HAW HAMBURG

Bachelorarbeit

Navruz Jabbarov

Visualisierung und Analyse der CAN-Bus Verkehrsströme von Sonderfahrzeugen

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Navruz Jabbarov

Visualisierung und Analyse der CAN-Bus Verkehrsströme von Sonderfahrzeuge

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Informatik Technischer Systeme*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Frans-Josef Korf
Zweitgutachter: Prof. Dr. Tim Tiedemann
Technische Begleiter: Andreas Hasenfuß

Eingereicht am: 02.07.2025

Navruz Jabbarov

Thema der Arbeit

Visualisierung und Analyse der CAN-Bus Verkehrsströme von Sonderfahrzeuge

Stichworte

KMWare, CAN Explorer, Filter, Grafik, CAN-Nachricht, Tabelle, CAN-Bus, Qt, C++, QML.

Kurzzusammenfassung

In dieser Arbeit wird die Entwicklung einer modernen Softwarelösung zur Visualisierung und Analyse von CAN-Bus-Nachrichten in Sonderfahrzeugen vorgestellt. Ausgangspunkt war die bisher verwendete CAN-Explorer-Anwendung, die technisch überholt, fehleranfällig und nicht dokumentiert war. Darüber hinaus war sie in C# implementiert – einer Programmiersprache, die im Unternehmen Kinemotion nicht mehr zum Einsatz kommt. Dies erschwerten die Wartung und Weiterentwicklung erheblich.

Das Ersetzen dieser veralteten Lösung ist für Kinemotion von zentraler Bedeutung. Eine leistungsfähige, benutzerfreundliche und zukunftssichere CAN-Analyse-Software verbessert nicht nur die Diagnose- und Wartungsprozesse, sondern bringt auch wirtschaftliche Vorteile mit sich. Keine Lizenzkosten sowie die Unabhängigkeit von Drittanbietern und die Möglichkeit der Integration in die bestehende Softwareumgebung (KMWare).

Im Rahmen dieser Arbeit wurde ein neuer CAN Explorer entwickelt. Die Implementierung erfolgte mit C++ und dem Qt-Framework im Backend sowie QML im Frontend. Die Anwendung ermöglicht den Empfang und die Analyse von CAN-Nachrichten in Echtzeit, inklusive Filterfunktionen, tabellarischer und grafischer Darstellung. Besondere Schwerpunkte lagen auf einer intuitiven Benutzeroberfläche, einer modularen Softwarearchitektur sowie der einfachen Erweiterbarkeit. Die entwickelte Software wurde erfolgreich in realen Fahrzeugsystemen getestet und erfüllt alle von Kinemotion.

Navruz Jabbarov

Theme of the work

Visualizing and analyzing the CAN bus traffic flows of special vehicles

Keywords

Kinemotion, KMWare, CAN Explorer, Filter, Graphics, CAN-Message, Table, CAN-Bus, Qt, C++, QML

Abstract

This thesis presents the development of a modern software solution for the visualization and analysis of CAN bus messages in special-purpose vehicles. The project was initiated in response to limitations of the previously used CAN Explorer application, which was outdated, error-prone, and lacked proper documentation. Furthermore, it was implemented in C#, a programming language no longer used at Kinemotion, making maintenance and further development increasingly difficult.

Replacing this legacy software is of strategic importance to Kinemotion. A powerful, user-friendly, and future-proof CAN analysis tool not only improves the diagnosis and maintenance of vehicles but also offers economic benefits. No licensing costs, independence from third-party vendors, and seamless integration into the existing software environment (KMWare).

As part of this work, a new CAN Explorer was developed using C++ and the Qt framework for the backend, and QML for the frontend. The application supports real-time reception and analysis of CAN messages, including filtering, tabular and graphical visualization. Particular emphasis was placed on usability, modular software architecture, and ease of extensibility. The software developed has been successfully tested in real vehicle systems and meets all Kinemotion requirements.

Inhaltverzeichnis

Glossar

CAN Explorer	Das gesamte Projekt wird „CAN Explorer“ genannt.
QT	Qt ist ein Anwendungsframework und GUI-Toolkit zur plattformübergreifenden Entwicklung von Programmen und grafischen Benutzeroberflächen. Die Programmiersprache ist C++ unter Verwendung der Qt Bibliotheken. Das Signal- und Slot-System von Qt wird intensiv genutzt, um Interaktionen innerhalb der Anwendung zu ermöglichen.
Qt Creator	Qt Creator ist Entwicklungsumgebung von Qt.
QML	QML ist eine deklarative Programmiersprache, die als Bestandteil von Qt entwickelt wurde. Zweck der Sprache ist die Entwicklung von Benutzeroberflächen für Desktop- und Mobile-Systeme.
Kinemotion GmbH	Die Firma Kinemotion GmbH ist ein Unternehmen und spezialisiert für elektronisch-hydraulische Regelungstechnik im Schwer- und Sonderfahrzeugbau und mobile Anwendungen.
KMWare	KMWare ist eine Software von Kinemotion, welche für die Diagnose, Steuerung und Parametereinstellungen der Fahrzeuge spezialisiert ist. Die Anwendung wird sowohl als Desktopversion, als auch auf Displays in Sonderfahrzeugen integriert. Die Software ist bereits im Einsatz und wird zurzeit weiterentwickelt. Die Entwicklungsumgebung ist der Qt Creator und die Programmiersprache ist C++. Für die Benutzeroberfläche wird QML verwendet.

1 Einleitung

In der schnell wachsenden Softwarewelt gewinnen innovative Lösungen und das Motto „All-in-One“ zunehmend an Bedeutung. Kunden legen großen Wert auf Modernität, Funktionalität, Benutzerfreundlichkeit und Fehlerfreiheit. Um wettbewerbsfähig zu bleiben, ist es unerlässlich, Softwareprodukte kontinuierlich zu modernisieren und weiterzuentwickeln. Auch die Kinemotion GmbH verfolgt dieses Ziel.
Im Rahmen dieser Arbeit entsteht eine neue Software namens **"CAN Explorer"**, die speziell dafür entwickelt wird, CAN-Nachrichten nicht nur zu empfangen und zu bearbeiten, sondern auch übersichtlich grafisch darzustellen.
Im Unterschied zu bereits existierenden Tools verfolgt der CAN Explorer nicht nur das Ziel, veraltete Programme abzulösen. Vielmehr soll er mit erweiterten Funktionen, einer modernen, benutzerfreundlichen Oberfläche und einem durchdachten Bedienkonzept echten Mehrwert bieten. Bei der Entwicklung wird bewusst auf aktuelle technologische Standards und die konkreten Anforderungen aus der Industrie geachtet.

1.1 Motivation

Es gibt zahlreiche Anbieter wie Peak-System, Sontheim Industrie Elektronik GmbH und Vector Informatik GmbH, die Diagnose- und Analyse-Software für CAN-Bus-Nachrichten anbieten. Dennoch bietet die Entwicklung eines eigenen, modernen CAN-Explorers für die Überwachung und Diagnose von CAN-Bus-Systemen erhebliche technische und wirtschaftliche Vorteile.
Ein unternehmenseigenes Tool ermöglicht eine stärkere Kontrolle und Flexibilität bei der Entwicklung. Es fördert Innovationen und den Wissensaufbau im Unternehmen. Besonders hervorzuheben ist die schnelle Anpassbarkeit, die es erlaubt, das Tool bei Bedarf umgehend zu modifizieren.
Zusätzlich können spezialisierte Funktionen implementiert werden, um die Leistung zu optimieren und spezifische Anforderungen zu erfüllen. Ein weiterer Vorteil ist die volle Kontrolle über die Software, die Unabhängigkeit von Drittanbietern sowie die Vermeidung von Lizenzgebühren, was langfristig Kosteneinsparungen und eine höhere Effizienz gewährleistet.

1.2 Problemstellung

Folgende Schwierigkeiten und Bedürfnisse entstehen bei der Nutzung der aktuellen, veralteten CAN-Explorer App:

1. **Veraltete Software und fehlende Dokumentation**

Die bestehende Version des CAN-Explorers wurde vor etwa zehn Jahren in C# von einem ehemaligen Mitarbeiter entwickelt, der weder eine Dokumentation noch Kommentare hinterlassen hat. Dies erschwert die Wartung erheblich. Darüber hinaus entspricht die grafische Benutzeroberfläche nicht mehr den aktuellen Standards moderner Softwareentwicklung.

2. **Ausstieg aus der Programmiersprache C#**

In der Kinemotion GmbH wird C# mittlerweile nicht mehr verwendet, was die Pflege und Weiterentwicklung der Software zusätzlich kompliziert macht.

3. **Integration des neuen CAN-Explorers in KMWare**

Nach erfolgreicher Entwicklung und Testphase soll der CAN-Explorer in die bestehende Softwareplattform VMware integriert werden. Eine Integration der aktuellen Version ist jedoch aufgrund der unterschiedlichen Programmiersprachen nicht möglich.

4. **Fehlerhafte Ergebnisse und Softwarebugs**

Die aktuelle Version des CAN-Explorers weist mehrere Fehler auf, die unter anderem zu unzuverlässigen Ergebnissen führen. Diese Probleme beeinträchtigen sowohl die Genauigkeit als auch die Stabilität der Software und machen eine Überarbeitung unumgänglich.

1.3 Zielsetzung

Im Rahmen dieser Arbeit wird eine Anwendung entwickelt, die in der Lage sein soll, Diagnose und Analyse bei den Fahrzeugen von Kinemotion durchzuführen. Das Ziel ist, die alte CAN-Explorer App zu ersetzen und eine zu dem aktuelle KMWare integrierbare Anwendung zu implementieren. Bei der Entwicklung des CAN-Explorers soll beachtet werden, dass die Anwendung eine moderne Benutzeroberfläche hat, wodurch die Entwicklung, Wartung und Erweiterung der Software vereinfacht werden können. Darüber hinaus kann die aktualisierte Benutzeroberfläche die Benutzerfreundlichkeit verbessern und die Effizienz bei der Systemintegration und Fehlerbehebung erhöhen.
Ein weiterer wichtiger Punkt ist eine klare Softwaredokumentation. Diese hilft enorm, um zukünftige Entwickler in Kinemotion bei der Wartung und Weiterentwicklung zu unterstützen. Durch die Modernisierung der Softwarearchitektur und Benutzeroberfläche kann Kinemotion sicherstellen, dass ihre Software den aktuellen Anforderungen entspricht und zukünftigen Entwicklungen gewachsen ist.

1.4 Abgrenzung des Themas

Diese Arbeit konzentriert sich auf den lesenden Zugriff auf die CAN-Bus-Schnittstelle und die Darstellung der CAN-Nachrichten in der Tabelle. Ein weiterer zentraler Aspekt ist die Implementierung benutzerspezifischer Filter, die es ermöglichen, die angezeigten Nachrichten einzugrenzen.
Darüber hinaus wird die grafische Darstellung einzelner CAN-Nachrichten als wichtiger Bestandteil der Arbeit betrachtet. Die genauen Anforderungen und weitere Details wurden im Rahmen der Anforderungsanalyse konkretisiert.

1.5 Der Aufbau der Arbeit

Zuerst wird in dieser Arbeit vermittelt die theoretischen und technischen Grundlagen, die zum Verständnis der Arbeit erforderlich sind. Zudem wird eine Anforderungsanalyse mit Kinemotion durchgeführt, um die Ausgangssituation systematisch zu erfassen. In dem drauf folgenden Kapitel werden die Softwarearchitektur und die zugrunde liegenden technischen Konzepte erläutert und begründet. Darüber hinaus wird der zu untersuchende Use Case definiert. Verschiedene Lösungsansätze zur Umsetzung werden dargestellt, bewertet und ein geeigneter Ansatz ausgewählt.
Daraufhin beschreibt das nächste Kapitel die praktische Realisierung des gewählten Konzepts. Im fünften Kapitel wird eine Qualitätsüberprüfung durchgeführt, um zu beobachten, ob die Ergebnisse der Umsetzung den zuvor definierten Anforderungen entsprechen. Außerdem wird die entwickelte Software technisch umgesetzt und in einem realen Fahrzeugsystem getestet. Daraufhin werden die Resultate der Umsetzung systematisch ausgewertet und analysiert. Die dabei erfassten Daten werden mit den aufgestellten Kriterien verglichen, um die Qualität und Zielerreichung zu beurteilen. Abgeschlossen wird die Arbeit mit dem Fazit der Arbeit und einer abschließenden Reflexion über den Gesamtprozess.

2 Grundlagen

In diesem Abschnitt werden die wesentlichen Grundlagen der in dieser Arbeit behandelten Themenbereiche detailliert erläutert, genauer gesagt wird hier einmal der CAN-Bus erklärt und die CAN-Bus Analyse Tools vorgestellt.

2.1 CAN-Bus (Controller Area Network)

Ein Bussystem ist ein Kommunikationssystem, das verschiedene Komponenten eines elektronischen Systems miteinander verbindet, indem es ihnen ermöglicht, Informationen über eine gemeinsame Datenleitung auszutauschen.

Das CAN-Protokoll[1] wurde 1983 von Bosch entwickelt und 1986 zusammen mit Intel vorgestellt, um die Kommunikation zwischen verschiedenen Steuergeräten im Fahrzeug zu optimieren und die Zuverlässigkeit und Effizienz zu erhöhen [1].

Die Kommunikation erfolgt mit Telegrammen. Innerhalb eines Telegramms gibt es Steuerbits und Nutzbits. Der genormte Aufbau eines solchen Telegrammrahmens wird als „Frame“ bezeichnet. Es gibt vier verschiedene Arten von Frames: den Daten-, den Remote-, den Error- und den Overload-Frame. Der Daten-Frame dient dem Transport von Daten. Der Remote-Frame dient der Anforderung von Daten-Frames anderer Teilnehmer. Der Error-Frame signalisiert allen Teilnehmern eine erkannte Fehlerbedingung in der Übertragung und der Overload-Frame dient als Zwangspause zwischen Daten- und Remote-Frames [2].

Nun zum Aufbau von Frames. Ein Frame besteht aus sieben Kennfeldern: Start-Kondition, Message Identifier, Steuerbits, Daten (0-8 Bytes), Prüfbits, Acknowledge-Bit und einer Stop-Kondition. Außerdem unterscheidet man die Frames nach der Länge des Identifiers: Es gibt den Standardframe mit einem 11-Bit Identifier und den Extended Frame mit einem 29-Bit Identifier [3].

2.2 CAN Analyse Tools

In heutigen Industriewelt gibt es schon viele Anbieter, die performante und effiziente Software für die Analyse und Diagnose der Fahrzeuge anbieten. Ein Beispiel die Firma Vector Informatik. Mit ihrem Produkt „CANoe Familiy“ bietet das Unternehmen vielseitigen Softwarelösungen je nach Einsatzgebiet an. Die Einsatzgebiete umfassen unter anderem die Automobilindustrie, Nutzfahrzeuge, Sonderfahrzeuge, Eisenbahn- und Bahntechnik, sowie Medizintechnik, Luft- und Raumfahrttechnik, IoT und Smart Devices [4].

3 Konzept

Dieses Kapitel dient der Erläuterung des grundlegenden Konzepts für die zu entwickelnde Software. Ziel ist es, eine gute Grundlage für die spätere Umsetzung zu schaffen.
Im ersten Unterkapitel Anforderungsanalyse wurden zunächst die Gründe für den Bedarf an einer eigenen Softwarelösung untersucht. Anschließend wird die Rolle des CAN Explorers als Werkzeug für die betroffenen Benutzergruppen beschrieben. Es folgen die Unterkapitel für die funktionalen und nichtfunktionalen Anforderungen. Dort werden sowohl die konkreten Anforderungen an die App als auch die Qualitätsanforderungen beschrieben. Anschließend werden die technischen Anforderungen definiert, die die Wahl der Programmiersprache und die Hardware-Komponenten festlegen. Zum Schluss wird die Systemarchitektur basierend auf Anwendungsfällen, Endarchitektur und den zugehörigen Diagrammen vorgestellt.

3.1 Anforderungsanalyse

Die Abbildungen 3.1 und 3.2 zeigen die Benutzeroberfläche der Diagnosesoftware von CANoe/pro [5] für den Desktop. Dabei ist zu sehen wie viel Komplexität das Tool durch die verschiedenen Bedienungsmöglichkeiten mitbringt. Die Anwendung enthält des Weiteren umfangreiche Funktionen, sowie erweiterte Diagnose- und Testmöglichkeiten. Dies ist zwar nützlich, stellt jedoch erhöhte Herausforderungen an den Umgang mit dieser Software.

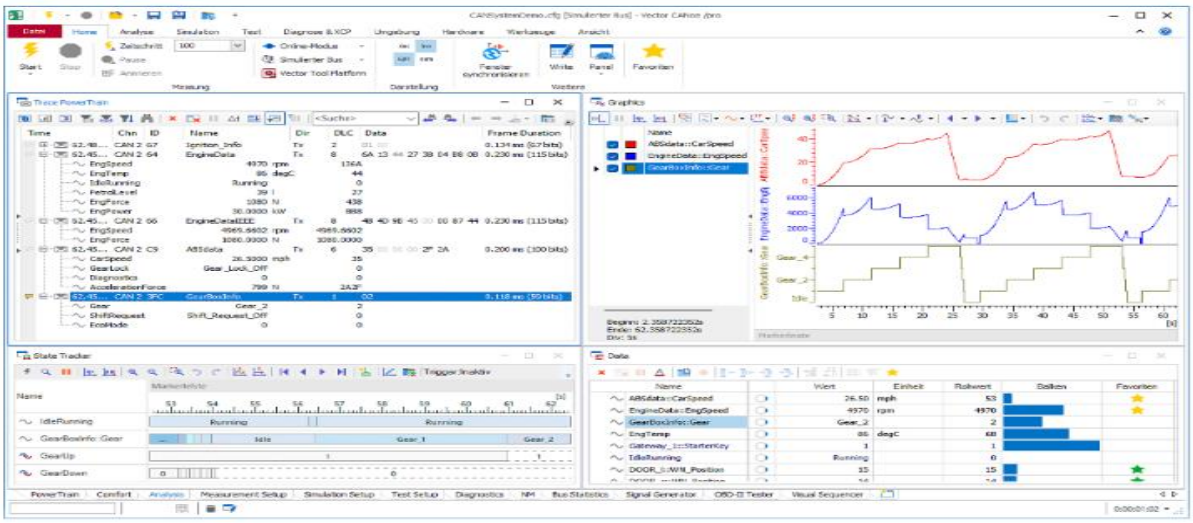


Abbildung 3.1: Das Analyse Fenster von Vector CANoe /pro.

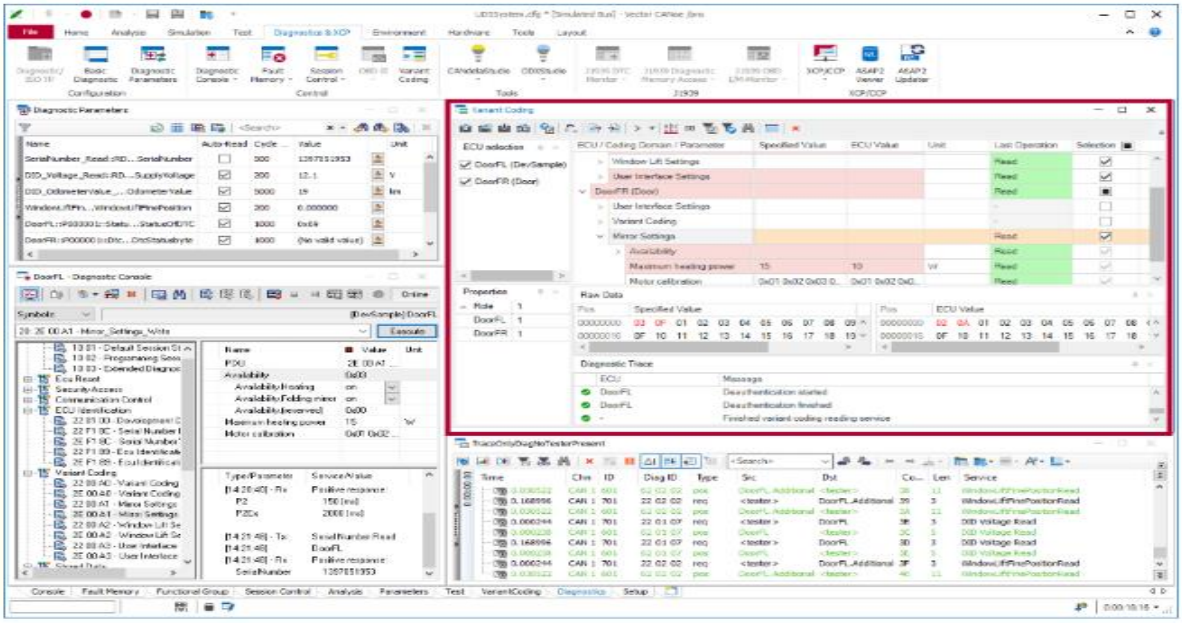


Abbildung 3.2: Das Diagnose Fenster von Vector CANoe /pro.

Die Entscheidung von Kinemotion gegen den Einsatz einer Standardsoftware und für die Eigenentwicklung basiert auf folgenden Gründen:

Komplexität

Die Vielseitigkeit und die umfangreichen Funktionalitäten von CANoe oder von anderen Tools können das Zurechtfinden für Benutzer herausfordernd machen. Das Tool bietet viele Optionen, die es notwendig machen, sich gut einzuarbeiten, um die Software effizient zu nutzen. Eine komplexe Benutzeroberfläche und viele konfigurierbare Funktionen erfordern oft eine längere Lernphase, um die Software vollständig zu verstehen. Die Endkunden von Kinemotion GmbH benötigen eine intuitive und einfach

bedienbare Softwarelösung, die ohne Schulungsaufwand oder Rückfragen an unser Unternehmen genutzt werden kann. Die Software sollte selbsterklärend und klar strukturiert sein, sodass die Nutzer sie ohne technische Vorkenntnisse problemlos einsetzen können.

Keine Abhängigkeit von Drittanbietern

Wenn Kinemotion seine Software selbst entwickelt, müssen sie sich nicht nach Plänen externer Anbieter richten, wenn es um Updates, neue Lizenzmodelle oder veränderten Support geht. Dadurch lassen sich Ausfälle oder Verzögerungen, die durch schlechten oder langsamen Kundensupport entstehen, oft vermeiden.

Schnelle Implementierung neuer Funktionen

Wenn Änderungen nötig sind, kann dies bei Standardsoftware nur mit Absprache mit dem Hersteller gemacht werden. Dahingegen lassen sich bei Eigenentwicklung diese intern umsetzen, ohne Wartezeiten oder Abhängigkeit von externen Partnern. Das macht das die Kinemotion flexibler und hilft, schneller auf technische Entwicklungen oder neue Bedürfnisse zu reagieren.

Speicherplatz und CPU-Ressourcen

Da die fertigen Softwaren wie CANoe sehr viele Funktionen und umfangreiche Datenverarbeitungsprozesse bietet, benötigt sie viel Speicherplatz und hohe CPU-Ressourcen, um effizient zu laufen. Zum Beispiel ist die Version 19 der Software von CANoe [6] ist schon über 5 GB groß. Das zeigt, wie umfangreich und ressourcenintensiv die Software ist. In dem Entwicklungsumfeld der Firma arbeiten die Entwickler parallel mit mehreren Anwendungen, daher ist es entscheidend, dass die eingesetzte Software möglichst wenig Speicherplatz beansprucht. Zudem ist geplant, das Tool als Bestandteil von CAN Explorer in die KMWare zu integrieren. KMWare wird auch in Displays aufgespielt und die Displays verfügen nur begrenzter Speicherkapazität. Daher ist es unerlässlich, dass die Software kompakt, effizient und auf das Wesentliche reduziert ist, um unnötige Speicherbelegungen zu vermeiden.

Kosteneinsparungen.

Softwarelösungen wie die CANoe-Produktlinie der Firma Vector Informatik – sind in der Regel mit hohen Lizenz- und Folgekosten verbunden. Laut einer persönlichen Rücksprache mit der Vertriebsabteilung von Vector (Juni 2025) liegen die Anschaffungskosten für eine Einzelplatzlizenz der aktuellen Version 19 bei etwa 12.000 Euro. Hinzu kommen optionale, aber in der Praxis oft notwendige Wartungs- und Supportgebühren in Höhe von 2.000 Euro pro Jahr und Lizenz.

Langlebigkeit
Die zu entwickelnde Softwarelösung muss langfristig an die Ziele und Entwicklungen des Unternehmens anpassbar sein. Es soll bei der Entwicklung unbedingt darauf geachtet werden, die nachhaltige Weiterentwicklung und Wartung zu ermöglichen.

Integration
Die Softwarelösung muss eine nahtlose Integration mit dem internen Desktopsystem KMWare ermöglichen, da dies wichtig für die Analyse von Fahrzeugnachrichten im Fahrzeug beim Endkunden ist. Eine Anbindung von Software anderer Anbieter ist technisch nicht möglich, weshalb eine eigenentwickelte Lösung zwingend erforderlich ist, um die spezifischen Schnittstellen und Abläufe innerhalb von KMWare vollständig zu unterstützen.

Langfristige Investition
Durch die Entwicklung eigener Softwarelösungen baut Kinemotion internes Wissen und technische Kompetenz im Unternehmen auf. Dieses Know-how stärkt nicht nur Unabhängigkeit, sondern ermöglicht es uns auch, schneller und effizienter auf neue Anforderungen oder Veränderungen zu reagieren. Langfristig steigert das die Wettbewerbsfähigkeit und reduziert die Abhängigkeit von externen Dienstleistern.

3.2 Zielgruppe

Die Zielgruppe der Anwendung CAN Explorer sind folgende Benutzergruppen: Softwareentwickler, Ingenieure und Servicetechnikern. Diese Benutzer sollen CAN-Bus-Daten analysieren und visualisieren. Dies umfassen das Lesen, Dekodieren und Darstellen der Daten in einer benutzerfreundlichen Weise. Folgend werden einmal die Benutzergruppen näher analysiert:

1. Fahrzeugingenieure
Fahrzeugingenieure sind Fachkräfte der Kinemotion GmbH sowie ihrer Partner aus dem Maschinenbau. Sie sind maßgeblich an der Entwicklung, Integration und Optimierung der Fahrzeugtechnik beteiligt und sollen das Tool gezielt zur Analyse und Systemvalidierung einsetzen.

2. Embedded-Systems-Entwickler

Alle Systementwickler von Kinemotion GmbH, die bei Kinemotion arbeiten und Software entwickeln.

3. Servicetechniker
Servicetechniker sind die verantwortlichen Fachkräfte bei verschiedenen Endkunden, die direkt mit den Fahrzeugen arbeiten. Sie nutzen das Tool im Alltag zur Diagnose,

Wartung und Fehleranalyse. Ihr Feedback ist entscheidend für die Weiterentwicklung und Praxistauglichkeit der Software.

3.3 Funktionale Anforderungen

Die funktionalen Anforderungen sind in fünf Bereiche gegliedert: die Datenaufnahme, die Datenanzeige, dem Filterfenster für die Nachrichtenfilterung und dem Grafikfenster für Datenanalyse und Visualisierung der CAN-Nachrichten. Diese werden jetzt folgend erläutert.

1. Datenaufnahme

Die App muss in der Lage sein, CAN-Daten in Echtzeit von einem CAN-Bus über eine CAN-USB Schnittstelle zu empfangen. CAN-Nachrichten aus einer Textdatei einlesen wird im Rahmen dieser Arbeit nicht betrachtet.

2. Datenanzeige

Starten, Stoppen und Reset der Aufnahme von CAN-Nachrichten.

Die Erfassung der CAN-Nachrichten muss gestartet, gestoppt und zurückgesetzt werden. Sobald das Tool läuft, beginnt die Tabellenansicht automatisch mit der Aufzeichnung. Ein „Start/Stop“-Button ermöglicht das Anhalten und erneute Fortsetzen der Aufzeichnung, während ein separater „Reset“-Button die aktuelle Aufzeichnung zurücksetzt.

Darstellung der CAN-Nachrichten in Form einer Tabelle.

Die aufgenommenen CAN-Nachrichten müssen für die Benutzer in Form einer Tabelle visualisiert werden. Eine gewünschte Form ist in Tabelle 3.1 dargestellt.

Nachricht-name	ObjektID (NachrichtID)	D0	D1	D2	D3	D4	D5	D6	D7	Zeit Stempel	Zähler
Lokal_LHV	000E0152	08	01	00	F0	12	FA	01	02	0.150	536
…	….	..	..	..	..	..	..	..	..	..	..

Tabelle 3.1: Eine Tabelle für die CAN-Nachrichten Ausgabe.

Die Tabellenspalten werden folgend erläutert:

Objekt ID (Nachricht ID)

Jede ankommende CAN-Nachricht soll mit seiner Nachrichten-ID in der Tabelle angezeigt werden.

Objektname

Es soll eine Spalte für die Namenvergabe von CAN-Nachrichten bereitgestellt werden,

um bei der zukünftigen Weiterentwicklung Nachrichtenamen möglichst einfach zu vergeben.

D0…D7

In den Spalten D0 bis D7 sollen der Inhalt der CAN-Nachrichten dargestellt werden. Diese Nutzdaten sind maximal 8 Byte groß. Jede Spalte repräsentiert jeweils ein Byte.

Zeitstempel

Diese Spate ist für den Zeitstempel einzelner Nachrichten vorgesehen. Es soll sowohl der relative als auch absolute Zeitstempel angezeigt werden.

Zähler

Diese Spalte soll die Anzahl der einzelnen empfangenen Nachrichten repräsentieren.

3. Funktionalitäten

Format der Nutzbytedaten.

Die Darstellung der Rohdaten soll variabel sein, sodass diese in Hexadezimal, Dezimal oder in ASCII- Zeichen angezeigt werden können. Darüber hinaus soll dem Nutzer ein Wechsel zwischen den Formaten möglich sein.

Nachrichtenanzeige in der Tabelle.

Es soll möglich sein, CAN-Nachrichten in zwei verschiedenen Formaten darzustellen. Die erste Möglichkeit beinhaltet eine relative Nachrichtenanzeige. Dabei überschreibt jede neueingelesene Nachricht ihren Vorgänger und nur die Nachrichtenanzahl wird hochgezählt. Die zweite Möglichkeit umfasst eine absolute Nachrichtenanzeige. Die absolute Nachrichtenanzeige zeigt jede neuangekommene CAN-Nachricht in einer neuen Zeile der Tabelle an.

Zeitstempel der CAN-Nachrichten

Es sollen zwei verschiedene Zeitstempel-Stile programmiert werden: der relative und der absolute Zeitstempel. Der relative Zeitstempel gibt den Zeitpunkt eines Ereignisses im Verhältnis zu einem anderen, referenzierten Zeitpunkt an, sodass nur eine Zeitspanne oder Differenz angezeigt wird und kein absoluter Zeitpunkt.

Der absoluter Zeitstempel ist eine präzise, unveränderliche Angabe eines bestimmten Zeitpunkts, der durch Datum und Uhrzeit eindeutig definiert ist. Dieser Zeitstempel gibt den genauen Zeitpunkt eines Ereignisses an, unabhängig von späteren Veränderungen oder dem aktuellen Zeitpunkt.

Anzeige die Gesamtanzahl CAN-Nachrichten.

Es soll ein kleines Fenster implementiert werden, um die gesamte Anzahl der CAN-Nachrichten anzuzeigen, die nach dem Start angekommen sind. Darüber hinaus soll jede einzelne Nachrichtenanzahl nach der Startzeit hochgezählt werden.

Die Auslastung der CAN-Busses

Die aktuelle Auslastung des CAN-Busses in Prozent soll im Hauptfenster für die User sichtbar sein.

Nachrichtenlange

Die CAN-Bus Nachrichten (Objekt ID) sind entweder 3 oder 8 Byte lang**.** 3 Byte lange Nachrichten sind Standardnachrichten und 8 Byte lange Nachrichten heißen Extended Nachrichten. Die Anwendung soll in der Lage sein beide Nachrichtentypen einzulesen und in der Tabelle darzustellen.

4. Filterfenster für die Nachrichtenfilterung

Durch eine Filtermöglichkeit soll es dem Benutzer möglich sein nur bestimmte CAN-Nachrichten anzuzeigen. Es soll ein Fenster für das Erzeugen eines Filters geben. Alle erzeugten Filter sollen in einer separaten Filtertabelle angezeigt werden. Der CAN Explorer soll zwei unterschiedliche Filtermöglichkeiten besitzen: Der Bereichsfilter und der Masken/ID-Filter.

Bereichsfilter

Für den Bereichsfilter sollen zunächst ein Bereich aus Start- und Endpunkt definiert werden. Wenn dieser Filter angewendet wird, werden nur Nachrichten durchgelassen, die in diesem Bereich liegen. Alle andere CAN-Nachrichten werden blockiert, sobald der Filter aktiv ist.

Masken/ID-Filter

Bei dem ID-Filter wird der Bytebereich von einer Objekt ID genau ausgewertet und beim Masken-Filter sollen nur Nachrichten, die auf die Maske passen, durchgelassen werden.

Jedoch stehen der Masken- und ID-Filter in einem Zusammenhang, denn es soll ein Entweder- Oder-Prinzip gelten, welches kleine Bereiche durch Größere überschriebt.

Filter speichern, entfernen importieren und exportieren

Der Benutzer soll Filter permanent speichern können und die gespeicherten Filterliste soll jederzeit verfügbar sein, beispielsweise nach einem Neustart. Außerdem sollen die erzeugten Filter wieder aus der Filterliste entfernt werden können.

Um die Filtertabelle mit anderen Benutzern teilen zu können soll die Filterliste in eine Textdatei gespeichert werden können. Um dieses Ziel zu erreichen, soll das Tool um einen Button „Exportieren“ und einen Button „Importieren“ erweitert werden, welches jedoch nicht im Rahmen dieser Arbeit umgesetzt werden soll.

5. Grafikfenster für Datenanalyse und Visualisierung der CAN-Nachrichten

Echtzeitnutzdaten sollen in einem Grafikfenster auf der Nutzeroberfläche zur Analyse visualisiert werden. Voraussetzung ist, dass mindestens vier verschiedene CAN-Nachrichten mit Nutzdaten geleichzeitig in einem Graphen dargestellt werden können. Dabei soll die X-Achse Zeit und die Y-Achse die Werte widerspiegeln.

Es soll außerdem ein Eingabefeld erstellt werden, womit der Benutzer gewünschte CAN-Nachrichten eintippen kann, um diese anzuzeigen. Dabei soll der User auch die Anzahl und die Stelle der Bytes auswählen können. Mögliche Auswahlmöglichkeiten für die Bytes sind 1 Byte, 2 Bytes, 3 Bytes und 4 Bytes.

Die Skalierungsmöglichkeit der X-Achse soll in 1, 2 ,5 oder 10 Sekunden möglich sein.

Bei der Y-Achse sollen Änderungsmöglichkeiten des Maximalwerts für die Nutzdaten programmiert werden. Außerdem soll eine Wahl zwischen signed und unsigned für den Maximalwert möglich sein. Eine weitere Anforderung an die App soll die Auswahl zwischen Motorola (Big-Endian) oder Intel (Little-Endian) sein. Diese Funktion soll die Ausrechnung der Nutzbytes möglich machen.

3.4 Nicht funktionale Anforderungen

Nicht funktionale Anforderungen sind essenziell, um die grundlegenden Fähigkeiten und Merkmale einer App zu definieren. Diese werden folgend beschrieben:

Leistung

Die App soll in der Lage sein, kontinuierlich 500 bis 3000 Nachrichten pro Sekunde in Echtzeit zu verarbeiten. Dabei soll die Verzögerung bei der Anzeige und Analyse der Daten stets im Bereich von nur 0,1 bis 0,5 Sekunden liegen.

Zuverlässigkeit

Die App muss auch bei sehr hohen Datenraten und großen Datenmengen dauerhaft stabil und zuverlässig arbeiten, ohne dass es zu Leistungsverlusten oder Ausfällen kommt.

Skalierbarkeit

Der User soll die Möglichkeit haben das Fenster der App dynamisch ändern zu können und die Komponenten sollen sich auch dementsprechend vergrößern oder verkleinern.

Benutzeroberfläche

Der Benutzeroberfläche soll dem User eine einfache Bedienung ermöglichen. Die Benutzerfreundlichkeit ist eine besonders wichtige Anforderung, da der Vorgänger hier besonders viele Probleme hatte. Eine weitere wichtige Anforderung ist, dass die

Oberfläche modern aussehen soll, um aktuelle Industrie- und Kundenstandards zu erfüllen.

Integration zu KMWare

Wie bereits erwähnt soll der CAN-Explorer zukünftig in die bestehenden Displaysoftware „KMWare“ integriert werden. Diese Anforderung soll bei allen Entscheidungen berücksichtigt werden, damit die abschließende Integration mit möglichst wenig Aufwand gelingt.

Benutzeranleitung

Ein wesentlicher Punkt. der Arbeit ist, dass eine gut lesbare Benutzeranleitung geschrieben werden soll. Diese soll sowohl Fahrzeugingenieuren als auch für Endkunden zur Einarbeitung helfen. Außerdem soll der Programmcode ausführliche Kommentare enthalten, damit das Programm leicht weiterentwickelt werden kann.

3.5 Technische Anforderungen

Kinemotion setzt voraus, dass die Implementierung der Software mit der Programmiersprache C++ unter Verwendung des Qt-Frameworks erfolgen soll. Für die Entwicklung der grafischen Benutzeroberfläche (GUI) kommt QML zum Einsatz.
Zudem wird ein CAN-Dongle zur Kommunikation über den CAN-Bus der Firma Sontheim eingesetzt. Nach erfolgreicher Implementierung wird die Software bei einem echten Fahrzeugsystem in der Werkstatt von Kinemotion getestet.

3.6 Systemarchitektur

Das Konzept robuster Anwendungsfälle und einer Systemarchitektur bildet das Fundament jeder erfolgreichen Softwareentwicklung. Eine saubere Struktur und klare funktionale Abläufe sind nur dann von Wert, wenn das System selbst korrekt umgesetzt ist. Denn *„Korrektheit als innere Qualität ist eine der wichtigsten Eigenschaften einer Software – das Erfüllen weiterer Qualitätsmerkmale ist nutzlos, wenn ein System falsch konstruiert, ist*“ [7]. Für die Erstellung und Gestaltung sämtlicher in dieser Arbeit verwendeter UML-Diagramme wurde auf das Fachwissen aus den folgenden Fachbüchern: *„UML 2.5 – Das umfassende Handbuch“* [8] und *„UML 2 glasklar – Praxiswissen für die UML-Modellierung“* [9] zurückgegriffen. Im Folgenden werden zentrale Anwendungsfälle erläutert und die Systemarchitektur vorgestellt, die als Grundlage für eine korrekte und wartbare Softwarelösung dienen. Anschließend folgen sowohl ein Komponenten-, als auch ein Klassendiagramm, um die Softwarelösung weiter zu beschreiben.

Anwendungsfälle

Zur strukturierten Darstellung der funktionalen Anforderungen an das System wurde ein Use-Case-Diagramm erstellt. Dieses veranschaulicht die Interaktionen zwischen verschiedenen externen Akteuren und den Hauptfunktionen des Systems. Aus der Anforderungsanalyse wurde folgendes Anwendungsdiagramm (Abbildung 3.3) erstellt.

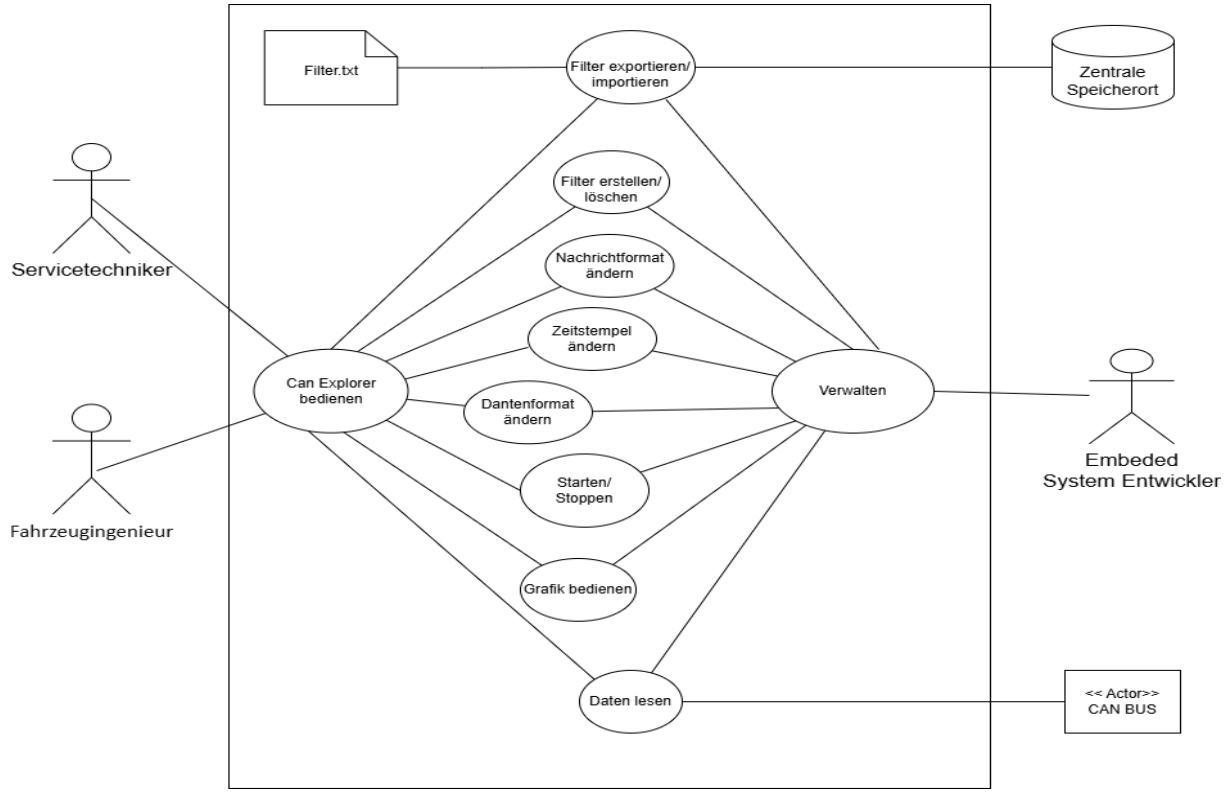


Abbildung 3.3: Anwendungsdiagramm des CAN Explorers.

Hier werden zunächst alle identifizierten Akteure beschrieben. Neben menschlichen Benutzern werden auch technische System beschrieben. Danach kommt die detaillierte Beschreibung der einzelnen Anwendungsfälle. Ziel ist es, ein klares Verständnis darüber zu schaffen, wer mit dem System wie interagiert und welche Funktionalitäten dabei unterstützt werden. Folgend werden alle betroffenen Akteure beschrieben:

1. **Fahrzeugingenieure**

Fahrzeugingenieure sind Fachkräfte der Kinemotion GmbH sowie ihrer Partner aus dem Maschinenbau. Sie sind maßgeblich an der Entwicklung, Integration und Optimierung der Fahrzeugtechnik beteiligt und setzen das Tool gezielt zur Analyse und Systemvalidierung ein.

2. **Embedded-Systems-Entwickler**

Der Embedded System Entwickler nutzt das System in einem erweiterten technischen Kontext. Er analysiert die Diagnosedaten, testet Systemverhalten in Entwicklungsumgebungen und analysiert die Fehler. In den meisten Fällen bei der Kinemotion GmbH ist ein Mitarbeiter sowohl Fahrzeugingenieur als auch Embedded-System-Entwickler.

3. **Servicetechniker**

Servicetechniker sind die verantwortlichen Fachkräfte bei verschiedenen Endkunden, die direkt mit den Fahrzeugen arbeiten. Sie nutzen das Tool im Alltag zur Diagnose, Wartung und Fehleranalyse. Sie haben Zugriff auf eine Vielzahl von Funktionen, jedoch keinen administrativen Zugriff, wie z. B. auf die vollständige Verwaltung oder Änderung systemkritischer Einstellungen. Ihr Feedback ist entscheidend für die Weiterentwicklung und Praxistauglichkeit der Software.

4. **CAN-Bus**

Der CAN-Bus ist ein weiterer technischer Akteur, der die Kommunikation mit dem eingebetteten Fahrzeugsystem ermöglicht. Das System liest über den CAN-Bus Nachrichten aus dem Fahrzeug aus oder schreibt entsprechende Daten hinein.

5. Zentrale Speicherort

Der zentrale Speicherort ist ein technischer Akteur und ist eine externe Speichereinheit von der Kinemotion. Über diesen Speicherort können Filterdateien exportiert oder importiert werden. Der Akteur nimmt also eine passive Rolle ein, indem er lediglich Daten empfängt oder bereitstellt, jedoch keine eigene Logik oder Entscheidungsfindung besitzt.

Nach der Beschreibung aller Akteure folgt die Beschreibung der einzelnen Use-Cases aus dem Diagramm in Abbildung 3.3.

1. Filter exportieren/importieren

Servicetechniker und Fahrzeugingenieure haben die Möglichkeit, vorhandene Filterkonfigurationen zu exportieren oder neue Filter zu importieren. Dies ermöglicht einen effizienten Austausch von Voreinstellungen und erleichtert die Wiederverwendung in ähnlichen Diagnoseszenarien. Die Filterdateien werden dabei in einem zentralen Speicherort abgelegt bzw. von dort geladen. Dieser Use-Case ist nur der Vollständigkeit halber aufgeführt, wird in dieser Arbeit jedoch nicht umgesetzt.

2. Filter erstellen/löschen

Über die Verwaltungsoberfläche können Nutzer eigene Filter definieren oder nicht mehr benötigte Filter entfernen. Diese Funktion hilft dabei, die Anzeige von CAN-Nachrichten gezielt auf relevante Inhalte zu begrenzen.

3. Nachrichtenformat ändern

Benutzer können das Format der empfangenen CAN-Nachrichten anpassen, beispielsweise durch die Umschaltung zwischen der Rohdaten- und der Interpretierten Ansicht. Diese Flexibilität ist besonders im Entwicklungs- und Wartungskontext nützlich.

4. Zeitstempel ändern

Für die präzise Nachverfolgung von Nachrichtenverläufen kann das Zeitstempelformat angepasst werden. Dies erlaubt eine besser strukturierte zeitliche Analyse der übertragenen Daten.

5. Datenformat ändern

Der CAN Explorer ermöglicht die Konfiguration des Datenformats (z. B. dezimal, hexadezimal), um eine möglichst verständliche Darstellung der Nachrichtendaten sicherzustellen.

6. Starten/Stoppen

Der Anwender kann die Erfassung der CAN-Nachrichten manuell starten oder stoppen. Dies ist vor allem bei der gezielten Analyse bestimmter Ereignisse oder Testläufe wichtig.

7. Grafik bedienen

Für eine visuelle Auswertung steht eine grafische Oberfläche zur Verfügung, die von den Nutzern bedient werden kann. Diese zeigt z. B. Signalverläufe, Zustandsänderungen oder Fehlercodes über die Zeit an.

8. Daten lesen/schreiben

Sowohl Servicetechniker, Fahrzeugingenieure als auch Embedded System Entwickler interagieren mit dieser zentralen Funktion, um Daten vom CAN-Bus zu empfangen oder gezielt Nachrichten zu senden.

9. Verwalten

Die Funktion „Verwalten“ dient als zentrales Steuerungselement zur Konfiguration aller genannten Funktionen. Sowohl Servicetechniker als auch Embedded Entwickler greifen hierauf zu, um systemweite Einstellungen vorzunehmen.

Architektur

In diesem Unterkapitel werden verschiedene Softwarearchitekturen vorgestellt und ihrer Vor- und Nachteile bewertet. Ziel ist es, eine passende Architektur für die vorliegende Arbeit zu finden.

Die Software-Architektur eines Systems „*dessen Software-Struktur respektive dessen - Strukturen, dessen Softwarebausteine sowie deren sichtbare Eigenschaften und Beziehungen zueinander*“ [10]. Somit stellt die Software-Architektur die zentrale Grundlage für den langfristigen Erfolg des im Rahmen dieser Arbeit entwickelten CAN Explorers dar. Sie definiert die grundlegende Struktur der Anwendung und legt fest, wie zum Beispiel der Nachrichtenempfang, die Filterung, die Darstellung und die Analyse von CAN-Nachrichten interagieren. Diese klare Trennung von Verantwortlichkeiten erleichtert nicht nur das Verständnis des Systems, sondern bildet auch die Basis für eine effiziente Weiterentwicklung.

Für das Projekt wurden zwei passende Architekturen im Vorfeld ausgewählt und eingehend analysiert. Im Hinblick auf die definierten Anforderungen lässt sich feststellen, dass folgende Architekturen grundsätzlich in Frage kommen: die ereignisgesteuerte Architektur und die Pipeline-Architektur.
Diese werden im Folgenden näher erläutert.

Ereignisgesteuerte Architekturen

Diese Architektur ermöglicht eine lose Kopplung zwischen Anwendungen. Sie müssen nicht direkt miteinander sprechen, sondern reagieren nur auf veröffentlichte Ereignisse. Im Vergleich zu klassischen Anfrage-Antwort-Modellen sind solche Architekturen flexibler, skalierbarer und leichter an neue Anforderungen anpassbar [10].

Ein zentraler Vorteil liegt in der asynchronen Verarbeitung eingehender CAN-Nachrichten. Sobald eine Nachricht vom Bus empfangen wird, löst sie ein entsprechendes Ereignis aus, das gezielt die dafür zuständigen Komponenten wie Filter oder Tabellen benachrichtigt. Dadurch entfällt die Notwendigkeit periodischer Abfragen, was die Systemlast reduziert und die Reaktionszeit erheblich verbessert.

Obwohl ereignisgesteuerte Architekturen die Verarbeitung von Hochgeschwindigkeitsdaten in Echtzeit ermöglichen, sind sie komplex in der Umsetzung. Die Verwaltung von kontinuierlichen Datenströmen beansprucht viele Ressourcen [11].

Außerdem In dieser Arbeit liegt der Fokus auf einem einzelnen Fahrzeug und somit auf einem lokal begrenzten System. Die beteiligten Komponenten befinden sich physisch

am selben Ort und können direkt miteinander kommunizieren. Eine ereignisgesteuerte Architektur, die vor allem in verteilten Systemen ihre Stärken ausspielt, ist hier nicht zwingend erforderlich.

Pipeline-Architektur

Die Pipeline-Architektur ist ein Konzept der Datenverarbeitung und überzeugt durch seine klare Struktur und hohe Modularität. Im Rahmen der Entwicklung des CAN Explorers erwies sich dieser Ansatz als besonders geeignet, da sich die Verarbeitung von CAN-Nachrichten in einer Abfolge logisch aufeinanderfolgender Schritte gliedern lässt.

Der Datenfluss innerhalb der Anwendung ist in klar definierte Verarbeitungsstufen unterteilt. Der erste Schritt ist das Einlesen von Rohdaten des CAN-Busses. Anschließend erfolgt die Filterung relevanter Nachrichten, gefolgt von der Umwandlung und Aufbereitung der Daten für die tabellarische und grafische Darstellung. Weitere Stufen umfassen das optionale Schreiben und Laden von Filterlisten aus und in Textdateien. Jede dieser Stufen kann als eigenständiges Modul implementiert werden. Die Daten werden unmittelbar nach ihrem Eintreffen erarbeitet, bevor sie an die nächste Verarbeitungsstufe weitergegeben werden.

Ein wesentlicher Vorteil dieser Architektur besteht in ihrer Modularität. Da jede Stufe eine klar abgegrenzte Aufgabe übernimmt, lassen sich einzelne Komponenten unabhängig voneinander entwickeln, testen und oder auch erweitern. Dies erleichtert nicht nur die Wartung der Software, sondern verbessert auch die Nachvollziehbarkeit des Datenflusses innerhalb der Anwendung [12].

Auch die Fehlersuche gestaltet sich durch die Struktur der Pipeline deutlich effizienter. Da jeder Verarbeitungsschritt isoliert betrachtet werden kann. Zwischenstände können getrennt analysiert werden, um die Fehlerquellen schnell zu identifizieren.

Insgesamt bietet die Pipeline-Architektur für den CAN Explorer eine ideale Grundlage für eine robuste, erweiterbare und wartungsfreundliche Softwarelösung, die sowohl den aktuellen Anforderungen gerecht wird als auch zukunftssicher weiterentwickelt werden kann.

Um die modulare Struktur der Anwendung und der allgemeine Verlauf der CAN-Bus Daten Bearbeitung besser zu veranschaulichen, wurde die gesamte Datenverarbeitung in einer Pipeline-Architektur (Abbildung 3.4) dargestellt. Die Grafik zeigt die einzelnen Verarbeitungsschritte sowie deren logische Abfolge. Dabei sind die einzelnen Stufen und deren Anordnung nicht an eine spezifische Norm gebunden. Die Pfeile zwischen den Stufen symbolisieren ausschließlich des sequenziellen Übergangs von einem Verarbeitungsschritt zum nächsten und besitzen keine logische Bedeutung.

Für die Erstellung aller UML-Diagramme wurde die Anwendung *draw.io* verwendet.

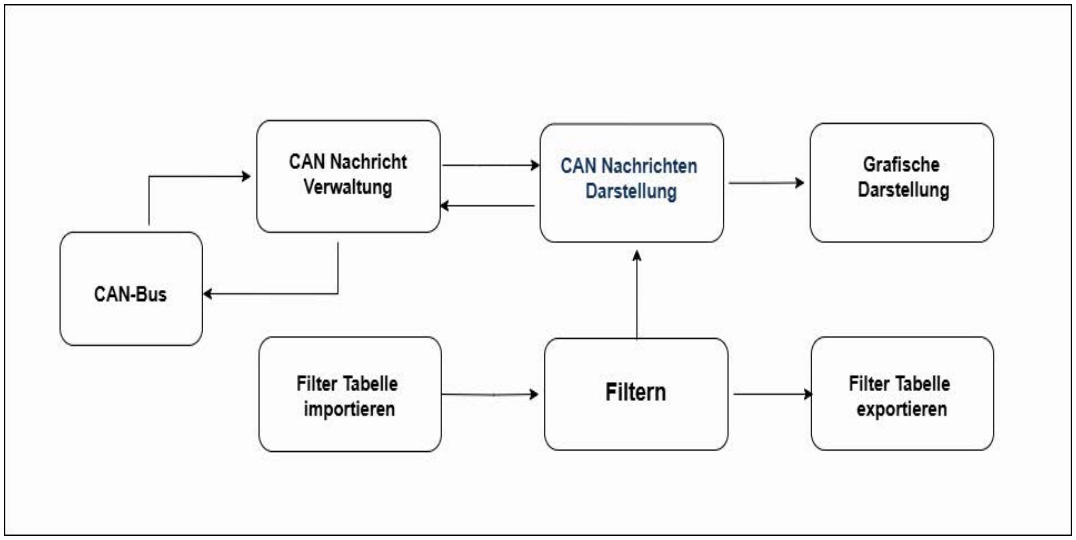


Abbildung 3.4: Pipeline Architektur des CAN Explorers.

Um die Funktionsweise und den Aufbau des Systems nachvollziehbar darzustellen, ist es hilfreich, die einzelnen Komponenten der Pipeline-Architektur im Detail zu beschreiben, da es sich dabei um eine selbstentworfene Struktur handelt, die nicht auf einem standardisierten Modell basiert. Deshalb ist es wichtig die einzelnen Stufen zu beschreiben. Sie ermöglicht ein besseres Verständnis der Datenflüsse und der modularen Logik hinter der Anwendung. Folgend sind die einzelnen Stufen beschrieben:

CAN-Bus

Der CAN-Bus stellt die physikalische Verbindung zum Fahrzeug bzw. zur Datenquelle dar. Hierüber werden Nachrichten empfangen und gesendet.

CAN Nachricht Verwaltung

Diese Komponente bildet die Schnittstelle zwischen dem CAN-Bus und der internen Verarbeitung. Sie empfängt Nachrichten vom Bus, verwaltet sie und leitet sie an andere Systemkomponenten weiter.

CAN Nachrichten Tabelle

Alle empfangenen oder geladenen Nachrichten werden in dieser Tabelle strukturiert abgelegt. Sie dient als zentrales Element für Visualisierung.

Filtern

Mithilfe definierter Filterregeln werden eingehende Nachrichten geprüft und gefiltert. Nur relevante Nachrichten werden anschließend weiterverarbeitet. Diese Filter können

über die GUI per Eingabe der User oder durch das Laden einer Filtertabelle festgelegt werden.

Filtertabelle

Diese Komponente ermöglicht das Speichern oder Laden von Filterregeln in Textdateien. Dies erhöht die Wiederverwendbarkeit und vereinfacht das Anpassen der Filterkriterien für verschiedene Anwendungsfälle.

CAN-Nachricht senden
Über diese Funktion werden CAN-Nachrichten aus der Tabelle auf den CAN-Bus zurückgesendet. Dies ist insbesondere für Test- oder Simulationszwecke sehr hilfreich.

Grafische Darstellung
Die in der Tabelle verwalteten CAN-Nachrichten können visuell aufbereitet und in grafische Form als Zeitdiagramme dargestellt werden. Dadurch wird eine schnelle Analyse und Bewertung der Daten ermöglicht.

Komponentendiagramm

Die zuvor dargestellte Pipeline-Architektur vermittelt bereits einen guten Überblick über die Hauptverarbeitungsschritte und deren Reihenfolge. Um die Architektur des Systems noch genauer zu verstehen, ist eine reine Ablaufdarstellung nicht genug. Besonders die Zusammensetzung einzelner Komponenten und deren Kommunikation untereinander können durch eine ergänzende strukturelle Betrachtung deutlich besser nachvollziehen.

Aus diesem Grund wird im nächsten Schritt ein Komponentendiagramm (Abbildung 3.5) verwendet, dass die internen Strukturen und Abhängigkeiten der Anwendung zeigen. Ziel dabei ist es, die einzelnen Module des Systems, ihre Zuständigkeiten und die Schnittstellen zwischen ihnen übersichtlich darzustellen. Somit wird es deutlich, welche Komponenten für das Einlesen und Verwalten von Daten zuständig sind, oder wie die Filterlogik eingebunden ist.

Ein zentrales Prinzip bei der Modellierung solcher Systeme ist die klare Trennung und Austauschbarkeit einzelner Komponenten. In diesem Zusammenhang lässt sich das Verhalten und die Flexibilität von Komponenten treffend wie folgt beschreiben:

„*Komponenten stellen eine abgegrenzte und über klar definierte Schnittstelle zugreifbares Verhalten bereit. Die konkrete Realisierung einer Komponente kann dabei gegen andere Komponenten, die über dieselben Schnittstellen verfügen, ausgetauscht werden, ohne Änderungen am System vorzunehmen*“ [13].

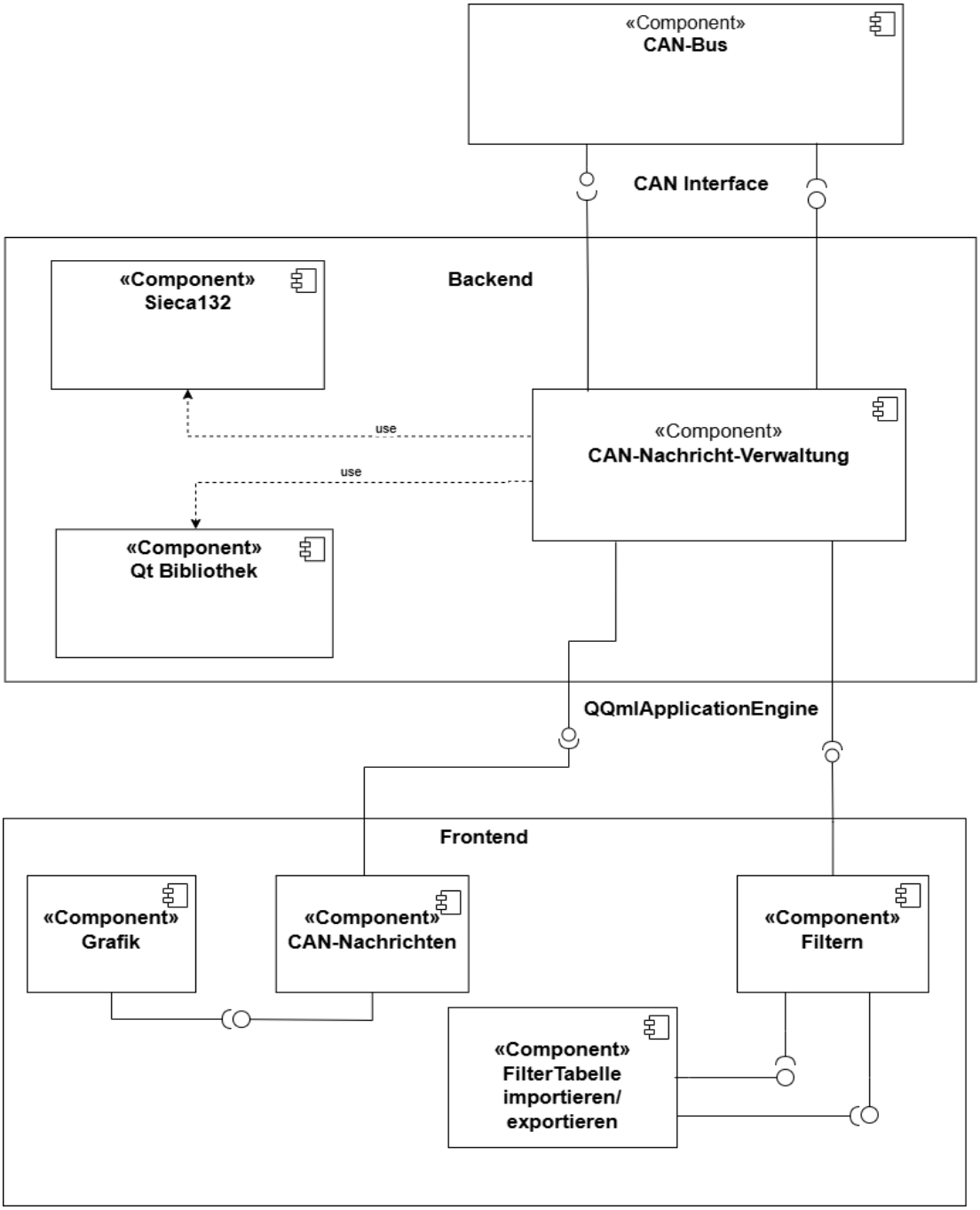


Abbildung 3.5: Komponentendiagramm des CAN Explorers.

Die Komponentendiagramm wurde sich für die bessere Verständlichkeit in zwei Hauptbereiche gegliedert. Der Backend-Bereich umfasst alle in C++ implementierten Komponenten und bildet die zentrale Logikschicht des Systems. Zudem wird von hier aus die Verbindung zur Hardware hergestellt und verwaltet. Der Frontend-Bereich

basiert auf QML und enthält die UI-spezifischen Komponenten, die für die Darstellung und Interaktion mit dem Benutzer verantwortlich sind.

Im Folgenden werden die einzelnen Systemkomponenten detailliert beschrieben und deren jeweilige Zuständigkeiten innerhalb der Systemarchitektur erläutert.

CAN-Bus
Der CAN-Bus stellt die physikalische Verbindung zur Übertragung von Nachrichten im Controller Area Network dar. Er dient als Schnittstelle zur Hardware und wird von der Software über ein spezielles CAN-Interface angesprochen.

CAN-Interface
Das CAN-Interface bildet die logische Schnittstelle zwischen der Softwarekomponente *CAN-Nachricht-Verwaltung* und der Hardware (*CAN-Bus*). Es ermöglicht das Senden und Empfangen von Nachrichten aus dem physischen Bus.

CAN-Nachricht-Verwaltung
Diese zentrale Backend-Komponente ist für das Empfangen, Verarbeiten und Weiterleiten von CAN-Nachrichten zuständig. Die eingehenden Daten werden analysiert und über die QML-Schnittstelle an das Frontend übergeben. Sie stellt die Verbindung zur Hardware her und übernimmt die Kommunikation mit dem CAN-Bus.

Sieca132
Bei *Sieca132* handelt es sich um eine Header-Datei, die die Anbindung an dem CAN-Treiber ermöglicht. Sie stellt die notwendigen Funktionen bereit, um auf den CAN-Adapter zuzugreifen und ist direkt in der *CAN-Nachricht-Verwaltung* eingebunden.

Qt-Bibliothek
Die Qt-Bibliothek ist eine zentrale externe Abhängigkeit des Systems. Sie bietet zahlreiche Funktionalitäten, darunter die GUI-Entwicklung und die objektorientierte Struktur zur Speicherverwaltung und Signalverarbeitung.

QQmlApplicationEngine
Diese Komponente stellt die Verbindung zwischen dem C++-Backend und dem QML-Frontend her. Sie ermöglicht die Instanziierung von QML-Komponenten aus dem C++-Code und erlaubt die wechselseitige Kommunikation über Signale und Slots. Dadurch ist eine klare Trennung von Logik (Backend) und Darstellung (Frontend) möglich [14].

CAN-Nachrichten (Frontend)

Diese QML-Komponente übernimmt die Darstellung der empfangenen CAN-Nachrichten in Tabellenform. Sie bildet das Hauptfenster der Anwendung und stellt die zentrale Benutzeroberfläche zur Anzeige und Interaktion mit den Nachrichten dar.

Filtern

Diese Komponente ermöglicht es dem Benutzer, gezielt Filterkriterien für die Darstellung der CAN-Nachrichten zu definieren. So können Nachrichten nach ID, Zeit oder Inhalt gefiltert und gezielt analysiert werden.

Grafik

Die Komponente *Grafik* dient der visuellen Aufbereitung von CAN-Daten. Sie zeichnet gewünschte Kurven oder Diagramme, beispielsweise zur Veranschaulichung zeitlicher Verläufe bestimmter Parameter.

Filtertabelle importieren/exportieren

Diese Komponente erlaubt das Laden und Speichern von Filtertabellen. Dadurch kann der Benutzer benutzerdefinierte Filtereinstellungen speichern und bei Bedarf erneut laden, ohne sie manuell neu eingeben zu müssen.

Klassendiagramm

Gerade bei komplexeren Komponenten, die intern wiederum aus mehreren Funktionseinheiten bestehen, wurde ergänzend ein Klassendiagramm (Abbildung 3.6) skizziert. Dieses verdeutlicht die Struktur innerhalb einzelner Komponenten und zeigt, welche Klassen welche Aufgaben übernehmen und wie sie miteinander interagieren. Dadurch wird die Implementierungslogik transparenter und die spätere Erweiterung oder Wartung des Systems erleichtert.

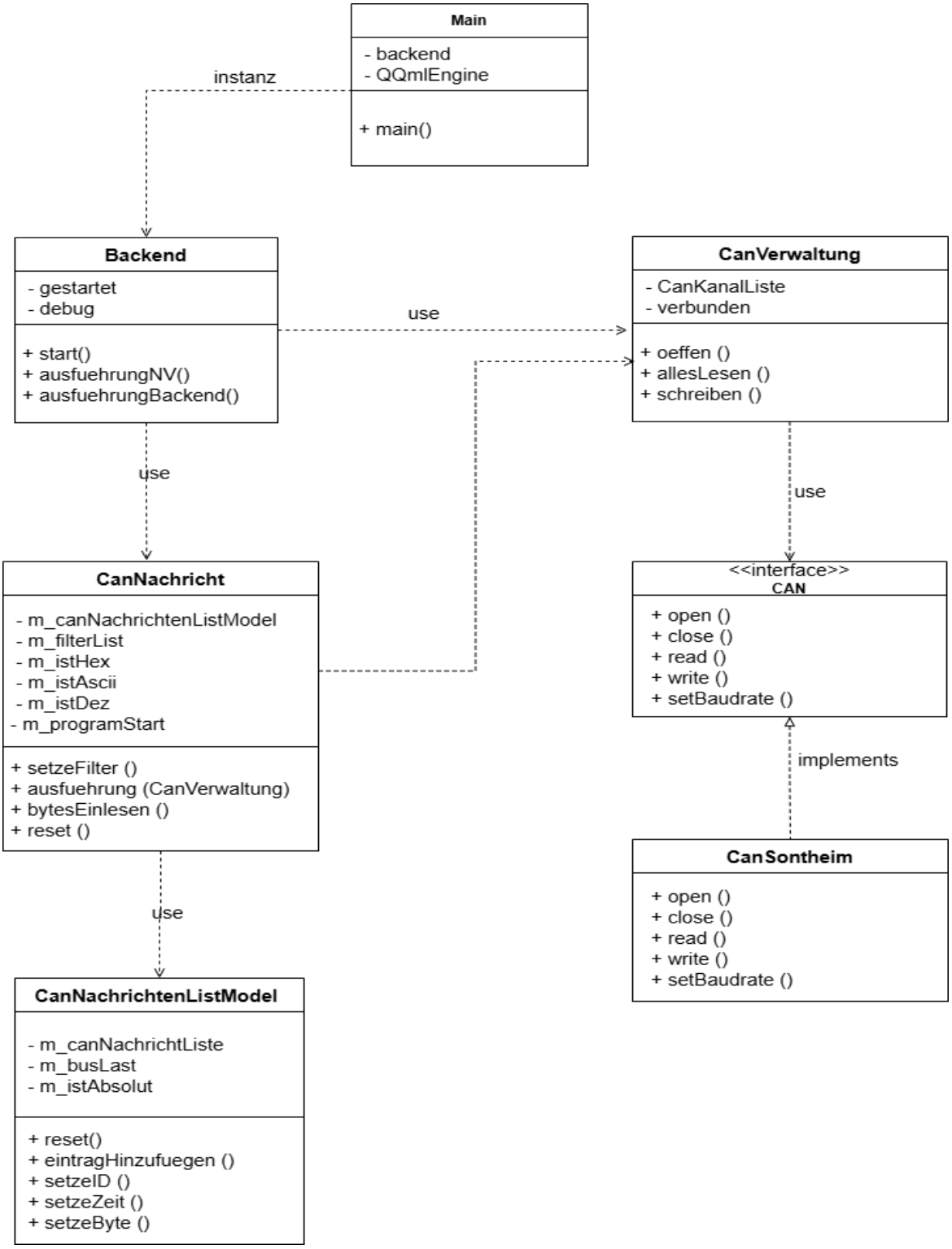


Abbildung 3.6: das Klassendiagramm für die Veranschaulichung der Komponentendiagramm.

Die Klassen und Interfaces mit ihren Interaktionen setzen sich wie folgt zusammen:

Main
Die Klasse erzeugt und initialisiert beim Programmstart eine Instanz der Klasse *Backend* und registriert den Typ *CanNachricht* als QML-Typ, um die GUI vom Backend zu entkoppeln.

Backend
Die Klasse hält einerseits eine Instanz von *CanVerwaltung*, über die alle CAN-Kanäle geöffnet, geschlossen, gelesen und beschrieben werden, und andererseits eine Instanz von *CanNachricht*, die für das Filtern und Aufbereiten der eingehenden und ausgehenden CAN-Nachrichten zuständig ist. Zusätzlich setzt *Backend* interne Timer ein, um periodisch Nachrichten zu verarbeiten und die Verbindung zu überwachen.

CanVerwaltung
In der *CanVerwaltung* sind verschiedene CAN-Schnittstellen gebündelt. Hierfür instanziiert sie alle Klassen, die das Interface CAN implementieren (zum Beispiel CanSontheim), und ruft deren open-, *close*-, *read*- und *write*-Methoden auf, um die eigentliche Kommunikation mit den Geräten zu steuern.

CAN
Das Interface CAN definiert die grundlegenden Methoden zur Geräteansteuerung.

CanSontheim
Diese Klasse implementiert die Methoden vom CAN-Interface konkret für Sontheim-Geräte und dabei enthält gerätespezifische Treiberlogik.

CanNachricht
Diese Klasse übernimmt die Verarbeitung der Nachrichteninhalte und befüllt dazu das *CanNachrichtenListModel*, das als Datenmodell für die GUI fungiert. Dabei erhält CanNachricht über Parameterübergabe aus seiner Methode *ausfuehrung (CanVerwaltung)* vorübergehend Zugriff auf *CanVerwaltung*, um bei Bedarf direkt weitere Lese- und Schreiboperationen anzustoßen.

CanNachrichtenListModel
Schließlich sorgt *CanNachrichtenListModel* dafür, dass die aufbereiteten Nachrichten in einer strukturierten Form vorliegen, die von der Benutzeroberfläche referenziert und dargestellt werden kann.

4 Umsetzung

In diesem Kapitel wird beschrieben, welche Werkzeuge, Softwarelösungen und Methoden zum Einsatz kommen, sowohl allgemein verwendbare als auch speziell auf das Unternehmen zugeschnittene. Zusätzlich werden für jede Anforderung des Unternehmens passende Lösungen vorgestellt.

In ersten Teil der Abschnitt werden zuerst die Methoden und Mittel für die Programmierung vorgestellt. Die Gründe für ihre Verwendung sowie ihre Vorteile werden anschließend diskutiert.

In dem Unterkapitel Implementierung kommen die Lösungen Schritt für Schritt auf den gesamten Anforderungen von der Datenaufbereitung über die Analyse bis hin zur Visualisierung der Ergebnisse ein. Ziel ist es, eine verlässliche technische Grundlage zu schaffen, die nicht nur den Anforderungen des Unternehmens entspricht, sondern auch mit den realen Herausforderungen der Implementierung umgehen kann.

4.1 Methode und Werkzeuge

Für die Entwicklung des CAN Explorers wurde C++ in Verbindung mit dem Qt-Framework für das Backend sowie QML für die Gestaltung der Benutzeroberfläche eingesetzt. Die Wahl dieser Technologien basiert auf den bestehenden technologischen Voraussetzungen bei Kinemotion. Weil die zentrale Unternehmenssoftware KMWare bereits vollständig in C++ unter Qt entwickelt ist und aktiv weitergepflegt wird.

Da der CAN Explorer perspektivisch direkt in KMWare integriert werden soll, ist die Verwendung derselben Entwicklungsumgebung und -technologien nicht nur naheliegend, sondern auch aus technischer und organisatorischer Sicht sinnvoll. Die Kombination aus C++ und QML erlaubt eine klare Trennung zwischen Logik und Oberfläche, fördert die Wiederverwendbarkeit von Komponenten und erleichtert die Integration in die bestehende Softwarelandschaft von Kinemotion.

Für den Softwaretest wird in der Werkstatt von Kinemotion ein Fahrzeugsystem verwendet, dass ein echtes System ist und später an den Kunden geliefert wird, um im Fahrzeug montiert zu werden.

Für die Kommunikation mit dem Fahrzeugsystem wird ein CAN-Dongle von Anbieter Sontheim verwendet, um CAN-Nachrichten zu empfangen.

Da der CAN Explorer später in KMWare integriert werden soll, ist es sinnvoll, das Tool bereits jetzt bestmöglich darauf vorzubereiten. Daher werden die vorhandenen

Header- und Source-Dateien in KMWare für diesem Projekt genutzt, um den Implementierungsaufwand zu reduzieren und die spätere Integration zu erleichtern. Diese werden nun im Folgenden beschrieben.

Für die Sontheim-Schnittstelle wird die Datei *SIECA132.h* verwendet. Der Sontheimproduktkatalog beschreibt diese wie folgt: *„SIECA132.h SiECA 132 MT-CANapi ist eine Schnittstelle zur Anwendungsprogrammierung, welche anderen Programmen die Software-Anbindung an die Sontheim-Produkte ermöglicht. Unterstützte Produkte: Die leistungsstarke SiECA132 multithread CANapi unterstützt alle internen und externen CAN-Interfaces wie unser CANUSB, den CANUSBlight oder auch die PowerCAN PCI“* [15].

Die beiden Dateien *canSothteim.h* und *canSothteim.cpp* definieren dabei die Schnittstellen und Funktionen, die für die Kommunikation mit Sontheim-CAN-Interface erforderlich sind. Sie implementiert *can.h* und benutzt *SIECA132.h***.** Die Datei wird bereit in KMWare benutzt, um mit CAN-Nachrichten zu arbeiten.

Das Interface für mögliche CANUSBs wird in *can.h* und *can.cpp* umgesetzt. Diese sind auch ein Teil des Projekts KMWare.

Beide Dateien *canverwaltung.h* und *canverwaltung.cpp* werden benutzt, um CAN-Nachrichten zu lesen, schreiben und andere Operationen durchzuführen.

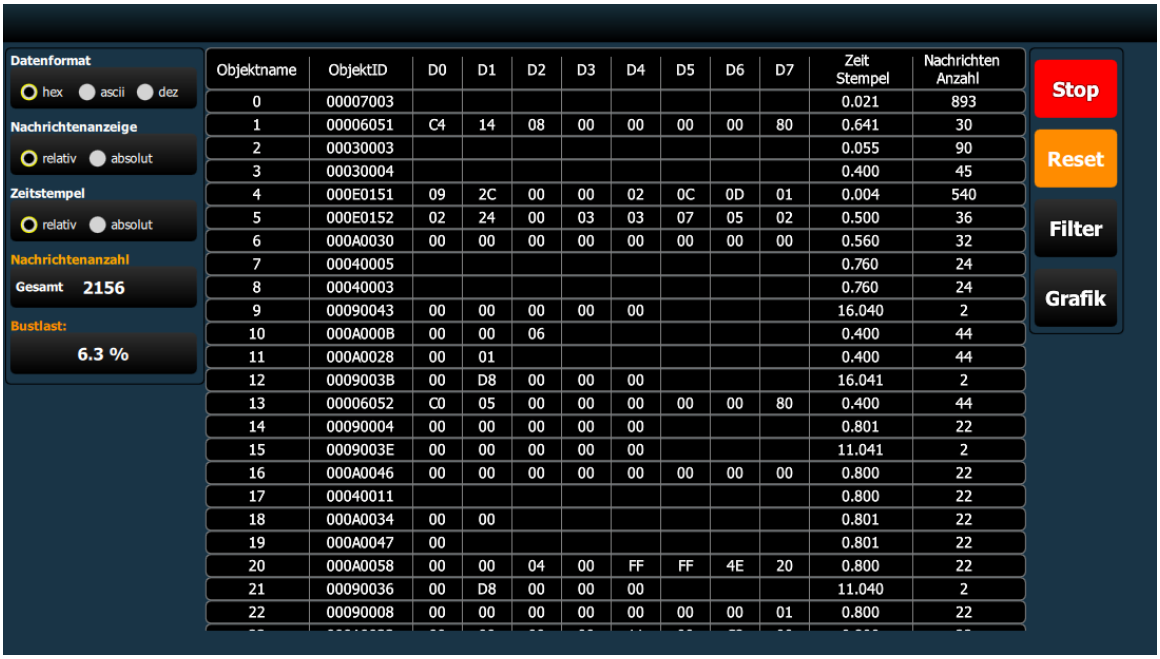
Für das Backend werden *backen.h* und *backend.cpp* genutzt und dienen dazu, das gesamte System zu starten und auszuführen.

4.2 Implementierung

In diesem Unterkapitel werden die Lösungen für die funktionalen Anforderungen präsentiert.

Anforderung 1 (Datenaufnahme)

Die Nachrichten (Abbildung 4.1) werden in Echtzeit erfasst und übersichtlich in der CAN-Nachrichtentabelle dargestellt.



Objektname	ObjektID	D0	D1	D2	D3	D4	D5	D6	D7	Zeit Stempel	Nachrichten Anzahl
0	00007003									0.021	893
1	00006051	C4	14	08	00	00	00	00	80	0.641	30
2	00030003									0.055	90
3	00030004									0.400	45
4	000E0151	09	2C	00	00	02	0C	0D	01	0.004	540
5	000E0152	02	24	00	03	03	07	05	02	0.500	36
6	000A0030	00	00	00	00	00	00	00	00	0.560	32
7	00040005									0.760	24
8	00040003									0.760	24
9	00090043	00	00	00	00	00				16.040	2
10	000A000B	00	00	06						0.400	44
11	000A0028	00	01							0.400	44
12	0009003B	00	D8	00	00	00				16.041	2
13	00006052	C0	05	00	00	00	00	00	80	0.400	44
14	00090004	00	00	00	00	00				0.801	22
15	0009003E	00	00	00	00	00				11.041	2
16	000A0046	00	00	00	00	00	00	00	00	0.800	22
17	00040011									0.800	22
18	000A0034	00	00							0.801	22
19	000A0047	00								0.801	22
20	000A0058	00	00	04	00	FF	FF	4E	20	0.800	22
21	00090036	00	D8	00	00	00				11.040	2
22	00090008	00	00	00	00	00	00	00	01	0.800	22

Abbildung 4.1 Echtzeit Aufnahme der CAN-Nachrichten

Die Übertragung von CAN-Nachrichten in einer Qt/QML-Anwendung kann auf verschiedene Arten realisiert werden. Es wurden zwei gängige Ansätze betrachtet: die Verwendung eines *QAbstractListModel*, oder die Nutzung einzelner *QObject*-Instanzen mit *Properties* und *Signalen*.

Im Rahmen dieser Arbeit wurde die Lösung mit *QAbstractListModel* gewählt. Im Folgenden werden die Gründe dafür erläutert und ein Vergleich mit der alternativen QObject-basierten Methode dargestellt.

Dieses Model kann problemlos in QML eingebunden werden. QML-Elemente wie *ListView* oder *TableView* können direkt auf das Model zugreifen und dessen Daten anzeigen. Darüber hinaus das ermöglicht die Definition benutzerdefinierter Rollen, wie zum Beispiel Nachricht ID, Zeit, oder Nachrichtenanzahl können direkt abgebildet werden. Diese Rollen stehen in QML als Properties zur Verfügung [16].

Zudem ermöglicht Das Signal-Slot-System von Qt eine automatische Synchronisierung zwischen dem Model und der Oberfläche in QML. Wenn das Model eine Änderung meldet, wird die Oberfläche automatisch aktualisiert, ohne dass manuelle Updates notwendig sind. Ein weiterer wichtiger Punkt ist, dass das *QAbstractListModel* für die großen, dynamischen Datenlisten geeignet ist. Auch bei vielen CAN-Nachrichten bleibt die Oberfläche reaktiv.

Bei der alternativen Lösung können CAN-Nachrichten z. B. als einzelne *QObject*-Instanzen mit Q_*PROPERTY*-Attributen dargestellt werden. Änderungen sollen dann über manuell ausgelöste Signale an QML übermittelt werden. Diese Methode kann für einzelne Werte oder gelegentliche Upates benutzt werden, aber sie ist für höheren Nachrichtenanzahl nicht geeignet. Außerdem ist schwierig mit diesem Model eine dynamische Liste zu erzeugen.

Anforderung 2 (Datenanzeige)

Für die grafische Umsetzung für Bedienelemente wie Start/Stop, Reset sowie Datenfelder, Buslastanzeige, Nachrichtenanzahl und Nachrichteninhalte wurden in dieser Arbeit überwiegend das *Rectangle*-Element von QML verwendet, obwohl es schon aus *QtQuick.Controls* vordefinierte *Button* Element gibt.

Der ein der Hauptvorteil von *Rectangle* ist, dass es die Größen flexibel anpassen lässt. Durch die Möglichkeit, Breite und Höhe an das Hauptfenster zu binden, skalieren die Elemente automatisch mit, wenn sich die Fenstergröße verändert. Dies unterstützt ein responsives Design, ohne zusätzlichen Layoutcode. Desweitern erlaubt *Rectangle* unkompliziert mit *Text*-Elementen kombinieren. Dadurch können Beschriftungen, Werte und Statusanzeigen direkt innerhalb des Elements dargestellt werden, ohne separate Komponenten zu benötigen. Zudem ist ermöglichen die Eigenschaften wie *radius*, *color* und *border* eine individuelle Gestaltung – z. B. mit abgerundeten Ecken oder farblichen Hervorhebungen, ohne den Einsatz externer Ressourcen zu benutzen.

Im Vergleich mit *Rectangle* zeigt das Element *Button* deutliche Schwächen in Bezug auf Layout-Flexibilität und Individualisierung. Dazu gehört die Größenanpassung: Das *Button*-Element ist nicht vollständig responsiv. Selbst wenn Breite und Höhe explizit gebunden werden, kann das interne Layout des in manchen Fällen nicht korrekt mitskalieren. Außerdem sind die Farbgestaltung und Design Anpassung eingeschränkt. Um ein individuelles Aussehen zu erreichen, muss entweder über komplexe Stilüberschreibungen wie *Style.qml* ein eigener Style implementiert oder das *Button*-Element vollständig durch eine benutzerdefinierte Komponente ersetzt werden, was im Vergleich zum flexiblen *Rectangle* deutlich umständlicher ist.

Start/Stop

Ein wurde ein Button (Abbildung 4.2) hinzugefügt, um die “Start” und “Stop” Funktionalitäten umzusetzen. Wenn „Start“ gedrückt wird, ändert sich der Zustand des Buttons auf „Stop“ und die Applikation startet die empfangenen Daten in der Tabelle zu visualisieren. Wenn dann „Stop“ gedrückt wird, ändert der Zustand des Buttons wieder auf „Start“ und das Programm hört auf Daten aufzunehmen.

Reset

Ein weiterer “Reset” Button wurde für den Neustart der Aufnahme implementiert. Wenn dieser Button geklickt wird, wird die Aufnehme der Nachrichten neu gestartet. Dieser regiert nur, solange die Aufnahme läuft, sonst regiert die Anwendung nicht.

Abbildung 4.2: Start, Stopp und Reset Buttons der CAN-Explorer.

Darstellung CAN-Nachrichten in Form einer Tabelle

Zur Darstellung der empfangenen CAN-Nachrichten wurde eine Tabelle (Abbildung 4.3) mit den folgenden Spalten Objekt-ID, Nachrichteninhalt (Rohdaten), Zeitstempel, Nachrichtenanzahl sowie Objektname bereitgestellt. Ankommende CAN-Nachrichten werden dabei automatisch und in Echtzeit in diese Tabelle eingetragen. In der Tabelle werden sowohl Standard- als auch Extended-CAN-Nachrichten dargestellt. Dabei spielt das Nachrichtenformat keine Rolle: Unabhängig davon, ob es sich um 11-Bit-Standard-IDs oder 29-Bit-Extended-IDs handelt, werden alle empfangenen Nachrichten einheitlich in die Tabelle eingetragen.

Objektname	ObjektID	D0	D1	D2	D3	D4	D5	D6	D7	Zeit Stempel	Nachrichten Anzahl
0	00007003									0.021	893
1	00006051	C4	14	08	00	00	00	00	80	0.641	30
2	00030003									0.055	90
3	00030004									0.400	45
4	000E0151	09	2C	00	00	02	0C	0D	01	0.004	540
5	000E0152	02	24	00	03	03	07	05	02	0.500	36
6	000A0030	00	00	00	00	00	00	00	00	0.560	32

Abbildung 4.3: CAN-Nachrichtentabelle.

Objektname
Zusätzlich wurde eine Spalte für den Objektnamen in der Tabelle erstellt. Um eine spätere der Objektnamen zu ermöglichen, wurde eine Variable eingeführt, die aktuelle Index jeweiligen Nachricht in der Tabelle zeigt. Diese Variable wird beim Eintrag in die Tabelle dynamisch verwendet und kann bei Bedarf mit minimalem Aufwand durch einen anderen Wert ersetzt werden. Dadurch lässt sich die Darstellung der Objektnamen flexibel anpassen, ohne Änderungen an der zugrunde liegenden Tabellenstruktur vornehmen zu müssen.

Anforderung 3 (Funktionalitäten)
Auf der linken Seite des Fensters (siehe Abbildung 4.1) befinden sich Auswahlfelder für das Datenformat, die Nachrichtenanzeige, den Zeitstempel, die Buslast sowie die Nachrichtenanzahl. Diese Steuerelemente wurden so angeordnet, dass die verschiedenen Einstellungsmöglichkeiten zur Nachrichtenanzeige in einer logischen Reihenfolge direkt neben der Nachrichtentabelle platziert sind. Dadurch wird eine intuitive Bedienung und eine klare Trennung zwischen Datendarstellung und Konfiguration ermöglicht.

Format der Nutzdaten
Die auswahlbaren Formate sind Hex, Dezimal und ASCI-Format, welches die Benutzer einfach durch das Klicken auf das gewünschte beschriftete Feld ändern können. Anschließend wird das CAN-Nachrichten Format wird geändert. Die Daten kommen im ASCI-Format aus dem CAN-Bus. Sobald ein anderes Format gewünscht wird, wird ein Signal von dem Frontend an das Backend geschickt und zum gewünschten Format umgewandelt.

Der Grund, warum die Daten nicht in QML selbst umgewandelt werden, liegt an der Stärke von C++, welche leistungsstarke String- und Byte-Handling Funktionen bietet und auch selbst bei den großen Datenmengen performant bleibt**.**

Die Daten hätte man auch mit Funktionen in QML umwandeln können. Solche rechenintensiven Umwandlungen sind in QML jedoch nicht ideal. Außerdem ist QML langsamer als C++ und man hat weniger Kontrolle über das Byte-Handling. Dieses Vorgehen wurde auch bei der Nachrichtenanzeige und der Zeitstempeländerungen umgesetzt.

Relative und absolute Nachrichtenanzeige
Je nach Benutzerauswahl setzt QML die entsprechende Variable in Backend auf *true*, wodurch die Nachrichtenliste entsprechend reagiert. Bei der relativen Anzeige werden eingehende Nachrichten aus dem CAN-Bus überschrieben, sobald neue Nachrichten eintreffen. Bei der absoluten Anzeige hingegen wird jede Nachricht in eine eigene Zeile

eingetragen. Sobald die Liste voll ist, werden die ältesten Einträge schrittweise überschrieben, beginnend mit dem ersten Element.

Relative und absolute Zeitstempel

Sobald der Benutzer eine Änderung vornimmt, wird die entsprechende Variable im Backend auf *true* gesetzt, wodurch die Anpassungen unmittelbar in der Tabelle sichtbar werden

Anzeige der Anzahl der Gesamtnachrichten

Die Gesamtanzahl der Nachrichten wird im Backend fortlaufend hochgezählt, sobald neue Nachrichten eintreffen. Im Frontend wird der aktualisierte Wert sofort angezeigt, sodass stets die aktuelle Anzahl sichtbar ist.

Die Auslastung der CAN-Busses

Die Buslast wird direkt vom CAN-Bus erfasst und im Backend zwischengespeichert. Anschließend wird der Wert ohne Verzögerung an das Frontend weitergeleitet, sodass eine sofortige Anzeige gewährleistet ist.

Anforderung 4 (Nachrichtenfilterung)

Auf der rechten Seite der Startfenster (Abbildung 4.4) wurde ein „Filter“ Button implementiert. Mit einem Mausklick erscheint neben der Nachrichtentabelle ein Filterfenster (Abbindung 4.4). Mit dem Doppelklick auf dem Filter Button verschwindet das Filterfenster wieder. Außerdem kann man mit dem „x“ Button oben rechts das Filterfenster wieder schließen.

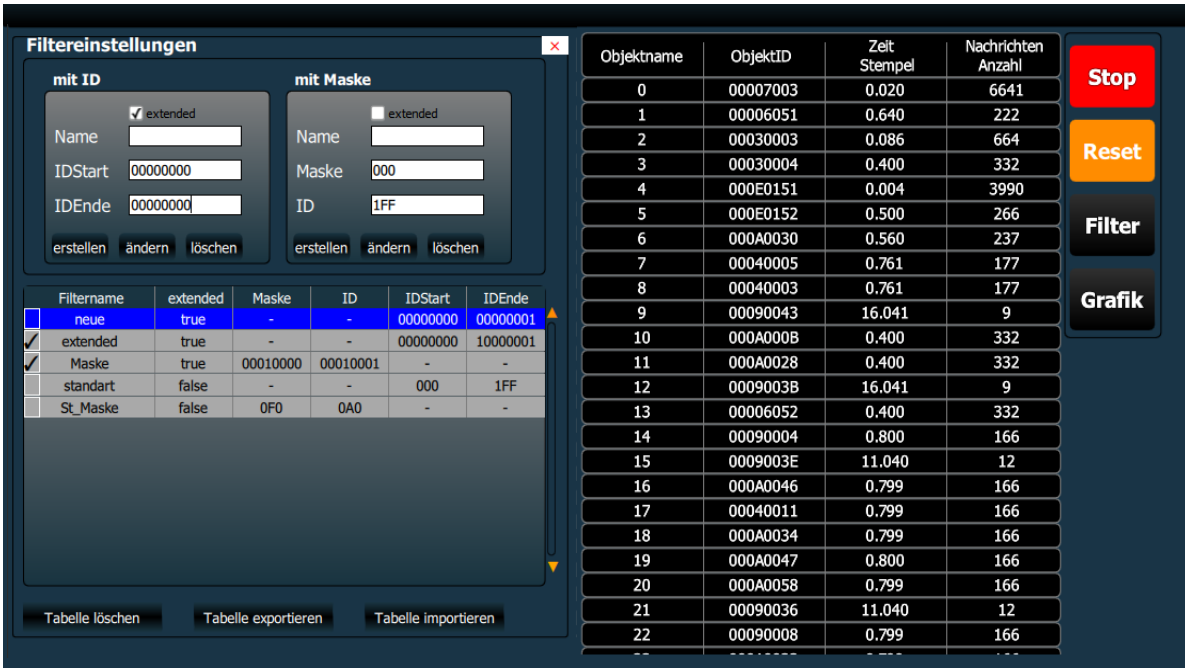


Abbildung 4.4: Fenster für das Filtern der Nachrichten in Anwendung

Bereichsfilter und Maskenfilter

Zur Erstellung und Verwaltung von Filtern wurden zwei getrennte Eingabefelder (Abbildung 4.4) implementiert. Eins davon ist für die Konfiguration eines Bereichsfilters und das andere ist für den Maskenfilter. Unterhalb dieser Eingabefelder befinden sich drei Schaltflächen mit den Funktionen **„**Erstellen**“**, **„**Ändern**“** und **„**Löschen**“**. Diese Buttons ermöglichen es, neue Filter anzulegen, bestehende Filtereinträge zu bearbeiten oder bei Bedarf zu entfernen. Es besteht die Möglichkeit, sowohl Standard- als auch Extended-Filter zu erstellen. Alle erstellten Filter werden in einer übersichtlichen Tabelle dargestellt. Um einzelne oder mehrere Filter zu aktivieren, befindet sich vor jedem Filtereintrag eine Checkbox. Wird ein Haken gesetzt, wird der entsprechende Filter sofort auf die eingehenden CAN-Nachrichten angewendet.

Filter speichern, entfernen, importieren und exportieren

Bei jeder Änderung in der Filtertabelle wird deren aktueller Zustand sofort automatisch in die Datei *Filter.txt* gespeichert. Beim Neustart der Anwendung wird der zuletzt gespeicherte Zustand der Filtertabelle wiederhergestellt.

Für das Speichern der Filtertabelle kamen zwei Ansätze in Betracht: die Verwendung von *XMLHttpRequest()* zur Dateiverwaltung sowie der Einsatz von *QML-Settings* zur internen Speicherung. Letztlich wurde *XMLHttpRequest()* gewählt, da diese Methode das Speichern in einer externen Textdatei ermöglicht, die sich später auch für den Export oder Austausch verwenden lässt. Ein weiterer Vorteil liegt in der Möglichkeit, strukturierte und komplexe Datenformate zu speichern.

QML-Settings ist zwar direkt in QML verwendbar und keine externe Dateiverwaltung erfordert, hat es seine Einschränkungen. Es ist schwieriger zu bearbeiten und eignet sich nicht für große Datenmengen, sondern eher für einzelne Variablen oder einfache Einstellungen. Ein wesentlicher Nachteil von *QML-Settings* ist, dass die gespeicherten Daten nicht zwischen verschiedenen Benutzern transportierbar sind.

Am unteren Rand des Filterfensters (Abbildung 4.4) wurden drei Schaltflächen bereitgestellt. **„**Löschen**“**, **„**Importieren**“** und **„**Exportieren**“**. Die Schaltfläche **„**Löschen**“** entfernt alle Einträge aus der Filtertabelle. Die Buttons **„**Importieren**“** und **„**Exportieren**“** wurden bereits in die Oberfläche integriert und dienen der zukünftigen Weiterentwicklung. Sie sollen es ermöglichen, eine Filterkonfiguration aus einer externen Datei zu importieren bzw. die aktuelle Filtertabelle als Datei zu exportieren. Die notwendigen Funktionalitäten dafür können bei Bedarf mit geringem Aufwand nachträglich ergänzt werden.

Datenanalyse und grafische Darstellung der CAN-Nachrichten

Zur Visualisierung ausgewählter CAN-Nachrichten wurde ein Button mit der Beschriftung **„Grafik“** implementiert. Ein Klick auf diesen Button öffnet unterhalb der Nachrichtentabelle ein zusätzliches Grafikfenster (Abbildung 4.5). Parallel dazu erscheint auf der rechten Seite der Benutzeroberfläche links ein Eingabebereich mit Feldern, über den festgelegt werden kann, welche CAN-Nachrichten in der Grafik dargestellt werden sollen. Die Anzeige des Grafikfensters und des Eingabebereichs kann durch einen Doppelklick auf den „Grafik“-Button oder alternativ über das „x“-Symbol oben rechts im Grafikbereich wieder geschlossen werden.

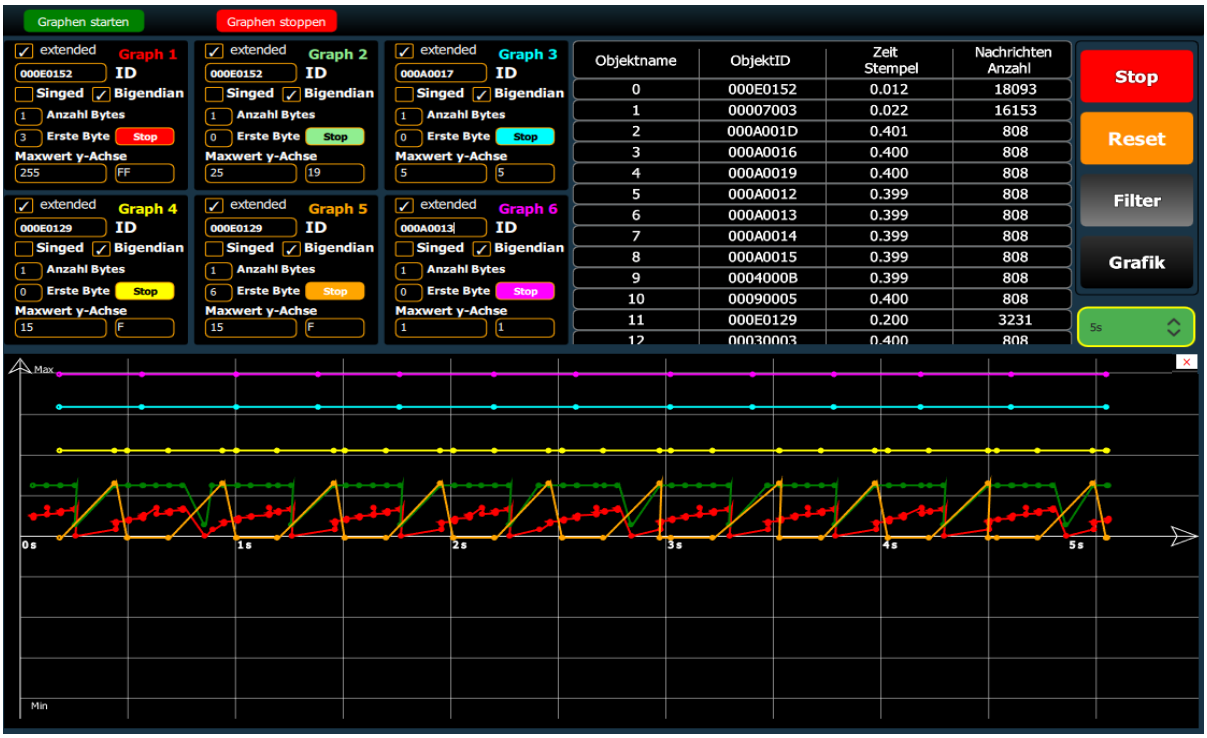


Abbildung 4.5: Grafische Darstellung der CAN-Nachrichten

Auswahl von CAN-Nachrichten über Eingabefelder.

Zur grafischen Analyse von CAN-Nutzdaten wurde ein leistungsfähiges und interaktives Grafikmodul implementiert. In diesem Modul werden Echtzeitdaten aus sechs unterschiedlichen CAN-Nachrichten gleichzeitig in unteren Grafikfenster dargestellt.

Die X-Achse der Grafik repräsentiert die Zeitachse, während auf der Y-Achse die Werte der ausgewählten Nutzdaten angezeigt werden. Für die X-Achse wurde eine Skalierungsauswahl in Sekunden (1, 2, 5, 10) integriert, um die Darstellung an unterschiedliche Analysezeiträume anzupassen. Diese Auswahl befindet sich oben rechts im Grafikfenster in der grün hinterlegten Box.

Für jede darzustellende Nachricht wurde ein konfigurierbares Eingabefeld realisiert, über das die gewünschte CAN-ID eingegeben werden kann. Zusätzlich können Nutzer die Byte-Anzahl (1 bis 4 Byte) sowie die Byte-Position (Startbyte) festlegen, ab wann der auszuwertende Datenwert beginnt.

Zur weiteren Individualisierung der Y-Achse wurde eine Option zur Festlegung des maximalen Wertebereichs integriert. Außerdem kann der Nutzer auswählen, ob die Werte als signed oder unsigned interpretiert werden sollen.

Ein weiterer wichtiger Aspekt der Implementierung ist die Unterstützung unterschiedlicher Byte-Reihenfolgen. Der Benutzer kann zwischen Motorola (Big-Endian) und Intel (Little-Endian) wählen.

5 Qualitätssicherung

In diesem Kapitel werden die Maßnahmen zur Qualitätssicherung des Projekts vorgestellt. Ziel ist es, zu überprüfen, ob das entwickelte System die gewünschten Funktionen erfüllt und zuverlässig arbeitet.

Das Kapitel gliedert sich in zwei Teile. Im ersten Unterkapitel werden Unit-Tests beschrieben. Dabei werden Tests durchgeführt, die einzelne Programmfunktionen isoliert prüfen. Unit-Tests bieten die Möglichkeit, die App mit verschiedenen Eingaben und Testdaten zu prüfen, um sicherzustellen, dass die Funktionen korrekt arbeiten. So können Fehler frühzeitig erkannt und behoben werden, bevor die Anwendung in einem realen System getestet oder eingesetzt wird. Zudem unterstützen sie eine modulare und wartbare Softwarearchitektur.

Im zweiten Teil des Kapitels werden praktische Tests im realen Fahrzeugsystem behandelt. Diese Tests dienen dazu, das System unter realen Bedingungen zu überprüfen und die Ergebnisse aus den Unit-Tests zu validieren. Dabei wird beobachtet, ob das System im Zusammenspiel mit der Hardware wie gewünscht funktioniert. Die Ergebnisse dieser praktischen Probe werden dokumentiert und ausgewertet, um mögliche Abweichungen oder Optimierungspotenziale zu finden.

5.1 Unit Test

Obwohl die Implementierung von Unit-Tests nicht Teil der ursprünglichen Anforderungen seitens des Unternehmens war, wurde im Rahmen dieser Arbeit bewusst entschieden, Unit-Tests eigeninitiativ zu entwickeln und einzusetzen. Das erhöht die Qualität der Software und hilft Implementierungsfehler zu finden. Durch den Einsatz des Unit-Tests konnte die funktionale Korrektheit zentraler Funktionen systematisch überprüft werden. Dies diente nicht nur der Fehlervermeidung, sondern auch der langfristigen Wartbarkeit und Erweiterbarkeit des Codes.

Um zu überprüfen, ob die entwickelten Softwarekomponenten korrekt funktionieren, wurde in diesem Projekt das Test-Framework Qt Test (QTestLib) eingesetzt. Diese Bibliothek ist Teil von Qt und eignet sich gut dafür, Unit-Tests in C++ zu schreiben. So konnten bestimmte Funktionen einzeln getestet und ihr Verhalten kontrolliert werden.

Insgesamt wurden elf Unit-Tests geschrieben, die sich auf die zentralen Funktionen des Programms konzentrieren. Dabei lag der Schwerpunkt auf den Hauptfunktionen der Anwendung, um sicherzugehen, dass diese auch unter verschiedenen Bedingungen wie gewünscht arbeiten. Diese Testet ermöglichen effiziente Nachverfolgung von Änderungen im Code. Wenn Funktionen verändert oder erweitert werden, können die

bestehenden Tests erneut ausgeführt werden, um zu überprüfen, ob die Anpassungen korrekt funktionieren und keine unbeabsichtigten Nebenwirkungen auftreten

Alle Testfälle liefen erfolgreich durch (Abbildung 5.1). Das spricht dafür, dass die getesteten Funktionen stabil implementiert wurden und wie vorgesehen arbeiten.

```
16:30:30: Starte C:\Users\Navruz Jabbarov\Desktop\CANExplorer\build\CANExplorerTest.exe...
********* Start testing of CanNachrichtTest *********
Config: Using QtTest library 5.15.18, Qt 5.15.18 (i386-little_endian-ilp32 shared (dynamic) release build; by GCC 8.1.0), windows 10
PASS   : CanNachrichtTest::initTestCase()
PASS   : CanNachrichtTest::testStandardWerte()
PASS   : CanNachrichtTest::testHexModus()
PASS   : CanNachrichtTest::testAsciiModus()
PASS   : CanNachrichtTest::testDezimalModus()
PASS   : CanNachrichtTest::testAbsolutModus()
PASS   : CanNachrichtTest::testZeitAbsolutModus()
PASS   : CanNachrichtTest::testProgramStart()
PASS   : CanNachrichtTest::testFilterFunktionen()
PASS   : CanNachrichtTest::testByteEinlesen()
PASS   : CanNachrichtTest::cleanupTestCase()
Totals: 11 passed, 0 failed, 0 skipped, 0 blacklisted, 7ms
********* Finished testing of CanNachrichtTest *********
```

Abbildung 5.1: Die Ergebnisse der Unit-Tests des CAN Explorers

5.2 Test in echtem Fahrzeugsystem

Zur Validierung der Anwendung unter realen Bedingungen wurde ein Test direkt an einem echten Fahrzeugsystem (Abbildung 5.2) durchgeführt, das für den Endkunden vorgesehen ist. Der Test fand in der Werkstatt von Kinemotion statt und verfolgte das Ziel, die Anwendung mit realen CAN-Daten zu prüfen und deren Verhalten unter praxisnahen Bedingungen zu evaluieren. Der gesamte Testablauf wurde in enger Zusammenarbeit mit dem technischen Leiter von Kinemotion, Andreas Hasenfuß, durchgeführt.

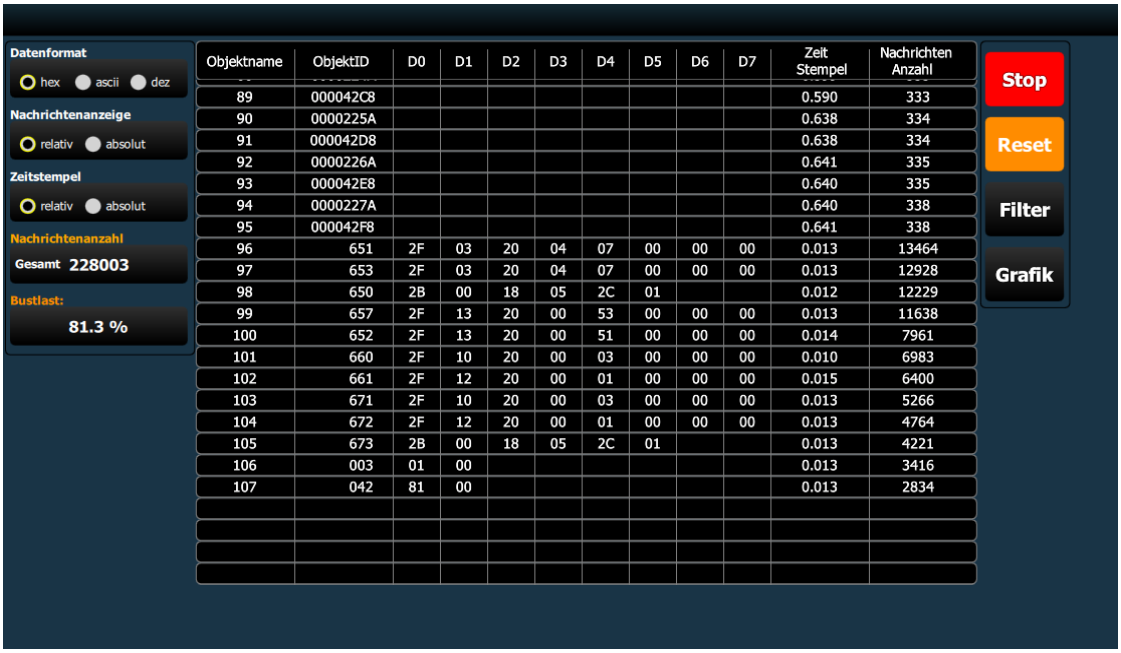
Abbildung 5.2: echtes Fahrzeugsystem in Kinemotion

Sämtliche Testfälle wurden gemeinsam besprochen, vorbereitet und anschließend direkt am realen Fahrzeugsystem ausgeführt. Herr Hasenfuß hat die Tests eigenständig durchgeführt sowie deren Verlauf und Ergebnisse aktiv beobachtet und bewertet. Auf diese Weise konnte sichergestellt werden, dass alle relevanten Anforderungen praxisnah geprüft.

Der Aufbau des Testsystems bestand aus einem Laptop mit der entwickelten Software, einem Sontheim-Interface-Dongle, der die Verbindung zwischen dem Fahrzeugsystem und dem Laptop herstellt, sowie einem realen Fahrzeugsystem mit aktivem CAN-Bus. Der Dongle ist über USB mit dem Laptop verbunden, wodurch die CAN-Nachrichten direkt in die Anwendung eingespeist und dort in Echtzeit verarbeitet wurden.

Buslast-Test

Gemäß den Vorgaben von Kinemotion darf die CAN-Buslast im regulären Betrieb maximal 50 % betragen. Zu Beginn des Tests wurde die Buslast bewusst erhöht, um ein realistisches und forderndes Szenario zu simulieren. Die Auslastung schwankte zwischen 50 % und 80 %, mit einem stabilen Maximalwert von 81,3 % (Abbildung 5.3). Diese erhöhte Buslast wurde während des gesamten Testverlaufs beibehalten, um die Software unter dauerhaft hoher Last zu erproben.



Objektname	ObjektID	D0	D1	D2	D3	D4	D5	D6	D7	Zeit Stempel	Nachrichten Anzahl
89	000042C8									0.590	333
90	0000225A									0.638	334
91	000042D8									0.638	334
92	0000226A									0.641	335
93	000042E8									0.640	335
94	0000227A									0.640	338
95	000042F8									0.641	338
96	651	2F	03	20	04	07	00	00	00	0.013	13464
97	653	2F	03	20	04	07	00	00	00	0.013	12928
98	650	2B	00	18	05	2C	01			0.012	12229
99	657	2F	13	20	00	53	00	00	00	0.013	11638
100	652	2F	13	20	00	51	00	00	00	0.014	7961
101	660	2F	10	20	00	03	00	00	00	0.010	6983
102	661	2F	12	20	00	01	00	00	00	0.015	6400
103	671	2F	10	20	00	03	00	00	00	0.013	5266
104	672	2F	12	20	00	01	00	00	00	0.013	4764
105	673	2B	00	18	05	2C	01			0.013	4221
106	003	01	00							0.013	3416
107	042	81	00							0.013	2834

Abbildung 5.3 Die Ergebnisse von Aufnahme der CAN-Nachrichten in der App.

Bei einer Buslast von über 70 % konnte eine minimale Verzögerung in der Verarbeitung beobachtet werden – konkret betrug diese etwa 0,2 Sekunden. Trotz dieser geringen Latenz reagierte die Anwendung weiterhin stabil und korrekt.

Nachrichtenanzahltest

Im Rahmen des Nachrichtenanzahl-Tests wurde untersucht, wie gut der CAN Explorer eine große Anzahl verschiedener CAN-Nachrichten verarbeiten und visualisieren kann. Während des Tests wurden maximal 107 unterschiedliche Nachrichten gleichzeitig empfangen, dekodiert und anschließend übersichtlich in einer Tabelle dargestellt (Abbildung 5.2). Insgesamt wurden etwa 500 bis 2000 Nachrichten pro Sekunde verarbeitet. Die Software zeigte dabei ein konsistentes Verhalten und erfüllte die definierten Anforderungen auch unter hoher Systemlast.

Datenformat-Test

In diesem Test wurde geprüft, ob die Software alle definierten CAN-Nachrichtenformate korrekt interpretieren und darstellen kann. Dabei wurde zwischen verschiedene Nachrichtenformate gezielt gesprungen und beobachtet ob die Nachrichtenbytes in richtiges Format korrekterweise umgestellt wurde. Alle Nachrichten wurden zuverlässig erkannt, korrekt dekodiert und visualisiert. Es traten während des gesamten Tests keine Formatierungs- oder Darstellungsfehler auf.

Filter-Test

Bei diesem Test wurde untersucht, ob das Filtersystem des CAN Explorers Nachrichten gezielt und zuverlässig herausfiltern kann. Dazu wurden sowohl „Extended“- als auch „Standard“-Nachrichten mit unterschiedlichen Filterkombinationen getestet, um unerwünschte Nachrichten effektiv auszublenden und relevante Nachrichten korrekt hervorzuheben. Im Rahmen des Tests kamen Bereichsfilter, Maskenfilter sowie mehrstufige Kombinationen beider Typen. Auch Mischformen, bei denen Bereichs- und Maskenfilter gleichzeitig angewendet wurden, wurden geprüft. Auf diese Weise konnte das Verhalten des Systems unter realitätsnahen und komplexen Filterkonfigurationen bewertet werden.

Durch die gezielte Auswahl und Kombination der Filtertypen wurde eine hohe Testabdeckung erreicht. Dabei wurden sowohl positive Szenarien überprüft, in denen gewünschte Nachrichten korrekt angezeigt wurden, als auch negative Szenarien, bei denen unerwünschte Nachrichten erfolgreich ausgeblendet wurden. Der Filtermechanismus reagierte in allen Fällen zuverlässig und erfüllte die definierten Anforderungen vollständig.

Grafik-Test

Die Tests im Rahmen der grafischen Darstellung wurden während der aktiven Aufzeichnung von Echtzeitdaten aus dem laufenden CAN-Bus durchgeführt. In diesem Test lag der Fokus darauf, die Visualisierung von CAN-Nachrichtenbytes hinsichtlich Geschwindigkeit, Stabilität und Genauigkeit zu prüfen. Unterschiedliche Bytes von einem Byte bis acht Bytes und Maximal Wert (Y-Achse) mit verschiedenen Eingaben

getestet. Dabei wurde auch besonders auf Echtzeit-Anzeigen Fokus gelegt und intensiv getestet. Darüber hinaus die wurde auch die Zeitachsenumstellung mit definierten Zeitenabschnitten getestet. Die grafische Darstellung lief stabil, flüssig und fehlerfrei. Selbst unter hoher Nachrichtenlast und bei kontinuierlicher Datenvisualisierung wurden keine Verzögerungen oder Darstellungsprobleme festgestellt.

Nach allen Prüfungen und Beobachtungen konnte festgestellt werden, dass der CAN Explorer in allen Tests einwandfrei funktioniert und keine Fehler aufweist.

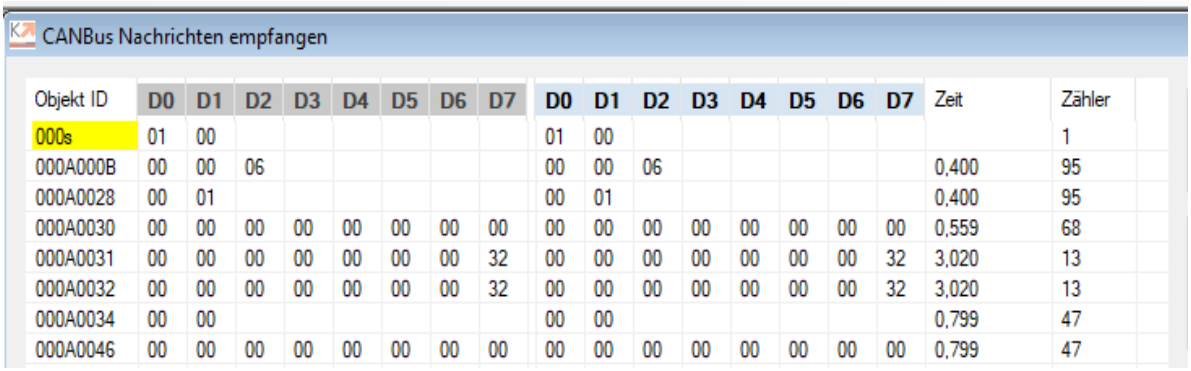
6 Evaluation

In diesem Kapitel wird die im Rahmen dieser Arbeit entwickelte Anwendung im Hinblick auf Zielerreichung, Qualität der Umsetzung sowie Stärken und Schwächen reflektiert. Darüber hinaus werden erste Rückmeldungen aus dem Praxiseinsatz sowie das Feedback von Kinemotion betrachtet und Verbesserungsmöglichkeiten unterhalten.

6.1 Vergleich mit dem alten CAN-Explorer

CAN-Nachrichtentabelle

Im alten CAN Explorer (Abbildung 6.1) wurden die empfangenen Nachrichten unnötigerweise doppelt dargestellt, was zu einer unübersichtlichen Darstellung und Platzverschwendung führte.



Objekt ID	D0	D1	D2	D3	D4	D5	D6	D7	D0	D1	D2	D3	D4	D5	D6	D7	Zeit	Zähler
000s	01	00							01	00								1
000A000B	00	00	06						00	00	06						0,400	95
000A0028	00	01							00	01							0,400	95
000A0030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	0,559	68
000A0031	00	00	00	00	00	00	00	32	00	00	00	00	00	00	00	32	3,020	13
000A0032	00	00	00	00	00	00	00	32	00	00	00	00	00	00	00	32	3,020	13
000A0034	00	00							00	00							0,799	47
000A0046	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	0,799	47

Abbildung 6.1: Alte Tabellenansicht der CAN-Nachrichten.

Im Gegensatz dazu wurde in der im Rahmen dieser Arbeit entwickelten Lösung die Nachrichtentabelle bewusst nur einmal dargestellt (Abbildung 6.2). Dadurch erhalten die Benutzer einen besseren Überblick über die Daten und profitieren von einer aufgeräumten Oberfläche. Außerdem wird viel Platz eingespart, der für weitere Funktionserweiterungen genutzt werden kann.

Objektname	ObjektID	D0	D1	D2	D3	D4	D5	D6	D7	Zeit Stempel	Nachrichten Anzahl
0	00007003									0.021	893
1	00006051	C4	14	08	00	00	00	00	80	0.641	30
2	00030003									0.055	90
3	00030004									0.400	45
4	000E0151	09	2C	00	00	02	0C	0D	01	0.004	540
5	000E0152	02	24	00	03	03	07	05	02	0.500	36
6	000A0030	00	00	00	00	00	00	00	00	0.560	32

Abbildung 6.2: neue Tabellenansicht der CAN-Nachrichten.

Ein weiterer Vorteil der neuen Darstellung ist die zusätzliche Spalte „Objektname“, die es ermöglicht, einzelnen CAN-Nachrichten bei späterer Weiterentwicklung eindeutige Namen zuzuweisen. Dies verbessert die Lesbarkeit und erleichtert insbesondere bei komplexeren Systemen die Identifikation und Interpretation der Daten.

Nachrichtenfilterung

Auf der linken Seite des Fensters (Abbildung 6.3) ist zu sehen, dass die Änderungsmöglichkeiten der verschiedenen Filtereinstellungen neben der Nachrichtentabelle angeordnet sind. Diese Anordnung dient den Benutzern dazu die Funktionen einfach zu benutzen und die Gesamtübersicht über die Anwendung beizubehalten.

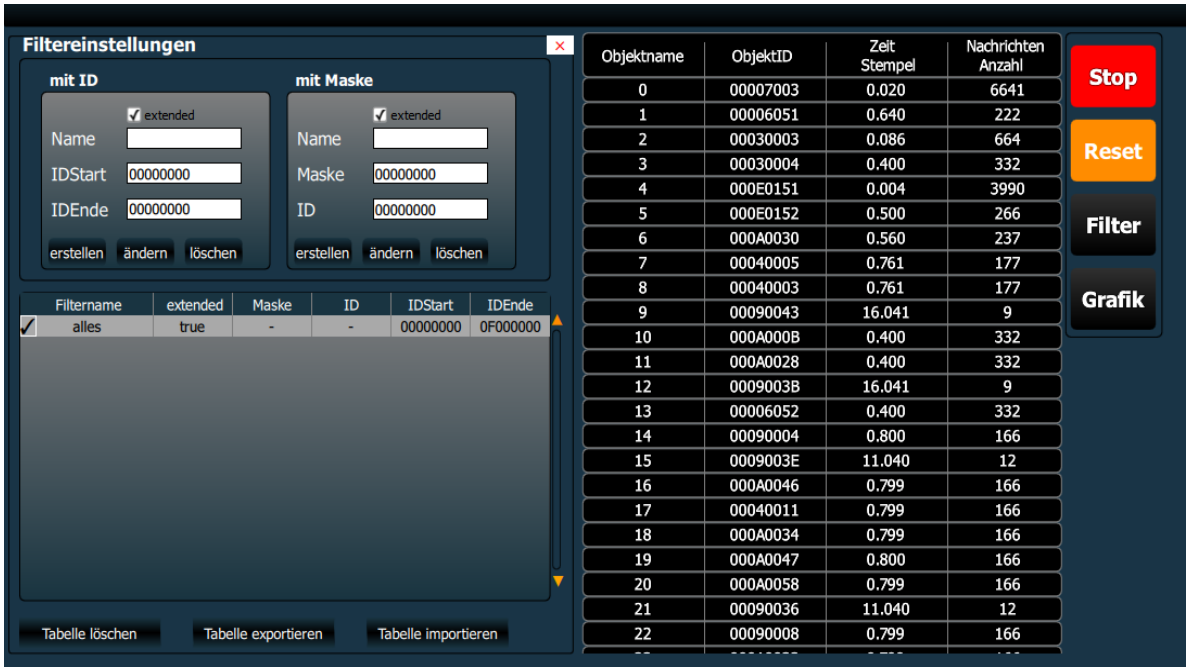


Abbildung 6.3: Das Filterfenster des neuen CAN Explorers

Im Vergleich zur Filterfunktion der alten Anwendung (Abbildung 6.4) bringt die neue Lösung eine ganze Reihe von Vorteilen mit sich. Das Filterfenster ist übersichtlich von oben nach unten aufgebaut, was die Bedienung deutlich intuitiver macht. Die Erstellung neuer Filter findet jetzt im oberen Bereich des Fensters statt, während die bestehende Filtertabelle weiter unten angezeigt wird. So bleibt alles klar strukturiert und leicht zugänglich. Dies sorgt für eine intuitive Nutzung und eine klare Übersicht. In der alten Anwendung war dies umgekehrt. Die Filtererstellung befand sich unten und die Filtertabelle oben, was die Bedienung weniger komfortabel machte.

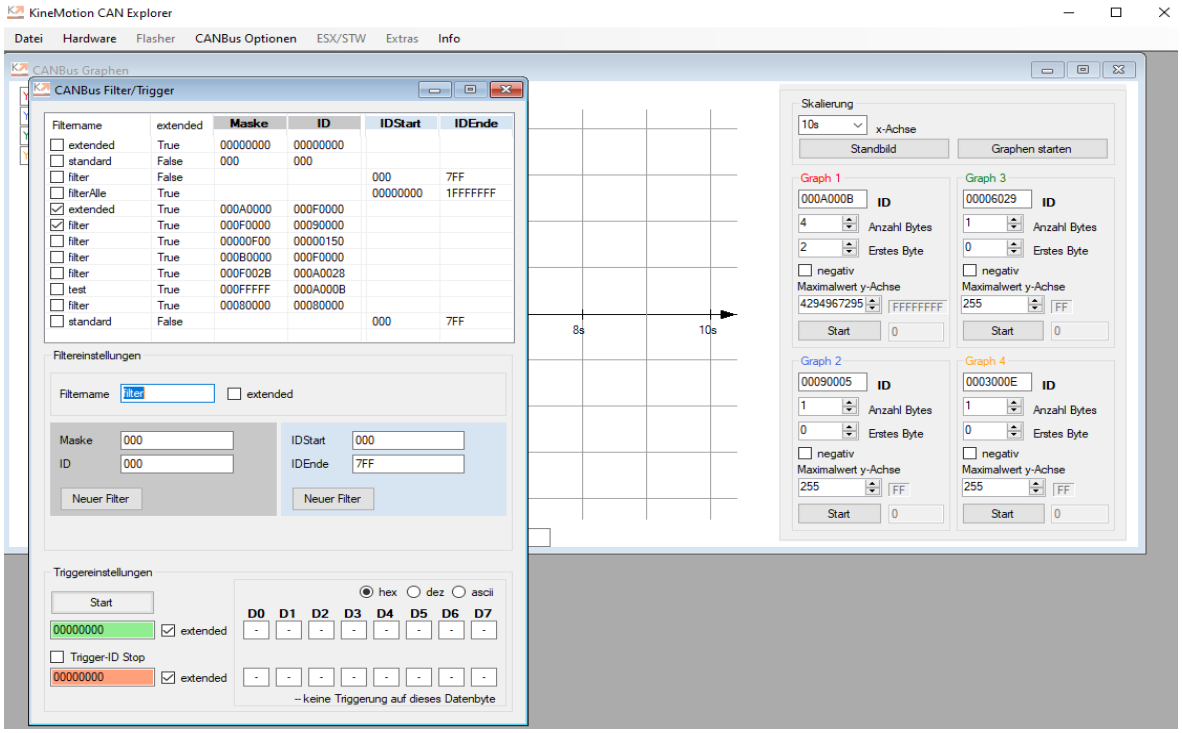


Abbildung 6.4: Das Filterfenster von alten CAN Explorer

Ein wesentlicher Vorteil der neuen Anwendung ist die gleichzeitige Sichtbarkeit der Nachrichtentabelle neben dem Filterfenster (Abbildung 6.3). Dadurch kann der Benutzer beim Anwenden eines oder mehrerer Filter in Echtzeit beobachten, welche Nachrichten weiterhin durchgelassen werden. In der alten Anwendung musste das Filterfenster zunächst geschlossen werden, um wieder zur Nachrichtentabelle zu sehen.

Ein weiterer wesentlicher Vorteil ist die klare Trennung der beiden Filterbereiche: Der Maskenfilter und der Bereichsfilter (ID-Start bis ID-Ende) verfügen nun jeweils über eigene Namensräume für die Vergabe von Bezeichnungen. In der Vorgängerversion gab es nur eine gemeinsame Bezeichnung für beide Filtertypen, was in der Praxis häufig zu Verwechslungen geführt hat.

Automatische Umwandlung der Buchstaben

Im Gegensatz zur alten Anwendung unterstützt das neu entwickelte Menü eine automatische Umwandlung eingegebener Zeichen in Großbuchstaben. Dies erleichtert die Eingabe, dass der nicht daran denken muss.

Zudem wurde die Bedienung durch eine verbesserte Tastaturnavigation optimiert. Der Benutzer kann mithilfe der Tabulatortaste schnell und unkompliziert zwischen der Namens- und ID-Eingabe wechseln. Dies ermöglicht schnelle Eingabe und freie Sprung, was zur Steigerung der Benutzerfreundlichkeit dient.

Warnungstext bei Fehlgaben

In der alten CAN Explorer-Anwendung war es möglich, dass Filter denselben Namen oder identische Bereiche und Masken haben konnten, was zu Verwirrung bei den Benutzern führte, da es schwer nachvollziehbar war, welches Filter genau welche Funktion hatte.

Im Rahmen dieses Projekts wurde dieses Problem gezielt behoben. Wenn der User versucht, einen bereits vergebenen Namen, einen identischen Bereich oder eine gleiche Maske erneut einzutragen, zeigt die Anwendung eine aussagekräftige Fehlermeldung an (Abbildung 6.5). Die Meldung weist den Benutzer darauf hin, dass der Eintrag bereits existiert und nicht doppelt angelegt werden kann.

Abbildung 6.5: ein Bespiel für kurze Warnung.

Darüber hinaus beim Erstellen eines Filters ohne Namen erscheint eine Hinweismeldung, die darauf aufmerksam macht, dass der Filter erst dann gespeichert werden kann, wenn ein eindeutiger Name vergeben wurde. Dies verhindert leere oder nicht nachvollziehbare Einträge in der Filtertabelle. Zudem wird überprüft, ob bei der Erstellung eines Filters für Extended-Nachrichten exakt 8 Byte und bei Standard-Nachrichten exakt 3 Byte angegeben wurden. Werden andere Längen eingegeben, weist die Anwendung mit einem Hinweis darauf hin, dass die Byte-Anzahl entsprechend den Vorgaben korrigiert werden muss.

Diese klaren und sofort sichtbaren Rückmeldungen helfen dabei, Fehleingaben direkt zu erkennen und zu korrigieren, wodurch die Zuverlässigkeit und Qualität der Filterdefinitionen deutlich verbessert wird.

Grafische Darstellung

Im Rahmen dieser Arbeit wurde eine neue grafische Darstellung für CAN-Nachrichten entwickelt, die im Vergleich zur alten CAN Explorer (Abbildung 6.1) mehrere wesentliche Verbesserungen bietet (Abbildung 6.2). Dies erhöhen die Lesbarkeit, Bedienbarkeit und Analysefähigkeit deutlich.

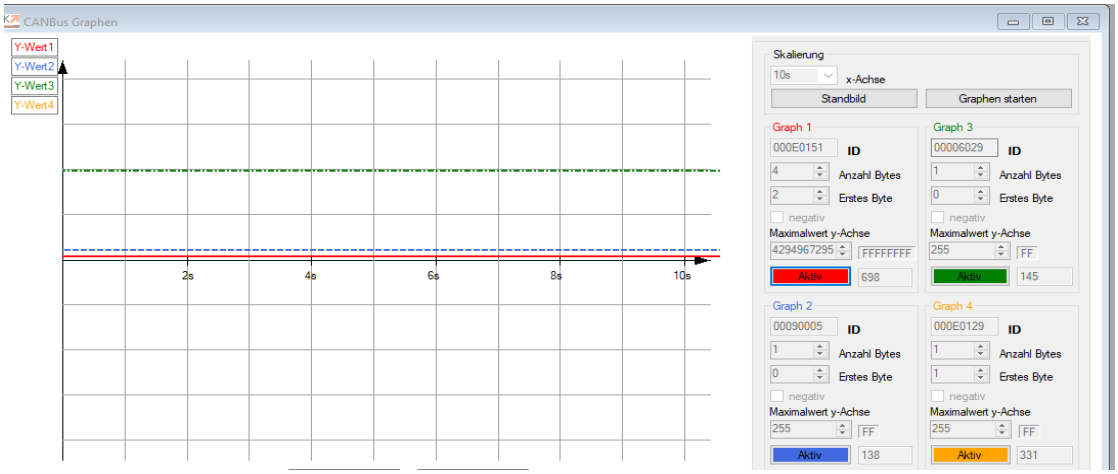


Abbildung 6.1: Alte grafische Darstellung der CAN-Nachrichten

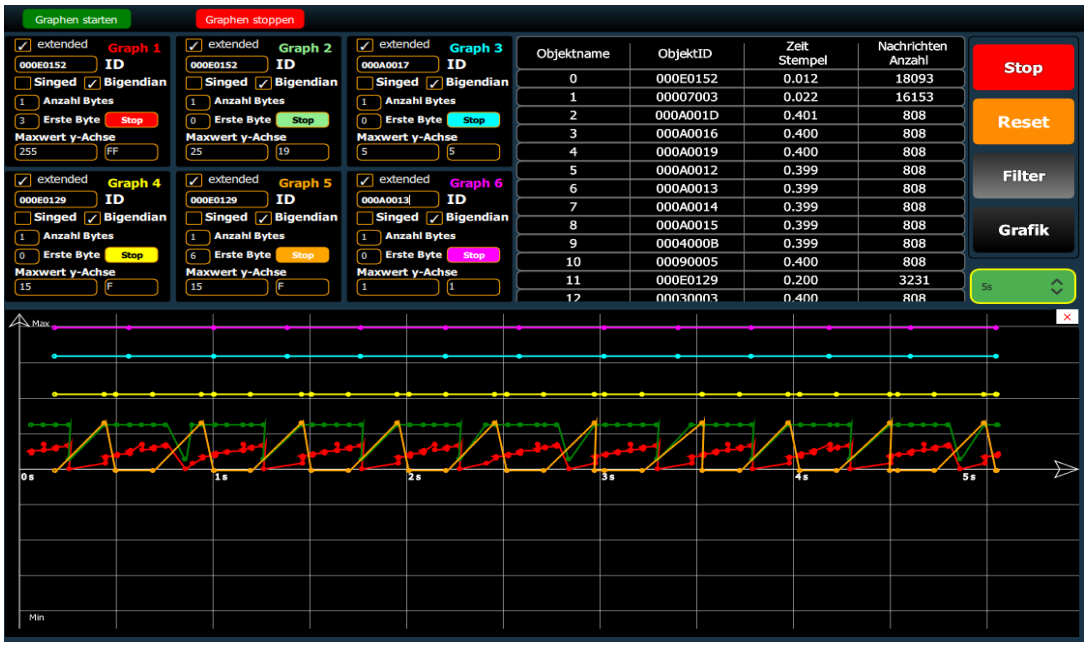


Abbildung 6.2: Grafische Darstellung der CAN-Nachrichten

Mehr Nachrichtenvisualisierung

Statt wie bisher vier werden jetzt bis zu sechs CAN-Nachrichten gleichzeitig angezeigt. Das erleichtert die gleichzeitige Analyse mehrerer Nachrichten.

Tabellenansicht
Bisher verschwand die tabellarische Darstellung, sobald die Grafikansicht aktiviert wurde. In der neuen Software bleiben beide Ansichten parallel in verkürzter Form sichtbar. So lassen sich Daten noch besser vergleichen und auswerten.

Unterstützung für Big-Endian
Da unterschiedliche Steuergeräte verschiedene Byte-Reihenfolgen verwenden, wurde die Unterstützung für Big-Endian-Darstellung ergänzt. Wird keine Auswahl getroffen, erfolgt die Berechnung automatisch im Little-Endian-Format. Diese Funktion macht die Software deutlich vielseitiger und kompatibler mit realen Fahrzeugarchitekturen.

Anpassung Während der Live-Aufzeichnung
Während der Live-Aufzeichnung in der grafischen Darstellung bietet die Anwendung die Möglichkeit, wichtige Anzeigeparameter direkt zur Laufzeit anzupassen. Der Benutzer kann beispielsweise die Anzahl der darzustellenden Bytes ändern oder den Maximalwert der Y-Achse anpassen, um den Wertebereich übersichtlicher darzustellen. Auch die Dauer der Zeitachse (X-Achse) lässt sich flexibel verändern, um kurze oder lange Zeitverläufe gezielt zu analysieren. Darüber hinaus kann während der laufenden Visualisierung festgelegt werden, ob die Daten als „signed“ oder „unsigned“ interpretiert werden sollen. Ebenso kann der Benutzer zwischen Big-Endian und Little-Endian wählen, indem er einfach auf jeweiligen Kästchen Haken setzt oder wegnimmt. Diese Möglichkeit zur dynamischen Anpassung während der Aufzeichnung macht die Analyse deutlich flexibler und effizienter, ohne die Visualisierung zu unterbrechen.

6.2 Erfüllung der nicht funktionale Anforderungen

Leistung und Zuverlässigkeit
Die Anforderungen an Leistung und Zuverlässigkeit wurden in den Tests beim Fahrzeugsystem vollständig erfüllt. Während der Tests zeigte die Anwendung ein stabiles Laufverhalten, selbst unter hoher Buslast und bei Echtzeitdatenverarbeitung. Es traten keine Abstürze oder Fehlfunktionen auf, was die Zuverlässigkeit der Software unter praxisnahen Bedingungen bestätigt.

Skalierbarkeit
In der neu entwickelten CAN-Explorer-Anwendung werden alle UI-Elemente dynamisch mitskaliert, wenn die Größe des Hauptfensters verändert wird. Dadurch bleibt die Benutzeroberfläche übersichtlich und durchgängig bedienbar, unabhängig von der Fenstergröße. Diese Flexibilität ermöglicht es den Nutzerinnen und Nutzern, die Anwendung auch bei kleineren oder größeren Bildschirmen komfortabel und effizient zu verwenden. Im Gegensatz dazu bot die bisherige Softwarelösung keine skalierbare

Oberfläche. Beim Ändern der Fenstergröße kam es dort häufig zu Überlappungen, abgeschnittenen Elementen oder unübersichtlicher Darstellung, was die Benutzerfreundlichkeit erheblich beeinträchtigt.

Benutzerfreundlichkeit.

Diese Anforderung wurde in der neuen Anwendung vollständig erfüllt. Die Farbkombination, sowie die Größe und Proportionierung der UI-Elemente, wurden gezielt so gewählt, dass die Oberfläche modern wirkt, gleichzeitig aber übersichtlich und gut bedienbar bleibt. Dadurch ist die Anwendung sowohl optisch ansprechend als auch funktional komfortabel in der täglichen Nutzung.

Integration zu KMWare

Im Laufe der Entwicklung wurde darauf geachtet, dass die neue Softwarelösung sich später in das bestehende Gesamtbild von KMWare (Abbildung 4.6 und Abbildung 4.7) einfügt. Besondere Blick lag auf der einheitlichen Benutzerführung und dem konsistenten Erscheinungsbild. Ziel war es, dass das Aussehen und die Bedienlogik der Anwendung dem Nutzer vertraut erscheinen, zumindest sehr ähnlich zu anderen Modulen innerhalb der KMWare-Plattform.

Darüber hinaus wurde bewusst darauf geachtet, bei der Implementierung ausschließlich bestehende Komponenten und Strukturen von KMWare zu verwenden. Dies minimiert potenzielle Kompatibilitätsprobleme und sorgt dafür, dass sich die neue Anwendung mit wenig Änderungen in die bestehende Systemumgebung integrieren lässt.

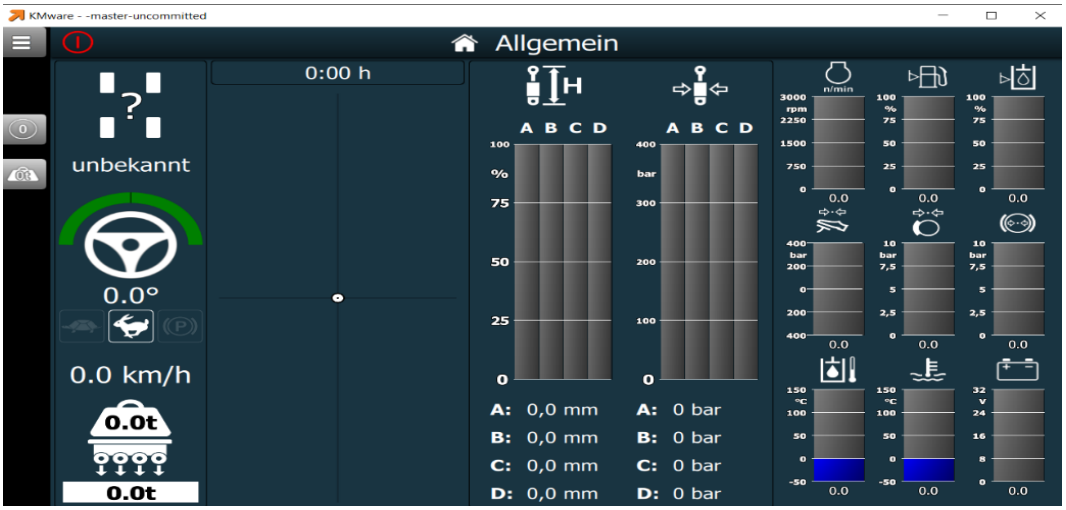


Abbildung 4.6: die Anwendung „KMWare“

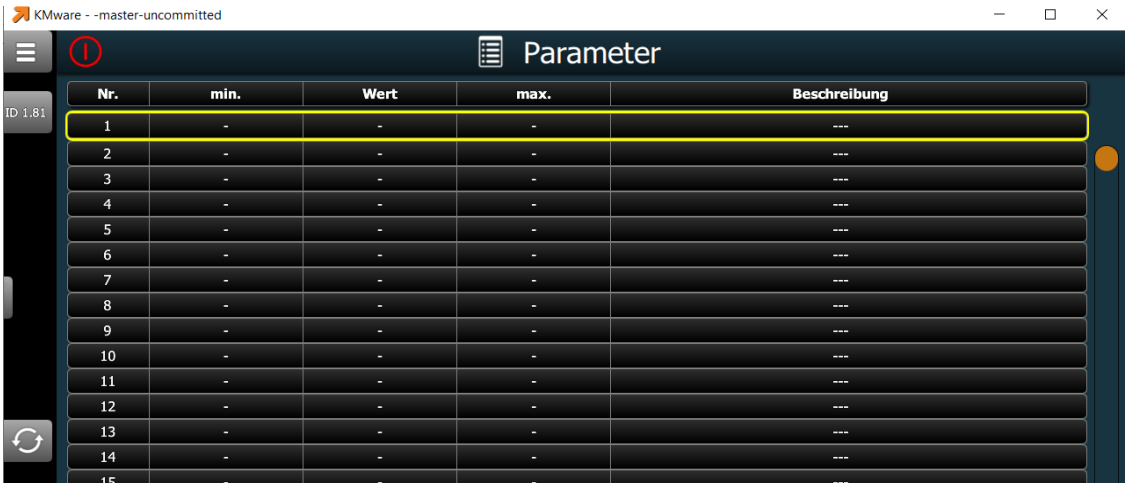


Abbildung 4.7: die Anwendung „KMWare“; Parametermenü

Benutzeranleitung.

Es wurde eine gut verständliche Benutzeranleitung erstellt, die sowohl Fahrzeugingenieuren als auch Endkundinnen und Endkunden den Einstieg in die Anwendung erleichtert. Die Anleitung erklärt alle wesentlichen Funktionen und unterstützt den Nutzer bei der eigenständigen Bedienung der Software. Die Benutzeranleitung ist im Anhang dieser Arbeit enthalten.

Zusätzlich wurde Wert auf die Kommentierung des Quellcodes gelegt. Der Programmcode enthält ausführliche, verständliche Kommentare, die eine nachvollziehbare Struktur und eine problemlose Weiterentwicklung durch Dritte ermöglichen.

6.3 Kosten und Speicherplatzanalyse

Kostenvergleich

Wie in der Anforderungsanalyse (vgl. Kapitel 3) dargelegt, belaufen sich die Anschaffungskosten für eine Lizenz des kommerziellen Tools CANoe, Version 19, auf rund 12.000 Euro. Zusätzlich kann ein Wartungs- und Supportpaket für ca. 2.000 Euro pro Lizenz erworben werden. Für den Einsatz bei Kinemotion werden mindestens fünf solcher Lizenzen benötigt. Im Gegensatz dazu wurde der CAN Explorer als Eigenentwicklung im Rahmen dieser Bachelorarbeit konzipiert und realisiert. Die Entwicklungszeit für den CAN Explorer erstreckte sich über einen Zeitraum von etwa einem Jahr. Während dieser Phase wurden nicht nur die Softwareentwicklung vorangetrieben, sondern auch weitere projektrelevante Aufgaben für Kinemotion bearbeitet. Es wurde ungefähr 40 bis 50 Stunde Zeit pro Monat für die Entwicklung des CAN Explorer investiert. 18 Euro pro Stunde beliefen sich die Gesamtkosten auf geschätzte 10.000 bis 12.000 Euro.

Speicherplatzbedarf

Der Speicherplatzbedarf ist ein weiterer entscheidender Faktor. Die Software CANoe, Version 19, erfordert eine Installation von über 5 GB und stellt zudem erhebliche Anforderungen an die Rechenleistung des Systems. Im direkten Vergleich dazu ist der CAN Explorer mit einer Gesamtgröße von lediglich 143 MB deutlich kleiner. Dieser geringe Speicherbedarf resultiert in einer reduzierten Beanspruchung von Systemressourcen, was die Integration in beispielsweise KMWare erheblich vereinfacht.

6.4 Feedback von Kinemotion

Um gezieltes Feedback zur neuen Softwarelösung zu erhalten, wurde ein strukturierter Fragenbogen erstellt und innerhalb der Firma Kinemotion übergeben. Ziel war es, sowohl die praktische Nutzbarkeit als auch die technische Weiterverwendbarkeit der Anwendung aus Sicht der Firma zu bewerten.

Zwei Schlüsselpersonen haben hierzu ihre Rückmeldung gegeben:

Andreas Hasenfuß, technischer Leiter und Geschäftsführer der Firma, wird die neue Software regelmäßig im Arbeitsalltag nutzen. Seine Rückmeldung war insbesondere auf die Bedienbarkeit, Funktionalität und den praktischen Einsatz im Werkstattumfeld ausgerichtet. Er hat folgende Fragen beantwortet.

1. Werden die Anforderungen und Vorgaben für den praktischen Einsatz am Fahrzeug durch die neue CAN Explorer App erfüllt? Bitte erläutern Sie.

   „Die Funktionalen Anforderungen werden vollständig erfüllt, da alle im Projektumfang geforderten Funktionen vorhanden sind.“

2. Wie bewerten Sie die Stabilität und Zuverlässigkeit der App bei der Nutzung am Fahrzeug? Gab es Situationen, in denen die App unerwartet reagiert hat?

   „*Die App arbeitete im Test absturzfrei*.“

   „*Unerwarteterweise werden beim Wechsel der Darstellungs-Einstellungen die bereits empfangenen Daten verworfen und nur neu empfangene Daten angezeigt. Es waren hierzu aber im Vorfeld keine konkreten Vorgaben gemacht worden*.“

3. Welche Funktionen oder Aspekte der neuen App waren aus Ihrer Sicht besonders hilfreich im praktischen Einsatz?

   „Die neue App verfügt im Vergleich zur Vorgänger-App über mehr Flexibilität im Grafikmenü. Es können 6 statt 4 Werte gleichzeitig dargestellt werden außerdem ist die Darstellung von CAN-Nachrichten mit 11-bit Identifiern

möglich, was bei der alten App nicht der Fall war. Außerdem kann zwischen Little Endian und Big Endian unterschieden werden."

4. Wie kreativ wurde die Anwendung entwickelt? Welche eigenständigen Lösungen des Entwicklers in der App gefallen Ihnen besonders gut und warum? Bitte erläutern Sie anhand konkreter Beispiele.

   „In der Ansicht zur Einstellung der Filter und der Grafikansicht wurden Kurzübersichtsfester eingefügt, die die eingehenden CAN-Nachrichten nur mit Identifiern zeigen."

   „Bei der Eingabe der Filter wurden Überwachungen umgesetzt, die falsche oder unplausible Eingaben abfangen."

5. Gab es Aspekte der App, die für den praktischen Gebrauch ungeeignet oder problematisch waren? Wenn ja, welche und warum?

   „Die Reihenfolge, in der die CAN-Nachrichten in der Liste angezeigt werden kann, nicht verändert werden. In bestimmten Situationen oder bei Arbeit mit wenigen Filtereinstellungen ist die Arbeit mit der App daher unübersichtlich."

6. Haben Sie Verbesserungsvorschläge zur Bedienbarkeit und Benutzerfreundlichkeit der App, die aus Sicht der praktischen Nutzung wichtig wären?

   „Das Schließen der Filtereinstellungen oder Grafikanzeige ist durch den nötigen Doppelklick etwas unintuitiv – eine Bedienung ähnlich der Start/Stopp Funktion wäre für ungeübte Bediener vermutlich eingängiger."

Dick Juchter wird die Software ebenfalls aktiv verwenden und ist zudem einer der Hauptentwickler der KMWare. Er bewertete neben der Benutzererfahrung vor allem die technische Integrationsfähigkeit und Wartbarkeit der neuen Lösung – mit Blick auf eine mögliche Weiterentwicklung innerhalb von KMWare. Er hat diese spezifische Frage beantwortet:

1. Sind die technischen Anforderungen und Spezifikationen für die neue CAN Explorer App vollständig umgesetzt worden? Bitte erläutern Sie.

   „*Die Aufgabe, Entwicklung einer CAN Diagnose Software, wurde vollständig umgesetzt. Laut Anforderungsanalyse wurden Vorgaben an den Studenten gestellt, welche wie beschrieben, umgesetzt wurden*."

2. Wie beurteilen Sie den Aufbau, die Struktur und die Qualität des Quellcodes der neuen App hinsichtlich Wartbarkeit und Erweiterbarkeit?

„Wie in den Anforderungen beschrieben, legt unser Unternehmen einen besonderen Wert auf nachhaltige Softwareentwicklung. Da kleinere Firmen wie Kinemotion aus Kaufmännischer Sicht automatisch gezwungen dies einzuhalten, um Ihre Entwicklungskosten gering zu halten.
Die Wartbarkeit des Codes wurde durch die Nutzung von GIT umgesetzt. Hierzu sind alle Punkte zur Zufriedenheit umgesetzt“

3. Welche Aspekte im Quellcode der App sind aus Ihrer Sicht besonders gelungen oder besonders gut umgesetzt? Bitte konkretisieren Sie Ihre Einschätzung.

   „Aus der Anforderungsanalyse hatte der Student die Aufgabe den Code so zu gestalten, damit der genutzte Rechner möglichst gering belastet werden sollte. Dies hat die Anwendung beim Test sehr gut umgesetzt. Es wurden Möglichkeiten zur einfachen Ausgabe genutzt, welche einen durchschnittlichen Arbeitsplatzrechner unmerklich belasten“

4. Ist aus Ihrer Sicht der eingesetzten Technologie (Programmiersprachen, Frameworks, Libraries) passend gewählt und nachhaltig?

5. Wie bewerten Sie die Kompatibilität und Integrationsfähigkeit der neuen App in bestehende technische Systeme und Abläufe Ihres Unternehmens?

   „Diese Frage kann zu 100% als erfüllt angesehen werden. Durch den Einsatz der Programmiersprache QT als Entwicklungsumgebung für Grafische Anwendungen in Verbindung mit QML kann die Anwendung in unsere Firmenanwendungen implementiert werden.“

6. Wie bewerten Sie die Kompatibilität und Integrationsfähigkeit der neuen App in bestehende technische Systeme und Abläufe Ihres Unternehmens?

   „Durch die schon oben erwähnten Punkte ist der entwickelte Code, auch durch die Unterstützung von GIT als Versions- Kontroll- und Dokumentationstool sehr einfach immer wieder in verschiedene weitere Tools zu integrieren. Kompatibilität wird durch den QT-Standard und die Klassen- Kapselung erreicht. Dies wird die Anwendung auf lange Sicht wiederverwendbar machen.“

7. Welche Schwächen oder kritischen Punkte sehen Sie aktuell im Quellcode oder der technischen Struktur der App, die zukünftige Weiterentwicklungen erschweren könnten?

   „Der Code wurde detailliert in viele „kleine“ Klassen unterteilt. Weiter sollte die Vererbung dieser viel tiefer im Fokus behalten werden. Auch die Nutzung von

Membervraiablen dieser Klassen muss erweitert werden um Details wie das Aussehen der einzelnen Elemente über Membervariablen automatisch anpassen zu können. Dies wird bei der Integration der CANexplorer- Funktion in die schon erwähnte KMware nötig werden. Grafisch haben die beiden Anwendungen derzeit größere Unterschiede."

8. Gibt es konkrete technische Verbesserungen, Optimierungen oder Änderungen, die Sie zur Weiterentwicklung der CAN Explorer App vorschlagen würden?

   „*Für die Darstellung der CAN-Nachrichten wird eine Tabellarische Darstellung benötigt. Sobald die Nachrichtenanzahl die Anzahl der Zeilen übersteigt, sollte ein Scrollbalken vorhanden sein oder dem Benutzer in irgendeiner weise dargestellt werden, dass hier weitere Informationen im Hintergrund bestehen. Weiter wäre eine Sortierfunktion durch Doppelklick auf die gewünschte Spalte eine sehr hilfreiche Funktion*."

7 Fazit

Ziel dieser Arbeit war es, eine moderne, robuste und benutzerfreundliche Alternative zu einer bestehenden CAN-Explorer-Anwendung zu entwickeln, die in den Sonderfahrzeugen von Kinemotion zum Einsatz kommt. Dieses Ziel wurde vollständig erreicht: Die entwickelte Anwendung erfüllt die funktionalen und nicht funktionalen Anforderungen, ist visuell und technisch zeitgemäß gestaltet und lässt sich intuitiv bedienen.

Die Entwicklung stellte insbesondere zu Beginn eine große Herausforderung dar, da sie weitgehend eigenständig erfolgte. Die Auseinandersetzung mit der Thematik CAN-Bus-Kommunikation und Verarbeitung von Fahrzeugdaten waren sehr komplex. GUI-Entwicklung mit Echtzeitdaten und das Einbauen von Filterfunktionen haben viel Zeit und Energie gekostet. Dennoch konnte die Anwendung erfolgreich umgesetzt werden. Dazu gehören inklusive umfassender Tests sowohl durch Unit-Tests als auch durch den Einsatz in einem realen Fahrzeugsystem mit echtem CAN-Datenverkehr.

7.1 Stärken der Lösung

Im Rahmen dieser Bachelorarbeit wurde eine moderne und skalierbare Benutzeroberfläche entwickelt. Damit wurde die Bedienbarkeit gegenüber der Vorgängerversion deutlich verbessert. Durch die Verwendung klarer Gestaltungselemente und intuitiver Interaktionsmuster konnten Komplexitäten reduziert und die Effizienz der Bedienung nachhaltig erhöht werden. Das Benutzererlebnis profitiert dabei insbesondere

von einer übersichtlichen Anordnung der Bedienelemente und einer flüssigen Navigation durch die verschiedenen Anwendungsbereiche.

Ein zentraler Schwerpunkt lag darauf, eine zuverlässige Funktionalität sicherzustellen, welche auch unter realen Bedingungen stabil funktioniert. Hierbei wurden umfangreiche Praxistests mit Live-Daten direkt aus dem Fahrzeugsystem durchgeführt, um ein möglichst realistisches Nutzungsszenario abzubilden. Diese Herangehensweise erlaubte es, mögliche Fehler frühzeitig zu identifizieren und zu beheben, das hat wesentlich zur Robustheit und Betriebssicherheit der Software beigetragen.

Besonderen Wert wurde bei der Entwicklung auf eine gute Wartbarkeit und Erweiterbarkeit der Software gelegt. Dazu wurde eine klar strukturierte und modular aufgebaute Codebasis geschrieben, welche zukünftige Anpassungen und Erweiterungen deutlich vereinfacht. Die konsequente Einhaltung von klar definierten Programmierprinzipien erleichtert sowohl das Verständnis als auch die langfristige Pflege der Softwarekomponenten.

Zusätzlich wurde eine ausführliche und gut strukturierte Dokumentation erstellt, welche den Umgang mit der Anwendung erheblich vereinfacht und zukünftigen Entwicklern einen schnellen Einstieg ermöglicht.

7.2 Schwächen und Verbesserungspotenzial

Das Feedback der Mitarbeitenden von Kinemotion ist von großer Bedeutung. Die darin genannten Schwächen und Verbesserungsvorschläge sind zentral und sollten ernst genommen werden. Darüber hinaus könnten die Tests am Fahrzeugsystem strukturierter durchgeführt und vollständig protokolliert werden, um eine bessere Nachvollziehbarkeit und Qualitätssicherung zu gewährleisten. Zudem könnten die Eingabefelder je nach Benutzerwünsch ausgeblendet werden, Nachdem der Objekt ID im Eingabefeld des Grafikfensters eingegeben und Grafik gestartet wurde. Damit würde ein vollständige Tabellenansicht zusammen mit dem Grafikfenster ermöglicht.

7.3 Ausblick

In der zukünftigen Weiterentwicklung von CAN Explorer sind mehrere funktionale Erweiterungen vorgesehen. Die CAN-Nachrichten sollen benutzerdefinierte Namen erhalten können, um die Lesbarkeit und eine einfachere Zuordnung zu ermöglichen. Darüber hinaus wird es möglich sein, CAN-Nachrichten aus einer Datei einzulesen und zu speichern, damit die Wiederverwendbarkeit und den Datenaustausch erleichtert werden können. Zusätzlich ist die Integration einer Funktion zum Importieren von anderem Benutzer und Exportieren zu anderem Benutzer von Filtertabellen geplant.

Ein weiterer wichtiger Punkt ist die Implementierung des schreibenden Zugriffs auf den CAN-Bus, sodass künftig nicht nur empfangene Nachrichten analysiert, sondern auch gezielt Nachrichten an das Fahrzeugsystem durch CAN-Bus gesendet werden können. Abschließend ist vorgesehen, den gesamten CAN Explorer vollständig in KMWare zu integrieren, um eine nahtlose und benutzerfreundliche Nutzung innerhalb der bestehenden Systemumgebung zu gewährleisten.

8 Literaturverzeichnis

[1] Rausch, M. (2022). *Kommunikationssysteme im Automobil: LIN, CAN, CAN FD, CAN XL, FlexRay, Automotive Ethernet* Carl Hanser Verlag 2022. S.74. eISBN 978-3-446-47457-4

[2] Rausch, M. (2022). *Kommunikationssysteme im Automobil: LIN, CAN, CAN FD, CAN XL, FlexRay, Automotive Ethernet* Carl Hanser Verlag 2022. S.91. eISBN 978-3-446-47457-4

[3] Wedemeyer, Thomas: *Grundlegende Informationen zum CAN-Bus*. S. 3. https://www.thomas-wedemeyer.de/files/CAN.PDF (Zugriff am 25.03.2025).

[4] Vector Informatik GmbH: *CANoe – Entwicklung, Simulation und Test von ECUs und Netzwerken*. https://www.vector.com/de/de/produkte/produkte-a-z/software/canoe/# (Zugriff am 30.03. 2025)

[5] Vector Informatik GmbH: *CANoe – Entwicklung, Simulation und Test von ECUs und Netzwerken*. https://www.vector.com/de/de/produkte/produkte-a-z/software/canoe/#c386473 (Zugriff am 30. 03. 2025).

[6] Vector Informatik GmbH: *CANoe User Manual.* https://www.vector.com/us/en/download/canoe-full-installer-19/ (Zugriff am 20.06.2025)

[7] Joachim Goll: *Entwurfsprinzipien und Konstruktionskonzepte der Softwaretechnik.* S. 94. Springer Vieweg 2017. ISBN 978-3-658-20054-1

[8] Christoph Kecher, Alexander Salvanos: *UML 2.5: das umfassende Handbuch* Rheinwerk Computing 2015. ISBN 978-3-8362-2977-7

[9] Chris Rupp, Stefan Queins, die SOPHISTen: *UML 2 glasklar: Praxiswissen für die UML-Modellierung* Hanser Verlag 2012. eISBN 978-3-446-43197-3

[10] O. Vogel, I. Arnold, A. Chughtai, E. Ihler, T. Kehrer, U. Mehlig, U. Zdun.: *Software Architektur Grundlagen – Konzepte – Praxis.* Auflage 2. Spektrum Akademischer Ferlag. S. 48. https://link.springer.com/book/10.1007/978-3-8274-2267-5 ISBN 978-3-8274-1933-0

[11] IBM: *Was ist eine ereignisgesteuerte Architektur?* https://www.ibm.com/de-de/topics/event-driven-architecture (Zugriff am: 24.03.2025).

[12] FasterCapital: *Pipeline-Architektur – So definieren und entwerfen Sie Ihre Pipeline Architektur mithilfe von Mustern und Best Practices*, https://fastercapital.com/de/inhalt/Pipeline-Architektur--So-definieren-und-

entwerfen-Sie-Ihre-Pipeline-Architektur-mithilfe-von-Mustern-und-Best-Practices.html (Zugriff am: 24.04.2025).

[13] Chris Rupp, Stefan Queins, die SOPHISTen: *Praxiswissen für die UML-Modellierung.* Auflage: 4. S. 216. Hanser Verlag 2012. ISBN 978-3-446-43197-3

[14] Qt Qml: Qt 6.8. https://doc.qt.io/qt-6/qqmlapplicationengine.html (Zugriff am: 02.02.2025)

[15] Sontheimproduktkatalog: https://www.sontheim-industrie-elektronik.de/fileadmin/user_data/Dokumente/DE/Produktkatalog/Uebersicht_Interfaces_Gateways_DE.pdf S. 27. (Zugriff: am 04.01.2025)

[16] Qt Qml: Qt 6.8. https://doc.qt.io/qt-6/qabstractlistmodel.html (Zugriff am: 03.02.2025)

9 Benutzeranleitung

1. Überblick

CAN Explorer App ist ein Tool zur Live-Aufnahme und Anzeige von CAN-Bus-Nachrichten. Nach dem Start der Anwendung beginnt automatisch die Aufzeichnung der Bus-Nachrichten. Die Default Oberfläche gliedert sich in drei Bereiche:

- Linke Seitenleiste
- Nachrichten-Tabelle (Mitte)
- Aktionsbuttons (Rechts)

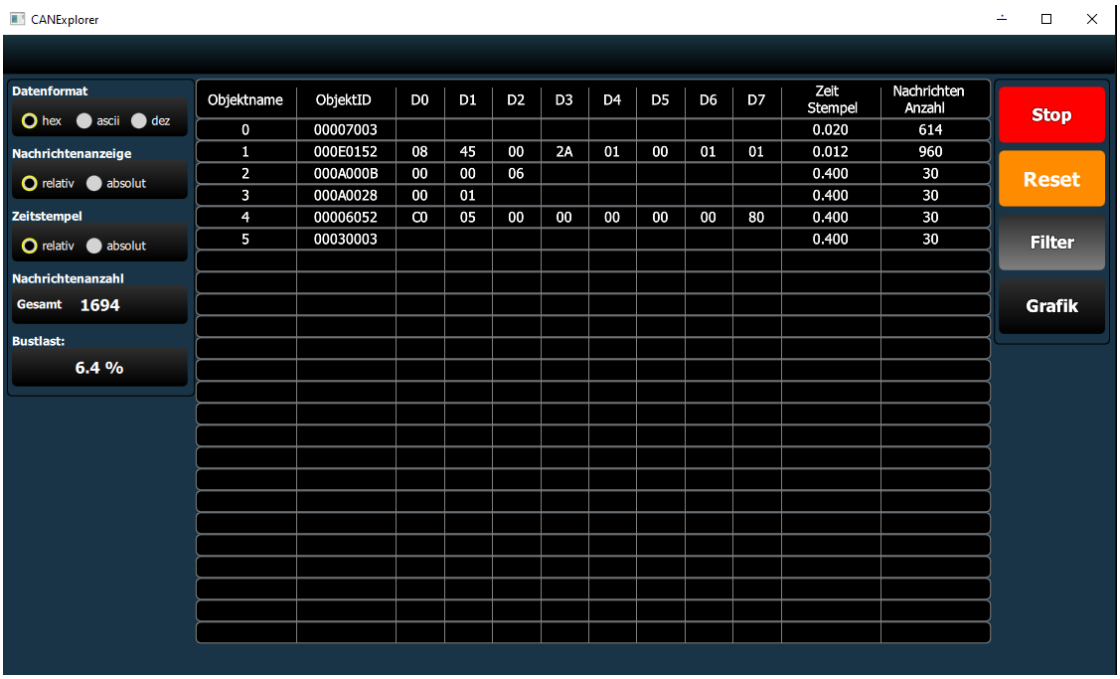


Abbildung 1: Default Fenster der Can Explorer nach dem Start.

2. Linke Seitenleiste

2.1 Datenformat

Hex: Wandelt die Daten Hexadezimal Format um

Ascii Wandelt die Daten Ascii Format um

Dez: Wandelt die Daten Dezimal Format um

Wählen Sie, in welchem Format die Datenbytes D0…D7 dargestellt werden sollen. Der Wechsel ist jederzeit möglich.

2.2 Nachrichtenanzeige

- **relativ** identische Nachrichten werden in der Tabelle überschrieben.
- **absolut** jede kommende Nachricht wird in neue Zeile gespeichert.

2.3 Zeitstempel

- **relativ**: nur Zeitdifferenz einer Nachricht seit letzter Ankunft.
- **absolut**: Gesamtzeit einer Nachricht seit Anfang/Reset

2.4 Nachrichtenzahl

- **Gesamt:** Zeigt die Gesamtzahl der seit Start/Reset empfangenen Nachrichten.

2.5 Buslast

- **Buslast:** Prozentuale Auslastung des CAN-Busses in Echtzeit.

Beispielserklärung von Abbildung 1:

- In diesem Beispiel ist das Datenformat auf **„Hex“** eingestellt. Dadurch werden die Rohdaten in der Tabelle im hexadezimalen Format angezeigt.
- Die Nachrichtenanzeige ist auf **„relativ“** gestellt. Das bedeutet, dass ältere Nachrichten durch neue überschrieben werden – es erfolgt also keine chronologische Auflistung, sondern eine fortlaufende Aktualisierung.
- Auch der Zeitstempel ist auf **„relativ“** eingestellt. Es wird dabei die Zeitdifferenz zwischen der aktuellen und der vorherigen Nachricht angezeigt, anstatt eines absoluten Zeitwertes.

3. Nachrichten-Tabelle

Objektname	Laufende Nummer (ab 0)
ObjektID	Hexadezimale CAN-Identifier inkl. Priorität
D0…D7	Datenbytes im gewählten Datenformat
Zeit Stempel	Zeit (relativ oder absolut)
Nachrichten Anzahl	Anzahl der Nachrichten mit dieser ObjektID

Neue Einträge werden unten angefügt.

4. Aktionsbuttons

Stop/Start: Stoppt (bzw. startet) die Aufnahme. Nach dem Stop-Vorgang wechselt die Beschriftung zu **Start**.

Zwischen Stop und Start können Sie jederzeit das Datenformat oder die Anzeigearten ändern, ohne die bisher gesammelten Daten zu verlieren.

Reset: Löscht alle bislang erfassten Daten und setzt Zähler (Nachrichten, Zeit) zurück. Nur aktiv, wenn die Aufnahme läuft.

Filter: Öffnet das Filterfenster (links in der Seitenleiste).

Grafik: Öffnet das Grafikfenster (unten in der Seitenleiste).

5.Filterfenster

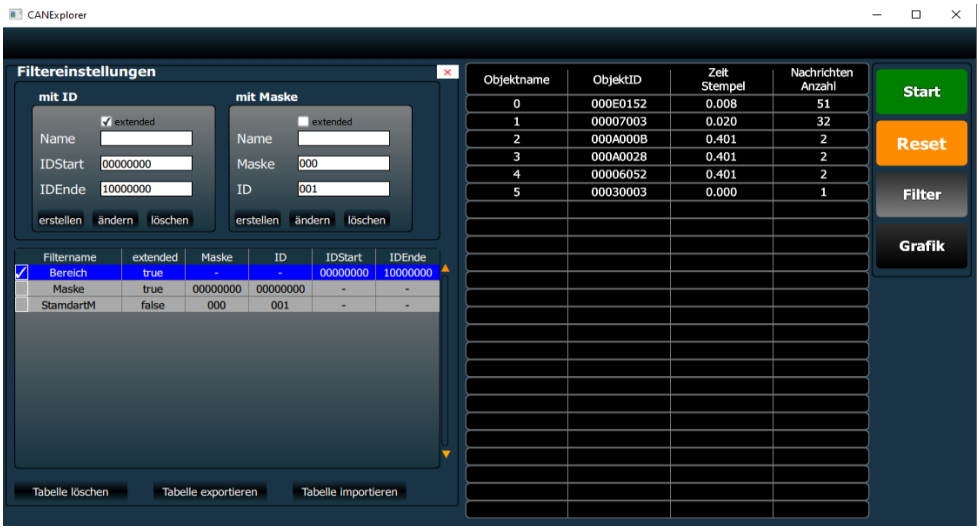


Abbildung 2: Filterfenster des Can Explorers.

- Mit einem Klick auf Filter erscheint ein zusätzliches Fenster in der linken Seitenleiste.
- Hier können Sie IDs, Dateninhalte oder Zeitbereiche eingrenzen.
- Schließen des Filterfensters:
  - ➢ Klick auf das “X” oben rechts im Filterbereich oder
  - ➢ Doppelklick auf den **Filter**-Button

6. Filtereinstellungen

Über den **Filter**-Button öffnen Sie das Filterfenster. Hier können Sie sowohl **Bereichsfilter (mit ID)** als auch **Maskenfilter (mit Maske)** anlegen, ändern und löschen.

6.1 Filtertypen

- **mit ID:** Definiert einen **Bereich** von IDs zwischen IDStart und IDEnde.
- **Mit Maske**: Definiert eine **Bitmaske** plus einen festen ID-Wert (ID).
- **extended-Haken:** Legt fest, ob Sie mit der vollen 29-Bit-ID (8 Hex-Digits) oder der Standard-11-Bit-ID (3 Hex-Digits) arbeiten.

6.2 Filter anlegen

1. **Filtertyp wählen**
   - Klicken Sie entweder auf den Bereich **mit ID** oder **mit Maske**.
   - Setzen Sie bei Bedarf den Haken **extended**, um 8-stellig (29 Bit) statt 3-stellig (11 Bit) zu filtern.
2. **Name eingeben**
   - Pflichtfeld! Ohne eindeutigen Namen erscheint eine Fehlermeldung.
3. **Felder ausfüllen**
   - **mit ID**: IDStart und IDEnde (Hex-Strings).
   - **mit Maske**: Maske (Hex-String) und ID (Hex-String).

4. **Erstellen**
    - Klick auf **Erstellen**.
    - Der neue Filter erscheint als letzte Zeile in der untersten Tabelle.
5. **Filter aktivieren für CAN-Nachrichten**
    - Um einen angelegten Filter **einzusetzen**, setzen Sie in der Filterliste das Häkchen ganz links vor dem gewünschten Filter-Namen (Bild 3).
    - **Nur** für alle aktiven (angekreuzten) Filter erfolgt anschließend eine **Live-Filterung** der eingehenden CAN-Nachrichten.
    - In der Nachrichten-Tabelle auf der rechten Seite sehen Sie sofort nur noch die Frames, die einem oder mehreren der aktivierten Filter entsprechen.

Bespielerklärung von Abbildung 2:

- In diesem Beispiel ist der **erste Filter in der Filterliste aktiv**, was durch das gesetzte Häkchen vor dem Namen des Filters sichtbar ist. Der Filter trägt den Namen **„Bereich“**.
- Die Option **„Extended“** ist auf **„true“** gesetzt, was bedeutet, dass es sich um einen **Extended-Filter für Extended-CAN-Nachrichten** handelt.
- Da es sich um einen **Bereichsfilter** handelt, bleiben die Spalten **„Maske“** und **„ID“** leer. Stattdessen sind die Spalten **„IDStart“** und **„IDEnde“** relevant:
    - **IDStart:** 00000000 – ab dieser ID beginnt der Filterbereich.
    - **IDEnde:** 01000000 – bis zu dieser ID wird der Bereich eingeschränkt.

Zusammengefasst:

Es werden **nur Nachrichten zugelassen**, deren CAN-IDs **zwischen 00000000 und 10000000** liegen. Alle Nachrichten außerhalb dieses Bereichs werden **gefiltert bzw. blockiert**. Das gleiche Prinzip gilt für **Standard** CAN-Nachrichten

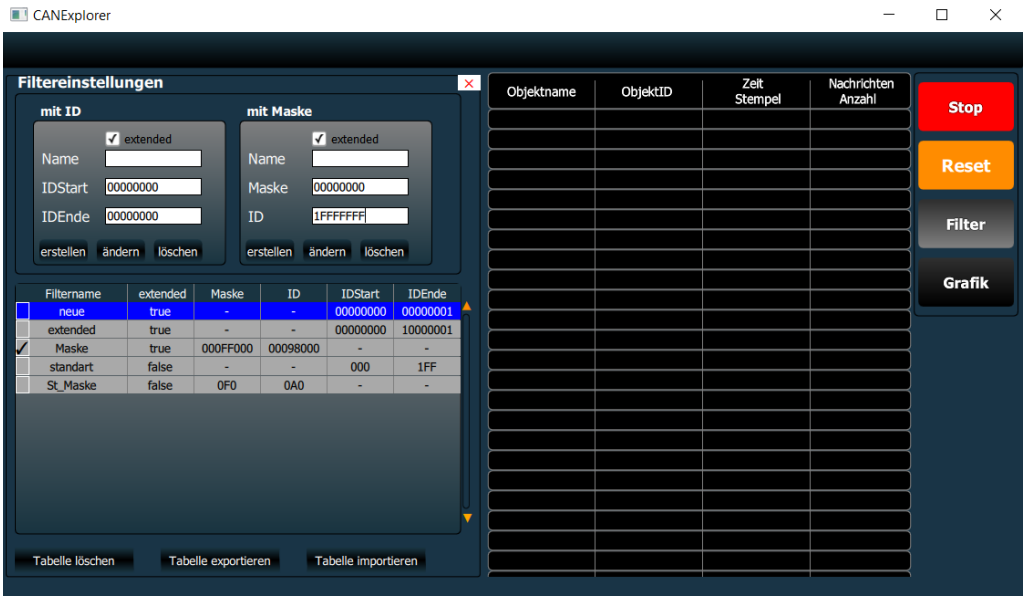


Abbildung 3: Maskenfilter des CAN Explorers

Bespielerklärung für Maskenfilter in Abbildung 3:

- In diesem Beispiel ist der **dritte in der Filterliste aktiv**, was durch das gesetzte Häkchen vor dem Namen des Filters sichtbar ist. Der Filter trägt den Namen **„Maske“**.
- Die Option **„Extended“** ist auf **„true“** gesetzt, was bedeutet, dass es sich um einen **Extended-Filter für Extended-CAN-Nachrichten** handelt.
- Da es sich um einen **Maskenfilter** handelt, bleiben die Spalten **„IDStart“** und **„ID Ende“** leer. Stattdessen sind die Spalten **„Maske“** und **„ID“** relevant:
    - **IDStart:** 000FF000 – ab dieser ID beginnt der Filterbereich.
    - **IDEnde:** 00098000 – bis zu dieser ID wird der Bereich eingeschränkt.

Erklärung der Funktion:

- Die **Maske** legt fest, **welche Bits bzw. Bytes** der CAN-ID bei der Filterung berücksichtigt werden sollen. In diesem Beispiel werden die **relevanten Bits im vierten und fünften Byte** ausgewertet.
- Die **ID** gibt an, **welcher exakte Wert** in den durch die Maske definierten Positionen vorhanden sein muss, damit eine Nachricht **durchgelassen** wird.

Zusammengefasst:

Es werden **nur CAN-Nachrichten zugelassen**, bei denen:

- im **vierten Byte** der Wert **9**, und
- im **fünften Byte** der Wert **8** gesetzt ist.

Alle anderen Nachrichten, bei denen diese Bedingungen nicht erfüllt sind, werden **gefiltert bzw. blockiert**. Das gleiche Prinzip gilt für **Standardmaskenfilter**.

6.3 Filter ändern oder löschen

1. **Filter auswählen**

- Klicken Sie auf die Zeile des Filters in der Filterliste doppelt.
- Die Werte werden wieder in die ursprünglichen Eingabefelder geladen (Abbildung 4).

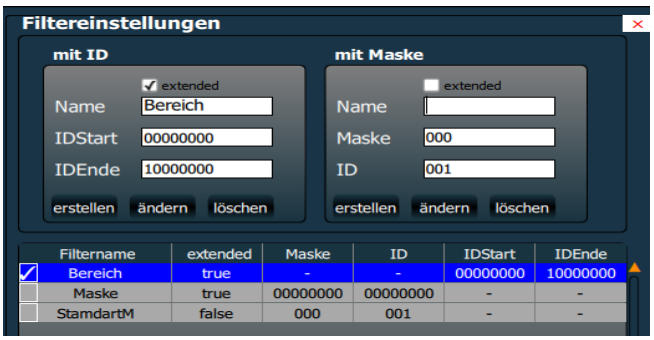


Abbildung 4: Filter laden zum Ändern oder Löschen

2. **Ändern**
    - Passen Sie Namen und Felder an und klicken Sie auf „**ändern“**.
    - Die Änderung wird an derselben Position in der Liste gespeichert.
3. **Löschen**
    - Nach Auswahl des Filters klicken Sie auf „**löschen“**, um ihn aus der Liste zu entfernen.

Beispielserklärung zum Ändern einen Filter von Abbildung 4.

1. **Doppelklick auf den Filter in der Tabelle:**
    - Durch einen **Doppelklick** auf einen bestehenden Filter in der Filtertabelle öffnet sich dieser wieder in seinem **ursprünglichen Eingabebereich**.
    - Handelt es sich z. B. um einen **Bereichsfilter**, erscheinen wieder die Felder **"IDStart"** und **"IDEnde"** zur Bearbeitung.
2. **Änderung des Filterbereichs:**
    - Nun können Sie den Filter wie gewünscht anpassen.
    - Beispiel:
        - Ändern Sie **IDStart** von "00000000" zu "00011000".
        - Ändern Sie **IDEnde** von "10000000" zu "09100000".
3. **Speichern der Änderungen:**
    - Klicken Sie anschließend auf den Button **„ändern“**.
    - Der Filter wird nun mit den neuen Werten **im selben Index** (an derselben Position) in der Filtertabelle aktualisiert und gespeichert (Abbildung 5).

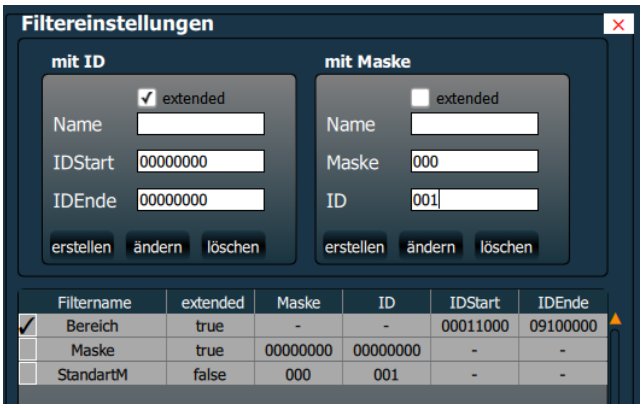


Abbildung 5 : geänderte Filter

6.4 Import, Export und Tabelle löschen

- **Tabelle löschen**: Entfernt **alle** eingetragenen Filter auf einmal.
- **Tabelle exportieren**: Speichert die gesamte Filterkonfiguration in eine Text Datei. (aktuell nicht aktiv)
- **Tabelle importieren**: Lädt Filterliste aus einer Text Datei. (aktuell nicht aktiv)

7. Grafische Auswertung („Grafik“)

- Mit einem Klick auf **Grafik** wechseln Sie von der Tabelle in die visuelle Darstellung. Das Fenster teilt sich in drei Bereiche:

    1. **Graph-Definition (oben links)**
    2. **Nachrichtentabelle (oben rechts)**
    3. **Grafikfenster (unten)**

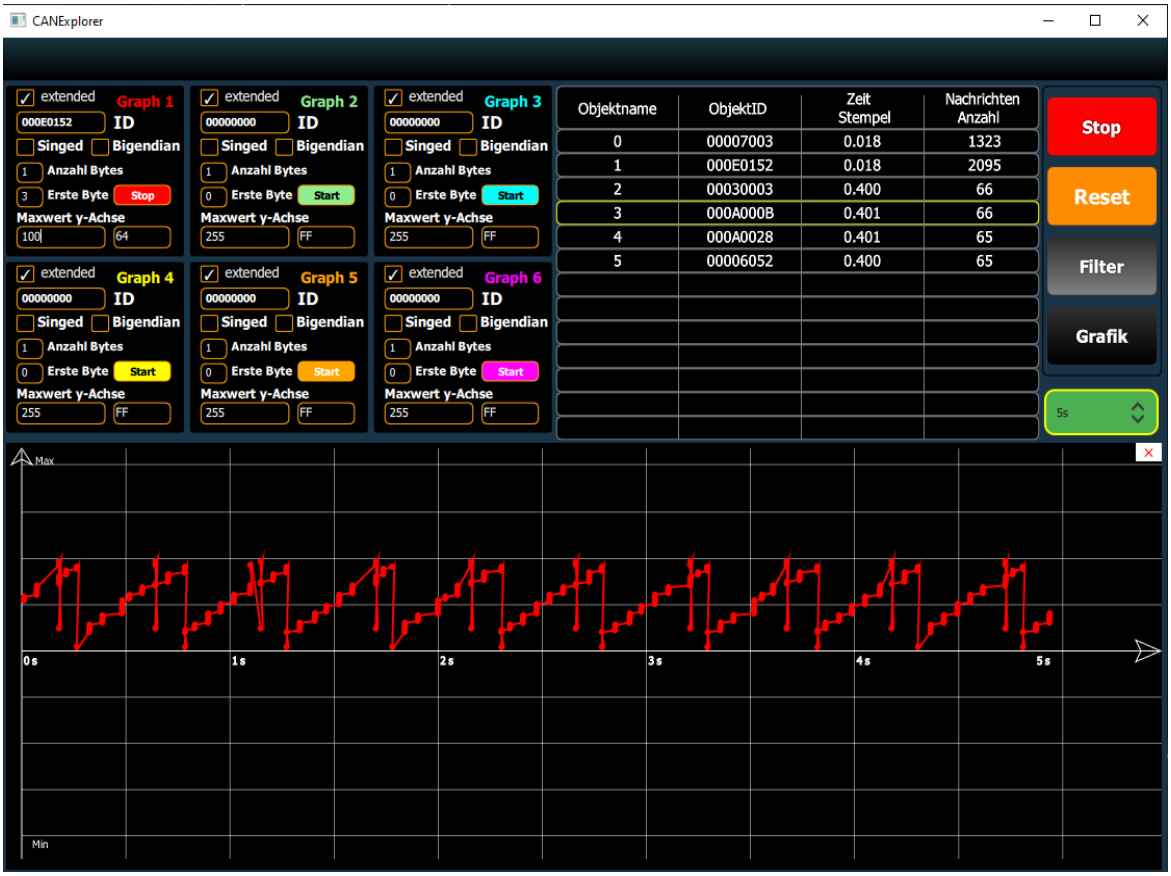


Abbildung 6: Grafik Fenster des Can Explorers

- Schließen des Filterfensters:
    - ➢ Klick auf das “X” oben rechts im Chartbereich oder
    - ➢ Doppelklick auf den **Grafik**-Button

8.1 Graph-Definition (bis zu 6 Kanäle)

Sie können bis zu **sechs** verschiedenen IDs gleichzeitig plotten. Jeder „Graph“ besitzt eigene Einstellungsmöglichkeiten:

Extended:	Bei Haken: 29-Bit-ID (8 Hex). Ohne Haken: 11-Bit-ID (3 Hex).
ID-Eingabe:	Tragen Sie die gewünschte CAN-ID (Hex) für diesen Graphen ein.
Signed:	Bei Haken: Werte als vorzeichenbehaftet interpretieren.

Bigendian:	Bei Haken: Big-Endian-Byte-Reihenfolge, andernfalls Little-Endian.
Anzahl Bytes:	Anzahl der zu verwendenden Datenbytes (1–8).
Erste Byte:	Index (0–7) des ersten angezeigten Bytes im Datenfeld.
Maxwert Y-Achse:	Obergrenze der y-Achse (dezimal und Hex) zur Skalierung.
Stop/Start:	Startet bzw. pausiert die Live-Aktualisierung dieses einzelnen Graphen.

- Sie können verschiedene Kombinationen aus Byte-Index und Länge nutzen, um z. B. Multi-Byte-Werte oder einzelne Bits im Zeitverlauf zu verfolgen.
- Filterfenster und Grafikfenster kann man nicht gleichzeitig öffnen. Wenn z B. Filterfenster offen ist und man öffnet das Grafikfenster dann schließt Filter Fenster automatisch.

Beispielserklärung von Abbildung 6.

Es ist nur eine Grafik aktiv – die **rote** Grafik

- Im gezeigten Beispiel wurde die *Objekt-ID* in das Eingabefeld eingetragen. In diesem Fall „000E0152“
- Die Optionen *Signed* sowie *Big Endian* sind nicht aktiviert.
- Es soll 1 Byte visualisiert werden, beginnend ab dem dritten Byte.
- Der Maximalwert der Y-Achse wurde auf 100 gesetzt. Auf der rechten Seite wird dieser Wert zusätzlich in hexadezimaler Darstellung angezeigt.
- Nach dem Drücken der *Start*-Taste wird im unteren Bereich des Grafikfensters die Visualisierung der Daten gestartet. In diesem Fall wird 1 Byte dargestellt.
- Die X-Achse (Zeitachse) ist aktuell auf 5 Sekunden eingestellt.

8.2 Zeitachsen-Intervall (oben rechts)

- Der grüne Button zeigt aktuell ausgewähltes Zeitfenster (z. B. **5s**).
- Klick auf die Pfeile oder direkt auf den Wert öffnet eine Auswahl:
  1 Sekunde, 2 Sekunde, 5 Sekunde, 10 Sekunde,
- Die X-Achse zeigt stets nur die letzten gestellten Sekunden.

8.3 Grafikfenster (unten)

- Live-Plott der aktiven Graph-Kanäle.
- **X-Achse:** Zeit (relativ zur letzten x-Sekunde Ansicht).
- **Y-Achse:** Werte gemäß Ihrer Einstellung (Min–Max).
- Jeder Graph hat eine eigene Farbe und Beschriftung (“Graph 1” … “Graph 6”).

10 Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

_____	_____	_____
Ort	Datum	Unterschrift im Original