UNIVERSITÄT LEIPZIG

Masterarbeit

System zur Sammlung, Analyse und Verwendung von Sensordaten in Fahrzeugen

Zur Erlangung des akademischen Grades eines
Master of Science
- Informatik -

TOMTOM
TELEMATICS

Fakultät Informatik
Referent: Prof. Dr. Martin Bogdan
Korreferent: Dr. Christian Meißner

eingereicht von:
Kai Trott
[redacted]

Leipzig, den 1. März 2017

Zusammenfassung

Die Arbeit behandelt das Mitschneiden von Sensordaten in einem Gerät und dem externen Speichern in einem erweiterbaren Dateiformat. Durch diese Daten soll eine fachgemäße Verwendung vollkommen simuliert werden können um automatisierte Software-Tests zu ermöglichen. Als Schwerpunkte werden betrachtet:

- Evaluierung geeigneter Geräte und eines Annotationssystems zur Nutzung während der Datenaufnahme,
- Definition von aufzunehmenden Sensordaten und Datenaufnahmeformaten,
- Entwicklung geräteseitiger Firmwareanpassungen zur Datenausleitung,
- exemplarische Reimplementierung eines bereits in der Firmware vorhandenen Algorithmus, welcher die aufgenommenen Daten als Eingabe verwendet und
- Betrachtung des entwickelten Systems in Bezug auf sicherheitsrelevante Aspekte.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Glossar

Bus

Datenleitung auf der mehrere Teilnehmer miteinander verbunden sind.

Flag

Vom Betriebssystem bereitgestellter Wert und damit verbundener Möglichkeit zum warten auf eine spezifische Wertänderung.

Header

Erstes Element in einer Datenstruktur, welches für Zusaetzliche Objektbeschreibende Informationen verwendet wird.

Queue

Eindimensionale Liste an Nachrichten mit definierter Reihenfolge.

Testfahrt

Autofahrt zum testen eines Gerätes im Betrieb.

Watchdog

Sicherheitsmechanismus zum Neustarten des Gerätes bei endlosen Arbeitsschritten.

Webfleet

Eine von TomTom betriebene Webseite zur Flottenverwaltung.

Acronyms

ADC	Analog to Digital Converter
CAN	Controller Area Network
CRC	Cyclic Redundancy Check
CSV	Comma Separated Values
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	Global System for Mobile Communications
MTU	Maximum Transmission Unit
OBD	On-Board-Diagnose
PCAP	Packet Capture
PCAPNG	Packet Capture Next Generation
PIN	Personal Identification Number
SQL	Structured Query Language
TG-C	TomTom Gerät CAN
TG-O	TomTom Gerät OBD
TLS	Transport Layer Security
USB	Universal Serial Bus

1 Einleitung

Softwareentwickler werden durch viele Hilfsmittel während der Arbeit unterstützt, welche die Effektivität und die Produktivität fördern und die Qualität und Stabilität der Software festigen sollen. So unter anderem durch

- Buildserver,
- Reviewserver,
- Elektronisches Team Management,
- Versionierungssysteme und
- Dokumentationen.

Alle diese Konzepte dienen zum koordinierten Zusammenarbeiten im Team. Besonders bei komplexen Systemen werden schnell Fehler produziert an Stellen, die nicht immer gleich ersichtlich sind. Bei großen Softwareprojekten sind *automatisierte Softwaretests* unumgänglich um die Funktionalität der Software auch auf lange Sicht zu gewährleisten. Diese können die Qualität schon frühzeitig während des Entwickelns erhöhen und geben einen direkten Überblick über die Stabilität des Projektes.

Diese Arbeit wurde in Verbindung mit der Firma *TomTom Telematics DE*[1] erstellt und befasst sich mit dem Bereitstellen von Testdaten für eine automatisierte Testumgebung im Umfeld einer TomTom eigenen Hardwareplattform.

1.1 Ziel der Arbeit

Diese Arbeit soll die Möglichkeit aufzeigen, gerätespezifische Sensordaten während des Betriebs zu exportieren. Diese Daten sollen dabei *vollständig* und vor allem *unverändert* sein, um eine gesamte Testfahrt darauf basierend simulieren zu können. Die Daten müssen in ihrer *Integrität* getestet und über ein Annotationssystem mit Metainformationen versehbar sein. Die Eingriffe in das bestehende System dürfen die Funktionalität der Software nur erweitern und keine existierenden Mechanismen abändern, um die Stabilität nicht zu gefährden. Es ist eine möglichst transparente Integration in das bestehende System zu erstreben.

[1] `https://telematics.tomtom.com`

1.2 Problemstellung

Im Verlauf dieser Arbeit ist eine Anpassung an eine Gerätesoftware zur externen Speicherung von internen Sensordaten im laufenden Betrieb zu entwickeln. Folgende Schwerpunkte sind gesetzt:

- Die Integrität der Daten und
- die Vollständigkeit der Daten.

Das Speicherformat der Sensordaten soll für zukünftige Erweiterungen ausgelegt sein und muss Kommentare zu den Sensordaten erlauben. Als Werkzeug zur Erhebung der Daten ist eine grafische Oberfläche zu erstellen. Diese muss für den Nutzer komfortabel bedienbar sein und Möglichkeiten der Erweiterung bieten.

Die resultierenden Exportdaten sollen als Grundlage für den Nachbau eines im System integrierten Algorithmus dienen und die Nutzbarkeit als Testdaten damit belegen.

1.3 Abgrenzung und Aufbau

Es wird eine öffentliche Version[2] dieser Arbeit angestrebt. Um dies zu erreichen werden firmenspezifische Informationen namentlich abgeändert oder beispielhaft ersetzt.

Die Arbeit beginnt mit einer Analyse des bestehenden Systems und der gestellten Geräte. Hierbei werden die vorhandenen Sensoren analysiert, die Schnittstelle für die Datenübertragung definiert und das Speicherformat für die Sensordaten gewählt. Die Analyse dient zur Erfassung des Ist-Zustandes der Plattform, der bestehenden Hardwarekomponenten und der Abgrenzung des möglichen Umsetzungsweges. In dieser Arbeit werden ebenfalls die Sicherheitsrelevanten Aspekte in der Kommunikation mit dem Gerät beschrieben aber nicht näher analysiert.

Folgend wird das Konzept einer ersten Implementierung gezeigt und eine Abänderung davon, um sich den Systemeigenheiten anzupassen. Die Anpassungen umfassen eine transparente Integration in das System, eine effizientere Übertragung der Sensordaten und das Optimieren der gewählten Schnittstelle.
Zudem wird die Nutzung des Speicherformates dargestellt und eine Benutzeroberfläche zum Erheben der Daten mit seinen Anforderungen und der Umsetzung beschrieben.

Zur Evaluation der Daten werden die Annotationen mit markanten Pegeln in den Beschleunigungs- und ADC-Werten verglichen und eine zeitliche Übereinstimmung geprüft.

[2] ohne Vertraulichkeitserklärung oder Sperrvermerk

Es wird ein interner Algorithmus nachgebaut und die exportierten Sensordaten darauf angewendet. Eine Betrachtung der originalen Resultate und der Ausgabedaten des nachgebauten Algorithmus, zeigen einen fehlerfreien Export und die Nutzbarkeit der Daten.
Für CAN-Daten wird ein Vergleich von bekannten Werten mit den erhobenen Daten angelegt. Hierbei wird die Reihenfolge, die Integrität und die Geschwindigkeit der Daten des Exportvorgangs analysiert.
Die aufgenommenen GPS-Punkte werden nachfolgend mit den erhaltenen Werten von Webfleet verglichen und eine Abweichung der Strecke analysiert.
Abschließend wird eine Gesamtauswertung der einzelnen Tests gegeben.

2 Analyse

Das Kapitel Analyse gibt eine Übersicht über die eingesetzten Geräte und die Software-Architektur des bestehenden Systems. Es werden die verschiedenen Schnittstellen nach außen als möglicher Datenkanal betrachtet, das Speicherformat wird bestimmt und Sicherheitsaspekte bezüglich der Kommunikation mit dem Gerät beleuchtet.

2.1 Sensoren

Als Sensor wird ein Objekt bezeichnet, welches definierte Eigenschaften seiner Umgebung qualitativ erfassen kann (vgl. [ITW17b]).
Tabelle 2.1 zeigt verschiedene Informationsquellen der in Kapitel 2.2.1 betrachteten Geräte und umfasst alle nutzbaren Sensoren. Diese werden nachfolgend näher beschrieben. Zur Erleichterung der Verständlichkeit werden alle aufgeführten Informationsquellen weitergehend unter der Bezeichnung Sensoren geführt, auch wenn nicht alle der Definition eines Sensors entsprechen.

Für den Export sind jene Informationen interessant, welche von Diensten an Applikationen entsprechend Kapitel 2.2.2 weitergegeben werden.

Sensor	**Geschwindigkeit**	**Verwendung**
ADC	100 Hz	✓
Beschleunigung	50 Hz	✓
CAN	40 Kbit/s - 1 Mbit/s	✓
GPS	1 Hz	✓
Drucktaster	-	
Modem	-	✓
Motorzustand	-	✓
OBD	-	✓
Temperatur	-	
Zündung	-	✓

Tabelle 2.1: Sensoren-Übersicht

ADC: Der Analog to Digital Converter (ADC) erfasst ein analoges Signal und wandelt es über eine sukzessive Approximation in ein digitales Signal um. Hierbei wird eine Referenzspannung der Eingangsspannung schrittweise angenähert und mit

jedem Schritt die Verschiebung der Referenzspannung um die Hälfte zur vorherigen Schrittweite verändert (vgl. [App17]). Der Sensor in allen betrachteten Geräten hat eine Präzision von 14 Bit. Die Abfrage wurde im Bereich dieser Arbeit von einem *On-Demand-System* zu einer *zyklischen* Werteerzeugung umgebaut. Die Abtastfrequenz ist softwareseitig einstellbar und innerhalb der Plattform auf 100 Hertz gesetzt worden.

Bechleunigung: Die Bewegung wird durch einen Beschleunigungssensor erfasst und auf einer X-, Y- und Z-Achse kapazitiv mit einer Taktfrequenz von 50 Hertz gemessen. Die Informationen werden hierbei in milli-g erfasst (vgl. [Sch07]). Die Einheit g steht für die mittlere Erdbeschleunigung und entspricht $1g = 9,81\frac{m}{s^2}$. Für eine präzise Lagenbestimmung wird der Sensor in einem mehrstufigen Verfahren kalibriert, um die genaue Lage des Gerätes im Fahrzeug bestimmen zu können. Für den Export macht es jedoch keinen Unterschied, da die Sensorinformationen sich dadurch nicht ändern.

CAN: Controller Area Network (CAN) ist in ISO 11898 definiert (vgl. [ISO15]) und dient in einem Fahrzeug der Kommunikation zwischen Sensoren, Aktoren und Prozessoren. Es handelt sich um ein serielles Bussystem, welches unterschiedlichste Steuergeräte im Fahrzeug miteinander verbindet und eine Kommunikation derer dadurch erlaubt. In sicherheitsrelevanten Mechanismen (z. B. Airbag) wird ein vom restlichen System abgeschotteter CAN-Bus verwendet, um eine möglichst zeitnahe Abarbeitung der Daten zu garantieren (vgl. [WWP04]).

Es wird zwischen einem *Highspeed-CAN* (ISO 11898-2) und einem *Lowspeed-CAN* (ISO 11898-3) unterschieden. Diese beiden Arten sind untereinander inkompatibel aufgrund der Übertragungsraten, ihrer Fehlertoleranz und der Terminierung des Busses und können daher nicht in einem Sytem zusammen verwendet werden (vgl. [Ins17]).
Innerhalb eines geschlossenen Systems nutzen alle mit dem Bus verbundenen Geräte dieselbe Taktung und diese kann nicht zur Laufzeit verändert werden. Der Highspeed-CAN unterstützt Geschwindigkeiten von 40 Kbit/s bis 1 Mbit/s. Die mögliche Höchstgeschwindigkeit wird jedoch durch die Kabellänge eingeschränkt. Der Bus wird an jedem Ende durch einen 120 Ohm Widerstand terminiert.
Lowspeed-CAN erlaubt Übertragungsraten im Bereich von 40 Kbit/s bis zu 125 Kbit/s. Jedes Gerät hat hierbei seine eigene Terminierung und erlaubt somit eine weitere Kommunikation im Falle einer Falschverkabelung einzelner Geräte.

Jedes angeschlossene Gerät prüft eigenständig, ob die Kommunikationsleitung frei ist und sendet nur in diesem Fall seine Daten. Bei einer zeitgleichen Kommunikation mehrerer Geräte wird über eine Arbitrierung entschieden welches Gerät senden darf. Das heißt, es darf die Nachricht mit der höchsten Priorisierung die Leitung belegen. Die Priorität wird über die CAN-ID bestimmt, welche eineindeutig im System vorhanden sein muss. Durch diese Regelung der Nachrichtenreihenfolge, können die CAN-Daten deterministisch versendet werden.

Eine Datennachricht beinhaltet:

- ein Extended Bit,
- eine CAN-ID mit einer Länge von 11 oder 29 Bits, abhängig vom gesetzten Extended Bit,
- ein Remote Bit,
- ein Bestätigungs-Bit,
- eine Datenlänge,
- die Daten selber mit einer maximalen Länge von 8 Byte und
- einen Cyclic Redundancy Check (CRC) um die Integrität der Daten testen zu können.

Es werden keine Informationen über den Absender oder den Empfänger mitgesendet. Deshalb wird jede Nachricht von jedem im Netz befindlichen Gerät erhalten und bei Bedarf verarbeitet. Dies ermöglicht ein schnelles Hinzufügen oder Entfernen von Geräten.

Es werden vier verschiedene Arten von Nachrichten unterschieden:

- Standard Frame,
- Extended Frame,
- Remote Frame und
- Error Frame.

Der **Standard Frame** sowie der **Extended Frame** sind für die Übermittlung der eigentlichen Daten verantwortlich und unterscheiden sich durch ein gesetztes *Extended Bit* und damit eine längere CAN-ID. In Abbildung 2.1 wird der Unterschied dieser beiden Nachrichten dargestellt.
Durch ein **Remote Frame** wird eine Anfrage auf bestimmte Daten einer CAN-ID gestellt. Als Antwort wird ein Standard oder Extended Frame erwartet und die CAN-ID bei der Anfrage ist hierbei gleich der erwarteten Antwort-ID.
Das **Error Frame** zeigt einen fehlerhaft erhaltenen Frame an und erzeugt ein erneutes senden der Nachricht.

Jedes Frame **startet** mit einem definierten Bitmuster, um den Beginn einer Nachricht anzuzeigen. Die nachfolgende **ID** wird zur Identifikation der Daten und zur Priorisierung verwendet. Das **Remote** Bit dient als Anfrage nach den zur ID gehörenden Daten. Die **Daten** werden aufgrund ihrer dynamischen **Länge**, mit Angabe dieser Länge versendet. Zur Überprüfung der Integrität wird ein zwei Byte langer **CRC** verwendet. Ein Paket muss mindestens einmal von einem Empfänger als Empfangen quittiert worden sein durch das Setzen eines dominanten Pegels beim **ACK** Bit. Eine Nachricht wird solange versendet, bis mindestens eine Bestätigung erhalten wurde. Das **Ende** wird wie der Anfang durch eine definierte Bit-Reihenfolge gekennzeichnet.

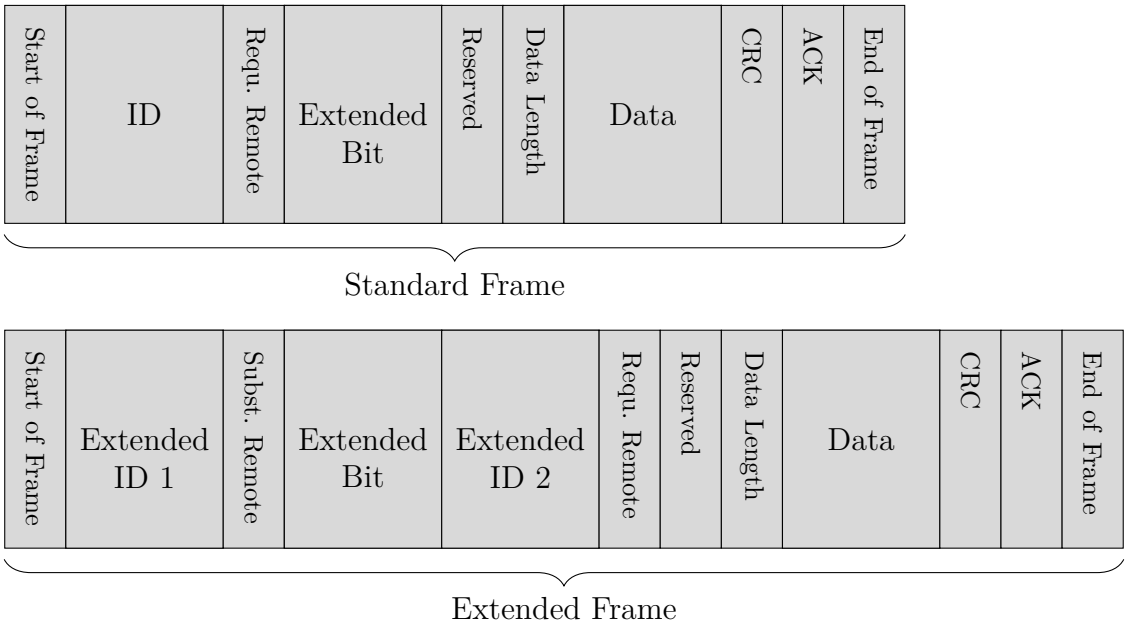


Abbildung 2.1: Standard und Extended CAN Frame,
Darstellung nach [Kum15]

GPS: Global Positioning System (GPS) ist ein Ortungssystem welches Zeitstempel von in der Nähe befindlichen Satelliten vergleicht und darüber die Lage des Gerätes bestimmt. Die Genauigkeit ist sehr von den in der nähe befindlichen Satelliten und des Sensors und seiner Abschirmung abhängig.
Die Satelliten sind untereinander synchronisiert und versenden zyklisch ihre aktuelle Zeit und Position. Mit diesen Informationen kann ein Gerät seine momentane Position errechnen, sobald es die Signale von mindestens vier Satelliten empfängt (vgl. [ITW17a]). Eine genaue Ortung ist theoretisch durch drei Satelliten zu reichen, jedoch wird zum Ausgleichen von Zeitdifferenzen ein vierter hinzugezogen.

Drucktaster: Die betrachteten Geräte verfügen beide über Drucktaster zum manuellen Neustarten. Dieses Signal wird jedoch nicht von der Plattformebene einer Applikation zugänglich gemacht und ist allein deshalb nicht für den Export relevant. Der Aufbau der Plattform wird in Kapitel 2.2.2 betrachtet. Zudem reicht die Zeit nach dem Drücken nicht aus, um die Information zu exportieren, da der Neustart umgehend ausgeführt wird.

Modem: Unter Modem fallen im Sinne der Sensoren weitergehend Zustandsinformationen wie Verbindungsstärke oder Fehlerrate, welche unregelmäßig abgefragt werden. Die eigentlichen Daten der Kommunikation werden nicht exportiert.
Es handelt sich um ein General Packet Radio Service (GPRS) Modem, welches zur Kommunikation eine Verbindung zu einem TomTom Server herstellt. GPRS ist ein Packet gesteuertes Protokoll im Global System for Mobile Communications (GSM) Netzwerk (vgl. [HRM04]).

Motorzustand: Dieser bezeichnet eine Information über den Betriebszustand des Motors. Er kann entweder *an* oder *aus* sein und wird über ADC- und OBD-Werte detektiert. In Kapitel 4.2 wird zur Evaluation ein Algorithmus zur Erkennung dessen reimplementiert.

OBD: Ein On-Board-Diagnose (OBD) System erlaubt das Auslesen von Systeminformationen im Fahrzeug durch das Anfragen der Information über ein Bussystem. Der Standard definiert die folgenden unterstützten Protokolle:

- ISO 9141-2,
- PWM J1850,
- VPW J1850,
- CAN Bus.

Es ist ein weltweit verbreiteter Standard zur Abfrage von Fahrzeuginformationen wie der Geschwindigkeit, Reifenumdrehungen oder der Temperatur. Die abfragbaren Informationen streuen sich je nach Autotyp und eingebauten Sensoren (vgl. [BGR12]).

Temperatur: Die Geräte verfügen über einen externen Temperatursensor, welcher jedoch noch nicht verwendet wird. Aktuelle Temperaturwerte werden bei Bedarf über den ADC Sensor bezogen. Dieser greift unter anderem einen im Prozessor integrierten Temperatursensor mit ab.

Zündung: Die Zündung ist ähnlich dem Motorzustand, eine Information über den Zustand der Zündung des Fahrzeuges. Diese kann die Zustände *an* oder *aus* annehmen. Die Detektion erfolgt entweder über einen digitalen Eingangs-Pin oder Informationen vom CAN Bus des Fahrzeugs.

2.2 Plattform und Geräte

In diesem Kapitel werden die verwendeten Geräte zur Aufnahme der Sensordaten betrachtet und die bestehende Software-Grundlage erklärt.

2.2.1 Geräte

Als Geräte werden die von TomTom entwickelten Produkte betrachtet, welche weitergehend unter der Bezeichnung

- TomTom Gerät OBD (TG-O) und
- TomTom Gerät CAN (TG-C)

geführt werden. Eine Gesamtübersicht der Unterschiede wird in Tabelle 2.2 gezeigt. Die Hardware wurde für diese Arbeit vorgegeben, deshalb entfällt eine Analyse weiterer Systeme.

	TG-O	**TG-C**
Takt	96 MHz	120 MHz
RAM	128 KiB	512 KiB
ROM	2048 KiB	2048 KiB
Flash	8 MiB	16 MiB
Beschleunigungssensor	✓	✓
ADC	4 Kanäle	2 Kanäle
Bluetooth	✓	-
CAN	-	✓
GPS	-	✓
Modem	-	GPRS
OBD	✓	-

Tabelle 2.2: Geräte Übersicht

TG-O: Das TG-O ist ein OBD-Dongle[3], welcher über eine Bluetooth Verbindung und eine Universal Serial Bus (USB) Schnittstelle verfügt. Es hat eine geringere Leistung hinsichtlich der Prozessorgeschwindigkeit und des Speicherplatzes als das TG-C. Ebenfalls verfügt es über keinen Anschluss zum CAN Bus. Per Bluetooth erhält es eine Verbindung zu Webfleet.

[3]Gerät wird als Aufstecker mit der OBD Schnittstelle des Fahrzeuges verbunden.

TG-C: Das TG-C wird an den CAN-Bus des Fahrzeuges angeschlossen. Dies kann über eine direkte Verbindung oder eine indirekte per Kabelklammer[4] erfolgen. Es verfügt über eine USB-Schnittstelle, ein GPRS-Modem und einen CAN-Anschluss.

2.2.2 Plattform

Beide Geräte verwenden das gleiche Basissystem, fortlaufend als Plattform bezeichnet. Abbildung 2.2 zeigt den grundlegenden Aufbau der Software. Einzelne Unterschiede in der Peripherie werden bei der Kompilierung berücksichtigt und haben lediglich Auswirkungen auf die *Plattform-Ebene*. Diese Ebene definiert sich durch verschiedene Treiber und Dienste, welche das Abgreifen der Daten von der Hardware koordinieren. Hierbei ist die Menge an Treibern, die ein Dienst verwendet, nicht beschränkt.

Ein *Treiber* greift die Informationen direkt von der Hardware ab und reicht sie einem *Dienst* weiter nach „oben“. Das heißt er bietet eine Schnittstelle, um die Daten abzufragen.
Dienste bereiten die Daten auf und stellen diese der *Plattform Abstraktionsschicht* zur Verfügung.
Applikationen greifen die Informationen von Diensten indirekt durch die *Plattform Abstraktionsschicht* ab oder reagieren auf Veränderung dieser.

Für die Koordinierung der Dienste sorgt das Betriebssystem. Es handelt sich um ein *Echtzeitsystem*. Das bedeutet, dass es eine deterministische Zeit zum Bearbeiten eines Problems garantiert (vgl. [Kan10]).
Das Betriebssystem kümmert sich um die Reihenfolge der laufenden Prozesse und vergibt die Systemzeit anhand von Prioritäten. Systemzeit bedeutet hierbei, dass der Prozess bearbeitet wird. Im betrachteten System kann immer nur ein Prozess zur selben Zeit bearbeitet werden.
Ein Prozess darf so lange arbeiten, bis dieser kenntlich macht, dass er auf ein Ereignis wartet, beispielsweise auf das Setzen eines Flags oder bis der Prozess mit seinen Rechenschritten am Ende ist und beendet werden kann. Ein Flag ist hierbei ein besonderer Speicherbereich mit einem Wert. Bei Änderung des Wertes wird jedem Prozess, der darauf wartet, wieder nacheinander Systemzeit gegeben, um das Ereignis abzuarbeiten.
Sollten Prozesse die gleiche Priorität haben, wird durch das *Round Robin-Verfahren* immer abwechselnd Systemzeit für einen bestimmten zeitlichen Intervall zur Verfügung gestellt (vgl. [Mat09]). Sobald die Zeitgrenze erreicht wurde, wird der aktuelle Prozess unterbrochen und die Systemzeit geht an den nächsten Prozess mit der gleichen Priorität über. Dies wechselt solange durch, bis alle Prozesse beendet sind oder ein Warten auf Ereignisse kenntlich machen.

[4] Kabelaufsatz zum Mitschneiden, der darüber laufenden Kommunikation

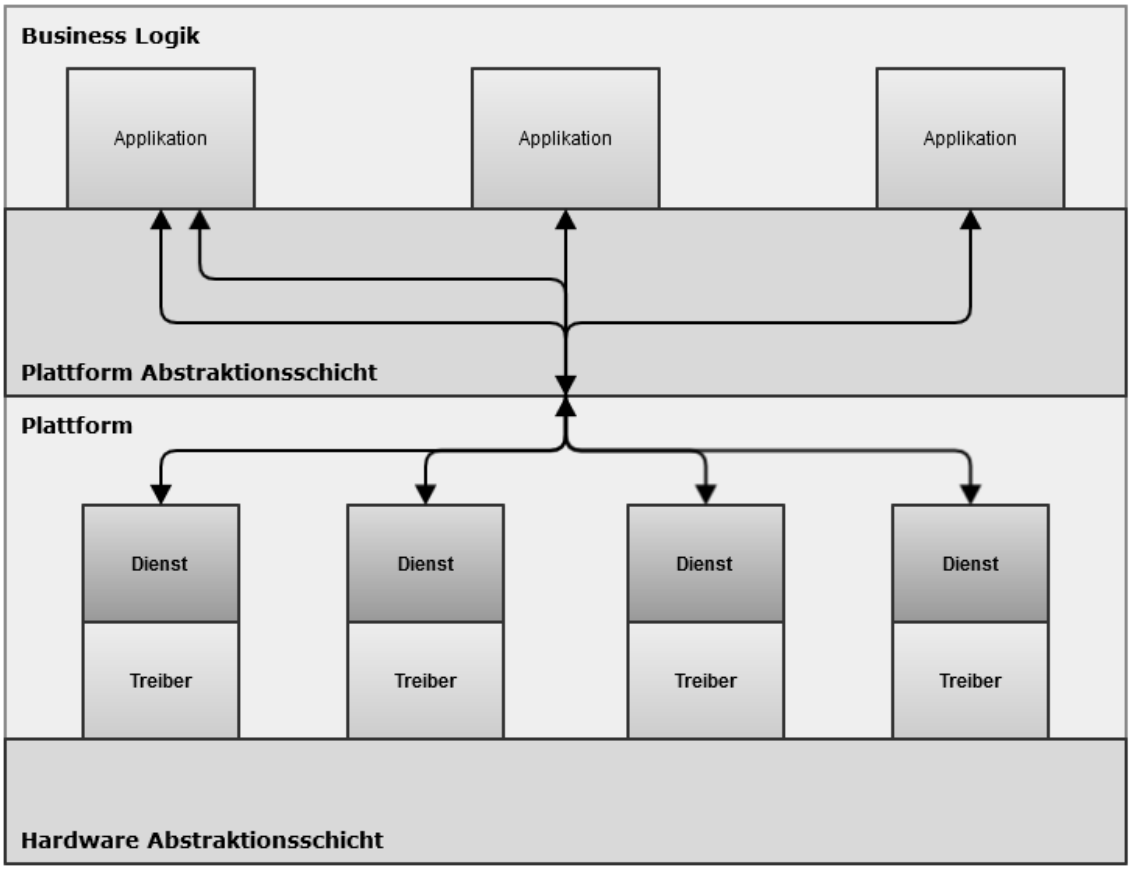


Abbildung 2.2: Plattform Übersicht,
Darstellung nach TomTom Telematics

Sollten zu viele höher priorisierte Prozesse Systemzeit brauchen, kann es bei nieder priorisierten Prozessen zu einem so genannten *Verhungern* kommen. Dies bedeutet, dass der Prozess keine Systemzeit erhält und dadurch möglicherweise für ihn relevante Ereignisse nicht verarbeiten kann. Besonders bei einem Fehlverhalten des Systems kann dies vorkommen, wenn ein Prozess in einer Berechnung feststeckt und die Systemzeit nicht mehr freigibt. Dadurch kann das gesamte Gerät nicht mehr reagieren. Für solche Fälle existiert ein *Watchdog*, ein von der Software unabhängig rückwärts zählender *Timer*. Sollte dieser den Wert 0 erreichen, wird automatisch das Gerät neu gestartet. Die Software stellt den Wert vom *Watchdog* immer auf einen ausreichend hohen Wert um die gesamte geplante Bearbeitungszeit bis zum nächsten Setzen des Wertes zu überdecken. Dadurch kann sichergestellt werden, dass zum einen nie ein Prozess ewig Systemzeit behält und zum anderen, dass das System ein gewisses Maß an Auslastung nicht überschreitet. Besonders für leistungsschwächere Geräte wie den TG-O ist die Einhaltung von minimalen Prozesszeiten wichtig.

2.3 Kommunikationsweg

Zur externen Speicherung müssen die Sensordaten über eine externe Verbindung transportiert werden. Dies muss in einer höheren Geschwindigkeit geschehen, als neue Sensordaten anfallen. Die logische Mindestgeschwindigkeit der Verbindung muss die größtmögliche Geschwindigkeit aller Sensoren übertreffen, um den Anforderungen gerecht zu werden. Da der CAN Sensor deutlich mehr Daten erbringt als die anderen, kann davon ausgegangen werden, dass ein Vielfaches seiner maximalen Geschwindigkeit ausreichend ist.
Die in Kapitel 2.2.2 beschriebenen Geräte gleichen sich in einem von vier möglichen externen Anschlüssen, die zum Exportieren der Sensordaten genutzt werden können. Nachfolgend werden alle Schnittstellen in den Geräten betrachtet.

CAN: CAN erlaubt, wie in Kapitel 2.1 beschrieben, Übertragungsraten bis zu 1Mbit/s und der TG-C verfügt über eine solche Schnittstelle. Doch abgesehen davon, dass der TG-C keine Schreibvorgänge auf den Bus tätigen kann, wird CAN als Sensor verwendet und eine Belastung durch einen Datenexport kann den Bus blockieren und das Fahrzeug damit in seiner Funktionalität behindern.

Bluetooth: Bluetooth ist ein kabelloses Verfahren zur Vernetzung einzelner Geräte untereinander in einer direkten Verbindung oder in Gruppen (vgl. [Haa98]). Das TG-O nutzt dies zur Verbindung mit Webfleet. Der verbaute Chip erreicht eine maximale Übertragungsgeschwindigkeit von 24 Mbit/s und ist damit die schnellste untersuchte Schnittstelle. Da das TG-C jedoch keine Bluetooth Verbindungen unterstützt und die Geschwindigkeit der USB Schnittstelle ausreicht, wird eine Geräte einheitliche Schnittstelle bevorzugt.

OBD: Die Schnittstelle ist Fahrzeug-abhängig und in Kapitel 2.1 beschrieben. OBD kann unter anderem auch über den CAN Bus laufen und in diesem Fall greifen die selben Gründe wie für eine direkte CAN Verbindung.

USB: USB ist eine gebräuchliche Verbindung, die weiträumig in Mobiltelefonen und Computern eingesetzt wird. Es ist ein weltweit einheitlicher Standard und jedes der betrachteten Geräte bietet diese Schnittstelle. Die maximale Übertragungsrate der betrachteten Schnittstellen beträgt 12 MiB/s, wobei diese abhängig von der Version der Schnittstelle ist. Bei den betrachteten Geräten wird eine USB 2.0 *Full Speed* Verbindung verwendet. Die Datenrate stellt ein Vielfaches der Geschwindigkeit des schnellsten Sensors dar und reicht demnach theoretisch aus um alle Sensoren im zeitlichen Rahmen zu exportieren. Neben *Full Speed* existieren noch *Low Speed*, *Hi-Speed*, *Super Speed* und *Super Speed+* als Namen für definierte Datenraten.

USB ist ein Bussystem bei dem der Host die einzelnen Endpunkte zyklisch anfragt, ob sie Daten senden möchten. Dadurch wird eine Kollision an Informationen verhindert und eine Priorisierung kann vom Host vorgenommen werden (vgl. [And97]).

Der Standard definiert vier verschiedene Übertragungsarten die ein Gerät nutzen kann.

- Control Transfers,
- Interrupt Transfers,
- Isochrone Transfers und
- Bulk Transfers.

Der **Control Transfer** wird für das Versenden von Statusinformationen und Anweisungen verwendet, um Geräte zu konfigurieren und beispielsweise eine Gerätebeschreibung zu erhalten. Durch dieses Prinzip ist ein Hinzufügen und Entfernen von Geräten wärend der Laufzeit zu erkennen.
Für zeitkritische Daten wie Audio- oder Video-Informationen sollte ein **isochrone Transfer** verwendet werden. Dieser sendet zyklisch und zusammenhängend die Daten über die Leitung. Die Größe der Datenpakete wird durch das Gerät selber bestimmt und kann lediglich von der Bandbreite noch eingeschränkt werden. Der Standard garantiert eine Bandbreite für die Daten und die Übertragung erfolgt unidirektional. Die Informationen werden durch einen CRC verifiziert, jedoch ist kein erneutes Senden von fehlerhaften Daten garantiert und führt in diesem Fall zu einem Datenverlust.
Mit **Interrupt Transfers** wird dem Hostsystem eine Notwendigkeit der Kommunikation signalisiert. Bis das Hostsystem die Daten abfragt, bleibt die Signalisierung bestehen.
Um große Mengen zeitunkritischer Daten zu transportieren ist der **Bulk Transfer** ausgelegt. Datenpakete können hierbei in großen Blöcken versendet werden, die im Falle von *Full Speed* eine Länge von 8, 16, 32, 64 oder bei *High Speed* 512 Bytes haben. Der Übertragungsart ist jedoch keine Bandbreite garantiert.

Die Bandbreite wird vorrangig für Isochrone und Interrupt Transfers verwendet. Ungefähr 10% werden für Control Transfers frei gehalten und alles Übrige wird für Bulk Transfers genutzt (vgl. [Gar+96]).

2.4 Kommunikationsprotokolle

Die Kommunikation mit einem Endgerät erfolgt durch ein *Verbindungsprotokoll* zur Authentifizierung und ein *Serviceprotokoll* zum Datenexport.

Die Geräte unterstützen mehrere verschiedene Protokolle zur Kommunikation mit Webfleet oder per USB verbundenen Endgeräten. Es werden jedoch lediglich die notwendigen Protokolle zum herstellen einer Verbindung über USB betrachtet, da dies die verwendete Schnittstelle ist.

2.4.1 Verbindungsprotokoll

Die Software hat zur Kommunikation über verschiedene Schnittstellen ein einheitliches Protokoll, um eine Steuerung von extern zu ermöglichen. Dies wird beispielsweise von Aktualisierungsprogrammen genutzt.
Die mögliche Nutzung verschiedener Funktionen wird über einen Zugangslevel bei der Anmeldung am Gerät bestimmt. Ab einem bestimmten Zugangslevel wird eine Authentifizierung benötigt, welche über einen Sicherheits-Dongle[5] ermöglicht wird.

Die Verbindung ist verbindungsorientiert und beendet sich automatisch nach einem Zeitintervall bei Inaktivität. Um die Verbindung aufrecht zu halten ist daher eine *Keep-Alive Nachricht* zyklisch zu senden. Zur frühzeitigen Abmeldung muss explizit eine *Logout-Nachricht* versendet oder die physische Verbindung getrennt werden.

Der Aufbau einer Nachricht basiert auf einem Binärprotokoll und ist sehr generisch gehalten. Grundsätzlich besitzt jede Nachricht eine eindeutige Nummer zur Unterscheidung des Inhaltes und eine dynamische Anzahl an Attributen. Ein Attribut bezeichnet hierbei einen Datentypen und seinen Wert.

2.4.2 Serviceprotokoll

Das Serviceprotokoll wird nur zu Entwicklungszwecken verwendet. Zur Zuordnung existieren wie beim Verbindungsprotokoll Identifikationsnummern. Zudem wird die Nachrichtenlänge im **Header** definiert. Der weitere Aufbau besteht aus einer dynamischen Anzahl an **statischen Blöcken**. Jeder statische Block verweist auf einen **dynamischen Block**. Es existieren somit immer mindestens genau so viele statische, wie dynamische Blöcke. Der Aufbau wird in Abbildung 2.3 dargestellt.

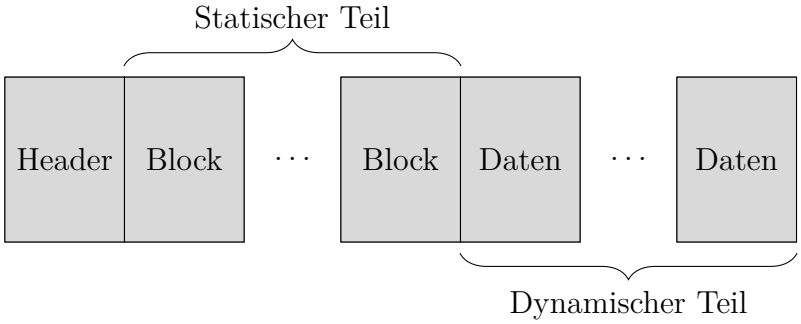


Abbildung 2.3: Serviceprotokoll Nachricht,
Darstellung nach TomTom Telematics

Statische Blöcke haben alle dieselbe Größe und denselben Aufbau innerhalb einer Nachricht. Dynamische Blöcke haben keinen definierten Inhalt und eine variable

[5] Über USB angeschlossener Kopierschutzstecker

Länge. Die genaue Position und die Menge an Daten eines dynamischen Blocks ist immer in einem statischen Block vermerkt.

2.5 Speicherformat

Die exportierten Daten werden für eine nachträgliche Verarbeitung in einem dafür geeigneten Format gespeichert. Die Werte sollen logisch geordnet und effizient abrufbar sein. Die Integrität der Daten muss erhalten bleiben und möglichst wenig Speicherplatz verbrauchen.
Folgende Informationen müssen darin platzierbar sein:

- Sensordaten,
- Sensorzeitinformationen,
- Metainformationen und
  - Geräteinformationen,
  - Aufnahmezeit,
  - Laufzeitinformationen und
  - Bytereihenfolge.
- Kommentare / Annotationen.

Die **Sensordaten** bezeichnen die vom Gerät erfassten internen Informationen, welche von den Applikationen verwendet werden. Die Erfassungszeit dieser soll als **Sensorzeitinformation** gespeichert werden.
Metainformationen dienen der nachträglichen Einordnung der gespeicherten Daten. Diese beinhalten unter anderem die aufgezeichneten Sensortypen, eine Gerätebezeichnung, die Zeit der Aufnahme, das verwendete Betriebssystem, eine Schnittstellenbezeichnung zum Gerät und die Bytereihenfolge der Sensordaten.
Besonders wichtig ist eine **Annotations- und Kommentarunterstützung**, damit besondere externe Zustände vermerkt und eine signifikante Veränderung der Sensorwerte in Verbindung gebracht werden kann.

In Tabelle 2.3 wird eine positive (+1), negative (-1) oder neutrale (0) Wertung der betrachteten Speicherformate gegeben, um die einzelnen Punkte zu bewerten.
Ein eindeutig **definiertes Format** steht nur bei Packet Capture Next Generation (PCAPNG) Daten bereit, da in den anderen Fällen die genaue Struktur noch selbst bestimmt werden muss. Im Falle einer Datenbank wird hierbei zumindest ein Modell für das Beschreiben der Struktur vorgegeben (Tabellen, Graphen, …), welche je nach Datenbank Typ unterschiedlich ausfallen kann.
Die **Abrufbarkeit** wird bei Datenbanken mit einer Structured Query Language (SQL) Abfrage leicht umgesetzt und bei PCAPNG Dateien ist zumindest der Aufbau der Blöcke klar definiert.
Die **Integrität** der Daten wird bei Datenbanken und dem PCAPNG-Format durch verschiedene Mechanismen wie der *Magic Number* oder Prüfsummen sichergestellt.
Der **Speicherverbrauch** ist lediglich bei einer Eigenimplementierung so gering wie möglich zu halten. Datenbanken sind stark von der verwendeten Struktur abhän-

gig, wie viel Speicherplatz sie für die Informationen benötigen. PCAPNG hat nach Standard mehr Felder für Metainformationen, als in diesem Anwendungsfall benötigt werden.
Kommentare sind in einer Eigenimplementierung so detailliert wie notwendig einzupflegen und im PCAPNG-Format kann jeder Block, jedes Interface und jede Datei einzelne Kommentare bekommen. Bei Datenbanken muss das Kommentieren in der Struktur selber mit berücksichtigt werden.

	Eigen-implementierung	Datenbank	PCAPNG
Definiertes Format	-1	0	+1
Abrufbarkeit	-1	+1	0
Integrität	0	+1	+1
Speicherverbrauch	+1	0	0
Kommentare	+1	0	+1
Gesamt	0	2	3

Tabelle 2.3: Wertung der Speicherformate

2.5.1 Eigenimplementierung

Ein speziell für diesen Anwendungsfall entworfenes Format hat als Vorteil, dass alle benötigten Kritikpunkte erfüllt werden können und die Größendimension optimal angepasst werden kann.
Als Nachteil ist der enorme Aufwand an Planung und Optimierung gegenüber gestellt. Zudem muss eine ausführliche Dokumentation existieren, um eine Weiterverwendung von Dritten zu ermöglichen.
Eine Betrachtung der Daten ist nur mit einer explizit dafür ausgelegten Software möglich.
Der Aufwand wird als zu umfangreich angesehen und ein bereits existierendes Format bevorzugt um auch eine längerfristige Verwendungsmöglichkeit zu garantieren.

2.5.2 Datenbank

Eine Datenbank erlaubt das geordnete Speichern aller Informationen und ermöglicht eine einfache Erweiterung des Formates. Sie besteht klassischerweise aus verschiedenen Tabellen mit untereinander verknüpften Informationen. Je nach Datenbankmodell kann sich das Konzept der Tabellen durch ein anderes, beispielsweise graphenbasiertes Modell, unterscheiden. Die betrachteten Kritikpunkte wurden unabhängig vom Datenbankmodell betrachtet.

Die Speicherung kann je nach verwendeter Datenbank abweichen. Um eine leichte Portabilität zu gewährleisten wird eine *single file* Funktionalität wie bei SQLite bevorzugt (vgl. [Kre10]). Diese erlaubt es alle nötigen Informationen zur Struktur der Daten und die Daten selbst in einer einzigen Datei zu speichern. Es sind jedoch immer mehr Informationen für die logischen Zusammenhänge der Daten untereinander nötig, wie eine Indizierung der Einträge oder Verweise auf andere Tabellen.
Eine Datenvisualisierung muss zur besseren Verständlichkeit über eine spezielle Software mit integrierter Datenbanklogik erfolgen.

2.5.3 PCAPNG

PCAPNG ist der sich in Entwicklung befindliche Nachfolger des *Packet Capture (PCAP)* Dateiformats (vgl. [Gro17]). Es dient zur Speicherung von Netzwerkpaketen und erlaubt die Kommentierung der gesamten Datei und einzelner Pakete.

Gegenüber dem Vorgänger wurde das Format unter anderem um folgende Punkte erweitert:

- Nutzer-spezifische Kommentare,
- hochauflösende Zeitstempel im Nanosekundenbereich,
- Schnittstellen-Informationen und
- Paketverlust-Zähler.

Jede gültige *PCAPNG* Datei beginnt mit der *magischen Zahl*[6] *0x0A0D0D0A*. Diese Zahl stellt ein Palindrom[7] dar, um zusätzlich zur Datentypenerkennung eine Veränderung der Byte-Reihenfolge nach Übertragungen feststellen zu können.

Der Aufbau besteht aus aneinander gereihten Blöcken, die zur Kennzeichnung von Metainformationen und für die Daten selbst verwendet werden. Insgesamt existieren vierzehn verschiedene Blöcke, wobei sieben davon experimentell sind und noch nicht zwangsweise im endgültigen Standard sein werden.
Die für den Datenexport relevanten Datenblöcke werden nachfolgend genauer betrachtet.

Section Header Block: Der *Section Header Block* beschreibt eine Aufnahmereihe von mehreren weiteren Blöcken und steht immer am Anfang einer solchen. Er selber beinhaltet keine Daten, sondern beschreibt die folgenden Blöcke.
Die genaue Struktur wird in Abbildung 2.4 dargestellt.

[6] Eine magische Zahl bezeichnet die ersten Bytes einer Datei, die dazu genutzt werden können den Dateityp zu bestimmen. Die Anzahl der Bytes ist Datei-spezifisch.
[7] Vorwärts wie rückwärts gelesen, derselbe Wert

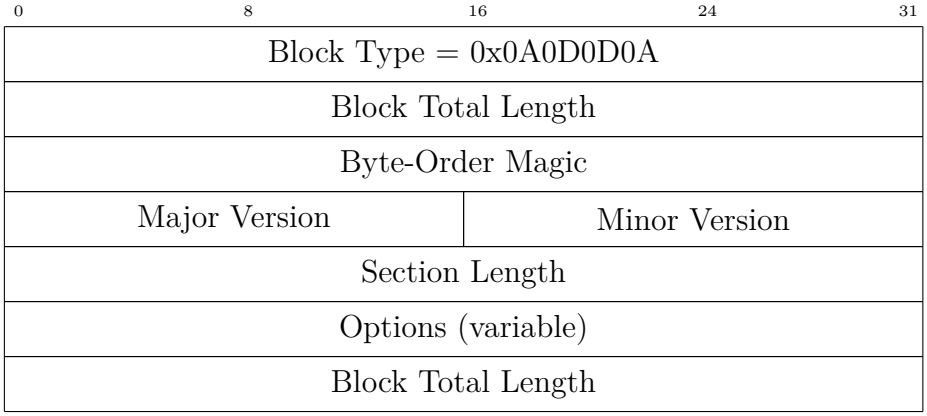


Abbildung 2.4: Section Header Block,
Darstellung nach [Gro17]

Der **Block Type** ist die oben genannte *Magic Number* und dient zur Identifizierung der Datei als PCAPNG-Datei.
Die **Blocklänge** begrenzt den Anfang und das Ende des Blocks, um die Integrität dessen in seiner Größe kontrollieren zu können.
Eine weitere **Magic Byte Order** existiert innerhalb, um die Bytereihenfolge der Daten darzustellen. Unterschieden wird hierbei zwischen *little-endian* und *big-endian*.
Zur Versionierung existiert ein Bereich für eine **Major**- und **Minor**-Versionsnummer, um wechselnde Strukturen in den Blöcken anzuzeigen.
Zum erleichterten Parsen der Datei hat der Abschnitt eine **Section Length**, damit bei Bedarf der Bereich schnell übersprungen werden kann. Die Länge kann unter Umständen auch negativ sein, um eine ungewisse Länge anzuzeigen.
Wie in jedem Block ist ein extra Platz für dynamisch lange **Optionen** gelassen, um Metainformationen zur Datei oder dem Abschnitt zu speichern.

Interface Description Block: Der *Interface Description Block* existiert zur Beschreibung der Schnittstelle, über welche die Daten erhoben wurden. Sein Aufbau wird in Abbildung 2.5 dargestellt.

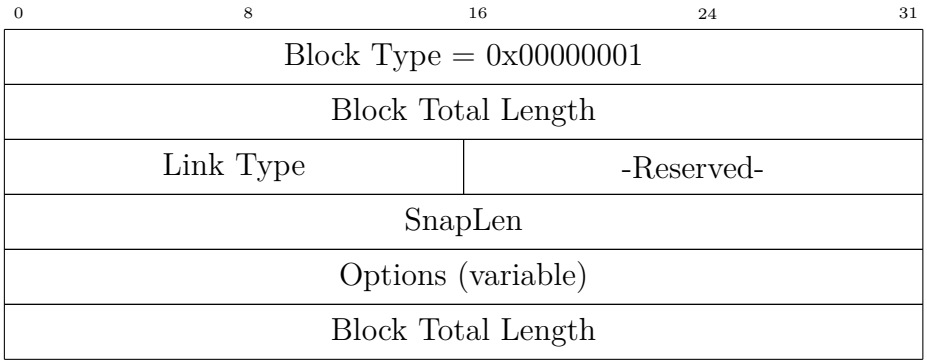


Abbildung 2.5: Interface Description Block,
Darstellung nach [Gro17]

Der **Block Type** dient zur Erkennung als *Interface Description Block*.
Der **Link Type** beschreibt einen Datentypen auf den sich einzelne Blöcke mit Daten beziehen können. Hierbei wird vom Standard auf eine Liste mehrerer definierter Typen verwiesen. Um einen eigenen Datentypen zu kennzeichnen wurde ein Bereich von 147 bis 162 freigehalten.
Der **reservierte** Bereich ist für eine zukünftige Verwendung definiert.
Wenn aufzunehmende Daten zu lang sein könnten, kann im Feld **SnapLen** eine maximale Datenlänge angegeben und dadurch ein abschneiden von zu langen Daten gekennzeichnet werden. Bei einer Länge von 0 wird keine obere Grenze gesetzt.

Enhanced Packet Block: Die eigentlichen Daten werden in *Enhanced Packet Blöcken* gespeichert und sind wie in Abbildung 2.6 dargestellt, aufgebaut.

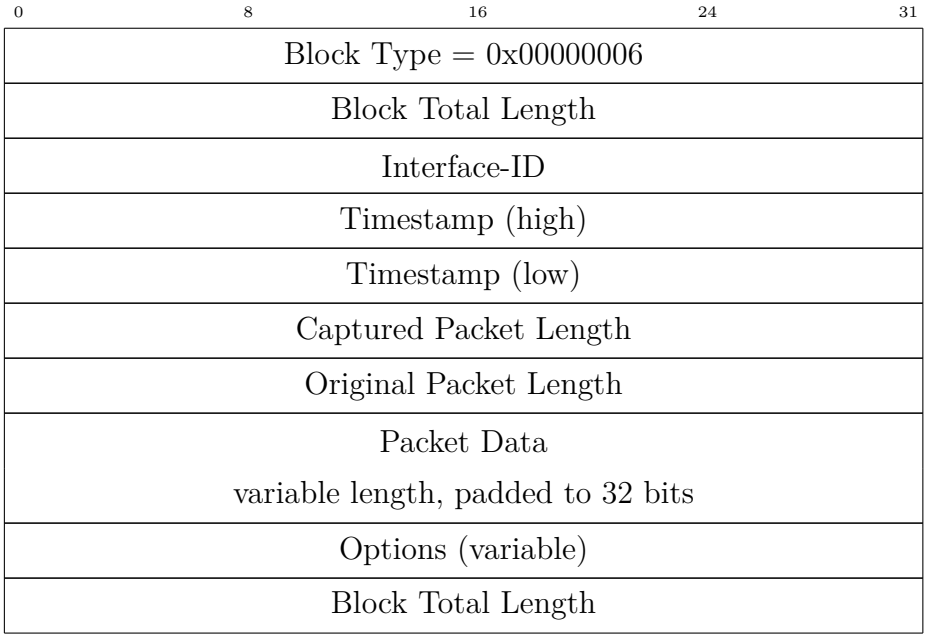


Abbildung 2.6: Enhanced Packet Block,
Darstellung nach [Gro17]

Der **Block Type** kennzeichnet einen *Enhanced Packet Block*.
Die **Interface-ID** stellt einen Verweis zu einem *Interface Description Block* her, um eine mögliche Redundanz an Typenbeschreibungen zu optimieren.
Da die Blöcke alle eine exakte Breite von 32 Bit haben, wird der 64 Bit große **Zeitstempel** in einen oberen und einen unteren Teil aufgespalten.
Die **Captured Packet Length** bildet zusammen mit der **Original Packet Length** eine Soll- und Ist-Wert Vergleichsmöglichkeit. Hierbei gibt die *Original Packet Length*, die zu erwartende Datenlänge an und die *Captured Packet Length* stellt die tatsächliche Länge der Daten dar. Somit kann ein Datenverlust festgestellt und gekennzeichnet werden.
Unter **Packet Data** sind die eigentlichen gespeicherten Daten zu finden. Sie werden immer in 32 Bit Teile aufgespalten und wenn notwendig bis zu einem vollen Block mit Nullen aufgefüllt.

Weitere Blöcke, wie beispielsweise der *Simple Packet Block*, der zur Abwärtskompatibilität dient oder experimentelle Blöcke werden nicht weiter betrachtet, da durch die bisher beschriebenen Blöcke alle Anforderungen an das Datenformat erfüllt werden.

Optionen: Jeder Block hat ein optionales Feld für Optionen, welche wie eine Liste nacheinander definiert werden und immer mit einer Option der Länge 0 und dem *Option Code opt_endofopt* endet, welcher ebenfalls dem Wert 0 entspricht. Somit hat jeder Block mindestens eine Option, welche das Ende der Optionsliste darstellt. Der Aufbau von Optionen wird in Abbildung 2.7 dargestellt.

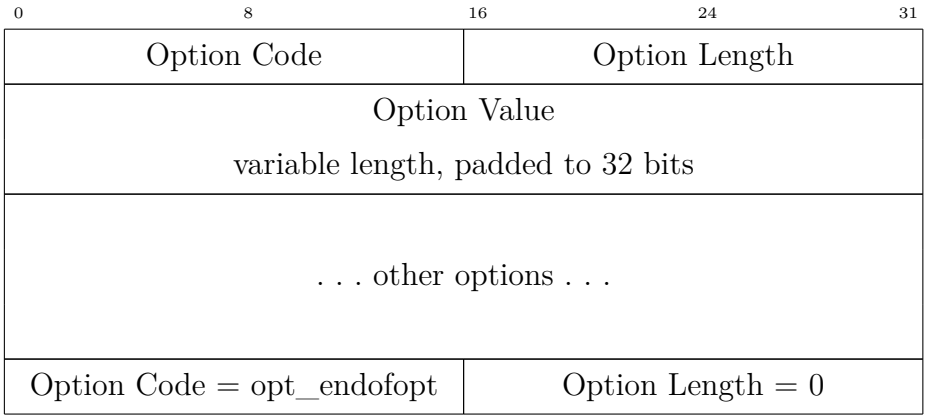


Abbildung 2.7: Optionen Formatierung,
Darstellung nach [Gro17]

Name	**Code**	**Beschreibung**	**Blöcke**
opt_endofopt	0	Ende der Optionen	Alle
opt_comment	1	Kommentar	Alle
shb_hardware	2	Verwendete Hardware zur Aufnahme	Section Header Block
shb_os	3	Verwendetes Betriebssystem zur Aufnahme	Section Header Block
if_name	2	Name des Aufnahmegerätes	Interface Block
if_description	3	Beschreibung des Aufnahmegerätes	Interface Block
if_tzone	10	Zeitzone des Interfaces	Interface Block
if_os	12	Betriebssystem des Interfaces	Interface Block

Tabelle 2.4: Ausgewählte Optionen Übersicht,
Darstellung nach [Gro17]

Der **Option Code** steht für einen benutzerdefinierten Kommentar oder eine im Standard definierte Option. Ein Auszug aus den möglichen Werten dafür wird in Tabelle 2.4 gezeigt.
Die **Option Length** gibt die dynamische Länge einer Option an und der **Option Value** ist der Kommentar. Dieser wird wie die Paketdaten auf 32 Bit Teile aufgefüllt.

2.6 Sicherheit der Kommunikation

Ein wichtiger Punkt in Produkten mit kritischen Nutzerinformationen ist die Absicherung der Zugriffe vor Unbefugten. Als mögliche Angriffspunkte werden die Schnittstellen der Geräte gesehen und der Verbindungsaufbau derer beschrieben.

2.6.1 Controller Area Network

Über den CAN Bus werden Daten vom Gerät erfasst und verarbeitet. Das TG-C hängt lesend am CAN Bus und kann von sich aus keine Informationen auf den Bus schreiben. CAN unterstützt in seinem Protokoll keine Authentifizierung und jedes Endgerät mit schreibenden Zugriff auf den Bus hat auch die volle Kontrolle über diesen (vgl. [WWP04]). Über CAN Daten können keine Befehle an das Gerät geschickt werden und fehlerhafte Informationen führen maximal zu einer Verwirrung der Applikationslogik.

2.6.2 On-Board-Diagnose

OBD hat wie die CAN Daten eine fehlende Möglichkeit Befehle an das Gerät zu senden und auch hierbei haben fehlerhafte Daten nur eine geringe Auswirkung auf die Funktionalität.

2.6.3 Bluetooth

Bluetooth wird von nur vom TG-O unterstützt und es wird ein Personal Identification Number (PIN) gestütztes *pairing Verfahren* verwendet.

2.6.4 Modem

Über GPRS werden Daten mit dem in einem abgeschotteten VPN befindlichen Webfleet Server übertragen. Der Zugangsschutz erfolgt über die eingebaute SIM-Karte und den Aufbau einer gesicherten Transport Layer Security (TLS) Verbindung. Die

Karte erlaubt lediglich eine Verbindung zu einem abgeschatteten TomTom Netzwerk und kann nicht für andere Dienste verwendet werden.

2.6.5 Universal Serial Bus

Über USB können die Geräte aktualisiert und zu Entwicklungszwecken auch weiter interne Funktionalitäten aufgerufen werden. Dies stellt eine hervorragende Angriffsfläche für potenzielle Angreifer dar, ist jedoch auch von einem direkten Kontakt zu dem Gerät abhängig. Die erste Sicherheit besteht also erst einmal in einem festen Verbau der Geräte im Fahrzeug. Um logische Anfragen zu stellen, wird ein proprietäres Kommunikationsprotokoll, welches in Kapitel 2.4.1 näher betrachtet wird, verwendet und durch mehrere Sicherheitsstufen gesichert. Auf der niedrigsten Stufe lassen sich sicherheitsirrelevante Informationen abfragen, die zu Supportzwecken dienen. Funktionen wie der Datenexport werden über eine Authentifizierung mit einem Sicherheits-Dongle geschützt. Der Zugriff auf den Dongle selbst wird über ein Passwort gesichert.

3 Implementierung

Das Kapitel Implementierung geht direkt auf die Integration in das bestehende Softwaresystem ein und beschreibt sowohl die angewandten als auch die verworfenen Konzepte und warum sie umgestellt wurden. Die Kernpunkte bilden die Integration der Funktionalität in die bestehende Software, die Verwendung des gewählten Speicherformates und die Erstellung einer Benutzeroberfläche zur Nutzung der Export-Funktionalität. Das Exportieren der Daten soll das System nicht einschränken und transparent dazu agieren. Die Daten sollen zudem vollständig und in jedem Fall fehlerfrei nach außen geleitet werden.

Es sind jene Sensordaten relevant, welche von den Diensten zur darüber liegenden Schicht bereitgestellt werden.

3.1 Export-Dienst

Der *Export-Dienst* definiert einen neuen zu implementierenden Dienst, welcher in die in Kapitel 2.2.2 beschriebene Plattform integriert werden soll. Dieser Abschnitt zeigt den theoretischen Entwurf und die veränderte Umsetzung aufgrund bestehender Eigenheiten des existierenden Systems.

3.1.1 Planung

Nach der Analyse der Plattform aus Abbildung 2.2 wurde ein erstes Konzept erstellt und implementiert. Der Aufbau und die Integration in die Plattform wird in Abbildung 3.1 gezeigt. Als Schnittstelle wurde USB gewählt, da beide Geräte diese aufweisen und die Datenrate weit über der Geschwindigkeit des schnellsten Sensors liegt.

Um genügend Abstraktion zum laufenden System zu haben, wurde ein eigener Dienst geplant, welcher die Initialisierung der benötigten Komponenten beim Systemstart übernimmt und komplett unabhängig die Sensordaten in eine Serviceprotokoll Nachricht verpackt, um sie über die USB Schnittstelle zu versenden. Diese Informationen werden am Punkt an dem die überliegende Schicht benachrichtigt wird, kopiert und einer Queue hinzugefügt. Sobald der Export-Dienst Systemzeit erhält, beginnt er mit dem Zusammenpacken der einzelnen Sensordaten aus der Queue zu einer größtmöglichen Nachricht von 8 KiB. Die maximale Größe ist hierbei begrenzt durch die Maximum Transmission Unit (MTU) der USB Verbindung, welche von dem entsprechenden Treiber bereitgestellt wird.

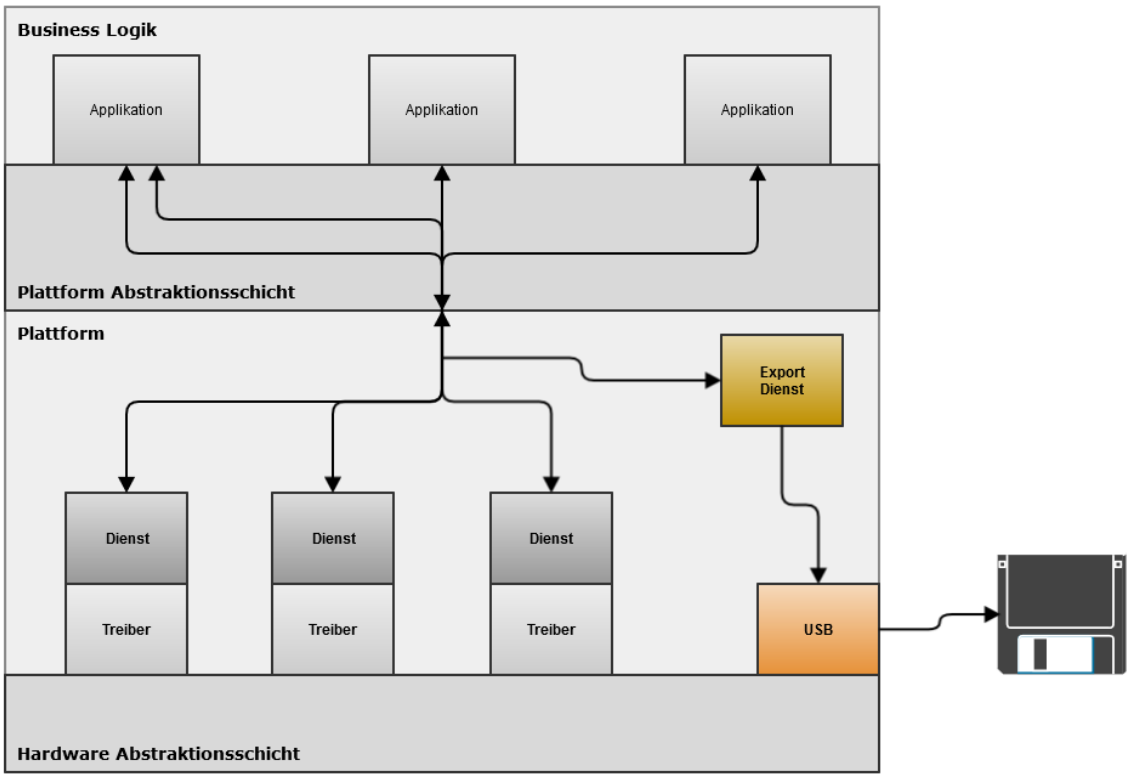


Abbildung 3.1: Theoretischer Entwurf,
Quelle: Eigene Darstellung

Alle anfallenden Sensordaten werden in einer Warteschlange gespeichert und bei erhaltener Systemzeit abgearbeitet. Als Queue dient eine im Betriebssystem integrierte Standard Datenstruktur. Sie unterstützt das automatische Setzen eines Flags bei einem hinzugefügten Element. Lediglich die Speicherverwaltung muss selbst implementiert werden. Somit kann der Export-Dienst auf ein gesetztes Flag der System Queue warten und diese bei erhaltener Systemzeit abarbeiten. Der Eingriff in das laufende System ist durch das Kopieren der Sensordaten und damit ein minimales aber notwendiges Blockieren von Systemzeit. Das System wird nur durch die Übertragung der Daten beeinträchtigt. Dies ist jedoch unumgänglich zum Exportieren.

3.1.2 Architektur-Verbesserungen

Die ersten Probleme bestehen in dem Design der Serviceprotokoll-Kommunikation. Diese wird über einen *System-Dienst* verarbeitet und ist nicht *Multitprozess*-kompatibel. Dies bedeutet, dass nur ein Dienst die Kommunikation über das Serviceprotokoll bearbeiten kann, da intern ein fester Speicherbereich für die zu versendende Nachricht verwendet wird und diese bis zur vollständigen Übertragung dort erhalten bleiben muss. Sollte die Funktion für diesen Prozess während der Abarbeitung erneut von einem anderen Dienst aufgerufen werden, würde dieser die Informationen in genanntem Speicherbereich überschreiben und beide Übertragungen beeinträchtigen.
Der Umbau und die damit verbundenen Tests für eine Mehrfach-Dienst Verwend-

barkeit stellt einen zu großen Eingriff in das existierende System dar und ist damit außerhalb der gesetzten Abgrenzungen.

Eine erste Erweiterung des geplanten Konzeptes ist eine asynchrone Verarbeitung über den *System-Dienst*. Der Export-Dienst reiht dabei das Versenden der Nachricht in eine Warteschlange ein, welche der System-Dienst abarbeitet. Sobald der *System-Dienst* das nächste mal Systemzeit erhält, packt er die Nachricht zusammen und bereitet sie zum Versenden über die USB Schnittstelle vor.
Das Versenden wird ausschließlich über einen *Nachrichten-Dienst* getätigt, welcher alle 4 Millisekunden prüft, ob eine Nachricht versendet werden muss.

Es ist durch die Architektur der Software bedingt, dass die Nachricht durch den *System-Dienst* zusammengepackt und vom *Nachrichten-Dienst* versendet werden muss. Der geplante *Export-Dienst* hätte somit die verbleibende Aufgabe alle zur Verarbeitung nötigen Datenstrukturen zu initialisieren und dem *System-Dienst* das Abarbeiten der Queue zu signalisieren.

Die restliche Funktionalität des *Export-Dienstes* ist gering genug, um sie direkt in den *System-Dienst* einzubetten, ohne dabei eine zu große Änderung am bestehenden Verhalten des Dienstes zu erfordern. Zusätzlich entfällt dabei die Benachrichtigungszeit des *Export-Dienstes* an den *System-Dienst*. Diese Architektur wird in Abbildung 3.2 dargestellt.

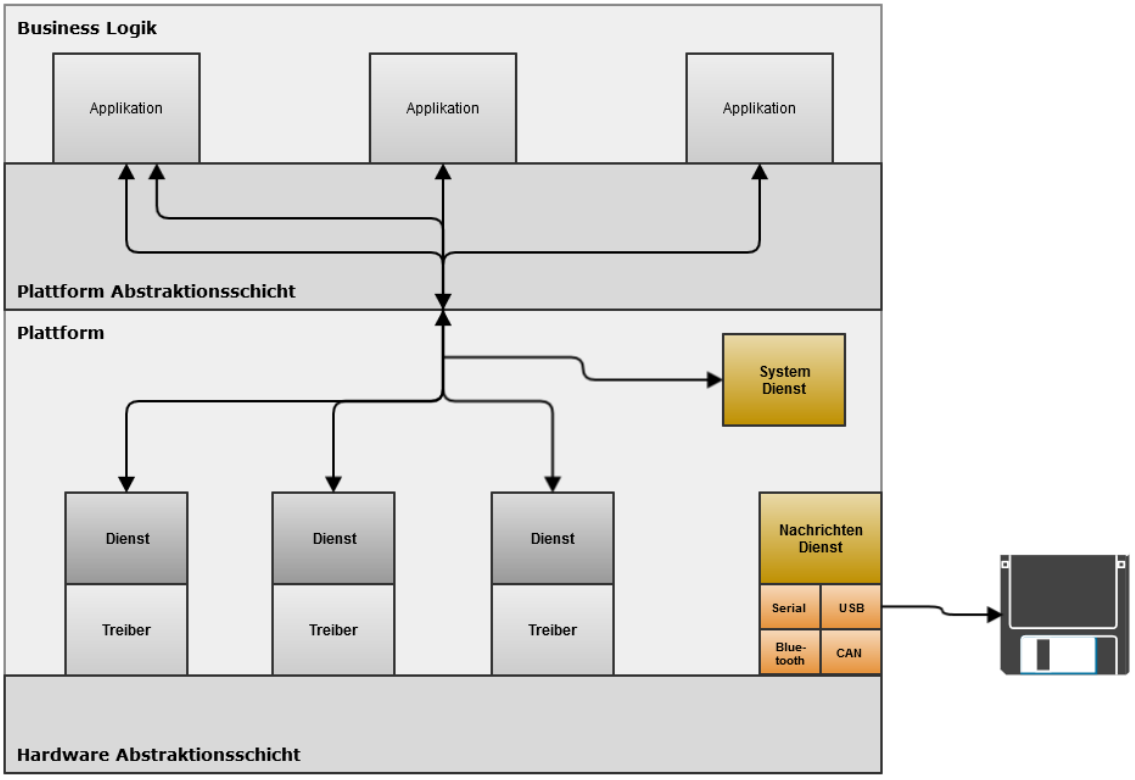


Abbildung 3.2: Praktische Umsetzung,
Quelle: Eigene Darstellung

Durch die Integration in den *System-Dienst* entfällt die Möglichkeit eine vom Betriebssystem bereitgestellte Queue zu verwenden, da der Dienst seine eigenen Flags hat auf die er reagiert und eine Verbindung von beidem zu starke Strukturänderungen bedingen würde.
Es wurden verschiedene bereits in der Plattform existierende Queues und Speicherstrukturen getestet, um eine möglichst Zeit- und Speicherplatzeffiziente Lösung zu finden. In Betrachtung waren Listen mit und ohne Speicherverwaltung, für statische und dynamische Elementlängen und Heap Implementierungen zur dynamischen Speicherallokierung. Die Wahl wurde aufgrund der bestmöglichen Umsetzungsmöglichkeit festgelegt unter Berücksichtigung einer möglichst geringen Laufzeit. Tabelle 3.1 zeigt die Betrachtung der Datenstrukturen.

	Vektor	**Ringspeicher**	**Liste**	**Heap**
Feste Reihenfolge	+1	+1	+1	-1
Feste Elementlänge	+1	+1	+1	+1
Dynamische Elementlänge	-1	-1	+1	+1
Lesezugriff	0	+1	0	+1
Schreibzugriff	+1	+1	+1	-1
Gesamt	2	3	4	1

Tabelle 3.1: Strukturen Vergleich zur Datenverwaltung

Der Aufbau der Serviceprotokollnachricht besteht aus zwei verschiedenen Teilen.
Im statischen Teil steht ein Zeitstempel für den Moment, in dem die Daten vom Gerät erhoben wurden, sowie ein Datentyp, um die Sensordaten zuzuordnen. Um den zugehörigen dynamischen Datenblock bestimmen zu können, werden die Datenlänge und der Anfangspunkt der Daten im dynamischen Teil durch Werte ausgedrückt.
Der dynamische Teil beinhaltet lediglich die Sensordaten direkt aneinandergereiht, um den Platzbedarf möglichst gering zu halten.

In der Umsetzung werden zwei Queues verwendet, welche den Nachrichtenaufbau aus Abbildung 2.3 nachbilden. Eine mit einer festen Größe der Elemente für den statischen Teil der Nachricht und eine mit dynamisch großen Elementen für den dynamischen Teil. Diese Aufspaltung ermöglicht ein effizientes Zusammensetzen der Nachricht, da die feste Elementgröße ein schnelles Springen zwischen den Einträgen erlaubt.
Die Datenstrukturen müssen die **Reihenfolge**, in der die Daten aufkommen, abbilden können und bis auf den Heap wird diese von allen unterstützt. Der Heap sucht sich die nächste passende Stelle für die Daten und die Reihenfolge muss abseits davon gespeichert werden.
Für den statischen Teil der Nachricht müssen Elemente **fester Länge** und für den dynamischen Teil Elemente **dynamischer Länge** gespeichert werden können. Der Vektor und der Ringspeicher unterstützen hierbei keine unterschiedliche Länge ihrer Elemente.

Der **Lesezugriff** beinhaltet sowohl das Auslesen von Elementen an zufälliger Position, sowie das Entfernen der ersten Position. Der Vektor muss beim Löschen des ersten Elementes alle verbleibenden verrücken, um die korrekte Reihenfolge beizubehalten. Eine Liste erlaubt nur das direkte Anspringen der ersten und der letzten Position.
Beim **Schreibzugriff** ist vor allem die Geschwindigkeit beim Einfügen eines neuen Elementes zu beachten und nahezu alle betrachteten Strukturen erlauben ein schnelles Springen an die letzte Position. Der Heap muss vorher noch die nächst freie Stelle suchen.
Damit ergibt sich der Ringspeicher für den statischen und die Liste für den dynamischen Teil der Daten.

3.1.3 Effiziente Ordnung

Um den zur Verfügung stehenden Speicherplatz auf dem Gerät besser auszunutzen und die Übertragungsgrößen der einzelnen Pakete zu minimieren, wurden bei verschiedenen Sensordaten die Informationen zusammengefasst. Das nachfolgende Beispiel behandelt diesen Ansatz bei CAN Daten.

CAN beinhaltet die hochfrequentesten Sensordaten, die im durchschnittlichen Betrieb anfallen und jedes gesparte Byte in der Übertragung kann Platz für fünf[8] weitere CAN Daten schaffen.
Auf dem Datenbus werden die Daten bitweise versendet und sind mit zusätzlichen Informationen wie Prüfsumme oder Start- und Endbit versehen, welche zur klaren Erkennung und zur Informationsintegrität notwendig sind. Im Gerät selbst werden nur Informationen des *Datenfeldes*, der *CAN-ID*, des *extended Bits*, ein *Zeitstempel* und die *Busnummer* gespeichert und verarbeitet. Die Busnummer ist nicht im CAN Protokoll vorhanden und wird im Gerät zur Unterscheidung des angeschlossenen CAN Busses verwendet.

Bei den zu exportierenden CAN Daten werden die Informationen der *CAN-ID* mit dem *extended Bit* und der *Busnummer* zusammengefasst. Dies wird in Abbildung 3.3 dargestellt. Zudem werden überflüssige Informationen wie redundante *Zeitstempel* vor dem Einreihen in die Queue entfernt. Der Aufbau einer CAN-Nachricht wird in Kapitel 2.1 beschrieben.

Als weitere Methode zur CAN Datenreduzierung werden nach der Erhebung doppelte Nachrichten gefiltert und nur veränderte Inhalte exportiert. Um die Integrität der Daten dabei nicht zu verletzen, muss eine Nachricht über die Filterung exportiert werden. Diese Nachricht braucht nur die CAN-ID beinhalten, um eine Referenz auf ihren doppelten Vorgänger zu geben. Sie würde keine Busnummer, das extended Bit oder die CAN Daten brauchen und wäre damit kleiner als ein ursprünglicher Daten-

[8] Beispielhafte Zahl unter Berücksichtigung der CAN Sensordaten Länge, der Häufigkeit der Nachrichten und der USB MTU Größe.

satz. Dadurch würde sich jedoch das Verarbeiten der exportierten Daten erschweren und eine strikte Reihenfolge der Daten wäre unumgänglich.

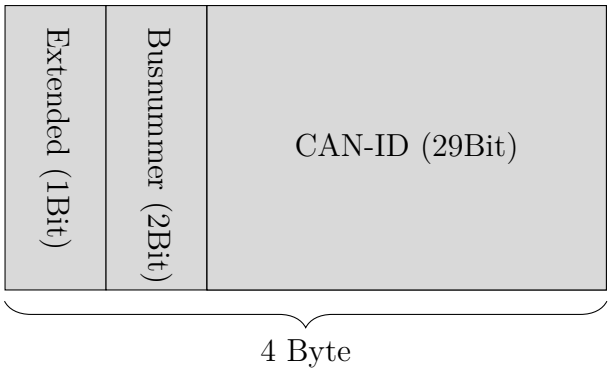


Abbildung 3.3: zusammengefasste CAN Daten,
Quelle: Eigene Darstellung

Die bereits verwendeten Optimierungen waren zur Erfüllung der zeitlichen Ansprüche ausreichend und deshalb wurde dieses Konzept nur theoretisch betrachtet und sich für eine erleichterte Verwendung der exportierten Daten entschieden.

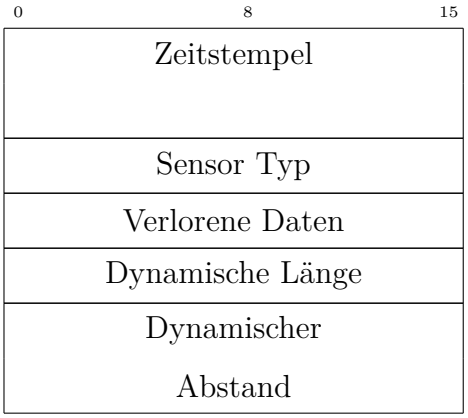


Abbildung 3.4: Statischer Teil Header,
Quelle: Eigene Darstellung

Neben der Optimierung der Sensordaten wurde an dem im Kapitel 2.4.2 betrachteten Serviceprotokoll Änderungen vorgenommen.
Jedem statischen Teil einer Nachricht wird ein vergrößerter Block, wie er in Abbildung 3.4 gezeigt wird, vorangesetzt. Da dieser Block immer am Anfang steht, wird er weitergehend als *Header* bezeichnet.
In den Feldern dieses speziellen Blockes bezieht sich der **Zeitstempel** auf die Ankunftszeit der zugehörigen Sensordaten im dynamischen Teil wie sie im Gerät erfasst worden sind. Der **Sensortyp** bezeichnet die Art der Sensordaten und die **verlorenen Daten** geben einen Datenverlust an, sollte beim Aufzeichnen kein Platz in der Queue vorhanden gewesen sein um die Sensordaten bis zum Export zu speichern. Dies tritt genau dann auf, wenn das Gerät durch blockierende Prozesse nicht zum Abarbeiten der Queue kommt.

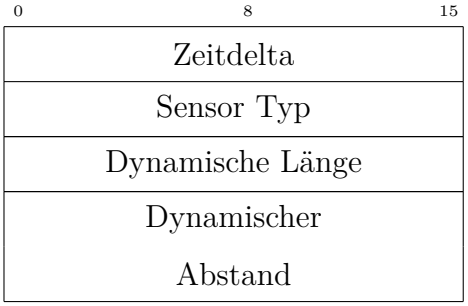


Abbildung 3.5: Statischer Teil Block,
Quelle: Eigene Darstellung

Jeder weitere Block enthält eine gekürzte Version des Headers, wie in Abbildung 3.5 dargestellt. Das **Zeitdelta** ist eine Differenzangabe zum Zeitstempel des ersten Blocks in der Nachricht und umfasst die Hälfte seiner Größe. Der Sensortyp verändert sich nicht, aber die Angabe für die verlorenen Daten fällt weg, da diese Information nur einmal pro Nachricht vorkommen muss.

Es wurde ein Speichergewinn durch Datenkompression betrachtet. Ein solches Verfahren hätte jedoch zusätzliche Rechenzeit und Speicherverbrauch in der Software in Anspruch genommen. Besonders die erhöhte Rechenzeit spielt bei diesem zeitkritischen Problem eine besondere Rolle und sollte in Bezug auf das zu erwartende Ergebnis in Vergleich gesetzt werden. Vor allem steht hier das TG-O aufgrund seiner verringerten Rechenleistung und dem geringeren Speicherplatz im Kritikpunkt.
Der Vergleich verschiedener verlustfreier Algorithmen zur Datenkompression wurde aufgrund einer ausreichenden Geschwindigkeit des Datenexportes und dem Umfang einer hinreichend ausgedehnten Analyse auf einen zukünftigen Zeitpunkt verschoben.

3.1.4 USB-Geschwindigkeit

Ein großer Zeitverbrauch liegt in der Übertragung der Daten über die USB 2.0 Schnittstelle. Theoretisch liefert die Verbindung eine Übertragungsgeschwindigkeit von bis zu 12 MiB/s. Praktisch werden immer nur 64 Byte kleine Teile der Nachricht pro Durchlauf des Nachrichten-Dienstes verschickt, um das System nicht zu lange durch die Kommunikation zu behindern. Dies hängt mit dem eingestellten Bulk Transfermodus der USB-Verbindung zusammen, welcher die Größe der zu sendenden Datenpakete beschränkt. Somit braucht eine maximal große 8192 Byte Nachricht 512 Millisekunden um vollständig verschickt worden zu sein. Dies ergibt sich aus der maximal größten Nachrichtenlänge geteilt durch die Größe der verschickten Nachricht multipliziert mit der Zeit in welcher der Nachrichten-Dienst zyklisch aufgerufen wird.

$$8192\ Byte\ /\ 64\ Byte\ *\ 4\ Millisekunden\ =\ 512\ Millisekunden$$

Das Senden der Daten erfolgt ausschließlich über den *Nachrichten-Dienst*, welcher mit einer festen Wiederholungsrate von 4 Millisekunden einen kleinen Block an Daten überträgt. Um dies zu beschleunigen, wurde die Wiederholungsrate auf eine Millisekunde heruntergesetzt und zusätzlich eine Reaktion auf ein Flag eingebaut, damit sich der Service selbst noch einmal unterhalb der ein Millisekunden-Grenze erneut aufrufen kann, sollten noch weitere Daten zu verschicken sein. Die minimale zyklische Aufrufzeit für einen Dienst ist durch das Betriebssystem auf eine Präzision im Millisekunden-Bereich festgelegt und muss deshalb zusätzlich über die Flag-Steuerung erhöht werden. Damit liegt die maximal benötigte Zeit zur Übertragung einer 8192 Byte großen Nachricht nur noch bei 128 Millisekunden und im durchschnittlichen Fall deutlich geringer. Eine genaue Zahl fluktuiert durch nebenbei laufende Prozesse zu stark, um sie verlässlich bestimmen zu können, daher wird von dem schlecht möglichsten Ergebnis ausgegangen.

Neben diesen Optimierungen wurde ein blockierendes Senden getestet, bei dem der *Nachrichten-Dienst* die Systemzeit nicht wieder frei gibt, bis die komplette Nachricht versendet wurde. Dieser Test hat gezeigt, dass das System unter entsprechenden Bedingungen dazu in der Lage ist die Geschwindigkeitsanforderung der Aufgabe zu erfüllen, jedoch hatte dies ungewollte Nebeneffekte zur Folge, welche den Normalbetrieb beeinträchtigten und einen Datenverlust an relevanten Stellen auslöste.

Der USB-Treiber selbst ist für ein Senden im *Bulk Modus* eingestellt, welcher in Kapitel 2.3 näher beschrieben wird. Es wurde eine Umstellung auf einen asynchronen Betrieb getestet, jedoch änderte dies die Programmlogik im existierenden System zu stark und wurde deshalb als mögliche Optimierung verworfen. Der Treiber hatte selbst schon eine angedeutete Unterstützung für diesen Modus, wurde jedoch im aktuellen Stand nicht vollständig dahingehend umgesetzt.

3.2 Datenspeicherung

Das zur Speicherung der exportierten Daten verwendete Format wird in Kapitel 2.5.3 beschriebenen. Hierbei werden die empfangenen Nachrichten des Serviceprotokolls direkt in *Enhanced Packet Blöcke* geschrieben. Der Dateiaufbau ist in Abbildung 3.6 dargestellt.

Zusätzlich besitzt jede Datei zwangsweise einen *Section Header Block* und einen *Interface Description Block*, welche die ersten beiden Blöcke jeder Datei darstellen. Im ersteren wurde die *Major- und Minor-Versionsnummer* auf den Wert Null gesetzt, um ein zukünftiges Schema der Versionierung zu erlauben. Als Optionen wurden die verwendeten Sensortypbeschreibungen mit ihrem entsprechenden Wert als Kommentar zur Verifizierung der verwendeten Typen beim Einlesen der Datei hinzugefügt. Die existierenden Typen können zukünftig durch einen Standard fest definiert und über eine Versionsnummer festgehalten werden, um eine Effizientere Verifikation zu erlauben. Zur Vollständigkeit wurden die *Section Header Block*

spezifischen Optionen über die verwendete Hardware bei der Aufnahme und das Betriebssystem hinzugefügt.

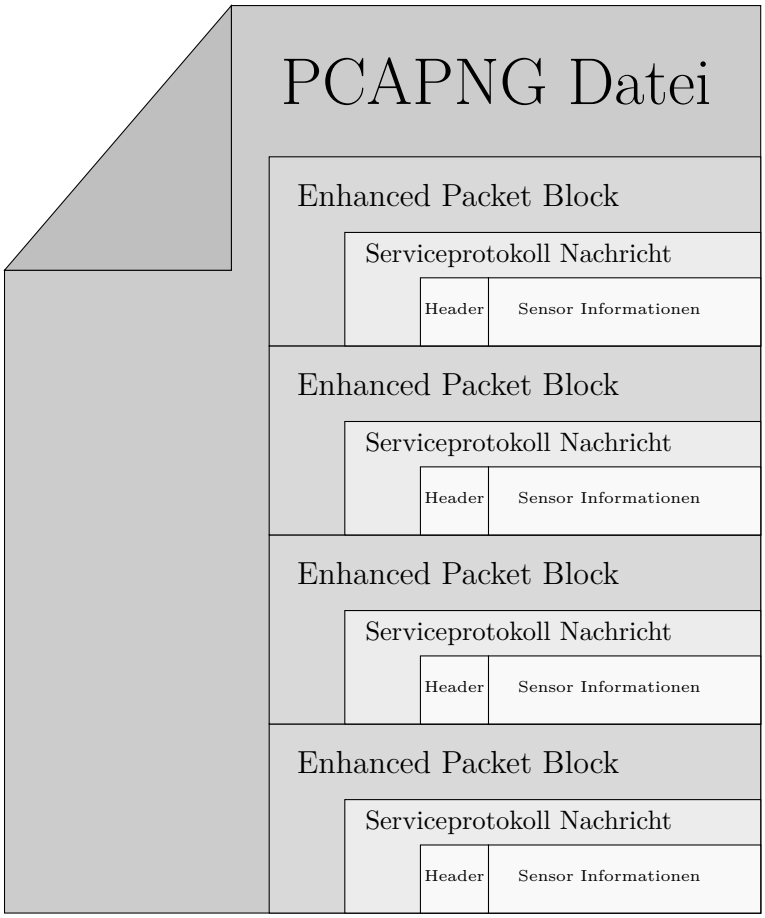


Abbildung 3.6: Darstellung des Dateiformates,
Quelle: Eigene Darstellung

Der *Interface Description Block* erhielt als *Link Type* den Wert 147, um damit einen nutzerdefinierten Datentyp auszudrücken. Eine *SnapLen* wurde nicht beschlossen und somit auf den Wert 0 gesetzt, da alle Daten benötigt werden und ein Abschneiden einer Nachricht nicht gewünscht ist. Als Teil der vordefinierten spezifischen Optionen wurde der Name der verwendeten Schnittstelle und seine Beschreibung eingetragen.

Alle folgenden Blöcke sind *Enhanced Packet Bloecke* und verweisen alle auf dasselbe Interface. Der *Zeitstempel* gibt die Erfassungszeit des Paketes an und die *Captured Packet Length* ist immer gleich der *Original Packet Length* gesetzt. In den Optionen befinden sich die Annotationen, wenn diese vorhanden sind.

3.3 Benutzeroberfläche

Um mit dem Gerät zu sprechen und einen Datenexport zu starten, bedarf es einer einfachen und bestenfalls intuitiven Softwarelösung. Realisiert wurde dies über eine grafische Benutzeroberfläche, welche für einen zur Verfügung gestellten Tablet-Computer in der Benutzung angepasst wurde. Die Applikation wird während der Testfahrt von einem Beifahrer, weitergehend als *Tester* bezeichnet, durchgeführt. Bei der Betrachtung der an das Gerät gestellten Anforderungen wird von einem ordnungsgemäß in ein Fahrzeug verbautem Gerät mit Zugang zur USB Schnittstelle für den Datenexport ausgegangen.

3.3.1 Anforderung

Die Hauptanforderungen liegen in einer einfachen Bedienbarkeit, um den Vorgang so einfach wie möglich zu gestalten. Die Steuerung der Applikation darf die Sicherheit der testenden Person nicht gefährden und ist dem entsprechend zu entwickeln.

Die Oberfläche muss

- *einfach benutzbar* sein, da man davon ausgehen muss, dass die Bedienungsumstände durch das Fahrverhalten und den verengten Platz eingeschränkt sind,
- *Informationen* über das korrekte Ausführen des Datenexportes liefern,
- *Fehler* der Anwendung oder des Benutzers ausreichend anzeigen oder umgehen,
- die *Sicherheit* der Kommunikation unterstützen und wahren,
- das Annotieren ohne Tastatureingaben erlauben und bestmöglich jedes vorkommende Ereignis abdecken,
- sich auf verschiedene *Anforderungen* in den zu testenden Sensoren und der verwendeten Geräte anpassen lassen können und
- eine einfache *Erweiterung* bei neuen Geräten oder Sensoren erlauben.

3.3.2 Umsetzung

Die Oberfläche wurde als Desktop-Applikation mit der Programmiersprache Python[9] und dem grafischen Framework QT4[10] realisiert.
Eine Steuerung über den Fahrer selbst wird ausgeschlossen, da dies eine Ablenkung vom Straßenverkehr und damit ein zu hohes Sicherheitsrisiko darstellt. Es wird dem entsprechend davon ausgegangen, dass die Bedienung von einem Beifahrer getätigt wird.

[9] siehe `https://www.python.org/`
[10] siehe `https://www.qt.io/`

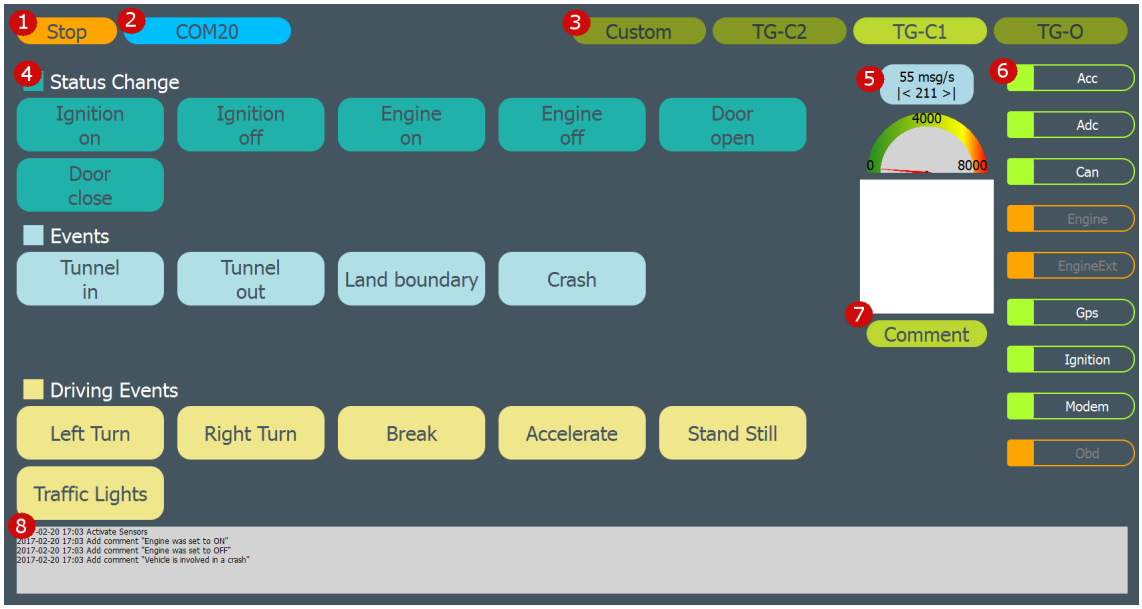


Abbildung 3.7: Grafische Benutzeroberfläche zum Datenexportieren, Quelle: Eigene Darstellung

In Abbildung 3.7 wird die Applikation während einer Aufnahme gezeigt. Mit **(1)** wird eine Aufnahme begonnen oder beendet. Bei **(2)** wird die Information der aktuell verwendeten Schnittstelle dargestellt und bei Klick darauf, die Auswahl aller verfügbaren Schnittstellen gezeigt wie in Abbildung 3.8 dargestellt. Vor dem Starten einer Aufnahme sollte unter **(3)** ein entsprechendes Gerät gewählt oder explizit die gewünschten Sensoren unter **(6)** definiert werden. Durch eine vordefinierte Geräteauswahl werden die unterstützten Sensoren automatisch aktiviert. Während der Aufnahme gibt die Anzeige **(5)** die aktuellen Pakete pro Sekunde und die maximal gemessene Länge dieser erhaltenen Pakete an. Die kleine Nadel stellt hierbei nur die Paketlänge anders dar und ist als optische Auffrischung zu betrachten. Mit den Knöpfen in Bereich **(4)** werden vorgegebene Annotationen getätigt und für einen benutzerdefinierten mehrzeiligen Kommentar kann das Eingabefeld **(7)** verwendet werden. Die Auswahl an vorgefertigten Annotationen bildet sich aus den Erfahrungen bisheriger Tests und zur Erhöhung der Übersichtlichkeit wurden diese in Gruppen unterteilt. Sämtliche Ereignisse werden in Bereich **(8)** mit Zeitstempel angezeigt und zusätzlich in eine *Logdatei* geschrieben.

Vor dem Starten einer Aufnahme muss eine Schnittstelle ausgewählt und zur Authentifizierung ein Sicherheitsdongle mit PIN angegeben werden, wie in Kapitel 2.6.5 beschrieben. Dies wird über die Anzeige in Abbildung 3.8 getätigt. Die PIN kann über ein Nummernfeld innerhalb der Oberfläche oder per Tastatur eingegeben werden. Die *Slot* Nummer dient der Auswahl bei mehreren angeschlossenen Dongles. An der linken Seite werden alle gefundenen möglichen Schnittstellen mit ihrem Namen und einem TomTom Logo angezeigt, sollte das Gerät schon als kompatibel erkannt worden sein. Als kompatibel werden alle Geräte angesehen, deren Gerätebezeichnung das Wort „TomTom“ beinhaltet.

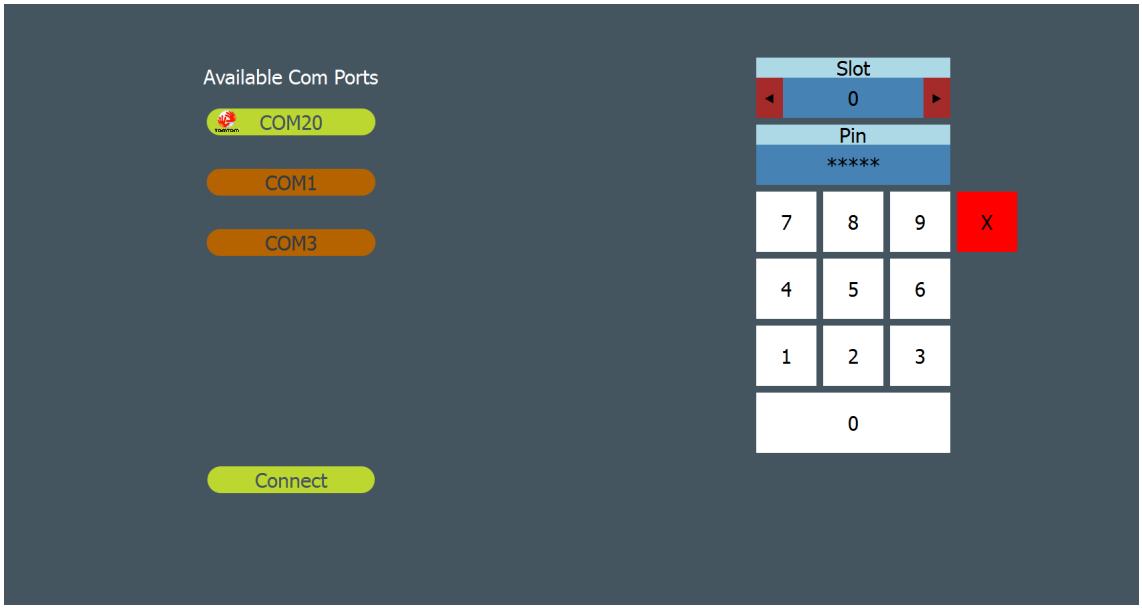


Abbildung 3.8: Einwahlbildschirm der grafischen Benutzeroberfläche, Quelle: Eigene Darstellung

Bei dem Aufnahmegerät wurde ein Tablet-Computer einem Laptop vorgezogen, da dieses zum einen aus Platzgründen und zum anderen aufgrund der Akkulaufzeit deutliche Vorteile gegenüber einem Laptop bietet. Die *Touch*-Eingabemöglichkeit erleichtert zudem das Auswählen von Annotationen. Eine genaue Übersicht über den Vergleich der möglichen Aufnahmegräte zeigt Tabelle 3.2. Die Betrachtung beinhaltete lediglich zwei vorgegebene Maschinen, welche zur Auswahl standen.

	Tablet	**Laptop**
Akkulaufzeit	+1	0
Platzverbrauch	+1	0
Eingabemöglichkeiten	0	+1
USB Anschlüsse	0	+1
Betriebssystem	+1	+1
Handhabung	+1	0
Gesamt	4	3

Tabelle 3.2: Wertung der Aufnahmegeräte

Die **Akkulaufzeit** wurde beim Laptop mit 2 Stunden bemessen und im Gegenzug beim Tablet mit 8 Stunden. Besonders bei einem für Testfahrten intensiv genutzten Tag wurde eine markant längere Lebensdauer unabdingbar. Der **Platzverbrauch** ist aufgrund des beengten Raumes in einem Auto einer der wichtigsten Punkte und macht sich auch in der **Handhabung** bemerkbar. Das Tablet ist mit einer Hand zu halten und mit der anderen bequem zu steuern. Dahingegen muss der Laptop

abgestellt werden und bietet kein sicheres Bedienen bei turbulentem Fahrverhalten. Um harte Kurven oder starkes Bremsen testen zu können, wird dieser Fahrstil benötigt. Die **Eingabemöglichkeiten** beschränken sich beim Tablet auf die *Touch Funktionalität* und bietet keine feste Tastatur oder Touchpad. Die fehlende Tastatur am Tablet wurde bei der Implementierung berücksichtigt und einer *Touch optimierten Oberfläche* eine höhere Priorität zugeteilt. Durch die Verbindung zum Gerät und dem Sicherheitsdongle werden mindestens zwei **USB Schnittstellen** benötigt, welche beim Tablet durch einen zusätzlichen USB-Verteiler bereitgestellt werden mussten. In beiden Maschinen wurde *Windows* als **Betriebssystem** verwendet und sie unterstützten damit die Anforderungen zum Ausführen der Oberfläche.

4 Evaluation

Die exportierten Daten werden durch die nachfolgenden Tests auf ihre Integrität geprüft. Die Testdaten werden durch *Testfahrten* erhoben und jede Testfahrt besteht aus mehreren *Trips*. Als *Trip* wird eine Zeitspanne vom Stillstand des Fahrzeuges mit ausgeschaltetem Motor über eine sachgemäße Verwendung bis zum erneuten Stillstand des Fahrzeuges mit ausgeschaltetem Motor betrachtet.
Legitime Testdaten erfüllen die nachfolgenden Anforderungen, um zur Evaluation in Betrachtung gezogen zu werden.

- Alle für den Test benötigten Sensordaten wurden lückenlos aufgezeichnet,
- Testdaten stehen aus mindestens drei verschiedenen Fahrzeugen zur Verfügung,
- Die Aufzeichnung wurde wenigstens eine Minute vor dem Anschalten des Motors begonnen und frühestens eine Minute danach erst beendet und
- Eine Testfahrt besteht aus mindestens 10 Trips.

Bei Aufzeichnungen kann es unter Umständen zu einer **Lücke** in den exportierten Daten kommen, weil das Gerät die Daten nicht schnell genug exportieren konnte und der interne Speicherplatz aufgebraucht war. In diesem Fall werden die Sensordaten verworfen und die Anzahl der verworfenen Daten wird beim nächsten Paket mit angegeben.
Um Fahrzeug-spezifische Eigenheiten zu erkennen, bedarf es **unterschiedlicher Fahrzeuge** zur Erhebung der Testdaten. Hierzu wurde ein *Volkswagen Touran* (Testfahrt 1), ein *Ford Fiesta* (Testfahrt 2) und ein *Renault Mégane* (Testfahrt 3) verwendet.
Für eine saubere Erkennung aller Signale wird eine **Vor- und Nachlaufzeit** von einer Minute angesetzt.
Um eine ausreichend große Menge an Trips zu vergleichen, wird eine Mindestanzahl von 10 Trips festgelegt.

4.1 Annotationen-Abgleich

Zur Überprüfung werden die erhaltenen Daten mit definierten Annotationen verglichen. Diese dienen zum Markieren von aufgetretenen und erwarteten Begebenheiten. Geprüft wird ein markanter Ausschlag von Sensoren bei beispielsweise scharfem Bremsen oder dem Ausschalten des Motors an den annotierten Zeitpunkten.

4.1.1 Anforderung

Der Benutzer gibt durch Kommentare an, zu welchem Zeitpunkt eine gewisse Datenänderung erwartet wird oder welches externe Ereignis gerade aufgetreten ist, um den exportierten Daten mehr Zusatzinformationen darüber zu geben.
Dies tritt beispielsweise beim Starten des Fahrzeugs auf, wenn der Zündschlüssel umgedreht wird und der Motor anspringt. Zu diesem Zeitpunkt sollte ein kurzer Abfall der Versorgungsspannung auftreten und danach ein erhöhter Wert dieser erkennbar sein. Der Benutzer markiert diesen Zeitpunkt durch einen Kommentar im nächst möglichen Datenblock.

4.1.2 Umsetzung

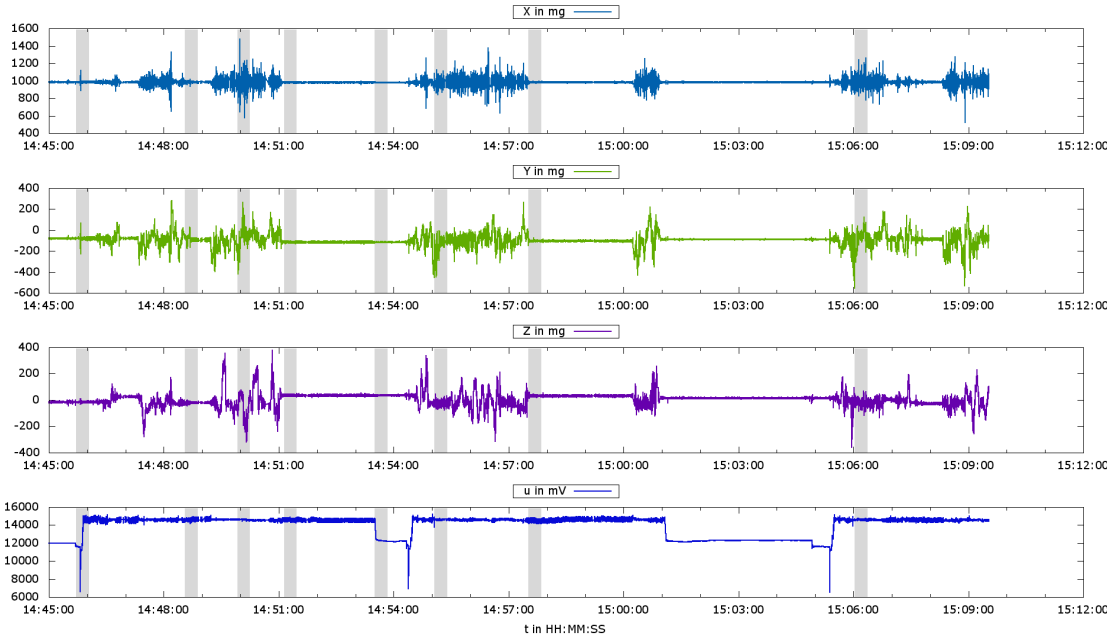
In der Abbildung 4.1 wird ein Ausschnitt der Beschleunigungsdaten und der Versorgungsspannung einer Aufzeichnung mit beispielhaften Annotationen gezeigt. Die Kommentare wurden in einem Zeitraum von 10 Sekunden, davor und danach, Grau hinterlegt.
Der Beschleunigungssensor stellt Informationen über die auf ihn wirkende Beschleunigung im dreidimensionalen Raum anhand der X-, Y- und Z-Achse bereit. Seine Werte werden in der Maßeinheit Milli-g angegeben, was für die mittlere Erdbeschleunigung steht.
Die Versorgungsspannung wird über den ADC Sensor abgegriffen und wurde in Millivolt umgerechnet.

4.1.3 Auswertung

Zur Auswertung wird eine Sichtung der Daten durchgeführt und eine Verbindung zwischen Pegeln und Kommentaren gesucht. Diese Tests sind sehr fehleranfällig und müssen mit erhöhter Toleranz betrachtet werden, da möglicherweise

- nicht alle markanten Stellen kommentiert wurden,
- eine Beschleunigung sich auf mehrere Achsen verteilt und damit nicht markant heraus sticht oder

- ein unbeachtetes Ereignis wie das Wackeln am Fahrzeug ein Rauschen auf die Daten setzt.



14:45:53 Motorzustand wurde auf AN geändert

14:48:43 Fahrzeug steht an einer Ampel

14:50:05 stärkeres Beschleunigen

14:51:18 Fahrzeug steht still

14:53:40 Motorzustand wurde auf AUS geändert

14:55:14 stärkeres Beschleunigen

14:57:41 Fahrzeug steht still

15:06:12 stärkeres Beschleunigen

Abbildung 4.1: Beschleunigungswerte und Versorgungsspannung im Diagramm, Quelle: Eigene Darstellung

Es werden die gebräuchlichsten Annotationen zur Auswahl vorgegeben, jedoch kann das Fahren über eine Bodenschwelle einen markanten Pegel in den Beschleunigungsdaten hervorrufen, dies aber gleichzeitig für den Tester nahezu unbemerkt bleiben. Ein Stillstand des Fahrzeuges ist leicht zu erkennen, da die Amplitude jeder Achse des Beschleunigungssensors dabei sehr gering ausfällt. Im Falle einer starken Beschleunigung existieren stärkere Pegel in einem kurzen Zeitfenster. Je nach Lage des

Gerätes im Fahrzeug ist die Verteilung der Kraft in entsprechende Richtungen des Raumes anders. Die Versorgungsspannung ist nicht durch das Fahrverhalten beeinträchtigt und weist lediglich beim Starten oder Stoppen des Motors eine markante Änderung auf. Besonders der kurze Pegelabfall beim Starten des Motors ist gut sichtbar.

4.2 Motorzustandserkennung

Bei Geräten ohne CAN Anschluss wird der Zustand des Motors durch eine Kombination aus OBD Daten und der Versorgungsspannung ermittelt. Dies wird über einen internen Zustandsautomaten geregelt und bei Zustandswechsel die Information zur Verfügung gestellt. Der Algorithmus wurde patentrechtlich geschützt (vgl. [Tel15]). Diese Erfassung wird mit den exportierten Daten nachimplementiert und das Zeitdelta zwischen den Änderungen in den Daten mit dem erzeugten Änderungssignal verglichen.

In der Plattform wird ein Zustandsautomat verwendet, um den aktuellen Zustand des Motors zu bestimmen. Dieser beinhaltet die verschiedenen Zustände und ihre Überführungs-Voraussetzungen, wie in Abbildung 4.2 dargestellt. Die Plattform selbst gibt nur die Zustände *Engine on* und *Engine off* in eine höhere Schicht bekannt. Intern wird jedoch ein weiterer Zustand *Pending* verwendet, um ein vorschnelles Springen nach *Engine off* zu verhindern und mögliche kurzzeitige Schwankungen in der Versorgungsspannung zu tolerieren.

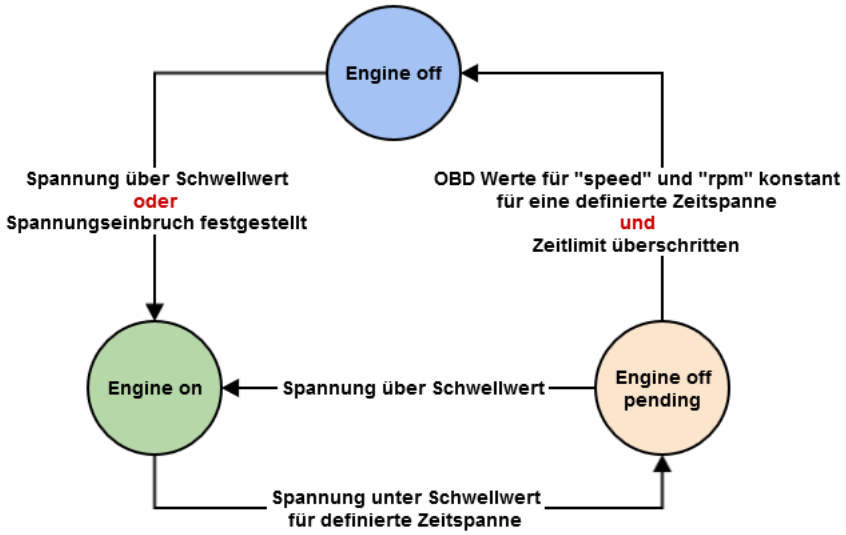


Abbildung 4.2: Motorzustandsautomat
Darstellung nach [Tel15]

4.2.1 Anforderung

Die exportierten Daten müssen OBD, ADC und *Motorzustands* Signale beinhalten. Der Algorithmus selbst basiert nur auf den OBD und ADC Werten, jedoch wird das Ergebnis danach mit den *Motorzustands*-Signalen verglichen. Seine Abweichung darf eine Toleranz von 0,05 Sekunden zu den exportierten Signalen aufweisen, um ein ausreichend gutes Ergebnis zu liefern. Zur Erhöhung der Genauigkeit des Vergleiches der Implementierung wird zusätzlich der interne Zustand der Motorzustandserkennung mit exportiert und alle drei möglichen Zustände damit betrachtet.
Als letzter Vergleichspunkt dienen noch die Annotationen über den Motorzustand, welche während der Testfahrt erstellt wurden. Diese sind jedoch nur als grobe Kennung zu betrachten und können nicht zur zeitlichen Analyse mit hinzugezogen werden.

Die Anforderungen an diesen Test werden nur vom TG-O erfüllt, da das TG-C keine OBD Information verwendet und den Motorzustand einstellungsabhängig über die CAN oder ADC Werte bestimmt.

4.2.2 Umsetzung

Zur Evaluation wurde der in Abbildung 4.2 gezeigte Zustandsautomat mit allen Zuständen und Übergangsbedingungen in der Skriptsprache *Python* implementiert. Die Resultate werden im Vergleich in Abbildung 4.3 dargestellt.

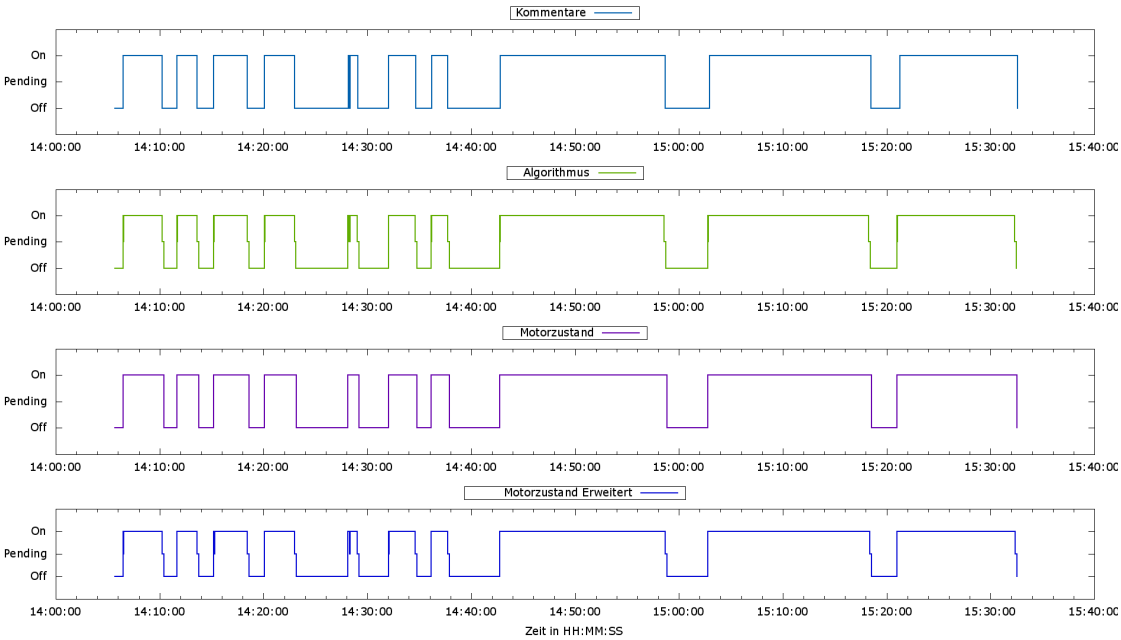


Abbildung 4.3: Motorzustandserkennung Diagramm,
Quelle: Eigene Darstellung

Kommentare haben keinen *Pending* Zustand, da der Tester keinen Zugriff auf den internen Zustand der Software hat und deshalb nur ein *Motor An* oder *Motor Aus* annotiert werden kann.
Die Abweichungen vom **Algorithmus** betragen im Durchschnitt 0,02 Sekunden und weisen eine erhöhte Sensibilität beim Zustandswechsel zwischen *On* und *Pending* auf.
Der **Motorzustand** stellt die Daten dar, welche von der Plattform nach oben weiter gereicht werden und nur diese Daten sind nach Kapitel 2.2.2 im engeren Sinne für das exportieren relevant.
Für einen präziseren Vergleich wurden die internen Zustände des Zustandsautomaten explizit exportiert und unter der Bezeichnung **Motorzustand Erweitert** dargestellt.

4.2.3 Auswertung

Die Ergebnisse des Tests werden in der Tabelle 4.1 gezeigt. Die gesetzte Toleranzzeit von 0,05 Sekunden wurde vom Algorithmus bei allen Zustandswechseln im Vergleich zum exportierten Signal eingehalten. Darauf basierend kann die Nutzbarkeit der Daten zur Reimplementierung einer internen Funktionalität als zutreffend angesehen werden.

	Testfahrt 1	**Testfahrt 2**	**Testfahrt 3**
Aufzeichnungslänge (HH:MM:SS)	01:27:40	00:48:13	00:21:30
Abweichung der Signale (Sekunden)	0,02	0,02	0,02
Anzahl Trips	10	10	11
Fehlerhafte Erkennung	0	0	0

Tabelle 4.1: Ergebnisse der Motorzustandserkennung

4.3 CAN Daten Abgleich

Bei CAN Exporten kann zur Überprüfung der extrahierten Daten ein Vergleich mit den eingespielten Werten erfolgen. Hierbei kann eine genaue Messung der erwarteten mit den erhaltenen Daten vorgenommen werden, um eine präzise Aussage über eine etwaige Abweichung zu treffen.

4.3.1 Anforderung

Es werden exportierte CAN Daten (Testdaten) genutzt, welche nicht durch die in dieser Arbeit implementierte Softwareanpassung erhoben wurden. Es handelt sich um reale CAN Aufzeichnungen von Testfahrten, welche mit einem USB CAN-Adapter[11] erhoben wurden. Die Testdaten sind in einem Comma Separated Values (CSV) Dateiformat gespeichert. In diesem Format werden Informationen durch ein *Komma* beziehungsweise ein anderes markantes Trennzeichen wie einem *Semikolon* oder einen *Tabulator* voneinander Unterschieden. Jeder CAN Datensatz ist in einer eigenen Zeile mit dem in Abbildung 4.4 dargestellten Aufbau gespeichert.

```
"Time";"Identifier (hex)";"Format";"Flags";"Data (hex)"
"00:00:25.07";"3D4";"Std";"";"00 00 00 00 00 00 00 00 "
"00:00:25.07";"497";"Std";"";"00 00 00 00 44 46 00 00 "
"00:00:25.07";"5D6";"Std";"";"00 00 00 00 00 "
"00:00:25.08";"720";"Std";"";"01 03 0D 00 04 00 00 "
"00:00:25.08";"5DC";"Std";"";"75 00 70 3F 9C 18 00 C0 "
"00:00:25.08";"7D0";"Std";"";"D8 00 00 00 00 00 00 C0 "
```

Abbildung 4.4: CAN Testdatenabschnitt

Die erste Spalte **Time** gibt die Zeit des Absendens des Paketes an mit einer Genauigkeit von zwei Nachkommastellen im Sekundenbereich. Der **Identifier** ist die CAN-ID der Nachricht in hexadezimaler Schreibweise dargestellt. Das **Format** zeigt, ob es sich um eine *Standard* oder *Extended* CAN Nachricht handelt. Die **Flags** Spalte ist ungenutzt und wird aus Kompatibilitätsgründen mitgeführt. Die Informationen der Nachricht sind mit dynamischer Länge in der letzten Spalte unter **Data** aufgeführt.
Zum Vergleich müssen die exportierten Daten aus dem PCAPNG Dateiformat in das CSV Format der Testdaten konvertiert werden. Besonders die unterschiedliche Länge der Nachricht muss beim Vergleich berücksichtigt werden, um fehlerhafte Unterschiede zu vermeiden. In der Plattform verliert sich die Information der Nachrichtenlänge und die eigentlichen Daten werden intern als Zahl weiterverarbeitet. Deshalb müssen beim Konvertieren alle Nachrichten mit voller 8 Byte Länge gespeichert werden.

[11] Adapter, um einen Computer über USB mit einem CAN Bus zu verbinden und Daten darauf zu schreiben und davon zu lesen

Für den Vergleich ist eine korrekte Reihenfolge und Vollständigkeit ausschlaggebend. Ein minimaler zeitlicher Unterschied der einzelnen Nachrichten untereinander ist aufgrund von verschiedenen unvorhersagbaren Faktoren, wie zum Beispiel einer ankommenden Nachricht über das Modem, zu tolerieren. Diese Zeitspanne wird nachfolgend als *Zeitdelta* bezeichnet und darf für eine erfolgreiche Wertung nicht größer als 0,01 Sekunden sein. Es muss von einer Abweichung aufgrund von Störungen in der Verarbeitung ausgegangen werden und 0,01 Sekunden stellt die kleinst mögliche Abweichung dar.

Ein Datensatz zählt dann als erfolgreich, wenn er an der selben Position zur Testdatei steht und sein Zeitdelta zum Vordatensatz sich um nicht mehr als 0,01 Sekunden unterscheidet.

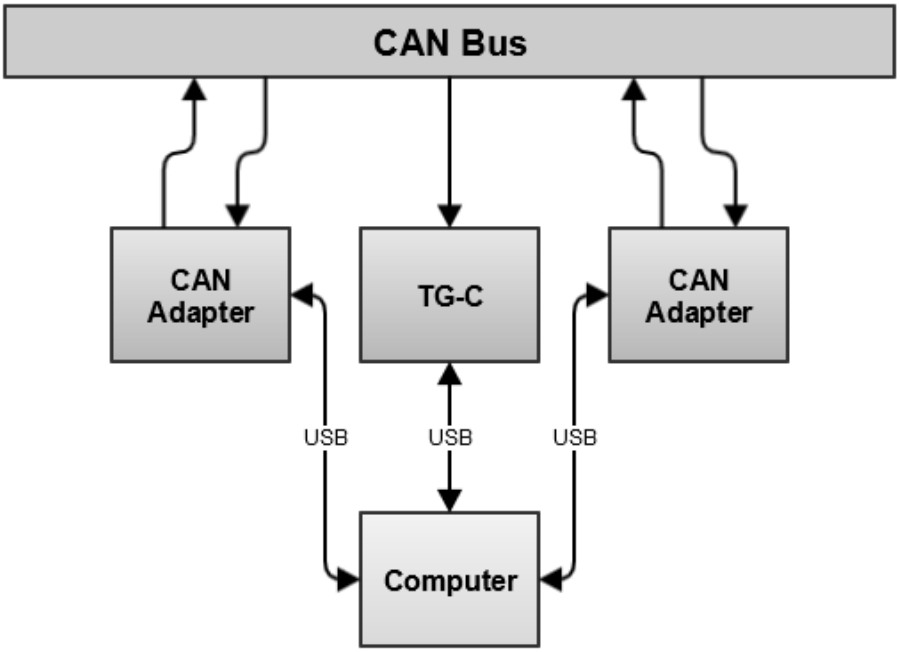


Abbildung 4.5: Labortest Aufbau,
Quelle: Eigene Darstellung

4.3.2 Umsetzung

Eine CAN Nachricht braucht eine Bestätigung der abgesendeten Nachricht und somit mindestens einen Abnehmer auf dem Bus. Ohne Bestätigung der Nachricht würde diese so lange wiederholt werden bis mindestens eine Quittierung dazu erhalten wurde.

Um dies zu erreichen wurden zwei CAN-Adapter mit einem CAN Bus verbunden und CAN Nachrichten zwischen ihnen ausgetauscht. Das TG-C wurde mit dem selben Bus verbunden, um die Kommunikation mitschneiden zu können. Das TG-C kann selber keine Nachrichten quittieren, da es keinen schreibenden Zugriff zum Bussys-

tem hat, daher werden zwei CAN-Adapter für den Nachrichtentransfer benötigt. Der Aufbau ist in Abbildung 4.5 abgebildet.

Als Testdaten wurden drei verschiedene CAN Aufzeichnungen in verschiedener Länge verwendet. Jede Testdatei wurde zehn mal abgespielt und mit dem TG-C exportiert.

4.3.3 Auswertung

Alle Daten haben eine unterschiedliche **Länge**, um verschiedene Fahrzeiten mit in die Messung einzubringen. Zudem beinhaltet Testdatei 1 auch eine Fahrpause, wodurch die Größe der **Datenmenge** in der kürzeren Testdatei 2 erklärt wird. Die **Abweichungen** im Zeitdelta sind alle unterhalb der gesetzten 0,01 Sekunden Grenze und werden somit als korrekt aufgefasst. Das Testergebnis ist in Tabelle 4.2 dargestellt.

	Testdatei 1	**Testdatei 2**	**Testdatei 3**
Aufzeichnungslänge (M:SS)	5:12	4:49	0:26
CAN Daten	565989	571883	51604
Test Wiederholungen	10	10	10
Abweichungen im Zeitdelta (Durchschnitt)	4,95%	4,76%	1,99%
Maximale Abweichung im Zeitdelta (Sekunden)	0,01	0,01	0,01
Erfolgreiche Exporte	100%	100%	100%

Tabelle 4.2: CAN Daten Abgleich Ergebnisse

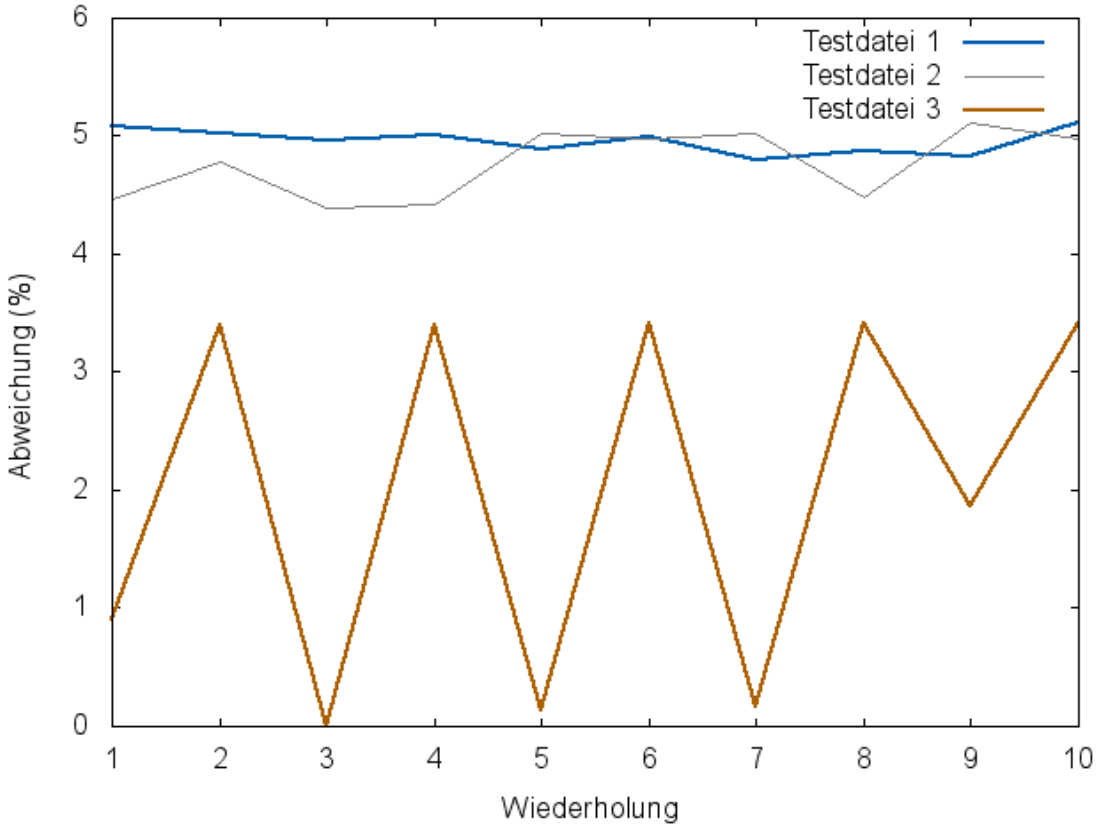


Abbildung 4.6: Exportdaten Zeitdelta Differenzen,
Quelle: Eigene Darstellung

Entsprechend dieser Tests sind die vom TG-C exportierten Daten vollständig und in einer unveränderten Reihenfolge. Der Unterschied in dem Zeitdelta ist durch andere Systemprozesse und Verzögerungen in der Datenübertragung zu erklären. Eine genaue Übersicht zeigt die Abbildung 4.6. Die Durchschnittliche Anzahl der Abweichungen liegt bei längeren Testdaten entgegen der 5%. Testdatei 3 hat aufgrund seiner geringen Laufzeit zu schwankende Werte, um eine klare Tendenz zu definieren. Es wird daher davon ausgegangen, dass die Zeitunterschiede durch beispielsweise das Aufnehmen der Modem-Verbindung entstehen. Da sich diese erst nach einer geringen Vorlaufzeit aktiviert und deshalb bei kurzen Aufzeichnungen den Datenexport nicht beeinflusst.

4.4 Webfleet Abgleich

Die von TomTom betriebene Webseite zur Flottenverwaltung (*Webfleet*) bietet die Verfolgung des Fahrzeuges über GPRS empfangener GPS Daten an.
Die Samplerate von GPS im Gerät entspricht einem Hertz. Die Genauigkeit von Webfleet basiert jedoch auf mehreren Faktoren. Strecken werden gröber dargestellt als es die Sensorabtastrate erlaubt, um die Datenübertragung geringer zu halten und dennoch eine klare Streckenführung zu erlauben. Dies bedeutet jedoch auch, dass einzelne Zwischenpunkte dabei verworfen werden. Die Webfleet GPS Daten sollten somit eine Teilmenge der exportierten Daten sein, jedoch eine deutlich geringere Auflösung an Punkten haben.
Durch die Ungenauigkeit im GPS werden trotz Stillstand unterschiedliche Positionen erkannt und das verfälscht die Strecke. Dieser Effekt wird *driften* genannt (vgl. [KH05]). Bei einem erkannten Stillstand des Fahrzeuges wird durch einen *Driftreduktions Algorithmus* eine Stabilität in den GPS Punkten erzeugt.

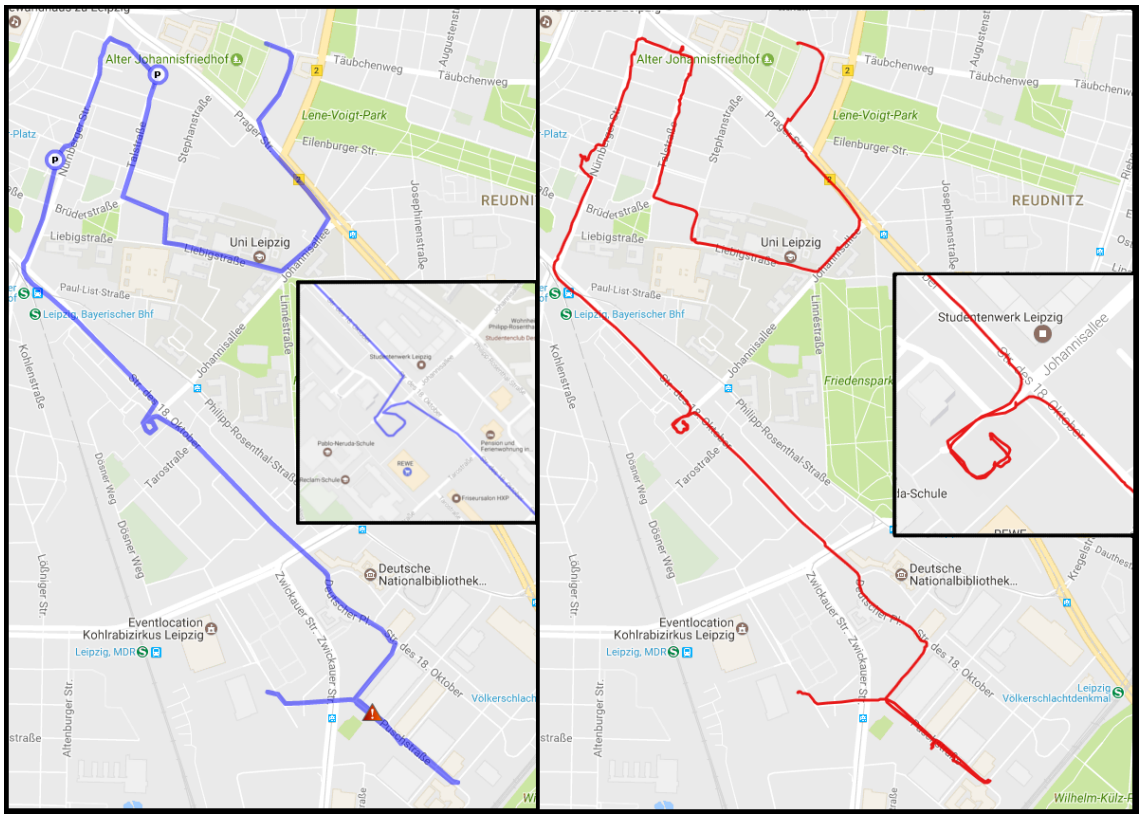


Abbildung 4.7: Vergleich Webfleet (Blau) und Datenexport (Rot),
Quelle: Eigene Darstellung

4.4.1 Anforderung

Das zu testende Gerät muss eine Verbindung mit dem Webfleet herstellen, um die GPS Daten versenden zu können. Dazu muss es registriert und signiert sein, um den sicheren Zugang aufbauen zu dürfen wie in Kapitel 2.6.4 beschrieben wurde.

Zum Vergleich müssen die exportierten GPS Punkte in einem gleichen Umfeld angezeigt werden und die von Webfleet bezogenen Punkte müssen vollständig in den exportierten Daten vorliegen.

4.4.2 Umsetzung

Es wurden verschiedene Testfahrten zur Datenerhebung durchgeführt. Ein Beispiel wird in Abbildung 4.7 gezeigt. Die blaue Linie im linken Bild zeigt hierbei die Darstellung von Webfleet auf der Google Karte[12]. Auf der rechten Seite ist mit einer roten Linie die Strecke anhand der exportierten Daten dargestellt[13]. In dem vergrößerten Abschnitt ist eine genauere Abtastung zu erkennen, da die exportierten GPS Daten höherfrequent sind.

	Testfahrt 1	**Testfahrt 2**	**Testfahrt 3**
Aufzeichnungslänge (HH:MM:SS)	55:50	37:15	15:30
Pausen (Minuten)	23	0	2
Datenmenge (Webfleet)	209	227	87
Datenmenge (Export)	3350	2235	930
Minimale Abweichung (Meter)	0,00	0,00	0,00
Durchschnittliche Abweichung (Meter)	0,04	0,04	0,04
Maximale Abweichung (Meter)	0,07	0,07	0,07

Tabelle 4.3: Webfleet GPS Vergleich

[12] siehe `https://maps.google.com`
[13] Darstellung durch `http://www.gpsvisualizer.com`

4.4.3 Auswertung

In Tabelle 4.3 werden die Ergebnisse der Tests dargestellt. Es ist eine Abweichung der GPS Punkte im hinteren Nachkommabereich festzustellen und dies wird auf einen Umrechnungsfehler durch Webfleet zurückgeführt. Die Abweichungen betragen im maximalen Fall 0,07 Meter und sind für eine präzise Streckenführung in einem akzeptablen Rahmen.

Die Differenz zwischen zwei Punkten wurde über die Haversine Funktion ermittelt (vgl. [Rob57]). Durch sie kann eine Distanz auf einer Sphere mittels der *geographischen Länge* und der *geographischen Breite* zweier Punkte berechnet werden.

4.5 Gesamtauswertung

Die Sensordaten wurden innerhalb der gesetzten Anforderungen an ihre Qualität getestet und jeder Test kann als erfolgreich angesehen werden.

Die Annotationen verweisen auf die korrekten Zeitpunkte in der Aufzeichnung und lassen eine markante Stelle in den Sensordaten nachträglich identifizieren.

Anhand der exportierten Daten lassen sich interne Algorithmen nachbilden, was das Erstellen einer Testumgebung zur automatisierten Prüfung der Funktionen erlaubt.

CAN Daten können fehlerfrei mitgeschnitten und in einer Testumgebung wieder abgespielt werden. Eine Veränderung im Zeitdelta ist hierbei minimal und nachvollziehbar, bedingt durch die Einwirkung anderer Prozesse und einer Bearbeitungsverzögerung.

Die aufgezeichneten GPS Positionen stellen eine genauere Abbildung der Webfleet abrufbaren Strecke dar. Die Schnittmenge der Punkte unterscheiden sich aufgrund von Rundungsfehlern um maximal 0,07 Meter und stellt für eine präzise Streckenführung eine akzeptable Abweichung dar.

5 Fazit

Die Resultate der Arbeit belegen, dass die Daten verlustfrei und fehlerfrei exportiert wurden. In die bestehende Plattform wurde eine Erweiterung integriert, um den Export von Sensordaten zu ermöglichen. Die Integrität der Daten und ein schnellstmöglicher Export durch die verwendete Schnittstelle wurde sichergestellt.
Es wurde zudem ein erweiterbares und standardisiertes Speicherformat ausgewählt, welches unter die Begrifflichkeit *State of the Art* fällt. Dies bedeutet, dass es zum Zeitpunkt dieser Arbeit dem aktuellsten Stand der Technik entspricht und aufgrund einer weiten Verbreitung und beständiger Weiterentwicklung auch zukünftig noch nutzbar bleiben wird. Es beinhaltet eine Unterstützung für Kommentare an den exportierten Daten, der verwendeten Schnittstelle und der Datei selbst. Zudem beugt es Fehlern in der Konvertierung vor und unterstützt die Versionierung bei sich ändernden Anforderungen.
Dem Tester wurde eine Benutzeroberfläche bereitgestellt, um effektiv Testdaten zu erfassen. Dabei wurden die Sicherheitsmechanismen der Geräte betrachtet und integriert. Es wurde explizit Wert auf eine leichte Bedienbarkeit und Erweiterbarkeit der Oberfläche gelegt.
Die Sensordaten wurden durch einen zeitlichen Abgleich mit den erstellten Annotationen geprüft, eine Soll- und Ist-Wert Analyse von eingespielten CAN Daten getätigt und die erfassten GPS Wegpunkte mit Webfleet verglichen. Zudem wurde ein bestehender Algorithmus nachimplementiert und das Ergebnis dessen, mit den originalen Werten abgeglichen.
Diese Arbeit hat somit die an sich gestellte Aufgabe zum Exportieren von Sensordaten aus vorgegebenen Geräten während der Verwendung in einem Fahrzeug erfüllt.

5.1 Problematiken

Die größten Probleme kamen durch die zeitlichen Einbußen beim Herstellen einer sicheren Modemverbindung, der Übertragungszeit der Daten über die USB Schnittstelle und dem Bedarf an internem Speicherplatz. Durch die Verbindung mit dem GPRS Netzwerk wurde das System aufgrund von kryptografischen Verfahren längerfristig blockiert und führte zu einem enormen Anstieg in der Datenmenge. Damit verbunden ist die beschränkte Speicherkapazität und eine geringe maximale Anzahl von Nachrichten in der Queue. Je mehr Daten zu exportieren sind, umso länger dauert der Exportvorgang und umso öfter muss der *Nachrichten-Dienst* aufgerufen werden, um einen Teil der Nachricht zu übertragen. Im schlimmsten Fall werden Daten verworfen und sind damit unwiderruflich verloren.
Um dies vorzubeugen, sollte bei jedem Datenexport auf eine vollständig hergestellte Verbindung zum Webfleet gewartet werden, um eine Last durch das Modem zu vermeiden.

5.2 Ausblick

Während der Analyse und der Implementierung sind verschiedene Ansätze zu weiteren Optimierungen aufgekommen. Diese wurden für eine weiterführende Arbeit in Aussicht gestellt und betreffen unterschiedlichste Bereiche des Gesamtprozesses.

Um die Größe der zu exportierenden Nachricht weiter zu minimieren kann eine Datenkompression zur effektiveren Verwendung näher betrachtet und verschiedene Kompressionsverfahren getestet werden. Zudem erlaubt eine nachträglich eingeführte Änderung in dem für die CAN Daten zuständigen Treiber, einen Zugriff auf die Länge des erhaltenen Pakets und bietet damit die Möglichkeit die zu exportierenden CAN Daten weiter zu komprimieren.

Das aktuelle Format zeigt die Anzahl der verworfenen Sensordaten an, gibt jedoch keine Auskünfte über die fehlende Sensorart. Zur besseren Fehlerbestimmung kann diese Information in einem expliziten Block untergebracht werden. Dazu muss das Dateiformat überarbeitet und mehrere Ansätze zum optimalen und damit minimalen Speicherverbrauch analysiert werden.

Weitergehend steht eine Rückführung der Daten in das Gerät in Aussicht, um eine komplette aufgenommene Fahrt vollständig simulieren zu können und die Funktionalität des Gerätes automatisiert zu testen. Hierfür muss ein Mechanismus zum Bereitstellen der Daten an eine höherliegende Schicht implementiert werden, welcher vor allem die zeitlichen Anforderungen einhält, um eine wirklich genaue Simulation zu erzeugen.

Die Benutzeroberfläche bedarf eines Beifahrers zur sicheren Nutzung bei Testfahrten, da der Fahrer durch die Bedienung zu stark vom Straßenverkehr abgelenkt ist. Durch eine Sprachsteuerung wäre es möglich diese Einschränkung zu umgehen. Dazu bedarf es jedoch einer ausführlichen Analyse von Spracherkennungssoftware und einer Anpassung der Oberfläche an die neue Bedienbarkeit.

Literatur

[And97] Don Anderson. *USB system architecture*. Addison-Wesley Professional, 1997.

[App17] Virtual Institute of Applied Science. *Analog-Digital-Wandler*. Deutsch. 27. Feb. 2017. URL: http://www.vias.org/mikroelektronik/adc_succapprox.html.

[BGR12] Klaus Beiter, Oliver Garnatz und Christoph Rätz. „Gesetzliche On-Board-Diagnose und ODX“. In: *Diagnose in mechatronischen Fahrzeugsystemen III S* 44 (2012).

[Gar+96] John Garney u. a. „An analysis of throughput characteristics of universial serial bus“. In: *Media and Interconnect Technology, Intel Architecture Labs, Tech. Rep.* Citeseer. 1996.

[Gro17] Network Working Group. *PCAP Next Generation (pcapng) Capture File Format*. Englisch. 27. Feb. 2017. URL: http://xml2rfc.tools.ietf.org/cgi-bin/xml2rfc.cgi?url=https://raw.githubusercontent.com/pcapng/pcapng/master/draft-tuexen-opsawg-pcapng.xml&modeAsFormat=html/ascii&type=ascii.

[Haa98] Jaap Haartsen. „Bluetooth-The universal radio interface for ad hoc, wireless connectivity“. In: *Ericsson review* 3.1 (1998), S. 110–117.

[HRM04] Timo Halonen, Javier Romero und Juan Melero. *GSM, GPRS and EDGE performance: evolution towards 3G/UMTS*. John Wiley & Sons, 2004.

[Ins17] National Instuments. *CAN Physical Layer Standards*. Englisch. 3. Feb. 2017. URL: http://digital.ni.com/public.nsf/allkb/84210794086E9C0886256C1C006BE6AE.

[ISO15] ISO. *11898-1:2015 Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signalling*. Norm. 2015.

[ITW17a] ITWissen. *GPS (global positioning system)*. Deutsch. 27. Feb. 2017. URL: http://www.itwissen.info/definition/lexikon/global-positioning-system-GPS-GPS-System.html.

[ITW17b] ITWissen. *Sensor*. Deutsch. 27. Feb. 2017. URL: http://www.itwissen.info/definition/lexikon/Sensor-sensor.html.

[Kan10] Krishna Kant. *Computer-Based Industrial Control, 2/e*. Eastern economy edition. Prentice-Hall Of India Pvt Limited, 2010. ISBN: 9788120339880. URL: https://books.google.de/books?id=3714jIryozYC.

[KH05] Elliott Kaplan und Christopher Hegarty. *Understanding GPS: principles and applications*. Artech house, 2005.

[Kre10] Jay Kreibich. *Using SQLite*. O'Reilly Media, Inc., 2010.

[Kum15] Dr. R. Kumaraswamy. *CAN (Controller Area Network) Protocol*. Englisch. 6. Okt. 2015. URL: https://www.slideshare.net/Darshanks3/canppt.

[Mat09] Rami J Matarneh. „Self-adjustment time quantum in round robin algorithm depending on burst time of the now running processes". In: *American Journal of Applied Sciences* 6.10 (2009), S. 1831.

[Rob57] C Carl Robusto. „The cosine-haversine formula". In: *The American Mathematical Monthly* 64.1 (1957), S. 38–40.

[Sch07] Wolf D Schmidt. *Sensorschaltungstechnik*. Vogel Business Media, 2007.

[Tel15] TomTom Telematics. *Engine state detection device*. Englisch. 14. Aug. 2015. URL: https://depatisnet.dpma.de/DepatisNet/depatisnet?docid=WO002016024023A1&xxxfull=1.

[WWP04] Marko Wolf, Andre Weimerskirch und Christof Paar. „Security in automotive bus systems". In: *Workshop on Embedded Security in Cars*. 2004.

Eidesstattliche Erklärung

Ich versichere an Eides Statt durch meine eigenhändige Unterschrift, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt habe. Alle Stellen, die wörtlich oder dem Sinn nach auf Publikationen oder Vorträgen anderer Autoren beruhen, sind als solche kenntlich gemacht. Ich versichere außerdem, dass ich keine andere als die angegebene Literatur verwendet habe. Diese Versicherung bezieht sich auch auf alle in der Arbeit enthaltenen Zeichnungen, Skizzen, bildlichen Darstellungen und dergleichen. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann.

Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Leipzig,
den 1. März 2017
Ort, Datum

Kai Trott