RWTH Aachen
Institut für Luft- und Raumfahrtsysteme

Diese Arbeit wurde vorgelegt am Institut für Luft- und Raumfahrtsysteme

von

Colin Klein

Matrikel-Nr.: 368642

Bachelor-Thesis

Entwicklung und Integration einer Benutzeroberfläche zur Darstellung von Flugzustandsdaten im Flugversuch

Aachen, 23. März 2021

Wissenschaftliche Leitung:	Univ.-Prof. Dr.-Ing. Eike Stumpf
Betreuender Mitarbeiter:	M. Sc. Dominic Schröder
Externe Leitung:	Dipl.-Ing. Christina Pätzold

1. Prüfer:	Univ.-Prof. Dr.-Ing. Eike Stumpf
2. Prüfer:	Dr.-Ing. Ralf Hörnschemeyer

Zusammenfassung

Zur Erhöhung der Effizienz und Sicherheit im Flugversuch soll in dieser Arbeit ein Onboard-Messdatenvisualisierungswerkzeug entwickelt werden. Diese Arbeit setzt bei den Messanlagen der Dornier 228-101 D-CODE und dem Helikopter Bo105 D-HDDP des Deutschen Zentrums für Luft- und Raumfahrt (DLR) an mit dem Ziel, die im Flugversuch ermittelten Messdaten zu entschlüsseln und in Echtzeit darzustellen, um Flugversuchsingenieure in ihrer Arbeit zu unterstützen und eventuelle Sensorfehler durch fehlerhafte Messanlageneingänge rechtzeitig zu erkennen.

Diese Arbeit beginnt bei den Grundlagen der Netzwerktechnik und stellt die proprietären binären Formate vor, in denen die Daten verschlüsselt sind, um den Datentransfer nachvollziehen zu können. Darauf aufbauend werden die Grundlagen der Mensch-Maschine Schnittstelle benutzt, um, basierend auf Anforderungen der Flugversuchsingenieure, einen Entwurf für die letztliche Nutzeroberfläche zu entwerfen. Zuletzt werden der Entwicklungsprozess und die entstandenen Funktionen um dieses Ziel zu erreichen dargestellt. Auf Basis der in dieser Arbeit entwickelten Schnittstellen und Benutzeroberflächen werden zukünftig die Messdaten aus den Messdatenerfassungssystemen der D-CODE und der D-HDDP sowohl offline dekodiert als auch im Flug dem Flugversuchsingenieur dargestellt.

Schlagworte: *Bachelor-Thesis, ARINC-429, Human Machine Interface, xidml, pcap, python, pyQT, iNET-X*

Abstract

To elevate efficiency and safety in flight experiments this thesis develops an onboard-flightdata-visualizing-system. This thesis begins by trying to decode and understand the data streams of the DLRs Dornier 228-101 (D-CODE) and Bo-105 (D-HDDP) helicopter. By decoding and displaying the stream's data in real time clearly, the strain of Flight Test Engineers can be reduced. The instant display of flight data also comes with the advantage of detecting faulty sensors instantly, avoiding flights without measuring data.

This work begins by showing the basics of network technology by introducing proprietary binary formats in which flight condition data gets transmitted in order to understand the encoding properties. This is followed by presenting the basics of Human Machine Interfaces (HMI). Those basics as well as the demands of flight test engineers that later will be working with this interface are used to create a design for the final user display. Subsequently, the development process as well as the created functions to solve this work are shown. Ultimately, this work creates the basis for a user interface that has potential to display data streams of other aircraft of the DLR at a later time and is capable to replay past flights.

Keywords: *Bachelor-Thesis, ARINC-429, Human Machine Interface, xidml, pcap, python, pyQT, iNET-X*

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Symbolverzeichnis

Abkürzungen

ARINC	Aeronautical Radio Inc.
BCD	Binary Coded Decimals
BCU	Bus Control Unit
BNR	Binary Number Representation
CAN	Controller Area Network
CSV	Comma Separated Values
DLR	Deutsches Zentrum für Luft- und Raumfahrt
FTI	Flight Test Instrumentation
FVI	Flugversuchsingenieur
FX	Flight Experiments
GUI	Graphical User Interface
HMI	Human Machine Interface
ILR	Institute für Luft- und Raumfahrtsysteme
IP	Internet Protocol
MAC-Adresse	Media-Access-Control-Adresse
MMS	Mensch-Maschine Schnittstelle
OSI	Open Systems Interconnection Model
PCap Datei	Packet Capture Datei
PFD	Primary Flight Display
PTP	Precision Time Protocol
RWTH	Rheinisch-Westfälische Technische Hochschule zu Aachen
SDI	Source-/Data Identifier
SSM	Sign/-Statusmatrix
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

1 Einleitung

Das Deutsche Zentrum für Luft- und Raumfahrt (DLR) ist das international agierende Forschungszentrum Deutschlands, das neben der Luft- und Raumfahrt auch in den Bereichen Sicherheit und Digitalisierung, Energie und Verkehr forscht. Mit der, in Abbildung 1.1 dargestellten, größten Forschungsflugzeugflotte Europas fliegt das DLR grob 100 bis 150 Flugstunden pro Luftfahrzeug und Jahr, die jeweils eine dazugehörige Menge an Flugzustandsdaten generieren. Diese Flugversuche werden von der DLR-Einrichtung Flight Experiments (FX) durchgeführt, welche Flugexperimente und wissenschaftliche Kampagnen von DLR-internen, sowie externen Nutzern durchführt [1]. Die Flugmessanlagen und die generierten Flugzustandsdaten wurden lange Zeit von einzelnen Instituten des DLR betreut. Durch die Digitalisierung und durch die Big Data Bewegung wurde in der Einrichtung FX die Gruppe FTI (Flight Test Instrumentation) ins Leben gerufen. Ziel dieser neugegründeten Abteilung ist es unter Anderem, im Zuge von Big Data die Datenverwaltung zu optimieren und sich mit erweiterter Darstellung der Flugversuche zu beschäftigen, sowie die Messanlagen der Luftfahrzeuge primär zu verwalten.

Abbildung 1.1: Die DLR Forschungsluftfahrzeuge [1]

Diese Arbeit setzt bei der Arbeit von FTI an und beschäftigt sich mit der Messanlage der Luftfahrzeuge Bo-105 und der Dornier 228 D-CODE (Abbildung 1.2). Für diese Messanlagen ist eine Darstellung der Messdaten während des Flugs erforderlich. Dazu ist es nötig, die Formate des Messdatenstroms zu verstehen und sich mit einer ergonomischen Darstellung der Flugzustandsdaten in der herausfordernden Umgebung eines Flugversuchs zu beschäftigen.

(a) Dornier Do228-101 D-CODE

(b) Airbus Helicopters Bo105 D-HDDP

Abbildung 1.2: Die Forschungsluftfahrzeuge D-CODE und Bo105 des DLR

Die Flugzustandsdaten der Messanlagen werden nach aktuellem Stand während des Fluges durch die Messanlage gespeichert und nach dem Flug in einem tendenziell umständlichen Verfahren in das Format der Drittsoftware IMC Famos und im Anschluss ein CSV (Comma Separated Value) Datenformat konvertiert. In dieser Arbeit wird versucht, diese Kodierung zu verstehen, neu und gut dokumentiert zu gestalten, sodass zukünftig die Daten von der FTI eigenständig ohne externe Anwendungen wie IMC Famos und dauerhafte Lizenzkosten verwaltet werden können. Zudem wird versucht, eine Anwendung zu entwickeln, die das Auslesen der Daten während des Flugs in Echtzeit ermöglicht. Nennenswerte Vorteile sind weitaus größere Flexibilität und Optimierbarkeit gegenüber kommerziellen Programmen, die nur eingeschränkte Funktionalität bieten. Die letztlich aus dieser Arbeit folgende Offlineumsetzung bereits erflogener Daten ist ein Nebenprodukt, die primäre Zielsetzung dieser Arbeit ist die Möglichkeit, die Daten im Flug unabhängig von externer Software darzustellen. Durch mangelnde Zugänglichkeit der Luftfahrzeuge durch unvorhergesehene Wartungsarbeiten und das Coronavirus verzögert sich diese Umsetzung allerdings.

Diese Arbeit ist aufgrund des Umfangs der Datenverarbeitung und deren Darstellung in die zwei Themengebiete der Netzwerktechnik und Mensch-Maschine-Schnittstelle strukturiert. Fokus wird hierbei auf das Format ARINC-429 und im weiteren Verlauf auf die Erstellung einer ergonomischen Benutzeroberfläche gelegt.

2 Theorie

In diesem Abschnitt werden die wesentlichen theoretischen Grundlagen der Arbeit erläutert. Diese werden für den Parser, der den Datenstrom sinnvoll zerlegt, sowie die Erstellung der Nutzeroberfläche, benötigt. Inhalte sind sowohl Grundlagen der Netzwerktechnik, Vorstellung von Bitformaten, sowie der Transport von Daten in Netzwerken und darauf aufbauend Grundlagen der Mensch-Maschine Schnittstelle mit Hinblick auf Gestaltungsmöglichkeiten einer Benutzeroberfläche.

2.1 Grundlagen der Netzwerktechnik

Die binäre Kodierung ist der zentrale Ausgangspunkt für den Transfer der Netzwerkdaten. Diese binären Daten in Netzwerkleitungen werden durch eine analoge, kontinuierliche Veränderung der Spannung übertragen, welche durch Auswertung in eine digitale, diskrete, binäre Darstellung aus Nullen und Einsen überführt werden kann. [2]

2.1.1 Bits und Bytes

Die fundamentale Dateneinheit in der Informationstechnik sowohl für Speichervorgänge als auch für die Datenübertragung ist der Bit. Diese kleinste Einheit, die aus dem Lateinischen bi- für zwei entlehnt ist, kann die *zwei* Werte **0** und **1** annehmen. Sollen mehr als nur zwei Werte gespeichert werden, ergibt sich für jedes zusätzliche Bit eine exponentielle Menge an Möglichkeiten, siehe Gleichung 2.1. [3]

$$n_{Möglichkeiten} = 2^{n_{Bits}} \tag{2.1}$$

Aufbauend auf der Basiseinheit Bit wird zur Übertragung von komplexeren Zeichen oder höherwertigen Zahlenfolgen der Byte eingeführt, welcher aus 8 Bits besteht. Ein Byte ist in der Lage, 256 verschiedene Werte abzubilden (vgl. Gleichung 2.1) und reicht somit für Darstellung von Buchstaben und Zeichen aus, wie zum Beispiel mit der Textkodierung ASCII (American Standard Code for Information Interchange) und ist das kleinste Datenformat in vielen Rechnerarchitekturen. Zu der binären Darstellung, die ein Zahlensystem auf der Basis 2 aufbaut, werden in dieser Arbeit zudem das Oktalsystem (griechisches und lateinisches *Octo* für 8)

und das Hexadezimalsystem (griechisches *Hexa* für sechs und hexadezimal für 16) eingeführt. Beispielhaft werden diese Repräsentationen eines Wertes in Tabelle 2.1 gegenübergestellt. [4, 5]

Der Datentyp float setzt sich aus 4 Bytes, der Datentyp Double aus 8 Bytes zusammen. Floats können für Darstellung von bis zu 7 signifikanten Stellen benutzt werden. Falls eine genauere Darstellung nötig ist, können mit doubles (doublefloats) 15 Stellen repräsentiert werden. Beliebige Zeichen, wie Buchstaben und Wörter, werden aufeinanderfolgend als Strings (Ketten), sowie Character Strings (Zeichenketten) bezeichnet. Diese Strings bestehen in der Regel aus einer Aneinanderreihung von ASCII- oder ASCII-ähnlichen Zeichen. [6, 7]

Name	Basis	Bits	Wert
Binär	2	1	11000101
Oktal	8	3	305
Dezimal	10	4	197
Hexadezimal	16	4	C5

Tabelle 2.1: Darstellung eines Byte in verschiedenen Zahlensystemen.

Über diese Bits und Bytes werden Dateien über das Internet nicht in großen Dateien, sondern in einzelne Pakete aufgeteilt, verschickt. Auch die Messanlage überträgt ihre Daten in Bitformaten, auf welche in den weiteren Abschnitten genauer eingegangen wird. In der weiteren Arbeit werden bei Kurzschreibweise Bits als b und Bytes als B abgekürzt.

2.1.2 ISO-OSI Modell

Die Bits bilden die unterste, erste Ebene des OSI Modells. Das ISO/OSI (Open Systems Interconnection Model) Modell stellt die Netzwerkarchitektur als Schichtenarchitektur in Abbildung 2.1 dar. Das Modell wurde 1984 vom DIN-Normenausschuss Informationstechnik und Anwendungen [8] veröffentlicht und bezweckt, die Kommunikation über weitreichende technische Anwendungen zu beschreiben. Das Modell stellt die Architekturschichten aufbauend von der Bitebene auf.

In dieser ersten Ebene wird festgelegt, wie einzelne Bits durch Stromschwingungen vom Analogen ins Digitale übertragen werden. Die Sicherungsschicht (Schicht 2) besteht aus Netzwerkrahmen (auch Frames) (vgl. Kapitel 2.1.2), die mithilfe von MAC (Media Access Control)-Adressen Ziel und Empfänger definieren. Diese Adressen sind auf jedem internetfähigen Gerät hardwareseitig festgelegte Adressen, die in lokalen Netzwerken zur Identifikation des Geräts dienen. In Netzwerkframes wird die MAC-Adresse des nächsten Geräts in der Netzwerkhierarchie mitgeschickt. Wenn man z.B. von einem beliebigen Rechner die RWTH-Webseite aufruft, wird die MAC-Adresse des Absenders die RWTH Server nie erreichen, weil das Paket eingekapselt in Frames von Gerät zu Gerät geschickt wird und somit das Frame vom Rechner

nur den nächsten Router erreicht. Um die Anfrage aber dennoch an die RWTH Webseite zu adressieren, muss eine global zugeordnete Adresse existieren. Dies ist die IP (Internet Protokoll)-Adresse in der Vermittlungsschicht (Schicht 3). Ein Paket wird also mit einer IP-Adresse bestückt abgeschickt und dann via einzelner Geräte über MAC-Adressen weitergeleitet (über Router geführt). Weitere Abstraktion zu Ebene 4 führt in die Transportschicht, in der Transportprotokolle, wie UDP und TCP die Pakete an Ports (Programmzuweisungen) in den Endgeräten weiterleiten. Für diese vierte Schicht existieren Protokolle, die entweder für eine effiziente oder sichere Übertragung sorgen. Für sichere Übertragung mit mehrfacher Rückversicherung und möglichst fehlerfreier Übertragung eines Pakets existiert das Transmission Control Protocol (TCP). Für schnelle Übertragung, bei der Verluste verkraftbar sind, wird das simpler aufgebaute User Datagram Protocol (UDP) genutzt. [9]

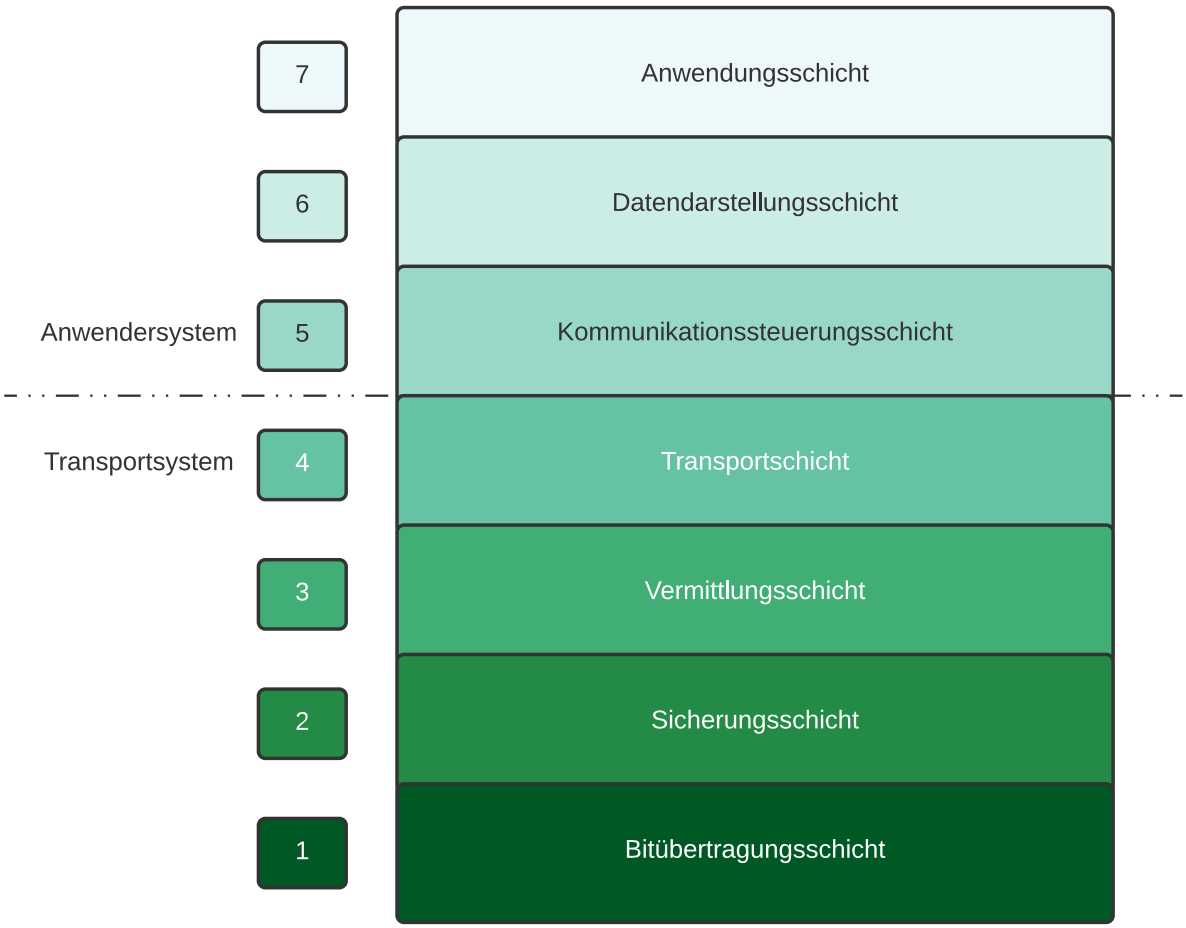

Abbildung 2.1: Das OSI Modell nach DIN ISO 7498 [3][8]

Die Schichten 5-7 des OSI-Modells nach Abbildung 2.1 beschäftigen sich mit der Datenverarbeitung in den jeweiligen Programmen, die diese Daten empfangen. Schicht 5 sorgt dafür, dass die Verbindung stabil ist und synchronisiert, sowie organisiert den Datenaustausch. Es werden z.B. Wiederherstellungspunkte erstellt, um nach Ausfall der Verbindung rasch wieder mit dem kommunizierenden Gerät synchron zu werden. Die Datendarstellungsschicht sorgt für eine Konvertierung der Daten in ein verarbeitbares Format. Verschlüsselung und Kompression gehören auch zu dieser Schicht. Die oberste, die Anwendungsschicht, Ebene Nummer 7

beschreibt den Fluss der Daten zum Anwender. Ein- und Ausgabe fließen durch diese Schicht. [3]

In dieser Arbeit werden Schichten 1-4 durch bereits existierende Bibliotheken geparst. Für Schichten fünf bis sieben werden Parsinganwendungen geschrieben, um das messanlagenspezifische Format zu entschlüsseln.

Frames

Daten in Netzwerken werden in einzelnen Blöcken, die als Frames (Rahmen) bezeichnet werden, zwischen Rechnern übertragen. Alle Frames besitzen die MAC-Adresse des Senders und die MAC-Adresse des Empfängers, welche 48 Bit lang ist, also 281 Billionen mögliche MAC-Adressen (vgl. Gleichung 2.1) darstellen kann. Das Kontingent an MAC-Adressen bietet also Wachstumspotenzial für künftigen Zuwachs an internetfähigen Geräten. Da die Darstellung mit Nullen und Einsen lange und unübersichtlich ist, werden MAC-Adressen in der Regel in Hexadezimalschreibweise dargestellt. Jedes Gerät, das einen Internetzugang hat, besitzt eine MAC-Adresse. [10]

Praktische Anwendung von Frames sind z.B. Frameprotokolle für die Raumfahrt [11], exotische Alternativen wie die von Waitzman [12] vorgestellte Methodik, sowie Ethernet Frames.

Ethernet Frame

Das Ethernet Frame ist der aktuell am weitesten verbreitete Framestandard und ist auch der Standard, in dem die Pakete aus der vorliegenden Messanlage abgeschickt werden. In Abbildung 2.2 aus der Dokumentation des Messanlagenherstellers ACRA Control [13] ist die Anwendung und die Segmentierung einer Netzwerkübertragung in OSI-Schichtenarchitektur dargestellt. Im Link Layer der Abbildung 2.2 sind die verschiedenen Ebenen von 2-7 dargestellt. Zunächst steht am Anfang des Pakets der MAC-Header (Schicht 2), um die direkte Verbindung zwischen zwei Geräten zu verwirklichen. Es folgt der IP Header (Schicht 3) für die globale Zuordnung, gefolgt vom UDP Header (Schicht 4), der Ports der jeweiligen Geräte enthält. Schicht 5-7 enthalten den App Header, die Metadaten, sowie die tatsächlich übertragenen Daten [13]. Die Zerlegung der einzelnen Frames erfolgt mithilfe eines sogenannten Parsers (parsen, vom lateinischen pars-aufteilen) [4]. Dieser segmentiert und liest die Frames bis zur Anwendungsschicht aus. Die schon im OSI-Modell erwähnten Schichten 3 und 4 sind auch in dieser Abbildung dargestellt. Die IP-Schicht enthält, wie vorherig schon erwähnt, die globale Zustellungsadresse und sorgt für den Transport von Netzwerkkarte zu Netzwerkkarte. In der Transportschicht von Abbildung 2.2 wird das UDP Protokoll verwendet. Dieses Protokoll findet bei z.B. Multimediaübertragungen Anwendung und wird auch bei der in dieser Arbeit vorliegenden Anwendung genutzt. [14]

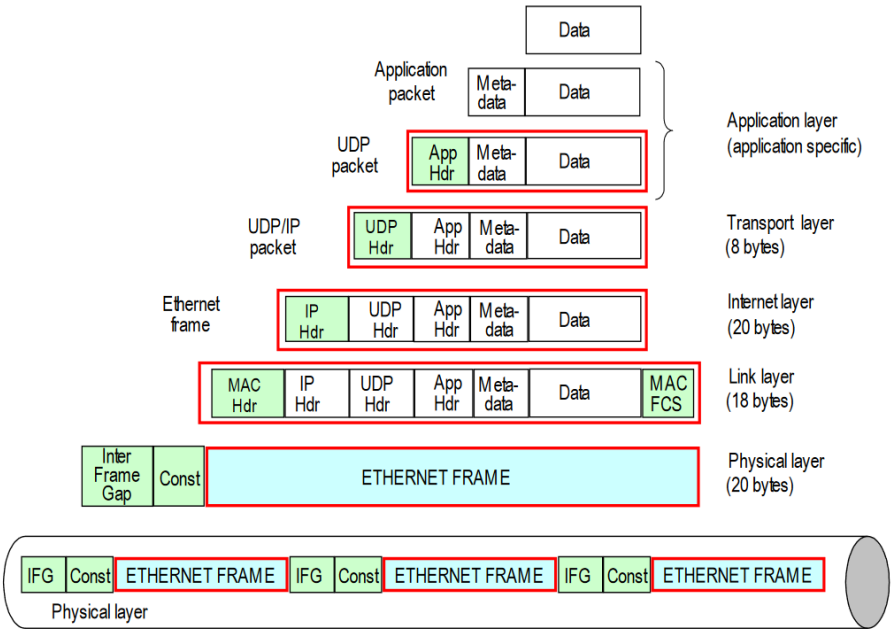

Abbildung 2.2: Kapselung bei Netzwerkübertragung gemäß der OSI-Architektur [13]

2.1.3 Das pcap Datenformat

Während des Fluges werden von der Messanlage die Messdaten über ein lokales Netzwerk ausgesendet und gespeichert. Diese Speicherung nutzt das pcap (packet capture) Dateiformat, dem Industriestandard für die Aufzeichnung von Netzwerkverkehr. Anwendungen für dieses Format sind vielfältig, eine davon wäre aber eine Netzwerkanalyse, bei der Netzwerkverkehr zunächst über einen Zeitraum aufgezeichnet wird und dann nach Absender oder ähnlichen Kriterien gefiltert werden kann. Das pcap Datenformat ist ein binäres Format, welches Netzwerkpakete aufzeichnet und danach, wie in Abbildung 2.3 dargestellt, verpackt. Eine pcap Datei beginnt hierbei stets mit Metadaten in Form eines globalen Dateikopfes, und jedes aufgezeichnete Netzwerkframe besitzt einen jeweiligen Kopf mit Metadaten des übertragenen Frames. Genauer Inhalt der beiden ist im Anhang in Tabelle A.1 dargestellt. [13]

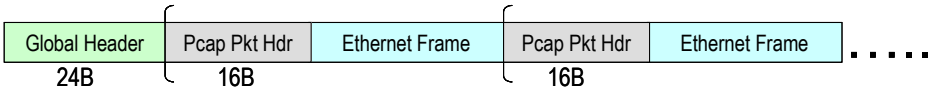

Abbildung 2.3: Packungsformat des pcap-formats

Zeitstempel in der Informationstechnik, die im weiteren Verlauf der Arbeit gebraucht werden, verwenden die Epochenzeit (Unix Epochenzeit). Diese zählt die Zeit seit dem 01.01.1970, 00:00 UTC in Sekunden und wurde von den Erstellern von Unix arbiträr festgelegt. Dieses Format wird in 32 Bit dargestellt und wird bis zum 19.01.2038 auf die 64 Bit Version aktualisiert werden müssen, da die 32 Bit Version nicht in der Lage ist, größere Zahlen darzustellen. Dieses Ereignis wird auch als Jahr 2038 Problem/Y2K38 bezeichnet, bei Umstellung auf 64 Bit wird dieses Format jedoch die Zeit bis zum Jahr 292 277 026 596 darstellen können. Das Format

der Zeit als natürliche Zahl ermöglicht leichte Arbeit in Anwendungen, da nicht mit unhandlichen Zeichenketten im Stunden:Minuten:Sekunden Format gearbeitet werden muss, sondern mathematische Operationen und Speicherung vereinfacht werden. Durch breite Unterstützung in der Informationstechnik existieren für dieses Format fest eingebaute Methoden, welche die Zeit in einem beliebigen Format ausgeben können. Das in dieser Arbeit verwendete, darauf aufbauende, PTP (Precision Time Protocol) ist ein Standard für die Uhrzeitsynchronität in Netzwerken. Das PTP wird in den vorliegenden Formaten als Epochenzeit übertragen. [14]

2.1.4 iNET-X

Da ein Überblick über allgemeine Netzwerkstruktur besteht, kann jetzt die Zuladung der Pakete, also Schicht 5-7 des OSI-Modells untersucht werden. Die vorliegende Messanlage wird vom Luftfahrtzulieferer Curtiss Wright hergestellt und funktioniert als Sammel- sowie Aufbereitungsstelle der Messdaten. So liest die Messanlage einzelne, skalare Variablenwerte aus und sammelt diese einzelnen Messwerte in einem Netzwerkpaket. Das Paket wird anschließend abgeschickt, sobald es entweder eine gewisse Größe erreicht hat oder eine gewisse Zeit verstrichen ist. Hierbei wird aber auch weiter differenziert, da die Messanlage Daten in verschiedenen Formaten empfängt. Die Konfiguration zur untersuchten Zeitperiode bestand aus drei verschiedenen Konfigurationen, in denen Messdaten verpackt wurden. Zunächst einmal in dem proprietären Format BCU (Bus Control Unit), im CAN (Controller Area Network)Format, welches aus der Automobilbranche stammt und drittens im ARINC-429 Format, einem Luftfahrtformat aus den 1980er Jahren. Die Zuladung beginnt zunächst, wie in Tabelle 2.4 dargestellt, nach dem UDP Header und enthält in Form des iNET-X Headers, also des iNET-X Kopfes, Metadaten. Das iNET-X Format ist ein von Curtiss Wright erstelltes, standardisiertes, proprietäres Format, welches durch jeweils andere *StreamID* (siehe Feld in Tabelle 2.4, Position 44) angibt, das von den Formaten BCU, CAN und ARINC-429 übertragen wird. Die Tabelle 2.4 stellt die genaue Bitstruktur eines Frames dar. Wie in der Spaltenüberschrift zu sehen, bestehen die Zeilen aus 4 Bytes bzw. 32 Bit. Die Zusammensetzung von iNET-X setzen sich aus den für die Arbeit relevanten Feldern: Stream ID, Timestamp und Payload zusammen, die in der Tabelle 2.4 in Byteposition 46-60 stehen. [14]

2.1.5 ARINC 429

Der ARINC-429 Standard, auch bekannt als *Mark 33 DITS Specification*, ist ein Übertragungsstandard, der von Aeronautical Radio Inc. (ARINC) veröffentlicht wurde. Im Standard wird die physikalische und elektrische Schnittstelle von einem Zwei-Draht Datenbus und dem zugehörigen Datenprotokoll definiert. ARINC-429 ist das am weitesten verbreitete Format für Datenbusse von Verkehrsflugzeugen. Genutzt wird dieses Format z.B. im Airbus A330 sowie in Boeing 747 und MD-11 als Datenbus für die Avionik.[15, 16]

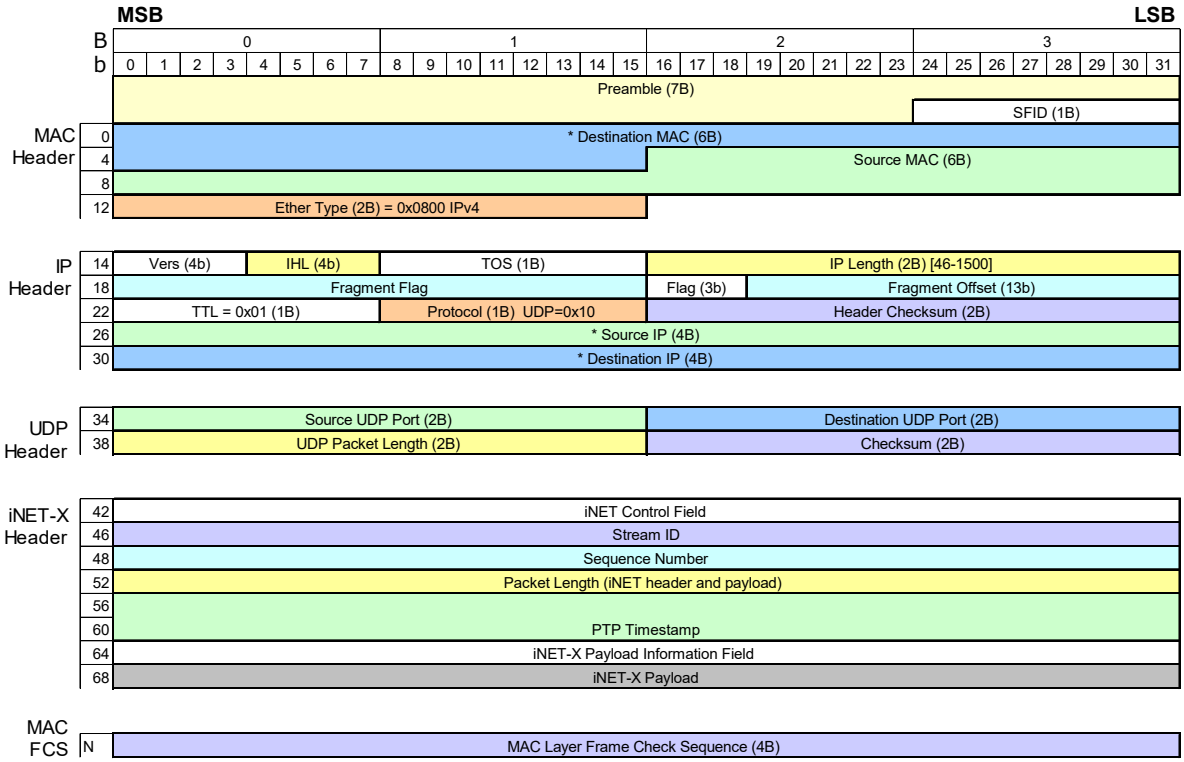

Abbildung 2.4: Bytestruktur eines Ethernet Frames mit iNET-X header [14]

Die Kapselung der ARINC-429 Daten vom iNET-X Header wird in Tabelle 2.2 dargestellt. Jede Nachricht übermittelt jeweils einen Datenwert und besteht aus 12 Bytes, die in Tabelle 2.2 in den mit *Msg#1* markierten Zeilen durch 3 Zeichenketten mit je 4 Bytes dargestellt werden. Das Feld *Elapsed Time* überträgt die Zeit, die seit Erstellung des Pakets in der Messanlage vergangen ist. Durch Addition mit dem iNET-X Zeitstempel kann so die genaue Zeit ermittelt werden, zu der der Messwert in der Messanlage angekommen ist. Die erste Zeile der Nachricht ist binär in einzelnen Feldern kodiert und übermittelt Werte wie Fehlerkodierungen. Der tatsächliche Messwert, der in der Abbildung als *ARINC-429 Traffic#1 (4B)* bezeichnet wird, steht in der dritten Zeile.

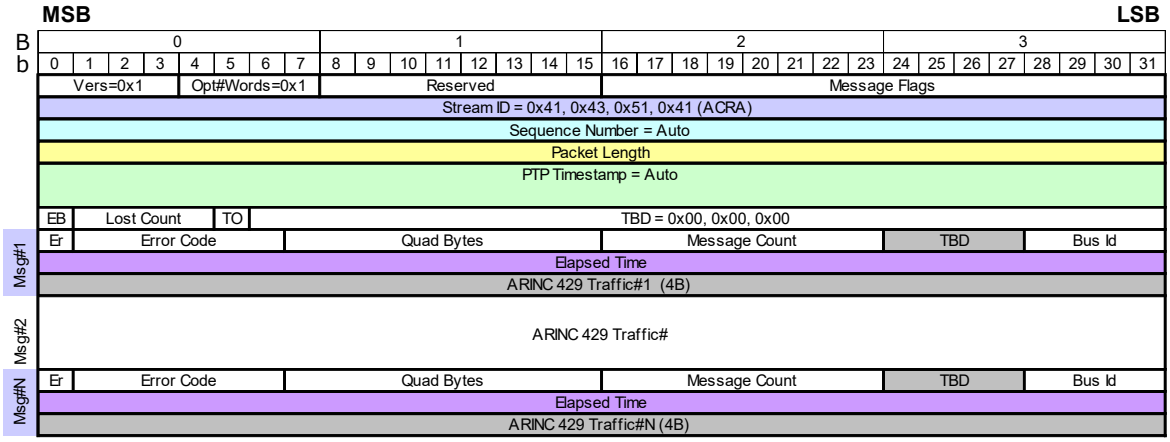

Tabelle 2.2: ARINC-429 Messages im iNET-X Format [14]

Die Zusammensetzung des ARINC-429 Traffics wird im kommenden Abschnitt erläutert und ist in Tabelle 2.3 dargestellt. Die Darstellung unterscheidet sich zu der von iNET-X in einigen wesentlichen Punkten: Der Index beginnt zunächst bei 1 anstelle von 0 und die Zählweise der Indizes startet, konträr zum Standard, von rechts nach links. Hintergrund dabei ist, dass bei ARINC-429 Bit 32 als erstes und Bit 1 als letztes übertragen wird. Die Zeichenkette muss also zum Auslesen durch Indizes zunächst umgekehrt werden. In Tabelle 2.3 sind zudem die verschiedenen möglichen Übertragungsformate untereinander dargestellt. [16]

<table>
<tr><td>32</td><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>09</td><td>08</td><td>07</td><td>06</td><td>05</td><td>04</td><td>03</td><td>02</td><td>01</td></tr>
<tr><td>P</td><td>SSM</td><td colspan="2">Vz</td><td colspan="8">MSB</td><td colspan="10">Data</td><td colspan="2">LSB</td><td>(SDI)</td><td colspan="8">Label</td></tr>
<tr><td>P</td><td>SSM</td><td colspan="4">Feld 1</td><td colspan="5">Feld 2</td><td colspan="5">Feld 3</td><td colspan="4">Feld 4</td><td colspan="4">Feld 5</td><td>(SDI)</td><td colspan="8">Label</td></tr>
<tr><td>P</td><td>SSM</td><td colspan="22">Discrete Data</td><td>(SDI)</td><td colspan="8">Label</td></tr>
</table>

Tabelle 2.3: Übertragungsreihenfolge des ARINC-429 Bus mit den Formaten (von oben) BNR, BCD und DISC

ARINC-429 besitzt drei verschiedene Formate zur Übertragung von Flugzustandsdaten. Zunächst die Formate BNR und BCD zur Darstellung von Zahlen und als Drittes zur Darstellung einzelner Bits, ähnlich den ersten 4 Bytes der ARINC-429 Message aus Tabelle 2.2, das Feld DISC. Die Binary Number Representation (BNR, Binäre numerische Darstellung) übertragt eine binär kodierte Zahl. Binary Coded Decimals (BCD, Binär Kodierte Dezimalstellen) ist ein Format, welches jede Stelle der übertragenen Zahl mit jeweils 4 Bit einzeln übermittelt. DISC (Diskrete Darstellung), ist ein einstellbares Datenfeld, welches ein im Vorhinein festgelegtes Datenformat überträgt. Auf die verschiedenen Felder aus Tabelle 2.3 wird im nachstehenden Abschnitt genauer eingegangen. [15]

Label

Das ARINC-429 Label ist ein 8 Bit langer Abschnitt. Dieser Abschnitt enthält eine dreistellige, oktalkodierte Nummer, welche die dargestellte Variable und den Datentyp identifiziert (vgl. Tabelle 2.4). Dabei gibt es für jede iNET-X StreamID ein dazugehöriges Set an Labels. Hierbei können allein Messwerte entschlüsselt werden, für die das Label bekannt ist. Die in Tabelle 2.3 dargestellten Bit 1-8 sind in Tabelle 2.4 in Detailansicht dargestellt, um die Konversion von binärer Darstellung in das dreistellige oktale Format zu verbildlichen. Die dreistellige Nummer wird durch eine Art BCD (also binäre Repräsentation der einzelnen Ziffern und nicht der gesamten Zahl) dargestellt. Zu beachten sind die Abkürzungen MSB und LSB. Diese stehen für Most Significant Bit (höchstwertigstes Bit) und Least Significant Bit (Niedrigstwertigstes Bit). Das LSB stellt die kleinste Stelle dar, während das MSB die größte darstellt. Grob vereinfacht ist bei der Zahl 524 das MSB 5 und das LSB 4.

Dies bedeutet, dass das gesamte Label umgekehrt werden muss, um als Zahl Sinn zu ergeben. Der beispielhafte Binärstring 011 011 11 resultiert nach Einsetzen der oktalen Werte der Tabelle in dem oktalen Wert 663 und ergibt nach Umkehren das Label 366. Das maximal darstellbare Label ist also 377, wenn die Bits 111 111 11 sind. Es lassen sich also $4 \cdot 8 \cdot 8 =$

	ARINC-Label Bit Struktur							
	LSB						MSB	
Bitposition	8	7	6	5	4	3	2	1
Oktaler Wert	1	2	4	1	2	4	1	2

Tabelle 2.4: Oktale Struktur des Labels

$2^8 = 256$ verschiedene Messwerte mit dem Label darstellen. Eine weitere Möglichkeit, um das Label auszulesen, ist es, die ARINC-429 Zeichenkette von Anfang an zu invertieren, da somit das Label nur eine reguläre Oktalkonversion durchlaufen muss. Als Beispiel ergibt die invertierte, oben genutzte, Abfolge 11 110 110 nach Konversion in Oktalschreibweise 366.

Source Data Identifier (SDI)

Dieses Feld an Stelle 9 und 10 kann bei mehreren Sendern und Empfängern genutzt werden, um das Gerät festzustellen, an welches die Nachricht gerichtet ist und von welchem Gerät die Nachricht gesendet wurde. Falls eine höhere Auflösung im BNR Zahlenfeld nötig ist, kann dieses Feld auch weggelassen werden und stattdessen auch Datenwerte übertragen.

Parity

Die Parität ist das letzte Element des ARINC-429 Traffics aus Tabelle 2.3 bzw. das zuerst übermittelte Bit (Index 32) und dient als Prüfsumme der übertragenen Daten. Die Parität beträgt entweder den Wert 0 oder 1, um dafür zu sorgen, dass die Summe aller Bits ein ungerader Wert ist. Ist die Quersumme gerade, ist von einer fehlerbehafteten Übertragung auszugehen.

Sign-/Statusmatrix (SSM)

Die Vorzeichen-/Statusmatrix (Index 30-31) ist in jeder ARINC Übertragung enthalten und enthält Fehlercodes und Vorzeichen. Bei BNR wird das Vorzeichenbit an Index 29 übertragen. Der Rest der Statusmatrix zeigt an, ob Fehler oder Funktionstest in der Übertragung vorliegen.

Die oben kurz erläuterten drei Datentypen, die der ARINC-429 Standard überträgt, werden nachfolgend genauer vorgestellt.

Datentyp BNR

In Tabelle 2.3 wird im Feld *Data* ein dezimaler Wert binär übertragen. Nach Konvertierung in Dezimalschreibweise kann dieser Wert mithilfe des mit dem Label verknüpften Skalierungsfaktors entsprechend skaliert werden und man erhält den vom Sensor gemessenen Wert. Wichtig zu beachten ist die Position des MSB und LSB, da die binäre Zeichenkette bei MSB auf der rechten Seite umgekehrt werden muss. Eine weitere Besonderheit ist auch das Vorzeichen des übertragenen Messwerts, welcher mit dem Bit 29 übertragen wird. Eine Null ist ein positiver-, eine 1 ein negativer Messwert.

Datentyp BCD

Die Übertragung Binär Kodierter Dezimalzahlen (Binary Coded Decimals) ist in Tabelle 2.5 abgebildet und beinhaltet 5 Ziffernfelder. Das erste Feld ist oktal repräsentiert. Die Felder 2-5 sind jeweils von 4 Bits (hexadezimal) dargestellt. Feld 1 stellt also nur die Werte von 0-7 dar. Falls die erste Stelle des übertragenen Messwerts größer als 7 ist, wird Feld 1 mit Nullen gefüllt und Feld 2 wird MSB. Die SSM stellt das Vorzeichen des Werts.

32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01
P	SSM		Feld 1			Feld 2				Feld 3				Feld 4				Feld 5				SDI		Label							

Tabelle 2.5: BCD Kodierung im ARINC-429 Bus

Datentyp DISC

Im Format Discrete (DISC) können speziell festgelegte Bitformate wie spezifische Gerätekonditionen übertragen werden. Auch ermöglicht die diskrete Übertragungsform individuelle Bits oder eine Mischung aus BNR und BCD Werten zu übertragen.

2.2 Grundlagen der Mensch-Maschine Schnittstelle

Die Entwicklung einer Mensch-Maschine Schnittstelle (MMS) stellt verschiedene Anforderungen an den Entwickler. Es ist nötig, den Kontext des Anwenders, die Aufgabe und auch das genutzte Werkzeug für die Entwicklung der MMS gut zu kennen.

Wandmacher [17] stellt die Interaktion als System nach Abbildung 2.5 vor. Dieses setzt sich aus Benutzer, Aufgabe und Werkzeug und Nutzungskontext zusammen, welche untereinander in Beziehung stehen. Ein Benutzer möchte mithilfe des Werkzeugs, einer Computeranwendung, die dargestellte Aufgabe lösen. Diese Aufgabe legt das Ziel des Benutzers fest. Das Computersystem stellt als Werkzeug verschiedene Mittel für die Lösung der Aufgabe bereit. Die Schwierigkeit der Aufgabe bestimmt sich aus der Passgenauigkeit der verwendeten Methoden auf die Anforderungen der Aufgabe, die Stärken und Schwächen des Nutzers allerdings auch Einfluss auf diese. Werden die Werkzeugeigenschaften verändert, ändert sich auch die Aufgabe aufgrund Anpassung der Lösungsschritte. Zuletzt stellt der Nutzungskontext, bzw. die Arbeitsumgebung einen einflussreichen Faktor dar, der die Qualität der Aufgabenbearbeitung beeinflusst. [18]

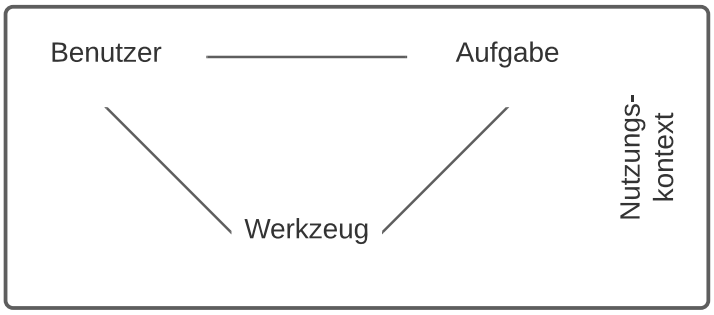

Abbildung 2.5: Schema eines Mensch-Maschine Systems

Im nachstehenden Abschnitt werden allgemeine Grundlagen zur Erstellung einer graphischen Benutzeroberfläche (GUI (Graphical User Interface)) zur Bedienung einer Maschine vorgestellt. Dazu zählen Überlegungen zur Übersichtlichkeit und Intuitivität, die die Bedienung in der besonderen Flugversuchsumgebung verbessern.

Murphys Gesetz wurde nach Bloch [19] vom amerikanischen Raketenforscher Edward Murphy nach einem Versuch formuliert, in dem an einem Raketenschlitten die Beschleunigung zur Sicherheit von 16 Sensoren gemessen werden sollte und alle von diesen Sensoren falsch in 90 Grad zur Fahrtrichtung ausgerichtet waren. In seiner Langfassung lautet das Zitat:

> "If there's more than one possible outcome of a job or task, and one of those outcomes will result in disaster or an undesirable consequence, then somebody will do it that way."

> „Wenn es mehrere Möglichkeiten gibt, eine Aufgabe zu erledigen, und eine davon in einer Katastrophe endet oder sonstwie unerwünschte Konsequenzen nach sich zieht, dann wird es jemand genau so machen."

Dieses zunächst zynische Zitat wurde jedoch wenige Zeit später als Sicherheitsstrategie bei der U.S. Air Force verwendet und bringt die menschliche Natur insofern auf den Punkt, dass es Verwendung redundanter Systeme empfiehlt. Butz und Krüger [18] interpretieren Murphys Gesetz in Hinblick auf die Mensch-Maschine Schnittstelle, dass alles was schiefgehen kann auch irgendwann mal schiefgehen wird und teilen die Implikationen in die drei Kategorien **Bedienfehler**, **Eingabefehler** und **Dimensionierung** auf. Insgesamt bedeutet dies, dass jede drei dieser Kategorien bei der Auslegung und Entwicklung jeder MMS berücksichtigt werden müssen.

2.2.1 Direkte Manipulation

Shneiderman [20] legt dar, dass direkte Manipulation der Elemente sinnvoll ist. Dazu gehören drei Prinzipen:

1. Sichtbarkeit aller Objekte und Aktionen
2. schnelle, umkehrbare, schrittweise ausführbare Aktionen
3. direkte, visuelle Manipulation der Objekte statt komplizierter Kommandos

Die Sichtbarkeit aller Objekte und Aktionen, also Vermeidung verschachtelter Menüführung in Form von flachen Hierarchieebenen bei der Menüführung und direkte und intuitive Eingabe ist ein wichtiges Attribut bei Erstellung jeglicher Anwendungen. Eine Implementierung schneller, umkehrbarer, schrittweise ausführbarer Aktionen projiziert auf die Anwendung eine Implementierung von Befehlen, die die letzte Aktion rückgängig machen (z.B. strg+z in Windows). Mächtige Befehle mit folgenschweren Auswirkungen wie Löschen von Elementen und Dateien sollten als solche gekennzeichnet sein. Im Zweifel soll ein Dialogfenster auftauchen, welches den Nutzer seine Aktion bestätigen lässt. Die direkte und visuelle Manipulation von Objekten soll wie im Windowsdatenexplorer direkt und visuell geschehen und nicht durch unübersichtliche Befehle unnötig verkompliziert werden. Auch wenn Kommandozeilen eine schnelle und effiziente Methode zur Eingabe sind, sind sie trotz schneller Ausführung und hoher Effektivität dennoch fehleranfällig, unintuitiv und mit einem erheblichen Lernaufwand verbunden. [18]

2.2.2 Kognitiver Durchgang einer Schnittstelle (Cognitive Walkthrough)

Der Cognitive Walkthrough setzt sich aus Fragen zusammen, aus deren Basis die Position des Nutzers eingenommen werden kann, um einzelne Funktionen einer MMS zu evaluieren und eine Ausführung zielgerichteter Handlungen möglich machen. Diese Fragen sind:

1. Identifizierung
2. Navigation
3. Rückmeldung

Die Identifizierung ist beispielhaft bei Speicherung einer Datei wichtig. Umsetzbar ist dies in einer klaren Benennung der Funktion und die Darstellung mit der Konvention eines Diskettensymbols, damit diese Aktion klar identifiziert werden kann. Weiterhin ist es wichtig, die Funktion für den Nutzer leicht zugänglich zu machen. Beim Speichersymbol ist dies durch eine vereinfachte Darstellung im Hauptdialogbildschirm der Benutzeroberfläche möglich. Zuletzt ist es sinnvoll, dem Benutzer nach Durchführung der Aktion ein angemessenes Feedback für seine Eingabeaktion zu geben. Im Bezug auf das Speichersymbol wäre dies ein Popup mit einer Meldung, dass die Datei abgespeichert ist. Die Berücksichtigung dieser Fragen als Leitfaden sorgen bei der Erstellung einer GUI dafür, dass obskure, unabsichtlich versteckte Funktionen vermieden werden und die Interaktion zwischen Benutzer und Maschine intuitiv gestaltet wird. [18]

2.2.3 Heuristische Evaluation

Nielsen [21] entwickelte die Methode der zehn Heuristiken zur Bewertung der Gebrauchstauglichkeit von Mensch Maschine Schnittstellen. Diese sind:

1. Sichtbarer System-Status
2. Übereinstimmung zwischen System und Realität
3. Möglichkeit der freien Navigation
4. Konsistenz und Standardisierung
5. Fehlervermeidung
6. Wiedererkennung geht vor Erinnerung
7. Flexibilität und Effizienz
8. Ästhetisches und minimalistisches Design

9. Information und Korrekturangebote bei Fehlern

10. Hilfe und Dokumentation

Zunächst ist es wichtig, dass der Nutzer vom System stets durch passende und rechtzeitige Rückmeldung über den Status des Systems und was im Hintergrund vor sich geht informiert werden sollte. Darüber hinaus spielt die Ähnlichkeit von Konventionen in System und Realität. Die Verwendung komplizierter Formulierungen oder missverständlicher Fachbegriffe sollte vermieden werden, um mögliche Missinterpretationen des Anwenders auszuschließen. Zusätzlich zur Einhaltung von Konventionen aus der realen Welt sollen Informationen zudem logisch und nachvollziehbar dargestellt werden. Systemfunktionen werden oft aus Versehen von Nutzern ausgewählt und deshalb sollte aus diesem unerwünschten Systemzustand ein erkenntlicher Ausweg vorliegen, z.B. in Form von Befehlen, die die letzte Aktion rückgängig machen (undo), bzw. diese erneut ausführen (redo). Konsistenz und Standardisierung sind für Nutzer wichtig, damit Klarheit herrscht, ob verschiedene Worte, Situationen oder Aktionen dasselbe bedeuten. Besser als Fehlermeldungen ist Fehlervermeidung. Dazu nötig ist eine aufmerksame Gestaltung der Schnittstelle, die Fehler gar nicht erst entstehen lässt. Bestätigungsdialoge sind vor potenziell inkorrekten Eingaben erforderlich und fehlerträchtige Situationen sollen gar nicht erst auftreten. Um die Belastung des Nutzers zu senken, können Objekte, Aktionen und Optionen deutlich und auffallend dargestellt werden. Informationen von einem Dialogschritt zum Nächsten sollen nicht im Kopf behalten werden müssen. Hilfen zur Bedienung sollen immer erkennbar oder wenigstens leicht aufzufinden sein. Durch Kurzbefehle und Tastenkombinationen können Experten Anwendungen schnell und effizient bedienen. Durch Nutzung von ihnen ermöglicht das System Neulingen sowie Experten eine effiziente Bedienung. Dialoge sollten zudem keine unnötigen Informationen beinhalten. Diese senken die Aufmerksamkeit des Nutzers für relevante Informationen und deren relative Sichtbarkeit. In Fehlermeldungen sollte das Problem einfach erklärt sein und Codes vermieden werden, um die Fehlerlösung für den Nutzer zu vereinfachen. Bestenfalls werden in der Fehlermeldung schon Lösungsmöglichkeiten benannt. Ein gut gestaltetes System sollte ohne Zunahme von Dokumentation bedienbar sein. Dennoch diese fallweise nötig, um die Nutzung eines Systems möglich zu machen. Hilfe und Dokumentation soll die Informationen effizient und übersichtlich darstellen und auf den Kontext und Aufgabe des Nutzers fokussiert, spezifische Bedienschritte nicht zu umfangreich darstellen. [18]

2.2.4 Farbgebung

Bei Auswahl der Farbgebung von Elementen einer Nutzeroberfläche ist es wichtig, dass das Farbschema so viele Farben wie nötig, aber nicht unnötig viele Farben enthält. Unterschiede im Farbton eignen sich dabei eher für die Unterscheidung von Kategorien. z.B. durch Varianz in Form von Helligkeit oder Sättigung. Diese können, um Wichtigkeit oder Aktualität darzustellen, eine begrenzte Anzahl von Werten auf einer kontinuierlichen Sättigungsachse annehmen. Durch Gruppierung von Elementen gleicher Helligkeit oder Sättigung wird logi-

sche Ähnlichkeit vermittelt (ausgegraute Elemente -> inaktiv). Für eine Signalwirkung helfen Kontraste zwischen unbunten (schwarz, weiß, grau) und bunten Farben. Itten [22] erklärt systematisch wie Farben, Kontraste und Farbklänge unsere Wahrnehmung beeinflussen. Farben werden vom Menschen anhand des Farbtons (von rot bis blau), Sättigung und Helligkeit (Grauwert) erkannt. Der aus diesen Werten entspringende HSV (Hue, Saturation, Value) Farbraum ist die mathematische Beschreibung dieser drei Dimensionen. Als kleiner Exkurs ist es außerdem sinnvoll, Farbdarstellungen in farbenblindheitskompatiblen Formaten zu erstellen, da von Farbenblindheit in der Bevölkerung über 8% der Männer und fast 1% der Frauen betroffen sind. Dies lässt sich zum Beispiel durch Änderung der Sättigung erzielen. Bei allgemeinen Anwendungen ist dies relevant, bei der vorliegenden jedoch vernachlässigbar, da Piloten und Flugversuchsingenieure durch medizinische Untersuchung davon ausgeschlossen sind.[18]

Im Luftfahrtkontext sind Hebel in Segelfliegern vereinheitlicht gekennzeichnet in Gelb für Ausklinkvorrichtung, Rot für Haubennotabwurf, Blau für Luftbremsen und Grün für Trimmung. In der allgemeinen Luftfahrt gilt zudem, dass die Farbe rot für Überschreitung des Betriebsbereichs gilt, gelb für Warnungen und grün für Aufhalten im Betriebsbereich steht. [23]

2.2.5 Menschliche Leistung

Bei Erstellung einer MMS ist es nicht nur wichtig, sich mit der Maschine zu beschäftigen, sondern auch den Menschen und seine Leistungsfähigkeit zu verstehen. Für die Interaktivität gilt laut Butz und Krüger [18], dass bei Reaktion des Programms innerhalb von 100ms die Ausgabe als direkt wahrgenommen wird. Liegt die Reaktion des Programms zwischen 100ms und einer Sekunde wird die Ausgabe des Programms noch immer als direkt mit der Eingabe verbunden. Reaktionszeiten des Programms lassen die Verknüpfung von Eingabe und Ausgabe zunehmend abnehmen bis der Nutzer keinen kausalen Zusammenhang zwischen Ein- und Ausgabe mehr feststellen kann. Lösung dafür sind "Lebenszeichen"des Programms in Form von Ladebalken und Popup-Dialogfenstern.

2.2.6 User Experience

Die User Experience (UX) steht für alle Erlebnisse und Eindrücke, die ein Benutzer bei Interaktion mit einem Gegenstand hat. Oft wird UX in der Softwareentwicklung, vor allem bei Webseiten- und Appdesign, benutzt, umfasst allerdings jede Art der digitalen und analogen Produktinteraktion. Nach den bereits vorgehenden Kapiteln fasst die UX alles unter einen Hut und versucht, die Arbeit des Nutzers zu optimieren. Dies erfolgt durch eine, von Feedback geprägte, Modellarchitektur. Nach Butz und Krüger [18] kann die Qualität eines Produktes in pragmatische und hedonische Qualitäten unterschieden werden:

Die pragmatischen Qualitäten sind Eigenschaften des Erlebnisses, die durch Benutzbarkeit (Usability) in Form von Effektivität, Effizienz und Zufriedenheit charakterisiert werden. Hedonische Qualitäten hingegen beschreiben Aspekte, die indirekt erfolgen. Beispielsweise Stimulation durch ein Produkt oder die Interaktion mit dem Produkt wie auch Identifikation mit dem Produkt.

Um ein bestimmtes Erlebnis hervorzurufen (z.B. hoher Informationsgrad und einfache Veränderbarkeit bei der Anwendung dieser Arbeit), muss dieses Erlebnis schon im Entwurfsprozess möglichst genau definiert sein und durch die Entwicklungsstadien hindurch konsistent beschrieben sein. Da es keine definierte Beschreibungssprache für UX gibt muss das Erlebnis anders festgehalten werden. Eine Möglichkeit ist es, Erlebnisse in Form von Geschichten zu fassen, die den Leser die beabsichtigten Gefühle und Bedürfnisse nachvollziehen lassen. Die Erstellung einer Geschichte, um das Erlebnis zu definieren, bietet zudem den Vorteil, dass alle am Prozess Beteiligten, auch wenn sie aus anderen Berufsgruppen stammen, gleichermaßen die Geschichte nachvollziehen können und kein Fachwissen über eine Beschreibungssprache brauchen. [18]

Evaluation der User Experience

Die Quantisierung von User Experience ist nicht mithilfe einer konkreten Messgröße erfassbar. Eine Möglichkeit, die Qualität einer Nutzerschnittstelle zu evaluieren, sind Interviews, in denen das Feedback ausgewählter Benutzer nach Nutzung der Schnittstelle festgehalten wird.

Diese Interviews lassen sich unterteilen in die Kategorien:

- unstrukturierten Interviews
- semistrukturierten Interviews
- strukturierten Interviews

Diese drei Kategorien können den gesamten Entwicklungsprozess begleiten.

Am Anfang der Entwicklung, wenn noch viel Freiraum in der Produktrichtung ist, macht es Sinn, ein unstrukturiertes Interview zu führen. Dieses setzt keinen festen Rahmen und festgelegte Fragen, sondern setzt sich lediglich auf einen Kontext fest, in dessen Richtung die Fragen orientiert sind. Semistrukturierte Interviews legen einige Fragestellungen im Vorhinein fest, sind aber in dem Sinne keine multiple choice, sondern Freitextfragen. Für eine Evaluation gegen Ende des Entwicklungsprozesses, wenn es um die Stellschrauben der Anwendung geht, sind strukturierte Interviews am nützlichsten. Es gibt z.B. den PANAS (Positive Negative Affect Schedule) Fragebogen, um Nutzererlebnisse darzustellen. Strukturierte Interviews sind in der Planung am aufwändigsten, bieten jedoch auch die größte Vergleichbarkeit. In diesem Interviewtyp werden allein im Vorhinein festgelegte Fragen gestellt. Dies ist vor Allem bei Vergleich von vielen Interviews vonnöten. Strukturierte Interviews sind von den Evaluations-

typen jedoch auch die aufwändigste Methode, was der Entwicklungszeit der Fragen und deren Funktionstests geschuldet ist.

3 Programmiertechnische Umsetzung

Kapitel 3 widmet sich der Entwicklung eines Programms zur Dekodierung des Messdatenstroms der in D-CODE und Bo-105 eingebauten Messanlage Kam 500, sowie des Entwicklungsprozesses der Nutzeroberfläche zur Ausgabe dieser Daten. Zunächst erfolgt eine Evaluation der Methoden, die für dieses Problem anwendbar sind. Darauf aufbauend befasst sich der zweite Teil mit der Lösung des Problems beginnend mit dem allgemeinen Ablauf der Dekodierungsfunktion und darauf der exemplarischen Dekodierung eines Netzwerkframes in einer pcap Datei von der Bitebene herauf. Der dritte Abschnitt beschreibt die Entwicklung der Nutzeroberfläche für die Darstellung dieser Daten.

3.1 Entschlüsseln des Netzwerkstroms

Um die Flugzustandsdaten aus der Messanlage zu dekodieren, soll zunächst evaluiert werden, welche Methode sich am besten dafür eignet. Idealerweise soll auch die Darstellung der Daten mit diesem Werkzeug realisiert werden.

3.1.1 Evaluation des Werkzeugs zum Parsing

Für eine einheitliche Lösung bieten sich Programmiersprachen gegenüber Anwendungen an, da Anwendungen weniger Flexibilität bieten. Nachfolgend werden Matlab, Javascript, sowie C++ und python untersucht. Die verschiedenen Anwendungen und Sprachen wurden in Tabelle 3.1 auf Basis verschiedener Punkte gegeneinander evaluiert und basierend auf dieser Evaluation bewertet. Zusätzlich zu den Programmiersprachen wird auch die Software Labview betrachtet. Labview wird aber nach initaler Betrachtung aufgrund mangelnder Flexibilität bezüglich Darstellungsmöglichkeiten verworfen. Eine der Kernanforderungen für die Programmiersprache ist das Parsen von pcap Datein. Für Matlab existieren derzeit für solche Vorhaben lediglich unoffizielle, ineffiziente von Matlabnutzern kreierte Lösungen. Das Lesen von pcap Dateien ist nicht offiziell unterstützt (Stand 24.02.2020: [24]). Zudem bietet Matlab eine lediglich wissenschaftlich orientierte Community und unterstützt keine Auslagerung von Bibliotheken.

Javascript (in OpenMCT genutzte Sprache) ist darstellungsorientiert und bietet sich eher für Frontend und Darstellungen an. Für den Umfang dieser Arbeit wird jedoch entschieden, eine

Kriterium	Labview	Matlab	OpenMCT/JS	python	C++
Kenntnisstand	0/5	4/5	0/5	2/5	2/5
Darstellungsoptionen	1/3	2/3	2/3	3/3	3/3
Open Source	nein	nein	ja	ja	ja
Geschwindigkeit	2/3	1/3	2/3	2/3	3/3
Syntaxkomplexität	1/3	2/3	2/3	1/3	3/3
Genereller Schwerpunkt	Daten-darstellung	numerische Analyse	Daten-darstellung	universell	universell
rapid Prototyping	3/3	2/3	2/3	3/3	1/3
source control	1/3	1/3	3/3	3/3	3/3
Objektorientierung	n/a	1/3	3/3	3/3	3/3
Community	wissen-schaftlich	wissen-schaftlich	wissen-schaftlich	breit	breit
Communitygröße	1/3	2/3	2/3	3/3	3/3
Syntax	2/3	2/3	1/3	3/3	1/3
Abkapselung	1/3	2/3	3/3	3/3	3/3
Niche vs. generisch	1/3	2/3	2/3	3/3	3/3

Tabelle 3.1: Vergleich der möglichen Werkzeuge

allgemeingültige Lösung zu finden und aus Gründen der Übersicht sich auf eine Programmiersprache zu beschränken. Aus denselben Gründen wird sich zunächst gegen openMCT entschieden.

Python bietet als wissenschaftlich oft genutzte Sprache mit der Numpy-Bibliothek die Möglichkeit, große Mengen an Daten effizient zu manipulieren, auszuwerten und abzuspeichern. Zudem hat python als weitreichend genutzte Programmiersprache viele Bibliotheken, eine große Community und dadurch auch gut dokumentierte Bibliotheken. pythons Syntax ermöglicht zudem auch Reduzierung des Programmieraufwands, da Logik mit vergleichsweise wenigen Zeilen dargestellt werden kann. Dies eignet sich vor allem für Rapid Prototyping, was sehr nützlich bei Neuentwicklungen wie z.B. dieser Arbeit ist. Allerdings muss durch pythons High-Level Architektur ein nicht zu vernachlässigender Mehraufwand betrieben werden, um eine C++ ähnliche Geschwindigkeit zu erreichen.

C++ zeichnet sich durch eine sehr schnelle Ausführung der Anwendung aus. Dies verlangt allerdings die explizite Programmierung von Variablen und Funktionsschnittstellen. Dies führt zu einer längeren Entwicklungszeit als die Programmierung derselben Funktion in beispielsweise python.

Zusammenfassend schneidet python von den verschiedenen Möglichkeiten am besten ab, da es für Arbeit mit Netzwerkdaten ausreichend gute Bibliotheken bietet und im Weiteren eine schnelle Entwicklung möglich macht. python ist zudem eine beim DLR genutzte Sprache und bietet diverse GUI Bibliotheken. Im kommenden Abschnitt wird python für die Dekodierung des Datenstroms, sowie die Programmierung der Nutzeroberfläche genutzt.

3.1.2 Ablauf des Programms

Der Ablauf der, in dieser Arbeit entwickelten Funktionen zur Entschlüsselung des Datenstroms, wird anschließend dargestellt. Zunächst wird die Struktur der höchsten Ebene für Auslesen der pcap Dateien und Ablage der entstandenen Dateien beschrieben. Darauf aufbauend wird die Entscheidungsstruktur für das Parsen des iNET-X Headers, also der reinen Zuladung der Netzwerkpakete, dargestellt und schließlich wird die Funktionslogik beim Auslesen einer ARINC-429 Nachricht beschrieben. Ursprünglich war eine Echtzeitdekodierung des

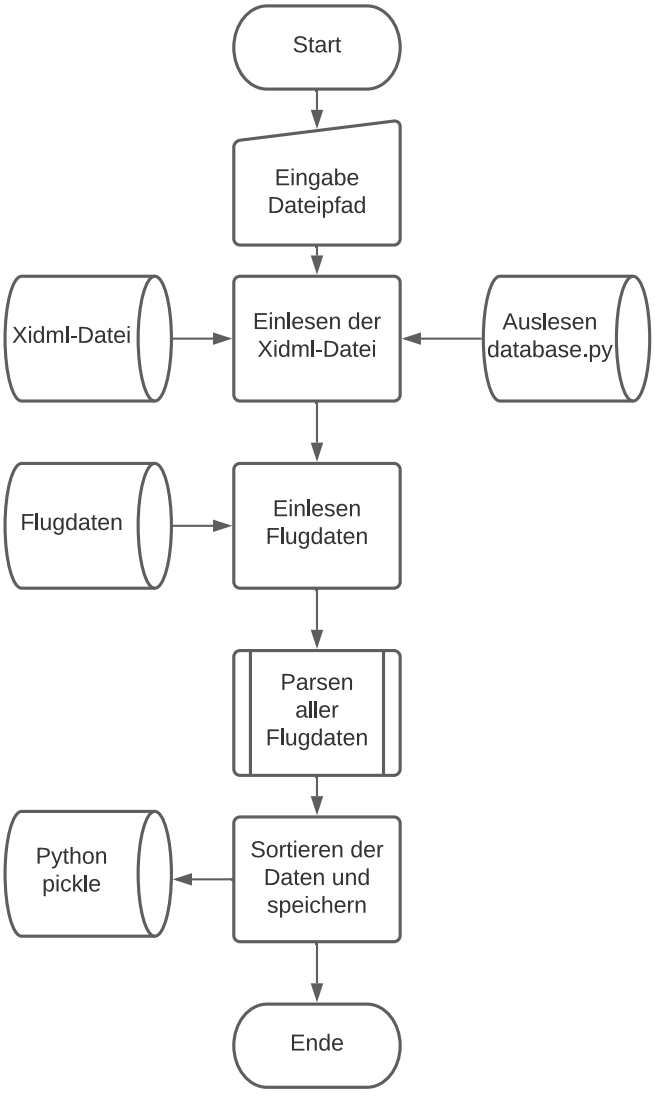

Abbildung 3.1: Ablauf des Konvertierens der im Flugversuch gemessenen Daten

Ethernetstreams vorgesehen. Dies kann jedoch im Rahmen dieser Arbeit durch mangelnde Zugänglichkeit der Messanlage durch unvorhergesehene Wartungen und das Coronavirus nicht realisiert werden und wird auf einen späteren Zeitpunkt verschoben.

In Abbildung 3.1 ist das Auslesen der pcap Dateien abgebildet, die von der Messanlage während des Flugs abgelegt werden. Das Flowchart beginnt mit der Eingabe des Dateipfads, in welchem sich die, meist mehreren, pcap Dateien eines Flugs befinden. Ausgehend davon wird die Messanlagenkonfigurationsdatei, die sich in einem .xidml Dateiformat befindet, ausgelesen. In diesem Format werden vor allem Infos wie Inhalt der Bus Control Unit (BCU) und StreamIDs festgelegt. Da diese Infos allein nicht ausreichen, wird basierend auf den StreamIDs der .xidml Datei aus einer weiteren, im Vorhinein feststehenden, Datei, die aktuell im python Wörterbuchformat ist, spezifischere Infos, wie Labelzuweisungen mit Variablennamen und Übertragungsformaten (z.B. ARINC-429 BNR) zu den StreamIDs ausgelesen. Diese beiden Dateien werden also zu einem gemeinsamen Konfigurationsformat fusioniert, welches im weiteren Verlauf für das Parsen genutzt wird. Sind die Konfigurationsdaten erstellt, wird

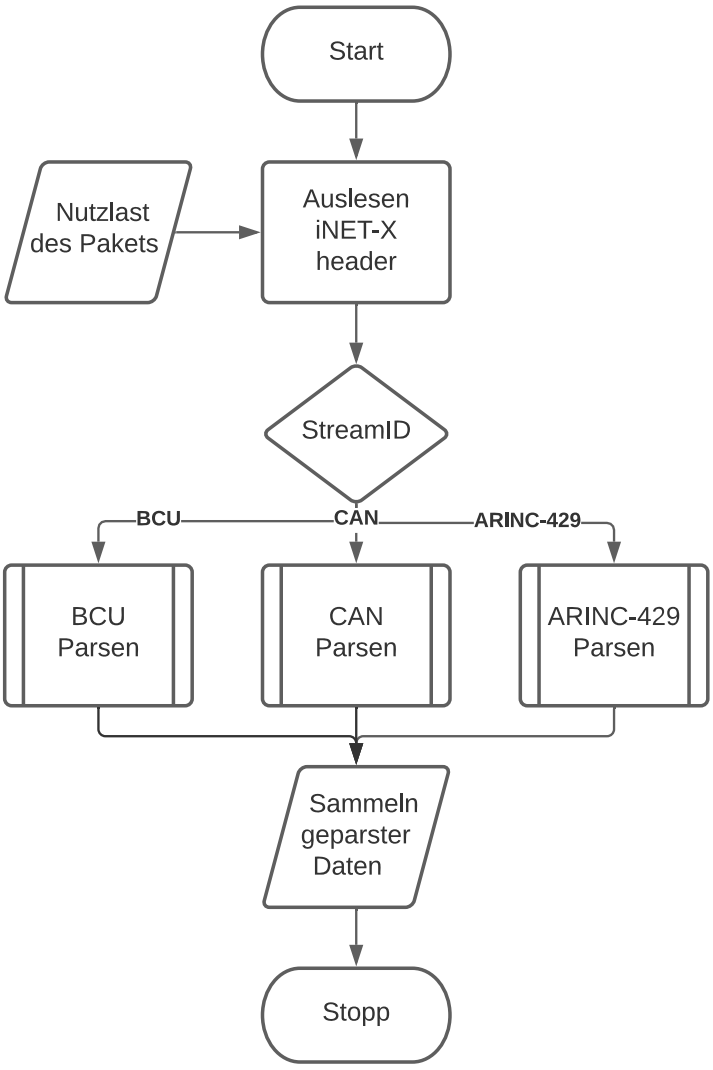

Abbildung 3.2: Algorithmus zum Parsen einer iNET-X Übertragung

durch die pcap Dateien iteriert. Für jede pcap Datei wird die Payload jedes Frames an eine Funktion zum Auslesen des iNET-X headers weitergegeben. Diese Funktion gibt python Listen mit Zeitstempel, Variablenname und Messwert zurück. Sind alle Messdaten aus pcap Dateien ausgelesen, werden diese gesammelt und in einem binären, pythonspezifischen *pickle* Format abgelegt.

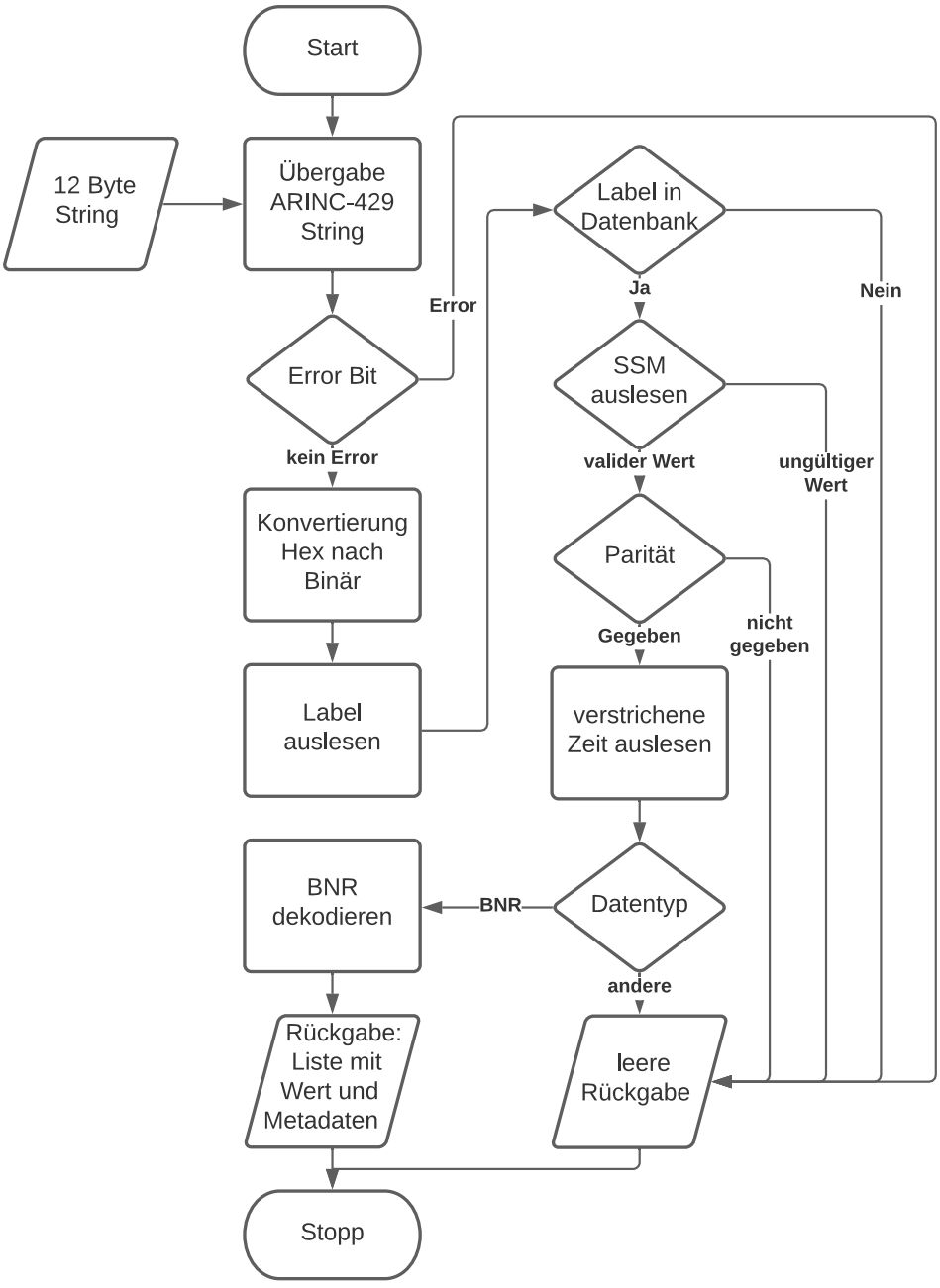

Abbildung 3.3: Ablauf des ARINC-Algorithmus

Die Zuladung des Netzwerkpakets wird im iNET-X Algorithmus in Abbildung 3.2 zunächst nach den in Kapitel 2.1.5 vorgestellten Bitmustern zerlegt, sodass die relevanten Felder Zeit und StreamID ausgelesen werden. Im Anschluss daran, wird die StreamID des Pakets mit den StreamIDs aus der .xidml Konfigurationsdatei abgeglichen und die dazugehörigen Konfigurationsdaten, sowie die verbliebene Nutzlast and Messdaten an die jeweilige Funktion weiterge-

geben. Nach dem Dekodieren in einer der Subroutinen ARINC, CAN oder BCU werden die ausgelesenen Daten in einer Liste gesammelt und an die Elternfunktion zurückgeschickt.

Funktionalität und Vorgehen bei Dekodieren des ARINC-429 Datenstroms werden in Abbildung 3.3 beschrieben. Zum ARINC-429 Algorithmus gehören viele leere Rückgaben, die aufgrund von vielen Fehlerindikatoren ausgelöst werden können. Zunächst aber wird der Funktion eine 12 Byte lange Zeichenkette, von der zunächst die erste Zeile auf das Error Bit überprüft wird, übergeben. Zeigt dieses Bit Fehler an, wird die Funktion umgehend beendet und gibt nichts zurück. Falls kein Fehler vorliegt, wird zunächst die empfangene Hexadezimaldarstellung für den ARINC-429 traffic in die Binärdarstellung konvertiert. Im Anschluss daran wird das oktale Label ausgelesen und, falls dieses nicht in den übertragenen Konfigurationsdaten gespeichert ist, wird die Funktion wie beim Error Bit auch umgehend beendet. Ist das Label jedoch bekannt, wird die Sign Status Matrix (SSM) auf Fehler überprüft und, falls es sich um den Datentyp BNR handelt, auch das Vorzeichen schon ausgelesen. Im Anschluss an die SSM wird im Zuge der Parität geprüft, ob die Prüfsumme aller Bits ungerade ist. Falls die Summe gerade ist, wird die Funktion beendet, falls sie ungerade ist wird der Zeitstempel ausgelesen und der Datentyp dekodiert und zusammen mit Variablenname und -Einheit an die Elternfunktion übertragen. In dieser Arbeit wird allein der BNR Datentyp dekodiert. Dieser Datentyp allein ist jedoch schon ausreichend, um 95% der Daten auszulesen. Zudem werden in den anderen ARINC-429 Datentypen weniger relevante Messdaten übertragen.

3.1.3 Dekodierung des Datenstroms mit python

Beispielhaft sei der Prozess der ARINC-429 Dekodierung erläutert. Dieser Abschnitt dient als detaillierte Ausführung zum vorherigen Kapitel 3.1.2 und beschreibt das Dekodieren einer ARINC-429 Nachricht und dessen iNET-X Header.

Pcap Auslese

Nach Abbildung 3.1 wird zunächst die Konfigurationsdatei der Messanlage eingelesen. Zunächst werden alle Pakete geöffnet und die Beladung an die iNET-X Funktion zum Parsen des Inhalts weitergegeben. In Abbildung 3.4 ist eine geöffnete pcap Datei eines D-CODE Fluges in dem Netzwerkanalysewerkzeug Wireshark geöffnet. Im oberen der drei Fenster befinden sich die empfangenen Frames, im mittleren Fenster die Voranalyse von Wireshark mit den verschiedenen OSI-Ebenen und im dritten Fenster der Inhalt des ersten Frames der pcap Datei. Im dritten Fenster ist aktuell die Zuladung des Pakets blau markiert. In diesem Bereich befinden sich die Flugdaten in Form des iNET-X Headers und der ARINC-429 Nachrichten, die im nachstehenden weiter behandelt werden.

Die blaue markierte Zuladung wird in Hexadezimalform an die iNET-X Funktion übergeben (siehe Abbildung 3.2). Die Paketzuladung und der dekodierte iNET-X Header wird in Tabelle 3.2 dargestellt. Die Darstellung liest die Daten entsprechend Tabelle 2.2 aus.

InfoLine:	11000000
StreamID:	ac000a17
Sequence Number:	00017431
Packet Length:	0000040c
PTP Seconds:	5e58e6d0
PTP Nanoseconds:	167c1715
Error Check:	00000000
ARINC-429 Msg#1	0003601100000000FFe649Fb
ARINC-429 Msg#2	0003611100057e40600085bb
	[...]
ARINC-429 Msg#N	0003b311025cdbc0FFF9197b

Tabelle 3.2: Dekodieren des iNET-X Headers eines Pakets der Kam-500 Messanlage

Erstens wird der iNET-X header abgetrennt und die Datenfelder werden dekodiert. Wichtig ist dabei der Zeitstempel und die StreamID. Die StreamID entscheidet, von welcher Quelle die Dateien stammen. In den untersuchten Dateien gab es beispielsweise 7 verschiedene StreamIDs. Drei von diesen sind ARINC-429 kodiert, drei im CAN (Controller Area Network)

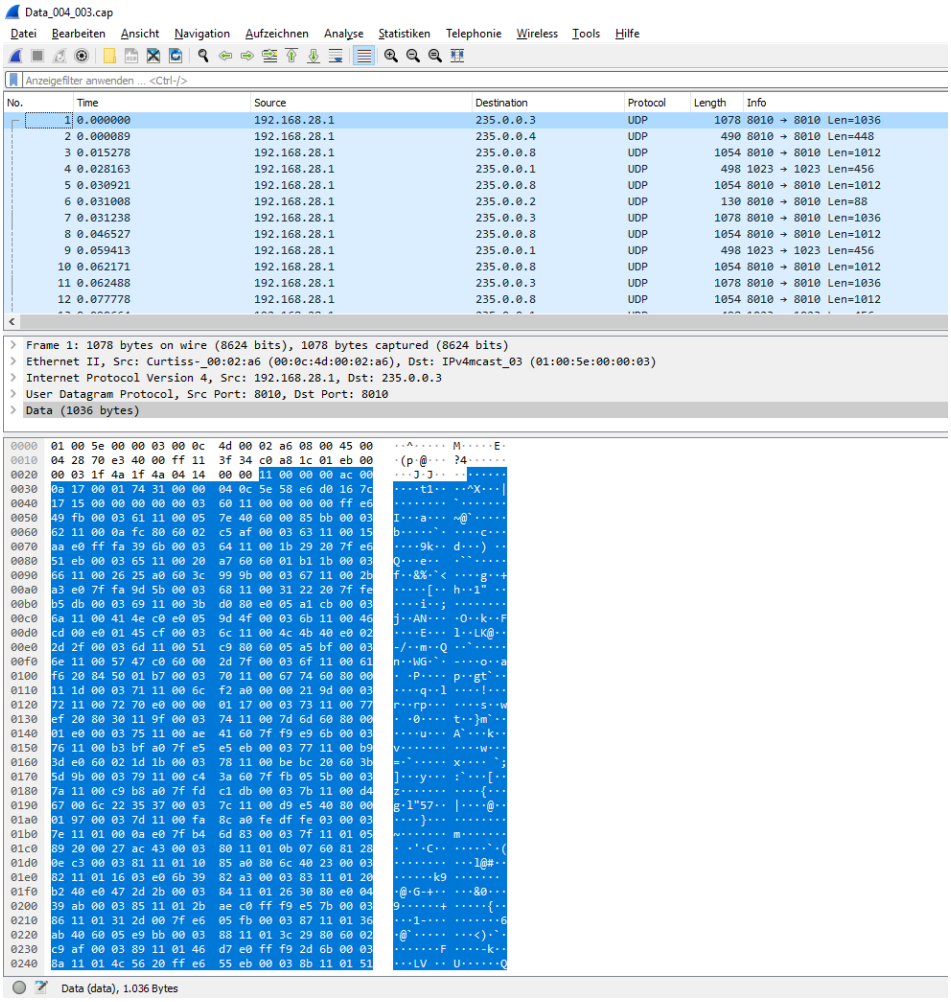

Abbildung 3.4: Eine pcap Datei in der Netzwerkanalysesoftware Wireshark

Format und eine der StreamIDs ist im Format der Bus Control Unit (BCU) dekodiert. BCU und CAN wurden als mittelfristige Ziele als weitere Dekodierungsmöglichkeiten gesetzt. Die Daten aus ARINC-429 sind für die aktuelle Darstellung allerdings ausreichend. Nachdem die StreamID erfasst ist, wird im Fall von ARINC-429 der Hex-String in 12 Byte lange Abschnitte (Messages) geteilt und nacheinander geparst. Die braun unterlegte ARINC Message aus Tabelle 3.2 wird im Weiteren untersucht. Die ARINC-Message aus dem iNET header 3.2 wird weiter aufgeteilt, siehe ARINC-Message Definition 2.2:

Infozeile:	00036011
Elapsed Time:	00000000
Datentraffic:	ffe649fb

Tabelle 3.3: Aufteilung einer ARINC-Message aus Tabelle 3.2

Der Datentraffic in binär konvertiert und auf die BNR Tabelle aus Kapitel 2.1.5 gelegt ergibt:

32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	08	07	06	05	04	03	02	01
P	SSM		MSB									Data									LSB			SDI		Label							
1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	0	0	1	0	0	1	0	0	1	1	1	1	1	1	0	1	1

Tabelle 3.4: Die untersuchte ARINC-429 Nachricht eingesetzt in die BNR-Kodierungsleiste

In Tabelle 3.5 werden die einzelnen Felder aus Tabelle ?? farbig unterlegt dargestellt und der traffic wird umgekehrt, um mit den Indizes zielführend zu arbeiten.

Binärtraffic	1 11 1 11111100110010010 01 11111011
Binärtraffic (umgekehrt)	11011111 10 01001001100111111 1 11 1

Tabelle 3.5: Farbig unterlegter ARINC-429 traffic

Der Datenverkehr muss folglich noch entschlüsselt werden und wird in der kommenden Passage dekodiert. Nach Kapitel 2.1.5 kann der ARINC-429 traffic nun wie in Tabelle 3.6 dargestellt entschlüsselt werden.

Label:	Im Beispiel indiziert das Label 337. Die zusammengehörige Variable bei StreamID ist die Roll Attitude Rate, also Nickrate.
SDI:	Die Source Data Identifier ist für die vorliegende Anwendung nicht relevant, wird aber dennoch übermittelt.
BNR-Wert:	Der binäre Zahlenwert wird in Tabelle 3.7 in dezimal und mit dem skalierwert multipliziert dargestellt. Zu beachten ist, dass bei negativen Zahlen von vollen Bytes heruntergezählt wird. Es macht also Sinn, zuerst das Sign Bit auszulesen, bevor man mit dem Zahlenfeld beginnt
Sign Bit:	Das Vorzeichenbit stellt bei 0: positiven Sinn 1: negativen Sinn dar. Ist also im vorliegenden Beispiel negativ.
SSM:	Für BNR gilt, dass bei 11 eine vollständige, reguläre Übertragung herrscht. Dies ist gegeben.
Parität:	Die einzelnen Daten werden mit Zeitstempel, Name und Wert zurückgegeben. Die Anzahl der Einsen im Binärtraffic in Tabelle 3.5 beträgt ohne das Paritätsbit 22. Um eine ungleiche Summe von Einsen zu gewährleisten muss das Paritätsbit 1 sein. So ist die Gesamtsumme der Einsen in dem übertragenen traffic ungerade und die Parität nach ARINC-429 wird eingehalten

Tabelle 3.6: Entschlüsselung der farbig markierten ARINC-429 Felder des Binärstrings aus Tabelle 3.5

Die binär dargestellten Daten aus Tabelle 3.5 sind in Tabelle 3.7 entschlüsselt dargestellt und stammen aus der in dieser Arbeit entwickelten pythonfunktion(Siehe Anhang A.2.2).

Bezeichnung	Wert
sign	False
label	337
DataOkParity	True
DataOkSSM	True
ElapsedTime	1582884560.3772311
PTPTime	1582884560.3772311
binaryFormat	BNR
ARINCwert	-0.8037109375
VarName	Roll Attitude Rate
VarUnit	deg/s

Tabelle 3.7: Auswertung des traffics mit python

Da die Message die erste des Pakets ist, sind das Feld Elapsed Time (entspricht 28.02.2020, um 11:09:20 2020) und PTP-Zeit in Tabelle 3.7 identisch. Schließlich ist die Zeit, die seit Initialisierung des Pakets in der Messanlage vergangen ist, null. Die Parität, sowie die Prüfung mit SSM indizieren Freiheit von Fehlern. Die gemessene Variable ist die Rollrate, in der Konfiguration gespeichert als Roll Attitude Rate. Die Rollrate ist zudem aufgrund des Vorzeichenbits negativ. Standardmäßig werden in einem iNET-X Paket viele ARINC-429 Messages abgeschickt, die letztlich durch fehlende Informationen nicht dekodiert werden können. Bei Bedarf können diese Informationen jedoch zu einem späteren Zeitpunkt in die Konfigurationsdatei aufgenommen werden, damit sie dekodiert werden können. [14]

Optimierung der Algorithmen

In der ersten Iteration benötigt das Parsen der Flugdatenaufzeichnungen eines ganzen Flugs mit pyshark 10 Minuten. Für Evaluation einer effizienteren Methode, welche durch die Pakete iteriert, wurden verschiedene Bibliotheken auf Funktionalität getestet.

Die Bibliotheken *pyshark*, *scapy* und *dpkt* wurden durch eine Beispielsdatei auf ihre Geschwindigkeit untersucht.

Das Ergebnis ist in Abbildung 3.5 dargestellt. Es zeigt sich, dass Zeitunterschiede von zwei Größenordnungen zwischen pyshark mit 12 Sekunden und dpkt mit 0.3 ersichtlich sind.

Somit bewegt sich dpkt in einer vollständig anderen Größenordnung als die anderen beiden Bibliotheken. Dies erklärt sich durch den Umfang von pyshark und scapy. Sie zerlegen den Netzwerkverkehr im Vorhinein schon stark. Da Metainformationen in der vorliegenden An-

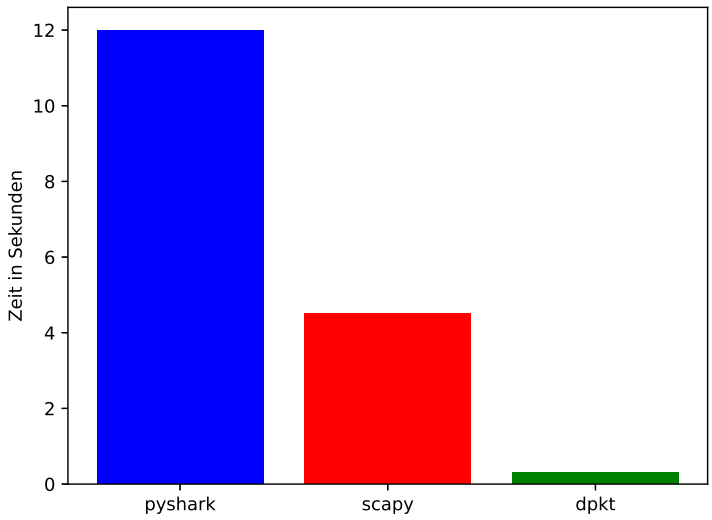

Abbildung 3.5: Geschwindigkeitsunterscheid bei Iteration durch eine exemplarische pcap-Datei

wendung nicht relevant sind, kann alles nicht automatisch geparste selber ergänzt werden werden.

Beim Parsen der Netzwerkaufzeichnungen eines 1,5h Fluges konnte die Zeit durch die Nutzung von dpkt von 10 auf 5 Minuten halbiert werden. Hierbei wird der Teil des Programms wesentlich effizienter ausgeführt, der die pcaps ausliest. Die Verarbeitung der pcaps, sowie das parsen des ARINC-429 Formats nimmt jedoch immer noch einen erheblichen Teil der Zeit ein.

Ursprüngliche Zielsetzung

Der Fokus dieser Arbeit und der Echtzeitauslesung wurde anfangs aufgrund von Wartungsverzögerungen von der Bo105 Messanlage auf die der D-CODE verändert. Dazu kamen zusätzliche Komplikationen aufgrund Corona und unvorhergesehen Wartungsverzögerungen auch bei der D-CODE. Aufgrund dessen konnte nicht mit Echtzeitdaten gearbeitet werden und es wurde auf Grundlage aufgezeichneter Flüge der D-CODE an der Entschlüsselung und Darstellung der Messdaten gearbeitet. Die Darstellung als Wiedergabe eines Fluges bringt andere datentechnische Implikationen mit sich, wird aber die später darauf aufbauende Echtzeitauslesung begünstigen, da viele der Grundbausteine wie Datenverwaltung und Darstellung schon entworfen sind. Auch kamen einige Features wie Pausieren und Vor- und Zurückspulen auf einer Zeitleiste dazu, welche bei Anfang der Entwicklung nicht zwingend vorgesehen waren.

3.2 Schnittstelle Mensch-Maschine

Der Entstehungsprozess der Benutzeroberfläche wird in den nächsten Absätzen dargelegt, indem zunächst die Anforderungen festgelegt werden, die Methoden zur Erfüllung dieser evaluiert werden und abschließend das Vorgehen bei der Erstellung erklärt wird. Die konkreten Schritte bei der Erstellung sind in Abbildung 3.6 dargestellt; diese stellt die Entwicklung der MMS dar, die in diesem Abschnitt beschrieben wird.

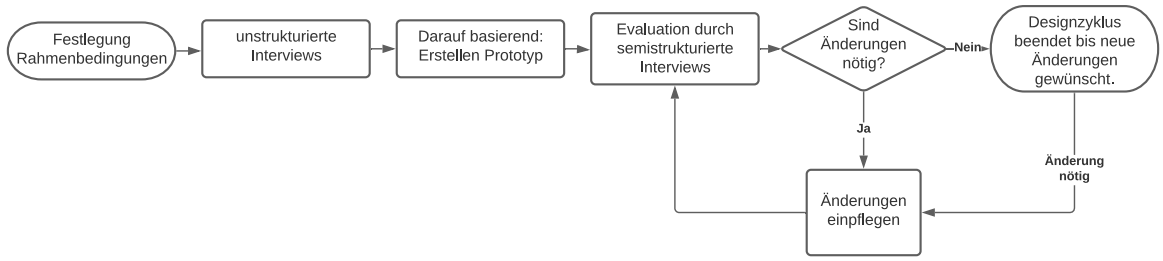

Abbildung 3.6: Schritte bei Erstellung der GUI

3.2.1 Überlegungen zur Darstellung der Daten

Das Vorgehen zur Erstellung der GUI besteht zunächst daraus, Überlegungen zur Darstellung der Flugzustandsdaten anzustellen. Erster Schritt dabei ist, die Anforderungen festzulegen und Gespräche mit Flugversuchsingenieuren, den zukünftigen Nutzern, zu führen und dann zu recherchieren, was in der Industrie verwendete Konventionen zur Darstellung von Daten sind. Nachdem ein aussagekräftiges Stimmungsbild eingeholt ist, wird sich auf die Konkretisierung dieser Anforderungen, wie gute Konfigurierbarkeit, Ergonomie und zeitgemäße Darstellung gekümmert.

Interviews mit Flugversuchsingenieuren

Im Rahmen der Erstellung der GUI wurde zunächst ein Anforderungsprofil der geforderten Anwendung erstellt. Dafür wurde als Grundlage die in der Ausschreibung definierte Arbeit genommen und unstrukturierte Interviews mit Flugversuchsingenieuren geführt.

Diese Interviews werden mit den Flugversuchsingenieuren des DLR Malte Kreienfeld und Waldemar Krebs geführt. Beide sind von der Darstellungsoberfläche überzeugt und bestärken, dass Fluginstrumente in der Oberfläche, sowie PFD und Moving Map relevant sind. Abgesehen davon wird darauf hingewiesen, dass die Konfigurierbarkeit der Datenanzeige ein wichtiger Aspekt ist, um die Werte für die unterschiedlichen Messparameter der Flugversuche angemessen anzuzeigen. Im Hintergrund dieser Interviews stand immer die übersichtliche Darstellung und auf dem Luftfahrtkontext basierte Darstellung der Daten. Die Früherkennung fehlerhafter Daten sorgt dafür, dass Flugversuche nicht doppelt geflogen werden müssen.

Menschliches Leistungsvermögen

Kapitel 2.2.5 stellt die Grundlagen der menschlichen Leistungsfähigkeit in Hinblick auf die Interaktivität vor. Darauf aufbauend setzt die menschliche Psychologie der Aufnahme von Daten einige Grenzen. Um den Nutzer deshalb nicht zu überlasten oder die Anzeige zu überladen, wird beschlossen, nicht alle 90 Variablen, die während eines Fluges gemessen werden, auf einer Seite anzuzeigen, sondern diese in drei bis fünf graphischen Anzeigen mit Plots darzustellen. Wichtig ist, dass der Nutzer während der Nutzung maximales Situationsbewusstsein bei minimaler Arbeitsbeanspruchung hat. Um dies zu realisieren, kann die bereits vorgestellte Farbgebung hilfreich sein, damit durch Einfärbung zusammengehörender Elemente und Einhaltung von Konventionen aus dem Cockpit dies bewerkstelligt wird.

Allgemeines Layout

Das aus den Interviews gewonnene Wissen kann nun auf den Anwendungsfall übertragen werden. Zunächst wird sich für eine Tabstruktur der einzelnen Thematiken entschieden, um verschiedene Daten zu strukturieren. So können einzelne Tabs beliebig konfiguriert werden und thematisch getrennt werden. Zudem wird die in Abbildung 3.7 dargestellte Nutzeroberfläche der Firma Aerodata untersucht. Darauf basierend wird entschieden, dass das in der linken Leiste obere und mittlere Element, eine Moving Map, sowie ein Primary Flight Display (PFD) in dieser Form in die Oberfläche dieser Arbeit aufgenommen werden. Zusätzlich soll es zur erhöhten Übersicht möglich sein, diese linke Leiste zu minimieren.

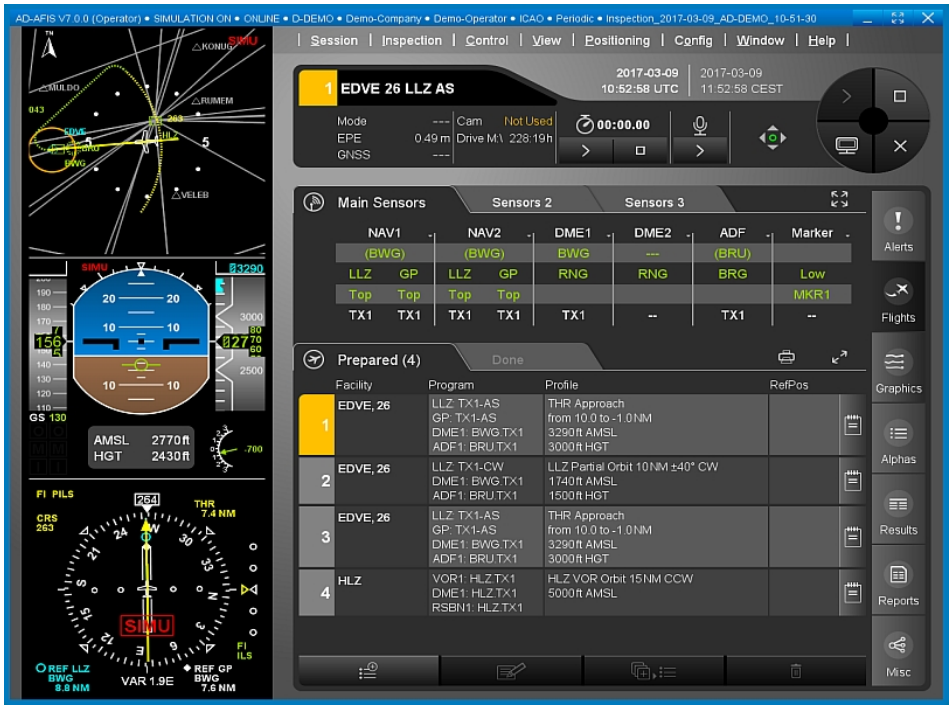

Abbildung 3.7: Benutzeroberfläche der Firma Aerodata [25]

Auswahl eines Mobile Device

Um die Daten im Luftfahrzeug darzustellen, wird festgelegt, ein Tablet mit optionaler Tastatur zu verwenden. Für Flugversuche, bei denen Beschleunigung variabler Stärke in allen Richtungen, stattfinden kann, besteht die Anforderung an ein mobiles Gerät, dass sich durch beispielsweise ein Kniebrett festmachen lässt. Ein Surfacelaptop mit optionaler Tastatur ist für diese Umstände geeignet, da eine Tastatur bei Bedarf in einer ruhigeren Flugphase ausgeklappt werden kann, um beispielsweise Notizen während des Flugs zu machen. Im Weiteren ist zu beachten, dass bei Nutzung von mobile devices(mobiles elektronisches Gerät) in Luftfahrzeugen Löschtaschen für Batteriebrand an Bord sein müssen. Durch mobile Anwendung und nicht festeingebaute Geräte kann zudem der Zertifizierungsprozess verkürzt werden. [26, 27]

Resultierend aus dem geplanten Einsatzumfeld Flugversuch im Hubschrauber und in einer unbedruckten Kabine der Do 228 mit starken Turbulenzen und Vibrationen, die man aus der Passagier-Luftfahrt nicht kennt, kommen hohe Anforderungen auf das Gerät zu. Das mobile Gerät sollte möglichst robust sein. Hinzu kommt eine notwendige Leistungsgfähigkeit für die Datenverarbeitung in python. Für die Anwendung in der D-CODE sollte es speziell dem Luftdruck bei 27000ft bei unbedruckter Kabine standhalten. Darüber hinaus gehört zu den Anforderungen, dass das Gerät gegen die Umwelteinflüsse Sonne, Hitze und Kälte resistent ist, es Vibration in den im Flugversuch üblichen Dosen aushält, es robust genug ist, um während Turbulenzen ein Herunterfallen oder etwaige lose Gegenstände auszuhalten und wasserdicht ist. Um den Datenstrom auszulesen ist ein Ethernet, sowie Bluetooth Anschluss nötig und eine Schnittstelle zu einer Tastatur soll auch bestehen. Weiterhin ist es wichtig, dass durch das Betriebssystem eine ausreichende Konfigurierbarkeit der Software gegeben ist, um es gegebenenfalls auf spezielle Anforderungen anzupassen.

Ein Gerät, dass diese Anforderungen zufriedenstellend erfüllt, ist das Latitude 7220. Von diesem Tablet erfüllte Bedingungen sind:

- zertifiziert für Betriebsstürze nach Militärnorm 810G und 810H
- Betriebstemperatur -29° bis 63°C
- mit Handschuh bedienbarer Bildschirm
- zusätzliche Makrotasten
- ausreichend leistungsfähige Hardware

Zusätzlich zum Latitude 7220 wird von den Flugversuchsingenieuren die Nutzung eines iPads vorgeschlagen, deren Nutzung dieser schon Standard im Cockpit ist und sie ein praktisches Format besitzen. In Tabelle 3.8 werden beide Geräte einander gegenübergestellt.

Anforderung \Gerät	Latitude 7220	iPad
technische Daten	16GB Ram, 8x1,9GHz	4GB Ram, 2x2,26 GHz
Bildschirmdiagonale	11,6"	11"
Robustheit	Getestet nach Mil Norm 810G und 810H	Nur durch zusätzliche Schutzhülle möglich
Betriebssystem	Windows	iOS
Entwicklungsaufwand	PyQt läuft auf gängigen Betriebssystemen	Swift ermöglicht Appentwicklung
Vorwissen	python Vorwissen vorhanden	kein Vorwissen in Swift vorhanden
Datenmanipulation	Mit Numpy Datenmanipulation	Datenmanipulation und Einbindung von Bibliotheken umständlich
Schnittstellen	Ethernetmodul vorhanden, sowie USB und gängige Formate	Apple Lightning, Smart Connector
Preis	2460	1209

Tabelle 3.8: Vergleich des iPad mit dem Dell Latitude 7220

iPads sind bei Piloten und Flugversuchsingenieuren schon in Gebrauch und haben ein intuitives und bekanntes Design. Sie sind allerdings nicht so performant wie Windows Tablets. Eine für den Anwendungsfall zunächst absolut notwendige Ethernetverbindung ist nur mit zwei seriell geschalteten Adaptern möglich. Dies gewährleistet keine feste Verbindung und begünstigt bei der Auswahl das Windows Tablet. Das Windows Tablet erfüllt somit nicht nur die Grundanforderungen, sondern auch darüber hinaus weitere Leistungsmerkmale wie erweiterte Betriebstemperaturen und einen mit Handschuhen bedienbaren Bildschirm. Dies ist für den Anwendungsfall im Helikopter geeignet, da bei bestimmten Einsätzen Hubschrauberpiloten und Flugversuchsingenieure (FVI) generell mit Handschuhen als Teil der Schutzausrüstung fliegen müssen.

3.2.2 Elemente der GUI

Die Flugversuchsingenieure haben einige Anforderungen im Kapitel 3.2.1 an die GUI geäußert. Diese soll neben der Konfigurierbarkeit auch Standardanzeigen der Luftfahrt besitzen, damit die Flugversuchsingenieure einen größeren Überblick der Flugsituation erlangen. Diese Standardanzeigen werden in den kommenden Absätzen genauer dargestellt und erläutert.

Abbildung 3.8: Das Instrumentensixpack [28]

Bedeutung von Farben in der Mensch-Maschine Schnittstelle

Wie in Kapitel 2.2.4 erwähnt können Farben in Bezug auf Anzeigen und Auswahlmöglichkeiten durch ähnliche Farben in Sättigungsstufen bei den Tabs gruppiert werden und eine Signalwirkung durch Kontraste erzielen. In dieser Anwendung wird der Einsatz von Farben bei z.B. gleichartigen Messdatenvariablen bzw. Tabmarkierungen in Erwägung gezogen. So ist bei beispielhafter Gruppierung von Navigations- und Fluglagedaten ein schnelles Erfassen und Auswählen der gewünschten Darstellung möglich. Weiterhin spielen Farben in der Luftfahrt eine wichtige Rolle, um Zustände darzustellen. So stellen die Farben grün, gelb und rot wie in einem Ampelsystem verschiedene Signale dar. Die grüne Farbe steht für Einhaltung der Betriebszustände bzw. normalen Betrieb, gelb hingegen als Hinweis, dass etwas geringfügig nicht in Ordnung ist und Achtung im weiteren Vorgehen geboten ist. Die Farbe rot schließlich bedeutet Eintreten eines schwerwiegenden Fehlers, bzw. Überschreiten der Betriebsgrenzen. Dies ist vor allem bei Geschwindigkeitsanzeigen hilfreich, um die Anzeige in Relation zu den Grenzen des Luftfahrzeugs zu setzen. In der Arbeit werden Farben eingesetzt, um durch eine Ampel den Status der Messwerte darzustellen. Diese Ampel funktioniert auf eine Weise, dass bei Messwertgrößen außerhalb eines festgelegten Limits kein grüner Status mehr angezeigt wird.

Fluginstrumente

In der Luftfahrt gibt es verschieden allgemeingültige Darstellungen von Fluginstrumenten. Einige Instrumente, die den Flugzustand messen sind in Abbildung 3.8 dargestellt. Das von der britischen Luftwaffe vor dem zweiten Weltkrieg eingeführte und seitdem international genutzte Sixpack nutzt den künstlichen Horizont als Hauptelement und stellt links vom künstlichen Horizont den Fahrt- und rechts den Höhenmesser dar. In der zweiten Zeile befinden sich von links nach rechts der Wendezeiger (Drehbewegung um Hochachse), der Kompass und das Variometer, welches die Höhenänderung anzeigt. Neben Anzeigen zum Flugzustand gibt es beispielsweise noch Anzeigen zum Status des Flugzeugs oder des Antriebs. Um auf den Luftfahrtkonventionen aufzubauen, ist es von Vorteil, diese zu nutzen und Variablen in dieser Form darzustellen. Der Fahrtmesser in Abbildung 3.8 stellt die Farbgebung z.B. sehr übersichtlich dar. Der grüne Bereich ist der Manöverbereich, in dem alle Ruder voll ausgeschlagen werden können. Im gelben Bereich sind starke und extreme Ruderausschläge zu vermeiden, um die Flugzeugstruktur zu schützen, und Geschwindigkeiten ab dem roten Strich sind zu vermeiden. Der weiße Bereich stellt zudem den Geschwindigkeitsbereich für ausgefahrene Landeklappen dar.

Primary Flight Display (PFD)

Ein Primary Flight Display (PFD) (Abbildung 3.9) ist eine effiziente Darstellung, die das ursprüngliche Sixpack (Abbildung 3.8) von Instrumenten kondensiert. PFDs sind in aktuellen Verkehrsflugzeugen der Standard.

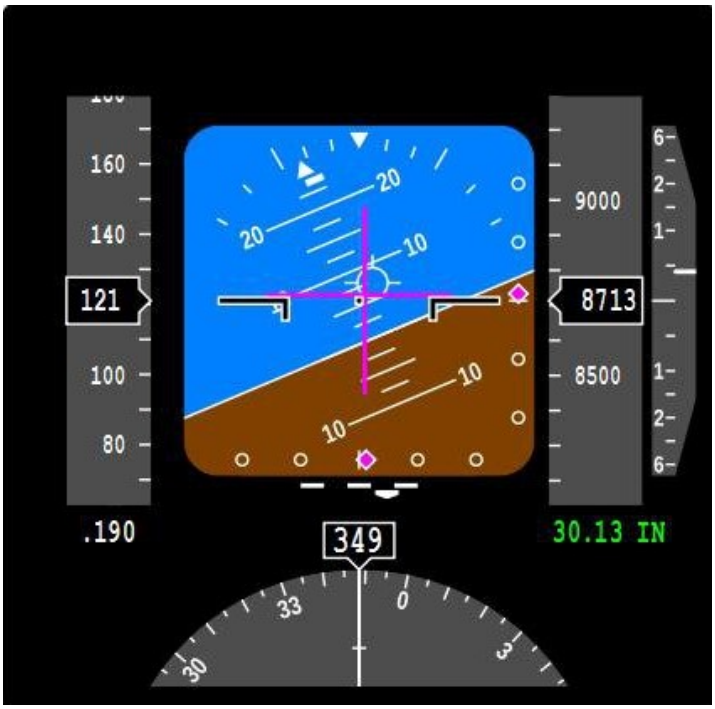

Abbildung 3.9: Das PFD [28]

Ein PFD zeigt die Werte Flughöhe, Geschwindigkeit, Anstellwinkel, Rollwinkel, Variometer und Kompasskurs an und bietet dem Piloten eine direkte Übersicht über den Flugzustand. Da jedem Piloten und Flugversuchsingenieur diese Ansicht bekannt ist, bietet es sich an, ein PFD einzubauen, da die Nutzung von diesem zusätzlich auch einen positiven Einfluss auf das Situationsbewusstsein hat.

Navigation Display

Eine bewegte Karte als Navigationsanzeige differiert in der Funktionsweise nicht stark von einem Automobilnavigationsgerät, ergänzt aber für Luftfahrzeuge wichtige Informationen wie Flughäfen und andere Flugzeuge, Wetterinformationen, Geschwindigkeit über Grund, angezeigte Geschwindigkeit, Windvektor und Autopilotenmodus dargestellt. Zudem ermöglicht eine Karte in der Luft auch bei unklarem Wetter eine rasche Übersicht. Das Navigation Display stellt Werte wie Kurs, Längen- und Breitengrad sowie Karteninformationen dar.

Datendarstellung im Plot

Für den Flugversuchsingenieur ist auch die Datenanzeige mit Blick in die Vergangenheit relevant. So wird in den Plots der GUI die letzten 10 Sekunden der Daten angezeigt. Auf diese Weise ist die Informationsdichte übersichtlich darstellbar. Zusätzlich zu dieser Art der kurzen Historie soll auch das Darstellen der Daten des gesamten Fluges möglich sein, um beispielsweise ein Höhenprofil darzustellen. Zur Zeit der Abgabe sind bei der Darstellung einige Optimierungsmöglichkeiten offen. Es wurde noch kein Leistungstest der Anwendung mit allen zukünftigen Graphen durchgeführt. Nach aktuellem Stand werden die Daten den Flugdaten entsprechend mit 25 Hz abgespielt. Eine Möglichkeit zu glattem Ablauf auch bei stockenden Daten ist die die Wiedergaberate von 25 Hz bei rechnerischer Überladung zu verringern. Ein downsampling, bei dem weniger Datenpunkte angezeigt werden, soll mit allen Mitteln vermieden werden, da so ein Tiefpassfilter implementiert werden kann, der die Daten verfälscht. Zudem können noch Arbeiten in Richtung von multiprozessorbasierten Funktionen hinzugefügt werden, die die Nutzung von mehreren Kernen ermöglichen, sowie eine spätere Implementierung effizient zu laufender Skripte in C++. Es wird beschlossen, die Daten wie in Abbildung 3.10 mit drei Feldern darzustellen. Diese stellen kompakt die Rohinformationen, das Instrument für eine schnelle Übersicht und die Historie für genauere Informationen.

3.2.3 Umsetzung

Der folgende Abschnitt zeigt die Entstehung der Nutzeroberfläche und stellt die die Funktionsweise der dabei entstandenen Schnittstelle vor.

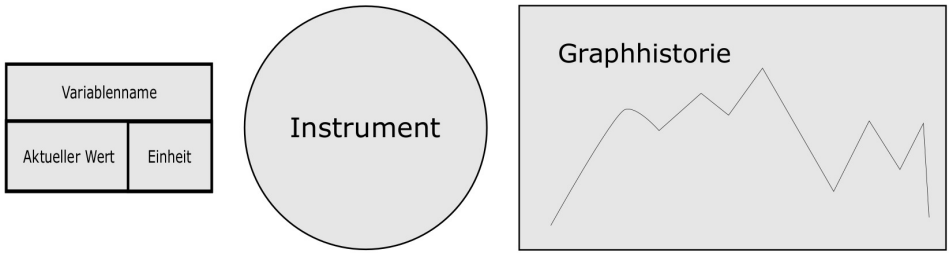

Abbildung 3.10: Gewählte Darstellung der Daten für die Benutzeroberfläche

Ergonomie

Da es sich um eine Luftfahrtanwendung handelt, bei der während des Versuchs schnell Daten dargestellt werden müssen, steht die Ergonomie und das schnelle Erkennen der vielen Daten im Vordergrund. Wichtig dabei ist vor allem die Robustheit gegen Anwenderfehler und möglichst intuitive Menüführung. Die Darstellung der Daten wird mit einer Darstellung ähnlich Abbildung 3.10 erstellt. Durch diese Darstellung ist ein sinnvolles, den regulären links- nach Rechtskonventionen entsprechendes Datenformat gewählt, welches auf schnellen Blick alle relevanten Informationen enthält. Zusätzlich wird das PFD und die Moving Map für die Nutzeroberfläche gewählt. Dies ist für die Endnutzer, Flugversuchsingenieure und Piloten ein gewohnter Anblick und führt zu einem schnellen Erfassen der Parameter und einer geringeren Aufmerksamkeitsbelastung.

Erstellung einer graphischen Flugversuchsoberfläche

Weiteres wichtiges Kriterium der GUI ist die einfache Anpassung an verschiedene Flugversuche, bei denen andere Variablen für den Benutzer interessant sind.

Somit ist eine Konfigurationsdatei unerlässlich. Als Ausblick steht, einen Generator für diese Datei zu entwickeln, welcher intuitiv zu bedienen ist, Eingabefehler vermeidet und im gleichen Zug die Nutzereinarbeitungszeit senkt.

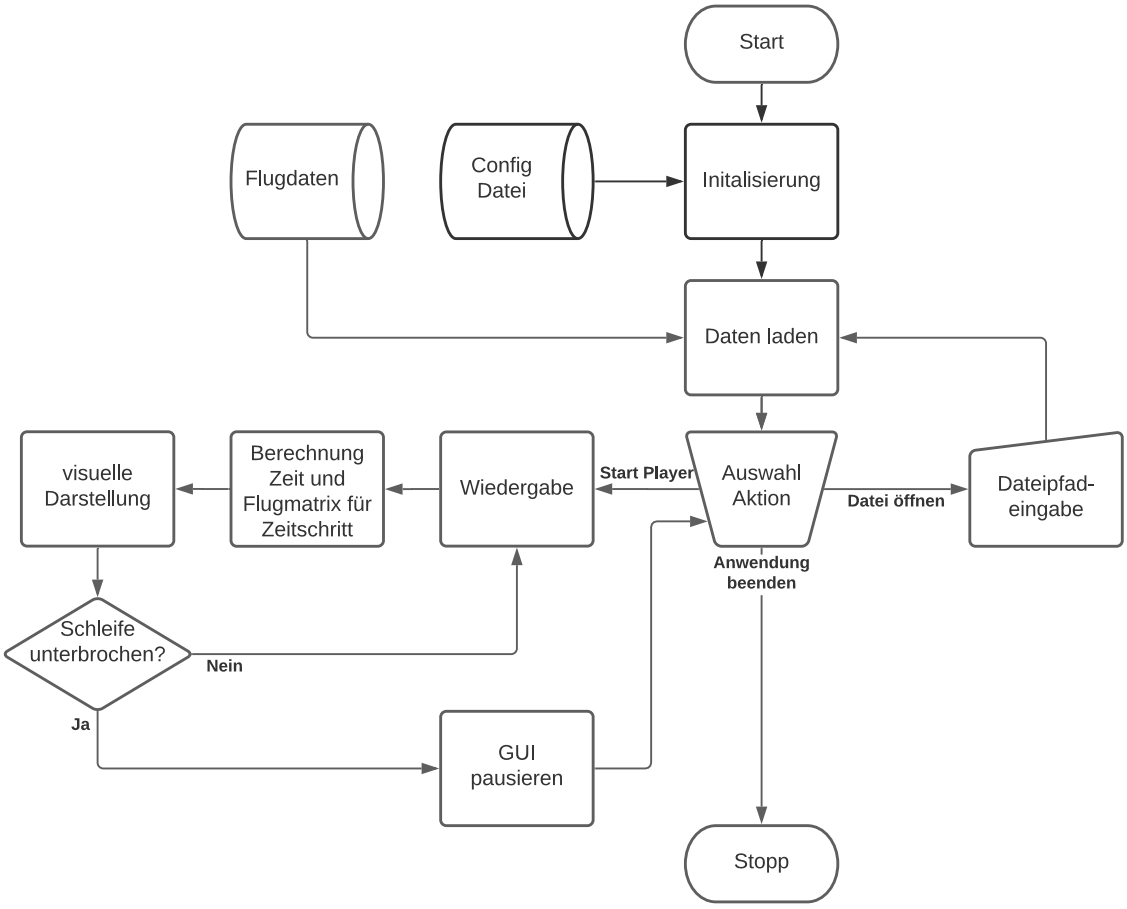

Abbildung 3.11: Ablauf der Anwendung

Programmablauf der Nutzeroberfläche

Abbildung 3.11 stellt den Ablauf für die aktuelle Benutzeroberfläche zur Wiedergabe von Flugdateien in einem Flowchart dar. Für die Darstellung von Flugdaten nach dem Flug gilt, dass der Nutzer zunächst die Daten von einem gewünschten Flug auswählt, die danach in die Anwendung geladen werden. Sobald dies geschehen ist, wartet das Programm auf eine manuelle Eingabe des Nutzers. Wird die Wiedergabe gestartet, begibt sich das Programm in eine Wiedergabeschleife, die bei Betätigung des Pause oder Stopp Knopfes unterbrochen wird. Während der Schleife wird zunächst die Zeit beim Drücken des Knopfes gespeichert, um eine Abhängigkeit von dieser Zeit für die nächsten Zeitschritte zu schaffen. Die Matrix mit dem gewählten Fenster an Flugdaten wird an eine Funktion zur visuellen Darstellung der Daten weitergeleitet und auf Unterbrechung der Wiedergabe geprüft.

Vorgehen

Bei der Erstellung wird sich an bestehenden Flugversuchsoberflächen orientiert, um in die GUI einen direkten Übergang zu bekannten Systemen zu erstellen. Durch die skalare Darstellung des aktuellen Werts auf einer Anzeige, sowie die Vergangenheitsanzeige der nahen Vergangenheit in einem Plot hat der Flugversuchsingenieur eine schnelle Übersicht über die Messdaten während seines Flugversuchs.

Bei Erstellung der Oberfläche ist zunächst von den bekannten Daten eines Flugs auszugehen und diese dann in einem Wiedergabemodus der Oberfläche darzustellen. Ein Modus zur Liveausgabe wird auf Basis des bestehenden Systems inplementiert.

Dieses Vorgehen fügt dem Endprodukt einige Eigenschaften hinzu. So ist es möglich, die Messdaten zu pausieren/einzufrieren, und es ist ebenso möglich, die Daten zurückzuspulen, um in z.B. einem unereignisreichen Flugabschnitt die Messdaten eines Flugversuchs im Nachhinein durchzugehen.

V1-grundlegende Funktionalität

Zunächst wird in der ersten Version eine simple Arbeitsversion erstellt, um die grundlegende Funktionalität zu prüfen. Es gibt verschiedene Tabs für die zusammengehörigen Themen. Flugdaten können in Plots dargestellt werden und die Programmlogik zur Wiedergabe funktioniert. Die Darstellung ausgewählter Daten über eine Konfigurationsdatei ist zu diesem Zeitpunkt bereits möglich.

Abbildung 3.12: Ablauf der Anwendung

V2-erweiterte Version

In der zweiten Iteration werden zwei Elemente in einer Sidebar hinzugefügt, die bei Bedarf nach besserer Übersicht ausgeblendet werden können. Die im Hintergrund laufenden Funktionen zum Abspielen der Flugdaten werden optimiert.

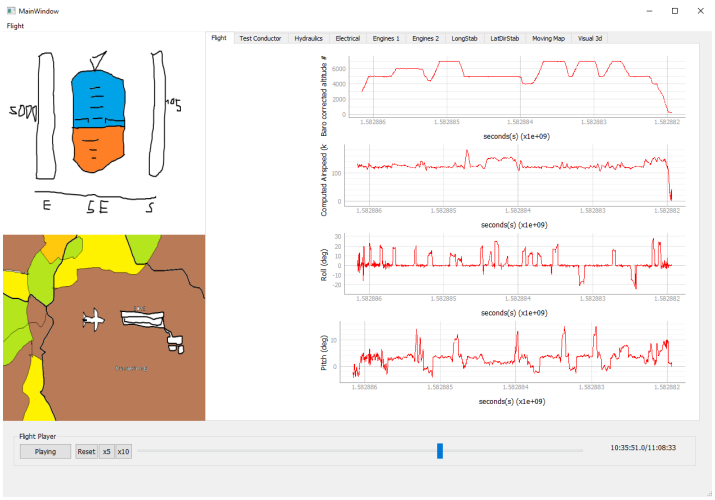

Abbildung 3.13: V2 der GUI

V3-Addition von Rundinstrumenten und Graphen, noch nicht implementiert

Zusätzliche Darstellungselemente wie Fluginstrumente, eine Ampel für den Status des Dateninputs werden hinzugefügt. Es ist möglich, den angezeigten Datenzeitraum von 5 bis 60s auszuwählen und es ist möglich, die Wiedergabegeschwindigkeit bei vergangenen Flügen anzupassen.

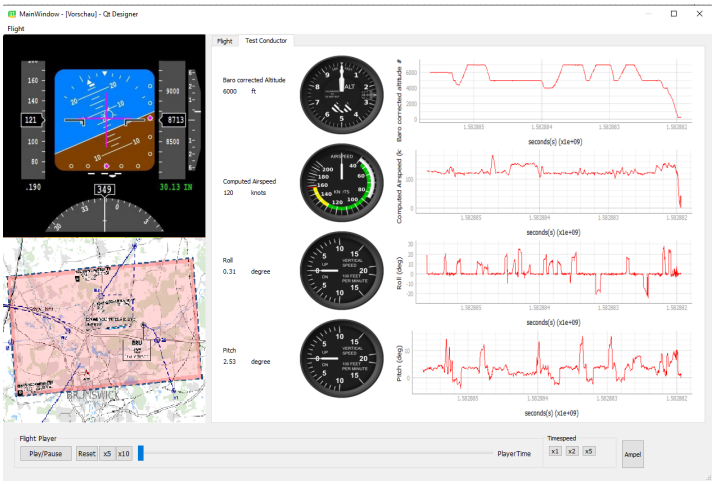

Abbildung 3.14: V3 der GUI (noch nicht implementiert)

Zusammenfassend kann die Interaktion des Nutzers mit Abbildung 3.15 zusammengefasst werden. Der Nutzer interagiert mit der MMS und kann die anzuzeigenden Daten und die Datenquelle auswählen. Diese Schnittstelle leitet die Anfrage an das Backend weiter, welches die Daten aus dem Echtzeitdatenstrom oder einer Aufnahme des Datenstroms auslesen kann. Diese Daten leitet das Backend an die Benutzeroberfläche weiter, in der diese dargestellt werden.

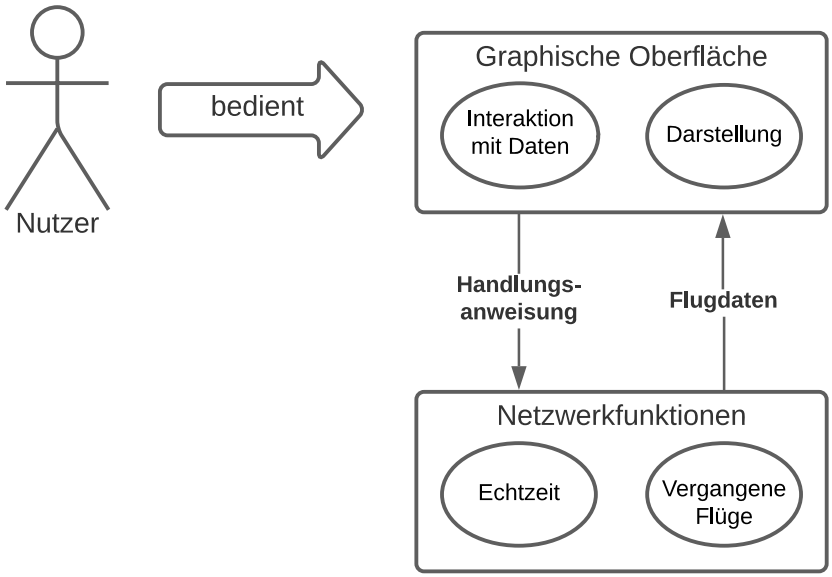

Abbildung 3.15: Systemskizze Front- und Backend

4 Zusammenfassung und Ausblick

In der vorliegenden Arbeit wird ein Datenstrom aus der Messanlage der Do228 D-CODE im ARINC-429 Format entschlüsselt, sowie eine Nutzeroberfläche nach Maßstäben der Mensch-Maschine Schnittstelle entwickelt, welche die Daten dieses Datenstroms wiedergibt und sich konfigurieren lassen soll. Motivation hierfür ist der Wunsch nach einer Livedarstellung der Flugzustandsdaten während des Flugversuchs, um primär die Möglichkeit zu erlangen, dem Flugversuchsingenieur Zustandsdaten im Flug anzuzeigen und sekundär eventuelle Sensorfehler möglichst früh zu erkennen. Zudem wurde die Dekodierung der Flugzustandsdaten im Vergleich zur vorherigen Dekodierung durch IMC Famos optimiert.

Das Vorgehen war, zunächst Flugdaten von vergangenen Flügen mit der kostenlosen Software Wireshark auszulesen. Aufbauend darauf wurde mit der Programmiersprache python ein Parser für die Netzwerkdaten geschrieben. Hierbei traten aufgrund zunächst unzugänglicher Dokumentation Verzögerungen beim Parsen auf. Das Format ARINC-429 ist hochspezialisiert für flugzeuginterne Datenübertragung. Die Dokumentation ist dementsprechend beschränkt zugänglich. Nachdem die Daten entschlüsselt und in einem binären pythonformat abgelegt sind, werden diese mit bereits über IMC Famos entschlüsselten Daten verglichen. Nachdem eine vollständige Übereinstimmung gewährleistet ist, beginnt die Entwicklung der Anwendung zur Darstellung der Daten. Da die Messanlagen aufgrund Verzögerungen in der Wartung und mangelnde Zugänglichkeit zum Standort durch den Coronavirus nicht zugänglich sind, wird anstelle der Echtzeitauslesung der Messdaten mit den dekodierten Flugmessdaten gearbeitet und ein Player für die Flugdaten entwickelt, dessen Oberfläche auch während der Flüge nutzbar sein wird. Dazu werden Interviews mit Flugversuchsingenieuren geführt, um die Benutzeroberfläche ihren Wünschen entsprechend zu erstellen. Mit diesen gewünschten Features wird nun die Nutzeroberfläche zunächst in Form eines Players für vergangene Flüge entwickelt. Dieser bietet den Flugversuchsingenieuren und Testpiloten bekannte Anzeigen wie ein PFD und bietet so in herausfordernder Umgebungen eine rasche Übersicht über den Flugzustand.

Im Rahmen einer ausführlichen Analyse hinsichtlich einer optimalen Gestaltung dieser Mensch-Maschine Schnittstelle wurden neben der Literaturrecherche auch Interviews mit Flugversuchsingenieuren geführt.

Durch Verzögerungen bei der Wartung des Hubschraubers Bo-105 wurde der Fokus dieser Arbeit auf die Messanlage der D-CODE gelegt. Bedingt durch Pandemieanordnungen am Standort konnte das Auslesen der Messanlage nicht vor Ort stattfinden. Der Test der entwickelten Anwendung im Flug wird jedoch noch nach der Abgabe dieser Arbeit erfolgen.

Mit dieser Arbeit wurde eine Grundlage für Weiterentwicklungen bei Dekodierung von Datenströmen über Ethernet Streams, sowie bei GUI-Entwicklungen für Oberflächen zur Darstellung der Daten während des Flugs geschaffen. Eine spätere Portierung der Anwendung von PyQt ins C++ QT Framework ist neben weiteren Optimierungen bei eventuellen späteren Performanceschwächen in python möglich. Portierung auf den Hubschrauber Bo105 wird nach Ende dieser Arbeit geschehen. Die entwickelte Arbeit kann darüber hinaus um Datenströme aus anderen Quellen, wie beispielsweise experimenteller Sensorik, erweitert werden.

Literatur

[1] DLR: Webseite Flugexperimente des DLR, 2021.

[2] Morpheus: Netzwerktechnik Tutorial #5 - Signalklassen, 25.07.2018.

[3] Bernstein, H.: Informations- und Kommunikationselektronik, Berlin/Boston: De Gruyter, ISBN 9783110360295, 2015.

[4] Westphalen, K.; Burdich, J., Hrsg.: Felix, Bamberg: Buchner, ISBN 978-3-7661-5200-8, 2009.

[5] Dahmen, W.; Reusken, A.: Numerik für Ingenieure und Naturwissenschaftler. Springer-Lehrbuch, Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg, ISBN 978-3-540-764922, 2008.

[6] Oracle Corporation: Data Types and Sizes, 2010.

[7] Ameling, W.: Digitalrechner. Technische Informatik, Braunschweig: Vieweg, ISBN 3528064951, 1992.

[8] DIN-Normenausschuss Informationstechnik und Anwendungen: Information technology - Open Systems Interconnection: Basic reference model: The basic model, ISO/IEC 7498-1, 1994.

[9] Kowalk, W. P.; Burke, M.: Rechnernetze: Konzepte und Techniken der Datenübertragung in Rechnernetzen. Leitfäden der Informatik, Stuttgart: Teubner, ISBN 3519021412, 1994.

[10] Meyers, M.: CompTIA A+ Prüfungsvorbereitung: Das umfassende Praxis-Handbuch für IT-Administration, Netzwerktechnik und Support : aktuell zu den neuen A+-Prüfungen 220-1001 und 220-1002. All in one, Frechen: MITP, ISBN 9783747501009, 2020.

[11] Consultative Committee for Space Data Systems: Internationales Forum für Entwicklung von Kommunikation und Datensystemstandards für die Raumfahrt, 1982.

[12] Waitzman, D.: IP over Avian Carriers with Quality of Service, 1999.

[13] ACRA Control: TEC/NOT/051: Ethernet frames, Wireshark and FAT32: Chapter 45.

[14] ACRA Control: TEC/NOT/067: IENA and iNET-X packet payload formats: Chapter 44.

[15] Avionics Interface Technologies: ARINC 429 Protocol Tutorial: Doc No. 40100001, 2017.

[16] Airlines Electronic Engineering Comittee: Mark 33 digital information transfer system(DITS) Part 1: Functional Description, Electrical Interface, Label Assignments and Word Formats: ARINC SPECIFICATION 429 PART 1-16.

[17] Wandmacher, J.: Software-Ergonomie. Mensch - Computer - Kommunikation : Grundwissen, Berlin und New York: De Gruyter, ISBN 311012971X, 1993.

[18] Butz, A.; Krüger, A.: Mensch-Maschine-Interaktion, De Gruyter Oldenbourg, ISBN 9783486989717, 2014.

[19] Bloch, A.: Gesammelte Gründe, warum alles schiefgeht, was schiefgehen kann: Murphy's Gesetze in 1 Bd. Goldmann, München: Goldmann, ISBN 3442100461, 1985.

[20] Shneiderman, B.: Direct manipulation: A step beyond programming languages, TR 1245, 1983.

[21] Nielsen, J.: 10 Heuristiken für UI Design, 1994.

[22] Itten, J.: Kunst der Farbe: Subjektives Erleben u. objektives Erkennen als Wege zur Kunst, Ravensburg: Maier, ISBN 347361551X, 1987.

[23] Kassera, W.: Flug ohne Motor: Das Lehrbuch für Segelflieger, Stuttgart: Motorbuch-Verl., ISBN 9783613032576, 2011.

[24] Vidyarthi, N.: Reading PCAP files on Matlab, 2020.

[25] aerodata: Flight Inspection Systems made by Aerodata: Graphical User Interface, 2021.

[26] Federal Aviation Administration: InFO13010SUP: FAA Aid to Operators for the Expanded Use of Passenger PEDS, 2014.

[27] Burwitz, S.: Loose Electronic Equipment(PED): FTOM: Project SOP Ausgabe B, 2020.

[28] Cel, M.: Qflightinstruments, 2015.

[29] Horn, T.: ASCII-, DOS-Latin-1-, Windows-1252- und HTML-Zeichencodes sowie deutsche Tastatur-Scancodes, 1998.

[30] Castor, R.: Binary, Hex, and Octal in Python: A stroll outside the decimal system. In:

A Appendix

A.1 binäre Kodierungen

Im Folgenden soll die Darstellung der drei Zahlensystem mit der Dezimalschreibweise beispielhaft verglichen werden. Die Konversion eines binären Datenformats in die Dezimalschreibweise erfolgt über eine Skalarmultiplikation der Bits und der zugehörigen Werte. Zur Verdeutlichung wird diese Rechenoperation für das gegebene Beispiel in Gleichung A.1 durchgeführt und die 8 Bit-Abfolge 11000101 in Dezimalschreibweise konvertiert, um im Weiteren das zugehörige ASCII Zeichen zu ermitteln.

$$Dezimal: \begin{bmatrix} 1_7 & 1_6 & 0_5 & 0_4 & 0_3 & 1_2 & 0_1 & 1_0 \end{bmatrix} \cdot \begin{bmatrix} 2^{bitIndex} = 128 \\ 2^{bitIndex} = 64 \\ 2^{bitIndex} = 32 \\ 2^{bitIndex} = 16 \\ 2^{bitIndex} = 6 \\ 2^{bitIndex} = 4 \\ 2^{bitIndex} = 2 \\ 2^{bitIndex} = 1 \end{bmatrix} = 197 \tag{A.1}$$

Das auf der Zahl 16 basierende System kann vier Bits darstellen und ermöglicht es somit, Zeichenketten auf einem Viertel des ursprünglichen Raums darzustellen. Da es im Hexadezimalsystem $2^4 = 16$ Möglichkeiten gibt, das Dezimalsystem lediglich nur die Werte von 0-9 abdeckt, werden die Werte von 10 bis 15 mit den Buchstaben A-F aufgefüllt. Bei Umrechnung von Binär zu Hex werden nicht wie in Gleichung A.1 alle Bits zusammengerechnet, sondern wie in Gleichung A.2 die Bits von 11000101 in 4er Päckchen zusammengefasst.

$$\begin{aligned} Hex: 11000101 \\ [1100] &=> (1*8+1*4+0*2+0*1) = 12 => C \\ [0101] &=> (0*8+1*4+0*2+1*1) = 5 \\ [1100][0101] &=> C5 \end{aligned} \tag{A.2}$$

Zusätzlich zur Binär-, Dezimal- und Hexadezimal-Schreibweise gibt es auch die oktale Darstellung, welche durch Zusammenfassung von 3 Bits entsteht, äquivalent zu Gl.A.2 jedoch durch Zusammenfassung von 3 Bits anstelle von 4. Diese Darstellung wird selten benutzt findet aber später in dieser Arbeit in Form des ARINC-429 Label Anwendung. Die verschiedenen

Datentypen sind in Tabelle 2.1 aufgelistet. Das zugehörige ASCII-Zeichen für das Byte C5 aus Tabelle 2.1 ergibt sich nach Abgleich mit einer ASCII-Tabelle [29] als das Zeichen Å.[30]

A.2 pcapDatei-binäre Kodierung

In Tabelle A.1 ist die Binärstruktur des globalen Dateikopfes und des lokalen Framekopfes einer Pcap Datei dargestellt.

Zur kurzen Einführung wird in dieser Abbildung die Belegung der einzelnen Bytes mit Informationen dargestellt. Die binärKette zählt von links nach rechts, sowie von oben nach unten und stellt in der linken Spalte den Index der Bytes, ab Index Null startend, dar. Die Spalten 2-5 teilen dies weiter auf Belegung pro Byte auf. Abbildung 2.3 die Kodierung einer pcap-Datei dar. Diese beginnt mit einem globalen Dateikopf von 24 Byte Länge, gefolgt von einem Framekopf mit 16 Byte Länge, welcher vor jedem einzelnen Netzwerkrahmen steht. Diese Ethernet Frames bezeichnen die vom Netzwerk tatsächlich übertragenen Pakete. In Tabelle A.1 sind die einzelnen Bytes vom Global Header und Paket Header aufgeführt. Der Globale Header besitzt zunächst das 4-Byte große Feld *Magic Number*, welches festlegt, ob die folgenden Felder umgekehrt oder direkt zu lesen sind. Die Felder des globalen Headers(grün) definieren einzelne Eigenschaften der pcap Datei, wie Umkehrung, Versionsnummer, Zeitstempel, Fehlererkennung und generelle Informationen zum Netzwerk. [13]

Bytes	0 1 2 3	Description
0	Magic Number	Used to detect the file format and the byte ordering. The writing application writes 0xa1b2c3d4(using its native byte ordering format) to this field.The reading application will read either 0xa1b2c3d4 (identical) or 0xd4c3b2a1 (swapped). If the reading application reads the swapped 0xd4c3b2a1 value, it indicates that all the following fields must also be swapped.
4	Vmajor | Vminor	The version number of this file format (current version is 2.4).
8	This zone	The correction time, in seconds, between UTC and the local time zone of the following packet header time stamps. For example, if time stamps are in UTC, this zone is simply 0; if time stamps are in Central European time, such as Amsterdam or Berlin (which is UTC + 1:00), this zone must be -3600.
12	Sig flags	The accuracy of the capture timestamp. Not supported in most pcap-compatible tools
16	Snap lengths	To capture the entire Ethernet frame, this is typically set to 65535.
20	Network	Data link layer type, for example 1 for Ethernet. This can be various types such as Token Ring or FDDI.
24	Ts Sec	The date and time when this packet was captured. This value relates to seconds since January 1, 1970 00:00:00 UTC.
30	Ts USec	The time, in microseconds, when this packet was captured, as an offset to ts_sec.
34	Incl Len	The number of bytes of packet data actually captured and saved in the file.
40	Orig Len	The length of the packet, as it appeared on the network when it was captured

Tabelle A.1: pcap Global Header(grün) und Paket Header(grau)[13]

A.2.1 Netzwerktopologien

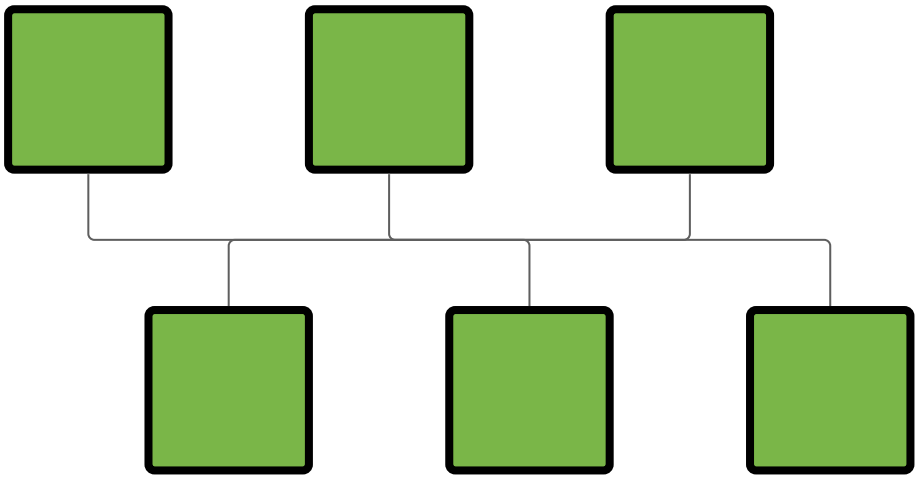

Abbildung A.1: Der Netzwerktopologietyp Bus

Auf der Anwendungsebene gibt es nun standardisierte Formate zur Übertragung von Daten. Das in der vorliegenden Messanlage auf der nächsten Ebene verwendete Format ist iNET-X. Dies ist ein proprietäres Format des Messanlagenherstellers, Curtiss-Wright.

A.2.2 Python Code für Parsen einer ARINC-429 Message

Anbei der Python Code für das Dekodieren einer ARINC-429 Message

```
"""
parse ARINC 429


"""
from Parsing.parseFunctions import *


def parseARINC429(data, config, PTPTime):
    """

    :param data:
    :param config:
    :param PTPTime:
    :return:
    """
    """this function only accepts simple 12 byte/24hex ARINC 429
       messages"""
    # first thing: check for errors in ARINC message
    if checkError(data):
        print("Error in parseARINC429: checkError ticked off")
        return None

    # separate ARINC traffic and convert into binary bites
    traffic = binaryConverter(data[16:24], 16, 2)[::-1]
```

```
25      # read label and then from configdict to know what to do for
            each label
26      label = readlabel(traffic)
27
28      try:
29          binaryFormat = config.get(label)[0]
30          # get sign-status matrix(SSM) to determine value sign and if
                computed data is alright
31          DataOkSSM, sign = SSMDecoding(traffic, binaryFormat)
32          DataOkParity = checkParity(traffic)
33      except TypeError:
34          # print("Label ", label, "in stream ", config["streamName"],
                " not defined in database.py. ")
35          pass
36      else:
37          if DataOkSSM is True and DataOkParity is True:
38              # decode elapsed time into nanoseconds
39              ElapsedTime = int(data[8:16], 16) * 1e-9 + PTPTime
40              # busId = int(binaryConverter(data[:8], 16, 2)[24:31],
                    2)
41              # try to get the format from the label. If label has no
                    according format. Catch error.
42              ARINCValue, ARINCAttributes = None, None
43
44              # iterate through the different decoding types
45              if binaryFormat == "BNR":
46                  ARINCValue, ARINCAttributes = parseBNR(traffic,
                        config.get(label), sign)
47                  return [ElapsedTime, ARINCValue, *ARINCAttributes
                        [3:6], ARINCAttributes[7]]
48              elif binaryFormat == "BCD":
49                  pass
50                  #ARINCValue, ARINCAttributes = parseBCD(traffic,
                        config.get(label), sign)
51              elif binaryFormat == "DISC":
52                  pass
53                  # ARINCValue, ARINCAttributes = parseDISC(traffic,
                        config.get(label), sign)
54              else:
55                  print("unknown bus format parseARINC429.py")
56
57          # return value only if there was valid data
58          # [3:6],[7] to not return bit information back upwards to
                inet header
59
60
61  def readlabel(Payload):
62      # TODO: read label and then decide which data type is at hand:
            BNR, BCD, discrete,...
63
64      """
65      #Label:Usage Subgroup
66
67      155-161: Maintenance
68      270-276:Discretes
69      350-354:Maintenances
70
```

```
71      :param Payload:
72      :return:
73      """
74      label = Payload[0:8]
75
76      # convert to  octal value and return
77      return binaryConverter(dataIn=label, formatIn=2, formatOut=8)
78
79
80  def parseBNR(traffic, config, sign):
81      """
82      this function will parse the Binary data in the ARINC-429 format
83
84      :param sign: tells program if the value is negative or not
85      :param traffic: get all 32 bytes of info
86      :param config: get defining parameters for ARINC-429 message
87      :return:
88      """
89      # TODO: Binary  BNR  Transmitted in fractional twos complement
            notation
90      # get SSM value
91      # get LSB and MSB position
92      # get value out of the payload
93      # convert to real number and revert the whole thing.
94      #i guess that's right. idk
95      value = int(traffic[config[2] - 1:config[1]][::-1], 2)
96
97      # Scaling. get MSB weight, LSB weight should be resolution,
            right? RIGHT? roundabout i guess.
98      # get MSB weight and work down from there
99      resolution = config[6] / (2 ** (config[1] - config[2]))
100     value = value * resolution
101
102     # TODO: check if range is okay. or perhaps do it at a later
            stage of the program.
103     """
104     a negative sign can only be added if
105         * the sign is negative and
106         * the positive Sense is not "no" ergo: there exists
107         a negative side of the variable
108     """
109     if not (config[7] == "No") and not sign:
110         value = 2 * config[6]-value
111         value = value*(-1)
112     return value, config
113
114
115 def parseBCD(traffic, config, sign):
116
117     # decode each letter for itself
118     # first letter
119
120     # Digit 1
121     if config[1]>28:
122         pass
123
124     # Digit 2
```

```
125     if config[1]>24 and config[2]<49:
126         pass
127     # Digit 3
128     # Digit 4
129     # Digit 5
130 
131     return value, config
132 
133     pass
134     # return value, config
135 
136 
137 def parseDISC(traffic, config, sign):
138     pass
139     # return value, config
140 
141 
142 def SSMDecoding(Payload, msgFormat):
143     """
144     this function gets the whole message string as well as the
            actual value format
145     it returns two booleans:
146     -isDataOk to determine whether the values are alright and it's
            computed correctly.
147     -signValue for the number sign(Vorzeichen) True for positive
            False for negative
148 
149     :param Payload: gives the whole arinc message string
150     :param msgFormat: contains if the value of ARINC message
151     is in binary coded decimals, binary numbers or discrete data.
152     :return: isDataOK and signValue
153     """
154 
155     # TODO: check for computed data and functional test. skip rest
            if any is evident.
156     SSMBits = Payload[29:31]
157     # preassign signValue
158     signValue = True
159     isDataOk = True
160     if SSMBits == "10":
161         # no computed data
162         isDataOk = False
163     elif SSMBits == "01":
164         # functionaltest
165         isDataOk = False
166     else:
167         if msgFormat == "BCD":
168             signValue = SSMBCD(Payload)
169         elif msgFormat == "BNR":
170             isDataOk, signValue = SSMBNR(Payload)
171         elif msgFormat == "DISC":
172             isDataOk = SSMDISC(Payload)
173         else:
174             raise Exception("Error 4 in SSMDecoding in parseARINC429
                    with format: ", msgFormat)
175     return isDataOk, signValue
176 
```

```
177
178 def SSMBCD(Payload):
179     """
180     evaluate SSM Binary Coded Decimal bits according to
            documentation
181     :param Payload:
182     :return:
183     """
184     SSMBits = Payload[29:31]
185     # for BCD
186     if SSMBits == "00":
187         signValue = True
188     elif SSMBits == "11":
189         signValue = False
190     else:
191         raise Exception("Error 1 in SSMDecoding in parseARINC429
                with format: BCD")
192     return signValue
193
194
195 def SSMBNR(Payload):
196     """
197     evaluate SSM Binary bits according to documentation
198     :param Payload:
199     :return:
200     """
201     SSMBits = Payload[29:31]
202     # for BNR
203     if SSMBits == "00":
204         # failure Warning
205         isDataOk = False
206     elif SSMBits == "11":
207         # working as intended
208         isDataOk = True
209         if Payload[28] == "1":
210             # sign negative
211             signValue = False
212         elif Payload[28] == "0":
213             # sign positive
214             signValue = True
215         else:
216             raise Exception("Error 2 in SSMDecoding in parseARINC429
                    with format: BNR")
217     return isDataOk, signValue
218
219
220 def SSMDISC(Payload):
221     SSMBits = Payload[29:31]
222     if SSMBits == "11":
223         # Failure Warning
224         isDataOk = False
225     elif SSMBits == "00":
226         # Verified data, normal operation
227         isDataOk = True
228     else:
229         raise Exception("Error 3 in SSMDecoding in parseARINC429
                with format: DISC")
```

```
230     return isDataOk
231
232
233 def checkError(Payload):
234     """
235     check and evaluate error Codes
236
237     :param Payload:
238     :return:
239     """
240     binaryInfoLine = binaryConverter(Payload[:8], 16, 2)
241     # convert to Error Bit & error code
242     errorBit, errorCode = binaryInfoLine[0], binaryConverter(
            binaryInfoLine[1:6], 2, 16)
243     # line to retrieve the bus id from info line
244     # busId = int(binaryConverter(Payload[:8], 16, 2)[24:31], 2)
245
246     if errorBit == 1:
247         isNotGood = True
248     else:
249         isNotGood = False
250
251     return isNotGood
252
253
254 def checkParity(traffic):
255     """
256     The number of Logic 1s transmitted in each word is an odd number
            ,
257     with bit 32 being set or cleared to obtain the odd count
258
259     :param traffic:
260     :return: isDataOk
261     """
262     if traffic.count("1") % 2 == 1:
263         isDataOk = True
264     else:
265         isDataOk = False
266         print("Parity is very much not ok")
267     return isDataOk
```