600 Jahre Universität Rostock

Traditio et Innovatio

Bachelorarbeit zum Thema

Sicherheitsanalyse durch Entwicklung eines Rogue Device zur Echtzeitmanipulation maritimer Steuerungssysteme

Studiengang:	Informatik
Vorgelegt von:	Jakob Engelbert Tomahogh
Matrikelnummer:	221201101
Bearbeitungszeitraum:	15. November 2024 – 04. April 2025
Erstgutachter:	M. Sc. Marvin Davieds
Zweitgutachter:	Prof. Dr. rer. nat. Clemens H. Cap

Lehrstuhl für Informations- und Kommunikationsdienste
Institut für Informatik
Fakultät für Informatik und Elektrotechnik
Universität Rostock

Gedruckt auf säurefreiem und elementar-chlorfrei gebleichtem Papier.

Stand:: 23. März 2025

Inhalt

1 Einleitung **1**

1.1 Motivation . . . 1

1.2 Ziel der Arbeit . . . 2

1.3 Aufbau der Arbeit . . . 2

2 Grundlagen **3**

2.1 CAN-Bus . . . 3

2.1.1 J1939-Protokoll . . . 4

2.2 NMEA-0183 . . . 6

2.3 Raspberry Pi . . . 6

3 State of the Art **7**

3.1 Rogue Device . . . 7

3.1.1 Angriff mit einem Rogue Device . . . 8

3.2 Arbeit mit dem CAN-Bus . . . 8

3.2.1 Anbindung von Raspberry Pi an CAN-Bus . . . 8

3.2.2 Übersetzung von CAN-Nachrichten . . . 8

4 Konzept und Systemdesign **11**

4.1 Aufbau Schiffsysteme . . . 11

4.2 Steuerungslogik des Spiele-Controllers . . . 12

4.3 Integration des Rogue Device . . . 14

4.3.1 Manipulation des Gashebels . . . 15

4.3.2 Manipulation der Rudersteuerung . . . 15

4.3.3 Rückmeldung der Eingaben . . . 16

5 Implementierung **17**

5.1 Umsetzung des Rogue Device . . . 17

5.2 Verbindung Rogue Device - Controller . . . 17

5.3 Übersetzung Signale Controller - Schiff . . . 19

5.3.1 Eingabe-Interface . . . 19

5.3.2 CAN-Bus Nachrichtenkodierung . . . 20

6 Sicherheit **23**

6.1 Schwachstellen . . . 23

6.1.1 Schwachstellen am CAN-Bus . . . 23

6.1.2 Schwachstellen am J1939-Protokoll 24
6.1.3 Schwachstellen an der Rudersteuerung 25
6.2 Schutzmaßnahmen 26

7 Abschließende Betrachtung **28**
7.1 Ergebnis 28
7.1.1 Wie kann die Manipulation von Steuergeräten erschwert werden? . . 29
7.1.2 Relevanz für andere Schiffe 30
7.2 Ausblick 30

VIII Anhang **32**
VIII.1 Quellcode 32
VIII.2 Schaltpläne 32

Abbildungen **33**

Literatur **34**

Abstract

In dieser Machbarkeitsstudie wird untersucht, ob die Steuerung eines Schiffes durch Manipulation von Steuergeräten möglich ist. Es wird ein Rogue Device entwickelt, das externe Steurungsbefehle erhält und manipulierte Nachrichten auf die Schiffskommunikation sendet. Dabei liegt der Fokus der Arbeit auf der Motorsteuerung über einen CAN-Bus. Die Rudersteuerung mit einer seriellen Schnittstelle wird kurz betrachtet. Eine zentrale Fragestellung ist dabei, wie die Manipulation von Steuergeräten erschwert werden kann. Durch die Simulation eines Angriffs wird gezeigt, dass die Steuerung eines Schiffes durch Manipulation von Steuergeräten möglich ist. Jedoch wird zusätzliche Forschung benötigt, um eine tatsächliche externe Steuerung zu ermöglichen. Dadurch werden verschiedene Sicherheitslücken an dem CAN-Bus und dem J1939-Standard aufgezeigt. Die Ergebnisse dieser Arbeit sind auch für andere Schiffe relevant, da der CAN-Bus weit verbreitet ist.

Kapitel 1

Einleitung

In diesem Kapitel

1.1 Motivation 1
1.2 Ziel der Arbeit 2
1.3 Aufbau der Arbeit 2

1.1 Motivation

In der Seefahrt ist die Anzahl an Cyberangriffen in den letzten Jahren stark angestiegen. Von 2021 auf 2022 hat sich die Anzahl von öffentlich bekannten Angriffen mehr als verdoppelt. Im Jahr 2021 sind lediglich 24 Angriffe bekannt geworden, im Jahr 2022 waren es bereits 65 Angriffe [App]. Darunter fallen Angriffe auf Häfen, Schifffahrtsunternehmen und Schiffe. Jedes Jahr werden zahlreiche Passagiere mit Schiffen befördert. Dabei kann es vorkommen, dass bösartige Aktoren ein Teil dieser Passagiere sind. Ein Angriff auf die Steuerung eines Schiffes kann katastrophale Folgen haben. Dennoch ist die Sicherheit von Schiffen nicht immer ausreichend gewährleistet [RJ16]. Um das zu verdeutlichen, soll in dieser Arbeit durch physischen Zugriff auf ein Schiff die Kommunikation manipuliert werden. Damit soll die Kontrolle über die Steuerung des Schiffes übernommen werden. Auf großen Passagierschiffen ist das Risiko besonders hoch, da es schwierig ist, einen solchen Angriff rechtzeitig zu bemerken. Es besteht die Gefahr, dass unbefugte Personen unbemerkten Zugriff zu kritischen Systemen erhalten.

1.2 Ziel der Arbeit

Diese Arbeit untersucht die Möglichkeit eines Angriffes auf die Steuerung eines gegebenes Beispielschiffes. Dabei soll auf die notwendige Sicherheit in Schiffen aufmerksam gemacht werden. Die Arbeit soll zeigen, wie wichtig es ist, die Systemkommunikation eines Schiffes zu schützen. Durch die Untersuchung eines Angriffes sollen zwei zusätzliche Forschungsfragen beantwortet werden:
Wie kann die Manipulation von Steuergeräten auf Schiffen erschwert werden?
Wie relevant sind die Ergebnisse für andere Schiffe?
Dazu wird ein Rogue Device entwickelt, welches in der Lage, ist externe Steuerungsbefehle zu erhalten und mit diesen Kontrollnachrichten auf die Schiffskommunikation zu senden. Dabei wird auf Netzwerke zugegriffen, die eine Kommunikation zwischen den verschiedenen Steuerungssystemen des Schiffes ermöglichen. Als Kommunikation spielt dabei der CAN-Bus eine wichtige Rolle. Unter anderem wird auch auf eine serielle Schnittstelle zugegriffen, um mehr Kontrolle über das System zu erlangen.
Durch die Simulation eines Angriffes werden die Auswirkungen auf die Steuerung des Schiffes analysiert. Zusätzlich sollen auch mögliche Gegenmaßnahmen betrachtet werden.
Um die Sicherheitslücken zu veranschaulichen, soll das Schiff mit einem Gaming-Controller gesteuert werden. Dies soll zeigen, wie einfach es ist, die Steuerung eines Schiffes zu übernehmen, wenn die Kommunikation nicht ausreichend geschützt ist.

1.3 Aufbau der Arbeit

Dazu werden zuerst die Grundlagen für die Arbeit erläutert. Um die Sicherheit von einem CAN-Bus zu diskutieren, muss dieser im Vorfeld verstanden werden. Daher wird im Detail auf den CAN-Bus und den J1939-Standard eingegangen. Im gleichen Kapitel wird auch auf den aktuellen Stand der Technik eingegangen. Ein wichtiger Teil dabei sind schon vorhandene Tools, die als Hilfsmittel für die Arbeit genutzt werden können. Im nächsten Schritt werden Konzepte für die Steuerung des Schiffes entwickelt. Dabei wird auf die Steuerungslogik eingegangen, aber vor allem Integration des Rogue Devices in das System. Mit diesen Konzepten wird in die Implementierung eingestiegen. Dabei sind die Verbindung des Spiele-Controllers mit dem Rogue Device und die Kommunikation mit dem Schiff wichtige Schritte. In der Kommunikation mit dem Schiff wird auf die CAN-Bus Kommunikation und die serielle Kommunikation eingegangen. Damit kann die Machbarkeit des Angriffs getestet werden. Im Anschluss werden Sicherheitslücken aufgezeigt und mögliche Gegenmaßnahmen vorgeschlagen. Zum Schluss wird die Relevanz der Ergebnisse für andere Schiffe betrachtet und ein Ausblick gegeben.

Kapitel 2

Grundlagen

In diesem Kapitel

2.1 CAN-Bus 3
2.2 NMEA-0183 6
2.3 Raspberry Pi 6

In diesem Kapitel werden die Grundlagen für die Arbeit erläutert. Dabei wird sich auf die wichtigen Begriffe und Technologien für den Rahmen dieser Arbeit fokussiert. Damit wird ein Verständnis für die Arbeit geschaffen und die Grundlage für die weiteren Kapitel gelegt.

2.1 CAN-Bus

Bei einem CAN-Bus handelt es sich um eine serielle Netzwerktechnologie, die mehrere Geräte mit einem Kabel verbindet. Die Entwicklung des CAN-Bus wurde von Bosch im Jahr 1983 begonnen [Vos08, Seiten 2–10]. Im Jahr 1986 wurde der erste CAN-Bus Standard veröffentlicht. Die Motivation der Entwicklung war die effiziente Kommunikation zwischen den Steuergeräten in einem Auto. Als günstigen Seiteneffekt konnte damit die Kabelmenge reduziert werden, da alle Geräte mit einem Bus verbunden werden können. Durch die höhere Zuverlässigkeit und Funktionalität des CAN-Bus, wurde dieser schnell in der Autoindustrie etabliert. Aber auch in anderen Sektoren, wie z. B. der Medizintechnik, der Gebäudeautomation oder der Luftfahrt, spielt der CAN-Bus mittlwerweile eine wichtige Rolle.
Man sprich von einem Bussystem, da mehrere Geräte miteinander über diesen miteinander verbunden werden [Vos08, Seiten 13–19]. Die Geräte auf an dem Bus sind in dem Fall die Knoten. Alle sind gleichberechtigt und können Nachrichten senden und empfangen. Die Nachrichten werden nach Broadcasting-Prinzip übertragen. Dabei wird jede Nachricht von allen

Knoten empfangen, aber nur die Knoten, welche die Nachricht benötigen, verarbeiten sie. Die Nachrichten werden nicht explizit bestätigt, da dies zu einer größeren unnötigen Last auf dem Bus führen würde. Trotzdem erwartet der Sender eine positive Bestätigung der Nachricht. Das Passiert mit einem Acknowledge (ACK) Bit im CAN-Frame, wie in Abbildung 2.1 zu sehen ist. Wenn eine Nachricht nicht bestätigt wird, sendet der Sender eine Fehlermeldung auf den Bus. Wenn ein Sender zu viele fehlerhafte Nachrichten sendet, soll dieser sich selbst logisch von dem CAN-Bus trennen. Eine Nachricht wird als fehlerhaft angesehen, wenn z. B. die Prüfsumme nicht stimmt oder die Nachricht nicht dem Standard entspricht.

Abbildung 2.1: Aufbau eines CAN-Frames [Vos08]

Die auf den Bus gesendeten Daten werden mit einer Nachrichten-ID versehen, welche die Priorität der Nachricht angibt. Die Priorität existiert, damit zeitkritische Nachrichten schneller verarbeitet werden können.Die Nachrichten mit der niedrigsten ID haben die höchste Priorität. Die Maximale Länge einer Nachricht beträgt 8 Byte. Auf dem CAN-Bus kann eine maximale Baudrate von 1Mbit/s erreicht werden. Zusätzlich kann durch die geringe Länge der Nachrichten geringe Latenz erreicht werden. Insgesamt ist der CAN-Bus damit gut für Echtzeitanwendungen geeignet, da die Reaktionszeit der Knoten möglichst gering gehalten wird.
Alle Knoten in einem CAN-Bus sind mit einem zweiadrigen Kabel verbunden [Vos08, Seite 132]. Diese werden als High (CAN_H) und Low (CAN_L) bezeichnet. Der Bus ist an beiden Enden mit einem Widerstand von 120 Ω abgeschlossen, um Reflexionen zu vermeiden.
Eine Nachricht auf einem CAN-Bus ist in einem CAN-Frame verpackt [Vos08, Seite 36]. Der Aufbau eines CAN-Frames ist in Abbildung 2.1 zu sehen. Dieser beginnt mit einem Startbit, welches Start of Frame (SOF) genannt wird. Darauf folgt das Arbitration Field, in dem die Nachrichten-ID und ein Bit RTR (Remote Transmission Request) gesetzt wird. Das RTR-Bit wird gesetzt, wenn der Sender eine Antwort auf die Nachricht erwartet. Das Control Field wird für die Datengröße und die Nachrichtenlänge verwendet. Im Data Field sind die eigentlichen Nutzdaten kodiert. Das CRC Field enthält eine Prüfsumme, welches die Richtigkeit der Nachricht überprüft. Hier folgt ein ACK Field, welches die Prüfsumme bestätigt. Die Nachricht endet mit einem End of Frame (EOF). Danacht folgt ein Interframe Space (IFS) von 3 Bit, welches eine Pause zwischen den Nachrichten darstellt.

2.1.1 J1939-Protokoll

Das Standard CAN-Protokoll hat einen 11 Bit Identifier, während das erweiterte CAN-Protokoll 29 Bit Identifier besitzt [MG18]. Dabei gibt es mehrere Standards, die den erweiterten Identifier verwenden. Der relevante Standard für diese Arbeit ist der SAE J1939-Protokoll.

Die Society of Automotive Engineers (SAE) hat das J1939-Protokoll entwickelt, um die Kommunikation in Nutzfahrzeugen zu verbessern. Der Fokus liegt auf der Kommunikation mit dem Antrieb des Fahrzeugs. Der 11-Bit-Identifier wird in J1939 auf 29 Bit erweitert, um eine größere Anzahl verschiedener Nachrichten zu ermöglichen.
Auf einem CAN-Bus können der Standard 11 Bit Identifier und der erweiterte 29 Bit Identifier gleichzeitig verwendet werden. Eine Nachricht mit dem 11 Bit Identifier wird bevorzugt vor einer Nachricht mit 29 Bit Identifier, wenn diese die gleiche Priorität haben. Die spezifierten Baudraten sind hier 250 kbit/s und 500 kbit/s.

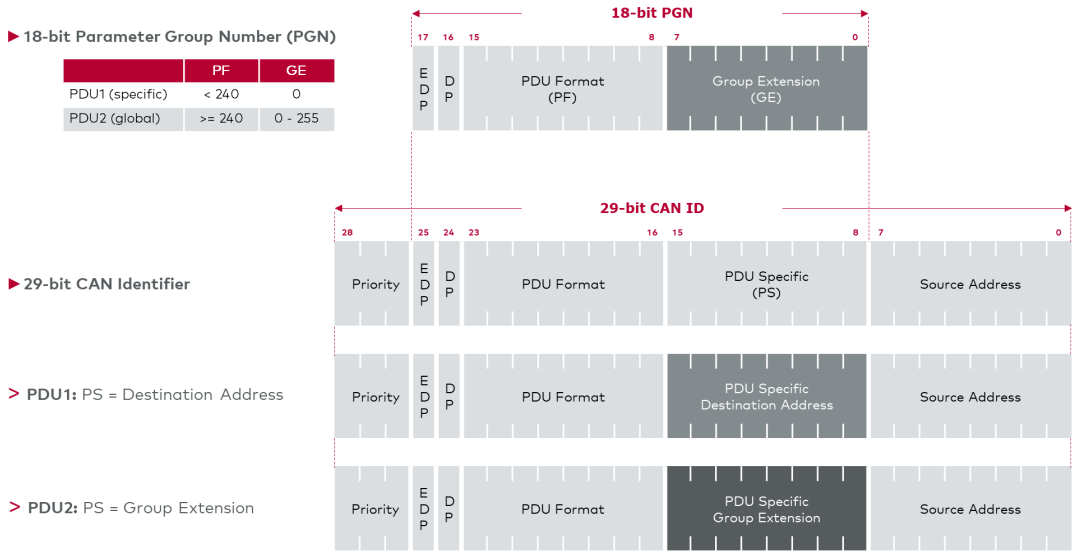


Abbildung 2.2: Header einer J1939-Nachricht auf dem CAN-Bus [Gmb]

In Abbildung 2.2 ist der 29 Bit Identifier einer J1939-Nachricht zu sehen. Die Hauptkomponenten des erweiterten Identifier sind die Priority, eine Parameter Group Number (PGN) und eine Quell-Adresse. Das PGN-Feld ist aus 1 Bit Data Page (DP), 1 Bit Extended Data Page (EDP), einer Protocol Data Unit (PDU) und einem PDU spezifischen Feld (PS) aufgebaut. Die DP und EDP bestimmen mit welchem Standard die Nachricht gesendet wird. Für SAE J1939 werden aber nur drei Kombinationen verwendet [Gmb]. Das PS-Feld kann entweder eine Zieladresse oder eine Gruppenerweiterung (GE) sein. Welches PS-Feld verwendet wird, hängt von der PDU ab. Wenn diese einen Wert von unter 240 hat, wird das PS-Feld als Zieladresse verwendet. In dem anderen Fall wird das PS-Feld als Gruppenerweiterung verwendet.
Trotz des Broadcasting-Prinzips soll nicht jede Nachricht von jedem Knoten verarbeitet werden [MG18]. Die Zieladresse ermöglicht es, dass eine Nachricht nur von dem designierten Knoten verarbeitet wird. Um eine Nachricht an alle Knoten zu senden, wird die Zieladresse auf 255 gesetzt. Mit der Gruppenerweiterung ist es nicht möglich eine Nachricht zielgerichtet an bestimmte Geräte zu senden.

2.2 NMEA-0183

NMEA-0183 ist ein Standard für die Kommunikation zwischen verschiedenen Geräten in der Schifffahrt [Ass]. Es handelt sich um einen Schnittstellenstandar, der von der National Marine Electronics Association (NMEA) entwickelt wurde. Er kann nur einen Sender haben, aber mehrere Empfänger. Die Daten werden in ASCII-Format übertragen. In den Daten können unter anderem Informationen über die Position, die Geschwindigkeit und die Richtung des Schiffes enthalten sein.

2.3 Raspberry Pi

Ein Raspberry Pi ist ein Einplatinencomputer, der von der Raspberry Pi Foundation entwickelt wurde. Dieser hat den Prozessor mit Grafikeinheit, Arbeitsspeicher, Speicher und Anschlüssen auf einem einzigen Board integriert. Es handelt sich um einen vollwertiger Computer, der kleiner als normale PCs ist. Zusätzlich kann dieser mit einem normalen Betriebssystem betrieben werden. Aus diesem Grund wurd der Raspberry Pi im Rahmen dieser Arbeit verwendet. Trotzdem gibt es ein spezielles Betriebssystem, das für den Raspberry Pi entwickelt wurde, das Raspberry Pi OS (ehemals Raspbian). Dieses basiert auf der Linux-Distribution Debian. Um viele Funktionen erfüllen zu können, hat der Raspberry Pi viele Anschlüsse. Der Raspberry Pi 5 hat 4 USB-Anschlüsse, 2 Micro-HDMI-Anschlüsse, 1 Ethernet-Anschluss, 1 USB-C-Anschluss für die Stromversorgung und einen Micro-SD-Kartensteckplatz.

Kapitel 3

State of the Art

In diesem Kapitel

3.1 Rogue Device 7
3.2 Arbeit mit dem CAN-Bus . . . 8

Für eine Sicherheitsanalyse von maritimen Steuerungssystemen ist es wichtig, die aktuelle Technik und die Möglichkeiten von Angreifern zu kennen. In diesem Kapitel wird der aktuelle Stand der Technik vorgestellt. Der Fokus liegt dabei auf Technologien, die für die Arbeit relevant sind.

3.1 Rogue Device

Unter einem Rogue Device versteht man ein Gerät, das sich unautorisiert und unauffällig in ein Netzwerk integriert [Sca+08]. Dies kann ein Raspberry Pi sein, der sich in ein Netzwerk einbindet und Daten abfängt oder manipuliert. Hierüber können sich Angreifer Zugriff auf das Netzwerk verschaffen. Ein solches Gerät kann dazu verwendet werden, um eigenen Code auszuführen. Der kann beispielsweise Daten veränden oder weitere Angriffe vorbereiten. Zudem kann es von Angreifern aus der Ferne gesteuert werden, wodurch gezielte Manipulationen oder Spionageangriffe vereinfacht werden. Darüber hinaus lässt sich ein Rogue Device nutzen, um Informationen über das Netzwerk zu sammeln, etwa durch das Mithören von Kommunikation oder die Analyse von Sicherheitsmechanismen. Das ermöglicht es, Schwachstellen im Netzwerk aufzudecken und gezielt auszunutzen.

3.1.1 Angriff mit einem Rogue Device

3.2 Arbeit mit dem CAN-Bus

Die Technologie des CAN-Bus wird in vielen Bereichen eingesetzt. Daher gibt es auch viele Tools und Bibliotheken, die es ermöglichen, mit dem CAN-Bus zu arbeiten. In diesem Abschnitt wird der aktuelle Stand der Technik vorgestellt.

3.2.1 Anbindung von Raspberry Pi an CAN-Bus

In der folgenden Arbeit soll ein Raspberry Pi an einen CAN-Bus angeschlossen werden. Im Folgenden werden einige verschiede Möglichkeiten vorgestellt.
Die erste Möglichkeit ist die Verwendung eines Microchip MCP251x, der als CAN-Controller dient [SKJ16]. Dieser wird mit dem Microchip MCP2551 ergänzt, der als CAN-Transceiver dient. Diese Kombination ermöglicht es, den Raspberry Pi mit dem CAN-Bus zu verbinden. Der MCP251x wird über die SPI-Schnittstelle (Serial Peripheral Interface) des Raspberry Pi angeschlossen. Dazu wird ein Treiber benötigt, der die Kommunikation zwischen dem Raspberry Pi und dem MCP251x ermöglicht. Ein solcher Treiber ist bereits seit dem 05.05.2015 im Raspberry Pi OS (ehemals Raspbian OS) integriert [SKJ16]. Um diese Möglichkeit zu vereinfachen, gibt es auch verschiedene HATs (Hardware Attached on Top), die solche Microchips bereits integriert haben [PL19]. Ein Beispiel dafür ist das PiCan2 von SK Pang Electronics. Dieses Board wird auf den GPIO-Pins (General Purpose Input/Output) des Raspberry Pi gesteckt. Es erlaubt dem Raspberry Pi mit dem CAN-Bus zu kommunizieren. Das PiCan2 unterstützt eine Geschwindigkeit von bis zu 1Mbit/s.
Es besteht auch die Möglichkeit, einen USB-CAN-Adapter zu verwenden. Damit ist die Anbindung eines beliebigen Computer an den CAN-Bus möglich. Ein Beispiel für einen solchen Adapter ist der UCAN von Fysetc [Fys]. Die Hardware und Firmware des Adapters sind opensource. Die Verbindung zum Raspberry Pi erfolgt über USB-C. Zum CAN-Bus müssen lediglich CAN_H und CAN_L und die Masse verbunden werden. Da CAN-Bus Treiber für Linux, Windows und Mac vorhanden sind, kann der UCAN mit diesen Betriebssystemen verwendet werden.

3.2.2 Übersetzung von CAN-Nachrichten

Viele Software Tools ermöglichen die Arbeit mit dem CAN-Bus. Diese Tools unterscheiden sich in ihrem Funktionsumfang und den Anwendungsbereichen. Einige Programme sind speziell für die Analyse und das Debugging von CAN-Nachrichten ausgelegt, während andere umfassende Entwicklungs- und Simulationsmöglichkeiten bieten. Im Folgenden werden

die wichtigsten Eigenschaften, Einsatzbereiche und Besonderheiten der einzelnen Tools vorgestellt.

- **canCommander**: Dieses Tool ermöglicht das Senden und Empfangen von CAN-Nachrichten. Es bietet eine einfache Benutzeroberfläche und umfangreiche Analysefunktionen. Mit canCommander können CAN-Nachrichten aufgezeichnet, analysiert und bearbeitet werden. Dabei können auch Nachrichten gesendet und injiziert werden. Es wird auch eine schon vorbereitete Platine verkauft, die es ermöglicht, mit dem CAN-Bus zu arbeiten. Dennoch ist canCommander ein open-source Projekt und kann auch auf anderen Plattformen verwendet werden. Diese Plattformen beschränken sich aber auf Mikrocontroller, wie Arduino Uno oder ESP32.[1]

- **cantools**: Diese Python-Bibliothek ermöglicht es, CAN-Nachrichten zu dekodieren. Mit cantools können CAN-Daten in ein Menschenlesbares Format umgewandelt werden. Die Bibliothek bietet eine Vielzahl von Funktionen und unterstützt verschiedene Protokolle und Datenformate. Damit ist cantools ist ein nützliches Werkzeug für die Analyse und Verarbeitung von CAN-Nachrichten und eignet sich für die Entwicklung von Anwendungen, die mit dem CAN-Bus arbeiten.[2]

- **CANoe**: Dieses eigenständige kommerzielle Tool wurde von der Firma Vector Informatik GmbH. entwickelt. Es bietet eine vielzahl an verschiedenen Funktionen. CANoe ermöglicht die Entwicklung, Simulation und Analyse von CAN-Netzwerken. Es unterstützt verschiedene Protokolle und Datenformate.[3]

- **can_decoder**: Das Open-Source-Projekt ermöglicht das Dekodieren von CAN-Nachrichten. Es wurde von der Firma CSS-Electronics entwickelt und ist auf GitHub verfügbar. Allerdings ist das Projekt nicht mehr aktiv und wird nicht mehr weiterentwickelt. Trotzdem kann es ein nützliches Werkzeug für die Analyse und Verarbeitung von CAN-Nachrichten sein.[4]

- **SavvyCAN**: Die C++ Anwendung kann CAN-Nachrichten aufzeichnen, welche zum Reverse-Engineering genutzt werden können. Hiermit können auch einzelne Nachrichten dekodiert werden. SavvyCAN ist ein Open-Source-Projekt und kann auf GitHub gefunden werden. Zusätzlich gibt es Unterstützung auf Linux und Windows.[5]

- **can-utils**: Diese Set an Linux-Tools ermöglichen die Arbeit mit dem CAN-Bus. Damit ist es möglich, CAN-Nachrichten zu senden und zu empfangen. Es ist ein Open-Source-

[1] CAN Commander (besucht am 15.02.2025)
[2] cantools (besucht am 15.02.2025)
[3] CANoe (besucht am 15.02.2025)
[4] can_decoder (besucht am 15.02.2025)
[5] SavvyCAN (besucht am 15.02.2025)

Projekt und kann auf GitHub gefunden werden. Diese Tools bieten eine Grundlage für die Entwicklung von Anwendungen, die mit dem CAN-Bus arbeiten.[6]

Es ist essentiell die Nutzlast der CAN-Bus Nachrichten Informationen zuzuordnen. Das kann in der Form einer DBC-Datei geschehen[Cho+21]. Diese Datei schlüsselt auf, welche Informationen in den verschiedenen CAN-Nachrichten enthalten sind. Da auch sensible Informationen in den Nachrichten enthalten sein können, wird diese Datei in den meisten Fällen nicht öffentlich verfügbar gemacht. Ohne diese Datei können zwar Brute-Force Ansätze verwendet werden, allerdings ist dies sehr aufwendig und nicht zielführend.
Um die vorher genannten Tools zu verwenden, sind DBC-Dateien notwendig, um die zu dekodierenden und kodierenden Nachrichten zu beschreiben. Bei canCommander sind einige DBC-Dateien bereits vorinstalliert. Bei cantools kann die DBC-Datei in das Programm geladen werden.

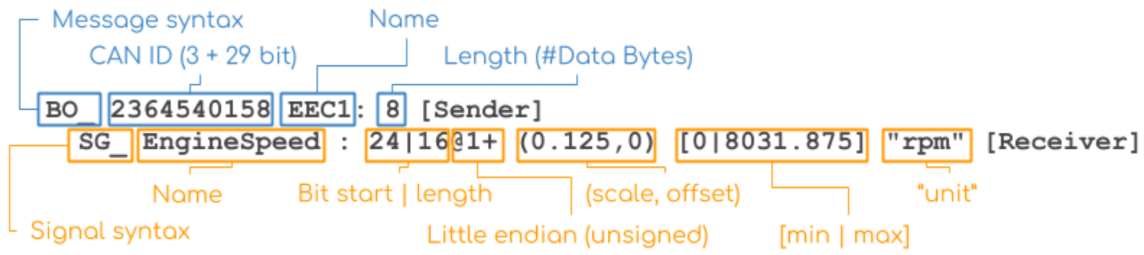


Abbildung 3.1: Auszug aus einer Beispiel DBC-Datei [Ele]

In Abbildung 3.1 ist ein Ausschnitt aus einer DBC-Datei zu sehen. In dieser sind einzelne Nachrichten aufgelistet. Jede Nachricht hat eine ID, eine Länge und Signale. Diese Signale haben eine Länge, einen Offset und einen Faktor. Die Signale sind die eigentlichen Informationen, die in den CAN-Nachrichten kodiert sind. Sie stellen also die Nutzlast der Nachrichten dar. Mit diesen Informationen kann eine Nachricht erstellt werden. Im Anschluss an die Nachrichtentypen werden in der DBC-Datei für bestimmte Signale die möglichen Eingaben definiert. Das hilft bei der richtigen Wahl der Eingabe.

[6] can-utils (besucht am 15.02.2025)

Kapitel 4

Konzept und Systemdesign

In diesem Kapitel

4.1 Aufbau Schiffsysteme 11
4.2 Steuerungslogik des Spiele-Controllers 12
4.3 Integration des Rogue Device . 14

In diesem Kapitel wird das Konzept und das Systemdesign für die Arbeit vorgestellt. Insgesamt soll das Forschungsschiff durch einen Spielecontroller gesteuert werden. Dafür soll ein Rogue Device entwickelt werden, welches in der Lage ist, die Kommunikation des Schiffes zu manipulieren. Das Rogue Device soll dabei unbemerkt in das System integriert werden und auf den CAN-Bus für die Motorsteuerung zugreifen. Darauf sollen manipulierte Steuerbefehle gesendet werden, um den Motor zu steuern. Zusätzlich soll auf den Autopiloten zugegriffen werden, um darüber die Rudersteuerung zu manipulieren.

4.1 Aufbau Schiffsysteme

Ein Schiff besteht aus vielen verschiedenen Systemen, welche verschiedene Aufgaben haben. In diesem Fall wird die Steuerung des Schiffes betrachtet. Dabei sind die wichtigsten Systeme die Gashebel und die Rudersteuerung, welche den Autopiloten enschließt.

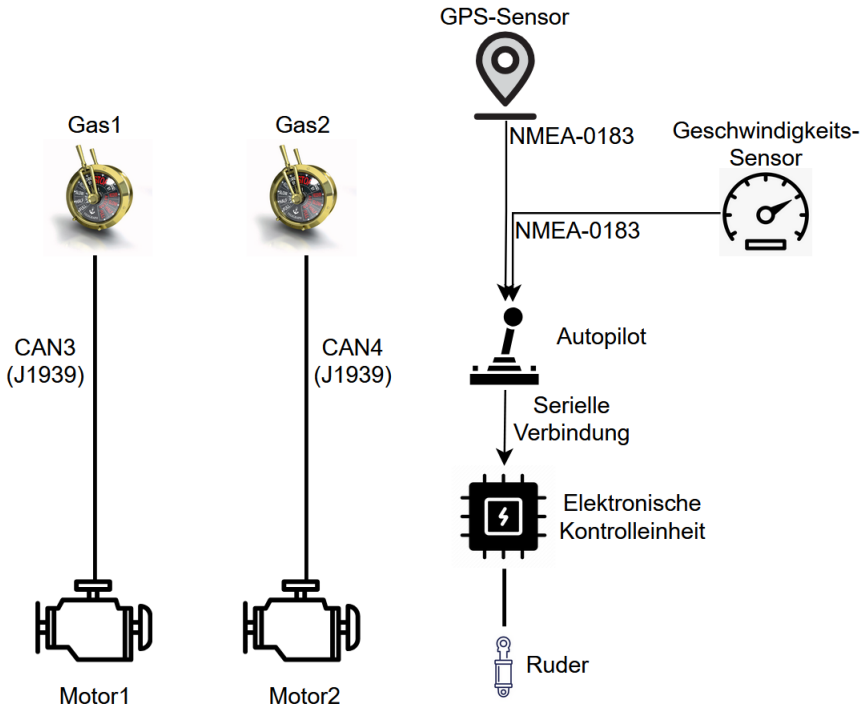


Abbildung 4.1: Vereinfachte Darstellung der Systeme auf der Limanda

In 4.1 sind die genannten Systeme veranschaulicht. Die beiden Gashebel sind jeweils mit einem CAN-Bus verbunden. Über diese Bussen senden die Gashebel Steuerbefehle an die Motoren. Als Higher-Layer-Protokoll wird J1939 genutzt.
Der Autopilot ist ein eigenständiges System, das über eine serielle Schnittstelle mit einer elektronischen Kontrolleinheit(ECU) für die Rudersteuerung verbunden ist. Dieser hat keine Verbindung zu den Gashebeln oder Motoren. Er kann lediglich Signale an die ECU für die Rudersteuerung über eine serielle Verbindung senden. Die Rudersteuerung ist mit dem Ruderstellmotor über eine unbekannte Verbindung angeschlossen. Diese Verbindung ist jedoch nicht relevant für diese Arbeit. Für die Kursberechnung erhält der Autopilot Daten von unter anderem einem GPS-Modul und Geschwindigkeitssensor. Diese Daten werden über NMEA-0183 Nachrichten übertragen.

4.2 Steuerungslogik des Spiele-Controllers

Der benutzte Spiele-Controller ist ein Xbox Series X Controller. Dieser wurde gewählt, da er kabellos ist und kann somit frei bewegt werden. Um die Steuerung des Schiffes zu ermöglichen, müssen die Eingaben des Xbox-Controllers in Steuerbefehle umgewandelt werden. Dies passiert auf dem Raspberry Pi. Der Xbox-Controller wird über Bluetooth mit dem Raspberry Pi verbunden. Dort werden die Eingaben des Controllers ausgelesen und in einem Python-Programm in Steuerbefehle umgewandelt.

Um eine einfache Steuerung zu ermöglichen, wird im folgenden die Tastenbelegung aufgeschlüsselt. Um alle gewünschten Funktionen umzusetzen, werden nicht alle Tasten benötigt.

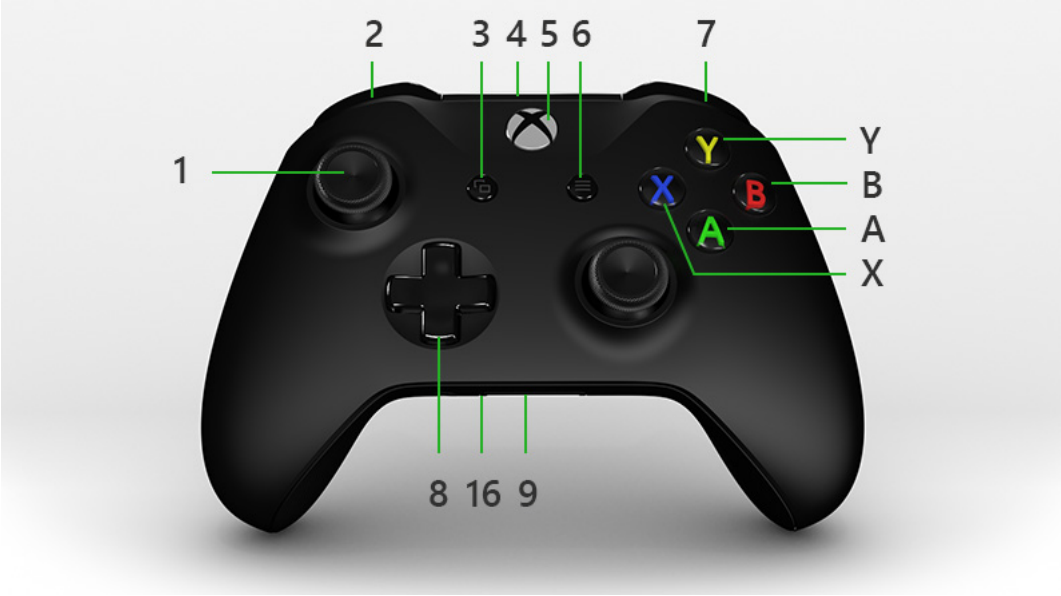


Abbildung 4.2: Vorderseite des Xbox-Controllers [Mic]

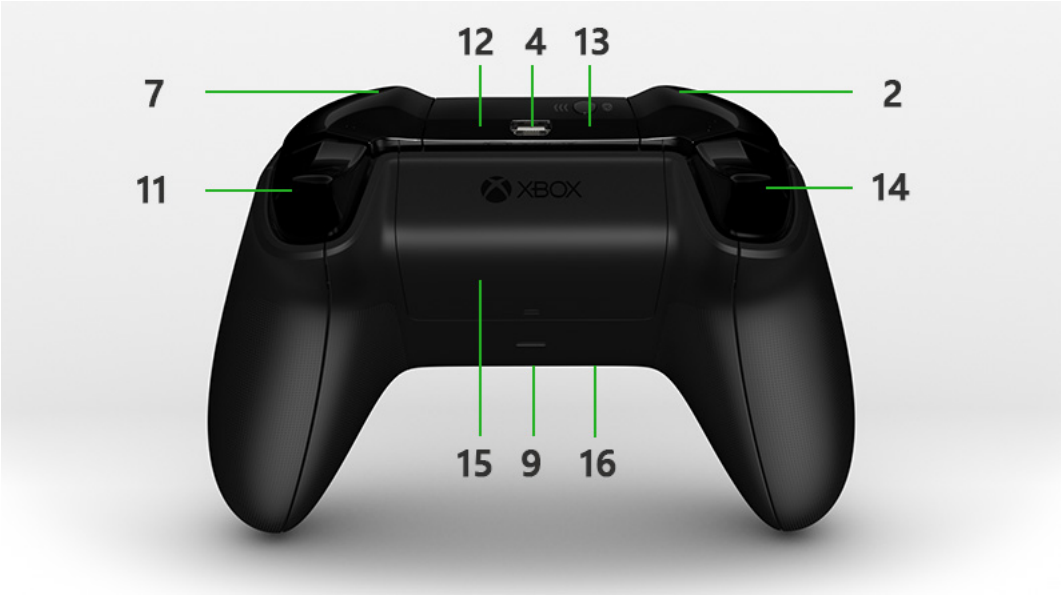


Abbildung 4.3: Rückseite des Xbox-Controllers [Mic]

In den Abbildungen 4.2 und 4.3 sind die Tasten des Xbox-Controllers zu sehen. Die Tastenbelegung ist wie folgt:

Nummerierung der Taste	Funktion
1	Bewegung des Ruders
2	Reduzierung der linken Gashebelposition
7	Reduzierung der rechten Gashebelposition
11	Erhöhung der rechten Gashebelposition
14	Erhöhung der linken Gashebelposition
B + 2	Umschalten des Rückwärtsgangs am linken Motor
B + 7	Umschalten des Rückwärtsgangs am rechten Motor
B + 2 + 7	Umschalten des Rückwärtsgangs an beiden Motoren (basierend auf dem derzeitigen Gang am rechten Motor)

Das Einlegen des Rückwärtsgangs ist durch eine Tastenkombination so gewählt, dass es nicht aus Versehen passieren kann. Mit jeweils der Taste 2 oder 7 wird die Gashebelposition reduziert. Mit der zusätzlichen Betätigung der Taste B wird der Rückwärtsgang eingelegt an dem jeweiligen Motor. Wenn die Tasten 2, 7 und B gleichzeitig betätigt werden, wird der Rückwärtsgang für beide Motoren gleichzeitig umgeschaltet. Damit das Getriebe während des Schaltvorgangs keine Gaseingabe erhält und möglicherweise Schaden nimmt, ist eine Verzögerung von 10 Sekunden eingebaut. Das soll dem Getriebe genug Zeit geben, um den Gang zu wechseln. Allerdings soll es auch eine Möglichkeit geben, beide Getriebe gleichzeitig umzuschalten, um nicht nacheinander die Verzögerung zu haben. Dies wird durch die Tastenkombination B + 2 + 7 realisiert. Dabei wird der Gang des rechten Motors als Referenz genommen.

4.3 Integration des Rogue Device

Damit der Controller die Steuerbefehle an das Schiff senden kann, muss das Rogue Device in das System integriert werden. In diesem Fall ist das Rogue Device der Raspberry Pi. Damit dieser möglichst unbemerkt in das System integriert werden kann, muss der Controller drahtlos verbunden werden. Zusätzlich muss das Rogue Device auch mit Strom versorgt werden. Dafür könnte ein Akku genutzt werden. Dieser müsste jedoch regelmäßig geladen werden. Daher ist es sinnvoller, die Stromversorgung über das Schiff zu realisieren.
Um die Kommunikation von dem Rogue Device zu dem Schiff zu ermöglichen, müssen die einzelnen Systeme angesteuert werden. Um die Gashebelposition zu verändern, wird der Raspberry Pi mit dem CAN-Bus des Schiffes verbunden.
Der Spiele-Controller soll über Bluetooth mit dem Raspberry Pi verbunden. Dieser liest die Eingaben des Controllers aus und übersetzt die in entsprechende Nachrichten. Daraufhin werden die Nachrichten an den CAN-Bus und an die Rudersteuerung gesendet.

4.3.1 Manipulation des Gashebels

Die Manipulation des Gashebels soll durch manipulierte Nachrichten auf dem CAN-Bus realisiert werden. Das Rogue Device wird dazu als externes Gerät an den CAN-Bus angeschlossen. Der Gashebel bleibt weiterhin angeschlossen. Sollte der Gashebel in der normalen Benutzung vom Schiffsführer benutzt werden, wird ein Signal an den CAN-Bus gesendet. Dieses Signal wird dann an die Motoren weitergeleitet. Wenn das Rogue Device keine Nachricht sendet, kann es durchaus sein, dass die der Gashebel die Befehle überschreibt. Um echte Eingaben zu verhindern, muss auf diese Nachricht erkannt und darauf reagiert werden. Dazu muss ein Programm auf dem Rogue Device die Nachrichten auf dem CAN-Bus abhören und analysieren. Wenn eine Nachricht für die Motorsteuerung erkannt wird, kann eine Nachricht des Rogue Device gesendet werden, um die Gashebelposition zu überschreiben. Da die manipulierten Nachrichten dem gleichen Typ entsprechen, wie die echten Nachrichten, ist es wichtig, die eigenen Nachrichten unterscheiden zu können. Das muss geschehen, um eine Endlosschleife von eigenen Nachrichten zu verhindern. Dafür könnte der Nachrichtenzähler überwacht werden. Aus den aufgezeichneten Nachrichten hat sich herausgestellt, dass der Nachrichtenzähler wenige verschiedene Werte hat. Daher sind diese Werte zu den eigenen erstellten Nachrichten verschieden. Dies ist eine einfache Methode, um die eigenen Nachrichten zu erkennen. Eine bessere Möglichkeit ist es aber, eine eigene CAN-ID zu berechnen. Diese kann dann genutzt werden, um die eigenen Nachrichten zu erkennen.
Wie in der Abbildung 2.2 zu sehen, besteht der Header aus einer PGN, einer Quelladresse und einer Priorität. Um nun Nachrichten an den Motor zu senden, kann eine Nachricht des Gashebels abgefangen werden. Die PGN kann für die eigene Nachricht genutzt werden. Die Quelladresse kann auch kopiert werden. Die Priorität sollte möglichst klein gewählt werden, damit die Nachricht des Rogue Devices bevorzugt wird. In der eigenen Nachricht kann dann die gewünschte Gashebelposition gesendet werden.

4.3.2 Manipulation der Rudersteuerung

Um die Rudersteuerung zu manipulieren, muss der Raspberry Pi mit dem Autopiloten verbunden werden. Bei dem Autopiloten handelt es sich um ein Navitron NT888G. Die Eingabeeinheit ist über eine serielle Schnittstelle mit einer elektronischen Kontrolleinheit. Diese Verbindung ist unverschlüsselt, benutzt jedoch ein Proprietäres Protokoll von Navitron. Durch Reverse-Engineering müssen einzelne Befehle herausgefunden werden, welche für die Ruderkontrolle von Bedeutung sind. Dazu müssen die Nachrichten des Autopiloten abgefangen und analysiert werden. Mit den gewonnenen Informationen können eigene Nachrichten erstellt werden oder bestehende Nachrichten in eigener Reihenfolge abgespielt werden. Zusätzlich kann das Ruder auch physikalisch über das Steuerrad bewegt werden. Dieser Weg wird jedoch nicht betrachtet. Ein weiterer Ansatz wäre es, die echten Kursdaten abzufangen und zu manipulieren. Dadurch müsste der Autopilot reagieren und das Ruder

steuern. Die Kursdaten werden nach dem NMEA-0183-Standard gesendet.
Aufgrund des Alters dieses Standards sind Informationen über das Format der Nachrichten verfügbar. Damit können die Kursdaten einfach verstanden und manipuliert werden. Mit dem Austauschen der echten Daten durch manipulierte, wäre ein physikalischer Man-in-the-Middle Angriff realisiert.

4.3.3 Rückmeldung der Eingaben

Es muss eine Art der Rückmeldung für den 'Angreifer' geben, um in etwa die Eingaben im Vergleich zum momentanen Zustand zu sehen. Dabei sollte die Rückmeldung möglichst unauffällig sein. Ein kleiner Bildschirm könnte benutzt werden, dieser kann physisch an den Raspberry Pi angeschlossen werden. Das würde das Verstecken des Rogue Devices erschweren. Ein eigenständiges Display könnte auch über eine Funkttechnologie mit dem Raspberry Pi verbunden werden. Eine andere Möglichkeit stellt eine App auf einem Handy dar. Diese App könnte dann die gewünschten Positionen anzeigen. Dafür muss der Raspberry Pi mit dem Handy verbunden werden. Dies könnte über Bluetooth geschehen. Hier ist zu beachten, dass die Verbindung stabil sein muss und nicht von anderen Geräten gestört wird. Jedoch kann ist hier mit einem größeren Aufwand zu rechnen, da die App erst entwickelt werden muss. Eine weitere Möglichkeit würden Vibrationen im Xbox Controller sein. Diese könnten genutzt werden, wenn die derzeitige Eingabe ein Maximum oder Minimum erreicht hat. Allerdings ist dies nicht so genau wie eine Anzeige auf einem Bildschirm. Es könnten auch nur wenige bestimmte Positionen mittgeteilt werden.
Eine solche Rückmeldung ist wichtig, um die Eingaben zu überprüfen und gegebenenfalls anzupassen. Da die Steuerung auf einem Schiff generell Trägheit aufweist, ist es wichtig, die Eingaben frühzeitig zu korrigieren. Daher können Änderungen in den Eingaben nicht sofort sichtbar sein. Es muss also eine Möglichkeit geben, die Eingaben zu überprüfen.

Kapitel 5

Implementierung

In diesem Kapitel

5.1 Umsetzung des Rogue Device . 17
5.2 Verbindung Rogue Device - Controller 17
5.3 Übersetzung Signale Controller - Schiff 19

In diesem Kapitel wird die Implementierung des Rogue Devices beschrieben. Dabei wird auf die Verbindung des Rogue Devices mit dem Controller und dem Schiff eingegangen. Zudem wird die Umsetzung der Kommunikation zwischen den einzelnen Komponenten beschrieben. Dabei hat die Übersetzung von CAN-Bus Nachrichten eine wichtige Rolle gespielt.

5.1 Umsetzung des Rogue Device

Wie bereits in Kapitel 2.3 beschrieben, wird ein Raspberry Pi 5 als Rogue Device benutzt. Als Betriebssystem wird Raspberry Pi OS benutzt. Der Standard Package Manager ist APT. Unter dessen Benutzung werden Python 3.12 und die benötigten Bibliotheken installiert. Damit sind die Vorraussetzungen für die weitere Implementierung gesetzt.

5.2 Verbindung Rogue Device - Controller

Im ersten Schritt wird der Spiele-Controller mit dem Raspberry Pi verbunden. Der Raspberry Pi wird mit Raspberry Pi OS betrieben. Als Standard-Bluetooth-Treiber wird BlueZ[1] verwendet. Dieser ist bereits vorinstalliert und muss nicht manuell installiert werden. Da es

[1] BlueZ (besucht am 02.01.2025)

sich in diesem Fall um einen Xbox-Controllers handelt, wurde der Treiber xboxdrv[2] installiert. Diese können im APT-Repository gefunden werden. Zusätzlich muss der Enhanced Re-Transmission Mode (ERTM) deaktiviert werden.[3] Dieser Modus ist standardmäßig aktiviert und verhindert die korrekte Verbindung des Controllers, aufgrund eines Kompatibilitätsproblem. Zusätzlich muss sichergestellt werden, dass die Firmware des Spiele-Controllers auf dem aktuellen Stand ist. Um den Xbox-Controller zu verbinden, kann die graphische Benutzeroberfläche benutzt werden.
Als Programmiersprache wird Python 3 benutzt. Die Sprache wurde gewählt, da für alle benötigten Funktionen bereits eine Bibliothek existiert. Die Struktur des Rogue Devices soll wie folgt aussehen:

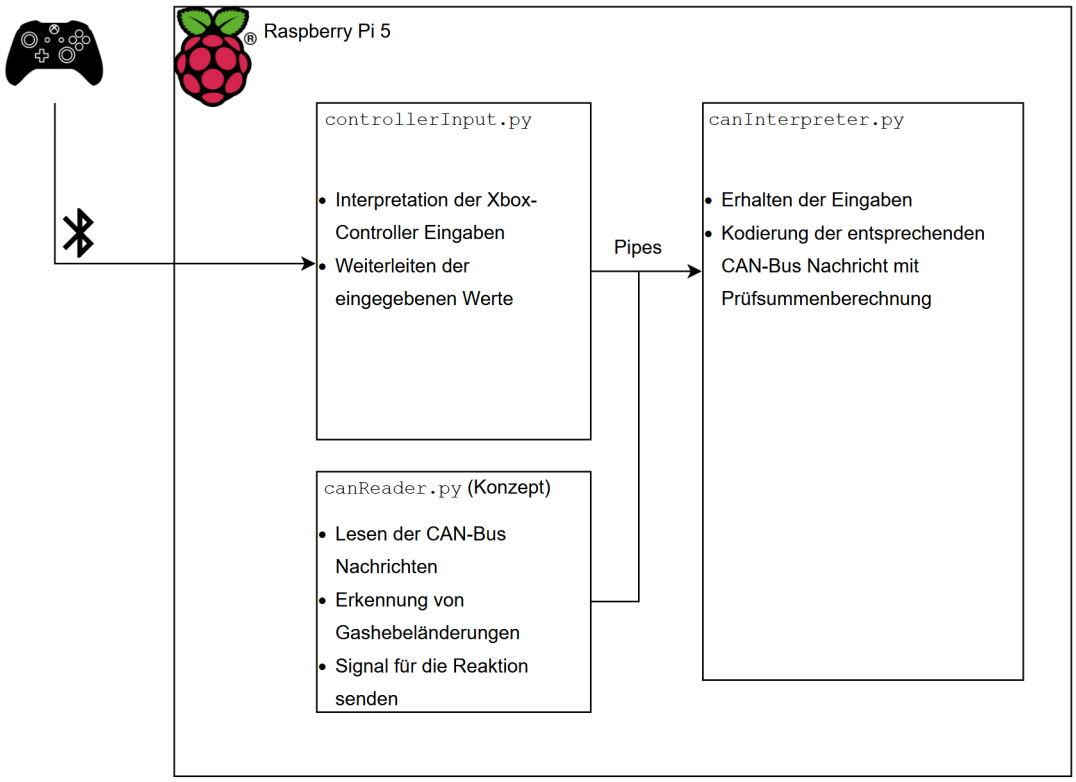


Abbildung 5.1: Programmstruktur auf dem Rogue Device

Für das im Rahmen der Arbeit entwickelte Programm `controllerInput.py` wird die Bibliothek `pygame` benutzt. Eine Einordnung des entwickelten Programms in die Programmstruktur ist in Abbildung 5.1 zu sehen. Die Bibliothek ermöglicht es, Eingaben von einem Controller zu empfangen. Diese Eingaben werden in Variablen gespeichert. Dabei gilt es zu beachten, dass der Gashebel oder die Ruderposition nicht zu schnell verändert werden. Dies könnte zu einem unkontrollierten Verhalten des Schiffes oder zu Schäden führen. Daher werden diese Werte mit Tasten des Controllers eingegeben, welche nicht nur eine binäre Eingabe haben.

[2] xboxdrv (besucht am 02.01.2025)
[3] Debian Wiki (besucht am 22.03.2025)

Diese werden als Achsen bezeichnet. Diese ermöglichen eine stufenlose Eingabe. Bei einer vollständigen Eingabe soll die Gashebelposition nach 20 Sekunden 100% erreichen. So ist sichergestellt, dass die Gashebelposition nicht zu schnell verändert wird. Die Ruderposition soll nach 2 Sekunden in jede Richtung jeweils 100% erreichen. Dies ist ein Kompromiss zwischen Geschwindigkeit und Genauigkeit.

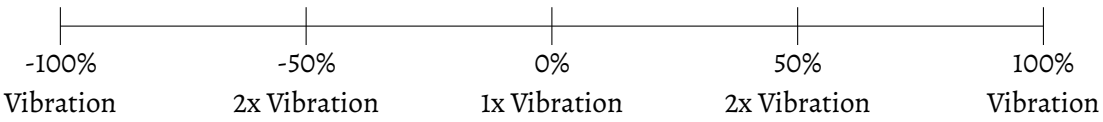
5.3 Übersetzung Signale Controller - Schiff

Wie in Abbildung 5.1 zu sehen ist, erhält `controllerInput.py` die Signale des Xbox-Controllers erhält Variablen in Python interpretiert. Diese Variablen müssen dann in Nachrichten für den CAN-Bus umgewandelt werden. Hierfür wurde ein weiteres Programm entwickelt, welches die Übersetzung die Kodierung der CAN-Bus Nachrichten realisiert. Dieses Programm wird `canInterpreter.py` genannt. Damit die beiden Programme miteinander kommunizieren können, wird Inter-Process-Communication (IPC) benutzt. Als Methode werden hierbei Pipes benutzt. Diese sind einfach zu implementieren und haben eine automatische Synchronisierung zwischen den Prozessen. Dadurch müssen Prozesse nicht aufeinander warten und können weiterarbeiten. Die Synchronisierung wird durch den Puffer der Pipe sichergestellt [VJ15]. Wenn dieser voll ist, wird der schreibende Prozess angehalten, bis der lesende Prozess den Puffer geleert hat. Dies ist ein einfaches und effizientes Verfahren, um die beiden Prozesse zu synchronisieren. Im Rahmen dieser Arbeit dafür ein weiteres Programm entwickelt, welches die Kommunikation zwischen `controllerInput.py` und `canInterpreter.py` ermöglicht. Dieses Programm wird `mainProcess.py` genannt und benutzt die Bibliothek `subprocess`. Dabei werden die beiden Prozesse als Subprozesse gestartet. Die Ausgaben von `controllerInput.py` werden in die Pipe geschrieben und von `canInterpreter.py` als Eingaben gelesen.

5.3.1 Eingabe-Interface

Durch die begrenzte Zeit dieser Arbeit, wurde die Rückmeldung mit Vibrationen im Xbox-Controller implementiert. Dabei wird bei Erreichen des Maximums oder Minimums der Gashebelposition eine Vibration ausgelöst. Das ist eine einfache Methode, um dem Benutzer eine Rückmeldung zu geben. Auch bei der hälfte der Gashebelposition wird eine kurze und leichte Vibration ausgelöst. Das soll ermöglichen, dass es eine ungefähre Vorstellung der Gashebelposition gibt. Bei der Ruderposition wird eine Vibration ausgelöst, wenn die Ruderposition 100% erreicht hat in jeweils beide Richtungen erreicht hat. Bei der Mitte der Ruderposition wird eine kurze und leichte Vibration ausgelöst. Wenn die Ruderposition auf 50% in eine Richtung ist, werden zwei kurze und leichte Vibrationen ausgelöst. Das soll die Bedienung erleichtern. Die Bedienung von Gashebel und Ruderposition soll dabei möglichst getrennt

voneinander vorgenommen werden.
Die Ruderstellung wird durch den linken Stick des Controllers gesteuert. Wenn die Eingabe eine bestimmte Schwelle überschreitet, wird dies wie Folgt durch Vibrationen signalisiert:



Dabei ist die Schwelle so gewählt, dass die Vibrationen nicht zu oft ausgelöst werden. Es ist wichtig zu wissen, dass der Wert basierend auf der Eingabe des Joysticks kontinuierlich berechnet wird. Die Eingabe beträgt -1 bis 1. Dabei ist -1 die maximale Position nach links und 1 die maximale Position nach rechts. Damit die Ruderstellung nicht zu schnell verändert wird, benötigt diese eine volle Eingabe von 1 oder -1 für 2 Sekunden, damit die Ruderstellung 100% erreicht. Entsprechend werden die 100% langsamer erreicht, wenn die Eingabe geringer ist.

5.3.2 CAN-Bus Nachrichtenkodierung

Die Kodierung der CAN-Bus Nachrichten ist der Hauptbestandteil des Programms `canInterpreter.py`, welches im Rahmen dieser Arbeit entwickelt wurde. Das Teilprogramm `canInterpreter.py` erhält Eingaben von `controllerInput.py`. Basierend auf den Eingaben des Xbox-Controllers wird dann die entsprechende Nachricht erstellt. Um eine Nachricht zu kodieren, wird die Bibliothek `cantools` benutzt. Mit dieser Bibliothek können DBC-Dateien gelesen und Nachrichten erstellt werden. Mit einer solchen Datei kann eine bestimmte Nachricht mit ihren Signalen definiert werden. In Kapitel 3.2.2 wurde bereits auf den Aufbau einer DBC-Datei eingegangen.
Bei der Verarbeitung der Eingabe ist es wichtig, dass diese in den richtigen Wertebereich umgewandelt wird. Beispielsweise dürfen die Umdrehungen pro Minute (RPM) nicht unter 500 liegen, wenn der Motor in Betrieb ist. Daher wird die Eingabe auf den Wertebereich von 500 bis 2500 Umdrehungen pro Minute umgewandelt. Dieser Bereich geht aus einer Analyse der aufgezeichneten Nachrichten hervor.
In der benötigten Nachricht ist eine Prüfsumme von 4 Bit notwendig. Jedoch gibt es zwei Methoden der Prüfsummenberechnung nach dem SAE J1939 Standard [Gmb]. Diese Quelle gibt aber auch die richtige Methode für den benötigten Nachrichtentypen an. Bei dem Nachrichtentyp handelt es sich um eine TSC1-Nachricht. Diese Nachricht wird für die Steuerung der Motoren benutzt. Allerdings stimmt die Methode nicht mit den aufgezeichneten Nachrichten überein. In den Nachrichten der Limanda scheint die Prüfsumme nur auf Basis des Nachrichtenzählers bestimmt zu werden. Daher wurde die Prüfsumme nur auf Basis des Nachrichtenzählers bestimmt und so den aufgezeichneten Nachrichten angepasst. Am Ende wird die Prüfsumme der Nachricht hinzugefügt und die Nachricht wird an den CAN-Bus gesendet.
Insgesamt sieht die Kodierung einer TSC1-Nachricht wie folgt aus:

```
1     def encodeThrottleMessage(speed, throttle, canFrameID):
2     torqueHiRes = throttle * 0.125
3     if torqueHiRes > 0.875:
4         torqueHiRes = 0.875
5     throttle = (throttle * 35) - 110
6     # laut aufgezeichneten Werten ist nur bereich von -110 bis -85 bekannt
7     if messageCounter != 4 and messageCounter != 15:
8         checksum = 0
9     elif messageCounter == 4:
10        checksum = 3
11    else:
12        checksum = 15
13    try:
14        throttleInput = gasLeverMessage.encode({
15            'EngOverrideCtrlMode': 3, #3: Speed / Torque Limit Control Mode
16            'EngRequestedSpeedCtrlConditions': 0,
17            # 0: Transient Optimized for driveline disengaged and non-lockup conditions
18            'OverrideCtrlModePriority': 0, # 0: Highest Priority
19            'EngRequestedSpeed_SpeedLimit': speed,
20            'EngRequestedTorque_TorqueLimit': throttle,
21            'TSC1TransRate': 4, # Transmission Rate of 100ms
22            'TSC1CtrlPurpose': 31, # Temporary PowerTrain Control
23            'EngRequestedTorqueHighResolution': torqueHiRes,
24            'MessageCounter': messageCounter,
25            'MessageChecksum': checksum
26        })
27        return can.Message(arbitration_id=canFrameID, data=throttleInput, is_extended_id=False)
```

Wie in dem Code zu sehen ist, werden die Eingaben des Xbox-Controllers vor der Kodierung in den richtigen Wertebereich umgewandelt. Das passiert in den Zeilen 2–5. Danach wird auf Grundlage des Nachrichtenzählers die Prüfsumme bestimmt. Die genauen Werte wurden den aufgezeichneten Nachrichten entnommen. In den Zeilen 14–25 wird die Nachricht erstellt. Dabei werden den Signalen die entsprechenden Werte zugewiesen. Mithilfe von `cantools` wird die Nachricht mit einer bestimmten ID erstellt. Nun liegt die Nachricht als CAN-Nachricht vor und kann an den CAN-Bus gesendet werden.

Zur Überwachung des CAN-Bus wurde das Programm `canReader.py` entwickelt. Es kann die wichtigen Nachrichten für diese Arbeit auf dem CAN-Bus lesen und in Echtzeit dekodieren. Dabei handelt es sich um die Nachrichten für die Motorsteuerung und die Gangschaltung. Dazu wurde wieder die Bibliothek `cantools` benutzt. Auch hier wird mit der gleichen DBC-Datei gearbeitet, da es sich um die gleichen Nachrichten handelt. In den dekodierten Nachrichten können die Signale und deren Werte gesehen werden. In den aufgezeichneten Nachrichten wurde entdeckt, dass Nachrichten für die Motorsteuerung immer eine PGN von 0 haben. Da nur diese Nachrichten mit der PGN 0 gesendet werden, kann nach dieser gefiltert werden. Die gefundenen Nachrichten werden dann dekodiert und spezifisch nach dem Signal `'EngRequestedTorque_TorqueLimit'` durchsucht. Das

Dekodieren wird mit Hilfe von `cantools` realisiert. Zuerst wird die ID der Nachricht einem Nachrichtentyp in der DBC-Datei zugeordnet. Danach wird die Nachricht dekodiert und die Signale und deren Werte ausgegeben. Wenn das gesuchte Signal entdeckt wird, dann muss eine eigene Nachricht gesendet werden, um die wahren Eingaben zu verhindern Dies passiert wieder durch `canInterpreter.py` nach einem über Pipes übermittelten Signal. Um hier zu verhindern, dass eigene Nachrichten vom eigenen System erkannt werden, wird ein eigener Identifier benutzt. Das allgemeine Problem wurde bereits in Kapitel 4.3.1 betrachtet. Um den Identifier zu berechnen, wurde mit einer Portierung von `canboatjs` auf Python gearbeitet [Ben+24]. In diesem Programm kann eine bestehende CAN-ID in Quelle, Priorität und PGN aufgeteilt werden. Diese Werte können dann einzeln verändert werden. Mit diesen Werten kann dann eine neue CAN-ID berechnet werden. Damit werden im eigenen Code die Quelladresse verändert, während die PGN beibehalten wird. Nun können eigene Nachrichten anhand der CAN-ID erkannt werden.
Für die Gangschaltung musste eigene Nachricht in der DBC-Datei definiert werden. Hierfür konnte auf die Bedienungsanleitung des Motors zurückgegriffen werden. In dieser sind die Nachrichten für die Gangschaltung definiert. Die Nachrichten wurden dann in `canInterpreter.py` erstellt und gesendet. Dabei hat sich eine Schwierigkeit ergeben, als die ID nicht als erweiterter J1939-Standard von `cantools` erkannt wurde. Dafür wurde eine Lösung gefunden, indem die ID mit der Hexadezimalen Zahl `80000000` addiert wurde. Mit der Addition dieser Zahl wird in der binären Darstellung das erste Bit von `0` auf `1` gesetzt. Alternativ kann bei der binären Darstellung der ID das erste Bit manuell auf `1` gesetzt werden. Die neu erzeugte ID wird nun ohne Probleme als erweiterte ID erkannt. Diese Lösung konnte nicht in einer offiziellen Dokumentation für den DBC-Standard gefunden werden. In einem Forum wurde berichtet, dass diese Lösung funktioniert [jul21]. Zusätzlich hat diese Lösung im Anwendungsfall dieser Arbeit funktioniert. Laut der gleichen Quelle hängt der Grund dafür damit zusammen, dass ein Bit für die ID in dem Dateiformat zweckentfremdet für die Erkennung einer erweiterten ID benutzt wird. Allerdings kann dies nicht bestätigt werden.
Da in der Nachricht für das Getriebe auch die aktuelle maximale zulässige Last (`'CurrentMaxPermissibleLoad'`) gesendet wird, reicht es nicht aus, diese Nachricht nur bei einer Änderung des Gangs zu senden. Daher wird die Nachricht bei jeder Änderung der Gashebelposition gesendet. Damit hat die aktuelle maximale zulässige Last immer den aktuellen Wert der Eingabe des Xbox-Controllers.

Kapitel 6

Sicherheit

In diesem Kapitel

6.1 Schwachstellen 23
6.2 Schutzmaßnahmen 26

6.1 Schwachstellen

Die durchgeführte Arbeit konnte in diesem Format nur durch einige Schwachstellen in der Kommunikation durchgeführt werden. Um einen solchen Angriff möglichst zu vermeiden, ist es wichtig, dass die Schwachstellen aufgedeckt und behoben werden. Dafür werden in dem folgenden Abschnitt die aufgedeckten Schwachstellen dieser Arbeit aufgeführt und diskutiert.

6.1.1 Schwachstellen am CAN-Bus

Der CAN-Bus ist eines der zentralen Systeme in modernen Fahrzeugen und Schiffen. So auch auf der Limanda. Nach dem CAN-Standard sind die Nachrichten auf dem Bus nicht verschlüsselt. Zusätzlich werden die Nachrichten nur auf ihre Richtigkeit überprüft, jedoch nicht auf ihre Authentizität. Das bedeutet, dass ein Angreifer Nachrichten auf den Bus senden kann, die von anderen Systemen als legitim angesehen werden [Vos08] (Seiten 13-19). Durch das Bus-Netzwerk wird die einfache Integration weiterer Geräte in das System ermöglicht. Ein Angreifer kann so durch physischen Zugriff auf den Bus ein Gerät hinzufügen, welches im schlimmsten Fall die Kontrolle über das gesamte System übernehmen kann. Die einzige Hürde, ein Gerät in ein CAN-Bus Netzwerk zu integrieren,

ist das Beschaffen der richtigen DBC-Datei. Diese Datei wird häufig von den Herstellern der Geräte nicht öffentlich zur Verfügung gestellt. Es sind jedoch einige DBC-Dateien im Internet verfügbar, die von anderen Nutzern erstellt wurden. Durch das Ausprobieren von dem Dekodieren der CAN-Nachrichten mit verschiedenen DBC-Dateien, kann eine passende Datei gefunden werden.
In dieser Arbeit wurde zuerst ausgenutzt, dass die Nachrichten durch physischen Zugriff auf den Bus mitgelesen werden können. Die Nachrichten wurden aufgezeichnet und mit einer passenden DBC-Datei dekodiert. Nachdem die Nachrichten verstanden wurden, konnte eine Nachricht mit dem gleichen Syntax und eigenen Werten erstellt werden. Die aufgezeichnete Nachrichten-ID wurde dabei in der Quelle verändert. Die Priorität ist gleich geblieben, da es bereits die höchste Priorität war. In anderen Fällen ist es möglich, die Priorität einfach zu verändern. Mit der eigenen Nachrichten-ID konnten nun die echten Nachrichten von den eigenen unterschieden werden. Das ist wichtig, um echte Nachrichten mit den eigenen zu überschreiben. Alle Geräte in einem CAN-Bus sind gleichberechtigt[Vos08] (Seite 14). Dadurch wird die Überschreibung der älteren Nachrichten möglich.
Nicht alle vorher beschriebenen Eigenschaften können als Schwachstelle bezeichnet werden. Jedoch können diese gemeinsam mit den Schwachstellen ausgenutzt werden.

6.1.2 Schwachstellen am J1939-Protokoll

Das J1939-Protokoll ist ein Protokoll, welches nicht für die Sicherheit, sondern für die Kommunikation zwischen den Steuergeräten entwickelt wurde. Das Protokoll basiert auf dem Standard des CAN-Bus. Das Protokoll ist nicht verschlüsselt und die Nachrichten sind nicht authentifiziert. Allerdings hat das Protokoll auch spezifische Schwachstellen, welche aber nicht alle in dieser Arbeit ausgenutzt wurden.
Eine Schwachstelle ist, dass Nachrichten mit der niedrigsten ID die höchste Priorität haben. Zusätzlich ist es möglich, eine eigene Priorität so zu setzen, dass die Nachrichten mit der eigenen ID die höchste Priorität haben. Durch das Senden von vielen Nachrichten mit hoher Priorität kann ein Denial-of-Service-Angriff durchgeführt werden. Dabei wird der Bus so stark belastet, dass die echten Nachrichten nicht in einem angemessenen Zeitraum gesendet werden können. Jedes Gerät in einem J1939-Netzwerk benötigt eine eindeutige Adresse und einen einzigartigen Namen [Jun10]. Da ein Gerät vollständige Kontrolle über den eigenen Namen und die eigene Adresse hat, kann ein Angreifer sich als ein anderes Gerät ausgeben. Die Namensübernahme führt zunächst zu einem Konflikt, da zwei Geräte den gleichen Namen haben. Diesen Konflikt gewinnt das Gerät, welches die niedrigere Adresse hat. Daher kann der Name recht einfach übernommen werden. Dadurch kann ein originelles Gerät aus dem Netzwerk ausgeschlossen werden. Mit der übernommenen Adresse kann der Angreifer nun Nachrichten senden, die von anderen Geräten als legitim angesehen werden. Die einzige Sicherheitsmaßnahme, die das J1939-Protokoll bietet, ist eine Zuweisungstabelle für die Adres-

sen und Namen. Allerdings bietet diese bei der Namensübernahme keinen Schutz, wenn der Angreifer auch die Adresse übernimmt. Diese Schwachstelle wurde in dieser Arbeit nicht ausgenutzt, da es zu unvorhersehbaren Folgen hätte führen können. Im schlimmsten Fall könnte ein Gerät dauerhaft aus dem Netzwerk ausgeschlossen werden, was zu einem Ausfall des Systems führen würde. Das spricht für die Schwere der Schwachstelle.
Eine weitere Möglichkeit den Bus zu stören, ist die globale Anfrage nach PGNs. Bei einer solchen Anfrage sollen alle Geräte mit der eigenen PGN antworten. Die Spezifikation rät dazu, maximal 3 Anfragen pro Sekunde für eine Parametergruppe zu senden. Es gibt aber keine feste obere Grenze für die Anzahl der Anfragen. Daher gibt es auch keine Gegenmaßnahmen bei zu vielen Anfragen. Ein Angreifer kann also durch das Senden von vielen Anfragen alle Geräte auf dem Bus dazu bringen, zu antworten. Damit ist ein Distributed-Denial-of-Service-Angriff (DDoS) möglich, weil die Überlastung von mehreren Geräten ausgeht.
[MG18]

6.1.3 Schwachstellen an der Rudersteuerung

Die Rudersteuerung wird nur durch das physikalische Steuerrad oder den Autopiloten gesteuert. Da ein physikalischer Angriff auf das Steuerrad nicht verdeckt durchgeführt werden kann, werden keine außerordentlichen Schutzmaßnahmen benötigt. Bei dem Autopiloten von Navitron handelt es sich um ein Gerät, welches Informationen zu dem Kurs, Geschwindigkeit und Position des Schiffes über den NMEA-0183-Bus erhält. Durch die Eigenschaft, dass NMEA-0183 direkt mit ASCII-Nachrichten arbeitet, ist es möglich mit physikalischem Zugriff auf den Bus Nachrichten zu lesen [Ass] (letzter Zugriff 07.03.2025).
Diese können zur Überwachung oder auch späteren Manipulation der Kommunikation genutzt werden. Mit aufgezeichneten Nachrichten kann ein Angreifer die Kommunikation von den Sensoren zu dem Autopiloten nachvolziehen. Durch das Senden von eigenen Nachrichten könnte es möglich, den Autopiloten indirekt zu steuern. Das könnte durch eine Manipulation der Kursdaten mittels einer Man-in-the-Middle-Attacke erreicht werden. Dies wurde in dieser Arbeit aber nicht getestet und ist daher nur eine Annahme. NMEA-0183 unterstützt eine Baudrate von 4800. Das bedeutet, dass die Nachrichten mit einer Geschwindigkeit von 9600 Bit pro Sekunde gesendet werden. Dadurch können Geräte mit diesem Standard möglicherweise durch eine Denial-of-Service-Attacke gestört werden [Tra+21]. Der Autopilot sendet basierend auf den Kursdaten und den eigenen Einstellungen Nachrichten an die Rudersteuerung. Für diese Nachrichten wurde im Rahmen dieser Arbeit keine Verschlüsselung festgestellt. Aber der Standard dieser Nachrichten ist nicht bekannt. Das stellt zwar keine Sicherheitsmaßnahme dar, aber es ist eine Hürde für einen Angreifer. Wenn jedoch der Standard bekannt ist, kann ein Angreifer die Nachrichten zumindest mitlesen. Eine Authentifizierung wurde auch nicht festgestellt, aber es kann nicht ausgeschlossen werden, dass eine Authentifizierung stattfindet.

6.2 Schutzmaßnahmen

Schutzmaßnahmen sind notwendig, um die Schwachstellen zu schließen und Angriffe zu verhindern. Eine Schwierigkeit bei dem Zugriff auf den CAN-Bus ist, dass dieser physisch zugänglich sein muss. Dies ist auch auf der Limanda der Fall. Wenn man allerdings unbemerkten Zugriff erhält, kann das Rogue Device versteckt in dem System integriert werden. Das stellt die einzige Schutzmaßnahme dar, die in dieser Arbeit umgangen wurde. Jedoch wird der Zugriff nur durch eine Abdeckung zu der Technik des Schiffes verhindert. Mit dem richtigen Werkzeug kann diese Abdeckung einfach entfernt werden. Eine weitere Hürde ist, dass die richtige Baudrate für den CAN-Bus des Schiffes gewählt werden muss. Ohne vorherige Kenntnisse muss dabei ausprobiert werden, welche Baudrate die richtige ist. Das stellt aber keine Schutzmaßnahme dar, sondern einfach eine wichtige Eigenschaft des CAN-Standards. Im CAN-Standard gibt es keine dedizierten Sicherheitsmaßnahmen. Daher müssen diese nachträglich implementiert werden. Eine vergleichsweise einfache Maßnahme ist die Implementierung von Intrusion Detection Systemen (IDS) [GGT16]. Diese Systeme überwachen den Bus und erkennen mögliche Angriffe. Ein Vorteil dabei ist, dass der CAN-Standard in keiner Form für den Anwendungsfall modifiziert werden muss. Es ist auch keine Implementation in jedem Gerät eines Netzwerkes notwendig. Das System kann auf einem zentralen Gerät laufen und den Bus überwachen. Ein IDS kann auf zwei Arten implementiert werden [HKD09]. Zum einen kann ein Signatur/Regel-basiertes System verwendet werden. Hier können bereits bekannte Angriffe erkannt werden. Dies wird mit einer Erkennung von bestimmten Kombinationen oder Sequenzen von Nachrichten erreicht. Um ständig neue Angriffe zu erkennen, müssen die Regeln ständig aktualisiert werden. Für bekannte Angriffe ist dieses System sehr zuverlässig. Zum anderen kann ein Anomalie-basiertes System verwendet werden. Hier wird das normale Verhalten des Systems überwacht. Abweichungen von diesem Verhalten werden als möglicher Angriff erkannt. Dieses System ist weniger wartungsintensiv, da keine ständige Aktualisierung der Regeln notwendig ist. Es ist auch besser für unbekannte Angriffe geeignet, da es keine explizite Regel für den Angriff geben muss. Allerdings gibt es auch mehr Fehlalarme, die in einigen Anwendungen nicht toleriert werden können.
Mit der Nutzung von J1939-Netzwerken ist keine höhere Sicherheit geschaffen. Ein großes Problem dabei ist, dass eine kryptographische Authentifizierung fehlt. Das bedeutet, dass Nachrichten nicht auf ihre Authentizität überprüft werden. Daher ist es möglich, dass ein Angreifer Nachrichten sendet, die von anderen Geräten als legitim angesehen werden. Eine einheitliche Lösung für dieses Problem ist schwer zu finden, da in einem solchen Netzwerk verschiedene Geräte von verschiedenen Herstellern zusammenarbeiten. Laut [MG18] ist eine Lösung für dieses Problem die Implementierung einer Public Key Infrastruktur (PKI). Hier werden die Geräte mit einem Zertifikat ausgestattet, welches von einer zentralen Stelle signiert wird, beispielsweise dem Fahrzeughersteller. Mittels dieses Zertifikats kann ein Gerät seine Identität beweisen und auch die Identität anderer Geräte überprüfen. Die Nutzung von PKI in J1939-Netzwerken ist nicht ungewöhnlich. Es gibt bereits einige Implementierungen,

die PKI nutzen. Vorallem in der Automobilindustrie ist die Nutzung von PKI teilweise etabliert. Außerdem ist es notwendig, eine PKI in Netwerken zu benutzen, in denen die Geräte nicht vom selben Hersteller sind. Desweiteren gibt es viele Open-Source-Bibliotheken, die PKI unterstützen.
Für Rudersteuerung sendet der nur der Autopilot Nachrichten an die Steuerungseinheit. Es ist aber nicht davon auszugehen, dass diese beiden Geräte miteinander verschlüsselt kommunizieren. Diese Geräte stammen auch nicht von dem selben Hersteller. Daher ist eine symmetrische Verschlüsselung zu aufwendig für den praktischen Einsatz. Bei dieser Verschlüsselung müssten die Geräte einen gemeinsamen Schlüssel aushandeln. Das kann nicht immer auf einem sicheren Weg gewährleistet werden. Auch wenn in diesem Fall nur zwei Geräte an der Kommunikation beteiligt sind, kann wieder eine PKI verwendet werden.

Kapitel 7

Abschließende Betrachtung

In diesem Kapitel

7.1 Ergebnis 28
7.2 Ausblick 30

7.1 Ergebnis

In dieser Arbeit sollte die Machbarkeit von Angriffen auf das Steuergerät eines Schiffes untersucht werden. Der Fokus lag dabei auf dem CAN-Bus für die Motorsteuerung und einer seriellen Verbindung für die Rudersteuerung. Um Schwachstellen zu veranschaulichen, sollte der Motor sowie das Ruder mittels eines Spiele-Controllers gesteuert werden. Im ersten Schritt wurde ein Konzept für die Steuerungslogik entwickelt, wie in 4.2 zu sehen. Damit die physichen Eingaben genutzt werden konnten, mussten diese in logische Eingabewerte für ein Programm umgewandelt werden. Das wurde in dem Programm `controllerInput.py` umgesetzt (5.3). Damit auf eigenen Systemen CAN-Nachrichten getestet werden konnten, wurde ein CAN-Bus aufgebaut und Testnachrichten gesendet. Im nächsten Schritt wurden aufgezeichnete CAN-Bus Nachrichten mithilfe einer DBC-Datei dekodiert und anschließend analysiert. Dabei wurden auch Nachrichten für die Rudersteuerung aufgezeichnet, diese konnten jedoch nicht dekodiert werden. Durch die Analyse der Nachrichten, konnten einzelne wichtige Nachrichten entdeckt werden. Durch die weitergehende Analyse der entsprechenden DBC-Datei zu den bestimmten Nachrichten, konnten diese auch verstanden werden (5.3.2). Damit konnte ein Programm entwickelt werden, welches die Eingaben des Spiele-Controllers erhält und in CAN-Bus Nachrichten umwandelt. Dazu musste auch eine entsprechende Prüfsumme berechnet werden, um die Nachrichten zu validieren. Das gleiche Vorgehen wurde auch für Nachrichten der Gangschaltung durchgeführt. Ein weiterer wichtiger

Teil war die Echtzeitdekodierung der CAN-Bus Nachrichten, damit auf Eingaben des Gashebels reagiert werden konnte. Dazu wurde das Programm `canReader.py` entwickelt. Die Rudersteuerung wurde nicht implementiert, da die Nachrichten nicht im Zeitrahmen dekodiert werden konnten.
Im ersten Test wurden die Nachrichten auf dem CAN-Bus des Schiffes im normalen Betrieb mit dem Programm `canReader.py` dekodiert. Dabei hat die Echtzeitdekodierung der Nachrichten funktioniert. Es konnten die wichtigen Nachrichten für die Motorsteuerung erkannt werden. Allerdings konnten nicht alle Nachrichten dekodiert. Zusätzlich wurden keine Nachrichten für die Gangschaltung dekodiert. Dabei ist wichtig zu erwähnen, dass diese Nachrichten nicht in der Standard DBC-Datei enthalten waren. Durch das Handbuch des Motors konnte eine Nachricht nachgestellt werden. Eine solche Nachricht wurde aber nicht in der Kommunikation des Schiffes gefunden. Im zweiten Test wurden manipulierte Nachrichten auf den CAN-Bus gesendet. In der Vorbereitung wurde die Gangschaltung von dem Kapitän in den neutralen Gang geschaltet. In diesem Zustand wurden Nachrichten mit einer Zieldrehzahl von 650 Umdrehungen pro Minute gesendet. Dabei ist die Drehzahl zügig angestiegen, aber über 650 Umdrehungen pro Minute hinaus. Bei einer Drehzahl von 1500 Umdrehungen pro Minute wurde die Nachrichtenübertragung gestoppt, um mögliche Schäden am Motor zu vermeiden. Dabei ist zu erwähnen, dass keine Fehlermeldung im Steuergerät aufgetreten ist. Das Verhalten des Motorsteuergeräts war nicht erwartbar nach den Nachrichten aus dem J1939-Standard. Daher konnte auch nicht die Reaktion auf Gashebelbewegungen getestet werden.

7.1.1 Wie kann die Manipulation von Steuergeräten erschwert werden?

Wie in dem Kapitel 6.2 genauer beschrieben, gibt es verschiedene Möglichkeiten, die Manipulation von Steuergeräten zu erschweren. Die einfachste Methode ist dabei den physischen Zugang zu den Steuergeräten zu erschweren und möglicherweise für Unbefugte dauerhaft zu unterbinden. Dazu kann die Verkabelung in einem geschlossenen Gehäuse untergebracht werden. Bei einem CAN-Bus ist eine weitere Maßnahme die Verwendung von Intrusion Detection Systemen (IDS). Diese können den CAN-Bus überwachen und bei ungewöhnlichen Nachrichten Alarm schlagen. Diese arbeiten entweder Regelbasiert oder erkennen Anomalien in den Nachrichten.
Eine Verschlüsselung von Nachrichten ist in der Theorie auch möglich, aber in der Praxis nicht umsetzbar. Das ist darauf zurückzuführen, dass die verschiedenen Geräte von verschiedenen Herstellern stammen. Daher ist der Schlüsselaustausch nur schwer möglich. Eine weitere Möglichkeit ist die Verwendung von digitalen Signaturen. Dies findet bei der Nutzung von einer Public Key Infrastructure (PKI) statt. Hier kann ein Gerät mit einem Zertifikat seine Identität beweisen. Eine solche PKI zu benutzen ist eine gerechtfertigte Maßnahme, wenn die Sicherheit der Kommunikation gewährleistet werden muss. In der

Automobilindustrie ist das bereits weit verbreitet.

7.1.2 Relevanz für andere Schiffe

Der Mehrwert dieser Arbeit fokussiert sich nicht nur auf die Limanda, sondern auch auf andere Schiffe. Aufgrund der weiten Verbreitung des CAN-Bus, ist davon auszugehen, dass auch viele andere Schiffe diesen nutzen. Der CAN-Bus der Limanda hat keine speziellen Sicherheitsmaßnahmen, die über den Standard hinausgehen. Die Limanda wurde erst 2021 in Betrieb genommen und ist daher auf einem aktuellen Stand der Technik [Sch21] (letzter Zugriff 14.03.2025). Es kann davon ausgegangen werden, dass auch andere Schiffe, die in den letzten Jahren in Betrieb genommen wurden, auf einem ähnlichen Stand der Technik sind. Daher ist es wahrscheinlich, dass der CAN-Bus auch auf anderen Schiffen keine speziellen Sicherheitsmaßnahmen hat.
In dem Fall der Limanda wurde nur der Motor mit einem CAN-Bus angesteuert. Es besteht jedoch die Möglichkeit, dass auch andere Systeme, wie die Rudersteuerung, mit einem CAN-Bus angesteuert werden. Auf größeren Schiffen kann das eher der Fall sein. Eine mechanische Rudersteuerung ist auf größeren Schiffen nicht mehr üblich, da die Brücke zu weit von dem Ruder entfernt ist. Daher kann die Rudersteuerung elektronisch sein.
In dieser Arbeit wurde gezeigt, dass NMEA-0183 Nachrichten einfach mitgelesen werden können. Dadurch können mindestens alle Informationen, die über NMEA-0183 übertragen werden, abgefangen werden. Es gibt bereits einen Nachfolger von NMEA-0183, NMEA-2000. der Standard ist auch auf der Limanda implementiert, allerdings war dieser nicht relevant für diese Arbeit. Deswegen kann keine Einschätzung über die Sicherheit von NMEA-2000 gemacht werden.
In dieser Arbeit konnten die Motoren nur manipuliert werden, wenn diese schon in Betrieb waren. Zusätzlich konnte Manipulation durch das ausschalten der Motoren verhindert werden. Im Rahmen dieser Arbeit kann keine Aussage über die Möglichkeit der Manipulation von Motoren gemacht werden, die nicht in Betrieb sind. Es würde eine größere Gefahr darstellen, wenn die Motoren von einem Angreifer ein- und ausgeschaltet werden könnten.

7.2 Ausblick

In dieser Arbeit wurde gezeigt, dass die Steuerung eines Schiffsmotor durch Manipulation von CAN-Bus Nachrichten möglich ist. Allerdings konnte die Manipulation nicht auf vorhersehbare Weise durchgeführt werden. Die Motordrehzahl konnte nicht vorhersehbar manipuliert werden. Eine Lösung dafür könnte in anderen Nachrichten von dem CAN-Bus liegen, welche nicht dekodiert werden konnten. Daher könnte die Dekodierung von

weiteren Nachrichten untersucht werden. In den dekodierten Nachrichten konnten keine Informationen über die Gangschaltung gefunden werden. Daher ist anzunehmen, dass diese Nachrichten auch nicht dekodiert werden konnten. Mit einer vollständigen Dekodierung der aller Nachrichten, ist es wahrscheinlich, die Gangschaltung zu manipulieren.
Zusätzlich konnte nicht gezeigt werden, dass die Rudersteuerung manipuliert werden kann. Dies ist der begrenzten Zeit und den fehlenden Informationen über die Rudersteuerung geschuldet. Es ist daher empfehlenswert, die Rudersteuerung in einer weiteren Arbeit zu untersuchen. Dazu sollten die Nachrichten für die Rudersteuerung dekodiert werden.
In dieser Arbeit wurden Sicherheitsmaßnahmen vorgeschlagen, die nicht implementiert wurden. Diese müssen in einer weiteren Arbeit auf Effektivität geprüft werden. Dazu können die Sicherheitsmaßnahmen implementiert und getestet werden. Dabei kann auch die Reaktion auf Angriffe getestet werden.

Kapitel VIII

Anhang

In diesem Kapitel

VIII.1 Quellcode 32
VIII.2 Schaltpläne 32

VIII.1 Quellcode

VIII.2 Schaltpläne

Abbildungen

2.1 Aufbau eines CAN-Frames [Vos08] 4
2.2 Header einer J1939-Nachricht auf dem CAN-Bus [Gmb] 5

3.1 Auszug aus einer Beispiel DBC-Datei [Ele] 10

4.1 Vereinfachte Darstellung der Systeme auf der Limanda 12
4.2 Vorderseite des Xbox-Controllers [Mic] 13
4.3 Rückseite des Xbox-Controllers [Mic] 13

5.1 Programmstruktur auf dem Rogue Device 18

Literatur

[App] NHL Stenden University of Applied Sciences. *Maritime Cyber Attack Database*. URL: https://maritimecybersecurity.nl/listview (besucht am 18. 03. 2025) (siehe S. 1).

[Ass] NMEA (National Marine Electronics Association). *NMEA 0183*. URL: https://www.nmea.org/nmea-0183.html (besucht am 07. 03. 2025) (siehe S. 6, 25).

[Ben+24] Scott Bender, Kees Verruijt, Jouni Hartikainen und Teppo Kurki. *Canboatjs*. Aug. 2024. URL: https://github.com/canboat/canboatjs/releases/tag/v2.10.0 (siehe S. 22).

[Cho+21] Wonsuk Choi, Seyoung Lee, Kyungho Joo, Hyo Jin Jo und Dong Hoon Lee. „An Enhanced Method for Reverse Engineering CAN Data Payload“. In: *IEEE Transactions on Vehicular Technology* 70.4 (2021), S. 3371–3381. DOI: 10.1109/TVT.2021.3063261 (siehe S. 10).

[Ele] CSS Electronics. *CAN DBC File Explained - A Simple Intro*. URL: https://www.csselectronics.com/pages/can-dbc-file-database-intro (besucht am 18. 02. 2025) (siehe S. 10).

[Fys] Fysetc. *UCAN*. URL: https://github.com/FYSETC/UCAN/ (besucht am 17. 02. 2025) (siehe S. 8).

[GGT16] Mabrouka Gmiden, Mohamed Hedi Gmiden und Hafedh Trabelsi. „An intrusion detection method for securing in-vehicle CAN bus“. In: *2016 17th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)*. 2016, S. 176–180. DOI: 10.1109/STA.2016.7952095 (siehe S. 26).

[Gmb] Vector Informatik GmbH. *SAE1939*. URL: https://www.vector.com/de/de/know-how/protokolle/sae-j1939/# (besucht am 18. 02. 2025) (siehe S. 5, 20).

[HKD09] Tobias Hoppe, Stefan Kiltz und Jana Dittmann. „Applying intrusion detection to automotive it-early insights and remaining challenges“. In: *Journal of Information Assurance and Security (JIAS)* 4.6 (2009), S. 226–235 (siehe S. 26).

[jul21] juleq. *cantools Issue*. März 2021. URL: https://github.com/cantools/cantools/issues/301#issuecomment-808726550 (besucht am 04. 03. 2025) (siehe S. 22).

[Jun10] Markus Junger. *Introduction to J1939*. Techn. Ber. Vector Informatik GmbH., 2010. URL: https://www.vector.com/int/en/download/introduction-to-j1939/ (siehe S. 24).

[MG18] Pal-Stefan Murvay und Bogdan Groza. „Security Shortcomings and Countermeasures for the SAE J1939 Commercial Vehicle Bus Protocol". In: *IEEE Transactions on Vehicular Technology* 67.5 (2018), S. 4325–4339. DOI: 10.1109/TVT.2018.2795384 (siehe S. 4, 5, 25, 26).

[Mic] Microsoft. *Xbox One Wireless Controller*. URL: https://support.xbox.com/de-DE/help/hardware-network/controller/xbox-one-wireless-controller (besucht am 24. 01. 2025) (siehe S. 13).

[PL19] Sudarshan Pant und Sangdon Lee. „Design and Implementation of a CAN Data Analysis Test Bench based on Raspberry Pi". In: *Journal of Multimedia Information System* 6.4 (Dez. 2019), S. 239–244. ISSN: 2383-7632. DOI: 10.33851/jmis.2019.6.4.239 (siehe S. 8).

[RJ16] George Reilly und John Jorgensen. „Classification considerations for cyber safety and security in the smart ship era". In: *Proceedings of the International Smart Ships Technology Conference*. 2016, S. 26–27 (siehe S. 1).

[Sca+08] Karen Scarfone, Murugiah Souppaya, Amanda Cody und Angela Orebaugh. „Technical guide to information security testing and assessment". In: *NIST Special Publication* 800.115 (2008), S. 2–25. URL: https://git.hsbp.org/six/PTD/raw/commit/9e642f4aa921f2bc83b012326ce0ed23c0af4e53/methodology_documents/NIST_SP800-115.pdf (siehe S. 7).

[Sch21] Hendrik Schubert. *Limanda*. Juli 2021. URL: https://www.uni-rostock.de/universitaet/kommunikation-und-aktuelles/medieninformationen/detailansicht/n/forschungskatamaran-der-universitaet-rostock-von-bildungsministerin-bettina-martin-auf-den-namen-limanda-getauft/ (siehe S. 30).

[SKJ16] A. A. Salunkhe, Pravin P Kamble und Rohit Jadhav. „Design and implementation of CAN bus protocol for monitoring vehicle parameters". In: *2016 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*. IEEE, Mai 2016, S. 301–304. DOI: 10.1109/rteict.2016.7807831 (siehe S. 8).

[Tra+21] Ky Tran, Sid Keene, Erik Fretheim und Michail Tsikerdekis. „Marine network protocols and security risks". In: *Journal of Cybersecurity and Privacy* 1.2 (2021), S. 239–251. URL: https://www.mdpi.com/2624-800X/1/2/13 (siehe S. 25).

[VJ15] Aditya Venkataraman und Kishore Kumar Jagadeesha. „Evaluation of interprocess communication mechanisms". In: *Architecture* 86.64 (2015). URL: http://pages.cs.wisc.edu/~adityav/Evaluation_of_Inter_Process_Communication_Mechanisms.pdf (siehe S. 19).

[Vos08] Wilfried Voss. *A comprehensible guide to controller area network*. Copperhill Media, 2008. ISBN: 9780976511601 (siehe S. 3, 4, 23, 24).

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen sind, sind als solche kenntlich gemacht.

Die Arbeit ist noch nicht veröffentlicht und ist in ähnlicher oder gleicher Weise noch nicht als Prüfungsleistung zur Anerkennung oder Bewertung vorgelegt worden.

Rostock, 23. März 2025