Norwegian University
of Life Sciences

Master's Thesis 2024 30 ECTS
Faculty of Science and Technology

"DREAMS" Drone Research and Affordable Mapping Solutions: Integrating DJI Tello with ORB-SLAM3

Jon Eirik Augensen
Applied Robotics

Acknowledgments

I am pleased to present my thesis, which is the culmination of my journey as an Applied Robotics student at the Norwegian University of Life Sciences. These years have been both challenging, exciting, and enriching. Even though this endeavor sometimes seemed farfetched or quite frankly impossible, I have accomplished this journey by having the "right" people by my side both outside of- and at the university. Therefore I want to start my thesis by extending gratitude to a select few who helped me along the way.

First, I would extend my most sincere thanks to my girlfriend, Kristin, for staying by my side and supporting me during the completion of this thesis and the long hours studying throughout my time at the University, effectively being my most needed pillar. Your love and encouragement have meant a lot to me and filled me with energy as I worked through seemingly never-ending writing and research to finish this work, even though I wasn't always aware of it during my most conceited or self-absorbed moments.

To my child, Henrik, my two bonus kids, Espen and Frank, thank you for filling our home with laughter. Although, sometimes frustrating, the beautiful chaos you bring is necessary and something I will cherish forever. The joy you guys provide is most dear to me, and I consider myself privileged to have you all in my life.

To my Supervisor, Weria Khaksar, your commitment, guidance, and insights have proven to be invaluable in the making of this thesis. Thanks for being of huge help, and aiding me in the darkest of hours during this thesis journey, leading to a pivotal decision and the shift in research focus to its current form.

To my friend and study buddy, Lars Øvergård, thank you for being a trusted companion throughout these five years of studying. From collaborating on numerous projects to endless study sessions. Thank you for going along with my shenanigans ranging from spending hours in Paint just to perfect a shading effect for a project's front page, to allowing the design of "Jaws-ome", our remote-controlled shark. The time in between study sessions filled with Bloons TD6 has been a most welcome stress relief.

To EikLab at NMBU and its affiliates, thank you for providing me with a platform to further develop myself through much-needed practical work through various projects. The knowledge gained from these projects and the confidence you showed in me, helped me discover my true interests and it paved the way forward for me professionally.

Lastly, to my family, my friends, and any others who have directly or indirectly played a part in the completion of this thesis, and for continued support throughout my studies for all these years**: Thank you for your support**.

Jon Augensen

Jon Eirik Augensen, May 2024, Ås

Abstract

The expanding world of drone technology has led to many exciting possibilities in various fields, such as surveillance and monitoring. Equipping drones with different sensors, depending on the use case makes drones invaluable in many applications, especially for environment mapping. Many of these sensors, or the drones equipped with them are costly and high maintenance, which opens up for the creation and implementation of cost-effective and efficient mapping solutions, making it available for a much broader audience, also those that do not have big budgets.

By focusing on building and implementing a low-cost mapping solution, this dissertation demonstrates how SLAM technology can be integrated with low-price drones while achieving accurate maps across various scenarios and environments. Through extensive research, a solution to this challenge presented itself with the DJI Tello drone and the ORB-SLAM3 package. Despite its modest specifications, the Tello drone's user-friendly and affordable nature makes it an ideal candidate for such a system, and with its monocular camera, proved to be highly compatible with the ORB-SLAM3 algorithm.

The open-source nature of ORB-SLAM enables customization, therefore making it applicable in a variety of scenarios. The option to change the number of features to be extracted and change the thresholds, allows the system to perform well even in low-contrast areas.

During this project, the system has been tested in various scenarios and environments and as a result, had its limitations and challenges showcased such as limited flight time or poor performance in strong winds. Regardless of its limitations, the system proves its ability to create accurate maps and shows how much potential such a system holds.

The result was a package solution for performing ORB-SLAM on the DJI Tello drone, and the insights gained from research conducted in this thesis show that it is entirely possible to achieve high-quality mapping even with a limited budget.

In conclusion, this project introduces the development of a drone-based mapping system capable of creating highly detailed maps with just a drone equipped with a 720p pinhole camera. It lays the groundwork for future research, continually improving and enhancing the field of low-cost drone-based mapping systems.

“Controversial but fabolous, drones do a good job” – Martha Stewart [1]

Table of Contents

Table of Figures

List of Abbreviations

UAV	**Unmanned Aerial Vehicle**
SLAM	**Simultaneous Localization And Mapping**
ROS	**Robot Operating System**
GPS	**Global Positioning System**
IOT	**Internet Of Things**
VTOL	**Vertical Take-Off and Landing**
LIDAR	**Light Detection and Ranging**
UI	**User Interface**
SDK	**Software Development Kit**
FOV	**Field Of View**
VLOS	**Visual Line Of Sight**
PoC	**Proof of Concept**
AR	**Augmented Reality**

TABLE 1: TABLE CONTAINING RELEVANT ABBREVIATIONS

1. Introduction

Drones have become increasingly popular in a variety of industries, including military, agriculture, surveillance, mapping, and entertainment. A drone is a type of Unmanned Aerial Vehicle (UAV), that does not have an on-board pilot [2]. Being able to navigate through unknown or dangerous environments without the endangerment of human lives can prove to be invaluable in many scenarios. Among these drones, the DJI Tello drone stands out with its low cost and substantial community-contributed resources setting the stage for a wide array of possibilities when it comes to drone programming.

Initially, the journey began with reviewing and researching literature on drone swarm technology and different control algorithms, driven by a vision of creating a robust, scalable, and affordable system aimed for aiding in scenarios and situations where human lives could be at risk. The project was set to revolutionize the search and rescue scene as well as contribute to other surveillance operations. However, as the project progressed from the theoretical aspect to the physical applications, it became clear that the limitations of the selected drone were more substantial than originally thought, some of these challenges were fundamental obstacles that could not be bypassed within the project time frame. This sudden realization led to a strategic pivot towards the domain of Simultaneous Localization And Mapping (SLAM), an area ripe for significant contributions.

As this project evolved, so did the research, wanting to stay true to using the DJI Tello, the research shifted towards so-called visual-based SLAM using cameras. In the midst of this research, one particular source of inspiration emerged in the work of Aman Kumar Singh, a student from the Department of Aerospace Engineering at IIT Kharagpur, India. His work with the Tello seen through a video from his YouTube channel [3] and documented in the GitHub repository [4], led to the discovery of ORB-SLAMs third iteration, ORB-SLAM3, an open-source feature extraction-based SLAM algorithm. This algorithm was chosen after reviewing its documentation, it was found to be adaptable with monocular cameras, and therefore highly compatible with the Tello drone.

This aided the process of setting a clear end game for this project: **The development of a user-friendly, efficient mapping system without the need for expensive high-end equipment** making it accessible to a broad audience of practitioners and scholars alike.

1.1 Conceptual Framework and Research

The mind map presented in **Figure 1** is a visual aid used to present the general thought process, key concepts, and ideas surrounding this master thesis. In general, a mind map is a way to brainstorm or organize thoughts and ideas, providing some structure to the work. [5] The mind map is created using an online tool called whimsical AI, which is a platform designed to provide a fast and intuitive way to visually represent ideas, projects, or workflows. [6]

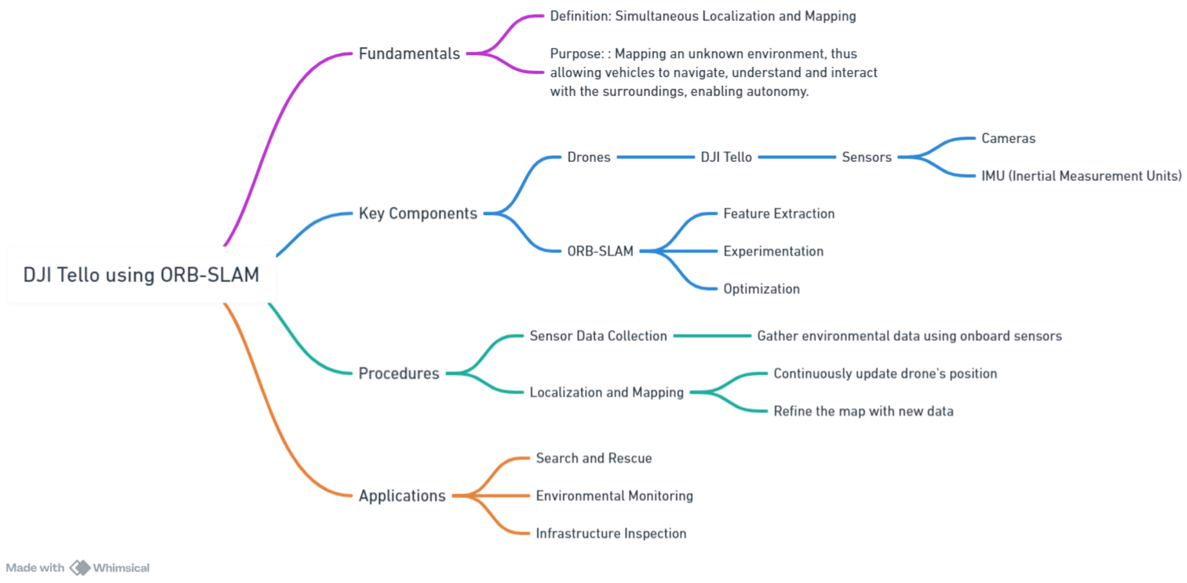

DJI Tello using ORB-SLAM
Fundamentals
Definition: Simultaneous Localization and Mapping
Purpose: : Mapping an unknown environment, thus allowing vehicles to navigate, understand and interact with the surroundings, enabling autonomy.
Key Components
Drones
DJI Tello
Sensors
Cameras
IMU (Inertial Measurement Units)
ORB-SLAM
Feature Extraction
Experimentation
Optimization
Procedures
Sensor Data Collection
Gather environmental data using onboard sensors
Localization and Mapping
Continuously update drone's position
Refine the map with new data
Applications
Search and Rescue
Environmental Monitoring
Infrastructure Inspection
Made with Whimsical

FIGURE 1: MINDMAP OVERVIEW OF THIS THESIS [6].

2. Drone Technology and Regulations

This chapter explores theories surrounding drones, from the most basic of principles, to the advantages and disadvantages of different types of drones and it briefly gives an overview of relevant regulations one might need to take into consideration when piloting drones in Norway. Additionally, it provides insight into the specific drone used in this project, the DJI Tello drone.

2.1 Fixed-Wing Drone

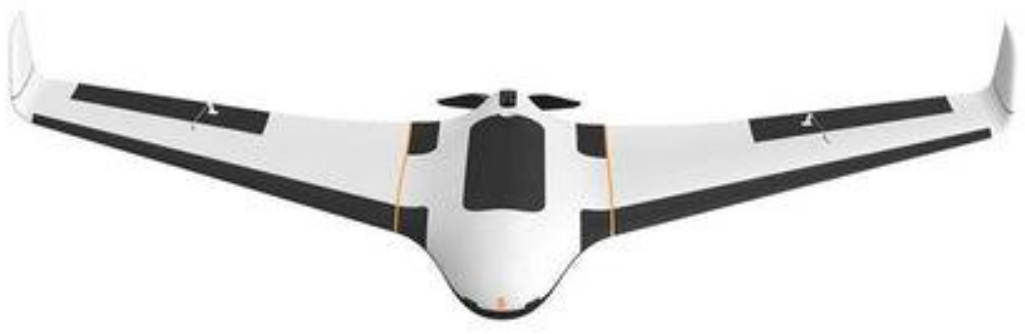


FIGURE 2: THIS FIGURE DEPICTS THE SKYWALKER FIXED-WING DRONE [7]

As the name suggests, the Fixed-Wing drone has rigid wings, designed to work much like an airplane, This type of drone only needs the energy to move forward providing lift rather than achieving lift from vertical rotors, which makes for a more energy-efficient drone. The means of forward thrust is usually gained by propellers.

In general, the main advantage of the Fixed-Wing is that it is capable of high-velocity flights, thus making it able to cover large areas more quickly as compared to other drones. Compared to other drones, like the multi-rotor drone, it can usually carry heavier payloads due to its robust frame, which is beneficial for several usage areas and enables the added weight of a gas tank for prolonged missions. [8]

The main disadvantages of this type of drone are that it takes more training to be flown efficiently and it is not ideal for operations that require frequent take-off or landings in varied terrain. Compared with the multi-rotor drone, it also has no VTOL or hover abilities and is a poor fit for tasks that require stationary observation or for operations in environments with no clear runway. [8]

The nature of this drone makes it ideal for operations when it comes to Aerial mapping over large areas or pipeline and power line inspections etc. Being able to carry heavier payloads also enables the user to add more sensors or bigger/better cameras to the drone.

2.2 Fixed-Wing VTOL Drone

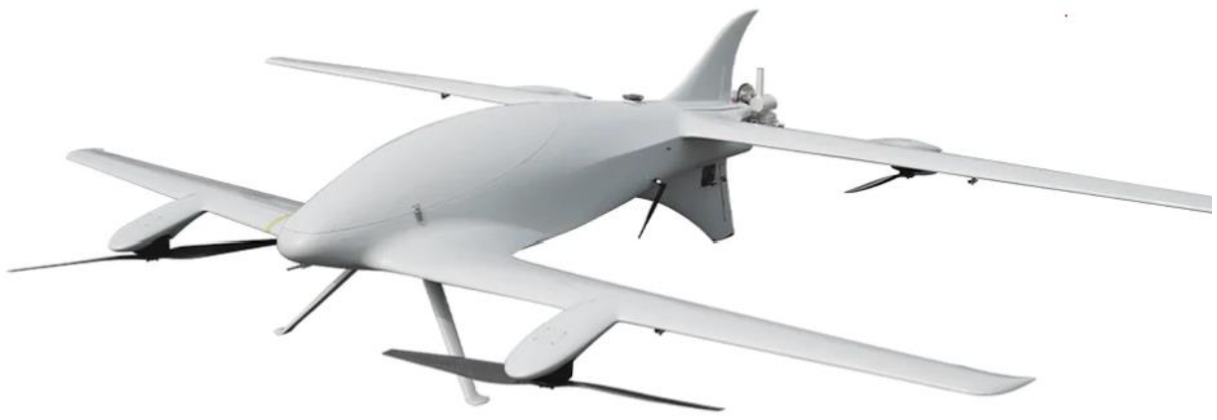


FIGURE 3: ONE TYPE OF VTOL DRONE, THE CUAV RAEFLY VTOL [9]

A hybrid drone aims to provide the best of both “worlds” of a multi-rotor drone and a Fixed-Wing drone. You get long-distance performance from the Fixed-Wing drone, and at the same time, it also provides the tools for precise positioning, hovering, and vertical flight.

The nature of this type of drone makes them perfect for tasks requiring both the ability of long-distance travel, as well as need for precision, the versatility of this drone is also its main feature. [8]

However, the complexity of this design and the cost associated with it makes it less affordable and accessible for smaller corporations and individuals alike. Additionally, by adding more components, the weight and/or size will also increase in contrast with the solutions. [8]

A hybrid drone is much like any other hybrid solution, it will never provide the best performance of either world, it will never reach the top speeds and long range of the regular Fixed-Wing solutions, nor will it be able to provide the same level of precision as a multi-rotor drone.

2.3 Single-Rotor Drone

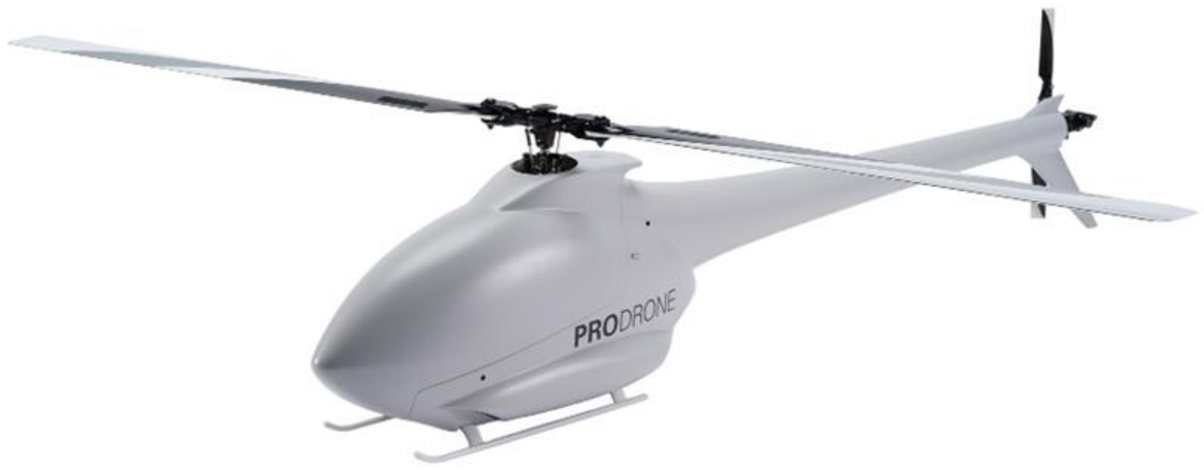


FIGURE 4: AN EXAMPLE OF SINGLE ROTOR DRONE, THE PRODRONE PDH-03 [10]

A Single-Rotor drone is often associated with a remote controlled helicopter, and in the context of UAVs, the terms are often used interchangeably. Throughout this section, it will be referred to as a Single-Rotor drone.

This type of drone makes it both more energy efficient and more power is generated in vertical flights than the multi-rotor drones. This design allows for precise maneuverability and hovering making them especially useful for tasks that require a mix of being able to carry larger payloads, with precise positioning. [8]

However, this type of drone is usually more complex than a multi-rotor drone and to fly efficiently in sensitive or populated areas, more training is needed. Having larger rotor blades also poses a safety hazard, particularly during take-off and landings in close proximity to humans [11], make sure that there is sufficient space for the drone to avoid accidents.

As previously mentioned, this type of drone excels in areas where carrying heavy payloads is a requirement, areas this drone can be ideal for include; delivering supplies in prolonged search and rescue operations, spraying in agriculture, or other scenarios where one needs to lift heavy equipment.

2.4 Multi-Rotor Drone

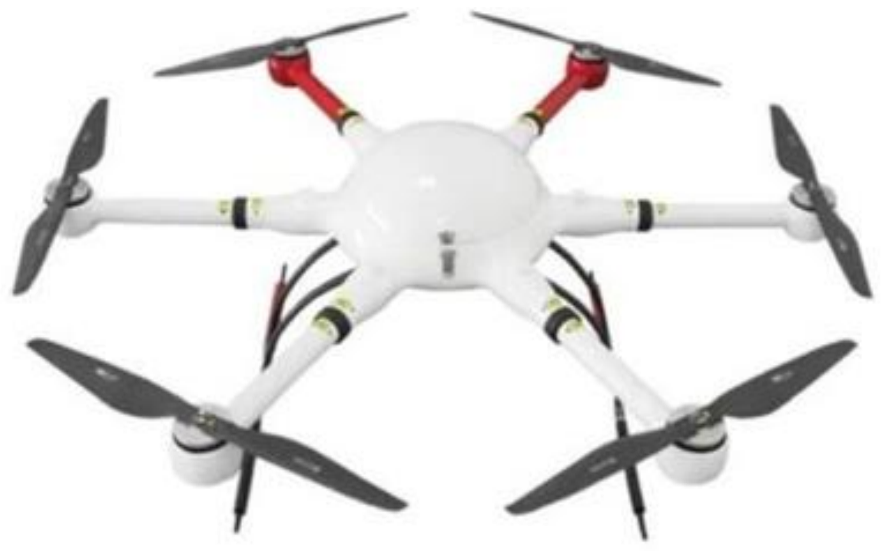


FIGURE 5: A MULTI-ROTOR DRONE, THE M6000WP MULTI-ROTOR DRONE. [12]

The Multi-Rotor drone is widely used for its stability enabling novices to achieve successful flight more effortlessly. The most common configuration is the quadcopter, which is equipped with 4 rotors.

The main advantage of this type of drone is its ease of use, stability, and maneuverability. The drone can perform VTOL and it is able to hover in place as well as flying sideways without changing its rotation. These drones are generally known to be easier to control due to the nature of their design and are also considered to be the most stable of the drone types. [8]

However, The energy required to keep the vehicle airborne is higher than other configurations and this type of configuration naturally comes at the expense of battery life, the drones are typically considered to have shorter flight times, especially compared with Fixed-Wing drones. The nature of this compact design naturally comes at the expense of the speed, which in turn limits its range. [8]

The design of this drone makes it ideal in areas like aerial photography, and it is widely used in film production and real estate capturing high-quality images and video.

2.5 Regulations by the Norwegian Civil Aviation Authority

The regulations are designed and set in place to ensure safety and privacy, they vary depending on the general use of said drone as well as the weight.

The weight of the drone used, and the use-case of this drone places it in the subcategory A1, more specifically, as it weighs significantly less than 250 grams and has a max velocity lower than 19 m/s, there is no need to register for the piloting license exam. There is, however, a need to follow general guidelines, the ones that apply to this specific use-case is are listed underneath:

- The age limit to fly alone is 16 years of age
- Max height of flight is 120 meters
- Flight should be in VLOS (Visual Line Of Sight)
- Avoid flight over outsiders and no flight over crowds of people.
- 1:1 rule, meaning, that the horizontal distance to outsiders cannot be less than the drone's vertical height above ground level, as depicted in the picture below:

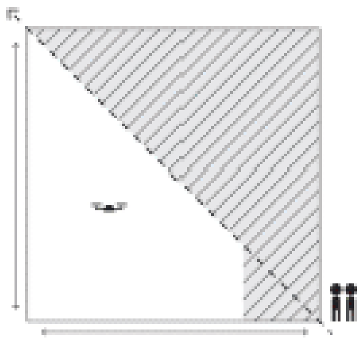


FIGURE 6: GRAPHICAL REPRESENTATION OF THE 1:1 RULE [13]

The lightweight and small stature of the DJI Tello does not pose a huge risk to the safety of humans, but it can still inflict some level of damage, therefore making sure that the drone is operated with enough distance for ample time of avoidance is necessary.

2.6 The DJI Tello: Drone of Choice

FIGURE 7: THE DJI TELLO DRONE [14]

Overview

The DJI Tello is designed and manufactured by DJI and Ryze Tech. The drone stands out for its compact and lightweight design, affordability, and ease of use. These properties make it appealing to beginners and more advanced users alike.

User Interface and programmability

The Tello's UI and programmability offers several features that enhance its usability: [15]

- **Tello App:** This app allows users to perform essential functions such as IMU calibration, monitoring battery life, and in-flight control directly from their mobile phone [14]. The user-friendly interface of the app offers an experience with many options for flight and media capture.
- **Scratch Programming:** The drone supports programming via Scratch, which is a visual block-based programming language developed by MIT, this support makes it a viable tool for introducing users to fundamentals in coding in a fun and engaging way.
- **Software Development Kit:** For more advanced users, Tello's Software Development Kit (SDK) provides further programming capabilities, enabling customization of flight patterns, data capturing, and more.
- **Community Contributions:** The Tello has a huge innovative community with engaging members reverse engineering API calls from the Tello for broader functionality and programmability, resulting in many unofficial tools and drivers being available online for further development and modifications.
- **TelloPy:** This is a Python library that provides more functionality than the official SDK. Designed for controlling the Tello drone, with key features including a way of sending control commands to the drone, resulting in movement in different directions. TelloPy also provides a way to start and receive video feeds from the camera, allowing the user

to access live video feeds as well as the ability to process video frames for further enhancements. The library can also retrieve flight data from the drone, such as battery level, speed, and altitude, which can be used to monitor the status of the drone.

Specifications [16]

- Physical specifications: Dimensions are 98*92.5*41 mm and weighing in at 80 grams, featuring 3-inch propellers. The drone comes with a built-in range finder, barometer, LED, vision system, and 2.4 GHz WiFi for 720p live view capabilities. It comes with a micro USB port for charging.
- Flight Performance: The maximum distance is 100 meters, with a top speed of 8 m/s and a maximum flight duration of 13 minutes.
- Battery: The drone is equipped with a detachable 1.1 Ah/3.8V battery
- Camera: 5 MP, 720p camera with an 82.6° Field of View (FOV), featuring Electronic Image Stabilization. Stored photos will be in. JPG format and videos will be in .MP4.

Design challenges for outdoor flight
While the Tello excels at indoor flight, the lightweight design makes it especially sensitive to windy conditions for outdoor flights. Experimentation has shown that optimal conditions are achieved when the wind speeds are below 5 meters per second. In worse conditions the drone might be subject to windthrows that can create disturbances to the video feed, leading to errors in feature extraction or sub-optimal readings, resulting in localization errors and map resets.

In summary, the DJI Tello drone was selected for its cost-effectiveness, accessibility, and programmable flexibility, which aligns well with the project’s goal of developing an economically viable option for drone deployment. As per current market evaluations using “Prisjakt”, a Norwegian price-comparison service, the Tello drone has a price range from 1 395,- NOK to 1 978,- NOK, [17] which remains within the financial scope of the project, supporting scalability and multi-drone operations.
Despite its hardware limitations when compared to other higher-end drones, especially in the area of sensors, the specifications for the Tello are sufficient for PoC (Proof of Concept) and other low-cost work.

The online community is vast, with plenty of resources for troubleshooting and development. This community-driven support provided by different forums like GitHub can be, and has been of tremendous help during this project, offering solutions and advice alike to overcome technical challenges.

3. SLAM Fundamentals and ORB-SLAM Review

This section aims to provide an introduction to what SLAM is and its use cases, together with giving an overview of a specific method of approach to visual SLAM, ORB-SLAM, and ORB-SLAM3.

3.1 Introduction to SLAM

Simultaneous Localization and Mapping (SLAM) is a technique for generating maps of unknown areas while locating and tracking the vehicle's position and movement. This technique was originally developed to achieve autonomous control in robotics [18]. Since then, SLAM has expanded into numerous fields of study, showing utility and promise on a broad level. Today, SLAM has areas of use in computer vision for 3D modelling, Augmented Reality (AR), development of self-driving cars, and especially in robotics [19].

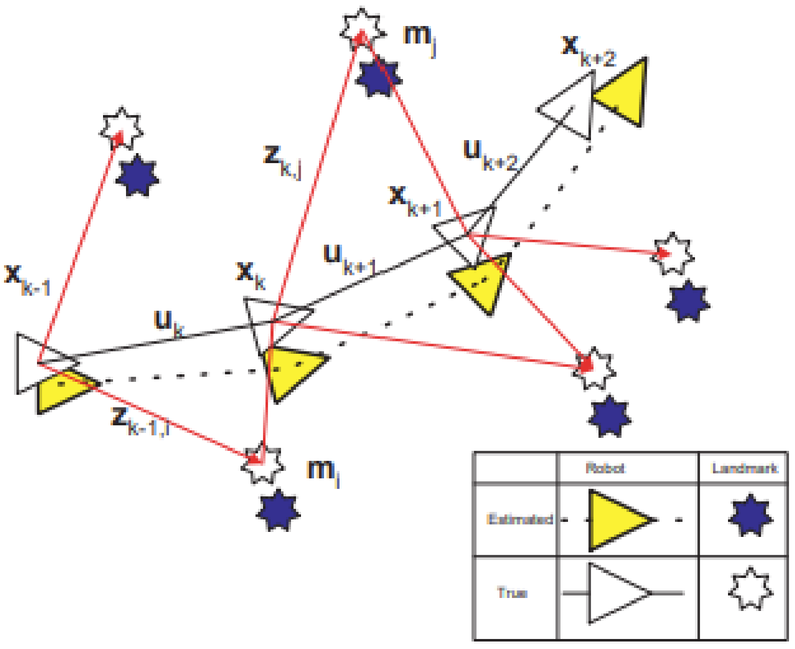


FIGURE 8: NAVIGATING THE UNKNOWN, A VISUAL REPRESENTATION OF THE SLAM PROBLEM [20]

Figure 8 gives a brief representation of the SLAM problem, having a robot navigating through an unknown environment gathering relative observations and measurements of several unidentified landmarks [20] [21] at any given time, *k,* labels can be defined as:

x_k; denoting the state vector as the location and orientation of the robot, u_k; being the control vector that is applied to the robot. This is an action that changes the robot's position and orientation x_k, at the specific time $k - 1$ to reach its new state. m_i being a vector describing the location of the i^{th} landmark. z_{ik} is an observation at any given time *k,* of a specific i^{th} landmark. [20]

Viewing the SLAM problem in probabilistic form, its distribution, $P(x_k, m|Z_{0:k}, U_{0:k}, x_0)$ requires computation for all times *k*. Here, $X_{0:k}$ signifies the vehicle location history, m, signifying the set of all landmarks, $Z_{0:k}$ is the set of landmark observations, $U_{0:k}$ signifies the control input history while x_0 signifies the initial pose. [20] [21]

There are many systems and variations of SLAM that can be used to solve this problem, with different algorithms and sensory capabilities showing strengths and weaknesses alike, depending on its use case.

One specific solution is a visual-based SLAM (vSlam) approach where cameras are used instead of other sensors including LiDARS, rotary encoders, inertial sensors, or GPS. This approach has been actively discussed and researched because of the affordability and its simpler configurations. [19]

For the work in this project, a 3D vSlam algorithm called ORB-SLAM is explored, using a monocular configuration because of its apparent compatibility with the DJI Tello Drone.

3.2 Exploring The ORB-SLAM Approach

ORB-SLAM3 is an accurate open-source system for visual, visual-inertial, and multi-map SLAM [22]. This comprehensive system is built on its previous configurations and the library offers versatility, efficiency, and accuracy, making it particularly suited together with UAVs. It integrates visual and inertial measurements to achieve seemingly robust pose estimation. This is achieved through optimization across the different phases of the actual SLAM process, which enables high accuracy in both the localization and the mapping.[8]

This method involves a unique multi-map functionality, using the Atlas framework, which allows for the management of multiple disconnected maps, which is crucial to maintaining consistent performance and handling long-term sessions or when the system encounters tracking losses. The tracking thread tries to re-localize the current frame in all the stored Atlas maps whenever the pose tracking is lost [22]. If it manages to re-localize in a stored map, the tracking resumes, and in need, switching the active maps. Otherwise, the map prior will be stored as non-active, and a new map will be initialized from scratch. [22] This happened frequently and will be detailed in the discussion part of this thesis.

ORB-SLAM3 has support for monocular, stereo, and RGB-D cameras, which enables the use across a wide variety of platforms, with various equipment. ORB-SLAM flexibility allows the users to perform robust SLAM without the need for high-end sensors or cameras. [22]the use of Oriented FAST and Rotated Brief (ORB) features to efficiently and accurately match points across images, enabling it to create or update its map.

the system has been tested quite extensively, proving its capability in various scenarios and with many different configurations. The Open-source nature allows for both a collaborative environment for improvements and continued testing. For academic and research purposes such as this project, the ability to access, modify, and tailor source code to meet specific objectives is invaluable and plays a huge role in the choice of algorithm.

The limitations of the system are also acknowledged in the research, which seems to be a limitation across the board when it comes to feature-based SLAM systems. Since the system detects and tracks distinct visual points or features within the environment to reconstruct the environment as a map, low-texture environments provide fewer unique features to extract and therefore have a higher uncertainty when it comes to the accuracy of the generated map. This applies especially in longer mapping sequences with prolonged flight and mapping. [22]

4. Methodology

Going from theory to practice, this chapter not only offers insight into relevant software tools, and prerequisites crucial for system setup, but it can also be viewed as a step-by-step guide enabling others to replicate the system, and further build and expand on the foundations provided by this work.

To enable clear and practical understanding, throughout this section, there will be text boxes designated for terminal commands or code blocks, as shown below:

```
Example text
```

These boxes are designed to differentiate between the practical executable commands from the theoretical and narrative work.

4.1 Ethical Considerations

Whenever data that can involve individual privacy is in question, there is a need to handle the data appropriately. In Norway, there are standards and regulations set in place when it comes to piloting drones. The regulations detailed in Chapter **2.5 Regulations by the Norwegian Civil Aviation Authority** are strictly followed. As the drone and its use case are well within the parameters for subclass A1, the guidelines given are mainly designed to ensure public safety and to protect their privacy.

Our flight data will mostly be comprised of areas where there are few-to-no bystanders, such as in parks, forests, or indoors for experimentation to ensure little to no disruption or infringements upon individuals or bystanders. In the case of bystanders, they will be told that the drone is equipped with a camera and that it is recording. This way the bystanders can remove themselves from the area if they do not want to be recorded. During this project, there are video feeds containing identifiable markers of individuals, these will be treated strictly confidential, and will not be included in the finished package that will be uploaded online.

Any sensitive information that is subject to being included in this thesis will be reviewed and carefully considered before use. By doing this, it is ensured that the data is not mishandled. The captured data obtained during this project will be deleted after completion.

4.2 Drone Setup and Basic Configuration

This drone is as close to plug-and-play as it could get. Upon unboxing, it comes fully assembled, simply insert and charge the battery to get started. From completely depleted it takes about 1.5 hours to reach a full charge, easily tracked by The LED indicator next to the frontal camera: blinking blue means charging, while a steady blue means that the battery is fully charged. During charging, the user has plenty of time to download the Tello app and get familiar with its UI. Once charged, connect to the drone's Wi-Fi, download the latest firmware, and calibrate the drone.

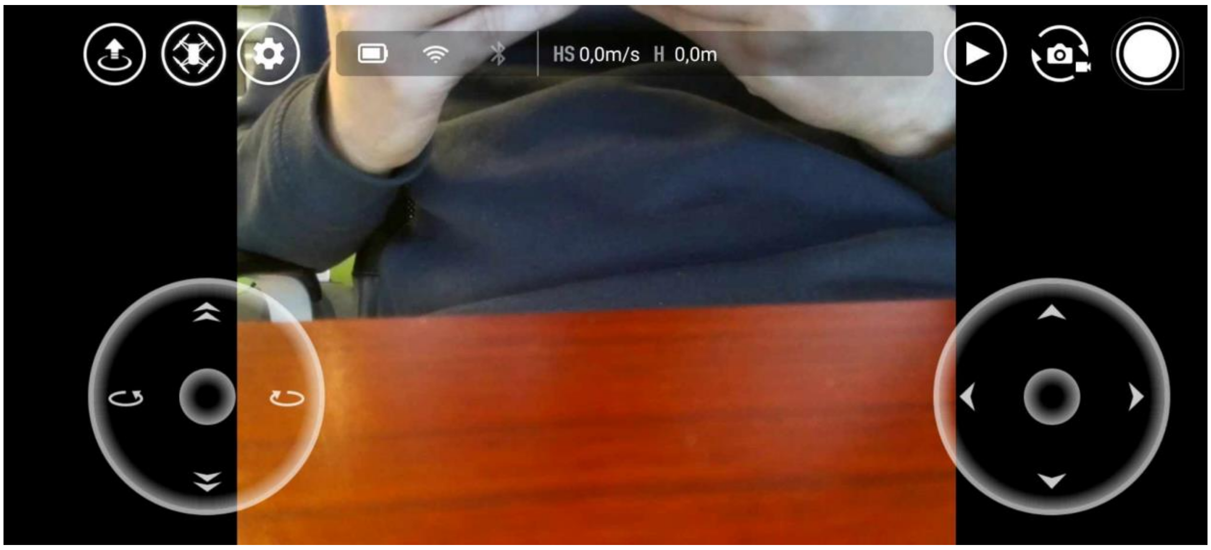
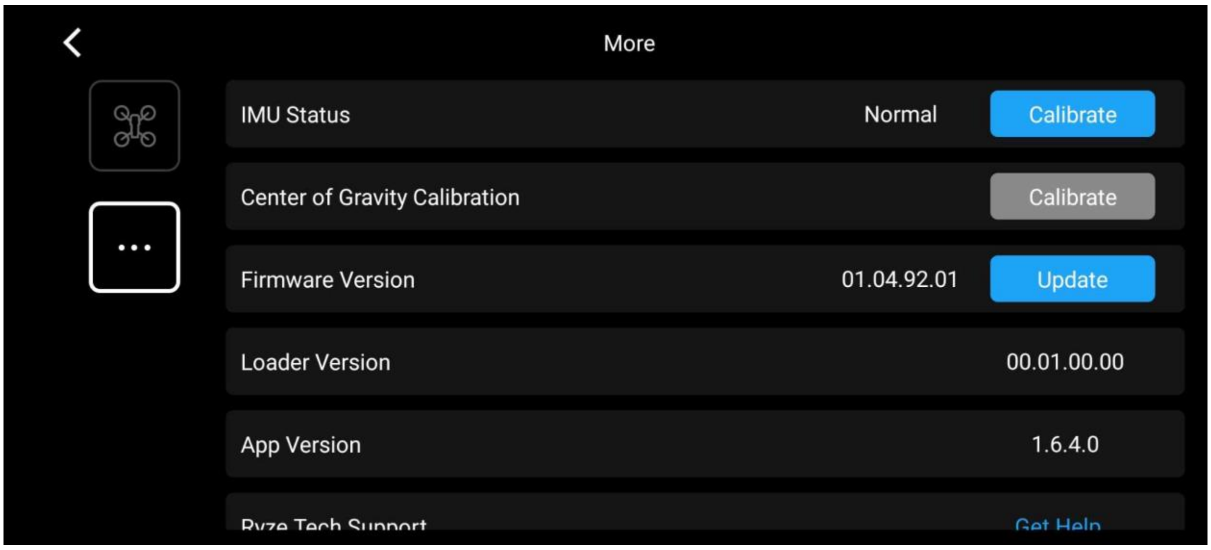


FIGURE 9: SCREENSHOT DIRECTLY FROM THE TELLO APP SHOWING THE USER-FRIENDLY INTERFACE.

As seen in **Figure 9,** The UI offers many sources of information and tools to the user. Accessing the “More” section, by pressing the cog-wheel icon, you will find a menu comprising the necessary elements for updating and calibrating the drone as seen in **Figure 10**.


More

IMU Status	Normal	Calibrate
Center of Gravity Calibration		Calibrate
Firmware Version	01.04.92.01	Update
Loader Version		00.01.00.00
App Version		1.6.4.0
Ryze Tech Support		Get Help

FIGURE 10: SCREENSHOT OF THE SETTINGS PAGE IN THE TELLO APP.

To calibrate the drone's IMU, you press the “Calibrate” button, and follow the necessary steps given by the app and the calibration tool. The steps involve placing the aircraft on a flat surface and placing it in six different positions as indicated on the screen.

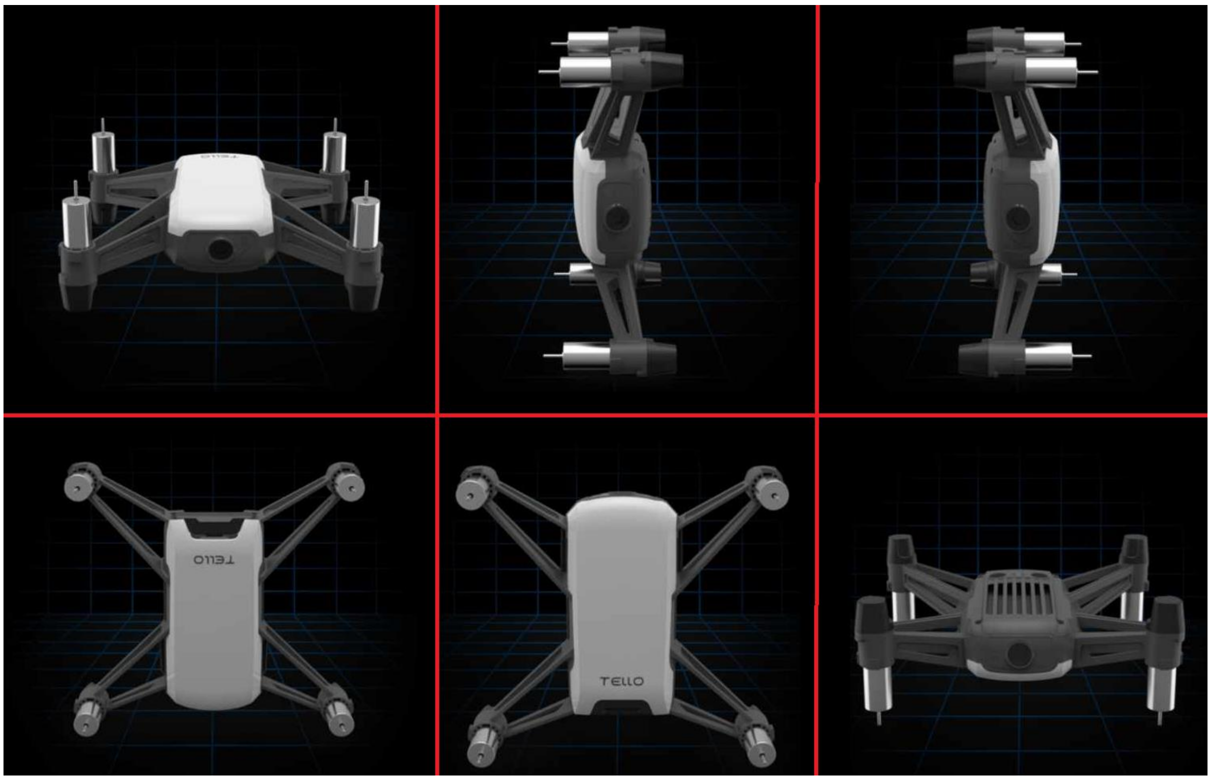


FIGURE 11: ILLUSTRATION OF THE POSITIONS USED FOR THE CAMERA CALIBRATION PROCESS.

After these steps, the user will be notified whether the calibration is successful and complete, or if the calibration steps need to be repeated, depending on the readings from the calibration.

These steps are essential to make sure that the drone is flight-ready and by concluding the IMU calibration, the user can test fly the drone, using the controls as seen in **Figure 9.**

4.3 System Setup, Software Tools, and Prerequisites

This section will detail the initial setup and configuration extending from the operating system to the necessary prerequisites for the functionality of the Robot Operating System (ROS) packages. Configurations that go beyond the initial software and tools and are directly related to the system implementation will be detailed in Chapter **5. Implementation and Experimentation**.

Installation of ROS Noetic

A foundational step in system setup is the installation of ROS. Considering the use of Ubuntu 20.04, the choice of ROS version lands on ROS Noetic, as this version of ROS is targeted for this distribution. To authenticate and allow the system to install packages from the official repository of ROS, it is essential to append ROS's source list to our system's list, as well as add the corresponding authentication keys. The installation process can be found on ROSWiki, and following the guidelines in this section will give a clear understanding of the process. [23]
If the user is familiar with the necessary steps from before, they can expedite the installation process by entering the following commands in the terminal:

```
sudo s h -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release
-sc) main" > /etc/apt/sources.list.d/ros-latest.list'

sudo apt install curl # if you haven't already installed curl

curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc
| sudo apt-key add -
```

These commands enable the system to recognize and accept software from packages.ros.org and configure the keys for package verification.

The most extensive way to install Noetic is by doing a full installation, that way you ensure that all the necessary ROS packages are included. Install ROS Noetic by using the command:

```
sudo apt install ros-noetic-desktop-full
```

Python3 is another prerequisite, although coming pre-installed with Ubuntu 20.04, by following these commands you can ensure that it is successfully installed:

```
Python3 --version
```

This command looks up the version of Python3 installed on the system, if by chance it is not installed properly, it can be installed by the following command:

```
Sudo apt install python3
```

Camera Calibration

For Camera calibration, the camera_calibration package available through 'rosdep' was used. This package allows for fairly easy calibration using the camera and a large checkerboard with known dimensions. The dimensions of this checkerboard are a grid of 8x6 with 108 mm squares. The command for installation is detailed in the text box underneath:

```
Rosdep install camera_calibration
```

This method facilitates precise calibration by capturing the interior vertex points where the checker lines of the checkerboard meet. This type of calibration is critical for extracting features from the images during the SLAM process. This method provides precise and easily detectable features that can be used when calculating the intrinsic and extrinsic parameters. Intrinsic values include focal length, optical center, and distortion coefficients, while the extrinsic values describe the camera's spatial orientation relative to objects. [24] Moving the checkerboard pattern around at a variety of angles and distances, including vertical- and horizontal movement allows for an estimation of the values.

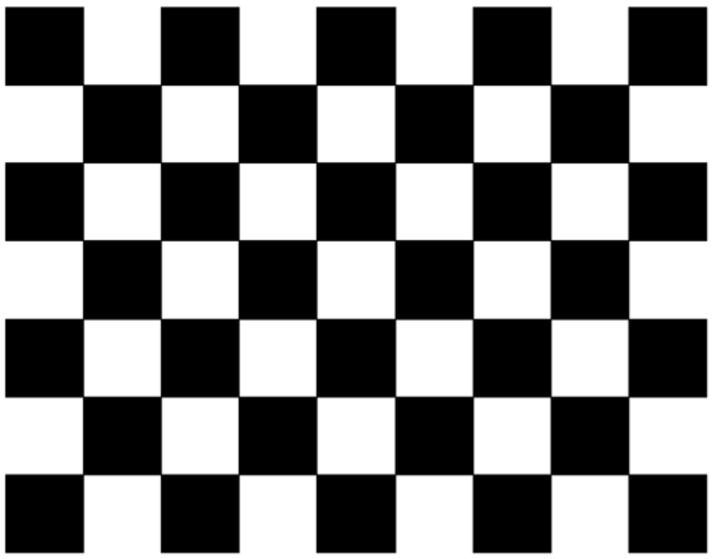


FIGURE 12: CHECKERBOARD USED FOR CALIBRATION

To run the calibration tool, there is a need to load the topics for the images. Running the following command in the terminal:

```
rosrun camera_calibration cameracalibrator.py --size 8x6 --square 0.108
image:=/tello/image_raw camera:=/camera
```

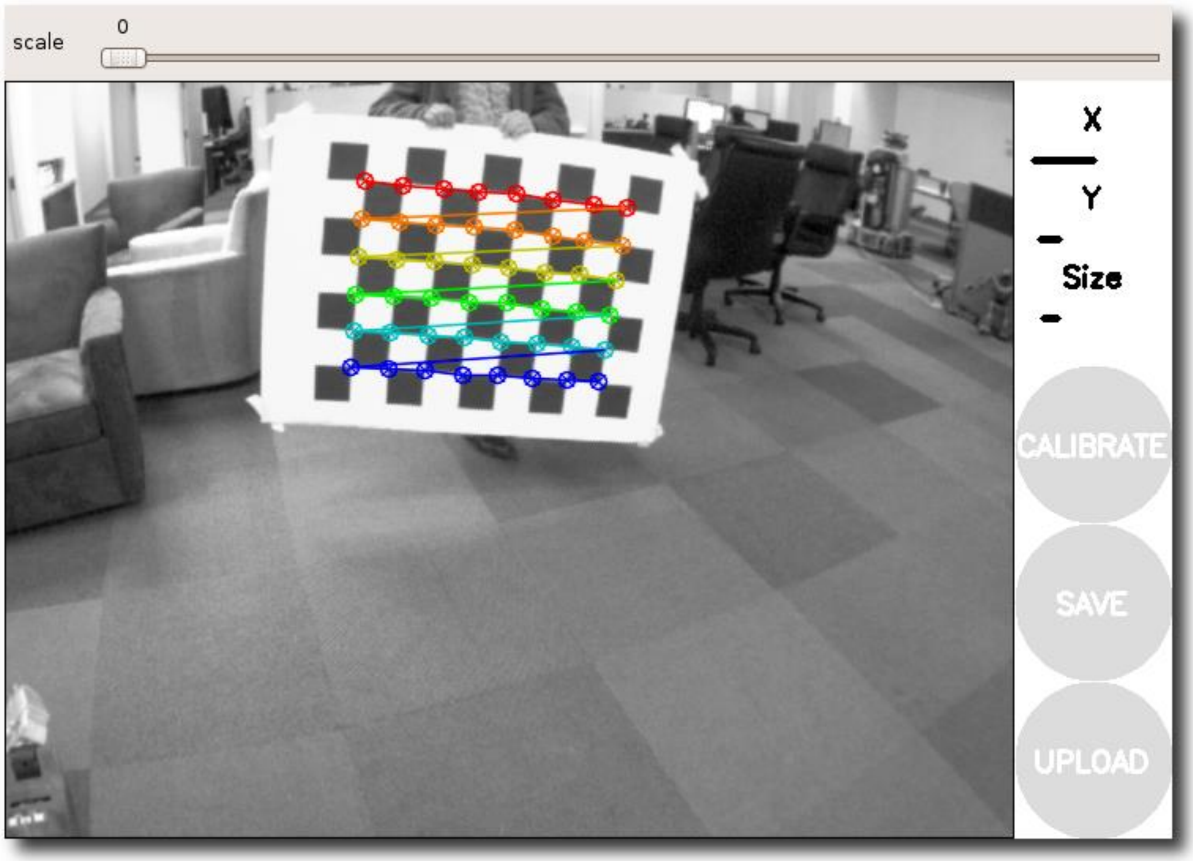


FIGURE 13: THE IMAGE USED IS FROM THE CAMERA CALIBRATION WIKI. [25]

As briefly mentioned above, to get a good calibration the checkerboard needs to be moved around. As seen in **Figure 13**, on the Hotbar to the right, there is “X”, “Y” and “Size”. The X corresponds to movement in the X-axis, or horizontal movement. The Y corresponds to movement in the Y-axis, or vertical movement. Size corresponds to the movement of the pattern either further away from the camera or closer towards the lens. The lines following these parameters signify how much data the calibration tool has managed to extract from the movement. In addition to the obvious movement gathered from the X, Y, and Size parameters, there is also a need for tilting the checkerboard while moving it around. An example of checkerboard movement can be seen in **Figure 14**.

When the calibration tool has gathered enough data the calibration button on the right hotbar as seen in **Figure 13** will start to blink, pressing this button will yield the calibration results and the parameters that you need will be displayed in the terminal. Should be stored and used in a calibration.yaml file for use with the ORB-SLAM3 package, ensuring the camera is correctly calibrated for precise spatial mapping.

FIGURE 14: ONLY FOR ILLUSTRATION PURPOSES, NOT IMAGES FROM ACTUAL CALIBRATION [25]

The D values in the output are the distortion coefficients and correspond to the radial distortion and tangential distortion labeled as k1, k2, and k3, p1, and p2.

The K values are associated with the camera matrix and correspond to the focal lengths (fx, fy) and principal point coordinates (cx, cy).

In addition to these values, you also need the width, height as well as FPS. The width and height are given directly while the fps is based on the camera specifications.

The values provided by this camera calibration can be stored in a text file to keep for later use, as they will be added and used when the ORB-SLAM3 package is introduced. The values obtained and stored can be located and viewed in **Appendix A**.

Pangolin
Pangolin is a utility library for 3D rendering and numerical analysis, making it a powerful and necessary tool for data visualization [26] in this project. To install Pangolin use the commands below:

```
mkdir ~/your_fav_code_directory

cd your_fav_code_directory

# Clone the repository

git clone --recursive https://github.com/stevenlovegrove/Pangolin.git

cd Pangolin

# Install dependencies

./scripts/install_prerequisites.sh recommended

cmake -B build

cmake --build build
```

To ensure that all submodules are included, the --recursive option is used. This is because the git clone does not initialize and update the submodules by default.

Eigen3

Eigen3 is the third iteration of the Eigen library, which is a template for linear algebra; matrices, vectors, and numerical solvers including a wide range of functionality from the most basic of arithmetic to the more complex algebraic operations like singular value decomposition and more. [27]

Eigen3 is already included in many ROS distributions, including ROS Noetic, and can be installed by using the command:

```
sudo apt-get install libeigen3-dev
```

This way, Eigen3 is available system-wide, and catkin_make or build will be able to locate it without requiring manual compilation and/or installation.

For projects that require a specific version or non-ROS applications, manual compilation using CMake might be necessary. As this system will be focusing on ROS, this approach will not be detailed, but the necessary steps for this approach can be found easily online.

4.4 Project Structure and Modifications

Tello driver

The system specification necessitated the evaluation of various libraries to identify the most suitable one designed for controlling the Tello drone. Initial tests included keyboard configurations and others which was compatible with other Ubuntu distributions, many of which were compatible or tested using Ubuntu 16.04. These were not optimal due to compatibility issues and they needed various degrees of modifications and tweaking to get it to work with the current system setup for this project. The driver of choice landed on tello_driver, a GitHub repository created by Jordy Van Appeven which is forked from anqixu/tello_driver, for its compatibility and its use of joystick controller configurations. This driver already includes the configuration of Playstation3 (PS3) controllers, which makes it highly modifiable to other compatible controllers as well.

The install process for this driver is rather straightforward, and done quickly by following these commands:

```
mkdir -p ~/your_tello_slam_ws/src && cd your_tello_slam_ws/src

git clone --recursive https://github.com/appie-17/tello_driver.git
```

Some modifications had to be done to the gamepad teleop node, to facilitate manual control of the drone. The code modifications add support for interpreting input from the Logitech F310 controller within the ROS system and allow for direct mapping of input or signals from the gamepad. The actual modifications and additions to the node can be found in

Appendix B.

FIGURE 15: THE F310 GAMING PAD [28]

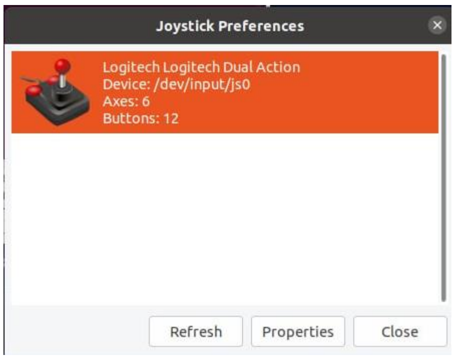
To find out which buttons or joystick movements from the controller corresponded with the buttons commands, 'jstest-gtk' was utilized, a graphical joystick testing application for Linux distributions. This tool allows for real-time identification and mapping of the buttons and axes of the controller.

Install and run the joystick tester by plugging in the controller and running the following commands:

```
sudo apt-get install jstest-gtk

jstest-gtk
```

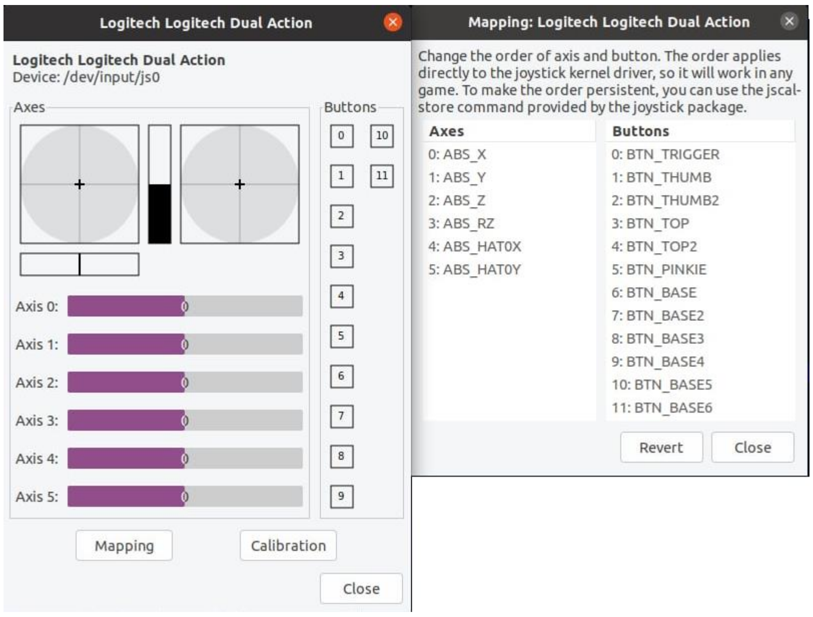
The first window that opens after running the program, is detailed in **Figure 16.** Here is a brief description of the connected controllers with several axes and buttons.


Joystick Preferences

Logitech Logitech Dual Action
Device: /dev/input/js0
Axes: 6
Buttons: 12

FIGURE 16: THE HOME WINDOW OF JSTEST

After selecting the controller of choice you will be met with the graphical interface shown in **Figure 17.** From this interface, the user can easily map and label buttons. Through this process, specific drone commands were assigned to the gamepad's input, ensuring intuitive controls for the drone.


Logitech Logitech Dual Action

Logitech Logitech Dual Action
Device: /dev/input/js0

Mapping: Logitech Logitech Dual Action

Change the order of axis and button. The order applies directly to the joystick kernel driver, so it will work in any game. To make the order persistent, you can use the jscal-store command provided by the joystick package.

Axes	Buttons
0: ABS_X	0: BTN_TRIGGER
1: ABS_Y	1: BTN_THUMB
2: ABS_Z	2: BTN_THUMB2
3: ABS_RZ	3: BTN_TOP
4: ABS_HAT0X	4: BTN_TOP2
5: ABS_HAT0Y	5: BTN_PINKIE
	6: BTN_BASE
	7: BTN_BASE2
	8: BTN_BASE3
	9: BTN_BASE4
	10: BTN_BASE5
	11: BTN_BASE6

FIGURE 17: GRAPHICAL INTERFACE OF JSTEST FOR MAPPING THE BUTTONS

The keybinding configuration can be seen in **Table 2: Keybind Configuration** as shown underneath:

Button	Action
L1	Take-off
R1	Landing
Left joystick UP	Vertical ascension
Left joystick DOWN	Vertical descension
Left joystick LEFT	Rotation towards the left (Counter clock-wise rotation)
Left joystick RIGHT	Rotation towards the right (clockwise rotation)
Right joystick UP	Fly forward
Right joystick DOWN	Fly backwards
Right joystick LEFT	Fly to the left without rotating the drone
Right joystick RIGHT	Fly to the right without rotating the drone

TABLE 2: KEYBIND CONFIGURATION

In addition to these actions, there are other action commands available like flips, etc, but these were not paid attention to, as they are not relevant to this project.

Camera Info Manager

To manage camera calibration information effectively, we use *'camera_info_manager.py'*, a Python version of the configuration written in C++ 'camera_info_manager', It is installed with the following command:

```
git clone https://github.com/ros-perception/camera_info_manager_py.git
```

Image viewer

For accessing the camera feed in real-time the user can use rqt_image_view, which is a tool that enables the users to view image data published over ROS image topics. This is easily installed by using the following command:

```
sudo apt-get install ros-noetic-rqt-image-view
```

If the ORB-SLAM is in effect while piloting the drone, this step is not needed, as you will be provided with a window showing the camera feed together with the feature extraction in full effect.

Testing the system

After successful installation and configuration, the operator is now set for a test flight ensuring that the system is fully operational and ready to use. To test the system the operators turn on the drone and connect to its WiFi, and run these commands in different terminals to initiate the Tello drone and associated control systems:

```
In Terminal 1: (Initializing the ROS environment)
roscore


In Terminal 2: (Initiating the Tello driver)
cd ~/your_tello_slam_ws
source devel/setup.bash
roslaunch tello_driver tello_node.launch


In Terminal 3: (Enabling the configuration for joystick controls)
cd ~/your_tello_slam_ws
source devel/setup.bash
roslaunch tello_driver joy_teleop.launch


In Terminal 4: #Optional (Camera broadcasting tool)
rqt_image_view


In Terminal 5: #Optional (To record the video feed)
rosbag record -a
```

Terminal 4 and Terminal 5 are as written, optional, and are included in case the user wants to see if the camera broadcasts successfully for a full system check. Terminal 4 opens up the image viewer provided by the rqt software, while terminal 5 records the video that is being broadcast. Normally, the users have to specify which ROS topics are to be recorded by using the command *rosbag record*. However, using *-a*, every topic that is being published gets recorded, which is a powerful option, however, if the user has limited disk space, in the case of prolonged missions, this should be used with some consideration as all topics might consume disk space.

ORB-SLAM3

The primary GitHub repository for the ORB-SLAM3 system is well established, however, the primary repository was not used in this project, instead, a segmented repository offering a ROS implementation has been utilized. This decision enables us to tailor the system to the specifications of ROS and Tello's operational framework. Installing this specific ORB-SLAM package can be done by using the following terminal commands:

```
cd ~/your_tello_slam_ws/src

git clone https://github.com/thien94/orb_slam3_ros
```

The parameters obtained previously when calibrating the camera are critical to ensure that the SLAM algorithm will work as intended, and they are detailed in **Appendix A**. These can be used alongside ORB-SLAM3-specific settings such as threshold detection and feature extraction numbers, and should be stored and used within the *tello_calibration.yaml* file. This file uses the same template as the other yaml files configured for monocular cameras, a template that has followed ORB-SLAM since ORB-SLAM2 and is included in the '*examples*' folder [29]. The contents of this file can be found in **Appendix D**.

Note: If there is intent to replicate this system for further work or testing, these parameters must be changed, and there is a need for calibration of the camera of your own Tello drone to ensure good results.

The final phase of the system's configuration is the creation of the *tello.launch* file. This file includes the parameters and settings essential for initiating the nodes necessary for this system. The parameters and settings for this launch file are specified and detailed in **Appendix C**.

With the system now fully set up, the stage is set for implementing the SLAM and experimenting with the parameters to achieve the results as needed.

5. Implementation and Experimentation

Following the initial setup and software configurations, this chapter will provide and present the practical application and the implementation of this system, experimenting with the parameters for the ORB-SLAM algorithm.

5.1 Real-time SLAM Operations

This work emphasizes the use of manual operation and control of the drone, real-time SLAM is a solution that enhances the situational awareness of the operator. By providing up-to-date mapping of the environment, operators can make decisions directly, identifying potential obstacles and being able to adjust paths taken or identifying areas requiring further exploration and adjustments. This course of action will ensure comprehensive coverage of the targeted area and might play a significant role in optimizing the mapping.

A main concern using this methodology is the drone's battery life. Despite running the SLAM algorithm on an external computer, the flight time remains a significant challenge. The restrictions that follow not only affect extended mapping missions but also limit the options for updating or adjusting operational parameters mid-mission as conducting multiple take-offs and landings for making these adjustments and relaunching the system will affect efficiency and mapping coverage capabilities.

To adopt this methodology and use the system in this manner, follow the sequence of commands outlined below:

```
In Terminal 1: (initializing the ROS environment)
roscore

In Terminal 2: (Launch ORB-SLAM3 with the Tello configuration)
cd ~/your_tello_slam_ws
source devel/setup.bash
roslaunch orb_slam3_ros tello.launch

In Terminal 3: (Initiate the Tello driver)
cd ~/your_tello_slam_ws
source devel/setup.bash
roslaunch tello_driver tello_node.launch

In Terminal 4: (Enable joystick controls)
cd ~/your_tello_slam_ws
source devel/setup.bash
roslaunch tello_driver joy_teleop.launch
```

*Note: It is crucial to ensure the connection to the drone's WIFI. For real-time flight, the 'sim_time' parameter is within the tello.launch file should be set to **false**, allowing the system to process live data timestamps accurately.*

5.2 Post-flight SLAM Analysis

Recording the drone data and applying the algorithm on the data post-flight offers distinct advantages. By applying the algorithm on recorded rosbags, the parameters used can be fine-tuned through repeated iterations. The flexibility this approach offers enables a degree of refinement and accuracy not easily obtained otherwise using real-time methods and it is especially beneficial where details in the environment are especially important.

Much like with the real-time SLAM methodology, this approach also comes with its own set of drawbacks and considerations. By using this approach, there is no way of reacting to real-time environmental feedback, meaning, the operator must rely solely on the recorded data, and in a dynamic environment, this data is subject to change. Additionally, relying on pre-recorded data, extensive data storage might become an issue, especially for lengthier missions or high-detailed mapping, which could be mitigated by the use of external hard drives or similar.

To conduct a post-flight SLAM analysis with the collected data, execute the following commands:

```
In Terminal 1: (enabling the ROS environment)
roscore


In Terminal 2: (launching the SLAM system)
cd ~/your_tello_slam_ws
source devel/setup.bash
roslaunch orb_slam3_ros tello.launch


In Terminal 3: (Replaying the record rosbag data with the -clock option)
cd ~/your_tello_slam_ws/recorded_rosbags
source devel/setup.bash
rosbag play --clock your_recorded_bag.bag
```

*Note: When using ORB-SLAM on rosbags for post-flight analysis change the 'use_sim_time' value to **true** within the tello.launch file. Rosbags contain data that was recorded at a specific time, this data also contains the time at which each message was recorded, by enabling 'use_sim_time', ROS is told to use the timestamps from the rosbags instead of the current system time. This allows the nodes to process the data as if it were being received in real-time, which is essential for accurately replaying and analyzing the recorded data.*

5.3 ORB-SLAM Experimentation

There was done extensive experimentation when it came to parameter adjustments for the ORB-SLAM algorithm, including changes to the number of features extracted per image and the thresholds. This was done using a trial-and-error approach by changing the values based on the feedback as seen on the screen. The drone flight path is depicted in green.

In **Figure 18,** we observe ORB-SLAM3's capacity to achieve highly detailed mapping of individual objects, these are screenshots of the same flight and the same object, only from different angles. These figures illustrate a wooden basket intended to contain firewood, and the resulting point cloud shows the precision and accuracy of the algorithm. Using a wooden basket containing a huge number of features to be extracted, the algorithm demonstrates its potential, even going beyond comprehensive environment mapping.

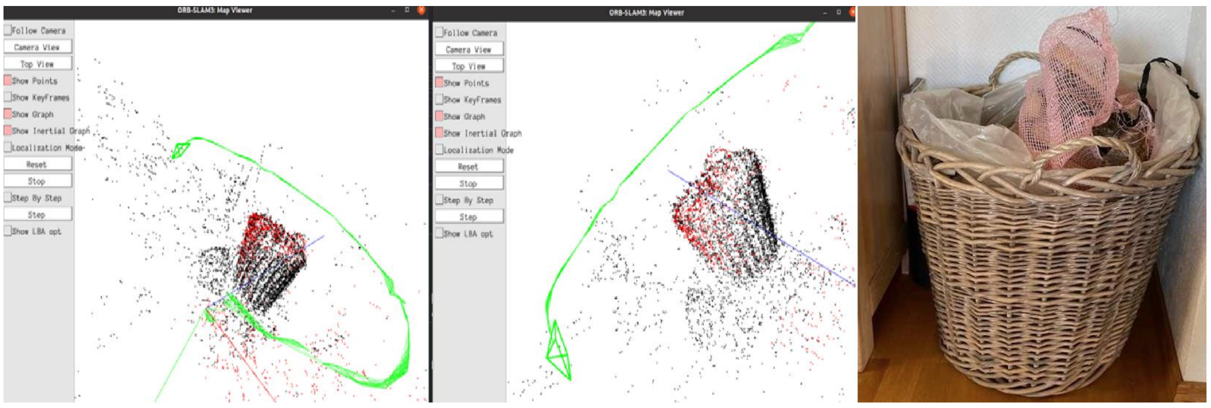
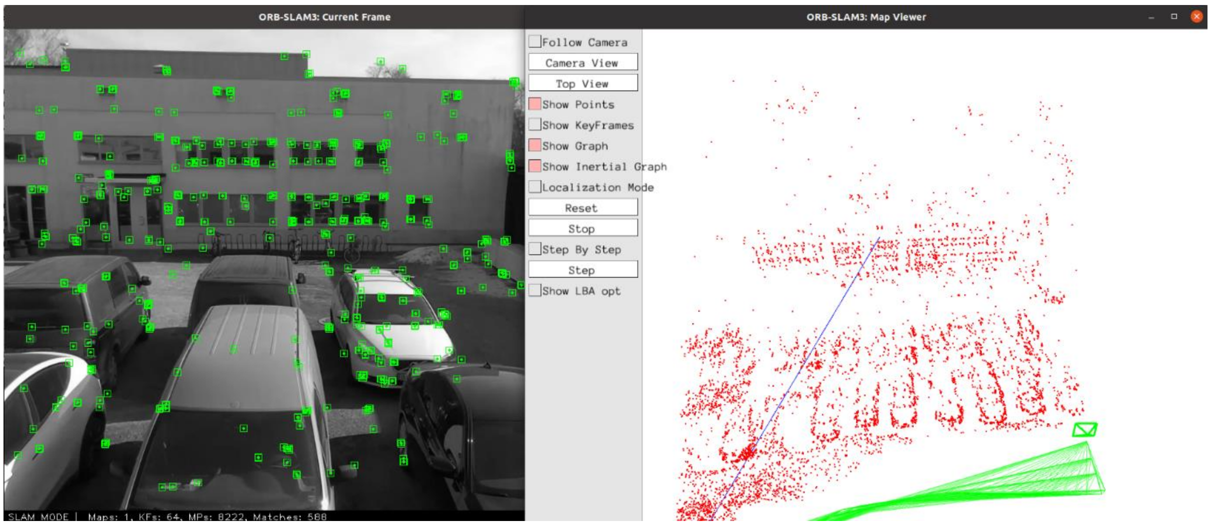


FIGURE 18: USING ORB-SLAM3 FOR PRECISION MAPPING OF AN ISOLATED OBJECT.

The following figures as seen in **Figure 21**, **Figure 20**, and **Figure 19** show different configurations of the ORB-SLAM algorithm. These are experiments conducted to explore its performance and to get a visual representation of the differences by changing the parameters. Here, the ORB-SLAM was applied to a short rosbag recording of a single parking lane near the University of Life Sciences. The main function was to determine the parameters that gave the most satisfying results. Specifically, these figures illustrate the feature extraction from a video feed and the resulting point cloud generated. The recordings were stopped at roughly the same timeframe, this was done manually, so slight deviations occurred.

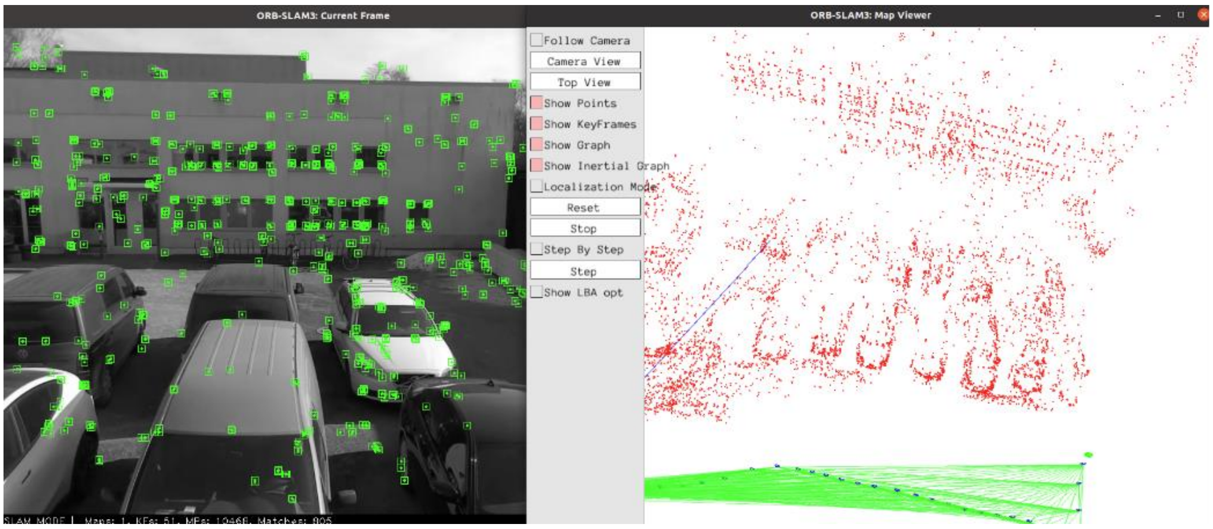
In **Figure 19,** a configuration with a higher threshold value and a maximum features extracted was set to 1250 per image yielding 8222 features, and for the frame in view, 588. Despite the higher threshold value, the accuracy of the point cloud remained relatively high.



SLAM MODE | Maps: 1, KFs: 64, MPs: 8222, Matches: 588

FIGURE 19: THE IMAGE IS FROM A PARKING LOT, FEATURE EXTRACTION CAPPED AT 1250 PER IMAGE.

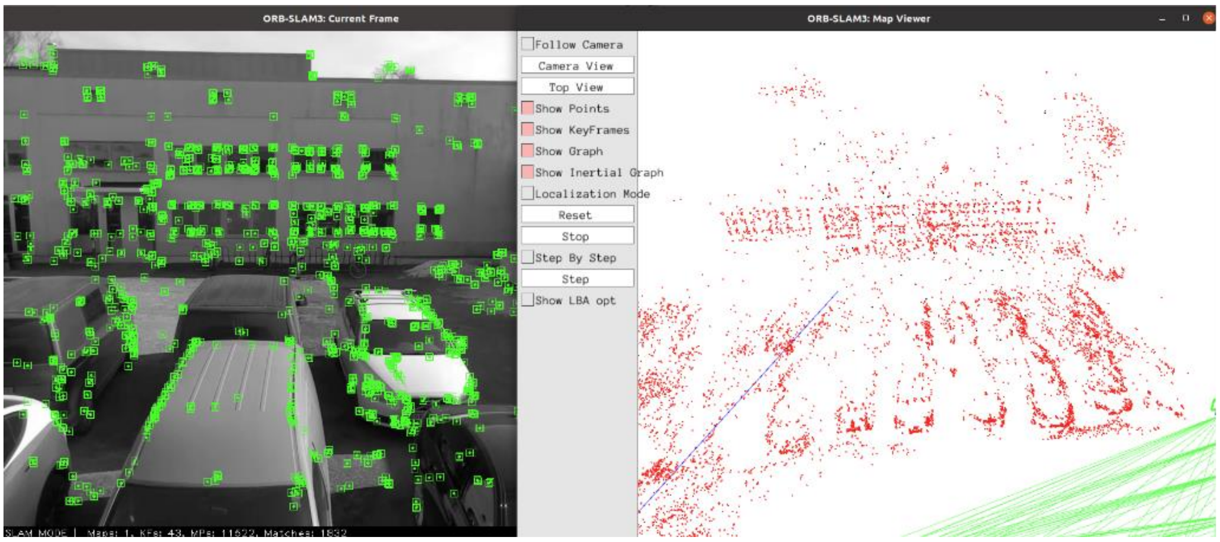
In comparison, **Figure 20** had a lower maximum extraction limit of 1000, and a reduced threshold, resulting in 10468 features extracted, with the specific frame in view extracted 905 features. This configuration showed an enhanced capacity to detect features, particularly in lower contrast regions. The resulting point cloud is quite similar and there are only small deviations between them, which is in favor of this being a threshold setting not generating too much noisy data.



SLAM MODE | Maps: 1, KFs: 51, MPs: 10468, Matches: 905

FIGURE 20: THE IMAGE IS FROM A PARKING LOT, FEATURE EXTRACTION CAPPED AT 1000 PER IMAGE.

Figure 21 represents an experiment where the feature cap was set to 5000, using the same threshold as in **Figure 20.** Interestingly, the system appeared to have an extraction limit of around 2000 features, as the features extracted seemingly never exceeded 2000, which seems odd as the ceiling was set so much higher. The system extracted 11522 features and for the snapshot in view, 1832. This configuration also produced a point cloud with minimal deviations from the others, suggesting a marginal return on feature count, suggesting a potential operational sweet spot.



```
SLAM MODE |  Maps: 1, KFs: 43, MPs: 11622, Matches: 1832
```

FIGURE 21: THE IMAGE IS FROM A PARKING LOT, FEATURE EXTRACTION CAPPED AT 5000 PER IMAGE.

The experimentation with the parameters of this algorithm has highlighted its efficiency and accuracy. This holds especially true in environments rich with distinct features. However, it is noteworthy that the system on multiple occasions did not reach the set ceiling for feature extraction, as seen in **Figure 21.** This observation and possible factors will be considered and discussed properly in Chapter **7. Discussion.**

6. Results

This chapter will further expand on the previous chapter, which involves the mapping of different environments. Key terminology presented in this chapter is map points and key frames. Map points refers to the actual points in the cloud which is generated by matching key points from multiple frames and triangulating them to compute their position in a coordinate system. Key frames is an essential part of this system as it is frames chosen by the system as they provide significant information about the environment and helps in tracking the drone's positioning in situations where the system loses track of it, which plays a major role in re-localization.

Figure 22 shows the results of a single flight through an indoor environment, specifically the living room of a family member of the researcher. This living room was chosen for its array of distinct features in the many wall paintings, photographs, and bookshelves stocked with books. This experiment was conducted to highlight the efficiency of this algorithm, and despite a single flythrough, of which the flight path shown in green, the algorithm successfully extracted a high level of detail from the surroundings.

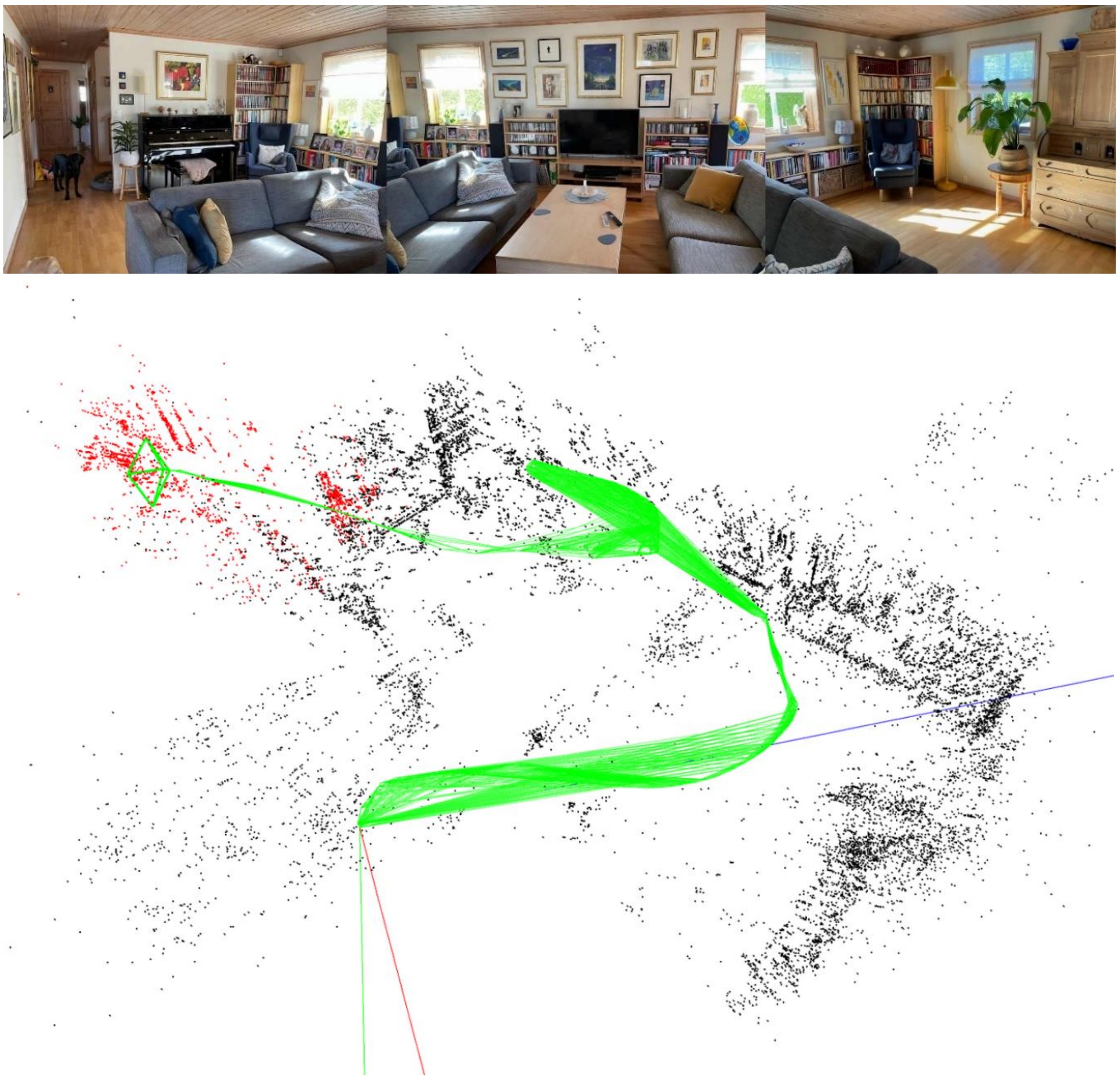


FIGURE 22: LIVING ROOM AND RESULTING POINT CLOUD

As seen in **Figure 23**, this figure shows a small parking lot with two lanes, and a single flight through as seen in green and blue, the green is the flight path of the drone, and the blue markers are keyframes used for localization. The cars do indeed have a lot of features to be extracted, and so do the buildings due to the number of windows, creating a lot of distinct markers. It is in these situations where the visual-based SLAM techniques excel, in areas where there is a lot of distinct features to be extracted. This map was created using 254 key frames and 17620 map points.

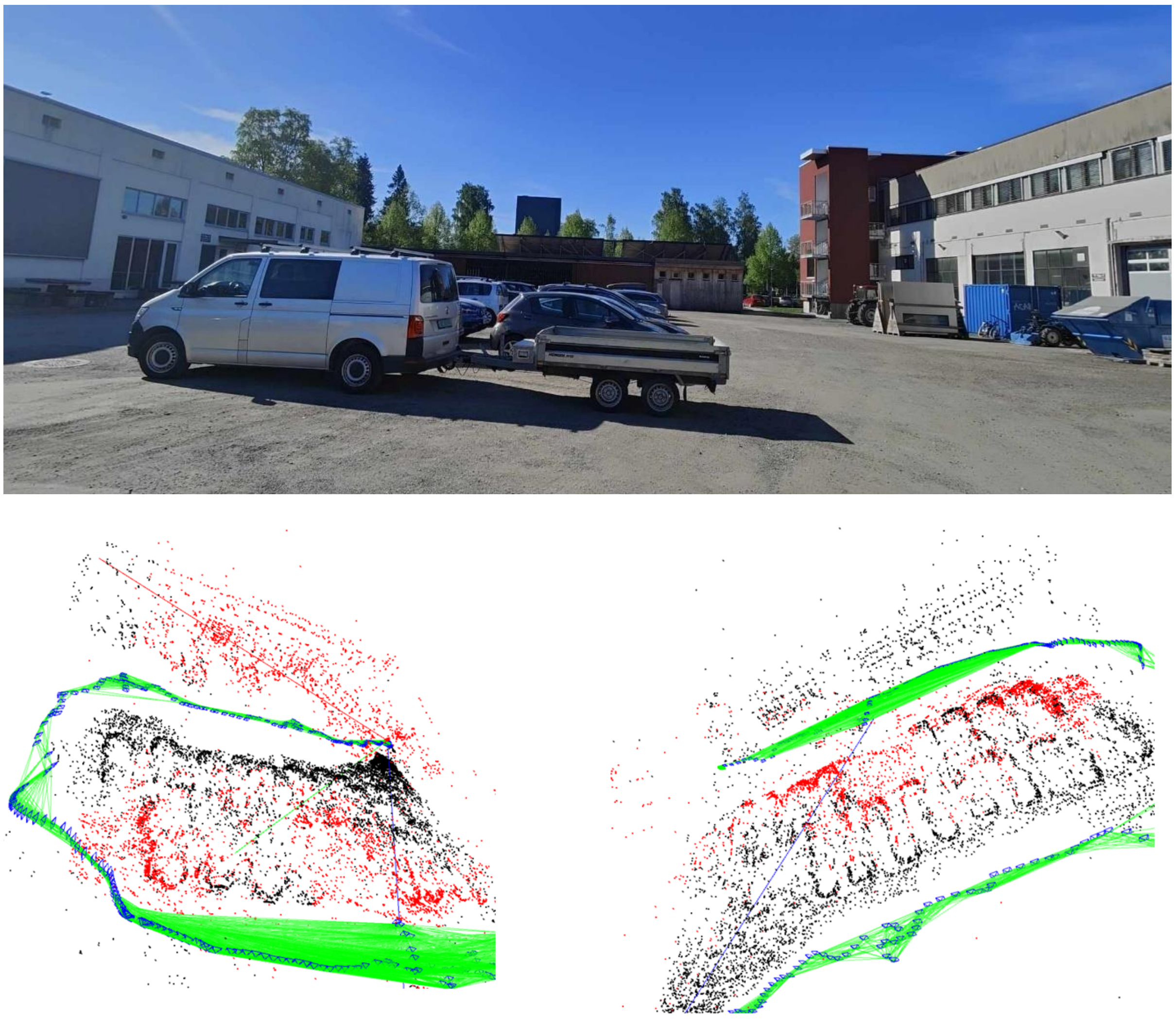


FIGURE 23: FIGURE SHOWS A PARKING AREA OUTSIDE OF NMBU WITH RESULTING POINT CLOUDS

A different scenario was an open field with a few trees, as depicted in **Figure 24**. This field has just a few trees in it, otherwise it is barren. **Figure 25** represents the resulting point cloud and this is included to show that even though trees themselves do not always present enough distinct features for the algorithm to pick up on, by lowering the threshold values there is a higher chance to recognize them. The result of tinkering with these values show that the algorithm is able to pick up enough features to map the trees, however, as this footage was acquired early spring, which means the trees did not have the amount of leaves as seen in Figure 24, smaller branches remain an issue and you need to fly rather close to being able to detect them. If this footage was acquired later in the season, a more detailed representation of the trees would be possible due to having more distinct features available due to the leaves and flowers on the trees, Creating detailed mapping of the trees itself was not the intention here, so in this particular instance, this was not an issue.

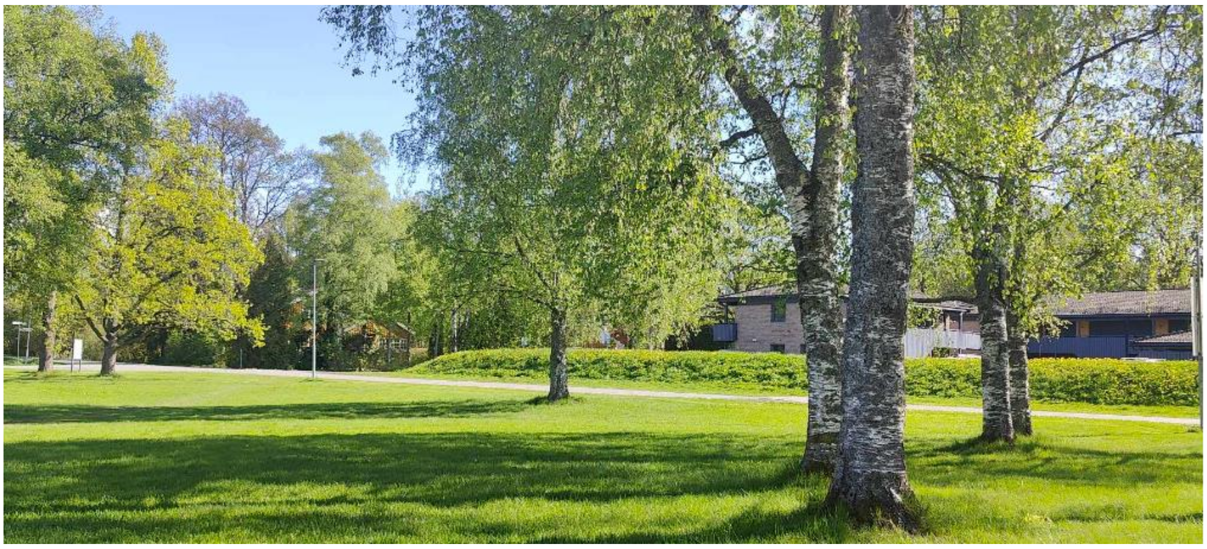


FIGURE 24: THIS FIGURE SHOWS THE SMALL GREEN AREA

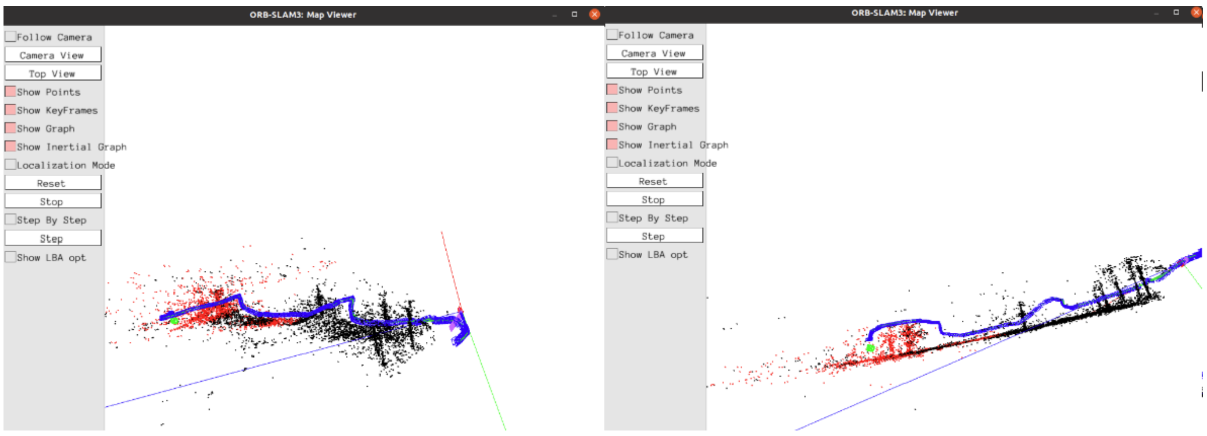


FIGURE 25: THIS FIGURE SHOWS THE RESULTING FLIGHT PATH AND POINT CLOUD

Figure 26 shows a tunnel made to research strawberries and applications surrounding picking and managing the berries. As seen on the resulting point clouds, which is generated using 9229 map points and 149 key frames, it is quite difficult to make out what the point cloud should represent without a reference, the reason for this is there are not enough distinct features to create a good enough mapping of this type of environment, at least not without using other methods to process the data afterwards. However, having a a reference picture provides enough context to understand what is going on. This figure is included in this section to show that there are certain limitations especially in environments with less features.

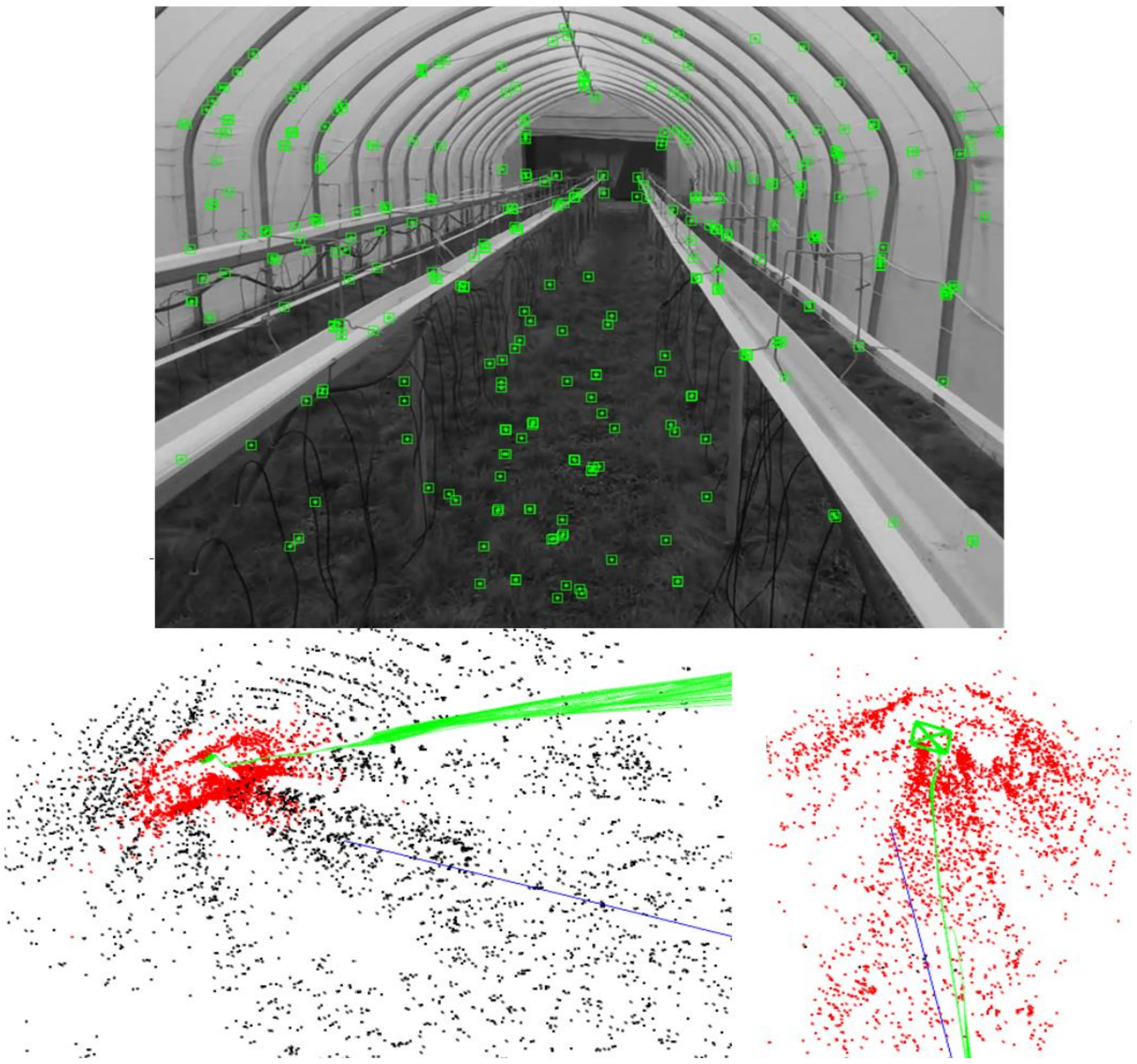


FIGURE 26: FLIGHT THROUGH TUNNEL, WITH RESULTING POINT CLOUDS

In **Figure 27** we see two flights from the surrounding area of AudMax, a concert/venue hall in Ås, the first flight was more focused on the surroundings while the other flight was more focused towards the face of the building itself. In the first flight, the lack of features for the surrounding road and pavement creates clear and distinguishable roads in the resulting point cloud. The first flight generated a point cloud with 12 159 map points and 411 key frames, while the second flight generated a cloud with 13722 points having 307 key frames.

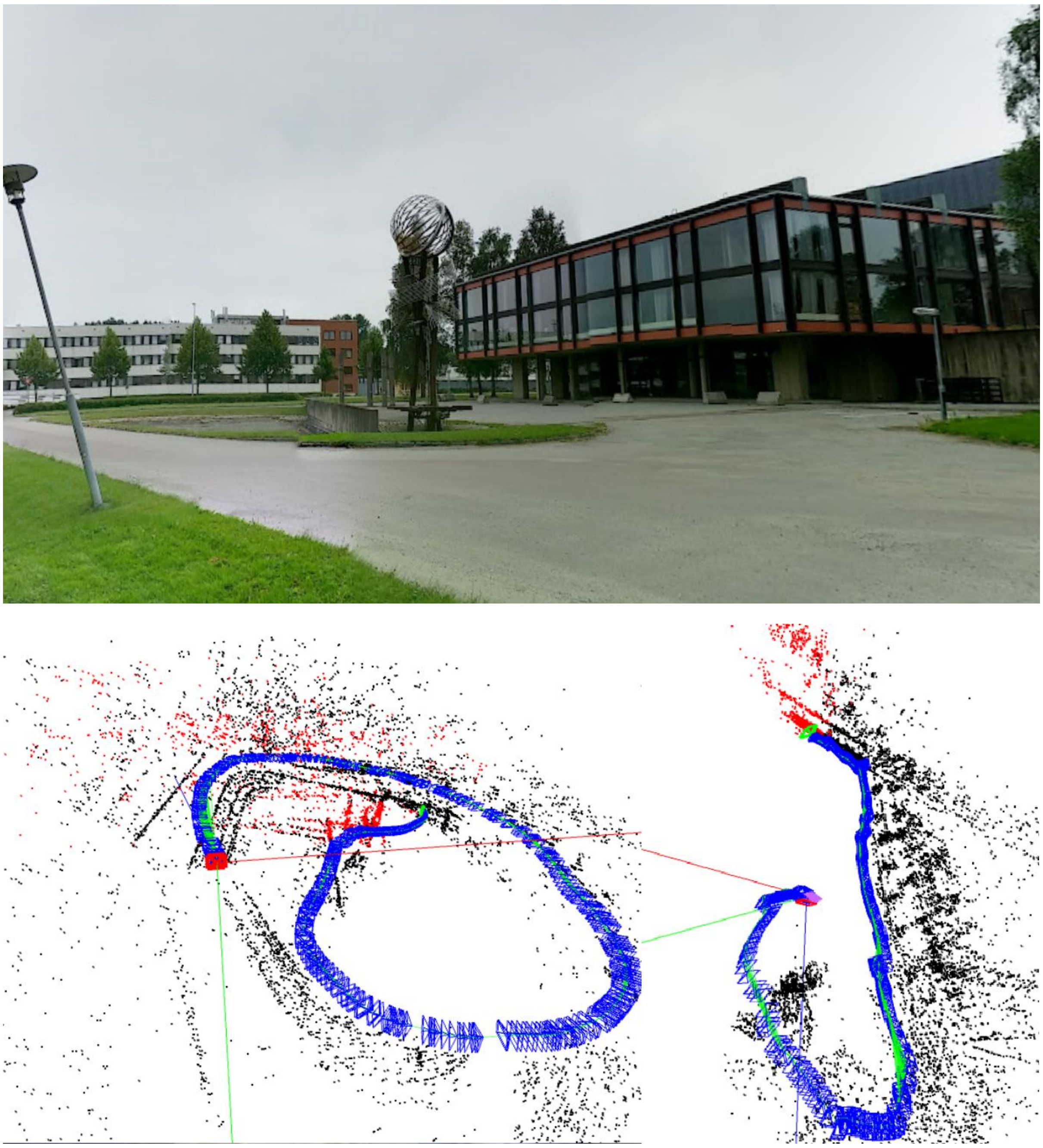


FIGURE 27: AUDMAX ÅS, AND RESULTING TWO POINT CLOUDS

Figure 28 details a specific part of the NMBU Campus, which involves several buildings including few of the "TF"-wings, Robotics lab, EikLab garage, and a student-study area in the top half. The bottom half is a satellite image of the area from google maps, traced manually to show the flight path of the drone for this particular mapping session. **Figure 29** shows the resulting point cloud of the entire area from two different angles while **Figure 30,** and **Figure 31** is two sections where we zoom in on specific parts of the area to show the systems accuracy. This flight involved a landing to change the battery of the drone, in which we could test out the Atlas system of ORB-SLAM, where the system stored the part of the map already generated, to bring it back up to join with the newly created one to further expand on the map. The resulting map is generated by joining the two maps, and with 26754 map points and 740 key frames.

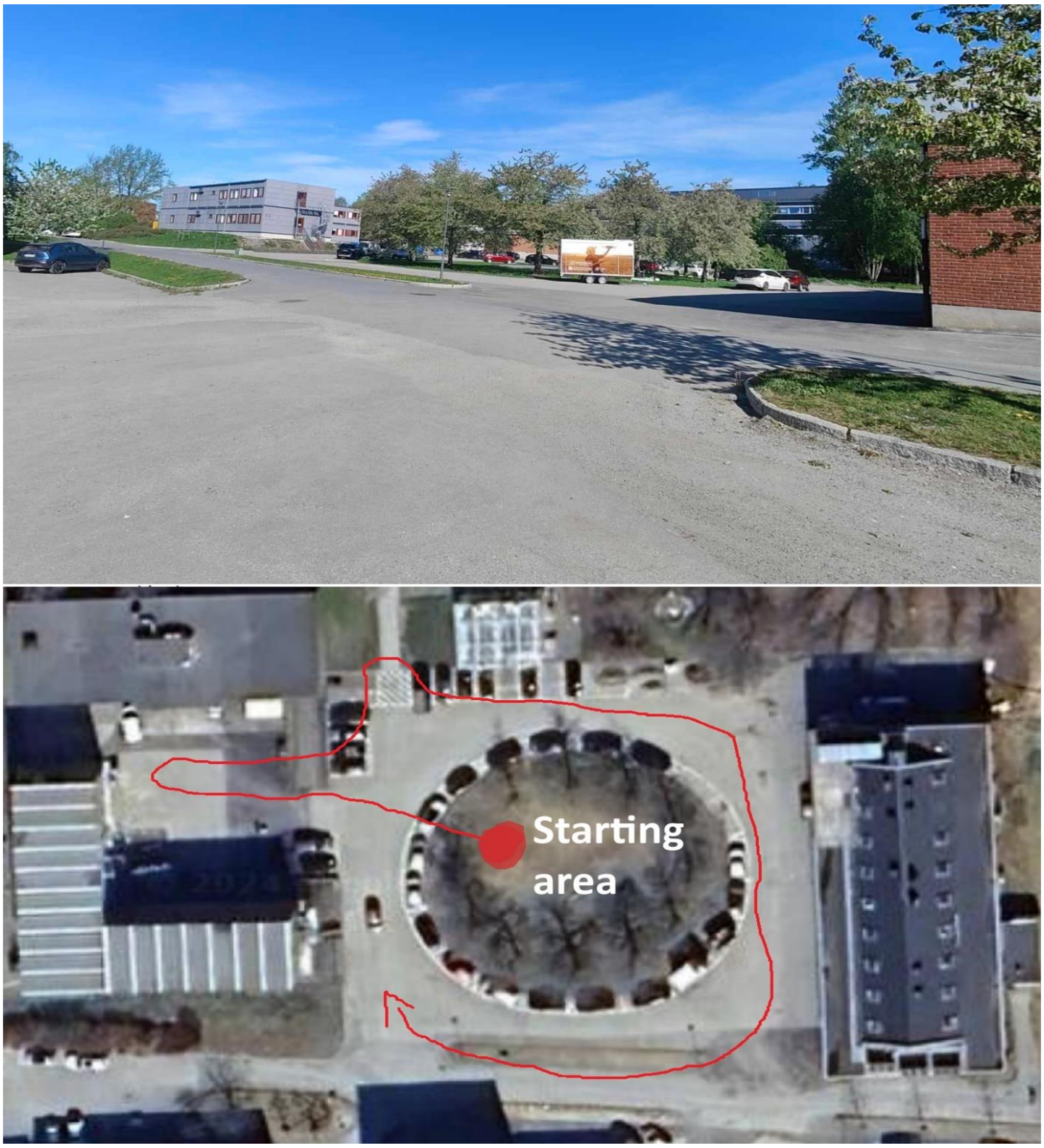


FIGURE 28: TF SECTION OF NMBU CAMPUS AND FLIGHT PATH TRACED USING IMAGE FROM GOOGLE MAPS

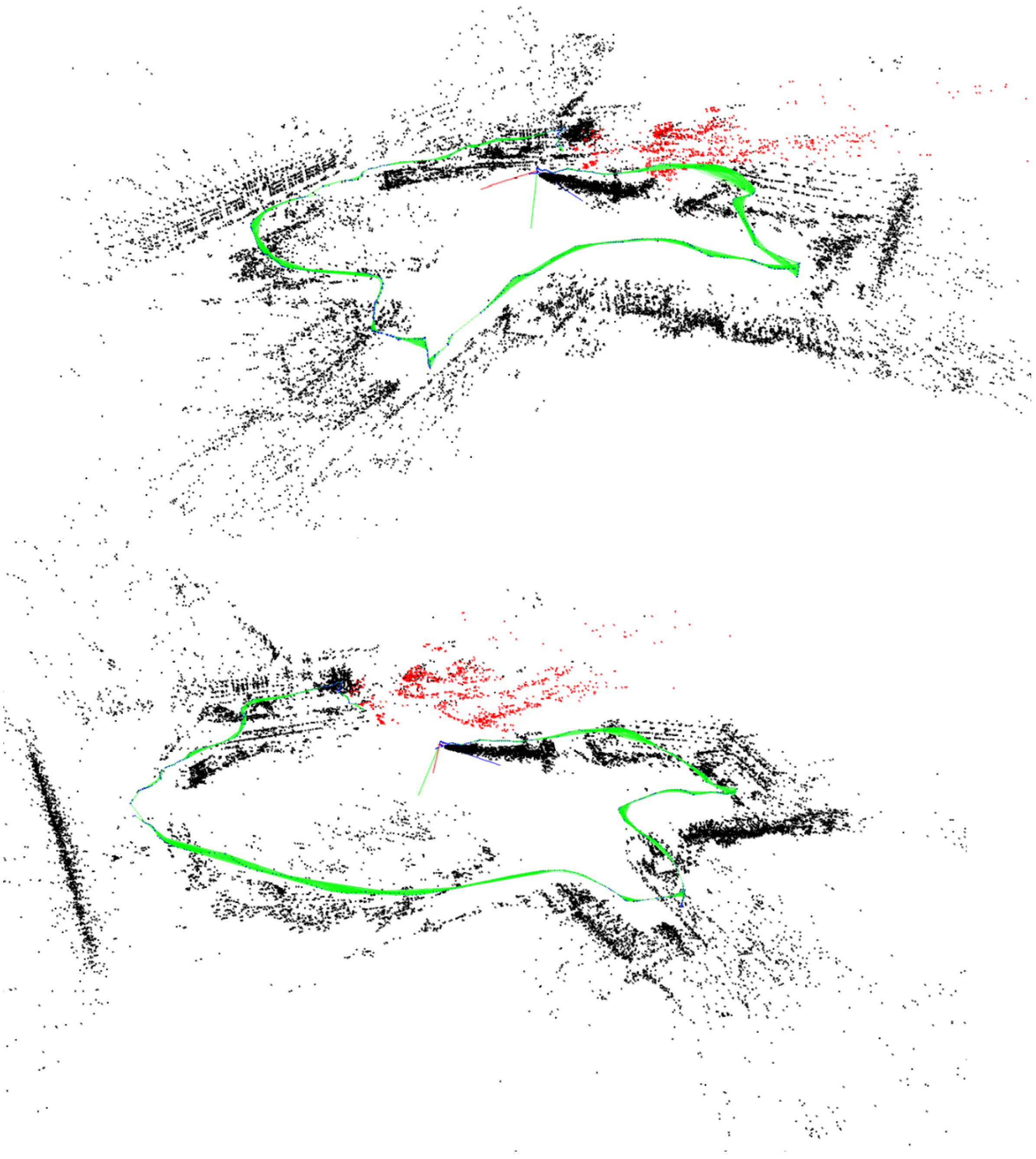


FIGURE 29: SHOWS A SECTION OF THE NMBU CAMPUS FROM TWO DIFFERENT ANGLES

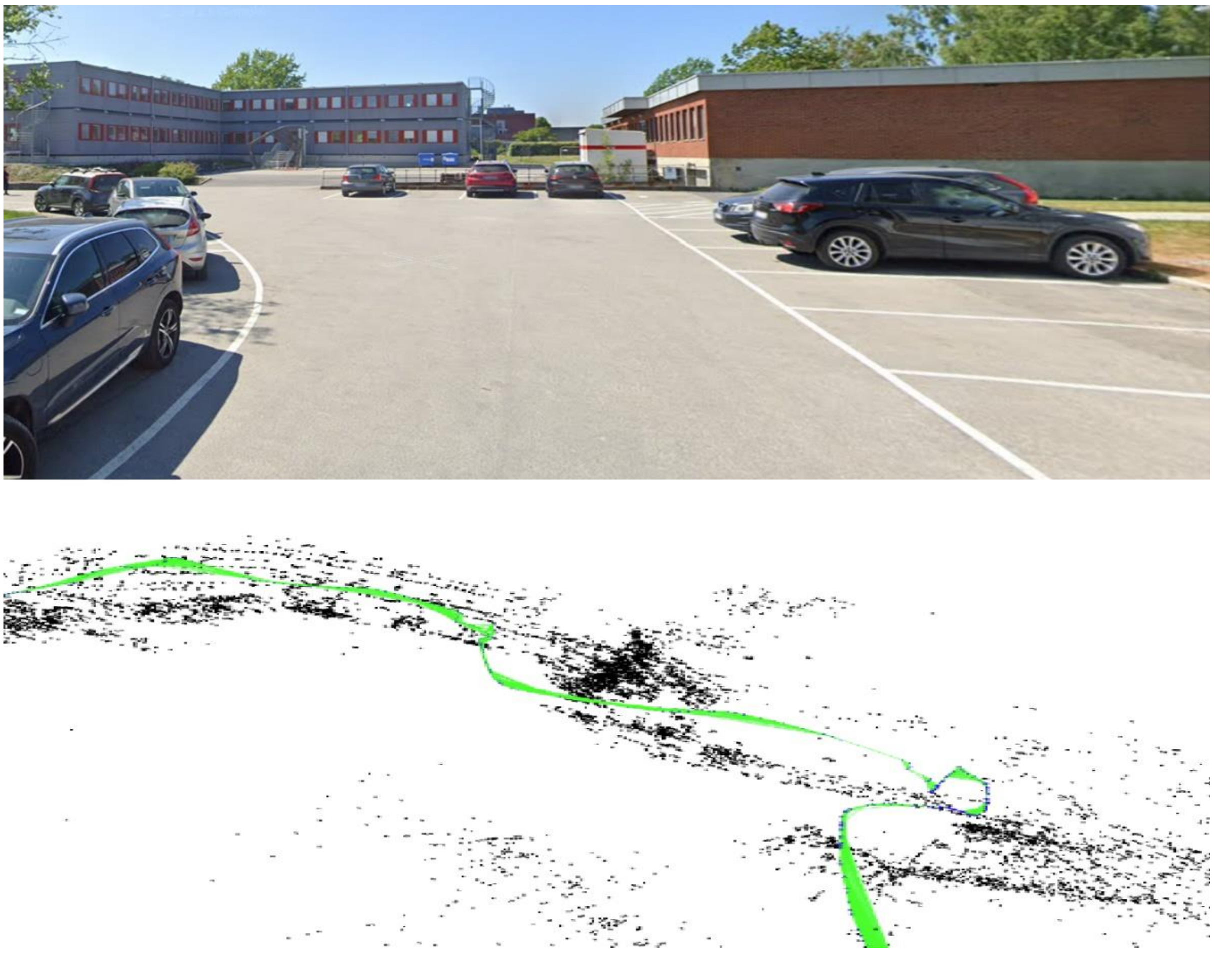


FIGURE 30: ZOOMED IN TO A SPECIFIC PART OF THE MAP

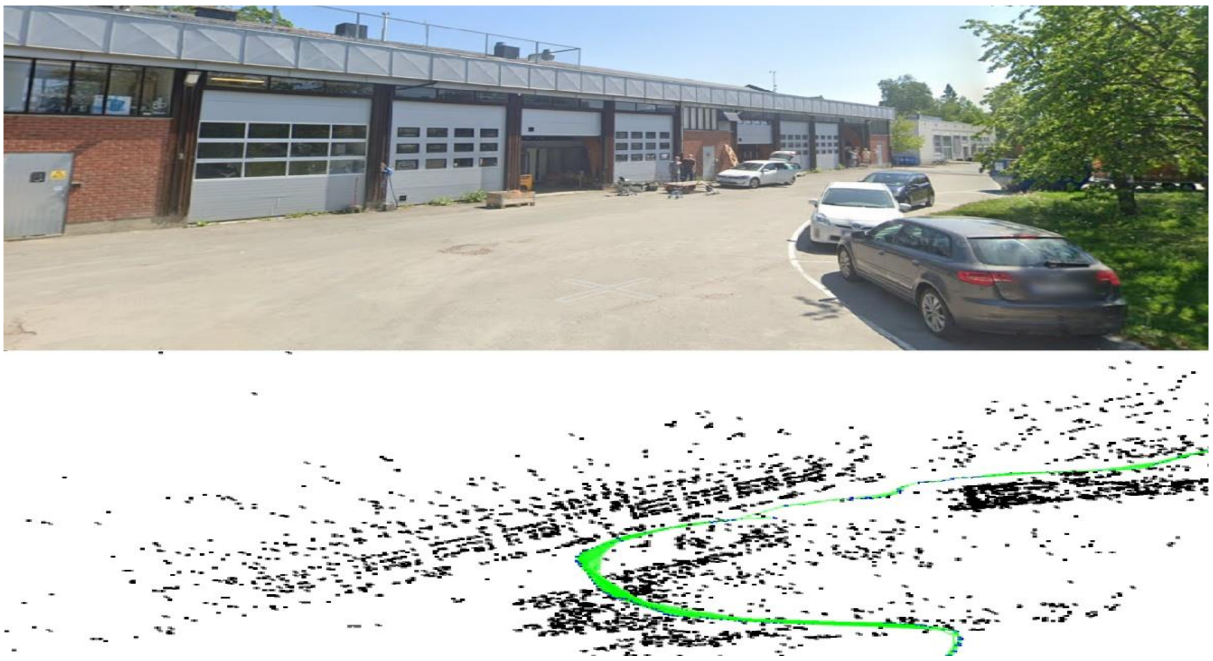


FIGURE 31: ZOOMED IN TO SHOW A DIFFERENT PART OF THE SAME MAP

The findings presented in this chapter together with the experimentation done in **5.Implementation and Experimentation** show that the method employed is capable of generating accurate and detailed mapping of various environments and in a broad array of scenarios. This proves the adaptability and versatility of drone technology in general and highlights the potential of combining low-cost drones with SLAM techniques.

This system proves to be a highly effective combination and a great budget-friendly alternative, capable of producing highly detailed maps.

7. Discussion

7.1 Practical limitations and challenges

When the drone was taken outside for outdoor testing, the limitations and sensitivity to wind became apparent. Its small frame and light weight often led to unnecessary battery consumption spent on corrections of flight paths, leading to redundant footage. Additionally, streaming issues, rapid motion blur, inadequate feature detection, or significant changes in the visual scene often led to localization issues and therefore map resets, which is a set feature to ensure the accuracy of the algorithm. The point cloud before the reset is stored locally in an attempt to merge with the newly generated point cloud, although more often than not, this feature led to the creation of a new map, starting from scratch, which resulted in a lot of redundant and unusable footage. Efforts to mitigate these issues involve flight patterns that reduce the need for sharp turns, with as few turns as possible combined with more training reducing the risk of localization issues significantly, resulting in more stable data gathering, and more precise maps. This also enabled the creation of bigger maps within the range of the capabilities of the drone.

The Atlas system, while enabling the connection of disconnected or “non-active” maps as the literature suggests, proved to be a valuable but frustrating function to work with. When a previously saved map is loaded, the system needs sufficient recognizable features to accurately relocalize the drone in the continued map. Unfortunately, this requirement often led to the map being reset at the beginning of the mapping session, resulting in losing said saved map. To effectively lessen the chance of these occurrences, the drone was placed approximately in the same location as it ended the previous session which proved to be somewhat effective.

The Atlas system seemed to work better for real-time applications as opposed to post-processing with the use of rosbags, which often triggered the previously mentioned map resets. Whether the root cause of these issues is related to how the function stores key features, or if there is an issue with how ROS handles rosbag data remains unclear and needs to be further researched.

7.2 Stream or Store; Real-time reactions or rosbag reflections

Using ORB-SLAM in real-time while flying the drone allows the users to interact and adapt to the changes in the environment immediately, allowing the operator to navigate complex spaces safely and efficiently. To effectively use this approach, one must ensure that the parameters of the algorithms are sufficient to provide as accurate and good maps as possible. The main drawback of this approach is the limited capacity when it comes to battery life. Even though the SLAM algorithm is run off-board, the flight time of the drone is limited. With ~10 minutes of flight time, the system does not allow for multiple take-offs and landing to update and tune the parameters mid-mission.

In comparison, post-flight SLAM allows for enabling the use of more computationally heavy algorithms or fine-tuning the parameters to generate higher-quality maps. However, the lack of real-time interaction puts a special emphasis on the operator's flying to ensure that the data gathered are both sufficient and exceptional for post-processing, as any errors or missed data are irreversible. Recording and storing rosbags might require excessive storage, this applies particularly if the operator records every topic published and especially for huge maps and prolonged mapping sequences.

In general, the choice between using these approaches depends highly on specific mission requirements. For instance, in search and rescue operations, surveillance, or any other applications where human lives are at risk, or time is crucial, real-time data is golden for immediate responses. Meanwhile, ecological surveys or mappings that might benefit from detailed and fine-tuned analysis would possibly benefit from post-flight processing enabling many options for tuning the data needed.

This project mainly leaned towards using a post-flight processing approach, running the ORB-SLAM on the rosbag. This method seemed logical as being able to tune the parameters without having to use excessive time and battery for doing the changes in real-time was considered superior.

7.3 Feature extraction limitations

As previously mentioned, it seems that the feature extraction limit was capped at around 2000 features per image no matter what the ceiling was set to, this seemed to hold true even with changing the threshold values as well. Probable factors can include:

- **Image content:** If the algorithm doesn't pick up enough distinct features due to a limitation to the video feed. Applying the algorithm to footage detailing a blank white wall will never provide enough feature points to make out a wall in the resulting point cloud. This is closely related to the next point of order, the threshold settings.
- **Threshold settings**: If the threshold settings are set too high, the detector might be disregarding features that otherwise could appear to be valid.
- **Computational resources:** This point might play a part and is a hypothetical one, however, it does make sense, especially if the hardware of the selected computer is insufficient, the algorithm might stop the feature extraction to maintain performance.

The reason why compute was included, is because even though the ceiling was set relatively higher than 2000, utilizing a low threshold, it never reached the maximum ceiling, another point if concern when it comes to compute, is that the hardware of the computer utilized might not be

Another possibility could be inherent limitations by the algorithm itself, but the literature does

7.4 Further work and research directions

Looking ahead, this project offers an area rich for further work. This section goes through possible segments for

Data preprocessing and augmentation: Being able to preprocess and augment the data is an area that should be further researched. By integrating deep learning, especially semantic segmentation the ORB-SLAM could be able to prioritize permanent features such as trees or buildings over dynamic ones that are not so relevant, like moving humans or animals. Semantic segmentation involves classifying pixels in images to categorize them, enabling the system to effectively understand the environment. [30] Applying techniques such as this could lead to more robust and accurate maps.

Autonomous navigation: By implementing some level of autonomy, the drones could effectively navigate through- and map the environment. The possibilities for this area are vast and include a variety of techniques, systems, and algorithms, depending on the intended use case. Reinforcement learning is a potential field for exploration for this type of project, and this machine learning process allows the agent/robot/drone to make decisions, based on the feedback on these decisions, learn from them, and choose the optimal decision. [31]

Charging function: Considering the battery life and the relatively short flight time of the Tello drone, there are also possibilities of incorporating a charging function into the autonomous navigation. Again, as the other areas mentioned for further work, there are many possibilities and many different solutions for this, at the core, the drones start by managing to read and react to how much battery is left before it is empty. Having a function that reacts to the battery level reaching a certain point, for example, 25- or 30% should allow the drone to reach back to a charging station. The location of the charging station could either be hardcoded into the drone's control software, using its relative position, or by using GPS coordinates. The location could also be marked using AruCo markers, which are black and white squared markers, with distinct readable binary patterns, and are useful for providing real-world coordinates. [32]
Integrating such a function with potential other drones on standby is ready to continue the work, potentially enabling collaboration between the drones, and sharing data for a more efficient and robust mapping process.

Expanding sensor- or drone use: This groundwork can also be expanded into the use of other sensors cameras or drones, creating systems that are more operation-specific, rather than the all-rounded, jack-of-all-trades system provided in this project. For the intended purposes, like in search and rescue. The incorporation of drones that are more reliable in strong winds, with a bigger battery and a longer range, equippable with IR or thermal cameras could dramatically improve the outcome of these types of operations, potentially being able to locate missing persons or other persons of interest.

8. Conclusion

The work in this thesis has resulted in the setup and implementation of a framework for performing ORB-SLAM3 on the DJI Tello drone. This project aimed to build a low-cost mapping solution offering an alternative to the other drones on the market with its significantly higher cost.

With its limited range of 100 meters and ~10-minute flight time, the Tello drone had its limitations, however, this was mitigated by the Atlas system in the ORB-SLAM package. This allows the users to save and load maps, to further expand and build on the already created maps. Through rigorous testing, it also became apparent that the drone had problems in windy conditions due to its lightweight design, however, it excels in controlled environments and indoors. Even though the hardware of the Tello cannot compare to other higher-end alternatives equipped with sophisticated sensors, the drone proves that it is very much suited for many research and development tasks due to its affordability and the many possibilities when it comes to customization and programming.

The integration of ORB-SLAM3 which proved to be highly compatible with the camera in the Tello paves the way for many possibilities and options in the world of drone-based mapping solutions, especially in the field of low-cost drone technology. Even though the algorithm has its limitations, especially when it comes to low-contrast environments with fewer distinct features to be extracted, ORB-SLAM3 proved to be a highly accurate SLAM algorithm, and with its open-source nature opens up for some level of customization, which allows for optimization and tweaks for better performance, while keeping the cost of the whole system to a minimum.

The research and experimentation conducted during this project resulted in a package for performing ORB-SLAM3 on the DJI Tello. By sharing it as a GitHub repository, the main idea for this framework reflects the general goals of this thesis; creating a foundational framework, and through its affordable nature, enabling a wide audience with other practitioners or scholars to further build and expand on the system.

Despite the issues and limitations of the system, it lays a solid foundation for future implementations, that could extend beyond the capabilities of the Tello, potentially revolutionizing the use of low-cost drones in life-saving operations.

Appendix: Supplementary Materials and Code

This section is dedicated to detailing code, value modifications, or additions that might prove necessary for further technical understanding of specific sections. The code removed or subject to modification will not be included to keep it as clear and concise as possible.

Appendix A: Stored values for camera calibration

```
# Camera calibration and distortion parameters

Camera1.fx: 966.87142

Camera1.fy: 954.66564

Camera1.cx: 534.18714

Camera1.cy: 395.60203


Camera1.k1: 0.096565

Camera1.k2: -0.162125

Camera1.p1: -0.003263

Camera1.p2: 0.025240

Camera1.p3: 0


Camera.width: 960

Camera.height: 720
```

Appendix B: Additions and modifications to the gamepad_teleop_node

```
#!usr/bin/env python3 #Comes as python2 by default


# Added method for f310 logitech controller
    def parse_f310(self, msg):
        if len(msg.buttons) != 12 or len(msg.axes) != 6:
            raise ValueError('Invalid number of buttons (%d) or axes
(%d)' % (
                len(msg.buttons), len(msg.axes)))
        self.L1 = msg.buttons[4]
        self.R1 = msg.buttons[5]
        self.LX = msg.axes[0]
        self.LY = msg.axes[1]
        self.RX = msg.axes[2]
        self.RY = msg.axes[3]


# Added parse_f310 to parse method for error handling and controller
parsing
    def parse(self, msg):
        err = None
        try:
            return self.parse_f310(msg)
        except ValueError as e:
            err = e
      raise err
```

Appendix C: The tello.launch file

```
<launch>

    <param name="use_sim_time" value="false" />


    <!-- Main node -->

    <node name="orb_slam3" pkg="orb_slam3_ros" type="ros_mono"
output="screen">

        <!-- change the topics according to the dataset -->

        <remap from="/camera/image_raw"    to="/tello/image_raw"/>


        <!-- Parameters for original ORB-SLAM3 -->

        <param name="voc_file"      type="string" value="$(find
orb_slam3_ros)/orb_slam3/Vocabulary/ORBvoc.txt.bin"/>

        <param name="settings_file" type="string" value="$(find
orb_slam3_ros)/config/Monocular/tello_calibration.yaml"/>


        <!-- Parameters for ROS -->

        <param name="world_frame_id"    type="string"   value="world" />

        <param name="cam_frame_id"      type="string"   value="camera"
/>

        <param name="enable_pangolin"   type="bool"     value="true" />

    </node>


    <!-- Visualization -->

    <node name="rviz" pkg="rviz" type="rviz" args="-d $(find
orb_slam3_ros)/config/ntuviral_no_imu.rviz" output="screen" />


    <!-- Trajectory path -->

    <node pkg="hector_trajectory_server" type="hector_trajectory_server"
name="trajectory_server_orb_slam3" output="screen" ns="orb_slam3_ros" >

        <param name="/target_frame_name"        value="/world" />

        <param name="/source_frame_name"        value="/camera" />

        <param name="/trajectory_update_rate"   value="20.0" />

        <param name="/trajectory_publish_rate"  value="20.0" />

    </node>

</launch>
```

Appendix D: The tello_calibration.yaml file [33]

```
%YAML:1.0


#-----------------------------------------------------------------------
--------

# Camera Parameters. Adjust them!

#-----------------------------------------------------------------------
--------

File.version: "1.0"


Camera.type: "PinHole"


# Camera calibration and distortion parameters

Camera1.fx: 966.87142

Camera1.fy: 954.66564

Camera1.cx: 534.18714

Camera1.cy: 395.60203


Camera1.k1: 0.096565

Camera1.k2: -0.162125

Camera1.p1: -0.003263

Camera1.p2: 0.025240

Camera1.p3: 0


# Camera frames per second

Camera.fps: 30


# Color order of the images (0: BGR, 1: RGB. It is ignored if images are
grayscale)

Camera.RGB: 1


# Camera resolution

Camera.width: 960

Camera.height: 720
```

Continuation of Appendix D:

```
# ORB Parameters

#---------------------------------------------------------------------------
--------


# ORB Extractor: Number of features per image

ORBextractor.nFeatures: 1250


# ORB Extractor: Scale factor between levels in the scale pyramid

ORBextractor.scaleFactor: 1.2


# ORB Extractor: Number of levels in the scale pyramid

ORBextractor.nLevels: 8


# ORB Extractor: Fast threshold

# Image is divided into a grid. At each cell, FAST is extracted imposing
a minimum response.

# In case of low contrast images, lower these images

ORBextractor.iniThFAST: 12

ORBextractor.minThFAST: 7


#---------------------------------------------------------------------------
--------

# Viewer Parameters

#---------------------------------------------------------------------------
--------

Viewer.KeyFrameSize: 0.05

Viewer.KeyFrameLineWidth: 1.0

Viewer.GraphLineWidth: 0.9

Viewer.PointSize: 2.0

Viewer.CameraSize: 0.08

Viewer.CameraLineWidth: 3.0

Viewer.ViewpointX: 0.0

Viewer.ViewpointY: -0.7

Viewer.ViewpointZ: -1.8

Viewer.ViewpointF: 500.0
```

Continuation of Appendix D:

```
#--------------------------------------------------------------------------------
# Atlas Parameters. To save and load the map. Creates a .osa extension file.
# Is only used when connecting maps for creating a bigger map detailing the same# area.
#--------------------------------------------------------------------------------
#System.SaveAtlasToFile: “map”
#System.LoadAtlasFromFile: “map”
```

Appendix E: minor tweak in System.cc to connect maps

```
//mpAtlas->CreateNewMap(); #swapping this for GetAllMaps to connect maps
mpAtlas->ChangeMap(mpAtlas->GetAllMaps()[0]);
```

References

[1] M. Stewart, “Twitter,” 10 July 2014. [Online]. Available: https://twitter.com/MarthaStewart/status/487060909165383680?ref_src=twsrc%5Etfw%7Ctwcamp%5Etweetembed%7Ctwterm%5E487060909165383680%7Ctwgr%5E9dd498955130a919de73a99c72b403c71e059915%7Ctwcon%5Es1_&ref_url=https%3A%2F%2Fwww.nbcnews.com%2Ftech%2Finnovation%2. [Accessed 14 May 2024].

[2] E. Alvarado, “The Commercial drone market in 2023: Insights and growth projections,” 2023. [Online]. Available: https://droneii.com/commercial-drone-market-2023#1525106654181-a2b63cd6-e0c3. [Accessed 18 November 2023].

[3] EverythingRobotics, “Visual SLAM on Tello Drone || ORB SLAM 3 || ROS Noetic,” 9 April 2023. [Online]. Available: https://www.youtube.com/watch?v=DFrfi85kKBY. [Accessed 2 February 2024].

[4] Aman226, “GitHub: Tello-ORB-SLAM-3,” 2023. [Online]. Available: https://github.com/aman226/Tello-ORB-SLAM-3/tree/main. [Accessed February 2024].

[5] MindMapping, “What is a Mind Map?,” [Online]. Available: https://www.mindmapping.com/mind-map. [Accessed 10 May 2024].

[6] whimsical, “mind-maps,” [Online]. Available: https://whimsical.com/mind-maps. [Accessed 10 May 2024].

[7] UAV Systems International, “Fixed-Wing Long-Range Drones,” 2022. [Online]. Available: https://uavsystemsinternational.com/pages/fixed-wing-long-range-drones. [Accessed 20 March 2024].

[8] AUAV, “Drone types: multi-rotor vs fixed-wing vs single rotor vs hybrid VTOL,” 2016. [Online]. Available: https://www.auav.com.au/articles/drone-types/. [Accessed 20 March 2024].

[9] RCDrone , “CUAV Raefly VT370 VTOL,” [Online]. Available: https://bit.ly/4cqAbT0. [Accessed 20 March 2024].

[10] PRODRONE, “PRODRONE develops the "Speed Delivery",” 2017. [Online]. Available: https://www.prodrone.com/release-en/2874/. [Accessed 23 March 2024].

[11] T. Zimmerman, “Single Rotor Drone - Things to know,” 11 May 2023. [Online]. Available: https://www.droidmen.com/single-rotor-drone-things-to-know/. [Accessed 11 April 2024].

[12] AVPAY, “AVPAY - The aviation marketplace and directory,” [Online]. Available: https://avpay.aero/company/embention/product/m600wp-multirotor-drone/. [Accessed 20 March 2024].

[13] Luftfartstilsynet, “Regelverk,” [Online]. Available: https://luftfartstilsynet.no/droner/regelverk-drone/. [Accessed 10 February 2024].

[14] DJI Oslo, “Ryze Tello,” [Online]. Available: https://djioslo.no/produkt/tello/dji-ryze-tello/. [Accessed 22 March 2024].

[15] Ryze Tech, “Tello,” 2018. [Online]. Available: https://www.ryzerobotics.com/tello. [Accessed 20 March 2024].

[16] Ryze Tech, “Tello Specs,” 08 January 2018. [Online]. Available: https://www.ryzerobotics.com/tello/specs. [Accessed 22 March 2024].

[17] Prisjakt, “Prisjakt - Ryze Tech Tello,” Ryze, 17 November 2023. [Online]. Available: https://www.prisjakt.no/product.php?p=4650753. [Accessed 23 March 2024].

[18] T. Taketomi, H. Uchiyama and S. Ikeda, “Visual SLAM algorithms: a survey from 2010 to 2016,” *IPSJ Transactions On Computer Vision and Applications,* 2017.

[19] A. M. Barros, M. Maugan, Y. Moline, G. Corre and F. Carrel, “A Comprehensive Survey of Visual SLAM Algorithms,” *Robotics,* vol. 11, no. 1, 2022.

[20] H. Durrant-Whyte and T. Bailey, “Simulatenaous Localisation and Mapping (SLAM): Part I The Essential Algorithms.,” *Robotics & Automation Magazine,* vol. 13, 2006.

[21] E. M. A. D. Ferreira, “Improving Lidar Odometry and Mapping in Real-time using Inertial Measurements,” 2021.

[22] C. Campos, R. Elvira, J. J. G. Rodriguez and J. M. M. T. J. D. Montiel, “ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial and Multip-Map SLAM,” *IEEE Transactions and Robotics,* vol. 37, no. 6, 25 May 2021.

[23] ROS.org, “Ubuntu install of ROS Noetic,” Open Robotics, [Online]. Available: http://wiki.ros.org/noetic/Installation/Ubuntu. [Accessed 03 January 2024].

[24] A. D. L. Escalera and J. M. Armingol, “Automatic Chessboard Detection for Intrinsic and Extrinsic Camera Parameter Calibration,” *MDPI - Instrumentation, signal treatment and uncertainty estimation in sensors,* vol. 10, no. 3, 15 March 2010.

[25] Roswiki, “How to Calibrate a Monocular Camera,” [Online]. Available: http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration. [Accessed 12 February 2024].

[26] StevenLoveGrove, “Pangolin on GitHub,” 2014. [Online]. Available: https://github.com/stevenlovegrove/Pangolin. [Accessed 12 February 2024].

[27] J. Benoit and G. Gael, “Eigen,” 2008. [Online]. Available: https://eigen.tuxfamily.org/index.php?title=Main_Page. [Accessed 2024].

[28] Logitech, “F310 Gaming-pad,” [Online]. Available: https://www.logitechg.com/no-no/products/gamepads/f310-gamepad.940-000138.html. [Accessed 15 February 2024].

[29] R. Mur-Artal, J. Tardos and J. G.-L. D. Montiel, “ORB-SLAM2,” 2016. [Online]. Available: https://github.com/raulmur/ORB_SLAM2. [Accessed March 2024].

[30] S. Hao, Z. Yuan and G. Yanrong, “A Brief Survey on Semantic Segmentation with Deep Learning,” *Neurocomputing,* vol. 406, 2020.

[31] Y. Li, “Deep Reinforcement Learning,” [Online]. Available: https://arxiv.org/pdf/1810.06339v1.pdf. [Accessed 21 April 2024].

[32] F. Romero-Ramirez, R. Munoz-Salinas and R. Medina-Carnicer, “Speeded up Detection for Squared Fiducial markers,” *Image and Vision computing,* vol. 76, 2018.

[33] R. Knowledgebase, “ORB SLAM2 Setup Guidance,” May 2019. [Online]. Available: https://roboticsknowledgebase.com/wiki/state-estimation/orb-slam2-setup/. [Accessed 20 March 2024].

Norges miljø- og biovitenskapelige universitet
Noregs miljø- og biovitskapelege universitet
Norwegian University of Life Sciences

Postboks 5003
NO-1432 Ås
Norway