascal Vincent u. a. „Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion“. In: *Journal of Machine Learning Research* 11 (2010), S. 3371–3408.

[Mas+11] Jonathan Masci u. a. „Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction“. In: *Artificial neural networks and machine learning - ICANN 2011*. 21st International Conference on Artificial Neural Networks (Espoo, Finland). Hrsg. von Timo Honkela

u. a. Lecture Notes in Computer Science 6791. ICANN und International Conference on Artificial Neural Networks. Berlin: Springer, 2011, S. 52–59. ISBN: 978-3-642-21734-0. DOI: 10.1007/978-3-642-21735-7_7.

[LBE15] Zachary C. Lipton, John Berkowitz und Charles Elkan. *A Critical Review of Recurrent Neural Networks for Sequence Learning*. 2015. URL: https://arxiv.org/abs/1506.00019v4 (besucht am 20. 08. 2018).

[HS97] Sepp Hochreiter und Jürgen Schmidhuber. „Long Short-Term Memory“. In: *Neural Computation* 9 (8 1997), S. 1735–1780.

[Bea15] Françoise Beaufays. *The neural networks behind Google Voice transcription*. Hrsg. von Google AI Blog. Google. 2015. URL: https://ai.googleblog.com/2015/08/the-neural-networks-behind-google-voice.html.

[SSB14] Haşim Sak, Andrew Senior und Françoise Beaufays. *Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition*. Google. 2014. URL: https://arxiv.org/abs/1402.1128v1 (besucht am 20. 08. 2018).

[Wu+16] Yonghui Wu u. a. *Google's Neural Machine Translation System. Bridging the Gap between Human and Machine Translation*. Google. 2016. URL: https://arxiv.org/abs/1609.08144v2 (besucht am 20. 08. 2018).

[Kha16] Pranav Khaitan. *Chat Smarter with Allo*. Hrsg. von Google AI Blog. Google. 2016. URL: https://ai.

googleblog.com/2016/05/chat-smarter-with-allo.html.

[Met18] Angeliki Metallinou. *Amazon Scientists Use Transfer Learning to Accelerate Development of New Alexa Capabilities*. Hrsg. von Alexa Blogs. Amazon. 2018. URL: https://developer.amazon.com/de/blogs/alexa/post/b6187c9a-2faa-4c17-a90c-142ab0e48b7e/amazon-scientists-use-transfer-learning-to-accelerate-development-of-new-alexa-capabilities.

[Xio+17] W. Xiong u. a. *The Microsoft 2017 Conversational Speech Recognition System*. Microsoft AI and Research. 2017. URL: https://arxiv.org/abs/1708.06073v2 (besucht am 20. 08. 2018).

[Ami+17] Shahin Amiriparian u. a. *Sequence to Sequence Autoencoders for Unsupervised Representation Learning from Audio*. Munich, Germany: Detection and Classification of Acoustic Scenes and Events 2017, 16 November 2017.

[Hsu17] Daniel Hsu. *Time Series Forecasting Based on Augmented Long Short-Term Memory*. 2017. URL: https://arxiv.org/abs/1707.00666v2 (besucht am 20. 08. 2018).

[Mal+15] Pankaj Malhotra u. a. „Long Short Term Memory Networks for Anomaly Detection in Time Series“. In: *Proceedings / 23rd European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2015*. 23rd European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (Bruges, Belgium).

Hrsg. von Michel Verleysen. Université catholique de Louvain u. a. Louvain-la-Neuve: Ciaco, 2015, S. 89–94. ISBN: 978-287587014-8.

[Mal+16] Pankaj Malhotra u. a. *LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection*. Accepted at ICML 2016 Anomaly Detection Workshop, New York, NY, USA, 2016. Reference update in this version (v2). 2016. URL: https://arxiv.org/abs/1607.00148v2 (besucht am 20. 08. 2018).

[Bon+17] Lo¨ıc Bontemps u. a. *Collective Anomaly Detection based on Long Short Term Memory Recurrent Neural Network*. 2017. URL: https://arxiv.org/abs/1703.09752v1 (besucht am 20. 08. 2018).

[Sin17] Akash Singh. „Anomaly Detection for Temporal Data using Long Short-Term Memory (LSTM)“. School of Information and Communication Technology. Master's Thesis. Stockholm, Sweden: KTH Royal Institute of Technology, 2017. 61 S.

[Du+17] Min Du u. a. „DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning“. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS'17*. 2017 ACM SIGSAC Conference on Computer and Communications Security (Dallas, TX, USA). Hrsg. von Bhavani Thuraisingham. Association for Computing Machinery, ACM Conference on Computer and Communications Security und CCS. New York, NY: Association for Computing Machinery, 2017, S. 1285–1298. ISBN: 978-1-4503-4946-8. DOI: 10.1145/3133956.3134015.

[Mar+15] Erik Marchi u. a. „A novel approach for automatic acoustic novelty detection using a denoising autoencoder with bidirectional LSTM neural networks“. In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (Brisbane, Queensland, Australia). Institute of Electrical and Electronics Engineers u. a. Piscataway, NJ: IEEE, 2015, S. 1996–2000. ISBN: 978-1-4673-6997-8. DOI: 10.1109/ICASSP.2015.7178320.

[Wol18a] Maxim Wolpher. „Anomaly Detection in Unstructured Time Series Data using an LSTM Autoencoder“. School of Electrical Engineering and Computer Science. Master's Thesis. Stockholm, Sweden: KTH Royal Institute of Technology, 2018. 69 S.

[Gro18] Aarish Grover. „Anomaly Detection for Application Log Data“. Master's Projects. San Jose: San Jose State University, Spring 2018. 49 S.

[LTZ08] Fei Tony Liu, Kai Ming Ting und Zhi-Hua Zhou. „Isolation Forest“. In: *Eighth IEEE International Conference on Data Mining, 2008. ICDM '08 ; Pisa, Italy, 15 - 19 Dec. 2008*. IEEE International Conference on Data Mining und ICDM. Piscataway, NJ und Piscataway, NJ: IEEE, 2008, S. 413–422. ISBN: 978-0-7695-3502-9. DOI: 10.1109/ICDM.2008.17.

[LTZ12] Fei Tony Liu, Kai Ming Ting und Zhi-Hua Zhou. „Isolation-Based Anomaly Detection“. In: *ACM Transactions on Knowledge Discovery from Data* 6 (1 2012), S. 1–39. DOI: 10.1145/2133360.2133363.

[DF13] Zhiguo Ding und Minrui Fei. „An Anomaly Detection Approach Based on Isolation Forest Algorithm for Streaming Data using Sliding Window“. In: *IFAC Proceedings Volumes* 46 (20 2013), S. 12–17. ISSN: 14746670. DOI: 10.3182/20130902-3-CN-3020.00044.

[Sun+16] Li Sun u. a. *Detecting Anomalous User Behavior Using an Extended Isolation Forest Algorithm. An Enterprise Case Study*. 2016. URL: http://arxiv.org/pdf/1609.06676.

[HS18] Julia Hofmockel und Eric Sax. „Isolation Forest for Anomaly Detection in Raw Vehicle Sensor Data“. In: *VEHITS 2018. Proceedings of the 4th International Conference on Vehicle Technology and Intelligent Transport Systems : Funchal, Madeira, Portugal, March 16-18, 2018*. 4th International Conference on Vehicle Technology and Intelligent Transport Systems (Funchal, Madeira, Portugal). Hrsg. von Markus Helfert und Oleg Gusikhin. Institute for Systems and Technologies of Information, Control and Communication, International Conference on Vehicle Technology and Intelligent Transport Systems und VEHITS. Setúbal, Portugal: SCITEPRESS - Science and Technology Publications Lda, 2018, S. 411–416. ISBN: 978-989-758-293-6.

[TTL11] Swee Chuan Tan, Kai Ming Ting und Tony Fei Liu. „Fast Anomaly Detection for Streaming Data“. In: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*. Hrsg. von Toby Walsh. International Joint Conference on Artificial In-

telligence. Menlo Park, California: AAAI Press, 2011, S. 1511–1516. ISBN: 978-1-57735-516-8.

[Pev16] Tomáš Pevný. „Loda: Lightweight on-line detector of anomalies“. In: *Machine Learning* 102 (2 2016), S. 275–304. DOI: 10.1007/s10994-015-5521-0.

[JL84] William B. Johnson und Joram Lindenstrauss. „Extensions of Lipschitz mappings into a Hilbert space“. In: *Conference on Modern Analysis and Probability*. Hrsg. von Richard Beals u. a. Bd. 26. Contemporary Mathematics. Providence, Rhode Island: American Mathematical Society, 1984, S. 189–206. ISBN: 9780821850305. DOI: 10.1090/conm/026/737400.

[Li07] Ping Li. „Very Sparse Stable Random Projections for Dimension Reduction in $l\alpha$ ($0 < \alpha \leq 2$) Norm“. In: *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. KDD 2007 (San Jose, CA, USA). Hrsg. von Pavel Berkhin. ACM Special Interest Group on Knowledge Discovery in Data und ACM Special Interest Group on Management of Data. New York, NY: ACM, 2007, S. 440–449. ISBN: 978-1-59593-609-7. DOI: 10.1145/1281192.1281241.

[Stu26] Herbert A. Sturges. „The Choice of a Class Interval“. In: *Journal of the American Statistical Association* 21 (153 1926), S. 65–66. ISSN: 0162-1459. DOI: 10.1080/01621459.1926.10502161.

[Sco79] David W. Scott. „On optimal and data-based histograms“. In: *Biometrika* 66 (3 1979), S. 605–610. ISSN: 0006-3444. DOI: 10.1093/biomet/66.3.605.

[FD81] David Freedman und Persi Diaconis. „On the histogram as a density estimator. L 2 theory“. In: *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete* 57 (4 1981), S. 453–476. DOI: 10.1007/BF01025868.

[BR06] Lucien Birgé und Yves Rozenholc. „How many bins should be put in a regular histogram“. In: *ESAIM: Probability and Statistics* 10 (2006), S. 24–45. DOI: 10.1051/ps:2006001.

[Dav+09] Laurie Davies u. a. „A comparison of automatic histogram constructions“. In: *ESAIM: Probability and Statistics* 13 (2009), S. 181–196. DOI: 10.1051/ps:2008005.

[Sch+00] B. Schölkopf u. a. „Support vector method for novelty detection“. In: *Advances in Neural Information Processing Systems 12*. Max-Planck-Gesellschaft. Cambridge, MA, USA: MIT Press, 2000, S. 582–588.

[Vap95] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Berlin, Heidelberg: Springer-Verlag, 1995. ISBN: 0-387-94559-8.

[TD99] David M. J. Tax und Robert P. W. Duin. „Data domain description using support vectors“. In: *Proceedings of the European Symposium on Artificial Neural Networks*. ESANN'1999 (Bruges (Belgium)). 1999, S. 251–256.

[Tax01] David Martinus Johannes Tax. „One-class classification. Concept-learning in the absence of counter-examples“. PhD Thesis. Delft: Delft University of Technology, 2001. 202 S.

[The14] Andreas Theissler. „Anomaly detection in recordings from in-vehicle networks“. In: *Big Data Applications and Principles. First International Workshop, BigDAP 14*. Proceedings (Madrid). Hrsg. von Alberto Mozo und Hernando, Antonio, Gómez (Eds.), Sandra. 2014, S. 23–38.

[Faw06] Tom Fawcett. „An introduction to ROC analysis“. In: *Pattern Recognition Letters* 27 (8 2006), S. 861–874. ISSN: 01678655. DOI: 10.1016/j.patrec.2005.10.010.

[Bal+00] Pierre Baldi u. a. „Assessing the accuracy of prediction algorithms for classification: an overview“. In: *Bioinformatics* 16 (5 2000), S. 412–424.

[van79] C. J. van Rijsbergen. *Information Retrieval*. eng. 2nd. London: Butterworth-Heinemann, 1979. ISBN: 978-0408709293.

[Chi92] Nancy Chinchor. „MUC-4 Evaluation Metrics“. In: *Proceedings of the Fourth Message Understanding Conference, MUC-4*. Message Understanding Conference (McLean, Virginia, USA). San Mateo, Calif., 1992, S. 22–29. ISBN: 1-55860-273-9. DOI: 10.3115/1072064.1072067.

[Sas07] Yutaka Sasaki. *The truth of the F-measure*. School of Computer Science, University of Manchester, October 26, 2007.

[Agg17] Charu C. Aggarwal. *Outlier Analysis*. eng. 2nd ed. 2017. Aggarwal, Charu C. (VerfasserIn). Cham und s.l.: Springer International Publishing, 2017. 13 S. ISBN:

978-3-319-47577-6. DOI: 10.1007/978-3-319-47578-3.

[Intb] International Organization for Standardization, Hrsg. *Road vehicles – Functional safety*. URL: https://www.iso.org/standards.html.

[Int10] International Electrotechnical Commission, Hrsg. *Functional safety of electrical/electronic/programmable electronic safety-related systems*. 1. Apr. 2010. URL: http://www.iec.ch/functionalsafety/standards/.

[Wal10] Fredrik Walderyd. „Hazard identification and safety goals on power electronics in hybrid vehicles“. Department of Energy and Environment. Master of Science Thesis. Göteborg: Chalmers University of Technology, 2010. 55 S.

[AUT17b] AUTOSAR, Hrsg. *Specification of Module E2E Transformer*. Version AUTOSAR CP Release 4.3.1. 8. Dez. 2017.

[AUT17c] AUTOSAR, Hrsg. *Specification of SW-C End-to-End Communication Protection Library*. Version AUTOSAR CP Release 4.3.1. 8. Dez. 2017.

[Intc] International Organization for Standardization, Hrsg. *Information technology – Security techniques – Evaluation criteria for IT security*. URL: https://www.iso.org/standards.html.

[Int13c] International Organization for Standardization, Hrsg. *Information technology – Security techniques – Information security management systems – Requirements*.

1. Okt. 2013. URL: https://www.iso.org/standard/54534.html.

[SAE16] SAE, Hrsg. *Cybersecurity Guidebook for Cyber-Physical Vehicle Systems*. Surface Vehicle Recommended Practice. Version JAN2016. 1. Jan. 2016.

[Nat16] National Highway Traffic Safety Administration. *Cybersecurity Best Practices for Modern Vehicles*. Washington, DC, October 2016.

[Aut] Automotive Information Sharing and Analysis Center, Hrsg. *Best Practices. Executive Summary*. URL: https://www.automotiveisac.com/best-practices/ (besucht am 22. 09. 2016).

[EVI12] EVITA. *E-safety vehicle intrusion protected applications. Project summary*. Hrsg. von EVITA. Apr. 2012.

[Rud+09] Alastair Ruddle u. a. *Deliverable D2.3: Security requirements for automotive on-board networks based on dark-side scenarios*. EVITA. Version 1.1. 30. December 2009.

[Tas18] Task Force on Cyber Security and Over-the-air issues. *Draft Recommendation on Cyber Security of the Task Force on Cyber Security and Over-the-air issues of UNECE WP.29 GRVA*. UNECE. 2018. URL: https://wiki.unece.org/download/attachments/60362218/GRVA-01-xx%20%28UN%20TF-CS_OTA%29%20Final%20Draft%20Recommendation%20on%20Cyber%20Security%20incl.%20Annex%20A-D.docx?api=v2 (besucht am 04. 01. 2019).

[Eur18] European Commission. *Cybersecurity Act*. 2018. URL: https://ec.europa.eu/commission/news/cybersecurity-act-2018-dec-11_en (besucht am 04. 01. 2019).

[AUT17d] AUTOSAR, Hrsg. *Specification of Secure Onboard Communication*. Version AUTOSAR CP Release 4.3.1. 8. Dez. 2017.

[IEE06] IEEE Computer Society, Hrsg. *IEEE Standard for Local and metropolitan area networks: Media Access Control (MAC) Security*. New York, NY, USA: IEEE, 18. Aug. 2006.

[Int05b] Internet Engineering Task Force, Hrsg. *Security Architecture for the Internet Protocol*. 1. Dez. 2005. URL: https://tools.ietf.org/html/rfc4301.

[Int08] Internet Engineering Task Force, Hrsg. *The Transport Layer Security (TLS) Protocol Version 1.2*. 1. Aug. 2008. URL: https://tools.ietf.org/html/rfc5246.

[Int18] Internet Engineering Task Force, Hrsg. *The Transport Layer Security (TLS) Protocol Version 1.3*. 1. Aug. 2018. URL: https://tools.ietf.org/html/rfc8446.

[Int12b] Internet Engineering Task Force, Hrsg. *Datagram Transport Layer Security Version 1.2*. 1. Jan. 2012. URL: https://tools.ietf.org/html/rfc6347.

[IEE18] IEEE Standards Association, Hrsg. *IEEE Standard for Local and metropolitan area networks—Bridges and Bridged Networks*. New York, NY, USA: IEEE, 2018.

[Int81] Internet Engineering Task Force, Hrsg. *Transmission Control Protocol*. 1. Sep. 1981. URL: https://tools.ietf.org/html/rfc793.

[Int14] International Organization for Standardization, Hrsg. *Road vehicles – Vehicle-to-Grid Communication Interface – Part 2: Network and application protocol requirements*. 1. Apr. 2014.

[Int80] Internet Engineering Task Force, Hrsg. *User Datagram Protocol*. 28. Aug. 1980. URL: https://tools.ietf.org/html/rfc768.

[Stu+13] Ivan Studnia u. a. „Security of embedded automotive networks: state of the art and a research proposal“. In: *SAFECOMP 2013 - Workshop CARS (2nd Workshop on Critical Automotive applications : Robustness & Safety) of the 32nd International Conference on Computer Safety, Reliability and Security*. Hrsg. von Matthieu Roy. Toulouse, France, 2013.

[MJ14] Jerome Miller und Radford Jones, Hrsg. *Personnel Protection. Threat Assessment Profile ; Proven Practices*. eng. Burlington: Elsevier Science, 2014. 10 S. ISBN: 9780128009260.

[vB16] Timo van Roermund und Andy Birnie. *A multi-layer vehicle security framework*. Whitepaper. May 2016. NXP B.V., 2016. 18 S.

[Arg] Argus Cyber Security. *Argus In-Vehicle Network Protection*. URL: https://argus-sec.com/argus-invehicle-network-protection/.

[esc] escrypt. *Angriffserkennung und Angriffsabwehr für Fahrzeuge*. URL: https://www.escrypt.com/de/loesungen/angriffsabwehr-fahrzeuge.

[Oku18] Masakazu Okuda. *Preventing the Next Cyber Attack on Autonomous Vehicles*. Harman - A Samsung Company, 2018.

[Gua] GuardKNOX. *GuardKNOXs Methodik. Communication Lockdown*. URL: https://www.guardknox.com/de/methodik/.

[Saf18] SafeRide Technologies. *Deterministic and Heuristic Technology for Future-Proof Security*. Hrsg. von SafeRide Technologies. 2018. URL: https://saferide.io/#solutions (besucht am 15. 01. 2019).

[Aur18] Aurora Labs. *Self Healing Software*. 2018. URL: https://www.auroralabs.com (besucht am 22. 10. 2018).

[cog18] cognomotive. *CHAI: Cognomotiv Hybrid AI – best match of edge- and cloud-AI*. 2018. URL: https://www.cognomotiv.com/product-overview (besucht am 22. 10. 2018).

[GBL16] Yaron Galula, Ofer Ben-Noon und Oron Lavi. „System and Method for Content Based Anomaly Detection in an In-Vehicle Communication Network“. US2016/0381067 A1 (United States). 29. Dez. 2016.

[BGL15] Ofer Ben Noon, Yaron Galula und Oron Lavi. „Global automotive safety system“. EP 2 892 199 A1 (European Patent Office). 8. Juli 2015.

[All+18] Yair Allouche u. a. „Cooperative Vehicle Monitoring and Anomaly Detection“. US 9,984,512 B2 (United States). International Business Machines Corporation. 29. Mai 2018.

[Moe+14] Douglas S. Moeller u. a. „Method for Vehicle Intrusion Detection with Electronic Control Unit“. US2014/0250531 A1 (United States). 4. Sep. 2014.

[SAE] SAE International. *SAE J3061™ “Cybersecurity Guidebook for Cyber-Physical Vehicle Systems”*.

[06] *SEVECOM (SE-cure VE-hicle COM-munication). General Introduction*. SEVECOM, 20. Sep. 2006.

[Gla+10] Michael Glass u. a. „"SEIS". Security in Embedded IP-based Systems“. In: *ATZ elektronik* (01 2010), S. 36–40.

[SES13] SESAMO, Hrsg. *Security and Safety Modelling. D2.1 - Specification of Safety and Security Mechanisms - Version 0 1*. Final. 29 May 2013.

[DIN02] DIN, Hrsg. *Industrielles Kommunikationssubsystem basierend auf ISO 11898 (CAN) - Teil 4: CANopen*. Beuth, 1. Dez. 2002.

[LNJ08] Ulf E. Larson, Dennis K. Nilsson und Erland Jonsson. „An approach to specification-based attack detection for in-vehicle networks“. In: *Intelligent Vehicles Symposium (IV), 2008 IEEE*. 2008 IEEE Intelligent Vehicles Symposium (Eindhoven, Netherlands). Institute of Electrical and Electronics Engineers. Piscataway, NJ: IEEE, 2008, S. 220–225. ISBN: 978-1-4244-2568-6. DOI: 10.1109/IVS.2008.4621263.

[MGF10] Michael Müter, Andre Groll und Felix C. Freiling. „A structured approach to anomaly detection for in-vehicle networks“. In: *Sixth International Conference on Information Assurance and Security (IAS), 2010. 23 - 25 Aug. 2010, Atlanta, GA, USA : [including workshop papers]*. 2010 Sixth International Conference on Information Assurance and Security (Atlanta, GA, USA). International Conference on Information Assurance and Security, International Workshop on Security and Performance in Emerging Distributed Architectures und International Workshop on Intelligent Technologies for Counter Terrorism and Security. Piscataway, NJ: IEEE, 2010, S. 92–98. ISBN: 978-1-4244-7407-3. DOI: 10.1109/ISIAS.2010.5604050.

[SB17] Noräs Salman und Marco Bresch. „Design and implementation of an intrusion detection system (IDS) for in-vehicle networks“. Department of Computer Science and Engineering. Master's Thesis in Computer Systems and Networks. Göteborg: Chalmers University of Technology and University of Gothenburg, 2017. 88 S.

[TLJ16] Adrian Taylor, Sylvain Leblanc und Nathalie Japkowicz. „Anomaly Detection in Automobile Control Network Data with Long Short-Term Memory Networks“. In: *3rd IEEE International Conference on Data Science and Advanced Analytics*. 2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA) (Montreal, QC, Canada). Hrsg. von Osmar R. Zaïane und Stan Matwin. IEEE International Conference on Data Science and Advanced Analytics u. a.

Piscataway, NJ: IEEE, 2016, S. 130–139. ISBN: 978-1-5090-5206-6. DOI: 10.1109/DSAA.2016.20.

[BP] Görkem Batmaz und Ildikó Pete. *Controller Area Network (CAN) Deep Packet Inspection*. Unter Mitarb. von NVIDIA. Silicon Valley: NVIDIA.

[SNH18] Hiroki Suda, Masanori Natsui und Takahiro Hanyu. „Systematic Intrusion Detection Technique for an In-vehicle Network Based on Time-Series Feature Extraction“. In: *2018 IEEE 48th International Symposium on Multiple-Valued Logic. ISMVL 2018 : proceedings*. 2018 IEEE 48th International Symposium on Multiple-Valued Logic (ISMVL) (Linz, Austria). Institute of Electrical and Electronics Engineers. Piscataway, NJ: IEEE, 2018, S. 56–61. ISBN: 978-1-5386-4464-5. DOI: 10.1109/ISMVL.2018.00018.

[Uni15] United States Congress - Senate. *SPY Car Act of 2015*. 114th Congress, 1st Session. 21. Juli 2015.

[Uni17a] United States Congress - Senate. *SPY Car Act of 2017*. 115th Congress, 1st Session. 21. März 2017.

[Uni17b] United States Congress - House of Representatives. *SPY Car Study Act of 2017*. 115th Congress, 1st Session. 24. Jan. 2017.

[CS16] Kyong-Tak Cho und Kang G. Shin. „Fingerprinting Electronic Control Units for Vehicle Intrusion Detection“. In: *Proceedings of the 25th USENIX Security Symposium*. 25th USENIX Security Symposium (Austin, TX). USENIX Association und USENIX Security. Berkeley, CA: USENIX Association, 2016. ISBN: 978-1-931971-32-4.

[Ji+17] Haojie Ji u. a. „Comparative Performance Evaluation of Intrusion Detection Methods for In-vehicle Networks“. In: *IEEE Access* (2017). ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2848106.

[Web13] Marc Weber. „AUTOSAR lernt Ethernet“. In: *HANSER automotive networks* (Special 2013 2013), S. 30–33.

[Web15] Marc Weber. „Neue Kommunikationsarten in der Automobilvernetzung. Ethernet und CAN FD sind die Wegbereiter“. In: *Automobil Elektronik* (7/8 2015).

[JWH17] Bernd Jesse, Marc Weber und Markus Helmling. „Die Zukunft mit SOA, POSIX, TSN. Automotive Ethernet: Trends und Herausforderungen“. In: *Automobil Elektronik* (11-12 2017).

[The17] Andreas Theissler. „Detecting known and unknown faults in automotive systems using ensemble-based anomaly detection“. In: *Knowledge-Based Systems* 123 (2017), S. 163–173. ISSN: 09507051. DOI: 10.1016/j.knosys.2017.02.023.

[Cal16] California Office of Administrative Law. *California Code of Regulations, Title 13, §1968.2. Malfunction and Diagnostic System Requirements–2004 and Subsequent Model-Year Passenger Cars, Light-Duty Trucks, and Medium-Duty Vehicles and Engines*. 25. Juli 2016.

[Int11a] International Organization for Standardization, Hrsg. *Road vehicles – Functional safety – Part 5: Product development at the hardware level*. 15. Nov. 2011.

[RZL17] Prajit Ramachandran, Barret Zoph und Quoc V. Le. *Searching for Activation Functions*. 2017. URL: http://arxiv.org/pdf/1710.05941.

[He+15] Kaiming He u. a. „Delving Deep into Rectifiers. Surpassing Human-Level Performance on ImageNet Classification“. In: *2015 IEEE International Conference on Computer Vision. 11-18 December 2015, Santiago, Chile : proceedings*. 2015 IEEE International Conference on Computer Vision (ICCV) (Santiago, Chile). Hrsg. von Ruzena Bajcsy, Greg Hager und Yi Ma. IEEE International Conference on Computer Vision, Institute of Electrical and Electronics Engineers und ICCV. Piscataway, NJ: IEEE, 2015, S. 1026–1034. ISBN: 978-1-4673-8391-2. DOI: 10.1109/ICCV.2015.123.

[GBB11] Xavier Glorot, Antoine Bordes und Yoshua Bengio. „Deep Sparse Rectifier Neural Networks“. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Hrsg. von Geoffrey Gordon, David Dunson und Miroslav Dudík. Bd. 15. Proceedings of Machine Learning Research. 11–13 Apr. Fort Lauderdale, FL, USA: PMLR, 2011, S. 315–323. URL: http://proceedings.mlr.press/v15/glorot11a.html.

[MHN13] Andrew L. Maas, Awni Y. Hannun und Andrew Y. Ng. „Rectifier Nonlinearities Improve Neural Network Acoustic Models“. In: *Proceedings of the 30th International Conference on Machine Learning*. 30th International Conference on Machine Learning (Atlanta, Georgia, USA). 2013.

[Zei+13] M. D. Zeiler u. a. „On Rectified Linear Units for Speech Processing“. In: *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE International Conference on Acoustics, Speech and Signal Processing (Vancouver, British Columbia, Canada). IEEE Signal Processing Society. Piscataway, NJ: IEEE, 2013.

[NH10] Vinod Nair und Geoffrey E. Hinton. „Rectified Linear Units Improve Restricted Boltzmann Machines“. In: *Proceedings of the 27th International Conference on Machine Learning*. 27th International Conference on Machine Learning (Haifa, Israel). 2010.

[KSH12] Alex Krizhevsky, Ilya Sutskever und Geoffrey E. Hinton. „ImageNet Classification with Deep Convolutional Neural Networks“. In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. 25th International Conference on Neural Information Processing Systems - Volume 1 (Lake Tahoe, Nevada, USA). NIPS'12. USA: Curran Associates Inc., 2012, S. 1097–1105. URL: http://dl.acm.org/citation.cfm?id=2999134.2999257.

[God18] Daniel Godoy. *Hyper-parameters in Action! Part II – Weight Initializers*. Towards Data Science. 2018. URL: https://towardsdatascience.com/hyper-parameters-in-action-part-ii-weight-initializers-35aee1a28404 (besucht am 13. 01. 2019).

[KB14] Diederik P. Kingma und Jimmy Ba. *Adam. A Method for Stochastic Optimization*. 2014. URL: http://arxiv.org/pdf/1412.6980.

[Aba+15] Martín Abadi u. a. *TensorFlow. Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[17] *31st Conference on Neural Information Processing Systems (NIPS 2017). NIPS 2017 Autodiff Workshop*. The future of gradient-based machine learning software and techniques (Long Beach, CA, USA). 2017.

[PyT] PyTorch. *PyTorch*. URL: https://pytorch.org/ (besucht am 30. 12. 2018).

[Jia+14] Yangqing Jia u. a. „Caffe. Convolutional Architecture for Fast Feature Embedding“. In: *arXiv preprint arXiv:1408.5093* (2014).

[Ped+11] F. Pedregosa u. a. „Scikit-learn. Machine Learning in Python“. In: *Journal of Machine Learning Research* 12 (2011), S. 2825–2830.

[LA15] Alexander Lavin und Subutai Ahmad. „Evaluating Real-Time Anomaly Detection Algorithms – The Numenta Anomaly Benchmark“. In: *2015 IEEE 14th International Conference on Machine Learning and Applications. Proceedings*. 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA) (Miami, FL, USA). Institute of Electrical and Electronics Engineers. Piscataway, NJ: IEEE, 2015, S. 38–44. ISBN: 978-1-5090-0287-0. DOI: 10.1109/ICMLA.2015.141.

[Ahm+17] Subutai Ahmad u. a. „Unsupervised real-time anomaly detection for streaming data“. In: *Neurocomputing* 262

(2017), S. 134–147. ISSN: 09252312. DOI: 10.1016/j.neucom.2017.04.070.

[13] *MISRA C:2012. Guidelines for the use of the C language in critical systems*. eng. Nuneaton: Misra, 2013. 226 S. ISBN: 978-1-906400-11-8.

[AUT17e] AUTOSAR, Hrsg. *General Requirements on Basic Software Modules*. Version AUTOSAR CP Release 4.3.1. 8. Dez. 2017.

[16] *MISRA C:2012 - Amendment 1. Additional security guidelines for MISRA C:2012*. Nuneaton: Misra, 2016. ISBN: 978-906400-16-3.

[18a] *MISRA C:2012 - Addendum 2. Coverage of MISRA C:2012 against ISO/IEC TS 17961:2013 "C Secure" - Second Edition*. Nuneaton: Misra, 2018. ISBN: 978-906400-18-7.

[18b] *MISRA C:2012 - Addendum 3. Coverage of MISRA C:2012 against CERT C*. Nuneaton: Misra, 2018. ISBN: 978-906400-19-4.

[Nis31] Steffen Nissen. „Implementation of a Fast Artificial Neural Network Library (FANN)“. Department of Computer Science. Copenhagen: University of Copenhagen, 2003-10-31. 92 S.

[Zel+94] Andreas Zell u. a. „SNNS (Stuttgart Neural Network Simulator)“. In: *Neural Network Simulation Environments*. Hrsg. von Josef Skrzypek. Bd. 254. The Kluwer International Series in Engineering and Computer Science. Boston, MA: Springer US, 1994, S. 165–186. ISBN: 978-1-4613-6180-0. DOI: 10.1007/978-1-4615-2736-7_9.

[Int17] Internet Engineering Task Force, Hrsg. *The JavaScript Object Notation (JSON) Data Interchange Format*. 1. Dez. 2017. URL: https://tools.ietf.org/html/rfc8259.

[Fac17] Facebook Inc. *ONNX. Open Neural Network Exchange Format*. The new open ecosystem for interchangeable AI models. 2017. URL: https://onnx.ai/ (besucht am 10. 09. 2018).

[Khr18] Khronos Group. *Neural Network Exchange Format (NNEF)*. 2018. URL: https://www.khronos.org/nnef.

[Tur] Chris Turner. *ARM processors driving automotive innovation*. Seoul und Taipei: ARM.

[Int11b] International Organization for Standardization, Hrsg. *Information technology – Programming languages – C*. Version 3. 1. Dez. 2011. URL: https://www.iso.org/standard/57853.html.

[Pyt] Python Software Foundation, Hrsg. *Time access and conversions*. Version 3.7.0. URL: https://docs.python.org/3/library/time.html.

[Lon] Leonardo Lontra. *Tensorflow-on-arm*. URL: https://github.com/lhelontra/tensorflow-on-arm (besucht am 07. 04. 2018).

[ACH97] H. Amin, K. M. Curtis und B. R. Hayes-Gill. „Piecewise linear approximation applied to nonlinear function of a neural network“. In: *IEE Proceedings - Circuits, Devices and Systems* 144 (6 1997), S. 313–317. ISSN: 13502409. DOI: 10.1049/ip-cds:19971587.

[Ten18b] TensorFlow, Hrsg. *Install TensorFlow on Raspbian*. 2018. URL: https://www.tensorflow.org/install/install_raspbian (besucht am 17. 09. 2018).

[WD98] R. C. Whaley und J. J. Dongarra. „Automatically Tuned Linear Algebra Software“. In: *SC '98: Proceedings of the 1998 ACM/IEEE Conference on Supercomputing. 10th anniversary : high performance networking and computing conference*. SC98 - High Performance Networking and Computing Conference (Orlando, FL, USA). Conference : Supercomputing, IEEE Computer Society und Sigarch. IEEE, 1998, S. 38–70. ISBN: 0-8186-8707-X. DOI: 10.1109/SC.1998.10004.

[ATL] ATLAS. *Automatically Tuned Linear Algebra Software*. URL: http://math-atlas.sourceforge.net/ (besucht am 17. 09. 2018).

[Sys] SysProgs. *Prebuilt GNU toolchain for powerpc-eabi*. URL: http://gnutoolchains.com/powerpc-eabi/ (besucht am 26. 12. 2018).

[IPG] IPG Automotive GmbH. *CarMaker. Pkw und leichte Nutzfahrzeuge virtuell testen*. URL: https://ipg-automotive.com/de/produkte-services/simulation-software/carmaker/ (besucht am 28. 12. 2018).

[Tes] Tesla. *Support. Software-Updates*. URL: https://www.tesla.com/de_DE/support/software-updates (besucht am 29. 09. 2018).

[MS17] Mirco Marchetti und Dario Stabili. „Anomaly detection of CAN bus messages through analysis of ID sequences“. In: *Intelligent Vehicles Symposium (IV),*

2017 IEEE. 2017 IEEE Intelligent Vehicles Symposium (IV) (Los Angeles, CA, USA). Institute of Electrical and Electronics Engineers. Piscataway, NJ: IEEE, 2017, S. 1577–1583. ISBN: 978-1-5090-4804-5. DOI: 10.1109/IVS.2017.7995934.

[Lee+18] Wei-Han Lee u. a. *Time Series Segmentation through Automatic Feature Learning*. 2018. URL: https://arxiv.org/abs/1801.05394v2 (besucht am 12. 01. 2019).

[Hei28] Thomas Heinz. „HiPAC. High Performance Packet Classification for Netfilter“. Fachbereich Informatik. Diplomarbeit. Saarbrücken: Universität des Saarlandes, 2004-02-28. 99 S.

[Ada16] Tony Adamson. *Migrating to CAN FD*. Unter Mitarb. von Vector Informatik GmbH. NXP, 16. Feb. 2016.

[NXP18] NXP. *NXP® TJA115x Secure CAN Transceiver Family. Secure CAN communication without cryptography*. Hrsg. von NXP. 2018.

[HKD08] Tobias Hoppe, Stefan Kiltz und Jana Dittmann. „Adaptive Dynamic Reaction to Automotive IT Security Incidents Using Multimedia Car Environment“. In: *Fourth International Symposium on Information Assurance and Security. ISIAS '08*. 2008 Fourth International Conference on Information Assurance and Security (IAS) (Naples, Italy). Hrsg. von Massimiliano Rak. IEEE Computer Society u. a. Piscataway, NJ: IEEE, 2008, S. 295–298. ISBN: 978-0-7695-3324-7. DOI: 10.1109/IAS.2008.45.

[HKD09] Tobias Hoppe, Stefan Kiltz und Jana Dittmann. „Applying Intrusion Detection to Automotive IT – Early In-

sights and Remaining Challenges“. In: *Journal of Information Assurance and Security* 4 (2009), S. 226–235.

[Ste57] Karl Steinbuch. „INFORMATIK: Automatische Informationsverarbeitung“. In: *SEG-Nachrichten* (4 1957).

[Ste61] Karl Steinbuch. „Die Lernmatrix“. In: *Kybernetik* 1 (1 1961), S. 36–45. DOI: 10.1007/BF00293853.

[Ste63] Karl Steinbuch. *Automat und Mensch. Kybernetische Tatsachen und Hypothesen*. ger. Zweite erweiterte Auflage. Steinbuch, Karl (author.) Berlin, Heidelberg und s.l.: Springer Berlin Heidelberg, 1963. 392 S. ISBN: 9783642531712. DOI: 10.1007/978-3-642-53170-5. URL: http://dx.doi.org/10.1007/978-3-642-53170-5.

[AUT17f] AUTOSAR, Hrsg. *Specification of Operating System*. Version AUTOSAR CP Release 4.3.1. 8. Dez. 2017.

[AUT17g] AUTOSAR, Hrsg. *Specification of Watchdog Manager*. Version AUTOSAR CP Release 4.3.1. 8. Dez. 2017.

[Mica] Microsoft. *Security Development Lifecycle*. URL: https://www.microsoft.com/en-us/sdl/default.aspx.

[Sho] Adam Shostack. *Experiences Threat Modeling at Microsoft*. Microsoft. URL: https://pdfs.semanticscholar.org/3003/487e89fdb3cf3698ae30c44fe34afd522c4a.pdf.

[Sho08] Adam Shostack. *SDL Threat Modeling: Past, Present and Future*. Microsoft, 2008.

[HL01] Michael Howard und David LeBlanc. *Writing secure code. Practical strategies and proven techniques for building secure applications in a networked world.* eng. Howard, Michael (VerfasserIn). Redmond, Washington, USA: Microsoft Press, 2001. 477 S. ISBN: 9780735615885.

[HL03] Michael Howard und David LeBlanc. *Writing secure code. Practical strategies and proven techniques for building secure applications in a networked world.* eng. 2. ed. Redmond, Washington, USA: Microsoft Press, 2003. 768 S. ISBN: 978-0735617223.

[Bun12] Bundesamt für Sicherheit in der Informationstechnik - BSI, Hrsg. *Leitfaden Informationssicherheit. IT-Grundschutz kompakt*. Bonn, Februar 2012.

[Micb] Microsoft, Hrsg. *Microsoft Azure Machine Learning: Algorithm Cheat Sheet*. URL: `https://download.microsoft.com/download/A/6/1/A613E11E-8F9C-424A-B99D-65344785C288/microsoft-machine-learning-algorithm-cheat-sheet-v7.pdf` (besucht am 26. 11. 2018).

[SW00] Uwe Schneider und Dieter Werner. *Taschenbuch der Informatik*. 3. Auflage. München: Fachbuchverlag Leipzig im Carl Hanser Verlag, 2000. 19 S.

[AUT17h] AUTOSAR, Hrsg. *Specification of Crypto Service Manager*. Version AUTOSAR CP Release 4.3.1. 8. Dez. 2017.

[AUT18] AUTOSAR, Hrsg. *Specification of Diagnostic Communication Manager*. Version AUTOSAR CP Release 4.4.0. 31. Okt. 2018.

Veröffentlichungen

[VWeb+18a] Marc Weber u. a. „Embedded Hybrid Anomaly Detection for Automotive CAN Communication“. In: *Proceedings of the 9th European Congress on Embedded Real Time Software and Systems*. 9th European Congress on Embedded Real Time Software and Systems (ERTS 2018) (Toulouse, France). 2018. URL: https://hal.archives-ouvertes.fr/hal-01716805.

[VWeb+18b] Marc Weber u. a. „A Hybrid Anomaly Detection System for Electronic Control Units Featuring Replicator Neural Networks“. In: *Advances in Information and Communication Networks. Proceedings of the 2018 Future of Information and Communication Conference (FICC), Vol. 2*. Future of Information and Communication Conference (FICC) 2018 (Singapore). Hrsg. von Kohei Arai, Supriya Kapoor und Rahul Bhatia. Springer, 2018, S. 43–62. ISBN: 978-3-030-03405-4. DOI: 10.1007/978-3-030-03405-4_4.

[VGWS18] Daniel Grimm, Marc Weber und Eric Sax. „An Extended Hybrid Anomaly Detection System for Automotive Electronic Control Units Communicating via Ethernet. Efficient and Effective Analysis using a Specification- and Machine Learning-based Approach“. In: *VEHITS 2018. Proceedings of the 4th International Con-*

ference on Vehicle Technology and Intelligent Transport Systems : Funchal, Madeira, Portugal, March 16-18, 2018. 4th International Conference on Vehicle Technology and Intelligent Transport Systems (Funchal, Madeira, Portugal). Hrsg. von Markus Helfert und Oleg Gusikhin. Institute for Systems and Technologies of Information, Control and Communication, International Conference on Vehicle Technology and Intelligent Transport Systems und VEHITS. Setúbal, Portugal: SCITEPRESS - Science and Technology Publications Lda, 2018, S. 462–473. ISBN: 978-989-758-293-6. DOI: 10.5220/0006779204620473.

[VWeb+] Marc Weber u. a. *Online Detection of Anomalies in Vehicle Signals using Replicator Neural Networks*. Unter Mitarb. von isits AG International School of IT Security. Ypsilanti, MI, USA. URL: https://www.escar.info/downloads.html.

[VWeb18] Marc Weber. *Automotive OBD-II Dataset*. Karlsruher Institut für Technologie (KIT) - Institut für Technik der Informationsverarbeitung, 2018. DOI: 10.5445/IR/1000085073.

Betreute Master- und Bachelorarbeiten

[MABAMaa17] Jonas Maas. „Anomalieerkennung für einen CAN-Bus, basierend auf maschinellem Lernen“. Institut für Technik der Informationsverarbeitung (ITIV). Masterarbeit. Karlsruhe: Karlsruher Institut für Technologie, 2017.

[MABARie17] Robin Ries. „Evaluierung eines Security Konzepts und Erweiterung um ein Intrusion Detection System“. Institut für Technik der Informationsverarbeitung (ITIV). Bachelorarbeit. Karlsruhe: Karlsruher Institut für Technologie, 2017.

[MABAAlm17] Pedro Alves Almeida. „Evaluation and Prototypical Implementation of Machine Learning to Detect ECU Misbehaviour“. Institut für Technik der Informationsverarbeitung (ITIV). Master Thesis. Karlsruhe: Karlsruher Institut für Technologie, 2017.

[MABAGri17] Daniel Grimm. „Statische und dynamische Anomalieerkennung in Ethernet-basierter Kommunikation“. Institut für Technik der Informationsverarbeitung (ITIV). Masterarbeit. Karlsruhe: Karlsruher Institut für Technologie, 2017.

[MABAWol18b] Georg Wolf. „Anomalieerkennung für Kommunikationssignale in Fahrzeugnetzwerken mit Hilfe von neuronalen Netzen“. Institut für Technik der Informations-

verarbeitung (ITIV). Bachelorarbeit. Karlsruhe: Karlsruher Institut für Technologie, 2018.

[MABALei18] Simon Leiner. „Realisierung und Evaluierung Neuronaler Netze zur Anomalieerkennung in Automotive Steuergeräten“. Institut für Technik der Informationsverarbeitung (ITIV). Bachelorarbeit. Karlsruhe: Karlsruher Institut für Technologie, 2018.

[MABAKlu18] Simon Klug. „Effiziente Anomalieerkennung in Fahrzeug-internen Kommunikationsnetzwerken mit selbstlernenden, Ensemble-basierten Algorithmen“. Institut für Technik der Informationsverarbeitung (ITIV). Bachelorarbeit. Karlsruhe: Karlsruher Institut für Technologie, 2018.

[MABAWie18] Thaddäus Wiedemer. „Host-basierte Anomalieerkennung für Automotive Steuergeräte“. Institut für Technik der Informationsverarbeitung (ITIV). Bachelorarbeit. Karlsruhe: Karlsruher Institut für Technologie, 2018.

[MABARah18] Mohammad Mahinur Rahman. „Anomalieerkennung in CAN-Kommunikation - eine ganzheitliche Betrachtung“. Institut für Technik der Informationsverarbeitung (ITIV). Master Thesis. Karlsruhe: Karlsruher Institut für Technologie, 2018.

[MABAGeo18] Ivo Georgiev. „Konzeption und prototypische Umsetzung einer effizienten CAN-Firewall für AUTOSAR-basierte Steuergeräte“. Institut für Technik der Informationsverarbeitung (ITIV). Masterarbeit. Karlsruhe: Karlsruher Institut für Technologie, 2018.

[MABARei18] Alexander Reichert. „Anomalieerkennung für Linux-Systeme anhand von Betriebssysteminformationen“. Institut für Technik der Informationsverarbeitung (ITIV). Masterarbeit. Karlsruhe: Karlsruher Institut für Technologie, 2018.

[MABALan18] Tobias Lang. „Anomalieerkennung in Automotive CAN-Kommunikation mittels maschinellem Lernen und generischer Klassifizierungsmerkmale“. Institut für Technik der Informationsverarbeitung (ITIV). Masterarbeit. Karlsruhe: Karlsruher Institut für Technologie, 2018.

Lebenslauf

Persönliche Daten

Name	Marc Weber
Geburtsdatum	27. Mai 1986
Geburtsort	Reutlingen
Staatsangehörigkeit	deutsch

Schulbildung

1992 – 1996	Mörikeschule Reutlingen-Sondelfingen
1996 – 2002	Hermann-Hesse-Realschule Reutlingen
2002 – 2005	Werner-Siemens-Schule Stuttgart
	Abschluss: Fachhochschulreife

Studium

2005 – 2009	Studium der Technischen Informatik an der Hochschule Esslingen
	Abschluss: Bachelor of Engineering
2009 – 2011	Studium in Automotive Systems an der Hochschule Esslingen
	Abschluss: Master of Engineering

Berufliche Tätigkeit

2002 – 2005	Ausbildung zum IT-Systemelektroniker bei der Acterna Germany GmbH
2005	Elektroniker/Prüfer bei der Acterna Germany GmbH
2007 – 2008	Praktikum bei der Dr. Ing. h.c. F. Porsche AG
2008 – 2009	Bachelor-Thesis bei der Robert Bosch GmbH
2009	Zivildienst am Universitätsklinikum Tübingen
2010 – 2011	Master-Thesis bei Mercedes-Benz Research & Development North America, Inc.
2011 – 2013	Software Development Engineer bei der Vector Informatik GmbH
seit 2013	Product Management Engineer bei der Vector Informatik GmbH
2016 – 2018	Akademischer Mitarbeiter am Karlsruher Institut für Technologie (KIT)