Zurich University
of Applied Sciences

School of
Engineering

Bachelor Thesis Computer Science

Building an Accessible and Affordable Drone Monitoring System Based on Remote ID

Authors	Sebastian Brunner Fabia Müller
Main supervisor	Marc Rennhard, ZHAW
Industrial partner	Cyber-Defence Campus, Zollstrasse 62, CH-8005 Zurich
External supervisors	Llorenç Roma, Cyber-Defence Campus Bernhard Tellenbach, Cyber-Defence Campus
Date	09.06.2023

Zusammenfassung

Die Anzahl Drohnen in der Luft haben in den vergangenen Jahren immer mehr zugenommen. Aus diesem Grund haben die EU und die USA jeweils ein neues Gesetz verabschiedet, welches Drohnen unter anderem dazu verpflichtet deren geografischen Standort und Informationen zum Piloten zu versenden. Diese Information wird Remote ID genannt. Ziel dieses Gesetzes und der Remote ID ist es, die Überwachung von Sicherheitsbereichen und Einrichtungen (z.B. Flughäfen oder Gefängnissen) zu erleichtern. Die Remote ID wird unverschlüsselt übermittelt und ist daher für alle zugänglich. Durch die baldige Einführung dieser Gesetze wird ein System benötigt, welches Drohnen anhand ihrer Remote ID überwachen kann. In dieser Bachelorarbeit wir die Übertragung via Wi-Fi, das Empfangen und das Extrahieren der Remote ID-Daten analysiert. Zusätzlich wird eine Lösung für ein funktionales Drohnen-Monitoring-System vorgeschlagen, entwickelt und getestet.

In der Bachelorarbeit wurde als erstes ein bereits bestehendes, von DJI entwickeltes System namens AeroScope, analysiert und dokumentiert. Danach wurde der aktuelle Stand der Remote ID inkl. Übertragungsmethoden, verschiedene Formate und deren Inhalt analysiert. Basierend auf diesen Erkenntnissen wurden die Anforderungen für ein besseres System abgeleitet. Dieses System soll im Vergleich zu AeroScope zugänglicher, preislich attraktiver und erweiterbar sein. Zuletzt wurde noch die Architektur sowie die Implementation des besseren Systems diskutiert, vorgestellt und ausgewertet.

Zum aktuellen Zeitpunkt gibt es kein vergleichbares System, welches Drohnen überwachen kann und verschiedene Remote ID-Formate und Drohnenhersteller unterstützt. Der Code wird zur freien Benutzung für die Öffentlichkeit zugänglich gemacht, was zur Weiterentwicklung einlädt. Vorgesehen ist die Installation des Systems auf einem Raspberry Pi und kann dann via Tablet oder Laptop verwendet werden. Dadurch ist das System im Vergleich zu AeroScope klein, günstig und zugänglicher. Zusätzlich überwiegt es AeroScope auch hinsichtlich Leistung und Benutzerfreundlichkeit. Das System ist jedoch in der Überwachungsreichweite eingeschränkt und enthält noch gewisse kleinere Unschönheiten. Hinzu kommt, dass die Remote ID bei Fehlkonfiguration oder an bestimmten Orten nicht versendet wird, was dazu führt, dass die Drohne vom System nicht erkannt werden kann. Da aber das Verwenden des Standards bald gesetzlich vorgeschrieben ist und Drohnenpiloten sich daran halten müssen, kann das System für die Überwachung von kleineren Umgebungen bis maximal 150 Drohnen verwendet werden und dient als exzellente Grundlage für zukünftige Erweiterungen.

Abstract

With the number of drones constantly increasing, the EU and the US are imposing policies that mandate drones to broadcast their geographical position and pilot information known as Remote ID. The purpose of this is to facilitate law enforcement in identifying and monitoring drones to protect restricted areas and facilities, such as airports or prisons, and enforce compliance with regulations. As this information is broadcasted unencrypted, it is openly accessible to everyone to monitor. With these policies soon taking effect, systems to monitor drones are needed. In this Bachelor thesis, the transmission method via Wi-Fi as well as the capturing and parsing of the Remote ID is analysed, a solution is proposed, and a working monitoring system is implemented and evaluated.

In a first step, an evaluation of an already existing monitoring system developed by DJI, AeroScope, is conducted and documented. Next, the current state of Remote ID including its transmission methods, different formats and content is analysed. Based on this, requirements for a better system, which is more accessible, lower-priced, and extensible, are defined. And finally, the architecture and implementation of a better system are discussed, demonstrated, and evaluated.

The developed application is, at the time of writing, the only existing system capable of monitoring drones in real-time supporting various Remote ID formats and manufacturers. The code will be publicly available and is extensible, making it free to use and open for contribution and customisation. Intended to be installed on a Raspberry Pi and accessed via a tablet or laptop, the system is smaller in size, lower-priced, and more accessible than AeroScope while outperforming it in terms of performance and user-friendliness. However, the system is limited in its effective range and carries some minor issues. In addition, some drones currently only broadcast a Remote ID in certain configurations and locations, which in some cases makes them undetectable with the current solution. Nonetheless, as the standard becomes mandatory and drone operators must comply with it, the system may be used to monitor areas on a smaller scale with up to 150 drones and offers a great basis for future extensions.

Table of Contents

1 Introduction

The purpose of this chapter is to give a quick overview of how the Bachelor thesis originated, the scope of the assignment and what is expected of the final product, namely a drone monitoring system. Lastly, an outline for each chapter is presented as a preview.

1.1 Background

In 2006 the first use case of a drone being utilized for commercial purposes was reported [1]. Since then, the number of drones has significantly increased. Currently, the U.S. Federal Aviation Administration registers around 345’000 commercial drones and 530’000 recreational drones [2]. Unfortunately, not only the number of drones increased, but so has the number of incidents where drones were used to transport illegal products or fly over restricted areas. According to DroneSec, a company providing drone threat intelligence solutions, the number of incidents in 2022 increased by 60 % compared to 2021, specifically from 708 to 1116 incidents [3]. Remote identification (**Remote ID**) for drones is, therefore, essential for security. It describes a drone's ability to transmit identification and location data to third parties while in flight. Authorities may track flying drones and determine who is controlling them by using Remote ID standards.

SZ DJI Technology Co., Ltd. (**DJI**), being one of the major companies in the commercial drone industry, created its own Remote ID solution, which has been included in the DJI model Mavic since July 2017 [4]. Moreover, DJI developed a portable drone monitoring system called DJI AeroScope Mobile (**AeroScope**), which can detect nearby DJI drones that broadcast Remote ID. AeroScope is designed to be used as an additional security measurement for airports, government sites, prisons as well as other important facilities [5].

1.2 Motivation / Goal

The main goal of this Bachelor thesis is the task of developing a drone monitoring system similar in functionality to AeroScope. The system must be able to receive and display drone Remote ID data which is sent unencrypted over 802.11 (Wi-Fi) broadcast. The motivation for a new drone monitoring system lies in the limited usability and excessive cost of AeroScope. The drone monitoring system developed in this Bachelor thesis aims to improve those issues.

Remote ID will be a requirement when manufacturing drones in the near future. Therefore, the system must be able to detect not only DJI drones but at least one more drone manufacturer. Parrot Drone SAS (**Parrot**) was defined as the second drone manufacturer in the assignment (see Appendix B), as it complies with the European regulations of the Remote ID [6].

In contrast to AeroScope, the drone monitoring system must be accessible to a variety of different communities and thus must be easy to use, affordable to operate and portable. The software must be open for reviews, extensions, and comments by the developing community in later stages. For that reason, the system will be open source.

1.3 Outline

This section serves as a quick overview of all the following chapters and thus contains a short outline for each chapter.

Chapter 2: Related Work consists of a brief introduction of previous work and studies done on this topic.

In **Chapter 3: Analysis**, the features of DJI's AeroScope are analysed and requirements for an improved system are gathered. In addition, it explains how a drone can be identified by its Remote ID.

In **Chapter 4: Technology and Architecture**, the design and architecture of the new drone monitoring system is documented. This includes decisions on which programming languages, technologies and libraries will be used to capture and translate the Remote ID, store the information, and display it in the frontend.

The implementation of the system is summarised in **Chapter 5: Implementation**. This includes the structure and API of the backend and the structure and UI of the frontend. In addition, it contains instructions on how to install the system, how drones can be spoofed using a script, and how to easily extend the system to meet individual needs.

In **Chapter 6: Testing and Evaluation**, the usability of the developed system is examined with real drones. Furthermore, the system's performance limits are tested with a spoofing script, ease-of-use and compatibility across different browsers are evaluated.

In **Chapter 7: Conclusion**, the results are discussed, Remote ID is critically reviewed, and potential future work is listed.

2 Related Work

In this chapter, the findings of prior research on Remote ID, its standards, and DJI's proprietary solution are presented.

DJI's proprietary Remote ID protocols, and their flaws, have first been studied and discussed in 2017 by Department 13 [4], an Australian drone technology company. The study shows that everyone with access to Wi-Fi and some technical know-how may receive the Remote ID protocols and track DJI drones. They spoofed the transmitted drone ID by hacking into the drone itself, reverse engineered the protocol, and implemented a simple tracking solution with Python.

Further research by Schiller et al. [7] in 2023 analysed the entire communication and information system of DJI drones. The research shows that "only a single DroneID packet is required to locate both the drone and the pilot" [7]. Furthermore, the pilot's location, which is transmitted by the smartphone application, can be spoofed by simply changing the phone's GPS coordinates, and there is no consistency or plausibility check performed by the drone itself. It additionally states in the research that a future standard is being drafted, which has been finalized at the time of writing this Bachelor thesis. Implementations for parsing said standard exist on GitHub including an Android application [8], [9]. This standard, however, is currently not implemented in all devices and not by all manufacturers.

At the same time, Dall'Omo [10] implemented and used a spoofer for DJI and Parrot drones as well as a tracking prototype for DJI drones to research Remote ID's and AeroScope's vulnerabilities for drone-specific attacks. That research is the basis for this Bachelor thesis with the purpose to develop a drone monitoring system as mentioned in Section 1.2.

3 Analysis

In this chapter, AeroScope and its features are summarised. This is followed by an introduction into Remote ID and its structure. Lastly, functional and non-functional requirements for an improved system are derived from AeroScope's features and the Remote ID standard. This serves as a basis for the development of the drone monitoring system.

3.1 AeroScope

AeroScope, as mentioned in Section 1.1, is a drone monitoring system in the form of a portable or stationary device. This Bachelor thesis focuses solely on the portable version, which allows its user to locate DJI drones and their pilots within a 5 km range [11] by means of DJI's proprietary Remote ID. It is not able to capture or parse other formats. As AeroScope supports not only Wi-Fi but also OcuSync and Lightbridge, and Wi-Fi usually reaches only as far as 2 km, the actual range of AeroScope for Wi-Fi monitoring is most likely lower [12]. With a size of 40.5 cm x 32.7 cm x 17.5 cm and a weight of 8.5 kg [11], AeroScope is rather large. Figure 1 shows the device and its dimensions.

Figure 1: DJI AeroScope Mobile [11]

Its features are simple yet not easy to use. The device contains a screen running Android, as displayed in Figure 2, which allows the user to interact with the software via a touchscreen. On the bottom left of Figure 2 the display shows a button list, which lists all the detected drones when clicked.

Figure 2: Screen of the portable version of AeroScope displaying a drone flight path [13]

Figure 3 shows an example of a pop-up window with data of a selected drone. It displays drone information such as the serial number, longitude, and latitude. The selected drone view also contains an option to replay previous flights. In such a replay the drone icon on the screen moves along the path of the previously flown route and leaves a line indicating the path travelled.

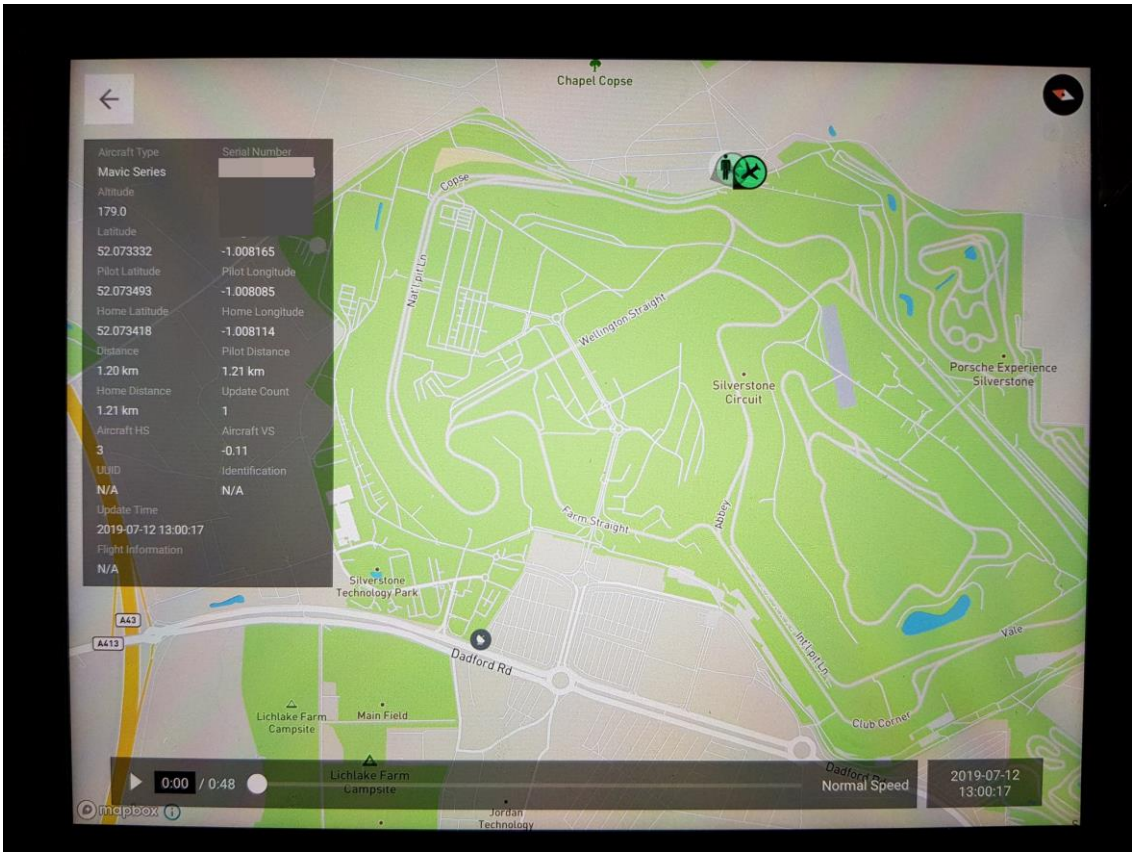

Figure 3: AeroScope displaying drone information of a specific selected drone on the left side [14]

To summarise, the key features of AeroScope are the following:

- Interactive map with locations of detected drones and optionally their pilots
- Display of a list of all detected DJI drones within a 5 km range
- Display of details of selected drone
- Display of flight path of a specific drone (history) on map

Although most mobile-native users can figure out the basic gestures required to use AeroScope, the UI's responsiveness and expressiveness needs improvement. Furthermore, when put under stress (e.g., through a spoofing attack), the application's performance suffers significantly, according to Dall'Omo [10]. During a test, conducted by Dall'Omo [10], with the use of 120 spoofed drones, the performance dropped continuously until a restart of the application was required to make it work again.

The most significant drawback of AeroScope is the costly investment of 7,900 Swiss francs [15] in hardware that customers are required to make despite drone information being broadcast over radio, which is freely accessible with a compatible (and rather low-priced) Wi-Fi adapter.

3.2 Identification of Drones

To be able to implement a drone monitoring system, one must first understand what Remote ID is, how it is transmitted, and how it is constructed. In this section these three subjects are discussed, beginning with what Remote ID is, followed by the structure of a transmission method, 802.11 Wi-Fi Beacon Management frame (**Wi-Fi Beacon frame**), and lastly the structure of the Remote ID.

3.2.1 Remote ID

Remote ID allows other parties to receive information about the identification of a drone in flight, its location, altitude, velocity, the pilot's location and more [16]. As of 16 September 2023, it will be mandatory in the USA to have a Remote ID-compliant drone system [16]. On 13 July 2022, the American Society for Testing and Materials (**ASTM**) published "Standard Specification for Remote ID and Tracking" [17] which defines transmission methods and message formats, among other things. On 1 February 2023, AeroSpace and Defence Industries Association of Europe - Standardization (**ASD-STAN**) published their own adaptation on regulating the Remote ID within the EU, which is compliant with the American standard to ensure interoperability [6], [18]. On 24 November 2022, it was decided that the EU drone regulation will also be adopted in Switzerland and will enter into force with a transitional period from 1 September 2023 [19]. Therefore, drones will have to comply with it.

The focus of this Bachelor thesis is on Wi-Fi as transmission method and will be discussed in Section 3.2.2. Therefore, no other transmission methods that are supported by the standards are discussed. The message format, however, is crucial for the development of the drone monitoring system, since it must be able to parse captured Wi-Fi Beacon frames to Remote ID-objects.

While researching the broadcasted packets by Parrot and DJI, several different formats were identified. Since these formats represent the transmitted Remote ID and need to be parsed by the drone monitoring system, they will be further discussed below.

3.2.2 Wi-Fi Beacon Frame

Remote ID can be transmitted via two radio frequencies (i.e., Bluetooth or Wi-Fi) [6], [7]. For the scope of this Bachelor thesis the main focus is on broadcasted messages via Wi-Fi. Remote ID data sent via Wi-Fi is constructed in a pre-defined structure called Wi-Fi Beacon frame [20]. Figure 4 shows said structure.

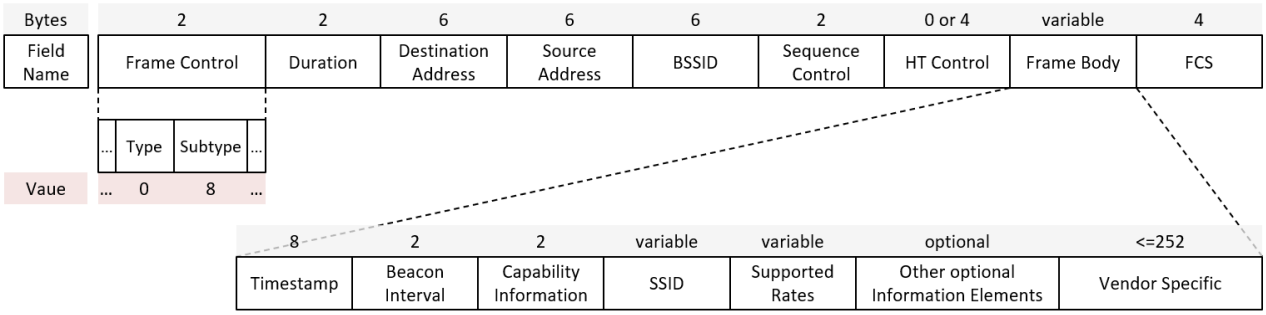

Figure 4: Structure of a Wi-Fi Beacon Frame including size in byte of each field (grey)

The identification of a Wi-Fi Beacon frame can be done by the *Frame Control* field value, which contains the type and subtype of the frame [20]. The value for management is 0 and the corresponding value for beacon is 8, which is displayed in Figure 4 [20]. These values are important for the implementation.

As displayed in Figure 4, a Wi-Fi Beacon frame also contains multiple other fields, all of which, apart from the *Frame Body* field, are of no significance for the Remote ID and are therefore not further discussed.

The *Frame Body*, or more precisely the *Vendor Specific* element of the *Frame Body*, is what is needed to access the Remote ID data. Therefore, all other elements of the *Frame Body* in Figure 4 are, again, not further discussed. The *Vendor Specific* element can be used by vendors to pack non-standard vendor information into the Wi-Fi Beacon frame [20], [21]. Since this element is used to transmit the Remote ID, it will be discussed more closely below.

Bytes	1	1	3	variable
Field Name	Element ID	Length	Organizationally Unique Identifier	Vendor-specific content
Value	221		00-00-00	

Figure 5: Structure of Vendor Specific Element

Figure 5 shows the structure of the vendor specific element consisting of four fields [20]. The *Element ID* contains the value 221 representing the type of the element, in this case a vendor specific element, according to IEEE [20]. The *Length* field contains the size of the vendor specific element, which consists of the combined length of the Organizationally Unique Identifier (**OUI**) and the vendor specific content. The OUI is a value to uniquely identify vendors. The OUI field can be used by vendors to transmit non-standardised data. To receive an OUI one must register with the IEEE registration authorities [22]. The OUI is used by the drone monitoring system to distinguish the drone providers as well as the applied standard to transmit the Remote ID. Table 1 lists the OUIs relevant for this Bachelor thesis [23], [24].

Table 1: OUIs of DJI, Parrot and ASD-STAN

DJI	Parrot	ASD-STAN
48-1C-B9	90-3A-E6	FA-0B-BC
60-60-1F	00-12-1C	
34-D2-62	90-03-B7	
	A0-14-3D	
	00-26-7E	

Finally, the last field of the vendor specific element, namely the V*endor-specific content* field, contains the transmitted Remote ID.

3.2.3 ASD-STAN Standard Format

The ASD-STAN format, as previously mentioned, is based on the EU regulations, which are compliant with the American standards published by ASTM [6]. It defines multiple message types to transport different information. The message structure, as displayed in Figure 6, consists of a message header of one byte and a message block of 24 bytes. This message structure applies to all message types.

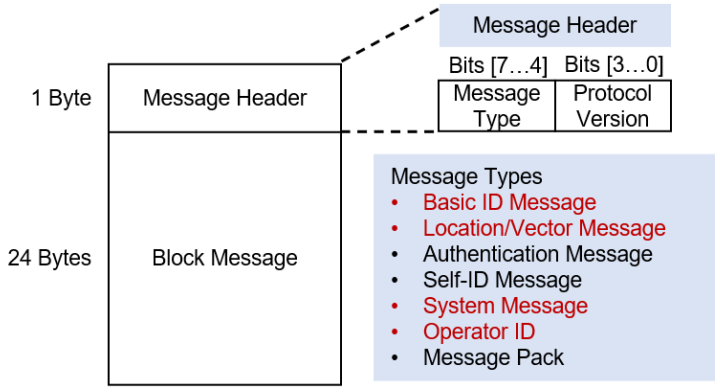

Figure 6: Message structure according to the ASTM and ASD-STAN standard

The message header contains the type of the message block and the protocol version. The message types highlighted in red font in Figure 6 are mandatory by the standard. The other message types are optional.

Figure 7 displays the structure of each mandatory message type. Multiple fields require a transformation equation, which can be dependent on a flag, to extract the actual value from the transmitted data. Figure 7 only lists the transformation equations which are not dependent on flags and are therefore simple. All other transformation equations can be found in Appendix A [6], [17].

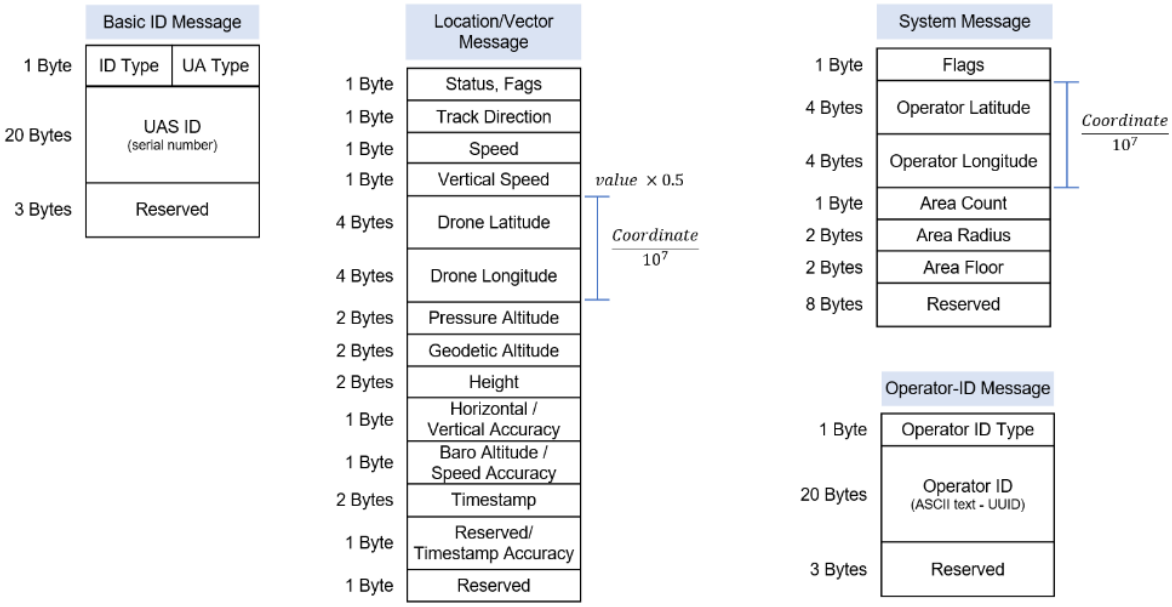

Figure 7: Structure of the four mandatory message types

The ASD-STAN standard requires the mandatory message types to be sent at least every three seconds [6], [17]. ASD-STAN lists the range over which the Remote ID can be broadcasted via Wi-Fi as 2 km (at best) [6]. However, they also mention that in reality, only half of this range can be expected due to interference and obstructions.

3.2.4 DJI Format

As mentioned in Section 1.1, DJI created their own Remote ID solution before the release of the standards provided by ASTM and ASD-STAN. Although DJI complies with the ASD-STAN standard, older drone models with outdated firmware might still broadcast a deprecated Remote ID. Thus, support for monitoring DJI's proprietary Remote ID is still desirable. This version of the Remote ID has been the subject of research by Department 13 [4], Dall'Omo [10] as well as by Bender [25]. These studies have identified that three packets

- license,
- flight information version 1,
- flight information version 2

were broadcasted via Wi-Fi. The license packet is of no further relevance for this Bachelor thesis, since it does not contain Remote ID information and will therefore not be discussed. The flight information packets, versions 1 and 2, act as a message container for Remote ID information.

Figure 8 displays the structure for both versions as well as the size in bytes for each field and the transformation equation for location values. The field sizes in byte according to Dall'Omo [10] differ from the ones made by Bender [25]. After extensively analysing a captured DJI drone format provided by the Cyber-Defence Campus (**CYD**) and comparing it with the information from the

studies of Dall'Omo [10] and Bender [25], the structure depicted in Figure 8 was composed. Unfortunately, it was not possible to verify this structure on a DJI drone. This will be discussed more closely in Section 6.1.

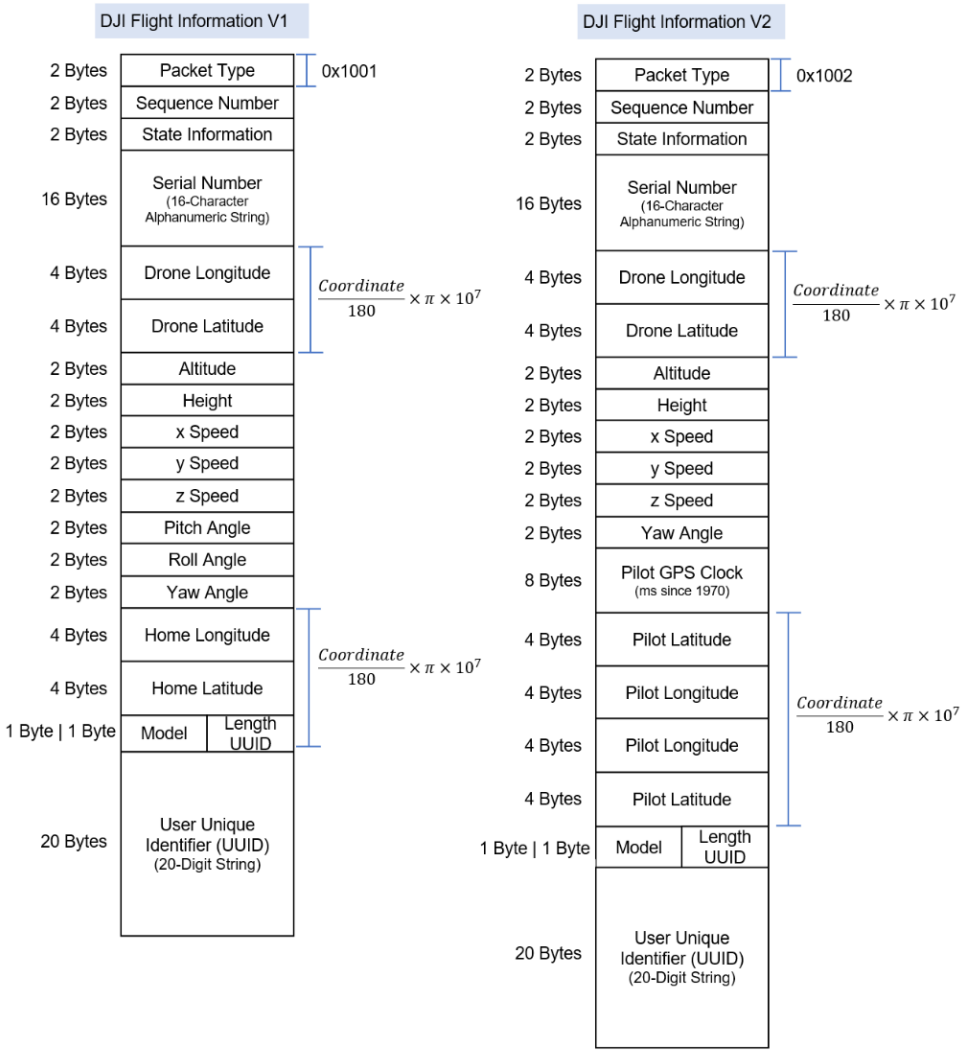

Figure 8: Comparison of Remote ID flight information packet version 1 (left) and version 2 (right)

Both versions contain the information required by the Remote ID standard [16]. Depending on the version some fields may vary. For example, version 1 contains the fields *Pitch Angle* and *Roll Angle*, whereas version 2 contains *Pilot GPS Clock* and *Pilot Latitude* and *Pilot Longitude* as displayed in Figure 8.

The two different versions can be distinguished by the version number found in the header of the Remote ID, which is important when implementing a parser for the different Remote ID versions. Version 1 contains the value 0x1001 in the field "Packet Type" and version 2 contains the value 0x1002 [25].

The specific fields of each version are not crucial to understand DJI's version of Remote ID. However, the following paragraphs describe some of those fields and their transformation equations. All information for those paragraphs is based on Bender [25].

The *Model*-byte is a value that can be mapped to a specific DJI drone model. The UUID is a 20-digit string linking the drone to a unique DJI account.

As mentioned above, some fields require the use of a transformation equation. For instance, for the location information (longitude and latitude) the following equation is applied:

$$\text{location information (longitude or latitude)} = (drone\ value/180) \times \pi \times 10^7$$

Equation 1: Transformation equation applied on the coordinate (latitude and longitude) values

The conversion of the height value is as simple as to multiply the value by 10. Other values, such as pitch, roll and yaw angle, require an algorithm to convert. The algorithm is displayed in Algorithm 1.

```
if angle_value == 0:
      return angle_value
else if angle_value < 0 or angle_value >= 180:
      reutn angle_value + 180
else:
      return angle_value % 180
```

Algorithm 1: Transformation algorithm for the angle values of the drone

3.3 Requirements for an Improved System

The features of AeroScope are simple but cover the basic functional requirements for a drone monitoring system, namely the live monitoring and replaying of previously captured drone flights. Table 2 lists all functional requirements (mandatory and optional) for a new and improved system. The optional functional requirements are provided by CYD as part of the assignment in Appendix B or are a result of the research of this Bachelor thesis.

Table 2: Functional requirements

Identifier	Requirement	Additional Remarks / Measures
Mandatory requirements		
REQ1	Support DJI's proprietary and the ASD-STAN Remote ID	Only via Wi-Fi
REQ2	Display all currently in-flight drones on a map	
REQ3	Update the live position of currently in-flight drone upon receiving an update	Update should be visible within 1 s of receiving it
REQ4	Selectively display historic flight path, pilot, and home location of drone	
REQ5	Display a list of all drones that were ever captured	

REQ6	Replay a flight of a previously captured drone	The flight must be traversable with an automatic play/pause button and a timeline to seek through
REQ7	Details for a replayed or in-flight drone can selectively be displayed	Details include all information available within a Remote ID
Optional extensions		
eREQ1	Read Remote IDs from captured pcap files	Useful for development and debugging purposes as well as analysing existing captures
eREQ2	Support Remote IDs broadcasted via LTE	
eREQ3	Detect spoofed Remote IDs	E.g., through Wi-Fi signal direction information

Of even higher importance for the usability of the drone monitoring system are the non-functional requirements. As described in Section 3.1, AeroScope faces challenges in terms of performance, reliability, and user-friendliness, along with a substantial price tag. Improving these factors and making the system more accessible are the foundation for Table 3, which shows the non-functional requirements for the new system. To measure whether the non-functional requirements are met, sensible target metrics were derived from AeroScope or the ASD-STAN standard.

Table 3: Non-functional requirements

Identifier	Requirement
NFR1	Continuously display at least 150 in-flight drones sending updates in 3 s intervals, as defined in the ASD-STAN standard, without crashing
NFR2	Update in-flight drones with live data within 1 s upon receiving a Remote ID
NFR3	Keep hardware costs under 10% of the costs of AeroScope (≤ 790 Swiss francs)
NFR4	Monitor drones within a 500 m radius (given a line of sight / no obstacles)
NFR5	Keep the installation process simple; a user with basic IT knowledge (e.g., executing Linux commands) must be able to install the software without specific training
NFR6	Make the usage of the application simple; a user without any IT knowledge or specific training must be able to use all key features described in Table 2
NFR7	Keep the code open for extensions; especially parsing new formats and adding new sniffing sources should be possible for a developer with coding experience
NFR8	Support all major devices, operating systems and/or browsers

To summarise, the system must replace at least all existing features of AeroScope and improve its accessibility, reliability, performance, and user experience. Contrary to AeroScope, the developed system is to remain open for extensions by the developing community and is intended to support the ASD-STAN as well as DJI's Remote ID.

4 Technology and Architecture

This chapter contains an outline of the analysis of potential solutions for various technical issues. This includes trade-offs as well as the decisions taken along with their justifications.

The analysis identified the following technical issues that need to be adequately addressed while designing the drone monitoring system:

- Capturing, parsing, and filtering Wi-Fi Beacon frames with drone-specific information
- Storing captured time-series information
- Displaying drones on an interactive map in an intuitive UI
- Making the usage (including installation) of the software as easy and simple as possible

The last point is important as, per NFR5 and NFR6, anyone, even without much technical experience or training, must be able to set up and use the system. This restricts the list of viable solutions drastically. Since this is a cross-cutting concern, it is addressed first, and each subchapter reiterates on how and why this shaped the solution.

4.1 Accessible and Easy-To-Use System

To make the drone monitoring system more accessible to a wide variety of users, the need for specific hardware must be reduced to as little as possible. While it is theoretically possible to run such monitoring software on a laptop without requiring any specific hardware, it is simpler to install and run the software on a Raspberry Pi (**Raspi**). This is because most integrated network interface cards (**NIC**) as well as some operating systems (**OS**) are limited in their capabilities. The Raspi, on the other hand, provides a controlled environment that guarantees that all prerequisites to capture data regarding OS and hardware are met, with one exception. The limits of NIC and the exception regarding the OS prerequisite will be discussed more closely in Section 4.2.

In addition, at a size of 10 cm x 6 cm x 2 cm and acquisition costs of 153 Swiss francs (on Galaxus as of 8 June 2023), the Raspi is smaller and cheaper than most devices. Thus, improving portability and accessibility. Figure 9 displays the size of the Raspi compared to an iPad Air 4th Generation.

Figure 9: Raspberry Pi 4 in comparison to iPad Air 4th Generation

Considering the above, the Raspi reduces the complexity of making multiple differing systems and devices compliant (NFR8) while keeping the hardware costs low (NFR3).

Throughout this Bachelor thesis when a “Raspi” is mentioned, it refers to the Raspberry Pi 4, which was used for research and testing purposes.

4.2 Capturing Wi-Fi Beacon Frames

To capture Wi-Fi Beacon frames, the application must access a Wi-Fi device in monitor mode and parse incoming binary data into a format that can be filtered and processed. Most integrated NICs do not support monitor mode including the built-in one of the Raspi. Therefore, the Raspi requires an additional capable adapter. Some adapters require the installation of additional drivers to support monitor mode, but the EDIMAX EW-7811Un adapter, shown in Figure 10, is natively supported by the Linux kernel, small and rather low-priced (e.g., 14 Swiss francs on Galaxus as of 2 June 2023). According to the Wireshark Wiki “changing the 802.11 capture modes is very platform/network adapter/driver/libpcap dependent, [sic!] and might not be possible at all” [26]. Further solidifying the decision to deliver the application on an external Raspi running Linux.

Figure 10: EDIMAX EW-7811Un Wi-Fi adapter used for testing [27]

The de facto standard for network **p**acket **cap**ture is called “pcap” defined by the “libpcap” Unix library [28] (under Windows also available via the “Npcap” library [29]) and thus represents the best format for the drone monitoring system to monitor packets. Both “libpcap” and “Npcap” are written in C but offer abstractions in different languages. This provides these solutions for packet capture as displayed in Table 4:

Table 4: Advantages and disadvantages of different network capturing solutions

Library	Advantages	Disadvantages
Scapy (Python)	• Massive popularity (8.5k stars on GitHub) • Easy to use / install as pip library • Supports most protocols (e.g., can replace 85 % of nmap) • Unlimited direct hardware access through native libraries • Easy installation on Linux / Raspi as Python is preinstalled	• Python better suited for scripting than web development • Python is interpreted and slower than compiled languages • Only scripting knowledge in development team, no web development experience in Python
Pcap4J (Java)	• Some popularity (1k stars on GitHub) • Great performance with JVM • Limited but extendable direct hardware access with JNI • Java was designed for web development, the ecosystem is massive (e.g., Spring) • Expert knowledge in development team	• Does not support all protocols by default (e.g., Wi-Fi Beacon frames are missing) but can be extended • Java Runtime Environment must first be installed on device
Libpcap (C)	• Unlimited direct hardware access • Best performance • No installation required / simple executable • Direct usage / no abstraction of libpcap (all features are usable)	• Not great for web development • Only basic knowledge in development team • Limited portability

Given that the library abstractions by Scapy and Pcap4J support most features, the performance and direct hardware access benefits of C do not outweigh the lack of knowledge within the development team. Therefore, C was not deemed a viable solution. Despite the development team having more experience working with Java (which is always an important factor to consider) and web development being more suited for the Java ecosystem with its vast number of frameworks, Scapy and Python were the better choice for the following reasons: First, Scapy has a wider community and supports more features, which makes its usage easier (NFR7). Second, Python is

preinstalled on Raspis and most Linux systems. In addition, it is interpreted and does not need to be compiled beforehand, which makes the installation straightforward by just copying the project files (NFR5). Third, most related work and especially the main basis for this thesis, Dall'Omo's research [10], have been done with Python, not only providing an existing code base but also making future integrations or improvements easier. And lastly, although Python was not primarily designed for web development, it has a large community with an enormous number of frameworks and libraries that simplify web development and database integration, which reduces Java's advantage in that regard.

4.3 Storing Captured Tracking Information

As all captured data is time-stamped, the use of a time series database like InfluxDB, which would index and optimize queries on the timestamp, could bring significant performance benefits. However, the installation process and the constraint of running on a Raspi again shaped the solution space. Hence, SQLite was chosen. SQLite only requires a runtime library, which is available for Python and most other languages. Additionally, no installation process is required as the database can be stored in a single file.

The main concern was restrictions on database or table size, which proved to be unfounded as SQLite supports up to 281 terabyte databases and 2^{64} rows per table [30]. As seen in Equation 2, the limiting factor is the database size. At these dimensions, the storage device would most certainly reach its limit well before the SQLite restrictions would set in. With sufficient storage available, these limits would allow the simultaneous recording of 100 drones every 10 ms for almost 14 years straight, as shown in Equation 3.

$$rowLimit = 2^{64}\ rows$$
$$dbSizeLimit = 281\ TB = 281 * \ 1024^4\ Bytes$$
$$packetSize = 68\ Info + 5\ Timestamp = \ 77\ Bytes$$
$$\frac{dbSizeLimit}{packetSize} = \frac{281 * 1024^4}{77} \approx 2^{42} < 2^{64} = \ rowLimit$$

Equation 2: Rows vs database size limit comparison

$$dbSizeLimit \ \approx \ 2^{42}$$
$$100\ Drones\ every\ 10ms = 10'000\ entries\ every\ s$$
$$\frac{dbSizeLimit}{10'000\ entries * 60 * 60 * 24 * 365} \approx 14\ years\ of\ recording$$

Equation 3: Theoretical recording limit

In practice, these limits are implausible to reach and therefore of no concern. With a storage capacity of 28 GB, which is the remaining free space on a 32 GB SD card after a clean Raspberry Pi OS installation, the recording with more realistic parameters (50 drones, every 3 s)

would last for about 271 days as shown in Equation 4. This is an acceptable limit. The packet size was estimated based on the database schema shown in Figure 11.

$$packetSize = 77\ Bytes$$

$$50\ Drones\ every\ 3s = 1000\ entries\ every\ min$$

$$totalCapacity = 28\ GB = 28 * 1024^3\ Bytes$$

$$\frac{totalCapacity}{1000\ entries * packetSize * 60 * 24} \approx 271\ \text{days}$$

Equation 4: Realistic recording limit

Concerns regarding database performance issues were not considered, as the system is intended to run on a portable, local device and is not supposed to monitor a heavily protected facility where hundreds of drones might simultaneously broadcast at a high frequency (see NFR1). While such applications might make sense to consider in the future, components such as the database can easily be replaced by a more scalable solution. Nonetheless, Section 6.4 evaluates and documents the performance of the current solution.

Since the only information the application stores are captured Remote IDs, the database schema is simple and oriented around the Remote ID standard. Figure 11 shows the only table in the database schema, *remoteid*, with all its columns and their data types. Noteworthy is the *id* column which is uniquely generated by the database itself whereas all other values are extracted from the Remote ID packet.

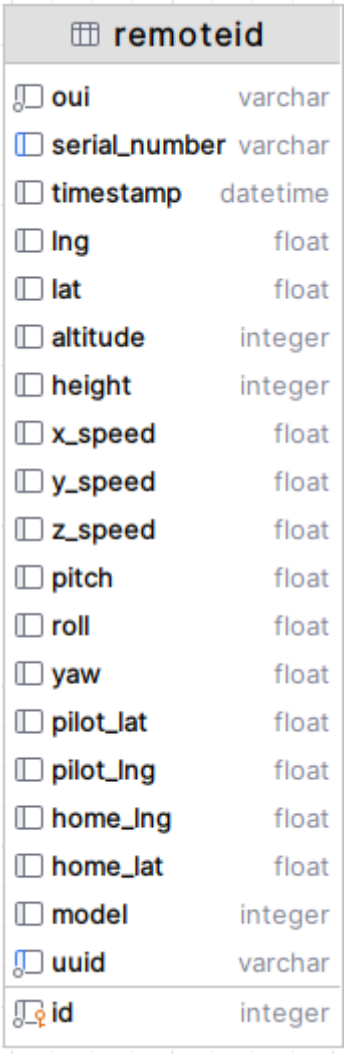

Figure 11: Database schema

4.4 Displaying Drones on a Map

Once again, the key constraint is ease of use of the application, and it was decided to provide the user interface (**UI**) as a web application served from a Raspi and accessed via a web browser over the local network. This allows the Raspi, once set up, to simply be plugged in and booted up. The application is then ready for use on any device with no further installation required on the client side (NFR5, NFR6 and NFR8).

Displaying the drones raised two distinct questions regarding the implementation: First, how to display an interactive map, or rather, which library to use, as coding this from scratch is unrealistic; and second, which UI framework or library should be used to make the frontend interactive and modular.

Google Maps is the most widely known web-based map library and was the best solution for the drone monitoring system. Google Maps offers all the features the system needs, namely markers, controls, and flight paths. Furthermore, it can be extensively customised and provides an up-to-date interactive map out of the box. The only downside of Google Maps is that it currently requires an

API key to run, which in its free version is limited to 28,500 map loads per month [31]. With the assumption of 20 working days per month and eight working hours per day, users with the same API key are allowed a little over 178 maps loads per minute. This is a limit that would be hard to reach in practice. Nonetheless, it adds complexity to securely store the API key and requires the additional installation step of requesting a new API key from Google Cloud Platform.

The handling of real-time location updates of the drones makes the entire application more complex. To structure the UI in a more modular fashion, the use of a component library was unavoidable. This allows the code to be abstracted and modularised by specific behaviour, such as displaying drones on a map or changing settings. Contrary to many commonly used component frameworks such as Angular, React, and Svelte that must build the code ahead of time, Vue can simply be integrated as a runtime library into a website and does not require any action during build time. This simplifies the installation of the application to merely copying files, proving Vue to be the sole viable option.

4.5 Architecture Overview

The architecture is composed of two main components: the sniffing service and the web UI. The sniffing service is written in Python and is responsible for monitoring, capturing, and filtering incoming Wi-Fi Beacon frames as well as checking whether the packets contain a valid Remote ID. In case they do, the Remote ID is parsed and stored in a database. Furthermore, it pushes current real-time data via WebSocket and provides stored data via HTTP to the web UI. The UI, written with JavaScript and Vue, displays the captured data and allows some configuration of the backend such as which Wi-Fi device to use. Figure 12 gives a visual overview of the entire system.

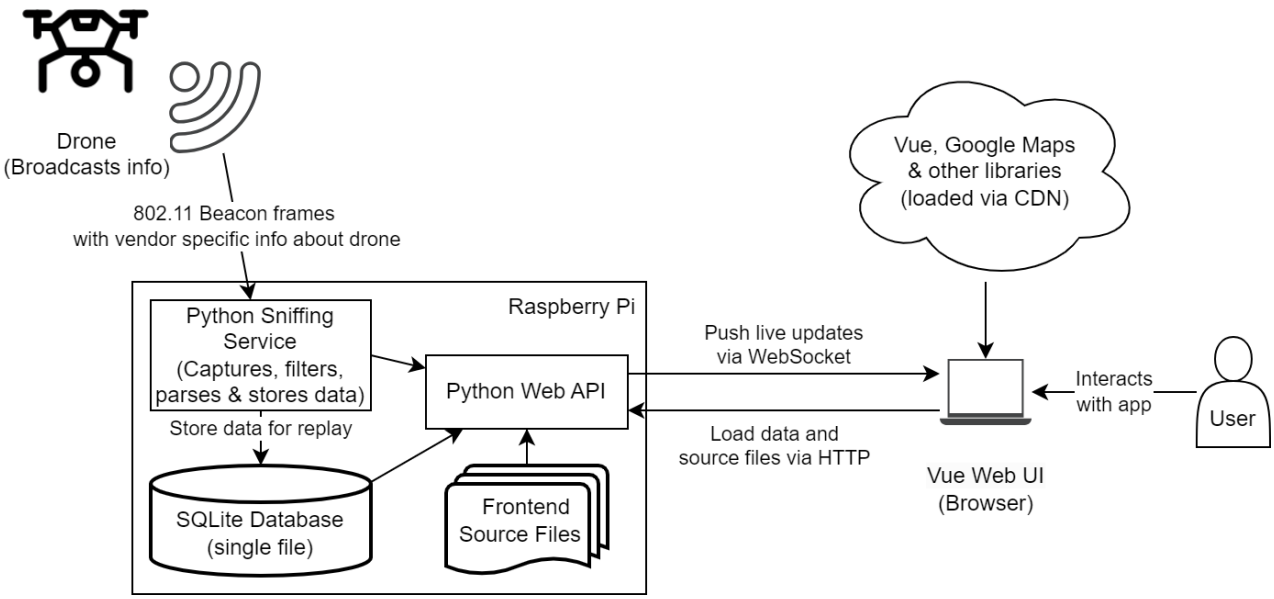

Figure 12: Architectural overview of the sniffing process

As visible in Figure 12, only the sniffing service, frontend source files and the database file are located on the Raspi. All frontend libraries are loaded via content delivery network (CDN) to ensure fast loading speeds and minimal storage space consumption on the Raspi. Only the sniffing service runs on the Raspi itself. The UI, once loaded, is solely executed in the client's browser.

5 Implementation

This chapter focuses on the implementation of the architecture previously defined in Section 4.5 on a more detailed, lower level. The structure of the backend is visualised using the example of the lifecycle of a captured Remote ID. The API endpoints as well as the frontend code structure are documented, and the different screens are described in form of a user manual. In addition, the chapter contains instructions for the installation on a Raspi, how drones can be spoofed via a script, and how the application can be extended. Extensions are demonstrated using LTE as a transmission method for the Remote ID and spoofing detection as examples.

All referenced code or files are located inside the *workspace* folder of the repository. The repository structure is explained in Appendix D. File, library, class, and component names are highlighted in *italic*.

5.1 Backend

The backend, responsible for capturing and storing data as well as providing said data to clients via an API, is written in Python. The library *FastAPI* [32] is used to specify the API and handle WebSocket connections. *Scapy* captures network packets, which are then parsed, filtered, and stored via *SQLModel* [33], an object-relational mapper, into the SQLite database. Both *FastAPI* and *SQLModel* were chosen because they integrate well with each other, and models can be reused for both API and database.

5.1.1 Structure

The backend code resides in the *backend* folder and is separated into the following files:

- **models/daomodels.py**: Defines Data Access Object (DAO) *RemoteId,* which contains all database fields as defined in Section 3.2.
- **models/dtomodels.py**: Defines all Data Transfer Objects (DTO) used in the API.
- **models/settings.py**: Defines the structure of the settings and its validators.
- **api.py**: Defines and implements all API endpoints and handles WebSocket connections.
- **info_handler.py**: Sets up the database and manages newly captured and parsed data by storing and sending it to connected clients.
- **main.py**: Entry point that parses arguments and bootstraps the entire application.
- **parser_handler.py**: Defines handlers for different formats that delegate the parsing of incoming packets to the right parser if possible.
- **parsers.py**: Contains actual parser logic for different formats.
- **settings.py**: Provides methods to load and save current settings.

- **sniffers.py**: Manages sniffers for various sources such as Wi-Fi interfaces and pcap files.
- **ws_manager.py**: Manages WebSocket connections and handles broadcasts to all of them.

Furthermore, all unit tests, which validate that the key features work, are located in the *tests* directory.

The logical structure is best explained with the sniffing and data processing process as visible in Figure 13.

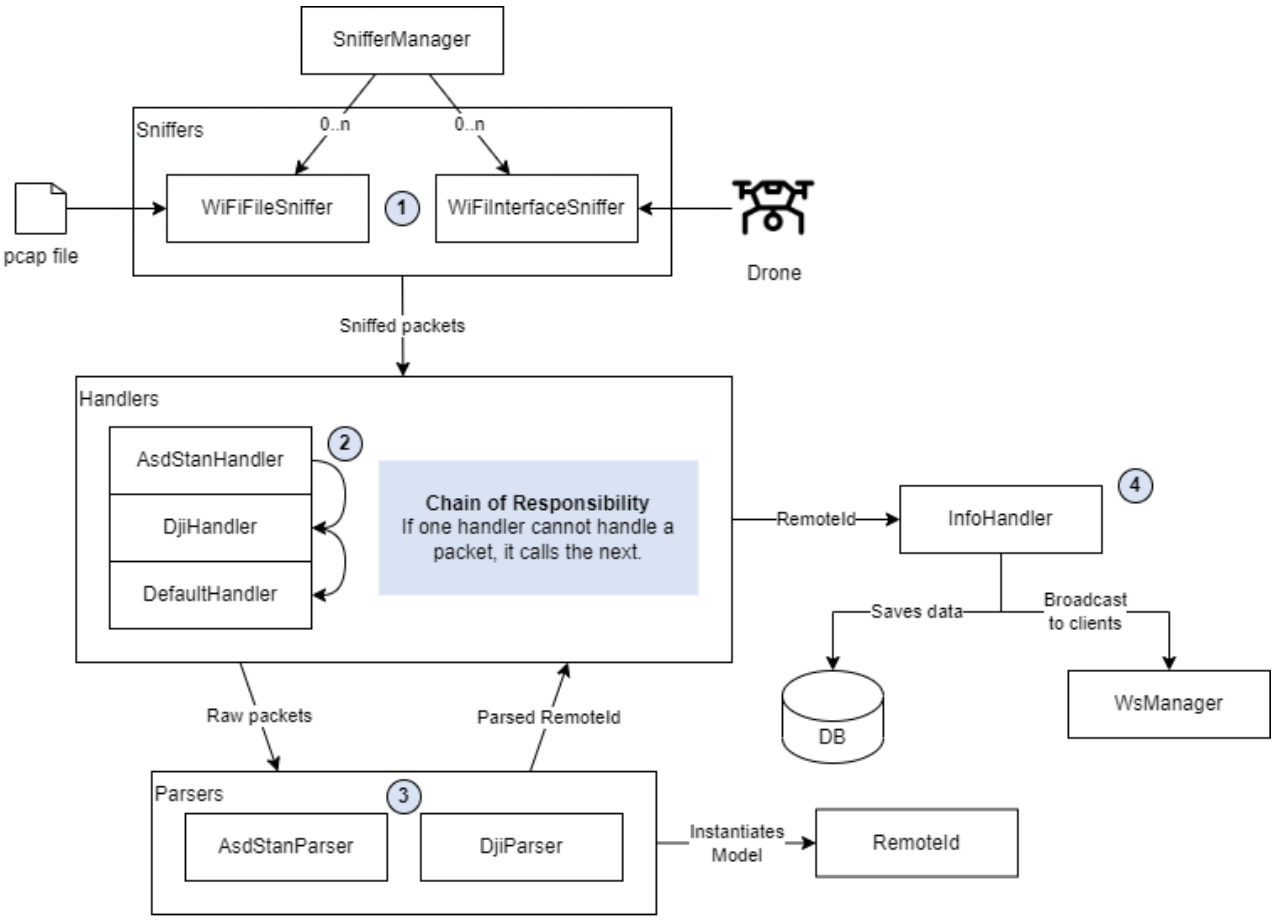

Figure 13: Structural overview of all components involved in the sniffing and data processing process

SnifferManager manages all sniffing processes from pcap files or actual Wi-Fi interfaces, by creating, starting, stopping, and destroying new or existing sniffers. These sniffers run each in a separate thread and, as shown in step 1 (marked with ① in Figure 13), capture or read raw packets from their respective source.

These packets are sent to handlers which try to extract a Remote ID. Since a Remote ID may have different formats, as mentioned in Section 3.2, the handlers act as a chain where the first handler checks if it can parse (accept) a packet and if so, it extracts its content itself. If the packet is unknown to the handler, it gets forwarded to the next handler and so on until the chain is exhausted and the packet is discarded. This pattern is commonly known as Chain of Responsibility and illustrated in step 2.

The actual extraction of data is performed by specific parsers shown in step 3 by reading raw values from the packet and applying conversions depending on the field. Since these parsers can get bloated, the code was separated from the handlers to provide a better overview. This data is then put into a *RemoteId* model and sent to step 4, the *InfoHandler*, which saves the model in the database and broadcasts the data via *WsManager* (Ws for WebSocket) to all listening clients.

5.1.2 API

The API mostly provides the UI with information and allows the user to change the settings dynamically. All data is sent as text in JSON format and the endpoints are defined in a RESTful manner. The API is not secured in any way, as generally only the user physically in possession of the Raspi has access to the data. Table 5 shows each endpoint with its purpose and expected input/output formats.

Table 5: API Endpoints

Method / Endpoint Purpose	Input / Parameters	Output
`GET /`		
Redirects to `/index.html` which contains the UI	None	Index.html file
`GET /ws`		
Accepts new WebSocket connections and handles them as long as they are connected	None	None
`GET /api/drones/active`		
Returns a list of all drones that are currently considered active (in flight)	None.	List of drones (*DroneDto*)
`GET /api/drones/all`		
Returns a list of all drones that have ever been captured (are currently in the database)	None	List of drones (*DroneDto*)
`GET /api/drones/{serial_number}/history`		
Returns the current flight path for the drone with the given serial number	`serial_number` (in URL): the serial number of the drone	The flight path of the drone (a list of *HistoryDto*)
`GET /api/drones/{serial_number}/flights`		
Returns a list of flight dates for the drone with the given serial number	`serial_number` (in URL): the serial number of the drone	A list of timestamps (start time) of all past flights of the drone (a list of strings formatted as UTC timestamps)
`GET /api/drones/{serial_number}/flights/{flight}`		
Returns the flight path for the drone with the given serial number for a particular flight	`serial_number` (in URL): the serial number of the drone `flight` (in URL): the timestamp of the flight (as returned by the flights endpoint) formatted in UTC and URL-encoded	The flight path of the drone of that particular flight (a list of *HistoryDto*)
`GET /api/settings`		
Returns the currently stored settings	None	Currently stored settings (*Settings*)

`GET /api/settings/interfaces`		
Returns a list of available network interfaces. This includes all interface (including Ethernet and loopback interfaces, not just Wi-Fi)	None	List of interface names as strings
`POST /api/settings`		
Update the current settings with new ones	HTTP Body: the new settings (*Settings*)	The newly stored settings (*Settings*)

5.2 Frontend

The frontend, which is written in JavaScript with Vue, is separated into various components. At the core, each component is connected to a *pinia* [34] store, which centrally manages all data. To explain the frontend more closely, the structure of the frontend will be explained in Section 5.2.1 alongside a section for each view in the UI.

5.2.1 Structure

All UI assets besides the *index.html* lie within a subfolder of the frontend directory separated by asset classes (*css*, *js*, *img*). The JavaScript sources are further separated into the following files:

- **Api.js**: Provides all API calls as asynchronous functions.
- **App.js**: Bootstraps the entire application at one entry point.
- **Components.js**: Contains all reusable components namely *ActiveDroneList*, *AllDroneList*, *DroneInfo*, *DroneInfoPanel* and *Settings*.
- **Drone.js**: Contains the *Drone* component which renders, animates, and moves each drone, displays information, and triggers actions if a drone is clicked.
- **MapView.js**: Contains the *MapView* component and its child views *MonitorView* and *ReplayView* that all display a map. *MapView* handles all interactions with Google Maps such as centring the map, drawing controls, and rendering markers (such as the drones and their flight paths).
- **SetupView.js**: This view is displayed before the application is set up correctly and only shows a setup form.
- **Store.js**: This is the heart of the application where all data is requested, stored, and transformed. Other components might request, subscribe to, and display different kinds of data that are all stored here.

Of further significance is the *index.html* file, which not only includes and bootstraps *App.js*, but also defines all dependencies in an import map. These import mappings enable the importing of dependencies via the *import* keyword throughout the frontend code and tell the browser from which source it must fetch said dependency. Even though the frontend only requires three direct

dependencies, *vue, vue3-google-map* and *pinia*, it further defines two indirect transitive dependencies required to run *pinia*: *vue-demi* and *@vue/devtools-api*. Both of which are rather small (≤ 1 kB).

While this setup is not optimal, as the import map must be maintained manually, it is necessary to enable the usage of *vue3-google-map* [35], which simplifies the usage of the Google Maps API with Vue significantly. The library is only available as an ES module and can only be imported like this. Import maps are currently not fully supported by all browsers; only about 76% of browsers are, according to caniuse.com [36]. Since testing with current versions of Chrome, Safari and Firefox was successful, further explained in Section 6.2, and other browsers are adding support for import maps soon, the downside seems tenable but noteworthy.

5.2.2 User Interface and Usage

The UI consists of two distinct views: *SetupView* and *MapView*. The former is only visible until Google Maps has been configured. Afterwards, only the latter is visible. *MapView* has two sub-views, one being *MonitorView*, which displays currently active drones, and *ReplayView*, which replays a specific flight of a drone. *MonitorView* is the homepage and acts as a hub to *ReplayView* via the selection of a drone flight to replay. The state diagram in Figure 14 shows when each view is active and how to switch between them.

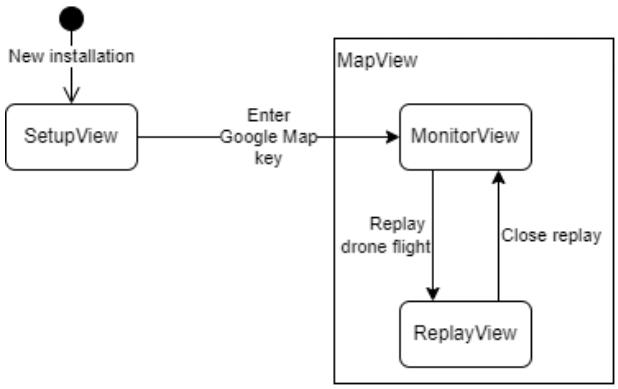

Figure 14: UI state diagram

5.2.2.1 Setup View

After a new installation of the software, the *SetupView,* visible in Figure 15, is displayed and asks the user to enter their Google Maps API key. A Google Maps API key can be obtained via Google Cloud Platform.

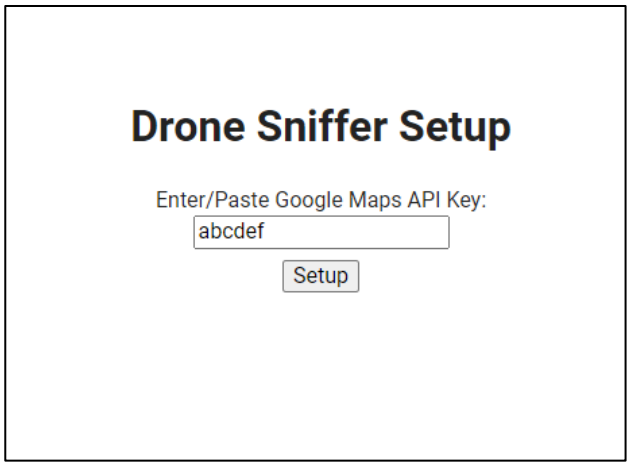

Figure 15: Setup view

5.2.2.2 Monitor View

Upon submission of the new setting, the view switches to the *MapView*, an example of which is shown in Figure 16.

Figure 16: Monitor view with schematic map

At the heart of the *MapView* is, as the name suggests, the map, which stretches across the entire screen. Markers and lines are drawn directly onto the map. Markers include drones, home, and pilot locations. Lines are the flight paths and drone-pilot links. Control elements are all placed at

the edge of the screens, anchored to either one of the corners or midway points of the edges and grow into the centre. Table 6 contains a list of all map control elements.

Table 6: List of map control buttons and their functionality

Symbol	Description	Side Note
	Toggle full screen mode	
	Tilt the map	
	Zoom in on the map	
	Zoom out of the map	
	Centre map to browser's current location	Works only if application is accessed via HTTPS or localhost due to security reasons

Figure 17 illustrates the two map options available to the user in the top left corner framed in orange. On the left side is the by default selected, more abstract schematic map and on the right side the more realistic satellite view. Either map style may provide a better overview of the flying drones as the contrast between drones and the background, especially on the satellite map, highly depends on the landscape underneath.

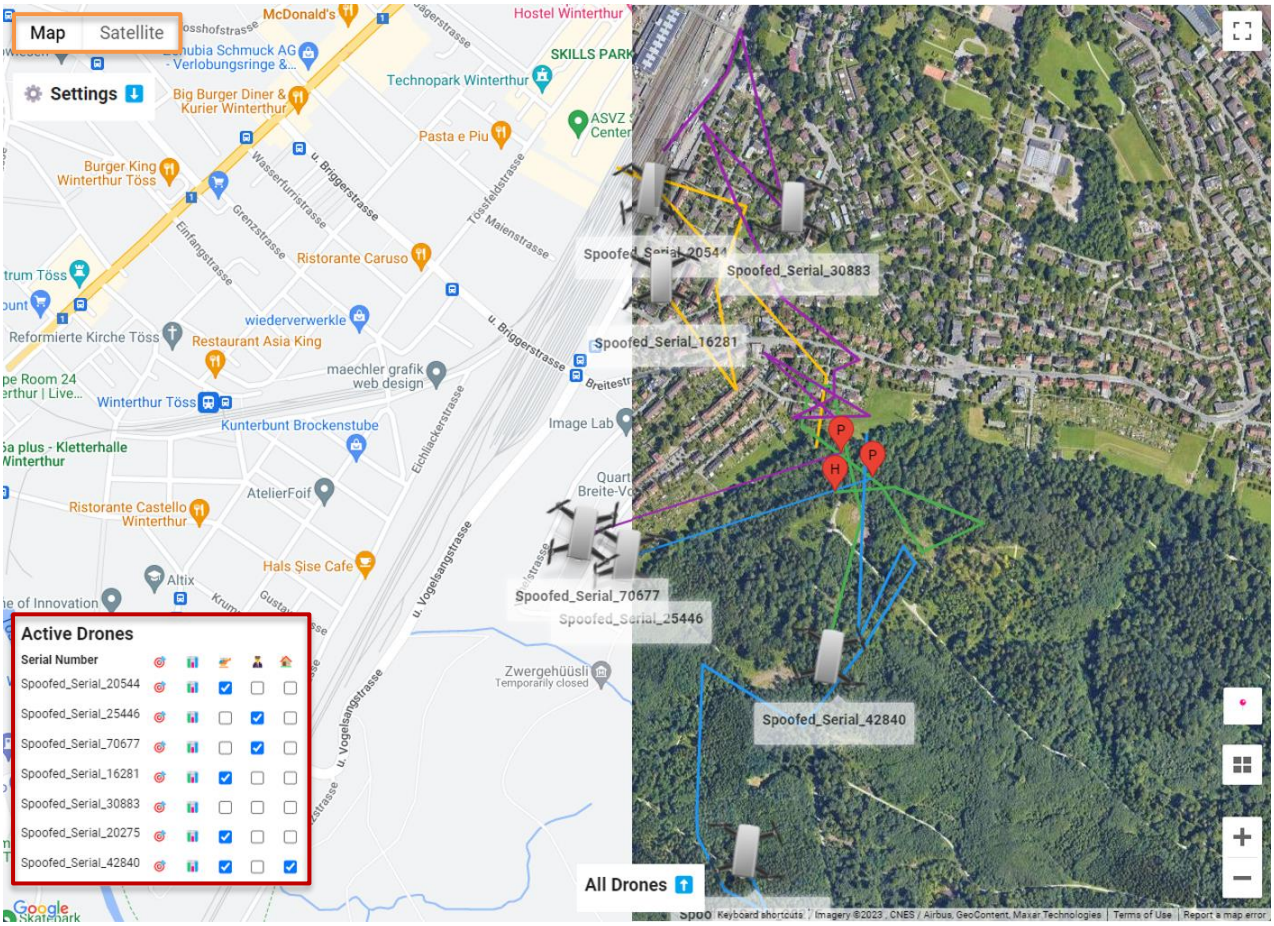

Figure 17: Monitor view with satellite map and the highlighted control elements

In the bottom left corner, framed red in Figure 17, all currently active drones are listed by serial number alongside some controls and options. The UI uses a lot of emojis rather than textual buttons to look leaner and more modern. The actual picture that is displayed depends on the system and might differ slightly.

The target icon (🎯), when clicked, centres the selected drone in the middle of the screen and tracks its movement from then on. If the user moves the map, the tracking is stopped. A click on the statistics icon (📊) opens the drone details explained further below.

The checkboxes underneath the helicopter (🚁) toggle whether the historic flight path of that particular drone should be displayed or not. It starts where the drone took off, called the home location, and follows the drone's flight path up to its current location. The paths are loaded lazily as soon as they are visible and each one has a distinct colour. In Figure 17, flight paths for four drones are displayed in blue, green, yellow, and purple. Due to the limited number of colours with sufficient contrast, duplicate colours are possible. The checkboxes beneath the pilot (👮) and home (🏠) icons each toggle the visibility of the pilot and home location markers. The markers are red and contain a single letter indicating the type of marker. "P" stands for pilot location and "H" for home location. The pilot location is dynamic and may change as the pilot moves. It is connected to the drone with a direct link in the same colour as the flight path to indicate which pilot controls which drone.

When a drone is selected, it is highlighted in the same colour as its path and its details are displayed beneath its serial number, framed red in Figure 18. Meanwhile, other drones are made slightly transparent to increase the contrast to the selected drone. After deselecting the drone, the details and highlighting of the drone disappear and the other drones become completely opaque again.

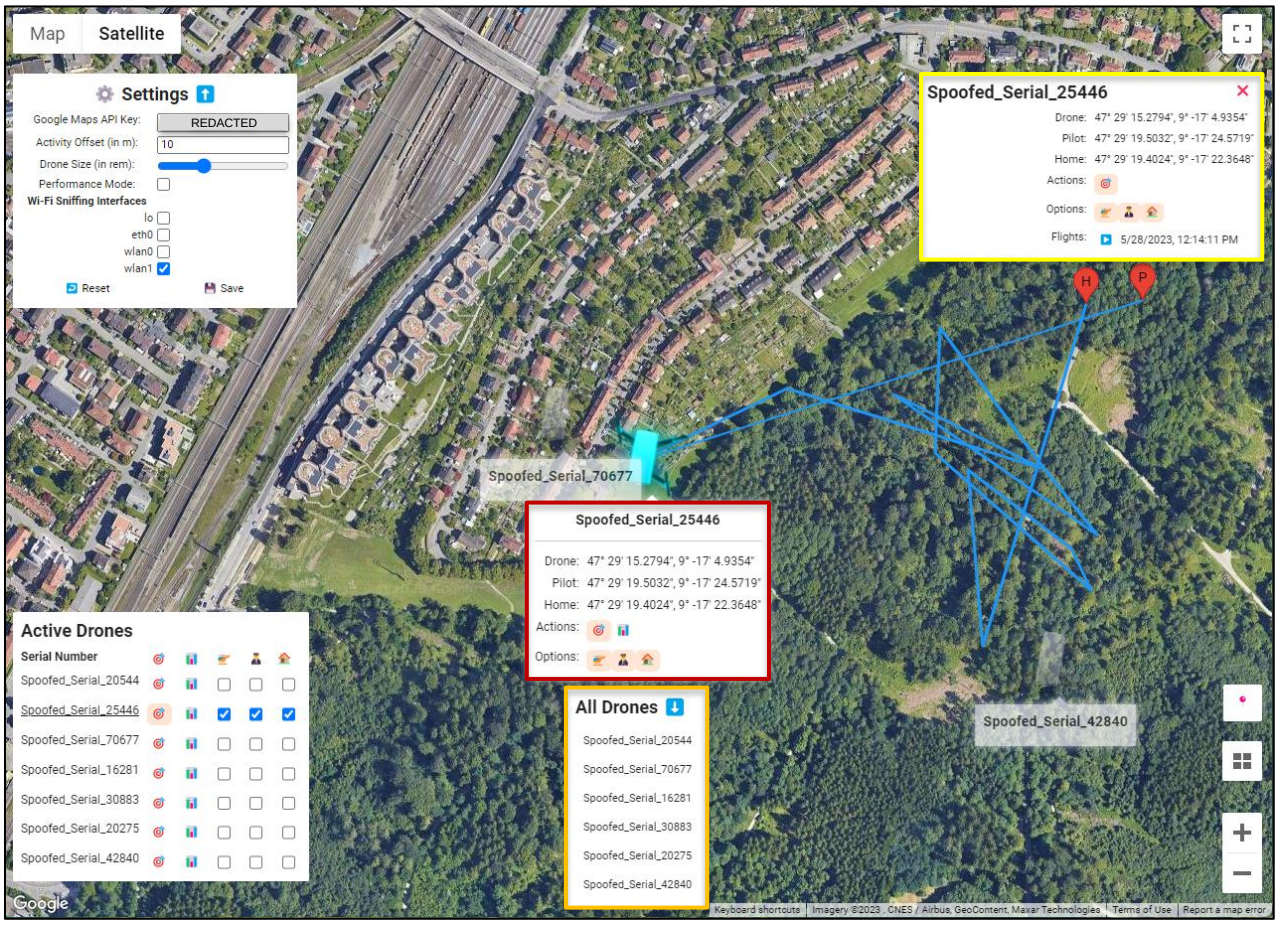

Figure 18: Drone details panels (red and yellow) and AllDroneList (orange)

The red framed drone details display all information available to the sniffer. In the sample of Figure 18, this includes the coordinates of the drone, the pilot and home location. As these are spoofed drones, identifiable by their serial number, not all information is transmitted. If available, altitude, height, speed in each direction, and the rotation would be displayed too. Furthermore, the rotation would be applied to the drone as well, indicating in which direction it is heading. Below the drone information are *Actions* and *Options*, which include the same controls as described in the list of active drones.

The same information is displayed in the top right corner in a separate pane, framed yellow in Figure 18, which is opened when the statistics icon (📊) is clicked. Contrary to the red framed pane, the yellow framed pane additionally displays a list of the respective drone's past flights. The past flights can be replayed by clicking on the replay icon (▶) opening the *ReplayView* for the selected flight, which is explained further in Section 5.2.2.4.

A list of all drones that were ever captured is located at the bottom centre of the screen, framed orange in Figure 18. The list, which is collapsed by default, can be extended (and collapsed) by clicking on its title. Clicking on a drone serial number in the list will display its details (yellow framed pane in Figure 18).

5.2.2.3 Settings

To configure the application, a settings panel is provided. It is located on the top left corner of the map. The collapsing and expanding of the panel can be toggled by clicking on the title. Figure 19 displays the expanded settings panel.

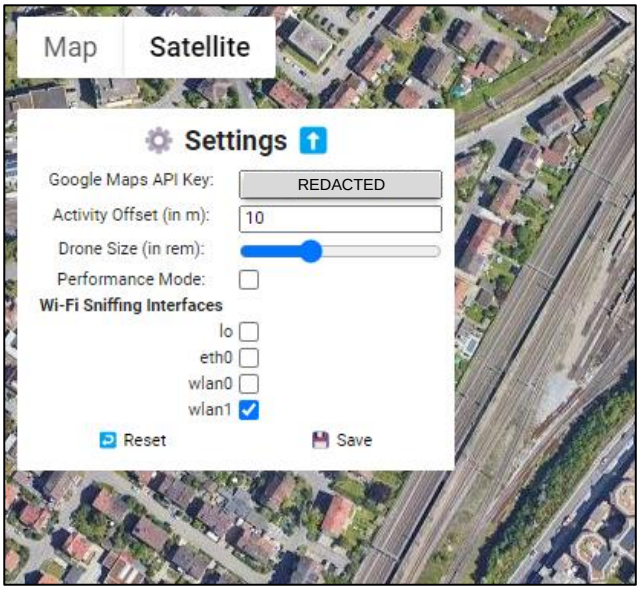

Figure 19: Expanded settings panel

The user may adjust the following settings:

- **Google Maps API Key**: The same API key that was set up in the *SetupView*.
- **Activity Offset**: The time delta in minutes that a drone is still considered active after its last sent update. This is needed to differentiate between different flights of the same drone as they do not send an update when they start or land. This means that if a drone lands and immediately takes off again, the system will recognise this as one flight. Must be a value between 1 and 60.
- **Drone Size**: A purely cosmetic option to adjust the size of drones on the map. The unit is rem where 1 rem is the same size as an M on screen (commonly used in CSS styles). Changing this setting immediately affects the UI but is not persisted unless saved. Note that drones are always the same size on screen regardless of the map scale. Must be a value between 1 and 10.
- **Performance Mode**: If enabled, the animated drones are replaced by a static, simpler image. This reduces performance issues browsers face when multiple drones are rendered

simultaneously, allowing for more drones to be monitored in parallel. Changing this setting immediately affects the UI but is not persisted unless saved.

- **Wi-Fi Sniffing Interfaces**: Displays a list of all detected network interfaces. If an interface is checked and the settings are saved, the system sets the interface into monitoring mode and starts sniffing on said interface. If unchecked and saved, the sniffing process is stopped, and the interface is set back to managed mode. The system supports sniffing on multiple interfaces simultaneously but might capture the same data multiple times, which may lead to side effects. Note that all detected interfaces are listed but not all support monitor mode (e.g., *wlan0* in Figure 18) nor are all Wi-Fi interfaces (e.g., *lo* and *eth0* in Figure 18). There is currently no mechanism to detect which interfaces are usable by the system and it is up to the user to know which interfaces to enable. On Raspberry Pi OS, Wi-Fi interfaces are usually named *wlanX* with “X” being the index increasing from 0 on and *wlan0* being the on-chip interface, which does not support monitor mode.

The settings are only persisted once the user clicks the save button. They can also reset the settings to the last persisted state by clicking the reset button.

5.2.2.4 Replay View

As mentioned in Section 5.2.2.2, upon replaying a flight, the application switches to the *ReplayView* as shown in Figure 20. In this view only the replayed drone is displayed. Additionally, *ActiveDroneList* and *AllDroneList* are replaced by the replay timeline at the bottom centre of the screen, framed red in Figure 20. By default, path, pilot and home location are turned on and the drone’s location, which was captured first, is centred.

Figure 20: Replay view with the path, home, and pilot location displayed

The timeline works just like a media timeline including a play/pause feature, which automatically replays the history one captured Remote ID per second and allows the user to freely forward and rewind through the flight. Closing the timeline with the red ✕ on the top right corner of the timeline returns the user to the *MonitorView*.

5.3 Installation

The application requires the Raspi to be set up with the Lite version of Raspberry Pi OS (**OS Lite**). The full version with desktop environment increases the CPU load and may cause the Raspi to miss packets [37]. This issue is further explained in Section 6.1. The application code can simply be cloned (or copied) onto the Raspi. As a second step, an installation script (*install.sh*) must be run, which

- installs Python, pip, and the required Python dependencies,
- copies the files required to run the application to the */opt* directory,
- sets up a system service,
- starts the system service.

This installation script must be executed with superuser privileges as, in order to sniff in monitor mode, the service itself must also run with superuser privileges. The user may pass the port the application should use with the parameter *-p* or *--port*. If not, port 80 will be used per default. Note that this installation script requires a working internet connection until the dependencies are installed. After this point, the Raspi does not require a working internet connection to run the application.

After the installation is complete, the user must visit the web application by entering the Raspi's hostname into the browser. As mentioned in Section 5.2.2.1, the user must enter a Google Map API key, which can be obtained from Google Cloud Platform. Afterwards, the application is ready to use.

The system service set up by the installation script starts as soon as the Raspi reaches the network target in its boot stage and restarts automatically on failure. This means that once the application is successfully set up, the Raspi can simply be plugged in, and the application is ready to be used within a few minutes.

5.4 Drone Spoofer

To be able to test the application without the use of a real drone, a script can be used to spoof one. As described by Dall'Omo [10], it is possible to spoof a drone's Remote ID by simply sending forged Wi-Fi Beacon frames. The script used by Dall'Omo [10] is publicly available and was therefore used as a basis for the spoofing script developed in this Bachelor thesis. Some functionality is added, and the existing code is improved.

Just like the drone monitoring system uses *Scapy* to monitor Wi-Fi Beacon frames, the script uses *Scapy* to send Wi-Fi Beacon frames. To do so, the script requires the input of an interface name through which the Wi-Fi Beacon frames will be sent. The spoofing script will set the interface into monitor mode, which requires superuser privileges, and sends the forged Wi-Fi Beacon frame repeatedly in an interval of three seconds, as defined by ASD-STAN and mentioned in Section 3.2.3.

Likewise, the structure of the forged Wi-Fi Beacon frame is designed in accordance with the standards [6], [17]. The mandatory message types, mentioned in Section 3.2.3, are created by the spoofing script, where the fields

- serial number,
- drone latitude,
- drone longitude

can be customised via arguments passed to the script.

Other fields, on the other hand, such as

- timestamp,
- operator latitude,
- operator longitude,

are set dynamically by the script and cannot be influenced. And lastly, all other fields in the mandatory message types are hard coded into the script.

The script uses a function to calculate a location close to specific coordinates within a certain radius. This function is used to calculate the starting location as well as the drone's next location prior to broadcasting. This intends to mimic the drone's movement as well as to assure that not all drones start at the same location when multiple drones are spoofed.

The starting location of the drone defaults to a location close to Kasernenareal in Zurich. As mentioned above, the starting location can be influenced by passing an argument to the script. The usage of the script arguments and their description are listed in Table 7.

Table 7: Arguments available to the spoofing script

Argument Flag		Value	Description
-i	--interface	Interface name	Interface to use
-m	--manual	None	Dynamically control drone movement with keyboard **Note:** cannot be combined with -r argument
-r	--random	Number of drones (defaults to one)	Define number of drones to be spoofed; drones move randomly to mimic movement **Note:** cannot be combined with -m argument
-s	--serial	Drone serial number	Define a custom serial number; only possible in manual and default mode (spoofing one random drone)
-n	--interval	Interval in seconds (defaults to 3 seconds)	Set interval how often Wi-Fi Beacon frames are sent
-l	--location	Latitude, longitude	Set drone starting location

If no argument is passed to the spoofing script it will run in random mode and spoof one drone. Figure 21 displays the behaviour and output of the script when it is started in manual mode. The logs contain, among other things, the mode, the movement of the drone as well as a notification the instant a Wi-Fi Beacon frame is sent.

```
user@ubuntu:~/DroneId-Monitoring/workspace$ sudo python3 spoof_drones.py -i wlx00c0ca916d -m
2023-06-02 18:27:04,123 INFO ########## STARTING DRONE SPOOFER ##########
2023-06-02 18:27:04,123 INFO Setting interface to: wlx00c0ca99160d
2023-06-02 18:27:04,123 INFO No location input detected. Using DEFAULT values.
2023-06-02 18:27:04,123 INFO Setting location to (473763399, 85312562).
2023-06-02 18:27:04,123 INFO Starting in MANUAL MODE - spoofing one user controlled drone.
2023-06-02 18:27:04,123 INFO Drone with SERIAL NUMBER b'Spoofed_Serial_10845' and LOCATION [LAT LNG] 473763399,
85312562 created.
2023-06-02 18:27:04,123 INFO Starting spoofing....
Use W, A, S, D to move the drone.
W (North)
A (West)
S (South)
D (East)
2023-06-02 18:27:04,479 INFO move NORTH
2023-06-02 18:27:04,980 INFO move NORTH
2023-06-02 18:27:06,512 INFO move WEST
2023-06-02 18:27:06,700 INFO move WEST
.
Sent 1 packets.
2023-06-02 18:27:07,944 INFO move SOUTH
2023-06-02 18:27:08,445 INFO move SOUTH
2023-06-02 18:27:08,805 INFO move EAST
2023-06-02 18:27:09,583 INFO move EAST
.
Sent 1 packets.
```

Figure 21: Terminal command and logs of the spoofing script started in manual mode with some drone movement

5.5 Extending the Application – LTE Extension

As defined in NFR7, the drone monitoring system must be designed in a way to be open for extensions by the developing community. To support a new transmission method, a sniffer can be added. To add a new Remote ID format, a new parser can be added.

In this section, the extension of the drone monitoring system is demonstrated using the example of LTE as transmission method. LTE has been chosen as an example because an open-source code example by Schiller et al. [7] already exists. This was advantageous because analysing and implementing the LTE functionality from scratch would not have been feasible in the limited time available. Furthermore, it is listed as an optional requirement (eREQ2).

Schiller et al. [7] analysed DJI's proprietary Remote ID and implemented a decoder for the DJI format version 2. The decoder can be used live or offline with pre-captured files. The live mode of the decoder was neglected when it came to integrating it into the drone monitoring system. Running the decoder in live mode not only requires an additional adapter but also a "quite powerful machine" [38], which the Raspi is not. Thus, the decoder can only be used whenever the drone monitoring system is started manually via a terminal and not in the UI.

Two DJI drone types, namely DJI Mavic Air 2 and DJI Mini 2, are used by Schiller et al. [7] to capture the raw data of the Remote ID. A sample file of each drone type exists in the repository to test the decoder [38]. The code in this repository was used as a basis. The files needed for the extensions were placed into a new folder called *lte* to separate the additional feature. The new modified project structure can be seen in Figure 22.

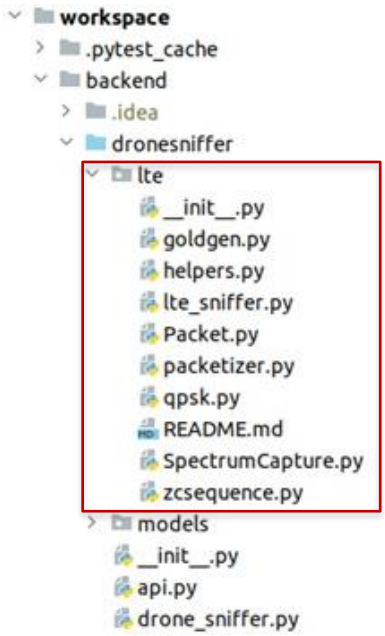

Figure 22: Project structure with LTE extension

The code was modified and adjusted where needed to be integrated into the drone monitoring system. The file *lte_sniffer.py* in Figure 22 is the main script, which requires the other files in the same folder to work properly. As the decoder works with DJI's proprietary format version 2, which is already integrated in the drone monitoring system, no additional parser must be added to the code. However, since a new transmission method, LTE, is used, an additional sniffer is required.

A sniffer class must be added in the *sniffers.py* file. The implemented *LteFileSniffer* class is shown in Figure 23.

```python
class LteFileSniffer:
    """
    Parses a file containing LTE data and forwards the parsed packets to the handler.
    """

    def __init__(self, filename: str = None) -> None:
        """
        Args:
            filename (str): The filename of the file to be read and parsed.
        """
        self.filename = filename
        self.stop_event = Event()
        self.sniffer = Thread(target=lte_sniffer, args=(self.stop_event, filename,), daemon=True)

    def start(self) -> bool:
        """
        Reads the file and parses its content.

        Returns:
            bool: Always succeeds and returns True.
        """
        logging.info(f"Starting to parse LTE file {self.filename}")
        self.sniffer.start()
        return True

    def stop(self) -> None:
        """
        Stops all sniffing efforts.
        """
        logging.info(f"Stop parsing file {self.filename}")
        self.stop_event.set()
        self.sniffer.join()
```

Figure 23: Added sniffer class for adding LTE compatibility

The sniffer class requires the initialization of a thread in the initialization method, as well as a start and stop function to properly start and stop the sniffer. The thread requires a target function to initialize correctly. This function will then be called once the start of the thread is requested. In Figure 23 the target function refers to *lte_sniffer*, which is the LTE sniffer's main function.

Next, the *LteFileSniffer* class needs to be integrated into the *main.py* file. This step requires two adjustments: First, a new argument needs to be added to the *main.py* file to be able to start the script with LTE functionality. Second, the *SnifferManager*'s method *parse_file* in the *sniffer.py* file, which handles file inputs, needs to be modified to handle LTE files alongside pcap files.

The argument, in this case *-l* or *--lte*, can simply be added to the main script via the in Python built-in *argparse* library as shown in Figure 24.

```python
def parse_args() -> argparse.Namespace:
    """
    Parses and returns arguments passed to script.

    Returns:
        Namespace: Parsed arguments.
    """
    arg_parser = argparse.ArgumentParser(prog="Super cool Drone Monitor System")
    arg_parser.add_argument("-p", "--port", help="port", type=int, default=8080)
    arg_parser.add_argument("-f", "--file", help="pcap file name")
    arg_parser.add_argument("-l", "--lte", action="store_true", help="sniff on lte")
    return arg_parser.parse_args()
```

Figure 24: Adding an argument to the main.py file with the help of the argparse library provided by Python

The action argument “store_true” defines, that if the flag is provided in the terminal command, then the boolean value true is stored in the variable *lte*. If the flag is missing, then false will be saved by default.

The *lte* variable must then be evaluated in the main function as shown in Figure 25.

```python
def main():
    args = parse_args()
    port: int = args.port
    file: str = args.file
    lte: bool = args.lte

    logging.info("Setting up database...")
    setup_database()

    # register shutdown manager
    atexit.register(shutdown)

    try:
        if file or lte:
            logging.info(f"Started with file argument, starting parsing of {file}")
            sniff_manager.parse_file(file, lte=lte)

        logging.info("Starting sniff manager...")
        settings = get_settings()
        sniff_manager.set_sniffing_interfaces(settings.interfaces)

        logging.info(f"Starting API on port {port}...")
        uvicorn.run(app, host='0.0.0.0', port=port)
    except KeyboardInterrupt:
        sys.exit(0)
```

Figure 25: Evaluating the newly added argument lte in the main function

And lastly, the method *parse_file* of the *SnifferManager* class needs to be adjusted. As mentioned in Section 5.1.1 the *SnifferManager*, or more precisely, the *parse_file* method handles pcap files. Therefore, this method can be reused since it already handles the filename input. The modified method is depicted in Figure 26.

```python
def parse_file(self, filename: str, lte: bool) -> None:
    """
    Starts a FileSniffer for file with filename. If lte is True a LteFileSniffer is started, otherwise a
    WiFiFileSniffer is started.

    Args:
        filename (str): Filename of the file to be parsed.
        lte (bool): If lte extension is used.
    """
    if lte:
        logging.info("Creating LTE Sniffer...")
        sniffer = LteFileSniffer(filename)
    else:
        logging.info("Creating Wi-Fi Sniffer...")
        sniffer = WiFiFileSniffer(filename)
    self.file_sniffers.append(sniffer)
    sniffer.start()
```

Figure 26: Integrated lte argument and LteFileSniffer class into SnifferManager's parse_file method

The LTE extension is now complete and can be used in the command line by simply using the *--lte* flag in combination with the file flag as shown in Figure 27.

```
user@ubuntu:~/DroneId-Monitoring/workspace$ sudo python3 ./backend/dronesniffer/main.py -l -f resources/lte/mini2_sm
```

Figure 27: Terminal command to start the LTE extension with a specified file

5.6 Extending the Application – Spoofing Detection Mechanism

A spoofing detection mechanism, as mentioned in eREQ3, was implemented into the system. The current logic is simple. It compares the drone's and the pilot's position. If the value of the distance is greater than 15 km, the drone is marked with an exclamation mark as suspicious. Figure 28 displays such a scenario.

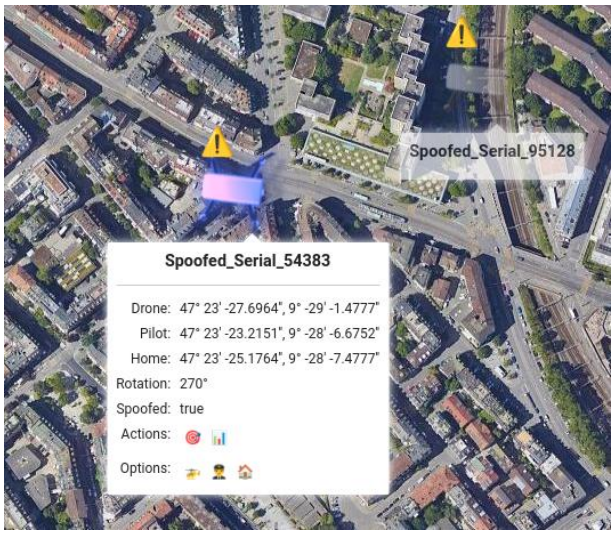

Figure 28: Example of a drone marked as possibly spoofed

The value of 15 km was set based on the maximum possible distance of the DJI Air 2S drone (12 km) to the pilot according to the technical specifications [39]. However, this fixed value is error-prone, as the maximum possible drone-pilot distance may vary depending on the drone model.

Using the distance between the drone and the capturing device (i.e., the Raspi) was also considered to improve the spoofing detection accuracy. However, to determine the location of the Raspi, an additional GPS module is needed. It was no longer possible to organise such a module in the time available and was therefore not pursued further.

6 Testing and Evaluation

The most important factor of a monitoring system is its ability to monitor its target objects, particularly when the system is under stress. Thus, the system was tested and evaluated outdoors in a realistic scenario, across different browsers and devices, and against a spoofing attack. Furthermore, user experience tests for feedback on the installation and usage were conducted.

6.1 Testing With Drones

Testing the application with the Parrot Anafi Thermal drone bore some challenges, which are mentioned below, but was successful. This shows that all mandatory functional requirements listed in Table 2 are working with ASD-STAN's Remote ID. The test, illustrated in Figure 29, was conducted at Bullingerhof in Zurich with a Raspberry Pi 4 connected to an iPad Air 4th Generation (iOS-Version 16.4.1 (a)) and the drone (Software-Version 1.8.2) controlled via the FreeFlight6 App (version 6.7.5) by Parrot on an iPhone 11 Pro (iOS-Version 16.5).

Figure 29: Real Scenario of a detected and monitored Parrot Anafi Thermal drone

The first challenge involved the drone not reliably broadcasting its Remote ID. Most of the time it did not broadcast anything at all and occasionally, it just sent a few initial packets after start-up and then stopped. The root cause could not be identified. A suspected issue might be the start location of the drone, as the testing location was in a restricted zone (no flying above 120 m). However, this would not explain why it sometimes did broadcast a Remote ID. Another possibility could be a flaw in the Parrot application or some sort of misconfiguration. None of the causes could be verified in the time given.

Second, with the full version of the Raspberry Pi OS installed, the Raspi was not able to process most Remote ID packets because of the increased CPU load. While this could not be fully verified either, installing OS Lite solved the issue and the application worked flawlessly. OS Lite does not contain a desktop environment, but only essential tools. The OS Lite version does not bear any disadvantage over the full version but requires Python and pip to be installed alongside the application, as they do not come preinstalled like in the full OS version.

Despite the two issues, capturing the live drone was successful up to 30 meters (between drone and capture device). Drone, pilot and home location, rotation, and height were all received, parsed, and displayed by the application, as seen in Figure 29. This test verifies, that REQ1 (for AST-STAN), REQ2, REQ3, REQ4 and NFR2 are met. Beyond 30 metres, no Wi-Fi Beacon frames were received by either the Raspi or a laptop. A more powerful adapter with good antennas might be able to extend the reach. This hypothesis could not be verified due to temporal constraints. Therefore, NFR4 is currently not satisfied but could be achievable with the current software.

Upon completing the test flights, the requirements REQ5, REQ6 and REQ7, which all involved replaying previous flights, were verified and are satisfied to the fullest.

DJI drones, on the other hand, prove to be more difficult to test than anticipated. The tests are conducted with two different drones (i.e., DJI Mavic Air and DJI Air 2S). Neither DJI's proprietary Remote ID nor the ASD-STAN format could be captured from any of the drones with the application. This results in REQ1 not being fully satisfied, as DJI's Remote ID was only processed via pcap files and not captured live. The root problem, however, does not lie with the application but with the broadcast of the DJI drones. The following paragraphs describe the problems encountered and how they were attempted to be overcome.

As mentioned in Section 3.2.4, DJI developed and implemented a proprietary solution before the publication of the standards. The discontinued drone DJI Mavic Air is known to have the proprietary Remote ID solution implemented in its firmware [25]. In addition, Wi-Fi Beacon frames of said drone were successfully captured at CYD prior to this Bachelor thesis. Therefore, this drone was used to test DJI's proprietary solution.

An assumption, which could not be verified, for why no Wi-Fi Beacon frames from the DJI Mavic Air drone are broadcasted may be due to incompatible versions of the smartphone application, DJI GO 4, and the drone's firmware. The smartphone application is required to control the drone. For the drone to properly broadcast DJI's proprietary Remote ID, the drone's firmware must be compatible with the version of the smartphone application. Otherwise, according to TheDronestop.com [40] and Medhane [41], this may lead to no GPS signal. The GPS signal is required to determine geolocation, without which the broadcasting of the Remote ID transmission is pointless. No meaningful source was found for this statement, but it is based on logical conclusion. The latest firmware version on the discontinued DJI Mavic Air is v1.0.1.0. This version

is, according to the DJI Mavic Air Release Notes [42], compatible with the DJI GO 4 application version 4.2.5 for iOS. However, only the newest version of the DJI GO 4 application can be downloaded in the (Apple) App Store, which is version 4.3.50. Downgrading the application on Android worked, but the application broke when trying to connect to the drone. This incompatibility of the versions may be the reason for the missing Remote ID broadcast.

To eliminate this in further testing, the successor of the DJI Mavic Air, namely DJI Air 2S, was acquired. The DJI Air 2S supports the Remote ID standard according to its English version manual [39] and is approved by the U.S. Department of Transportation [43]. However, according to the DJI support, the Remote ID will only be broadcasted if the following requirements are met:

1. The drone is registered with the proper authorities [44]
2. The drone owner is resident in the US [44]
3. The drone flight is within US airspace [45]
4. The drone's motors are spinning [45]

Three of those four requirements could not be met within the constraints of this Bachelor thesis, and tests by spoofing the geolocation to within the US were unsuccessful. This shows that the first and/or second requirements must be met for the drone to broadcast the Remote ID. Thus, the focus was henceforth on testing the application with the Parrot Anafi Thermal drone.

6.2 Compatibility Across Devices

As stated in NFR8, compatibility with all major devices and systems is key to make the application accessible. A useful advantage of installing the system on a Raspi is its portability. Once set up, it can simply be plugged into most devices and is ready to go without requiring any client-side installation. Unfortunately, not all devices can connect to the Raspi via the USB C cable, which also delivers power to it. Devices not able to connect via cable, such as the Apple iPad, can be linked to the Raspi via a Wi-Fi hotspot that both devices connect to. From then on, the Raspi can be accessed via its hostname. Note that ".local" might need to be added to the hostname in case it does not work.

While the sniffing service runs on the Raspi, the UI is executed on the client side and must thus support a variety of different browsers. No breaking issues were detected on any of the tested major browsers, Safari on iPad, Chrome, and Firefox. On Safari, however, the highlighting of the drones in the same colour as the path does not work. Since this is a minor cosmetical issue, it was not further investigated or fixed.

6.3 User-Friendliness and Ease-of-Use

To verify whether NFR6 and NFR7 are satisfied, user experience tests were carried out on 7 June 2023. One tester works in IT security and was tasked with installing and setting up the

system with just the instructions in Section 5.3 given. While they did face challenges setting up the Raspi itself with the correct configuration, which is not further explained in Section 5.3, installing the system was successful [46].

Another tester, who works for the tax department and has no formal IT education, was given a laptop with the system already running and displayed in a Google Chrome browser. They were tasked to test the requirements of Table 2 without further instructions given. The overall reaction was good, and the application looks “cool” [47]. However, the following issues and improvements were pointed out:

- It is not clear whether the settings are saved or not, due to lack of feedback
- Emoji-buttons are very small
- An optional introduction tutorial or “?” button would be great
- The drone-pilot link and historic path look the same and can be confused, making it dashed might help
- Resizing the map does not work, when the browser size is adjusted

The UI is currently designed for desktops and tablets first. Therefore, user-friendliness suffers on a mobile device (e.g., iPhone 11 Pro and Fairphone 3). Most buttons are small, making them hard to accurately click, and control panels start to overlap on smaller screen sizes. However, the issue is negligible, since the system is not primarily designed for mobile devices, and work well on tablets. According to the testers, both NFR6 and NFR7 are fulfilled.

6.4 Performance

While the developed monitoring system is still susceptible to spoofing, tests show that its performance and responsiveness remain much longer intact compared to AeroScope, which crashed while monitoring 120 drones.

The evaluation of the developed system started at 10 spoofed, randomly-moving drones each sending updates in a 3 second interval. A single, manually-controlled drone with a 0.5 second update cycle was used to verify the systems responsiveness. By increasing the number of randomly-moving drones by increments of 10, the first slight delay occurred at 30 drones. This delay was only noticeable on the manually controlled drone as it moved more frequently than the others. The delay started to increase slightly after 80 spoofed drones, but the application was still functioning properly. Figure 30 shows a screenshot of 60 spoofed drones. It illustrates the issue that, with such a dense drone clustering, the UI becomes cluttered and individual drones are difficult to track.

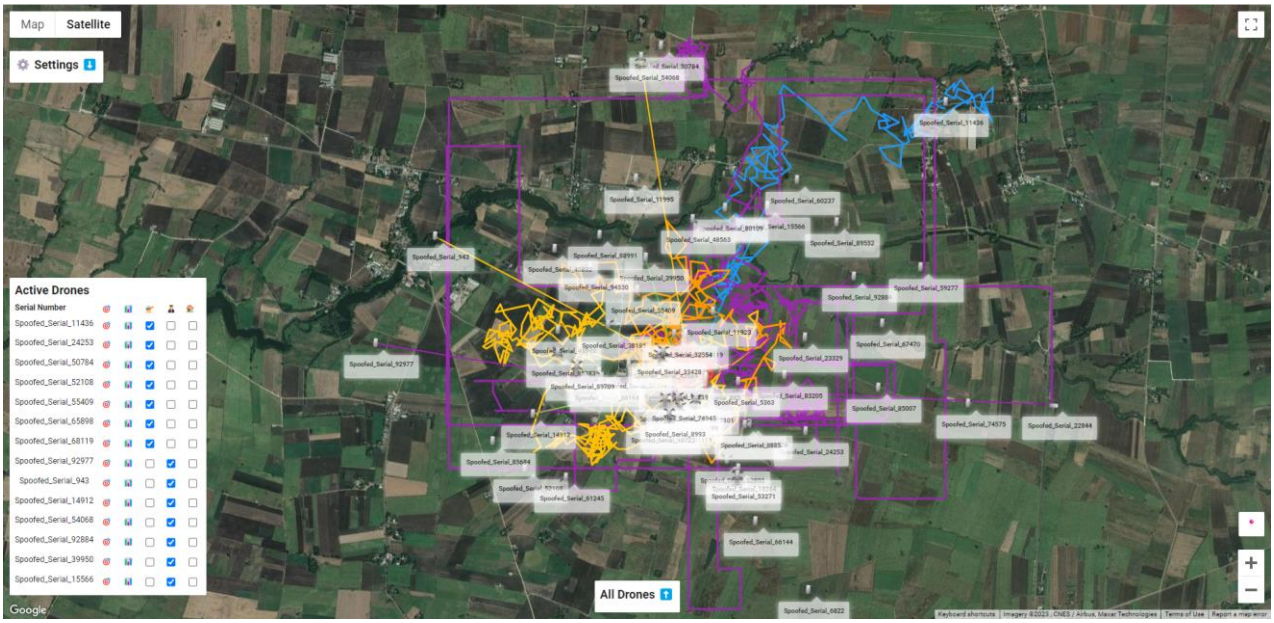

Figure 30: Satellite view of 60 spoofed drones

At 100 drones, the UI was struggling with updating the map and the drones with their paths and animations. Switching to the schematic map and turning on performance mode, as seen in Figure 31, helped immensely and allowed the system to comfortably handle up to 150 drones, successfully satisfying NFR1. Note that the drones in Figure 31 spread over an area of around 20 km, at which point the drones would be out of reach for even AeroScope to detect.

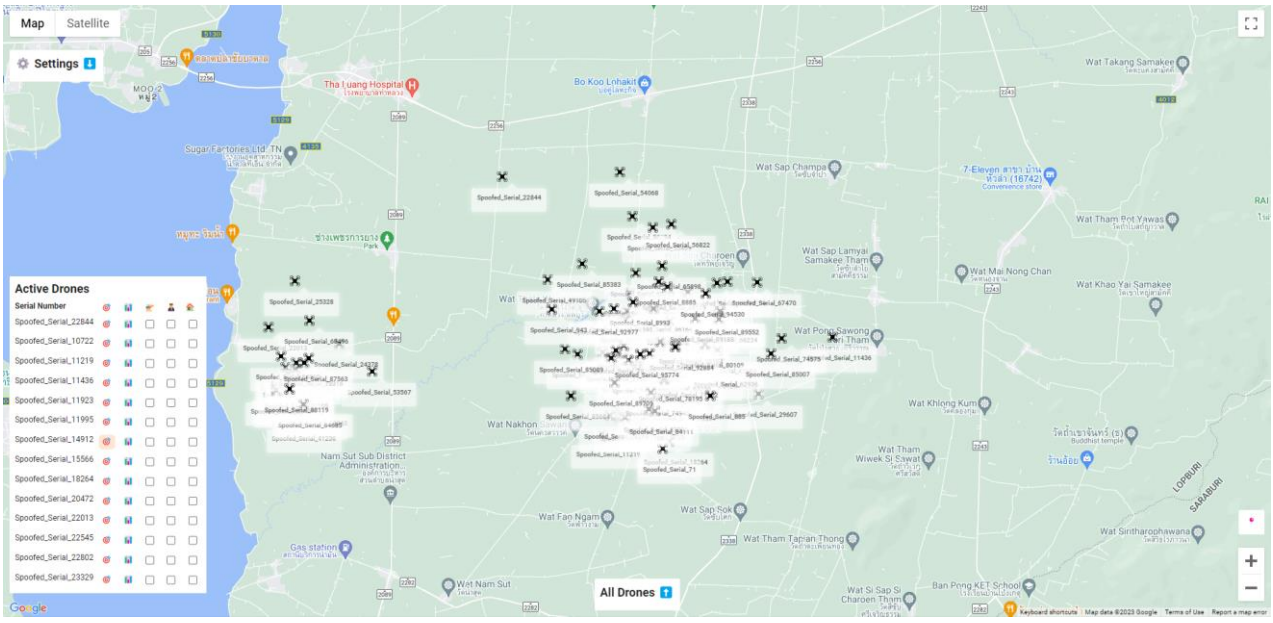

Figure 31: Schematic view with performance mode and 100 spoofed drones

To get a better idea of the scale, Figure 32 shows 150 spoofed drones around Los Angeles International Airport (LAX), which has an area of 14.77 km^2 [48].

Figure 32: 150 spoofed drones at Los Angeles International Airport (LAX)

A limit on the number of drones before the application stops working was detected at 500 spoofed drones. The application still works at first, but the responsiveness decreases with every Remote ID received until it stops working. At this point, the Raspi reboots itself. After a few seconds, the application is available again.

The evaluation demonstrates that the application is functional, resilient and may be used operationally. Most non-functional and functional requirements were met except for REQ1 and NFR3. Both requirements could, however, be achievable with the same software and the appropriate hardware but could not be verified within the remaining time.

7 Conclusion

In this chapter, results and the potential for future work are discussed. The system and the associated Remote ID are critically reviewed, and their limitations are pointed out.

7.1 Results

The goal of this Bachelor thesis was to develop an improved and affordable drone monitoring system with the aim to facilitate the monitoring of drones via the new Remote ID for authorities and law enforcement, but also ordinary users.

All mandatory **functional requirements** listed in Table 2 were fulfilled and successfully tested for the ASD-STAN format. The system successfully captures, parses and stores broadcasted Remote IDs and displays them on an interactive map. Live updates are sent to the UI and drones display their historic flight paths, pilot and home locations, and other details. The user can adjust various settings and customise the UI. Furthermore, all captured drone flights can be replayed upon request. Unfortunately, support for DJI's proprietary Remote ID was implemented but could not be

verified in a real scenario. Thus, REQ1 (support for DJI and ASD-STAN Remote ID) is only partially satisfied.

By design, the **non-functional requirements** NFR3 (hardware cost ≤ 10% of AeroScope) and NFR7 (open for extension) were achieved. The usage of a Raspi in combination with a Wi-Fi adapter and tablet or laptop makes the system significantly more affordable and accessible than AeroScope. At below 200 Swiss francs, the hardware costs of the Raspi and the EDIMAX Wi-Fi adapter meet the required maximum of 790 Swiss francs. Even a better Wi-Fi adapter with increased sniffing range would be within budget. The code is open for extensions as shown in Section 5.5. The evaluation of the system verified that NFR1 (handle ≥ 150 drones without crashing), NFR2 (real-time updates within 1 s), and NFR8 (support all major devices) are all met. Only NFR4 (range of 500 m) was not achieved. With the devices used in this Bachelor thesis, only an effective range of 30 m was measured. This is significantly below the required 500 m but could be improved by using an adapter with a good antenna.

Through user tests, NFR5 (simple installation process) and NFR6 (easy usage) were verified to be met as well. Although the UI could be improved, no major issues were reported by the testers.

All **optional extensions** are at least partially addressed. The option to read and parse pcap files (eREQ1) is available when starting the application via a command line. This is useful for development purposes but not available to the end user. Support for LTE (eREQ2) was added experimentally, but live capturing of LTE traffic is not supported. However, like pcap files, captured raw data files can be loaded and parsed. A minimal anti-spoofing detection (eREQ3) was implemented as well. While it does only check the bare minimum, the detection mechanism can be extended in future versions.

The extension "Drone-mounted Remote ID spoofer", which is mentioned in the assignment in Appendix B, was omitted altogether. A simple solution for this would be to tape a Raspi running the spoofing script in random mode to a drone.

To summarise the statements above, Table 8 lists all functional and non-functional requirements alongside their completion state.

Table 8: Summary of all functional and non-functional requirements

Identifier	Requirement	State
Mandatory requirements		
REQ1	Support DJI's proprietary and the ASD-STAN Remote ID	Support is implemented for both; only ASD-STAN was verified with live drone
REQ2	Display all currently in-flight drones on a map	Fulfilled
REQ3	Update the live position of currently in-flight drone upon receiving an update	Fulfilled
REQ4	Selectively display historic flight path, pilot, and home location of drone	Fulfilled
REQ5	Display a list of all drones that were ever captured	Fulfilled
REQ6	Replay a flight of a previously captured drone	Fulfilled
REQ7	Details for a replayed or in-flight drone can selectively be displayed	Fulfilled
Optional extensions		
eREQ1	Read Remote IDs from captured pcap files	Only supported via command line; not available to end users
eREQ2	Support Remote IDs broadcasted via LTE	Only files supported via command line; no live capture; not available to end users
eREQ3	Detect spoofed Remote IDs	Only simple mechanism implemented; open for extensions
Non-functional requirements		
NFR1	Continuously display at least 150 in-flight drones sending updates in 3 s intervals, as defined in the ASD-STAN standard, without crashing	Fulfilled
NFR2	Update in-flight drones with live data within 1 s upon receiving a Remote ID	Fulfilled
NFR3	Keep hardware costs under 10% of the costs of AeroScope (≤ 790 Swiss francs)	Fulfilled
NFR4	Monitor drones within a 500 m radius (given a line of sight / no obstacles)	Not fulfilled, only 30 m radius supported with current hardware
NFR5	Keep the installation process simple; a user with basic IT knowledge (e.g., executing Linux commands) must be able to install the software without specific training	Fulfilled
NFR6	Make the usage of the application simple; a user without any IT knowledge or specific training must be able to use all key features described in Table 2	Fulfilled
NFR7	Keep the code open for extensions; especially parsing new formats and adding new sniffing sources should be possible for a developer with coding experience	Fulfilled
NFR8	Support all major devices, operating systems and/or browsers	Fulfilled, minor issues on Safari

Overall, the goal of this Bachelor thesis was achieved. The result can be seen in Figure 33. It contains a picture of the Raspi being powered by the iPad Air 4th Generation, and the application running in the browser, detecting a spoofed drone.

Figure 33: Drone monitoring system running Raspi and displayed on iPad Air 4th Generation

The developed system has a more responsive and easier-to-use UI, can detect spoofed drones on a simple level, and can monitor drones by different manufacturers. The last point is not only thanks to the Remote ID standards, which unifies the protocol for sending the data, but also because the system is designed to be easily extensible where needed.

Nonetheless, the system is not yet production ready for large scale drone surveillance. For instance, even though the system does detect that a drone might be spoofed, it is not yet fully protected against denial-of-service (DOS) attacks through drone spoofing. With a high tolerance for the number of drones that can be displayed simultaneously, the application can safely be used in smaller monitoring use cases.

Furthermore, the drone's distance to the capturing device (i.e., Raspi) poses an additional limitation. Depending on the surrounding, the Wi-Fi signal can be too weak to be caught by the application with a simple Wi-Fi adapter or the target itself could be too far away to be monitored.

Additionally, because the system is dependent on the transmission of the Remote ID, it is directly dependent on the drone owner, who must configure the drone correctly, and the manufacturers, who implement the Remote ID support into their drones.

For instance, a drone owner can simply buy a new drone and manipulate the drone, unintentionally through misconfiguration or maliciously, to no longer transmit the Remote ID. As a result, the drone will no longer be detected by the drone monitoring system at all.

The registration of the drone with the proper authorities poses a similar issue. Some drones will not send a Remote ID until the drone is properly registered. If this is not done by the drone owner, the drone cannot be detected by the system. This can be equated with a driving license check. By law, a car driver is required to take and pass a test before the person is allowed to drive. However, a person is not actively checked to see if they have a driving licence, neither when buying a car nor before every journey. Occasionally, checks are carried out, but the basic principle is that drivers are responsible for their own actions. This can also be applied to drones.

In addition, older drone models, that were built before Remote ID was introduced, can still be used. These drones cannot be recognised by the system because no Remote ID is transmitted.

In summary, the developed system can monitor drones on a smaller scale. The system provides an excellent foundation for further extension. It is user-friendly, easy-to-use, low-priced, and small in size. Therefore, it is accessible to a large community, including people without much technical know-how. It supports sniffing over Wi-Fi as well as parsing LTE files and can detect possibly spoofed drones. However, due to the system's limitations, it should not be used as the sole safety measure but only as a supporting measure.

7.2 Future Work

Throughout development and especially during evaluation, many ideas for new features emerged and some issues were discovered. Since software is never fully finished but time was limited, some improvements were left for future work.

No breaking issues were discovered during evaluation. However, several minor issues remained unfixed either because of a lack of importance compared to other issues or due to their complexity. The following list, which is not ranked in any way, records said issues:

- When replaying a drone that is also currently active, location updates disrupt replay (displace replaying drone)
- Occasionally, when the sniffing service is stopped, a background process does not terminate correctly thus blocking the application from properly shutting down
- Titles in *ActiveDroneList* and *AllDroneList* disappear when scrolling and it is not clearly visible when a list is scrollable
- All interfaces appear in Wi-Fi Sniffing interface setting (filter out non-Wi-Fi interfaces and interfaces that do not support monitor mode)
- Setup view appears when Google Map key is deleted in settings even without saving
- Contrast between drones, paths, and terrain can be insufficient in some constellations
- Pilot and home location markers are indistinguishable between different drones (same icon)

- The map style preference (schematic or satellite) does not persist and reset upon page refresh
- Whenever *AllDroneList* is empty, it would be more appropriate to display a “No drones found” text instead of not expanding the list
- The UI is not very mobile friendly as the buttons are too small
- The drone icons sometimes overlap in the UI and hide other content
- The settings panel does not show that there are unsaved settings
- It is unclear to the user when the settings have successfully been saved (a success toast could help there)

In addition to the minor issues listed above, the LTE extension could, unfortunately, not be tested in a realistic setting and therefore might hide potential errors or inaccuracies. Thus, it is recommended to test the LTE extension before it is fully integrated into the application and advertised as functional. Furthermore, the spoofing detection currently only performs minimal checks, which should be extended to increase detection accuracy.

Finally, a list of improvements or potential new features that were left out of scope is provided in the following list:

- Replay of not just one drone but a timespan with multiple drones (e.g., all drones captured the previous Thursday between 12:00 to 14:00)
- Directly upload and parse a pcap file via UI (currently only via command line supported)
- Display the logs of the service in the UI (significantly simplifies debugging)
- Do not process spoofed drones to prevent DOS attacks
- Add a help menu to explain the software and UI

8 Lists

8.1 List of References

[1] J. Alkobi, “The Evolution of Drones: From Military to Hobby & Commercial,” 15 January 2019. [Online]. Available: https://percepto.co/the-evolution-of-drones-from-military-to-hobby-commercial/. [Accessed 25 April 2023].

[2] Federal Aviation Administration, “Drones by the Numbers,” (n.d.). [Online]. Available: https://www.faa.gov/node/54496. [Accessed 25 April 2023].

[3] DroneSec, “Weekly Threat Intelligence,” *Notify,* no. 160, p. 11, Jan. 2023.

[4] Department 13, “White Paper: Anatomy of DJI’s Drone Identification Implementation,” AUS., Manuka, 2017.

[5] DJI, “DJI AEROSCOPE,” (n.d.). [Online]. Available: https://www.dji.com/aeroscope. [Accessed 17 March 2023].

[6] ASD-STAN, *Direct Remote ID Introduction to the Eruopean UAS Digital Remote ID Technical Standard,* Brussels, 2021, p. 13.

[7] N. Schiller, M. Chlosta, M. Schloegel, N. Bars, T. Eisenhofer, T. Scharnowski, F. Domke, L. Schönherr and T. Holz, “Drone Security and the Mysterious Case of DJI's DroneID,” 2023.

[8] Dronetag, “Drone Scanner,” (GitHub Repository), 15 March 2023. [Online]. Available: https://github.com/dronetag/drone-scanner. [Accessed 17 March 2023].

[9] Open Drone ID, “OpenDroneID Android receiver application,” (GitHub Repository), 17 January 2023. [Online]. Available: https://github.com/opendroneid/receiver-android. [Accessed 17 March 2023].

[10] B. Dall'Omo, “Evaluation of practical attacks on drone monitoring device,” Armasuisse Science + Technology, Thun, CH, 2023.

[11] copters.eu, “DJI Aeroscope Mobile Station,” (n.d.). [Online]. Available: https://www.copters.eu/remote-controllers/1030-dji-aeroscope-mobile-station.html. [Accessed 5 June 2023].

[12] heliguy, “DJI Transmission Systems – Wi-Fi, OcuSync & Lightbridge,” 27 May 2022. [Online]. Available: https://www.heliguy.com/blogs/posts/dji-transmission-systems-wi-fi-ocusync-lightbridge. [Accessed 8 June 2023].

[13] SkyLab, “Aeroscope and what it sees!,” SkyLab, 9 June 2022. [Online]. Available: https://www.youtube.com/watch?v=eJhONTxUQ00. [Accessed 26 May 2023].

[14] HELIGUY.com, “DJI AeroScope Image,” 17 July 2019. [Online]. Available: https://www.facebook.com/heliguydotcom/photos/a.146719441508/10159703105116509/?type=3&theater. [Accessed 25 May 2023].

[15] L. Romà, *private communication,* June 2023.

[16] Federal Aviation Administration, “UAS Remote Identification,” 15 March 2023. [Online]. Available: https://www.faa.gov/uas/getting_started/remote_id. [Accessed 17 March 2023].

[17] ASTM, “Standard Specification for Remote ID and Tracking (ASTM F3411-22a),” 13 July 2022. [Online]. Available: https://www.astm.org/f3411-22a.html. [Accessed 17 March 2023].

[18] ASD-STAN, “ASD-STAN prEN 4709-002 Corrigendum 1,” (n.d.). [Online]. Available: https://asd-stan.org/downloads/pren-4709-002-corr/. [Accessed 6 June 2023].

[19] C. Hegner, “Ausnahmen von den Bestimmungen über den Betrieb von unbemannten Luftfahrzeugen,” 19 January 2023. [Online]. Available: https://www.fedlex.admin.ch/eli/fga/2023/96/de. [Accessed 6 June 2023].

[20] *IEEE Standard for Information Technology--Telecommunications and Information Exchange between Systems - Local and Metropolitan Area Networks--Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications,* IEE, New York, NY, 2021, pp. 212, 897, 945, 1059.

[21] V. Gupta and M. K. Rohil, “Information Embedding in IEEE 802.11 Beacon Frame,” *IJCA,* no. 3, pp. 12-16, Nov. 2012.

[22] IEEE, “IEEE Registration Authority,” (n.d.). [Online]. Available: https://standards.ieee.org/faqs/regauth/. [Accessed 18 March 2023].

[23] IEEE, “MAC Address Block Large - OUI,” (n.d.). [Online]. Available: https://standards-oui.ieee.org/. [Accessed 28 April 2023].

[24] IEEE, “Company ID - CID,” (n.d.). [Online]. Available: https://standards-oui.ieee.org/cid/cid.txt. [Accessed 28 April 2023].

[25] C. Bender, “DJI drone IDs are not encrypted,” The University of Tulsa, Tulsa, OK, 2022.

[26] Wireshark Foundation, “WLAN (IEEE 802.11) capture setup,” 11 August 2020. [Online]. Available: https://wiki.wireshark.org/CaptureSetup/WLAN. [Accessed 2 June 2023].

[27] EDIMAX, “150 Mbit/s Wireless IEEE802.11b/g/n nano USB Adapter EW-7811Un,” (n.d.). [Online]. Available: https://www.edimax.com/edimax/merchandise/merchandise_detail/data/edimax/de/wireless_adapters_n150/ew-7811un/. [Accessed 6 June 2023].

[28] The Tcpdump Group, “LIBPCAP 1.x.y by The Tcpdump Group,” (GitHub Repository), 18 May 2023. [Online]. Available: https://github.com/the-tcpdump-group/libpcap. [Accessed 18 May 2023].

[29] G. Lyon, “Npcap,” (n.d.). [Online]. Available: https://npcap.com/. [Accessed 18 May 2023].

[30] SQLite, “Limits In SQLite,” (n.d.). [Online]. Available: https://www.sqlite.org/limits.html. [Accessed 12 March 2023].

[31] Google, “Pricing that scales to fit your needs,” (n.d.). [Online]. Available: https://mapsplatform.google.com/pricing/. [Accessed 6 June 2023].

[32] S. Ramírez, “FastAPI,” 16 May 2023. [Online]. Available: https://fastapi.tiangolo.com/. [Accessed 26 May 2023].

[33] S. Ramírez, “SQLModel,” 30 August 2022. [Online]. Available: https://sqlmodel.tiangolo.com/. [Accessed 26 May 2023].

[34] E. San Martin Morote, “Pinia,” 18 May 2023. [Online]. Available: https://pinia.vuejs.org/. [Accessed 26 May 2023].

[35] Inocan Group, “vue3-google-map,” (GitHub Repository), 3 September 2022. [Online]. Available: https://github.com/inocan-group/vue3-google-map. [Accessed 25 March 2023].

[36] A. Deveria, “Can I use... Import maps,” (n.d.). [Online]. Available: https://caniuse.com/import-maps. [Accessed 25 March 2023].

[37] n. cAMel, “On Raspberry Pi WiFi monitor mode sniffing - not getting data packets,” StackOverflow, 26 January 2016. [Online]. Available: https://stackoverflow.com/a/35012796. [Accessed 30 May 2023].

[38] RUB-SysSec, “Drone-ID Receiver for DJI OcuSync 2.0,” (GitHub Repository), 10 March 2023. [Online]. Available: https://github.com/RUB-SysSec/DroneSecurity. [Accessed 10 May 2023].

[39] DJI, “DJI AIR 2S - User Manual,” October 2022. [Online]. Available: https://dl.djicdn.com/downloads/DJI_Air_2S/DJI_Air_2S_User_Manual_v1.2_en.pdf. [Accessed 2 May 2023].

[40] TheDronestop.com, “(FIXED) DRONE WON’T CONNECT TO GPS & GPS ERRORS. (WHY, HOW TO FIX),” 12 April 2022. [Online]. Available: https://thedronestop.com/drone-wont-connect-to-gps/. [Accessed 2 May 2023].

[41] s. medhane, “No GPS Signal on New Mavic Mini Drones,” 21 November 2021. [Online]. Available: https://lccsdrone.com/no-gps-signal-on-new-mavic-mini/. [Accessed 2 May 2023].

[42] DJI, “Mavic Air Release Note,” 2018. [Online]. Available: https://dl.djicdn.com/downloads/Mavic%20Air/Mavic_Air_Release_Notes_en.pdf. [Accessed 3 June 2023].

[43] U.S. Department of Transportation, “UAS Declaration of Compliance,” 7 September 2022. [Online]. Available: https://uasdoc.faa.gov/listDocs/RID000000007. [Accessed 2 May 2023].

[44] Federal Aviation Administration, “How to Register Your Drone,” 11 April 2023. [Online]. Available: https://www.faa.gov/uas/getting_started/register_drone. [Accessed 2 May 2023].

[45] DJI, “What are the conditions for a drone to start broadcasting Remote ID signals?,” 18 March 2023. [Online]. Available: https://forum.dji.com/forum.php?mod=viewthread&tid=286879&page=1#pid3003932. [Accessed 2 May 2023].

[46] A. Joss, *private communication,* June 2023.

[47] L. Mosimann, *private communication,* June 2023.

[48] Los Angeles International Airport, “Airport Basics,” LAWA, (n.d.). [Online]. Available: https://www.lawa.org/lawa-our-lax/airport-basics. [Accessed 31 May 2023].

8.2 List of Figures

8.3 List of Tables

8.4 List of Equations

8.5 List of Algorithms

8.6 List of Abbreviations

ASD-STAN	AeroSpace and Defence Industries Association of Europe - Standardization
ASTM	American Society for Testing and Materials
CDN	Content Delivery Network
CYD	Cyber-Defence Campus
GPS	Global Positioning System
OUI	Organizationally Unique Identifier
Raspi	Raspberry Pi
UI	User Interface
UUID	Universally Unique Identifier

9 Appendices

Appendix A Extract of ASTM F3411-19

TABLE 7 Encoding Table

Field Type	To Encode (sender)	To Decode (receiver)
Direction	If Value <180 EncodedValue = Value Set Direction Segment bit to 0 else EnclodedValue = Value – 180 Set Direction Segment bit to 1	if Direction Segment bit = 0 Value = EncodedValue else Value = EncodedValue + 180
Speed (UInt8)	If Value- ≤ 225*0.25 EncodedValue = Value/0.25[A] Set Multiplier Flag to 0 else if Value > 225*0.25 and Value <254.25 EncodedValue = (Value – (225*0.25))/0.75[A] Set Multiplier Flag to 1 else (Value ≥ 254.25 m/s) EncodedValue = 254[A] Set Multiplier Flag to 1	If Multiplier Flag = 0 Value = EncodedValue * 0.25 else if Multiplier Flag = 1 Value = (EncodedValue * 0.75) + (255*0.25)[B, C] Encoding Rationale: This allows for a higher speed precision of 0.25 m/s for lower speeds (0.5 kts) and 0.75 m/s (1.5 kts) at higher speeds.
Lat/Lon	(Int32) EncodedValue = Value * 10^7 Default/Unknown: 0,0	(Double) Value = EncodedValue / 10^7
Vertical Speed	(Int8) EncodedValue = Value / 0.5	(Float) Value = EncodedValue * 0.5
Altitude	(UInt16) EncodedValue = (Value + 1000) / 0.5 Unknown: –1000, encode as 0	(Float) Value = (EncodedValue * 0.5) – 1000 Encoding Rationale: Eliminated unused negative integer space and increases precision to 1/2 m If decoded Value = –1000, then real value is unknown

Time Stamp	(UInt16) Encoded Value = Tenths of seconds since current hour	if Encoded Value > Tenths of seconds since the current hour at time of receipt, then
		ValueTenths = tenths of seconds since previous hour
		else
		ValueTenths = tenths of seconds since current hour
		Value = Current GMT Date/Time + ValueTenths
		This is the "Time of Applicability"

[A] Encoded Value must be rounded to nearest Integer.
[B] If value decodes to 255, then an unknown value is being represented.
[C] If value decodes to 254.25, then speed is at least 254.25.

Bachelor Thesis FS 2023

Students: Sebastian Brunner, Fabia Müller

Advisor: Prof. Dr. Marc Rennhard, Llorenc Roma (Cyber-Defence Campus)

Start: 13. February 2023 Credits: 12 ECTS

End: 9. June 2023

Drone Monitoring System

Background

With a massive rise in the number of commercial drones, the skies have become an increasingly dangerous place, and drones' remote identification (RemoteID) is crucial for security. RemoteID is the ability of a drone in flight to provide identification and location information that can be received by other parties. RemoteID creates a common and consistent way for authorities to monitor airborne drones and identify who is flying them.

As one of the main players in the commercial drone market, company DJI developed its own RemoteID solution, and many DJI drones already implement this feature. In addition, DJI has also developed a system called AeroScope, which can quickly identify all DJI drones with RemoteID implemented. AeroScope is meant to be used as a protection mechanism for highly critical facilities, e.g., prisons, airports, and governments. Moreover, this device was recently seen in use within the Ukraine-Russia conflict, where one party used AeroScope to identify the other party's drone pilot’s position. At the same time, the Ukrainian army claimed that their DJI AeroScope devices had been patched to NOT show the Russian drones (nor the pilot’s positions).

Previous Work

Previous research investigated how the RemoteID feature is implemented in WiFi-based DJI drone models and reversed engineered it. Based on this research, the Cyber-Defence Campus developed a simple RemoteID spoofer software that could deceive Aeroscope successfully. Hence, such a spoofer could be used as a countermeasure to DJI’s drone monitoring system AeroScope.

In a second phase, the spoofer was evaluated further against AeroScope. This included spoofing multiple drones on a location, spoofing random pilot locations, and spoofing drones in motion. At this point, it could be confirmed that AeroScope does not have any protection against fake RemoteID information.

Goals and Task

The goal of this bachelor thesis is to develop a drone monitoring system that implements functionality similar to AeroScope: receive and display drone RemoteID information. This is possible due to RemoteID information not being encrypted and transmitted over 802.11 (WiFi) broadcast packets.

Despite AeroScope being focused on DJI drones, the RemoteID feature is also implemented by other manufacturers such as Parrot. According to the USA, all the manufacturers will have to implement this feature. Therefore, the developed drone monitoring system should be able to monitor drones of at least one additional drone manufacturer, Parrot. The Parrot RemoteID version is to be compliant with European regulations, and there exists software able to receive it, which can be used as a code reference.

Task Details

In the context of this bachelor thesis, the following main task should be completed:

- Familiarization with DJI RemoteID and Parrot RemoteID
    - Study how the RemoteID is transmitted (data formats and fields).
    - Study the 802.11 standard and how beacon packets are leveraged to transmit RemoteID information.
- Identify AeroScope features
    - Analyze AeroScope’s functionality as a basis to replicate it in the new drone monitoring system.
- Software development
    - Plan, design, implement and test an Aeroscope-like drone monitoring system. The software should be capable of monitoring drones that transmit RemoteID information. It should display information about each of the detected drones in real time and store it for showing and replaying observed situations later (history time).
- Documentation

Depending on the progress and challenges encountered with the main task, there are several extensions for the project to be considered. It is expected that at least one of them can be partially addressed during this project.

- Support for DJI LTE RemoteID
    - Develop functionality for receiving RemoteID information over LTE.
- Drone-mounted RemoteID spoofer
    - Implementation of the RemoteID spoofer into a hardware board that can be mounted (taped) on drones. This can be used as a countermeasure against AeroScope (and other drone monitoring systems) by spoofing multiple drones around the real drone, making the real drone harder to identify by AeroScope.
- Implement anti-spoofing detection mechanisms
    - Survey and add/implement/test functionality that helps with detecting fake drones. For example, compare position information with signal directional information (see, e.g., Wi-Fi Direction Finding with Frequency-Scanned Antenna and Channel-Hopping Scheme, IEEE Sensors Journal, Volume: 22, Issue: 6, March 2022).

Deliverables

- All developed software components. The software should have a good quality level (good design, good code quality, good readability) and be further usable, but it does not have to be production ready.
- The written report.
- A presentation of the work. The final presentation will be held in front of an external expert and the advisors and takes place after the thesis submission deadline during the official exam weeks.

General Remarks

- Regular meetings are held with the advisors to discuss the progress of the work and open questions.
- After one week, the students must have prepared a project plan that contains the planned activities throughout the project work, including the most important milestones. This plan is to be discussed with the advisors.
- The written report must be in English and must include an abstract in both German and English. The report must clearly and comprehensibly describe all steps and important decisions that were made during the thesis and the results that were achieved. In addition, it must include this thesis assignment document and the project plan in the appendix. The report must also be accompanied by a data carrier or a data repository containing the report in source and PDF format and all developed software components.

Appendix C Project Management

Milestones	Deadline	In Charge	Comment
Capture and filter Wi-Fi Beacon frames (with Python or Pcap4J) - POC	4.3.2023	Sebastian	
Analyse AeroScope features and derive features (and improvements) for improved system	4.3.2023	Fabia	
Define basic architecture and make basic setup (repository, frameworks, language, modules, database)	5.3.2023	Both	Decision point, decide who does what and how in the future
Sniff, filter, and parse packets and store the info in database	26.3.2023	Fabia	
Design and implement first UI version (e.g., with Google Maps API, d3.js, etc.)	26.3.2023	Sebastian	
Load and display data in UI (replay up to current, no pushes)	2.4.2023	Sebastian	
Real time pushes of drone updates to UI	16.4.2023	Sebastian	Fabia will analyse the captured Parrot packet & Sebi will in time try to get the Mavic Drone to work (with older Android App) Should we get a new DJI drone?
Package application running on Raspberry Pi	23.04.2023	Both	
Research and decide for extension (LTE, Wi-Fi direction info, on-drone spoofer)	30.4.2023	Both	Another decision point: (LTE: https://github.com/RUB-SysSec/DroneSecurity)
Develop extension(s)	28.5.2023	Both	LTE file parsing and simple spoofing detection will both be implemented as extensions
Beautify documentation and prepare presentation	9.6.2023	Both	

Appendix D GitHub Repository Structure

Repository Link:
https://github.com/cyber-defence-campus/2023_Mueller-Fabia_Brunner-Sebastian_DroneID-Monitoring (might require access by CYD)

Tagged version (v1.0) at the date of submission:
https://github.com/cyber-defence-campus/2023_Mueller-Fabia_Brunner-Sebastian_DroneID-Monitoring/releases/tag/v1.0

The repository structure was given by CYD. Most folder are empty. The important ones are:

- *workspace*: contains all code including install scripts, frontend and backend
  - *backend:* all backend Python code
    - *dronesniffer:* code for the drone sniffer
    - *tests:* all unit tests for the backend
  - *frontend:* all frontend source files
    - *css:* CSS style sheets
    - *img:* raw images
    - *js:* Javascript source files
  - *dsniffer.service:* template for the system service
  - *install.sh:* install script
  - *spoof_drones.py:* spoofing script
- *report*: contains this report in PDF and Word format
- *presentation*: will contain the presentation as soon as it is finished