Entwicklung einer App zur Steuerung und Datenauswertung einer Drohne zur Luftqualitätsmessung

Studienarbeit

an der Dualen Hochschule Baden-Württemberg Stuttgart

von

Julian Riegger

04.06.2018

Bearbeitungszeitraum	xx Wochen
Matrikelnummer, Kurs	1577610, STG-TINF15-ITA
Ausbildungsfirma	Robert Bosch GmbH, Stuttgart
Betreuer	Thilo Ackermann, Rene Lasse

Erklärung

Ich erkläre hiermit ehrenwörtlich:

1. dass ich meine Studienarbeit mit dem Thema *Entwicklung einer App zur Steuerung und Datenauswertung einer Drohne zur Luftqualitätsmessung* ohne fremde Hilfe angefertigt habe;
2. dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe;
3. dass ich meine Studienarbeit bei keiner anderen Prüfung vorgelegt habe;
4. dass die eingereichte elektronische Fassung exakt mit der eingereichten schriftlichen Fassung übereinstimmt.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Stuttgart, 04.06.2018

_____

Julian Riegger

Abstract

Logfiles beinhalten eine große Menge an Daten, deren Analyse bei der Suche nach Fehlern und der Überwachung einer IT Infrastruktur eine große Hilfe sind. Dabei stellen sich mehrere Herausforderungen. Die Erste ist, dass Logfiles textbasiert sind. Der Nachteil hierbei ist, dass im Vergleich zu einer Datenbank oder einer XML Datei textbasierte Dateien keine klar auslesbare oder durchsuchbare Struktur besitzen. Die Zweite ist, dass Systeme so viele Informationen wie möglich loggen und dadurch die nützlichen bzw. wichtigen Informationen erst herausgefiltert werden müssen.

Für die Analyse von textbasierten Daten eignet sich sehr gut das MapReduce Modell. Außerdem lässt sich das Modell sehr einfach skalieren und auf mehrere Programmläufe verteilen (Master-Worker). Das Apache Hadoop Projekt stellt sowohl für MapReduce, als auch für die Verwaltung von mehreren Programmläufen, ein Basisframework bereit, mit welchem die Entwicklung eines Analyseprogramms durchgeführt werden soll.

Ziel dieser Arbeit ist die Entwicklung einer prototypischen Anwendung zur formatunabhängigen Analyse von Logfiles unter Zuhilfenahme von Apache Hadoop MapReduce. Die Anwendung soll die bisher vorhandenen Monitoring Systeme innerhalb der Infrastruktur ergänzen, wodurch Informationen über den Zustand des Systems schneller erhoben werden können. Des Weiteren sollen aufkommende Fehler besser erkannt werden, um die Reaktionszeit auf diese zu optimieren.

Inhaltsverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
AQI	Air Quality Index
CAQI	Common Air Quality Index
CO	Kohlenstoffmonoxid
CO2	Kohlenstoffdioxid
EPA	Environmental Protection Agency
IDE	Integrated Development Environment «««< HEAD
DJI	DJI
IP	Internetprotokoll
KB	Kilobyte
LTS	Long Term Support
MB	Megabyte ======= »»»> 7ddb84756ee9ca9703000a9627a78569d24b8fbb
GUI	Graphical User Interface
GPS	Global Positioning System
MVC	Model View Controller
NO2	Stickstoffdioxid
NOx	Stickoxide
O3	Ozon
SDK	Software Development Kit
SO2	Schwefeldioxid
OS	Betriebssystem
WLAN	Wireless Local Area Network
XDK	Cross Domain Development Kit
IOT	Internet of Things
UART	Universal Asynchronous Receiver-Transmitter
ADC	Analog-to-Digital Converter
MCU	Microcontroller Unit
V	Volt
GPIO	General Purpose Input/Output
CO	Carbon Monoxide
CO2	Carbon Dioxide

O3	Ozone
PM	Particulate Matter
UDP	User Datagram Protocol
CAD	Computer-Aided Design
PCB	Printed Circuit Board
Pa	Pascal
TCP	Transmission Control Protocol
SSID	Service Set Identifier

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1 Einleitung

In den lezten Jahren bekam das Thema der Luftqualität immer mehr Aufmerksamkeit und gewinnt immer mehr an Bedeutung in der Tagespolitik sowie in der Industrie. Hier ist vor allem die Automobilindustrie in den Fokus gerückt, da die Verbrennungsmotoren in einer sehr emotional geführten Debatte für einen Großteil der schlechten Luft in Großstädten verantwortlich gemacht werden. Nun trägt nicht nur der Verkehr sondern auch die Industrie mit verschiedenen Fabriken, wie auch andere Faktoren, wie zum Beispiel das heizen mit Holz im Winter, zur Verschlechterung der Luftqualität bei. Es wurden in den letzten Jahren immer mehr Messstationen in großen und kleineren Städten platziert, um die Luftqualität zu überwachen.
Zum Thema Luftqualität stellen sich folgende Fragen, welche in der folgenden Arbeit teilweise beantwortet werden sollen.

- Was ist Luftqualität?
- Kann man die Luftqualität messen?
- Was sind Faktoren für die Luftqualität?
- Was sind für den Menschen gefährliche Faktoren in der Luft?

1.1 Problemstellung

Von den im Kapitel Einleitung genannten Messtationen ist in der Region Stuttgart die Messstation am Neckartor die bekannteste. Diese misst die Luftqualität aber nur an einer Stelle. Hierbei kann man diskutieren, ob dieser Wert überhaupt aussagekräftig ist oder nicht. Es könnte sein, dass die Wahl für den Ort der Messtation missglückt ist und die gemessenen Werte deshalb nicht aussagekräftig sind.
Ein weiterer Aspekt, welcher zu berücksichtigen ist, sind die Auswirkungen des Wetters auf die Luftqualität.

1.2 Aufgabenstellung

Um die genannten Probleme zu umgehen, soll eine Air-Quality-Drone erstellt werden. Hierbei soll eine bereits existierende Drone mit Sensoren ausgestattet werden, welche klassische Werte zur Beurteilung der Luftqualität und zur Beurteilung der Umgebung, wie zum Beispiel die Luftfeuchtigkeit und Temperatur erfassen können. Eine Drone ist agil und kann an verschiedenen Orten und in verschiedenen Luftschichten Messungen durchführen. In dieser Arbeit soll ein Prototyp für eine Drohne zur Messung der Luftqualität erstellt werden. Zu der Drone soll eine App erstellt werden, über die die Drohne bedient werden kann.

2 Theoretische Grundlagen

Für die Erstellung der App, sowie für die Auswahl der Sensoren und Erstellung des Messaufbaus ist verschiedenes Wissen notwendig. Diese theoretischen Grundlagen werden im folgenden erläutert.

2.1 Luftqualität

Die Luftqualität gibt den Gütegrad der Luft an. Dabei handelt es sich um die Bewertung und den Einfluss von Verunreinigung durch Fahrzeugabgase oder Kraftwerke. Bestandteile der ausgestoßenen Luft, wie beispielsweise Stickoxide (NOx) oder Feinstaub, sind für die Luftverschmutzung verantwortlich und werden in Kapitel? näher erläutert. Um die Luftqualität möglichst hoch zu halten und den Menschen wenig zu schaden werden Richtlinien mit Grenzwerten eingeführt sowie fortlaufend verfeinert. Diese Richtlinien geben die Höhe sowie deren Eintrittshäufigkeit pro Zeiteinheit an. Das überschreiten der Grenzwerte hat Sanktionen zur Folge.

2.1.1 Indices

Zur allgemeingültigen Bewertung der Luft gibt es diverse Luftqualitätsinidices, die basierend auf den gemessenen Werten von Feinstaub (PM2,5 und PM10), bodennahem Ozone (O3), Stickstoffdioxid (NO2) und Schwefeldioxid (SO2), den Gütegrad der lokalen Luft bestimmen. Zur Messung dieser Werte werden derzeit meist stationäre Sensoren, oftmals an viel befahrenen Straßen, eingesetzt.
Es gibt Länder- und Regionenspezifische Indices. Im folgenden werden die Indices für Europa und für die Vereinigten Staaten erläutert.

Air Quality Index (AQI)

Der sogenannte Air Quality Index (AQI) bildet für die Vereinigten Staaten eine Skala von 0 bis 500 ab. Mit steigendem Wert wird die Luftqualität, bezogen auf den Tag, schlechter

und die Risiken für den Menschen schwerwiegender. Die folgende Tabelle stellt die Zusammenhänge von AQI-Wert und den Folgen sowie deren Bedeutung dar. Laut Environmental Protection Agency (EPA) ist ein Wert von maximal 100 als gesundheitlicher Standard angesetzt.

Air Quality Index Levels of Health Concern	Numerical Value	Meaning
Good	0 to 50	Air quality is considered satisfactory, and air pollution poses little or no risk.
Moderate	51 to 100	Air quality is acceptable; however, for some pollutants there may be a moderate health concern for a very small number of people who are unusually sensitive to air pollution.
Unhealthy for Sensitive Groups	101 to 150	Members of sensitive groups may experience health effects. The general public is not likely to be affected.
Unhealthy	151 to 200	Everyone may begin to experience health effects; members of sensitive groups may experience more serious health effects.
Very Unhealthy	201 to 300	Health alert: everyone may experience more serious health effects.
Hazardous	301 to 500	Health warnings of emergency conditions. The entire population is more likely to be affected.

Tabelle 2.1: AQI Übersicht

Common Air Quality Index (CAQI)

Europäische Länder haben eine eigene, angepasste Skala zur Bewertung der Luftqualität: den Common Air Quality Index (CAQI). Man unterscheidet hierbei kurzfristige (stündlich oder täglich) von langfristigen (jährlich) Luftqualitätsindices. Die stündlich und täglich aktualisierte Luftverschmutzung wird als relatives Maß aus den für Europa bedeutendsten Schadstoffen. Dazu gehören der Feinstaub (PM2,5 und PM10), NO2 und O3. Bei entsprechend verfügbaren Daten können ebenfalls Carbon Monoxide (CO) sowie SO2 einbezogen werden. Zur Berechnung der Index-Klasse unterscheidet man zwischen dem Verkehrs- sowie dem Hintergrundindex. Ersteres soll die Verkehrsbelastung anhand Messsensoren in direkter Nähe von vielbefahrenen Straßen und letzteres die allgemeine Belastung einer Stadt darstellen.
Über den für ein ganzes Jahr hinweg berechneten Index lässt sich der Abstand zu den EU-Grenzwerten darstellen. Der Schwellwert ist dabei 1. Ein höherer Wert des Luftqualitätsindex zeugt von einer Überschreitung eines oder mehrerer Schadstoffwerte. Die Vergleichswerte entsprechen einer Empfehlung durch die Welt Gesundheitsorganisation und dienen dem Schutz der Gesundheit.
Die aktuellen Luftqualitätswerte der Messstationen Europas werden von der EUA und der Europäischen Kommission online unter

2.1.2 Schadstoffe

Im Folgenden soll auf messbare und für die Luftqualität bzw. die menschliche Gesundheit entscheidende Schadstoffe eingegangen werden. Neben der Erläuterung der luftverunreinigenden Teilchen wird auch auf deren Konsequenzen für den Menschen eingegangen.

Feinstaub

Ein für die Gesundheit des Menschen entscheidender Schadstoff ist der Feinstaub oder auch Schwebstaub genannt. Diese kleinen, für das menschliche Auge nicht sichtbaren Teilchen, die nur langsam zu Boden sinken, sind zum größten Teil menschlicher Herkunft. Dazu gehören die Emissionen, die durch Kraftfahrzeuge, Öfen und Heizungen in Innenräumen sowie durch die Metallerzeugung, auftreten. Dabei wird der durch Kraftfahrzeuge entstehende Feinstaub nicht nur aus dem Motor emittiert, sondern auch Bremsen- und Reifenabrieb sowie die Aufwirbeln des Straßenstaubes verunreinigen die Luft. Die eben genannten Faktoren tragen zum primären Feinstaub bei. Der sekundäre Feinstaub hat seine Herkunft in der Landwirtschaft. Hierbei entstehen in der Tierhaltung gasförmige Schadstoffe durch die Ammoniakemissionen.
Dabei unterteilt man die Partikel nach ihrer Größe. Alle Teilchen mit einem aerodynamischen Durchmesser kleiner 10 Mikrometer werden als PM10, wobei man diejenigen mit einem Durchmesser von kleiner 2,5 Mikrometer als PM2,5 bezeichnet. Eine weitere Aufschlüsselung innerhalb der eben genannten Grenzen ergibt die Begrifflichkeiten Grobfraktion, für alle Partikel zwischen 2,5 und 10 Mikrometer, sowie die Feinfraktion, für Partikel mit einem Durchmesser kleiner 2,5 Mikrometer. Die allerkleinsten Partikel, aerodynamischer Durchmesser kleiner als 0,1 Mikrometer nennt man ultrafeine Partikel. PM10 wandern in die Nasenhöhlen, PM2,5 gelangen über die Atemwege in die Bronchien sowie Lungenbläschen und setzen sich dort ab. Die ultrafeinen Partikel können bis in die Blutgefäße eindringen können. Als Folgen können abhängig von der Partikelgröße Atembeschwerden entstehen sowie die Gefahr eines Herzinfarkts oder einer Lungenerkrankung steigen.

NOx

Stickstoffoxide sind die Verbindung aus unterschiedlich vielen Stickstoff- und Sauerstoffatomen. Die wichtigsten Vertreter dieser Gruppe sind das Stickstoffmonoxid und das Stickstoffdioxid. Diese Oxide entstehen bei unerwünschten Nebenreaktionen während der Verbrennung von Benzin, Öl, Gas oder Kohle. Sie stellt einen sehr reaktionsfreudigen Stoff dar, wodurch es nicht nur zur Ozonbildung, sondern auch zur Feinstaubbelastung beiträgt. Stickstoffdioxide können vor allem bei Asthmatikern zur Bronchienverengung führen.

SO2

Das farblose aber stark riechende Schwefeldioxid wird vor allem bei der Verbrennung von Kohle oder Öl erzeugt und liegt im Normalfall als Gas vor.

Für den Menschen hat dieser Schadstoff Schleimhaut- und Augenreizungen sowie Atemwegsprobleme zur Folge.
Heutzutage ist die Belastung durch SO2 jedoch nicht mehr kritisch und die Gesundheitsrisiken akut nicht vorhanden.
Zudem entstehen aus Schwefeldioxid Sulfatpartikel in der Atmosphäre, die die PM10 Belastung verstärken.

O3

Bodennahes O3 gilt als sekundärer Schadstoff, da es erst durch photochemische Prozesse entsteht und nicht direkt emittiert wird. Es ergibt sich vor allem aus NOx sowie flüchtigen organischen Verbindungen. Diese beiden sogenannten Vorläuferstoffe werden überwiegend vom Menschen erzeugt. Während NOx von Kraftfahrzeugen emittiert werden, entstehen flüchtige organische Stoffe bei der Verwendung von Lösemitteln, wie zum Beispiel in Farben, Klebstoffen, Reinigungsmitteln oder durch die Verbrennung von Kraftstoff.
Zu den gesundheitlichen Folgen gehören eine geringere Lungenfunktion sowie Atemwegsbeschwerden. Diese Wirkungen treten vor allem bei körperlicher Belastung sowie bei besonderes anfälligen oder vorgeschädigten Personen auf.

CO

Das gasförmige, farb- und geruchslose CO wird bei der unvollständigen Verbrennung von Kraftstoffen freigesetzt. Der Grund hierfür ist Sauerstoffmangel, der bei extrem niedriger Dosierung zu einer CO Konzentration fährt und in einem solchen Fall als Atemgift wirkt. Das liegt an der Beeinträchtigung der Sauerstoffaufnahme und zieht negative Konsequenzen für das zentrale Nervensystem mit sich.
Des Weiteren ist CO ein Bestandteil bei der Bildung von bodennahem O3.

Carbon Dioxide

CO2 ist ein farb- und geruchsloses Gas. Es ist als Treibhausgas für das auf der Erde entstehende Klima verantwortlich, indem es einen Teil der Wärme, die ins Weltall ausgestrahlt wird zurück auf die Erde emittiert. Neben den großen Vorkommnissen im Weltall wird es ebenfalls bei der Zellatmung von Menschen und vielen Tieren ausgeschieden. Ein weiterer Entstehungsort ist der Verbrennungsvorgang von Öl, Holz oder Kohle. Ein Problem von CO2, ist dass sich dieser Schadstoff nicht selbstständig wieder abbauen kann. Die einzige Möglichkeit zur Reduzierung des CO2-Gehaltes ist die Photosynthese oder die physikalische Speicherung in Gewässern. Dabei entsteht Glucose und Sauerstoff.

Die folgende Tabelle zeigt die Auswirkungen von diversen CO2-Konzentration auf den Menschen.

2.2 Hardware

Um die Luftqualität in verschiedenen Luftschichten messen zu können ist diverse Hardware notwendig.
Für die Messung werden unterschiedliche Sensoren benötigt, die die wichtigsten Aspekte der Luftqualität, wie zum Beispiel den Feinstaub, messen.
Um die Daten zu verarbeiten und auszulesen ist ein µ-Controller notwendig. In dieser Arbeit wurde ein XDK der Firma Bosch verwendet.
Um agil messen zu können wird eine Drone der Marke DJI eingesetzt. Die Modellbezeichnung lautet Phantom 3 Standard.

2.2.1 DJI Phantom 3 Standard

Der Aufbau der Drone ist in folgender Abbildung zu sehen. Die Drone hat vier Propeller, welche die Antriebskraft leisten. Um Bilder und Videos aufzunehmen ist auf der Unterseite der Drone ein Gimbal mit einer Kamera befestigt. Für eine sichere Landung und einen guten Stand hat die Drone zwei Beine.

Abbildung 2.1: DJI Phantom 3 Standard

Die Drone lässt sich auf verschiedenen Wegen steuern. Es gibt die Möglichkeit der klassischen manuellen Steuerung über die Fernbedienung oder man lässt die Drone autonom fliegen.
Allgemein ist eine App notwendig, um die Drohne zu steuern und alle Features, welche mit der Drohne kommen, zu nutzen. Hierfür stellt die Firma DJI eine eigene App, die DJI GO App zur Verfügung. Es wird aber auch ein Software Development Kit (SDK) bereitgestellt, mit dem sich eigene Apps entwickeln lassen. Durch die durch das Software Development Kit bereitgestellten Funktionen lassen sich die Features der DJI GO App replizieren und erweitern.
Das manuelle fliegen lässt sich einfach über die mitgelieferte Fernbedienung realisieren. Die zwei Steuerknüppel dienen hierbei zur Steuerung. Die Funktion der einzelnen Steuerknüppel lässt sich über die DJI eigene App konfigurieren. Hier kann der Nutzer seine Vorlieben einstellen.
Um die App bequem während des Fluges bedienen zu können ist an der Fernbedienung eine Halterung montiert in die man das Endgerät, auf der die App ausgeführt wird, befestigen kann.
Der Aufbau der Fernbedienung ist in folgender Abbildung zu sehen.

Abbildung 2.2: DJI Phantom 3 Standard - Fernbedienung

Die zweite Möglichkeit ist, die Drohne autonom fliegen zu lassen. Hierbei werden sogenannte Waypoints einer Mission hinzugefügt, welche gestartet wird. Die Waypoints beinhalten die Koordinaten mit Längengrad und Breitengrad, sowie der Flughöhe und weiteren Informationen, wie der Fluggeschwindigkeit oder dem Radius mit welchem der Punkt umflogen werden soll. Um den autonomen Flug zu starten wird die Mission gestartet. Während des Fluges erkennt die Drohne über die Kamera Hindernisse und vermeidet eine Kollision mit diesen.
Hierbei spielt das Global Positioning System (GPS) der Drohne eine wichtige Rolle. Das intelligente System merkt sich den Startpunkt der Drohne. Je nach Einstellung kehrt die Drohne automatisch zum Startpunkt zurück, sollte der Akkustand eine bestimmte

Grenze unterschreiten, die Drohne außer Reichweite der Fernbedienung sein oder die return-to-home-Funktion ausgeführt wird.

Die folgende Tabelle gibt eine Übersicht über die wichtigsten, für diese Arbeit relevanten technischen Daten.

Gewicht	1216g
Diagonale Größe	350mm
max. Steiggeschwindigkeit	$5\frac{m}{s}$
max. Sinkgeschwindigkeit	$3\frac{m}{s}$
max. Fluggeschwindigkeit	$16\frac{m}{s}$
max. Flughöhe über NN	6000m
max. Flugzeit	ca. 25 min
Betriebstemperatur	0° bis 40°C
Positionsbestimmung	GPS
max. Sendereichweite	FCC: 1000m CE: 500m Flughöhe: 120m

Tabelle 2.2: DJI Phantom 3 Standard - technische Daten

2.2.2 iPad 3

Um die in dieser Arbeit zu erstellende App auszuführen und das Produkt zu testen ist ein mobiles Endgerät notwendig.
Bei dem mobilen Endgerät handelt es sich um ein iPad der 3. Generation von der Firma Apple.
Das iPad hat die Abmessungen:

- Höhe: 241,2 mm
- Breite: 185,7 mm
- Tiefe: 9,4 mm

- Gewicht: 652 g

Die Bedienfläche ist ein 9,7"großer Multi-Touch Display. Das iPad ist **WLAN!** fähig und kann eine Bluetooth Verbindung aufbauen.

Abbildung 2.3: iPad 3

2.3 iOS-Appentwicklung

Bei der Appentwicklung für iOS Geräte bietet sich die Apple eigene Programmiersprache Swift an, welche für die in dieser Arbeit erstellten App auch verwendet wurde.
Die Entscheidung für ein für das Projekt sinnvolles Design-Pattern fiel auf das Model View Controller (MVC) Pattern.
Für die Ansteuerung der DJI-Drone ist das DJI-SDK notwendig, sowie für die Einbindung externer Bibliotheken ist Wissen über Cocoa Pods notwendig.
Im folgendem werden die genannten Grundlagen in einzelnen Unterkapiteln kurz beschrieben.

2.3.1 Model View Controller (MVC)

Das MVC-Pattern besteht, wie der Name sagt, aus drei verschiedenen Teilen. Dem Model, dem Controller und der View. Das Model dient ausschließlich zur Speicherung von Daten. Zum Beispiel werden aktuelle Daten der Anwendung, wie zum Beispiel eine Flugroute in einem Model abgespeichert. Die View ist für die Darstellung der Inhalte und Daten

zuständig. Ebenso ist die View dafür zuständig die Eingaben eines Nutzers an den entsprechenden Controller weiterzuleiten. Die View beinhaltet auch die gesamte Graphical User Interface (GUI). Der Controller beinhaltet die Anwendungslogik und ist für die Steuerung der Anwendung verantwortlich.

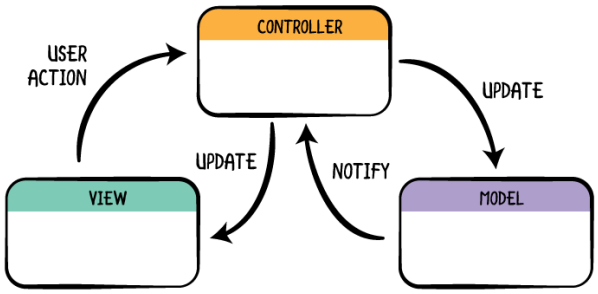

Abbildung 2.4: MVC Design-Pattern Diagramm **mvc1**

Das MVC Pattern kann verschieden streng implementiert werden. In dieser Arbeit wird das Pattern in einer leichten Form angewendet. Es wird sich nicht exakt an die Spezifikation aus der Literatur gehalten.

Eine mit Xcode und Swift gschriebenen App hat als Kern eine UIApplication-Klasse. Diese Klasse verarbeitet die Interaktion zwischen System und Objekten der App. Sie verwaltet die geöffneten Views und leitet Ereignisse durch Nutzereingaben oder vom System kommend an die entsprechenden Controller weiter.
Die *ViewController* Objekte stellen in einer iOS-Anwendung die Controller des MVC Patterns dar. Eine *ViewController*-Klasse verarbeitet Nutzereingaben und aktualisiert die View.
Die View ist in der iOS-Programmierung durch sogenannte Storyboards realisiert.

2.3.2 Cocoa

Application Programming Interface (API)

Cocoa ist eine Application Programming Interface (API) zur Programmierung unter den Betriebssystem (OS) MacOS. Für die Apple OS von mobilen Endgeräten, welche über Touch-Displays verfügen, wurde die Cocoa API zur CocoaTouch API erweitert. CocoaTouch beinhaltet Funktionen für die Nutzereingabe über Eingaben durch Gesten. Somit wird für die Entwicklung von iOS Apps, wie in dieser Arbeit, die CocoaTouch API verwendet.
Die Entwicklung für Apps mit der Cocoa API erfolgt mit den Apple eigenen Developer

Tools, Xcode, welches im nächsten Kapitel beschrieben wird, und dem Interface Builder. Die hauptsächlich für die API gedachten Programmiersprachen sind Objective-C und die Apple eigene Programmiersprache Swift. Die Programmierung in C und C++ ist generell auch möglich.
Der Aufbau von Cocoa ist im Allgemeinen einfach gehalten. Cocoa besteht aus drei verschiedenen Frameworks.

- *Foundation*: beinhaltet alle relevanten Basisklassen, wie Strings, Arrays, Iterators, et cetera.
- *AppKit*: stellt Klassen zur Entwicklung von Graphical User Interface (GUI) zur Verfügung. Zum Beispiel Buttons, Labels, Menüs, usw..
- *Core Data*: dient zur Erstellung von Objektgraphen.

Klassen des Cocoa-Frameworks sind im Quellcode durch die Buchstaben *NS* im Objektnamen zu erkennen.

Pods

CocoaPods ist ein application level dependency manager für Objective-C, Swift und andere Programmiersprachen, welche in XCode laufen. Pods stellt ein Standardformat zum managen von externen Bibliotheken bereit.
Die Projektabhängigkeiten werden mittels Podfile-Dateien in einem Projekt beschrieben. Im Folgendem ist ein Beispiel zu sehen.

```
1  # platform: iOS, '9.0'
2  use_frameworks!
3  project 'AirQualityDrone.xcodeproj'
4  target 'AirQualityDrone' do
5  pod 'DJI-SDK-iOS' '~> 4.4'
6  pod 'DJI-UILibrary-iOS', '~> 4.4'
7  pod 'CocoaAsyncSocket'
8  pod 'DTMHeatmap'
9  end
```

Listing 2.1: Podfile Beispiel

Um die externen Bibliotheken zu installieren, wird der Befehl *pod install* aufgerufen. Dadurch werden die Quellen der Bibliotheken geladen und das Projekt in Xcode eingerichtet, sodass die Bibliotheken separat gebaut werden. In das Projekt werden die Dateien über eine statische Bibliothek (*libPods.a*).

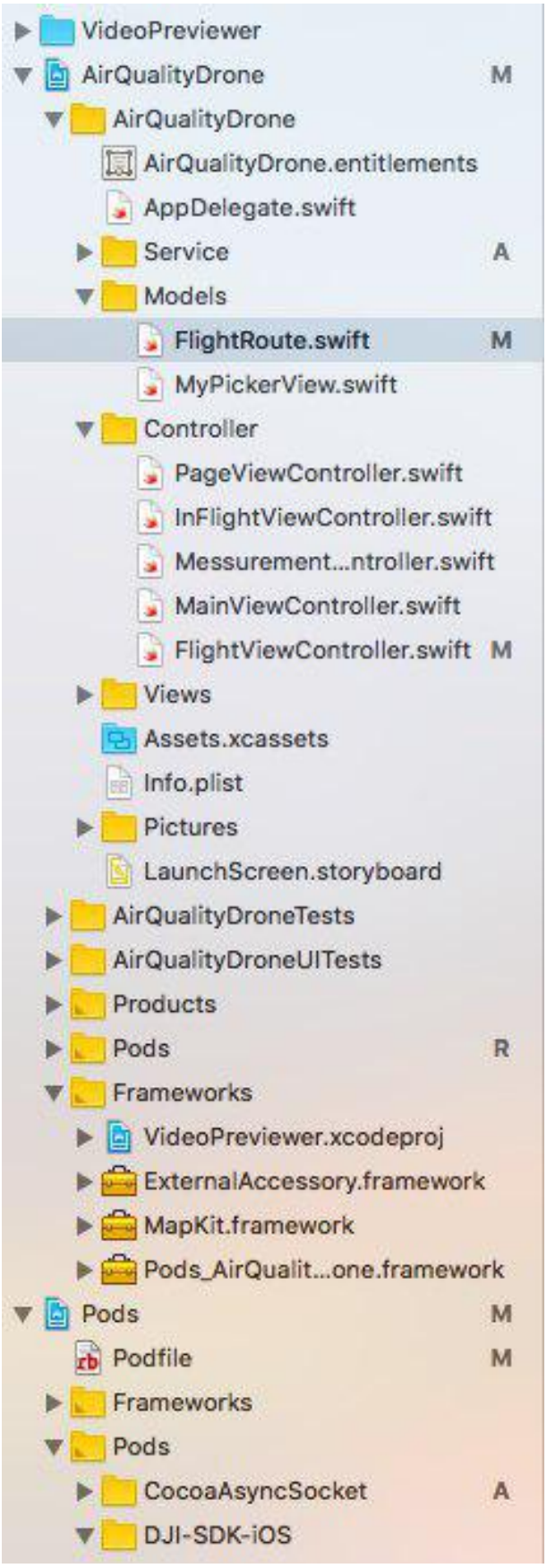

Abbildung 2.5: Externe Bibliotheken in Xcode

2.3.3 Xcode

Xcode ist eine Integrated Development Environment von Apple für das OS macOS. Xcode ist für die Entwicklung von Programmen und Apps für macOS, iOS, tvOS und watchOS gedacht. Die IDE ist BEstandteil der Xcode Tools.
Die Xcode Tools beinhalten:

- Xcode IDE
- Interface Builder
- Instruments
- Xcode Core
- Dashcode
- Quanz Composer
- iPhone Simulator

Die iOS App, welche Bestandteil dieser Arbeit ist, wird ausschließlich mit Xcode entwickelt. Die folgende Abbildung zeigt eine Übersicht der Xcode-Oberfläche.

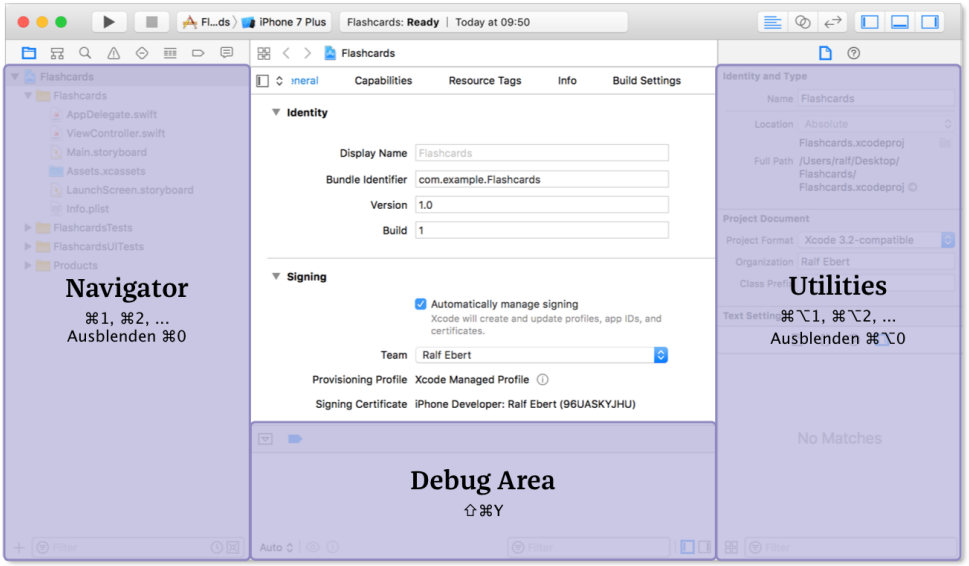

Abbildung 2.6: Xcode-Oberfläche

Zentral ist in Xcode der Editor zu finden. Auf der linken Seite befindet sich der Navigator. Unterhalb des Editors befindet sich die Debug Area und auf der rechten Seite die Utilities. Navigator, Debug Area und Utilities lassen sich über die Buttons in der oberen rechten

Ecke ausblenden und ermöglichen es so das Editor Fenster größer zu machen.
Die Projektstruktur ist in Abbildung 2.5 zu sehen.

2.3.4 SWIFT

Swift ist eine Programmiersprache des Technik Konzerns Apple. Sie wurde für die Entwicklung von Apps für iOS, Mac, Apple TV und Apple Watch kreiert. Swift ist kostenlos und Open Source. Die Sprache steht unter der Apache 2.0 Open Source Lizenz. Somit kann eine große Community direkt zum Swift Quellcode beitragen.
Swift vereint unterschiedliche Konzepte verschiedener Programmiersprachen, wie zum Beispiel Objective-C und Python. Dies führt dazu, dass Sie sich durch Paradigmen wie objektorientiert und imperativ beschreiben lässt. Swift greift Mechanismen, wie Klassen, Vererbung, Closures, Typinferenz, generische Typen, etc., auf, welche von anderen Programmiersprachen bereits bekannt sind.
Für die Appentwicklung wurde die Programmiersprache Swift unter der Version 4.1 verwendet.

2.3.5 DJI-Software Development Kit (SDK)

DJI bietet neben dem Verkauf von Drohnen auch noch andere Leistungen an. Hierzu gehören diverse SDK, die die Entwicklung von Apps und Programmen für die Drohnen von DJI erleichtern. SDK die angeboten werden sind:

- Mobile SDK
- Onboard SDK
- Guidance SDK
- Payload SDK

In dieser Arbeit ist nur das Mobile SDK relevant.
Das DJI Mobile SDK unterstützt die Plattformen iOS 9.0 oder höher, sowie Android 5.0.0 oder höher. Da das mobile Endgerät mit einem iOS OS läuft, wird in dieser Arbeit die Ausführung des DJI Mobile SDK für iOS verwendet. Hierbei ist die Programmierung in den Sprachen Swift und Objective-C möglich. Wie im Kapitel 2.3.4 erwähnt wird in dieser Arbeit die Programmiersprache Swift 4.1 verwendet.
Das SDK bietet verschiedene Kernfunktionalitäten an. Zu diesen wichtigen und nützlichen Features gehört die Obstacle avoidance, High and low level flight control, Aircraft state through telemetry and sensor data, Live video feed, Pre defined missions, wie Waypoint,

HotPoint oder FollowMe und State information und control of Battery und Remote Controller.
Für die Entwicklung von GUI stellt DJI eine UXLibrary zur Verfügung, welche graphische Elemente für die wichtigsten Funktionen des mobile SDK bereitstellt.

2.4 Bosch Cross Domain Development Kit (XDK)

2.4.1 Allgemeines

Abbildung 2.7: Bosch XDK

Das Bosch Cross Domain Development Kit (XDK) ist ein kabelloses Sensor-Kit, welches Rapid Prototyping von Produkten im Rahmen des Internet of Things (IOT) ermöglicht. Es ist ausgelegt für die Entwicklung von Prototypen vor der tatsächlichen Serienentwicklung. Hierfür wird die eigens dafür entwickelte Entwicklungsumgebung, die sogenannte XDK-Workbench verwendet. Zur Implementierung der eigenen Software kann zwischen den beiden Programmiersprachen C sowie Eclipse Mita gewählt werden, wobei letztere im Kontext des XDK oft auch als XDK Live bezeichnet wird. Im Rahmen der Studienarbeit haben wir uns für die Programmiersprache C entschieden, da hierfür die Dokumentation zur Zeit noch ausführlicher ausfällt als bei XDK Live.

2.4.2 XDK Workbench

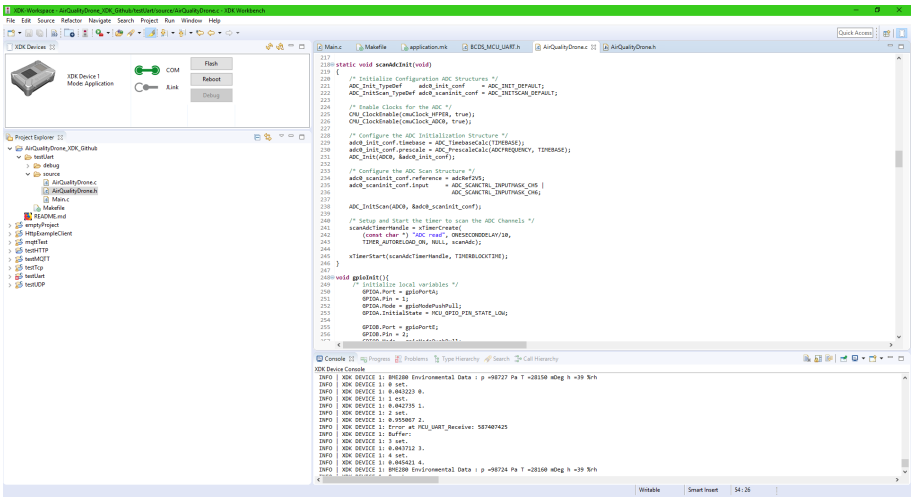

Abbildung 2.8: Bosch XDK Workbench

Die XDK Workbench ist eine auf Eclipse basierende Entwicklungs-Umgebung, um Programmcode für das XDK zu entwickeln, zu debuggen und den Code auf das Gerät zu flashen. Um ein lauffähiges Programm aus dem Code zu generieren muss das Projekt nach jeder Änderung zunächst gebuildet werden, bevor es anschließend über den Flash-Button auf das Gerät geladen werden kann. Um die beim Built erzeugten Binärdateien erfolgreich auf das Gerät zu übertragen, muss sich dieses zunächst im Bootloader-Modus befinden. Diesen kann man herstellen, indem das Gerät ausgeschaltet wird und es anschließend unter Drücken von Button 1 des Gerätes wieder eingeschaltet wird.
Einen wirklichen Debug-Mechanismus bietet die Workbench alleine bis dato nicht. Um dennoch zu debuggen muss ein externes Gerät angeschlossen werden wie beispielsweise die J-Link Debug Probe.
Falls nicht auf dieses externe Debug-Gerät zurückgegriffen werden kann bleibt nichts anderes übrig als über Kommandozeilen-Ausgaben das Programm zu debuggen. Dies ist im Falle des XDK besonders mühsam, da der Prozess des Buildens, Bootens und Flashens relativ langwierig sein kann. Für diesen Prozess können mitunter mehrere Mintuen vergehen, falls beispielsweise noch ein Clean erforderlich ist.

2.4.3 Sensoren

Das XDK ist mit Sensoren zur Messung folgender Größen ausgestattet:

- Beschleunigung (BMA280)
- Gyroskop (BMG160)

- Magnetische Feldstärke (BMM150)
- Licht (MAX44009)
- Sound (AKU340)
- Temperatur (BME280)
- Relative Luftfeuchtigkeit (BME280)
- Luftdruck (BME280)

Im Rahmen dieser Arbeit werden die letzten drei genannten Sensoren verwendet, um Informationen über die Umwelt zu erlangen. Tabelle 2.3 zeigt die Messbereiche und Toleranzen des Sensors BME280.

Messgröße	Messbereich	Einheit	Absolute Genauigkeit	Relative Genauigkeit
Temperatur	0 - 65	°C	± 1	-
Luftfeuchtigkeit	0 - 100	% (relativ)	± 3	-
Luftdruck	300 - 1100	hPa	± 1	$\pm 0{,}12$

Tabelle 2.3: Übersicht Umgebungssensoren XDK

2.4.4 Extension Board

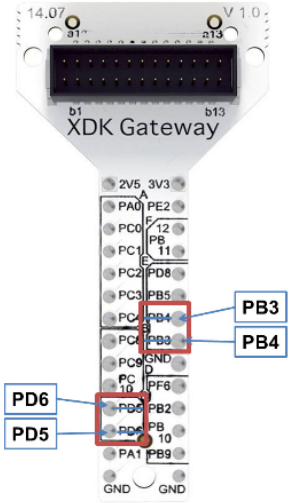

Abbildung 2.9: Bosch XDK Extension Board

Zusätzlich zu den eingebauten Sensoren bietet das XDK die Möglichkeit das zugehörige Extension Board über eine serielle Schnittstelle anzuschließen. Das Extension Board bietet so die Möglichkeit externe Komponenten mit dem XDK zu verbinden, wie beispielsweise weitere Sensoren. Das Board ist bedingt durch die interne Microcontroller Unit (MCU) des XDK auf Spannungen zwischen 0V und 2,5V beschränkt, weshalb eingehende Signale mit größeren Spannungen immer zuerst umgewandelt werden müssen, um den Prozessor nicht zu schädigen. Das Board verfügt über 28 Pins, von denen 23 als General Purpose Input/Output (GPIO)-Pins per Software konfiguriert werden können. Neben dieser Standard-Funktionalität der Pins, können einige Pins besondere Funktionalitäten umsetzen, wenn sie entsprechend konfiguriert werden. Das Board verfügt beispielsweise über zwei Pins, die als Analog-to-Digital Converter (ADC) genutzt werden können, um mit deren Hilfe analoge Spannungspegel zu lesen. Neben der Möglichkeit analoge Signale zu erkennen, können auch einige Pins serielle Daten auslesen, indem sie beispielsweise als Universal Asynchronous Receiver-Transmitter (UART) konfiguriert wurden.

2.5 Sensoren Luftqualität

Aufgrund der Tatsache, dass das XDK lediglich Aussagen zu Temperatur, Luftfeuchtigkeit und Luftdruck der Umgebung tätigen kann, werden für die tatsächliche Auswertung der Luftqualität weitere Sensoren benötigt. Die folgende Tabelle listet alle für diese Arbeit verwendeten zusätzlichen Sensoren mit deren wichtigsten Eigenschaften auf:

Sensor Name	Messgröße	Messbereich	Versorgungsspannung	Spannung Ausgang	Signalart
MQ7	Carbon Monoxide (CO)	20ppm-2000ppm	5V	5V	Analog
MG811	Carbon Dioxide (CO2)	350 - 10000 ppm	5V	5V	Analog
MQ131	Ozone (O3)	10PPB-2PPM	5V	5V	Analog
MQ131	Ozone (O3)	1000ppm-10ppm	5V	5V	Analog
MQ135	Schädliche Gase (NO_x)	Abhängig vom Gas	5V	5V	Analog
SDS011	Particulate Matter (PM)	0.0-999.9 ug/m^3	5V	3,3V	UART

Tabelle 2.4: Übersicht Sensoren

Neben dem Feinstaub-Sensor SDS011, welcher die Messdaten digital über UART an das XDK übermittelt, liefern alle anderen Sensoren lediglich ein analoges Ausgangssignal. Die Umsetzung der Spannung an den Ausgängen in Messwerte erfolgt auf Basis der Datenblätter logarithmisch zu den gemessenen Werten.
Abbildung **??** zeigt einen Ausschnitt aus dem Datenblatt des MG811-Sensors und zeigt,

wie genau die Impedanz des Sensors mit steigenden Messwerten logarithmisch ansteigt. Für alle weiteren analogen Sensoren sind diese Kennlinien ähnlich und lassen sich im Anhang in Abschnitt **??** finden.

Das Kriterium, das den größten Einfluss auf die Wahl der Sensoren hat, ist in diesem Projekt deren Kostengünstigkeit.
Dies hat zur Folge, dass die Toleranzen der Sensoren aus Tabelle 4.1 nicht optimal sind und die logarithmische Umsetzung der Messwerte in Spannungen einige Ungenauigkeiten mit sich bringt.
Da der Fokus dieser Arbeit jedoch auf der Entwicklung der Drohne als Messstation liegt und die Auswertung eher vernachlässigt wird, hat dies keine zu großen Auswirkungen.

2.6 FreeRTOS

Das Betriebssystem des XDK ist FreeRTOS, ein Echtzeitbetriebssystem für eingebettete Systeme. FreeRTOS stellt de facto den Standard für Mikrocontroller und kleine Mikroprozessoren dar, nicht zuletzt aufgrund dessen sehr freizügiger Open-Source-Lizenz. Das Betriebssystem ist veröffentlicht unter der MIT Lizenz, welche die kostenlose Nutzung der Software selbst zu kommerziellen Zwecken erlaubt. FreeRTOS ist zum größten Teil in der Programmiersprache C entwickelt, wobei es einzelne Funktionalitäten gibt, die auf Assembler-Ebene umgesetzt sind. Im Folgenden werden die beiden Begriffe Tasks und Timers näher betrachtet, da diese bei der Entwicklung von Software für das XDK eine große Rolle spielen.

2.6.1 Tasks

Eine auf einem Echtzeitbetriebssystem laufende Applikation kann in mehrere Tasks unterteilt werden. Die Tasks sind unabhängig voneinander und haben jeweils einen eigenen Task-Kontext. Es kann immer nur genau ein Task innerhalb des System aktiv sein, sodass im Falle mehrerer parallel laufender Tasks ein Scheduler diesen zur Laufzeit Rechenleistung zuweist und entscheidet, welcher Task gerade aktiv ist.

2.6.2 Timers

Ein Timer wird verwendet, wenn eine Funktion zu einem festen Zeitpunkt ausgeführt werden soll. Die Funktion, die nach Ablauf des Timers ausgeführt wird, bezeichnet man als Callback-Funktion. Neben Timern, die ein einmaliges Ereignis zu einem festen Zeitpunkt

triggern, ist es möglich den Timer jedes mal nach dessen Ablauf neu zu starten, sodass die Callback-Funktion periodisch aufgerufen wird.

3 Planung

3.1 Anforderungsanalyse

Basierend auf der grundlegenden Idee, eine Drohne mit Sensoren auszustatten, um damit die Luftqualität messen zu können, ist eine Anforderungsanalyse entstanden, welche die Funktionalitäten der ausgestatteten Drone eindeutig spezifiziert. Die vollständige Anforderungsanalyse findet sich im Anhang.

3.1.1 Muss-Kriterien

Im Folgenden werden zunächst die Kernanforderungen näher erläutert und warum diese absolute Priorität genießen.

i. Der Drohnenflug muss mittels der App gestartet werden können

ii. Die Flugroute muss mittels der App festgelegt werden können

iii. Der Drohnenflug muss mittels der App abgebrochen werden können

iv. Die Messwerte müssen über die App einsehbar sein

v. Die Messwerte müssen über die App exportierbar sein

vi. Die App muss dem Benutzer ermöglichen, einen Flugbereich auf einer Karte zu markieren

vii. Die App muss dem Benutzer ermöglichen, die Flughöhe für die Messung auszuwählen

viii. Die auf der Karte ausgewählte Flugroute muss gestartet werden können

ix. Die App muss dem Benutzer ermöglichen, aus verschiedenen Messhäufigkeiten auszuwählen

x. Die Drohne muss Feinstaub messen können (2.5 & 10 μm)

xi. Die Drohne muss Druck messen können

xii. Die Drohne muss Feuchtigkeit messen können

xiii. Die Drohne muss Temperatur messen können

xiv. Die Drohne muss Stickoxide (NO_x) messen können

Wie aus den obigen Anforderungen ersichtlich ist, liegt das Hauptaugenmerk darauf, die Drohne mittels einer App ansprechen und steuern zu können, während die Sensoren die gemessenen Umgebungsdaten an die App senden, damit diese dort angezeigt werden. Das Senden sowie das Verarbeiten der Sensordaten geschieht mittels des Bosch Cross Domain Development Kit (XDK), welches als Schnittstelle zwischen den zum Teil analogen Sensoren und der App dient.

3.1.2 Soll-Kriterien

In Ergänzung zu diesen kritischen Anforderungen, beschreiben die folgenden Anforderungen Funktionalitäten die ebenfalls für sinnvoll erachtet wurden, die jedoch nicht besonders kritisch sind.

i. Die Messwerte sollen in der App visualisiert werden

ii. Die App soll die Messdaten in Abhängigkeit der Höhe zweidimensional auf der Karte anzeigen können

iii. Basierend auf dem auswählbaren Flugbereichs auf einer Karte, soll die App eine Flugroute automatisch berechnen können.

iv. Die App soll dem Benutzer ermöglichen, Bereiche explizit aus dem Flugbereich auszuschließen

v. Die App soll die Drohnenposition während des Fluges auf einer Karte anzeigen

vi. Die App soll eine Fortschrittsanzeige zur laufenden Messung bieten

vii. Die App soll einen permanenten Video-Stream während des Drohnenflugs anzeigen

viii. Die Drohne soll Kohlenstoffdioxid (CO_2) messen können

ix. Die Drohne soll Ozon (O_3) messen können

Wie man erkennen kann, werden hier einige zu messende Größen ergänzt, sowie einige graphischen Verbesserungen, die eine bessere Rückmeldung zur laufenden Messung bieten sollen.

3.1.3 Kann-Kriterien

3.1.4

Sollten die oben genannten Funktionalitäten noch immer nicht den Rahmen des Projektes sprengen, sind mit den folgenden Anforderungen noch einige optionale Funktionalitäten erfasst, die potentiell einen weiteren Mehrwert liefern.

i. Der Benutzer kann in der App Flugrouten erstellen, speichern und abrufen

ii. Der Benutzer kann in der App Messprofile erstellen, speichern und abrufen

iii. Die App kann die Benutzerprofile exportieren können

iv. Die App kann durch Verrechnung von verbleibendem Akkustand und der vorgegebenen Flugroute einen Warnhinweis an den Benutzer geben, dass die Messung potenziell nicht ausgeführt werden kann

v. Die App kann die Messdaten dreidimensional darstellen

vi. Die Drohne kann Kohlenstoffmonoxid (CO) messen können

vii. Die Drohne kann Schwefeldioxid (SO_2) messen können

viii. Die Drohne kann flüchtige organische Verbindungen (VOC) messen können

ix. Die Drohne kann Methan (CH_4) messen können

x. Der Anwender kann die Messdaten mittels der App einem Server übermitteln können

3.2 Aufgabenteilung

Um eine klare Trennung und Zuordnung der Aufgaben zu erreichen, ist das Projekt grob in 2 Bereiche eingeteilt.
Die erste Komponente ist die iOS-App, welche für die Ansteuerung der Drohne, sowie die Visualisierung der Messdaten zuständig ist.
Die zweite Komponente ist der Entwurf des Messsystems mit allen Sensoren, deren Verschaltung und der Verarbeitung deren Daten mittels des Bosch XDK.

Alle Aufgaben hinsichtlich der App werden von Julian Riegger bearbeitet, während alle Aufgaben zum Messsystem von Sebastian Breit bearbeitet werden.

3.3 Zeitplan

In Ergänzung zur bereits erwähnten Aufgabenteilung, wird zur besseren Planung und Nachverfolgung des Projektes ein Zeitplan verwendet, der auf einer hohen Abstraktionsebene die Aufgaben zeitlich einplant. Der Zeitplan ist mittels des frei verfügbaren Online-Tools Agantty erstellt, mit dessen Hilfe man einfach und unkompliziert Projekt-Pläne erstellen kann und dabei spezifischen Aufgaben bestimmten Personen zuordnen kann.

Abbildung 3.1 zeigt den erstellten Zeitplan.

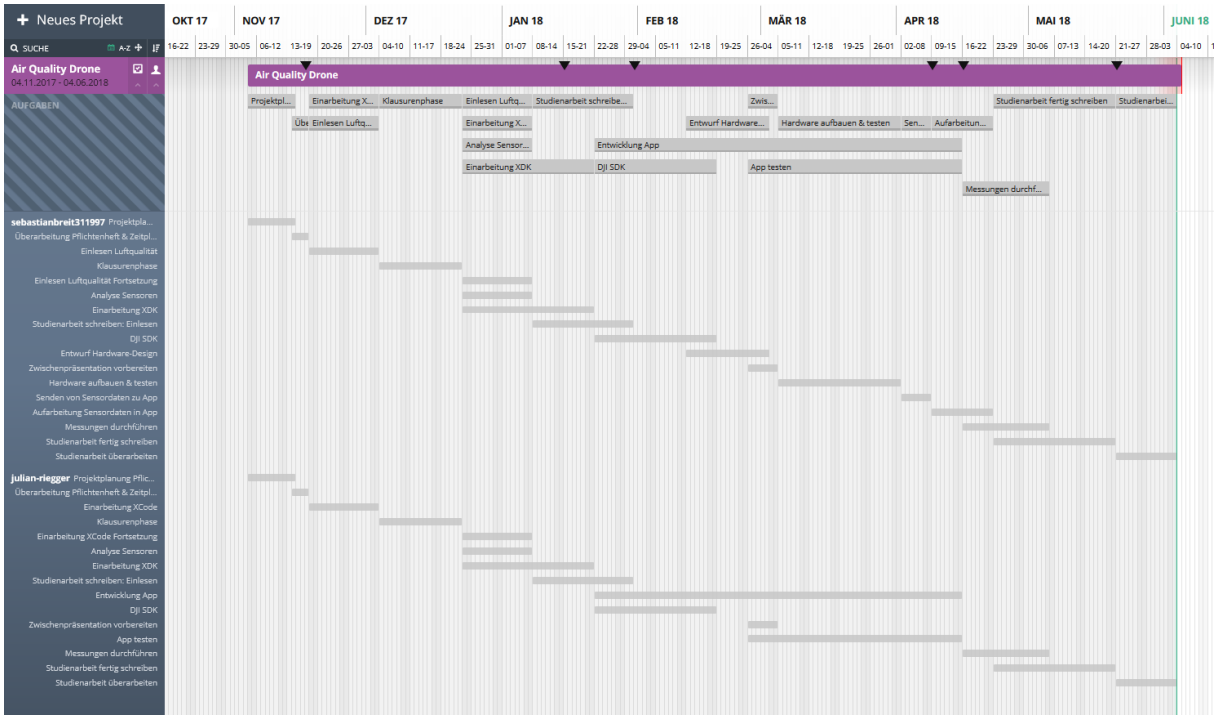

Abbildung 3.1: Zeitplan

4 Architektur

4.1 Hardware

Aufgrund der Tatsache, dass der XDK lediglich Aussagen zur Temperatur, Luftfeuchtigkeit und dem Luftdruck in der Umgebung liefern kann, werden für das Auswerten der Luftqualität der Umgebung weitere Sensoren benötigt. Die folgende Tabelle listet alle verwendeten externen Sensoren mitsamt deren wichtigsten Eigenschaften auf:

Sensor Name	Messgröße	Messbereich	Versorgungsspannung	Spannung Ausgang	Signalart
MQ7	Carbon Monoxide (CO)	20ppm-2000ppm	5V	5V	Analog
MG811	Carbon Dioxide (CO2)	350 - 10000 ppm	5V	5V	Analog
MQ131	Ozone (O3)	10PPB-2PPM	5V	5V	Analog
MQ131	Ozone (O3)	1000ppm-10ppm	5V	5V	Analog
MQ135	Schädliche Gase (NO_x)	Abhängig vom Gas	5V	5V	Analog
SDS011	Particulate Matter (PM)	0.0-999.9 ug/m^3	5V	3,3V	UART

Tabelle 4.1: Übersicht Sensoren

Eine genaue Scalierung der Sensorwerte

4.2 Architektur Konzept

Wie in Abbildung 4.1 zu sehen ist, besteht das System im Kern aus 3 Komponenten. Die Drohne, das XDK sowie das Tablet sind alle mit dem Wifi der Drohne verbunden. Über dieses läuft dann die komplette Kommunikation zwischen den einzelnen Komponenten. Die App kommuniziert über das DJI-SDK mit der Drohne, das XDK baut hingegen eine UDP-Verbindung zur App auf, über die es dann die gemessenen Sensordaten schicken kann.

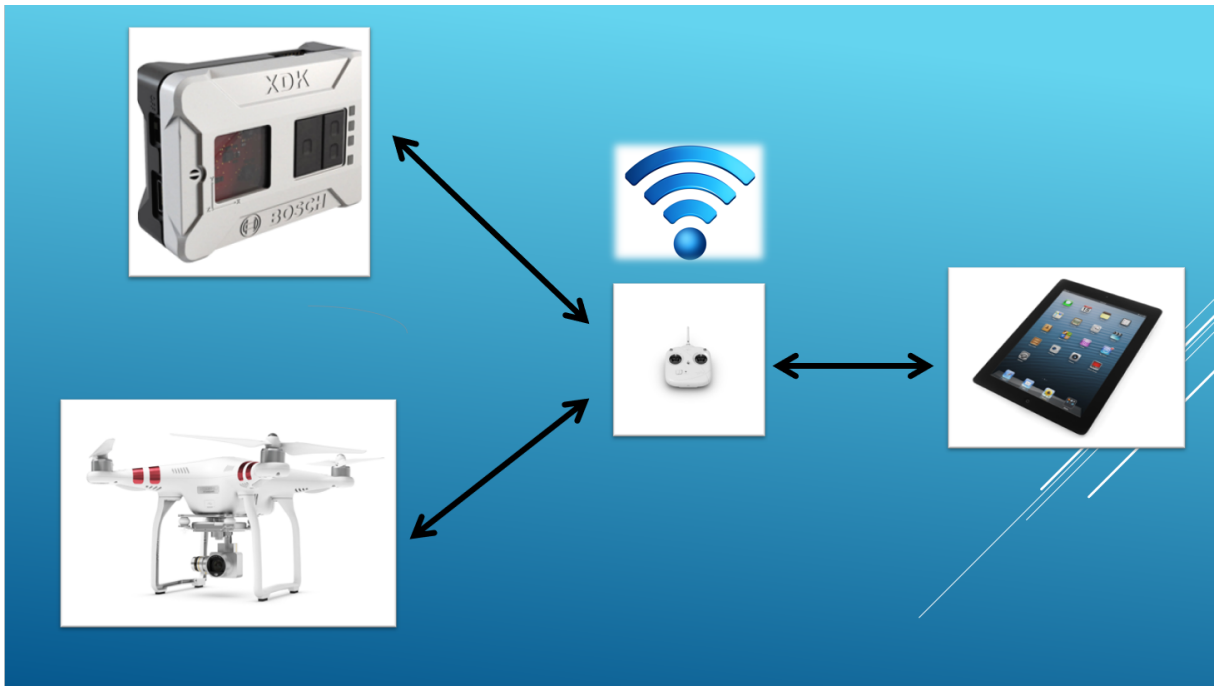

Abbildung 4.1: Architektur allgemein

4.3 Lesen der Sensordaten

Abbildung 4.2 zeigt schematisch das Auslesen der Sensordaten am XDK. Das XDK ist mit dem dazu passenden Extension-Board direkt über eine serielle Schnittstelle verbunden, über die es die Messwerte der externen Sensoren empfängt.
Für die analogen Sensoren werden prinzipiell deren Spannungspegel am Ausgang auf einen Multiplexer geleitet, der in Abhängigkeit des Signales vom Extension-Board entscheidet, welcher Sensor-Ausgang auf den Ausgang des Multiplexers übertragen wird. Die Ansteuerung des Multiplexers erfolgt auf Seiten der Software auf dem XDK, die festlegt, wie genau die GPIO-Pins des Extension-Boards zu beschalten sind. Abschließend wird dann der Wert am Ausgang des Multiplexers vom Analog-Digital-Wandler des Extension-Boards erfasst und an das XDK übermittelt.
Die Messwerte des Feinstaub-Sensors SDS011 können lediglich über UART erfasst werden. Aus diesem Grund wird der Sensor nicht mit dem Multiplexer verbunden, sodern direkt, beziehungsweise nach Anpassung des Spannungs-Pegels, mit dem Extension-Board verbunden. In der Software des XDK lässt sich dann konfigurieren, dass die mit dem Sensor verbundenen Pins für UART genutzt werden sollen, womit die Messwerte somit direkt in digitaler Form am XDK erfasst werden können.
Aufgrund dessen, dass der Feinstaub-Sensor die gemessenen Daten höchstens im Sekundentakt per UART übermitteln kann, ist die maximale Messdichte ebenfalls auf 1 Messung pro Sekunde beschränkt. Damit die 5 analogen Sensoren auch mindestens einmal in diesem

Intervall ihre Daten an das XDK weitergeben, werden die GPIO-Pins von der Software alle 100ms geändert, sodass immer ein anderer Sensor am Analog-Digital-Wandler anliegt. Sind nun in einer Sekunde alle Daten einmal gemessen, so sendet das XDK diese sofort per UDP über das Wifi der Drohne an die iOS-Applikation, wo die Messdaten dann visualisiert werden können.

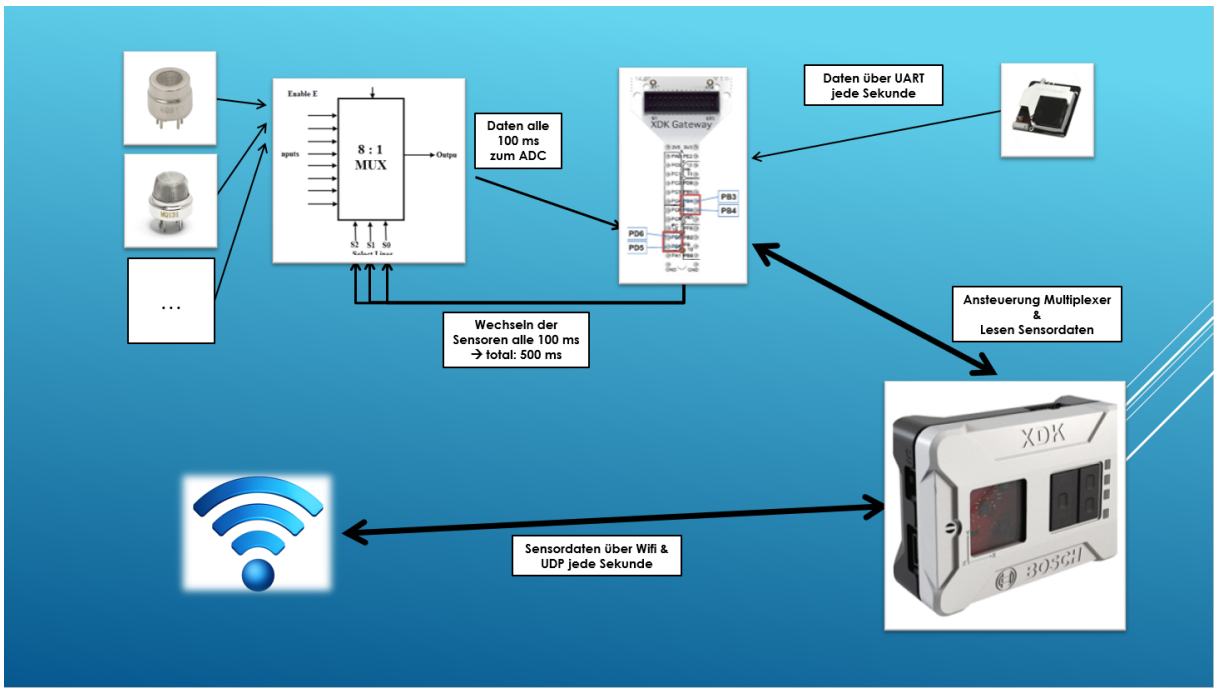

Abbildung 4.2: Architektur Sensorik

Wie wir basierend auf der vorangegangenen Tabelle erkennen können, liefert kein Sensor einen Pegel von 2,5V, wie er vom XDK benötigt wird. Damit die von den Sensoren gelieferten Daten nun also verarbeitet werden können, müssen die Pegel angepasst werden. Um dies zu erreichen, wird ein simpler Spannungsteiler angewendet, der die Ausgangsspannung um die Hälfte reduziert. Neben der Anpassung der Pegel ist noch einweiterer Schritt erforderlich, um die analogen Sensordaten lesen zu können. Bedingt dadurch, dass 5 analoge Sensoren auszuwerten sind und dafür lediglich 2 Pins verfügbar sind, welche als ADC genutzt werden können, müssen mehrere der Sensoren mit einem ADC gemessen werden. Um dies zu ermöglichen, werden die Sensordaten mit Hilfe eines Multiplexers in verschiedenen Zeitschlitzen übermittelt. Um den Multiplexer anzusteuern werden dabei die GPIO-Pins des XDK verwendet.

Neben den 5 analogen Sensoren muss auch das Signal des Feinstaub-Sensors angepasst werden, welcher die Daten seriell über UART mit 3,3V übermittelt. Um dieses Signal auf die erforderlichen 2,5V umzuwandeln, wird ein fertiger Pegelwandler verwendet, der diskrete Pegel konvertieren kann.

4.4 Schaltungsdesign

Da die Sensoren aus Tabelle 4.1 Ausgangs-Spannungen zwischen 0V und 5V liefern, müssen diese Spannungen noch umgewandelt werden, bevor sie mit dem Extension-Board und somit dem XDK verbunden werden dürfen. Aus diesem Grund werden mehrere Pegel-Wandler und Operationsverstärker eingesetzt, damit die einzelnen Komponenten kompatibel werden.
In diesem Abschnitt wird nun näher auf das konkrete Schaltungs-Design eingegangen, wie die Bauteile elektrisch miteinander verbunden sind und wie die Spannungen umgewandelt werden, sodass die einzelnen Komponenten miteinander kommunizieren können. Ein Überblick über das gesamte Schaltungslayout lässt sich im Anhang finden.
Die gezeigten Schaltungs-Layouts und die dabei verwendeten Komponenten sind mit Hilfe der gratis Computer-Aided Design (CAD)-Software PCBWeb entworfen worden, die sowohl zum Entwurf schematischen Schaltplänen, als auch zur Erstellung von Printed Circuit Board (PCB)-Layouts verwendet werden kann.

4.4.1 Extension-Board

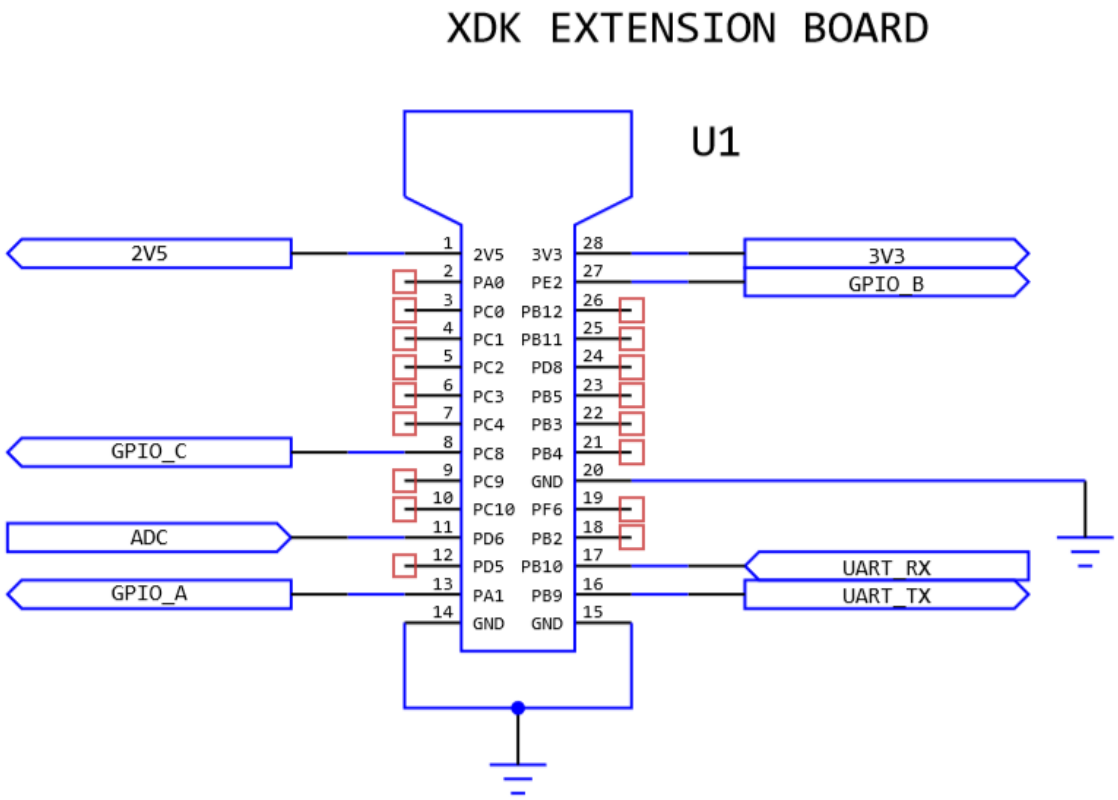

Abbildung 4.3: Layout Extension-Board

Abbildung 4.3 zeigt ganz konkret die Beschaltung des Boards und wie die angeschlossenen Pins verwendet werden. Die Pins 1 und 28 dienen als Spannungsquelle für alle Komponenten, die Spannungen von 2,5V oder 3,3V benötigen.
Die Pins PA1, PE2 sowie PC8 werden als GPIO-Pins benutzt, mit denen der Multiplexer angesteuert wird.
PD6 ist konfiguriert um als Analog-Digital-Wandler die Spannungen der analogen Sensoren zu digitalisieren. Dazu wird er nach einer Anpassung des Spannungspegels mit dem Multiplexer verbunden.
Die beiden Pins PB9 und PB10 sind für UART konfiguriert und werden mit letztlich mit dem Feinstaub-Sensor SDS011 verbunden.

4.4.2 Multiplexer

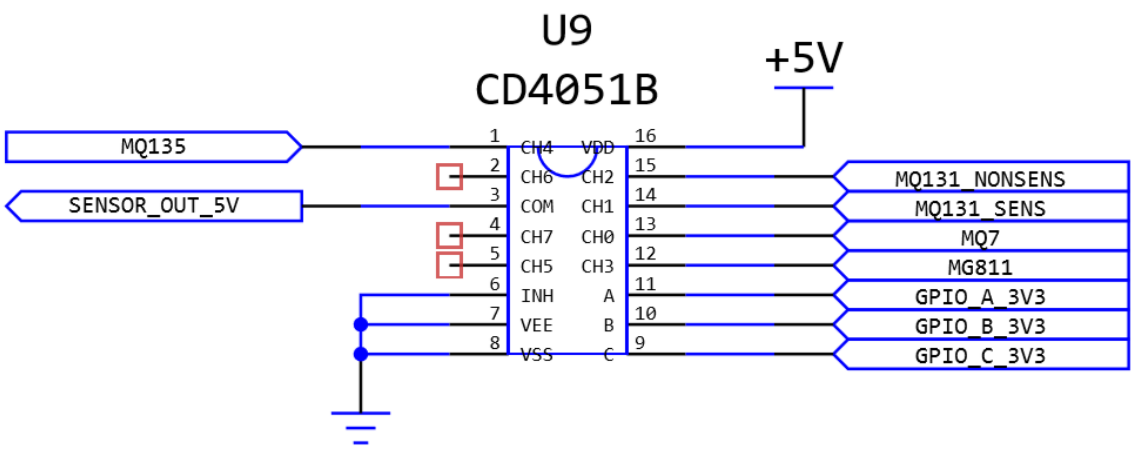

Abbildung 4.4: Layout Multiplexer

Damit der Multiplexer die Spannungs-Werte der Sensoren auf den Ausgang durchschalten kann, benötigt er an den Steuerungs-Eingängen A, B und C Spannungen von mindestens 3,3V. Aus diesem Grund können nicht einfach die GPIO-Pins des Extension-Boards direkt mit dem Multiplexer verbunden werden, da diese lediglich 2,5V liefern. Damit diese Pegel angepasst werden, werden die GPIO-Pins somit zunächst mit einem Pegelwandler verbunden, wie in Abbildung 4.5 zu sehen ist.
Wenn somit die Steuerung des Multiplexers funktioniert, wird abhängig von der Beschaltung der Pins A, B und C der Ausgang COM auf den passenden Eingangskanal durchgeschaltet. Tabelle 4.2 zeigt die logischen Verknüpfungen.

A	B	C	Output (Channel)
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

Tabelle 4.2: Logische Ansteuerung Multiplexer

Ist nun ein bestimmter Eingangskanal auf den Ausgang durchgeschaltet, so kann das Signal trotzdem noch nicht mit dem ADC verbunden werden, da die Spannungswerte des Signals zwischen 0V und 5V liegen können. Da das XDK nur Spannungen bis zu 2,5V verträgt, muss die Spannung auf diesen Bereich umgewandelt werden. Wie dies zu erreichen ist, wird in Abschnitt 4.4.4 näher erläutert.

4.4.3 Pegelwandler

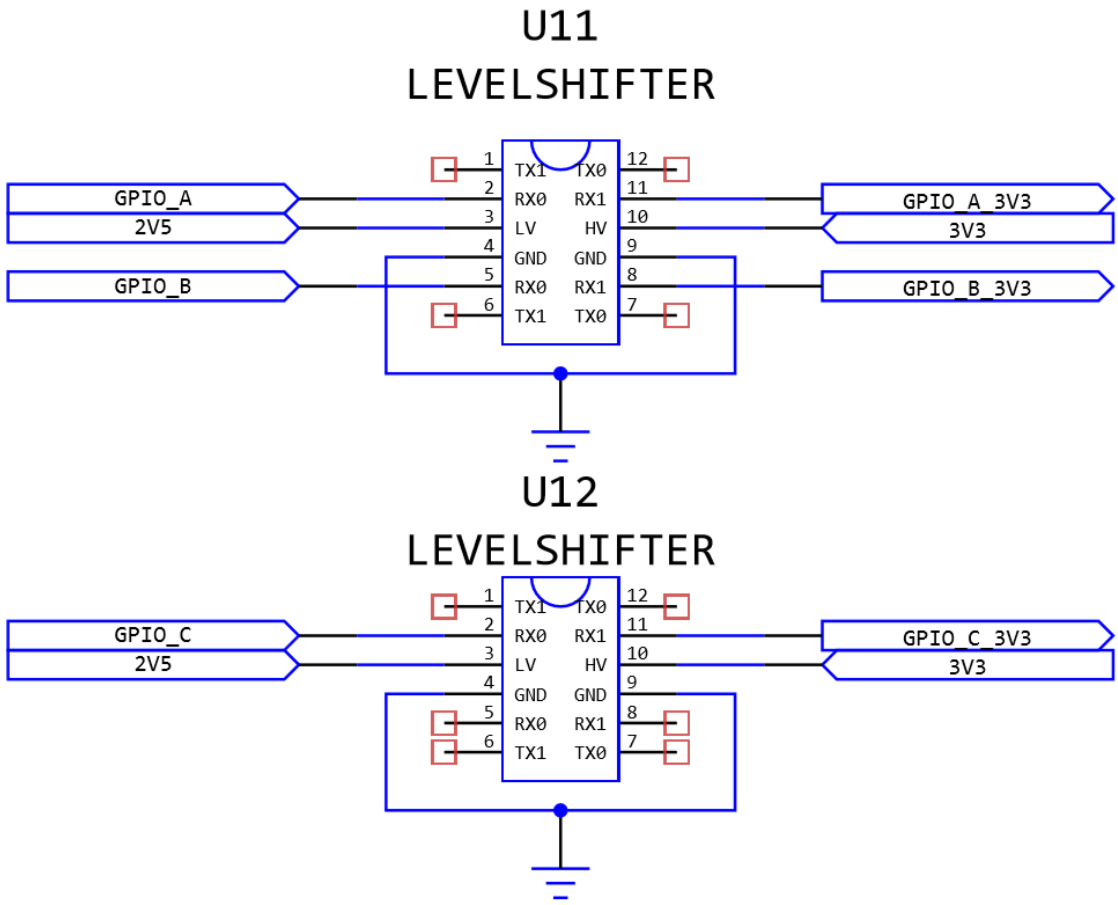

Abbildung 4.5: Layout Pegelwandler

Wie bereits zuvor erwähnt werden die beiden Pegelwandler benutzt, um die Spannung der von den GPIO-Pins eingehenden Signale auf 3,3V anzuheben, damit der Multiplexer diese verarbeiten kann.

4.4.4 Operationsverstärker

Wie bereits im Abschnitt 4.4.2 erwähnt, muss das Ausgangssignal dessen auss dem Intervall von 0V bis 5V in das Intervall von 0V bis 2,5V konvertiert werden. Dies geschieht mittels eines Operationsverstärkers und der dazugehörigen Grundschaltung des invertierenden Verstärkers, welche in Abbildung **??** zu sehen ist.

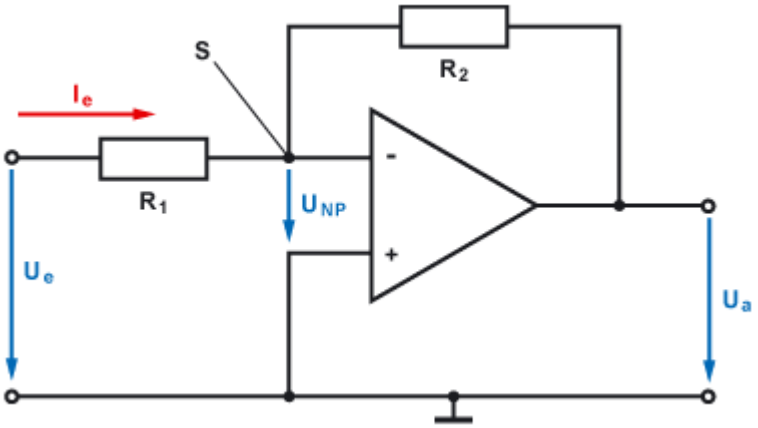

Abbildung 4.6: Invertierender Verstärker

Laut der Grundgleichung des invertierenden Verstärkers wird der Verstärkungsfaktor V berechnet aus:

$$V_u = \frac{U_a}{U_e} = -\frac{R_2}{R_1}$$

Um somit eine Dämpfung um den Faktor 2 zu erhalten, beschalten wir einen ersten Operationsverstärker mit $R_1 = 20K\Omega$ und $R_2 = 10K\Omega$. Da nun das Signal allerdings im Intervall von -2,5V bis 0V liegt, wird ein weiterer Operationsverstärker damit beschaltet, der die Wiederstände $R_3 = 10K\Omega$ und $R_4 = 10K\Omega$ besitzt, wodurch das Signal erneut invertiert wird und es somit im korrekten Intervall erscheint.
Diese schematische Beschaltung ist in Abbildung 4.7 zu erkennen.

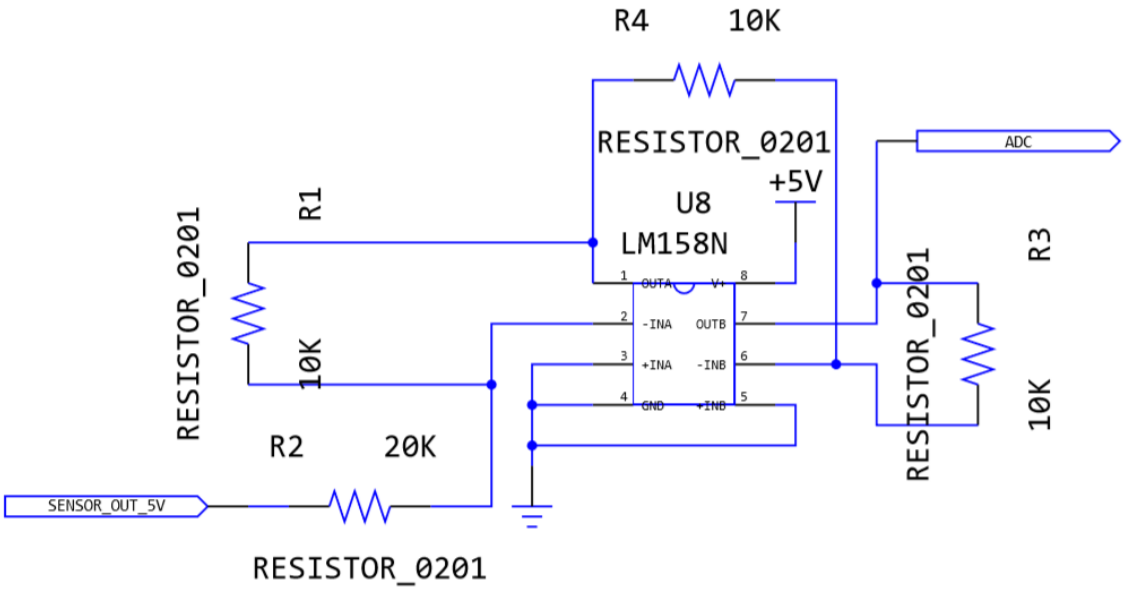

Abbildung 4.7: Layout Operationsverstärker

4.4.5 Feinstaub-Sensor

Abbildung 4.8 zeigt die Beschaltung des Feinstaub Sensors. Man kann erkennen, dass auch hier die Signale des Sensors zunächst auf einen Pegelwandler geschickt werden, damit dieser die Spannungspegel von 3,3V auf 2,5V umwandelt. Nach der Umwandlung am Pegelwandler werden die daraus resultierenden Signale direkt mit den entsprechenden Pins am XDK Extension-Board verbunden, an denen die digitalisierten Messwerte über UART ausgelesen werden.

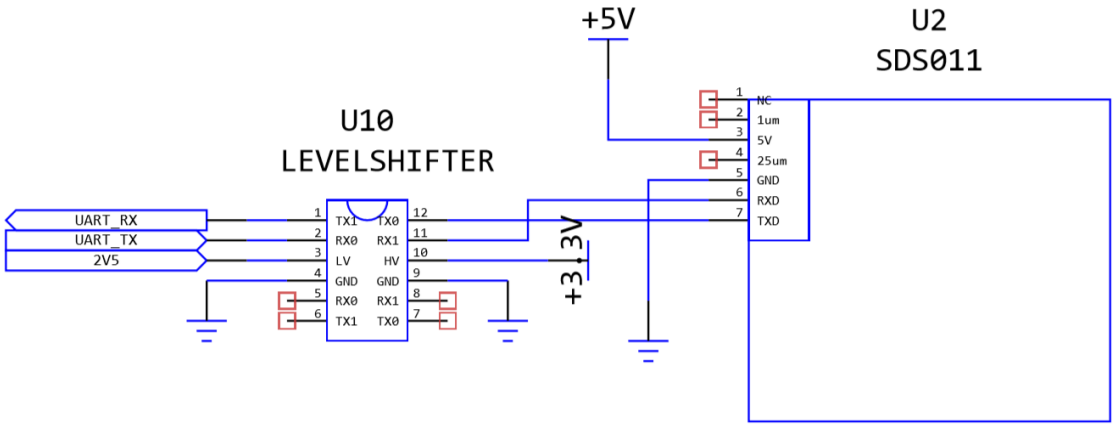

Abbildung 4.8: Layout Feinstaub-Sensor

4.4.6 Layout analoger Sensor

Zum Abschluss der Schaltungs-Layouts zeigt Abbildung 4.9 eine beispielhafte Beschaltung eines der analogen Sensoren. Da dies bei allen praktisch gleich aussieht, wird hier nur die Abbildung des MQ7-Sensors zur Veranschaulichung dargestellt. Die vollständigen Layouts der Sensoren finden sich im Anhang.

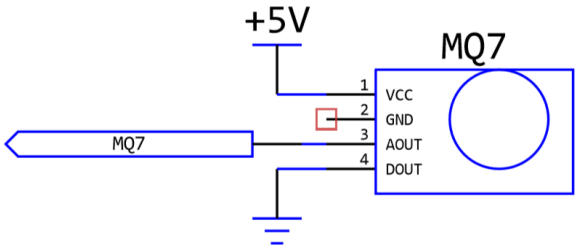

Abbildung 4.9: Layout analoger Sensor

5 Umsetzung

5.1 XDK Code

Zur Steuerung und zum Erfassen der Messwerte der Sensoren wird C Code verwendet der auf das Bosch XDK aufgespielt wird. Wie in Kapitel 2.4 erwähnt, läuft auf dem XDK das Echtzeitbetriebssystem FreeRTOS, das Programme in verschiedene Tasks unterteilt. Auch der Code zum Verwalten der Sensoren ist in diverse Tasks unterteilt, welche alle quasi-parallel ablaufen.
In den folgenden Abschnitten wird näher auf den Aufbau des Programmes eingegangen, sowie ein kurzer Überblick über die wichtigsten Funktionalitäten geboten.

5.1.1 Programm Aufbau

Ein Programm für das Bosch XDK ist Grundsätzlich immer gleich aufgebaut. Es besteht aus einer Funktion zur Initialisierung des Prozessors, sowie einem oder mehreren Tasks, welche zur Prozessor-Initialisierung angelegt und gestartet werden. Zusätzlich gibt es wie in gewöhnlichem C Code auch normale Funktionen, Subroutinen und Variablen, welche von den unterschiedlichen Tasks aufgerufen und verändert werden können. Im entwickelten Programm gibt es genau 2 Variablen, auf die alle Tasks zugreifen können und somit diese lesen und schreiben können.

```
1  int sensorNo=0;
2
3  struct messagePayload
4  {
5    float temperature;    //degrees
6    float pressure;     //Pa
7    float humidity;     //% relativeHum
8    float pm25;       //in ug/m3
9    float pm10;       //in ug/m3
10   float co;       //relativ (0-1)
11   float co2;        //relativ (0-1)
```

```
12   float o3sensitive;     //relativ (0-1)
13   float o3lessSensitive;  //relativ (0-1)
14   float hazardousGas;    //relativ (0-1)
15 } payload;
```

Listing 5.1: Geteilte Variablen

Beide Variablen sind in Listing 5.1 zu erkennen.
Zum einen ist dies die Variable 'SensorNo', in welcher immer der aktuelle Sensor angegeben ist, dessen Ausgang am ADC anliegt.
Zum anderen ist die Struktur 'messagePayload' von allen Tasks aus zugreifbar, da hierin die letzten gemessenen Daten zwischengespeichert werden, bevor sie an die App versendet werden oder von neuen Daten überschrieben werden.

Code-Listing 5.2 zeigt die Initialisierung des Prozessors für die entwickelte Applikation.

```
1  void appInitSystem(void * CmdProcessorHandle, uint32_t param2)
2  {
3  if (CmdProcessorHandle == NULL)
4  {
5  printf("Command processor handle is null \r\n");
6  assert(false);
7  }
8  AppCmdProcessorHandle = (CmdProcessor_T *) CmdProcessorHandle;
9  Board_EnablePowerSupply3V3(EXTENSION_BOARD);
10 BCDS_UNUSED(param2);
11
12 initUART();
13
14 //-------- Init Tasks -------------
15 initEnvironmental();
16 gpioInit();
17 scanAdcInit();
18 initUDP();
19 createNewUARTTask();
20 }
```

Listing 5.2: Prozessor Initialisierung

Nach der Prüfung auf einen erfolgreichen Zugriff auf den Prozessor und nach der Zuweisung eines Prozessor-Handles, welcher für die Ausführung des Programms erforderlich ist, wird zunächst die 3,3V Spannungsversorgung des Extension-Boards aktiviert. Anschließend wird eine gewöhnliche Funktion aufgerufen, welche Pins des Extension-Boards für eine UART Kommunikation konfiguriert.

Die nächsten 5 Funktionsaufrufe sind nun verschieden vom Vorhergehenden. In jeder dieser Funktionen wird ein neuer Task angelegt und gestartet. Anhand deren Namen lässt sich leicht die weitere Gliederung des Programmes erkennen.
Es gibt jeweils einen Task für jede der folgenden Teilfunktionen des Programmes:

- Auslesen der internen Sensoren zur Bestimmung von Temperatur, Luftfeuchtigkeit und Luftdruck
- Ansteuerung des Multiplexers durch GPIO-Pins
- Auslesen der Spannung am Analog-Digital-Wandler
- Senden der Messdaten an die iOS-App über UDP
- Auslesen der Sensordaten des Feinstaub-Sensors über UART

Die folgenden Abschnitte gehen nun jeweils näher auf jeden dieser Tasks ein.

5.1.2 Auslesen interner Sensoren

```
1  static void readEnvironmental(xTimerHandle xTimer)
2  {
3    (void) xTimer;
4
5    Retcode_T returnValue = RETCODE_FAILURE;
6
7    /* read and print BME280 environmental sensor data */
8
9    Environmental_Data_T bme280 = { INT32_C(0), UINT32_C(0), UINT32_C(0)
         };
10
11   returnValue = Environmental_readData(xdkEnvironmental_BME280_Handle, &
         bme280);
12
13   if ( RETCODE_OK == returnValue) {
14     printf("BME280 Environmental Data : p =%ld Pa T =%ld mDeg h =%ld %%
           rh\n\r",
15     (long int) bme280.pressure, (long int) bme280.temperature, (long int
           ) bme280.humidity);
16     payload.temperature=(float)bme280.temperature / 1000;
17     payload.pressure=(float)bme280.pressure;
18     payload.humidity=(float)bme280.humidity / 100;
19   }
20 }
```

Listing 5.3: Read Environmental Data

Code-Listing 5.3 zeigt den Task, der die Daten der internen Sensoren erfasst. Dazu wird zunächst initialisiert, welche Sensoren ausgelesen werden sollen, um anschließend deren Daten abzufragen. Die erhaltenen Messdaten werden danach noch formatiert, sodass die Temperatur in Grad Celsius, der Luftdruck in Pascal (Pa) und die Luftfeuchtigkeit als relativer Wert zwischen 0 und 1 angegeben wird. Abschließend werden die gemessenen Daten in einem temporären Zwischenspeicher abgelegt, auf den die anderen Tasks ebenfalls zugreifen können.

```
1  static void initEnvironmental(void)
2  {
3    Retcode_T returnValue = RETCODE_FAILURE;
4    Retcode_T returnOverSamplingValue = RETCODE_FAILURE;
5    Retcode_T returnFilterValue = RETCODE_FAILURE;
6
7    /* initialize environmental sensor */
8
9    returnValue = Environmental_init(xdkEnvironmental_BME280_Handle);
10   if ( RETCODE_OK != returnValue) {
11     printf("BME280 Environmental Sensor initialization failed\n\r");
12   }
13
14   returnOverSamplingValue = Environmental_setOverSamplingPressure(
         xdkEnvironmental_BME280_Handle,ENVIRONMENTAL_BME280_OVERSAMP_2X);
15   if (RETCODE_OK != returnOverSamplingValue) {
16     printf("Configuring pressure oversampling failed \n\r");
17   }
18
19   returnFilterValue = Environmental_setFilterCoefficient(
         xdkEnvironmental_BME280_Handle,ENVIRONMENTAL_BME280_FILTER_COEFF_2
         );
20   if (RETCODE_OK != returnFilterValue) {
21     printf("Configuring pressure filter coefficient failed \n\r");
22   }
23
24   environmentalHandle = xTimerCreate((const char *) "readEnvironmental",
          ONESECONDDELAY,TIMER_AUTORELOAD_ON, NULL, readEnvironmental);
25
26   xTimerStart(environmentalHandle,TIMERBLOCKTIME);
27 }
```

Listing 5.4: Task-Initialisierung: Interne Sensoren

Listing 5.4 zeigt die Erstellung eines neuen Tasks und das starten des selben am Beispiel der internen Sensoren. Zunächst werden hier die internen Sensoren konfiguriert, um nach dem erfolgreichen Abschluss den zuvor beschriebenen Task zu starten, welcher die Messdaten liest.

In diesem Fall geschieht dies mittels eines Timers, welcher die in Listing 5.3 beschriebene Callback-Funktion immer wieder aufruft, sobald der Timer abgelaufen ist. Genau genommen ist also nicht die aufgerufene Funktion der Task, sondern der Timer läuft in einem Task und ruft lediglich bei seinem Ablauf die Callback-Funktion auf.
Der Zeitpunkt an dem dies geschieht ist durch die Zeit 'ONESECONDDELAY' und den Parameter 'TIMER_AUTORELOAD_ON' fest definiert. In diesem falle würde es bedeuten, dass der Timer nach einer Sekunde abläuft, die Callback-Funktion aufruft und wieder von vorne beginnt, womit die Callback-Funktion periodisch im Sekundentakt aufgerufen wird.

5.1.3 Ansteuerung des Multiplexers

```
1  void gpioInit(){
2    /* initialize local variables */
3    GPIOA.Port = gpioPortA;
4    GPIOA.Pin = 1;
5    GPIOA.Mode = gpioModePushPull;
6    GPIOA.InitialState = MCU_GPIO_PIN_STATE_LOW;
7
8    GPIOB.Port = gpioPortE;
9    / ...
10
11   /* Initialization activities for PTD driver */
12   MCU_GPIO_Initialize(&GPIOA);
13   MCU_GPIO_Initialize(&GPIOB);
14   MCU_GPIO_Initialize(&GPIOC);
15
16   gpioTimerHandle = xTimerCreate(
17   (const char *) "ADC read", ONESECONDDELAY/10,
18   TIMER_AUTORELOAD_ON, NULL, gpioTask);
19
20   xTimerStart(gpioTimerHandle, TIMERBLOCKTIME);
21 }
```

Listing 5.5: GPIO Task Initialisierung

Listing 5.5 zeigt die Task Initialisierung für die Ansteuerung der GPIO-Pins. Neben der Erstellung und dem Start eines Timers, der periodisch nach 100ms die Callback-Funktion aufruft, werden die entsprechenden Pins einmalig konfiguriert, damit sie als GPIO-Pins genutzt werden können.

```
1  static void gpioTask(xTimerHandle pxTimer2)
2  {
3    int retcode;
```

```
4
5   sensorNo+=1;
6   if(sensorNo>=10){
7     sensorNo=0;
8   }
9
10  switch(sensorNo){
11    case 0: retcode = MCU_GPIO_WritePin(&GPIOA,MCU_GPIO_PIN_STATE_LOW);
12        retcode = MCU_GPIO_WritePin(&GPIOB,MCU_GPIO_PIN_STATE_LOW);
13        retcode = MCU_GPIO_WritePin(&GPIOC,MCU_GPIO_PIN_STATE_LOW);
14        printf("0 set.\n\r");
15        break;
16    case 1: retcode = MCU_GPIO_WritePin(&GPIOA,MCU_GPIO_PIN_STATE_HIGH);
17        retcode = MCU_GPIO_WritePin(&GPIOB,MCU_GPIO_PIN_STATE_LOW);
18        retcode = MCU_GPIO_WritePin(&GPIOC,MCU_GPIO_PIN_STATE_LOW);
19        printf("1 est.\n\r");
20        break;
21    // ...
22  }
23  if(retcode!= RETCODE_OK){
24    printf("Write low failed!\n\r\n\r");
25  }
26 }
```

Listing 5.6: GPIO Task

Listing 5.6 zeigt nun die tatsächliche Ansteuerung des Multiplexers über die GPIO-Pins des XDK in der Callback-Funktion. Hierzu wird einfach mit den logischen Werten der Pins ein Zähler implementiert, der hochzählt und somit die entsprechenden Kanäle des Multiplexers anspricht. Die selbe Logik wurde bereits in Tabelle 4.2 gezeigt. Sobald die Sensornummer 10 überschreitet, wird diese wieder auf 0 gesetzt, womit genau zu beginn eines neuen Zyklus der erste Kanal des Multiplexers wieder angesprochen wird.

5.1.4 Auslesen der Spannung am ADC

```
1  static void scanAdc(xTimerHandle pxTimer)
2  {
3    /* Initialize the Variables */
4    (void) (pxTimer);
5    uint32_t _adc0ChData = 0;
6    uint8_t _channelsScanned = 0;
7
8    /* Start the ADC Scan */
9    ADC_Start(ADC0, adcStartScan);
10
```

```
11    for (_channelsScanned = 0; _channelsScanned < NUMBER_OF_CHANNELS-1;
          _channelsScanned++) {
12      /* Wait for Valid Data */
13      while (!(ADC0->STATUS & ADC_STATUS_SCANDV));
14
15      /* Read the Scanned data */
16      _adc0ChData = 0xFFF & ADC_DataScanGet(ADC0);
17      switch(sensorNo){
18        case 0: payload.co=(float)_adc0ChData  /4095;
19            printf("%f 0.\n\r", payload.co);
20            break;
21        case 1: payload.o3sensitive=(float)_adc0ChData  /4095;
22            printf("%f 1.\n\r", payload.o3sensitive);
23            break;
24        // ...
25      }
26    }
27  }
```

Listing 5.7: ADC Scan Task

Listing 5.7 zeigt das Lesen der digitalisierten, quantisierten Spannungswerte am Analog-Digital-Wandler. Es wird immer die Spannung am Eingang des ADC gemessen, jedoch wird abhängig von der globalen Variable 'sensorNo', welche den letzten aktiven Sensor angibt, das Ergebnis in der Struktur 'payload' der passenden Variable zugeordnet. Die Struktur dient lediglich als temporärer Zwischenspeicher der Daten vor dem Versenden an die App.

5.1.5 Senden der Messdaten

```
1  static void wifiUdpSend(void * param1, uint32_t port)
2  {
3    // Socket Initialisierung
4    SockID = sl_Socket(SL_AF_INET, SL_SOCK_DGRAM, (uint32_t) ZERO); /**<
         The return value is a positive number if
5    // Prüfe Socket ...
6
7    char outBuf[1024];
8    sprintf(outBuf, "%f,%f,%f,%f,%f,%f,%f,%f,%f,%f",payload.temperature,
         payload.pressure,payload.humidity,
9    payload.pm25,payload.pm10,payload.co,payload.co2,payload.o3sensitive,
         payload.o3lessSensitive,payload.hazardousGas);
10
```

```
11   Status = sl_SendTo(SockID, (const void *) outBuf, strlen(outBuf), (
         uint32_t) ZERO, (SlSockAddr_t *) &Addr, AddrSize);/**<The return
         value is a number of characters sent;negative if not successful*/
12   // Check Status ...
13
14   Status = sl_Close(SockID);
15   // Check Status ...
16   printf("UDP sending successful\r\n");
17   return;
18 }
```

Listing 5.8: Daten senden über UDP

Wie in Code-Listing 5.8 zu sehen ist, wird im Task jedes Mal ein neues Socket geöffnet und geprüft ob das Socket korrekt initialisiert wurde. Anschließend werden die aktuellsten Messdaten in einen Char-Array gepackt, in dem sie im Format einer **CSV!** (**CSV!**)-Datei zwischengespeichert werden. Erst danach wird das Array mit den so formatierten Daten über UDP an die App geschickt.

Aufgrund der Nutzung von UDP bekommt der Nutzer hier keinerlei direkte Rückmeldung, ob das Paket tatsächlich bei der App ankam oder nicht, jedoch lässt sich in der App recht schnell feststellen, dass Verbindungsprobleme auftreten, sollten keine Messwerte über mehrere Sekunden hinweg empfangen werden, da das Senden der Daten üblicherweise im Sekundentakt erfolgen sollte.
Damit das Paket überhaupt versendet werden kann muss zunächst allerdings eine stabile Verbindung zum Wifi der Drohne aufgebaut sein. Auf die Umsetzung einer Wifi-Verbindung wird in Abschnitt 5.1.7 näher eingegangen.

An dieser Stelle ist darauf hinzuweisen, dass bei der Implementierung bewusst die Entscheidung gegen das Transmission Control Protocol (TCP)-Protokoll gefällt wurde, welches aufgrund der hierbei offenen Verbindung Rückmeldung über verlorene Pakete geben könnte. Diese Entscheidung basiert auf der Feststellung, dass das XDK keinerlei Verbindung über TCP zur App aufbauen konnte, wenn es mit dem Wifi der Drohne verbunden war.

5.1.6 Auslesen des Feinstaub-Sensors

```
1 static void uartTask(){
2   int status;
3   uint8_t buffer [10] ;
4   uint32_t bufLen = sizeof(buffer);
5   for (;;)
6   {
```

```
7       for(int i=0;i<10;i++){
8         buffer[i]=null;
9       }
10      vTaskDelay (1000);
11      status = MCU_UART_Receive(uartHandle,  buffer, bufLen);
12      if(status!=RETCODE_OK){
13        printf("Error at MCU_UART_Receive: %d\r\n",status);
14        printf("Buffer:%s\r\n",buffer);
15      }
16      else{
17        printf("MCU_UART_Receive worked correctly\r\n");
18        for(int i=0;i<10;i++){
19          if(buffer[i]!=null){
20            printf("Bit %d aus Buffer: %d\r\n",i,buffer[i]);
21          }
22        }
23        payload.pm25=((float)buffer[3]*256+buffer[2])/10;
24        payload.pm10=((float)buffer[5]*256+buffer[4])/10;
25      }
26    }
27  }
```

Listing 5.9: UART Task

Die in Listing 5.9 gezeigte Funktion ist der erste wirkliche Task der nicht nur eine Callback-Funktion eines Timers ist. Dieser Task wird zu Programmbeginn initialisiert und gestartet. Aufgrund der Endlosschleife bleibt dieser Task dann solange das Programm läuft und der Prozess nicht von außen beendet wird aktiv. In diesem konkreten Fall ruft der Task immer wieder die Daten am Empfangss-UART-Pin des Extension-Boards ab, rechnet die erhaltenen Bytes um in Feinstaub-Werte und schreibt diese in den temporären Speicher als aktuellste Messwerte.
Die Verzögerung 'vTaskDelay (1000);' sorgt dafür, dass das Lesen der UART-Daten nicht permanent geschieht, sondern lediglich ein mal jede Sekunde. Eine höhere Abtast-Rate hätte zur Folge, dass der Task deutlich öfter aktiv sein muss. Dies ist allerdings nicht sinnvoll, da der Sensor ohnehin nur einmal jede Sekunde neue Daten übermittelt und durch die vermehrten Abfragen nur der Resourcenverbrauch des Tasks steigt, ohne zusätzliche Daten zu generieren.

5.1.7 Wifi Verbindung

```
1  Retcode_T wifiConnect(void)
2  {
3    WlanConnect_SSID_T connectSSID;
```

```
4   WlanConnect_PassPhrase_T connectPassPhrase;
5   Retcode_T ReturnValue = (Retcode_T) RETCODE_FAILURE;
6
7   ReturnValue = WlanConnect_Init();
8
9   if (RETCODE_OK != ReturnValue)
10  {
11    printf("Error occurred initializing WLAN \r\n ");
12    return ReturnValue;
13  }
14  printf("Connecting to %s \r\n ", WLAN_CONNECT_WPA_SSID);
15
16  connectSSID = (WlanConnect_SSID_T) WLAN_CONNECT_WPA_SSID;
17  connectPassPhrase = (WlanConnect_PassPhrase_T) WLAN_CONNECT_WPA_PASS;
18  ReturnValue = NetworkConfig_SetIpDhcp(NULL);
19  if (RETCODE_OK != ReturnValue)
20  {
21    printf("Error in setting IP to DHCP\n\r");
22    return ReturnValue;
23  }
24  ReturnValue = WlanConnect_WPA(connectSSID, connectPassPhrase,
        WlanEventCallback);
25  if (RETCODE_OK != ReturnValue)
26  {
27    printf("Error occurred while connecting Wlan %s \r\n ",
          WLAN_CONNECT_WPA_SSID);
28  }
29  return ReturnValue;
30 }
```

Listing 5.10: Wifi Verbindung

Listing 5.10 zeigt die Funktion, die zum Aufbau einer Wifi-Verbindung verwendet wird. Hierbei werden die Service Set Identifier (SSID) des Netzwerks und das Passwort aus der Header-Datei benutzt, um die Verbindung aufzubauen. Diese Art der Implementierung hat gleich mehrere Nachteile. Zum einen muss jedes Mal der Quellcode geändert werden, das Projekt neu gebuildet werden und die dabei erzeugten Binärdateien auf das XDK geflasht werden. Darüber hinaus ist es aus Sicht der Security nicht ideal, das Passwort im Klartext abspeichern zu müssen und dieses auch so über das Netzwerk zu versenden.
Trotz dieser beiden Aspekte wurde die Funktionalität auf diese Weise implementiert, da keine funktionierende Alternative dazu gefunden werden kann.

5.1.8 Scheduling

Da es sich bei dem Betriebssystem des XDK um ein Echtzeitbetriebssystem handelt und in dem entwickelten Programm mehrere Tasks parallel ablaufen sollen, muss zwangsläufig das Scheduling dieser Tasks betrachtet werden. Im entwickelten Programm werden spezifische Programmtechniken wie die Implementierung von Semaphoren zur Sicherstellung von bestimmten Reihenfolgen der Tasks komplett vernachlässigt. Dies geschieht begründet auf der Tatsache, dass die Tasks weitestgehend unabhängig voneinander laufen und nur an 2 Stellen eine Überschneidung bei der Nutzung gemeinsamer Variablen auftreten kann.

Im ersten Fall könnte es passieren, dass die Daten des Zwischenspeichers gesendet werden, bevor von jedem Sensor die neuesten Messwerte darin abgespeichert sind. Dies ist allerdings nicht als kritisch einzustufen, da die Änderungen der Messwerte innerhalb eines Sende-Zyklus meist nur minimal ausfallen.
Der zweite Fall würde eintreten, wenn der ADC-Task bereits ein weiteres Mal liest, bevor der GPIO-Task den Sensor weitergeschaltet hat. In diesem Fall würde einfach der selbe Wert zwei Mal gelesen und die Messwerte eines anderen Sensors würden nicht erfasst werden. Auch dieses Szenario ist als unkritisch anzusehen, da in diesem Fall für den Sensor, dessen Daten nicht erfasst werden einfach die letzten gemessenen Daten übertragen werden, welche keine allzu großen Abweichungen aufweisen sollten.

6 Die iOS App

Dieses Kapitel beschreibt die Struktur und den Aufbau der für die Ansteuerung der Drohne benötigten iOS App. Es wird ein überblick über die verwendete Architektur und Benutzeroberfläche, sowie über die verwendeten Klassen gegeben.

6.1 Aufbau

Die iOS-App ist mit Hilfe des MVC Pattern entwickelt. Dadurch gibt sich ein Aufbau aus verschiedenen Controllern und Views. In diesem Projekt gibt es für jede einzelne View einen eigenen Controller. Die Daten, wie die Flugroute sind als Models implementiert.
Die Views sind alle in einem Storyboard zu finden. Die Views werden im Kapitel GUI genauer beschrieben.
Die App stellt die Schnittstelle zum Anwender dar. Darunter sitzt das DJI Mobile SDK, welches Funktionen der Drohne implementiert. Das SDK stellt die äußere Schnittstelle zur Drohne dar.
Die Kommunikation zwischen mobilem Endgerät und Drohne wird über die **WLAN!**-Verbindung der Fernbedienung ermöglicht.

6.2 Graphical User Interface (GUI)

Die GUI besteht aus verschiedenen Views. Die folgende Abbildung zeigt den Aufbau der GUI in einem Storyboard.

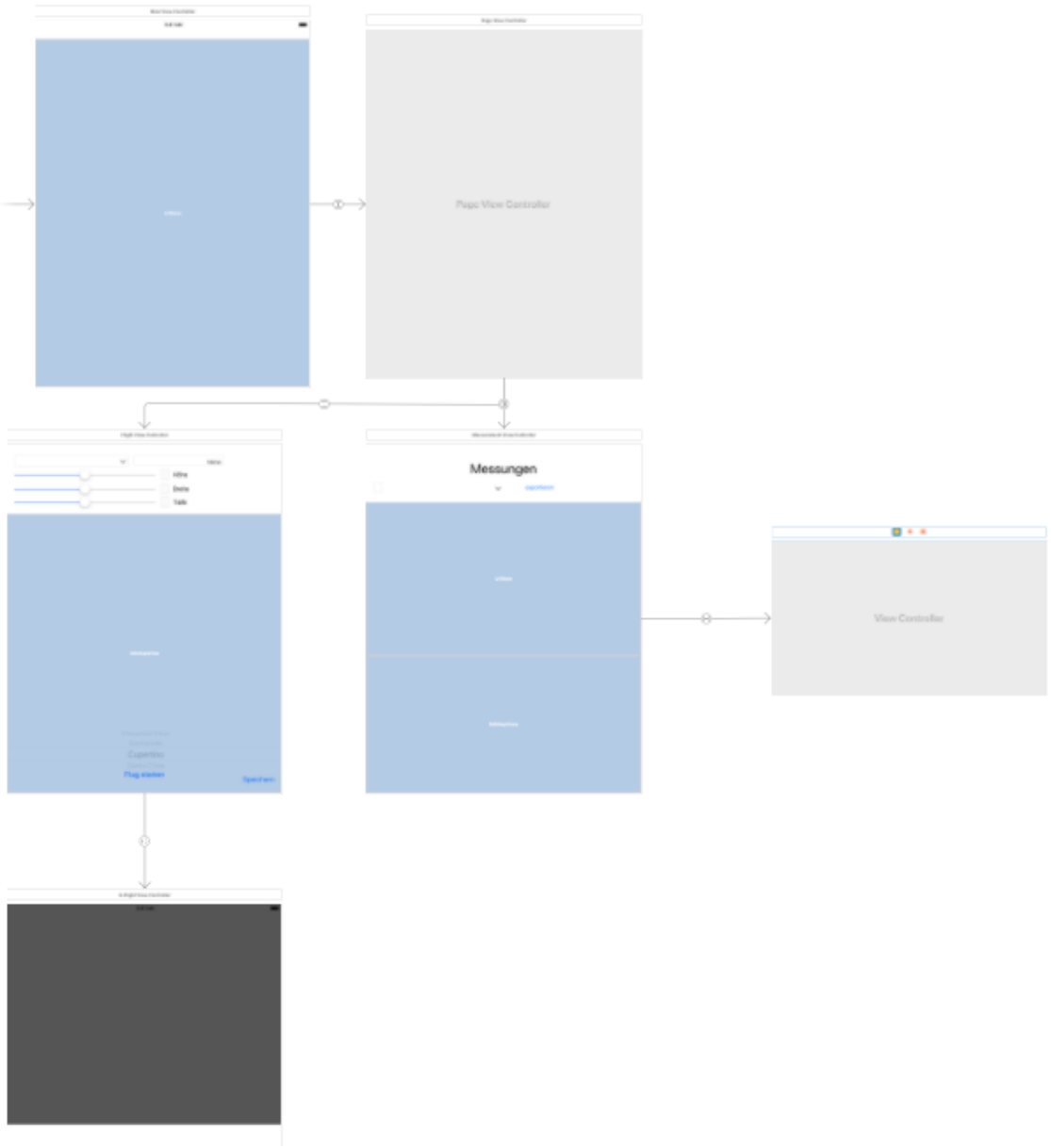

Abbildung 6.1: Aufbau GUI

6.2.1 MainView

Es gibt eine MainView, welche an der oberen Kante des Displays eine Statusleiste abbildet, die Informationen über den Standort und Zustand der Drohne visualisiert. Neben der Statusleiste beinhaltet die MainView noch eine ContainerView.

Statusleiste

Die Statusleiste bildet genauer folgende Informationen ab.

- Batteriestatus der Drohne
- Stärke des WiFi-Signals
- Qualität des GPS-Signals
- Die aktuelle Flughöhe der Drohne
- Flugstatus

Realisiert ist die Statusleiste über Elemente der DJI UXLibrary. Die einzelnen View-Elemente erben von den UXLibrary-Klassen. Die UXLibrary Klassen sind an dem Präfix *DUL* zu erkennen. So sind die Obengenannten Widgets durch die UXLibrary Klassen,

- DULBatteryWidget
- DULWifiSignalWidget
- DULGPSSignalWidget
- DULAltitudeWidget
- DULPreFlightStatusWidget

wie folgt realisiert:

```
1 @IBOutlet var batteryWidget: DULBatteryWidget!
2 @IBOutlet var wifiWidget: DULWifiSignalWidget!
3 @IBOutlet var GPSWidget: DULGPSSignalWidget!
4 @IBOutlet var altitudeWidget: DULAltitudeWidget!
5 @IBOutlet var flightStatusWidget: DULPreFlightStatusWidget!
```

Listing 6.1: Statusleiste

Die folgende Abbildung zeigt die Darstellung der Statusleiste in der App.

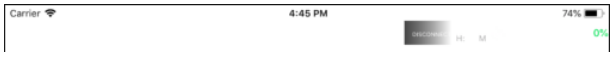

Abbildung 6.2: Statusbar

ContainerView

Die ContainerView nimmt, außer der Statusleiste am oberen Rand, den restlichen Platz des Displays ein. In die ContainerView werden die anderen Views geladen. Somit bekommt der Anwender in jeder Ansicht, über die Statusleiste, eine Übersicht über den Zustand der Drohne.

6.2.2 PageView

Um eine Trennung der Ansichten auf die Messungen und auf den Flugbetrieb zu bekommen, wird eine PageView implementiert. Diese PageView beinhaltet eine View für die Messungen und eine weitere View für den Flugbetrieb.
Um zwischen der *FlightView* und der *MessearumentView* zu wechseln ist eine einfache Wischgeste notwendig. Man kann die Views unendlich wechseln. Damit ist gemeint, dass man nicht nur einmal nach rechts wischen kann und dann wieder nach links wischen muss, um auf die andere View zu kommen. Egal welche Wischgeste, nach links oder nach rechts, man ausführt, man wechselt die View.

FlightView

Die *FlightView* beinhaltet eine Kartenansicht, sowie zwei Slidebars, um die Höhe und Tiefe der abzufliegenden Route auszuwählen. Es soll noch zusätzlich eine Combobox implementiert werden, welche es ermöglicht bereits abgespeicherte Routen auszuwählen. Um eine neue Route mit Namen hinzuzufügen ist ein Textfeld neben der Combobox zu finden. Im unteren Abschnitt befinden sich noch die Buttons *Flug starten* und *Speichern*. Der Button *Speichern* soll bewirken, dass die Flugroute welche man erstellt hat abgespeichert wird. Der Button *Flug starten* soll einen automatischen Drohnenflug starten. Es wird in ein Livefeed der Kamera der Drohne gewechselt.

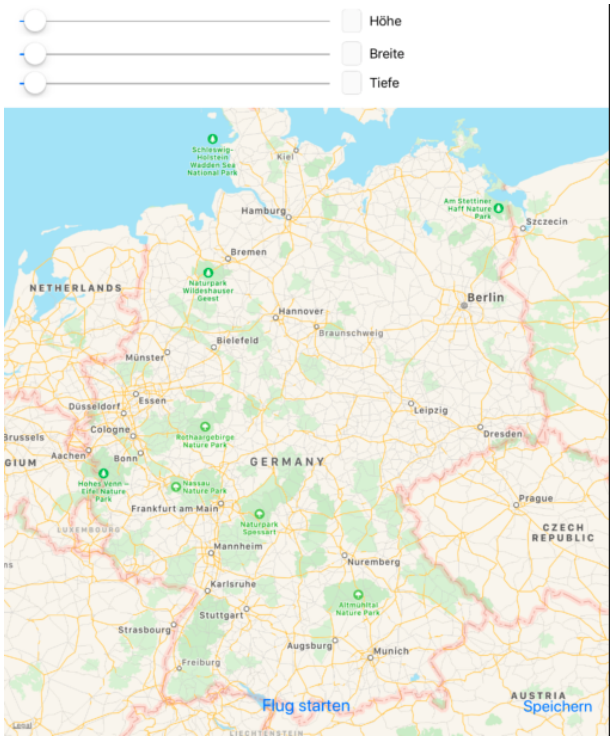

Abbildung 6.3: FlightView

MeassurementView

Die *MessearumentView* besteht aus einer Combobox über die man eine Messung auswählen können sollte. Daneben befindet sich ein Button exportieren, welcher dazu gedacht ist eine Messung als csv-File zu exportieren.
darunter existiert eine ContainerView welche Platz für eine Tabelle bietet, um die Messwerte als Tabelle anzuzeigen. Darunter existiert wiederum eine Kartenansicht um die Messung in einer Heatmap darzustellen.

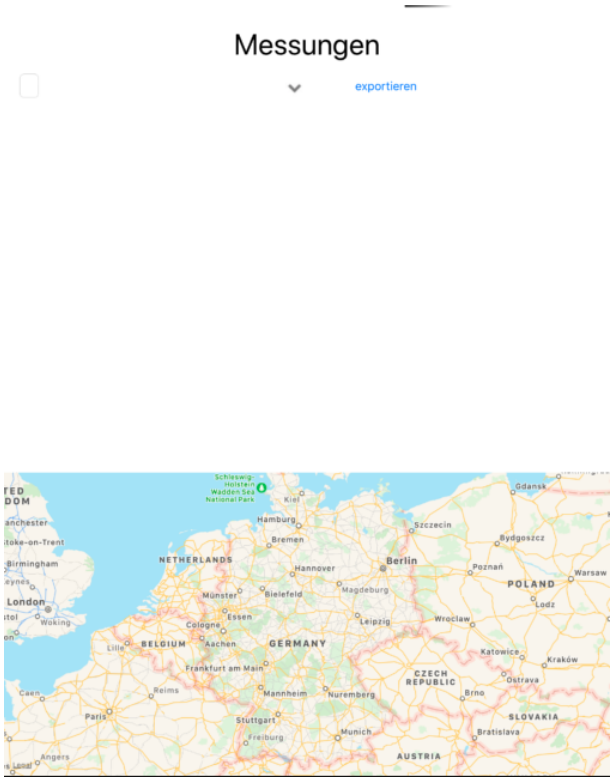

Abbildung 6.4: MessearumentView

6.3 Das autonome Fliegen

Der autonome Flug der Drohne ist über eine sogenannte Waypoint Mission realisiert. Diese beschreibt die Funktion des DJI Mobile SDK, dass die Drohne nacheinander vorgegebene Punkte anfliegt. Diese Mission muss auf die Drohne hochgeladen werden.

6.3.1 Hochladen der Waypoint Mission

Um eine Waypoint Mission zu starten, muss diese zuerst auf die Drohne hochgeladen werden. Dies geschieht über die *DJIMissionControl*. Die *DJIMissionControl* handhabt das ausführen von verschiedenen Missionen. Es gibt die Möglichkeit zweckbestimmte Mission Operators oder eine Serie von Missionen in einer Zeitachse auszuführen.
Die *DJIMissionControl* hat dafür die sogenannten Mission Operators. In dieser Arbeit ist nur der *DJIWaypointMissionOperator* wichtig. Dieser Mission Operator kann über Listener den aktuellen Stand der Mission abfragen. Wie in diesem Projekt das hochladen der Waypoint Mission auf die Drohne implementiert ist, zeigt folgendes Quellcode Beispiel.

```
func addWaypointMissionToTimeline(waypointMission:
    DJIMutableWaypointMission) -> Bool {

  var success = false

  let toScheduleWaypointMission = DJIWaypointMission(mission:
      waypointMission)

  let missionError = toScheduleWaypointMission.checkParameters()

  if missionError == nil {
    NSLog("Mission valide")
  }
  else{
    NSLog("Mission nicht valide: " + (missionError?.localizedDescription
        )!)
  }

  let errorAddingOperator = DJISDKManager.missionControl()?.
      waypointMissionOperator().load(toScheduleWaypointMission)
  if errorAddingOperator == nil {
    NSLog("Loaded Mission")
    success = true
  }
  else {
    NSLog("Error loading WPM: " + (errorAddingOperator?.
        localizedDescription)!)
  }

  if DJISDKManager.missionControl()?.waypointMissionOperator().
      currentState == DJIWaypointMissionState.readyToUpload
    && missionOperator.loadedMission?.checkParameters() == nil{
    DJISDKManager.missionControl()?.waypointMissionOperator().
        uploadMission(completion: { (errorUpload) in
      if errorUpload != nil {
        NSLog("Error Uploading Mission to Aircraft" + (errorUpload?.
            localizedDescription)!)
      }
      else {
        NSLog("Uploaded Mission to Aircraft")
        //self.addMissionOperatorUpdateListener()
      }
    })
  }
  return success
}
```

```
39
40 func addMissionOperatorUploadListener() {
41   missionOperator.addListener(toUploadEvent: self, with: DispatchQueue.
         main, andBlock: { (error) in
42     if error.error != nil {
43       NSLog("Fehler uploading: " + error.description)
44     }
45     else {
46       if self.missionOperator.currentState == .readyToExecute {
47         NSLog("Upload Done")
48         self.missionOperator.startMission(completion: { (errorExecuting)
               in
49           if errorExecuting != nil {
50             NSLog("Fehler bei Ausfuehrung: " + String(describing:
                   errorExecuting))
51         }
52         else {
53           NSLog("Kein Fehler bei Ausführung")
54         }
55
56         self.missionOperator.removeAllListeners()
57         })
58       }
59     }
60   })
61 }
```

Listing 6.2: Upload und ausführen der Waypoint Mission auf der Drohne

Wurde die Mission erfolgreich hochgeladen, kann diese über den Befehl *missionOperator.startMission()* ausgeführt werden. Im obigen Quellcode Beispiel wurde über den *DJIWaypointMissionOperator*, ein Listener hinzugefügt, welcher überprüft ob die Mission erfolgreich hochgeladen wurde. Ist dies der Fall wird die Mission ausgeführt.
Bevor der Listener hinzugefügt wird, wird die Mission der Zeitachse der *DJIMissionControl* hinzugefügt. War dies erfolgreich wird der Listener hinzugefügt.

6.3.2 Erstellen der Waypoint Mission

Der autonome Flug der Drohne ist in einer *DJIWaypointMission* implementiert. Diese fliegt die in Ihr enthaltene Waypoints an.
Bei betätigen des Buttons *Flug starten*, wird eine *DJIWaypointMission* angelegt. Diese beinhaltet noch keine konkreten Waypoints. In dieser Mission werden die Rahmenbedingungen für den gesamten Flug gesetzt. Die Mission ist vom Typ *DJIMutableWaypointMission*,

was bedeutet sie kann verändert werden.
Als für den gesamten Flug geltende Einstellungen werden folgende Parameter der Waypoint Mission gesetzt. Die Waypoint Mission wird mittels der Funktion *createMission()* erstellt, welche im Folgenden zu sehen ist.

```
1  private func createMission() -> DJIMutableWaypointMission? {
2    let mission = DJIMutableWaypointMission()
3
4    mission.autoFlightSpeed = Float(activeFlightRoute.velocity)
5    mission.finishedAction = .goHome
6    mission.headingMode = .auto
7    mission.flightPathMode = .normal
8    mission.exitMissionOnRCSignalLost = true
9    mission.gotoFirstWaypointMode = .safely
10   mission.repeatTimes = 1
11
12   return mission
13 }
```

Listing 6.3: Erstellen einer Waypoint Mission

Die Mission soll genau einmal ausgeführt werden und die Waypoints sollen direkt ohne Umweg angeflogen werden. Die Geschwindigkeit mit der sich die Drohne bewegt wird durch die Messhäufigkeit berechnet. Diese beträgt eine Messung pro Sekunde. Somit kann gewährleistet werden, dass ein regelmäßiges Messgitter entsteht und alle Messungen gleich weit von einander Entfernt durchgeführt wurden.
Die Geschwindigkeit berechnet sich aus dem kürzesten Abstand zweier Waypoints.
Zum Ende der Mission, soll die Drohne zurück zum Ausgangspunkt fliegen.
Ist die Waypoint Mission angelegt, werden der Mission die einzelnen Waypoints hinzugefügt.

6.3.3 Berechnen der Waypoints

Die Waypoints werden automatisch von der aktuellen Position der Drohne aus berechnet. Die maximale Anzahl der Waypoints ist auf 99 begrenzt.
Im Allgemeinen wird von einem Rechteck ausgegangen, welches sich vom Standort der Drohne nach Norden und Westen ausbreitet. Die maximale Fläche, welche abgeflogen werden kann ist auf 30m x 30m begrenzt und lässt sich über die Entsprechenden Slider in der GUI auswählen. Der Wert width entspricht der Richtung nach Norden, der Wert depth der Richtung nach Westen.
Aufgrund von Zeitproblemen bei der Implementierung werden immer 6 Luftschichten

im Bereich von 1,2 Metern bis 30 Metern abgeflogen. 30 Meter entspricht der maximal erlaubten Flughöhe für Drohnen bis 5 kg in Deutschland.
Durch das abfliegen der verschiedenen Luftschichten bleiben für jede Luftschicht 16 Waypoints übrig. Das heißt man kann pro Luftschicht 16 Waypoints zum anfliegen verwenden. Dadurch ergibt sich eine mindest Messhäufigkeit. Um ein gelichmäßiges Messgitter zu gewährleisten werden die Waypoints, wie folgt angeflogen.

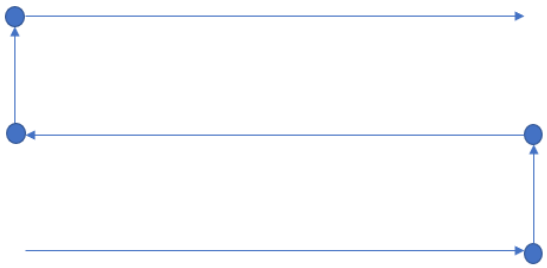

Abbildung 6.5: Anfliegen der Waypoints

Um die begrenzte Anzahl der Waypoints optimal zu nutzen wird zuerst überprüft, welche Strecke der zu messenden Fläche größer ist. Aus der kürzeren Strecke, geteilt durch acht, ergibt sich die Geschwindigkeit mit der sich die Drohne fortbewegt ($v = \frac{s}{t}$ mit s = kurze Distanz; t = eine Messung pro Sekunde). Die Geschwindigkeit legt auch den Mindestabstand der Messungen fest.
Die verschiedenen Waypoints werden vom Startpunkt ausgehend, welcher dem HomePoint der Drohne entspricht, berechnet. Die Waypoints werden über *CLLocationCoordinates2D* initialisiert. Deshalb muss die Entfernung von einer Entfernung in der Einheit Meter in eine Entfernung in Längengrad und Breitengrad umgerechnet werden.
Hierfür wird die Erde als Kugel angenähert. Dies ist im Fall der Anwendung der App legitim, da die tatsächliche Form der Erde bei Entfernungen von 30 Metern keine Auswirkungen hat. Für die Umrechnung einer Entfernung in Metern in Breitengrad, wird die Entfernung mit dem Faktor 1/111120.0 multipliziert. Für die Umrechnung einer Entfernung in Meter in Längengrad wird die Entfernung mit dem Faktor 1/25700000.0 multipliziert. Die entsprechenden Funktionen sehen wie folgt aus.

```
1  private func meter2degree_latitude (dist_meter: Double) -> Double {
2    var dist_lat: Double
3    let lat_const: Double = 111120.0 //Näherung
4    dist_lat = dist_meter * (1/lat_const)
5
6    return dist_lat
```

```
7   }
8
9   private func meter2degree_longitude (dist_meter: Double) -> Double {
10    var dist_long: Double
11    let long_const: Double = 25700000.0 //Näherung für Deutschland www.
          iaktueller.de
12    dist_long = dist_meter * (1/long_const)
13
14    return dist_long
15  }
```

Listing 6.4: Umrechnung Meter in Längengrad und Breitengrad

Über die erhaltenen Distanzen werden die einzelnen Positionen der Waypoints berechnet und einem Array hinzugefügt. Nach berechnen von 16 Waypoints wird die Höhe um 4,8 Meter erhöht und die Waypoints werden nach gleichem Vorgehen mit einer neuen Flughöhe dem Array hinzugefügt. Somit erhält man 6 verschiedene Waypoints an einer Position mit verschiedener Höhe.
Ein Waypoint wird mit den entsprechenden Koordinaten erstellt. Danach werden dem Waypoint Geschwindigkeit und Flughöhe hinzugefügt.

Literatur

Anhang

A. Anforderungsanalyse

Anforderungsanalyse

Air Quality Drone

Pflichtenheft Studienarbeit

an der Dualen Hochschule Baden-Württemberg Stuttgart

von

Julian Riegger, Sebastian Breit

17.11.2017

Bearbeitungszeitraum	04.11.2017 - 17.11.2017
Matrikelnummer, Kurs	1577610, 8870320, STG-TINF15ITA
Ausbildungsfirma	Robert Bosch GmbH, Stuttgart
Betreuer	Rene Lasse, Thilo Ackermann

Inhaltsverzeichnis

1 Zielbestimmung

1.1 Musskriterien

i. Der Drohnenflug muss mittels der App gestartet werden können

ii. Die Flugroute muss mittels der App festgelegt werden können

iii. Der Drohnenflug muss mittels der App abgebrochen werden können

iv. Die Messwerte müssen über die App einsehbar sein

v. Die Messwerte müssen über die App exportierbar sein

vi. Die App muss dem Benutzer ermöglichen, einen Flugbereich auf einer Karte zu markieren

vii. Die App muss dem Benutzer ermöglichen, die Flughöhe für die Messung auszuwählen

viii. Die auf der Karte ausgewählte Flugroute muss gestartet werden können

ix. Die App muss dem Benutzer ermöglichen, aus verschiedenen Messhäufigkeiten auszuwählen

x. Die Drohne muss Feinstaub messen können (2.5 & 10 μm)

xi. Die Drohne muss Druck messen können

xii. Die Drohne muss Feuchtigkeit messen können

xiii. Die Drohne muss Temperatur messen können

xiv. Die Drohne muss Stickoxide (NO_x) messen können

1.2 Sollkriterien

i. Die Messwerte sollen in der App visualisiert werden

ii. Die App soll die Messdaten in Abhängigkeit der Höhe zweidimensional auf der Karte anzeigen können

iii. Basierend auf dem auswählbaren Flugbereichs auf einer Karte, soll die App eine Flugroute automatisch berechnen können.

iv. Die App soll dem Benutzer ermöglichen, Bereiche explizit aus dem Flugbereich auszuschließen

v. Die App soll die Drohnenposition während des Fluges auf einer Karte anzeigen

vi. Die App soll eine Fortschrittsanzeige zur laufenden Messung bieten

vii. Die App soll einen permanenten Video-Stream während des Drohnenflugs anzeigen

viii. Die Drohne soll Kohlenstoffdioxid (CO_2) messen können

ix. Die Drohne soll Ozon (O_3) messen können

1.3 Kannkriterien

i. Der Benutzer kann in der App Flugrouten erstellen, speichern und abrufen

ii. Der Benutzer kann in der App Messprofile erstellen, speichern und abrufen

iii. Die App kann die Benutzerprofile exportieren können

iv. Die App kann durch Verrechnung von verbleibendem Akkustand und der vorgegebenen Flugroute einen Warnhinweis an den Benutzer geben, dass die Messung potenziell nicht ausgeführt werden kann

v. Die App kann die Messdaten dreidimensional darstellen

vi. Die Drohne kann Kohlenstoffmonoxid (CO) messen können

vii. Die Drohne kann Schwefeldioxid (SO_2) messen können

viii. Die Drohne kann flüchtige organische Verbindungen (VOC) messen können

ix. Die Drohne kann Methan (CH_4) messen können

x. Der Anwender kann die Messdaten mittels der App einem Server übermitteln können

1.4 Abgrenzungskriterien

i. Die App muss keinen Mehrbenutzerbetrieb mittels Login Daten ermöglichen

ii. Die App muss den Anwender nicht benachrichtigen, wenn der Nutzer in verbotenen Bereichen einen Drohnenflug durchführt.

iii. Die Drohne verfügt über keinerlei Kollisionserkennung, somit muss der Benutzer die sichere Benutzung selbst sicherstellen

2 Produkteinsatz

2.1 Anwendungsbereiche

Das Produkt ist für den Einsatz im Freien gedacht. Es darf nur in Gegenden genutzt werden, in denen es grundsätzlich erlaubt ist, mit Drohnen zu fliegen. Hauptanwendungsbereiche stellen Gebiete dar, in denen eine erhöhte Luftverschmutzung vermutet werden kann, um die dortige Luftqualität und mögliche Gesundheitsrisiken zuverlässig bestimmen zu können.

2.2 Zielgruppen

Das Produkt ist für Personen und Organisationen gedacht, welche einen Beitrag zu einem transparenteren Umgang mit dem Thema Luftqualität leisten wollen. Mögliche Interessengruppen wären hierbei beispielsweise staatliche Organisationen, die die Luftqualität in Städten überwachen wollen, Firmen, die die Luftverschmutzung in der Nähe ihrer Produktionsstätten messen wollen, sowie alle gemeinnützigen Organisationen und Privatpersonen, denen das Thema Luftqualität am Herzen liegt. Das Produkt ist nicht für Personen bestimmt, welche den Umgang mit Drohnen und mobilen Endgeräten nicht beherrschen oder der ihnen verboten ist.

2.3 Betriebsbedingungen

Das Produkt ist nur für den Einsatz mit ausreichendem Akkustand gedacht. Das Produkt ist nicht für den Einsatz unter extremen Wetterbedingungen gedacht. Das Produkt ist nicht für den Einsatz innerhalb von Gebäuden und geschlossenen Räumen gedacht.

3 Produktumgebung

3.1 Software

- iOS

3.2 Hardware

- Drohne (Phantom 3)
- iPhone/iPad (Gerät mit iOS)

4 Produktfunktionen

ID	**«F-010»**
Funktion	App starten
Akteur	Anwender
Beschreibung	Der Anwender muss die App auf seinem Endgerät (iPhone/iPad) starten können.

ID	**«F-100»**
Funktion	Flugroute anlegen
Akteur	Anwender
Beschreibung	Der Anwender muss in der Lage sein eine Flugroute in der App anzulegen. Durch setzen von Punkten auf einer Karte, kann die Flugroute gesetzt werden. Eine weitere Möglichkeit ist die Flugroute manuell abzufliegen, sodass diese gespeichert wird.

ID	**«F-110»**
Funktion	Flugroute bearbeiten
Akteur	Anwender
Beschreibung	Der Anwender muss in der Lage sein eine bestehende Flugroute zu bearbeiten.

ID	**«F-120»**
Funktion	Flugroute löschen
Akteur	Anwender
Beschreibung	Der Anwender muss in der Lage sein eine bestehende Flugroute zu löschen.

ID	**«F-130»**
Funktion	Flugroute auswählen
Akteur	Anwender
Beschreibung	Der Anwender muss in der App eine Flugroute auswählen können. Es kann eine Flugroute ausgewählt werden. Existiert noch keine Flugroute kann eine neue Flugroute erstellt werden (F-100).

ID	**«F-200»**
Funktion	Messprofil erstellen
Akteur	Anwender
Beschreibung	Der Anwender muss ein neues Messprofil erstellen können. Beim Erstellen muss die Messhäufigkeit und die zu messenden Daten ausgewählt werden.

ID	**«F-210»**
Funktion	Messprofil ändern
Akteur	Anwender
Beschreibung	Der Anwender muss ein von ihm erstelltes Messprofil ändern können.

ID	**«F-220»**
Funktion	Messprofil löschen
Akteur	Anwender
Beschreibung	Der Anwender muss ein von ihm erstelltes Messprofil löschen können.

ID	**«F-230»**
Funktion	Messprofil auswählen
Akteur	Anwender
Beschreibung	Der Anwender muss vor dem Starten einer Route ein Messprofil auswählen können.

ID	**«F-300»**
Funktion	Drohnenflug starten
Akteur	Anwender
Beschreibung	Der Anwender muss den ausgewählten Drohnenflug (F-130) starten können. Vor dem Start muss ein Messprofil ausgewählt werden (F-230).

ID	**«F-310»**
Funktion	Drohnenflug abbrechen
Akteur	Anwender
Beschreibung	Der Anwender muss einen Drohnenflug, den er gestartet hat (F-300) abbrechen können.

ID	**«F-400»**
Funktion	Messdaten als Tabelle anzeigen
Akteur	Anwender
Beschreibung	Der Anwender muss sich die Messdaten in einer Tabelle anzeigen lassen können.

ID	**«F-410»**
Funktion	Messdaten in Karte anzeigen
Akteur	Anwender
Beschreibung	Der Anwender muss sich die Messdaten in einer Karte anzeigen lassen können.

ID	**«F-420»**
Funktion	Messdaten exportieren
Akteur	Anwender
Beschreibung	Der Anwender muss die gemessenen Daten als csv-Datei exportieren können.

ID	**«F-430»**
Funktion	Messdaten an den Server übermitteln
Akteur	Anwender
Beschreibung	Der Anwender muss die gemessenen Daten als csv-Datei mittels der App an den Server übermitteln können.

5 Produktdaten

5.1 Non-persistente Daten

ID	**«D-001»**
Inhalt	Video-Stream
Bestandteile	• Video-Stream der Kamera an der Drohne

5.2 Persistente Daten

ID	**«D-010»**
Inhalt	Flugrouten-Koordinaten
Bestandteile	• Beschreibung der Flugroute • GPS-Koordinaten der abzufliegenden Punkte

ID	**«D-020»**
Inhalt	Messprofil
Bestandteile	• Beschreibung des Messprofils • Genauigkeit der Messung (in Messungen/Zeiteinheit) • Zu messende Werte (Feinstaub, NO_x, CO_2, ...)

ID	**«D-030»**
Inhalt	Messdaten
Bestandteile	• Flugkoordinaten • Zeitstempel • Temperatur • Feuchtigkeit • Druck • Feinstaub-Partikel-Konzentration • Stickoxid-Konzentration • Kohlenstoffdioxid-Konzentration • Ozon-Konzentration • Methan-Konzentration Jeweils ein Datensatz pro vorgegebener Zeiteinheit

6 Produktleistungen

ID	**«L-010»**
Leistung	Systemanforderungen
Beschreibung	Die App muss auf den folgenden Betriebssystemen lauffähig sein: • iOS 9 und neuer

ID	**«L-020»**
Leistung	GPS Standort ermitteln
Beschreibung	Die App muss über die Drohne den GPS Standort der Drohne ermitteln können.

ID	**«L-030»**
Leistung	NO_x Werte messen
Beschreibung	Das Produkt muss NO_x Werte in seiner Umgebung messen können.

ID	**«L-040»**
Leistung	Feinstaub Werte messen (2,5 μm & 10 μm)
Beschreibung	Das Produkt muss Feinstaub Werte in seiner Umgebung messen können.

ID	**«L-050»**
Leistung	O_3 Werte messen
Beschreibung	Das Produkt muss O_3 Werte in seiner Umgebung messen können.

ID	**«L-060»**
Leistung	CO Werte messen
Beschreibung	Das Produkt muss CO Werte in seiner Umgebung messen können.

ID	**«L-070»**
Leistung	SO_2 Werte messen
Beschreibung	Das Produkt muss SO_2 Werte in seiner Umgebung messen können.

ID	**«L-080»**
Leistung	CO_2 Werte messen
Beschreibung	Das Produkt muss CO_2 Werte in seiner Umgebung messen können.

ID	**«L-090»**
Leistung	CH_4 Werte messen
Beschreibung	Das Produkt muss CH_4 Werte in seiner Umgebung messen können.

ID	**«L-100»**
Leistung	VOCs Werte messen
Beschreibung	Das Produkt muss VOCs Werte in seiner Umgebung messen können.

ID	**«L-110»**
Leistung	Luftfeuchtigkeit messen
Beschreibung	Das Produkt muss die Luftfeuchtigkeit in seiner Umgebung messen können.

ID	**«L-120»**
Leistung	Temperatur messen
Beschreibung	Das Produkt muss die Temperatur in seiner Umgebung messen können.

ID	**«L-130»**
Leistung	Luftdruck messen
Beschreibung	Das Produkt muss den Luftdruck in seiner Umgebung messen können.

ID	**«L-140»**
Leistung	Zeit messen
Beschreibung	Das Produkt muss die Zeiten der Messungen messen können.

7 Qualitätsanforderungen

	Wichtig	Mittel	Niedrig	Nicht relevant
Robustheit	x			
Verfügbarkeit			x	
Kompatibilität				x
Benutzerfreundlichkeit		x		
Zeitverhalten			x	
Änderbarkeit				x
Portierbarkeit			x	

8 Testszenarien und Testfälle

ID	**«TC-010»**
Beschreibung	App starten
Vorbedingung	App ist auf dem Endgerät installiert
Testschritte	1. Der Anwender drückt auf das App Icon auf dem Endgerät
Zu erwartendes Ergebnis	App wird geöffnet

ID	**«TC-100»**
Beschreibung	Flugroute anlegen
Vorbedingung	-
Testschritte	1. Der Anwender wählt verschiedene aufeinanderfolgende Punkte auf einer Karte aus 2. Der Anwender speichert die Flugroute
Zu erwartendes Ergebnis	Die Flugroute wird angelegt und gespeichert

ID	**«TC-110»**
Beschreibung	Flugroute bearbeiten
Vorbedingung	Mindestens eine Flugroute existiert
Testschritte	1. Der Anwender wählt eine existierende Flugroute (TC-130) 2. Der Anwender ändert Punkte der Flugroute 3. Der Anwender speichert die Flugroute
Zu erwartendes Ergebnis	Die Flugroute wird geändert und gespeichert

ID	**«TC-120»**
Beschreibung	Flugroute löschen
Vorbedingung	Mindestens eine Flugroute existiert
Testschritte	1. Der Anwender wählt eine existierende Flugroute (TC-130) 2. Der Anwender löscht die Flugroute
Zu erwartendes Ergebnis	Die Flugroute wird gelöscht

ID	**«TC-130»**
Beschreibung	Flugroute auswählen
Vorbedingung	Mindestens eine Flugroute existiert
Testschritte	1. Der Anwender wählt eine existierende Flugroute
Zu erwartendes Ergebnis	Die Flugroute ist ausgewählt

ID	**«TC-200»**
Beschreibung	Messprofil erstellen
Vorbedingung	-
Testschritte	1. Der Anwender wählt eine Messhäufigkeit aus 2. Der Anwender wählt die zu messenden Daten aus 3. Der Anwender speichert das Messprofil
Zu erwartendes Ergebnis	Das Messprofil wird erstellt und gespeichert

ID	**«TC-210»**
Beschreibung	Messprofil bearbeiten
Vorbedingung	Mindestens ein Messprofil existiert
Testschritte	1. Der Anwender wählt ein existierendes Messprofil (TC-230) 2. Der Anwender ändert die Messhäufigkeit oder die zu messenden Daten 3. Der Anwender speichert das Messprofil
Zu erwartendes Ergebnis	Das Messprofil wird geändert und gespeichert

ID	**«TC-220»**
Beschreibung	Messprofil löschen
Vorbedingung	Mindestens ein Messprofil existiert
Testschritte	1. Der Anwender wählt ein existierendes Messprofil (TC-230) 2. Der Anwender löscht das Messprofil
Zu erwartendes Ergebnis	Das Messprofil wird gelöscht

ID	**«TC-230»**
Beschreibung	Messprofil auswählen
Vorbedingung	Mindestens ein Messprofil existiert
Testschritte	1. Der Anwender wählt ein existierendes Messprofil
Zu erwartendes Ergebnis	Das Messprofil ist ausgewählt

ID	**«TC-300»**
Beschreibung	Drohnenflug starten
Vorbedingung	-
Testschritte	1. Der Anwender wählt den Menüpunkt "Flug startenäus 2. Der Anwender wählt eine existierende Flugroute der Combobox "Flugroutenäus 3. Der Anwender wählt ein existierendes Messprofil aus der Combobox "Messprofileäus 4. Der Anwender startet den Flug durch drücken des Buttons „Flug starten“
Zu erwartendes Ergebnis	Die Drohne startet und fliegt die ausgewählte Flugroute ab. Messungen werden nach dem ausgewählten Messprofil durchgeführt.

ID	**«TC-310»**
Beschreibung	Drohnenflug abbrechen
Vorbedingung	Drohnenflug ist im Gange & Drohne in Reichweite der Steuerung
Testschritte	1. Der Anwender drückt auf den Button "Messung abbrechen" 2. Der Anwender bestätigt die auftretende Warnmeldung
Zu erwartendes Ergebnis	Der Drohnenflug wird abgebrochen und die Drohne kehrt zum Startpunkt zurück

ID	**«TC-400»**
Beschreibung	Messdaten als Tabelle anzeigen
Vorbedingung	-
Testschritte	1. Der Anwender wählt den Menüpunkt "Messdaten anzeigenäus 2. Der Anwender wählt den Unterpunkt "Tabelleäus 3. Der Anwender wählt einen Zeitraum aus, für den er alle Messwerte angezeigt bekommen möchte
Zu erwartendes Ergebnis	Alle Messdaten werden in einer Tabelle angezeigt

ID	**«TC-410»**
Beschreibung	Messdaten in Karte anzeigen
Vorbedingung	Es gibt existierende Messdaten
Testschritte	1. Der Anwender wählt den Menüpunkt "Messdaten anzeigenäus 2. Der Anwender wählt den Unterpunkt "Karteäus 3. Der Anwender wählt einen Zeitraum aus, für den er alle Messwerte angezeigt bekommen möchte
Zu erwartendes Ergebnis	Alle Messdaten werden in einer Karte angezeigt

ID	**«TC-420»**
Beschreibung	Messdaten exportieren
Vorbedingung	-
Testschritte	1. Der Anwender wählt den Menüpunkt "Messdaten exportierenäus 2. Der Anwender wählt den Speicherort aus
Zu erwartendes Ergebnis	Alle Messdaten werden als csv-Datei exportiert

ID	**«TC-430»**
Beschreibung	Messdaten an den Server übermitteln
Vorbedingung	-
Testschritte	1. Der Anwender wählt den Menüpunkt "Messdaten an Server übermittelnäus
Zu erwartendes Ergebnis	Alle Messdaten werden als csv-Datei an den Server übermittelt

9 Benutzeroberfläche

Im folgenden sind Mockups zu sehen, die die zu entwickelnde App darstellen. Folgende Bildschirme werden hier dargestellt:

- Menü
- Menü Pop-Up
- Flug starten
- Flugrouten
- Messprofile
- Messwerte anzeigen (Karte)
- Messwerte anzeigen (Tabelle)

Abbildung 9.1: Menü

Abbildung 9.2: Menü Pop-Up

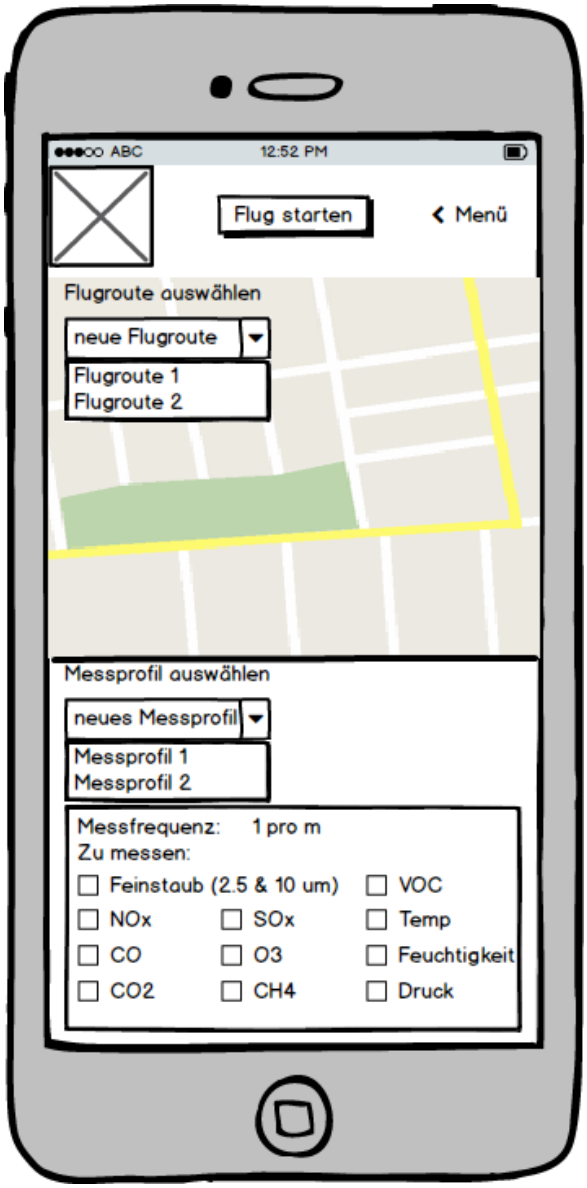

Abbildung 9.3: Flug starten

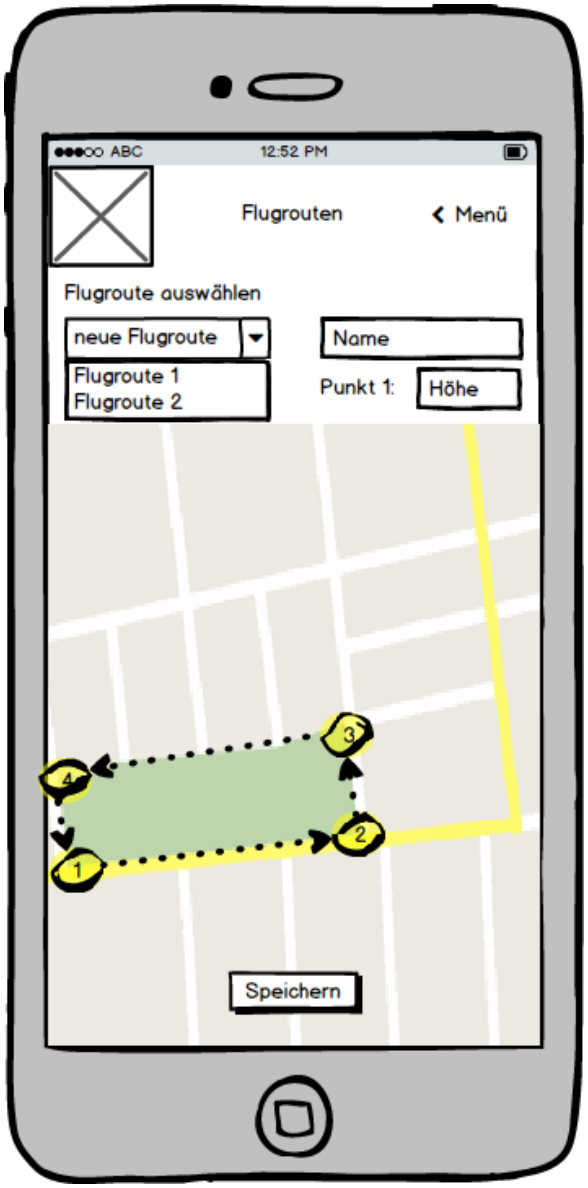

Abbildung 9.4: Flugrouten

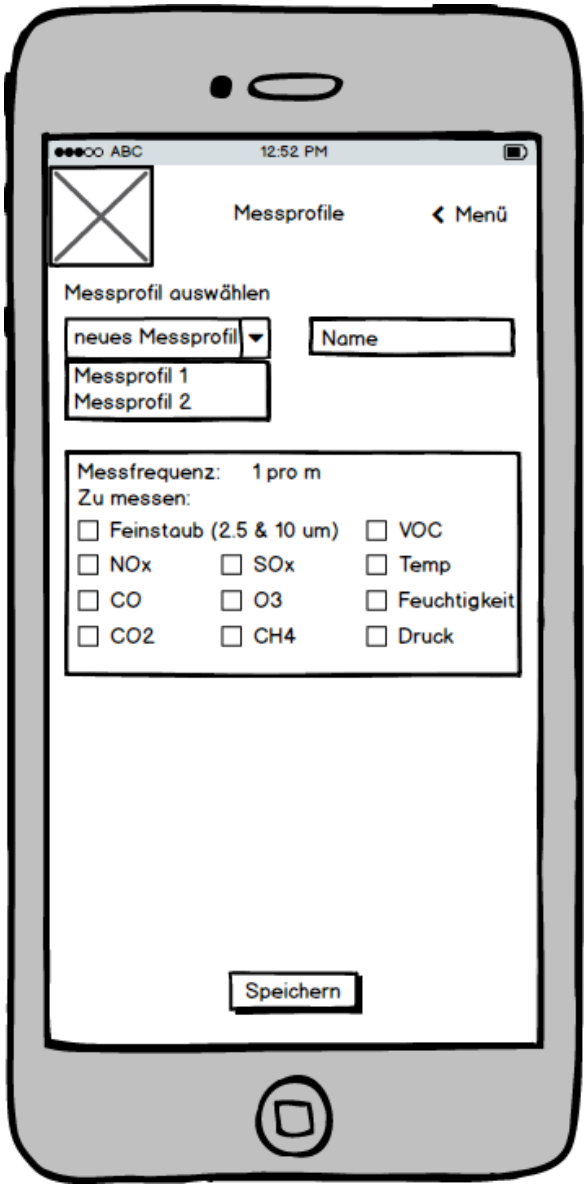

Abbildung 9.5: Messprofile

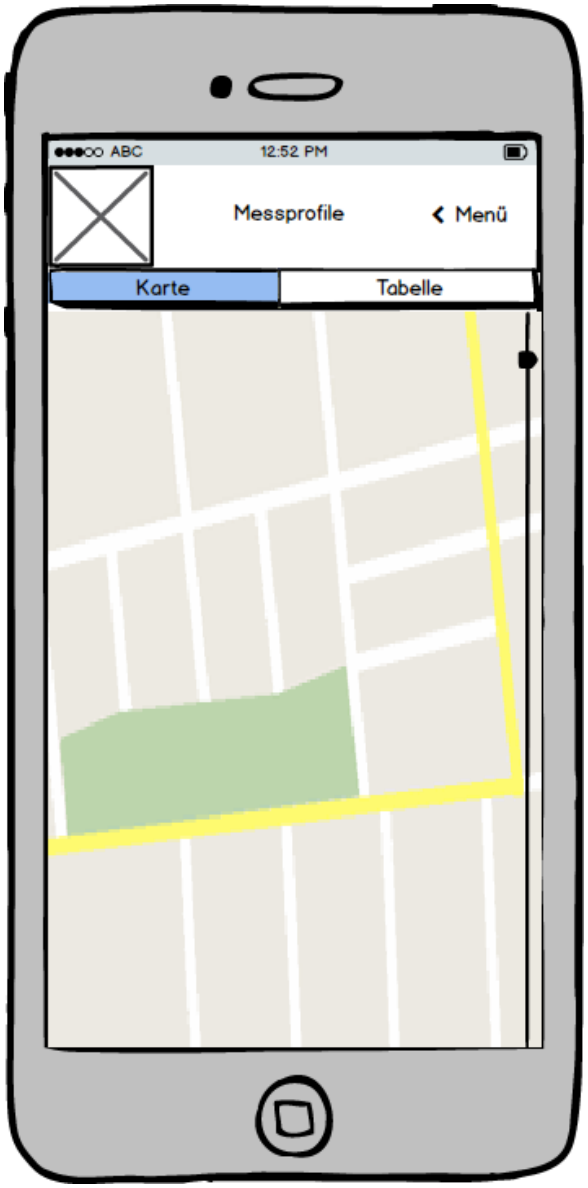

Abbildung 9.6: Messwerte anzeigen (Karte)

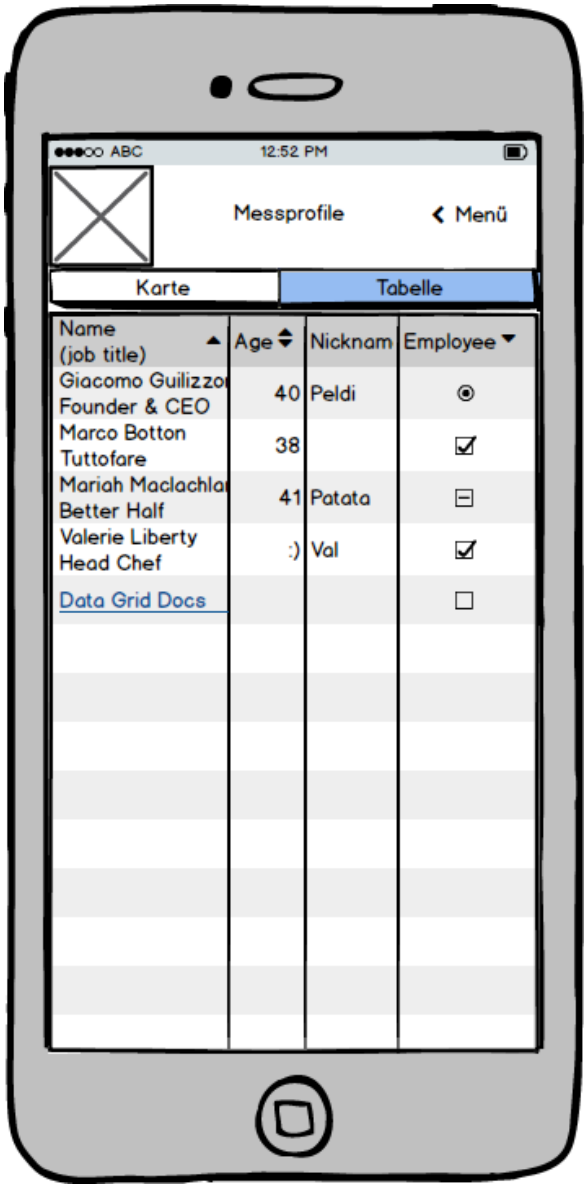

Abbildung 9.7: Messwerte anzeigen (Tabelle)

10 Entwicklungsumgebung

- Die App wird mit der Entwicklungsumgebung XCode entwickelt werden.
- Die Drohnenaspekte werden mit der Entwicklungsumgebung DJI Mobile SDK
- Die Anbindung der Sensoren erfolgt mithilfe des Bosch XDKs und der zugehörigen Entwicklungsumgebung

11 Abschlussbewertung