



OST Ostschweizer Fachhochschule

Design and Development of a Quadcopter Control System for Wind Data Acquisition and Analysis

Bachelor's Thesis by Israel Leuenberg, Pietro Lehmann, André
Schmid

OST Ostschweizer Fachhochschule,

Rapperswil, 2026-06-11

supervised by

Thomas Letsch

Prof. Dr. Martin Zimmermann

Prof. Dr. Christoph Gebhardt

Abstract

Wind and turbulence have a major impact on all types of operations in aviation. Smaller aircraft and especially paragliders are most susceptible to their influence. Therefore, having accurate wind information available is of vital importance. While airports and airfields usually provide detailed weather information, this is often not the case for paraglider launch sites, which could be anywhere from known launch sites to remote locations chosen spontaneously.

This bachelor's thesis aims to provide a portable, quadcopter based system that can be used at those locations to get an overview of the local wind and turbulence conditions. Part of this system are a small, custom built quadcopter and a mobile app, that allows the pilot to control the quadcopter directly and receive live local wind information.

The developed system lets the pilot define measurement locations and a route, which the quadcopter then follows autonomously and sends the estimated wind speed, wind direction and turbulence intensity directly to the app. The control software used for the quadcopter is ArduPilot and the communication to the app is established over a WiFi connection using MavLink.

Field tests in light to moderate wind conditions showed the average deviation in the estimated wind speed from the reference wind speed to be 0.307 m/s, which is a sufficiently accurate result for the intended use case. The calculated turbulence intensity followed the overall trend of the reference data. However, further testing is still required, because the turbulence encountered during testing was generally weak.

The overall system is very portable, as the quadcopter weighs 190 g (including batteries) and does not require a separate remote control. The expected maximum flight duration with one 4S battery with 850 mAh capacity lasts around seven minutes, which is deemed sufficient when used to sporadically check the wind conditions before a paragliding launch.

Improvements can mainly be made by enhancing the stability of the manual flight controls and further testing the system in more heterogeneous wind conditions. The mobile app could also be extended with additional functionality, such as allowing to save multiple routes or showing the wind barbs of the estimated wind conditions during flight.

In summary, the thesis successfully implements a functional system that can be used to supplement the available weather information at paragliding launch sites with localized wind and turbulence estimations.

Acknowledgements

We are grateful to everyone who supported us during the course of this thesis:

- **Laurin Hilfiker** and **Kevin Haizmann** for providing us with the Windshaper and the necessary infrastructure to perform the wind tunnel tests, as well as for their support during the testing process.
- **The Giger family** for letting us use their appartement as a base for our testing and development activities in Meiringen and their general support and hospitality.
- **Manuela Giger** and **Bérénice Bütschi** for their assistance with the field tests and their help in collecting reference data.
- **Andreas Bachmann**, **Natalie Breu** and **Sabrina Forster** for proofreading the thesis and providing valuable feedback on the writing.

We would like to thank these people for their support and contributions, which were invaluable to the successful completion of this thesis.

Contents

Glossary	1
Abbreviations	1
Concepts	1
1 Initial Situation and Problem Statement	2
1.1 Current Situation	2
1.2 Problems and Challenges	2
1.3 Type of Thesis	2
2 Task Description	3
2.1 Vision	3
2.2 Hardware Requirements	3
2.3 Methods and Expected Results	3
3 Framework Conditions	4
3.1 Purpose and Scope	4
3.2 Time, Budget and Resources	4
3.3 ECTS	4
4 Context and Domain Analysis	5
4.1 Paragliding	5
4.2 Wind	6
4.3 Turbulence	7
4.4 Regulatory Context	11
4.5 System Context	12
5 Market Analysis	13
5.1 Current Solutions	13
5.2 Competitive Analysis	13
6 Previous Studies	14
6.1 Related Research by Other Authors	14
6.2 Semester Thesis Autumn Term 2025	14
7 Requirements	15
7.1 Functional Requirements	15
7.2 Non-Functional Requirements	26
7.3 Mockup	28
7.4 MVP	30
8 Methods	31
8.1 Wind Speed and Direction Estimation	31
8.2 Turbulence Intensity Estimation	32
8.3 Reference Measurements	35
9 Design and Architecture	37
9.1 Introduction and Goals	37
9.2 Constraints	38
9.3 Context and Scope	38

9.4 Solution Strategy	39
9.5 Building Block View	40
9.6 Runtime View	42
9.7 Crosscutting Concepts	44
9.8 Architectural Decisions	47
9.9 Technical Debt	51
9.10 UX Design	52
10 Implementation	55
10.1 Quadcopter	55
10.2 Telemetry	60
10.3 Companion App	67
11 Results	83
11.1 Quadcopter System	83
11.2 Companion App	84
11.3 Wind and Turbulence Estimation	88
12 Conclusions and Outlook	95
12.1 Conclusions	95
12.2 Outlook	95
Bibliography	97
Appendix	100
A.1 Tools	100
A.2 Additional Artifacts	100
A.3 Companion App Class Diagram	101
A.4 Quadcopter Build Notes	105

List of Figures

Figure 1	Asymmetrical Canopy Collapse [1]	5
Figure 2	Frontal Canopy Collapse [1]	5
Figure 3	Wind Barbs [2]	6
Figure 4	Eddies of different Sizes in turbulent Airflow [3, Fig. 5.23]	7
Figure 5	Mechanical Turbulence caused by Buildings and Trees [4, Fig. 12-4]	7
Figure 6	Thermal Turbulence caused by Dry Convection Currents [4, Fig. 12-7]	8
Figure 7	Mountain Wave Turbulence [4, Fig. 12-15]	8
Figure 8	ICAO Turbulence Scale [5]	10
Figure 9	Domain Model	12
Figure 10	Manual Control	28
Figure 11	Automatic Control	28
Figure 12	Automatic Control - Fixed Altitude	28
Figure 13	Live Wind Measurements Overlay	29
Figure 14	Analysis Page with Wind Barbs	29
Figure 15	Detail Page for Measurement at Waypoint with Single Altitude	29
Figure 16	Detail Page for Measurement at Waypoint with Altitude Range	29
Figure 17	Example Fan Speed over Time using the Turbulence Function	35
Figure 18	Field Measurement in Meiringen	36
Figure 19	Known Turbulence Zone in Innertkirchen [6]	36
Figure 20	Business Context	38
Figure 21	Technical Context	38
Figure 22	Component Diagram	40
Figure 23	Database Schema	41
Figure 24	Runtime View - Programming Waypoints and Route	42
Figure 25	Runtime View - Uploading and starting the Mission	43
Figure 26	Logging Database Schema	44
Figure 27	State Machine Diagram	45
Figure 28	UX Layout	52
Figure 29	Header Bar	52
Figure 30	Header Bar Overlay	52

Figure 31	Tab Selection	53
Figure 32	Route Iconography	53
Figure 33	Wind Barb Iconography	53
Figure 34	Light and Dark Themes	54
Figure 35	The custom 3D-printed Quadcopter Frame	55
Figure 36	295 g – Improvised Prototype from the Semester Thesis	56
Figure 37	190 g – Quadcopter used in the Bachelor Thesis	56
Figure 38	ArduCopter’s Parameter Configuration Interface	57
Figure 39	Byte-Level Architecture of a MAVLink Packet	62
Figure 40	Telemetry Messages configured via MAV_CMD_SET_MESSAGE_INTERVAL and their respective Update Frequencies	65
Figure 41	Screenshot of the Emergency Buttons in the App Header	67
Figure 42	Manual Control Toggle Button	67
Figure 43	Screenshot of the Manual Control Interface in the App	67
Figure 44	Auto Flight Toggle Button	68
Figure 45	Screenshot of the Waypoint Creation Form in the App (Single Altitude on the left, Altitude Range on the right)	69
Figure 46	Button to Create a Waypoint at the Current Position of the Quadcopter	69
Figure 47	Screenshot of a Route	70
Figure 48	Illustration of the need for a Helper Waypoint to ensure cruising at the higher Altitude	71
Figure 49	Illustration of the need for Helper Waypoints for Altitude Range Waypoints ..	71
Figure 50	Mission Waypoints as displayed in Mission Planner after a successful Upload .	72
Figure 51	Diagram of the takeOff() Function’s Logic	73
Figure 52	Screenshot of the App Header showing Live Wind Data	76
Figure 53	Screenshot of the History Page	77
Figure 54	Screenshot of the Wind Barb Detail Card. Single Height Waypoint on the left, Altitude Range Waypoint on the right	78
Figure 55	Screenshots of the different Map Provider Selections in the App	79
Figure 56	Screenshot of the Settings Tab	81
Figure 57	The Quadcopter clipped onto a standard Hiking Backpack using the Battery Latch	83
Figure 58	Companion App Screenshots	84
Figure 59	Windshaper Testing 04.05.2026	88

Figure 60	Field Testing Meiringen 30.05.2026	90
Figure 61	Wind Speed Estimation Analysis Meiringen 30.05.2026 11:19 UTC	91
Figure 62	Correlation Reference Measurement and Quadcopter Wind Estimation Meiringen 30.05.2026 11:19 UTC	92
Figure 63	Turbulence Intensity Estimation Analysis Meiringen 30.05.2026 11:19 UTC	92
Figure 64	Turbulence Intensity Comparison Meiringen 30.05.2026 11:19 UTC	93
Figure 65	Flight Details Innertkirchen 31.05.2026 12:38	94
Figure 66	Companion App Class Diagram (Part 1 of 4)	101
Figure 67	Companion App Class Diagram (Part 2 of 4)	102
Figure 68	Companion App Class Diagram (Part 3 of 4)	103
Figure 69	Companion App Class Diagram (Part 4 of 4)	104
Figure 70	XIAO ESP32-C3 pinout: Physical Pins and their digital References	107
Figure 71	SpeedyBee F405 Mini Flight Controller used in this Project	107
Figure 72	ArduPilot Custom Firmware Builder (custom.ardupilot.org)	108

List of Tables

Table 1 MVP Features	30
Table 2 Quality Goals of the Architecture	37
Table 3 Solution Strategy	39
Table 4 Technical Debt	51
Table 5 Turbulence Calculation Parameters	75
Table 6 Windshaper Measurements 04.05.2026	89
Table 7 Field Measurements 30.05.2026 - 31.05.2026	91
Table 8 Tools used in the Project	100

Glossary

Abbreviations

AGL	Above Ground Level: The height of an object above the local terrain.
Beaufort	Beaufort Scale: A measure of wind speed, categorizing it into 13 levels.
EDR	Eddy Dissipation Rate: A measure of the rate at which turbulent energy is dissipated in the atmosphere.
EMA	Exponential Moving Average: Filter that gives more weight to recent data points.
ESC	Electronic speed controller: Electronic component that controls and regulates the speed of an electric motor.
FC	Flight controller: Control unit of a UAV.
GNSS	Global Navigation Satellite System: The general term for satellite navigation systems, including GPS, GLONASS, and Galileo.
IMU	Inertial Measurement Unit: A sensor package combining accelerometers and gyroscopes (and often a magnetometer) that measures a vehicle's linear acceleration and rotational rates, allowing the flight controller to track its orientation and motion.
LPF	Low Pass Filter: A filter that allows low-frequency signals to pass through while attenuating high-frequency signals.
MAVLink	Micro Air Vehicle Link: Lightweight binary protocol for drone communication.
MTOW	Maximum Takeoff Weight: The maximum weight including payload, fuel, and the vehicle itself that an aircraft is certified to take off with.
PID	Proportional-Integral-Derivative: A control algorithm used in flight controllers to maintain stable flight by adjusting control outputs based on the error between desired and actual states.
TI	Turbulence Intensity: A measure of the level of turbulence.
UART	Universal Asynchronous Receiver-Transmitter: A serial communication interface widely used for point-to-point data exchange between embedded devices such as flight controllers and peripheral modules.
UAV	Unmanned Aerial Vehicle: An aircraft that operates without a pilot on board and is controlled remotely or autonomously.

Concepts

Altitude	The height of an object above mean sea level.
Height	The height of an object above the local terrain (AGL).
Pitch	Rotation around the lateral (side-to-side) axis — tilts the nose up or down, controlling forward and backward movement.
Roll	Rotation around the longitudinal (front-to-back) axis — tilts the drone left or right, controlling lateral movement.
Yaw	Rotation around the vertical axis — turns the nose left or right, controlling the heading direction.
Waypoint	A predefined location defined by its coordinates, and possibly an altitude / height.

Chapter 1

Initial Situation and Problem Statement

Wind and turbulence are critical meteorological factors that affect every area of aviation. Bigger airports provide accurate real-time weather data, including wind and gust information. This project on the other hand aims to provide accurate wind data at remote locations, such as hangglider launch sites, where traditional weather stations are often unavailable. To acquire and analyze wind data, a portable quadcopter-based system is developed.

1.1 Current Situation

The semester thesis from the same project team in the autumn term 2025 laid the foundation for this project [7]. This bachelor's thesis builds upon that work and continues to combine the fields of wind measurement and UAV (Unmanned Aerial Vehicle) technology. Both fields are briefly introduced in the following sections.

1.1.1 Wind Measurements

Common wind measurement techniques include different kinds of anemometers and wind vanes [8]. Anemometers measure wind speed, while wind vanes determine wind direction. These methods are deployed at airports and weather stations but are often unavailable at smaller sites, such as hangglider launch sites [9], creating uncertainty for many recreational pilots. They also tend to be expensive and too bulky for portable use, making them unsuitable for use in remote locations.

1.1.2 UAV Technology

UAVs, particularly quadcopters, have become increasingly popular for various applications, including environmental monitoring and data collection [10]. They can be equipped with a variety of sensors to measure different parameters. Modern UAV control systems even integrate wind analysis [11] and allow real-time transmission of data to ground stations [12].

1.1.3 Similar Studies

Several studies have explored the use of UAVs for measuring wind conditions, as detailed in Chapter 6. However, none of these is catered specifically to be portable and usable by paragliding pilots.

1.2 Problems and Challenges

The initial hurdle of designing a quadcopter-based system that can estimate wind conditions and transmit data in real-time has been overcome in the semester thesis. However, there are still several challenges to be addressed in this thesis, including:

- Building an improved and more stable quadcopter based on the prototype developed in the semester thesis.
- Calibrating the algorithms for accurate wind data acquisition and analysis.
- Developing a mobile application to control the quadcopter and visualize the collected data in an intuitive way for pilots.

1.3 Type of Thesis

The focus of the thesis lies in the design and development of the quadcopter hardware and a corresponding control system. Part of the thesis is also the research and testing of algorithms for wind and turbulence estimation.

Chapter 2

Task Description

The overall goal of this thesis is to develop a quadcopter-based system for acquiring and analyzing wind data. An overview of the tasks can be found in this chapter.

2.1 Vision

As an extension of the semester thesis “Expansion of open-source drone software to include wind analysis” from the autumn term 2025 [7], this thesis aims to complete the quadcopter build and develop a mobile application for data visualization and quadcopter control. A prototype of the quadcopter has been built in the semester thesis, but lacked some crucial features such as the ability to fly. In this thesis, a final version of the quadcopter is to be built and it has to be capable of measuring wind and turbulence data.

Additionally, a mobile application (companion app) is developed to allow users to visualize the acquired wind data and control the quadcopter remotely. The complete system should be easy to use by inexperienced users and help to gather valuable wind data especially for paragliding pilots. The estimation algorithms were already researched in the semester thesis. However, testing, calibration and optimization of the algorithms are part of this thesis.

2.2 Hardware Requirements

The quadcopter is custom built using off-the-shelf components and open-source software. An important requirement for the quadcopter is a small and light form factor. The MTOW (Maximum Takeoff Weight) must stay below 250 g to ensure portability and a minimum of legal restrictions. The quadcopter is equipped with sensors to estimate wind speed, direction, and turbulence. This data is transmitted through WiFi in real-time to the mobile application.

2.3 Methods and Expected Results

The expected results of the thesis include:

- a completed quadcopter capable of flying and measuring wind data.
- real-time data transmission from the quadcopter to the mobile application.
- a mobile application for visualizing the acquired wind data and controlling the quadcopter remotely.

These results will be achieved by building upon the previous work and further developing the system in the following three areas:

1. **Construction of the quadcopter:** This includes the finalization of the prototype, as well as developing an optimized second version of the quadcopter.
2. **Development of the mobile application:** This includes designing and implementing a user-friendly interface for visualizing the wind data and controlling the quadcopter.
3. **Testing and optimization of the estimation algorithms:** This includes testing the system in real-world conditions as well as conducting wind tunnel tests to optimize the algorithms for accurate wind data acquisition.

Chapter 3

Framework Conditions

OST, the Eastern Switzerland University of Applied Sciences, provides the framework conditions for this bachelor's thesis as follows:

3.1 Purpose and Scope

This project is a bachelor's thesis, which is required for the completion of the bachelor's degree in Computer Science at OST. The project is to be carried out independently by the participants, under the guidance of a supervisor from OST.

3.2 Time, Budget and Resources

The total time budget for this project is 360 hours per participant. This results in a total of 1'080 hours for all three participants.

OST directly or indirectly provides the necessary resources, in particular access to a fixed workplace at the campus in Rapperswil, as well as access to the library and online resources. This also includes the financial resources for the hardware necessary to build the quadcopter.

Following are the dates given for the completion of this project:

- **16.02.2026:** Start of the project
- **13.04.2026:** Interim presentation
- **08.06.2026:** Submission of the brochure abstract
- **10.06.2026:** Release of the brochure abstract by the supervisor
- **12.06.2026:** Submission of the complete documentation
- **12.06.2026:** Exhibition
- **15.07.2026:** Presentation and Examination

3.3 ECTS

The project is worth 12 ECTS credits (30 hours per credit).

Chapter 4

Context and Domain Analysis

The thesis revolves around the contexts of paragliding, meteorology and quadcopter technology. This chapter provides an overview of these contexts and analyzes the domain of the thesis. In general, this chapter is a summary of the more detailed context and domain analysis that was conducted in the semester thesis [7], which can be consulted for more details.

4.1 Paragliding

Paragliding pilots use a fabric wing (canopy) to glide through the air, relying on natural lift sources such as thermals and ridge lift to stay airborne [13]. The sport is heavily influenced by weather conditions, especially wind and turbulence, which can lead to dangerous situations such as canopy collapses [14]. Therefore, paragliding pilots require accurate and up-to-date weather information to make informed decisions about their flight. However, many paragliding sites, especially when chosen spontaneously, lack nearby weather stations or up-to-date weather information.

4.1.1 Canopy Collapse

Strong wind conditions and turbulence can lead to canopy collapses. These are the most common cause of paragliding accidents, accounting for 36% of all paragliding incidents according to research by F. Soldati, C. Jackowski and A. Schön in 2022 [15]. There are essentially two types of canopy collapses: asymmetrical and frontal collapses.

Asymmetrical Collapse

In an asymmetrical collapse, one side of the canopy folds inwards. They can be caused by thermic conditions, downwind of obstructed airflow or when passing through wind shear layers. The collapsed wing creates drag which results in a turn and tilt towards the collapsed side. [1]



Figure 1: Asymmetrical Canopy Collapse [1]

Frontal Collapse

In a frontal collapse, the leading edge of the canopy folds back, which can also lead to a loss of lift and control. They happen when flying into strong downdrafts, for example when leaving thermals or flying into rotor turbulence. [1]



Figure 2: Frontal Canopy Collapse [1]

4.2 Wind

Wind is caused by masses of air moving from areas of high pressure to areas of low pressure. The topology of the terrain heavily influences wind patterns, especially in the Alps [16], where a lot of paragliding activity takes place.

4.2.1 Measurement of Wind

Traditionally, wind is measured using anemometers, which can be mechanical (e.g. cup anemometers) or ultrasonic (measuring wind through sound waves). These are typically part of weather stations and provide measurements of wind speed and direction at a fixed location [8].

The average windspeed is traditionally calculated over a period of 10 minutes. Gusts are defined as the highest windspeed measured over a 1 second interval within that 10 minute period [16].

Generally, the following two parameters are used to describe wind conditions [16]:

- **Wind Speed** (usually in m/s, km/h, knots or Beaufort)
- **Wind Direction** (usually in degrees from North)

To display the wind conditions, many different visualizations exist. In aviation, a common visualization is the use of wind barbs, which are graphical representations of wind speed and direction. Each barb consists of a line pointing in the direction the wind is coming from, with feathers or barbs indicating the wind speed [2].

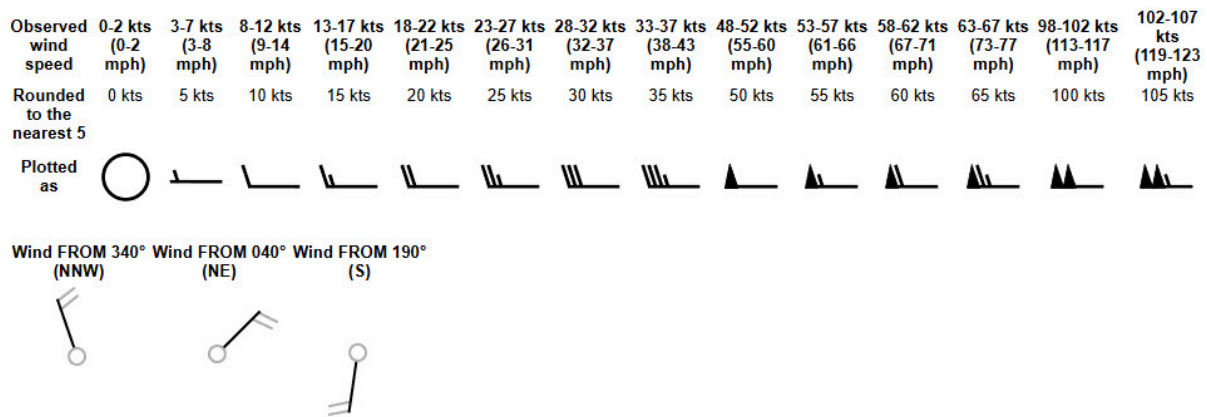


Figure 3: Wind Barbs [2]

These wind barbs are usually placed on a map to give an overview of the wind conditions in a certain area. However, it must be noted that wind barbs are only able to display the average wind conditions and do not provide any information about gusts or turbulence. To address this issue, the wind barbs in this thesis will be color coded to also show the level of turbulence, which, is not a standard practice.

As the main unit of measurement of speed in this thesis is m/s, it is helpful to know that 5 kts (the short feather) corresponds to roughly 2.5 m/s, and 10 kts (the long feather) corresponds to roughly 5 m/s.

4.3 Turbulence

Turbulence refers to irregular and chaotic fluctuations in wind speed and direction. There are different types of turbulence, which are detailed below. Turbulence can be considered as being made up of eddies. These are swirling motions of air that can be from a few centimeters to several kilometers in size.

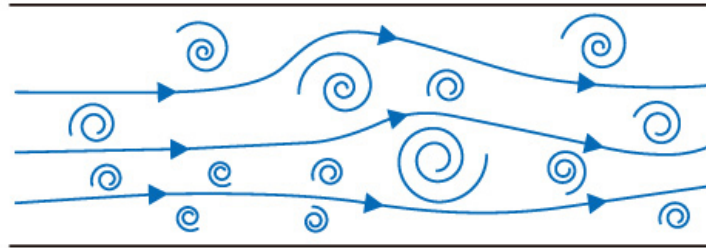


Figure 4: Eddies of different Sizes in turbulent Airflow [3, Fig. 5.23]

4.3.1 Causes and Types of Turbulence

As this thesis focuses on lower altitude measurements, only the types of turbulence that are relevant for this altitude range are discussed here. The descriptions and illustrations are based on the book “Aviation Weather” by Peter Lester [4].

Mechanical Turbulence

Mechanical turbulence refers to any disturbance in the airflow caused by physical obstructions, such as buildings, trees, or even mountains. When otherwise laminar wind flows over or around these obstructions, it can create eddies and vortices that result in turbulence. This type of turbulence is most commonly encountered at lower altitudes, especially during takeoff and landing phases of flight. It is particularly prevalent in urban or mountainous areas.

With higher windspeeds, the effects of mechanical turbulence can be felt in altitudes in excess of 3'000 feet (ca. 900 meters) AGL

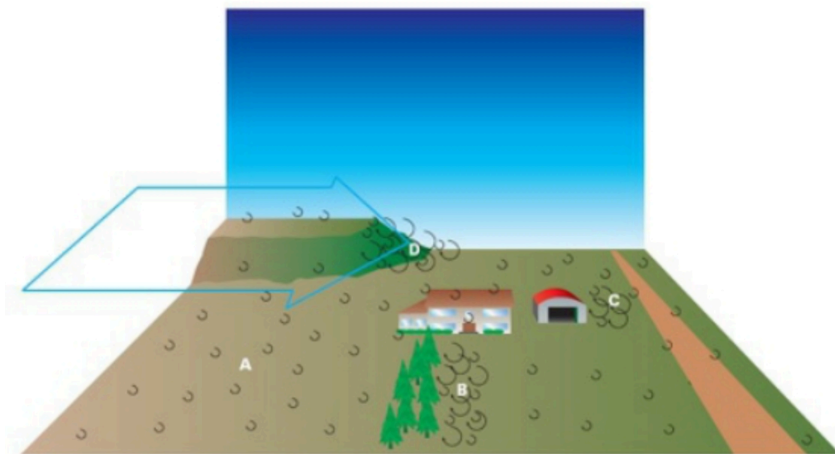


Figure 5: Mechanical Turbulence caused by Buildings and Trees [4, Fig. 12-4]

Mechanical turbulence is **highly relevant** for this thesis, as many paragliding sites are located in mountainous areas, where this is one of the main types of turbulence encountered due to trees, houses and the terrain itself.

Thermal Turbulence

Thermal turbulence occurs when the sun heats the ground, causing warm air to rise and cooler air to sink. These vertical air movements can lead to turbulence. Thermal turbulence is most commonly encountered during daytime, especially in sunny conditions over land. It is often more pronounced in the afternoon when the ground has had time to heat up.

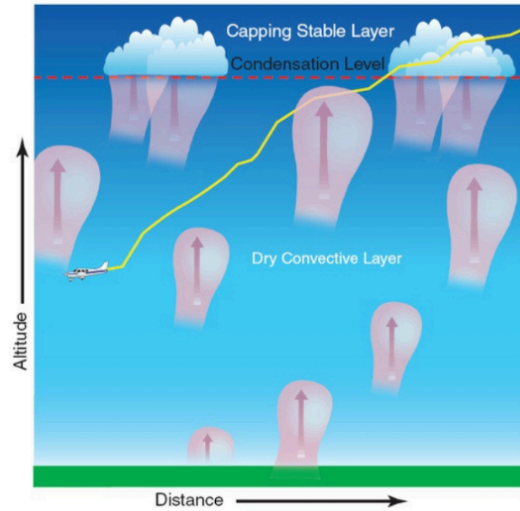


Figure 6: Thermal Turbulence caused by Dry Convection Currents [4, Fig. 12-7]

Thermal turbulence is also **highly relevant** for this thesis, as it is commonly encountered in the afternoons at many paragliding sites, especially in summer. Additionally, thermals often have a higher impact on flying than horizontal wind, as they can cause sudden and strong updrafts.

Mountain Wave Turbulence

Mountain wave turbulence occurs when stable air flows over a mountain range, creating a series of waves on the leeward (downwind) side of the mountains. These waves can extend for many miles downwind and can create rotors, or areas of turbulence.

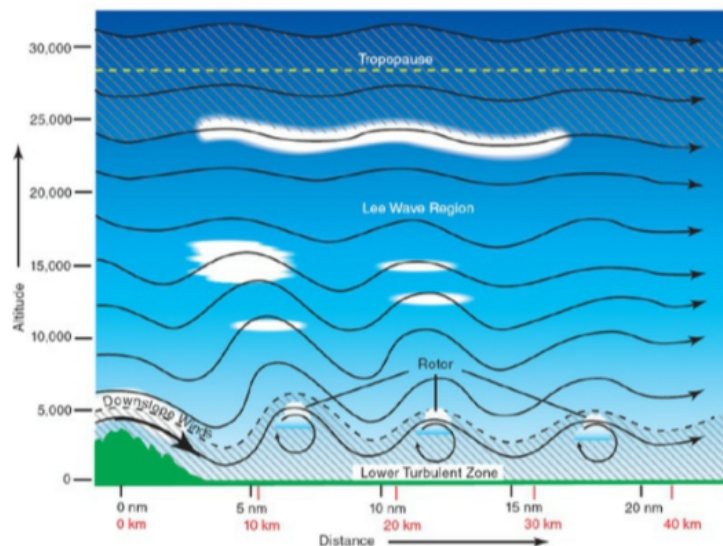


Figure 7: Mountain Wave Turbulence [4, Fig. 12-15]

Mountain wave turbulence is **highly relevant** for this thesis, as many paragliding sites or ad-hoc launch sites are located in mountainous areas. .

4.3.2 Turbulence Measurement

Since turbulence is difficult to measure directly, there is no single standard parameter to describe the level of turbulence. Different quantitative and qualitative measurements and classifications exist, which are based on different parameters, such as the eddy dissipation rate (EDR) [17], turbulence intensity (TI) [18], or the ICAO turbulence scale [5]. Relevant for this thesis are the turbulence intensity and the ICAO turbulence scale, which will be used to calculate and display the level of turbulence respectively.

Turbulence Intensity

Turbulence intensity is based on the standard deviation of the wind speed fluctuations in relation to the average wind speed, calculated over a certain time period. This time period by convention is 10 minutes and the sample rate at least 1 Hz [19]. However, since the measurement flights in this thesis will be much shorter than 10 minutes, the turbulence intensity will be calculated over a shorter time window.

The formula for calculating the turbulence intensity is as follows [19, Formula 2.23]:

$$TI = \frac{\sigma_u}{U}$$

Where TI is the turbulence intensity, σ_u is the standard deviation of the wind speed fluctuations, and U is the average wind speed.

The formula for calculating the standard deviation of the wind speed fluctuations is as follows [19, Formula 2.24]:

$$\sigma_u = \sqrt{\frac{1}{N-1} \sum_{k=1}^N (u_k - U)^2}$$

Where N is the number of samples, u_k is the wind speed at sample k , and U is the average wind speed. TI is calculated from a sample of wind measurements and not from the entire population of wind measurements during the flight, therefore the formula for the standard deviation uses $\frac{1}{N-1}$ instead of $\frac{1}{N}$ to account for the bias in the estimation of the population standard deviation from a sample.

ICAO Turbulence Scale

While TI is a useful unit of measure for research, it does not find widespread application in aviation. Instead, turbulence is often described qualitatively in a scale defined by ICAO (International Civil Aviation Organisation).

Intensity	Impacts	Accelerometer +/-
Weak	Noticeable Slight vibrations	0,05 à 0,2 g 
Moderate	Aircraft speed fluctuations between 15 and 25 kt Difficulty walking, irregular jolts	0,5 à 1,0 g 
Severe	Unsecured objects flying around Speed fluctuations >25 kt Short-term loss of aircraft control	>1,0 g 
Very severe	Aircraft damage possible	>1,5 g

Figure 8: ICAO Turbulence Scale [5]

As the perception of turbulence varies depending on the size and weight of the aircraft, the values of this scale are highly subjective. Nonetheless it gives pilots a good understanding of how severe the expected turbulence is, and is therefore commonly used.

To be useful in this thesis, the calculated TI is mapped to an ICAO turbulence intensity corresponding to the impact of turbulence on paragliding pilots.

4.4 Regulatory Context

In Switzerland, responsibility of regulations in civil aviation lies with the Bundesamt für Zivilluftfahrt (BAZL). It publishes regulations and guidelines for all categories of civil aircraft, including drones [20].

Additional guidelines are published by Skyguide, the Swiss air navigation service and air traffic control provider. Skyguide's responsibility is the management and control of all air traffic in Swiss airspace, including drones (under which category the developed quadcopter falls). Therefore, Skyguide provides further guidelines in terms of airspace usage and restrictions [21].

4.4.1 Category

The BAZL divides drones in two categories, depending on their weight and capabilities: "Offen" and "Speziell". The quadcopter in this thesis is lighter than 25 kg and therefore falls into the category "Offen". Additionally, classes and subclasses are defined to regulate the use of drones further [20]. Because the quadcopter in this thesis is privately built and below 250 g, it falls into the class **C0** and subclass **A1**. [22]

This classification results in the following requirements for the use of the quadcopter:

- Overflight of uninvolved people is allowed, but not over crowds of people [22]
- The quadcopter must be kept within visual line of sight (VLOS) at all times [22]
- The maximum flight altitude is 120 m AGL [22]
- Manned aircraft have the right of way [22]
- The quadcopter must not be flown in controlled airspace, such as near airports or in certain urban areas, without special permission from the BAZL [22]
- When emergency services are operating in the area, the quadcopter must not be flown [22]
- No remote-pilot certificate or registration is required [22]
- No Remote ID is required [22]

4.4.2 Airspace Restrictions

In addition to the general altitude restriction of 120 m AGL, there are certain zones where flying drones is prohibited or restricted. Which of these zones are active at any given time can be checked in current NOTAMS (Notice to Airmissions) and DABS (Daily Airspace Bulletin Switzerland) [23]. Additionally, the BAZL provides an online map with all active airspace restrictions for the operation of drones in Switzerland, which should be consulted before any flight: <https://map.geo.admin.ch>

4.4.3 Incidents and Insurance

In case of an incident involving the quadcopter, reports must be made to the BAZL and or SUST (Schweizerische Sicherheitsuntersuchungsstelle) [24].

The following types of incidents have to be reported [24]:

To BAZL	To SUST
• A person is seriously injured or killed • A manned aircraft is involved • A certified UAV is involved • The operating permit (Spezielle Kategorie) requires the operator to comply with a reporting obligation.	• A person is seriously injured or killed • A manned aircraft is involved • A certified UAV is involved

In terms of insurance, UAVs with an MTOW below 250 g are covered under the general liability insurance of the operator [25].

4.5 System Context

The following section deals with the technical context of the thesis, which includes the quadcopter, the companion app and the external wind measuring equipment used for calibration.

4.5.1 Users

The main user of the system is the paragliding pilot, who will use the companion app to receive real-time wind and turbulence information before their flight. Even though the measurements taken by the quadcopter are of interest to other target groups as well, the focus on paragliding pilots allows to tailor the capabilities and user experience of the system to the specific needs of this user group.

4.5.2 Domain

The domain contains the paragliding pilot, the quadcopter and the companion app. External wind measuring equipment is used for calibrating the measurements during the development process, but is not part of the final system and therefore not included in the domain model.

The domain model below shows the interactions between the different entities in the system. The **Paragliding Pilot** is the user of the system, who interacts with the system through the **Companion App**, which sends commands to the **Quadcopter**. The **Quadcopter** is the main component of the system, which estimates the wind conditions and sends this data to the **Companion App**, where the measurements are used to calculate the level of turbulence and show the combined information to the **Paragliding Pilot**.

Thanks to the onboard flight controller, the **Quadcopter** operates largely autonomously, which means that it can continue to operate safely even if the connection to the **Companion App** is interrupted.

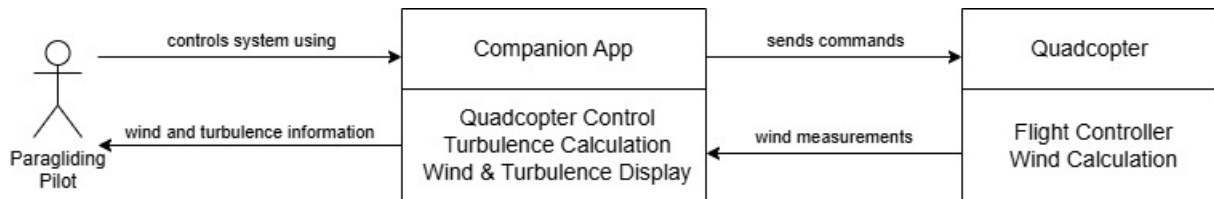


Figure 9: Domain Model

4.5.3 Quadcopter Software

The quadcopter is equipped with an onboard flight controller, whose software is part of the system domain and therefore is introduced here.

Ardupilot and MAVLink

The quadcopter is controlled using the open-source flight control software *ArduPilot*, which provides a wide range of features for controlling and stabilizing quadcopters and many other types of drones and RC vehicles. The software enables the quadcopter to fly autonomously, follow preprogrammed routes, and respond to various flight conditions.

Using the messaging protocol *MAVLink* it is possible to communicate with the quadcopter and receive telemetry data in real-time. This allows the quadcopter to be controlled remotely through a mobile application, and for the acquired wind data to be transmitted in real-time to the app for visualization and analysis. For a more detailed explanation of the MAVLink protocol, refer to Section 10.2.1.

Chapter 5

Market Analysis

This chapter analyses the current solutions available for measuring wind and turbulence conditions for paragliding pilots. This is a condensed version, more details can be found in the autumn term 2025 semester thesis [7].

5.1 Current Solutions

Current solutions to gather wind information include nearby weather stations, visual observation and experience, portable anemometers, and larger UAVs with advanced sensors. However, each of these solutions has limitations in terms of accuracy, portability, or cost.

While some paragliding sites have installed weather stations [26], they often provide infrequent updates, and nearby weather stations may not accurately reflect local conditions at the flying site. Visual observation of the terrain, vegetation and cloud formations and experience can be effective for experienced pilots [9] but are subjective and can lead to misjudgments. Portable anemometers only measure wind speed at ground level and also don't provide any information about wind direction or turbulence. Larger drones with advanced sensors, as discussed in Chapter 6, can provide high-quality data but are often expensive and are not portable enough for use in remote locations.

5.2 Competitive Analysis

There are currently no direct competitors specifically targeting paragliding pilots. The offered solutions fall short in terms of accuracy, portability or cost to meet the needs of this niche market. This project fills this gap by developing a system that is designed to be portable, accurate, and cost-effective for paragliding pilots.

Chapter 6

Previous Studies

Previous work in this field includes the semester thesis “Expansion of open-source drone software to include wind analysis” [7], as well as the work of other researchers who have explored the use of drones for wind speed estimation. Brief summaries of selected studies are provided in the following sections, while a more detailed analysis can be found in the semester thesis.

6.1 Related Research by Other Authors

Several researchers have previously explored the possibilities of wind measurement using drones. Most of them concentrated on the use of larger and heavier drones, which are not suitable for this project. Nevertheless, their results and methods provide valuable insights for our work.

6.1.1 High-resolution wind speed measurements with quadcopter uncrewed aerial systems: calibration and verification in a wind tunnel with an active grid [27]

A drone with a weight of 0.645 kg was used by Johannes Kistner et al. to measure wind speed and turbulence. They used the built-in weathervaning mode of their autopilot, which automatically turns the drone into the wind, to measure wind direction. Wind speed was estimated by observing the tilt angle of the drone during hovering. By measuring the fluctuations in windspeed, they calculated turbulence intensity (TI, see Section 4.3.2.1). The study was able to achieve accurate results, but the drone used is too heavy for our use case.

6.1.2 SAMURAI-S: Sonic Anemometer on a Multi-Rotor drone for Atmospheric turbulence Investigation in a Sling load configuration [28]

The study by Ghirardelli et al. fitted a 3D sonic anemometer to a drone with a weight of 4 kg. The main hurdle to be overcome was the placement of the anemometer to avoid erroneous measurements caused by the air movements generated by the rotors. They solved this by placing the anemometer as a sling load below the drone. While this is a functional concept, the use of an external anemometer is not suitable for this thesis.

6.1.3 A balloon-borne accelerometer technique for measuring atmospheric turbulence [29]

Graeme J. Marlton et al. used a weather balloon to measure atmospheric turbulence by suspending an accelerometer 4 m below the balloon. The turbulence intensity calculation was based on the accelerations encountered by the sensor. This proves the feasibility of using acceleration measurements to estimate turbulence, but they would have to be adapted for use on a quadcopter.

6.2 Semester Thesis Autumn Term 2025

The team’s previous semester thesis “Expansion of open-source drone software to include wind analysis” [7] focused on the research of the feasibility of using a lightweight quadcopter for wind and turbulence estimation. The thesis researched several methods for measuring wind and turbulence, coming to the conclusion that estimating wind and turbulence using a small quadcopter is possible, with the following methods being identified as the most promising:

- Estimating the level of turbulence by measuring the quadcopter’s acceleration measurements
- Estimating wind speed and direction by analyzing the attitude necessary to maintain a fixed position in the air

A prototype solution was developed, which still needs to be developed further into a fully functional system in this bachelor’s thesis.

Chapter 7

Requirements

The requirements of the system are based on the original task description, and the system context described in Chapter 4.

7.1 Functional Requirements

The functional requirements are divided into the quadcopter specific requirements (FR-Q) and the requirements for the companion app (FR-A). They are further grouped and sorted by type.

Prioritization: Priority 1 requirements are essential for the MVP defined in Section 7.4, while priority 2 requirements are additional features that improve the usability of the system. Priority 3 requirements are considered optional and are not expected to be implemented within the scope of this thesis, but are included for completeness and to provide a vision for future development of the system.

Actors: The only actor is the paragliding pilot who uses the system. For brevity, the paragliding pilot is hereafter referred to as “pilot”.

7.1.1 Quadcopter Requirements

FR-Q1	Wind Measurement
Type	Data Acquisition / Analysis
Priority	1
Workitem	Story 224: Wind Measurement
Description	As a pilot I want the quadcopter to estimate the wind speed and direction at its current location, so that I can analyze the wind conditions during my quadcopter flights.
Acceptance Criteria	• A real-time estimate of the wind speed and direction is calculated and sent to the companion app.

FR-Q2	Wireless Data Transmission
Type	Data Acquisition / Analysis
Priority	1
Workitem	Story 225: Wireless Data Transmission
Description	As a pilot I want the quadcopter to send the data in real-time to the companion app, so that I can see the estimations live.
Acceptance Criteria	• The quadcopter streams the data to the companion app in real-time. • The data can be received and processed by the companion app.

FR-Q3	Execute Control Commands
Type	Control
Priority	1
Workitem	Story 226: Execute Control Commands
Description	As a pilot I want the quadcopter to execute control commands from the companion app, so that I can control its movements and actions remotely and don't need a physical remote control.
Acceptance Criteria	• The quadcopter receives and executes control commands from the companion app in real-time. • The quadcopter follows the route programmed in the companion app. • No physical remote control is required to control the quadcopter.

7.1.2 Companion App Requirements

FR-A1	Quadcopter Manual Control
Type	Control
Priority	2
Workitem	Story 193: FRA-A1 Quadcopter Manual Control
Description	As a pilot I want to be able to control the quadcopter manually via the app, so that I can position it in the best possible way to get accurate estimations.
Acceptance Criteria	• The companion app provides a way to activate the manual control mode for the quadcopter. • The companion app provides a manual control interface for the quadcopter. • In manual control mode, if the quadcopter is on the ground and turned on, the pilot can arm the quadcopter by pressing a "Arm" button and take off by using the on screen controls. • During flight, the quadcopters position and heading is shown on a map. • During flight, the current wind conditions are shown on the screen. • Status information about the current wind, quadcopter speed, altitude, and battery remaining is shown on the top of the screen.

FR-A2 Quadcopter Automatic Control	
Type	Control
Priority	1
Workitem	Story 194: FRA-A2 Quadcopter Automatic Control
Description	As a pilot I want to enable the automatic flight of a preprogrammed route, so accurate and repeatable measurements are made automatically.
Acceptance Criteria	• In automatic flight, the quadcopter follows the preprogrammed single- or multi-waypoint route. • Automatic flight of the programmed route can be started by pressing a button. • During flight, the quadcopters position and heading is shown on a map. • Status information about the current wind, quadcopter speed, altitude, and battery remaining is shown on the top of the screen.

FR-A3 Start Mission	
Type	Control
Priority	1
Workitem	Story 195: FRA-A3 Start Mission
Description	As a pilot I want to start the automatic flight of a preprogrammed route, so accurate and repeatable measurements are made automatically.
Acceptance Criteria	• A mission can be started by pressing a “Start Mission” button, which initiates the automatic flight of the preprogrammed single- or multi-waypoint route. • When the button is pressed, the quadcopter arms and takes off if it is on the ground. • During flight, the “Start Mission” button is hidden.

FR-A4 End Mission	
Type	Control
Priority	1
Workitem	Story 196: FRA-A4 End Mission
Description	As a pilot I want to end the automatic flight of a preprogrammed route, so I can safely conclude the mission.
Acceptance Criteria	• During automatic flight, a “Return to Base” button is visible, which when pressed, initiates a return to base and landing procedure, where the quadcopter flies back to the takeoff location, lands, and disarms.

FR-A6 Switch between Manual and Automatic Control	
Type	Control
Priority	2
Workitem	Story 198: FRA-A6 Switch between Manual and Automatic Control
Description	As a pilot I want to be able to switch between manual and automatic control, so I can take over control from the autopilot in case of an emergency.
Acceptance Criteria	• The companion app provides a toggle to switch from automatic to manual control. • When switching from automatic to manual, the quadcopter holds position. • When switching from automatic to manual, the route still is displayed on the map. • Switching from manual to automatic control is only possible on the ground.

FR-A7 Adding a Waypoint	
Type	Flight Planning
Priority	1
Workitem	Story 199: FRA-A7 Adding a Waypoint
Description	As a pilot I want to program a waypoint for the quadcopter, so it can fly to a specific location and measure the wind there.
Acceptance Criteria	• The companion app provides ways to program a waypoint • When on the ground and quadcopter disarmed: ▸ Location can be selected by tapping on the map. ▸ Location can be adjusted by dragging the waypoint on the map. ▸ On the bottom of the screen a small form is shown where the altitude can be selected and the waypoint can be named. • The programmed waypoint is shown on the map. • When in flight, a waypoint can be stored by pressing a button, which saves the current location and altitude of the quadcopter as a waypoint. • When a waypoint isn't named by the pilot, it is automatically named with "WP" and a number corresponding to the order of programming (e.g. WP 1, WP 2, etc.).

FR-A8 Editing a Waypoint	
Type	Flight Planning
Priority	1
Workitem	Story 200: FRA-A8 Editing a Waypoint
Description	As a pilot I want to edit a programmed waypoint for the quadcopter, so it can fly to a specific location and measure the wind there.
Acceptance Criteria	• The companion app provides a way to edit a waypoint by tapping on the waypoint on the map • On the ground and quadcopter disarmed: ▸ Location can be changed by dragging the waypoint on the map. ▸ Altitude can be selected in a form at the bottom of the screen. ▸ The waypoint can be renamed in the form. • In flight: ▸ Waypoints can not be changed anymore.

FR-A9 Deleting a Waypoint	
Type	Flight Planning
Priority	1
Workitem	Story 201: FRA-A9 Deleting a Waypoint
Description	As a pilot I want to delete a programmed waypoint for the quadcopter, in order to declutter the map and remove waypoints that are no longer relevant.
Acceptance Criteria	• The companion app provides a way to delete a waypoint by tapping on the waypoint on the map when on the ground and selecting “Delete” in a form at the bottom of the screen. • When a waypoint is deleted, it is removed from the map and any programmed routes.

FR-A10 Planning Single Waypoint Route	
Type	Flight Planning
Priority	1
Workitem	Story 202: FRA-A10 Planning Single Waypoint Route
Description	As a pilot I want to select a programmed waypoint during flight planning, so that the quadcopter flies to that waypoint and measures the wind there.
Acceptance Criteria	• The companion app provides a way to select a programmed waypoint as the mission target during flight planning, which can be done by tapping on the waypoint on the map and selecting “Add To Route” in a form at the bottom of the screen.

FR-A11 Planning Multi Waypoint Route	
Type	Flight Planning
Priority	1
Workitem	Story 203: FRA-A11 Planning Multi Waypoint Route
Description	As a pilot I want to program multiple waypoints for the quadcopter, so it can follow a predefined route and measure the wind at each point.
Acceptance Criteria	• A route can be preprogrammed between multiple waypoints. • A line is shown between the programmed waypoints.

FR-A27 Planning Altitude Range	
Type	Flight Planning
Priority	1
Workitem	Story 227: FRA-A27 Planning Altitude Range
Description	As a pilot I want to program a range of altitudes with steps at a waypoint, so that the quadcopter measures the wind at different altitudes at the same location and I get a vertical overview of the wind profile.
Acceptance Criteria	• When planning a waypoint, the pilot can select: ▸ A minimum altitude for the waypoint, which is the lowest altitude at which the quadcopter will measure the wind at that waypoint. ▸ A maximum altitude for the waypoint, which is the highest altitude at which the quadcopter will measure the wind at that waypoint. ▸ An altitude step for the waypoint, which is the vertical distance between each measurement point at the waypoint. • The time, which the quadcopter will remain at each step can be configured.

FR-A12 Cruising Altitude between Waypoints	
Type	Flight Planning
Priority	1
Workitem	Story 204: FRA-A12 Cruising Altitude between Waypoints
Description	As a pilot I want the quadcopter to maintain a cruising altitude between waypoints, so that it avoids obstacles and ensures a safe flight.
Acceptance Criteria	• During flight in automatic mode, the quadcopter takes the higher altitude of two waypoints as the cruising altitude to avoid crashes into obstacles. • If the next waypoint is higher than the current altitude, the quadcopter climbs to the cruising altitude before flying towards the next waypoint. • If the next waypoint is lower than the current altitude, the quadcopter maintains the cruising altitude until it reaches the next waypoint, where it then descends to the waypoint's altitude.

FR-A13 Show Range of Quadcopter on Map	
Type	Flight Planning
Priority	2
Workitem	Story 206: FRA-A13 Show Range of Quadcopter on Map
Description	As a pilot I want the quadcopter to show the WiFi range on the map, so that I can plan my flights within the connectivity limits.
Acceptance Criteria	During flight in manual or automatic mode, the quadcopter displays its WiFi range as a circle on the map.

FR-A14 (optional) Show if Route is within Range	
Type	Flight Planning
Priority	3
Workitem	Story 207: FRA-A14 Show if Route is within Range
Description	As a pilot I want the quadcopter to show if the planned route is within its battery range, so that I can plan my flights within the endurance limits.
Acceptance Criteria	- The planned route is checked against the quadcopter's battery range to ensure it is within endurance limits. - If the planned route is out of range, a warning is displayed to the pilot.

FR-A15 Show Live Wind Estimations	
Type	Data Acquisition / Analysis
Priority	1
Workitem	Story 208: FRA-A15 Show Live Wind Estimations
Description	As a pilot I want to see live wind estimations from the quadcopter, so that I can observe the wind conditions at a certain location in real-time.
Acceptance Criteria	- The companion app displays live wind estimations from the quadcopter in real-time. - The wind estimations include speed and direction as well as turbulence classification, and are updated continuously while the quadcopter is in flight. - The live estimations are displayed on an overlay in the companion app, which can be accessed during flight by pressing the header bar. - The overlay can be shown both in manual and automatic flight mode. - In manual flight mode, the overlay does not cover the manual control interface. - In automatic mode, the overlay does not cover the "End Mission" button.

FR-A16 Save Wind Measurement and show Wind Barb on Map	
Type	Data Acquisition / Analysis
Priority	1
Workitem	Story 209: FRA-A16 Save Wind Measurement and show Wind Barb on Map
Description	As a pilot I want the measurements made during automatic flight mode to be saved and displayed on the map as wind barbs.
Acceptance Criteria	• During flight in automatic mode while measuring at a waypoint, the measurements for that location are saved. • While remaining at that waypoint, the current wind conditions are shown as a wind barb at the quadcopters position on the map. • When leaving the waypoint, the wind barb remains at the location of the waypoint and represents the average wind speed and direction estimation at that waypoint. • The color of the wind barb represents the turbulence level at that waypoint.

FR-A17 Show Wind Barbs on Map on History Page	
Type	Data Acquisition / Analysis
Priority	2
Workitem	Story 210: FRA-A17 Show Wind Barbs on Map on History Page
Description	As a pilot I want the quadcopter to show wind barbs on the map on the history page, so that I can analyze wind conditions during my flights.
Acceptance Criteria	• The map displays wind barbs representing the wind estimations taken by the quadcopter. • A slider at the bottom of the screen allows the pilot to select the data of a specific flight, which updates the wind barbs on the map to show the estimations taken during that flight. • The wind barbs are color-coded based on the level of turbulence.

FR-A18 Show Detail Page	
Type	Data Acquisition / Analysis
Priority	2
Workitem	Story 211: FRA-A18 Show Detail Page
Description	As a pilot I want to click on a wind barb on the history map and see a detail page for the selected wind barb, so that I can analyze wind conditions in more detail.
Acceptance Criteria	• The detail page displays a graph of the wind speed and direction over time at the selected location. • If the measurement was at a waypoint with an altitude range, the graph shows the measurements at the different altitudes. • The detail page shows the waypoint where the measurement was taken and the time of the measurement. • If multiple measurements were taken at that location, the detail page allows the pilot to select which measurement to view. • The detail page shows the raw data in a table. • The detail page provides a classification of the turbulence level. • The detail page allows data export in JSON format.

FR-A19 Data Export	
Type	Data Acquisition / Analysis
Priority	1
Workitem	Story 212: FRA-A19 Data Export
Description	As a pilot I want to be able to export the data from the detail page in JSON format, so that I can analyze it further or share it with others.
Acceptance Criteria	• The detail page provides an option to export the data in JSON format. • When the export option is selected, a JSON file containing the data is generated and downloaded to the user's device.

FR-A20 Emergency Return Button	
Type	Safety
Priority	1
Workitem	Story 214: FRA-A20 Emergency Return Button
Description	As a pilot I want to be able to press an emergency return button to safely return the quadcopter to its base in case of an emergency.
Acceptance Criteria	• During flight in manual or automatic mode, pressing the emergency return button initiates a safe return to base. • The quadcopter's built-in return to land function is activated. • After landing, the quadcopter disarms. • The emergency return function can be activated at any time during the flight.

FR-A21 Hold Position Button	
Type	Safety
Priority	1
Workitem	Story 215: FRA-A21 Hold Position Button
Description	As a pilot I want to be able to press a hold position button to keep the quadcopter at its current location in case of an emergency.
Acceptance Criteria	• During flight in manual or automatic mode, pressing the hold position button keeps the quadcopter at its current location. • The quadcopter maintains its altitude and position until either: ▸ The pilot makes inputs in manual mode. ▸ An emergency return is initiated. ▸ The battery charge is low, which initiates an automatic return to base. • The hold position function can be activated at any time during the flight.

FR-A22 Automatic Return to Base when Battery Charge Low	
Type	Safety
Priority	1
Workitem	Story 216: FRA-A22 Automatic Return to Base when Battery Charge Low
Description	As a pilot I want the quadcopter to automatically return to base when the battery charge is low, so that I don't lose the quadcopter due to an empty battery.
Acceptance Criteria	• During flight in manual or automatic mode, when the battery charge is low, the quadcopter initiates a safe return to base, following the same logic as the emergency return function in FR-A20.

FR-A23 Emergency Disarm Button	
Type	Safety
Priority	1
Workitem	Story 217: FRA-A23 Emergency Disarm Button
Description	As a pilot I want to be able to immediately disarm the quadcopter in case of an emergency, so that I can prevent accidents or damage.
Acceptance Criteria	• During flight in manual or automatic mode, pressing the emergency disarm button immediately stops all motors and disarms the quadcopter. • The disarm button can be activated at any time during the flight, and takes precedence over all other controls and functions.

FR-A24 Units Selection	
Type	Settings
Priority	2
Workitem	Story 219: FRA-A24 Units Selection
Description	As a pilot I want to be able to select the units of measurement for the quadcopter's settings, so that I can use the units I am most comfortable with.
Acceptance Criteria	• The pilot can choose between metric and imperial units for all relevant settings. • The selected units are consistently applied throughout the user interface and telemetry data. • Changing the units does not affect the underlying functionality or performance of the quadcopter.

FR-A25 Health Check	
Type	Settings
Priority	2
Workitem	Story 220: FRA-A25 Health Check
Description	As a pilot I want to be able to monitor the health status of the quadcopter and its sensors, so that I can ensure it is functioning correctly and safely.
Acceptance Criteria	• The pilot can view the health status of the quadcopter in real-time. • The health status includes information on battery level, motor performance, and sensor functionality. • (Optional) Alerts are generated for any critical issues that require immediate attention.

FR-A26 Log Display	
Type	Settings
Priority	1
Workitem	Story 220: FRA-A26 Log Display
Description	As a pilot I want to see the logs made by the quadcopter and the companion app, so that I can identify any issues.
Acceptance Criteria	• The pilot can view the logs generated following NFR4 in the settings tab of the companion app

FR-A28 (optional) Calibrate Sensors	
Type	Settings
Priority	3
Workitem	Story 221: FRA-A28 Calibrate Sensors
Description	As a pilot I want to be able to calibrate the quadcopter's sensors before flight, so that I can ensure accurate sensor readings and optimal performance.
Acceptance Criteria	• The pilot can initiate sensor calibration from the settings menu. • The calibration process provides feedback on the status and results of each sensor. • Calibration can be performed without affecting other settings or flight operations.

7.2 Non-Functional Requirements

The non-functional requirements are largely defined by the original task description.

NFR1 Quadcopter Size and Category	
Requirement	The Quadcopter shall have a maximum takeoff weight (MTOW) of less than 250g, so that it can be flown in category A1 in Switzerland [20].
Workitem	N/A
Verification	The weight of the Quadcopter is regularly measured during the development process to ensure it remains below 250g.
Time of Verification	During Development and Final Testing.
Acceptance Criteria	• The Quadcopter is lighter than 250g. • The Quadcopter can be legally flown in category A1 in Switzerland. • The Quadcopter is small and portable enough to be easily transported by the pilot.

NFR2 Open Source	
Requirement	The Quadcopter control system should be implemented by extending an existing open source solution.
Workitem	N/A
Verification	Checking the source code for used software and frameworks.
Time of Verification	During Development and Final Testing.
Acceptance Criteria	• All software and frameworks that are used are either open source or a custom implementation.

NFR3 Enterprise Architect	
Requirement	Enterprise Architect should be used to create a class-diagram of the implemented code.
Workitem	Story 222: FRA-A27 Enterprise Architect
Verification	The source code of the Quadcopter and the remote software is documented using Enterprise Architect.
Time of Verification	During Final Testing
Acceptance Criteria	• A class-diagram representing the source code has been generated.

NFR4 Logging	
Requirement	An intensive logging system with 2-dimensional runtime configuration is required: • Category (e.g. Package, Layer, etc.) • Log-Level In dense logging situations (e.g. loops) unnecessary processing of logs which are not being used must be prevented (e.g. if a log level is set to ignore certain messages, those messages should not be processed at all).
Workitem	Story 223: FRA-A28 Logging System
Verification	During testing of the software the logs are usable to improve debugging.
Time of Verification	During Development.
Acceptance Criteria	• The log-level can be selected at runtime. • The desired log categories can be selected at runtime. • There are no unnecessary processing of logs in dense code execution such as loops.

7.3 Mockup

The following mockup shows the initial draft of the user interface of the companion app. The mockup is based on the requirements defined in Section 7.1, and includes the main features and functionalities of the app. The design and naming scheme in the mockup are not final and are subject to change during the development process.

To create the mockup, Figma was used, which is a popular design tool for creating user interfaces and prototypes. It can be found under:

<https://www.figma.com/make/xkCEmF4XdJ2HAXRd3XYkyh/Mobile-App-with-Tab-Navigation?p=f&t=fnAzddT4q8ddhlaZ-0>

The mockup includes three main tabs: Flight, Analysis and Settings. The Flight and Analysis tabs are described in more detail below.

7.3.1 Flight Tab

The flight tab allows the pilot to plan and execute flights both manually and automatically, as well as view live wind and turbulence data during flight. Switching between manual and automatic control is possible using the toggle button on the top right of the screen.

The header bar shows live information about the current flight and contains the buttons to initiate an emergency return to base and to hold the quadcopter's position, which can be used in case of an emergency. These buttons are visible in every flight mode.

In automatic mode, waypoints can be defined on the map with either a fixed altitude or an altitude range, and the quadcopter can be sent directly to a waypoint using the "Direct To" function.

Note: The Direct To function is not available in the final product. Instead, waypoints can be added to a preprogrammed route, which the quadcopter then follows in automatic mode. This decision was made at a later stage in the project, which is why the Direct To function is still present in the mockup. The final design of the user interface is updated to reflect this change during the development process.

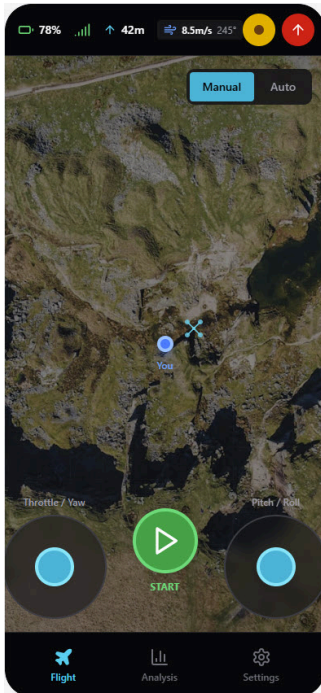


Figure 10: Manual Control

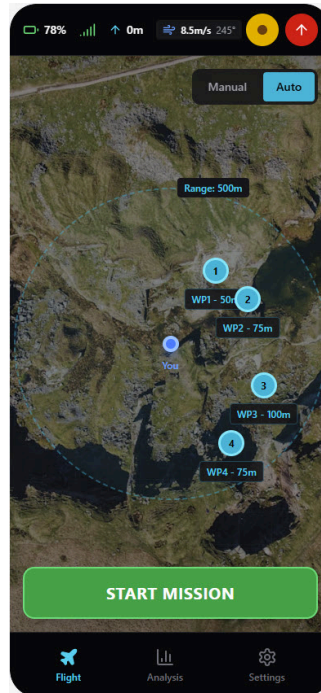


Figure 11: Automatic Control

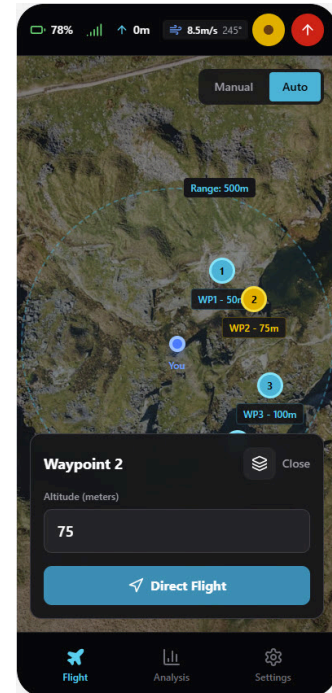


Figure 12: Automatic Control - Fixed Altitude

Tapping on the header bar in flight opens an overlay, that shows live wind estimations, including the current wind speed, direction, and turbulence classification.



Figure 13: Live Wind Measurements Overlay

7.3.2 Analysis Tab

The analysis tab allows the pilot to view the history of their flights. The map on the analysis page displays wind barbs representing the wind estimations taken by the quadcopter during each flight, which can be selected using a filter in the header bar. Clicking on a wind barb opens a detail page for that measurement, which has a different user interface, depending on whether the measurement was taken at a waypoint with an altitude range or at a single altitude.

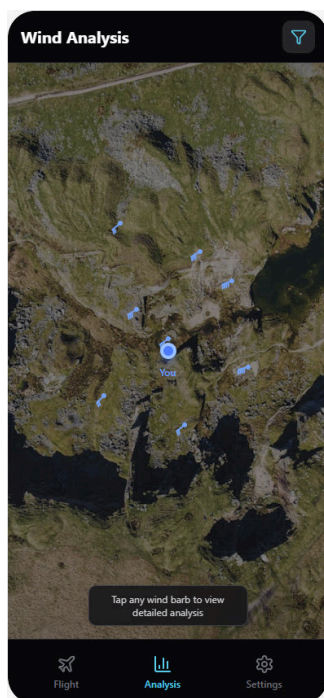


Figure 14: Analysis Page with Wind Barbs

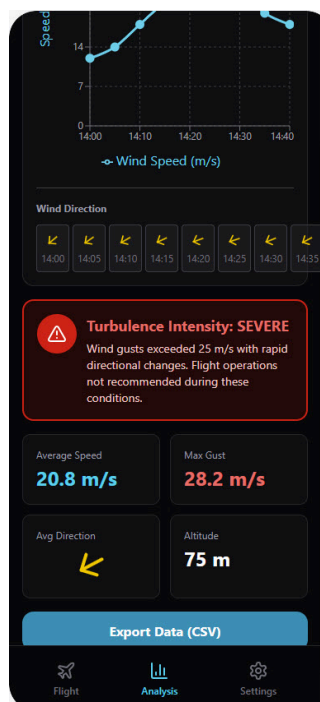


Figure 15: Detail Page for Measurement at Waypoint with Single Altitude

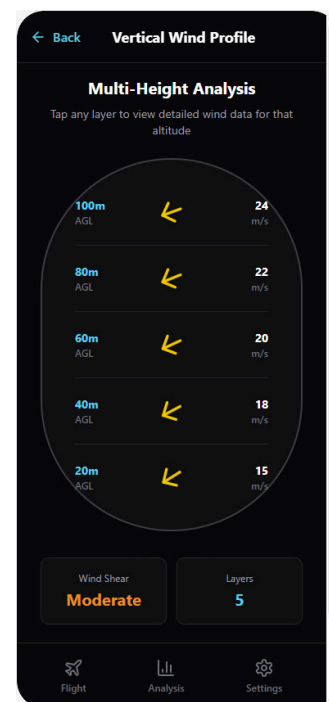


Figure 16: Detail Page for Measurement at Waypoint with Altitude Range

7.4 MVP

The minimum viable product (MVP) for this thesis includes the following features and functionalities:

Table 1: MVP Features

Feature	Functional Requirements	Non-Functional Requirements
A quadcopter fitting the requirements of category A1 in Switzerland, with a maximum takeoff weight of less than 250g is developed	-	NFR1
The quadcopter control system is implemented by using existing open source software .	-	NFR2
Waypoints can be defined on the map with a single altitude or an altitude range .	FR-A2, FR-A7, FR-A8, FR-A9, FR-A27	-
The quadcopter can be controlled in automatic mode and fly a preprogrammed route .	FR-Q3, FR-A2, FR-A3, FR-A4, FR-A10, FR-A11, FR-A12	-
The three emergency buttons (emergency return , hold position , and emergency disarm) are functional and can be activated at any time during the flight.	FR-A20, FR-A21, FR-A22, FR-A23	-
The live measurements overlay is functional and shows the current wind speed, direction, and turbulence classification during flight.	FR-Q1, FR-Q2, FR-A15, FR-A16	-
The measured data can be exported in JSON format.	FR-A19	-
The settings tab allows access to the logs .	FR-A26	NFR4
The app implementation is documented using Enterprise Architect , including a class-diagram representing the source code.	-	NFR3

These features represent the minimum set of functionalities that are required to test the core concept of the project. Additional features and improvements can be added after the MVP has been successfully implemented and tested.

Chapter 8

Methods

As several methods of estimating wind and turbulence were identified as promising in the semester thesis (see Section 6.2), the accepted methods are further elaborated in this chapter. These methods rely on the quadcopter's built in sensors and the wind estimation function of ArduPilot [30].

8.1 Wind Speed and Direction Estimation

The quadcopter is equipped with a GNSS receiver, which allows it to determine its position and velocity. This allows the flight controller to maintain a fixed position in the air, even when affected by wind, by adjusting the quadcopter's attitude, tilting in the direction of the wind. The necessary corrections are made automatically by ArduPilot [31]. By analyzing the attitude necessary to maintain a fixed position in the air, and the unintentional acceleration and GPS movement of the quadcopter, it is possible to calculate the wind speed and direction.

This calculation is inbuilt in ArduPilot [32], and the results are transmitted to the companion app in real-time. In order to use the built in wind estimation function of ArduPilot, several parameters need to be calculated. These parameters are the body drag coefficients and the propeller drag coefficient.

8.1.1 Body Drag Coefficients

The body drag coefficients are dependent on the area of the quadcopter's body and its weight. ArduPilot's wind estimation function calls for the body drag coefficients to be calculated for the front (EK3_DRAG_BCOEF_X) and the side (EK3_DRAG_BCOEF_Y) of the quadcopter using the following formulae [32]:

$$\text{EK3_DRAG_BCOEF_X} = \frac{m}{A_f}$$

$$\text{EK3_DRAG_BCOEF_Y} = \frac{m}{A_s}$$

Where m is the mass of the quadcopter in kg, A_f is the frontal area of the quadcopter in m^2 , and A_s is the side area of the quadcopter in m^2 . The frontal and side areas are calculated based on a two-dimensional projection of the quadcopter's body.

The final quadcopter of this project has a frontal area (A_f) of 0.002184 m^2 and a side area (A_s) of 0.002815 m^2 . With a mass (m) of 0.1960 kg , the resulting body drag coefficients are:

$$\text{EK3_DRAG_BCOEF_X} = 89.744 \text{ kg/m}^2$$

$$\text{EK3_DRAG_BCOEF_Y} = 69.627 \text{ kg/m}^2$$

8.1.2 Propeller Drag Coefficient

The propeller drag coefficient makes up the largest part of the total drag of the quadcopter and stems from the momentum drag created by the propellers [30, 6:16]. It can only be calculated experimentally, either by measuring the acceleration and deceleration of the quadcopter when facing into the wind as described by Dr. Paul Riseborough [30], or by comparing the output of ArduPilots calculation to a reference measurement and adjust the parameter accordingly.

In this project, the latter method is used to finetune the propeller drag coefficient, using a Windshaper and field measurements as reference (see Section 8.3). The ideal propeller drag coefficient, giving the most accurate wind estimation results, is found to be:

$$\text{EK3_DRAG_MCOEF} = 0.30 \text{ 1/s}$$

The corresponding measurement flights are documented in Table 7.

8.2 Turbulence Intensity Estimation

Two methods of estimating turbulence were found to be promising in the semester thesis. Both of these methods are combined in this project to provide a more comprehensive estimation of turbulence intensity and are enhanced by a third method using the variance in wind direction.

8.2.1 Turbulence Intensity from Wind Speed Variance

The established method of calculating turbulence intensity is to use the variance of the wind speed, as described in Section 4.3.2. By calculating the variance of the wind speed estimations obtained from the quadcopter's built-in wind estimation function [30], it is possible to estimate the turbulence intensity. However, this method depends on the accuracy of the wind speed estimation and is influenced by the time it takes the quadcopter to adjust to changes in wind speed, which can lead to an underestimation of turbulence intensity. Therefore, this method is only used in combination with additional data to enhance the overall calculation of the level of turbulence.

8.2.2 Turbulence Estimation using Wind Direction Variance

To enhance the estimation of turbulence intensity, the variance of the wind direction is also taken into account, which is also an indicator of turbulence [33]. Due to the circular nature of wind direction and therewith the jump from 359° to 0°, a circular variance needs to be used. A precedent for this method can be found in a paper by Farrugia and Micallef [34], whose method is adapted for this project and demonstrated below.

Initially, the x and y components of the wind direction vector on a two-dimensional plane are calculated from the wind direction measurements:

$$\begin{aligned}\bar{C} &= \frac{1}{N} \sum_{i=1}^N \cos(\theta_i) \\ \bar{S} &= \frac{1}{N} \sum_{i=1}^N \sin(\theta_i)\end{aligned}$$

Where θ_i is the radial wind direction for the i -th measurement, and N is the total number of measurements in the measurement window. Next, the length of the vector R is calculated:

$$R = \sqrt{\bar{C}^2 + \bar{S}^2}$$

If R is close to one, the direction measurements are concentrated around a mean direction, indicating low turbulence. Finally, the circular variance is calculated using the following formula:

$$\text{Circular Variance} = 1 - R$$

8.2.3 Turbulence Magnitude from Accelerometer

The quadcopter's IMU measures the acceleration in three dimensions. By analyzing the variance of the accelerometer data, it is possible to estimate the level of turbulence. Higher variance in the accelerometer readings indicates stronger turbulence.

This method has precedents in previous research, such as the National Oceanic and Atmospheric Administration's (NOAA) work calculating turbulence using the vertical acceleration of commercial aircraft [17]. Another previous study by Marlton et al. [29] used an accelerometer attached to a weather balloon to estimate the level of turbulence.

The first calculation method considered was to use the established formula for turbulence intensity, but replace the wind speed with the magnitude of the acceleration vector. However, as the average acceleration of the quadcopter normalized by gravity will usually remain around zero, this would result in regular divisions by zero. Additionally, vibrations of the quadcopter's motors and propellers, as well as control inputs to maintain position, can lead to an overestimation of turbulence intensity. The solution used in this project is to apply a high pass filter to the acceleration vector to filter out the low frequency vibrations, and normalize the resulting smoothed acceleration vector by gravity.

The method demonstrated below is a simplified version of similar methods used in previous research, such as the work of Ji et al. [35], using a more performant high pass filter to reduce the load on the phone's processor.

High Pass Filter

As a first step, in order to filter out the low frequency vibrations, a high pass filter is applied to the acceleration vector a . In order to use a simpler filter, the inverse of a low pass filter (LPF) is used, which is calculated by subtracting the output of the low pass filter from the original acceleration vector:

$$a_{\text{turb},x} = a_x - \text{LPF}(a_x)$$

$$a_{\text{turb},y} = a_y - \text{LPF}(a_y)$$

$$a_{\text{turb},z} = a_z - \text{LPF}(a_z)$$

The low pass filter used in this project is an Exponential Moving Average (EMA) [36]:

$$y_t = \alpha * x_t + (1 - \alpha) * y_{t-1}$$

Where y_t is the current output of the low pass filter, x_t is the current input (the original acceleration vector), and y_{t-1} is the previous output of the low pass filter. Since only the last output of y is stored, this filter requires minimal memory and decreases the weight of older measurements exponentially. α determines the amount of smoothing applied to the acceleration vector and needs to be tuned to filter out the disturbing factors.

Using the low pass filter and removing its output from the original acceleration vector as described above returns a smoothed acceleration vector, containing the high frequency vibrations, which are the ones caused by turbulence. This calculation is done for each of the three dimensions of the acceleration vector (a_x, a_y, a_z) , resulting in the smoothed components $a_{\text{turb},x}$, $a_{\text{turb},y}$, and $a_{\text{turb},z}$.

Normalization

The magnitude of the combined smoothed acceleration vector is calculated using the Euclidean vector norm [37]:

$$m_{\text{turb}} = |a_{\text{turb}}| = \sqrt{a_{\text{turb},x}^2 + a_{\text{turb},y}^2 + a_{\text{turb},z}^2}$$

The resulting magnitude m_{turb} is then used in the turbulence intensity estimation.

Until here, the calculations are done for each individual measurement within the defined measurement window, which is defined as the time period over which the turbulence intensity is calculated. To get a single value for the turbulence intensity over the measurement window, the root mean square (RMS) of the magnitudes of the smoothed acceleration vector is calculated and then normalized by gravity, as shown in the following formulas:

$$\text{RMS}(m_{\text{turb}}) = \sqrt{\frac{1}{N-1} \sum_{i=1}^N m_{\text{turb},i}^2}$$

$$\text{Turbulence Magnitude} = \frac{\text{RMS}(m_{\text{turb}})}{G}$$

Where N is the number of measurements in the measurement window, $m_{\text{turb},i}$ is the magnitude of the smoothed acceleration vector for the i -th measurement, and G is the acceleration due to gravity ($9'806.65 \text{ mm/s}^2$). This results in a normalized turbulence magnitude value that can be used to estimate the level of turbulence.

8.2.4 Calculating the Final Turbulence Intensity

The final turbulence intensity estimation is calculated by combining the turbulence intensity estimated from the variance of the wind speed, the circular variance of the wind direction, and the turbulence magnitude from the accelerometer. A weighted average of these three values is calculated to get a final turbulence intensity estimation, which will then be categorized into the different levels of turbulence as defined in Section 4.3.2.1.

The weighted average is calculated using the following formula:

$$\text{Final Turbulence Intensity} = \frac{w_1 * \text{TI} + w_2 * \text{CV} + w_3 * \text{TM}}{w_1 + w_2 + w_3}$$

Where w_1 , w_2 , and w_3 are the weights assigned to each of the three turbulence intensity estimation methods TI (turbulence intensity from wind speed), CV (circular variance from wind direction), and TM (turbulence magnitude from accelerometer). The weights need to be tuned based on reference measurements to find the best combination for accurate turbulence intensity estimation.

8.3 Reference Measurements

In order to calibrate the wind speed and turbulence measurements, a series of reference measurements takes place. These take place at different locations with varying objectives.

8.3.1 Wind Shaper Testing

Wind tunnel tests allow flight in controlled and repeatable conditions. The OST Techpark in Rapperswil provides a Windshaper [38], that can be used for this purpose. It consists of an array of 6x6 small fans, which can be individually controlled to generate dynamic wind conditions. During the tests, a portable anemometer (testo 410i) is used alongside the quadcopter to collect reference data.

Turbulence Simulation Function

To simulate turbulence using the Windshaper, mathematical functions can be used to control the fans intensity over time and depending on their position in the grid. To create an irregular turbulent flow that can be used for repeated deterministic tests, the following combination of sine and cosine waves is used:

$$f(A_{\max}, A_{\min}, \gamma, k, t) = \left(\frac{A_{\max}}{2} - \frac{A_{\min}}{2} \right) * (\cos(\sin(\gamma + kt) + 0.2kt)) + \frac{A_{\max}}{2} + \frac{A_{\min}}{2}$$

Where:

- $A_{\max}$: Maximum fan speed (configurable between 0 and 100%)
- $A_{\min}$: Minimum fan speed (configurable between 0 and 100%)
- γ : Phase shift (configurable between 0 and 2π)
- k : Frequency (configurable)
- t : Time in Seconds (not configurable, increases continuously during the test)

Here is an example of the fan speed over time using the above function with $A_{\max} = 15\%$, $A_{\min} = 5\%$, $\gamma = 4$ and $k = 1$, where t is represented on the x-axis and the resulting fan speed is represented on the y-axis:

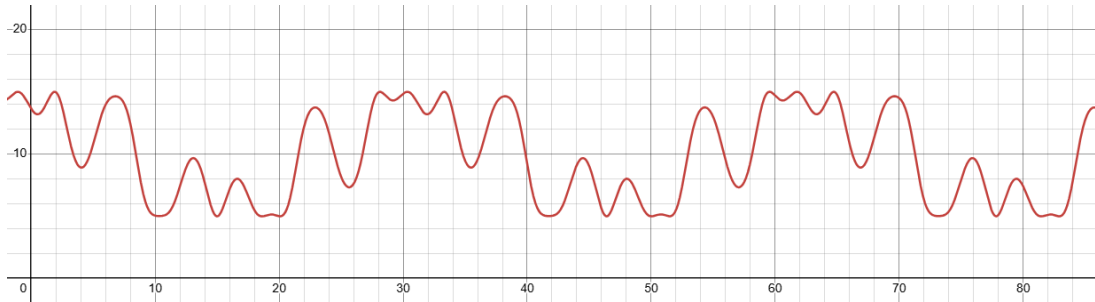


Figure 17: Example Fan Speed over Time using the Turbulence Function

8.3.2 Field Measurements

Field measurements allow for testing the quadcopter's wind estimation and turbulence intensity estimation in real-world conditions. Reference measurements will be taken using a portable anemometer (testo 410i) mounted on a tripod alongside the quadcopter to collect reference data. To get an overall picture of the wind conditions, a wind sock is mounted on the tripod as well, allowing for a visual reference of the wind direction and speed.



Figure 18: Field Measurement in Meiringen

Field measurements take place in Meiringen BE and Innertkirchen BE, to calibrate the wind speed and turbulence estimation and test the overall system. These locations are chosen for their varying wind conditions due to the surrounding mountains. Several thermals are expected in the surroundings of Meiringen, while Innertkirchen is situated in a basin, causing mountain wave turbulence and strong eddies (see Section 4.3).

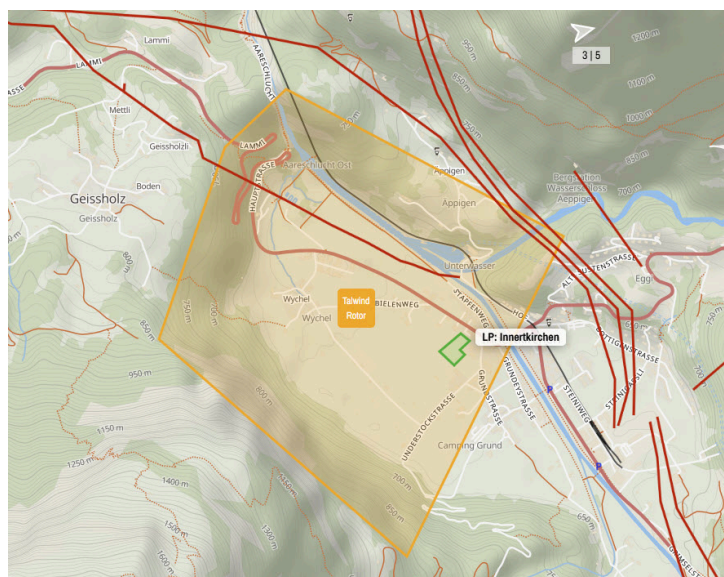


Figure 19: Known Turbulence Zone in Innertkirchen [6]

Chapter 9

Design and Architecture

The architecture of the quadcopter control system consists of two main components: the quadcopter itself and the companion app. The quadcopter is responsible for collecting wind data and transmitting it to the companion app, which processes the data and provides insights to the user. The documentation of the architecture is based on [arc42](#).

9.1 Introduction and Goals

The overall goal of the architecture is to integrate the sensor capabilities of the quadcopter into the companion app and in return control the quadcopter in a way that no external components like a remote control are required. Any outside components used are only used for development and testing purposes, but not in the final product. The companion app is an Android application built with React Native. The architecture is designed in a way that it can be easily extended in the future, for example by adding new sensors or features to the quadcopter or the companion app.

9.1.1 Requirements

The requirements for the quadcopter control system are detailed in [Chapter 7](#).

9.1.2 Quality Goals

The following quality goals are defined:

Table 2: Quality Goals of the Architecture

Category	Quality	Description
Security	Flight Safety	Safety measures are in place to prevent incidents during flight.
Maintainability	Extendability	The architecture should be designed in a way that it can be easily extended with new features.
Operability	User-Friendly Interface	The companion app should have an intuitive and user-friendly interface to ensure ease of use.
Reliability	Stable and Performant Connection	A stable and reliable connection between the quadcopter and the companion app needs to be ensured to prevent data loss or loss of control during flight.
Reliability	Synchronization	The quadcopter's state should be consistently synchronized with the companion app.
Compatibility	Support for alternate ArduPilot Quadcopters	The architecture should be designed to support other ArduPilot quadcopters using the same firmware in the future.
Compatibility	Support for alternate control methods	The quadcopter should be able to be controlled by other control methods like a different app or desktop application that supports MavLink.

The corresponding architectural approaches to achieve these quality goals are detailed in [Section 9.4](#).

9.1.3 Stakeholders

The stakeholders are mainly the paragliding pilots, the students and the supervisor. The architecture is designed with their needs and expectations in mind, especially the paragliding pilots who will be using the companion app to collect the wind data by flying the quadcopter.

9.2 Constraints

The project is subject to a number of constraints. These are detailed in Section 2.2 (technological constraints). The architecture is designed in a way that it can be implemented within these constraints.

9.3 Context and Scope

The context of the architecture involves the interaction between the quadcopter and the companion app. Reference data from external sources, such as weather stations or anemometer readings are used during testing, but are not in the scope of the architecture.

9.3.1 Business Context

The two main components of the architecture, the quadcopter and the companion app, are designed so the paragliding pilot (here forward referred to as “pilot”) is able to directly interact with the quadcopter and receive the collected wind data in the companion app without the need for any external components like a remote control.

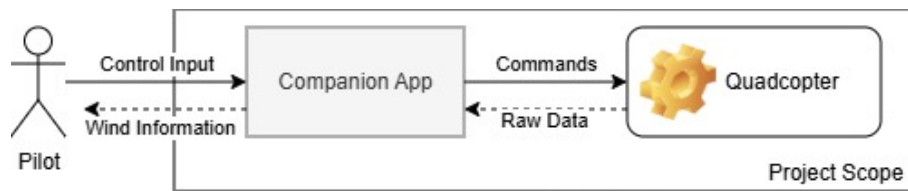


Figure 20: Business Context

9.3.2 Technical Context

The technical context of the companion apps architecture consists of the quadcopter, the companion app, and the communication between them. The quadcopter is equipped with various sensors to collect wind data, which is then transmitted to the companion app via a wireless connection. The center piece of this connection is the WiFi receiver, which is part of the quadcopter.

The companion app connects directly to the WiFi receiver and processes the received data, fulfilling requirement FR-Q2. The same WiFi receiver also allows the companion app to send commands to the quadcopter and control its flight, to fulfill requirement FR-Q3.

The quadcopters sensors are influenced by environmental factors such as weather conditions, the location or the terrain.

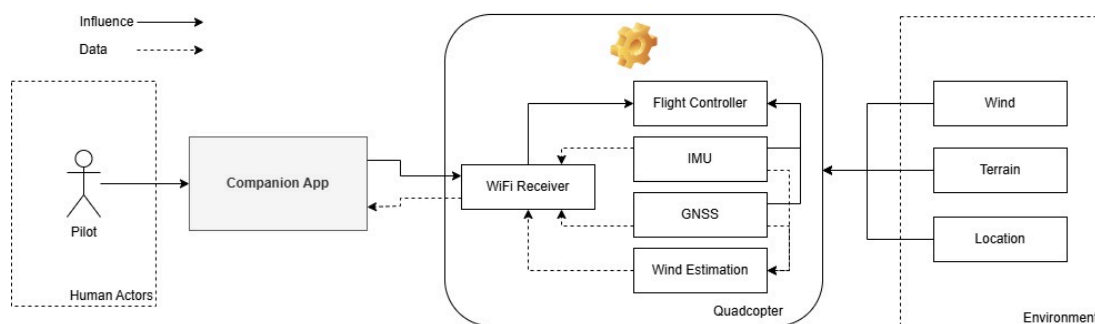


Figure 21: Technical Context

9.4 Solution Strategy

The aim is to choose technologies and design the architecture in a way that it meets the defined quality goals in Section 9.1.2 while also being feasible to implement within the given constraints.

Table 3: Solution Strategy

Quality Goal	Architectural Approach	Details
Stable Connection	Robust WiFi Communication Protocol	The WiFi receiver on the quadcopter connects via UDP to the pilot's phone. Details in Section 9.7.1 and Section 10.2.1.
Flight Safety	Emergency Functions always accessible	The companion app is designed to have emergency functions that are always accessible to the pilot during flight. Details about the design and implementation of these functions can be found in Section 10.3.1, while the management of the corresponding states in the companion app is detailed in Section 9.7.3.
Flight Safety	Use Intelligent Flight Control Firmware	Although the pilot can control the quadcopter through the companion app, the onboard flight-control algorithms continuously stabilize the aircraft, maintain its position, and execute missions autonomously – continuing to operate safely even if the connection to the ground station is interrupted.
Extendability	Modular Design	The architecture is designed to be modular, with clear separation of concerns between the different components. The companion app uses React Native, which allows for a modular and component-based architecture, making it easier to add new features in the future. Details about this decision can be found in Section 9.8.3.
User-Friendly Interface	Intuitive Design	The companion app is designed to have an intuitive and user-friendly interface to ensure ease of use for the pilot. Details in Section 9.10.
Consistent State Management	Synchronization	The companion app should have a consistent state management approach to facilitate the synchronization of the quadcopter's state with the app. Details in Section 9.7.3.
Support for alternate control methods	Standard MavLink Protocol	The quadcopter uses the standard MavLink protocol for communication, which allows it to be controlled by other control methods in the future. Details in Section 9.7.1.

9.5 Building Block View

As the companion app is implemented in React Native, the architecture is designed in a modular and component-based way. Therefore, a component diagram is borrowed from C4 to display the building blocks of the architecture. The component diagram only gives a high level overview. Please refer to the full class diagram in Appendix A.3 for a detailed view.

The `drone-status.store.ts` store is the central link between the UI logic and the quadcopter communication logic in the companion app. The stores use Zustand as the state management library, which is described in more detail in Section 9.7.3. It synchronizes the quadcopter's state with the app and vice versa. The UI reacts to state changes by the quadcopter and updates the displayed information accordingly (FR-A15, FR-A16). The quadcopter receives commands from the UI through the store via state changes as well as from `manual-flight.service.ts` and `auto-flight.service.ts` services for direct calls. This allows the manual or automatic control of the quadcopter, fulfilling the requirements FR-A1 and FR-A2.

Note: Throughout the code base of the companion app, the term “drone” is used to refer to the quadcopter. In the diagrams and the following architecture sections these terms are used interchangeably.

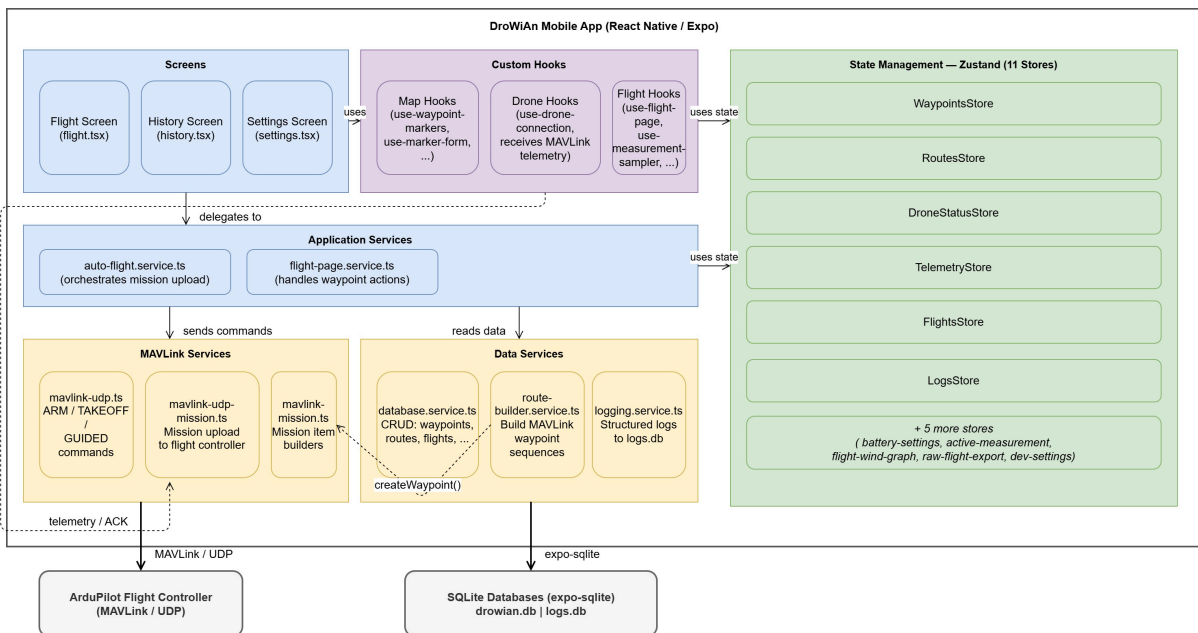


Figure 22: Component Diagram

The underlying persistent storage uses SQLite, which is a lightweight and efficient database engine that is well-suited for mobile applications. The Database runs locally on the device as an embedded library rather than a separate server process. This provides us the advantage of rapid data access and offline capability.

The database schema is shown in the following diagram:

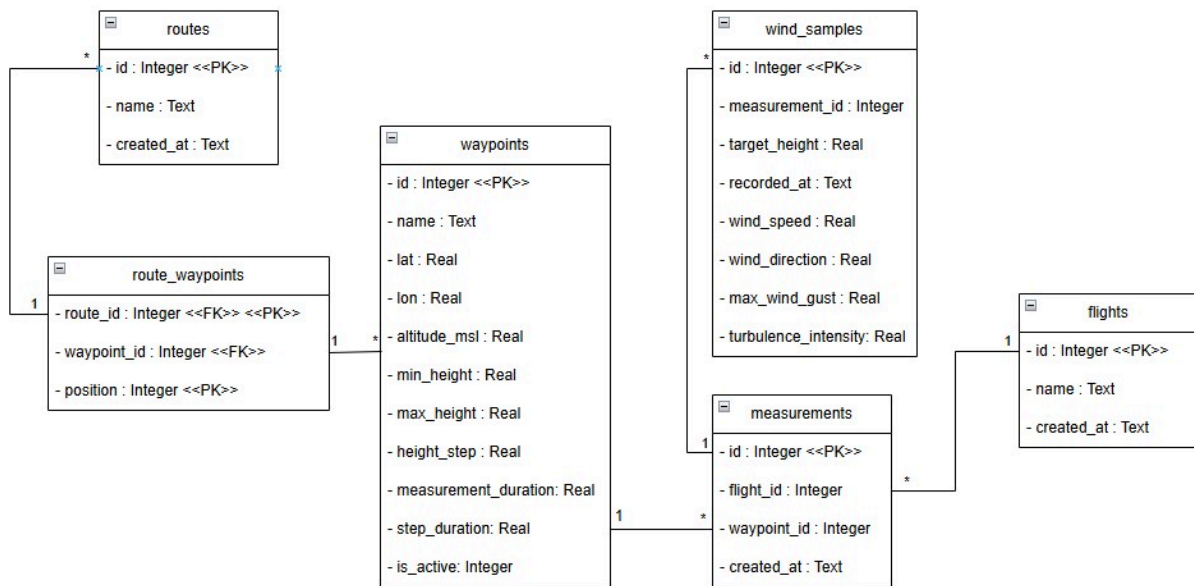


Figure 23: Database Schema

For all flights the measurements made during that flight are stored in the measurements table. A measurement is linked to the waypoint at which the measurement was taken, as well as the wind_samples that were collected as part of that measurement. A measurement can have multiple wind samples if the wind conditions were measured at multiple altitudes at the same waypoint. The wind_samples table stores measured wind conditions for each altitude, including average wind speed and direction, turbulence intensity, and the maximum gust factor.

The waypoints table contains the waypoints that have been planned by the user. Each waypoint is defined by its coordinates, and height or height range. Waypoints are soft-deleted by setting an is_active flag rather than being fully removed from the database. This ensures that historical measurements remain linked to the waypoint at which they were taken, which is required for the history page to display past flights correctly.

Routes can be planned by creating a route in the routes table and linking waypoints to that route in the route_waypoints table. Currently, only one route can be created in the UI. Having the routes table could be useful if the app is extended in the future to support the saving of multiple routes (which supports the “extendability” quality goal).

9.6 Runtime View

As an example of the runtime behaviour of the system, the process of creating waypoints, programming a route, and sending the route to the quadcopter is described below.

The pilot starts by creating a waypoint on the map (FR-A7). The corresponding services, hooks and stores save the waypoint in the database and update the map accordingly. This process is repeated until the pilot has created all the waypoints for the intended route. Editing an existing waypoint (FR-A8) or deleting one (FR-A9) follows the same process, but instead of creating a new entry in the database, the existing entry is updated or deleted.

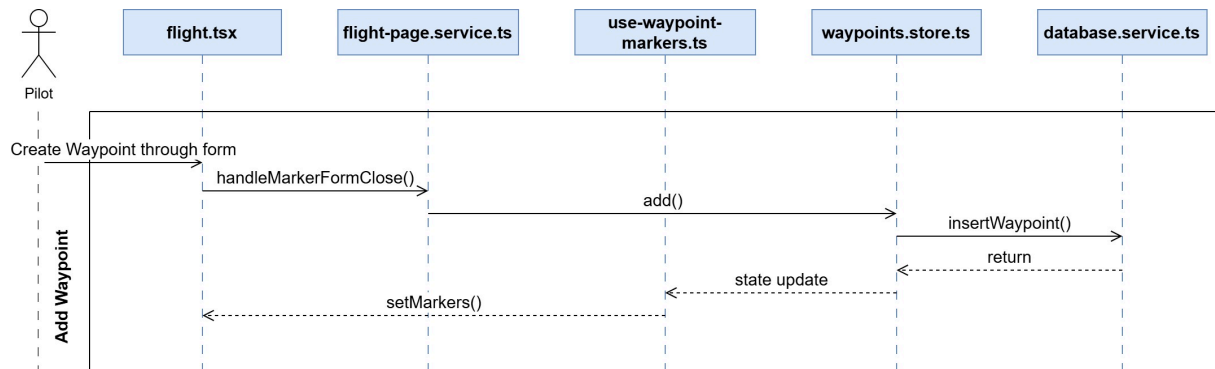


Figure 24: Runtime View - Programming Waypoints and Route

Once all required waypoints are created, the pilot can add them to the route by clicking “Add to Route” (FR-A10, FR-A11) in the waypoint form. This stores the waypoint in the `route_waypoints` table and links it to the route. Once all waypoints are added to the route, the pilot can send it to the quadcopter by clicking “Upload Mission” on the flight page. This triggers the process in the auto-flight service that builds the mission by adding helper-waypoints that are required to establish the cruising altitude between user waypoints and enable the measuring of multiple altitudes at waypoints (FR-A12, FR-A27). A take off and land mission item are added at the beginning and end of the mission respectively, and the user waypoints are added in between. The mission is then sent to the quadcopter via MavLink. For more details about the building of the route and the helper waypoints, refer to Chapter 10.

Once the mission is uploaded to the quadcopter, the pilot can start the flight by clicking the “Start Mission” button. This triggers the arming of the quadcopter and therewith the take off (FR-A3). Then the quadcopter will fly the defined route and rest at each waypoint for the defined time.

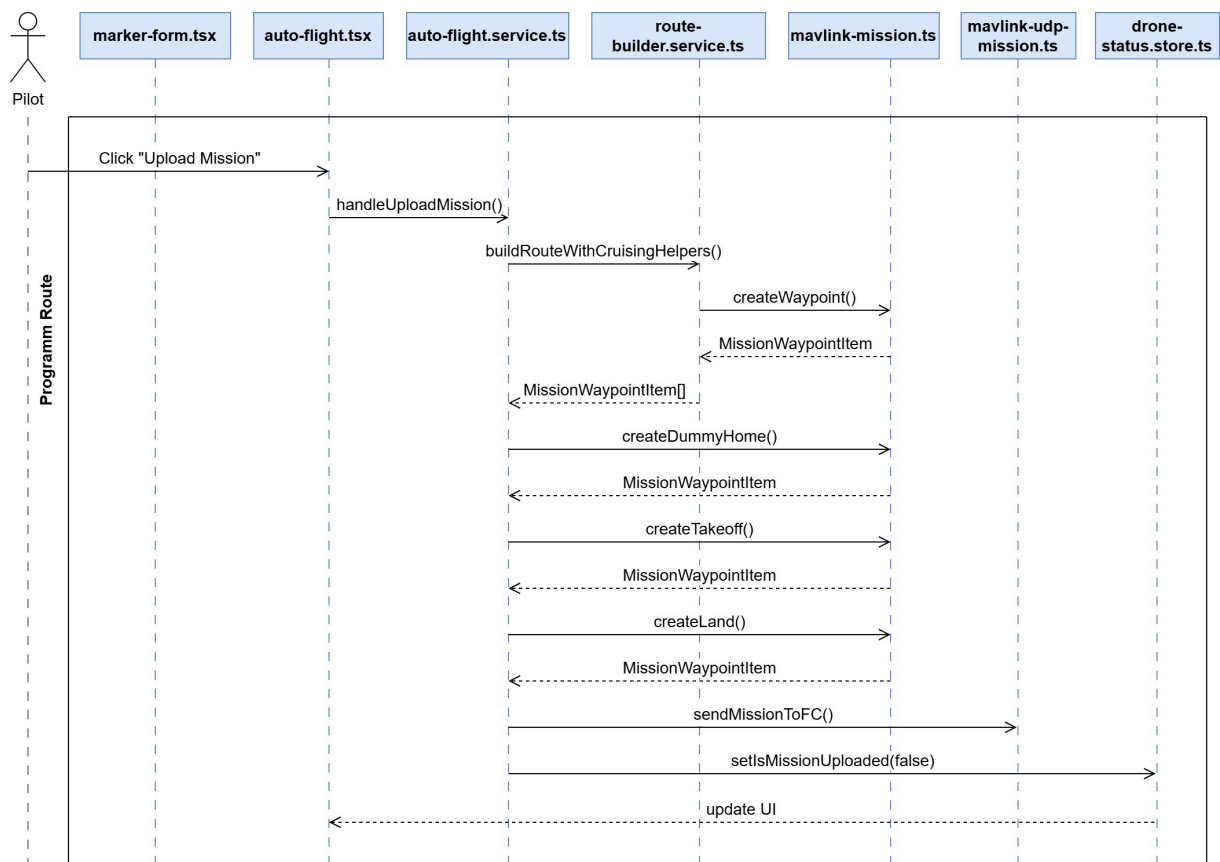


Figure 25: Runtime View - Uploading and starting the Mission

9.7 Crosscutting Concepts

Some concepts are relevant across multiple components of the architecture.

9.7.1 UDP Connection

To exchange real-time telemetry data and control commands, a stable and reliable UDP connection must be established between the companion app and the quadcopter. This is achieved by the pilot's phone connecting directly to the quadcopter's WiFi receiver. Through this connection, the companion app receives telemetry data such as the aircraft's position, altitude, battery status, and wind estimations, while transmitting control commands including arming/disarming, mission uploads, flight settings, and Return-to-Launch (RTL) commands.

9.7.2 Logging

The logging framework is implemented as a custom solution, as documented in ADR 06, to satisfy the requirements of NFR4 while maintaining full compatibility with React Native. Log entries are classified along two orthogonal dimensions: *level* (debug, info, warn, error) and *category* (mavlink, flight, app, database), which together enable two-dimensional runtime filtering as required by NFR4. Levels carry a numeric rank (0–3), so that selecting a minimum level such as info implicitly includes all higher-severity levels without requiring explicit configuration.

Entries are persisted in a dedicated SQLite database (`logs.db`). Each entry stores the level, category, message, and an ISO 8601 timestamp and indexes on all three filter dimensions keep query performance acceptable. To bound storage consumption, entries are automatically pruned to the 5'000 most recent records after each insertion. Because database initialisation is asynchronous, log calls issued during application startup are buffered in memory and flushed once the database is ready. Each message is also emitted to the development console to facilitate debugging without requiring a database query. The schema of the logging database is shown below:

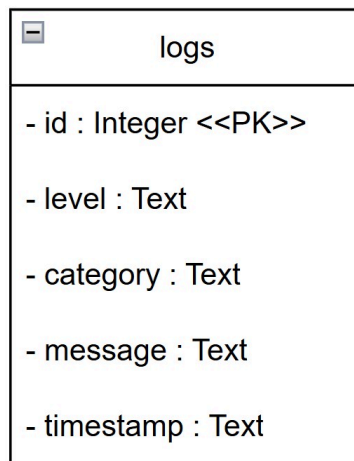


Figure 26: Logging Database Schema

The Zustand store `useLogsStore` bridges the service and the UI, managing the active filter, the fetched entries, and a loading state. The log viewer is accessible in a dedicated tab within the Settings section of the companion app, fulfilling FR-A26.

9.7.3 State Management

Zustand is a lightweight state management library for React applications. It provides a simple, hook-based API to define global stores that any component or service can read from and write to, without the boilerplate of more complex solutions like Redux. Zustand is used as the state management library for the companion app, as decided in ADR 04. The following state machine diagram illustrates the different states of the quadcopter that need to be managed by the companion app and the transitions between them:

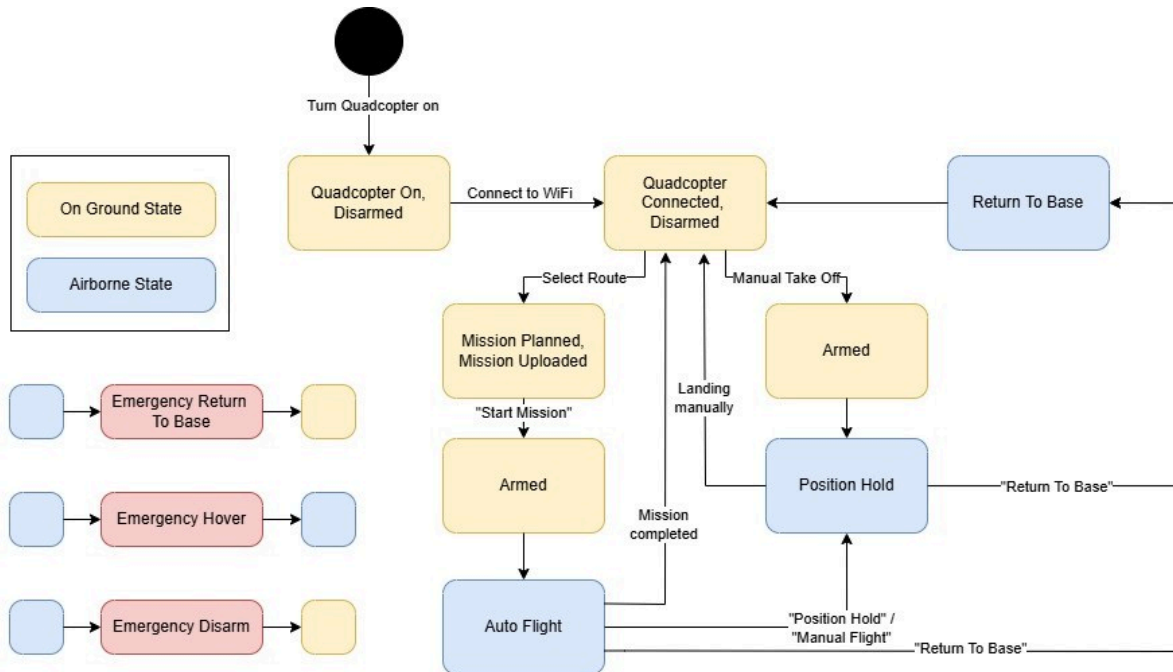


Figure 27: State Machine Diagram

Ground based states are shown in yellow, while airborne states are shown in blue. When switching from a ground based state to an airborne state, the quadcopter is armed, which means that the motors are ready to be powered up, and the quadcopter takes off.

For clarity, the emergency states are not included in the main diagram, but shown exemplarily on the bottom left in red. Each emergency state can be entered from any airborne state. After the emergency function is executed, the quadcopter returns back into the corresponding non-emergency states:

- “**Emergency Return To Base**” returns to “**Quadcopter Connected, Disarmed**” (FR-A20)
- “**Emergency Hover**” returns to “**Position Hold**” (FR-A21)
- “**Emergency Disarm**” returns to “**Quadcopter Connected, Disarmed**” (FR-A23)

This state architecture allows the pilot to switch between flight modes, such as from automatic to manual flight (FR-A6) or initiate a return to base from any flight mode (FR-A4).

9.7.4 Data Export

Two categories of measurement data can be exported from the companion app, fulfilling FR-A19: aggregated wind sample data and raw telemetry data.

Wind sample data is exported as JSON from the history view, where the pilot can export samples per altitude level or for a complete single-altitude measurement. Each export contains metadata (timestamp, altitude AGL, sample count) alongside the individual samples with wind speed, wind direction, turbulence intensity, and maximum gust factor. The file is written to the device's cache directory and shared via the native share dialog provided by expo-sharing.

Raw telemetry is recorded continuously throughout a measurement at the frequency at which the quadcopter sends telemetry data by the `useRawFlightRecorder` hook, which buffers entries in the `useRawFlightExportStore`. Each entry captures a timestamped snapshot of position, wind vector, accelerometer, and gyroscope readings. When a measurement ends, the buffer is flushed to `latest-flight-raw.json` in the device's documents directory via expo-file-system. The pilot can share this file from the Settings tab using the same native share dialog. In order to keep the storage size manageable, the raw data will only be saved for the latest flight.

9.8 Architectural Decisions

The architectural decisions made during the design of the system are documented below. These decisions are based on the defined quality goals, constraints, and the technical context of the project.

9.8.1 ADR 01: Quadcopter Software

The semester thesis [7] by the project team defined ArduPilot as the quadcopter software for the project. This decision is carried forward in this thesis, as it builds on the existing prototype. More information about the decision can be found in the documentation of the previous semester thesis as “ADR 01: UAV Software”.

9.8.2 ADR 02: Communication Protocol

Title	Communication Protocol
Context	The communication protocol between the quadcopter and the companion app needs to ensure a stable, reliable and performant connection.
Decision	We decided to use UDP as the communication protocol, because it allows for low-latency transmission of real-time telemetry data and control commands, which is crucial for the responsive operation of the quadcopter.
Status	Accepted
Consequences	The framework used for the companion app needs to support a direct UDP connection to the quadcopter.

9.8.3 ADR 03: Mobile App Framework

Title	Mobile App Framework
Context	The mobile app framework is responsible for providing the foundation for the development of the companion app. It must support the UDP connection to the quadcopters WiFi receiver and allow for a modular approach. Additionally it should be possible to port the app to multiple operating systems without rewriting the entire codebase.
Decision	We decided to use React Native as the mobile app framework and Expo as the development toolchain for the companion app. Establishing a direct UDP connection to the quadcopter’s WiFi receiver requires a native approach, ruling out web-based frameworks. This approach also facilitates cross-platform compatibility, allowing the companion app to be developed for both iOS and Android devices. Additionally, its reactive data binding and component-based architecture supports a modular design, making it easier to add new features in the future.
Status	Accepted
Consequences	The mobile app must be developed using React Native, and the team must ensure compatibility with the target platforms.

9.8.4 ADR 04: State Management in the Companion App

Title	State Management in the Companion App
Context	The companion app is highly dependent on the state of the quadcopter, as its interface and functionality vary based on the quadcopter's state. Therefore, managing the state globally across the app facilitates the synchronization of the quadcopter's state with the app and ensures a consistent user experience.
Decision	We decided to use Zustand as the state management library for the companion app, because it offers a simple hook-based API with minimal boilerplate and no need for reducers or dispatchers, while being lightweight enough to integrate seamlessly with React Native.
Status	Accepted
Consequences	The companion app must be developed using Zustand for state management, and the team must ensure proper integration with React Native.

9.8.5 ADR 05: Data Persistence in the Companion App

Title	Data Persistence in the Companion App
Context	The measurements collected by the quadcopter and the programmed waypoints in the companion app need to be persisted across sessions. Restricting factors in the choice are the compatibility with React Native and the intended cross-platform compatibility of the app.
Decision	We decided to use SQLite via expo-sqlite as the embedded database for data persistence. SQLite runs locally on the device without requiring a separate server process, and is well-supported on both Android and iOS through the Expo ecosystem.
Status	Accepted
Consequences	The companion app must be developed using SQLite for data persistence, and the team must ensure proper integration with React Native via expo-sqlite.

9.8.6 ADR 06: Logging Framework for the Companion App

Title	Logging Framework for the Companion App
Context	Various parameters and events from the quadcopter and the companion app need to be logged for debugging and monitoring purposes. The logging framework needs to be compatible with React Native and support the classification of logs into different levels and categories, as defined in NFR4.
Decision	We decided to create a simple custom implementation for the logging in the companion app, which uses SQLite for data persistence. This approach allows for a logging framework that is tailored to the specific needs of the companion app and ensures compatibility with React Native.
Status	Accepted
Consequences	All aspects of the logging in the companion app need to be custom implemented, including the classification of logs into different levels and categories, as defined in NFR4. The logging framework needs to be usable throughout the entire code base of the companion app.

9.8.7 ADR 07: Styling Library for the Companion App

Title	Styling Library for the Companion App
Context	The companion app requires a consistent and maintainable styling approach. The styling library needs to be compatible with React Native and support responsive design.
Decision	We decided to use Nativewind as a styling library, because it allows using Tailwind CSS classes in React Native, which have already been used for the design of the mockup in Section 7.3. This approach also facilitates a consistent and maintainable styling across the app. We decided against using a component library like “NativewindUI” or “Gluestack”, because of our specific design requirements, which are not fully met by the available components in these libraries.
Status	Accepted
Consequences	As this decision was made after starting the implementation of the companion app, existing styles need to be refactored to use Nativewind. New components need to consequently be styled using Nativewind as well.

9.8.8 ADR 08: Map Provider for the Map Component in the Companion App

Title	Map Provider for the Map Component in the Companion App
Context	The companion app requires a map component to display the quadcopter’s location and waypoints as well as the measurements on previous flights. The map provider needs to be compatible with React Native and support custom markers.
Decision	We decided to use React Native Maps as map provider, because it is integrated with Expo, which is the development toolchain for the companion app, and it supports custom markers. On Android, React Native Maps uses Google Maps as the underlying map provider. We chose it over “Expo Maps”, which is also integrated with Expo, because Expo Maps is still in alpha stage and frequently encounters breaking changes, which would have posed an unnecessary risk for the project.
Status	Accepted
Consequences	To use React Native Maps, a Google API key is required. This key can only be used in an app, if a credit card is linked to the Google account, as the demo key does not allow the use of the maps SDK in an app. The costs generated through the map integration therefore need to be monitored closely, especially during development when the map is used frequently.

9.8.9 ADR 09: Supported Mobile Platforms for the Companion App

Title	Supported Mobile Platforms for the Companion App
Context	The companion app needs to be developed for one or more mobile platforms. The choice of the supported platforms has implications for the implementation effort, additional costs, and the potential user base of the app.
Decision	We decided to develop the companion app only for Android , as the majority of the team uses Android devices, and the development for iOS requires either a Mac or a developer account, which would have generated additional costs. Additionally, the development and testing effort for one platform is significantly lower than for two platforms.
Status	Accepted
Consequences	The companion app must be developed for Android devices. As one developer uses an iOS device, an emulator needs to be set up for development and testing purposes.

9.9 Technical Debt

The technical debt of the architecture is documented below. This includes any shortcuts or compromises made during the design and implementation of the system that may need to be addressed in the future.

Table 4: Technical Debt

Area	Debt	Mitigation Strategy
Map Integration	Rendering the very first waypoint marker on the map results in the waypoint marker being only partially visible. Currently, a workaround is implemented, which involves setting its name hardcoded to “WP 1” to trigger a re-render of the waypoint marker. This workaround doesn’t impact the performance or usability of the app, but it is not an ideal solution and should be addressed in the future.	The marker logic in the map component needs to be revised to handle the initial rendering of waypoint markers correctly.
Quadcopter Altitude Error	The quadcopter uses a barometer to measure its altitude. This barometer is subject to measurement errors due to the rotor wash and environmental wind influencing the local air pressure around the barometer. The erroneous reading created by this phenomenon can amount to an imprecision of around 1 m with the quadcopter used in this project. ArduPilot provides a feature that helps reduce the error caused by the wind speed [32], however, due to time constraints this correction has not yet been made.	For the time being, waypoints should be defined with an altitude of at least 2 m to have enough margin to the ground. In the future, the correction parameter in ArduPilot can be calculated to reduce the altitude error.
Quadcopter Connection	To establish a connection and control the quadcopter through the companion app, the cellular data of the phone needs to be disabled.	As the system is fully functional with the cellular data disabled, this is not addressed in the current implementation.

9.10 UX Design

The UX design of the companion app focuses on being intuitive and user-friendly so all the implemented features are easily accessible to the pilot before and during flight. The design also ensures that the most important information and functions are always accessible to the pilot during the flight.

The design is closely oriented on the mockup, found in Section 7.3, which was created based on the requirements specified in Chapter 7.

To ensure consistency and offer a familiar user experience, the overall design of the companion app uses as many native components as possible.

The user interface is divided into three sections, which adapt depending on the quadcopter's status and the selected tab in the app:

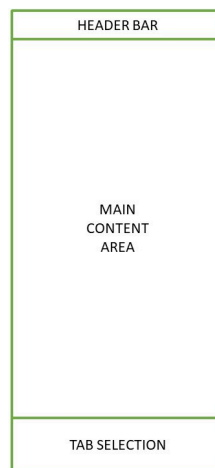


Figure 28: UX Layout

9.10.1 Header Bar

The header bar is located at the top of the app and provides a persistent status overview and quick access to emergency functions. Details are described in Chapter 10.

Tapping the header bar opens an overlay with more detailed wind and turbulence information. Details are described in Chapter 10.

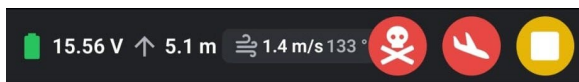


Figure 29: Header Bar

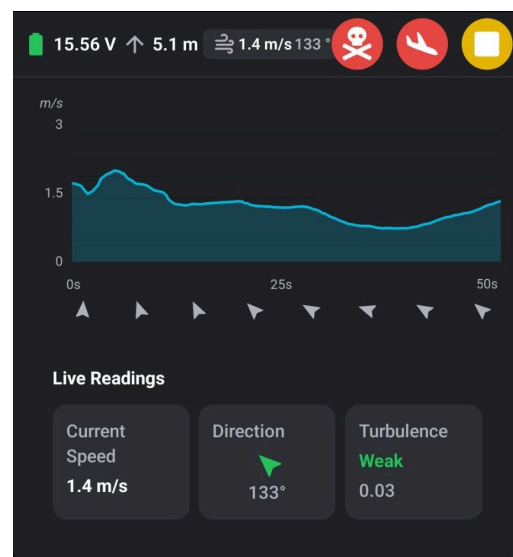


Figure 30: Header Bar Overlay

9.10.2 Main Content Area

The main content area takes up the majority of the screen and displays different content based on the selected tab. More details about the content of the different tabs can be found in Chapter 10.

9.10.3 Tab Selection

The lowest part of the screen houses the tab selection. “Flight”, “History”, and “Settings” are the three available tabs.

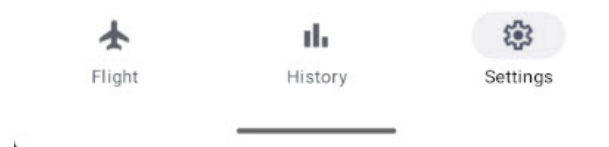


Figure 31: Tab Selection

9.10.4 Iconography

Wherever possible, the app uses either native icons that are familiar to the user or icons and colors that are commonly used in aviation. Examples for the latter include:

- Red and amber colors for the emergency buttons
- A magenta line to indicate the programmed flight path on the map
- A triangle for the waypoint marker
- Showing the waypoint’s altitude as a number next to the waypoint marker on the map, with lines above and below the number to indicate the altitude constraints
- Wind barbs to indicate the wind speed and wind direction

The wind barb iconography deviates slightly from the standard aviation iconography, as they are color coded to indicate the estimated turbulence intensity. Green wind barbs indicate weak turbulence, yellow wind barbs indicate moderate turbulence, and red wind barbs indicate severe turbulence. This deviation from the standard aviation iconography was made to provide the pilot with more information about the wind conditions at a glance.

Another deviation from standard aviation iconography is the use of a house icon on the map, which indicates the home location of the quadcopter, the location from which the quadcopter took off.

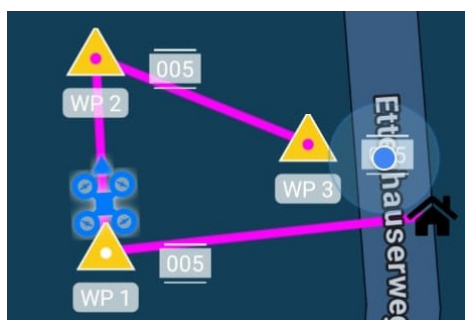


Figure 32: Route Iconography

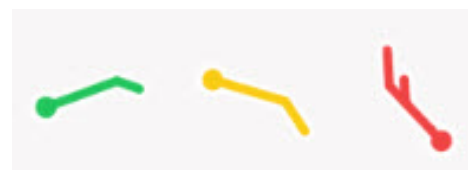


Figure 33: Wind Barb Iconography

9.10.5 Theming

The app adapts automatically to the system theme of the user's phone. The handling of the theming is implemented by using the theming capabilities of Nativewind, using a central theme file to define the different colors for the light and dark themes.



Figure 34: Light and Dark Themes

Chapter 10

Implementation

Whereas the previous chapter gives an overview over the overall architecture of the system, this chapter focuses on the implementation details of the two main components: the quadcopter and the companion app. It describes the technologies used, the development process, and any challenges faced during implementation.

10.1 Quadcopter

This section describes the design, hardware selection, firmware configuration, and parameter tuning of the quadcopter used in this project. The construction process was heavily guided by the ArduCopter Methodic Configurator documentation [39], which provided the foundational knowledge for component selection, firmware setup, and systematic parameter tuning. ArduCopter is the implementation of ArduPilot for multicopters.

10.1.1 Quadcopter Frame

As required by NFR1, the quadcopter must weigh less than 250 g. To maximize flight time, the weight was reduced even further wherever possible. Special thanks to Jakub who provided the base design [40].

To achieve this, a custom quadcopter frame was designed and manufactured using 3D printing with PA6-CF (carbon-fibre reinforced polyamide) filament, chosen for its high stiffness-to-weight ratio. The resulting frame weighs only 16 g while providing mounting points for all required components. Designing the frame specifically for the selected hardware eliminated the need for additional adapters or mounting hardware, with the exception of a few light rubber bands.



Figure 35: The custom 3D-printed Quadcopter Frame

To minimize weight further, no protective housing was placed around the electronic components. This approach reduces overall mass but leaves components more exposed to impacts and environmental influences. One example is the barometer, which is directly exposed to airflow. Wind pressure can distort altitude measurements, although this effect can largely be reduced as discussed in Section 12.2.3.

10.1.2 Flight Controller

The selected FC is the SpeedyBee F405, built around an STM32F405 microcontroller [41] running at 168 MHz with 1 MB of flash. This controller is designed for lightweight quadcopters while still providing support for a wide range of sensors and communication interfaces.

During the early stages of the project, it was unclear whether a dedicated wind speed sensor would be integrated into the system. Therefore, the availability of multiple serial interfaces and broad sensor support were important selection criteria. Additionally, the SpeedyBee F405 is supported by the ArduCopter firmware, which was a key requirement for this project.

A central characteristic of this architecture is that the flight controller is a self-contained autopilot rather than a simple signal relay. All flight-critical computation runs directly on the controller's own processor: it stabilizes the quadcopter in the air, navigates to a given position, performs takeoff and landing, returns to the home location, holds position over a measuring point, and estimates the wind without any input from the companion app. The companion app on the other hand issues high-level instructions, such as uploading a mission or commanding a mode change. This separation is what allows the quadcopter to continue operating safely even if the link to the app is lost.

10.1.3 Final Build

The assembled quadcopter consists of three vertically stacked modules. The **top stack** holds the ESP32-C3 WiFi bridge, whose antenna is routed to the bottom-left corner of the frame and connects via UART (a serial point-to-point communication interface) to the flight controller below, forwarding MAVLink telemetry wirelessly to the companion app. The **middle stack** is the SpeedyBee F405 flight controller: it hosts all flight-critical sensors, executes the uploaded mission autonomously, and reads its GPS position from the external GPS module mounted in the top-right corner of the frame, then sends top-level motor instructions to the ESC directly below. The **bottom stack** is the 4-in-1 ESC, which connects directly to all four motors and translates the throttle signals from the flight controller into the corresponding motor speeds.



Figure 36: 295 g — Improvised Prototype from the Semester Thesis



Figure 37: 190 g — Quadcopter used in the Bachelor Thesis

The final build is 105 g lighter than the semester thesis prototype. The old quadcopter carried a heavy protective enclosure together with a significantly larger, more material-intensive frame that alone accounted for 47 g without providing any measurable stability benefit. Its GPS module was placed directly adjacent to high-current ESC wiring, causing magnetic interference that degraded compass accuracy. The WiFi antenna was also left unsecured and not properly routed away from the electronics, reducing signal reliability. All of these issues were addressed in the final design.

10.1.4 Firmware

As required by NFR2, the flight-control software had to be open source. Furthermore, the project required onboard wind estimation capabilities. Among the evaluated flight stacks, ArduCopter was the only viable option because it supports both wind estimation discussed in Section 8.1 and autonomous flight operations.

Due to the SpeedyBee F405 having only 1 MB of flash memory available, not all ArduCopter features could be included in the standard firmware build. Consequently, a custom firmware image was generated using the ArduPilot Custom Firmware Builder (see Appendix A.4.7.1). This custom build allowed the inclusion of the wind-estimation functionality required for the project while excluding unnecessary features to remain within memory constraints.

10.1.5 Quadcopter Parameters

ArduCopter is a highly configurable flight-control system containing thousands of parameters, many of which influence one another. Although autonomous flight is supported by the firmware, reliable operation could not be achieved using the default configuration.

A significant portion of the development effort was therefore dedicated to researching, testing, and tuning parameters to achieve stable autonomous flight, reliable takeoff and landing behaviour, and robust operation under varying wind conditions. For some parameters, the official documentation was insufficient to fully understand their behaviour, and the ArduPilot source code [42] had to be consulted directly to determine their true effect.

Examples of particularly important parameters are shown in Figure 38.

Name	Δ	Value	Default	Units	Options	Desc
ATC_RAT_RLL_I		0.135	0.135	1/rad	0.01 2.0	Roll axis rate controller I gain. Corrects long-term difference in desired roll rate vs actual roll rate
ATC_RAT_RLL_IMAX		0.5	0.5		0 1	Roll axis rate controller I gain maximum. Constrains the maximum that the I term will output
ATC_RAT_RLL_P		0.135	0.135	s/rad	0.01 0.5	Roll axis rate controller P gain. Corrects in proportion to the difference between the desired roll rate vs actual roll rate
ATC_RAT_RLL_PDMX		0	0		0 1	Roll axis rate controller PD sum maximum. The maximum/minimum value that the sum of the P and D terms can exceed
ATC_RAT_RLL_SMAX		0	0		0 200	Sets an upper limit on the slew rate produced by the combined P and D gains. If the amplitude of the control action would be greater than this, the feed-back is saturated
ATC_RAT_YAW_D		0	0	s ² /rad	0.000 0.02	Yaw axis rate controller D gain. Compensates for short-term change in desired yaw rate vs actual yaw rate
ATC_RAT_YAW_D_FF		0	0	s ² /rad	0 0.02	FF D Gain which produces an output that is proportional to the rate of change of the target
ATC_RAT_YAW_FF		0	0	s/rad	0 0.5	Yaw axis rate controller feed forward
ATC_RAT_YAW_FLTD		0	20	Hz	5 50	Yaw axis rate controller derivative frequency in Hz
ATC_RAT_YAW_FLTE		2	2.5	Hz	0 20	Yaw axis rate controller error frequency in Hz
ATC_RAT_YAW_FLTT		23	20	Hz	1 50	Yaw axis rate controller target frequency in Hz
ATC_RAT_YAW_I		0.018	0.018	1/rad	0.010 1.0	Yaw axis rate controller I gain. Corrects long-term difference in desired yaw rate vs actual yaw rate
ATC_RAT_YAW_IMAX		0.5	0.5		0 1	Yaw axis rate controller I gain maximum. Constrains the maximum that the I term will output

Figure 38: ArduCopter's Parameter Configuration Interface

Stabilization Parameters

The stabilization performance of a quadcopter depends heavily on the tuning of its attitude and rate controllers. All these parameters had to be carefully adjusted to achieve stable flight.

INS_GYRO_FILTER

This parameter controls the gyroscope filtering. Setting `INS_GYRO_FILTER` to **0** significantly reduced vibrations on the quadcopter. In our case, the onboard gyroscope consistently reported non-existent vibrations, causing the controller to generate unnecessary corrective actions that introduced additional oscillations so in our case this filter was disabled.

ATC_RAT_RLL_P/I/D and ATC_RAT_PIT_P/I/D

These parameters configure the inner rate-control PID loops for roll and pitch. The proportional gain (P) is particularly sensitive:

- Values that are too high cause oscillations.
- Values that are too low make the aircraft sluggish and unresponsive.

The final values used were **P = 0.135**, **I = 0.135**, **D = 0.0036** for both roll and pitch axes.

ATC_ANG_RLL_P and ATC_ANG_PIT_P

These parameters configure the outer attitude-control loop. The controller converts a desired angle into a desired rotational rate, which is then processed by the inner rate controller.

- Higher values increase responsiveness.
- Excessively high values lead to oscillations.
- Low values result in slow attitude corrections.

Both were set to **4.5**.

ATC_RAT_RLL_FF and ATC_RAT_PIT_FF

These feedforward gains determine how aggressively the controller responds to commanded rotational rates. Both were left at **0**, as the standard P/I/D control proved sufficient.

Automatic Flight Parameters

Several parameters were essential for achieving reliable autonomous takeoff and navigation.

TKOFF_THR_MAX

The ArduCopter takeoff algorithm considers takeoff complete only after reaching the altitude specified by this parameter. The default value of **0.9** caused the quadcopter to continue accelerating for approximately four seconds, resulting in excessively aggressive take off behaviour and altitude overshoots of up to 40 m. While simply reducing this value to about **0.4** (40% of maximum throttle) fixed the issue, understanding this behaviour required inspection of the ArduCopter source code.

AUTO_OPTIONS

By default, ArduCopter does not allow arming directly in Auto mode. This safety mechanism assumes that a pilot performs a manual takeoff before switching to autonomous operation.

Since the developed system is intended to take off autonomously, this safeguard had to be disabled. `AUTO_OPTIONS` was set to **3**, a bitmask enabling both arming directly in Auto mode (bit 0) and takeoff without first raising the throttle (bit 1).

WP_NAVALT_MIN

This parameter defines the minimum altitude at which the navigation controller begins correcting horizontal position errors after takeoff.

If configured too high, especially in windy conditions, the quadcopter may drift significantly before position corrections are applied. This can result in unstable takeoffs and landings. The parameter was set to **1 m**, meaning position corrections begin as soon as the quadcopter reaches 1 m altitude.

Failsafe Parameters

The quadcopter must be capable of operating safely even when communication with the ground station is interrupted. Several ArduCopter failsafe mechanisms were therefore configured and tested.

RC Loss (FS_THR_ENABLE)

Gets triggered when the RC transmitter signal is lost. Depending on configuration, the vehicle can:

- Return to Launch (RTL), see “Return to Launch and SmartRTL” in Section 10.2.1
- Land
- Execute SmartRTL, see “Return to Launch and SmartRTL” in Section 10.2.1
- Continue the mission

Since this project does not use an RC transmitter, this failsafe was disabled (**FS_THR_ENABLE = 0**).

Ground Station Loss (FS_GCS_ENABLE)

Triggered when no MAVLink heartbeat is received from the ground station for longer than **FS_GCS_TIMEOUT** seconds.

Possible actions include:

- Return to Launch (RTL), see “Return to Launch and SmartRTL” in Section 10.2.1
- Land
- Execute SmartRTL, see “Return to Launch and SmartRTL” in Section 10.2.1
- Continue the mission

This failsafe was enabled (**FS_GCS_ENABLE = 1**) with a timeout of 2 seconds (**FS_GCS_TIMEOUT = 2**), causing the quadcopter to perform RTL if the companion app link drops.

Battery Failsafe (BATT_FS_LOW_ACT, BATT_FS_CRT_ACT)

Triggered when battery voltage or remaining capacity falls below configured thresholds.

Possible actions include:

- Land
- Return to Launch (RTL), see “Return to Launch and SmartRTL” in Section 10.2.1
- Execute SmartRTL, see “Return to Launch and SmartRTL” in Section 10.2.1

The low-battery threshold was set to 14.4 V (**BATT_LOW_VOLT = 14.4**) with RTL as the action (**BATT_FS_LOW_ACT = 2**). The critical-battery failsafe was disabled (**BATT_FS_CRT_ACT = 0**), as the low-battery RTL is expected to return the quadcopter before the critical threshold is reached.

EKF Failsafe (FS_EKF_ACTION)

This parameter defines the action to take when the Flight controller can no longer provide a sufficiently accurate estimate of vehicle position or attitude.

It was configured to land immediately (**FS_EKF_ACTION = 1**), as a degraded position estimate makes RTL unreliable.

These examples represent only a small subset of the available ArduCopter parameters but illustrate the extensive configuration effort required to achieve stable and reliable autonomous operation.

10.2 Telemetry

The constraints and capabilities of the MAVLink API, explained in this chapter, directly dictate how the user interface must be structured and what interactions it can offer. Understanding the communication layer first makes the UI design decisions that follow considerably easier to follow.

Unlike traditional remote controllers that broadcast commands over dedicated radio frequencies, a smartphone does not have the hardware to transmit on the 2.4 GHz or 915 MHz bands used by standard quadcopter radios. Instead, the companion app uses the smartphone's built-in WiFi antenna. The quadcopter carries a small WiFi access point, and the phone connects to this network. All command and telemetry data flows over this WiFi link as ordinary IP traffic, replacing the radio link found in conventional setups.

This section is crucial for understanding the reasons behind the design of the companion app's user interface and control logic. The capabilities and limitations of the MAVLink protocol directly influence how the app can interact with the quadcopter, and thus shaped many of the design decisions described in the following sections.

10.2.1 MAVLink Introduction

MAVLink (Micro Air Vehicle Link) is a lightweight binary communication protocol used for communication between drones, ground control stations, and onboard computers. Developed in 2009 by Lorenz Meier, it has become the de-facto standard in the drone industry and is supported by the major autopilot firmwares, including ArduPilot and PX4. The protocol defines a set of standardised message types, each identified by a unique numeric message ID, that are exchanged continuously between the app and the flight controller. Each message is encoded into a compact binary packet consisting of a fixed header, a variable-length payload, and a two-byte CRC checksum, and can be carried over any byte-stream or datagram transport. In this project, UDP is used.

Before turning to the technical details of how these packets are built and parsed, the following subsections introduce the most important flight concepts that the companion app relies on.

Arming and Disarming

Arming activates the quadcopter's motors and flight controller, transitioning it from a safe, inert state into one that is ready for takeoff. Until the quadcopter is armed, the motors cannot spin and no flight commands are executed. **Disarming** is the reverse operation: it cuts power to the motors and returns the quadcopter to its inert state, normally after a safe landing. Disarming can also be issued as a last-resort emergency command while the quadcopter is airborne, immediately cutting motor power; while this results in an uncontrolled descent, it prevents the quadcopter from continuing to fly when all other control has been lost.

Geofence

A geofence is a virtual boundary that constrains where the quadcopter is allowed to fly. In this project it is configured as a cylindrical safety volume defined by a maximum horizontal radius and a maximum altitude around the home position. If the quadcopter reaches the boundary, the flight controller automatically takes a configured action such as holding at the fence or returning to launch, to keep it within a safe operating area.

Return to Launch and SmartRTL

Return to Launch (RTL) is a mode in which the quadcopter autonomously navigates back to its recorded home position, the location where it was armed, and lands. It can be triggered manually by the pilot or automatically by a failsafe event such as a loss of the control link. **SmartRTL** (Smart Return to Launch) is a variant that, instead of flying a direct straight-line route home, retraces the path the quadcopter actually flew. By following this recorded breadcrumb trail of the outbound flight, it returns along a route that is already known to be obstacle-free.

10.2.2 MAVLink over UDP

Communication between the app and ArduPilot uses **MAVLink**, transported over a UDP socket bound to port 14550. This wireless link carries both the telemetry received from the quadcopter (FR-Q2) and the control commands sent to it, such as arming, mode switches, and manual control inputs (FR-Q3). Implementing quadcopter control at this level is not a typical task for web engineers. The protocol operates considerably closer to the hardware than the HTTP APIs that are common in web development, and introduces a set of unfamiliar challenges that are described in the section below.

10.2.3 The Challenges of Asynchronous Communication

The MAVLink communication model introduces paradigms that are fundamentally different from the request–response model most developers are familiar with.

Commands to the quadcopter are sent as **COMMAND_LONG** packets, and the quadcopter acknowledges each one asynchronously with a **COMMAND_ACK** packet. There is no guarantee of delivery or ordering, and the app must correlate incoming acknowledgements with previously sent commands by matching the command identifier. If no acknowledgement arrives within a given timeout, the command must be retransmitted. This fire-and-forget pattern with asynchronous confirmation, combined with retransmission logic and the need to detect duplicate or out-of-order packets, represents a significant departure from the synchronous programming model.

10.2.4 MAVLink Message Encoding and Decoding

MAVLink is a lightweight binary communication protocol. Before transmission, a MAVLink message is encoded into a compact binary packet. Upon reception, the packet is decoded back into the original message structure.

Encoding

```
\services\mavlink\mavlink-udp-core.ts
```

When a MAVLink message is created, the sender first populates the fields defined for the specific message type. For example, a **COMMAND_LONG** message contains parameters, target system information, and the command identifier.

It is important to note that the fields are not serialized in the same order as they appear in the MAVLink XML definition. Instead, MAVLink reorders fields according to their native data size:

- **8-byte fields** (uint64_t, int64_t, double)
- **4-byte fields** (uint32_t, int32_t, float)
- **2-byte fields** (uint16_t, int16_t)
- **1-byte fields** (uint8_t, int8_t, char)

Fields of equal size retain their relative order from the XML definition. This reordering minimizes memory alignment issues and allows efficient serialization on embedded processors. The resulting wire layout is therefore determined by field size, not by the order shown in the MAVLink documentation.

After reordering, all multi-byte values are serialized in little-endian byte order meaning the least significant byte is transmitted first. For example, the 32-bit value 0x12345678 is transmitted over the wire as:

78 56 34 12

The serialized payload is then framed inside a MAVLink v2 packet together with header fields: the magic byte (0xFD) signaling the start of a MAVLink packet, payload length, incompatibility and compatibility flags, sequence number, system ID, component ID, message ID (3 bytes), and a two-byte CRC. In the companion app, this encoding and framing is implemented by the `buildMavlinkV2Packet` function in `\services\mavlink-udp-core.ts`, which assembles the header, payload, and CRC into the final packet.

Figure 39 shows the byte-level layout of a MAVLink packet, contrasting the older v1 format with the v2 format used in this project.

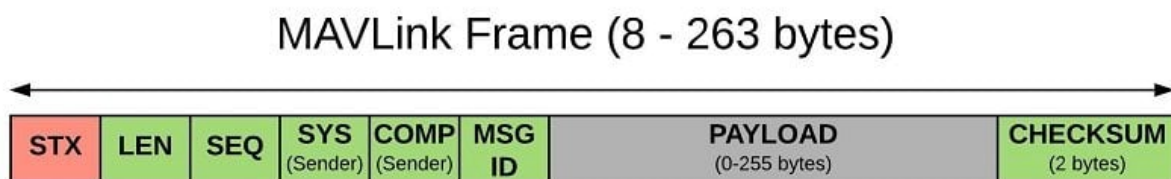


Figure 39: Byte-Level Architecture of a MAVLink Packet

Decoding

```
\services\mavlink\mavlink-parser.ts
```

The receiver continuously scans the incoming byte stream for the MAVLink start-of-frame marker. Once a complete packet has been received, the checksum is verified to ensure data integrity. The message ID contained in the packet header identifies the message type and therefore the expected payload structure. Using the shared MAVLink message definition, the receiver reads each field from the correct byte offset, converts multi-byte values from little-endian back into native TypeScript types, and exposes the decoded message to the application layer. This decoding is implemented by the `parseMavlink` function which scans for the start-of-frame marker and validates the packet with the `decodePayload` function that reconstructs the individual message fields, both located in `\services\mavlink-parser.ts`.

Importance of Field Reordering

A common misconception is that MAVLink parameters must be transmitted in exactly the same order as listed in the XML documentation. In reality, compatibility depends on the serialized payload layout produced by the MAVLink protocol rules. As long as both sender and receiver apply the same field-size-based ordering and little-endian encoding, the message is decoded correctly. The MAVLink checksum calculation (CRC_EXTRA) also incorporates this reordered layout, so any mismatch between message definitions on sender and receiver is detected at the checksum stage rather than silently producing wrong values.

10.2.5 MAVLink Message Types

MAVLink defines three broad categories of messages that the app exchanges with the flight controller:

- **Telemetry messages** are sent by the quadcopter periodically and carry status information such as GPS position (`GLOBAL_POSITION_INT`), attitude (`ATTITUDE`), battery state (`SYS_STATUS`), wind estimate (`WIND`), and home position (`HOME_POSITION`). The app subscribes to these streams and updates its state stores and UI accordingly.
- **Command messages** (`COMMAND_LONG`) are sent by the app to instruct the quadcopter to perform actions like arming the motors, changing the flight mode, or setting the update rate for a telemetry message. The quadcopter acknowledges each command with a `COMMAND_ACK` message indicating whether it was accepted, temporarily rejected, or is being processed.
- **Mission protocol messages** form a stateful handshake sequence used to upload the list of waypoints the quadcopter can fly to autonomously. The app initiates the upload by sending a `MISSION_COUNT` packet that tells the quadcopter how many items the mission contains. The quadcopter then requests each item individually by sending a `MISSION_REQUEST_INT` for each index. The app responds to every request with the corresponding `MISSION_ITEM_INT` packet. Once the quadcopter has received all items, it concludes the handshake with a single `MISSION_ACK` confirming that the full mission has been stored successfully.

10.2.6 Key MAVLink Commands and Messages

The following messages are the most important ones exchanged between the companion app and the flight controller.

MANUAL_CONTROL

`MANUAL_CONTROL` is the primary message used for manual flight. Unlike most control commands, it is not wrapped in a `COMMAND_LONG` frame but uses its own dedicated message type (ID 69) and is sent continuously at a fixed rate of 40 Hz. The message carries four signed 16-bit integers, each representing one axis of pilot input in the range -1000 to $+1000$:

- **Pitch:** forward and backward tilt
- **Roll:** left and right tilt
- **Throttle:** upward and downward thrust
- **Yaw:** rotation around the vertical axis

The way these axis values are produced from the on-screen joysticks is described in Section 10.3.2. Each time the pilot moves a virtual joystick, the corresponding axis value is updated in a module-level state object. A 40 Hz timer then continuously transmits the current axis state to the quadcopter, regardless of whether the values have changed. This ensures that ArduPilot always receives a fresh control signal and can detect a loss of input if packets stop arriving. When manual control is exited, a final packet with all axes set to zero is sent to signal a neutral state.

No `COMMAND_ACK` is expected in response to `MANUAL_CONTROL` packets as the flight controller consumes them as a continuous stream.

MAV_CMD_COMPONENT_ARM_DISARM

Arming and disarming the motors is performed by sending a `COMMAND_LONG` with command ID 400 (`MAV_CMD_COMPONENT_ARM_DISARM`). The first parameter selects the operation: 1.0 arms the quadcopter and 0.0 disarms it. For a normal arm, the second parameter is set to 0. For a force disarm, intended as an emergency measure to cut motor power immediately while the quadcopter is airborne, the second parameter is set to the magic value 21196, which instructs ArduPilot to bypass the usual safety interlocks and disarm unconditionally.

The flight controller responds to both commands with a `COMMAND_ACK`. A successful arm transitions ArduPilot into an active state where the motors can spin; a successful disarm cuts all motor power.

MAV_CMD_DO_SET_MODE

Switching the flight mode of the quadcopter is done by sending a `COMMAND_LONG` with command ID 176 (`MAV_CMD_DO_SET_MODE`). The first parameter must be set to `MAV_MODE_FLAG_CUSTOM_MODE_ENABLED` (128) to indicate that the desired mode is expressed as an ArduCopter-specific custom mode number. The second parameter then carries the mode:

- **Stabilize** (0): the pilot controls attitude directly; the quadcopter does not hold position
- **Position Hold** (16): the flight controller uses GPS to hold the current horizontal position; the pilot can still apply inputs to move the quadcopter
- **Auto** (3): the flight controller executes the uploaded mission autonomously

The companion app uses mode switches to transition between manual flight (Position Hold), autonomous mission execution (Auto), and the initial connection state (Stabilize).

MAV_CMD_NAV_RETURN_TO_LAUNCH

The Return-to-Launch command instructs the flight controller to autonomously navigate the quadcopter back to its recorded home position and land. The quadcopter defines its home position as its current location when it's being armed. The RTL command is sent as a `COMMAND_LONG` with all navigation parameters set to zero, since no target coordinates are needed. In the companion app, RTL is exposed as one of the three emergency buttons in the app header (see Section 10.3.1) and serves as a rapid recovery measure in case the pilot loses orientation or the quadcopter drifts out of range.

HEARTBEAT

In addition to the explicit control commands, the app continuously transmits a `HEARTBEAT` message (ID 0) to the quadcopter once per second. The heartbeat identifies the sender as a ground control station (type `MAV_TYPE_GCS`) and announces that the link is alive. This is not merely informational: the flight controller is configured so that if no heartbeat is received for two seconds, it assumes the connection to the controller has been lost and autonomously returns to its recorded home position and lands there. Sending a steady one-second heartbeat therefore keeps the connection healthy and prevents the quadcopter from prematurely entering this failsafe. Until the quadcopter's network address has been learned from its incoming packets, the heartbeat is broadcast to all candidate addresses; once the address is known, it is sent directly to that target.

PARAM_SET

The geofence explained in “Geofence” in Section 10.2.1 is configured by writing individual ArduPilot parameters to the flight controller using the PARAM_SET message (ID 23). Unlike a COMMAND_LONG, which triggers an action, PARAM_SET directly modifies a named configuration parameter on the autopilot. When the pilot enables and configures the geofence, the app sends following sequence of PARAM_SET messages:

- FENCE_ENABLE
- FENCE_TYPE
- FENCE_RADIUS
- FENCE_ALT_MAX
- FENCE_ACTION

Each carries the parameter name and its new value. The flight controller stores these values persistently and applies them to constrain the quadcopter to a defined cylindrical safety volume during flight.

10.2.7 Telemetry Stream Configuration

```
\services\mavlink\mavlink-udp.ts
```

Rather than relying on the older and deprecated REQUEST_DATA_STREAM mechanism, the app configures the quadcopter’s telemetry output by sending individual MAV_CMD_SET_MESSAGE_INTERVAL (command 511) commands. Each command instructs the flight controller to stream a specific message type at a defined interval in microseconds. This allows fine-grained control over which data is received and at what frequency, avoiding unnecessary network load from messages that the app does not need.

Figure 40 shows the messages that the app requests along with their configured update frequencies.

```
81  const STREAM_INTERVALS: ReadonlyArray<{ msgId: number; intervalUs: number; label: string }> = [
82    { msgId: MavlinkMessageId.SYS_STATUS,   intervalUs: 500_000, label: 'SYS_STATUS (battery, sensors, CPU load @ 2 Hz)' },
83    { msgId: MavlinkMessageId.GPS_RAW_INT,   intervalUs: 1_000_000, label: 'GPS_RAW_INT (fix quality, satellites @ 1 Hz)' },
84    { msgId: MavlinkMessageId.SCALED_IMU,    intervalUs: 250_000, label: 'SCALED_IMU (accelerometer, gyroscope @ 4 Hz)' },
85    { msgId: MavlinkMessageId.ATTITUDE,     intervalUs: 250_000, label: 'ATTITUDE (roll, pitch, yaw @ 4 Hz)' },
86    { msgId: MavlinkMessageId.GLOBAL_POSITION_INT, intervalUs: 1_000_000, label: 'GLOBAL_POSITION_INT (lat/lon/alt/heading @ 1 Hz)' },
87    { msgId: MavlinkMessageId.VFR_HUD,      intervalUs: 250_000, label: 'VFR_HUD (airspeed, groundspeed, altitude, climb @ 4 Hz)' },
88    { msgId: MavlinkMessageId.WIND,         intervalUs: 100_000, label: 'WIND (speed, direction, vertical component @ 10 Hz)' },
89    { msgId: MavlinkMessageId.HOME_POSITION, intervalUs: 25_000_000, label: 'HOME_POSITION (home lat/lon/alt @ 0.04 Hz)' },
90    { msgId: MavlinkMessageId.MISSION_CURRENT, intervalUs: 100_000, label: 'MISSION_CURRENT (active waypoint index @ 2 Hz)' },
91  ];
```

Figure 40: Telemetry Messages configured via MAV_CMD_SET_MESSAGE_INTERVAL and their respective Update Frequencies

The app tracks acknowledgements for each SET_MESSAGE_INTERVAL command sent and logs a confirmation once all intervals have been accepted by the flight controller. If the quadcopter disconnects and reconnects, the full set of interval commands is automatically re-sent so that the telemetry stream is always correctly configured for the current session. The requested messages and their intervals are defined in the STREAM_INTERVALS array and sent by the sendSetMessageIntervals method, both located in \services\mavlink-udp.ts.

10.2.8 Safety: Byte Order and Its Consequences

A critical challenge during implementation was ensuring that every value sent to the quadcopter is encoded with the correct byte order. ArduPilot, the open-source autopilot firmware running on the flight controller, is written in C and C++. C programs interpret raw memory directly: if the app sends a 4-byte float value with the bytes in the wrong order, the flight controller silently reconstructs a completely different number than intended.

An incorrectly encoded value could cause the quadcopter to arm unexpectedly, switch to an unintended flight mode, or execute a navigation command with corrupted coordinates. In the worst case, a corrupted command packet could cause the quadcopter to crash or behave in a way that endangers bystanders. Bugs of this type are particularly difficult to diagnose because they produce no error messages and the flight controller simply acts on plausible-looking but meaningless numbers.

To prevent this, all multi-byte values in the app are written and read using JavaScript's `DataView` API with the `littleEndian` flag explicitly set to `true` . The resulting byte sequences were verified against reference implementations and packet captures using Mission Planner before being used in actual flights.

10.3 Companion App

The following sections detail the implementation of the companion app, which serves as the central control interface for the system.

10.3.1 Header: Emergency Buttons

```
\components\layout\app-header.tsx
```

```
\services\mavlink\mavlink-udp.ts
```

In flight, the header bar of the app hosts three emergency buttons. From left to right, they allow the pilot to emergency-disarm the quadcopter (FR-A23), to return to the takeoff location (FR-A20), and to hold the current position (FR-A21). The position hold button simply switches the quadcopter into a manual mode that holds its position, as explained in the `MAV_CMD_DO_SET_MODE` description in Section 10.2.6.

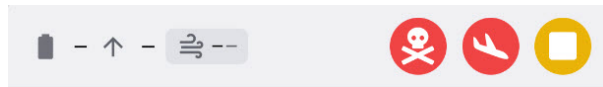


Figure 41: Screenshot of the Emergency Buttons in the App Header

The buttons set the corresponding state in the quadcopter status store, as elaborated in Section 10.3.5 and send the corresponding command through the MAVLink interface in `\services\mavlink\mavlink-udp.ts`. The logic for this is implemented in the `\components\app-header.tsx` file, which defines the header component and the functions that are called when the buttons are pressed.

10.3.2 Manual Control

```
\components\flight\manual-flight.tsx
```

The app allows for manual control of the quadcopter, fulfilling FR-A1. To enable manual control, the pilot can open the “Flight” tab in the app and press the “Manual Control” button. This switches the quadcopter into Position Hold mode.

The control interface in the app consists of virtual joysticks and buttons that send commands to the quadcopter. The joysticks are implemented using `react-native-axis-pad`, which provides a simple and responsive way to capture joystick input.

The output of the two virtual joysticks is directly mapped to the four axes of the `MANUAL_CONTROL` message (Section 10.2.6). The left joystick controls throttle and yaw, while the right joystick controls pitch and roll. Each axis is encoded as a signed 16-bit integer in the range -1000 to $+1000$ and sent to the quadcopter at a fixed rate of 40 Hz, giving the flight controller a continuous and up-to-date picture of the pilot’s intent. Moving a joystick immediately updates the corresponding axis value, which is then picked up by the next transmission cycle.

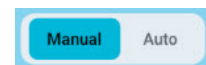


Figure 42: Manual Control Toggle Button

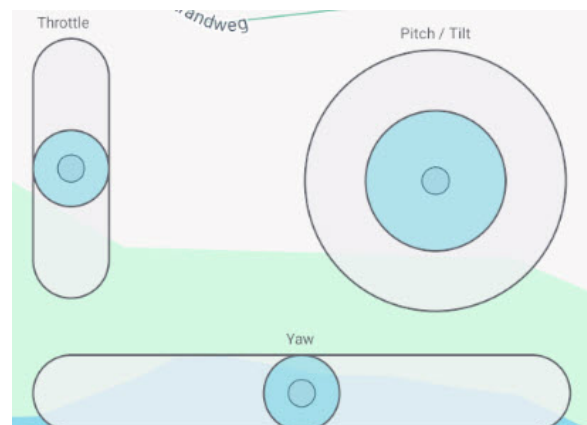


Figure 43: Screenshot of the Manual Control Interface in the App

Challenges of Touchscreen-Based Flight Control

Steering a quadcopter with a smartphone is considerably more demanding than flying with a conventional dedicated remote controller, and this difference has a direct influence on the design choices of the manual control interface.

A physical remote controller features joysticks that provide tactile resistance proportional to deflection. This resistance gives the pilot continuous haptic feedback about the current stick position, allowing fine adjustments to be made without looking at the controller at all. The pilot who uses a smartphone must split their visual attention between watching the quadcopter in the air and monitoring the touchscreen for joystick position.

For this project, the limitations of touchscreen control are acceptable because the primary use case is automatic flight.

10.3.3 Automatic Control

```
\components\flight\auto-flight.tsx
```

The automatic control feature can be accessed on the “Flight” tab of the app by pressing the “Auto Flight” button. This enables the Auto flight mode, which allows the pilot to create a route by adding waypoints to the map and then upload this route to the quadcopter for autonomous execution.

The process and implementation of the required steps is detailed in the following sections.

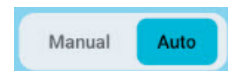


Figure 44: Auto Flight Toggle Button

10.3.4 Flight Planning

```
\app\flight.tsx
```

Before starting a flight, the pilot can plan a route by creating waypoints to the map and adding them to the route on the flight page of the app. This corresponds to the requirements FR-A2, FR-A3, FR-A7, FR-A8, FR-A9, FR-A10, FR-A11, FR-A27, and FR-A12.

Editing waypoints and the route is possible only while the quadcopter is on the ground, to prevent discrepancies between the route that the quadcopter is currently flying and the route that is being edited in the app.

The runtime view of creating a waypoint and a route is detailed in Section 9.6. This section here focuses on the implementation details.

Waypoints

```
\components\map\waypoint-marker.tsx
```

```
\components\map\marker-form.tsx
```

```
\stores\waypoints.store.ts
```

A waypoint is a fixed location in space that the quadcopter can be commanded to fly to. Each waypoint is defined by its latitude, longitude, and altitude, as well as additional parameters such as the type of measurement to be taken at that point and the time to remain there.

In the UI, waypoints can be created by pressing on the map, and edited by pressing on an existing waypoint. Doing so opens a form (Figure 45), in which the user can enter a name, the height, the type of the measurement, and the time to remain measuring at the waypoint. In case the type “Altitude Range” is selected, the user can enter multiple heights by specifying the minimum and maximum altitude, as well as the altitude step. Closing the form either by pressing the “X” button or by swiping down, saves the waypoint to the database and updates the waypoint marker on the map.

Figure 45: Screenshot of the Waypoint Creation Form in the App
(Single Altitude on the left, Altitude Range on the right)

While in flight, the pilot can also create waypoints at the current position of the quadcopter by pressing the button depicted in Figure 46.



Figure 46: Button to Create a Waypoint at the Current Position of the Quadcopter

The button sits on the top right of the map and can be clicked if the current position of the quadcopter is known, which is the case as long as the quadcopter has a GPS lock and is transmitting its position. When clicked, a waypoint is created at the exact location and height of the quadcopter, without the waypoint form being opened. This is the only way to create a waypoint in flight.

Route Creation

```
\components\map\default-route-polyline.tsx
```

```
\stores\routes.store.ts
```

By clicking “Add to Route” in the waypoint form (as seen in Figure 45), the waypoint is added to the current route. Even though the database schema allows for multiple routes, the app currently only supports the creation of a single route, to reduce complexity of the user interface and development effort.

Once multiple waypoints have been added to the route, a magenta line connecting the selected waypoints appears on the map, indicating the planned flight path, as shown in Figure 47. The indicated route remains, regardless of the position of the quadcopter, until one or more waypoints are removed from the route. It is also important to mention, that to simplify the user interface, a waypoint can only be added to the route once.



Figure 47: Screenshot of a Route

If a route has been created and the quadcopter is connected, the route can be uploaded to the quadcopter by pressing the “Upload Mission” button. This starts the creation of all required mission items on the quadcopter, including the takeoff and landing commands, as well as all required helper waypoints. Helper waypoints are required in the following cases:

Cruising Altitude Helper Waypoints

```
\services\data\route-builder.service.ts
```

In accordance with FR-A12 Section 7.1.2, the quadcopter should cruise between two waypoints at the altitude of the higher waypoint. However, since ArduPilot flies the quadcopter in a direct line between the two waypoints, a helper waypoint is required at the cruising altitude, to ensure that the quadcopter either first climbs to the cruising altitude and then flies horizontally, or first flies horizontally and then descends to the altitude of the lower waypoint, depending on the position of the waypoints.

Figure 48 illustrates the need for a helper waypoint in this case. The blue line indicates the planned flight path without a helper waypoint, while the magenta line indicates the flight path with a helper waypoint at the cruising altitude.

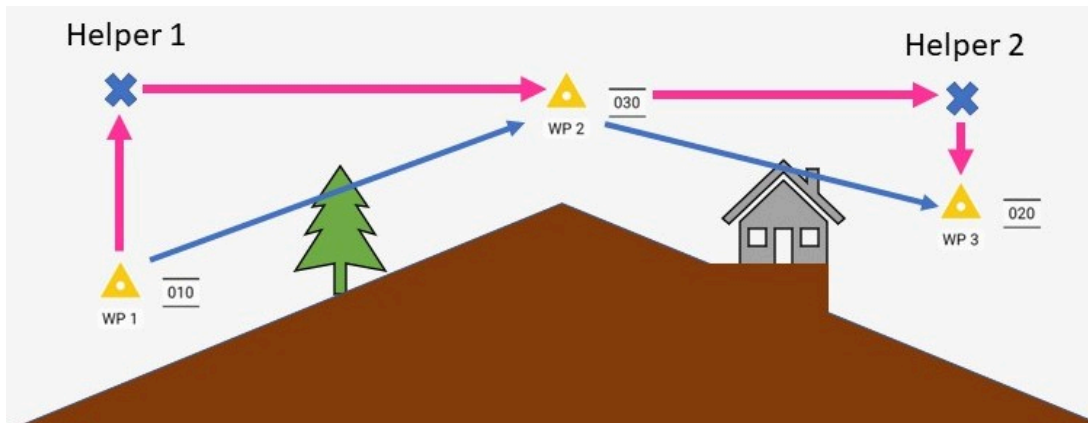


Figure 48: Illustration of the need for a Helper Waypoint to ensure cruising at the higher Altitude

Altitude Range Helper Waypoints

```
\services\data\route-builder.service.ts
```

If a waypoint with the type “Altitude Range” is added to the route, a helper waypoint is required for each altitude step, so the quadcopter measures at the specified altitudes. The first waypoint is at the minimum altitude of the waypoint, and each subsequent helper waypoint is at the altitude of the previous waypoint plus the specified altitude step, until the maximum altitude is reached. Figure 49 illustrates this process for a waypoint with a minimum altitude of 10 m, a maximum altitude of 20 m, and an altitude step of 5 m.

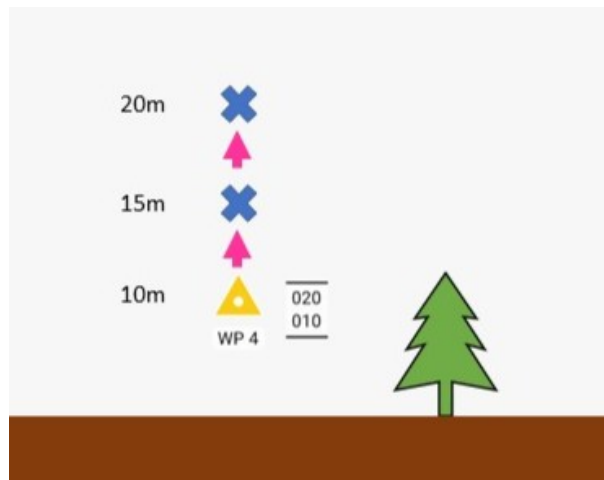


Figure 49: Illustration of the need for Helper Waypoints for Altitude Range Waypoints

Takeoff and Landing Helper Waypoints

```
\services\mavlink\mavlink-mission.ts
```

ArduPilot requires a takeoff and landing command at the beginning and end of the route. The takeoff command is added as the first mission item and receives the altitude of the first waypoint as parameter, so that the quadcopter takes off to the correct altitude. The landing command is added as the last mission item and receives the coordinates of the intended landing location, which currently is the location of the last waypoint.

Route Upload

`\services\flight\auto-flight.service.ts`

`\services\mavlink\mavlink-udp-mission.ts`

Once the “Upload Mission” button is pressed, the app creates the required mission items as detailed above, and uploads them to the quadcopter using `\services\mavlink\mavlink-udp-mission.ts`.

MAVLink’s mission protocol uses NAV_WAYPOINT mission items, so each waypoint first needs to be converted into the corresponding NAV_WAYPOINT item. Each of these carries a loiter time parameter that defines how long the quadcopter hovers at the waypoint before proceeding to the next one. The hold duration is encoded directly in the waypoint itself. For measurement waypoints, the hold duration is set to the time specified by the user in the waypoint form. For transit waypoints, the hold duration is set to zero, so that the quadcopter flies directly to the next waypoint without hovering.

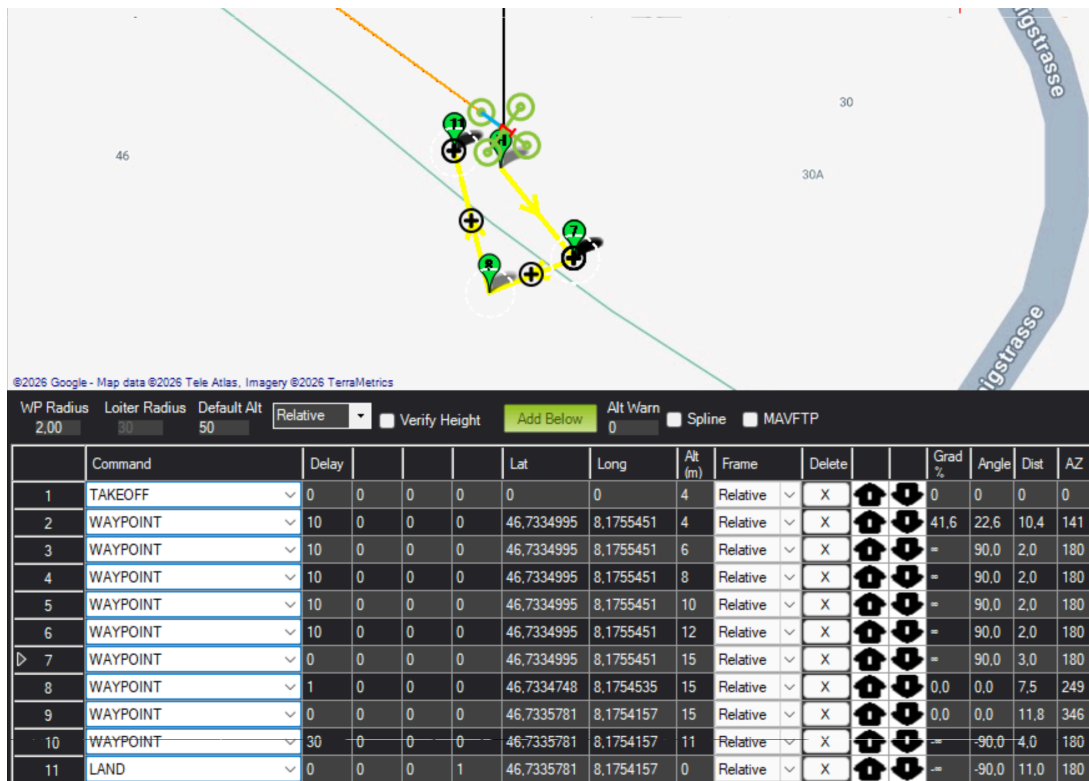


Figure 50: Mission Waypoints as displayed in Mission Planner after a successful Upload

Once the pilot has placed the desired waypoints on the map and the route has been built and converted into the corresponding sequence of MAVLink mission items, the mission is uploaded to the flight controller via the mission protocol handshake (Section 10.2.5). After a successful upload, no further interaction is required from the app: The pilot sets the quadcopter to Auto mode (see “MAV_CMD_DO_SET_MODE” in Section 10.2.6) and presses the start mission button which arms the quadcopter (see “MAV_CMD_COMPONENT_ARM_DISARM” in Section 10.2.6), at which point ArduPilot executes the mission autonomously from takeoff to landing.

10.3.5 State Management

```
\stores\drone-status.store.ts
```

As defined in Section 9.7.3, the app uses Zustand for state management. In that section, the corresponding state machine diagram can also be found. The states are defined in corresponding enum types, which are used throughout the app to indicate the current state of the quadcopter. Zustand allows for easy management of state transitions and ensures a consistent state across the app. The state management logic is implemented in the `\stores\drone-status.store.ts` file, which defines the states and the functions to transition between them.

Complex Transitions

Certain transitions should only be possible through specific functions, as there is some logic involved. An example of this is the transition from “onground” to any of the airborne states, which can have different outcomes depending on the selected flight mode and planning state. Therefore, this transition is only to be carried out through the `takeOff()` function, whose logic is shown in the following diagram as an example:

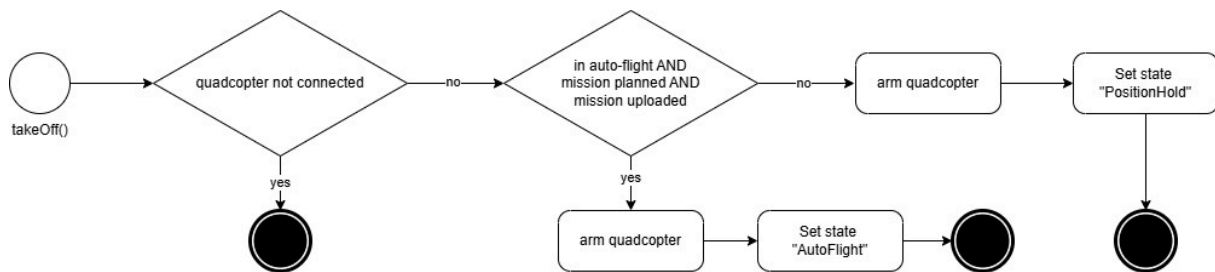


Figure 51: Diagram of the `takeOff()` Function's Logic

Other functions, that contain complex state logic include:

- `setConnected()`
- `setArmed()`
- `setAirborneState()`
- `setIsInAutoFlight()`

Collaboration

One additional benefit of using Zustand during the development process is that it eases collaboration between developers. The state management logic serves as a boundary between the control logic and the UI logic. If the UI needs to send an input to the quadcopter, it can simply alter the state through the defined functions, to which the control logic will react accordingly. Vice versa, changes by the quadcopter, such as the arrival at a measuring waypoint or a change of the arming state, can be easily reflected in the UI by simply changing the corresponding state variable.

This separation of concerns allows for parallel development of the control logic and the UI, as well as simplifies debugging and testing of the app, as the app's functionality can be tested by altering the state manually, without the need of the quadcopter being present.

10.3.6 Data Retrieval and Storage

```
\services\data\database.service.ts
```

All structured data in the companion app is persisted in a single SQLite database (`drowian.db`) via the `DatabaseService` class, which acts as a singleton with lazy initialisation. On the first call to `DatabaseService.getInstance()`, the database is opened using `expo-sqlite` in WAL (Write-Ahead Logging) mode and the schema is created if it does not yet exist. The schema comprises the tables described in Section 9.5. The `DatabaseService` exposes typed CRUD methods for each entity, which are called directly by the Zustand stores rather than through an additional repository layer.

Store–Database Interaction

Each data-related Zustand store calls `DatabaseService.getInstance()` directly, performs the required database operation, and then updates its own in-memory state to keep the UI consistent without requiring a full reload. All mutations are wrapped in a shared `withLoading` utility, which manages the `isLoading` and `error` fields uniformly across all stores. For example, adding a waypoint inserts it into the database, receives the generated ID, and appends the new entry to the store’s in-memory list without reloading the full table from disk.

The `routes.store.ts` additionally carries a side effect: whenever the route list changes, it synchronises the `isMissionPlanned` flag in `drone-status.store.ts`, so the flight control logic always reflects whether a plannable mission exists.

Active Measurement Recording

```
\stores\active-measurement.store.ts
```

```
\hooks\flight\use-measurement-sampler.ts
```

During a flight, wind samples are recorded by the `useMeasurementSampler` hook. When the quadcopter reaches a waypoint, `startMeasurement()` is called, which creates a `Flight` record in the database if none exists yet for the current session, and a `Measurement` record for the active waypoint. From then on, `recordSample()` is called every second, persisting each `WindSample` to the database and adding it to the in-memory `windSamples` array in the store. Writing to the database immediately prevents data loss if the app is interrupted, while keeping the samples in memory allows the live visualisation to display up-to-date data without requiring additional database reads.

Historical Data Retrieval

```
\services\data\history.service.ts
```

Reading flight history for the history view (FR-A17, FR-A18) is handled by the stateless `HistoryService`. It provides methods to retrieve the dates on which flights were recorded, the flights for a given date, and the full set of measurements and wind samples for one or multiple flights. By isolating this aggregation logic in a dedicated service rather than loading all data into global state, the history view can query exactly the data it needs in a single call.

10.3.7 Wind and Turbulence Calculation

```
\utils\wind.utils.ts
```

The data retrieved from the quadcopter contains the sensor data, as well as the calculated wind data, as described in Section 8.1. From this data, the average wind speed and direction, as well as the level of turbulence is calculated to enable the fulfillment of requirements FR-Q1, FR-A15, FR-A16, FR-A17, and FR-A18.

Average Wind Conditions

To display the average wind conditions to the user, the average speed and direction is calculated from the samples sent by the quadcopter in high frequency in flight, or from the data stored locally for historical data.

Calculating the average wind speed from the selected samples is straight forward (function `avg(...)`), while calculating the average wind direction requires a circular mean due to it's circular nature (function `circularMean(...)`), comparable to the circular variance described in Section 8.2.2.

Turbulence Calculation

The methods and formulas deployed to calculate turbulence are described in Section 8.2. The described formulas are implemented in the function `calculateTurbulenceIntensity()` and it's helper functions.

For testing and refinement purposes, several parameters were made configurable at runtime. The following table lists these parameters, their purpose, and the values that were found to produce good results during testing:

Table 5: Turbulence Calculation Parameters

Parameter	Purpose	Optimal Value
windAnalysisWindowMs	Length of the sliding window used to calculate turbulence intensity during live measurements.	3000 ms
tiWindSpeedWeight	The weight of the TI measured using the wind speed variance in the weighted average calculation.	0.5
tiWindDirectionWeight	The weight that the circular variance calculated from the wind direction changes is given.	0.2
tiAccelerometerWeight	The weight of the turbulence magnitude calculated from the accelerometer variance.	0.3
accelerometerLowPassAlpha	The parameter α used in the low pass filter applied to the turbulence magnitude calculation.	0.05
tiModerateThreshold	The TI threshold above which turbulence is classified as "Moderate".	0.10
tiSevereThreshold	The TI threshold above which turbulence is classified as "Severe".	0.20
tiVerySevereThreshold	The TI threshold above which turbulence is classified as "Very Severe".	0.50

10.3.8 Data Visualization

The measured wind data is displayed in the app in various ways. Live data is displayed in the app header while historical data is visualized in the history page.

Live Data Visualization

```
\components\layout\app-header.tsx
```

The app header, described in Section 9.10, permanently shows the quadcopter’s battery voltage, relative altitude, and current wind speed and direction during flight, fulfilling FR-A15. The battery voltage is colour-coded based on the charge level to give the pilot an immediate indication of the battery status.

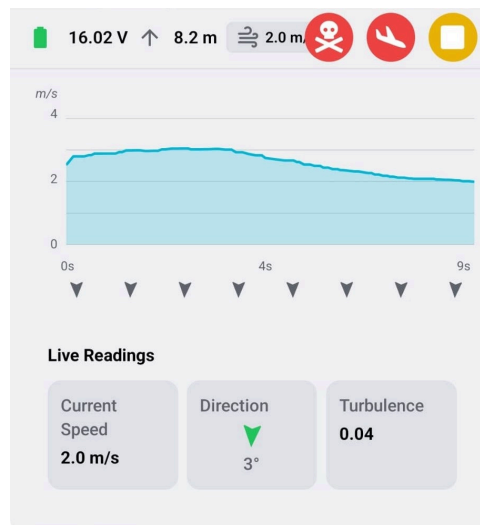


Figure 52: Screenshot of the App Header showing Live Wind Data

Tapping the header expands an overlay that provides a more detailed view of the current wind conditions. The overlay contains the same wind speed graph component used in the history detail card, but populated with live data. The `useFlightWindGraphRecorder` hook subscribes to the telemetry store and appends a new data point to the `useFlightWindGraphStore` on each incoming wind packet. Recording starts with the first non-zero wind reading and resets when a new flight session begins. Below the graph, a “Live Readings” section displays three cards: the current wind speed, the current wind direction shown as a colour-coded arrow, and the current turbulence intensity with a numeric value. The arrow and turbulence card use the same colour scheme as the wind barbs on the map, so the pilot can quickly cross-reference the two views.

Historical Data Visualization

```
\app\history.tsx
```

Wind data from previous flights is visualized on the history page according to FR-A17. A menu at the bottom allows to search for a flight by day and time, or to show the data of all flights for a given day. The selector is implemented in `\components\history-bottom-bar.tsx`. The main page displays a map with the waypoints and overlaying wind barbs, showing the estimations for the selected flight(s).

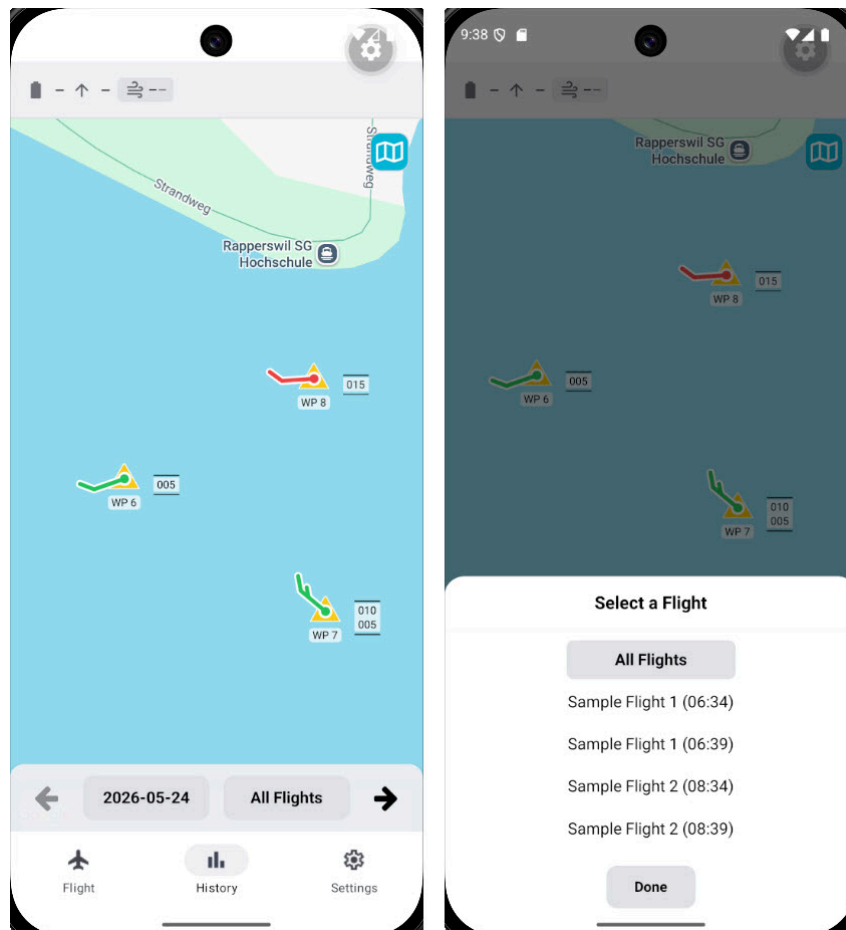


Figure 53: Screenshot of the History Page

Wind Barbs

```
\components\wind\windbarb.tsx
```

```
\components\wind\wind-barb-marker.tsx
```

The wind barbs are implemented as custom SVG component (`\components\wind\windbarb.tsx`), which take the wind speed, wind direction, and turbulence intensity as input and render the corresponding wind barb. The color of the barb indicates the turbulence intensity, with the color gradient defined in the app's theme. The categorization is done through the central `\utils\wind.utils.ts` file, which contains the logic to calculate and classify the turbulence intensity based on the measured data.

Detail View

```
\components\wind\wind-sample-card.tsx
```

When clicking on a wind barb, a detail card opens, showing the exact values and a graph of the wind data over time for the corresponding measurement point (FR-A18). The card is implemented in `\components\wind-sample\*`.

The layout varies depending on the type of waypoint, for which the card is shown. For single altitude waypoints, the card shows a graph over the entire time the quadcopter was at that waypoint, and the average values below that. For altitude range waypoints, the card shows the vertically stacked average estimations for the different altitudes, as well as the total average below that. Clicking on an altitude entry in the stack opens a detailed view for that altitude, showing the graph of the estimations at that altitude over time.

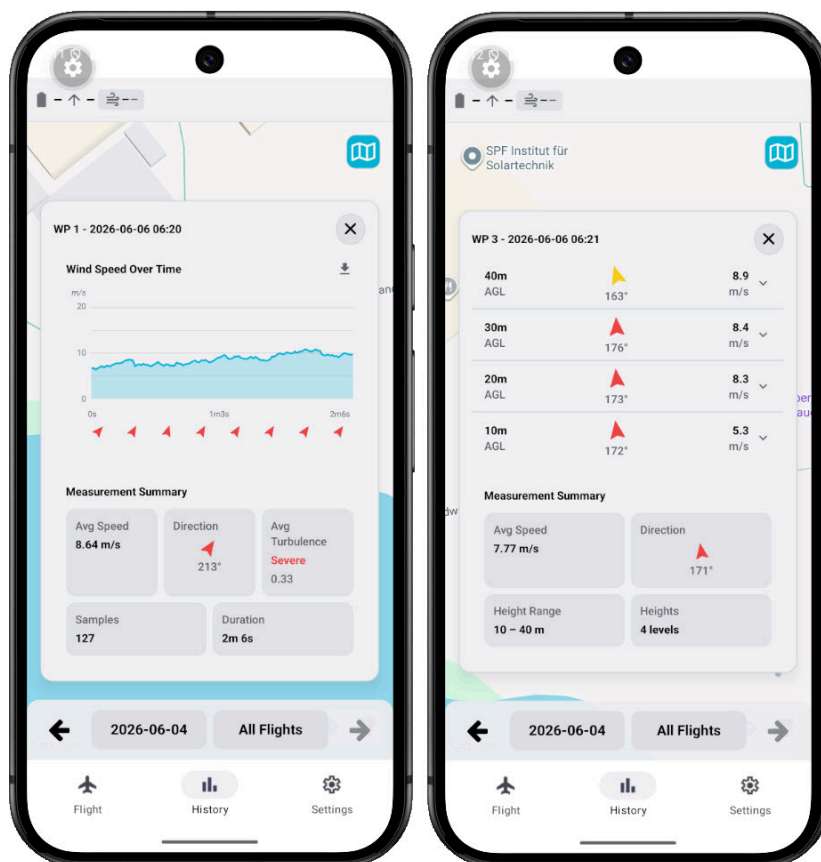


Figure 54: Screenshot of the Wind Barb Detail Card. Single Height Waypoint on the left, Altitude Range Waypoint on the right

```
\components\wind\wind-speed-chart.tsx
```

The wind speed graph is implemented using `victory-native`, a charting library for React Native built on Skia for high-performance rendering. The chart renders a cartesian line chart with a semi-transparent area fill below the wind speed curve. The X-axis shows the elapsed time from the start of the measurement, while the Y-axis shows the wind speed in m/s. Tapping on the chart activates an interactive crosshair that follows the touch position and displays the exact wind speed, elapsed time, and turbulence intensity at that point. Below the chart, evenly-spaced wind direction arrows provide an overview of how the wind direction changed over the course of the measurement, with their colour indicating the turbulence intensity.

10.3.9 Map Integration

```
\app\flight.tsx
```

```
\app\history.tsx
```

```
\app.json
```

The map library used in the app is `react-native-maps`, which provides a powerful and flexible integration of Google Maps into the app. This implementation allows for all the required map features to be implemented, as detailed in the below sections.

As this library uses Google Maps, it requires a Google Maps API key, and since the API is used in an app, the free demo key is not sufficient. Therefore, a paid plan is required, which can incur additional costs if the free usage limits are exceeded. To ensure that no exorbitant costs arise, a disposable credit card with a spending limit of CHF 1.00 is used for the API key.

The key can be found and replaced in the `app.json` file under **`androidGoogleMapsApiKey`**.

Map Providers

```
\components\map\osm-tile-layer.tsx
```

The base map of `react-native-maps` is the standard Google Maps map. It is possible to set the map mode to “satellite”, and to add custom tile providers to overlay different maps on the grid. Both of these possibilities are used in the app, so the user can switch between Google Maps, satellite and Open Street Maps (OSM) in order to choose the map style that best suits their environment and provides the best situational awareness.

As tile provider, the basic OSM tile server was used:

<https://tile.openstreetmap.de/{z}/{x}/{y}.png>

Whereby the `{z}`, `{x}` and `{y}` parameters are automatically replaced by the library with the corresponding zoom level and coordinates.

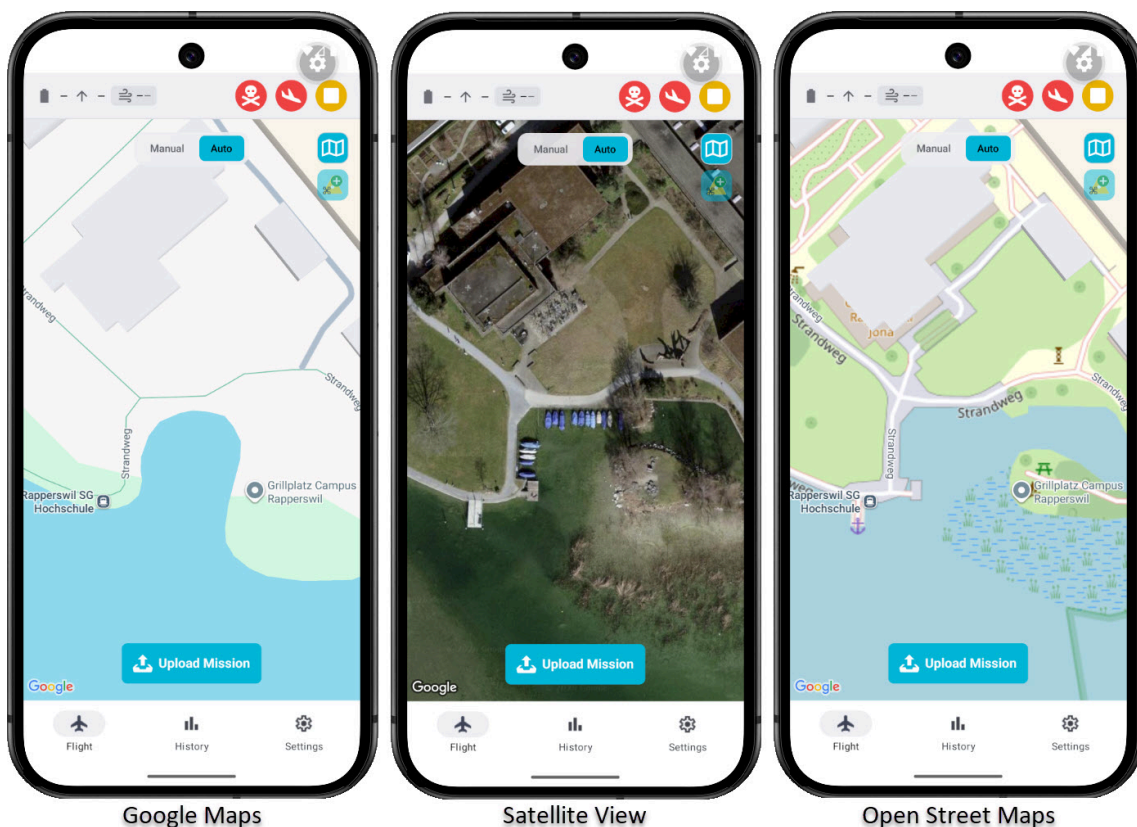


Figure 55: Screenshots of the different Map Provider Selections in the App

Caching and Offline Use

When connected to the quadcopter, the app cannot establish internet connection, and therefore cannot load new map tiles. To make the app available offline, react-native-maps caches the map tiles that have already been loaded, so that the map is still available during flight. This requires that the user opens the app and navigates to the area of operation before connecting to the quadcopter, so that the required map tiles are loaded and cached.

This caching mechanism works automatically with the standard Google Maps and satellite views, but needs to be explicitly enabled for the custom Open Street Maps tile provider. This is done by setting the `offlineMode` property of the `UrlTile` component to `true`, which ensures that the OSM tiles are also cached and available offline.

An option to pre-cache map tiles for a specific area and zoom levels or shipping the app with pre-cached tiles for the area of Switzerland was considered, but ultimately decided against. Both options would have a significant impact in the file size of the app, and introduce additional complexity in form of a custom local tile server, which might negatively affect the app's performance and reliability. The automatic caching has been tested to work reliably, as long as the user loads the required map area before flight.

Map Markers

```
\components\map\default-route-polyline.tsx
```

```
\components\map\drone-marker.tsx
```

```
\components\map\waypoint-map-marker.tsx
```

```
\components\map\waypoint-marker.tsx
```

```
\components\wind\wind-barb-marker.tsx
```

Google Maps markers are used to indicate the position of the quadcopter, the waypoints, the wind barbs, and the planned flight path. As react-native-maps supports the various kinds of overlays which Google Maps provides, the quadcopter, waypoint and wind barbs are implemented as `Marker` components, while a `Polyline` component is used to indicate the planned flight path. To show the quadcopter's approximate range, according to FR-A13, a `Circle` component is displayed around the users present position.

10.3.10 Settings

```
\app\settings.tsx
```

The settings tab in the app allows the user to set basic parameters and see the status of the quadcopter's sensors. Additionally, the tab hosts settings used during development and the logs generated during runtime.

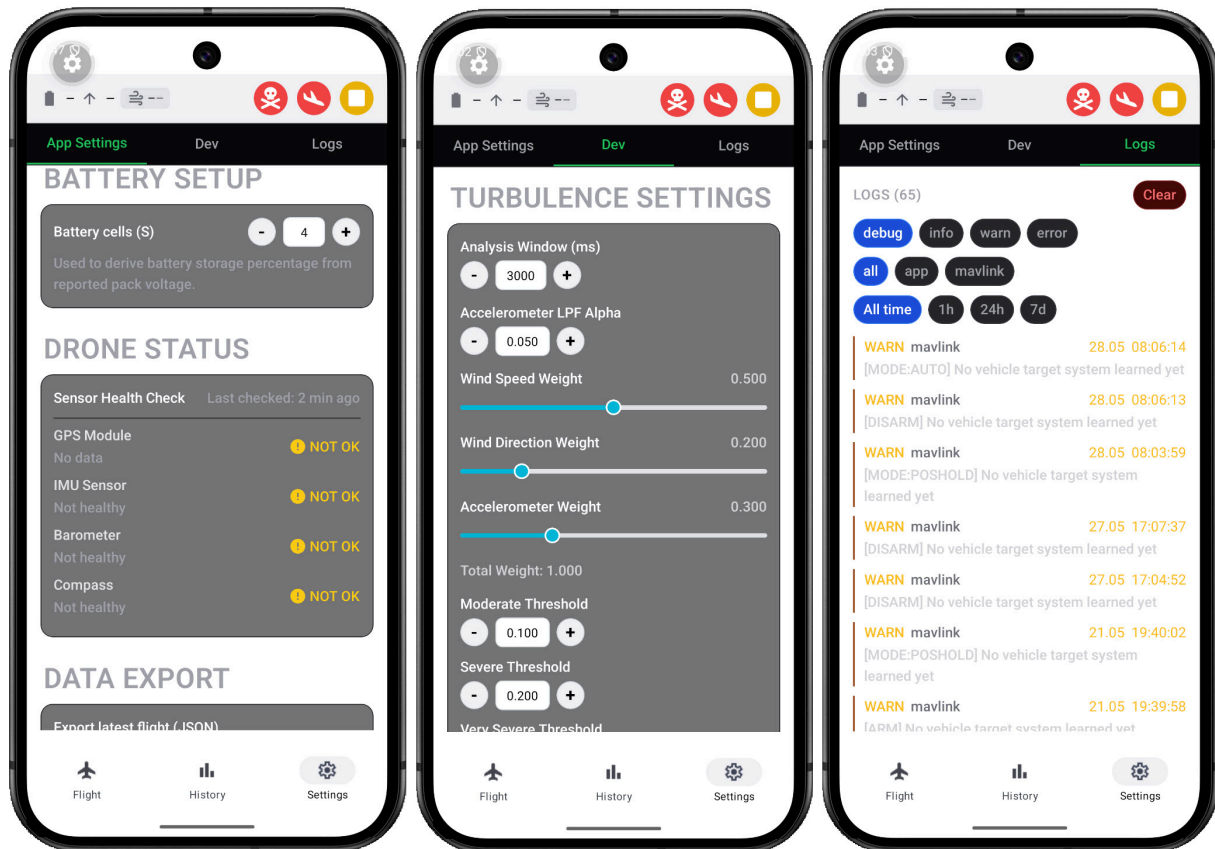


Figure 56: Screenshot of the Settings Tab

App Settings

```
\app\settings-tabs\app\AppSettingsTab.tsx
```

The app settings include:

- The number of battery cells of the quadcopters battery, defining its maximum voltage and used to calculate the remaining flight time based on the current battery voltage
- The option to export the last flight's data as a JSON file, which can be used for further analysis and testing.
- The health status of the quadcopter's sensors can be found here (FR-A25).

Dev Settings

```
\app\settings-tabs\dev\DevTab.tsx
```

The dev settings include:

- The configurable parameters for the turbulence calculation
- An interface to manually set the quadcopter states for testing purposes
- Buttons to create and remove test data in the local database.

This tab is only intended to be used during development and testing, and should be removed before publishing the app. As explained in [Appendix A.2](#), the tab is still visible in the submitted version of the app to allow for easier further development.

Logs

```
\app\settings-tabs\logs\LogsTab.tsx
```

This section shows the logs generated by the app or the quadcopter during runtime. They can be filtered by category, level, and time, and can be used for debugging and testing purposes, thereby fulfilling the non-functional requirement FR-A26 and NFR4.

Chapter 11

Results

The overall project was successful in producing a portable quadcopter based system that can be used to get localized wind and turbulence information at paragliding launch sites.

11.1 Quadcopter System

The quadcopter fulfils all critical requirements defined in Section 7.1.1. Particularly noteworthy is its portability: as specified by FR-Q3, no physical remote controller is required to operate the quadcopter. The quadcopter is controlled entirely through the companion app over WiFi, meaning a paraglider only needs to carry their smartphone, which is a significant practical advantage when hiking to a remote launch site. At 190 g, the quadcopter also satisfies NFR1, which requires a maximum takeoff weight of less than 250 g, keeping it in the A1 category under Swiss quadcopter regulations. NFR2 was also fulfilled, as the flight-control software ArduCopter is completely open source [42].

To protect the electronics against moisture, all components were treated with silicone spray, though how effective this proves in practice remains to be seen. Transporting the quadcopter is straightforward: the same latch that secures the battery during flight can be used to clip the quadcopter directly onto a standard hiking backpack, as shown in Figure 57. Storing the propellers separately would be impractical, but mounting the fully assembled quadcopter externally avoids this entirely and keeps the pack compact.

Overall, the quadcopter represents a very well-suited platform for paragliders: it is lightweight, requires no dedicated radio equipment, and is fully capable of conducting safe, autonomous wind measurements while the pilot is on the ground preparing for flight.



Figure 57: The Quadcopter clipped onto a standard Hiking Backpack using the Battery Latch

11.2 Companion App

The developed companion app allows the pilot to define waypoints and a route, start and stop the quadcopter's autonomous flight, and receive the estimated wind information directly in the app. Historical data can be viewed in the app and the raw data of the last flight can be exported as a JSON file.

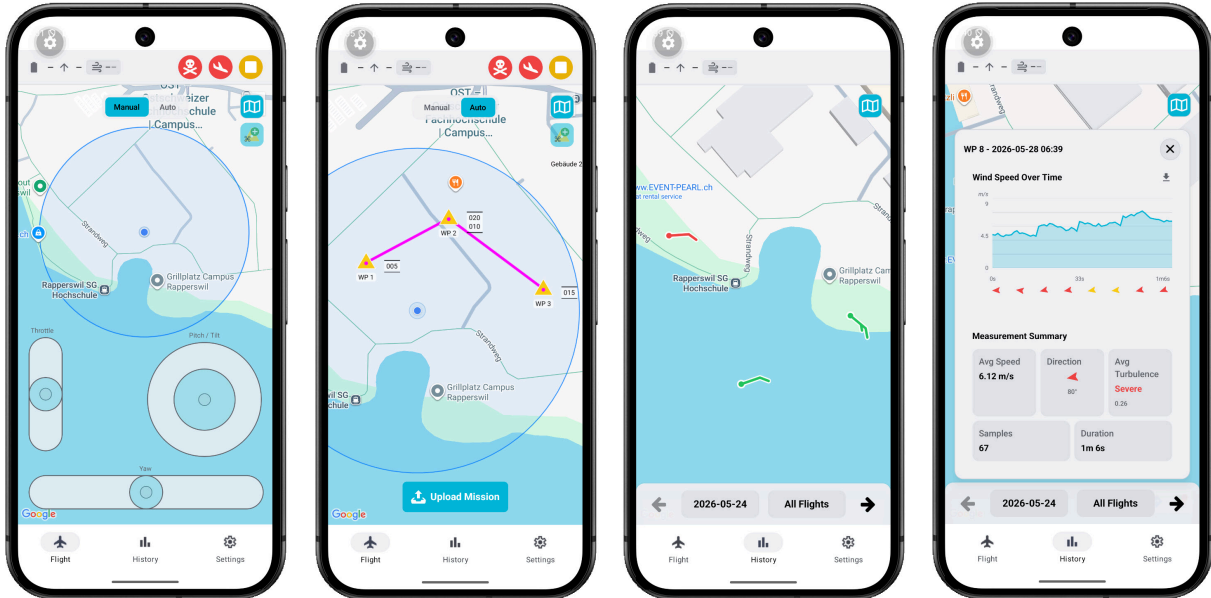


Figure 58: Companion App Screenshots

During development, for technical reasons some requirements had to be adjusted to accommodate for the limitations of quadcopter's sensors and firmware. For example, initially a "Direct To" functionality was planned, that would have allowed the pilot to redirect the quadcopter to a specific waypoint during the flight. However, since it was uncertain if ArduPilot would accept commands to change the flight path during an active mission, this requirement was changed so that the complete flight path has to be planned before taking off. This is not a major issue, as all use cases of the system are still covered with the current functionality of the app, and the "Direct To" functionality can be implemented in a future iteration of the app if desired.

This change also implied, that it is not possible in flight to switch from manual flight mode to automatic flight mode, as the flight path has to be defined before takeoff. Therefore, the "Position Hold" button in the app header, which was supposed to switch the quadcopter to position hold mode, also switches the quadcopter and the app to manual flight mode. As the manual flight mode is not stable enough to be used reliably, this button is not recommended to be used in its current state.

Another necessary change was to the live wind display: it no longer shows the current waypoint. This is because ArduPilot only sends a waypoint notification to the ground station after the quadcopter has already departed from that waypoint and the configured delay period has elapsed, meaning the information would always be one waypoint behind. This implies that for altitude range waypoints, the live wind information does not show an immediate overview of the vertical profile of the wind. However, this vertical profile is still available in the history page after the flight, and the live wind information still gives a good overview of the current wind conditions during the flight, even without the current waypoint information.

Overall, the requirements to the MVP in Section 7.4 were met, with the above adjustments. The results of these tests are summarized in Section 11.2.1.

11.2.1 Requirements Fulfillment

In detail, the following functional requirements were met:

Requirement	Description
FR-A2	Quadcopter Automatic Control The quadcopter can be controlled in automatic flight mode using the app.
FR-A3	Start Mission A mission in automatic flight mode can be started.
FR-A4	End Mission A mission in automatic flight mode can be stopped.
FR-A6	Switch Manual/Automatic Control It is possible to switch between manual and automatic flight mode using the app.
FR-A7	Adding a Waypoint Waypoints can be defined in the app.
FR-A8	Editing a Waypoint Waypoints can be edited in the app (e.g. name, height, and position).
FR-A9	Deleting a Waypoint Waypoints can be deleted in the app.
FR-A10	Planning Single Waypoint Route Routes with one waypoint can be defined in the app.
FR-A11	Planning Multi Waypoint Route Routes with multiple waypoints can be defined in the app.
FR-A27	Planning Altitude Range An altitude range can be defined for a waypoint, for which the quadcopter will fly and measure in defined intervals.
FR-A12	Cruising Altitude Between Waypoints When cruising between two waypoints, the quadcopter will chose the higher altitude of the two waypoints to fly at.
FR-A13	Show Range of Quadcopter on Map The quadcopter's range is shown as a circle on the map around the pilot's position.
FR-A15	Show Live Wind Measurements Live wind information is shown in the app during the flight.
FR-A17	Show Wind Barbs on Map on History Page The estimated wind information is saved and shown as wind barbs on the history map.
FR-A18	Show Detail Page Detailed information about the estimated wind at a certain waypoint for a certain flight can be viewed on the history page.
FR-A19	Data Export The raw data of the last flight can be exported as a JSON file.
FR-A20	Emergency Return Button A button in the app header allows the user to automatically return the quadcopter to the home position and land in an emergency situation.

FR-A22	Automatic Return to Base When Battery Charge Low The quadcopter automatically returns to the home position and lands when the battery charge is low.
FR-A23	Emergency Disarm Button A button in the app header allows the user to disarm the quadcopter in an emergency situation.
FR-A25	Health Check The settings page in the app shows the current health status of the quadcopter's sensors.
FR-A26	Log Display Logs can be viewed in the app, categorized by type and level.

Due to time constraints, the following functional requirements were not or only partially met:

Requirement	Reason
FR-A1	Quadcopter Manual Control The quadcopter can be controlled in manual flight mode using the app, but flight is rather unstable and therefore not recommended. Further development and testing is needed to improve the stability of the quadcopter in manual flight mode.
FR-A14	Show If Route Is Within Range There is no functionality to show if a programmed route is within the quadcopter's range, as this is highly dependent on the quadcopter and the battery in use. The range circle on the map only gives a general indication of the quadcopter's range. This feature, however, was already defined as optional in the requirement specification.
FR-A16	Save Wind Measurement and Show Windbarb on Map The estimated wind information is saved to the app per waypoint in automatic flight mode. However, the wind barbs are only shown on the history map, not during the flight.
FR-A21	Hold Position Button The "Position Hold" button in the app header does switch the quadcopter to position hold mode, but that also means that the quadcopter and the app switch to manual flight mode. As discussed above, the manual flight mode is not stable enough to be used reliably, and in the tests resulted in a rapid descent of the quadcopter.
FR-A24	Units Selection The measurement units of the wind information shown in the app are fixed to m/s and cannot be changed in the settings. This feature was left out due to time constraints, but can easily be implemented in the future.
FR-A28	Calibrate Sensors Calibrating the sensors of the quadcopter was regularly necessary with the prototype quadcopter. With the final quadcopter, however, this was not necessary anymore. Therefore, and because it was an optional requirement to begin with, this feature was left out in the app.

The non-functional requirements regarding the companion app were all met:

Requirement	Description
NFR3	Enterprise Architect A class diagram of the app's architecture is included in Section 9.5. However, due to integration issues with Enterprise Architect and TypeScript, the class diagram was created using plantUML, in agreement with the supervisor.
NFR4	Logging A logging mechanism is implemented in the app, which allows to log messages from the app and the quadcopter, classified by category and level.

Overall, the companion app is functional and fulfills the main requirements. There are still some shortcomings, especially regarding the manual flight mode. These can be addressed in future iterations of the app.

11.3 Wind and Turbulence Estimation

As detailed in Chapter 8, the wind speed estimation was tested using the Windshaper and on location during field tests. The following sections give an overview of the results of these tests and measurements.

11.3.1 Windshaper Testing

On 04.05.2026, the prototype quadcopter was tested using the Windshaper to evaluate the accuracy of the wind speed estimation, as well as the reaction time of the wind estimation to changes in wind speed (turbulence simulation).



Figure 59: Windshaper Testing 04.05.2026

Objectives: Using the prototype quadcopter to test the accuracy of the wind speed estimation, as well as the reaction time of the wind estimation to changes in wind speed (turbulence simulation).

Location: Due to initial issues with the GNSS signal at the OST Techpark that were caused by too many buildings close by and not enough open area, the Windshaper was ultimately moved to the campus in Rapperswil. This allowed for testing on an open field and with adequate GNSS signal.

Procedure: The Windshaper was placed on top of a table, about 1.2 m above ground, on an open field. The quadcopter was flown in front of the Windshaper in a distance between 2-5 m, due to some difficulties with holding the precise position in strong wind conditions. Reference measurements were taken using the portable anemometer after each test flight at the same position the quadcopter was flying. Still, some inaccuracies are to be expected due to the exact positioning of the quadcopter and the anemometer, as well as the environmental wind.

For the laminar wind measurements, the Windshaper was run at constant power settings. To simulate turbulence, the function detailed in Section 8.3.1 was used.

Results: The below table gives an overview over the measurements made during testing. For laminar wind measurements the used power setting of the Windshaper is given, while for the turbulence measurement the parameters used in the turbulence function are given. The reference minimum and maximum values are taken from the anemometer measurements, while the quadcopter minimum and maximum values are taken from the wind estimation of the quadcopter. Measurements below 30% power were influenced too much by the environmental wind, which is why these were omitted from the table.

Table 6: Windshaper Measurements 04.05.2026

Goal	Parameters	Reference Minimum	Quadcopter Minimum	Reference Maximum	Quadcopter Maximum
Laminar Wind	30% Power	3.5 m/s	N/A	4.0 m/s	4.0 m/s
Laminar Wind	50% Power	N/A	N/A	4.5 m/s	5.0 m/s
Laminar Wind	60% Power	5.0 m/s	5.0 m/s	5.8 m/s	5.5 m/s
Laminar Wind	80% Power	6.0 m/s	6.0 m/s	9.0 m/s	9.2 m/s
Laminar Wind	100% Power	8.0 m/s	9.0 m/s	9.3 m/s	9.8 m/s
Turbulence	$A_{\max} = 100\%$ $A_{\min} = 30\%$ $\gamma = 4$ $k = 5.$	6.7 m/s	9.0 m/s	8.2 m/s	13.0 m/s

A general observation for these lower wind speeds was that the quadcopter's wind estimation was generally 1-2 m/s higher than the reference measurement. This makes the quadcopter estimations for lower wind speeds less reliable, however, these wind speeds are less relevant for the intended use case of this project. Therefore, this inaccuracy is accepted, especially since this test was conducted using the prototype quadcopter.

The measurements for the higher wind speeds show a good correlation between the quadcopter's wind estimation and the reference measurement, with a maximum deviation of 1.0 m/s. This maximum deviation occurs at the highest power setting of the Windshaper and for the minimum measured wind speed at that setting, which could be explained by a measurement inaccuracy of the anemometer. For all other measurements, the deviation is 0.5 m/s or less, which is considered to be a good result.

When simulating turbulence, it became apparent that the quadcopter's wind estimation reacts to changes in wind speed with a delay of between 10 and 30 seconds. This delay can be reduced by finetuning the corresponding parameters in ArduPilot, but some delay is still present. Accurate calculation of the turbulence intensity using the variance of the wind speed alone is therefore rather difficult, as the delay leads to an underestimation of turbulence intensity.

11.3.2 Field Testing

On the Weekend from 30.05.2026 to 31.05.2026, the final quadcopter was tested in the field in Meiringen BE and Innertkirchen BE.



Figure 60: Field Testing Meiringen 30.05.2026

Objectives: Testing the overall system in real-world conditions, as well as calibrating the wind speed estimation and turbulence intensity estimation of the final quadcopter using reference measurements.

Location: Meiringen and Innertkirchen, chosen for the prevailing wind conditions due to the surround terrain, and since the region is an active area for paragliding, which is the intended use case for the system.

Innertkirchen in particular was chosen, as it features a known “Talwindrotor”, which is a turbulent area with strong gusts, making it ideal for testing the turbulence intensity estimation.

Procedure: The quadcopter was flown near the reference anemometer, which was mounted on a tripod, to collect reference measurements for the wind speed estimation and turbulence intensity estimation. The drag parameters, as described in Section 8.1.2, was adjusted between flights to optimize the wind speed estimation. For tests that did not directly involve the measuring of the wind speed, but only the integration of the companion app with the quadcopter, the exact flightpath was not bound to the position of the reference anemometer.

The tests on the first day in Meiringen were focused on testing the overall system and calibrating the wind speed estimation, while the tests on the second day in Innertkirchen were focused on testing the turbulence intensity estimation, as the wind conditions in Innertkirchen were expected to be more turbulent than in Meiringen.

Results: The test protocols of the system tests are summarized in Section 11.2. Table 7 below gives an overview of the most important measurement flights that were used to calibrate and verify the wind speed estimation.

Table 7: Field Measurements 30.05.2026 - 31.05.2026

Date / Location	EK3_DRAG_COEF	Reference Minimum	Quadcopter Minimum	Reference Maximum	Quadcopter Maximum
30.05.2026 10:57 UTC Meiringen	0.16	2.0 m/s	3.2 m/s	3.8 m/s	5.9 m/s
30.05.2026 11:09 UTC Meiringen	0.30	1.2 m/s	1.2 m/s	3.0 m/s	2.2 m/s
30.05.2026 11:19 UTC Meiringen	0.30	0.5 m/s	0.1 m/s	4.1 m/s	2.6 m/s
31.05.2026 12:32 UTC Innertkirchen	0.25	1.3 m/s	2.4 m/s	3.7 m/s	2.6 m/s

These are selected results of the flights that were used to tune the drag parameter EK3_DRAG_COEF, which is used in the wind estimation algorithm as described in Section 8.1.2. The column EK3_DRAG_COEF shows the value of this parameter for each flight. Adjusting the parameter directly influenced the wind speed estimation accuracy. Setting the drag parameter EK3_DRAG_COEF to **0.30** yielded the best results, when observing the average difference between the wind speed estimation and the reference measurement, even though the quadcopter was still slightly underestimating the wind speed, as explained in the following analysis.

Meiringen

Meiringen was chosen as the first testing location, as it is located near many paragliding launch sites and the wind conditions at the time were expected to be moderate.

A detailed analysis of the measurement data from the flight in Meiringen at 11:19 (bold row in Table 7) shows the following results for the wind speed estimation:

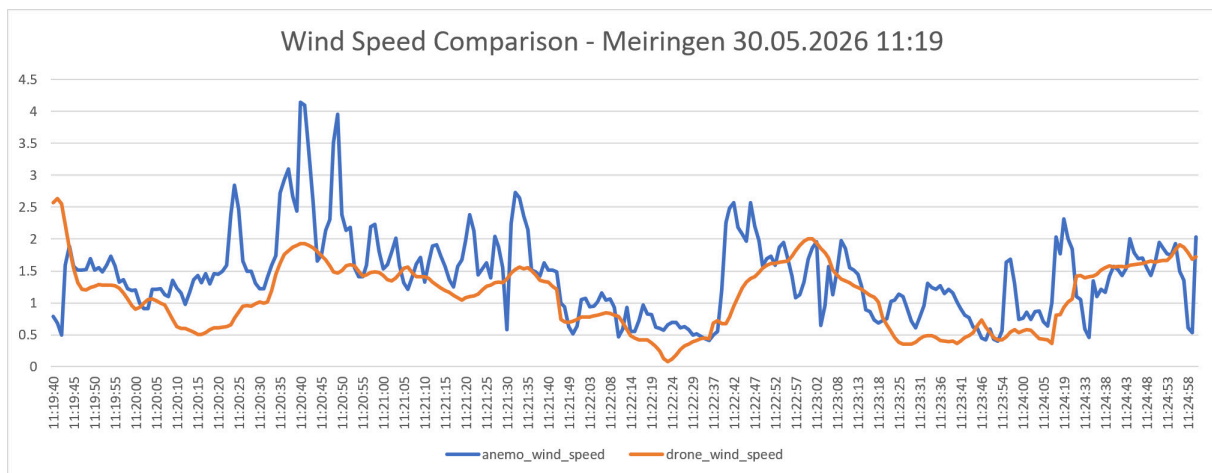


Figure 61: Wind Speed Estimation Analysis Meiringen 30.05.2026 11:19 UTC

The graph shows, that the quadcopter's wind estimation (orange line) follows the general trend of the reference measurement (blue line), but with a latency of several seconds. This leads to an underestimation of the wind speed during gusts, which is also reflected in the turbulence intensity estimation, as can be seen later in this section. This delay is caused by the time it takes for the

quadcopter to adjust its position and attitude to compensate for changes in wind speed, which is a limitation of this method of measurement.

The average deviation between the quadcopter's wind estimation and the reference measurement is 0.307 m/s, which is considered to be a good result for the intended use case of this project.

Figure 62 shows the correlation between the reference measurement and the quadcopter's wind estimation for the same flight:

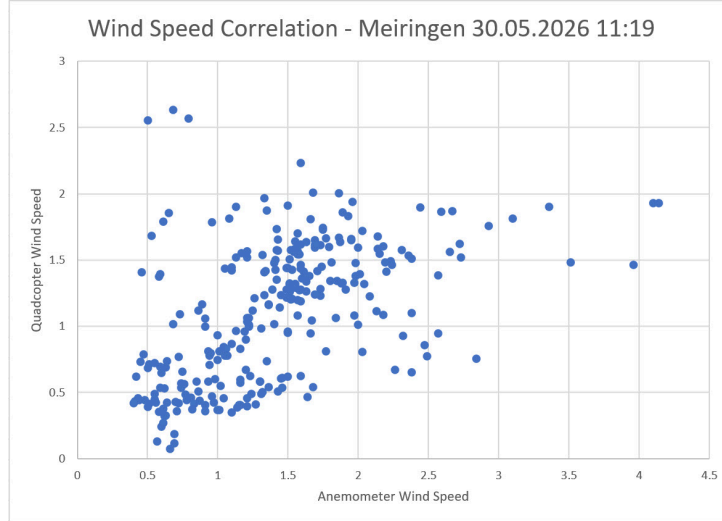


Figure 62: Correlation Reference Measurement and Quadcopter Wind Estimation Meiringen 30.05.2026 11:19 UTC

The correlation coefficient is 0.511, which indicates a moderate positive correlation between the reference measurement and the quadcopter's wind estimation. The correlation is not stronger due to the latency of the quadcopter's wind estimation, however, the general trend is captured well and visible in Figure 62.

Figure 63 shows the turbulence calculations from the quadcopter's sensors: The TI from the wind speed measurements (blue line), the circular variance of the measured wind direction (yellow line), and the turbulence magnitude from the accelerations (dark green line).

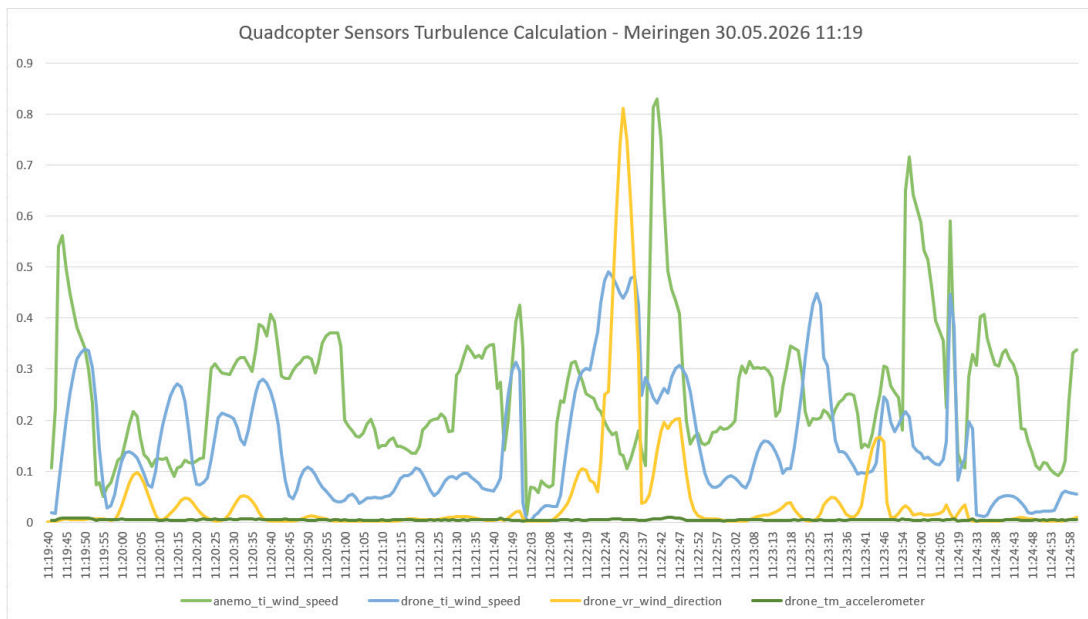


Figure 63: Turbulence Intensity Estimation Analysis Meiringen 30.05.2026 11:19 UTC

As can be seen from Figure 61, around 11:22:37 UTC a rapid change in wind speed occurs according to the reference measurement. The quadcopter's wind estimation reacts to this change with a rather high, but expected, delay of around 20 seconds. This rapid change in wind speed leads to a spike in the turbulence intensity from the reference data (light green line in Figure 63).

The wind direction variance spikes a few seconds before the reference TI, which could explain the spike in reference wind speed, as the anemometer only measures wind in one direction. If the wind changes direction rapidly, before the anemometers orientation could be adapted manually, this would result in a falloff in measured wind speed. The quadcopter's wind speed TI also captures this wind speed variance, but not to the same magnitude. This is an excellent demonstration, why wind speed and direction variance need to be combined to calculate an accurate turbulence estimation. The turbulence magnitude, on the other hand, remains stable at a very low level, showing that the wind direction change did not have a significant impact on the acceleration of the quadcopter. This goes to show that, at least in weakly to moderately turbulent environments, the measured accelerations are not a reliable indicator of turbulence by themselves.

The total weighted average of the three turbulence calculations combined compared to the reference TI is shown in Figure 64:

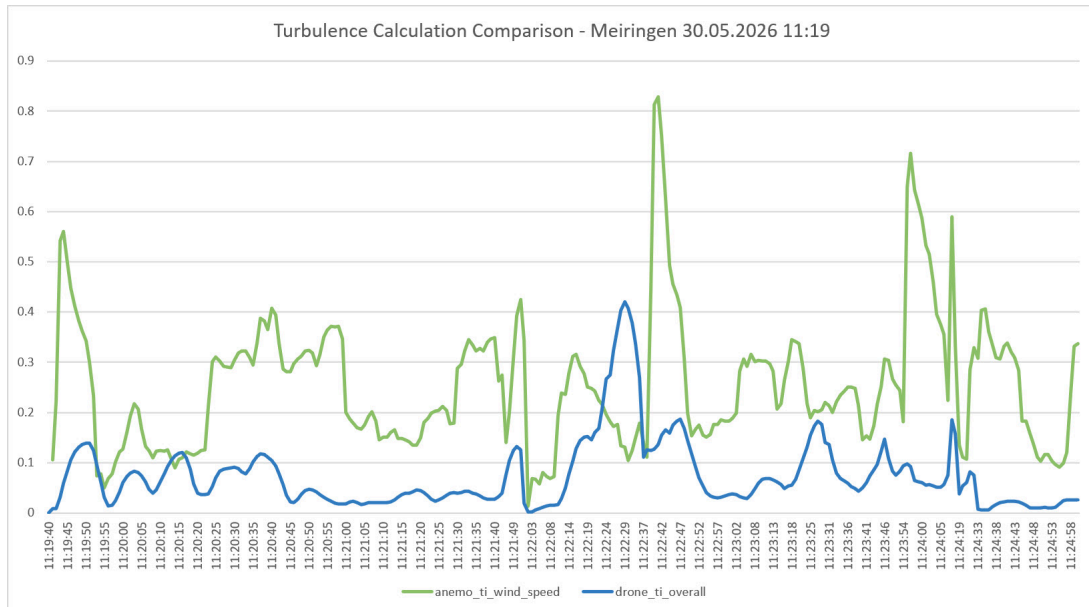


Figure 64: Turbulence Intensity Comparison Meiringen 30.05.2026 11:19 UTC

The calculated TI (blue) follows the trend of the reference TI (green) with a delay of a few seconds. It is, however, evident, that the magnitude of both curves shows a significant difference. As TI is a dimensionless number and needs to be interpreted into a standard classification, this can be easily accounted for by setting the threshold values accordingly. According to our results, we recommend the following categorization:

- Weak turbulence: $TI \in [0; 0.1[$
- Moderate turbulence: $TI \in [0.1; 0.2[$
- Severe turbulence: $TI \in [0.2; 0.5[$
- Very severe turbulence: $TI \in [0.5; \infty[$

However, these values need to be further verified in more heterogeneous and more turbulent environments, as the wind and turbulence conditions were calm to moderate throughout the testing period.

Innertkirchen

Innertkirchen was chosen as a second testing location, as the wind conditions were expected to be more turbulent than in Meiringen, due to the surrounding terrain. However, the wind in Innertkirchen was significantly calmer than expected. This prevented meaningful testing of the TI estimation in more turbulent conditions. The following screenshot shows the history details of an altitude range flight that showed laminar wind, but no turbulence:

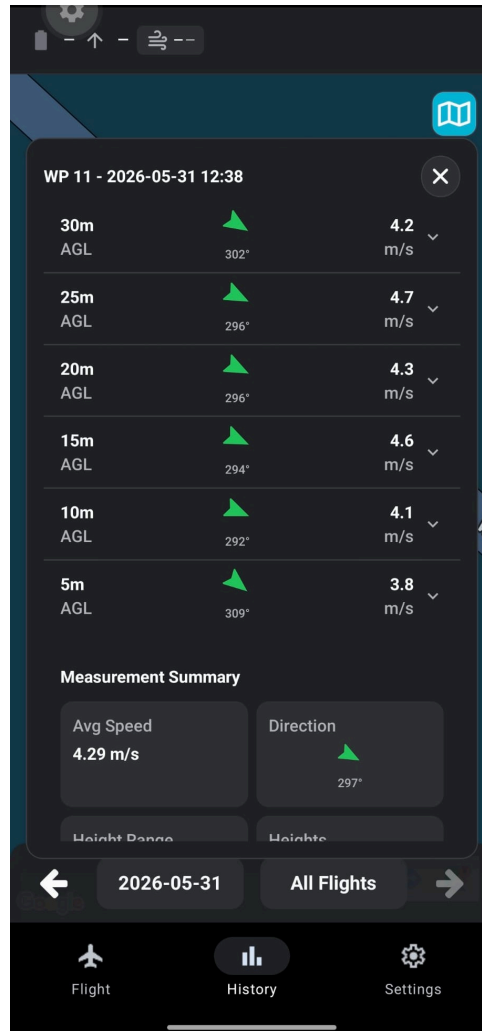


Figure 65: Flight Details Innertkirchen 31.05.2026 12:38

Because of this limitation, the focus of the analysis of the field testing is on the results from the tests in Meiringen.

Chapter 12

Conclusions and Outlook

12.1 Conclusions

The main objective of this thesis was to build on the semester thesis of the autumn term 2025 and implement a fully functional solution for measuring wind and turbulence using a quadcopter. Ultimately, this thesis successfully uses the methods established in the previous semester and builds on the created prototype quadcopter to implement an operational system that can be used in real-world conditions.

Compared to similar existing solutions, the implemented system is more compact and lightweight. By using a sub 250 g quadcopter and controlling it with a mobile app, the system is portable and usable by the targeted user group of paragliding pilots. Regulatory requirements for flying a quadcopter of this category are minimal, which allows the system to be used at most locations without special permissions.

With the achievable range of the WiFi telemetry link, the system can be used to estimate wind speed and direction, as well as low-level turbulence (namely mechanical turbulence, thermal turbulence, and mountain wave turbulence) in the critical takeoff and landing zones of paragliding sites, which are often inaccessible to traditional weather measurement methods.

The companion app implements all MVP features and provides a user-friendly interface for controlling the quadcopter and visualising the estimated wind and turbulence data. Some intended features had to be deferred due to time constraints, but the implemented app provides a solid foundation for future development and improvement.

The algorithms and parameters for wind and turbulence estimation have been implemented and fine tuned to achieve good performance in flight tests. Further testing in more diverse wind and turbulence conditions is required to fully evaluate the performance of the system, but the results so far are promising and demonstrate the feasibility of the developed system.

Overall, the thesis successfully fulfills the objectives defined in the task description and delivers a functional solution for measuring wind and turbulence with a quadcopter, concluding the work that was started in the previous semester and providing a solid foundation for future improvements and development.

12.2 Outlook

The developed system is fully functional. Still, there are several improvements that can be made to enhance the performance, reliability, and usability of the system.

12.2.1 Bluetooth Telemetry via SpeedyBee F405

The SpeedyBee F405 flight controller used in this thesis includes an onboard Bluetooth receiver. At the time of writing, ArduCopter does not support Bluetooth as a telemetry transport, meaning this hardware capability remains unused and a separate dedicated telemetry radio module is required to establish the wireless MAVLink link between the quadcopter and the companion app.

A promising avenue for future improvement would be to implement Bluetooth telemetry support directly in ArduCopter. This would allow MAVLink packets to be transmitted over the built-in

Bluetooth interface of the SpeedyBee F405, eliminating the need for an external telemetry hardware module, reducing both weight and cost of the overall system.

12.2.2 Barometer Protection and Wind Compensation

A simple and well-established mitigation for barometer inaccuracy is to cover the sensor with a small piece of open-cell foam (styrofoam or similar), which acts as a low-pass filter that allows slow pressure equalisation while blocking fast pressure pulses caused by propeller wash and gusts [43]. Adding such foam shielding would improve altitude hold reliability, particularly in gusty conditions.

A further improvement is to compensate for wind-induced barometer error using ArduCopter's built-in wind estimation feature. Airflow around the airframe disturbs the static pressure measured by the barometer, degrading the altitude estimate. Since ArduCopter already estimates the wind vector in flight, this estimate can be used to correct the barometric pressure reading and thereby stabilise the altitude estimate. As noted in Section 9.9, the required EKF drag parameters have not yet been tuned; realising this improvement requires repeated test flights and careful evaluation of the recorded flight logs.

12.2.3 Enclosure and Component Protection

Currently the technical components of the quadcopter are largely exposed; in particular the flight controller and the WiFi module are mounted openly on the frame. Encasing these components with additional framing or a lightweight partial enclosure would protect them from mechanical damage and make the quadcopter considerably easier to transport, as the sensitive electronics and their wiring would no longer be exposed. A complete enclosure is not feasible, however, as several components — in particular the ESCs and telemetry modules — generate significant heat during operation and require constant airflow for cooling. Any enclosure must therefore remain open or vented.

12.2.4 Improved Manual Flight Characteristics

Manual flight using the app is currently possible, but the flight characteristics of the quadcopter are unstable and difficult to control. More development in this area is required to make manual flight more intuitive and easier to control.

12.2.5 Improved Quadcopter / Waypoint Height Calculation

By default, ArduPilot uses the altitude of the home position as the reference for all height calculations. This means that the height AGL of both the quadcopter and waypoints is always relative to the takeoff point. On flat terrain this is not a problem, but on mountainous terrain this can lead to significant discrepancies between the calculated height and the actual height above ground level. Implementing a more accurate height calculation that takes into account the local terrain elevation would improve the accuracy of the height data and make it more useful for pilots.

12.2.6 Show Wind Barbs on the Flight Map

The companion app currently only shows the wind barbs on the history page. In the future it would be desirable to show the wind barbs on the flight page as well for waypoints, that have already been flown.

12.2.7 Improved Wind and Turbulence Estimation

The parameters of the wind speed and turbulence estimation algorithms have been tuned in relatively calm conditions, and the performance of the system in more turbulent conditions is not yet fully evaluated. Further testing in more diverse wind and turbulence conditions is required to fully evaluate the performance of the system and to further fine-tune the estimation algorithms for improved accuracy and reliability.

Bibliography

- [1] “Paraglider Control: Stall, Spin, Collapse!” Accessed: Feb. 27, 2026. [Online]. Available: <https://flybubble.com/blog/paraglider-control-stall-spin-collapse>
- [2] N. US Department of Commerce, “Information about wind barbs.” Accessed: Mar. 03, 2026. [Online]. Available: <https://www.weather.gov/hfo/windbarbinfo>
- [3] “Basic Course of Thermo-Fluid Analysis 18: Chapter 5 Basics of thermo-fluid analyses - 5.9.1 Reynolds-averaged Navier–Stokes (RANS), 5.9.2 Large eddy simulation (LES).” Accessed: Oct. 01, 2025. [Online]. Available: <https://www.cradle-cfd.com/media/column/a91>
- [4] P. F. Lester, *Aviation Weather*, 4th ed. Jeppesen, 2013. Accessed: Oct. 01, 2025. [Online]. Available: <https://www.yumpu.com/en/document/read/66264683/aviation-weather-4th-edition-peter-f-lester>
- [5] “Turbulence - MeteoSwiss.” Accessed: Oct. 01, 2025. [Online]. Available: <https://www.meteoswiss.admin.ch/weather/weather-and-climate-from-a-to-z/turbulence.html>
- [6] “burnair Map - Gleitschirm Fluggebiete.” Accessed: June 01, 2026. [Online]. Available: <https://www.burnair.cloud/>
- [7] I. Leuenberg, P. Lehmann, and A. Schmid, “Expansion of open-source drone software to include wind analysis,” Semester Thesis, 2025. Accessed: Mar. 02, 2026. [Online]. Available: <https://eprints.ost.ch/id/eprint/1356/>
- [8] “Messinstrumente - MeteoSchweiz.” Accessed: Nov. 29, 2025. [Online]. Available: <https://www.meteoschweiz.admin.ch/wetter/messsysteme/bodenstationen/automatisches-messnetz/messinstrumente.html>
- [9] “Wind am Start einschätzen - Wissen / Tipps / News - Chill Out Paragliding AG.” Accessed: Sept. 24, 2025. [Online]. Available: <https://chilloutparagliding.com/infos/news-reports/wind-am-start-einschaetzen/>
- [10] S. Manfreda *et al.*, “On the Use of Unmanned Aerial Systems for Environmental Monitoring,” *Remote Sensing*, vol. 10, no. 4, p. 641, Apr. 2018, doi: 10.3390/rs10040641.
- [11] “Weathervaning and Wind Hold — Copter documentation.” Accessed: Dec. 06, 2025. [Online]. Available: <https://ardupilot.org/copter/docs/weathervaning.html>
- [12] “Making a MAVLink WiFi bridge using the Raspberry Pi — Dev documentation.” Accessed: Feb. 21, 2026. [Online]. Available: <https://ardupilot.org/dev/docs/making-a-mavlink-wifi-bridge-using-the-raspberry-pi.html>
- [13] “Paragliding.” Accessed: Nov. 29, 2025. [Online]. Available: <https://en.wikipedia.org/w/index.php?title=Paragliding>
- [14] “Beginners meteorology | SkyNomad Paragliding School: training, tours and courses in Bulgaria, Europe. Articles.” Accessed: Oct. 22, 2025. [Online]. Available: https://www.skynomad.com/articles/beginners_meteorology.html
- [15] F. O. Soldati, C. Jackowski, and C. A. Schön, “Foot-Launched Flying Sport Fatalities in the Canton of Berne, Switzerland,” *Wilderness & Environmental Medicine*, vol. 33, no. 1, pp. 50–58, Mar. 2022, doi: 10.1016/j.wem.2021.11.004.

- [16] “Wind - MeteoSchweiz.” Accessed: Nov. 29, 2025. [Online]. Available: <https://www.meteoschweiz.admin.ch/wetter/wetter-und-klima-von-a-bis-z/wind.html>
- [17] A. Laing, M. S. Wandishin, J. (. E.). Hart, and M. A. Petty, “Assessment of Graphical Turbulence Guidance, Global (GTG-G), Part 1.” Accessed: Oct. 01, 2025. [Online]. Available: <https://repository.library.noaa.gov/>
- [18] “Turbulenzeinflüsse bei der Messung der Windgeschwindigkeit.” Accessed: Oct. 01, 2025. [Online]. Available: https://www.researchgate.net/publication/242725565_Turbulenzeinflüsse_bei_der_Messung_der_Windgeschwindigkeit
- [19] J. F. Manwell, J. G. McGowan, and A. L. Rogers, *Wind Energy Explained: Theory, Design and Application*, 2nd ed. Wiley, 2009. Accessed: May 10, 2025. [Online]. Available: https://ee.tlu.edu.vn/Portals/0/2018/NLG/Sach_Tieng_Anh.pdf
- [20] “Drohnen.” Accessed: Mar. 05, 2026. [Online]. Available: <https://www.bazl.admin.ch/de/drohnen>
- [21] “Drohnenflüge – Skyguide.” Accessed: Oct. 08, 2025. [Online]. Available: <https://www.skyguide.ch/de/dienstleistungen/drohnenfluege>
- [22] “Flugregeln für Drohnen.” Accessed: Mar. 05, 2026. [Online]. Available: <https://www.bazl.admin.ch/de/flugregeln-drohnen>
- [23] “Geografische Flugeinschränkungen.” Accessed: Mar. 05, 2026. [Online]. Available: <https://www.bazl.admin.ch/de/geografische-flugeinschraenkungen>
- [24] “Unfälle melden.” Accessed: Mar. 05, 2026. [Online]. Available: <https://www.bazl.admin.ch/de/unfaelle-melden>
- [25] “Versicherung für Drohnen unter 250g: Wirklich notwendig?” Accessed: Oct. 08, 2025. [Online]. Available: <https://drohneversichern.ch/unter-250g/>
- [26] “Unser Club betreibt beim Start- / Landeplatz eine Holfuy Wind-Messtation, wie kommt diese auf die burnair Map?” Accessed: Dec. 06, 2025. [Online]. Available: <https://help.burnair.cloud/hc/de/articles/360018234617-Unser-Club-betreibt-beim-Start-Landeplatz-eine-Holfuy-Wind-Messtation-wie-kommt-diese-auf-die-burnair-Map>
- [27] J. Kistner, L. Neuhaus, and N. Wildmann, “High-resolution wind speed measurements with quadcopter uncrewed aerial systems: calibration and verification in a wind tunnel with an active grid,” *Atmospheric Measurement Techniques*, vol. 17, pp. 4941–4955, Aug. 2024, doi: 10.5194/amt-17-4941-2024.
- [28] M. Ghirardelli, S. T. Kral, E. Cheynet, and J. Reuder, “SAMURAI-S: Sonic Anemometer on a Multi-Rotor drone for Atmospheric turbulence Investigation in a Sling load configuration,” *Atmospheric Measurement Techniques*, vol. 18, no. 9, pp. 2103–2124, May 2025, doi: 10.5194/amt-18-2103-2025.
- [29] G. J. Marlton, R. Giles Harrison, K. A. Nicoll, and P. D. Williams, “Note: A balloon-borne accelerometer technique for measuring atmospheric turbulence,” *Review of Scientific Instruments*, vol. 86, no. 1, p. 16109, Jan. 2015, doi: 10.1063/1.4905529.
- [30] ArduPilot, “Wind Estimation in multicopters with Dr Paul Riseborough.” Accessed: Mar. 07, 2026. [Online]. Available: <https://www.youtube.com/watch?v=xVVtvVuZGQE>
- [31] “PosHold Mode — Copter documentation.” Accessed: Mar. 05, 2026. [Online]. Available: <https://ardupilot.org/copter/docs/poshold-mode.html>

Bibliography

- [32] “Windspeed Estimation and Baro Compensation — Copter documentation.” Accessed: Dec. 06, 2025. [Online]. Available: <https://ardupilot.org/copter/docs/airspeed-estimation.html>
- [33] R. O. Weber, “Estimators for the Standard Deviation of Horizontal Wind Direction,” *Journal of Applied Meteorology and Climatology*, vol. 36, no. 10, pp. 1403–1415, Oct. 1997, doi: 10.1175/1520-0450(1997)036<1403:EFTSDO>2.0.CO;2.
- [34] P. S. Farrugia and A. Micallef, “Vectorial statistics for the standard deviation of wind direction,” *Meteorology and Atmospheric Physics*, vol. 129, no. 5, pp. 495–506, Oct. 2017, doi: 10.1007/s00703-016-0483-8.
- [35] L. Ji *et al.*, “Research on Low-Pass filtering noise reduction method for Multi-Rotor Heavy-duty unmanned aerial vehicles based on fast fourier transform,” *Journal of Physics: Conference Series*, vol. 2955, 2025, doi: 10.1088/1742-6596/2955/1/012036.
- [36] “Exponential Moving Average (EMA) Filters.” Accessed: May 25, 2026. [Online]. Available: <https://blog.mbedded.ninja/programming/signal-processing/digital-filters/exponential-moving-average-ema-filter/>
- [37] “Euclidean Norm - an overview | ScienceDirect Topics.” Accessed: May 25, 2026. [Online]. Available: <https://www.sciencedirect.com/topics/engineering/euclidean-norm>
- [38] “WindShaper.” Accessed: June 11, 2026. [Online]. Available: <https://www.windshape.com/products/windshaper>
- [39] A. Lucas, “Arducopter tuning guide.” Accessed: June 09, 2026. [Online]. Available: https://ardupilot.github.io/MethodicConfigurator/TUNING_GUIDE_Heli.html
- [40] Jakub, “Sub 250g Autonomous Drone Platform.” Accessed: June 09, 2026. [Online]. Available: <https://www.thingiverse.com/thing:6699961>
- [41] ST, “Specsheet FC Chip.” Accessed: June 09, 2026. [Online]. Available: <https://www.st.com/en/microcontrollers-microprocessors/stm32f405-415.html>
- [42] A. T. Peter Barker, “Ardupilot.” Accessed: June 09, 2026. [Online]. Available: <https://github.com/ArduPilot/ardupilot>
- [43] Betaflight, “Barometer.” Accessed: June 08, 2026. [Online]. Available: <https://betaflight.com/docs/wiki/guides/current/Barometer>

Appendix

A.1 Tools

The following table lists the various tools that were utilized throughout the project for different purposes.

Table 8: Tools used in the Project

Area	Tools
Literature Research	Google, Google Scholar, Swisscovery, ChatGPT, Zotero
Location Scouting	burnair Map, Google Earth, Google Maps, Windy, MeteoSwiss, map.geo.admin.ch
Data Analysis and Visualization	Copilot, ChatGPT
Diagrams and Modeling	Enterprise Architect, PlantUML, draw.io, Desmos, Figma
Translation	DeepL
Coding	Visual Studio Code, Claude, Github Copilot
Text creation, text optimization, spelling and grammar checking	Typst, Copilot
Collaboration and Projectmanagement	Teams, Azure DevOps, Youtrack, 7Pace Timetracker

A.2 Additional Artifacts

A.2.1 Settings “Dev” Tab

The “Dev” tab in the companion app contains various settings and tools that were used during development and testing of the app, as detailed in [Section 10.3.10](#). As of the date of submission of this thesis, the dev settings are still accessible in the app, as they are provided for future developers to aid in further development and testing of the app. However, they are not to be considered as part of the final submission of the thesis.

A.2.2 Auxiliary Repository Directories

The main directory of the code repository contains several subdirectories that were used during the development and testing of the system, but are not part of the final submission of the thesis and should not be considered in the assessment. These folders are left for future reference only.

- **drowian-windtunnel-app**: This directory contains an early version of the app, which only contains the display of live information from the quadcopter and was used during initial Windshaper tests. It is left in the repository for reference and to aid future testing and development of quadcopters but should not be counted to the final submission of the thesis.
- **measurements**: This directory contains the data exported from the field tests, as well as scripts and Excel files used to analyze the data. It remains there as evidence of the results of the field tests, but is not intended for further use beyond the scope of this thesis, and should not be considered in the thesis assessment apart from verifying the test results.

A.3 Companion App Class Diagram

The following pages show the full class diagram of the companion app, generated from the TypeScript source code using tplant. The diagram covers stores, services, hooks, and utilities. It is split across multiple pages for readability.

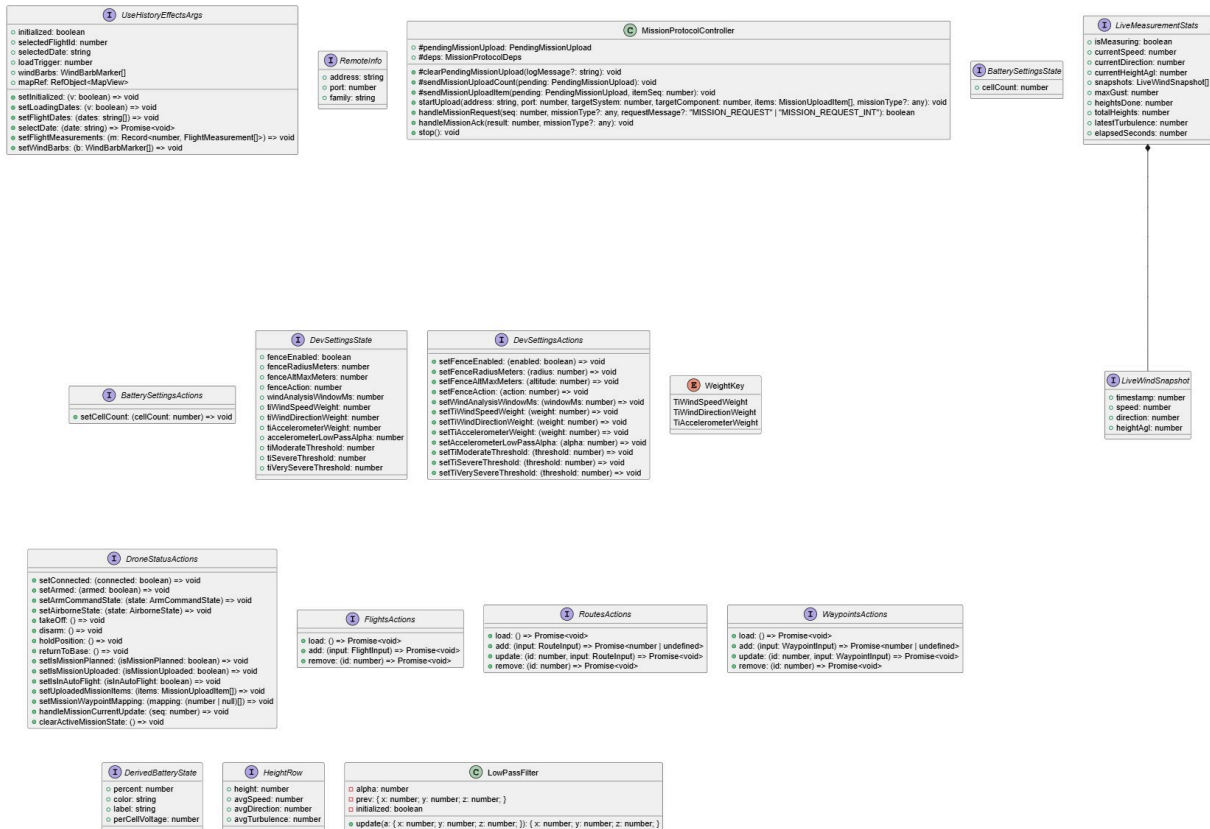


Figure 66: Companion App Class Diagram (Part 1 of 4)

Appendix

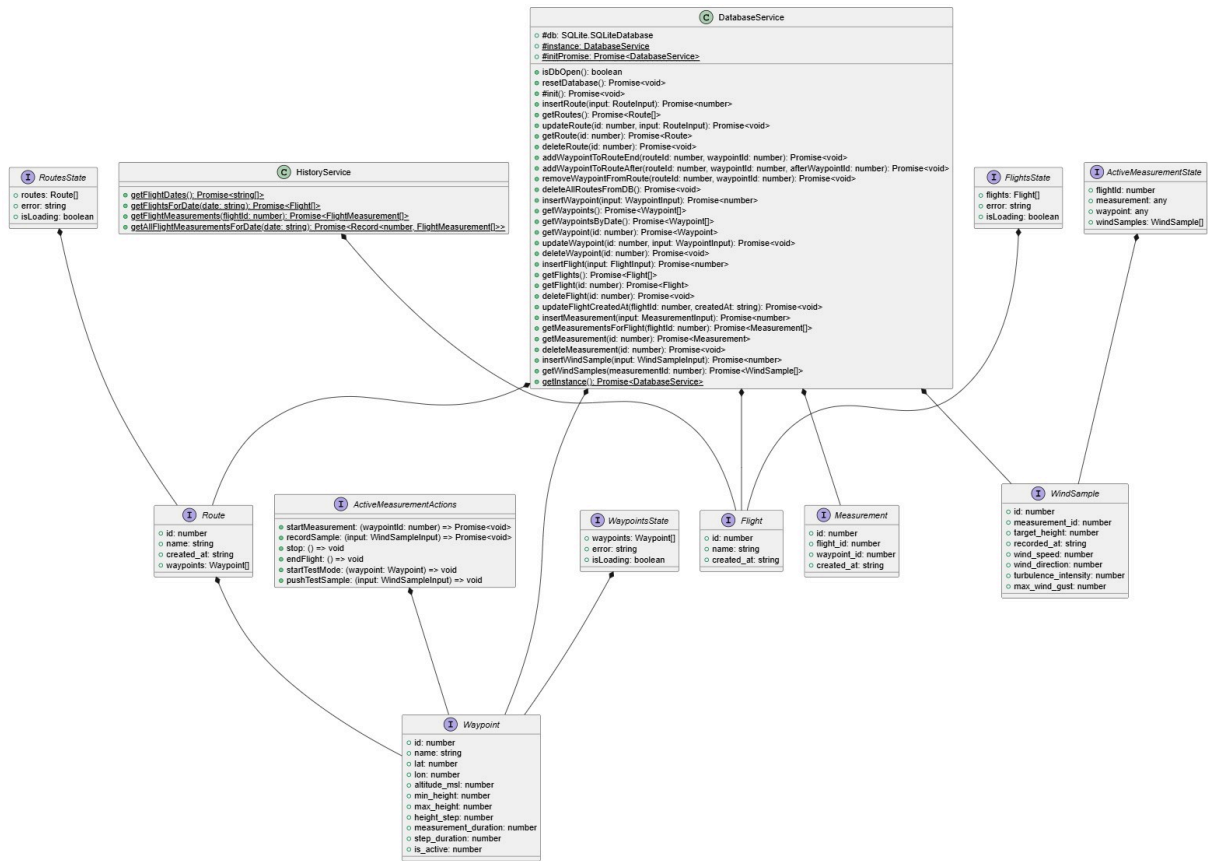


Figure 67: Companion App Class Diagram (Part 2 of 4)



103

Appendix

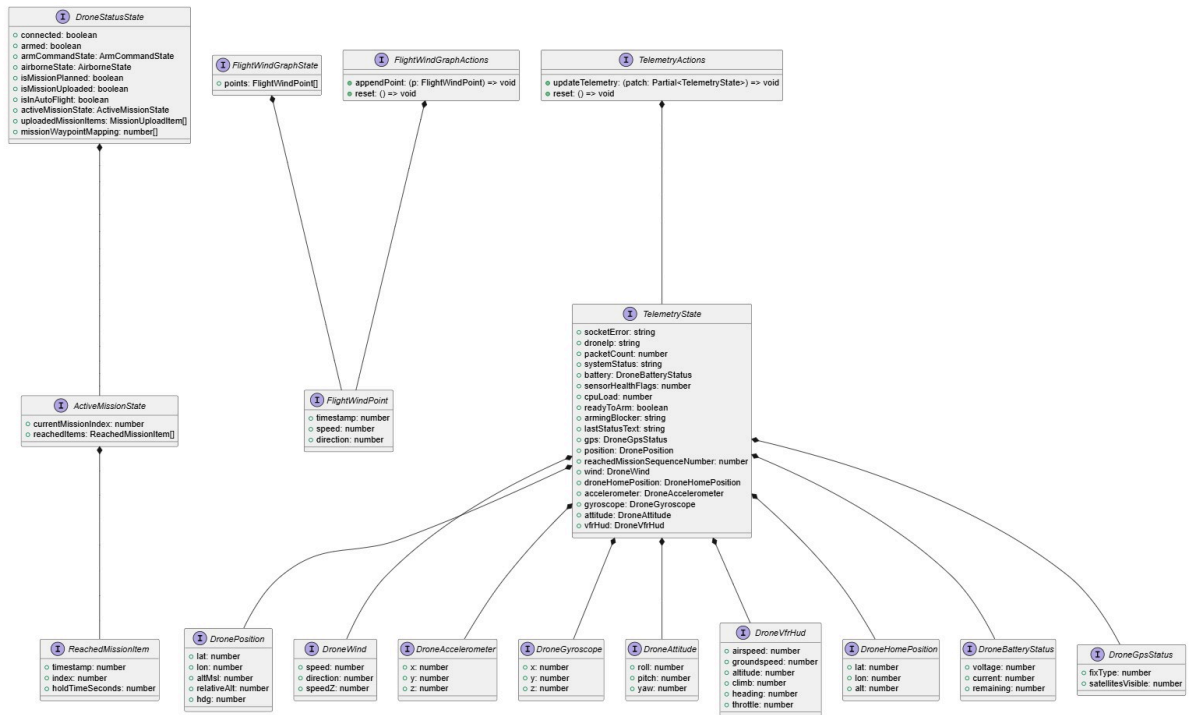


Figure 69: Companion App Class Diagram (Part 4 of 4)

A.4 Quadcopter Build Notes

This appendix is a collection of the knowledge gained during the construction of a quadcopter. Its purpose is to accelerate future quadcopter development by documenting key insights, design decisions, proven solutions, and common challenges. This allows upcoming projects to build directly on established experience and reduce redundant work.

A.4.1 Flight controllers

At the time this was written, advanced chip exports to China were being boycotted, which made advanced 2 MB FCs unavailable.

Flight-Controller (FC) boards are essentially a single main chip on a small PCB, while the remaining area is mostly pads that route signals to sensors, ESCs, and peripherals. The chip itself acts as the central control unit of the quadcopter and determines how much logic, and which features, the FC can support.

Many common chips, such as the STM32F405 series, provide only around 1 MB of RAM/flash, which is sufficient for lightweight firmware like Betaflight. However, this memory limit is too restrictive for more advanced systems such as ArduCopter or INAV, which include features like wind estimation, automatic navigation, precision landing, and other high-level autonomy functions. As a result, these firmwares often need to be trimmed down to fit into 1 MB, losing important capabilities in the process.

ArduPilot offers the option to build a custom firmware configuration, allowing you to include only the features you need so the firmware fits the hardware constraints while still providing the required functionality.

A.4.2 Electronic Speed Controllers (ESCs)

In most quadcopter builds, ESCs come as 4-in-1 modules that connect directly to a serial interface on the flight controller. This is the standard approach because it reduces wiring, simplifies installation, and ensures clean signal routing. It is also technically possible to drive ESCs via PWM servo outputs, but this is uncommon and, in our case, did not work on the Matek F405.

The most reliable setup is to use a 4-in-1 ESC from the same manufacturer as the flight controller, ideally as a matched stack. This improves compatibility and avoids unnecessary troubleshooting.

Before using the ESC, it is important to verify which firmware it is running. To check or flash the firmware, connect the ESC to the flight controller and enable `SERVO_BLH_AUTO` in the ArduCopter firmware. This parameter forwards ESC telemetry/data through the FC. Without it, the ESC will not be detected, and ESC-Configurator will tell you that another device is using the bus.

Once forwarding is active, you can use esc-configurator.com to read and flash the ESC firmware. Bluejay is recommended, as it has become the standard for modern builds. BLHeli_30 or BLHeli_60 are also options for compatibility, but in most cases the choice has no significant impact on performance or compatibility.

The FC is the most crucial part of the quadcopter, and as the requirements of this project are very specific, we recommend the following attributes:

- Must have a SDA-SCL (I2C) bus.
- Must have at least 3 accessible serial ports.
- Should have a compatible ESC from the same manufacturer.
- Should have 2 MB flash memory to load all ArduCopter firmware features.

A.4.3 Frame Design Choices and Vibration Prevention

Frame design requires careful placement of components and attention to vibration sources. The GPS module should be mounted as far away from any high-current wiring as possible, with at least 6 cm of separation. ArduPilot provides an ESC-GPS interference test that helps determine the ideal GPS location for minimal noise. The frame itself must stay lightweight while still offering good rigidity; PA6-CF carbon-fiber filament proved to be a strong and reliable choice.

Vibrations must be kept under control. Identify which motor introduces the strongest vibration by checking logs or running simple motor tests. Ensure all screws on motors and propellers are properly tightened and that there is no mechanical play. If a motor is imbalanced, it should be replaced immediately. Check for weird noises from the motor. A dangerous but effective test is to hold the quadcopter down and let the rotors spin: the arm with the vibrating motor will visibly vibrate.

For software-side damping, ArduPilot offers several filtering parameters. A simple and effective option is adjusting the `INS_GYRO_FILTER`, which reduces high-frequency noise from the gyro. In our case, tuning this filter was enough to bring vibrations into an acceptable range. Additional options such as `INS_ACCEL_FILTER` or harmonic-notch filters exist, but starting with the gyro filter usually provides quick results.

A.4.4 Sensors and Serial Configuration

Sensor and serial wiring requires attention to signal direction, bus behavior, and power availability. For UART-based peripherals, RX and TX must always be crossed between the flight controller and the device. For I2C devices (SDA/SCL), the bus allows multiple sensors on the same two pins, but identical sensors with the same address cannot share the bus, otherwise address conflicts occur. Most peripherals support a wide input-voltage range, but it is essential to confirm this in their specifications.

Many ports on flight controllers only supply power when a battery is connected. During USB-only debugging these pads may remain unpowered, and this behavior is often not documented by the FC manufacturer. Camera-UART ports typically provide weak or unstable voltage and should be avoided for powering sensors. You can still use their RX/TX lines, but power and ground should be taken from more reliable pads elsewhere on the board.

When selecting the correct protocol and baud rate for a sensor or peripheral, the protocol must match what the device expects, while the baud rate is usually auto-detected by ArduPilot. However, certain components, especially radio receivers, require an exact baud rate specified by the manufacturer and will not function otherwise. It is also important to verify the mapping between logical serial ports and physical UARTs, since the numbering on the board does not always correspond to the `SERIALx` entries in the configuration. Always double-check the serial port configuration tab to ensure each device is assigned to the correct logical port with the correct protocol and baud settings.

A.4.5 WiFi Telemetry

To make WiFi telemetry possible, we used a Seeed Studio XIAO ESP32-C3 chip running the DroneBridge for ESP32 firmware. This bridges a serial port of the flight controller to a WiFi link, allowing a ground station to connect wirelessly over MAVLink.

On the quadcopter side, the serial port that the ESP32-C3 is wired to must be configured as MAVLink2 (protocol 2).

When configuring the GPIO ports of the chip in the DroneBridge firmware flasher, you must ensure that you do **not** configure the physical GPIO pin numbers but their digital reference instead. The

mapping between the physical pins and their digital references is shown in Figure 70. Using the physical pin numbers here is a common mistake and will result in a non-working serial connection.

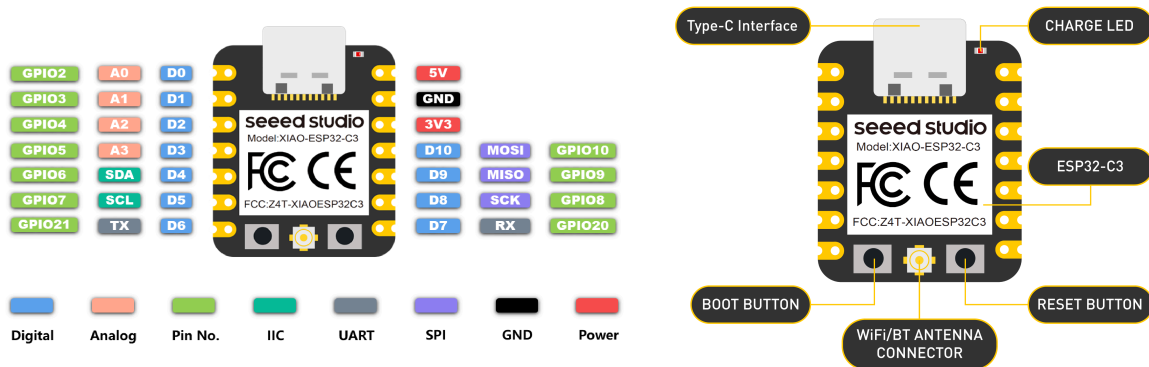


Figure 70: XIAO ESP32-C3 pinout: Physical Pins and their digital References

A.4.6 Firmware Flashing

Flashing firmware onto the flight controller was not always reliable with the usual tools. Whenever the standard ground-station flashing failed, STM32CubeProgrammer was always able to flash the hex file with bootloader.

The first time any ArduCopter firmware is installed on a flight controller, the **hex file with bootloader** must be installed, which for us did not reliably work using Mission Planner. Once the bootloader is present, the .apj file can be flashed afterwards, which conveniently does not override the ArduCopter configuration parameters.

A.4.7 Motors, Rotors and Battery Sizing

Use Oscar Liang's website¹ to understand what you need and buy the components accordingly. Use LLMs with caution. For this quadcopter the SpeedyBee F405 Mini (Figure 71) was used, whose documentation is included in this appendix.

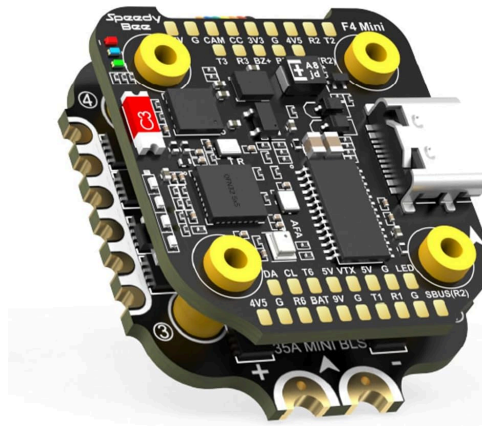


Figure 71: SpeedyBee F405 Mini Flight Controller used in this Project

To save any future user of this thesis time, the full parameter set used to configure ArduCopter is appended in this appendix and can simply be imported onto the quadcopter using Mission Planner.

¹<https://oscarliang.com>

Building a Custom Firmware

Because the 1 MB SpeedyBee F405 Mini cannot hold every ArduCopter feature, a custom firmware has to be built. ArduPilot provides an online custom firmware builder at custom.ardupilot.org. Select the SpeedyBee F405 Mini board and the ArduCopter vehicle, then enable only the features you need before downloading the resulting firmware.

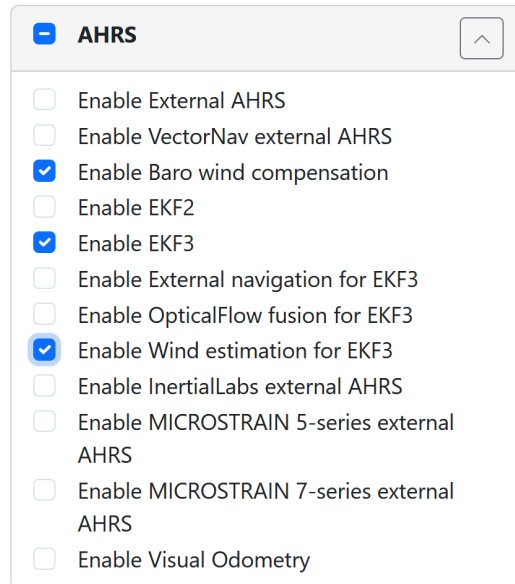


Figure 72: ArduPilot Custom Firmware Builder (custom.ardupilot.org)

For this project the two features that must be enabled are:

- **Enable Baro wind compensation**
- **Enable Wind estimation for EKF3**

Be aware that the parameters used to configure the wind estimation feature (for example the EKF3 drag and barometer wind-compensation parameters) will appear in the Mission Planner parameter list **even if the feature is not actually installed** in the running firmware. Seeing the parameters is therefore not proof that the feature is present. Always double-check that the feature is truly installed before relying on it.

Preparing for the first Flight

Strictly follow ArduPilot's guide on how to prepare a quadcopter for its first flight as described in the ArduPilot Methodic Configurator tuning guide.² Preflight calibrations such as the compass calibration may never be skipped.

Even small errors, like a slightly wrong compass offset, can lead to the quadcopter crashing. When the sensors disagree, the EKF becomes unsure which sensors to trust and starts all kinds of corrective actions to figure it out, such as completely resetting the accelerometer tuning. In our case this led to the quadcopter intentionally flipping onto its head, only to then realize via the barometer that it was descending swiftly, after which it reversed the manoeuvre to catch itself. This example illustrates why following these guides is extremely important.

Safety

As a conclusion, you must always be aware of the danger involved in coding such a quadcopter. An uncontrollable quadcopter can seriously harm other people. You must at least always have an emergency disarm feature ready, with the force parameter set, to ensure safety.

²https://ardupilot.github.io/MethodicConfigurator/TUNING_GUIDE_Heli.html