Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Michael Przybilla

Telemetrie eingebetteter Systeme mit softrealtime Anforderungen am Beispiel eines autonomen Modellfahrzeugs

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Michael Przybilla

Telemetrie eingebetteter Systeme mit softrealtime Anforderungen am Beispiel eines autonomen Modellfahrzeugs

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.-Ing. Franz Korf
Zweitgutachter : Prof. Dr. rer. nat. Stephan Pareigis

Abgegeben am 26. Februar 2010

Michael Przybilla

Thema der Bachelorarbeit

Telemetrie eingebetteter Systeme mit softrealtime Anforderungen am Beispiel eines autonomen Modellfahrzeugs

Stichworte

Telemetrie, Wireless-LAN, Soft-Realtime, Qt

Kurzzusammenfassung

Gegenstand dieser Arbeit ist der Entwurf und die Realisierung eines Telemetriesystems mit Soft-Realtime-Anforderungen für ein autonomes Modellfahrzeug. Dafür wird eine Anwendung entwickelt, die auf der Seite des Fahrzeuges die erforderlichen Daten sammelt, bearbeitet und an eine Empfangsstation versendet. Hierbei werden die Daten über eine Wireless-LAN Verbindung transportiert. Als Gegenstück wird eine Anwendung auf Basis des Qt-Frameworks entwickelt, welche die empfangenen Daten visualisieren, speichern und erneut abspielen kann.

Michael Przybilla

Title of the paper

Telemetry of embedded systems with softrealtime demands at the example of an autonomous model vehicle

Keywords

Telemetry, Wireless-Lan, Soft-Realtime, Qt

Abstract

Subject of this thesis is the design and implementation of a telemetry system with soft real-time requirements for an autonomous model vehicle. For this purpose an application is developed to gather the necessary data of the vehicle, to process the data and to send the data to a remote station. Therefor the data is send via a wireless-lan connection. As remote station an application based on the Qt framework is developed to visualize, store and replay the received data.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1. Einleitung

In diesem Kapitel wird ein kurzer Überblick über die Zielsetzung und Gliederung dieser Arbeit gegeben.

1.1. Motivation

Im Rahmen des CaroloCups [Maurer (2010)] treten autonom fahrende Modellfahrzeuge in unterschiedlichen Disziplinen, wie z.B. Einparken oder Befahren eines Parcours mit Hindernissen, gegeneinander an. Auch die HAW Hamburg ist mit einem eigenen Team in diesem Wettbewerb vertreten.

In diesem Rahmen wird, unter anderem zur Steuerung dieser Fahrzeuge, Software entwickelt. Diese Software muss in entsprechendem Umfang getestet werden, um eine korrekte Funktionalität zu gewährleisten. Damit das Testen effektiv und erfolgreich durchgeführt werden kann, ist es nötig mit den Sinnen des Fahrzeuges zu sehen, um daraus Rückschlüsse auf die Interpretation dieser Daten zu treffen. Für die korrekte Interpretation dieser Daten ist es notwendig, dass alle in zeitlicher Relation stehenden Daten auch in diesem zeitlichen Zusammenhang sichtbar gemacht werden.

1.2. Zielsetzung

Ziel dieser Arbeit ist es, eine Telemetrie-Anwendung [Carden u. a. (2002)] zu entwickeln. Mit Hilfe dieser Telemetrie-Anwendung sollen ausgewählte Daten per Wireless-LAN vom Fahrzeug auf einem beliebigen PC oder Laptop übertragen werden können. Dort soll es möglich sein die empfangenen Daten zu visualisieren, bei Bedarf zu speichern und zu einem beliebigen späteren Zeitpunkt wiederzugeben. Im Zuge dieser Entwicklung sollen einige unterschiedliche Implementierungsansätze diskutiert und analysiert werden.

1.3. Gliederung

Die Arbeit teilt sich in folgende Abschnitte auf:

- **Kapitel 2 - Grundlagen:**

  Erläuterung verschiedener Begriffe um das Verständnis der Arbeit zu erleichtern.

- **Kapitel 3 - Analyse und Design:**

  Ermittlung des Ist-Zustands des vorhandenen Systems und Analyse der Anforderungen an das Telemetriesystem unter Abwägung verschiedener Implementierungsmöglichkeiten, sowie Erläuterung verschiedener Designmöglichkeiten.

- **Kapitel 4 - Realisierung:**

  Beschreibung der Realisierung des Telemetriesystems unter Berücksichtigung der, im vorherigen Kapitel, beschriebenen Designmöglichkeiten.

- **Kapitel 5 - Fazit:**

  Zusammenfassung und Beurteilung der Ergebnisse.

2. Grundlagen

Zweck dieses Kapitels ist es, eine Basis zu schaffen mit der die Entscheidungen, die im Verlauf dieser Arbeit getroffen werden, leichter nachvollziehbar sind.
In diesem Sinne werden zuerst die Kenntnisse im Bereich Netzwerktechnik aufgefrischt. Im späteren Verlauf wird dann eine Einführung in den Bereich der Echtzeitsysteme gegeben.

2.1. Netzwerk

Zweck dieses Kapitels ist es, mögliche Wissenslücken im Bereich Netzwerk-Protokolle und Wireless-LAN[1] zu schließen, um in späteren Kapiteln aufgezeigte Designmöglichkeiten und getroffene Entscheidungen besser verstehen zu können. Zuerst werden die Netzwerk-Protokolle User Datagram Protocol (UDP) und Transmission Control Protocol (TCP) beschrieben und erläutert. Im darauf folgenden Abschnitt wird eine kurze Einführung zu den verschiedenen Wireless-LAN IEEE 802.11x Standards gegeben.

2.1.1. Netzwerkprotokolle

Anfang der achtziger Jahre wurde von der Internationalen Organisation für Normung (ISO) das Open Systems Interconnection (OSI) - Referenzmodell als Designgrundlage für Kommunikationsprotokolle verabschiedet. Dieses Modell besteht aus sieben Schichten, welche sich iterativ von der Hardwareebene zu den Software-Applikationen bewegen [Zimmermann (1980)].

Die vierte Schicht ist die Transportschicht. Diese Schicht wird oft als die wichtigste Schicht des OSI-Modells bezeichnet. Ihre Aufgabe ist es, unabhängig von den physischen Gegebenheiten, zuverlässigen, kosteneffektiven Datentransfer vom Sender zum Empfänger sicher zu stellen [Tanenbaum (2003)]. Sie stellt somit ein Bindeglied zwischen den Hardware- und den Softwareebenen dar. Außerdem ist sie, von unten gesehen, die letzte Schicht, welche

[1] LAN = Local Area Network. Ein lokales, räumlich begrenztes Rechnernetz. LANs umfassen in der Regel mehrere Räume, aber selten mehr als ein Grundstück.

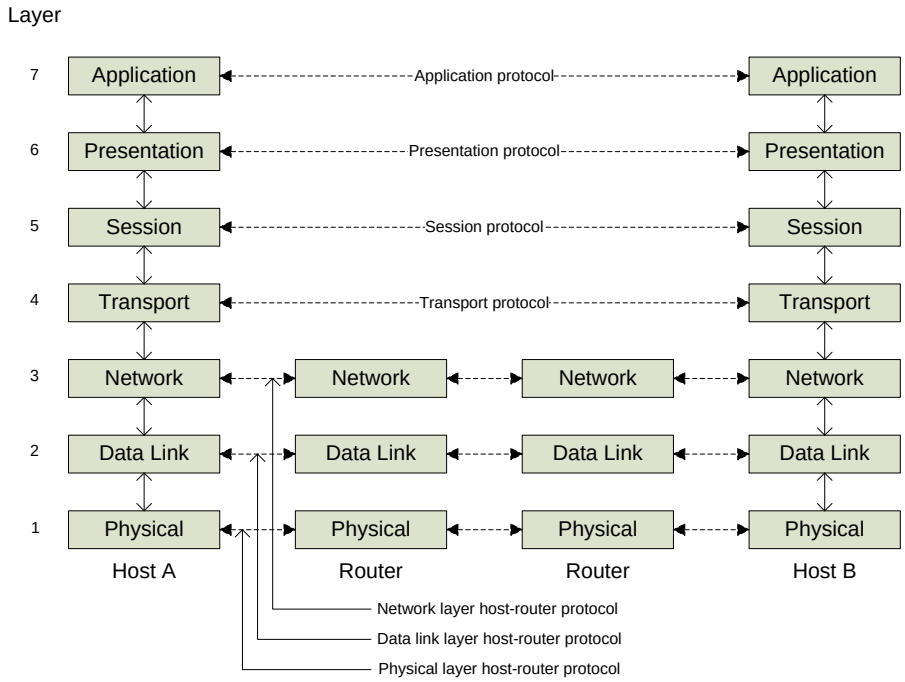


Abbildung 2.1.: OSI Standard Modell

eine vollständige Ende-zu-Ende Kommunikation zur Verfügung stellt. In dieser Schicht gibt es zwei Protokolle die am häufigsten genutzt werden, das User Datagram Protocol und das Transmission Control Protocol. Beide Protokolle mit ihren Vor- und Nachteilen, sowie die sich daraus ergebenden Einsatzgebiete, werden im Folgenden erläutert.

User Datagram Protocol (UDP). UDP ist ein verbindungsloses Protokoll. Das bedeutet, dass es möglich ist, IP-Datagramme zu verpacken und zu versenden ohne vorher eine feste Verbindung aufbauen zu müssen. Dabei werden Segmente versendet, die aus einem nur acht Byte großem Header und den eigentlich zu versendenden Nutzdaten bestehen.

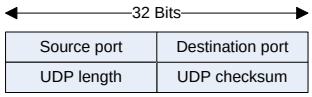


Abbildung 2.2.: UDP-Header

Der Header besteht aus zwei Portadressen, Source und Destination, sowie einer Größenangabe und einer Prüfsumme. Der Hauptunterschied zum IP-Datagram (Schicht 3) sind die

Portangaben. Mit Hilfe der Ports werden die Daten der UDP-Segmente bei Ankunft zu dem Prozess geleitet werden, der an den entsprechenden Port gebunden ist. Ohne die Portangaben wüsste die Transportschicht nichts mit dem Paket anzufangen. Der Source Port dient hauptsächlich zu Antwortzwecken, wenn eine Antwort nötig ist.
Die Größenangabe umfasst das gesamte UDP-Segment, also Nutzdaten und Header. Durch das 16 Bit große Feld ergibt sich somit eine maximale Größe von 64 Kilobyte pro UDP-Segment. Die Prüfsumme ist optional und kann bei Bedarf ausgeschaltet werden.
UDP bietet keine Mechanismen zur Fluss- oder Fehlerkontrolle. Das schließt mit ein, dass Segmente die fehlerhaft empfangen worden sind nicht automatisch erneut angefordert und/oder versendet werden. UDP bietet ein Interface zum IP Protokoll, das dieses um die Möglichkeit erweitert, Daten an mehrere Prozesse gleichzeitig zu senden (Demultiplexing).
UDP eignet sich damit besonders für Netzwerk Anwendungen in denen die Daten jeweils in ein Segment passen und eine zuverlässige Übertragung der Daten nicht notwendig ist, wie z.B. beim Domain Name Service (DNS) Dienst. Dieser dient zur Auflösung von URLs (Uniform Resource Locator), wie z.B Internetadressen wie *www.informatik.haw-hamburg.de* in IP-Adressen. Trifft innerhalb einer gewissen Zeitspanne keine Antwort vom DNS-Server ein an den die Anfrage zur Namensauflösung gestellt wurde, wird einfach eine neue Anfrage ggf. an einen anderen DNS-Server gestellt.

Transmission Control Protocol (TCP). TCP wurde entworfen um zuverlässig einen Bytestrom über ein unzuverlässiges Netzwerk versenden zu können.
Es ist ein verbindungsorientiertes Protokoll. Das bedeutet es wird vor der eigentlichen Datenübertragung eine Ende-zu-Ende Verbindung zwischen dem Socket auf dem Sendersystem und dem Ziel-Socket auf dem Zielsystem hergestellt. Diese wird während der gesamten Übertragung aufrechterhalten und gegebenenfalls neu hergestellt. Da eine TCP-Verbindung voll duplexfähig ist, kann der Sender auch zum Empfänger werden und umgekehrt. Ein Socket kann hierbei ein Endpunkt für mehrere Verbindungen sein, aber eine Verbindung hat immer genau einen End- und einen Startpunkt. TCP unterstützt kein Multi- und kein Broadcasting.
Im Gegensatz zum UDP müssen die Daten nicht in ein Segment passen. Übersteigt die Datenmenge die Größe eines Segments, so teilt die TCP Bibliothek die Daten in entsprechend große Teile, versieht diese mit einer Sequenznummer und versendet diese Teile dann. Auf der Empfängerseite wird der Strom dann anhand der Sequenznummern wieder zusammengesetzt. Dabei müssen die einzelnen Teile nicht zwingend in der Reihenfolge empfangen werden in der sie abgeschickt worden sind. Beim Versand wird ein Timer gestartet. Wird dieses Segment auf der anderen Seite empfangen so wird eine Bestätigungsnachricht (ACK) mit der Sequenznummer des empfangenen Teils geschickt. Kommt keine Bestätigung an bevor der Timer abgelaufen ist, so wird das entsprechende Segment erneut verschickt. Dabei wird immer nur die höchste in die Reihenfolge passende Sequenznummer bestätigt.

Haben wir also zehn Segmente (1-10) und auf Empfängerseite sind 1,2,3,7,9,10 angekommen, so wird nur ein ACK bis Sequenznummer 3 geschickt.

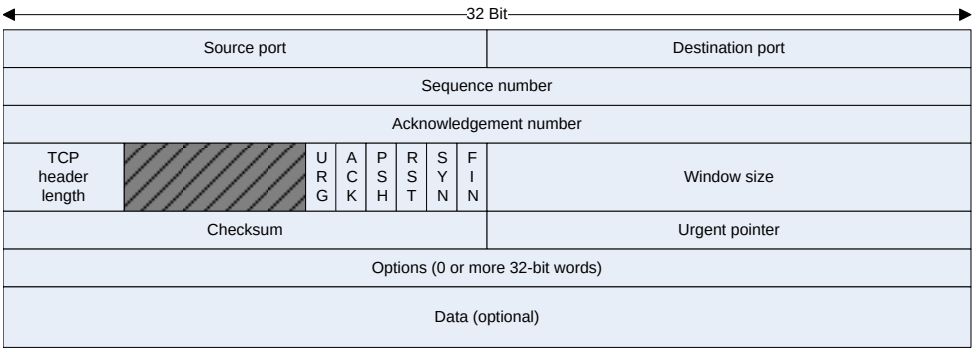


Abbildung 2.3.: TCP-Header

Ein TCP-Segment besteht aus einem fixen 20 Byte Header, optionalen Header-Optionen und optionalen Daten. Die Daten können wie beim UDP-Segment knapp 64 Kilobyte umfassen. In der Praxis werden die Daten aber meist an die MTU des physischen Mediums angepasst, im Falle von Ethernet währen das 1460 Byte. Der Header besteht aus zwei Portadressen, Source und Destination, einer Sequenz- und Acknowledgementnummer, einer Headerlänge, einem ungenutzten Feld, sechs Flags, einer Fenstergröße, einer Prüfsumme und einem Urgent Pointer.

- Die Portadressen enthalten eine 16 Bit Portadresse.

- Die Sequenz- und die Acknowlegdementnummern erfüllen die vorher schon beschriebene Aufgabe. Sie sind jeweils 32 Bit groß, da in einem TCP-Strom jedes Byte nummeriert wird. Dabei gibt die Acknowledgementnummer nicht das zuletzt empfangene, sondern das als nächstes erwartete Byte an.

- Die Headerlänge gibt die Länge des TCP-Headers in 32 Bit Worten an. Dies ist nötig, weil das Header-Optionen Feld eine variable Größe hat.

- Das nicht genutzte, in Abb. 2.3 grau schraffierte, Feld wurde damals beim Design eingebaut, um eventuelle Fehler oder Schwachstellen beseitigen zu können, aber bis

heute nicht gebraucht.

- Das *URG*-Flag ist gesetzt, wenn der Urgent Pointer gesetzt ist. Dieser zeigt innerhalb der aktuellen Sequenznummer auf einen Offset, an dem sich eine Interrupt-Nachricht befindet. Wird z.B. bei einer Remote-Verbindung genutzt, wenn durch ein CTRL-C die Verbindung abgebrochen bzw. die Anwendung geschlossen wird.

- Das *ACK*-Flag gibt an ob es sich um eine gültige Acknowledgement Nummer handelt.

- Das *PSH*-Flag bewirkt, falls gesetzt, dass die Daten sofort bei Empfang an die entsprechende Anwendung weitergeleitet werden. Es wird also nicht gewartet wird bis der Buffer voll ist.

- Das *RST*-Flag initiiert einen Verbindungsneuaufbau. Es wird ausserdem benutzt um ungültige Segmente oder einen Verbindungsaufbau abzulehnen.

- Das *SYN*-Flag wird benutzt um die Punkt-zu-Punkt Verbindung aufzubauen.

- Das *FIN*-Flag dient zum Beenden einer Verbindung. Es signalisiert, dass der Sender keine weiteren Daten zum Versenden hat. Nach Empfang eines FIN-Segments kann der Empfänger trotzdem noch Daten über diese Verbindung empfangen, da auch das FIN-Segment eine Sequenznummer hat und somit in entsprechend korrekter Reihenfolge verarbeitet wird.

- Window size gibt an, wie viele Bytes der Empfänger momentan zu empfangen bereit ist. Diese kann von Segment zu Segment unterschiedlich sein. Sie kann sogar 0 sein. In dem Fall möchte der Empfänger erstmal eine Pause haben, weil er mit der Abarbeitung der bereits empfangenen Segmente oder anderen Aufgaben ausgelastet ist [Tanenbaum (2003)].

2.1.2. WLAN

Wenn im Rahmen dieser Arbeit von Wireless-LAN oder WLAN gesprochen wird, ist damit der IEEE Standard 802.11 gemeint. Seit der Entwicklung dieses Standards 1997, sind einige weitere „Unterstandards“ entwickelt worden. Nach dem Originalen 802.11, welcher nie wirklich in praktischem Gebrauch war, verabschiedete sich 1999 die Wireless-LAN Technologie mit den Standards 802.11b und 802.11a von seinem Nischendasein. Mittlerweile sind beide Standards weit verbreitet. Danach folgten weitere, teilweise weniger verbreitete Standards, aber auch der momentan als der Standard angesehene 802.11g.
Die einzelnen Standards unterscheiden sich in ihrer Bandbreite (siehe Tab.2.1), ihres genutzten Frequenzbandes, den Sicherheitsmaßnahmen, sowie einiger anderer Eigenschaften [Roshan und Leary (2004)]. Im Rahmen dieser Arbeit ist dabei Hauptsächlich die Bandbreite, sowie die Anfälligkeit gegenüber Störungen relevant. Eine genauere Analyse dieser Eigenschaften, in Bezug auf diese Arbeit, erfolgt in Kapitel 3.2.

Klasse	**Übertragungsrate Brutto**	**Übertragungsrate Netto max.[a]**
IEEE 802.11	2 MBit/s	- [b]
IEEE 802.11a	54 MBit/s	18,4 MBit/s
IEEE 802.11b	11 MBit/s	4,7 MBit/s
IEEE 802.11g	54 MBit/s	20,8 MBit/s
IEEE 802.11n	540 MBit/s	- [c]

[a] bei Punkt zu Punkt Verbindung mit 1m Abstand
[b] nie richtig im Betrieb gewesen
[c] Standard noch zu neu

Tabelle 2.1.: Datentransferraten der IEEE 802.11x Standards [Pietzko (2009)]

2.2. Echtzeit

Dieser Abschnitt befasst sich mit dem Begriff der Echtzeit im Zusammenhang mit Computersystemen.
Normale Systeme ohne Echtzeitanforderungen werden meistens durch Eingaben gesteuert, die zu diskreten Zeitpunkten erfolgen, während Echtzeitsysteme meistens durch angeschlossene, analoge Peripherie, wie z.B. Sensoren, mit kontinuierlichen Datenströmen gesteuert werden. Echtzeitsysteme besitzen quasi ihre eigenen Sinne [Ward und Mellor (1991)].
Während es bei Nicht-Echtzeitsystemen ausschließlich auf die Korrektheit der Berechnungsergebnisse ankommt, ist es bei Echtzeitsystemen zusätzlich wichtig, dass vorhandene Zeitbedingungen eingehalten werden. Ein Echtzeitsystem stellt bestimmte Anforderungen an Rechtzeitigkeit, Gleichzeitigkeit sowie einer Reaktion auf spontane Ereignisse in einem angemessenen Zeitfenster [Wörn und Brinkschulte (2005)].

- **Rechtzeitigkeit** bedeutet, dass das zu berechnende Ergebnis innerhalb eines vorgegenen Zeitfensters vorliegen muss. Dieses Zeitfenster sollte weder über- noch unterschritten werden. Zum Beispiel bei Abtastzeitpunkten von Sensoren, etwa bei einem Airbag in einem Fahrzeug.
- **Gleichzeitigkeit** bedeutet, dass mehrere Aufgaben, jeweils mit ihren eigenen zeitlichen Anforderungen, parallel bearbeitet werden müssen. Der Autopilot eines Flugzeugs muss zum Beispiel parallel die Geschwindigkeit, sowie Kurs und Höhenmesser auslesen und gegebenenfalls nachregeln.
- **Spontane Reaktion auf Ereignisse** bedeutet, dass auf zufällig auftretende interne oder externe Ereignisse innerhalb eines festgelegten Zeitfensters reagiert werden muss.

Diese Anforderungen sind für jedes Echtzeitsystem individuell und von unterschiedlicher Gewichtung. Dabei unterscheidet man zwischen zwei Gruppen: Anwendungen mit Soft-Realtime Anforderungen und Anwendungen mit Hard-Realtime Anforderungen.

Soft-Realtime. Tasks und Anwendungen mit Soft-Realtime Anforderungen geben in Bezug auf Rechtzeitigkeit zwar eine Zeitgrenze vor, aber wird diese nicht eingehalten passiert dem System bzw. der Umgebung nichts [Stallings (2005)]. Dennoch sollte diese Zeitgrenze nicht regelmäßig überschritten werden. Beispiele hierfür sind unter anderem das im Rahmen dieser Arbeit zu entwickelnde Telemetriesystem oder Telekommunikationsanwendungen. Hierbei ist es nicht sinnvoll die empfangen Daten zu puffern oder bei Datenverlust die Daten neu zu senden, da sich sonst das Gespräch durch die entstehenden Verzögerungen sehr statisch gestaltet. Als Echtzeitanwendung hingegen ist es hier wichtig, dass die Daten in einem bestimmten Zeitfenster verarbeitet und wiedergegeben werden, um diese Verzögerung zu verhindern. Wird dieses Zeitfenster überschritten, führt dies eventuell zu Aussetzern im Gespräch. Das Gespräch wird aber nicht unterbrochen und die Audio-Daten werden nicht aus dem zeitlichen Zusammenhang gerissen. Es wird also kein Schaden an der Hardware oder der Umgebung angerichtet.

Hard-Realtime. Tasks und Anwendungen geben in Bezug auf alle oben genannten Anforderungen einen strikten Zeitrahmen vor. Wird dieser Zeitrahmen nicht eingehalten, führt das zu Schäden, entweder am System oder an der Umgebung des Systems. Bei Echtzeitsystemen mit Hard-Realtime-Anforderungen ist ein Fehler im Zeitbereich genauso kritisch wie ein Fehler im Wertebereich [Rechenberg u. a. (2006)]. Ein klassisches Beispiel hierfür ist die Steuerung des Airbags in einem Kraftfahrzeug. Wird hier nach Auswertung der Sensordaten der Zeitrahmen in einer Verkehrsunfallsituation der Airbag nicht im korrekten Zeitpunkt angesteuert und somit zu früh oder zu spät geöffnet, führt das höchstwahrscheinlich zu einem Schaden. In diesem konkreten Fall an der Gesundheit des zu schützenden Fahrgastes. Die Einhaltung der Zeitgrenze ist also kritisch.

3. Analyse und Design

Für die Anwendung werden zwei eigenständige Teilprogramme benötigt. Eines zum Senden der Daten und eines zum Empfangen, Anzeigen und Abspeichern der Daten. Die Anwendung zum Senden der Daten ist auf dem Netbook, das auf dem Fahrzeug (Abb. 3.1) mitfährt

Abbildung 3.1.: Fahrzeug des FAUST-Teams. Foto: S.Pareigis

und für die Steuerung des Fahrzeugs zuständig ist, zu realisieren. Die Anwendung zum Empfangen, Anzeigen und Abspeichern der Daten, im weiteren Verlauf als User-Interface bezeichnet, ist so zu realisieren, dass sie auf einem beliebigen PC oder Laptop ausführbar ist.
In den folgenden Unterkapiteln werden zunächst die Anforderungen an die Gesamtanwendung erläutert und anschließend eine Ist-Analyse der Plattformen für die Teilanwendungen durchgeführt, sowie Designmöglichkeiten aufgezeigt. Die Designentscheidungen erfolgen daraufhin in Kapitel 4.

3.1. Anforderungen

Die folgenden Anforderungen werden vom CaroloCup-Team der HAW Hamburg an die Anwendung gestellt:

- **Datenübertragung per WLAN-Verbindung**
  Es soll möglich sein sich mit einem beliebigen PC oder Laptop per WLAN eine Verbindung mit dem Netbook des Fahrzeugs aufzubauen. Über diese Verbindung sollen ausgewählte Daten vom Netbook versendet und auf Seiten des User-Interfaces empfangen werden können.
- **Betriebssystemunabhängigkeit**
  Das User-Interface soll so entwickelt werden, dass es für die gängigen windows- und linuxbasierten Betriebssysteme verwendbar ist. Eine Portierung des User-Interfaces auf ein anderes Betriebssystem soll durch einfaches Neuübersetzen, also ohne Veränderung, des Codes realisierbar sein.
- **Flexibilität**
  Beide Anwendungsteile sollen einfach erweitert und auf die Bedürfnisse des Anwenders zugeschnitten werden können. Der Anwender soll in der Lage sein, die grafische Oberfläche des User-Interfaces nach seinen Anforderungen, aus einem vorgegebenen Pool von Widgets, selbst zusammenzustellen. Außerdem soll der Anwender in der Lage sein zu bestimmen, ob und in welchen Zyklen welche Daten gesendet werden. Weiter soll es möglich sein, die fertige Anwendung flexibel zu erweitern, falls neue Sensoren etc. hinzukommen.
- **„Replay“ - Mechanismus**
  Es soll möglich sein, die übertragenen und angezeigten Daten zu speichern. Diese sollen zu einem späteren Zeitpunkt in korrekter zeitlicher Abfolge wieder abgespielt werden können.
- **Verlaufsgraphen**
  Stark dynamische Sensorwerte sollen, zur besseren Analyse, in einem Verlaufsgraphen dargestellt werden können. Diese Verlaufsgraphen sollen einen Zeitraum von etwa zwei Sekunden überspannen.
- **geringer Zyklus**
  Beide Teilanwendungen müssen ihre Berechnungen in einem geringen Zeitintervall durchführen. Auf Seite der Teilanwendung zum Senden der Daten im einstelligen Millisekundenbereich und auf der Seite der User-Interfaces im Bereich von unter 25 Millisekunden.

- **zur Übertragung stehen folgende Daten an:**
  - **Sensordaten**
    Alle verfügbaren Sensorwerte sollen übertragen und angezeigt werden. Dies umfasst Abstandssensoren, Energiestatussensoren, Inkrementalgeber, Beschleunigungssensoren, sowie einen Kompass.
  - **Kamerabild (Bilddaten)**
    Das Bild der Kamera inklusive aller in das Bild eingezeichneten Markierungen. Die Größe des Bildes muss hierbei nicht zwingend erhalten bleiben, sollte aber eine Größe haben in der die Informationen wahrnehmbar sind.
  - **Karteninformationen (Mapdaten)**
    Eine lokale Umgebungskarte soll übertragen und dargestellt werden. Diese Umgebungskarte besteht aus einzelnen Punkten, welche sich aus einer X- und einer Y-Koordinate, sowie einer Typenbeschreibung in Form eines Integers bestehen. Die Typenbeschreibung gibt dabei an, ob es sich bei dem Punkt z.B. um eine Fahrbahnmarkierung, ein Hindernis etc. handelt.
  - **Debug-Nachrichten (Debugdaten)**
    Es soll möglich sein, eigene Debug-Nachrichten in Form eines Strings zu formulieren, zu übertragen und darzustellen.

3.2. Teilanwendung: Sender

Die Plattform für die Software zur Übertragung der Daten ist das, auf dem Fahrzeug montierte, Netbook. Dieses Netbook ist mit einer Intel Atom CPU, 1 GB RAM und einer internen 802.11g WLAN-Karte ausgestattet. Das Betriebssystem ist ein konsolenbasiertes Linux.
Das Fahrzeug wird durch eine Anwendung gesteuert welche, wie in Abbildung 3.2 schematisch dargestellt, aus verschiedenen, voneinander unabhängigen Tasks, sowie einigen Utilities und Treibern besteht. Dabei rufen die Treiber, die jeweils voneinander unabhängige Threads sind, die Daten von der Hardware, wie z.B. der Kamera, ab. Die Tasks analysieren diese Daten, werten sie aus und geben Steuerbefehle an die Hardware zurück. Die Utilities sind kleine Hilfsprogramme für die Tasks oder den Entwickler. Bei den hier erwähnten Treibern handelt es sich nicht um Treiber im herkömmlichen Sinne, die unter Kontrolle des Betriebssystems stehen. Diese Treiber sind unabhängige Threads innerhalb der Anwendung, die dieser Treiberfunktionalität für die angeschlossene Peripherie zur Verfügung stellen.

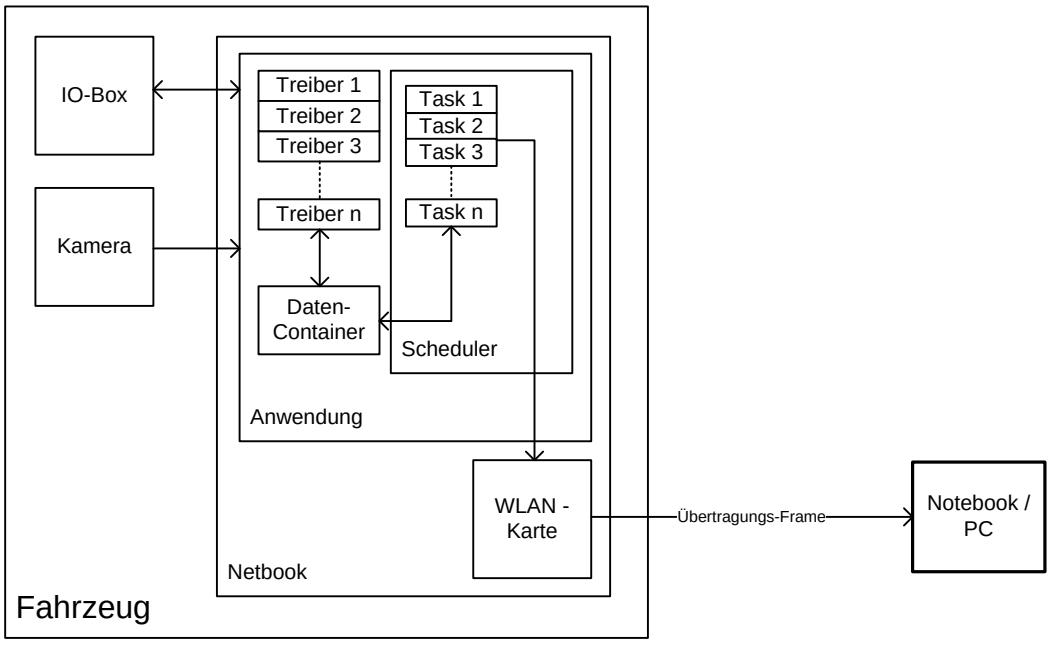


Abbildung 3.2.: Schematische Darstellung der Fahrzeugplattform

Vorhandenes Design nutzen oder ignorieren? Die Anwendung hat ein internes Scheduling (Abb.3.3). Bei diesem Scheduling handelt es sich um ein kooperatives Scheduling. Das bedeutet, dass in diesem Scheduling eine Liste mit allen Tasks vorhanden ist, die in

der Reihenfolge ihrer Prioritäten abgearbeitet wird, ohne das eine der Tasks mehrmals pro Zyklus zum Zuge kommt. Abgesehen von dem Fall bei dem der Timer des Schedulers abgelaufen ist, geben die Tasks, nachdem sie ihre Arbeit beendet haben, die CPU selbstständig wieder frei. Während die Liste von Tasks durchlaufen wird, werden Datenstrukturen, wie z.B. die Ansteuerung der Motoren, durch den jeweils aktuellen Task, ggf. immer wieder neu interpretiert und überschrieben. Dadurch kann gewährleistet werden, dass ein Task mit einer hohen Priorität am Ende auch „das letzte Wort“ hat, um z.B. eine Kollision zu vermeiden.
Die Treiber sind, wie bereits erwähnt, eigenständige Threads, innerhalb der Anwendung, und laufen parallel und asynchron zum Scheduler.

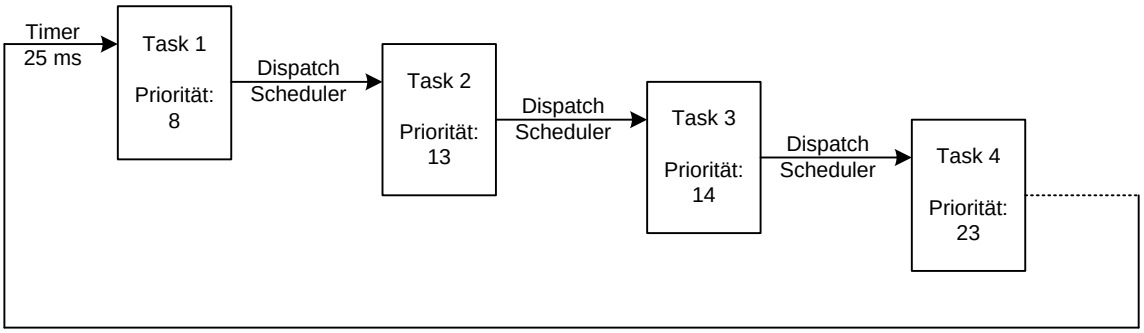


Abbildung 3.3.: Schematische Darstellung des Schedulings aus Abb. 3.2

Der Scheduler hat eine variable, vom User festlegbare, Zyklusdauer von minimal 25 Millisekunden, in der alle ausgewählten Tasks ihre Arbeit erledigt haben müssen. Läuft der Timer des Schedulers aus, so wird der aktuelle Task, unabhängig von seinem aktuellen Status terminiert und der Scheduler wird zurückgesetzt. Anschließend beginnt der Scheduler von vorne die Liste mit den Tasks abzuarbeiten. Dies führt zum einen dazu, dass alle Tasks die nach dem terminierten Task in der Liste standen natürlich auch nicht mehr zum Zuge kamen und keinen Einfluss auf das Fahrzeug ausüben konnten. Allerdings wird dadurch auch eine feste Zyklusdauer garantiert. Bei Unterschreitung der Zyklusdauer (alle Tasks sind vor Ablauf des Timers abgearbeitet) wartet der Scheduler auf Ablauf des Timers, um neu gestartet zu werden. Innerhalb der Anwendung kann der User über ein Web-Interface die benötigten Softwareelemente auswählen, sowie die gewünschte Zyklusdauer festlegen und anschließend das System starten.
Bei der Analyse und Fehlersuche innerhalb einer Anwendung, z.B. in einer Funktion oder einem Task, sind Daten die aktuell keinen Einfluss auf diesen Teil der Anwendung haben, weil sie weder gelesen noch geschrieben werden, oder veraltet sind, nicht von Bedeutung. Unter gewissen Umständen behindern sie diesen Vorgang sogar, indem sie dem Entwickler falsche Informationen darstellen, die zu falschen Schlussfolgerungen führen können. Des-

halb ist es sinnvoll das Sendeprogramm als, innerhalb des Schedulers laufenden, Task zu implementieren. So kann sichergestellt werden, dass die Daten die gesendet werden sollen aktuell und korrekt sind. Dadurch ist keine zusätzliche Synchronisation mit den anderen Tasks nötig, da bereits durch den Scheduler alle Tasks auf einen Zyklus synchronisiert sind. Eine hundertprozentige Synchronität lässt sich allerdings auch hiermit nicht erreichen, da die Treiber, welche die Daten von der Hardware liefern, als eigenständige Threads nicht unter der Leitung des Schedulers laufen und somit asynchron zu ihm sind. Das bedeutet, dass es passieren kann, dass der nachfolgende Task bereits mit einem neueren Datensatz arbeitet als der aktuelle Task, weil einer der Treiber-Threads den Datensatz in der Zwischenzeit aktualisiert hat. Setzt man die Sende-Task durch die Priorität direkt hinter die zu analysierende bzw. debuggende Task minimiert man die Wahrscheinlichkeit für einen solchen Fall. Denn nach der Formel

$$Wahrscheinlichkeit\ P = \frac{Zeit\ zwischen\ Tasks}{Zeit\ aller\ Tasks}$$

ist die Wahrscheinlichkeit am geringsten, wenn die Tasks direkt hintereinander stehen. Ganz ausschließen lässt sich dieser Fall jedoch leider nicht.
Um die Wahrscheinlichkeit für den Fall zu verringern, dass nachfolgende Tasks „verhungern“, sollte außerdem auf eine möglichst geringe Laufzeit des Tasks geachtet werden, um so die Gesamtlaufzeit aller Tasks in der Liste nicht zu stark zu erhöhen.

Aufkommendes Datenvolumen Die Sensorwerte befinden sich in einem Daten-Container, der in einem regelmäßigen Zeitintervall, standardmäßig alle 25 ms, von einem der Treiber-Threads aktualisiert wird.
Standardmäßig werden 33 Schwarz-Weiß Bilder pro Sekunde, ebenso durch einen, asynchron zum Scheduler laufenden Treiber, von einer USB-Kamera als Einzelbilder im Image Processing Library (IPL) Format [Intel (1998)], mit einer Auflösung von 752 x 480 Pixeln, geliefert. Die Anzahl der Bilder pro Sekunde ist variabel über das Web-Interface wählbar, wobei maximal 87 Bilder pro Sekunde möglich sind. Da beim IPL-Format für ein Schwarz-Weiß Bild ein Byte pro Pixel Speicher benötigt wird, errechnet sich bei dieser Auflösung ein Speicherbedarf von 352,5 kB für die Bilddaten, sowie einige Bytes für den Header des IPL-Formats, pro Bild.
Die opencv Bibliotheken, die innerhalb der Anwendung zur Bildverarbeitung verwendet werden, bringen einige Methoden zur Speicherung bzw. Konvertierung in ein anderes Bildformat wie z.B. JPG oder BMP mit. Im Falle von JPG ist es dadurch möglich das Bild zu komprimieren und den Bandbreitenbedarf so zu senken.
Zum Zeitpunkt der Erstellung dieser Arbeit sind pro Zyklus einundzwanzig 32-Bit Sensorwerte, 352,5 kB Bilddaten, maximal 50 Mappunkte á 3 mal 32-Bit, sowie maximal 10 Strings mit einer maximalen Länge von 100 Zeichen zu senden. Dadurch ergibt sich, wie in Tabelle

Informations-typ	Datentyp	Anzahl	Speicherbedarf Datentyp	Gesamtspeicherbedarf pro Zyklus
Bilddaten	byte	360.960	1 Byte	360.960 Byte
Sensordaten	float	21	4 Byte	84 Byte
Mapdaten	3 x int	50	12 Byte	600 Byte
Debugdaten	100 x char	10	100 Byte	1.000 Byte
			Gesamt	**362.644 Byte**

Tabelle 3.1.: Zu sendendes Datenvolumen pro Zyklus

3.1 zu sehen, eine Datenmenge von insgesamt 362.644 Byte bzw. ca. 354 kB. Durch die Formel

$$Datenvolumen\ pro\ Sekunde = Datenvolumen\ pro\ Zyklus * \frac{Sekunde}{Zyklusdauer}$$

errechnet sich bei einer Zyklusdauer von 25 Millisekunden ein Datenvolumen 110MBit/s.

Datenvolumen verringern Wie bereits geschildert, besitzt das Netbook, das auf dem Fahrzeug mitfährt eine WLAN Karte die dem 802.11g Standard entspricht. Der Tabelle 2.1 aus Kapitel 2.1.2 ist zu entnehmen, dass dem 802.11g Standard ein Bruttodatendurchsatz von 54MBit/s und ein Nettodatendurchsatz von maximal 40%, also etwa 21MBit/s möglich ist. Das bedeutet das die Datenmenge deutlich verkleinert werden muss.

Da die Bilddaten ca. 99,5% des Datenvolumens darstellen, lässt sich eine Verkleinerung der Gesamtdatenmenge am effektivsten durch Verkleinerung der Menge der Bilddaten erreichen. Dafür bieten sich vier Möglichkeiten an.
Die erste ist die Kompression des Bildes, entweder durch die Funktion der opencv-Bibliotheken oder durch eine andere Funktion, wie sie z.B. von zipähnlichen Bibliotheken zur Verfügung gestellt werden.
Die zweite Möglichkeit ist, das Bild nicht jeden Zyklus, sondern nur jeden zweiten, dritten oder x-ten Zyklus zu senden. Hierbei ist es wichtig darauf zu achten, dass auf der Empfängerseite immer noch ein verwertbarer Strom von Bildern ankommt. Ein Strom von unter 10 Bildern pro Sekunde macht wenig Sinn. Dem Anwender würden so zum einen zu viele Informationen entgehen, zum anderen würde die Anzeige der Bilder sehr ruckelig aussehen, was eine Analyse der Daten unnötig erschweren würde.
Eine weitere Möglichkeit die Datenmenge zu verkleinern ist das Bild selbst zu verkleinern, also die Auflösung zu verringern. Bei dieser Methode ist darauf zu achten, dass nicht zu viele Informationen verloren gehen, bzw. das Bild nicht zu klein für den Betrachter wird.
Letzte Variante ist eine Kombination aus den vorher genannten Möglichkeiten.

Designmöglichkeiten der Übertragungsframes aus Abb. 3.2. Benötigt für die Frames, in denen die unterschiedlichen Datentypen übertragen werden, wird ein Index, eine Größenangabe, Informationen, um welche Datentypen es sich jeweils handelt und natürlich die jeweils dazugehörigen Daten selbst.
Dabei nötig für eine korrekte Zuordnung in die richtige Reihenfolge ist ein eindeutiger Index. Hierfür eignet sich zum Beispiel eine Variable die hochgezählt wird. Besser noch wäre ein Zeitstempel, der bei der späteren Wiedergabe der Daten noch den Vorteil mit sich bringt, dass man relativ genau den zeitlichen Abstand zwischen zwei Datensätzen bestimmen kann und somit die Ausgabe der Daten in der ursprünglichen zeitlichen Abfolge dargestellt werden können.
Eine weitere Notwendigkeit ist eine Information über die Länge, beziehungsweise das Ende des Datensatzes. Dazu könnte man entweder am Anfang des Frames einen Wert mitsenden der die Länge des Datensatzes angibt oder man definiert ein „Ende“-Label, welches an das Ende des Frames gehängt wird, damit der Sender dann bemerkt, dass er keine weiteren Daten einlesen muss. Falls die Menge der Daten stets gleich ist, weil immer dieselbe Art und Menge an Daten gesendet wird, kann man dies auch in einer externen Konfigurationsdatei vermerken. Dadurch würde man etwas weniger Daten zu senden haben, aber man verliert an Dynamik.
Die Sensorwerte sind alle als float Wert im Datencontainer vorhanden. Deshalb ist es nicht nötig den Datentyp für jeden Sensorwert mitzuschicken. Nötig wäre nur jeweils ein Label zur Unterscheidung zwischen den einzelnen Wertekategorien wie Sensorwerte, Kamerabild, etc..
Daraus ergeben sich einige mögliche Kombinationen. Im Folgenden werden einige dieser Kombinationen betrachtet und auf ihre Vor- und Nachteile analysiert.

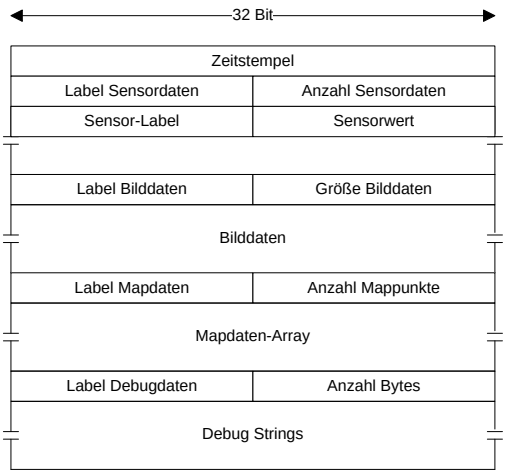


Abbildung 3.4.: Variante mit einem Frame

Die erste Variante wäre alle Daten in ein Frame zu stecken. Das könnte dann wie in Abb.3.4

aussehen. Da ein UDP-Datagram, abhängig vom Betriebssystem, maximal 64 kB groß sein kann, müsste man das Datenvolumen auf diese Größe reduzieren oder in mehr als ein Paket aufteilen. Der Vorteil dabei wäre, dass man nur einen Socket auf jeder Seite benötigen würde. Der Nachteil jedoch wäre, dass wenn ein Paket verloren ginge, die Daten möglicherweise unbrauchbar wären und auf Empfängerseite verworfen würden.

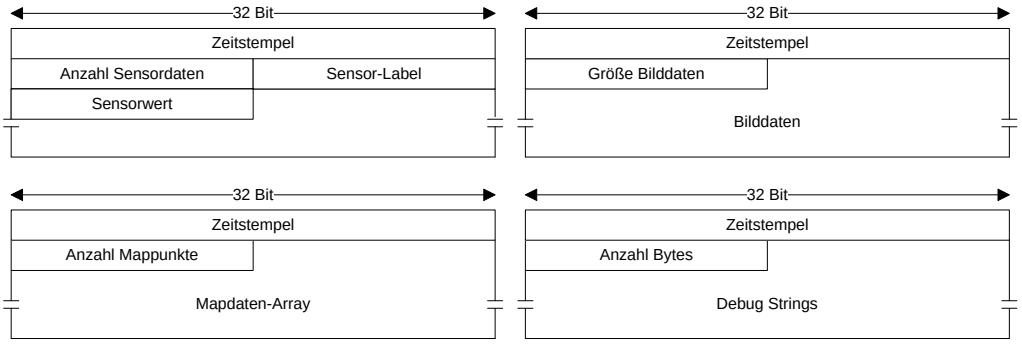


Abbildung 3.5.: Variante mit mehreren Frames

Die zweite Variante teilt die Daten in mehrere Frames auf (Abb.3.5). Dabei hat man dieselbe Flexibilität wie in der ersten Variante, verliert bei Paketverlust aber nur einen Teil der Daten. Des weiteren können, durch die Benutzung mehrerer Sockets, auf Empfängerseite die Daten einfacher parallel verarbeitet werden. Der einzige Nachteil an dieser Variante ist, dass das Datenvolumen immer noch nicht optimal ist.
Die dritte Variante ist eine Optimierung der zweiten Variante. Hierbei werden die Labels der Sensoren nicht mitgesendet, sondern es wird die Reihenfolge der Sensordaten innerhalb des Arrays in einer Konfigurationsdatei festgelegt. Diese sollte dann aus Kompatibilitätsgründen zu anderen Versionen allerdings nur ergänzt, aber nie abgeändert werden. Außerdem kann man das Feld, das die Anzahl der Mappunkte mitteilt einsparen, wenn man in der Konfigurationsdatei eine feste Anzahl angibt. Der Nachteil dabei ist, dass man Flexibilität zur Laufzeit einbüßt.

3.3. Teilanwendung: User-Interface

Das User-Interface dient zur Darstellung der gesendeten Daten sowie zur eventuellen Speicherung derselben.
Da keine bestimmte Plattform spezifiziert wurde, ist, um einen Richtwert für die Performance zu erhalten, ein Laptop mit einer 2 x 1 GHz CPU, 2 GB RAM und einer internen 802.11g WLAN-Karte verwendet worden.

Betriebssystemunabhängigkeit. Um die Betriebssystemunabhängigkeit zu realisieren gibt es mehrere Ansätze.
Der Erste ist, man kümmert sich selbst darum, dass alles in den entsprechenden Betriebssystemen kompilierbar ist, indem man alle Eventualitäten mit entsprechenden Defines im Code abfängt. Diese Möglichkeit ist recht aufwendig und damit auch fehleranfällig, aber man hat maximalen Einfluss auf die Performance.
Der Nächste wäre die Anwendung in einer Sprache zu implementieren die von Haus aus schon betriebssystemunabhängig ist, wie z.B. Java. In diesem Fall hat man, speziell im Falle Java, einen Mechanismus, wie die Java Virtuell Machine, unter dem Code laufen der diese Unabhängigkeit realisiert. Dadurch hat man nur bedingt Einfluss auf die Performance der Anwendung.
Ein weiterer Ansatz wäre die Anwendung mit Hilfe eines Frameworks, wie z.B. GTK+ [Krause (2007)] oder Qt [Blanchette und Summerfield (2009)], zu realisieren. Diese Frameworks setzen auf objektorientierte Programmiersprachen wie C++ auf und bieten eigene Datentypen an, hinter denen dann, die in der ersten Möglichkeit erwähnten, Defines stehen. Dadurch ist es möglich den Code auf unterschiedlichen Betriebssystemen zu kompilieren und auszuführen. Hierbei hat man wie in der ersten Möglichkeit guten Einfluss auf die Performance, muss sich aber nicht selbst um die, unter Umständen sehr aufwendige, Portierung auf die verschiedenen Betriebssysteme kümmern.

GUI-Frameworks. Das GUI-Framework sollte der Realisierung der Betriebssystemunabhängigkeit keinesfalls im Wege stehen. Außerdem sollte das Framework wegen eventueller zukünftiger Ergänzungen und Neuerungen in der Anwendung, die dann wahrscheinlich nicht vom Verfasser dieser Arbeit implementiert werden, mit einer objektorientierten Programmiersprache arbeiten, die von den Nachfolgern auch beherrscht wird. Da an der HAW Hamburg hauptsächlich C++ und Java unterrichtet wird, ist eine relativ hohe Wahrscheinlichkeit gegeben, dass eine dieser beiden Sprachen im Repertoire des Nachfolgers ist. Aus diesem Grund werden hier nur Frameworks erwähnt, die auf einer dieser beiden Sprachen aufbauen. Im Folgenden werden einige Frameworks erläutert.

- **Java Swing** ist ein GUI Toolkit mit dem sich relativ einfach große und komplexe Anwendungen realisieren lassen. Vorhandene Komponenten lassen sich erweitern und neue mit gewissem Aufwand erstellen. Dadurch, dass es auf Java basiert ist es betriebssystemunabhängig und fordert nur eine vorhandene Java Virtual Machine. In Swing geschriebene Anwendungen werden in der Regel in einem Browser ausgeführt, können aber, mit entsprechender Vorbereitung, auch als eigenständige Anwendung laufen. Swing ist, dadurch dass es auf Java basiert, potentiell ein wenig langsamer als auf C++ basierende Frameworks [Loy (2003)].

- **GTK+** ist ein flexibles Framework zur Erstellung von *graphical user interfaces* (GUI), welches eine plattformübergreifende Kompatibilität implementiert. Das bedeutet, dass ein Programm nur einmal geschrieben werden muss und dann einfach für jedes gewünschte Betriebssystem kompiliert werden kann. Zur Zeit ist dies für Windows-, Mac- sowie Linux/Unix-Systeme möglich. Außerdem ermöglicht GTK+ das Entwickeln in verschiedenen Sprachen, unter anderem C++, Python und C#.
  GTK+ steht unter der GNU LGPL 2.1. Dadurch ist es möglich ohne Kosten Software zu entwickeln. Unter anderem wird GTK+ für die Entwicklung des GNOME Desktops für Linuxsysteme verwendet.
  Es enthält alle Standard-Elemente die für die meisten Programme nötig sind. Weitere Elemente können mit gewissem Aufwand selbst erstellt und hinzugefügt werden [Krause (2007)].

- **Qt** ist ein Framework das speziell dafür entwickelt wurde, GUIs für verschiedene Plattformen entwickeln zu können, ohne das für den Entwickler ein Mehraufwand für die Portierung anfällt oder sich an der Funktionalität der GUIs etwas ändert. Dies ist für alle gängigen Betriebssysteme möglich.
  Qt basiert auf C++ und bietet alle Standardelemente, die für den Bau einer GUI nötig sind. Alle vorhandenen Elemente können angepasst werden. Fehlende Elemente und Funktionalität kann mit entsprechendem Aufwand in C++ nachgebildet werden.
  Außerdem bietet es mit *Signale und Slots* ein System welches auf einfache Weise interuptähnliche Funktionalität zur Verfügung stellt [Blanchette und Summerfield (2009)].

- **LabVIEW** ist eine grafische Programmierumgebung, mit der sich Mess-, Prüf-, Steuer,- und Regelsysteme entwickeln und darstellen lassen. Dabei werden intuitive grafische Symbole eingesetzt und miteinander verbunden, so dass eine Art Flussdiagramm entsteht. Es bietet eine recht große Auswahl an Widgets und Symbolen in diesem Bereich und es ist möglich fehlende Elemente und Funktionalität durch einbinden von C-Code und .dll Bibliotheken zu ergänzen. Mit LabVIEW lässt sich auf und für Windows-, Mac-,

Linux-, sowie einige Echtzeitsysteme entwickeln [Jamal und Hagestedt (2005)].

Flexibilität. Gemäß den Anforderungen soll es möglich sein, die Anwendung ohne allzu großen Aufwand erweitern und/oder auf die eigenen Bedürfnisse anpassen zu können.
Eine Möglichkeit dies zu erreichen ist, die Anwendung so zu designen, dass Änderungen nur an wenigen, im Idealfall nur an einer Stelle, im Code getätigt werden müssen. Zum Beispiel indem die Anwendung durch Parameter in einem Header-File konfiguriert wird. Der Nachteil an dieser Möglichkeit ist, dass der Code nach jeder Änderung neu kompiliert werden muss.
Eine andere Möglichkeit wäre dieses Header-File aus dem Code auszulagern und z.B. in ein oder mehrere XML-File/s zu verlegen. Diese Möglichkeit hat den Vorteil, dass man die Änderungen vornehmen kann ohne anschließend den Code neu kompilieren zu müssen. Man braucht also nicht über die entsprechende Entwicklungsumgebung zu verfügen.
In diesen Files, in welcher Variante sie vorliegt spielt hier erst einmal keine Rolle, würde dann festgelegt werden:

- wo welches Element auf der Oberfläche positioniert werden soll
- wo sich welcher Wert im Datenpaket befindet
- welcher Wert welchen Elementen zugeordnet wird
- welche möglichen Elemente zur Verfügung stehen
- wie die Elemente beschriftet werden sollen

Speicherung und Wiedergabe. Wie den Anforderungen zu entnehmen ist, soll es außerdem möglich sein die empfangenen Daten abzuspeichern und zu einem späteren Zeitpunkt in korrekter zeitlicher Relation, wieder abspielen zu können.
Um das zu bewerkstelligen kann man eine Log-Datei erstellen, in die alle Daten geschrieben werden, wie in Abb. 3.6 dargestellt.

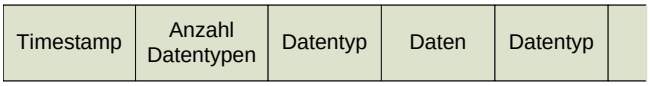


Abbildung 3.6.: Alle Daten in einer Log-Datei speichern

Das hat den Vorteil, dass alle Daten an einem Ort gespeichert sind, man nur eine Datei zu öffnen brauch und man die File-Table der Festplatte nicht belastet. Allerdings wird diese Datei, vor allem durch die Bilddaten, relativ schnell recht groß werden. Das hat zur Folge, dass Suchoperationen in der Datei immer langsamer funktionieren und die Ausführung

eines Replays oder das Raussuchen von Zeitmarken für das Replay zur Geduldsprobe für den Anwender wird. Zusätzlich kommt je nach Formatierung des Datenträgers noch die maximale Dateigröße als obere Grenze hinzu die allerdings nur bei FAT16-Datenträgern, wie sie heute zutage hauptsächlich nur noch auf USB-Sticks zu finden sind, mit zwei GB relevant sind. Bei realistischen zwei MB pro Sekunde Datenaufkommen wäre diese Größe bereits nach knapp 16 Minuten erreicht.
Alternativ kann man für jeden Zeitstempel die Daten jeweils in eine Datei speichern (Abb.3.7). Dabei wird einmalig pro Zeitstempel nur eine Datei geöffnet. Wenn man dabei die Datei nach dem Zeitstempel benennt, erspart man sich beim Suchen nach einem bestimmten Zeitstempel das Öffnen und Durchsuchen der Dateien und man hat so einen schnelleren Zugriff auf die Daten des gewünschten Zeitstempels. Dabei würden dann bei einem 25 ms Zyklus pro Sekunde 40 Dateien erstellt werden, was z.B. eine NTFS Filetable mit einer maximalen Dateianzahl von 2^{32} nicht sonderlich belastet. Der Nachteil dabei ist, dass man sich außerhalb der Anwendung z.B. die Bilddaten, z.B. zwecks einer Löschung von nicht benötigten Daten, nicht anschauen kann.

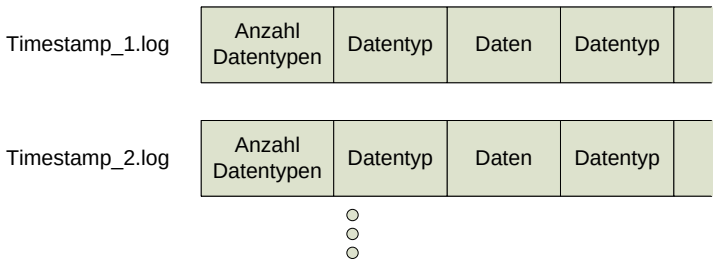


Abbildung 3.7.: Eine Log-Datei pro Timestamp

Eine weitere Alternative stellt das Speichern nach Datentyp dar. Dabei werden Bild-, Debug-, Sensor- und Mapdaten jeweils in eine eigene Datei gespeichert. Dabei würden pro Sekunde 160 neue Dateien angelegt werden. Trotz dieser recht großen Menge an Dateien müsste man sich keine Sorgen um eine überfüllte File-Table machen, denn nach 72 Tagen Dauerbetrieb wäre eine NTFS Filetable nur etwa ein Viertel mit Log-Dateien gefüllt. In dieser Hinsicht gibt es also keine Bedenken. Der Vorteil dieser Variante ist, dass man z.B. die Bilder auch betrachten kann ohne die Anwendung starten zu müssen. Der Nachteil dabei ist die große Menge an Dateien, bei der man schnell die Übersicht verlieren kann.

Zur Wiedergabe der Daten liest man erst die Zeitstempel ein. Danach startet man einen Timer mit der Differenz zwischen dem aktuellen Zeitstempel und dem nächsten und liest anschließend die zum aktuellen Zeitstempel gehörenden Daten ein und gibt diese an die GUI weiter. Der Timer startet dann bei Ablauf einen neuen Thread der den eben beschriebenen Vorgang durchführt und den Timer neu stellt. Alternativ kann der Timer einen Interrupt auslösen, in dessen Routine dieser Vorgang dann durchgeführt wird. Erstere Variante würde

eine parallele Verarbeitung mehrerer Vorgänge ermöglichen, da viele der heutigen Computer eine Mehrkern-CPU verbaut haben. So könnte ein Thread die Daten jeweils einlesen und ein weiterer die GUI aktualisieren. Dieses Verfahren würde auch dem nächsten Punkt zugute kommen.

Einhaltung des Zeitrahmens. Da die Teilanwendung des Senders standardmäßig in einem 25 Millisekunden Zyklus ausgeführt wird und damit auch Daten gesendet werden, muss die Teilanwendung des User-Interfaces in der Lage sein diese Daten innerhalb von jenen 25 Millisekunden zu verarbeiten.
Um eine möglichst geringe Durchlaufzeit eines Bearbeitungszykluses zu erreichen, ist es wichtig auf einige, normalerweise selbstverständliche Dinge besonders acht zugeben. So sollte man, falls nicht unbedingt nötig, auf langsame Festplattenzugriffe außerhalb der Initialisierungsphase verzichten. Zum Beispiel indem man in jedem Zyklus die Konfigurationsdateien zu Rate zieht, um zu evaluieren wie ein bestimmter Sensorwert behandelt werden soll. Weiter sollte man auf Kopieroperationen im Speicher wann immer möglich verzichten. So ist es zum Beispiel häufig nicht nötig eine Datenstruktur, wie z.B. ein Array zu kopieren, sondern es reicht meist einen Zeiger auf diese Struktur zu übergeben und diese vor gleichzeitigem Zugriff über z.B. einen Mutex zu schützen.
Zusätzlich Zeit sparen lässt sich auch durch die Parallelisierung von Abläufen, in Fällen wo es möglich ist. So könnte man zum Beispiel den Datenempfang und die Aktualisierung der GUI in zwei Threads aufteilen. Der eine Thread würde dann die Daten vom Netzwerk einlesen, vielleicht schon analysieren und auswerten, in einen Datencontainer schreiben und Thread Nummer zwei benachrichtigen. Während Thread Nummer zwei nun die GUI aktualisiert, könnte der erste Thread schon wieder Daten empfangen. Ein weiterer Thread könnte die Speicherung der Daten auf Festplatte übernehmen. Dies ist natürlich nur wirklich effektiv, wenn der Computer über mehr als eine Recheneinheit verfügt.

4. Realisierung

In diesem Kapitel wird beschrieben, welche der, im vorherigen Kapitel aufgezeigten, Designmöglichkeiten ausgewählt wurden. Diese Entscheidungen werden begründet und die Umsetzung des jeweiligen Designs wird beschrieben.

4.1. Teilanwendung: Sender

An dieser Stelle wird die Entscheidung für das Design der Teilanwendung des Senders beschrieben und erläutert. Zu Anfang wird die Entscheidung diskutiert, ob die Teilanwendung als externes Programm realisiert oder als Task in die Anwendung des FAUST-Teams integriert wird. Anschließend wird beschrieben mit welchen Mitteln das Datenvolumen reduziert wurde und in welchem Format die Daten übertragen werden. Im weiteren Verlauf wird der Sender selbst beschrieben und die Ergebnisse dargelegt.

Vorhandenes Design nutzen oder ignorieren? Die Teilanwendung „Sender“ wurde als Task in die bereits vorhandene FAUST-Anwendung integriert. Dadurch sind keine zusätzlichen Synchronisationsmechanismen zwischen der Teilanwendung des Senders und der FAUST-Anwendung nötig, wodurch auch potentielle Fehlerquellen wegfallen. Somit ist auch gewährleistet, dass die Teilanwendung immer im gleichen Zyklusintervall ausgeführt wird wie die FAUST-Anwendung. Wird dort der Zyklus von 25 ms auf z.B. 50 ms erhöht, muss diese Änderung nicht zusätzlich auch in der Teilanwendung durchgeführt werden, wodurch auch wieder eine potentielle Fehlerquelle eliminiert wird.
Außerdem kann so über den Parameter „Priority“ genau festgelegt werden, an welcher Position in der Taskliste des Schedulers sich die Teilanwendung befinden soll, bzw. auf welchen Task sie folgen soll. Möchte man z.B. den Task der sich an vierter Stelle in der Liste befindet und die von ihm genutzten Daten genauer betrachten, so setzt man den Sender durch die Priorität an die fünfte Position. Dadurch wird, wie in Kapitel 3.2 beschrieben, nach der Formel

$$Wahrscheinlichkeit\ P = \frac{Zeit\ zwischen\ Tasks}{Zeit\ aller\ Tasks}$$

die Wahrscheinlichkeit minimiert, dass in der Zeit zwischen Nutzung des Datensatzes

durch den Task und dem Senden des Datensatzes durch den Sende-Task, von einem der asynchron zum Scheduler laufenden Treiber, ein aktualisierter Datensatz bereitgestellt wird.

Verringerung des Datenvolumens. Wie in Tabelle 3.1 zu sehen ist und in Kapitel 3.2 erläutert wurde, ist eine Reduzierung des Datenvolumens nötig. Wie dort ebenfalls zu sehen, ist mit ca. 99,5% des anfallenden Datenvolumens der Großteil auf die Bilddaten zurückzuführen. Aus diesem Grund ist es am sinnvollsten das Datenvolumen der Bilddaten zu verringern. Bei einer perfekten Verbindung zwischen zwei Teilnehmern ist im 802.11g WLAN-Standard eine maximale Transferrate von ca. 21 MBit/s möglich. Bei dieser Transferrate dürfte die maximale Datenmenge, bei einem 25 ms Zyklus, 66,56 kB nicht übersteigen.

$$20,8MBit * \underbrace{1024/8}_{\text{zu kB umrechnen}} / \underbrace{\frac{1}{25ms}}_{\text{Anzahl Zyklen pro Sekunde}} = 66,56\ kB$$

Da eine perfekte Verbindung unter den gegebenen Bedingungen während des Carolo-Cups (sich bewegendes Fahrzeug, andere Fahrzeuge mit WLAN-Verbindung im Raum, etc.) ziemlich unwahrscheinlich ist, wäre eine Datenmenge von ≤ 30 kB optimaler.
Um dies zu erreichen wurde zuerst die, in den opencv-Bibliotheken enthaltene, Methode verwendet, mit der sich das IPL-Image in ein komprimiertes JPG-Image wandeln lässt. Das Ergebnis dabei ist aber für diesen Anwendungsfall keines Falls zufriedenstellend. Zum einen weil das Ergebnis dieser Umwandlung stark variiert, das JPG-Image hat, je nach Ausgangsbild, eine Größe zwischen 75 kB und 130 kB, was immer noch zu groß ist. Zum anderen dauert der Prozess der Umwandlung, hauptsächlich bedingt durch die Kompression, zwischen 50 ms und 70 ms. Eindeutig zu lange für eine Zyklusdauer von 25 ms.
Da das IPL-Format [Intel (1998)] bei einem schwarz-weiß Bild die Bilddaten in einem Array ablegt in dem jede Stelle einem Pixel entspricht, lassen sich gezielt Informationen verändern bzw. weglassen. Wenn man aus dem Bild jede zweite Zeile und jede zweite Spalte entfernt, bekommt man ein Bild, welches für den Anwender noch die nötigen Details liefert, aber nur noch ein viertel der ursprünglichen Größe besitzt und somit auch noch noch ein viertel des ursprünglichen Datenvolumens benötigt. Nach diesem Schritt hat das Bild, bei einer verbleibenden Auflösung von 376 x 240, also nur noch eine Größe von 88 kB (ca. 25%), ist aber immer noch zu groß.
Auch an dieser Stelle erreicht die opencv Funktion leider kein zufriedenstellendes Ergebnis. Zwar ist das JPG-Image nur noch zwischen 20 kB und 30 kB groß, aber die Ausführung verbraucht immernoch zwischen 35 ms und 50 ms.
Durch Rücksprache mit dem FAUST-Team wurde ermittelt, dass sich im Bild Bereiche befinden, die weder zur Fahrbahnerkennung noch zur Hinderniserkennung oder sonstigen Orientierungsmaßnahmen verwendet werden. Diese Bereiche befinden sich am Rand. Da diese Bereiche von der Software nicht ausgewertet werden, ist es für den Anwender auch nicht unbedingt nötig diesen Bereich angezeigt zu bekommen. Diese Bereiche haben folgende

Ausmaße: oben 60 Zeilen, unten 20 Zeilen, an den Seiten jeweils 38 Spalten. Entfernt man diese Randbereiche bleibt ein Bild mit einer Auflösung von 300 x 160 Pixeln und einem Datenvolumen von ca. 47 kB (ca. 16%) übrig. Wenn man nun die Bildinformationen nur jeden zweiten Zyklus sendet, bleiben noch 20 Bilder pro Sekunde übrig und das Datenvolumen reduziert sich weiter auf etwa 23,5 kB (ca. 8%). Bei 20 Bildern pro Sekunde ist allerdings schon ein leichtes Ruckeln wahrnehmbar. Dennoch ist es nach Angaben des FAUST-Teams zufriedenstellend. Durch diese Maßnahmen lässt sich das Datenvolumen auf 8,2 MBit/s (ca. 7,5%) reduzieren.

Parametereinstellungen über das Web-Interface. Eine weitere Reduktion ist durch Verändern des Parameters *send_pic_every_x_cycles* (Tab. 4.1) im Web-Interface möglich. Dieser Parameter gibt an, in welchem Intervall, bzw. ob das Bild überhaupt, gesendet werden soll. Dies ist zum Beispiel in Räumen mit starker Funkbelastung, wie etwa während des CaroloCups, welche das WLAN-Signal stark stören kann, von Vorteil und oft auch notwendig. Ein solcher Parameter existiert auch jeweils für die Sensor-, Map- und Debugdaten. Diese

Parameter	Bedeutung	Standard-wert
send_pic_every_x_cycles	Das Kamerabild alle X Zyklen senden (0 = nie)	2
send_map_every_x_cycles	Die Mapdaten alle X Zyklen senden (0 = nie)	0
send_debug_every_x_cycles	Den Debugstring alle X Zyklen senden (0 = nie)	0
send_sensors_every_x_cycles	Die Sensordaten alle X Zyklen senden (0 = nie)	1
priority	Bestimmt die Position des Tasks in der Taskliste des Schedulers	15

Tabelle 4.1.: Liste der Parameter mit Erläuterung und Standardwert

Parameter gelten immer für den ganzen Datentyp. Das bedeutet, es ist nicht möglich, z.B. bei den Sensordaten einzelne Sensorwerte zu selektieren. Ein Grund dafür ist, dass das dadurch eingesparte Datenvolumen nur minimalst ins Gewicht fällt, da es sich pro Sensor nur um 4 Byte handelt. Ein weiterer Grund ist die Schnittstelle zum User-Interface. Dies wird zum einen Teil im späteren Verlauf dieses Kapitels und zum anderen Teil in Kapitel 4.2 genauer erläutert. Über den Parameter *priority* lässt sich einstellen an welcher Position der Task in der Taskliste des Schedulers eingefügt wird. So wird festgelegt wann der Task ausgeführt

wird. Idealerweise wählt man, wie bereits dargestellt, den Wert hier so, dass der Sender direkt hinter dem Task ausgeführt wird, dessen Datensatz man betrachten möchte.
Die in Tab. 4.1 gezeigten Standardeinstellungen für die Parameter bewirken, dass die Bilddaten alle zwei Zyklen, die Sensordaten jeden Zyklus und die Map- sowie die Debugdaten gar nicht übertragen werden. Die Sensor- und die Bilddaten sind laut FAUST-Team die Daten, die von den meisten Teammitgliedern benötigt werden. Die Mapdaten werden von den wenigsten Teammitgliedern benötigt und die Debugdaten sind jeweils, wenn verwendet, sehr individuell auf den Anwender zugeschnitten. Daraus und aus Gründen der Datenvolumenreduzierung ergeben sich die in Tab. 4.1 dargestellten Standardeinstellungen.

Übertragung der Daten. Für die Übertragung der Daten wird das UDP-Protokoll verwendet. Dadurch werden die Daten nicht automatisch neu versendet, falls Datagramme unterwegs verloren gehen. Durch das erneute Senden der Daten würde sich die Wiedergabe auf der Seite des User-Interfaces verzögern und der Anwender würde nicht die aktuellen Daten zu sehen bekommen. Außerdem ist der Overhead beim UDP-Protokoll durch den kleineren Header geringer als beim TCP-Protokoll. Dadurch wird die Bandbreite der WLAN-Verbindung nicht zusätzlich belastet.

Informationstyp	Information	Datentyp	Datenvolumen
Frame-Header			
	Zeitstempel	64-Bit Integer	8 Byte
	Inhaltscodierung	char/Byte	1 Byte
Sensordaten			
	Anzahl Werte	char/Byte	1 Byte
	Sensorwerte	float	21 x 4 Byte
Mapdaten			
	Anzahl Punkte	char/Byte	1 Byte
	Mappunkte	3 x 32-Bit Integer	50 x 12 Byte
Debugdaten			
	Anzahl Bytes	16-Bit Integer	2 Byte
	Debug Strings	100 x 1 char/Byte	10 x 100 Byte
		Gesamt	**1.697 Byte**

Tabelle 4.2.: Aufschlüsselung der Framegröße nach Header-, Sensor-, Debug- und Mapdaten

Die Bilddaten werden getrennt von den anderen Daten über einen eigenen Socket verschickt. Falls ein Datagram unterwegs verloren geht, sind nicht gleich alle Daten des Zykluses verloren, sondern nur ein Teil. Alle Datentypen jeweils einzeln zu verschicken ist jedoch nicht sinnvoll, da es zu einem unrentablen Overhead-Nutzlast-Verhältnis kommen würde.

Die Sensordaten umfassen pro Zyklus z.B. nur ein Datenvolumen von 84 Byte (Tab. 3.1) + 8 Byte für den Zeitstempel + 1 Byte für Hilfswerte, also ein Gesamtvolumen von 93 Byte. Der UDP-Header ist 8 Byte groß, der IP-Header 20 Byte und der Ethernet-Header 14 Byte. Das bedeutet es würden in diesem Fall auf 93 Byte Nutzdaten 42 Byte Overhead anfallen, also nur ca. 68% des übertragenen Datenvolumens wären Nutzdaten. Das ist immer der Fall, da worst-case und best-case hier identisch sind. Wenn man nun die anderen Datentypen, ausser den Bilddaten, mitsendet kommt man auf 1.697 Byte Nutzdaten (siehe Tab. 4.2). Somit währen ca. 97% des versendeten Volumens Nutzdaten, was deutlich effektiver ist. Dies ist natürlich wegen dem maximal möglichen Datenvolumen der best-case. Entscheidet sich der Anwender über die Parameter im Web-Interface einzustellen, dass er nur die Sensordaten übertragen haben möchte, wegen minimal möglichen Datenvolumen also der worst-case, so kommt man ebenfalls nur auf ca. 69% Nutzdatenanteil.

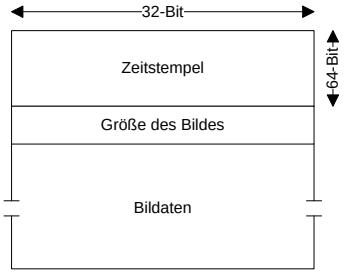


Abbildung 4.1.: Nutzdaten im Bildübertragungsframe

Aufbau der Frames / Schnittstelle zum User-Interface. Die beiden Frames sind wie folgt aufgebaut. Das Frame, welches die Bilddaten enthält (Abb.4.1), besteht aus einem 64-Bit Zeitstempel als eindeutigen Index, eine Größenangabe des Arrays welches die Bilddaten enthält und das Array mit den Bilddaten. Da Netzwerk-Pakete nicht unbedingt in der Reihenfolge empfangen werden in der sie versendet worden sind, dient der Zeitstempel auf der Empfängerseite, also auf der Seite des User-Interfaces, dazu, festzustellen ob das eben empfangene Paket älter ist als das zuvor empfangene und somit veraltet ist und verworfen werden kann. Da ein Array kein Terminierungszeichen besitzt, gibt die Größenangabe Aufschluss über die Größe des Bilddaten-Arrays. In dem Array befinden sich die Bildinformationen in keinem Bildformat, sie besitzen also keinen Format-Header. Vielmehr liegen die Daten im Rohformat vor. Pro Pixel wird ein Byte belegt, in welchem eine Graustufeninformation enthalten ist. Auf Seiten des User-Interfaces wird aus diesem Array dann ein Bild im Bitmap-Format(.bmp) erstellt. Wie dies durchgeführt wird, steht in Kapitel 4.2.
Wird eine neue Kamera mit abweichender Auflösung verbaut, so ist die *region of interest (roi)* so einzustellen, dass das IPL-Image, von den Parametern her, dem ursprünglichen IPL-Image entspricht. Ist dies nicht der Fall, muss der Prozess der Verkleinerung des Bildes individuell auf die neuen Parameter angepasst werden. Dabei ist, wegen des Bitmap-Formats

auf Seiten des User-Interfaces, darauf zu achten, dass die Anzahl der Pixel pro Zeile durch vier teilbar ist. Außerdem sind auf Seite des User-Interfaces einige Änderungen nötig, die in Kapitel 4.2 genauer beschrieben werden.

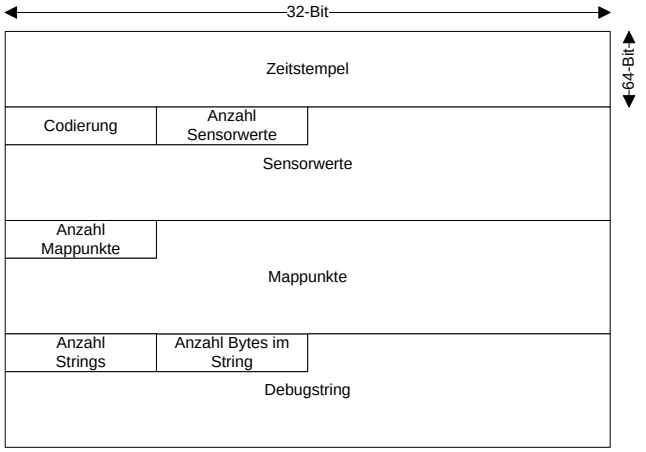


Abbildung 4.2.: Frame zur Datenübertragung

Das Frame, welches die anderen Daten enthält (Abb.4.2), besteht aus einem 64-Bit Zeitstempel, einer 8-Bit Inhaltscodierung, der Anzahl der Sensorwerte im Array, dem Array welches die Sensorwerte enthält, der Anzahl der Mappunkte, den Mappunkten, der Anzahl der Debugstrings, der Anzahl der Bytes im jeweiligen Debugstring und dem jeweiligen Debugstring. Der Zeitstempel erfüllt hier, genau wie im Bild-Frame, die Funktion eines eindeutigen Indexes.

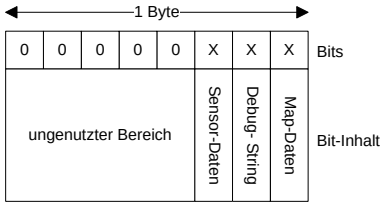


Abbildung 4.3.: Bit-Codierung des Codierungsbytes im Datenframe(siehe Abb.4.2)

In der Inhaltscodierung (Abb.4.3) ist festgelegt welche Daten in dem Paket enthalten sind. Dies wird über eine sogenannte „One-Hot“-Codierung festgelegt. Das bedeutet jedes Bit repräsentiert einen Datentyp (Sensor-, Map-, Debugdaten). Ist das Bit auf 0 gesetzt ist der Datentyp nicht im Paket enthalten, ist das Bit auf 1 gesetzt ist der Datentyp enthalten.
Die Elemente, die jeweils die Anzahl der Sensorwerte, der Mappunkte oder die Anzahl der Bytes im Debugstring angeben, dienen dazu das Ende des jeweiligen Datentyps im Array

auf der Empfängerseite zu finden bzw. zu registrieren. Alternativ hätte man auch ein „Ende“-Label implementieren können. Insgesamt betrachtet nehmen sich beide Varianten nichts, was Speicherbedarf oder Komplexität angeht.
Grundsätzlich werden immer alle Sensorwerte im Array versendet. Dadurch können auch andere Anwender die gespeicherten Daten nutzen, ohne die Konfiguration des User-Interfaces zum Zeitpunkt der Speicherung kennen zu müssen.
Ist zukünftig ein weiterer Sensorwert zu senden, weil ein oder mehrere neue Sensoren verbaut wurden, so wird dieser an das Ende des Arrays angehängt in dem die Sensordaten übertragen werden. Außerdem muss natürlich auch der Zähler entsprechend erhöht werden, welcher angibt wie viele Werte in dem Array enthalten sind. Dadurch, dass neue Werte immer hinten an das Array angehängt werden, ist eine Änderung der Konfiguration auf Seite des User-Interfaces nur nötig, wenn der neue Wert auch in irgendeiner Form dargestellt werden soll. Es ist aber nicht Voraussetzung für die Ausführbarkeit des User-Interfaces. Aus diesem Grund ist eine Versionierung nicht notwendig.

Durchführung / Ergebnisse. Wie bereits dargestellt, wurde der Sender als Task in die Anwendung des FAUST-Teams integriert. Nötige Bestandteile die ein Task innerhalb dieser Anwendung haben muss sind ein Konstruktor, sowie ein Destruktor, eine execute-Funktion, eine reset-Funktion, sowie eine create-Funktion.

- Im Konstruktor werden die Werte für die Parameter gesetzt, die als Standard beim Programmstart im Web-Interface angezeigt werden sollen. Diese sind wie in Tabelle 4.1 gesetzt. Weiter werden im Konstruktor alle Member-Variablen des Sender-Tasks vorinitialisiert, um potentielle Fehlerquellen zu reduzieren.
- Der Destructor ruft die reset-Funktion auf, welche dann die Sockets frei gibt und zurücksetzt. Der Destruktor wird bei Beendigung der Anwendung aufgerufen, die reset-Funktion hingegen jedes mal wenn das Programm angehalten wird. Eine erneute Initialisierung der Member-Variablen ist in der reset-Funktion nicht nötig, da bei jedem Neustart des Programms der Konstruktor wieder aufgerufen wird.
- Die create-Funktion dient dazu den Task zu erstellen und ihn beim Scheduler zu registrieren, damit er in die Taskliste aufgenommen wird.
- Die execute-Funktion stellt das eigentliche Programm dar. Diese Funktion wird in jeden Zyklus vom Scheduler aufgerufen, um den Task auszuführen. Das UML-Diagramm in Abb. 4.4 zeigt den Ablauf dieser Funktion.

Um zu testen, ob der Task funktionsfähig ist und ob die gewählten Datentypen versendet werden, wurde ein weiterer Task entwickelt. Dieser Task hat den Zweck Testdaten zu

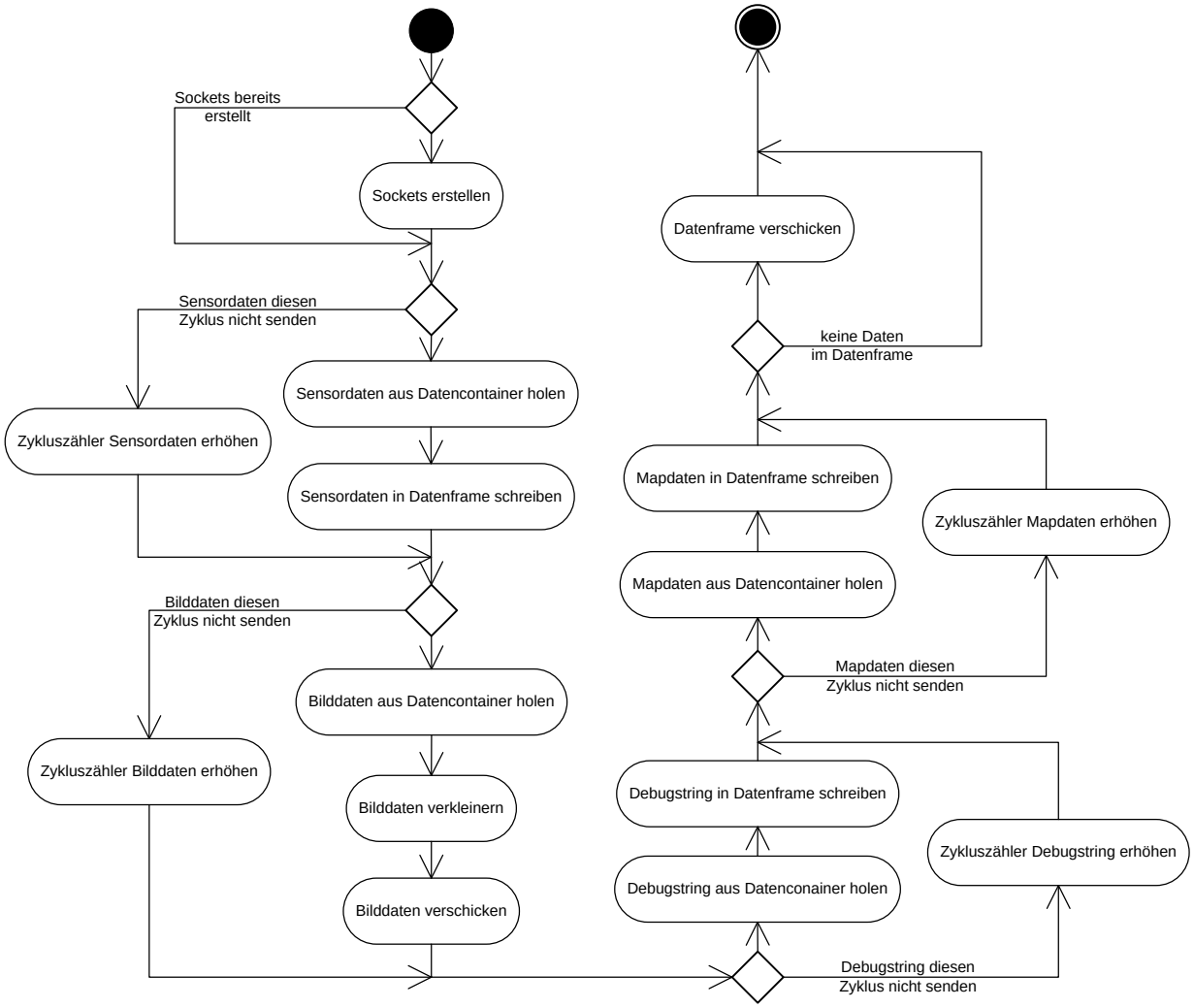


Abbildung 4.4.: UML-Aktivitätsdiagramm des Sende-Tasks

generieren, die den jeweiligen Datentypen entsprechen und mit diesen Daten die entsprechenden Datencontainer zu befüllen, damit der Sende-Task sich exakt so verhalten kann wie unter realen Bedingungen. Um die Kamerabilder zu simulieren wurden einige gespeicherte Kamerabilder verwendet. Der Sende-Task wurde mit den, in Tab. 4.1 zu sehenden, Standardeinstellungen konfiguriert. Die Daten wurden daraufhin über eine WLAN-Verbindung an einen anderen PC mit etwa 2 m Abstand versendet. Dort wurde mit einem Sniffer-Tool die WLAN-Karte auf die zu erwartenden Pakete untersucht und es wurde festgestellt das etwa 90% der Daten eintrafen. Ein positives Ergebnis.
Als nächstes wurde die Laufzeit des Tasks gemessen. Um dies zu tun wurden die genutzten CPU-Zyklen über 100.000 Durchläufe des Schedulers gemessen und anschließend daraus die durchschnittliche Laufzeit errechnet. Dafür wurden die Parameter im Web-Interface so gewählt, dass alle Daten jeden Zyklus gesendet werden. Dies stellt sowohl vom Datenvolumen, als auch von der Berechnungsdauer die maximale Last dar. Dabei kam eine durchschnittliche Laufzeit von ca. 7,8 Millisekunden heraus. Dieser Wert befindet sich schon relativ nah am, vom FAUST-Team vorgegebenen, Maximum, nach welchem sich die Laufzeit des Tasks sich im einstelligen Millisekundenbereich halten soll.
Außerdem wurde dieser Test noch einmal mit dem Standardeinstellungen, die in Tab. 4.1 zu sehen sind, durchgeführt. Dabei kam eine durchschnittliche Laufzeit von ca. 4,3 Millisekunden heraus. Da dies die wahrscheinlichsten Einstellungen sind, befindet sich der Task definitiv in den zeitlichen Vorgaben.
Eine weitere Reduktion der Laufzeit ist durch eine Erhöhung der Parameter *send_Datentyp_every_x_turns*, also durch eine Reduzierung der Sendehäufigkeit der Daten, zu erreichen.

4.2. Teilanwendung: User-Interface

An dieser Stelle wird die Entscheidung für das Design der Teilanwendung des User-Interfaces beschrieben und erläutert. Zu Anfang wird die Realisierung der Betriebssystemunabhängigkeit diskutiert. Anschließend wird erörtert welche Designvariante bezüglich der Flexibilität gewählt wurde und wie diese umgesetzt wurde. Daraufhin wird auf die Speicherung und Wiedergabe der Daten eingegangen und es wird erläutert welche Maßnahmen getroffen wurden, um den geforderten Zeitrahmen einzuhalten. Am Ende wird das User-Interface beschrieben und Ergebnisse dargelegt.

Betriebssystemunabhängigkeit. Um die Betriebssystemunabhängigkeit des User-Interfaces zu realisieren, wurde das grafische Framework Qt gewählt. Zusätzlich zu der, im Vergleich zu den anderen in Kapitel 3.3 genannten Frameworks, relativ großen Auswahl an standardmäßig vorhandenen Auswahl an Widgets, werden durch eine große Community viele weitere frei nutzbare Widgets zur Verfügung gestellt. Eine Analyse ergab, ist eine grafische Oberfläche mit Java Swing realisiert worden, so ist die Ausführungsgeschwindigkeit im Vergleich zu einer Implementation mit GTK+ oder Qt um bis zu 15% geringer. Da gemäß den Anforderungen die Anwendung in der Lage sein muss, alle Neuberechnungen für die grafische Oberfläche und die Aktualisierung dieser in maximal 25 ms abzuschließen, und noch genügend Spielraum für zukünftige Erweiterungen, wie z.B. neue Elemente vorhanden sein muss, schneidet Java Swing [Loy (2003)] in diesem Punkt am schlechtesten ab. Die übrigen Frameworks sind performancetechnisch potentiell etwa auf gleicher Höhe. Im Bereich Erweiterbarkeit ist LabView im Vergleich zu GTK+ und Qt relativ unflexibel was Erweiterbarkeit mit neuen Elementen angeht. Da eine einfache Erweiterbarkeit aber zu den Anforderungen gehört, schneidet LabView aus diesem Grund in Puncto Erweiterbarkeit am schlechtesten ab. Von den hier diskutierten Frameworks bieten mit Qt erzeugte Oberflächen für den Anwender das beste Look&Feel, da Anwendungen unter anderem die betriebssystemspezifische Optik besitzen. Wie in Tabelle 4.3 zusammenfassend zu sehen, ergibt sich damit, ein kleiner Vorsprung für Qt gegenüber GTK+ und den anderen Frameworks.
Da eine Anwendung die mit Hilfe von Qt und unter Benutzung der Qt eigenen Klassen geschrieben wurde, nicht für jedes Betriebssystem umgeschrieben, sondern der Code für das gewünschte System nur neu übersetzt werden muss, sind keine weiteren Maßnahmen zum Erreichen einer Betriebssystemunabhängigkeit nötig. Einzige Voraussetzungen sind, dass anstatt betriebssystemspezifischer Klassen und Bibliotheken die Qt eigenen genutzt werden und, dass das Zielsystem auch von Qt unterstützt wird, was im Rahmen dieser Arbeit der Fall ist.

Flexibilität. Um es zu ermöglichen das User-Interface nach den Wünschen des Anwenders anzupassen, werden XML-Konfigurations-Dateien verwendet. Diese werden beim

Kriterium	**Gewichtung** in %	**Java Swing**	**LabView**	**Qt**	**GTK+**
Portabilität	20	8	2	6	5
vorhandene	15	6	5	8	7
Performance	25	6	7	8	9
Netzwerkfähigkeit	25	9	9	9	9
Look&Feel	5	7	4	8	7
Erfahrung mit Framework/Editor	5	5	0	0	0
Erfahrung mit Sprache	5	5	7	7	7
Ergebnis	100	7,1	5,7	7,4	7,25

Bewertungsbereich von 0 - 10; 10 = Maximum

Tabelle 4.3.: Entscheidungstabelle Frameworks

Starten der Anwendung eingelesen und die grafische Oberflächen anhand der Angaben in den Dateien aufgebaut. Dadurch kann das User-Interface angepasst werden, ohne dass im Quellcode der Anwendung Änderungen vorgenommen werden müssen.
Die XML-Datei mit der die Elemente des User-Interface konfiguriert wird, ist wie in Abb. 4.5 aufgebaut.

XML Label	Element Type	Description	Description Orientation	X-Position	Y-Position	Position in Data-Array	Elementspecific Data

Abbildung 4.5.: Aufbau der XML-Konfigurationsdatei für die Elemente des User-Interfaces

Das *XML-Label* dient dem Parser zum Auffinden der Zeile.
Element Type gibt an, um welches Widget es sich handelt.
Description bietet die Möglichkeit ein Label mit dem in diesem Feld übergebenen Text als Beschreibung neben das Widget zu platzieren. Mit *Description Orientation* kann dann gewählt werden, ob dieses Label links oder rechts vom Widget platziert wird.
Die Felder *X-Position* und *Y-Position* legen die Position des Widgets fest.
Das Feld *Position in Data-Array*, ist nicht nur für die numerischen Widgets relevant. Zur Zeit sind die Sensorwerte allerdings der einzige Datentyp mit mehr als einer Position im Array. Aber die nicht numerischen Widgets haben für zukünftige Erweiterungen auch die Möglichkeit auf eine bestimmte Arrayposition verwiesen zu werden. Falls zukünftig einmal z.B. mehr als eine Karte oder ähnliches übertragen wird, kann somit auch bei diesen Widgets bestimmt werden, welchen Wert sie darstellen sollen. Zur Zeit steht an dieser Stelle aber stets eine 0 für die erste und einzige Position.

Das Feld *Elementspecific Data* unterscheidet sich bei einigen Widgets ein wenig. Die verschiedenen, aktuell genutzten, Varianten werden in Tabelle 4.4 dargestellt. Bei allen anderen bleibt dieses Feld zur Zeit leer.

Datentyp	XML-Feld	Beschreibung
Verlaufsgraph-Widget	GraphType	Gibt an, welcher Typ von Graph dargestellt werden soll. (siehe Manual im Anhang)
Control-Widget	SavePath	Speicherpfad für die Daten.
	BitmapHeader	Pfad des Bitmap-Headers.
Bild-Widget	Height	Gibt die Höhe des Bildelements an
	Width	Gibt die Breite des Bildelements an
Debug-Widget	Height	Gibt die Höhe der Debuganzeige an
	Width	Gibt die Breite der Debuganzeige an

Tabelle 4.4.: Aufschlüsselung der Elementspecific Data aus Abb. 4.5

Bei der Erstellung der XML-Datei hat der Anwender die Möglichkeit aus einer vorgegebenen Menge von Widgets zu wählen. In dieser Menge sind Widgets zur Darstellung von Text, numerischen Werten, sowie der Darstellung von Bildern vorhanden. Außerdem ein Element zur Steuerung der Oberfläche. Eine genaue Darstellung und Erläuterung zu den einzelnen Elementen ist im Manual im Anhang zu finden. Alle diese Elemente sind von der Basis-Klasse, welche in Abb. 4.6 zu sehen ist abgeleitet. Dadurch können alle Elemente, unabhängig von ihrer eigentlichen Klasse, auf dieselbe Weise, also generisch, aktualisiert werden.

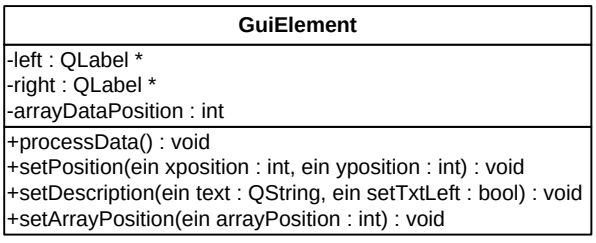


Abbildung 4.6.: UML-Klassendiagramm der Basis-Klasse der User-Interfaces Widgets

Ein Ausschnitt aus dieser XML-Datei könnte beispielsweise folgendermaßen aussehen:

```
...
< GuiElement ElementType="0" Text="" TextAlign=""
             XPos="0" YPos="200" SavePath="../save/"
             BitmapHeader="../ressources/bitmap.header" />
< GuiElement ElementType="3" Text="IR_Front_Left"
             TextAlign="1" XPos="0" YPos="150"
             ArrayValue="8" Type="0" />
< GuiElement ElementType="3" Text="IR_Front_Right"
             TextAlign="1" XPos="50" YPos="150"
             ArrayValue="9" Type="0" />
< GuiElement ElementType="2" Text="steering angle"
             TextAlign="0" XPos="100" YPos="150"
             ArrayValue="3" />
< GuiElement ElementType="1" Text="" TextAlign=""
             XPos="0" YPos="0" Height="160" Width="300"
             ArrayValue="0" />
...
```

Durch diesen Ausschnitt werden fünf Widgets angelegt.

- Ein Control-Panel angelegt, mit dem sich das User-Interface steuern lässt. Dem Widget werden der Speicherpfad und der Pfad an dem sich der Bitmap-Header befindet übergeben.
- Zwei Verlaufsgraphen. Beide vom Typ *Normaler Sensor*. Dieser Typ kann nur positive X- und Y-Werte darstellen. Die Beschreibung (*Description*) wird in beiden Fällen links vom Widget angelegt.
- Ein Widget zur Darstellung eines numerischen Wertes, in diesem Fall der Lenkwinkel. Hier wird die Beschreibung rechts vom Widget angelegt.
- Ein Element zur Darstellung des Kamerabildes mit einer Größe von 300 x 160 Pixeln.

Beim Start der Anwendung werden anhand der XML-Datei die entsprechenden Elemente erstellt und, gemäß ihres Datentyps, in die Liste der Bildanzeigeelemente oder die Liste der Datenanzeigeelemente eingefügt. Werden neue Daten empfangen (siehe Abb. 4.8) oder der nächste Datensatz zum erneuten Abspielen fällig (siehe Abb. 4.9), wird der Datencontainer im Speicher mit den neuen Daten aktualisiert und ein Signal emittiert, welches eine Aktualisierung der einzelnen Elemente initiiert. Während die Elemente aktualisiert werden, können so schon neue Daten empfangen oder bereitgestellt werden. Die Aktualisierung selbst erfolgt

folgendermaßen. Beide Elementlisten werden nacheinander durchiteriert und die Aktualisierungsfunktion *processData()* jedes Elements wird aufgerufen. In dieser Funktion holt sich das Element den Wert aus dem Datencontainer, der ihm in der XML-Datei zugewiesen wurde und ruft im Anschluss die Aktualisierungsfunktion von Qt *update()* auf. Daraufhin wird auf neue Werte gewartet.

Empfang der Daten. Um die Daten zu empfangen werden jeweils ein Socket für die Bilddaten und ein Socket für die restlichen Daten verwendet. An das *readyRead* Signal der Sockets sind die in Abbildung 4.8 dargestellten Slots gebunden. Im Slot zum Empfang der restlichen Daten (rechts im Bild) wird ein empfangenes Datagram auf seinen Inhalt analysiert und anschließend nach Datentypen zerteilt. Mit diesen Daten wird dann der Inhalt des Datencontainers aktualisiert. Daraufhin wird die Liste die die entsprechenden Widgets enthält durchgearbeitet und jedes Widget zieht sich seinen neuen Wert aus dem Datencontainer. Die Zuordnung zu einem Wert ist durch das Feld *ArrayPosition* in dem XML-File festgelegt, mit dem das User-Interface konfiguriert wird.
Der Slot zum Empfang der Bilddaten funktioniert prinzipiell genauso, nur ist der Datentyp hier ein anderer. Ein weiterer Unterschied ist, dass die Daten nicht direkt vom Datagram in den Datencontainer kopiert werden. Die Bilddaten kommen ohne ein Datei-Format, also als Rohdaten beim User-Interface an. Diese Rohdaten können von Qt so aber nicht dargestellt werden, also müssen die Daten in ein Datei-Format gewandelt werden. Hierfür bietet sich das Bitmap-Format wegen seiner sehr ähnlichen Struktur an. Anders als beim IPL-Format [Intel (1998)] werden beim Bitmap-Format die Zeilen von unten nach oben eingelesen. Darum muss zuerst die Zeilenreihenfolge invertiert werden. Anschließend wird der in Tab. 4.5 zu sehende Header vor die Daten gesetzt. Dieser Header liegt als Datei vor und wird beim Start der Anwendung eingelesen. Danach wird mit dem Bild so verfahren wie mit den anderen Daten auch. Das Bild im Datencontainer wird aktualisiert und eine Aktualisierung der Bild-Widgets wird initiiert.
Ändert sich z.B. durch die Verwendung einer neuen Kamera, die Auflösung der Bilddaten, müssen nur die Felder *width* und *height* des Bitmap-Headers entsprechend aktualisiert werden. Der Rest des Headers bleibt identisch, solange es sich bei den neuen Bildern nicht um Farbbilder handelt.

Speicherung und Wiedergabe. Die Daten werden nach Bilddaten und restliche Daten getrennt gespeichert. Dabei wird jedes Bild als eigene Bitmap-Datei gespeichert, die restlichen Daten hingegen komplett in eine Datei geschrieben, die dann mit den jeweils neuen Daten ergänzt wird. Zum einen wird dadurch verhindert, dass mittelfristig eine Datei entsteht, welche die maximale Dateigröße der lokalen Festplattenpartition übersteigt. Zum anderen würden die vielen kleinen Dateien, welche entstehen würden wenn man für die restlichen

Offset	Feld	Größe	Inhalt	verwendeter Wert in Hex
0000h	Identifier	2 Byte	Identifiziert den Bitmaptyp	42 4d
0002h	File size	4 Byte	Dateigröße in Bytes	b6 64 01 00
0006h	reserved	4 Byte	Reserviert	00 00 00 00
000ah	bitmap data offset	4 Byte	Gibt Startbit der Daten an	36 04 00 00
000eh	bitmap header size	4 Byte	Größe des Headers	28 00 00 00
0012h	width	4 Byte	Breite des Bildes in Pixeln	2c 01 00 00
0016h	height	4 Byte	Höhe des Bildes in Pixeln	a0 00 00 00
001ah	planes	2 Byte	Anzahl Farbebenen. Muss 1 sein	01 00
001ch	bits per pixel	2 Byte	Bits pro Pixel für Farbinformation	08 00
001eh	compression	4 Byte	Kompressionsmethode	00 00 00 00
0022h	bitmap data size	4 Byte	Größe der Bitmapdaten	00 00 00 00
0026h	hresolution	4 Byte	Breite in Pixel pro Meter	00 00 00 00
002ah	vresolution	4 Byte	Höhe in Pixel pro Meter	00 00 00 00
002eh	colors	4 Byte	Anzahl genutzter Farben	00 00 00 00
0032h	important colors	4 Byte	Anzahl wichtiger Farben	00 00 00 00
0036h	palette	n * 4 Byte	Farbpalette. 4 Byte pro Farbe	256 Graustufen

Tabelle 4.5.: Bitmap-Header

Daten pro Zeitstempel eine neue Datei anlegen würde, auf der Festplatte mehr Speicher belegen, da diese im Regelfall die Blockgröße der Festplattenpartition deutlich unterschreiten würden. Des weiteren wird das Öffnen und Ergänzen umso zeitintensiver je größer die betroffene Datei bereits ist. Demnach wird hier Rechenzeit gespart.
Die Bilddaten werden, wie eben bereits erwähnt, im .BMP Format, mit dem Zeitstempel als Dateinamen, gespeichert. Die anderen Daten werden, wie in Abb. 4.7 zu sehen, in einer .log Datei gespeichert. Der Zeitstempel erfüllt auch hier die Rolle eines eindeutigen Indizes für jeden Datensatz. Die Inhaltscodierung (Abb. 4.3) ist genau wie bei den Übertragungsframes aufgebaut und codiert. Die darauf folgenden Daten sind jeweils optional, wobei aber natürlich einer der Datentypen (Sensor-, Map-, Debugdaten) immer vorhanden ist, sonst wäre kein Eintrag vorhanden. Im unteren Drittel der Abbildung 4.8 ist der Speichervorgang dargestellt. Die Debugstrings werden als einzelner String gespeichert. Da die Strings später alle zur gleichen Zeit angezeigt werden, macht es für den Anwender optisch keinen Unterschied, ob die Strings einzeln nacheinander in das Widget geschoben werden oder alle auf einmal als einzelner String.

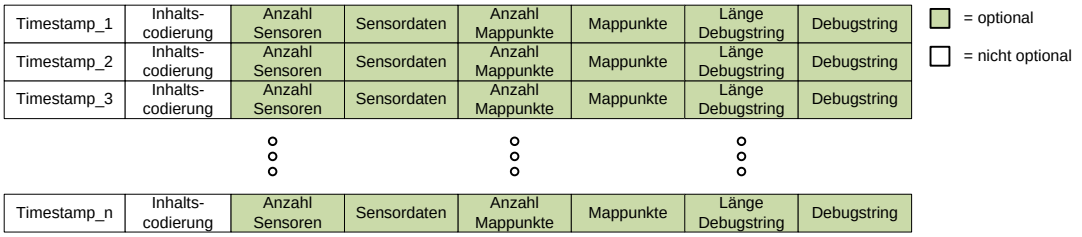


Abbildung 4.7.: Aufbau der Log-Datei

Beim Start des User-Interfaces werden von allen gespeicherten Daten die Zeitstempel eingelesen und in einer Liste gespeichert. Die Zeitstempel werden außerdem im Control-Widget in lesbarer Form, also als echte Zeitinformation, angezeigt, damit der Benutzer bei einem Abspielwunsch den Start- und den Endpunkt festlegen kann. Neben den Zeitstempeln werden auch die gespeicherten Map-, Sensor- und Debugdaten beim Anwendugsstart eingelesen und in einer Datenstruktur abgelegt. Dies wird gemacht, weil das Durchsuchen einer Datenstruktur im Speicher weniger Zeit in Anspruch nimmt, als das Durchsuchen einer Datei. Besonders wenn die Datei sehr groß wird, ist hier die Zeitersparnis zur Laufzeit wichtig, um die zeitlichen Rahmenbedingungen einhalten zu können. Deshalb werden neben dem Speichern auf Festplatte auch diese Datenstrukturen im Arbeitsspeicher aktualisiert bzw. ergänzt, falls das Speichern aktiviert wurde. So hat der Benutzer die Möglichkeit gerade gespeicherte Daten sofort wiederzugeben ohne die Anwendung neu starten zu müssen damit die neu verfügbaren Zeitstempel im Control-Widget sichtbar werden.

Für die Wiedergabe wird der „Signale und Slots“-Mechanismus von Qt [Blanchette und Summerfield (2009)] verwendet. Da nur die Bilddaten von Festplatte eingelesen werden und die restlichen Daten bereits bei Systemstart in den Speicher eingelesen wurden, ist der Zeitbedarf relativ gering. Bei Analysen stellte sich heraus, dass bei einer zweistufigen Pipeline, wie sie hier vorzufinden wäre (Daten laden - Anzeigen aktualisieren), durch eine Multi-Threading Implementation keine zeitlichen Vorteile erreicht werden. Daher wird hier auf Multi-Threading verzichtet.
Im Prinzip funktioniert „Signale und Slots“ wie ein Interrupt mit einer Interrupt Service Routine (ISR). Anstatt dass man ISRs an Interrupts bindet, bindet man Slots an Signale. Startet man einen Qt Timer (QTimer), so wird bei Ablauf anstatt eines Interrupts ein Signal emittiert. Dadurch werden dann nacheinander alle an das Signal gebundene Slots aufgerufen. Bei den Slots handelt es sich prinzipiell um Funktionen die einer ISR ähneln. Jedes Objekt in Qt kann sowohl Signale emitieren, als auch Slots besitzen, sofern es von QWidget abgeleitet ist.
In Abbildung 4.9 ist der Ablauf des Slots zu sehen, welcher durch das *timeout*-Signal des Timers aufgerufen wird. Die neue Timerzeit wird dabei aus der Differenz zwischen dem aktu-

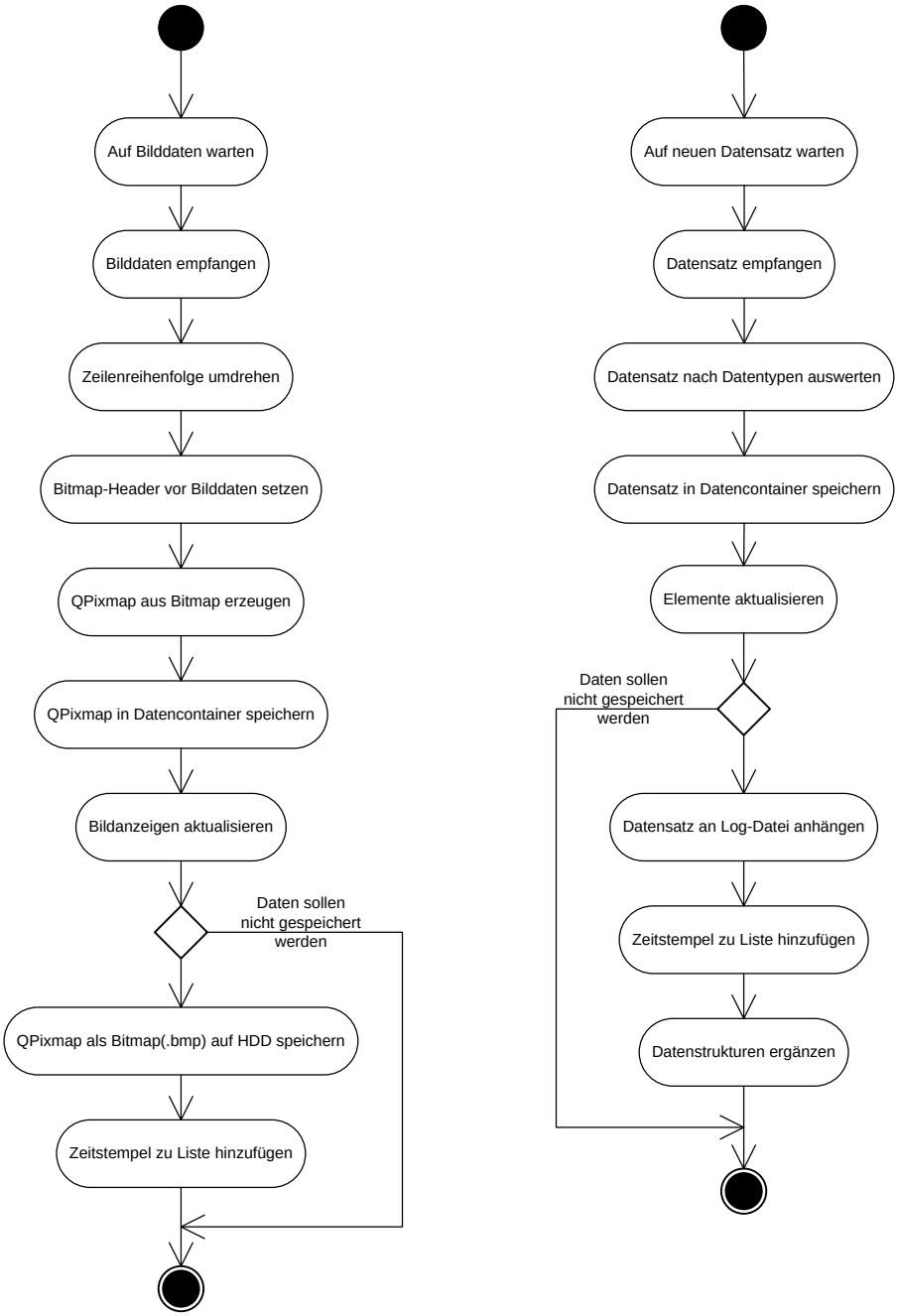


Abbildung 4.8.: Slots zum Datenempfang. Empfang der Bilddaten links, restliche Daten rechts.

ellen und dem nächsten Zeitstempel berechnet. Ist der aktuelle Zeitstempel der Letzte in der Liste, so wird der Timer pauschal mit 25ms gestartet und der Zeiger auf den aktuellen Zeitstempel wird auf das erste Element in der Liste gesetzt. Die Wiedergabe läuft also stets als Endlosschleife ab. So kann der Nutzer dem selektierten Bereich seine volle Aufmerksamkeit widmen, ohne sich um die Bedienung des User-Interface kümmern zu müssen.

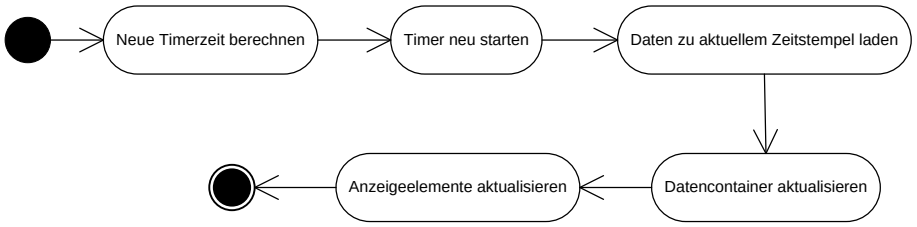


Abbildung 4.9.: UML-Aktivitätsdiagramm des Replay-Slots

Durchführung / Ergebnisse. Um die korrekte Funktion des User-Interface zu testen wurde ein Testprogramm geschrieben, das Daten aller Datentypen generiert und diese an das User-Interface verschickt. Zusätzlich wurde ein User-Interface konfiguriert auf dem alle Datentypen komplett vertreten sind. Außerdem wurde, wie beim Sende-Task, der Code für den Test so modifiziert, dass eine Zeitmessung erfolgen konnte. Die Zeiten werden in den jeweiligen Empfangsslots (Abb. 4.8), sowie im Wiedergabe-Slot (Abb. 4.9) gemessen. Die Messungen wurden über eine Dauer von 100.000 Zyklen durchgeführt, was bei dem Standard-Zyklus von 25 ms einer Dauer von etwa 40 Minuten entspricht.
Zuerst wurde das User-Interface gestartet, anschließend das Testprogramm. Dabei stellte sich heraus, dass das User-Interface alle gesendeten Daten darstellen konnte. Die empfangenen Daten wurden während diesem Testlauf mit der Speicherfunktion des User-Interface aufgezeichnet, um diese Funktionalität und im Anschluss die Wiedergabefunktion zu testen. Im Testlauf ohne Datenspeicherung ergab sich über 100.000 Zyklen im Bildempfangsslot eine durcgschnittliche Laufzeit von 6 ms und im Datenempfangsslot eine durchschnittliche Laufzeit von 5 ms. Bei aktivierter Speicherfunktion erhöhte sich der Durchschnittswert des Bildempfangsslots auf 13 ms und der des Datenempfangsslots auf 9 ms. Ein Ergebnis das noch genügend Spielraum für zukünftige Erweiterungen und Schwankungen durch äußere Einflüsse, wie z.B. eine hohe Festplattenlast, lässt.
Abschließend wurde die Funktionsfähigkeit der gesamten Anwendung unter realen Bedingungen getestet. Die FAUST-Anwendung, inklusive Sende-Task, wurde auf das Netbook überspielt und das Netbook mit dem Fahrzeug verbunden. Die Parameter des Sende-Tasks wurden so gesetzt, dass alle Datentypen übertragen wurden. Das User-Interface wurde auf

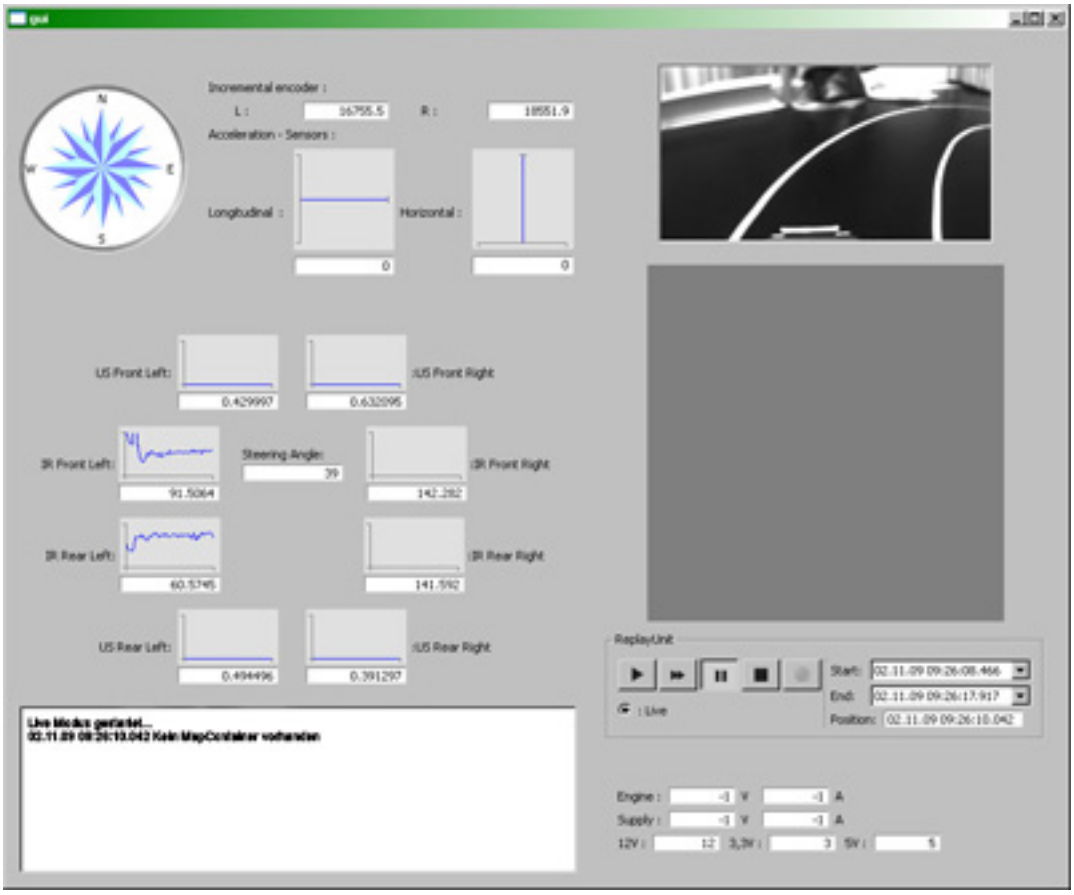


Abbildung 4.10.: Test User-Interface

dem in Kapitel 3.3 beschriebenen Laptop ausgeführt und wie in Abb. 4.10 zu sehen konfiguriert.
Der Test bestätigte die vorhergehenden Ergebnisse. Die Datenverluste lagen bei sich bewegendem Fahrzeug in etwa 3m Entfernung bei ca. 40%, was keinen akzeptablen Wert darstellt. Verbessern ließ sich dieser Wert durch die Parameter des Sende-Tasks. Durch Reduktion der Sendehäufigkeit der Bilddaten auf jeden zweiten Zyklus ließ sich der Datenverlust auf ca. 10% senken. Ein viel akzeptablerer Wert. Eine weitere Verbesserung dieses Wertes ließe sich durch die Verwendung eines Access Points erreichen. Da dieser im Gegensatz zur integrierten WLAN-Karte des verwendeten Laptops eine Antenne besitzt, ist dieser nicht so störanfällig. Der Laptop würde dann in diesem Fall über ein Kabel mit dem Access Point verbunden werden.

5. Fazit

In dieser Arbeit konnte erfolgreich ein Telemetriesystem für das autonome Modell-Fahrzeug des FAUST-Teams entwickelt werden. Ziel war es einen, vom Anwender frei wählbaren, Datensatz vom einem fahrenden Fahrzeug aus über eine WLAN-Verbindung an eine Empfangsstation zu verschicken. An dieser Empfangsstation sollten diese Datensätze visualisiert, gespeichert und zu einem späteren Zeitpunkt erneut abgespielt werden können. Im Laufe dieser Arbeit wurde ein Telemetriesystem entwickelt, das diese Anforderungen erfüllt. Die Hauptaufgabe dabei bestand in der nötigen Verringerung des Datenvolumens und in der Realisierung einer flexiblen Konfigurier- und Erweiterbarkeit.
Dieses System kann nun als Basis für zukünftige Erweiterungen verwendet werden.

5.1. Bewertung der Ergebnisse

Das Sammeln, Bearbeiten und Verschicken der Daten läuft erfolgreich. Einzige Ausnahme stellen die Mapdaten dar, da diese zum Zeitpunkt der Erstellung dieser Arbeit noch nicht abrufbar waren. Die Möglichkeit dazu ist vom Telemetriesystem her aber gegeben.
Das User-Interface (Empfangsstation) ist flexibel vom Anwender konfigurierbar und alle Datentypen (inkl. Mapdaten) können erfolgreich empfangen, visualisiert, gespeichert und zu einem späteren Zeitpunkt wiedergegeben werden. Die Erweiterbarkeit für zukünftige Neuerungen ist ebenfalls gegeben.
Auch die Anforderungen an die Laufzeit wurden erfolgreich umgesetzt. Mit einer durchschnittlichen Durchlaufzeit des Sende-Tasks, bei maximalem Datenaufkommen, von ca. 7,8 ms liegt dieser innerhalb der Anforderungen. Auch das User-Interface liegt mit im Schnitt ca. 22 ms im Soll.

Literaturverzeichnis

[Blanchette und Summerfield 2009] BLANCHETTE, Jasmin ; SUMMERFIELD, Mark: *C++ GUI Programmierung mit Qt 4: Die offizielle Einführung*. 2., aktualisierte Aufl. München : Addison Wesley, 2009 (Programmer's Choice). – ISBN 3-82-732729-6

[Carden u. a. 2002] CARDEN, Frank ; JEDLICKA, Russell P. ; HENRY, Robert: *Telemetry systems engineering*. Boston, Mass. : Artech House, 2002 (Artech House telecommunications library). – ISBN 1580532578

[Intel 1998] INTEL: *Intel Image Processing Library Reference Manual*. 1998. – URL `http://www.cs.nott.ac.uk/~jzg/nottsvision/old/ipldoc.pdf`

[Jamal und Hagestedt 2005] JAMAL, Rahman ; HAGESTEDT, Andre: *LabVIEW: Das Grundlagenbuch ; [bis Version LabVIEW 7.1]*. 4. Aufl., [Nachdr.]. München : Addison-Wesley, 2005 (Scientific Computing). – ISBN 3827320518

[Krause 2007] KRAUSE, Andrew: *Foundations of GTK+ Development*. Berkeley, CA : Andrew Krause, 2007 (Springer-12059 /Dig. Serial]). – ISBN 1590597931

[Loy 2003] LOY, Marc: *Java Swing: [covers Java 2 SDK 1.4 ; developing GUIs in Java]*. 2. ed. Beijing : O'Reilly, 2003. – ISBN 0-596-00408-6

[Maurer 2010] MAURER, Markus: *Carolo Cup*. 2010. – URL `http://www.carolo-cup.de/`. – Zugriffsdatum: 25.02.2010

[Pietzko 2009] PIETZKO, Stephan: *WLAN Standards*. 2009. – URL `http://wiki.uni-konstanz.de/wiki/bin/view/Wireless/WlanStandards`. – Zugriffsdatum: 02 Oct 2009 - 18:01

[Rechenberg u. a. 2006] RECHENBERG, Peter ; POMBERGER, Gustav ; PIRKLBAUER, Klaus: *Informatik-Handbuch*. 4., aktualisierte und erw. Aufl. München : Hanser, 2006. – ISBN 3446401857

[Roshan und Leary 2004] ROSHAN, Pejman ; LEARY, Jonathan: *802.11 wireless LAN fundamentals: [a practical guide to understanding, designing, and operating 802.11 WLANs]*. 2. print. Indianapolis, Ind. : Cisco Press, 2004. – ISBN 1587050773

[Stallings 2005] STALLINGS, William: *Operating systems: Internals and design principles*. 5. ed., internat. ed. Upper Saddle River, NJ : Pearson/Prentice Hall, 2005. – ISBN 0-13-127837-1

[Tanenbaum 2003] TANENBAUM, Andrew S.: *Computer networks*. 4. ed., internat. ed. Upper Saddle River, NJ : PH PTR Pearson Education Internat., 2003. – ISBN 0-13-038488-7

[Ward und Mellor 1991] WARD, Paul T. ; MELLOR, Stephen J.: *Strukturierte Systemanalyse von Echtzeit-Systemen*. Coed. München : Hanser [u.a.], 1991. – ISBN 3-446-16198-8

[Wörn und Brinkschulte 2005] WÖRN, Heinz ; BRINKSCHULTE, Uwe: *Echtzeitsysteme: Grundlagen, Funktionsweisen, Anwendungen ; mit 32 Tabellen*. Berlin : Springer, 2005 (eXamen.press). – ISBN 3-540-20588-8

[Zimmermann 1980] ZIMMERMANN, Hubert: *OSI Reference Model-The ISO Model of Architecture for Open Systems Interconnection*. 1980. – URL `http://www.comsoc.org/livepubs/50_journals/pdf/RightsManagement_eid=136833.pdf`

A. Bedienungsanleitung

Bedienungsanleitung

für das

FAUST Telemetriesystem

Inhaltsverzeichnis

1. Sende-Task

Dieses Kapitel beschäftigt sich mit dem Task, der zum Senden der Daten entwickelt wurde.

1.1 Was wird benötigt?

Benötigt wird:

- der Task selbst.
  - Zu finden unter: https://svn.cpt.haw-hamburg.de/svn/CaroloCup/Onyx/branches/przybilla/FAUSTplugins/Tasks/Gui
  - einzubinden ist der Ordner „Gui“ in FAUSTplugins/Tasks/
- der Datencontainer für die Debug-Strings
  - Zu finden unter: https://svn.cpt.haw-hamburg.de/svn/CaroloCup/Onyx/branches/przybilla/FAUSTplugins/Data/
  - benötigt wird die Datei *DebugStrings.h*

1.2 Funktion des Tasks

Der Zweck dieses Tasks ist es, die Werte der Sensoren, das Kamerabild (ggf. schon in bearbeiteter Form mit Markierungen o.ä.), Daten zur Kartenerstellung, sowie Strings zu Debuggingzwecken an eine Empfangsstation zu versenden.

1.2.1 Versenden von Daten

Um Daten zu versenden müssen diese in den entsprechenden Datencontainer gelegt werden. Folgende Datentypen können versendet werden:

- **Sensordaten**: Die Sensordaten werden von einem Treiber zyklisch im Datencontainer *SensorValues* abgelegt. Das Hinzufügen oder Verändern von Daten durch Tasks ist in diesem Fall nicht vorgesehen.
- **Bilddaten:** Die Bilddaten werden von einem Treiber zyklisch im Datencontainer *CameraImage* abgelegt. Diese können für eigene Zwecke verändert oder mit Markierungen versehen werden. Diese Markierungen sollten mindestens 2 Pixel hoch bzw. breit sein. Ist dies nicht der Fall besteht das Risiko, dass die Markierungen während des

nötigen Verkleinerungsprozesses verloren gehen. Außerdem sollte die Priorität des Tasks so gewählt werden, dass dieser nach dem Task ausgeführt, der die Bilddaten modifiziert hat. Wird die Auflösung des Bildes verändert, ist wie unter Punkt 1.3 beschrieben zu verfahren.

- **Debug-Strings:** Es besteht die Möglichkeit Strings z.B. zu Debugzwecken zu übertragen. Diese Strings werden über die Funktion *addString* des Datencontainers *DebugStrings* im selbigen abgelegt. Ein solcher String darf eine Länge von 100 Zeichen nicht überschreiten. Der Datencontainer fasst maximal 10 Strings gleichzeitig. Bei Überschreitung einer dieser Grenzen wird der String nicht in den Datencontainer eingefügt, sondern verworfen und *false* zurückgegeben.
- **Mapdaten:** Es besteht die Möglichkeit Punkte zur Kartenerstellung zu versenden. Pro Sendezyklus können maximal 50 Punkte versendet werden. Zu versendende Punkte sind im Datencontainer *MapData* abzulegen.

1.2.2 Parameter

Mit folgenden Parametern kann der Task über das Web-Interface konfiguriert werden:

Parameter	Beschreibung	Standardwert (Empfehlung)
priority	Gibt die Priorität an	15
send_pic_every_x_turns	Gibt an in welchem Intervall die Bilddaten versendet werden. Eine 2 bewirkt, dass die Bilddaten jeden 2. Zyklus gesendet werden. Eine 0 verhindert das Senden.	2
send_sensors_every_x_turns	Gibt an in welchem Intervall die Sensordaten versendet werden.	1
send_map_every_x_turns	Gibt an in welchem Intervall die Mapdaten versendet werden.	0
send_debug_every_x_turns	Gibt an in welchem Intervall die Debug-Strings versendet werden.	0

Außerdem können im Header *gui.h* die Zieladresse und beide Zielports für die jeweiligen Datentypen eingestellt werden.

1.3 Erweitern des Tasks

Soll der Task erweitert werden ist für die jeweiligen Datentypen folgendes zu tun:

- **Sensordaten:** Ist ein neuer Sensor verbaut worden und soll der Wert des neuen Sensors mit übertragen werden, sind einige kleine Änderungen im Code nötig. In der *execute*-Funktion des Tasks muss der Wert des neuen Sensors durch die entsprechende *getter*-Methode aus

dem Datencontainer geholt werden und hinter dem aktuell letzten Sensorwert in den Sende-Buffer eingefügt werden. Momentan ist dies der 12V-Wert (*getVoltageOf12Line*). Wie dies genau funktioniert kann man sich dort am Beispiel des 12V-Werts anschauen. Außerdem muss im Header *gui.h* das Define *NUM_SENS_VALUES* pro neuem Sensor um eins erhöht werden. Ist dies erledigt, wird der neue Wert zukünftig mit versendet.

- **Debug-Strings:** Hier ist es möglich die Grenzwerte des Datencontainers anzupassen. Folgende Änderungen sind im Header *DebugStrings.h* möglich. Die Anzahl der Strings die im Datencontainer abgelegt werden können wird durch *MAX_STRING_COUNT* festgelegt. Die maximale Länge eines Strings wird mit *MAX_STRING_SIZE* bestimmt.
- **Bilddaten:** Wird eine andere Kamera mit abweichender Auflösung verwendet, muss der Code ein wenig angepasst werden. Die Funktion *subSample*, welche für die Verkleinerung des Bildes zuständig ist, muss ein wenig modifiziert werden. In der ersten for-Schleife wird jede zweite Zeile und jede zweite Spalte des Bildes entfernt um die Größe auf ein Viertel zu reduzieren. Diese Schleife kann unverändert bleiben. In der zweiten for-Schleife werden bisher ungenutzte Randbereiche entfernt. Diese Schleife muss angepasst werden. Dabei ist darauf zu achten, dass die Anzahl Pixel pro Zeile durch vier teilbar ist, damit die Daten auf der Empfängerseite in das Bitmap-Format konvertiert werden können. Außerdem ist darauf zu achten, dass als Resultat maximal eine Bildgröße von 50kB herauskommt. Dies ist nötig um die WLAN-Verbindung nicht zu überlasten. Dieser Vorgang ist nur mit schwarz-weiß Bildern möglich. Farbbilder haben ein anderes Format (siehe IPL-Dokumentation).

2. User-Interface

Dieses Kapitel beschäftigt sich mit dem User-Interface. Das User-Interface dient dazu, empfangene Daten zu visualisieren, bei Bedarf abzuspeichern und zu einem späteren, beliebigen Zeitpunkt wieder abspielbar zu machen.

2.1 Was wird benötigt?

Für den Benutzer:

- Das User-Interface: https://svn.cpt.haw-hamburg.de/svn/CaroloCup/Telemetrie_GUI/trunk/compiled_binaries/
- Editor zum Erstellen und Bearbeiten von XML-Dateien
- Für Linux- und Mac-User: siehe *Für den Entwickler*

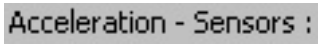
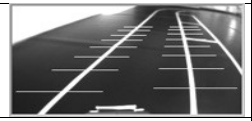
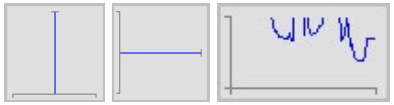
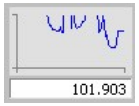
Für den Entwickler:

- Qt 4 Bibliotheken. Zu beziehen unter: http://qt.nokia.com/downloads
- gcc-Compiler. Z.B. MinGW unter Windows
- Qt Plugin für die bevorzugte Entwicklungsumgebung oder die Qt eigene Entwicklungsumgebung Qt-Creator
- Quellcode des User-Interface: https://svn.cpt.haw-hamburg.de/svn/CaroloCup/Telemetrie_GUI

2.2 Konfiguration / individuelle Gestaltung

Die grafische Oberfläche des User-Interface wird über eine XML-Datei erstellt. Diese Datei muss sich im Hauptverzeichnis der Anwendung befinden und *config.xml* genannt werden.

Bei der Konfiguration stehen dem Benutzer einige Widgets zur Auswahl. Welche Widgets das sind, welchen Zweck sie erfüllen, wie sie aussehen und wie man sie konfiguriert wird in der folgenden Tabelle dargestellt.

ID	Name	Bild	Funktionsbeschreibung	Beispielcode
0	ControlPanel		Mit dem ControlPanel lässt sich das User-Interface steuern. Modi: Live, Play, SlowMotion, Record, Stop	< GuiElement ElementType="0" Text="" TextAlign="" XPos="0" Ypos="200" SavePath="../save/" BitmapHeader="../ressources/bitmap.header" />
1	Label		Zum zusätzlichen Beschriften, z.B. von Widget-Gruppen.	< GuiElement ElementType="1" Text="Acceleration-Sensors :" TextAlign="" Xpos="0" Ypos="10" />
2	VideoPanel		Zum Anzeigen von Bilddaten.	< GuiElement ElementType="2" Text="" TextAlign="" Xpos="0" Ypos="10" Height="160" Width="300" ArrayValue="0" />
3	ValueField		Zur Anzeige eines numerischen Wertes.	< GuiElement ElementType="3" Text="12V :" TextAlign="" Xpos="0" Ypos="10" ArrayValue="20" />
4	Compass		Ein Kompass.	< GuiElement ElementType="4" Text="" TextAlign="" Xpos="0" Ypos="10" ArrayValue="16" />
5	HistoGraph noValueField		Zur Anzeige von numerischen Werten. Zeigt die letzten 80 Werte als Graph an.	< GuiElement ElementType="5" Text="" TextAlign="" Xpos="0" Ypos="10" ArrayValue="8" />
6	HistoGraph		Zur Anzeige von numerischen Werten. Zeigt die letzten 80 Werte als Graph + aktuellen als Zahl an.	< GuiElement ElementType="6" Text="" TextAlign="" Xpos="0" Ypos="10" ArrayValue="1" />
7	Map		Zur Anzeige von Kartendaten.	< GuiElement ElementType="7" Text="" TextAlign="" Xpos="0" Ypos="10" ArrayValue="0" />
8	DebugField		Zur Ausgabe von Debug-Strings.	< GuiElement ElementType="8" Text="" TextAlign="" Xpos="0" Ypos="10" Height="160" Width="300" ArrayValue="0" />

Bei jedem der Widgets kann über die Benutzung des *Text*-Feldes ein Label mit einer Beschreibung erzeugt werden wie es in der Tabelle beim ValueField (ID 3) zu sehen ist. Mit TextAlign kann bestimmt werden, ob dieses Label Rechts oder Links vom Widget platziert wird. Eine 1 bewirkt eine Platzierung links vom Widget, eine 0 bewirkt eine Platzierung rechts.

Mit dem *ArrayValue*-lässt sich ein Wert aus dem Datenarray auf das Widget zuweisen. Momentaner Inhalt dieses Arrays ist wie folgt:

Arrayfeld	**Sensor**
0	distance left
1	distance right
2	velocity
3	steering angle
4	us front left
5	us front right
6	us rear left
7	us rear right
8	ir front left
9	ir front right
10	ir rear left
11	ir rear right
12	engine battery current
13	engine battery voltage
14	supply battery current
15	supply battery voltage
16	compass angle
17	accelerometer value
18	3,3V supply
19	5V supply
20	12V supply

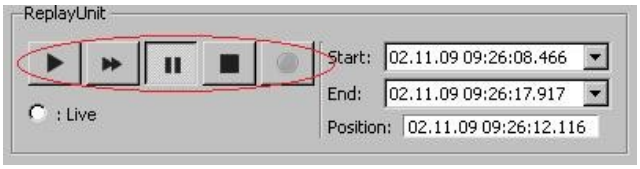
Beim ControlPanel existiert zusätzlich noch ein Feld für den Speicherpfad der Log-Dateien, also Bilddaten etc., und ein Feld für den Pfad des Bitmap-Headers. Dieser ist im Normalfall immer *„../ressources/bitmap.header“*. Der Bitmap-Header wird nur interessant, wenn eine neue Kamera mit abweichender Auflösung verbaut wird.

2.3 Bedienung des User-Interfaces

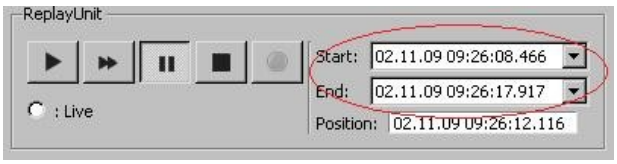
Das User-Interface wird über das ControlPanel bedient. Dort kann zwischen zwei Modi gewählt werden. Über den Live-Radio-Button kann in den Live-Modus geschaltet werden. In dem Modus werden die aktuell empfangenen Daten angezeigt.

Über die Aufnahme-Taste (ganz links) können die Daten gespeichert werden.

Weiter sind auf dem ControlPanel fünf Tasten vorhanden, wie man sie z.B von einem Videorekorder kennt. Die Tasten erfüllen auch dieselbe Funktion wie bei einem Videorekorder.



In den beiden Drop-Down Menüs *Start* und *End* werden alle zur Zeit verfügbaren Zeitmarken für gespeicherte Daten angezeigt. Wählt man dort einen Bereich aus und betätigt die Play-Taste, wird der ausgewählte Zeitbereich abgespielt.



2.4 Erweiterungen

Wird ein neuer Sensor verbaut und der Sensorwert, wie oben beschrieben an das Ende des Datenarrays angehängt, ist auf der Seite des User-Interface nur eine kleine Ergänzung zu tätigen. Damit die Daten des neuen Sensors auch angezeigt werden, muss in der XML-Datei nur ein Widget existieren, das auf die *ArrayPosition* verweist an der sich die neuen Werte befinden. Es muss also entweder eine neue Zeile hinzugefügt werden, oder eine bestehende Zeile im Feld *ArrayPosition* abgeändert werden.

2.4.1 Neues Widget

Soll ein neues Widget dem Pool hinzugefügt werden, sind folgende Schritte nötig:

- das neue Widget muss von der Basis-Klasse *BaseClass* und von *QWidget* abgeleitet werden
- die vererbten Funktionsprototypen der Basis-Klasse sind entsprechend zu überschreiben
- die switch-case Anweisung in der *createGui*-Methode der Klasse *CaroloGui* muss um einen entsprechenden case-Fall erweitert werden.

2.4.2 Neue Kamera / andere Auflösung des Bildes

Wird eine neue Kamera verbaut und ändert sich dadurch die Auflösung des gesendeten Bildmaterials, muss die Bitmap-Header Datei *bitmap.header* modifiziert werden. An Offset 0012h ist die neue Bildbreite in Pixeln einzutragen und an Offset 0016h ist die neue Bildhöhe in Pixeln einzutragen. Bei beiden Werten handelt es sich um 4 Byte große Hexadezimalwerte.

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 26. Februar 2010	
Ort, Datum	Unterschrift