DHBW
Duale Hochschule
Baden-Württemberg
Mannheim

Studiengang Informationstechnik

Bericht zum Modul Praxis II

Praxisphase 4: 25.6. - 07.09.2018

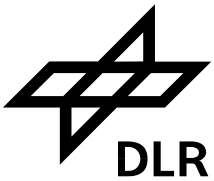
Thema 1: Aufbereitung einer SQLite-Datenbank für AIS-Signale

Thema 2: Analyse von Kalibrierpulsen für das V-SAR System

von **Beitler, Daniel**

- Matrikelnr.: 9247761 -

- Kurs: TINF16ITIN -



Deutsches Zentrum für
Luft- und Raumfahrt e.V.
in der Helmholtz-Gemeinschaft

Standort: Oberpfaffenhofen

Institut für Hochfrequenztechnik und
Radarsysteme
Abteilung: SAR-Technologie

Dr. Rolf Scheiber

Kurzfassung

Das Ziel dieser Arbeit ist die Entwicklung und Umsetzung von Programmen zur Verarbeitung von Daten, die im Rahmen der DBFSAR- und V-SAR-Missionen erzeugt werden. Hierbei haben sich zwei Projekte ergeben, die diese Daten behandeln, beziehungsweise Kalibrierpulse analysieren. Zuerst wird primär auf Daten, wie sie von Schiffen zur Regelung des Schiffsverkehrs gesendet werden, eingegangen. Auf dieser Basis werden hier Erkenntnisse zur Entwicklung von Methoden zum Umgang mit Datenbanken und JSON-Dateien, sowie zur Modifikation von bereits vorhanden Algorithmen präsentiert. Dabei wird zuerst auf die jeweilige Aufgabenstellung eingegangen. Anschließend erfolgt eine Entwicklung, beziehungsweise Darstellung des verwendeten Programmentwurfes. Im Bereich der Datenbankmanipulation wird auch näher auf die Implementierungsaspekte, die sich in dieser Hinsicht ergeben haben, eingegangen, um mögliche Lösungen für ähnliche Probleme darzustellen. Dabei werden im Bereich des Entwurfs Graphen, beziehungsweise im Implementierungsteil Ausschnitte aus dem Quellcode verwendet, um die dargestellten Inhalte anschaulicher zu beschreiben.

Abstract

The main aim of this report is the concept and implementation of methods to manipulate data gained by the most recent DBFSAR and V-SAR sensor of the DLR. During this work the primary focus will be set on AIS data, information gathered by the automatic identification system used by most ships today. During the course of this work two projects have emerged. The first one focuses mostly on the combination of tables containing the previously mentioned AIS-data. The second one discusses the modification of an already existing algorithmn to create comparing graphs from previously collected callibration data. In both parts of this report the main goal of each project will be displayed and the architecture of the developed algorithm will be explained. The part considering the manipulation of databases will include a chapter discussing various aspects of the actual imlementation. In order to display the described features several graphics and extractions from the source code will be used.

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich
die vorliegende Arbeit selbstständig und nur unter
Verwendung der angegebenen Quellen und
Hilfsmittel angefertigt habe.

Oberpfaffenhofen, der 7. Januar 2019

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Tabellenverzeichnis

Abkürzungsverzeichnis

DLR	**D**eutsches Zentrum für **L**uft- und **R**aumfahrt
HR	Institut für **H**ochfrequenztechnik und **R**adarsysteme (im DLR)
SAR	**S**ynthetic **A**perture **R**adar
DBFSAR	**D**igital **B**eamforming **S**ynthetic **A**perture **R**adar
AIS	**A**utomatic **I**dentification **S**ystem
V-SAR	**SAR**-System für **V**erkehrsanwendungen

1 Einleitung

Das Institut für Hochfrequenztechnik und Radarsysteme (HR) des Deutschen Zentrum für Luft- und Raumfahrt (DLR) erforscht hauptsächlich Radarsysteme mit synthetischer Apertur (Synthetic Aperture Radar (SAR)) und befasst sich dabei unter anderen mit der Konzeption und Entwicklung neuartiger Systeme [2]. Innerhalb des Instituts HR entwirft die Abteilung SAR Technologie flugzeuggestützte Radarsysteme und Anwendungen zur Signalverarbeitung [3]. Dabei befasst sich die Gruppe SAR Signalverarbeitung mit der zuvor schon erwähnten Entwicklung von Algorithmen zur Analyse und Verarbeitung von eingehenden SAR-Daten [4].

In diesem Rahmen wurden 2018 mehrere Testflüge mit dem System DBFSAR (Digital Beamforming Synthetic Aperture Radar), beziehungsweise V-SAR (SAR für Verkehrsanwendungen) durchgeführt. Dabei wurden verschiedene Neuerungen getestet, wobei zur Analyse und Verarbeitung benötigte Methoden noch nicht vorhanden waren. Auf zwei dieser Veränderungen wird hier genauer eingegangen und die Konzeption und Entwicklung von Verarbeitungsmodulen zur Behandlung von durch diese Änderungen bereitgestellten Daten, vorgestellt.

Dabei wird im ersten Teil auf die Verarbeitung von AIS-Daten (Automatic Identification System), die zur Verfolgung und Überwachung des Schiffsverkehrs dienen und hier erstmals bei einem durch das Institut HR durchgeführten Radarflug aufgezeichnet wurden, eingegangen.

Der zweite Teil dieser Arbeit befasst sich mit dem Entwurf eines Moduls zum Auswerten von Radar-Kalibrierpulsen, die hierbei über einen zuvor nicht verwendeten Modus aufgezeichnet wurden.

2 Aufbereitung einer SQLite-Datenbank für AIS-Signale

Bei neu durchgeführten Radaraufzeichnungen mittels des SAR-Systems DBFSAR/V-SAR durch das Institut HR wurden auch sogenannte AIS-Daten (Automatic Identification System) aufgezeichnet. Diese Daten werden automatisch von den meisten Schiffen gesendet und beinhalten dabei einige Angaben bezüglich des Senders, wie zum Beispiel zu den Ausmaßen, dem Zielhafen oder der Beladung des Schiffes, und die aktuellen Position desselben. Dabei werden diese Informationen gesendet und - in diesem Fall - an Bord der Radarplattform (des Flugzeugs) empfangen, anschließend entschlüsselt und in zwei verschiedenen Tabellen innerhalb einer Datenbank gespeichert. Diese Datensätze enthalten respektive statische Informationen und Positionsdaten.

Nun sollen zur vereinfachten Verarbeitung diese Aufzeichnungen in einer neuen Tabelle kombiniert (sogenanntes "mergen", der Prozess wird als "Merging" bezeichnet) werden Dafür soll ein Python-Skript verwendet werden, das sich über einen Methodenaufruf starten lässt. Ein weiteres Skript soll es im Anschluss ermöglichen, aus den gesammelten Daten eine JSON-Datei zu erstellen, die zur weiteren Analyse der Radardaten und der Schiffsdaten verwendet werden kann.

2.1 Datendefinition

Bevor erste Entwürfe erstellt werden können, ist es substantiell, dass zuerst ein Überblick über die Struktur und Definition der gegebenen und geforderten Daten geschaffen wird. Dafür ist für die vorhandene Datenbank und für die neu zu implementierende Tabelle schon eine Definition vorhanden [1].

Im Gegensatz dazu steht für die verlangte JSON-Datei lediglich eine vorab generierte Datei als Vergleichsobjekt zur Verfügung. Aufgrund dessen war es hier nur über das sogenannte "Reverse-Engineering" (rückwärts gerichtete Entwicklung, vom Code zur Spezifikation) möglich Objekt-, beziehungsweise Klassendiagramme zu erstellen.

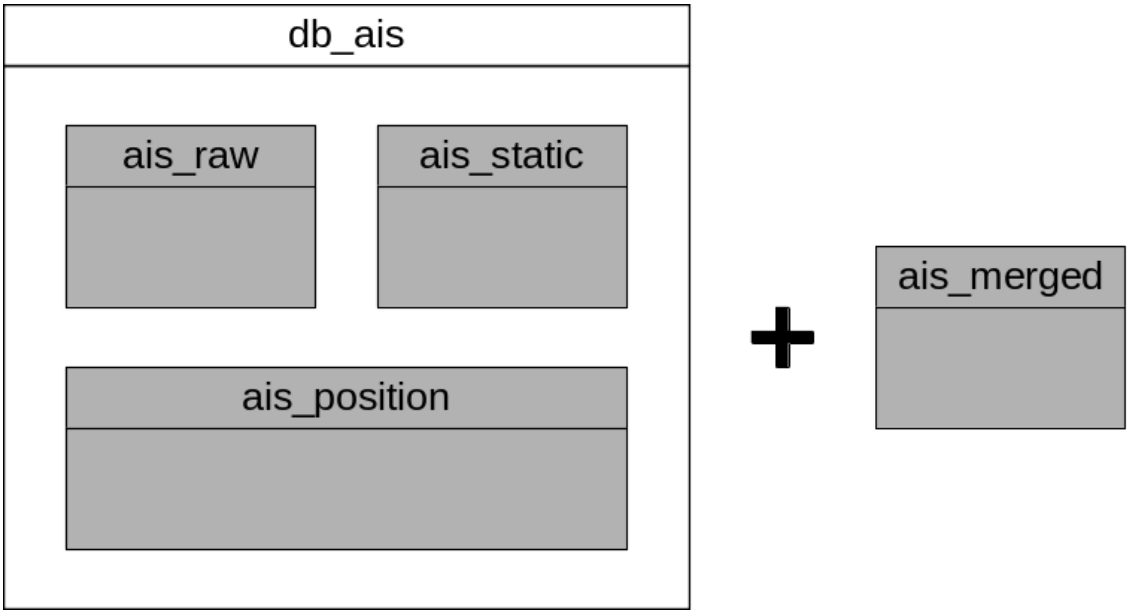


Abbildung 1: Schema der Datenbank nach [1]

Datenbank: [1] definiert für die Datenbank drei Tabellen, die schon implementiert sind und bei der Datenerfassung mit gefüllt werden, sowie eine Tabelle, die die zusammengefassten Daten enthalten soll, wobei nicht festgeschrieben ist, in welchen Datensatz diese einzufügen ist [1]. Dies wird in Abbildung 1 dargestellt.

Dabei ist die Tabelle *ais_raw* so definiert, dass sie sich in *db_ais* befindet und die durch AIS gesendeten Rohdaten als String, sowie den Zeitpunkt des Schreibens enthält. Diese Sequenz wird anschließend verarbeitet und abhängig vom Typ des AIS-Signals entweder in *ais_position* oder *ais_static* übertragen. Dort werden Informationen zur Position, wie zum Beispiel die aktuellen Koordinaten oder der Bewegungsvektor (Geschwindigkeit und Rotation), oder zum Schiff allgemein, wie zum Beispiel Daten zum Schiffstyp oder zum Ziel, gespeichert. Hierbei zeigt Abbildung 2 die Definition der Tabellen nach [1].

Bei genauer Betrachtung der in Abbildung 2 dargestellten Tabellen ist ersichtlich, dass bestimmte Attribute von *ais_merged* in den beiden zugrunde liegenden Tabellen auftauchen, oder einen anderen Datentyp besitzen. Auf diese Umstände wird im Verlauf dieser Arbeit noch weiter eingegangen.

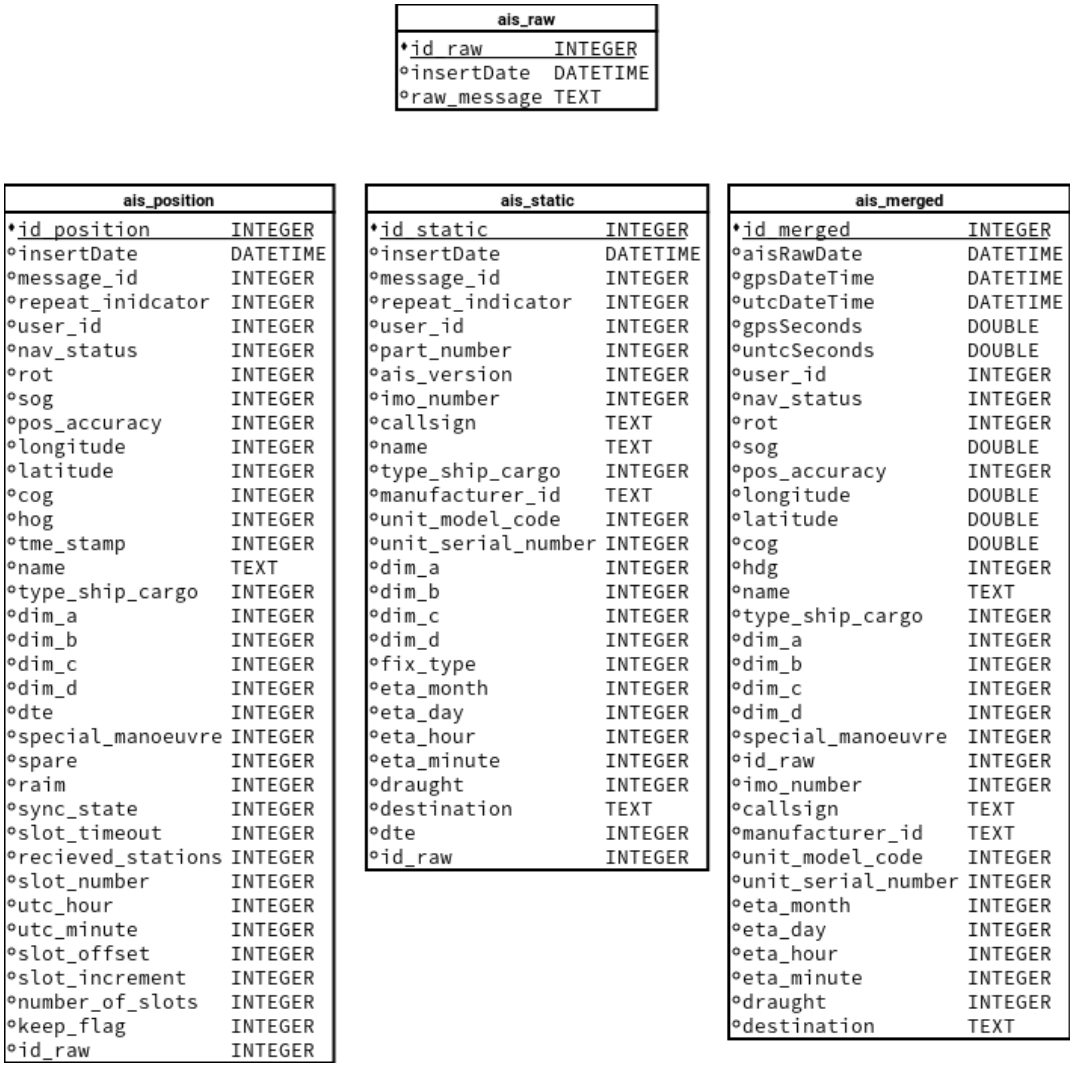


Abbildung 2: Definition der Tabellen nach [1, S. 3–9]

JSON: Da zur Definition der JSON-Datei lediglich eine fertig generierte Datei, aber keine Dokumentation vorhanden ist, kann nur diese Datei zur Erstellung einer Definition herangezogen werden. Hieraus ergibt sich die in Abbildung 3 dargestellte Struktur. Dabei können in einer JSON-Datei mehrere Objekte des Typs *ShipClass* abgelegt werden, wobei der Name dieser Objekte jeweils die ID des entsprechenden Schiffes ist. Jede Schiffsinstanz vom Typ *ShipClass* enthält ein Datenblatt *DataClass*, in dem einige Kerndaten, wie zum Beispiel die Größe des Gefährts oder sein Rufname, hinterlegt sind. Außerdem kann jedes Schiff

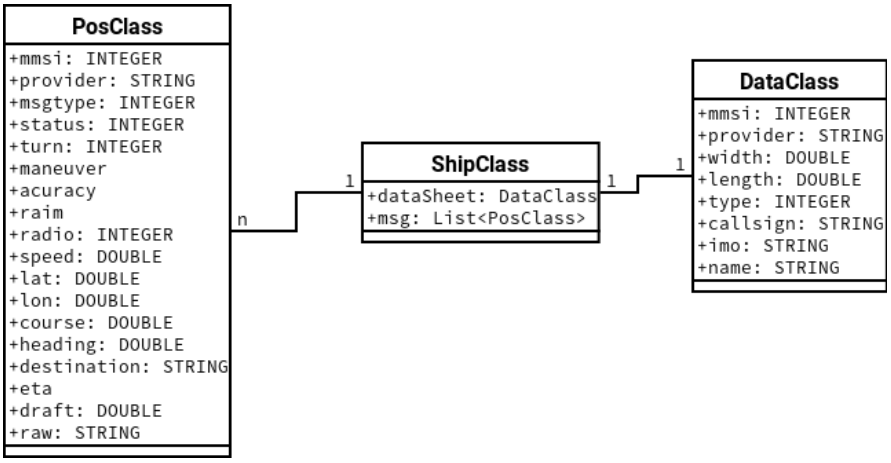


Abbildung 3: Klassendiagramm der JSON-Datei

mehrere Elemente des Typs *PosClass* beinhalten. Diese Positionsobjekte werden nach den *gpsSeconds* benannt, die schon in Abbildung 2 in der Tabelle *ais_merged* definiert sind. Außerdem werden hier Informationen zur aktuellen Position des Schiffes, wie die Koordinaten oder der Bewegungsvektor, gespeichert.

2.2 Programmentwurf

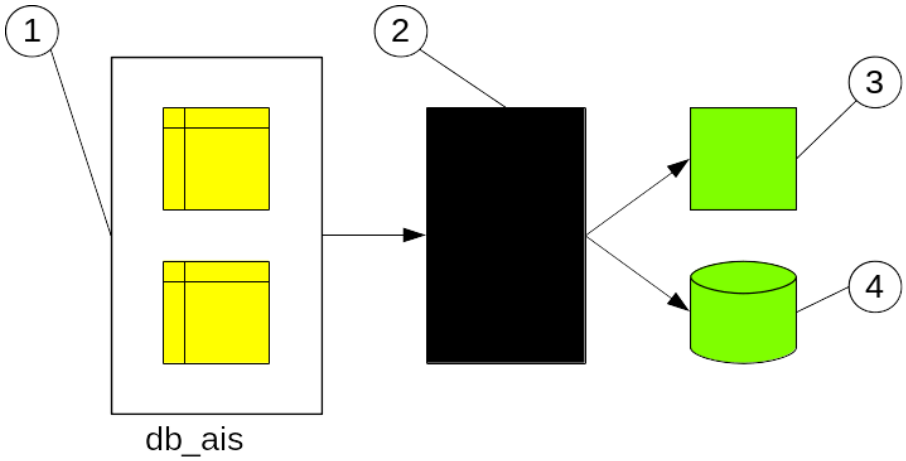


Abbildung 4: Erster Entwurf des Algorithmus.

Erste Überlegungen in enger Anlehnung an die Aufgabenstellung führten zu einem Entwurf, der ein Verfahren vorsah, das als Eingangsparameter den SQL-Dump, der durch die Radarflüge erzeugt wird, benötigt und abschließend eine SQLite-Datenbank mit den zusammengefassten Daten und die geforderte JSON-Datei ablegt. Dieser frühe Entwurf wird in Abbildung 4 dargestellt. Dabei wird der Eingangsdump mit (1) , das Verfahren als "Blackbox" mit (2) und die Ausgangs-JSON-Datei, sowie die SQLite-Datenbank mit (3) , bzw. (4) markiert.

Aufgrund späterer Überlegungen zum Einsatzbereich des Verfahrens (diesmal unter dem Gesichtspunkt der eventuellen Verwendung während der laufenden Aufnahme im Flugzeug) stellte sich die Lösung mit den getrennten Datenbanken für die Ursprungs-und Enddaten als umständlich heraus. Außerdem kann hierbei auch auf eine Verfahren zur gemeinsamen Zusammenfassung der Tabelleneinträge und Generierung der JSON-Datei verzichtet werden. Hierbei ist eine Methode für das Merging und das Auslesen und separate Speichern von Schiffsdaten aufgrund der Operationszeit (solch eine Methode wird im Betrieb in bestimmten Zeitintervallen aufgerufen) und der Tatsache, dass die JSON-Daten erst nach Beendigung der Aufnahme nutzbar sind, ohne eine fundierte Grundlage. Daher sind in diesem Zusammenhang zwei getrennte Operationen sowohl aus der Sicht des Zeitmanagements und des Aufwandsmanagements eine optimale Lösung. Auch ist hierbei die Verwendung von zwei getrennten SQL-Datenbanken fragwürdig. Da aufgrund der Signalaufnahme schon eine Datenbank als SQL-Instanz vorhanden ist und sich bei der Verwendung eines gemeinsamen Systems für die Ursprungs- und die zusammengefassten Daten diese an dem selben Ort befinden. So können beim späteren Erzeugen der JSON-Datei, wobei auf Einträge aus verschiedenen Tabellen zugegriffen werden muss, alle Parameter in einem Objekt vorhanden sein.

In Abbildung 5 werden die Schreib- und Lesezugriffe der aufgrund der vorher angestellten Überlegungen entworfenen Methoden gezeigt. Hieraus wird ersichtlich, dass zur Erzeugung der JSON-Datei (dieser Prozess wird in der linken Seite der Abbildung dargestellt) in diesem Fall auf drei Tabellen, die sich innerhalb von nur einer Datenbank befinden, zugegriffen werden muss. Dabei erfolgt die Abfrage der meisten Informationen aus *ais_merged* (zum Beispiel die Ausmaße des Schiffes oder seine Identifikationsnummer). Andere Daten müssen hingegen aus *ais_raw* (*raw_message*, die Rohnachricht) und *ais_position* (Nachrichtentyp und Daten zur GPS-Überprüfung).

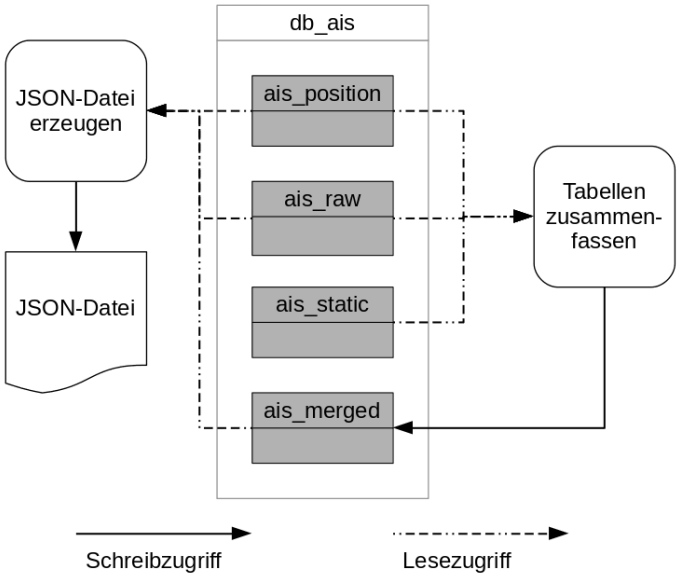


Abbildung 5: Zugriffsstruktur des finalen Entwurfs

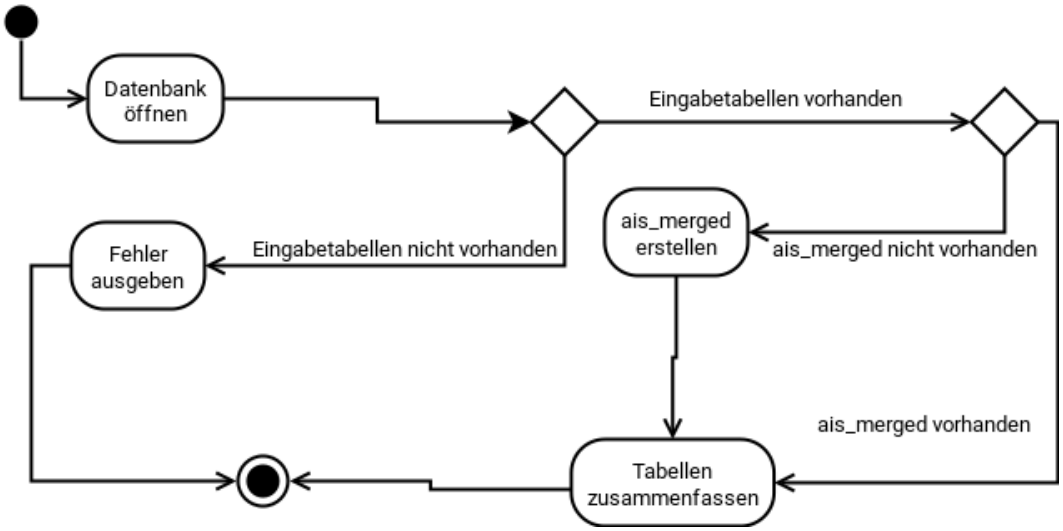


Abbildung 6: Aktivitätsdiagramm des Algorithmus zum Kombinieren von Daten

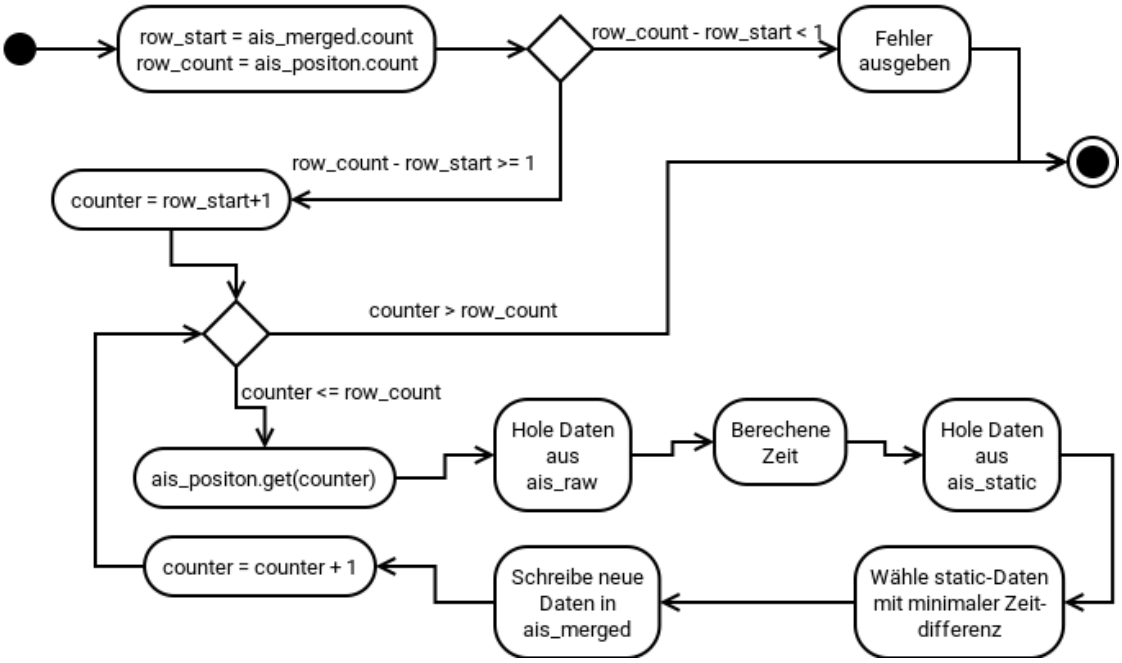


Abbildung 7: Aktivitätsdiagramm zum Zusammenfassen von Tabellen in Abbildung 6

2.2.1 Entwurf des Kombinationsalgorithmus

Aus diesen Vorgaben lässt sich ein erstes Aktivitätsdiagramm für das Kombinieren der Tabellen, wie es in Abbildung 6 dargestellt ist, entwickeln. Hierbei wird zuerst die Datenbank geöffnet, beziehungsweise meldet sich der Algorithmus am Datenbankserver mit den Zugangsdaten, die zuvor übergeben wurden, an. Anschließend wird überprüft, ob die Tabellen *ais_raw*, *ais_position* und *ais_static* in dieser Datenbank vorhanden sind. Wenn dies nicht der Fall ist, so wird eine Fehlermeldung ausgegeben und die Ausführung dieses Prozesses beendet. Wenn diese Bedingung aber erfüllt ist, so wird die Tabelle *ais_merged* erstellt, sofern sie nicht bereits vorhanden ist. Daraufhin wird mit dem eigentlichen Vorgang des Mergen, wie er in Abbildung 7 gezeigt wird, begonnen.

Dabei wird zuerst die Anzahl an Zeilen in *ais_position* und *ais_merged* (als *row_count*, bzw. *row_start*) ermittelt. Danach wird die Differenz aus diesen beiden Zahlen ($row\_count - row\_start$ gebildet. Ist das Ergebnis kleiner als 1, so sollte es in *ais_position* keine Einträge geben, die nicht schon in *ais_merged* vorhanden sind, sofern keine der beiden Tabellen manuell verändert wurde. Folglich kann daraufhin der Prozess beendet werden. Hat die zuvor gebildete

Differenz aber eine Wert, der größer oder gleich 1 ist, so wird eine Schleife ausgeführt, die solange läuft, bis alle noch nicht abgedeckten Einträge in *ais_positon* behandelt worden sind. Dafür wird zuerst der entsprechende Eintrag abgefragt und anschließend der Schreibzeitpunkt für den entsprechenden Datensatz in *ais_raw* ausgelesen. Aus dieser Zeit werden in Verbindung mit einigen anderen Parametern aus *ais_position* die *gpsDateTime*, die *utcDateTime*, die *gpsSeconds* und die *utcSeconds* berechnet. Anschließend lädt der Algorithmus alle mit der aktuellen Position verbundenen Einträge aus *ais_static*. Diese werden über die ID des Senders, also des sendenden Schiffes bestimmt. Aus dieser Menge wird der Datensatz mit der geringsten Zeitdifferenz zum Positionseintag bestimmt und mit diesem zusammengeführt. Abschließend werden die zusammengefassten Daten in *ais_merged* geschrieben.

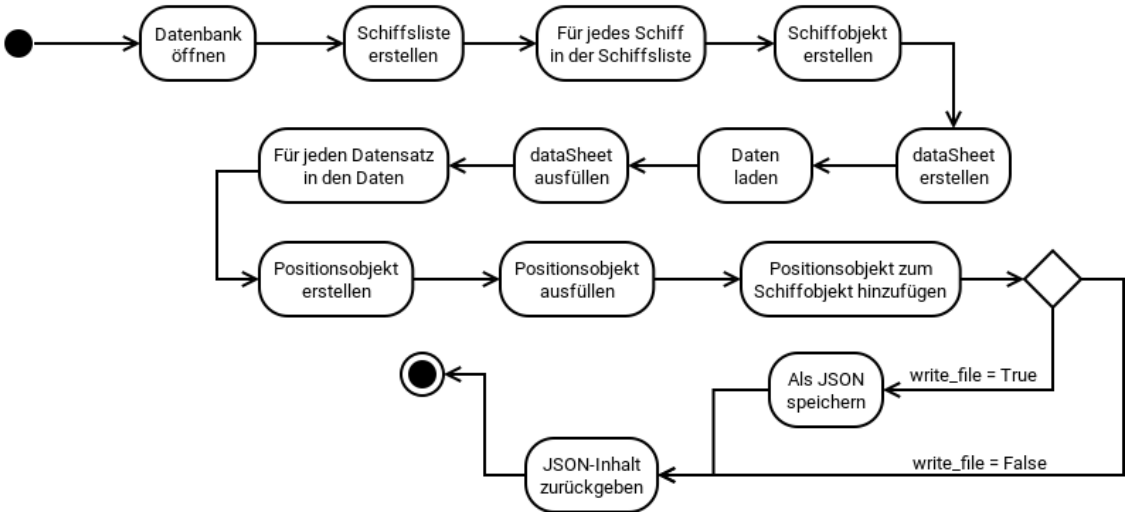


Abbildung 8: Aktivitätsdiagramm zum Generieren von JSON-Dateien aus AIS-Aufzeichnungen

2.2.2 Generieren von JSON-Dateien für AIS-Signale

Für das Erstellen von JSON-Dateien meldet sich der Algorithmus, wie auch schon zuvor, bei dem Datenbanksystem mit den AIS-Daten mit übergebenen Anmeldedaten an. Daraufhin wird eine Liste mit allen Schiffen, die in *ais_merged* vorhanden sind, erstellt. Diese Übersicht wird, wie in Abbildung 8 dargestellt, verwendet, um für jedes Schiff in der Schiffsliste ein Objekt zu erstellen. Für jedes dieser Objekte wird ein *dataSheet* erstellt und hinzugefügt,

welches die Metadaten des Schiffes, wie seine Länge oder den Zielhafen, enthält. Dafür werden alle vorhandenen Datensätze für das aktuell zu behandelnde Schiff über die Schiffs-ID geladen. Diese Datensätze sind weiterhin der Grundstock für die Positionsliste, die an jedes Schiffsobjekt angehängt wird. Dabei wird für jede aus der Datenbank abgefragte Zeile ein neues Objekt erstellt, welches die Daten zu dieser Position enthält. Abschließend wird geprüft, ob der Nutzer die erfassten Daten als eine JSON-Datei speichern möchte oder nicht. Sollte dies der Fall sein, so wird die Datei abgelegt. Schlussendlich werden die im Arbeitsspeicher vorhandenen Elemente als Objekt an den Aufrufer der Methode zurückgegeben.

2.3 Implementierungsaspekte

Für das gesamte Projekt wird die Programmiersprache Python zugrunde gelegt, wobei in diesem Fall die Distribution *Python 3.5* verwendet wird. Als Datenbankverbindung kann theoretisch jedes MySQL-Paket verwendet werden, da die Datenbank auf dem System MariaDB, welches ein freier Ableger von MySQL ist, aufbaut, wobei hier*pymysql* zum Einsatz kommt. Zur Umrechnung von Daten kommt bei der Verschmelzung der Tabellen eine von Stefan V. Baumgartner entwickelte Methode zum Einsatz. Die Generierung der JSON-Datei beruht auf den Pythonpaketen *json* und *collections*.

2.3.1 Implementierung des Kombinationsalgorithmus

Hierbei wird zuerst, wie schon zuvor in Kapitel 2.2.1 gezeigt, eine Datenbankverbindung geöffnet, deren Parameter über den Aufruf der Methode *ais_merge* übergeben werden. Da dieser Algorithmus keine weiteren Parameter benötigt sind die Anmeldedaten die einzigen Variablen, die hier benötigt werden. Daraus ergibt sich der im Quelltext 1 gezeigte Methodenkopf.

```
1 def ais_merge(host, user, password, db):
```

Quellcode 1: Kopf der Methode zum Kombinieren der Tabellen

Anschließend werden die Variablen für das Zählen der bereits vorhandenen und der gesamten Zeilen definiert. Hier erfolgt auch die Erstellung einer leeren Liste für das Überprüfen der vorhandene Tabellen, wie es der Quellcode 2 zeigt.

```
1 row_count = 2
2 row_start = 1
```

```
3 existing_tables = []
```

Quellcode 2: Variablendeklaration

Anschließend erfolgt ein Abfragen alle vorhandenen Tabellen über ein *SELECT* auf das *INFORMATION_SCHEMA* der Datenbank. Dabei erhält die Methode ein Datenfeld, das anschließend in eine Liste umgewandelt wird (Quellcode 3).

```
1 cursor.execute("SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE
      TABLE_SCHEMA=%s;", (db,))
2 existing_tables_tmp = cursor.fetchall()
3
4 for el in existing_tables_tmp:
5     existing_tables.append(el[0])
```

Quellcode 3: Auslesen der Tabellen

Anschließend werden die in Kapitel 2.2.1 gezeigten Schritte durchgeführt. Bei der Berechnung von *row_start*, beziehungsweise *row_count* wird die von MySQL bereitgestellte Funktion *COUNT* verwendet, die bei der Verwendung , wie sie in Quellcode 4 gezeigt wird, die Summe aller Zeilen in der angesprochenen Tabelle zurückgibt [5]. Diese Methode ist aber nicht die einzige Möglichkeit, um die Anzahl an Zeilen in einer Tabelle abzufragen. Eine der weiteren Optionen ist die Verwendung von *SHOW TABLE STATUS*. Dabei wird auch die Anzahl an Zeilen angezeigt [6]. Diese Herangehensweise ist zwar um einige schneller als die hier verwendete, aber auch umständlicher zu integrieren, da *SHOW TABLE STATUS* nicht unbedingt mit anderen SQL-Befehlen verwendet werden kann. Auch ist die Ausführungsgeschwindigkeit in diesem Fall nicht ausschlaggebend, da sich erst ab einer Anzahl von mehreren Millionen Zeilen ein deutlicher Unterschied zeigt. Außerdem ist die hierbei ermittelte Zahl nur eine Annäherung, wohingegen der Befehl *COUNT(*)* die tatsächliche Anzahl zurückgibt [6].

```
1 cursor.execute("SELECT COUNT(*) FROM ais_position;")
```

Quellcode 4: Verwendung von COUNT()

Nach dem zuvor in Kapitel 2.2.1 erwähnten Abgleich zwischen der aktuell in *ais_merge* vorhandenen und der tatsächlichen Anzahl an Zeilen in *ais_position* wird eine *for*-Schleife verwendet. Dabei ist *row_start* die untere und *row_count* die obere Grenze des verfügbaren Zahlenraumes. Anschließend werden, wie in Quellcode 5 die Variablen für die Berechnung der Zeiten und für das Ermitteln des zugehörigen Eintags in *ais_static* deklariert.

```
for i in range(row_start, row_count):
    pos_time_stamp = None
    pos_utc_hour = None
    pos_utc_minute = None
    static_list = []
```

Quellcode 5: Schleife für das durchgehen der verfügbaren Zeilen und Variablendeklaration

Hierauf erfolgt die Erzeugung einer Pythonstruktur, um die Daten für den Eintrag in *ais_merged* zwischenzuspeichern. Mit dem aktuellem Index der *for*-Schleife werden daraufhin die Daten der entsprechenden Zeile in *ais_position* abgerufen, als Liste zwischengespeichert und in die entsprechenden Positionen, auch mit möglichen Umrechnungen, eingetragen. Hierbei ist zu beachten, dass manche Parameter, die hierbei erfasst werden, auch in *ais_static* vorhanden sind, wie es aus Abbildung 2 im Kapitel 2.1 hervorgeht. Sollten diese Parameter in *ais_position* vorhanden sein (sie haben nicht den Wert *Null*), so werden diese bevorzugt verwendet. Die durch die Definition festgelegten Zeiten, wie zum Beispiel *gpsDateTime* oder *utcSeconds*, werden mit dem *timeCalc*-Algorithmus, den Stefan V. Baumgartner freundlicherweise bereitgestellt hat berechnet. Dafür wird auch die Aufzeichnungszeit aus *ais_raw*, wie es im Quellcode 6 gezeigt wird, verwendet.

```
ais_data = {"aisRawDate":None,
            "gpsDateTime": None,
            ...
            "dim_a":None,
            ...
            "draught": None,
            "destination": None}

cursor.execute("SELECT user_id,"\
               ...
                       "dim_a,"\
                       ...
                       "id_raw,"\
                       "time_stamp,"\
                       "utc_hour,"\
                       "utc_minute"\
                       " FROM ais_position WHERE id_position=%s;", (i,))
pos_data=cursor.fetchone()

```

```
20 ais_data["user_id"]=pos_data[0]
21 ...
22 ais_data["dim_a"]=pos_data[11]
23 ...
24 ais_data["id_raw"]=pos_data[16]
25
26 pos_time_stamp = pos_data[17]
27 pos_utc_hour = pos_data[18]
28 pos_utc_minute = pos_data[19]
29
30 cursor.execute("SELECT insertDate FROM ais_raw WHERE id_raw=%s;", (
       ais_data["id_raw"],))
31 ais_data["aisRawDate"]=cursor.fetchone()[0]
32 ...
33 time_vals = ais_timeCalc.timeCalc(ais_data["aisRawDate"], pos_utc_hour,
       pos_utc_minute, pos_time_stamp)
34 ais_data["gpsDateTime"]=time_vals[0]
35 ...
36 ais_data["utcSeconds"]=time_vals[3]
```

Quellcode 6: Pythonstruktur für die temporäre Aufnahme der Daten und Lesen, bzw. Einfügen derselben

Im nächsten Schritt wird hierauf, wie im Quellcode 7 dargestellt, eine Liste aller zu der Nutzernummer (*user_id*) gehörenden Einträge in *ais_static* abgefragt. Dabei werden, um eine übermäßige Belastung des Arbeitsspeichers zu vermeiden, lediglich die zur Bestimmung der Zeitdifferenz benötigten Felder abgefragt. Da einer dieser Parameter aber in *ais_raw* liegt werden die beiden Tabellen über ein *INNER JOIN* auf *id_raw* kombiniert.

Anschließend wird aus der Rückgabe, die die Datenbank bereitstellt, die zuvor schon initialisierte Liste mit benannten Pythonstrukturen (sogenannten Dictionaries) erstellt. Abschließend wird über eine Schleife ermittelt, welcher Datensatz die geringste Zeitdifferenz zu dem Eintrag in *ais_position* besitzt.

```
1 cursor.execute("SELECT ais_static.id_static, ais_raw.insertDate FROM"\
2                " ais_static INNER JOIN ais_raw ON ais_static.id_raw=
                     ais_raw.id_raw"
3                " WHERE ais_static.user_id=%s;", (ais_data["user_id"],))
4 static_tmp = cursor.fetchall()
5
6 for static_el in static_tmp:
```

```
7     static_list.append({"id_static":static_el[0],\
8                         "time_diff":abs((ais_data["aisRawDate"] -
                              static_el[1]).total_seconds())})
9 min_curr = None
10 for static_el in static_list:
11     if min_curr is None:
12         min_curr = static_el
13     else:
14         if min_curr["time_diff"]<static_el["time_diff"]:
15             min_curr = static_el
```

Quellcode 7: Ermitteln des *ais_static*-Eintrages mit der geringsten Zeitdifferenz zum *ais_position*-Eintrag

Schlussendlich werden die für das vollständige Ausfüllen des neuen Eintrags in *ais_merged* benötigten Daten abgefragt und übertragen, wobei, wie schon zuvor erwähnt, Daten aus *ais_position* bevorzugt werden (Quellcode 8).

```
1 cursor.execute("SELECT imo_number,"\
2                 ...
3                 "dim_a,"\
4                 ...
5                 "draught,"\
6                 "destination"\
7                 " FROM ais_static WHERE id_static=%s", (min_curr["
                      id_static"],))
8 static_data = cursor.fetchone()
9
10 ais_data["imo_number"]=static_data[0]
11 ...
12 ais_data["draught"]=static_data[15]
13 ais_data["destination"]=static_data[16]
14
15 if ais_data["dim_a"] is None:
16     ais_data["dim_a"]=static_data[7]
```

Quellcode 8: Einfügen von Daten aus *ais_static* in die Pythonstruktur zu *ais_merged*

Abschließend werden die so kombinierten Daten in die Tabelle *ais_merged* geschrieben und die Änderungen werden, wenn es keine weiteren Zeilen zum Übertragen gibt, mit einem *COMMIT* in der Datenbank fest gespeichert, um einen Verlust zu vermeiden.

2.3.2 Implementierung des Algorithmus zum Generieren einer JSON-Datei aus einer AIS-Datenbank

Wie zuvor schon beim Kombinieren der Tabellen muss auch hier eine Datenbankverbindung geöffnet werden. Daher müssen auch die schon zuvor beschriebenen Anmeldedaten übergeben werden. Außerdem wird noch ein Pfad zum Abspeichern der JSON-Datei und ein Wahrheitswert, ob überhaupt gespeichert werden soll benötigt. Daraus ergibt sich der im Quellcode 9 dargestellte Methodenkopf.

```
1 def ais_to_json(host, user, password, db, file_path="standard.json",
      write_file=True):
```

Quellcode 9: Methodenkopf des Algorithmus zum Generieren einer JSON aus einer Datenbank

Nach dem Öffnen der hier benötigten Datenbankverbindung wird zuerst eine geordnete Pythonstruktur erstellt, um Objekte der im Kapitel 2.1 definierten Klasse *ShipClass* aufzunehmen. Die in dem erwähnten Kapitel definierten Klassen werden zwar nicht als Klasse, sondern als Strukturen definiert, sind aber in sonstiger Hinsicht identisch mit der Definition.

Anschließend wird von der Tabelle *ais_merged* eine Liste mit allen individuellen Schiffsnummern bereitgestellt. Hierfür wird der SQL-Befehl *DISTINCT*, der alle Duplikate entfernt, in der Abfrage verwendet.

Für jedes hierfür erhaltene Schiff wird ein Objekt erstellt, dessen Positionsdaten nun abgefragt werden. Da die in der Klasse *DataClass* enthaltenen Werte für jede abgefragte Position gleich sind, können sie aus dem ersten verfügbaren Datensatz kopiert werden. Anschließend wird für das aktuelle Schiff jede Position unter der Verwendung der für *ais_merged* berechneten *gpsSeconds* gespeichert. Das Schiffsobjekt seinerseits wird unter seiner Nummer in die zuerst generierte Struktur abgelegt. Abschließend wird das JSON-Objekt auf Wunsch des Nutzers gespeichert und als Rückgabewert zurückgegeben.

2.4 Praktischer Einsatz der entwickelten Anwendungen

Bisher ist der Einsatz der Methode zum Kombinieren von Tabellen bei einer AIS-Datenbank nur mit bereits abgeschlossenen Datensätzen möglich, da ein Einsatz im laufenden Betrieb an Bord eines Flugzeugs eine operative Radarmission erfordert, die bisher aber noch nicht

durchgeführt wurden.

Im offline Betrieb wurde das Verfahren bisher auf die Vollständigkeit der erzeugten Tabelle hin untersucht, wobei sich durch einen einfachen Vergleich der Ergebnisse einer SQL-Operation auf den Tabellen *ais_position* und *ais_merged* ergeben hat, dass die Kardinalität dieser Mengen gleich ist. Außerdem haben stichpunktartige Kontrollen jeweils das erwartete Ergebnis gezeigt.

Die durch den zweiten hier vorgestellten Algorithmus generierte JSON-Datei kann als solche nicht auf eine Korrektheit geprüft werden. Aber auf Nachfrage hat sich herausgestellt, dass der Algorithmus, für den dieses Format bestimmt ist die erwarteten Resultate ergeben hat. Dies kann man in Abbildung 9 und Abbildung 10 sehen. Diese Bilder zeigen eine Überlagerung in Google Earth. Hierfür wird ein Radarbild (in diesem Fall von Regensburg) auf den Globus gelegt und mit einer .kml-Datei (enthält Schiffsdaten), die aus der hier generierten JSON-Datei erzeugt wird. Vor allem Abbildung 10 zeigt die Übereinstimmung zwischen Schiffsposition und Markierung mittels AIS-Daten.

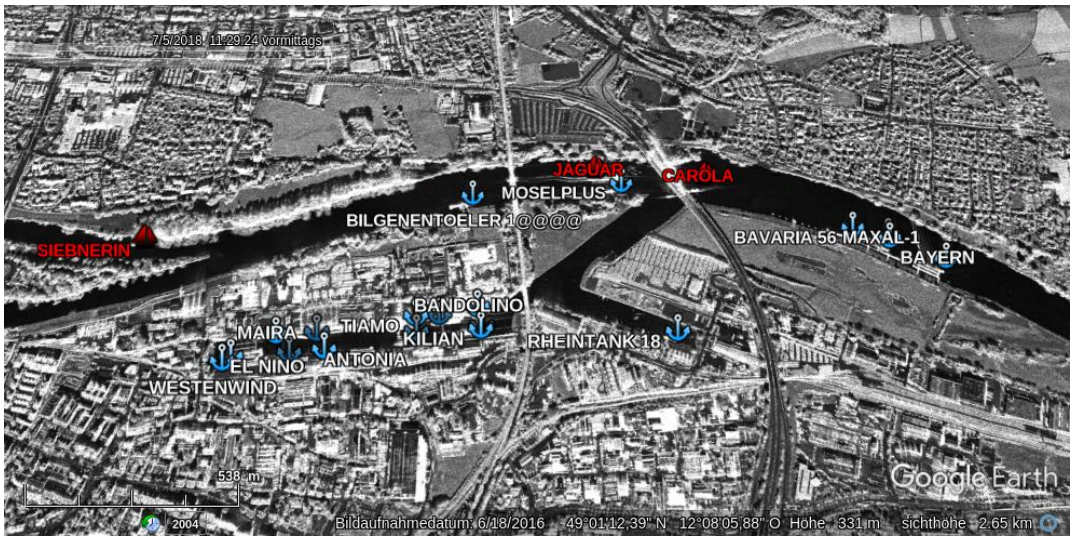


Abbildung 9: Darstellung der Ergebnisse als Überlagerung in Google Earth

2.5 Resultat

Zunächst zeigt sich, dass Algorithmen gemäß den zuvor vorgestellten Anforderungen erzeugt wurden und auch entsprechend ihrer Bestimmung einsetzbar sind. Auch wenn bisher keinerlei

Abbildung 10: Auschnitt aus Abbildung 9

Tests im laufenden Betrieb durchführbar waren, sind die bisherigen Resultate zufriedenstellend. Zur vollständigen Validierung dieses Projekts fehlen lediglich ausführlichere Erprobungen, unter anderem während der Signalaufnahme im Messflug.

3 Analyse von Kalibrierpulsen für das V-SAR System

Bei der Radaraufnahme mittels DBFSAR beziehungsweise V-SAR werden auch Kalibrierpulse mit aufgezeichnet. Um diese Daten auszuwerten verfügt das Institut HR über einen Algorithmus, der die empfangenen Signale als Graphen, wie in Abbildung 11 gezeigt, darstellt.

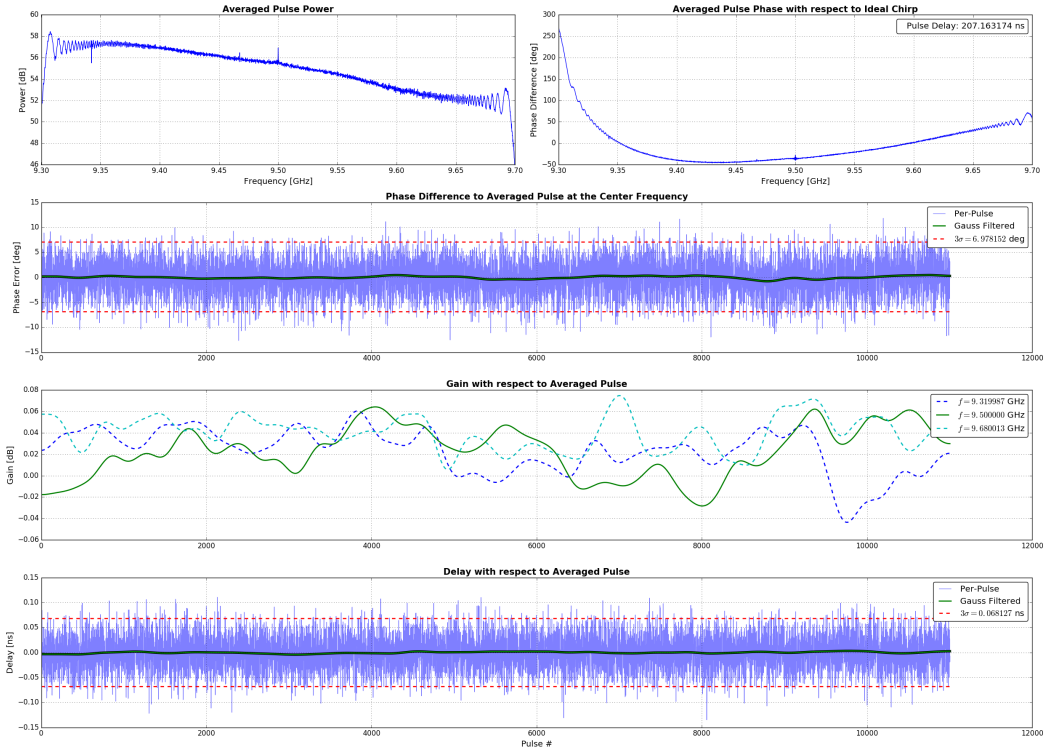


Abbildung 11: Graphische Auswertung der Kalibrierpulse

Diese Diagramme zeigen aber jeweils nur einen Kanal der ganzen Aufnahme. Um aber genauere Informationen zu erhalten sollten die Kanäle mit einem Referenzkanal verglichen werden. Um dieses Ziel zu erreichen wird eine Modifizierung der schon vorhandenen Methode vorgeschlagen. Im Folgenden wird aufgezeigt, wie solch eine Modifikation erzielt werden kann.

3.1 Entwurf des modifizierten Algorithmus

Die bisher vorhandene Methode - ein Konsolenprogramm - benötigt vom Nutzer eine Spezifikation der benötigten Daten, welche über den Missionsnamen und einen Parameter zur Bestimmung des Pfades in diese erzielt wird. Aus diesem Pfad werden alle verfügbaren Kanäle geladen und nacheinander berechnet und gespeichert.

Da hier aber ein Vergleich angestellt werden soll, muss auch noch der Vergleichskanal definiert werden. Hierzu wird der Aufruf um einen zusätzlichen Missionsnamen, und zwei Parameter zur Charakterisierung dieses Pulses erweitert.

Da schon eine Funktion vorliegt, die alle benötigten Werte berechnet, wird diese als Grundstock für die Kalkulation der Vergleichswerte benutzt und gibt diese zurück, anstatt sie abzuspeichern.

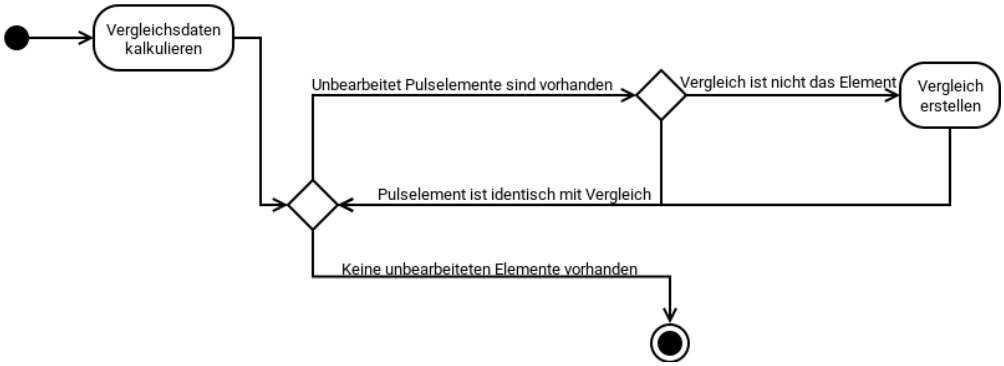


Abbildung 12: Aktivitätsdiagramm zum Vergleich von Kalibrierpulsen

Folglich kann der ganze Ablauf des Algorithmus anhand von Abbildung 12 beschrieben werden: Direkt nach dem Aufruf der Methode werden die Vergleichsdaten berechnet und zwischengespeichert. Für diese Aufgabe wird die schon zuvor erwähnte Modifikation verwendet. Anschließend wird über eine *for*-Schleife geprüft, ob in dem angegebenem Zielverzeichnis noch unbearbeiteten Kanäle des benötigten Typs vorhanden sind. Wenn dies zutrifft, so wird zuerst über die Lokalisierungsparameter festgestellt, ob die anfangs benutzten Daten zum Erstellen des Vergleichs aus diesem Kanal stammen. Wenn dies nicht der Fall ist, so wird eine weitere modifizierte Version des ursprünglichen Algorithmus verwendet, um die eigentlichen Diagramme zu generieren. Ist dieser Vorgang abgeschlossen, so wird wieder nach unbearbeiteten Elementen gesucht.

Die erzeugten Graphen zeigen die durchschnittliche Pulsstärke, durchschnittliche Pulsphase im Verhältnis zum idealen Signal und die Phasen-, die Verstärkungs- und die Zeitverschiebungsdifferenz. Beim Vergleich werden bei den beiden zuerst aufgezählten Diagrammen die zu vergleichenden Werte übereinander gelegt und gleichzeitig angezeigt. Die restlichen Darstellungen zeigen den Absolutteil der Differenz aus den Vergleichswerten. Daraus ergeben sich Graphen, wie sie in Abbildung 13 dargestellt sind.

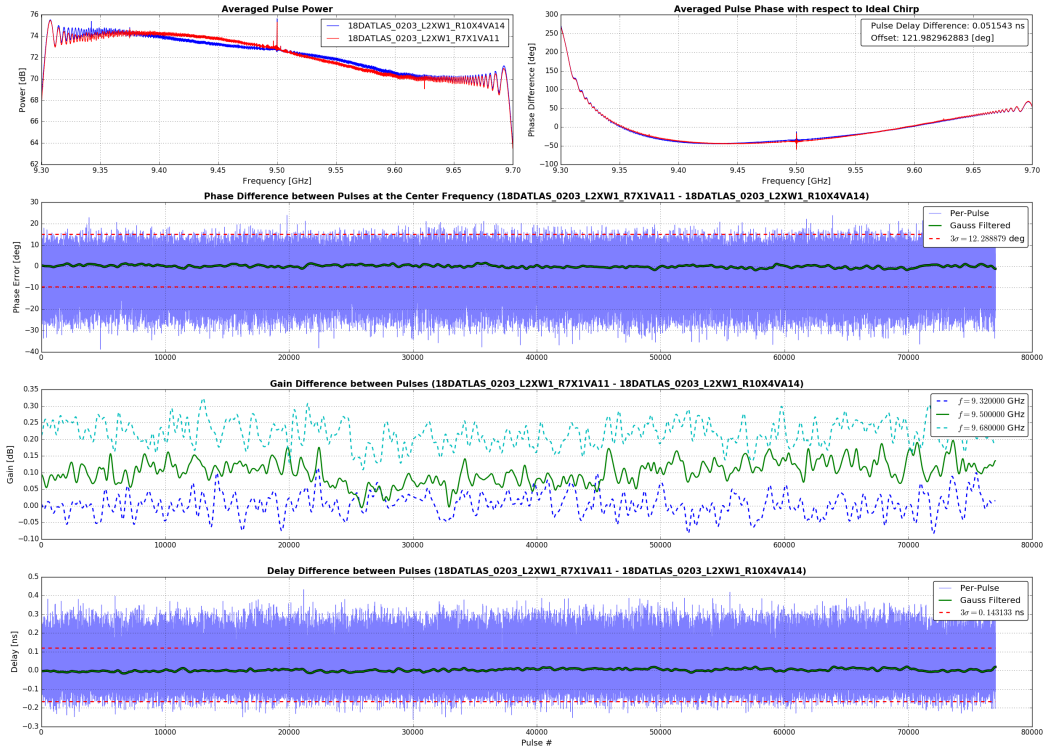


Abbildung 13: Ergebnis des vergleichenden Algorithmus

3.2 Fazit

Mithilfe des zuvor vorgestellten Algorithmus lassen sich Diagramme erstellen, die dem Betrachter genaue Unterschiede zwischen den aufgezeichneten Kanälen zeigen können. Mithilfe dieser können fehlerhafte Kanäle leichter identifiziert und Kalibrierparameter für die kombinierte Mehrfachverarbeitung zuverlässiger abgeschätzt werden. Aus den zuvor generierten Daten

Signallaufzeit (ns)	Calibration	Datatake	
L2XW1_R3X8VA3	-0,076	-0,046	0,020
L2XW1_R4X8HA4	-0,076	-0,029	0,047
L2XW1_R6X5VA10	0,026	0,031	0,005
L2XW1_R8X2VA12	-0,06	-0,027	0,033
L2XW1_R9X3VA13	0,009	0,001	0,008
L2XW1_R10X4VA14	0,015	0,052	0,037
L2XW1_R12X6VA16	0,072	0,079	0,007
Mean	-0,013	0,009	0,022

Tabelle 1: Die Differenz zwischen den Pulsverzögerungen (zum Referenzcoding L2XW1_R7X1VA11) zwischen dem Kalibrierpass und dem Aufnahmepass. Außerdem wird der Durchschnitt der jeweiligen Spalten gezeigt.

können Tabelle 1 und Tabelle 2 erstellt werden, wodurch es möglich wird die Daten zu vergleichen.

Hieraus lässt sich errechnen, dass sich internen Signallaufzeiten im Mittel auf +/- 0.01 nsec bei einer Standardabweichung von 0.059, bzw. 0.04 nsec belaufen. Der Unterschied zwischen Kalibrier- und Messpulsen liegt bei etwa 0.02 nsec und hat eine Standardabweichung von zirka 0.01 nsec. Die Zeit 0.01 nsec entspricht bei der Radarmessung einem Positionsfehler von in etwa 1.5 mm. Da die bestmögliche Auflösung aber 0.1 m ist, so sind die gemessenen Abweichungen vernachlässigbar.

Weiterhin sind die Phasenoffsets für die unterschiedlichen Kanäle sehr verschieden, jedoch ist die Differenz zwischen Messflug und Kalibriermessung vergleichsweise gering. Der Mittelwert der Phasendifferenzen liegt bei zirka 1 Grad mit einer Standardabweichung von 1.13 Grad.

Damit erscheint das DBFSAR/V-SAR System stabil und die Kalibriermessungnen können für die relative Kanalkalibirierung verwendet werden. Die Angleichung der Signallaufzeiten im Radarsystem während der Aufnahme war erfolgreich.

Phasenoffset (deg)	Calibration	Datatake	
L2XW1_R3X8VA3	-73,92	-73,35	0,57
L2XW1_R4X8HA4	-154,61	-158,34	3,73
L2XW1_R6X5VA10	-34,31	-33,99	0,32
L2XW1_R8X2VA12	-15,91	-15,71	0,20
L2XW1_R9X3VA13	-113,97	-112,75	1,22
L2XW1_R10X4VA14	122,58	121,98	0,6
L2XW1_R12X6VA16	-41,65	-40,69	0,96
Mean			1,085

Tabelle 2: Die Differenz zwischen den Offsets (zum Referenzcoding L2XW1_R7X1VA11) zwischen dem Kalibrierpass und dem Aufnahmepass. Außerdem wird der Durchschnitt des Offsets gezeigt.

Literaturverzeichnis

[1] Baumgartner, Stefan V.: *AIS- Messages and Database Structure for Archiving*. Juli 2018

[2] *Institut für Hochfrequenztechnik und Radarsysteme*. https://www.dlr.de/hr/desktopdefault.aspx/tabid-2304/3442_read-39287/. Version: August 2018

[3] *Institut für Hochfrequenztechnik und Radarsysteme - SAR-Technologie*. https://www.dlr.de/hr/desktopdefault.aspx/tabid-2326/. Version: August 2018

[4] *Microwaves and Radar Institute - SAR Signalverarbeitung*. https://www.dlr.de/hr/en/desktopdefault.aspx/tabid-2326/3776_read-5688/. Version: August 2018

[5] Corporation, Oracle: *MySQL :: MySQL 5.5 Reference Manual :: 12.16.1 Aggregate (GROUP BY) Function Descriptions*. https://dev.mysql.com/doc/refman/5.5/en/group-by-functions.html#function_count. Version: September 2018

[6] Corporation, Oracle: *MySQL :: MySQL 5.7 Reference Manual :: 13.7.5.36 SHOW TABLE STATUS Syntax*. https://dev.mysql.com/doc/refman/5.7/en/show-table-status.html. Version: September 2018