Politecnico di Torino

Department of Electronics and Telecomunications

Master degree in Mechatronic Engineering

~ · ~

Cohort 2023-2024

Real-time Stereo Vision and RTK GPS-Based Mapping Framework for UAV Powerline Inspection

Supervisors

Marcello Chiaberge — *Politecnico di Torino*

Dieter Steiner — *Fraunhofer Italia*

Georg Egger — *Fraunhofer Italia*

Candidate

Andrea Ceola

Final exam October 2025

Abstract

Powerline inspection is a critical maintenance task performed by helicopters or, more recently, by manually piloted drones, both without automation or any real-time awareness of data completeness. Autonomous UAVs with advanced sensors have the potential to overcome these issues, but real-time mapping and autonomous navigation in complex outdoor environments remain challenging. Fraunhofer Italia aims to address these challenges by developing a UAV and ground station system to support maintenance teams in real-time infrastructure inspection, with a focus on powerlines.

This thesis, in collaboration with Fraunhofer Italia, proposes a perception framework for geo-referenced 3D point cloud mapping using stereo vision and high-precision GNSS. The resulting maps can serve as a basis for path planning and autonomous navigation. The system is developed and tested using a Pixhawk flight controller, Jetson Orin NX companion computer and ZED 2 stereo camera. While the Pixhawk provides GNSS and IMU data for pose information, the ZED 2 camera capabilities were exploited to handle the complex geometry of transmission towers, highlighting both its strengths and limitations. Built on the Robot Operating System (ROS), the framework is modular and interconnected, enabling real-time mapping performance. Due to limited onboard computational resources, further processing tasks such as path planning must be offloaded to a ground station. Since testing on a real drone was not feasible due to legal restrictions, a mock-up was built, demonstrating the potential of real-time UAV-based mapping of transmission towers.

Contents

Introduction

Unmanned Aerial Vehicles have become popular in recent years across several different fields. Firstly developed for military usage, nowadays they find applications in surveying, photography, agriculture, disaster management, security, entertainment and even in competitive drone racing [1]. Usually, UAVs are remotely-controlled by human pilots, but recent advances in research and technology are bringing autonomous flight one step closer to reality. Autonomy in UAVs can be interpreted in different ways, spacing from flight-assistance tasks to fully autonomous missions that require a full environmental awareness and decision making capabilities. To enable such autonomy, UAVs can be equipped with a variety of sensors that support human pilot maneuvering, provide flight decision information, and acquire application-specific data for either real-time use or offline processing. A distinction that should be made is between the two main types of UAV: fixed-wing and rotatory-wing. The first type, comparable to airplanes, are more popular in the military field due to high speed and heavy payload capacity. Rotatory-wings, or multi-copters, on the other hand, resemble helicopters and are the more interesting for many civil applications due to their ability of stationary flight, so called hovering.

0.1 Problem Statement

Nowadays, powerline inspection is typically performed by manually-piloted UAVs, sometimes with limited level of autonomy to assist the pilot in maneuvering the drone in such difficult environment, but piloted solutions require significant training effort for human operators, which is costly and time-consuming. In some cases inspections are still carried out by helicopters or by manual ground-based surveys. This thesis focuses on multi-copters and their role in autonomous infrastructure inspection, with the intention of replacing traditional helicopters and manually-piloted UAV operations. Drones have the potential to increase safety, speed, repeatability and decrease the cost of powerline inspections. At the same time, autonomous solutions provide more accurate data, also needed to let the drone have a full awareness of the environment. Despite recent advances in sensing technologies and onboard computing [2], inspection of infrastructure using UAVs still lacks fully autonomous solutions due to the challenging nature of infrastructures themselves, the difficulty of embedding the high computational load of autonomous mission in a flying platform, and the need for high precision to ensure safety. Moreover, autonomous robots themselves are a complex system that requires a multi-level architecture to divide the problem into sub-tasks:

- **Perception:** Acquiring and processing sensor measurements to extract useful information about the environment.

- **Localization:** Estimating the location of the drone within the environment.
- **Planning:** Planning the path according to mission objectives, environmental and legal constraints.
- **Control:** Control the actuators (propellers) to follow the defined trajectory.

Some of these tasks can be computationally expensive, UAVs typically rely on a ground-station to perform some of them. Tasks such as perception and control are usually done onboard since are interfaced directly with the environment, and control algorithms require direct feedback from sensors. Localization and planning, on the other hand, are tasks that may require high data storage and powerful computation, so are suitable to be performed off board on a ground computer.

Powerlines represent a particularly challenging case of study: UAVs are well suited to cover wide spatial area, but their thin and elongated structures are difficult to detect [3]. An additional challenge is represented by the different structure of transmission towers, that can change in relation to the voltage (high and medium tension), role (e.g. line endpoint) and environmental constraints. Moreover, achieving full autonomous inspection of infrastructure typically requires Artificial Intelligence for tasks such as object detection. However, this thesis focuses specifically on the mapping challenges of powerlines and leaves AI implementation as a future work.

Figure 1: Different structure of transmission towers

Since powerlines are outdoor infrastructures spanning large areas, UAVs must sustain long flights and handling high data throughput. To maximize endurance, both payload and computational resources must be carefully limited to reduce weight and power consumption. For this reason, in this work the main sensor for perceiving the environment is a depth camera, which provides both colored and depth

images. However, it will be shown that small and complicated structures like transmission towers and wires are difficult to detect for stereo vision systems, making it hard to obtain precise 3D maps (as LiDAR-based survey), implying that a different methodology must be adopted. In contrast, powerlines do not obstruct satellite visibility, making it possible to rely on Global Navigation Satellite Systems (GNSS) for position estimation. Moreover, high voltage represents a risk for the avionics components of the drone, requiring the UAV to maintain a safety distance to avoid potential hazards [4]. The dynamic nature of the environment must be also accounted, as well as errors in the map. An obstacle avoidance system able to detect any obstacles and react quickly to ensure safety should be considered. Together, these factors illustrate the need for a balanced, integrated approach that considers sensor selection, onboard processing, communication reliability, and mission safety to achieve fully autonomous UAV infrastructure inspection.

0.2 Contributions and Objectives

Autonomous UAV powerlines inspection is a complex and multidisciplinary problem that cannot be fully addressed in a single thesis. The work presented here is the result of six months of work, in collaboration with Fraunhofer Italia, to explore the feasibility of performing such inspection using a stereo camera and a high-precision GNSS. In particular, the development of a perception and localization framework to build 3D maps of the environment, for path planning usage, is the goal of this thesis. Since the company is based in a no-fly zone due to the vicinity of an airport, real UAV experiments could not be performed. It was therefore decided to build a ground platform, ensuring a rigid mounting of components. This drone mock-up would have been more suitable for rapid testing, at least in this phase of the project, where only the perception and localization have been analyzed. Of course this came at the cost of the impossibility to sense the wires, restricting to focus on the transmission towers.

The contributions of this work can be summarized as:

1. **System Design:** A survey of the state of the art in UAV-based infrastructure inspection, with a specific focus on perception and localization approaches. This provided the foundation for identifying suitable software solutions given the available hardware.

2. **Hardware and Software Integration:** A rigid structure to emulate a drone platform was built. This drone mock-up integrates a ZED 2 stereo camera, a high-precision GNSS receiver, a companion computer and a PX4 flight controller. The software framework was based on ROS 2, well-suited to interface with all the components.

3. **RTK GNSS Correction Pipeline:** Development of a custom pipeline to transmit GNSS corrections from a base station to the onboard GNSS module, enabling centimeter-level positioning.

4. **Perception and Localization Framework:** A modular ROS 2 pipeline was designed to:

    - Manage the acquisition and synchronization of data streams.

- Implement a geo-referencing algorithm to project camera pixels into the global world frame.

The subsequent adaptation of this pipeline for mapping, including filtering and CUDA-based programming, was carried out by a colleague at Fraunhofer Italia. [5]

5. **Evaluation and Testing:** Validation of the integrated framework through simulation (SITL with Gazebo) and field experiments[1]. Evaluation focused on software integration, GNSS precision, geo-referencing accuracy and performance analysis.

In summary, the main contribution of this thesis is the design, integration, and validation of a perception and localization framework for UAV powerline inspection, demonstrating the feasibility of combining stereo vision and RTK GNSS to achieve reliable mapping under real-world constraints.

0.3 Thesis Outline

The structure of this thesis is designed to introduce the background, system design, implementation and evaluation of the proposed framework. Following this introductory chapter, the thesis is organized as follows:

1. **Chapter 1 - State of the Art:** Provides an overview of related work in UAV-based infrastructure inspection, stereo vision systems for mapping and the modern approach to robotics.
2. **Chapter 2 - System Design:** Describes the hardware and software components and the architecture adopted. The integration of every component is also explained.
3. **Chapter 3 - Development:** Explains the implementation of the perception and localization framework, including the RTK GNSS correction pipeline, geo-referencing process and mapping.
4. **Chapter 4 - Results and Discussion:** Presents the outcomes of field experiments, analyzing the accuracy, performance and limitations of the developed pipeline. Results are discussed in relation to the objectives of the work.

Finally, **Conclusion** summarizes the main challenges encountered, the lessons learned and proposes directions for future work.

[1] Mapping tests were performed on the lower parts of transmission towers using the UAV mock-up, due to the impossibility of real flight tests.

Chapter 1

State of the Art

1.1 UAV Powerline Inspection Approaches

Over the recent years, UAVs have become more common for infrastructures inspection. They provide highly detailed inspection without exposing human operators to any hazardous situations. At the same time, they reduce cost and time of inspections, also allowing to increase the repeatability. Although manually piloted drones are widely used, research is gradually moving towards fully autonomous systems, where UAVs can operate with minimal human intervention. Despite recent progress, autonomous inspection of powerlines remains challenging due the highly constrained operational environment. UAVs must navigate around tall towers, avoid thin wires and maintain safe distances from high-voltage equipment. Moreover, powerlines extend in a large area, which imposes additional limits on flight duration, and consequently on payload and computational resources. Research on various approaches has produced innovative solutions, both regarding the drone configurations and the processing algorithms. An example is the AERIAL-CORE project, a collaborative initiative coordinated by the University of Seville. It aimed to advance state of the art technologies for the inspection and maintenance of aerial infrastructure using unmanned aerial cognitive vehicles[1]. A large numbers of paper have been published since 2020, covering topics such as perception, manipulation, multi-UAV collaboration and AI integration. The work in [6] presents several application of the technologies developed during the project, demonstrating how different challenges have been addressed using different type of UAV configuration and sensors.

Autonomous UAV Architectures

Autonomous UAVs are designed with a multi-level architecture, where each layer is responsible for a specific function while interacting with the others [7]. As shown in Figure 1.1, the first action of an autonomous robot is perception: read sensors measurements and elaborate them to achieve knowledge of the interaction between itself and the environment. These information are processed by a localization module that elaborate a map of the environment and the pose of the robot within this map. Then, according to mission goals, the optimal path for the robot is planned. Finally the control module computes the actuators commands to make the robot following the computed path. This is typically done with a feedback loop, where

[1] Link to the online page of the AERIAL-CORE project

sensors measurements are considered as well in the computation of the actuator commands, compensating for any disturbances within the environment, the robot or the interaction of them.

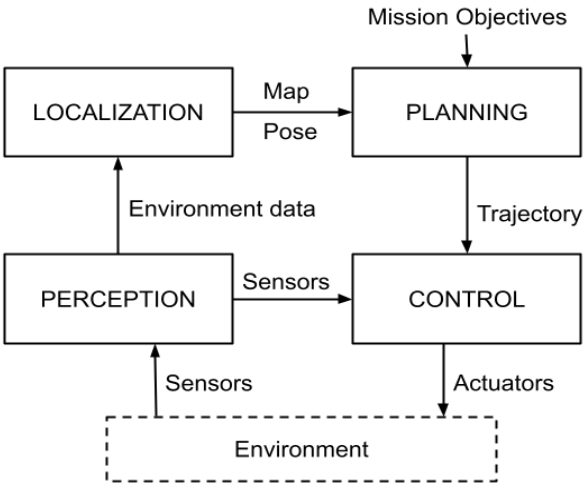


Figure 1.1: Typical architecture for autonomous robots

Perception Techniques for Powerline Inspection

To implement UAVs perception module, typical sensors that could be found for powerline inspection purposes are reported in Table 1.1. Sensors can be specific for the inspection task or for autonomous flight. Ideally, a sensor should provide useful data for both, minimizing the overall payload.

Sensor	Output	Purpose	Pros	Cons
Camera (RGB)	2D color images	Visual inspection of powerlines; object detection and scene recognition	Lightweight, passive low cost	Strongly affected by lighting conditions
Stereo Camera	Depth map	3D reconstruction of environment; obstacle detection	Lightweight, passive	Difficult to detect thin object; performance decreases exponentially with the distance
Thermal Camera	Infrared images	Detect hot spots in conductors or insulators	Works in low-light; useful for identifying defects not visible in RGB	Expensive; low spatial resolution
LiDAR	3D point cloud	Precise mapping of towers and wires; obstacle detection	High accuracy, robust to lighting	Heavy, expensive, power-consuming
Rangefinder (Ultrasonic, Radar or Laser)	Distance to object	Simple obstacle avoidance, altitude estimation	Lightweight, low cost	Limited range and field of view
IMU (Inertial Measurement Unit)	Gyroscope, barometer, magnetometer	Attitude estimation, control stabilization	Small, lightweight, high-frequency data	Drift accumulates over time
GNSS	Global position estimate	Global localization, map georeferencing	Provides global coordinates; with RTK reaches cm-level precision	Requires open sky view; needs correction link for RTK

Table 1.1: Overview of common sensors used in autonomous UAV powerline inspection.

Cameras are the most common sensors for inspection because of their low weight and flexibility. Stereo cameras can estimate depth, enabling 3D reconstruction of the environment. However, in powerline inspection, this approach faces challenges. The structure of transmission towers is geometrically complex and made of thin steel lattices, while wires are thin and barely visible in the image, depending on the background and lighting conditions.

LiDAR sensors provide more reliable 3D measurements and can capture thin structures accurately, but they are often heavy, expensive, and power-consuming, which limits flight duration.

Thermal cameras are useful to detect critical components, such as overheated power equipment, but their contribution to flight-related information is limited. Current research is adopting a hybrid approach, to take advantage of the strengths of some sensors and using others to compensate the weaknesses. For example, the work in [8] presented an interesting solution to detect overheated components using a thermal camera, and localize them using a stereo camera.

Another study analyze the use of RGB camera and mmWave (millimeter Wave) radar to detect cables and accurately estimate their pose. Integrating these information in the ROS 2 framework allow to use these pose for path planning, achieving a safe navigation [9].

Multiple approaches are based on camera and LiDAR, which generally work well, but the weight of the drone, power consumption and consequentially the flight time is not satisfactory. Similar to this thesis, the work in [10] uses a stereo camera to provide RGB-D images to feed a Neural Network that performs color segmentation to detect pipes of an industrial plant, and fuses depth information to compute their pose. While this approach works well for detecting pipes, thinner structures like powerlines remain challenging for stereo vision system.

Localization and Mapping

For powerline inspection, mapping is not only about reconstructing the local environment: maps must be georeferenced to be useful for maintenance and repeatability. High-precision GNSS, often with Real-Time Kinematic (RTK) correction, provides centimeter-level positioning. In areas where GNSS signals are weak or obstructed, visual-inertial odometry or SLAM techniques can provide supplemental positioning. Although these methods are mainly developed indoors, some outdoor solutions exist. As reported in [11], most outdoor localization and mapping algorithms studied in the recent years relied on vision-based sensors alone, as well as combined with LiDAR or ranging sensors. The common approach is to feed SLAM algorithm (Simultaneous Localization and Mapping), then fuse data with GNSS to obtain an highly-detailed map with georeferenced landmark.

Almost all SLAM methods rely on visual features to estimate their pose, with the advantage of having a smooth odometry source. However, in contrast, are more likely to drift over time and are affected by various sources of error. For this reason, only certain points are georeferenced, maintaining the map precise. However, [12] revealed that visual inertial odometry (VIO)-based SLAM algorithms struggle under bad lighting conditions and with propeller-induced vibrations.

Using GNSS as the primary pose source for mapping would allow all points to be georeferenced in a single step. However, GNSS signal are noisy and generally do not provide a smooth trajectory. RTK systems on the other hand, significantly enhance the precision of GNSS positioning, and simple filters are enough to obtain smooth trajectory [13]. This approach will be evaluated in this thesis.

Obstacle Avoidance

Mapping, due to its heavy computational load, is usually performed at a low frequency. For this reason, typically, a faster obstacle avoidance module is implemented in autonomous robots. It is responsible for detection and avoidance of obstacles that would collide with the UAV, including dynamic objects and covering for error in main the map. The architecture of this system is similar to standard mapping, but with high frequency sensor and fast algorithms with small memory storage, limiting the map to the area next to the UAV. In the context of infrastructure inspection, the drone is moving slowly, allowing a redundant detection of the same objects. Anyway, dynamic objects and noise-related errors induce to include this module also in inspection platforms for safety reasons.

Usually, this system is independent from the main mapping pipeline. It can be achieved with many types of sensors, but a trade off between robustness and cost[2] must be achieved. Fusing multiple sensors, including the main mapping ones, is also an option. In particular, [14] shows that multi-sensors fusion is the most promising solution, as it improves detection efficiency and perception completeness. On the other hand it adds complexity in calibration, synchronization and data alignment, However, for UAV missions, it seems to be impossible to rely on a single sensor, since its weaknesses must be compensated by other sensors to handle edge cases and ensure safety. Another study highlights that the extra complexity of fusion approaches require additional computational load, emphasizing the need for efficient algorithms and methods to enable real world applications [15].

In this thesis, obstacle avoidance is acknowledged but not implemented, since the focus is on perception and localization.

Data Management

Autonomous missions involving mapping deal with huge amounts of data. Sensors themselves provide high-throughput data that the perception module must handle in real-time, typically onboard the UAV. Mapping algorithms usually generate point cloud data, which grows with the map dimension. Managing this data onboard would require a lot of computational resources, which is not always feasible. For this reason, a common approach is to perform perception and control onboard, since these modules directly interact with the environment and control module requires real-time perception data for feedback control algorithms. Localization, mapping and planning, on the other hand, can be offloaded to a powerful ground-station machine. This approach has the advantage of enabling human supervision and saving important onboard resources, which translates into longer flight time.

Real-time operation is critical for UAV missions, and the communication link between drone and ground station becomes a key element in the pipeline. This link must be stable and capable of transmitting large amounts of data quickly. Typical options include Wi-Fi, LTE or 5G as can provide high bandwidth and relatively stable connections. However, standard Wi-Fi is limited in range and LTE depends on the availability and quality of cellular infrastructure. Another study explored the use of a dedicated Wi-Fi-like data link module to enable real-time transmission of high-resolution images during SaR (Search and Rescue) missions, demonstrating solid performances at distances up to few kilometers [16]. While LTE can offer slightly better performance, it has the significant drawback of relying on external infrastructure, in terms of coverage, network load and quality of service.

Environmental Challenges

UAV flights face several environmental challenges, to which infrastructure inspections add further complexity. Aerial vehicles operate in a 3D environment that requires advanced perception methods to extract all relevant features, and the amount of mapping data generated increases quickly. Moreover, the eventuality of dynamic obstacles should be considered, making path planning and obstacle avoidance modules more difficult [17]. UAVs are typically small and lightweight, making them

[2]Cost intended as the overall impact on the system

sensitive to weather conditions and requiring robust control algorithms to ensure safe and reliable flight.

In addition to flight stability, weather and terrain also affect the inspection task itself. For example, vegetation can obstruct the view of part of the infrastructure, either by covering certain components or by limiting the ideal trajectory of the drone. Additionally, the infrastructure itself can present challenges, such as electromagnetic interference from high-voltage components, or thin cables that are difficult to detect and avoid during flight. All these factors contribute to the complexity of the system, requiring careful attention during the design of the platform to ensure effectiveness and safety.

Safety Regulations

Safety is a key aspect of UAV operations: drones are subject to numerous restrictions and constantly changing laws. Although there are some common directories, such as those given by the European Union, each Country has its own regulations that define areas and methods in which drones can fly. For example, no-fly zones and altitude limits in the vicinity of airports affected this thesis, since Fraunhofer Italia is based right next to Bolzano airport. For this reason, in this work a ground platform that emulates the drone is adopted for testing.

Regulations also dictate safety aspects of the technology employed, both hardware and software, adding necessary workload to engineers. Due to the hazardous nature of flight, strict certification processes and redundancy requirements are essential to mitigate risks and ensure functional safety [18].

1.2 Stereo Vision for Mapping

As mentioned, stereo vision systems are particularly attractive for autonomous tasks, since they can provide both RGB and depth information with a relatively lightweight, passive sensors. However, their performance is tightly linked to image resolution, scene texture, and distance from the target, which poses significant challenges for powerline inspection. In the following, the principles and limitations of stereo vision are reviewed, together with the main algorithms and a comparison between popular commercial stereo cameras.

Principle of Stereo Vision

A stereo vision system relies on two cameras mounted at a fixed baseline B, usually parallel. By analyzing the disparity d between corresponding points in the left and right image, the depth Z of each point can be estimated according to:

$$Z = \frac{f \cdot B}{d} \tag{1.1}$$

where f is the focal length of the cameras. Depth accuracy is therefore inversely proportional to the accuracy of disparity estimation. This principle highlights two important limitations. First, disparity becomes very small when the UAV is far from the object, leading to high uncertainty in depth. Second, the discrete nature of digital images imposes a minimum measurable disparity limited by sensor resolution, which effectively limits the maximum reliable range of the system [19]. In particular,

stereo vision accuracy is determined by the relationship between depth, disparity, and image resolution. The depth resolution D_r varies quadratically with distance Z, according to

$$D_r = Z^2 \cdot \alpha$$

where α is a camera-dependent constant. This implies that accuracy decreases rapidly as the distance increases.

For powerline inspection, additional difficulties arise from their thin structures and wires. Most of the algorithms define a matching window in pixels units, used to compare left and right images. Objects like wires may be thinner than this window and thus discarded as noise. In other cases, lack of texture or low contrast with the background can lead to ambiguous matches. These limitations are intrinsic of stereo vision principle and can only be compensated by hardware or software performance.

Stereo Matching Algorithms

The key to stereo vision is the stereo matching algorithm, which finds corresponding pixels between the two images, allowing to compute the disparity. Common approaches can be broadly divided into two groups:

- **Computer Vision:** traditional approaches based on computer vision techniques. For example:
  - **Local Methods:** Use a fixed window to compare pixel intensities between images. They are simple and fast, but struggle with thin objects and textureless regions.
  - **Global Methods:** Formulate stereo matching as an optimization problem: they find a disparity map by minimizing a global energy function. They can achieve high accuracy, but are computationally expensive.
  - **Semi-Global Methods:** Approximate global optimization by aggregating matching costs along multiple one-dimensional paths across the image [20], providing a good trade-off between accuracy and efficiency.
  - **Feature-Based Methods:** Instead of comparing pixel intensities, these methods extract geometric features such as corners, edges, or planes to establish correspondences [21]. They are more robust in textureless regions, but generally more complex and often produce sparse disparity maps.
- **Deep Learning:** Modern approaches use neural networks trained on large datasets to compute the matching cost and directly regress disparity. They achieve state-of-the-art performance, but require substantial GPU resources. In this thesis, it is shown that it was the only approach able to reliably detect transmission towers with acceptable accuracy, using the ZED 2 camera.

The choice of the algorithm should be determined by the scene of interest and the available computational resources [22].

Overview of Stereo Cameras on the market

In addition to the choice of matching algorithm, the hardware of the stereo camera has a significant effect on performance. Several commercial stereo cameras are available today, offering different trade-offs between resolution, range, size, and onboard

processing capabilities. Popular examples include Intel RealSense D400 Series, Luxonins OAK-D and ZED 2 by Stereolabs. A comparison of these cameras is provided in Table 1.2.

Specification	**ZED 2**	**RealSense D430**	**OAK-D Color**
Baseline	120 mm	50 mm	75 mm
Depth Range	0.3 – 20 m	0.2 - 10 m	0.8 - 12 m
Resolution (depth)	1920×1080 at 30 fps	1280×720 at up to 30 fps	1280×800 at 120 fps
Additional Sensors	IMU	IMU	IMU
Field of View (FOV)	110° H × 70° V × 120° D	86° H × 57° V × 94° D	69° H × 55° V × 81° D
Frame Rate (Depth)	Up to 100 fps (depends on resolution)	Up to 90 fps (depends on resolution)	Up to 120 fps
Interface / Power	USB interface / Average 1.9 W	USB interface / Up to 3.5 W	USB interface / Up to 3 W
Size / Weight	175×30×43 mm / 230g	71 x 14 x 11 mm / 70g	97×30×23 mm / 115 g

Table 1.2: Comparison of technical specifications between ZED 2, Intel RealSense D430, and OAK-D.

The ZED 2 camera was used in this thesis. According to the datasheet [23], it should satisfy requirements such as:

- Wide field of view and long depth range, to capture large scenes or outdoors.
- Great accuracy at longer distances thanks to the wide baseline.
- Integrated IMU can help with pose estimation and fusion with SLAM or odometry pipelines.
- High resolution RGB-D data.
- Strong SDK support, including tools for depth, point-cloud generation and ROS 2 integration as well.

However, a trade-off needs to be considered. Using high resolution data adds computational load, which means that more CPU, GPU and bandwidth is needed.

A performance evaluation of the ZED stereo camera under variable illumination and range conditions is presented in [24]. The authors report that within short to medium ranges (e.g. 1–10 m), depth estimation is relatively reliable, but error grows nonlinearly beyond that, especially when disparity becomes small or image contrast decreases. Their observations confirm the theoretical limitations of disparity-based depth estimation. However, those tests have been done on a large checkerboard on an indoor wall, which makes it hard to define what to expect when it comes to detect thin outdoor objects. Nevertheless, an important lessons learned from this study, is that the depth range for accurate measurements is well below the 20 m declared in the datasheet.

1.3 ROS 2 Framework for Real-Time Autonomous Robots

The ROS 2 (Robot Operating System 2) framework is a middleware that has improved robot development, especially for autonomous systems that require real-time

capabilities, distributed architectures and reliability. This section attempts to provide an overview of how this middleware has changed the world of robotics in the last decade.

Early Middleware and Frameworks (1990s–2006)

Initially, robotics software was developed ad hoc by each research group specifically for their robots, without any framework or middleware. It was typically written in C or assembly, and lacked a structured architecture. During late 1970s the concept of software architecture began to grown, but still built for specific robotic platforms. As robots became more complex, the need arose for standard communication protocols to interface hardware and software components, as well as reusable frameworks [25]. By the 1990s, the idea of open-source middleware/framework became popular, some of the most famous in robotics are:

- **Player/Stage Project:** Released in 2000, it is a community free software project, based on a client-server architecture. Player provides a network server for robot and sensor control, while Stage is a 2D/3D environment simulator [26].
- **YARP:** Yet Another Robot Platform was published in 2006 with the goal of facilitating code reuse and modularity. It is a software package that provide support for inter-process communication, image processing and hardware integration [27].
- **RT-Middleware:** Developed in Japan around 2005, the Robot Technology Middleware is a standard that aimed to promote open robotic architectures. It defines robotics elements, such as sensors, as RT-Components and proposes a framework for a system development based on these components [28].

ROS 1 Era

The Robot Operating System was first introduced in 2007. Building on robotics research at Stanford University, Willow Garage[3] developed software to run on one of its robots, naming it ROS. It was designed to be an open-source middleware, not an operating system (OS) in the traditional sense. It provided a structured communication architecture that could run on various host operating systems [29]. A large number of institutions contributed to ROS, forming a worldwide ecosystem from the beginning. In the following years, it was further developed and gained popularity, also thanks to the launch of the Q/A forum named ROS Answers in 2011. Some key elements that distinguished it from previous solutions are:

- Publish/subscribe model, node modularity, several message types.
- Large active community, robotic kits for academic/research use.
- Tools for simulation, visualization, logging (e.g., rviz, rosbag).
- Language support: C++03 and Python 2 were standard. C++ allowed lower latency implementations, while Python offered easier prototyping but tended to be slower (due to interpreter overhead, garbage collection, etc.).

[3]Willow Garage was a robotics lab and incubator in California, USA

However, it had some important limitations, especially relevant for UAV and real-time applications:

- **Central master node architecture:** A central ROS Master that handled communication, which could become a bottleneck in multi-robot or distributed systems.
- **Lack of real-time execution requirements:** The communication system used a networks-based transport (UDP/TCP) that does not provide guarantee about message delivery timing.
- **Lack of security mechanisms:** It was designed for academic usage, where security was not a major concern.

To overcome these limitations and respond to the needs of both the industry and the research community which was developing more complex systems, Open Robotics released ROS 2 in 2017. It was a completely new project, designed from scratch to avoid breaking ROS 1 due to the substantial changes.

The Advent of ROS2

During the design of ROS 2, other third-party middleware were explored by ROS 2 developers to overcome ROS 1 limitation. Data Distribution Service (DDS) was the most promising solution, and was chosen as the ROS 2 middleware. It is a standard for real-time and secure data distribution, which was exactly what ROS 2 was aiming at. Relevant ROS 2 feature [30], especially for UAV/autonomous flight, are:

- **Decentralized communication via DDS:** Real-time capabilities, better suited for multi-robot and large-scale distributed systems.
- **QoS policies on topics:** Quality of Service allows tuning the trade-off between reliability and best-effort for each topic [31].
- **Better language support:** Uses modern C++ and Python libraries built on a common C library.
- **Security:** DDS security, encryption and authentication options.
- **Cross-platform support:** Runs natively on Linux, Windows, embedded systems, real-time OS, etc.

Even with these improvements, ROS 2 has its own challenges::

- **Real-Time:** ROS2 provides the framework and tools to reach real-time performance, but achieving it depends on the system configuration [32], as well as hardware and software components. Optimal system design is required.
- **Language Choice:** Using higher-level languages such as Python, which are interpreted rather than compiled, introduces overhead which can increase latency. On the other hand, C++ allows for lower level programming, and the compiler translates it to machine code, which is executed faster. Chosing the correct language for each node is crucial.

- **Resource Constraints:** Embedded hardware limits the available computational resource. ROS2 introduces extra overhead, which increases computational load compared to lightweight protocols. This can reduce the resources available for the application software, requiring a careful trade-off between ROS2 functionalities and system efficiency.

Overall, ROS 2 can be considered a solid choice for research, academia, and industrial applications, thanks to its rich set of tools, flexibility, and strong community support. In the context of this thesis, ROS 2 facilitated the integration of multiple sensors and enabled the implementation of perception and mapping algorithms across different programming languages and by different team members. Furthermore, it provided powerful tools for simulation, data visualization, and performance evaluation, contributing to a modular and efficient development process.

A more detailed analysis of ROS 2 environment is provided in Section 2.2.1.

Chapter 2

System Design

2.1 Hardware Components

The hardware platform used for this work was selected to balance flexibility, computational requirements and onboard resource limitations. As mentioned, flying a real drone was unfeasible due to legal restrictions in the area of interest, but a rigid frame on which to mount the components was necessary for hardware-in-the-loop and field testing. For this reason, a non-flying drone mock-up was built to accommodate the hardware needed for the framework. It consists of a flight controller, a companion computer, a stereo vision camera and a RTK-capable GNSS system, along with the battery and the necessary connectors.

(a) *Pixhawk 6c* (b) *Jetson Orin NX* (c) *ZED 2*

Figure 2.1: Images by the respective official website already cited: (a) Flight Controller. (b) Companion Computer. (c) Stereo Camera

2.1.1 Flight Controller

The flight controller is central for every flying platforms. It is responsible for low-level vehicle control, including sensor fusion for attitude estimation, motor control, communication with onboard and off-board sensors and platforms. For this work, a Pixhawk 6C was chosen, running the PX4 Autopilot firmware (v1.15). PX4 is an open-source platform widely used in both research and industry, ensuring reliability and community support. Pixhawk offers a high-performance processor, a wide set of communication interfaces and multiple IMU redundancies, making it a solid choice for a research platform. Although a real drone was not flown in this work, it ensures realistic hardware-in-the-loop integration and data exchange with the rest of the system. In particular, it provided IMU and GNSS data to the companion computer,

which were essential for the perception and mapping framework. Moreover, PX4 includes an Extended Kalman Filter (EKF) for all sensors, providing consistent data up to 100 Hz.

2.1.2 Companion Computer

A companion computer that provides onboard computational performance is mandatory for real-time perception tasks such as stereo depth estimation and 3D mapping. On the other hand, more resources imply more weight and power consumption, limiting the drone's flight time. A trade-off is found in the NVIDIA Jetson Orin NX module. The Orin NX provides a 6-core ARM CPU and an Ampere GPU with 1024 CUDA cores, optimized for parallel computation. This allowed the execution of the depth estimation and the point cloud mapping algorithms onboard. A more detailed analysis of performance is provided in Section section 4.5.

Feature	**Value**
Memory	8 GB 128-bit
Ampere GPU	1024 CUDA cores
Max Operating Frequency	765 MHz
Size	69.6 x 45 mm
Peripheral Interfaces	3x USB 3.2, 3x USB 2.0
	3x1 + 1x4 PCIe
	3x UART
	2x SPI
	4x I2C
	1x CAN
	DMIC
	DSPK
	2x I2S
	15x GPIOs

Table 2.1: Jetson Orin NX 8GB Specifications [33]

Despite its considerable computational capability, the Jetson Orin NX remains limited for achieving full UAV autonomy, especially when additional modules such as path planning or advanced navigation are considered. A practical solution is to offload computationally demanding tasks to a more powerful ground-station computer, with the Jetson transmitting processed mapping updates over a wireless link. This architecture balances real-time onboard perception with off-board computation, and represents a realistic design path for UAV-based inspection systems.

2.1.3 Stereo Camera

As mentioned in the previous Chapter, stereo vision is a passive depth-sensing technique that estimates the distance of objects by comparing two slightly offset images, mimicking human binocular vision. By computing the disparity between corresponding points in the left and right images, it is possible to reconstruct a dense three-dimensional representation of the scene. Unlike active sensors such as LiDAR,

stereo cameras do not emit their own signal, which makes them lightweight and low power consuming, well suited for onboard applications where power efficiency is an important constraint. The Stereolabs ZED 2 camera was used in this project. However, accuracy in outdoor scenario with thin object was unknown.

The ZED 2 relies on machine-learning models to improve depth estimation, but offloads the computation to the host GPU, which makes the choice of a capable companion computer critical. In addition to stereo depth and RGB imagery, the ZED 2 includes an integrated IMU, barometer and magnetometer that can be fused with visual data to provide orientation estimates at a rate up to 400 Hz. This estimate was compared against the Pixhawk's IMU measurements during testing. Another significant advantage of the ZED 2 is its factory calibration and built-in self-calibration capability, which allows rapid deployment with minimal setup effort.

On the software side, the ZED SDK provides APIs for integration into Python, ROS2, and other open-source environments. The SDK offers multiple settings for depth estimation, sensor fusion, and camera configuration, making the platform flexible but also requiring careful tuning to balance accuracy and computational performance. Moreover, more tools like object detection and mapping are available in the SDK (Figure 2.2).

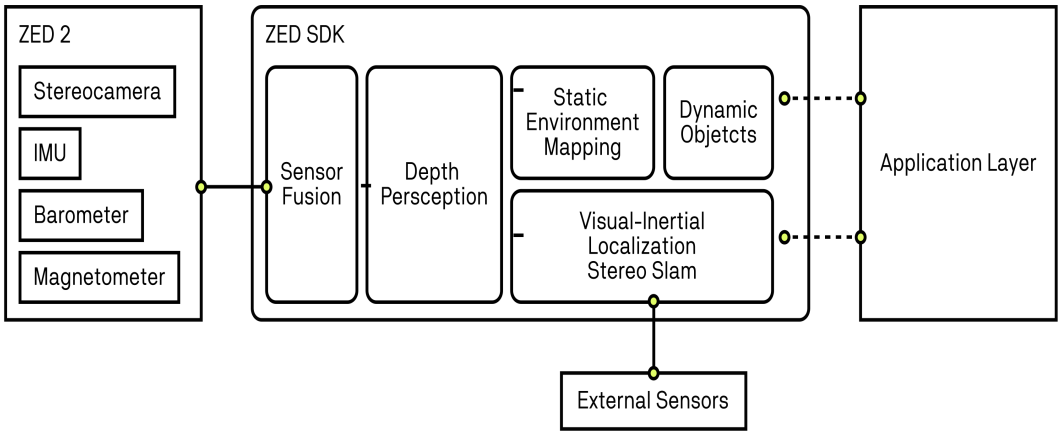


Figure 2.2: Functional SDK Diagram. Source: Stereolabs

2.1.4 RTK GPS

Global Navigation Satellite Systems (GNSS) are commonly used for UAV localization, but standard sensors typically provide accuracy on the order of several meters, which is insufficient for precise mapping of small-scale structures such as powerlines. To overcome this limitation, Real-Time Kinematic (RTK) positioning is employed. RTK improves accuracy by using the carrier wave phase measurements from the satellite signals combined with correction data provided by a nearby reference station. RTK GPS operates using a two-receiver system: a base station and a rover. The base station is placed at a known, fixed location and continuously receives signals from multiple GNSS satellites. By comparing its known coordinates with the incoming signal-derived coordinates, the base station computes correction data that account for errors caused by atmospheric delays, satellite clock inaccuracies, and other factors. This correction data is then transmitted to the rover, which is the

mobile receiver whose precise position is required. The rover applies the base station corrections in real time, effectively eliminating most of the common errors between the two receivers [34]. Since the base and rover are typically separated by only a few kilometers, the shared errors are highly correlated, allowing the rover to achieve centimeter-level positional precision relative to the fixed base station, provided that a stable correction link is maintained. In contrast, the absolute positioning accuracy depends on the quality of the base station's coordinates, which affect the computed error correction, and is typically on the order of a few meters. For mapping applications, relative precision is more important than absolute accuracy, as it ensures that the reconstructed point clouds and spatial measurements are consistent and reliable. A deeper explanation can be found in Section 3.1, that shows how the RTK pipeline for this thesis was done.

In this project, the onboard rover module is a Holybro F9P Helical GNSS receiver, which is based on the u-blox ZED-F9P chipset and capable of receiving multi-constellation signals (GPS, GLONASS, Galileo, BeiDou). The reference station is an Emlid Reach RS2, which transmits correction messages to the rover. The rover integrates these corrections with its own satellite observations to compute a precise, real-time position estimate. This configuration ensured that the UAV mock-up had access to globally referenced position data with sufficient precision for geo-referenced mapping.

While RTK requires additional infrastructure (a base station and a reliable telemetry link), it provides a deterministic positioning reference that can outperform typical passive odometry approaches such as visual-inertial odometry, particularly in outdoor inspection scenarios where absolute global coordinates are required.

(a) *Holybro F9P Helical* (b) *Emlid Reach RS2*

Figure 2.3: (a) Onboard GPS. (b) Fixed Base GPS

2.2 Software Stack

The perception framework developed in this thesis is built upon ROS 2, that provides a distributed architecture where individual nodes can exchange data via publish-subscribe topics, services, or actions, supporting both real-time and non-real-time workflows. Its modularity and strong community support make it an ideal choice for research platform that integrate multiple sensors, processing units, software, and requires a shared working environment.

The ZED 2 camera was easily integrated into the ROS 2 framework using the official ZED ROS 2 wrapper [35], which bridges ZED data to ROS topics. In contrast, PX4 Autopilot software required a bit of manual integration. In order to easily test

PX4-ROS integration, a simulation environment was set up using the open-source simulator Gazebo [36]. Finally, the mapping algorithms are built as ROS 2 nodes, using RGB-D, IMU and GNSS topics.

The Operating System chosen is Ubuntu 22.04 LTS, as it offers a stable integration of many software and hardware components. On the Jetson platform, it is installed via JetPack 6.1, NVIDIA's SDK that provides the Linux for Tegra (L4T) operating system, along with optimized libraries and tools for GPU-accelerated applications.

2.2.1 ROS 2 Humble

ROS 2 [37] represents the evolution of the original ROS framework (Section 1.3), introducing support for real-time communication and enhanced cross-platform compatibility. Moreover, it provides a wide set of tools for developing robotic applications. In this work, the Humble Hawksbill distribution of ROS 2, released in 2022, was used. Humble is compatible with Ubuntu 22.04 and is well supported in the current robotics ecosystem. It offers stable drivers and wrappers for a wide range of sensors, including the ZED 2 camera, and it is commonly used in state-of-the-art research involving autonomous aerial vehicles and real-time perception tasks. On top of the native ROS 2 libraries and tools, many other community-developed packages are available.

Architecture and Communication

ROS 2 retains the core concepts of ROS 1 such as nodes, messages, topics, services, and actions, but introduces a decentralized architecture for improved performance and real-time operation.

Nodes	Fundamental unit that perform computation. They encapsulate specific functionalities and communicate with each other via the ROS middleware.
Messages	Data structures used for communication between nodes. They define the format of exchanged information.
Topics	Named communication channels used for unidirectional, many-to-many message transfer. A publisher writes messages to a topic, while subscribers receive them asynchronously.
Services	Synchronous communication based on a request/response pattern, suitable for tasks requiring immediate feedback.
Actions	Asynchronous extension of services designed for long-running tasks. They provide continuous feedback and allow preemption.

Table 2.2: Main ROS 2 communication interfaces

Its communication layer is based on the Data Distribution Service (DDS) standard, which allows peer-to-peer message exchange without a central master node that handles everything (discovery and communication), as in ROS 1. This design improves scalability, robustness, and multi-robot support. For this project,

eProsima's Fast DDS was employed, enabling low-latency and high-throughput message delivery. In addition to Simple Discovery Protocol (peer-to-peer), Fast DDS also offers a Discovery Server mode, where all nodes register with a central server that handles discovery information (i.e., which nodes are publishing/subscribing to which topics), reducing the network overhead while still having peer-to-peer data communication.

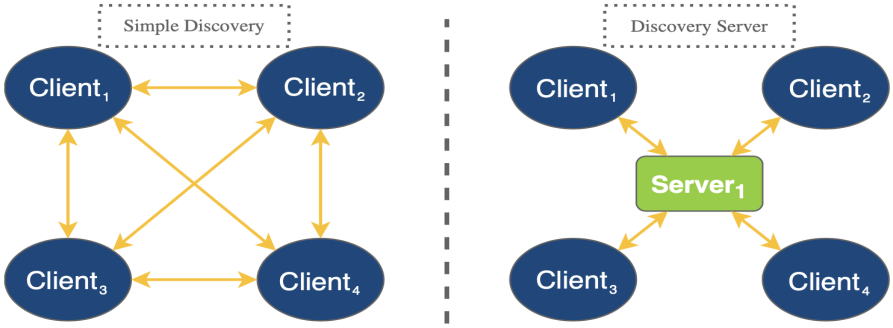


Figure 2.4: Comparison of Peer-to-Peer Discovery and Discovery Server in Fast DDS comparison. Source: eProsima Docs

DDS also provides fine-grained Quality of Service (QoS) controls, critical for transmitting high-frequency sensor data, such as stereo images, IMU measurements and RTK GPS readings. QoS settings can be defined per topic, allowing communication policies to be tuned depending on the type of data stream. For example, non-critical and high-bandwidth topics could be transmitted in `best-effort` mode (similar to UDP) to maximize throughput, while more critical messages could be sent in `reliable` mode (similar to TCP) to guarantee delivery. For this work, all topics were configured with a reliable policy to prioritize robustness and data consistency.

Development and Workspaces

ROS 2 organizes code into packages, which contain source files, message definitions, and metadata. Packages are built using the Colcon build system, which allows parallel compilation, dependency management, and workspace overlays. A workspace consists of the source (`src`), build (`build`), install (`install`), and log (`log`) directories, enabling modular development and easy integration of multiple packages. Colcon build system takes source files in `src` folder, compiles them in `build` and then saves the final output in `install`. This modularity is particularly useful for robotics applications such as a UAV perception framework, where sensor drivers, mapping nodes, and navigation modules can coexist without interfering with each other.

Advantages of the ROS 2 Ecosystem

ROS 2 provides a comprehensive ecosystem that simplifies the integration of heterogeneous components. To sum up, key advantages that facilitated this work include:

- **Modularity:** Nodes can be developed and tested independently, then easily combined into a full system.

- **Integration:** ROS 2 facilitates integration of a wide range of sensors, hardware drivers, and third-party libraries (e.g., ZED SDK, PX4).

- **Tooling support:** Utilities, parameter management, logging, visualization (`rviz2`, `rqt`, `Foxglove`), and offline data analysis enhance development, debugging, and testing.

This architecture and ecosystem enable the UAV perception framework to integrate sensors, controllers, and mapping algorithms in a robust and modular manner, while ensuring real-time data exchange across the system. A more detailed analysis of communication latency and real-time performance is provided in Section section 4.5.

2.2.2 PX4 Autopilot

PX4 is an open-source autopilot software stack widely used for UAVs, capable of providing full flight control, sensor integration, and communication with companion computers and ground stations. It runs on a variety of flight controllers, including the Pixhawk series, and supports multiple vehicle types such as multirotors, fixed-wing aircraft, and VTOLs. PX4 offers modular firmware architecture, extensive configuration options, and simulation tools that facilitate both research and development in UAV applications [38].

MAVLink Communication

MAVLink (Micro Air Vehicle Link) is a lightweight messaging protocol widely used in UAV ecosystems for telemetry, command/control, and parameter transfer usually between ground station and unmanned vehicles, and between onboard components. It was first released in 2009 by Lorenz Meier and has become popular in the robotic field [39]. The key features of the MAVLink protocol include [40]:

- **Efficiency:** Packets have 14 bytes of overhead and a maximum of 255 bytes of payload as shown in Table 2.3; it is designed to meet minimal bandwidth requirements.

- **Reliability:** Provides methods to detect packet drops, corruption, and support packet authentication.

- **Bidirectionality:** Supports a two-way communication, allowing the use of commands that requires responses or acknowledgments.

- **Compatibility:** Works independently of the network link, hardware components specifications and programming languages.

Field	Description
STX (1 byte)	Start-of-frame marker
LEN (1 byte)	Payload length
SEQ (1 byte)	Packet sequence number
SYS_ID (1 byte)	System ID of the sender
COMP_ID (1 byte)	Component ID of the sender
MSG_ID (3 bytes)	Message type identifier
PAYLOAD (0–255 bytes)	Actual message data
CRC (2 bytes)	Checksum for error detection

Table 2.3: MAVLink packet structure

SITL with Gazebo

To validate system integration and data workflows without actual flight, a Software-In-The-Loop (SITL) simulation was employed. SITL allows the PX4 firmware to run natively on a computer while simulating the behavior of the flight controller and UAV sensors. Harmonic is the version of Gazebo chosen since it offers the best compatibility with Ubuntu 22.04, ROS 2 Humble and PX4 v1.15. For this work, a custom Gazebo world was created, featuring a drone equipped with a depth camera in a powerline inspection scenario (Figure 2.5). The PX4 firmware offers default models and tools for SITL simulation, in particular the multicopter model `X500` with front-facing depth camera was adopted. The powerline model on the other hand, was taken from the work in [41] and integrated in a default gazebo world. To interact with the vehicle, the software QGroundControl (QGC) was employed, a MAVLink protocol ground station with proven PX4 integration and community support available. QGC, PX4 and Gazebo run on a Ubuntu 22.04 PC during the simulation. Even if the purpose was just to verify the data stream, flying the vehicle was also possible using a joystick or with a keyboard input, implemented based on the work in [42].

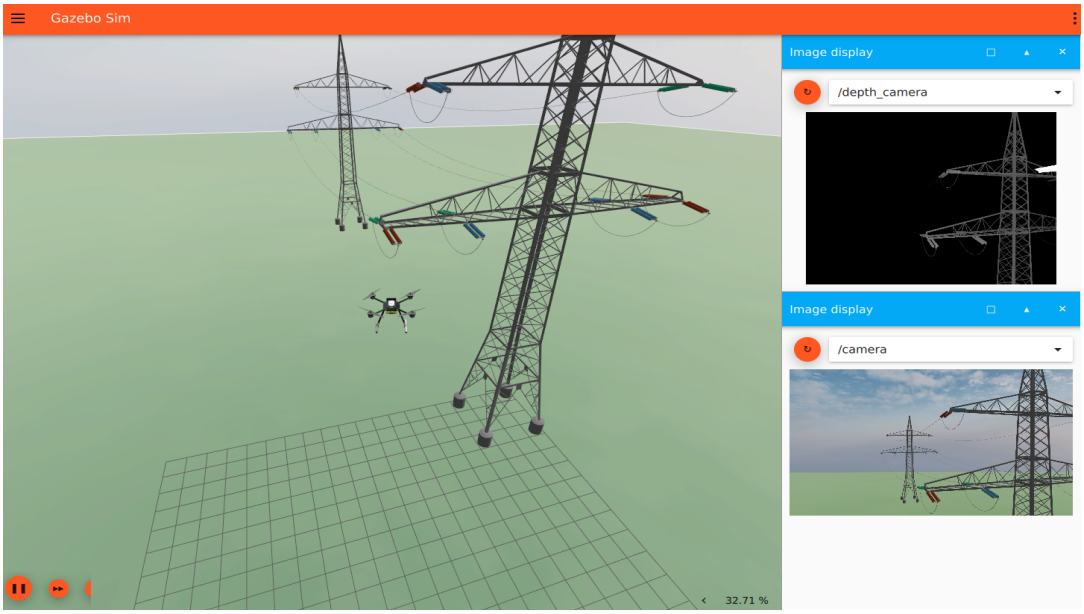


Figure 2.5: SITL simulation in a custom Gazebo world

The SITL helped understanding the data format and communication flow, allowing to make the proper firmware adjustment in a simulated environment. In particular, the XRCE topics `gps_inject_data`, necessary to provide RTK GNSS correction through ROS 2, was added to the list of subscription inside the `dds_topics.yaml` file in the firmware. Further explanation of the XRCE bridge between PX4 and ROS 2 can be found in 2.3.

QGC was then used also to install the modified PX4 firmware on the Pixhawk and to modify the necessary parameters to accommodate the specific hardware setup, including the integration of the Holybro F9P GPS module and companion computer. Serial port parameter were tuned to allow communication between hardware components.

Extended Kalman Filter (EKF) — State Estimation

PX4 firmware includes an Extended Kalman Filter to estimate the UAV's state by fusing multiple sensors, as well as to filter noisy measures. Some important features provided by the EKF are:

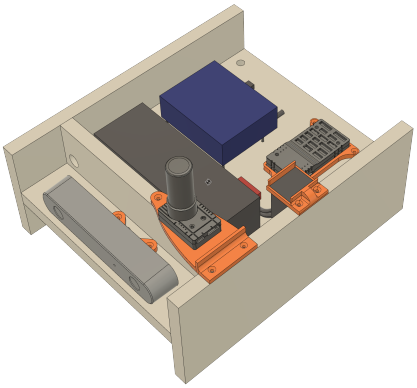
- **State estimation:** orientation (quaternion from world frame to body frame), velocity, position, IMU biases, magnetic field, wind velocity, terrain altitude, etc.
- **Synchronization:** Measurements from different sensors may have differing delays: EKF employs a *fusion time horizon* buffer system to align data properly over time before fusing.
- **Parameters:** Many EKF tuning parameters are available: e.g. IMU position offset relative to body frame, delay settings, covariance and process noise settings, etc. Note that proper tuning is critical for good performance, but was done only partially in this work.

2.3 Methodology

The objective of this work is to develop and evaluate a perception framework capable of providing geo-referenced 3D maps for UAV navigation in real time. Due to flight restrictions in the working area and to maintain flexibility during development, a drone mock-up was constructed. This rigid platform enabled realistic hardware integration and sensor testing without the risks and legislative challenges associated with outdoor flights. Simulation tools were also exploited, but their use was limited to validating the integration of the flight controller with the ROS 2 framework, as explained in the previous Section. Since the major challenges of this work were hardware-related and required real sensor data, simulations alone would not have been sufficient.

2.3.1 Drone Mock-Up

The drone mock-up structure was designed using the 3D CAD software Autodesk Fusion 360. The main platform was fabricated from 16 mm MDF (Medium Density Fiber) to ensure rigidity and low cost, while custom supports for the ZED camera, Pixhawk flight controller and GPS antenna were designed and 3D-printed. This setup provided stable mounting points and allowed for modular hardware reconfiguration during testing.



(a)

(b)

Figure 2.6: (a) Fusion 360 Model (b) Drone Mock-Up

Regarding hardware connections, the ZED 2 stereo camera connects directly to the Jetson Orin NX via its built-in USB 3.0 interface, ensuring high bandwidth for data streams. The Pixhawk flight controller and the Holybro F9P GPS are connected together via a standard telemetry serial cable through the GPS-1 port of the Pixhawk. Then, a custom-made serial cable was used to connect the Pixhawk TELEM-2 serial port to the Jetson GPIO pins (6: GND, 8: TX, 10: RX). The baud rate of this connection was configured to 921600 bps, which provides sufficient throughput for high-frequency telemetry while maintaining stable communication.

The RTK GPS setup was completed by establishing a link with the Emlid Reach RS2 base station. The RS2 transmits correction messages via USB to a ground computer at 115200 bps, which then forwards the corrections to the Jetson over a UDP socket. This ground computer also served as a control station for remote Jetson access via SSH (Secure Shell). During early development, the Jetson was equipped with a Wi-Fi module and antennas to enable wireless communication. However, for field tests where Wi-Fi infrastructure was weak or unavailable, a direct Ethernet connection was preferred, as it provided greater stability.

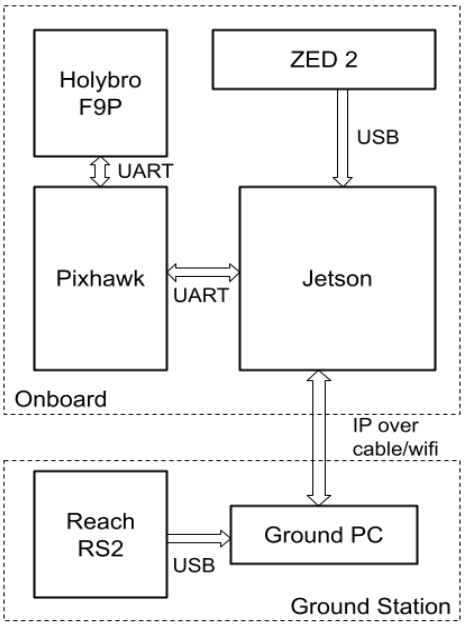


Figure 2.7: Hardware Communication Scheme.

2.3.2 Software Integration

With the hardware components interconnected, the next step was to integrate all data sources into the ROS 2 framework. As mentioned, Stereolabs provides a wrapper to bridge ZED data into ROS 2 environment, ensuring time synchronization with the host (Jetson) and ROS 2-formatted messages, services and actions.

PX4 on the other hand, required manual integration due to some inconsistency issue with the PX4-ROS 2 bridge. For this reason, initial development was performed in a SITL environment, allowing rapid tests after code implementation. However, this extra step required more integration since simulation topics from Gazebo needed to be bridged into the ROS 2 framework as well. For this purpose, *ros_gz* bridge [43] was installed. Then a further step to convert depth camera data to match ZED 2 format was necessary. To address this task, together with the PX4 topics issue, a custom bridge node was implemented. This node retrieved raw messages from PX4 and Gazebo topics and republished them in ROS 2-compliant format, while also ensuring consistency with hardware setups.

Two types of bridges were implemented:

- **SITL bridge:** adapted PX4 and Gazebo simulation topics to match ROS 2 message formats and naming conventions, as well as real RGB-D data format.
- **HITL bridge:** extended the same logic to hardware-in-the-loop experiments, additionally renaming ZED 2 wrapper topics for consistency.

This allowed to implement the perception framework regardless the use of SITL or HITL input data, and to have topics consistent with the ROS 2 environment:

- `/drone/camera` of type `sensor_msgs/Image`
- `/drone/rgb_camera_info` of type `sensor_msgs/CameraInfo`
- `/drone/depth_camera` of type `sensor_msgs/Image`
- `/drone/depth_camera_info` of type `sensor_msgs/CameraInfo`
- `/drone/imu` of type `sensor_msgs/Imu`
- `/drone/global_position` of type `sensor_msgs/NavSatFix`

PX4 ROS 2 XRCE Agent

In order to integrate PX4 with ROS 2, the ROS 2 XRCE (eXtremely Resource Constrained Environment) agent was chosen. The XRCE agent acts as a bridge between the flight controller firmware and the ROS 2 environment using the eXtremely Resource Constrained Environment DDS (Micro XRCE-DDS) protocol, enabling lightweight and efficient transmission of MAVLink messages from PX4 to ROS 2 nodes. This approach allows high-frequency telemetry, sensor data, and vehicle state information to be available within the ROS 2 framework while minimizing computational overhead on the flight controller. However, XRCE agent has two limitations about its topic being not fully compliant with ROS 2 conventions. In particular:

- It provide custom message types (e.g. `VehicleAttitude`, `SensorCombined`, etc.), which are not directly compatible with state-of-the-art ROS 2 tools.
- Messages lack a standard ROS 2 header (timestamp and frame ID).

To address this, the bridge nodes appended the missing headers and republished the data as standard ROS 2 messages.

IMU Message

The ROS 2 IMU message (`sensor_msgs/Imu`) was constructed by combining information from multiple PX4 topics:

- `VehicleAttitude`: provided orientation as a quaternion (q_w, q_x, q_y, q_z) representing the rotation from the vehicle body frame **FRD** (Forward-Right-Down) to the **NED** (North–East–Down) world frame.
- `SensorCombined`: provided raw gyroscope measurements in rad/s and accelerometer measurements in m/s^2, expressed in the **FRD** body frame.

ROS, on the other hand, adopts a different convention:

- The world frame is **ENU** (East–North–Up).
- The body frame is **FLU** (Forward–Left–Up).

Therefore, a frame transformation is required before publishing IMU data in ROS 2. Specifically:

1. The **body frame** conversion FRD $\rightarrow$ FLU corresponds to a positive rotation about X axis of π according to the Right-Hand Rule (RHR).

2. The **world frame** conversion NED → ENU corresponds to a positive rotation about Z axis of $\pi/2$.

The PX4 quaternion q_{PX4} encodes the rotation from the FRD body frame to the NED world frame. To express this orientation in the ROS convention (FLU body frame to ENU world frame), two steps are required:

1. **Body-frame conversion (FRD → FLU):** Converting the body frame axes from FRD to FLU is a change of basis. In quaternion form, such a change is expressed by conjugation:

$$q' \;=\; r_x \otimes q_{\text{PX4}} \otimes r_x^{-1},$$

where r_x is the quaternion corresponding to a 180° rotation about the x-axis. Since $r_x = [0,\, 1,\, 0,\, 0]$ is its own inverse (up to sign), this simplifies to

$$q' \;=\; r_x \otimes q_{\text{PX4}} \otimes r_x.$$

2. **World-frame conversion (NED → ENU):** Aligning the NED world frame with ENU requires a fixed 90° rotation about the z-axis, represented by the quaternion

$$r_z = \left[\tfrac{\sqrt{2}}{2},\, 0,\, 0,\, \tfrac{\sqrt{2}}{2}\right].$$

This is applied by left multiplication:

$$q_{\text{ros}} \;=\; r_z \otimes q'.$$

Thus, the final quaternion expressed in the ROS convention is

$$\boxed{q_{\text{ROS}} \;=\; r_z \otimes \big(r_x \otimes q_{\text{PX4}} \otimes r_x\big)}.$$

Once this transformation is applied to q_{PX4} to express orientation, the same was done to angular velocity and linear acceleration, but in matrix form:

$$\mathbf{v}_{\text{ROS}} = R\,\mathbf{v}_{\text{PX4}}, \qquad \mathbf{a}_{\text{ROS}} = R\,\mathbf{a}_{\text{PX4}}$$

where:

$$R \;=\; R_z\!\left(\tfrac{\pi}{2}\right) R_x(\pi) = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}.$$

Finally the IMU information were published as a standard ROS2 `sensor_msgs/Imu` message:

```
std_msgs/Header header
geometry_msgs/Quaternion orientation
geometry_msgs/Vector3 angular_velocity
geometry_msgs/Vector3 linear_acceleration
```

Field `header.stamp` is filled with the timestamp of the original PX4 message and `frame_id` is set according to the ZED 2 camera's frame tree, ensuring consistency across the system.

GNSS Message

PX4's global position output was similarly adapted. The raw[1] GNSS data `VehicleGlobalPosition` was mapped into a standard `sensor_msgs/NavSatFix` message. This step required only to fill latitude, longitude, altitude and header fields with PX4 message data, without additional computation.

RGB-D Messages

Depth camera messages were already published in ROS 2 format, both from ZED 2 wrapper topics and from Gazebo simulation topics. Hence, the HITL bridge node simply change the topics name, while the SITL bridge node also convert the camera format to match the ZED. Although this step was not necessary even because some ZED 2 settings like resolution was tuned afterwards, it ensures that the data encoding was the same.

Real-time Considerations

Clearly, the bridge node introduced an additional processing step, that could potentially increase the message delay. However, during testing, latency was measured to be negligible compared to the intrinsic delays of sensor acquisition and data transfer. It will be shown that with this framework an overall delay of approximately 500 ms can be expected. The SITL bridge node introduce a latency of less than 5 ms (Figure 2.8). The HITL bridge node is slightly slower, but a more detailed analysis of real-time performance is presented in Section 4.5.

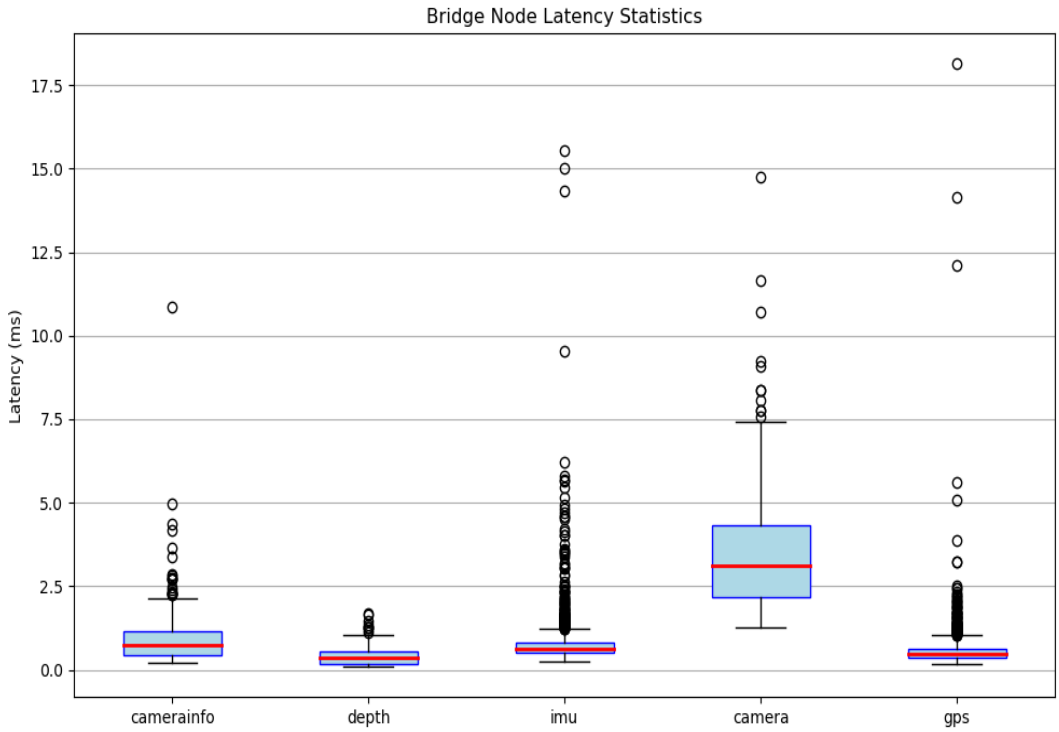


Figure 2.8: SITL Bridge Node Topics Latency

[1]Raw in this context means the PX4 topic, which is not actually raw but filter with EKF as explained previously.

Modular ROS 2 Workspaces

To maintain modularity and avoid package conflicts, the perception algorithms were implemented across multiple overlay workspaces. This structure leveraged the modular design of ROS 2 and reduced the risk of accidental modifications to upstream packages. It also allowed independent development and testing of different perception modules before integration into the overall framework.

Evaluation Tools

For evaluation and debugging, several visualization and logging tools were employed. ROS native tools such as `rosbag`, `rqt`, `plotjuggler`, `rviz2`, and `Foxglove Studio` were used at different stages of development. However, due to the heavy computational load of tools like `rviz2`, the primary approach was to log data as CSV files directly from ROS nodes. These datasets were then analyzed offline using custom Python scripts, which allowed detailed inspection without overloading the Jetson at runtime.

2.4 Mapping Framework

The proposed framework for mapping using RTK-GNSS and stereo camera is illustrated in Figure 2.9.

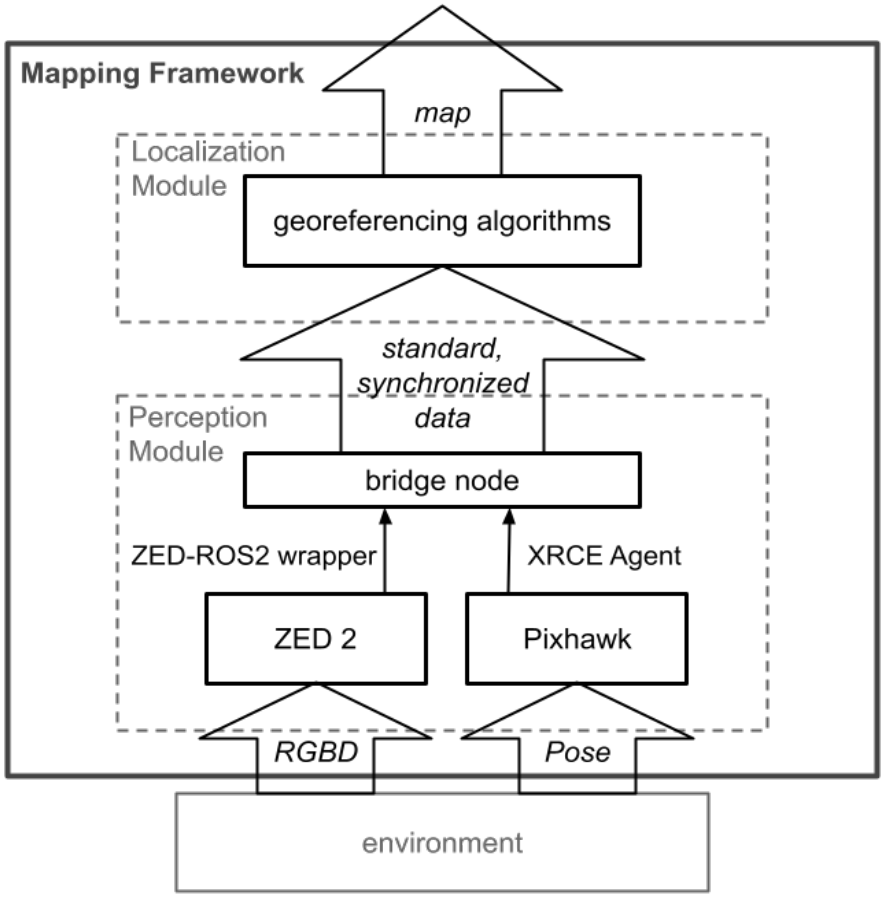


Figure 2.9: Stereo vision and RTK GNSS-based mapping framework

Depth data are sensed by the stereo camera and integrated into ROS 2 through the `zed-ros2-wrapper`. RGB images are also retrieved, although they are not strictly required for mapping in this work, their availability could support future extensions such as object detection modules. Using Pixhawk flight controller and GNSS system provided filtered pose data (position and orientation) straight-forward through PX4 and the XRCE Agent. This is a considerable advantage, as it reduces the localization problem to estimating the position of the environment.

The perception module retrieve data from ZED and PX4 and re-elaborate them in a bridge node to align and make them compliant to ROS 2 standard. This additional step also increased the modularity of the development, allowing to implement and test different software components independently.

The localization module then computes the position of points in the environment. Using the intrinsic parameters of the camera, each depth pixel is projected into 3D coordinates in the camera frame. Orientation from the IMU is used to transform these coordinates into the drone body frame, and finally the GNSS position allows them to be georeferenced in a global frame. The result is a point cloud map referenced in world coordinates. However, directly georeferencing every pixel is unfeasible, both for computational and practical reasons. The raw point cloud would be heavy and noisy: filtering techniques must be applied. In particular, the maps presented in this thesis are the result of the work by a colleague [5], which applies down-sampling and CUDA-accelerated processing to achieve real-time feasibility.

The contribution of this thesis is the design and implementation of the acquisition and georeferencing pipeline, demonstrating the feasibility of constructing a global map suitable for UAV mission planning.

Chapter 3

Development

3.1 RTK GNSS Pipeline

As mentioned, GNSS signals are affected by different sources of error (e.g. atmospheric effects) that need to be mitigated in order to obtain a precise and reliable signal that can be used for more advanced applications such as UAV mapping. A practical solution is to apply differential correction by indirectly measuring the error with another GNSS receiver at a known location and subtracting it from the GNSS signal of interest. To accomplish this, a base GNSS station at a known position retrieves its estimated position from the GNSS signal and compares it against the ground-truth position to estimate the error.

$$P_{\text{BASE GNSS}} = P_{\text{BASE TRUE}} + err$$

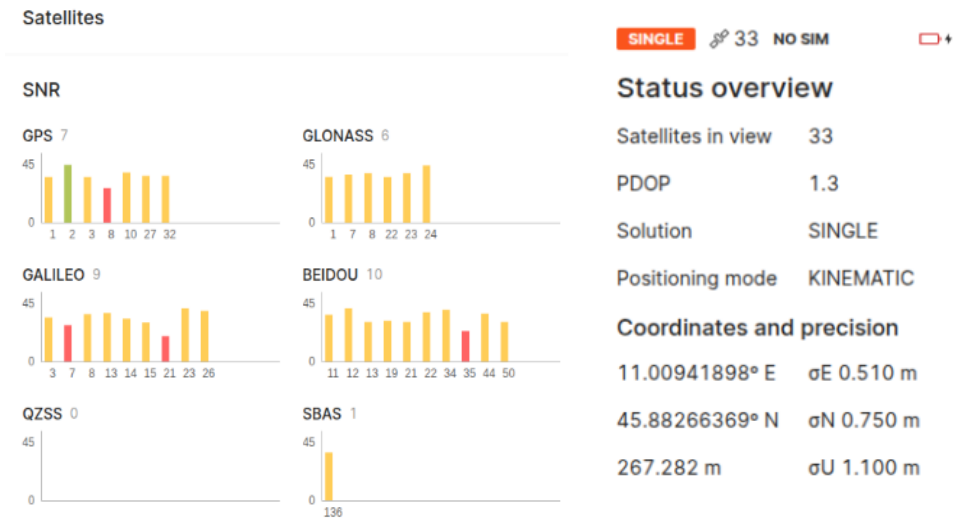
$$P_{\text{BASE GNSS}} - P_{\text{BASE TRUE}} = P_{\text{BASE TRUE}} + err - P_{\text{BASE TRUE}} = err$$

Since the base and the onboard GNSS receiver are placed within few a kilometers, atmospheric noise and satellites in view are similar, so err will be similar as well. The base station then sends the estimated error to the UAV, which can subtract it from its GNSS readings to obtain a corrected position estimate.

$$P_{\text{UAV GNSS}} - err = P_{\text{UAV TRUE}} + err - err = P_{\text{UAV TRUE}}$$

Although the resulting precision is within centimeters, global accuracy can still be on the order meters, depending on the accuracy of the known position of the base GNSS receiver. The computed err in fact, includes a static term due to inaccuracy of the ground-truth position, which leads to a global offset error. For the purpose of this work, precision is key to obtaining good mapping results, while global accuracy does not affect the quality of the map, but only the absolute global location of all points.

The base GNSS position is not always at a known and reliable location, so it must be estimated as well. In this work, the *average single* positioning method was used. This is a simple but effective method that averages many single-point readings of the base GNSS signal over a period of time. During tests, the averaging time was set to 15 minutes, and the typical accuracy error was approximately within 1 meter (Figure 3.1). It was observed that the main factor in reducing the error was not increasing the averaging time, but placing the base station in an open area with clear sky visibility, which was not always possible. Nevertheless, for the purposes of this work, this level of accuracy was considered sufficiently small.



(a) *Satellites in view* (b) *Status overview*

Figure 3.1: Control panel for the Reach base station

RTCM Message Structure

RTCM is a communication protocol used to transmit differential GNSS correction messages since the 1980s, but it became widely used in the 2000s with the RTCM-Version 3 format that introduced support for multiple satellite constellations, variable-length messages and increased number of possible message types. RTCM messages consist of a header, payload, and checksum, and are usually transmitted as a continuous stream over serial or network connections [44]. A typical RTCM message has the following structure:

- **Preamble (8 bits):** Start-of-message marker, usually `0xD3`.
- **Message Length (10 bits):** Indicates the number of bytes in the payload.
- **Payload (variable length):** Encodes the GNSS correction data, satellite-specific information, or other system parameters. Multiple message types are supported, such as:
  - Type 1005/1006: Reference station coordinates
  - Type 1077/1087: GPS/GLONASS RTK corrections
- **Checksum (24 bits):** CRC (Cyclic Redundancy Check) for detecting transmission errors.

RTCM messages are compact and efficient, minimizing bandwidth overhead. Although is not the only possibility to transmit GNSS data, these characteristics make them well-suited for differential correction data.

3.1.1 Implementation

To implement the RTK GNSS pipeline, a communication link between the base and onboard GNSS must be established. As mentioned, the Emlid Reach RS2 is

connected via USB to the ground computer, where a Python script reads raw data from the serial port and forwards it via a UDP (User Datagram Protocol) socket to the IP address of the Jetson (typically via Ethernet during tests). UDP, in contrast to TCP (Transmission Control Protocol), offers a faster data stream at the cost of potential packet loss. In this setup UDP was chosen to reduce the latency typical of TCP.

On the Jetson, the ROS 2 node `rtcm_forwarder` opens the UDP socket on the specified port and reads a continuous stream of RTCM (Radio Technical Commission for Maritime Services) correction data. The `pyrtcm` Python library [45] was used to parse this stream, extracting individual RTCM messages. These messages populate the PX4–ROS 2 message `GpsInjectData`, which is then published on the `gps_inject_data` topic and bridged to PX4 through the XRCE agent.

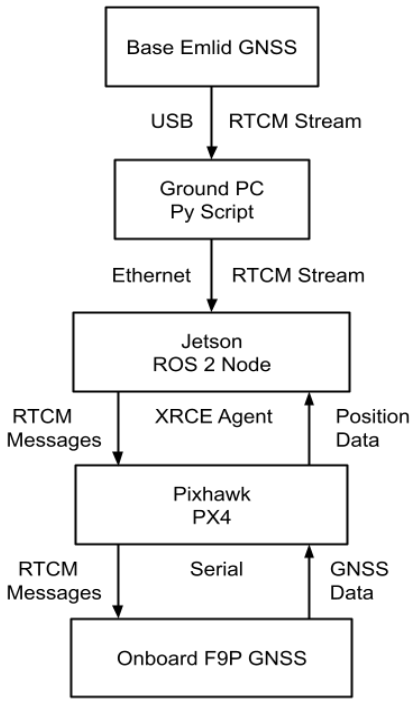


Figure 3.2: RTK GNSS Pipeline: communication interface and data streaming

Thanks to the update of the PX4 firmware, the flight controller now listens to this topic and forwards the correction messages to the F9P GNSS module. The onboard GNSS, configured in QGC to apply the corrections, returns high-precision GNSS data to the flight controller. The PX4 EKF filters these data and publishes precise position estimates, which are bridged into ROS 2 via the XRCE agent.

RTCM Forwarder ROS 2 Node

The RTCM forwarder node `rtcm_forwarder` is responsible for receiving RTCM correction messages from the base station via UDP and injecting them into PX4 through ROS 2 messages. The main steps of the node are:

1. **Initialization:** The node creates a ROS 2 publisher for `GpsInjectData` mes-

sages and a non-blocking UDP socket bound to a specific port.

2. **Receive UDP data:** Incoming data are appended to a circular buffer for processing.

```
while True:
    data, _ = sock.recvfrom(4096)
    buffer.extend(data)
```

3. **Parse RTCM Messages:** The buffer is wrapped in a `BytesIO` stream and fed to `pyrtcm.RTCMReader` to extract complete RTCM messages.

```
stream = io.BytesIO(buffer)
reader = RTCMReader(stream)
for raw_data, parsed in reader:
    send_chunk(raw_data)
```

4. **Publish to PX4:** Each RTCM message is packed into a `GpsInjectData` message and published to the ROS 2 topic. Messages larger than the maximum allowed length are dropped, while shorter messages are padded with zeros to fill the fixed-size array.

```
if len(raw) > MAX_LEN:
    return  # drop message too large
msg = GpsInjectData()
msg.data = list(raw) + [0] * (MAX_LEN - len(raw))  # pad if necessary
pub.publish(msg)
```

5. **Buffer Maintenance and Logging:** Consumed bytes are removed from the buffer, and statistics on packets received and messages sent are logged periodically. In Listing 3.1, the logs (every 60 seconds) show that packets are received at about 1 Hz, with each packet containing about 50 messages (i.e., 300 messages in 60 packets).

Listing 3.1: Sample terminal logs during the execution of the node

```
[INFO][rtcm_forwarder_node]: Packets recv: 180, Messages sent: 918
[INFO][rtcm_forwarder_node]: Packets recv: 240, Messages sent: 1224
[INFO][rtcm_forwarder_node]: Packets recv: 300, Messages sent: 1530
```

3.1.2 Limitations of the Proposed Pipeline

Although the proposed pipeline provided effective correction and allowed obtaining precise measurements, some limitations must be highlighted. It was known that the communication link was critical, but maintaining its stability and checking its integrity proved more challenging than expected.

Even though all communications were wired, the USB cable between the base Reach GNSS and the ground computer was slightly loose. If the computer was moved roughly, the communication was easily lost. It was noted that if the correction link was broken, as in such cases, the onboard GNSS simply fell back to standard

positioning without any re-alignment or fix. The difference in coordinates could be on the order of meters, resulting in a sudden jump in position. However, the PX4 EKF smoothed this jump, which then appeared as a gradual offset, as shown in Figure 3.3.

These breaks in the pipeline were initially noticed only in the final map results, where the offset became evident. To mitigate this, some checks were added in the code to monitor the stability of the link between the base GNSS and the ground computer. On the Jetson side, however, the `pyrtcm` library does not provide built-in features for error detection, highlighting the need for a future implementation based on a different library or additional methods. Additionally, some PX4 topics also provide the status of the GNSS, allowing detection of whether corrections are applied and handling of such cases is needed. However, in this thesis none of these features were implemented, since the main issue was related to the USB cable of the base Reach receiver. Once it became clear when this link was interrupted, tests were simply stopped and repeated. For future implementations, however, it is strongly recommended to add error detection and handling mechanisms, especially if the correction link is not wired.

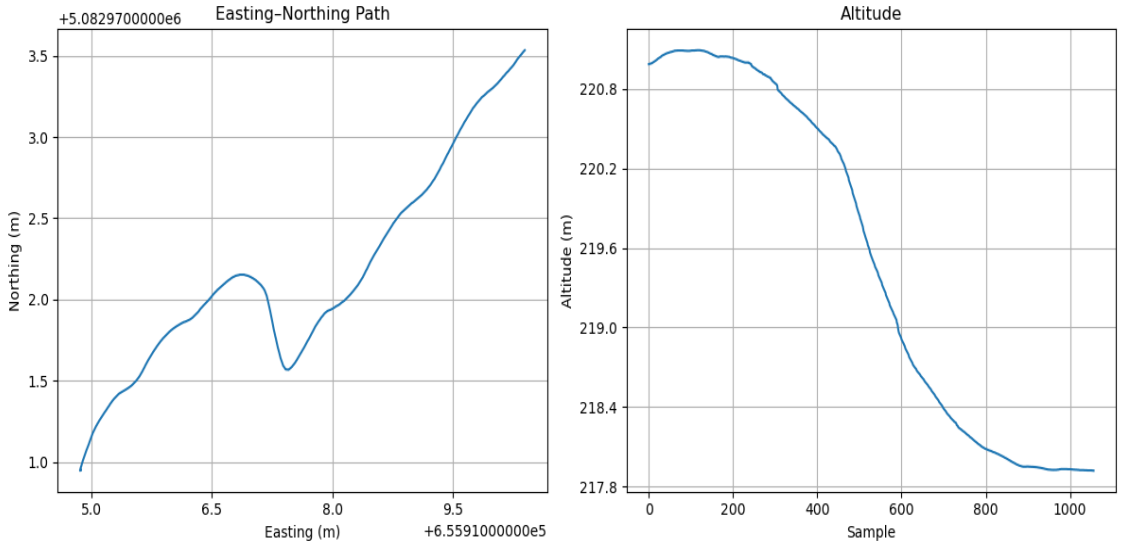


Figure 3.3: Effect of correction loss on the estimated position.

3.2 Georeferencing Pipeline

In this section, the implementation of the georeferencing pipeline to perform mapping in world coordinates is explained. The pipeline can be considered a deterministic sensor fusion algorithm: it transforms pixel coordinates from the depth camera into world coordinates, combining information from the camera, IMU, and GNSS. As with any sensor fusion approach, synchronization of data streams is critical, since misaligned timestamps can propagate errors into the 3D reconstruction. In this work, the Jetson was used as the reference system time, ensuring that the ZED camera and the flight controller shared a common temporal base.

In order to implement the full transformation pipeline, the following data are needed:

- **Camera Intrinsics:** Focal lengths and principal points, obtained from the `CameraInfo` topic, are necessary for projecting pixel coordinates into 3D camera coordinates.
- **Depth:** Depth maps from the stereo camera are published as `Image` messages.
- **IMU:** Attitude data (quaternions) are provided on the `Imu` topic.
- **Position:** The drone's position in global coordinates is provided on a topic of type `NavSatFix`.

3.2.1 Camera Model

A depth camera can be modeled using the **pinhole camera model**, which relates 3D points in the camera frame to 2D points on the image plane. Mathematically, a point (X, Y, Z) in the camera frame projects to pixel coordinates (x, y) as:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X/Z \\ Y/Z \\ 1 \end{bmatrix}$$

The matrix, usually referred to as K, contains the intrinsic parameters:

- **Focal lengths:** f_x, f_y are the effective distances between the camera lens and the projection plane, also called the image plane. They are expressed as two parameters since cameras often have non-square pixels, so the effective focal length along the horizontal and vertical axes can differ.
- **Principal points:** c_x, c_y are the coordinates of the intersection point between the optical axis and the image plane. They represent the origin of the pixel coordinate frame. Figure 3.4 provides a visual representation.

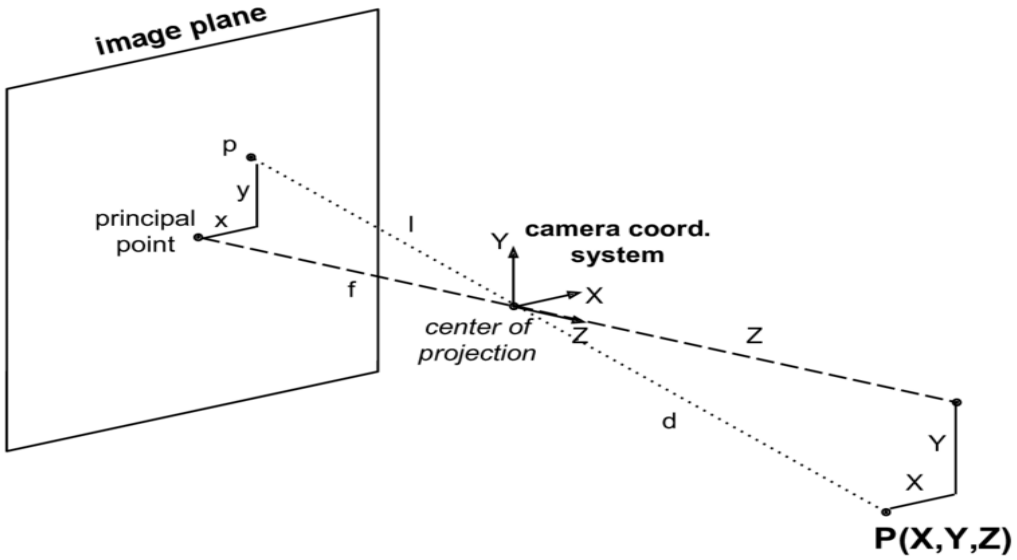


Figure 3.4: Pinhole camera model. Figure by [46].

In this work, the intrinsic parameters are published by the camera on the topic `CameraInfo` in ROS 2 and captured once during initialization. They depend on the camera settings and resolution.

3.2.2 Message Synchronization

Synchronizing sensor measurements is critical in a georeferencing pipeline. In ROS 2, sensors publish data asynchronously at different rates, and small timing mismatches can produce incorrect 3D points. To address this, the `message_filters` package is used. By subscribing to multiple topics and defining a synchronization policy, a single callback function is called only when messages from all sensors are aligned in time (within a small tolerance). This ensures that the depth, IMU, and GNSS measurements correspond to the same instant.

`message_filters` initialization in a ROS 2 node looks like:

```
sub_depth = Subscriber(self, Image, '/drone/depth_camera')
sub_imu   = Subscriber(self, Imu,   '/drone/imu')
sub_gps   = Subscriber(self, NavSatFix, '/drone/global_position')

self.sync = ApproximateTimeSynchronizer(
    [sub_depth, sub_imu, sub_gps], queue_size=20, slop=0.06
)
self.sync.registerCallback(self._sync_cb)
```

The `slop` parameter defines the maximum allowed time difference between messages (set to 0.06 seconds), and the `queue_size` handles different topic rates and possible transmission delays.

Actual Timestamp

It is important to clarify what the ROS 2 timestamps represent. The timestamp associated with each topic, in fact, does not necessarily match the exact moment of sensor acquisition. For the ZED 2 camera, this represents the time at which data enter the USB buffer, and it is closest to the acquisition time, according to Stereolabs developers. For the PX4 flight controller, the timestamp in ROS 2 topics is derived from the system time of the autopilot when the message is generated. This usually introduces a small delay relative to the actual sensor measurement, as IMU and GNSS readings are collected at high frequency and then packaged by PX4 for transmission.

3.2.3 Pixel-to-World Transformation

The core of the georeferencing pipeline is the transformation of coordinates that starts from pixels and ends up in global coordinates. It is a step-by-step process that could be merged into a single transformation matrix. This involves three steps:

1. back-projecting a pixel into 3D camera coordinates using the inverse pinhole camera model;

2. transforming the point from the camera frame into the UAV body frame;

3. transforming the point into the world frame using the UAV pose (position and orientation).

UTM Coordinate System

To obtain readable results, the Universal Transverse Mercator (UTM) coordinate system is used [47]. Latitude and longitude are expressed in degrees and are not suited to be represented as 3D points. UTM projects the coordinates onto a grid-based coordinate system: the Earth is divided into 60 vertical zones, each spanning 6° of longitude, and 20 horizontal bands, each spanning 8° of latitude, for a total of 1200 cells. Within these cells, the Earth can be approximated as flat, with a small margin of error. The distortion, in fact, grows as the longitude moves away from the central meridian, due to the Transverse Mercator projection model. However, for mapping purposes it is safe to treat each cell as flat, allowing us to obtain an (X_{world}, Y_{world}) coordinate system in meters.

Pixel-to-Camera Transformation

The depth map on the `Image` topic contains the depth value Z (meters) associated with each pixel (x, y). Inverting the pinhole camera model, it is possible to back-project a pixel on the image plane to the corresponding point in the camera reference frame:

$$\begin{aligned} X_c &= (x - c_x)\frac{Z}{f_x} \\ Y_c &= (y - c_y)\frac{Z}{f_y} \\ Z_c &= Z \end{aligned}$$

where (X_c, Y_c, Z_c) are the coordinates in the camera reference frame, in meters. In matrix form, this is equivalent to a homogeneous transformation:

$$P_c = T_{K^{-1}} P_{pixel}$$

$$T_{K^{-1}} = \begin{bmatrix} 1/f_x & 0 & -c_x/f_x & 0 \\ 0 & 1/f_y & -c_y/f_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

with P_c and P_{pixel} being the homogeneous coordinates in the camera and body frames, respectively.

Camera-to-Body Transformation

The camera reference frame is RDF (Right-Down-Forward) and, for the ZED 2, is placed on the left lens. The drone body frame is defined as FLU (Forward-Left-Up) to be consistent with the ROS 2 convention, and thus aligned with all topics. The origin of the body frame is placed at the F9P GNSS module, since the global position is relative to it. This introduces only a small offset, as the camera and GNSS are a few centimeters apart. This offset would be irrelevant for mapping, because a constant shift in the global position of all points does not affect the relative structure. Nevertheless, it is considered here for the sake of completeness.

The rotation between RDF and FLU is a rotation of π about the X axis, as explained in Section 2.3.The translational offset of the camera relative to the F9P is estimated to be 3 cm forward, 1 cm to the right, and 1 cm downward. This corresponds to the homogeneous transformation matrix T_{cb}, which combines the rotation R_{cb} and translation t_{cb}:

$$\mathbf{P}_b = T_{cb}\,\mathbf{P}_c$$

$$T_{cb} = \begin{bmatrix} 1 & 0 & 0 & 0.03 \\ 0 & -1 & 0 & -0.01 \\ 0 & 0 & -1 & 0.01 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where P_b are homogeneous coordinates in the body frame.

Body-to-World Transformation

Given a point in the body frame, it needs to be rotated and translated according to the pose of the drone in global coordinates. To account for rotation, IMU data are used. Aligning with the ROS 2 convention allows the rotation matrix from FLU to ENU (world frame) to be safely computed. As for translation, the vector of UTM coordinates (Easting, Northing), combined with altitude, is used. The code snippet that define the transformation matrix T_{bw} using the quaternion from `Imu` and coordinates from `NavSatFix` is shown below:

```
# Orientation (quaternion → rotation matrix)
q = imu.orientation
R_bw = R.from_quat([q.x, q.y, q.z, q.w]).as_matrix()

# Position in UTM
e, n, _, _ = utm.from_latlon(gps.latitude, gps.longitude)
t = np.array([e, n, gps.altitude])

# Homogeneous transform
T_bw = np.eye(4)
T_bw[:3,:3] = R_bw
T_bw[:3, 3] = t
```

The last step of this pipeline to obtain the 3D point P_w corresponding to a single pixel in world coordinates is:

$$\mathbf{P}_w = T_{bw}\,\mathbf{P}_b,$$

Once the intrinsic parameters of the camera are initialized, the first two transformation are fixed and can be combined into a single homogeneous matrix from pixel to body:

$$T_{pb} = \begin{bmatrix} 1/f_x & 0 & -c_x/f_x & 0.03 \\ 0 & -1/f_y & c_y/f_y & -0.01 \\ 0 & 0 & -1 & 0.01 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

allowing the body-frame coordinates to be computed in a single step:

$$\begin{bmatrix} X_b \\ Y_b \\ Z_b \\ 1 \end{bmatrix} = T_{pb} \begin{bmatrix} x \\ y \\ Z \\ 1 \end{bmatrix}$$

The final body-to-world transformation is better left separate, since this matrix is dynamic and changes whenever new data arrive through `message_filters`. The complete transformation chain can be expressed compactly as:

$$\mathbf{P}_w = T_{bw}\, T_{pb}\, \mathbf{P}_{pixel}$$

This compact formulation highlights the deterministic structure of the georeferencing pipeline: once the static transformations ($T_{K^{-1}}$, T_{cb}) are calibrated, only T_{bw} varies over time, driven by the UAV's global pose.

3.2.4 Point Cloud Mapping

While the georeferencing pipeline allows to project each pixel into a 3D georeferenced point, it is infeasible to store and publish a full dense point cloud. A 1920×1080 HD image already contains over 2 million pixels, and after projection each frame would result in millions of points. For real-time mapping purposes, this translates in:

- Large memory consumption;
- High CPU/GPU load;
- Large network bandwidth;
- High point density with significant redundancy and noise.

To overcome this, a filtering stage is introduced after the georeferencing pipeline, but before publishing and accumulating the point cloud. This allows defining spatial filtering criteria directly in world coordinates. Moreover, in order to achieve real-time performance, the use of GPU power is mandatory. The mapping algorithm must be written with a mid/low level programming language with GPU acceleration. As mentioned, the algorithm used in this work is written by a colleague at Fraunhofer Italia [5].

In this way real-time capabilities were achieved, as well as good quality maps.

Filtering Techniques

Two filtering techniques are applied sequentially in two subsequent steps:

1. **Spatial Hash Voxel Grid Filter**
   - Reduces point density by discretizing space into voxels (3 dimensional boxes) of fixed size, and assigning each point a hash key to uniquely identify the voxel they are in. Only the first point that claims the voxel is kept; other point within the voxel are discarded.

- This is a trade-off between accuracy and performance: it is very efficient due to the hash check, but the resulting point distribution is not perfectly uniform, as the first point in each voxel is kept, regardless of how central it is.

2. **Radial Outlier Removal**

   - Removes isolated points with too few neighbors within a given radius.
   - Enhanced performance by discretizing the space in voxels. Each point in a voxel is compared only against points in the neighboring voxels, reducing the amount of comparison.

Algorithm 1 Point Cloud Generation and Filtering Pipeline

Require: Depth, IMU, GNSS, camera intrinsics K, camera extrinsics T_{cb}

1: $P_{cam} \leftarrow T_{K^{-1}} \cdot P_{pixel}$ ▷ Backproject pixels
2: $P_{body} \leftarrow T_{cb} \cdot P_{cam}$ ▷ Camera-to-body
3: $P_{world} \leftarrow T_{bw} \cdot P_{body}$ ▷ Body-to-world
4: $\mathcal{P} \leftarrow \{P_{world}\}$ ▷ Set of georeferenced points
5: $\mathcal{P}' \leftarrow$ `Voxel Grid Downsampling`$(\mathcal{P}, \Delta)$ ▷ Δ defines the voxels' dimension
6: $\mathcal{P}'' \leftarrow$ `RemoveOutliers`$(\mathcal{P}', r, k)$ ▷ r, k define the radius of search and minimum number of neighbors, respectively
7: Publish $\mathcal{P}''$ to ROS 2 topic and append to global map

Parameters of both filter were tuned using real data recorded with the drone mock-up.

Real-Time Considerations

To achieve real-time processing onboard the UAV, the computational power of the Jetson is leveraged. The algorithm was implemented in C++ with CUDA acceleration:

- **C++**: Enables low-level memory management, efficient copying of large point arrays, as well as integration with ROS 2.
- **CUDA**: Offloads voxel filtering, outlier removal, and projection computations to the GPU.

An evaluation of the performance is given in Section 4.5.

3.3 Testing Setup

Various tests were conducted to evaluate the framework at different stages of development. Initially, SITL and HITL tests were used to verify the integration of PX4 with ROS 2, ensuring that the correct data streams entered the framework.

Tests for single sensors were also performed to evaluate the quality of the individual input data for the mapping pipeline. These tests involved simpler setups and will be discussed in detail in Section 4.1.

The georeferencing pipeline was first evaluated using an ArUco marker to assess the precision of repeated measurements of a single point. Then, the full mapping algorithm was tested, starting the tuning process of the ZED camera parameters. Both

the ArUco experiments and initial mapping tests were conducted while recording `rosbag` data to allow offline modifications to the algorithm and detailed inspection of raw measurements. Finally, the complete framework was tested onboard, logging all relevant data to evaluate both the quality of the generated maps and real-time performance.

Figure 3.5: Setup during a field test on transmission tower

General Mapping Test Procedure

The mapping algorithm was tested first in a controlled environment, in a yard near the company facilities in Bolzano, to verify correct functionality. Later, tests were conducted in the vicinity of transmission towers to collect data relevant to the purpose of this work.

The ground computer was connected via USB to the base GNSS station and via Ethernet to the Jetson companion computer onboard the drone mock-up. The procedure can be summarized in four steps:

1. **Base GNSS:** Connect the Reach RS2 to the ground computer to automatically start the averaging procedure. Once completed, the Reach begins sending RTK correction through the USB port.
2. **Companion Computer:** Power on the Jetson and access it from the ground computer over SSH via Ethernet.
3. **Pipeline:** On the ground computer, launch the script to forward correction to the Ethernet link. On the Jetson terminal, launch a script that starts a `tmux` session with the necessary processes: `rtcm_forwarder`, ZED wrapper, XRCE, and the bridge node.
4. **Mapping:** Open a separate SSH session to either launch the mapping node directly or record raw sensor data using `rosbag` for offline analysis.

Chapter 4

Results and Discussion

A lot of effort during the development of this work has gone into testing. Working with a real drone platform required ensuring that every hardware component was correctly calibrated and provided high-quality data. Testing algorithms and components in the field was time-consuming, and evaluation was limited on-site due to the limited battery life of the drone mock-up. For this reason, `rosbag` was used to record raw data and test algorithms offline. Tools such as PlotJuggler, RViz and Foxglove were useful to inspect `rosbag` data offline, assessing the quality of the tests.

Another challenge encountered was related to the evaluation of the tests. Usually, test results are compared against a ground truth[1], and several metrics are computed. In this work, another methodology was adopted since determining a ground truth would have required additional resources and work, often unnecessary given the purpose of the test. For instance, since both Pixhawk and ZED 2 include IMUs, the two were compared, providing additional confirmation of their consistency. ZED fusion tools were also tested, as they fuse GNSS data with visual-inertial odometry to theoretically improve the pose estimate, evaluating the quality of raw data.

Regarding depth measurements, only a few tests were conducted using an ArUco marker to measure the actual distance from objects. ArUco marker are easy to detect on RGB data, and therefore inspect the depth value of corresponding pixels. However, this cannot be considered an mm-level precise ground truth, since measurements were taken manually with a measuring tape rather than with more precise instruments such as laser measurement tools. Moreover, the main goal is measuring thin structures in a powerline scenario, which the ArUco marker can only provide limited insight into.

For georeferencing and mapping, no ground truth was available, so only visual tools and some data statistics were computed. During this phase, relevant data were logged from the ROS 2 nodes as CSV files and later inspected using Python scripts. During mapping, tuning ZED parameters required multiple tests, one for each modification.

Finally, the real-time performance of the system was analyzed to assess whether the implemented framework could sustain onboard processing and mapping under real-time constraints.

[1] Ground truth refers to reference measurements assumed to be correct, against which experimental results are compared.

4.1 Sensor Testing

Individual sensor testing was done for the following reasons:

- Evaluate the quality of sensors data in the ROS 2 framework.
- Ensure correct data flow enter the framework.
- Estimate the expected error of the georeferencing pipeline to establish whether this approach to mapping was consistent enough.

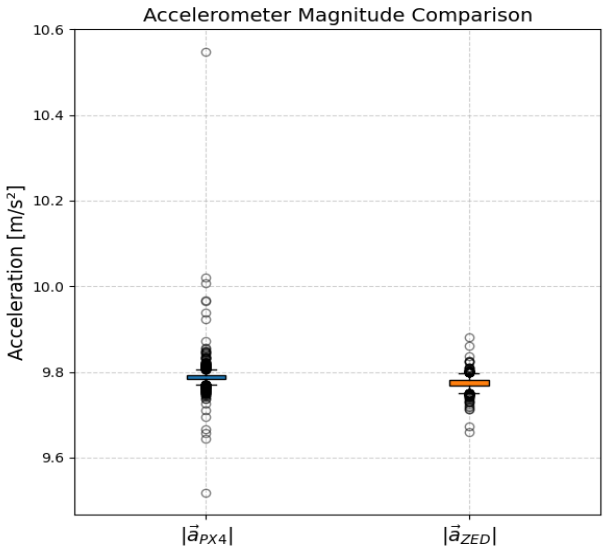
4.1.1 IMU

Within the drone mock-up, two IMUs are available: Pixhawk and ZED. According to the component datasheets, the IMU integrated in the flight controller is expected to provide better performance. However, a simple comparative test was performed between the two to evaluate both their consistency and data quality. The PX4 IMU data are obtained from the bridge node output, while the ZED wrapper already provides ROS `Imu` messages. These messages were renamed in the bridge node, ensuring that both IMU data streams undergo the same processing. In the mapping framework, only the attitude information from IMU data is used. Although quaternions are employed in the georeferencing pipeline, here the rotation is analyzed in terms of roll, pitch, and yaw for easier interpretation. Moreover, raw sensor data from the accelerometer and gyroscope are also analyzed.

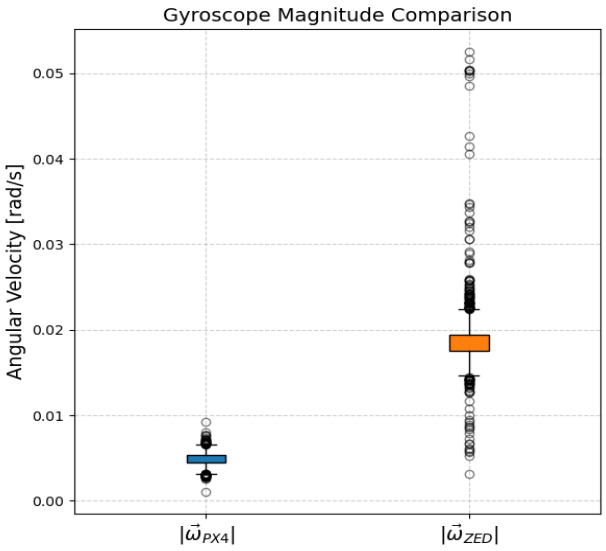
Static Test

A static test was performed by keeping the drone mock-up still on an approximately flat surface for about 60 seconds. Raw data results for the linear acceleration and angular velocity vectors are shown in Figure 4.1. The norm of each vector was computed, and the magnitude results demonstrate that the two IMUs behave very similarly:

- **Linear Acceleration:**
  PX4: Mean=9.7884 Std.=0.0153 m/s^2
  ZED: Mean=9.7742 Std.=0.0101 m/s^2
- **Angular Velocity:**
  PX4: Mean=0.0049 Std.=0.0007 rad/s
  ZED: Mean=0.0186 Std.=0.0022 rad/s



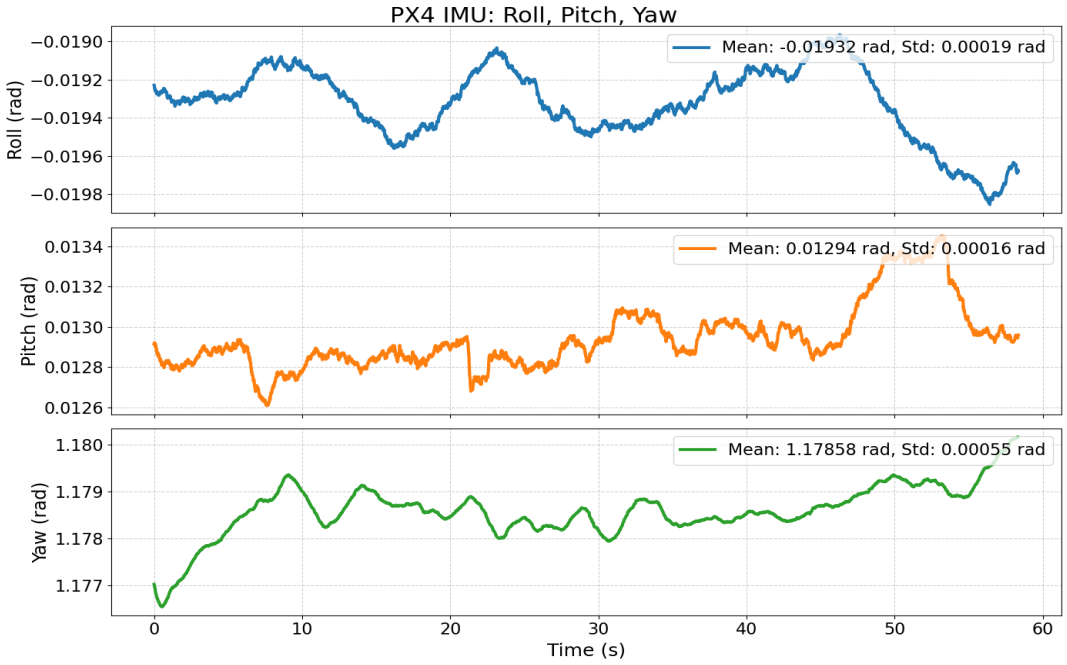
(a) *Linear Acceleration Vector*



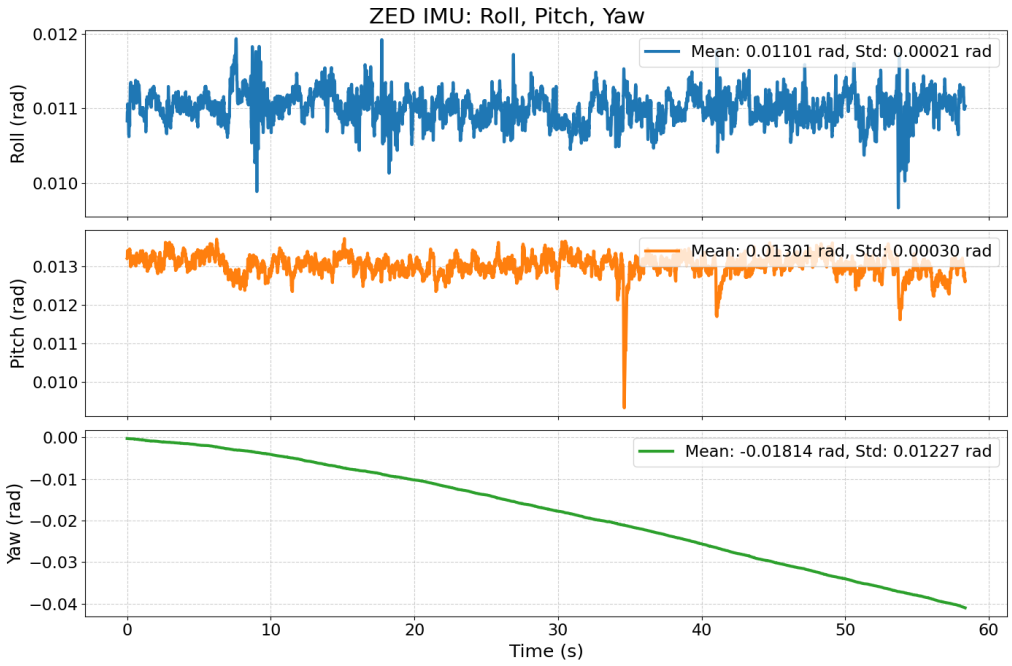
(b) *Angular Velocity Vector*

Figure 4.1: Static test raw data: (a) Accelerometer (b) Gyroscope

Figure 4.2 shows the attitude in terms of roll, pitch, and yaw for both the PX4 and ZED IMUs. An interesting observation can be made about the ZED yaw: it appears not to be aligned with the world frame. Although the ZED is equipped with a magnetometer and the ZED Fusion tool was enabled to fuse GNSS and visual data, its orientation seems aligned with the initial camera position. Over time, it seems to adjusts according to the fusion algorithm, but it remains offset from the world axes. The ZED fusion algorithm is proprietary, so the underlying cause remains unclear. However, examining roll and pitch data, the PX4 IMU exhibits slightly lower standard deviations, suggesting marginally better stability.



(a)



(b)

Figure 4.2: Static attitude: roll (blue), pitch (orange), yaw (green)

An offset between the ZED and PX4 roll values was noted. Further tests, not reported here, revealed a consistent trend: the PX4 roll zero was typically negative, while the ZED's was positive. However, the value varied between tests, preventing identification of a fixed offset. A possible cause could be a slight misalignment in the mounting orientation on the mock-up. However, the standard deviation was too small to justify the variability. Since the PX4 IMU was used in the mapping

framework, and a constant offset would not affect mapping precision, this issue was not considered critical.

Dynamic Test

A dynamic test was performed to evaluate IMU performance during motion, ensuring reliability of both units under dynamic conditions. Due to the yaw misalignment observed in the static test, only roll and pitch maneuvers were intentionally executed, while yaw was still recorded to confirm previous findings. Although no ground truth is available, Figure 4.3 shows that the IMUs remain consistent with each other, both capturing each oscillation in a similar way. To mitigate the influence of the previously observed static offset, data were normalized for a fair dynamic comparison. In particular, the absolute distance between the two IMUs' attitude estimates was computed:

- **Roll:**
  Mean=0.03162, Max=0.04163 *rad*
- **Pitch:**
  Mean=0.00517, Max=0.04130 *rad*

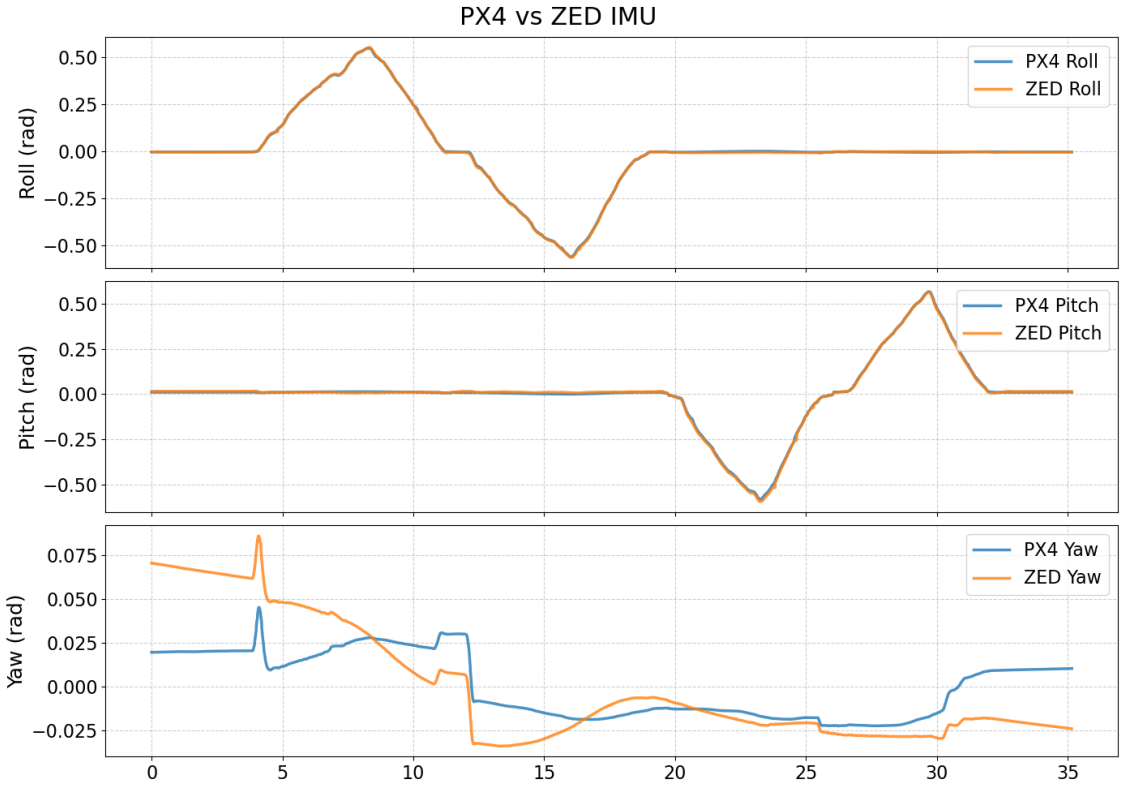


Figure 4.3: Dynamic maneuver: roll + pitch

During this maneuver, a sequence of positive and negative roll followed by positive and negative pitch was performed, attempting to maintain constant yaw. As observed in the static test, the ZED Fusion's contribution to yaw estimation is evident. Nonetheless, both IMUs show nearly identical roll and pitch dynamics, confirming good agreement and reliability.

4.1.2 RTK GNSS

To evaluate the performance of the RTK pipeline, both static and dynamic tests were performed to assess accuracy and consistency. As in the IMU tests, the ZED Fusion algorithm was fed with GNSS data from PX4 (already filtered by its internal EKF) to provide an additional position estimate, which could potentially be better.

Static Test

During the static test, the base GNSS station position was averaged for 15 minutes to obtain a stable reference. Afterwards, the RTK pipeline was started and allowed to stabilize for about a minute to let the corrections and EKF converge. At this point, the drone mock-up was kept stationary for 30 seconds, and GNSS data were recorded at 1 Hz. Figure 4.4 shows the recorded data projected onto the UTM plane, indicating a bi-dimensional RMSE (Root Mean Square Error) with respect to the 2D average of approximately 1 cm. This level of precision is consistent with the expected performance of the RTK setup. Figure 4.5 reports data statistics for the UTM easting, northing, and altitude coordinates. Altitude shows the largest variation, with a standard deviation of 2 cm, compared to less than 1 cm for both the horizontal components.

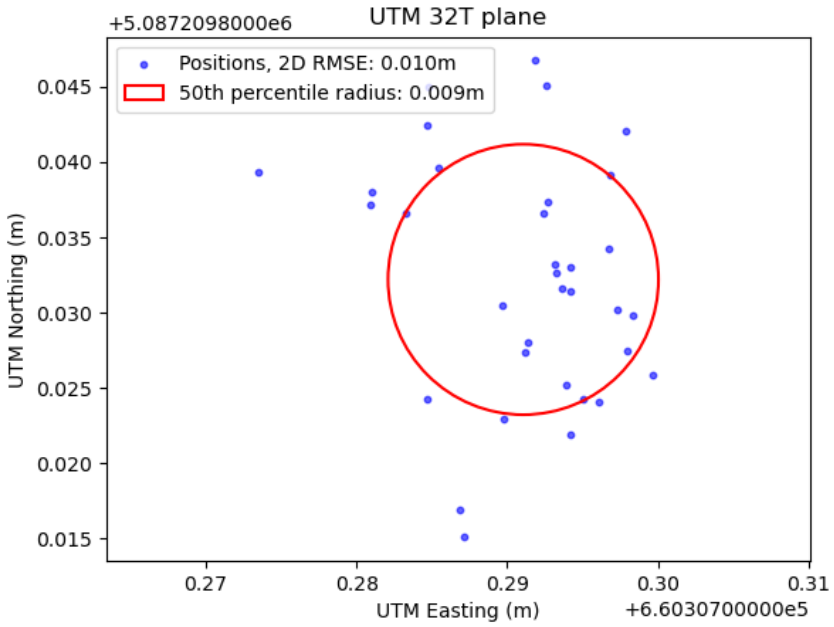


Figure 4.4: UTM Coordinate Plane (Easting, Northing) during static test.

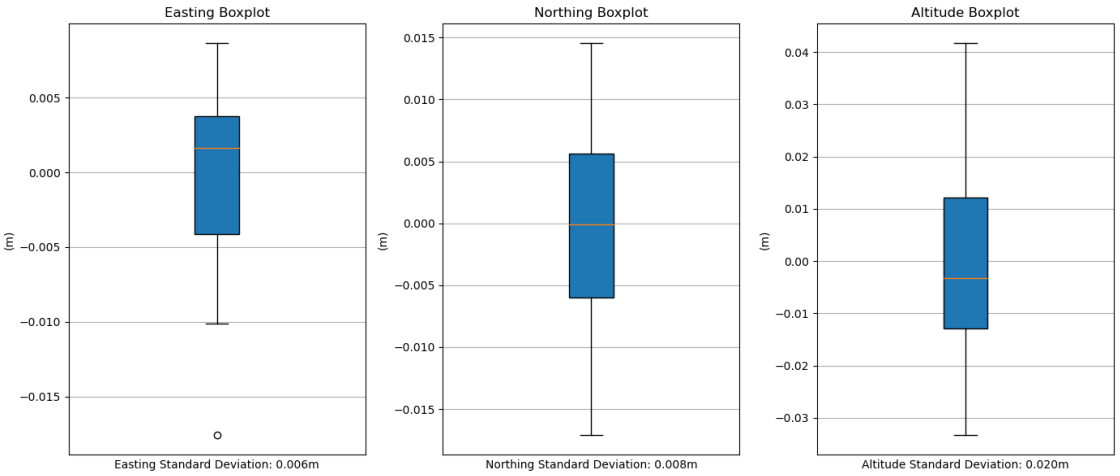


Figure 4.5: Static test statistics for the 3D coordinate vector (Easting, Northing, Altitude).

Dynamic Test

To evaluate GNSS data consistency, a dynamic test was performed by simultaneously recording PX4 and ZED Fusion positions at 1 Hz. In the first test, a rectangular trajectory of measured size $3 \times 4\,\mathrm{m}$ was repeated three times. During this test, the drone mock-up maintained a forward-facing orientation along the trajectory, performing rapid 90° rotations at each corner. The ZED Fusion algorithm attempted to improve position estimates by incorporating visual data, but in this case the results were actually worse (Figure 4.6).

A second test, following a circular trajectory (Figure 4.7), was performed to validate the observations. In this case, the ZED Fusion and PX4 positions were closely aligned, suggesting that the ZED fusion performs better under smooth visual changes. However, overall, the ZED Fusion data proved less reliable and consistent than the PX4 GNSS output alone, and therefore it was not used in the final mapping pipeline.

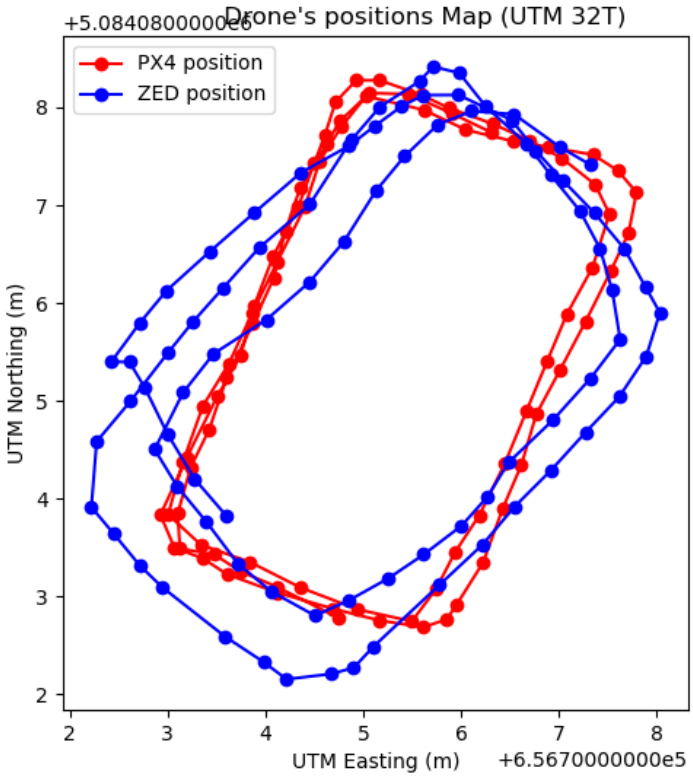


Figure 4.6: RTK GNSS rectangular pattern on the UTM plane.

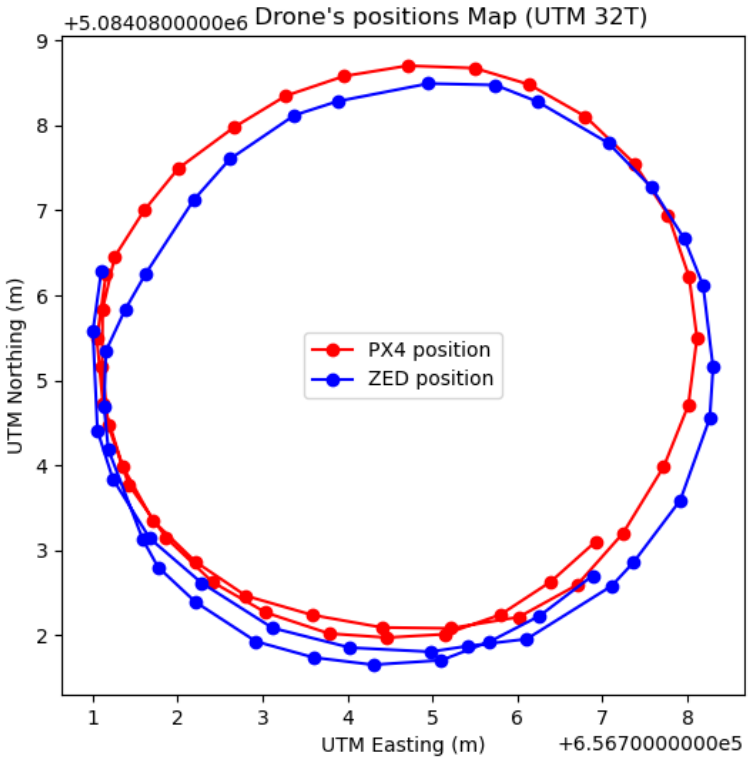


Figure 4.7: RTK GNSS circular pattern on the UTM plane.

4.1.3 Depth Camera

Testing the ZED depth estimation is more challenging than other sensors, as several parameters influence the results in different ways. Moreover, disparity computation strongly depends on the characteristics of the scene: thin structures or textureless surfaces are generally harder to detect. For this reason, different scenes and objects were analyzed to evaluate how scene complexity affects depth accuracy.

- **Easy scene:** An ArUco marker placed on a wall and measured at different distances, with various parameter combinations.

- **Challenging scene:** Depth frames of transmission towers acquired with different parameter configurations.

The ZED SDK allows choosing the stereo matching algorithm to use. In this work, two neural network–based algorithms were tested: `NEURAL_LIGHT` and `NEURAL_PLUS`. The latter provides the highest accuracy and is recommended for inspection purposes, but it is computationally heavier. Therefore, `NEURAL_LIGHT` was also considered, as it offers a good trade-off between accuracy and processing speed.

Both algorithms output a confidence value for each pixel that defines its uncertainty. The higher the value the more uncertainty. The parameters `depth_confidence` (d) and `depth_texture_conf` (t) define thresholds above which pixels are discarded. Reducing these thresholds reduces noise, but excessive filtering can leave too few valid points, affecting completeness.

ArUco Distance

To evaluate depth estimation performance, an ArUco marker test was performed using the `NEURAL_LIGHT` mode with different confidence threshold combinations. Measurements were taken at known distances of 1, 4, 7, and 10 m. However, since the drone mock-up was manually positioned, a small human placement error is expected. Figure 4.8 shows the absolute error for each configuration, together with the RMSE across all distances. Because the ArUco was placed on a planar wall, setting the threshold to 100 (no filtering) yielded very accurate results; however, this would not generalize to more complex scenes. The best overall RMSE was obtained with `depth_confidence`=50 and `depth_texture_conf`=90, a configuration that also performed well in transmission tower mapping.

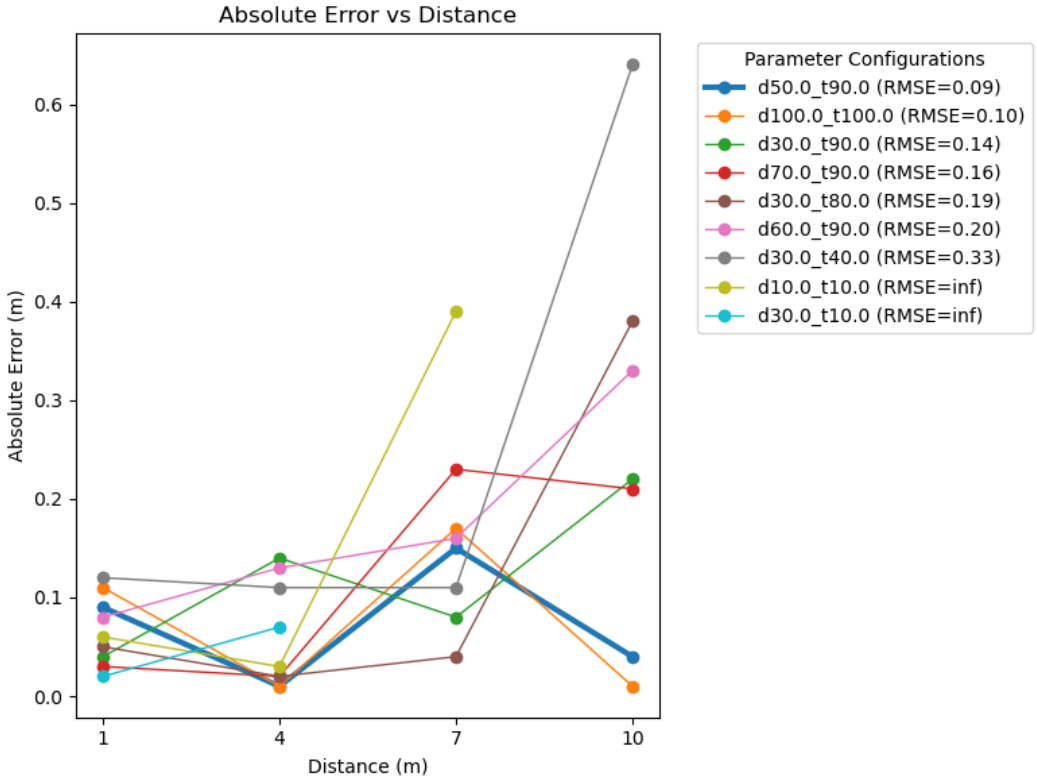


Figure 4.8: ArUco detection at known distances with different parameter combinations.

Depth Frames on Transmission Towers

Depth images of transmission towers were captured using different depth modes, parameter configurations, and distances. As shown in Figures 4.9–4.10, the `NEURAL_LIGHT` mode was insufficient to capture fine structural details, while `NEURAL_PLUS` (Figures 4.11–4.13) provided notably better performance. Although the `NEURAL_LIGHT` frames suggest that low `depth_confidence` values lead to too few valid pixels, this difference was less pronounced with `NEURAL_PLUS`. Furthermore, in mapping applications, the accumulation of multiple frames mitigates this limitation.

Since it was not evident which `depth_confidence` value yielded the best trade-off, thresholds of 25, 50, and 75 were selected for evaluation during mapping. Regarding `depth_texture_conf`, a constant value of 95 was used in all cases and during mapping. This setting provides minimal outlier removal while preserving pixels with lower confidence due to textureless regions.

Finally, Figure 4.14 shows a frame captured at a greater distance (about 8 m), demonstrating the expected degradation in accuracy increasing depth range.

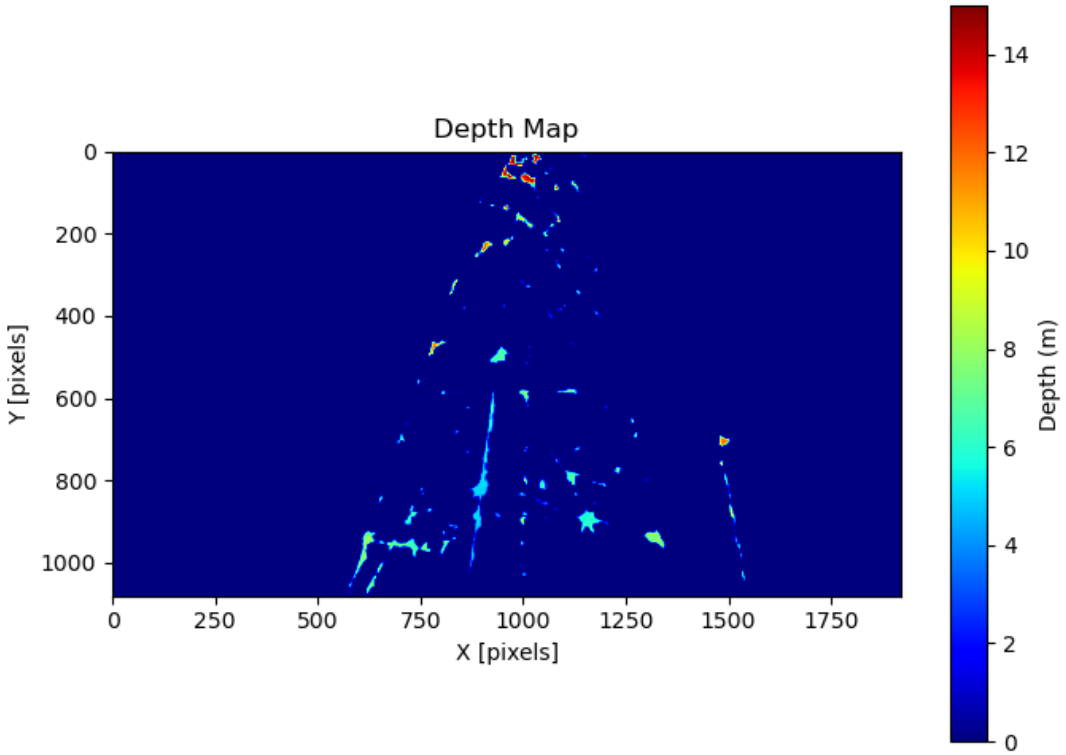


Figure 4.9: Depth frame at about 4 m with `NEURAL_LIGHT`, `depth_confidence`=50, `depth_texture_conf`=95

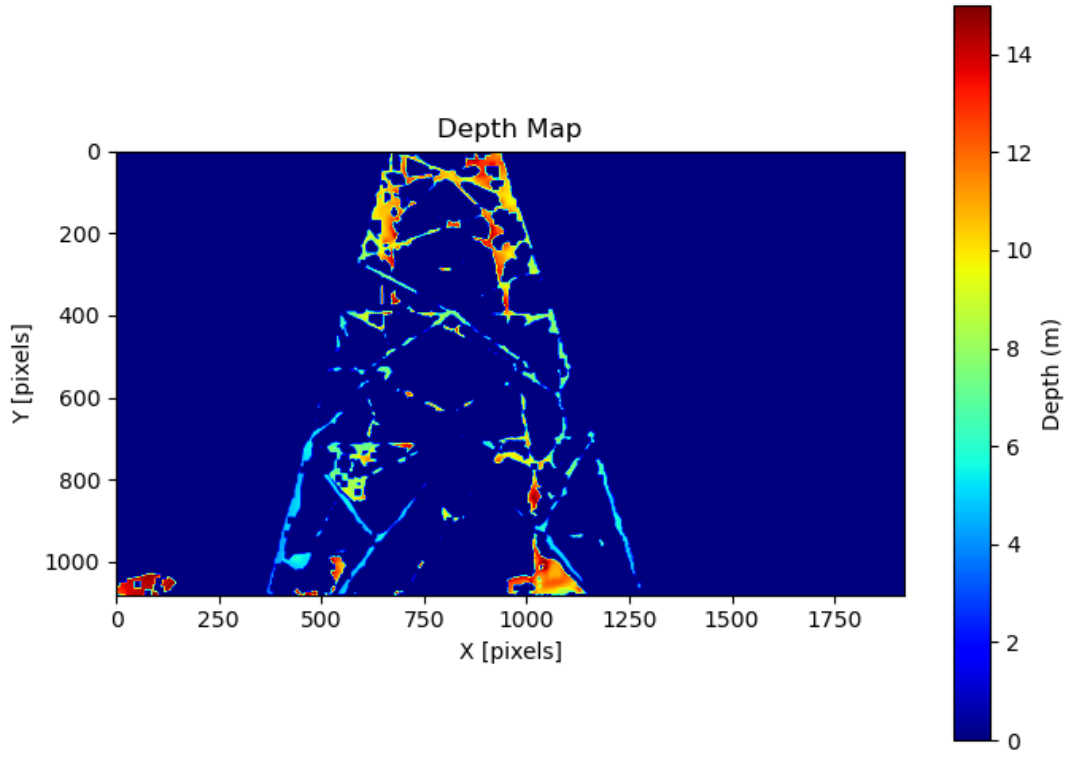


Figure 4.10: Depth frame at about 4 m with `NEURAL_LIGHT`, `depth_confidence`=75, `depth_texture_conf`=95

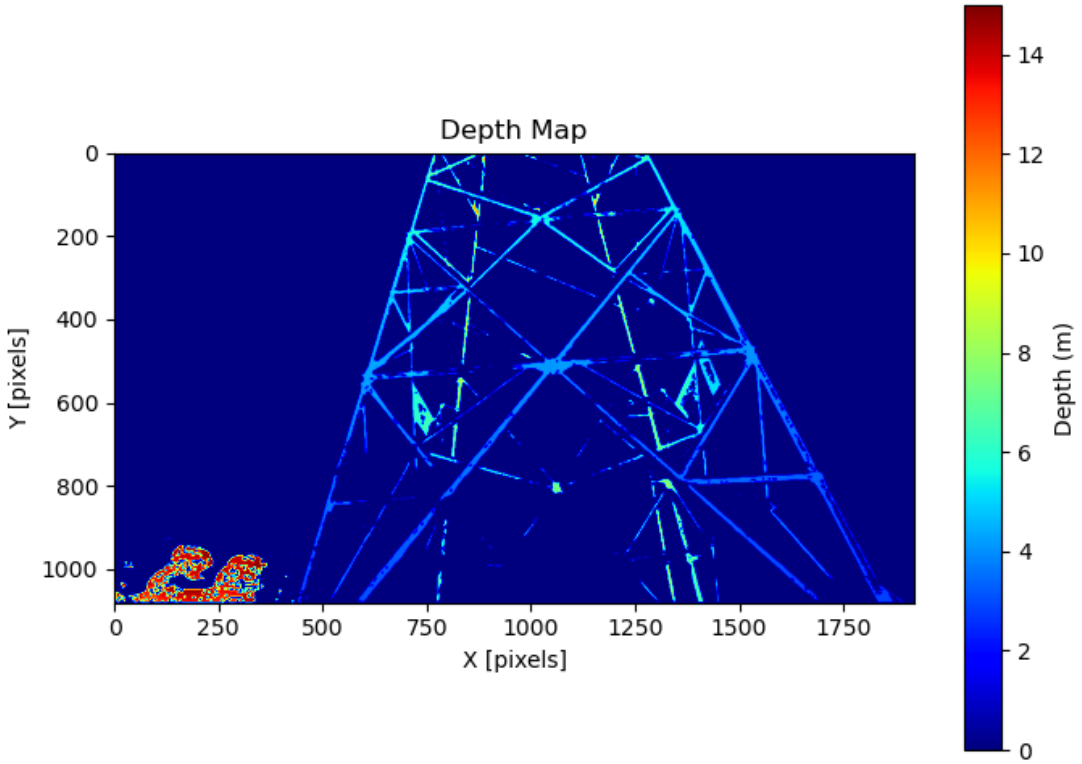


Figure 4.11: Depth frame at about 4 m with `NEURAL_PLUS`, `depth_confidence`=25, `depth_texture_conf`=95

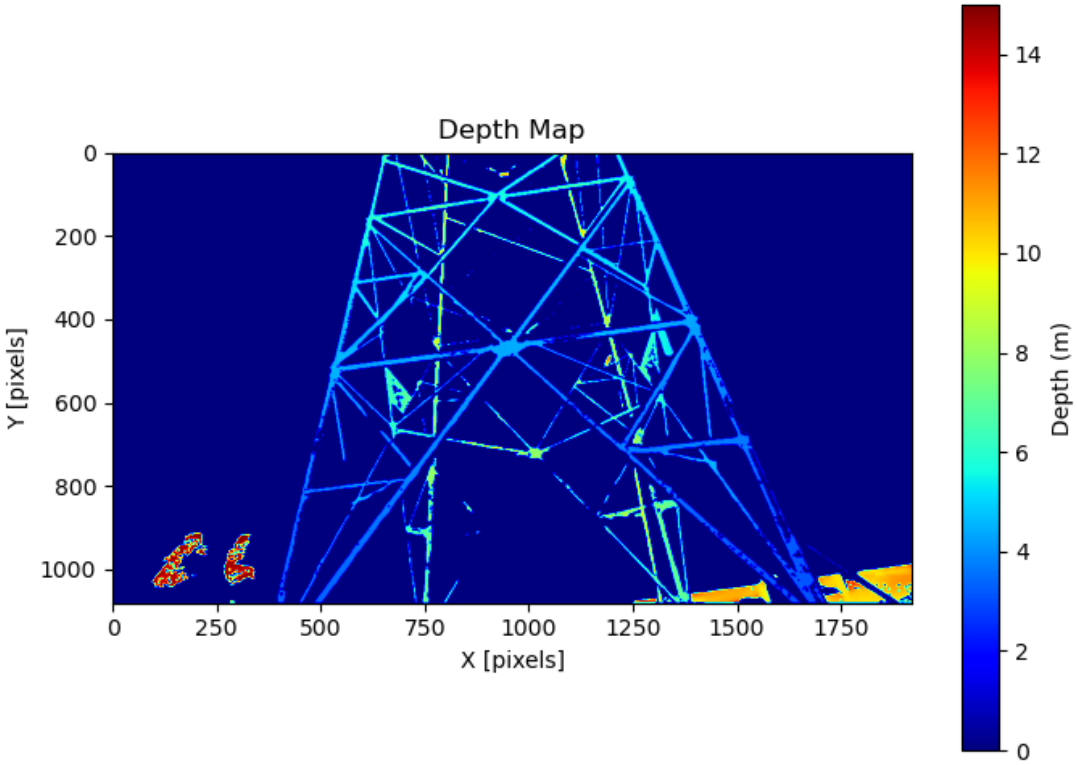


Figure 4.12: Depth frame at about 4 m with `NEURAL_PLUS`, `depth_confidence`=50, `depth_texture_conf`=95

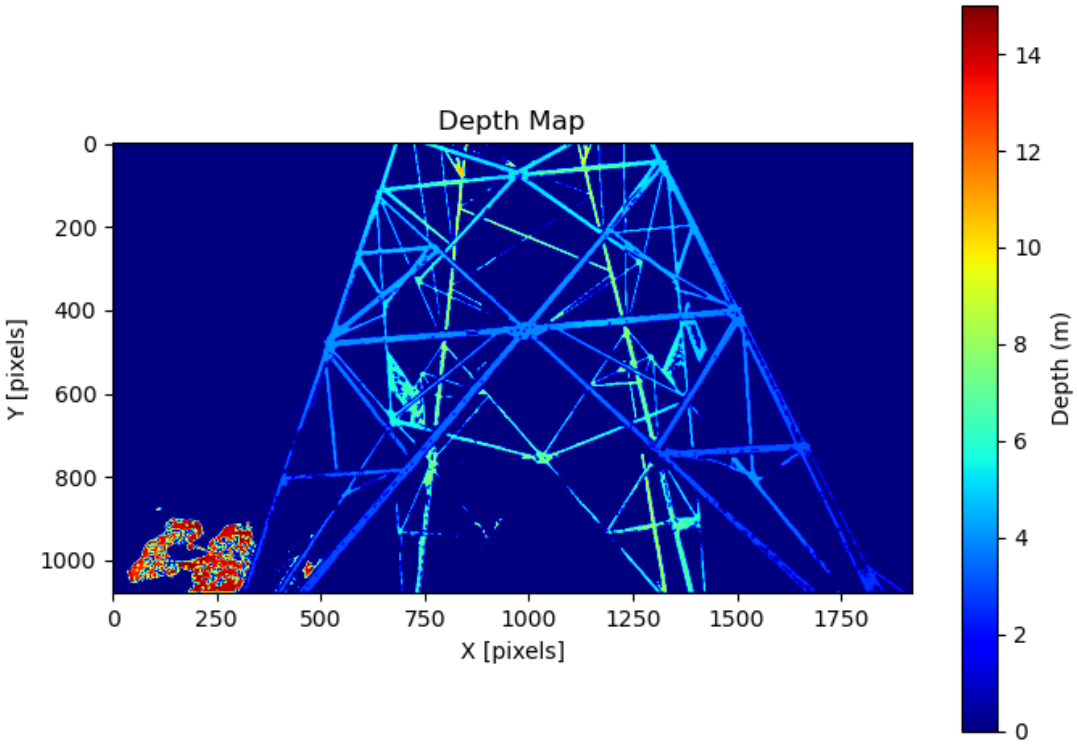


Figure 4.13: Depth frame at about 4 m with `NEURAL_PLUS`, `depth_confidence`=75, `depth_texture_conf`=95

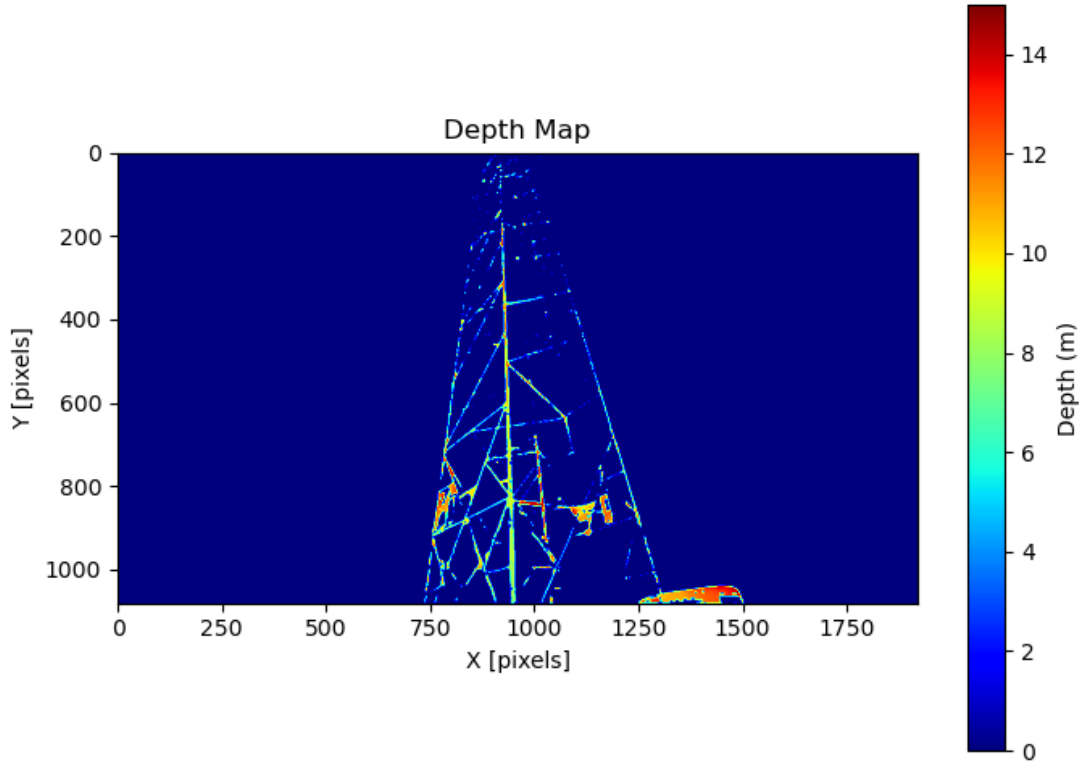


Figure 4.14: Depth frame at about 8 m with `NEURAL_PLUS`, `depth_confidence`=75, `depth_texture_conf`=95

4.2 Error Propagation

An estimate of the expected error of the georeferencing pipeline can be computed using standard error propagation theory. Due to the non-linearity of the pipeline, a linearization of the model is applied. Static sensor measurements are used as reference errors for IMU and GNSS, while the best-case depth standard deviation is used. Pixel coordinate errors (image quantization, etc.) are neglected. These approximations provide a **best-case estimate**. Moreover, in practice, IMU and GNSS are dynamic and depth may be less accurate (e.g., in challenging structures such as powerlines).

The non-linear georeferencing model can be expressed as

$$y = f(x) \iff P_w = T_{bw}\, T_{pb}\, P_{\text{pix}}, \tag{4.1}$$

where:

- $P_{\text{pix}} = [x, y, Z, 1]^\top$ is the pixel coordinate with associated depth Z;
- T_{pb} is the calibrated (static) pixel-to-body homogeneous transform;
- $T_{bw} = \begin{bmatrix} R_{bw} & t \\ 0_{1\times 3} & 1 \end{bmatrix}$ is the body-to-world homogeneous transform, with rotation R_{bw} (from IMU) and translation t (from GNSS).

The standard deviations measured during static sensor tests are:

$$\begin{aligned}&\text{Depth: } \sigma_Z,\\ &\text{IMU (Roll, Pitch, Yaw): } \sigma_R,\ \sigma_P,\ \sigma_Y,\\ &\text{GNSS (E,N,U): } \sigma_E,\ \sigma_N,\ \sigma_U.\end{aligned}$$

The input vector of the model and its covariance is:

$$x = \begin{bmatrix} Z \\ \theta_r \\ \theta_p \\ \theta_y \\ t_x \\ t_y \\ t_z \end{bmatrix}, \qquad \Sigma_x = \operatorname{diag}\left(\sigma_Z^2, \sigma_R^2, \sigma_P^2, \sigma_Y^2, \sigma_E^2, \sigma_N^2, \sigma_U^2\right) \tag{4.2}$$

Assumptions

- **Small errors:** perturbations are small, justifying a first-order Taylor expansion.
- **Independence:** depth, IMU, and GNSS errors are uncorrelated.

First-order Error Propagation

For small uncertainties, the first-order Taylor expansion at the mean $\bar{x}$ approximates the model as:

$$y \approx f(\bar{x}) + J(\bar{x})(x - \bar{x}) \tag{4.3}$$

where $J(\bar{x})$ is the Jacobian evaluated at $\bar{x}$. For small perturbations, the covariance of y can be approximated by[2]

$$\Sigma_y \approx J \Sigma_x J^T \tag{4.4}$$

For the georeferencing model, the Jacobian is the 3×7 matrix $J = [J_Z \quad J_\theta \quad J_t]$, hence:

$$\Sigma_{P_w} \approx J_Z \sigma_Z^2 J_Z^T + J_\theta \Sigma_\theta J_\theta^T + \Sigma_t \tag{4.5}$$

Small Perturbations

Writing the non-homogeneous form of the model:

$$p_b = R_{pb} p_{\text{pix}} + t_{pb}, \qquad p_w = R_{bw} p_b + t \tag{4.6}$$

And linearizing p_w about inputs perturbations:

$$\delta p_w \approx \frac{\partial p_w}{\partial Z}\delta Z + \frac{\partial p_w}{\partial \theta}\delta\theta + \frac{\partial p_w}{\partial t}\delta t \tag{4.7}$$

where

- δZ is the depth error;
- $\delta\theta = [\delta\text{roll}, \delta\text{pitch}, \delta\text{yaw}]^\top$ is the IMU orientation error;
- $\delta t = [\delta E, \delta N, \delta U]^\top$ is the GNSS position error.

[2]For a linear map $y = Ax \quad \rightarrow \quad \delta_y = A\delta_x \quad \rightarrow \quad \Sigma_y = (A\delta_x)(A\delta_x)^T = A(\delta_x \delta_x^T)A^T$

IMU Contribution For small angles, a rotation can be approximated using the first-order Rodrigues formula [48]:

$$R(\delta\theta) \approx I + [\delta\theta]_\times, \tag{4.8}$$

with $[v]_\times$ the skew-symmetric matrix. The IMU perturbation thus propagates as[3]:

$$\delta p_w^{\text{IMU}} \approx -R_{bw}[p_b]_\times \delta\theta, \quad \|\delta p_w^{\text{IMU}}\|^2 = (Y_b^2+Z_b^2)\sigma_R^2+(X_b^2+Z_b^2)\sigma_P^2+(X_b^2+Y_b^2)\sigma_Y^2 \tag{4.9}$$

From Equation 4.9, it is evident that each angle affects the components perpendicular to its rotation axis.

Depth Contribution Defining $\alpha = \frac{x-c_x}{f_x}, \beta = \frac{y-c_y}{f_y}$, the Jacobian relative to the depth contribution $\frac{\partial p_w}{\partial Z} = R_{bw}R_{pb}[\alpha\,\beta\,1]^T$. Since rotation matrices preserve norms:

$$\delta p_w^{\text{depth}} \approx R_{bw}R_{pb}[\alpha\,\beta\,1]^T\delta Z, \quad \|\delta p_w^{\text{depth}}\|^2 \approx (\alpha^2+\beta^2+1)\sigma_Z^2 \tag{4.10}$$

GNSS Contribution Applying a perturbation on the translation vector t has a direct linear effect on the global error:

$$\delta p_w^{\text{GNSS}} = \delta t, \quad \|\delta p_w^{\text{GNSS}}\|^2 \approx \sigma_E^2+\sigma_N^2+\sigma_U^2 \tag{4.11}$$

Total Standard Deviation

Assuming independence of errors:

$$\|\delta p_w\|^2 = \|\delta p_w^{depth}\|^2 + \|\delta p_w^{[}IMU]\|^2 + \|\delta p_w^{GNSS}\|^2$$

allows to express the total standard deviation as:

$$\sigma_{\text{tot}} \approx \sqrt{(\alpha^2+\beta^2+1)\sigma_Z^2 + (Y_b^2+Z_b^2)\sigma_R^2 + (X_b^2+Z_b^2)\sigma_P^2 + (X_b^2+Y_b^2)\sigma_Y^2 + \sigma_E^2+\sigma_N^2+\sigma_U^2} \tag{4.12}$$

Central Axis Approximation

A further simplification can be applied considering a point near the optical axis, where $\|p_b\| \approx Z$. This implies $\alpha \approx \beta \approx 0$ and $[X_c, Y_c, Z_c]^T \approx [0, 0, Z]^T$. The total standard deviation then depends only on Z:

$$\sigma_{\text{tot}}(Z) \approx \sqrt{\sigma_Z^2 + Z^2(\sigma_R^2+\sigma_P^2+\sigma_Y^2) + \sigma_E^2+\sigma_N^2+\sigma_U^2} \tag{4.13}$$

Stereo Depth Error

While IMU and GNSS errors can be considered constant, the stereo depth uncertainty grows quadratically with distance. From the stereo camera model (Equation 1.1):

$$\sigma_Z = \frac{\partial Z}{\partial d} = \frac{f\,b}{d^2}\sigma_d = \frac{Z^2}{f\,b}\sigma_d, \qquad d = \frac{f\,b}{Z} \tag{4.14}$$

Here σ_d can be assumed approximately constant, although in reality it depends on the stereo matching algorithm and texture conditions. Thus:

$$\sigma_Z(Z) \approx kZ^2, \quad k = \frac{\sigma_Z(Z_0)}{Z_0^2} \tag{4.15}$$

[3]The rotation matrix preserve the norm, so the contribution on the squared error is given by $\|[p_b]_\times\delta\theta\|^2$

Expected Error of the Pipeline

Using the stereo depth model (Equation 4.15) and experimental data, a value of $\sigma_Z(Z_0) = 0.05$ m at $Z_0 = 5$ m was estimated from ZED camera tests. Combined with IMU and GNSS errors (Table 4.1), the expected standard deviation of the georeferencing pipeline can be computed.

Table 4.1: Sensor standard deviations used for georeferencing error estimation.

Sensor	Quantity	Std.
Depth (ZED stereo)	σ_Z	0.05 m at $Z_0 = 5$ m
IMU	σ_R	0.00019 rad
IMU	σ_P	0.00016 rad
IMU	σ_Y	0.00055 rad
GNSS	σ_E	0.006 m
GNSS	σ_N	0.008 m
GNSS	σ_U	0.020 m

Simplified Error vs Depth

Figure 4.15 shows $\sigma_{\text{tot}}(Z)$ for the central-axis approximation, comparing constant and distance-dependent depth error. The effect of the constant σ_Z appears almost flat with respect to $\sigma_Z(Z)$, making the stereo camera depth error the limiting factor.

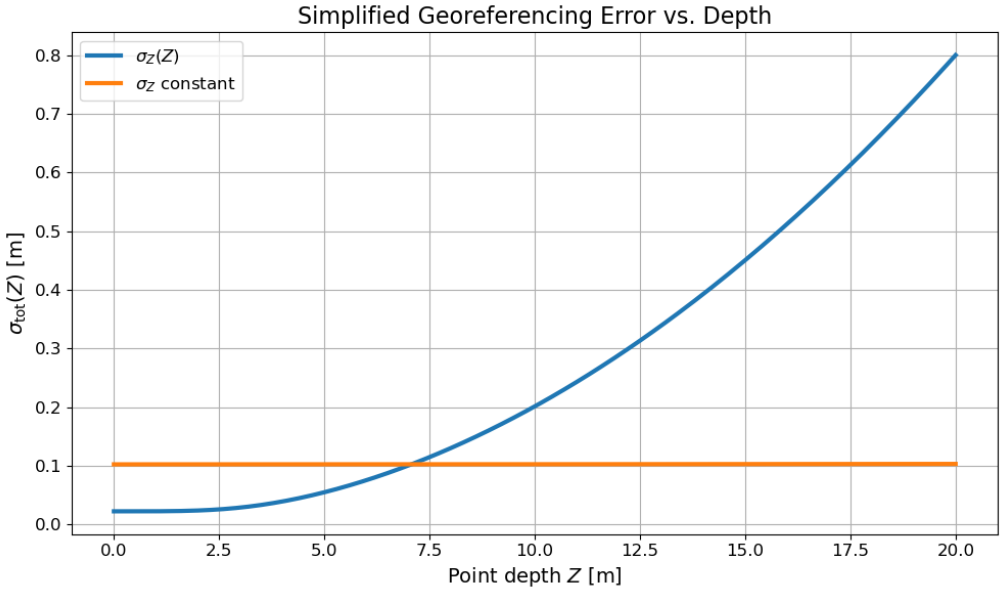
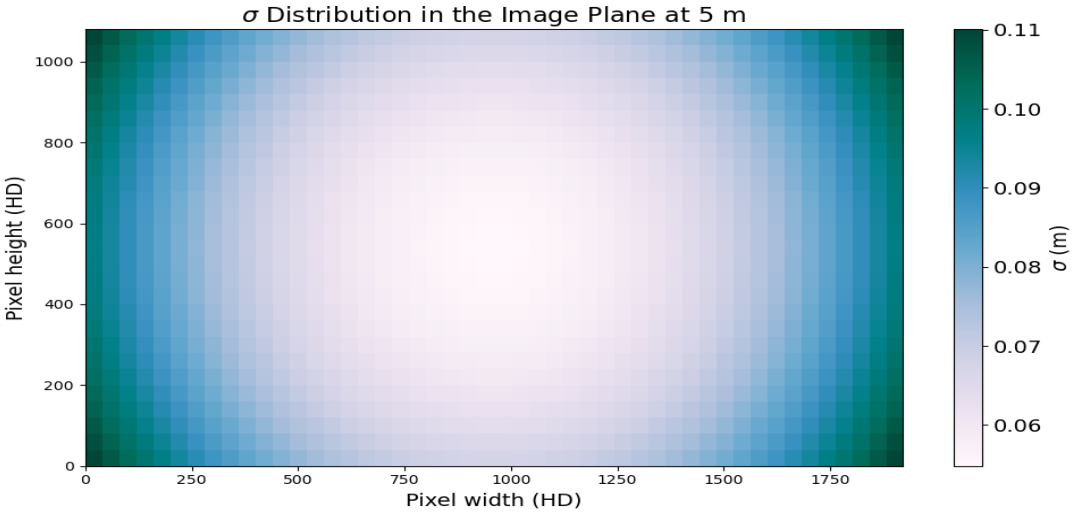


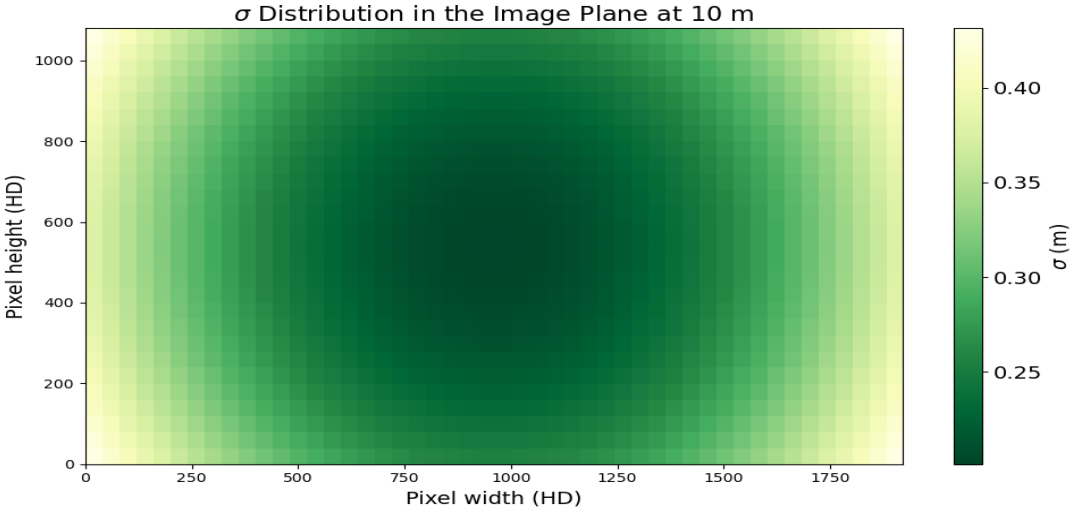
Figure 4.15: Expected standard deviation of the pipeline as a function of distance. Depth uncertainty dominates at larger ranges.

Error Distribution in the Image

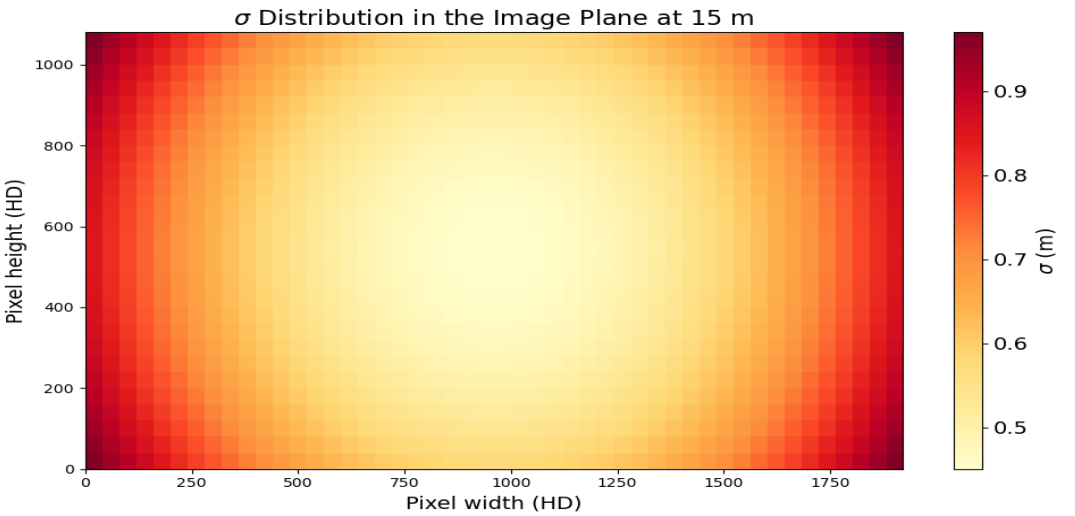
The total standard deviation also depends on pixel location. An HD image (1080×1920) was discretized into macro-pixels of 40×40 pixels, and representative depths of $Z = 5, 10, 15$ m were analyzed. Figure 4.16 shows σ_{tot} distribution across the image plane: errors increase toward the image edges due to off-axis IMU contributions, while depth remains the dominant source of uncertainty.



(a)



(b)



(c)

Figure 4.16: Expected σ_{tot} distribution in the image plane at different depths.

Remarks

- Orientation errors grow linearly with distance $\|p_b\|$ from the UAV.
- Depth error is affected by:
  - the pixel location in the image plane;
  - the object distance from the camera (due to stereo principle).
- GNSS errors contribute linearly and remain approximately constant over distance.
- The dominant uncertainty source is the depth estimate, especially at long ranges.
- The expected mapping error in practice will likely exceed this analytical best-case estimate.

4.3 ArUco Georeferencing Test

In order to evaluate the effectiveness of the georeferencing pipeline, a test detecting an ArUco marker was performed due to its reliable visual detection. In Figure 4.17, a test keeping the drone mock-up on a line with constant orientation facing the marker was conducted. The depth computation mode was set to `NEURAL_LIGHT`, with `depth_confidence`=50 and `depth_texture_conf`=95.

The absence of a ground-truth position for the marker constrains the analysis to rely only on data statistics, such as the standard deviation from the average detected position, and on visual inspection of results. Moreover, the distance from the camera to the marker can be estimated from RGB images, knowing the true dimension of the marker according to:

$$Z = \frac{f_x W_{real}}{W_{pix}} \tag{4.16}$$

where f_x is the focal length along the horizontal axis, W_{real} is the real side length of the (square) ArUco marker, and W_{pix} is the corresponding side length in pixels.

Since the ArUco marker is not challenging to detect, the stereo depth is expected to be similarly accurate. However, Figure 4.18 shows a second test where a continuous measurement of the ArUco marker was performed while following a semi-square trajectory. Interestingly, the average absolute difference between the two distance estimations is about 50 cm, which is higher than expected. Although the specific cause of this offset is not fully identified, repeated experiments confirmed its consistency, indicating a systematic rather than random error. The resulting standard deviation of the georeferenced marker position of this test is about 15 cm, which is significantly higher than in the straight-line test (around 2 cm). Nevertheless, this value is consistent with the expected error estimated in Section 4.2, confirming that the proposed georeferencing pipeline achieves accuracy consistent with analytical expectations, but remains sensitive to depth estimation quality.

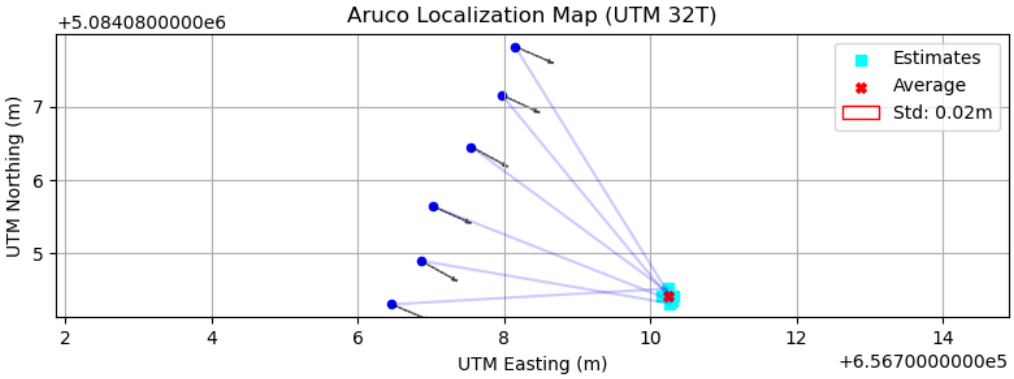
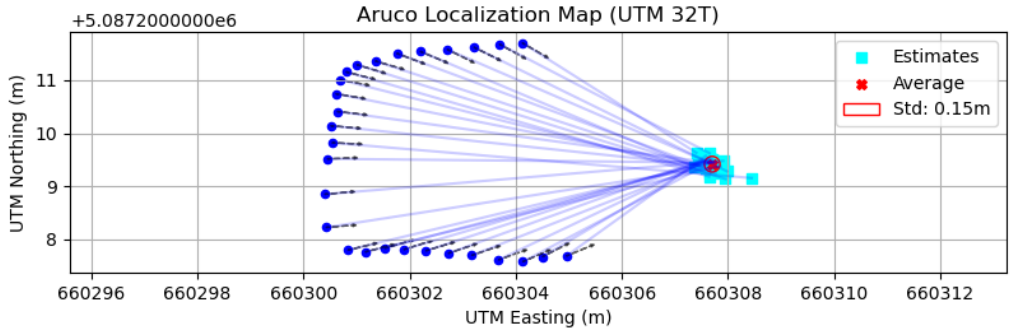
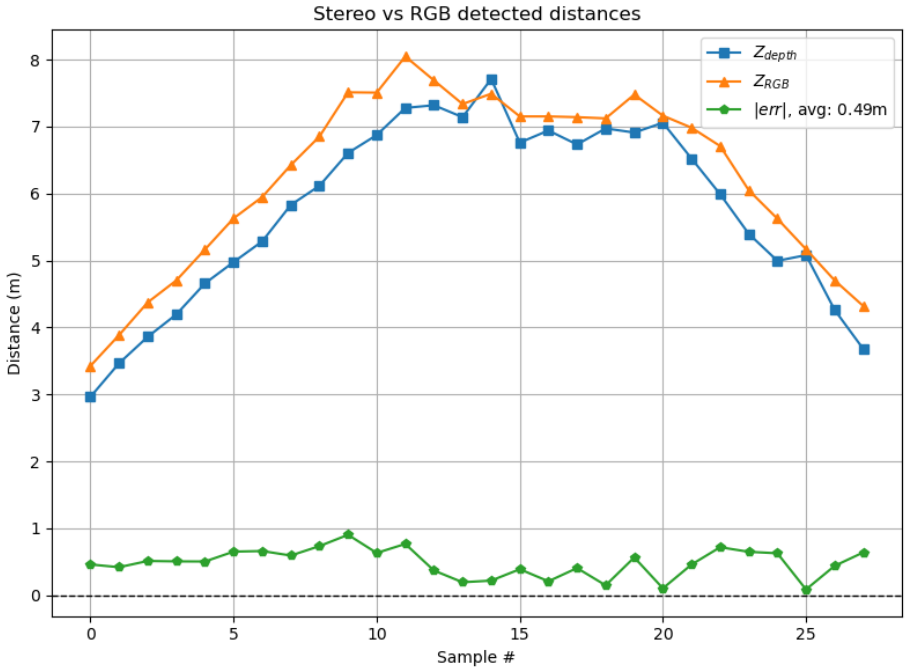


Figure 4.17: Georeferencing of ArUco marker following a straight-line trajectory.



(a) *The georeferenced position of the ArUco shows a standard deviation of 15 cm.*



(b) *Average absolute difference between ZED depth and RGB-based estimate is about 50 cm.*

Figure 4.18: Continuous detections of the ArUco marker. (a) UTM plane view. (b) Difference between stereo depth estimate and RGB estimate according to Equation 4.16

4.4 Transmission Towers Point Cloud

The final test of this work, to validate the effectiveness of the framework in the context of powerline inspection, was mapping transmission towers. Since several parameters needed to be tuned for this purpose, a series of field experiments were done.

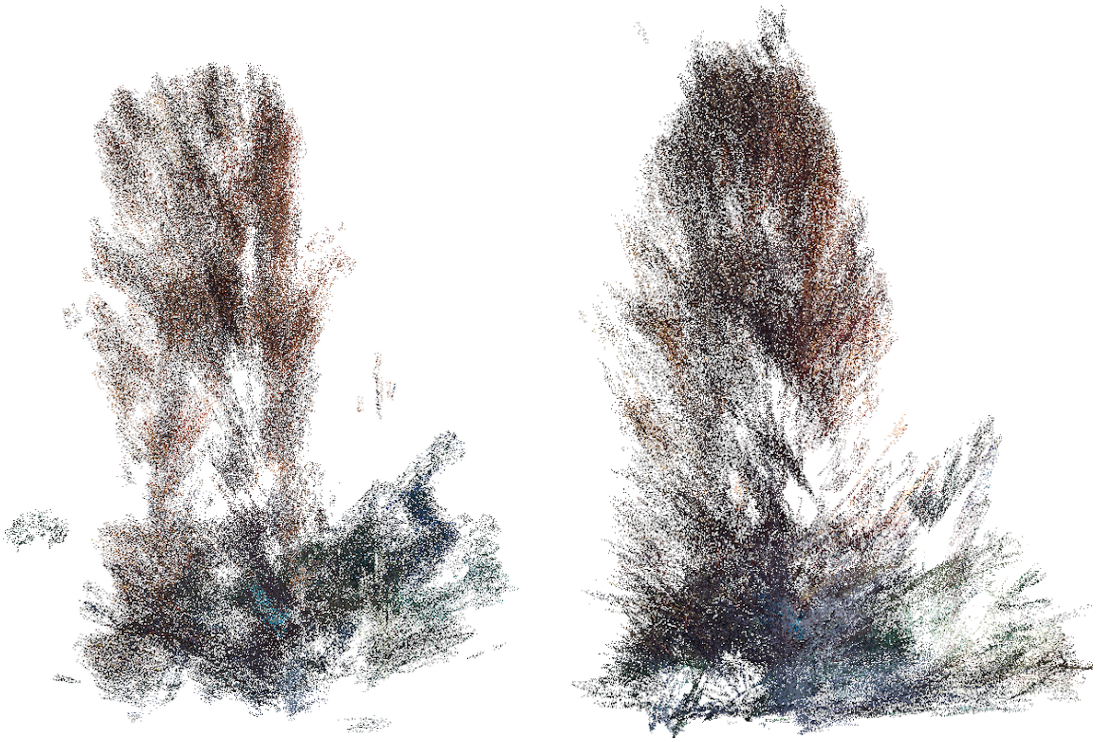
Finding a clear scene suitable for testing was challenging. Most transmission towers in the region are typically hidden by vegetation, vineyards, or, if close to the city, by buildings. After an accurate research for a clear transmission tower, the test scene is reported in Figure 4.19. Moreover, in the background there is another tower of a different type, which is interesting to evaluate the performance of this framework. The tests were performed at a distance of about 5 meters from the tower.

Figure 4.19: Transmission tower used to validate the mapping framework. Following results are referenced to this tower.

The parameters were tuned starting from the depth mode, depth confidence threshold, and finally adjusting the filter parameters of the mapping algorithm. Even though it was not strictly necessary, the maps were colored according to RGB images, averaging the color of pixels in the same grid after the voxel filter. The additional computational load is almost negligible compared to the whole system.

4.4.1 Tuning Depth Mode

Even though the ZED depth test shows that the depth mode `NEURAL_LIGHT` is probably underperforming on transmission towers, a mapping test was done to validate this result. Figure 4.20 shows that this matching algorithm is not sufficient to capture thin details of the tower, independently of the rest of the parameters.



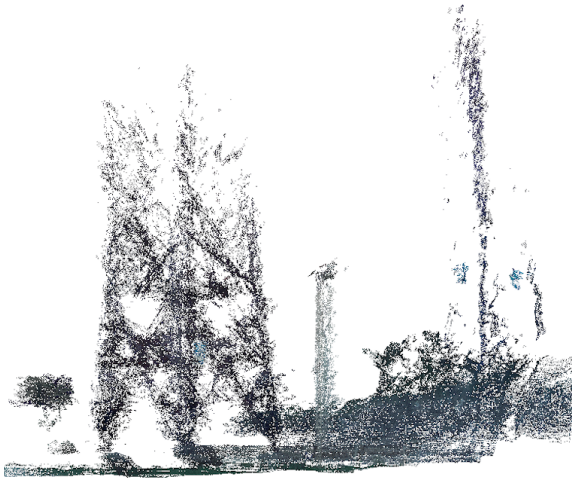
(a) *depth_confidence=75* (b) *depth_confidence=50*

Figure 4.20: Maps with depth mode `NEURAL_LIGHT` and `depth_texture_conf`=95

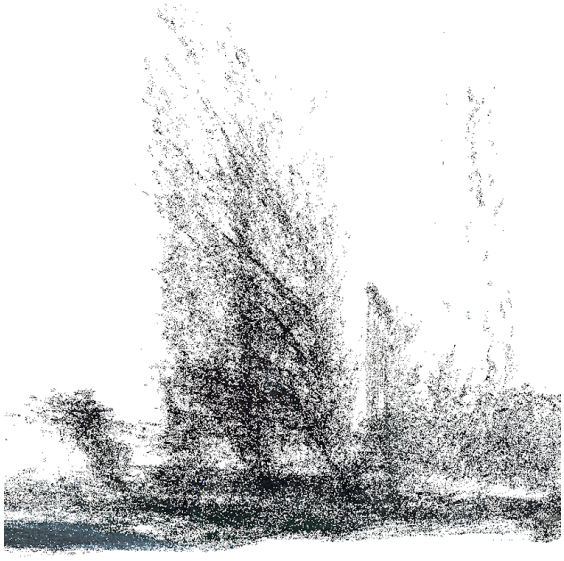
4.4.2 Tuning ZED Parameters

After establishing which depth mode to use, the confidence threshold parameters needed to be tuned. From the depth test on a single frame, it is clear that `depth_texture_conf` can be safely set to 95, while for `depth_confidence` the best trade-off should be assessed. A low confidence would discard more data, but accumulating more frames could potentially compensate for that.

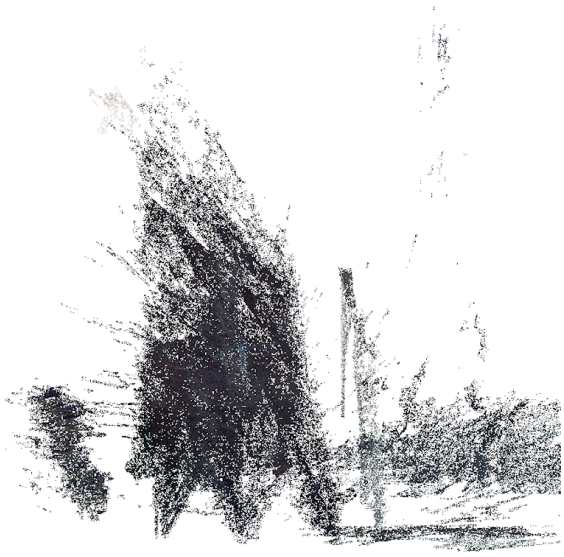
Figure 4.21 demonstrates that this guess was correct. Frame accumulation compensates fully for the few points kept, allowing a more accurate map.



(a) *depth_confidence=25*



(b) *depth_confidence=50*



(c) *depth_confidence=75*

Figure 4.21: Maps with depth mode `NEURAL_PLUS` and `depth_texture_conf`=95

Results allow concluding that challenges of transmission towers can be captured with the ZED 2 camera using the right settings (Table 4.2), and a map in global coordinates can be computed using the RTK GNSS system.

Given the degradation of depth estimation with distance and considering that tests were conducted at approximately 5 m from the base of the tower, a maximum distance for reliable mapping can be considered around 8 m.

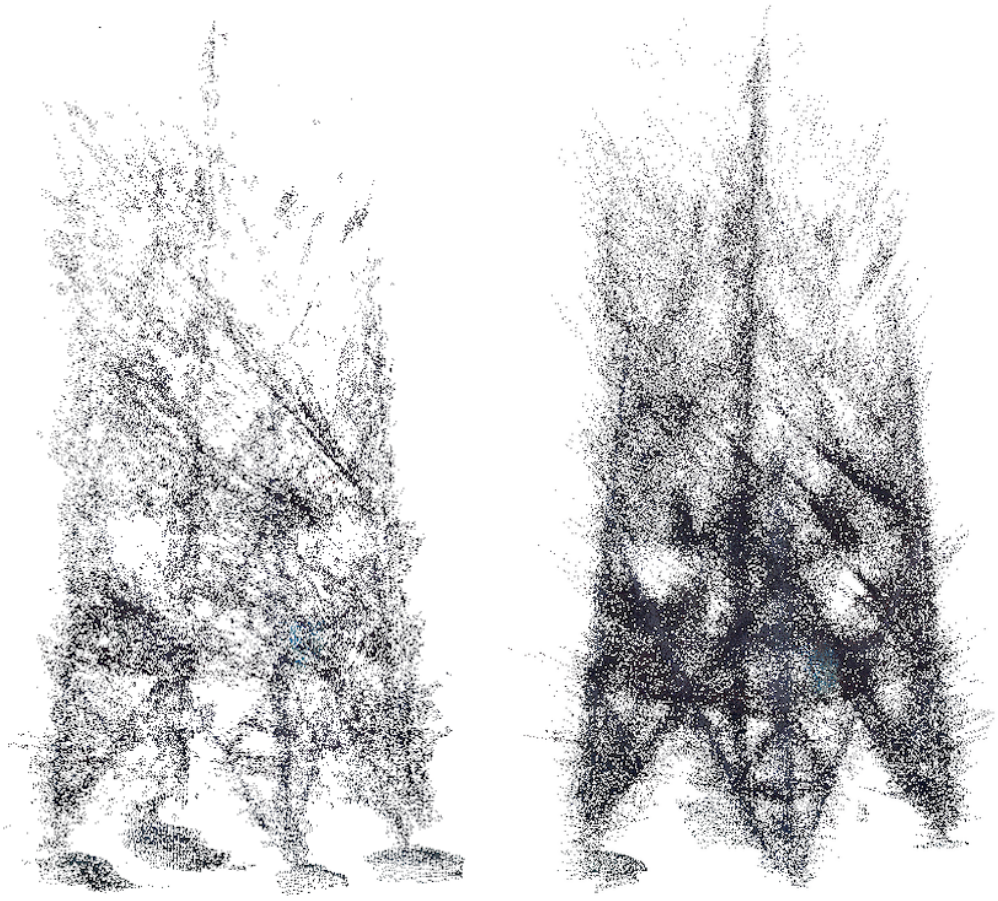
Parameter	**Value**
Resolution	1080p
Frame Rate	15
Minimum Depth	0.5 m
Maximum Depth	15 m
Depth Mode	NEURAL PLUS
Depth Confidence Threshold	25-50
Texture Confidence Threshold	90

Table 4.2: ZED 2 Parameters after tuning.

4.4.3 Tuning Filters Parameters

A further tuning step was performed by choosing the parameters of the filter in the mapping algorithm. This step was done by recording a ROS bag and running the algorithm offline.

As shown in Figure 4.22, a more aggressive filter can reduce the number of points while keeping an acceptable level of detail. Even though the computational load of the mapping framework is not reduced, this can significantly reduce the computational load of successive steps, like path planning. Moreover, since successive tasks are likely done on a ground station, reducing the point cloud size reduces the bandwidth needed for streaming the data.



(a) *Aggressive filtering* (b) *Minor filtering*

Figure 4.22: Tuning mapping filter with depth mode `NEURAL_LIGHT`, `depth_confidence`=25 and `depth_texture_conf`=95

4.5 Real-Time Performance

The proposed mapping framework provides high-quality maps of transmission towers in global coordinates. However, a key requirement for UAV inspection is real-time performance. Onboard resource usage is monitored on the Jetson using `tegrastats` (CPU, GPU, and RAM), and framework latency is measured using ROS 2 topic timestamps. Two types of latency are considered:

- **Input Latency:** The difference between the original timestamp[4] of ZED and Pixhawk topics and the time at which these topics are retrieved in the bridge node.
  This latency is the most relevant, as it depends on both the Jetson and ROS 2 performance, including depth computation time and ROS 2 message handling.
- **Bridge Node Latency:** The difference between the time topics enter the bridge node and the time they exit.
  This is expected to be much lower than the input latency and is measured to

[4]As mentioned earlier, this does not necessarily correspond to the sensor acquisition time

evaluate whether the bridge node introduces a significant delay and must be optimized (e.g., rewritten in C++ or removed entirely).

This section evaluates real-time performance at different stages of the framework: first ZED computation only, then the data acquisition pipeline and finally the full mapping pipeline.

4.5.1 ZED 2 Computation

ZED 2 offloads all computation to the host GPU, which can be a disadvantage for an onboard system with other concurrent workloads. Figure 4.23 shows Jetson resource usage during execution of the ZED wrapper only, with depth mode `NEURAL_PLUS`. As expected[5], the GPU load reaches an average of 95%.

The `NEURAL_PLUS` mode also affects the camera frame rate and input latency in ROS 2. The frame rate measured with `ros2 topic hz` was lower than expected:

$$\text{mean} = 2.758\,\text{Hz} \qquad \text{std} = 0.315\,\text{Hz}$$

Comparing input latency between the two depth modes shows a significant increase with `NEURAL_PLUS`:

	`NEURAL_LIGHT`	`NEURAL_PLUS`
Latency avg (ms)	179.84	300.92

Table 4.3: Average latency of the depth topic from the ZED ROS 2 wrapper, comparing depth modes.

[5]Stereolabs documentation states an average GPU load on the Jetson Orin NX 8 GB of 94%, a CPU usage of 10% and a frame rate of 8 fps.

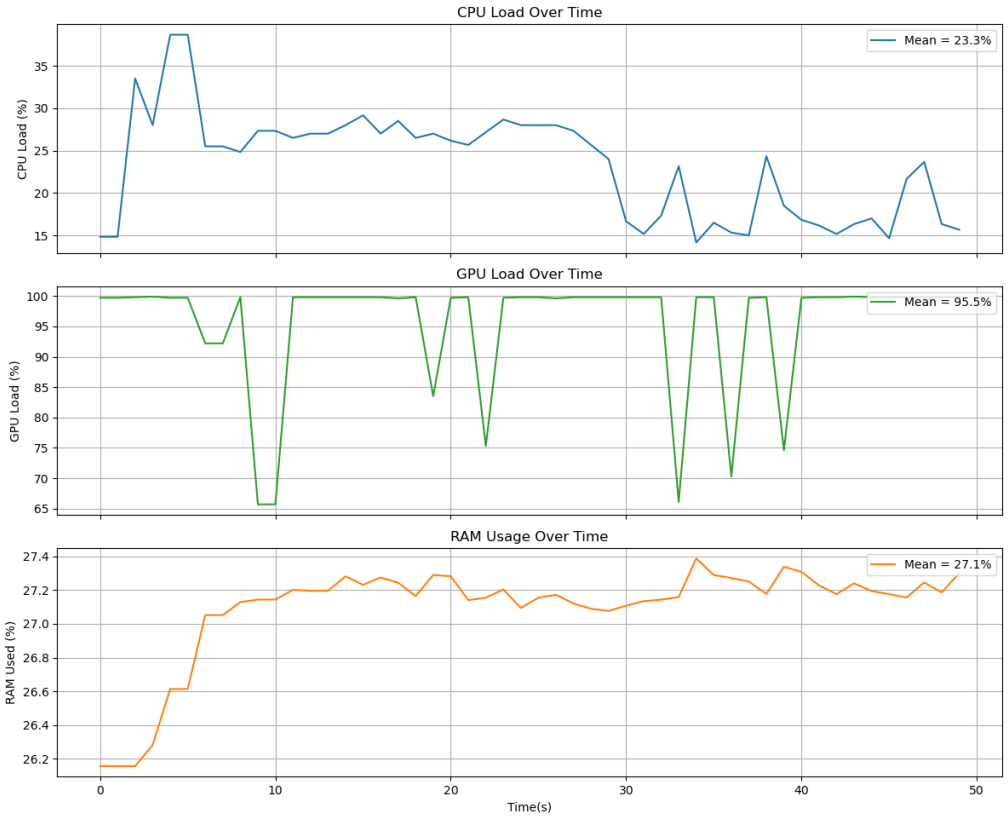


Figure 4.23: Jetson resource usage during execution of the ZED ROS 2 wrapper with depth mode `NEURAL_PLUS`.

4.5.2 Data Acquisition Pipeline

Adding PX4 topics and the bridge node, the average frame rate decreases to:

$$\text{mean} = 1.860\,\text{Hz} \qquad \text{std} = 0.375\,\text{Hz}$$

The Jetson resource utilization during data acquisition is shown in Figure 4.24. When the mapping algorithm is executed concurrently with the ZED 2 camera, the GPU load remains consistently high (around 95%), while the CPU utilization increases compared to running the camera alone. This behavior indicates that the GPU is already heavily engaged by the ZED neural depth estimation, but still allows additional workloads to be scheduled concurrently. The utilization metric in fact, is the proportion of active time and not the absolute computational saturation. Since different workloads (e.g. NN depth estimate, CUDA-based mapping) may use distinct GPU subsystems the scheduler can overlap their execution. However, the additional processing demand and memory transfers increase CPU coordination and I/O latency, and is likely contributing to the observed reduction in frame rate.

The increase in CPU load is also highly related to the additional ROS 2 nodes and data handling overhead.

ROS 2 latency for this pipeline is shown in Figure 4.25. PX4 data maintain low latency, whereas ZED depth computation introduce a delay of about 300 ms.

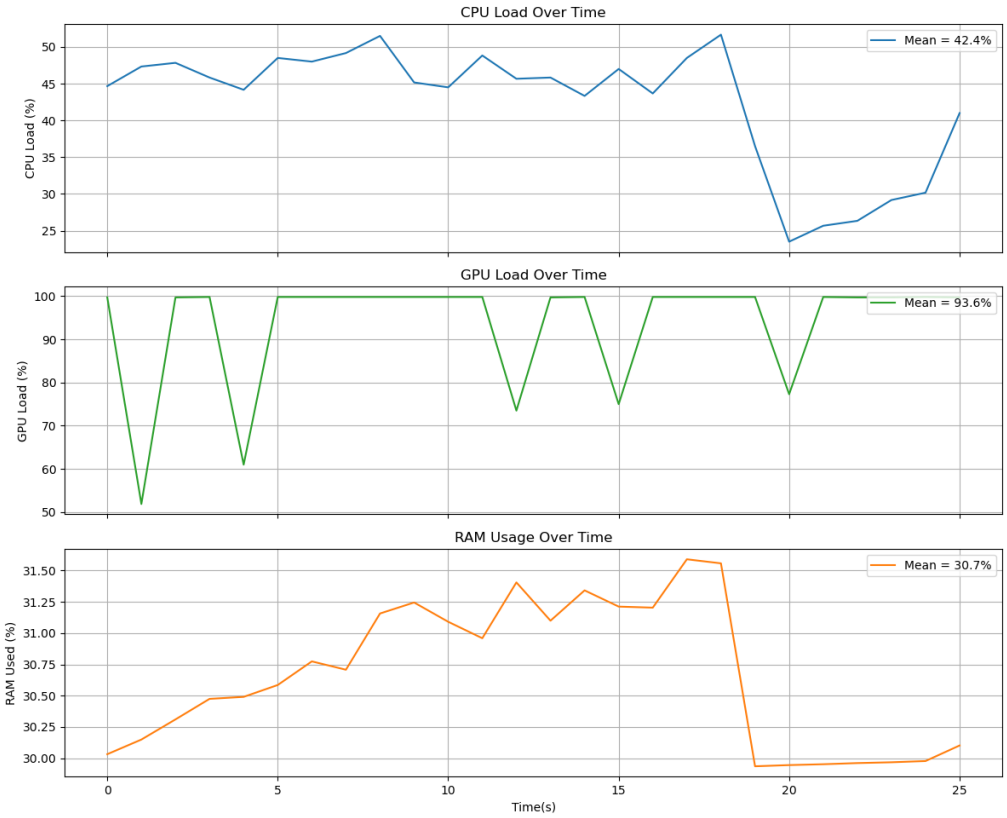
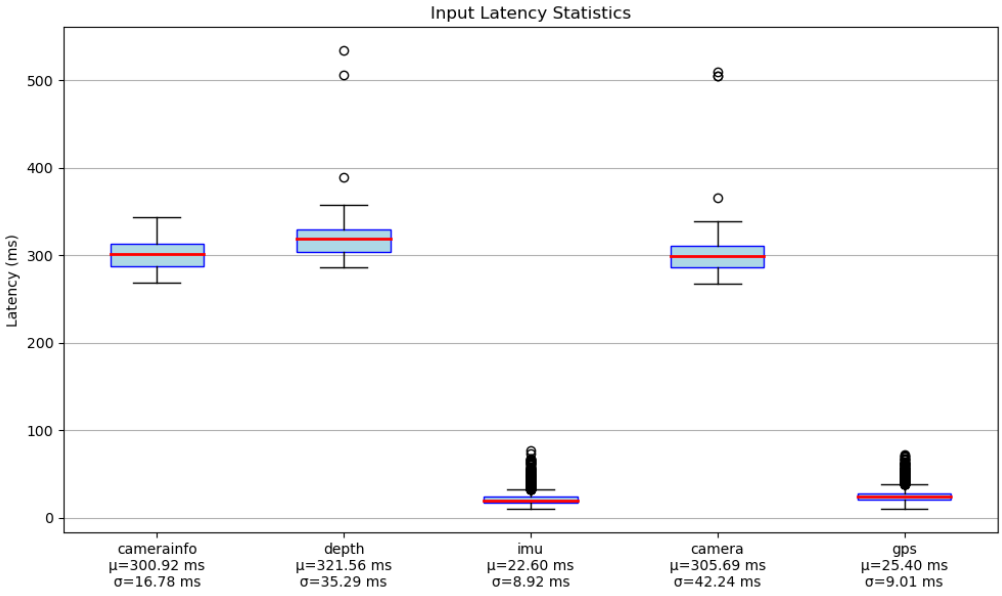
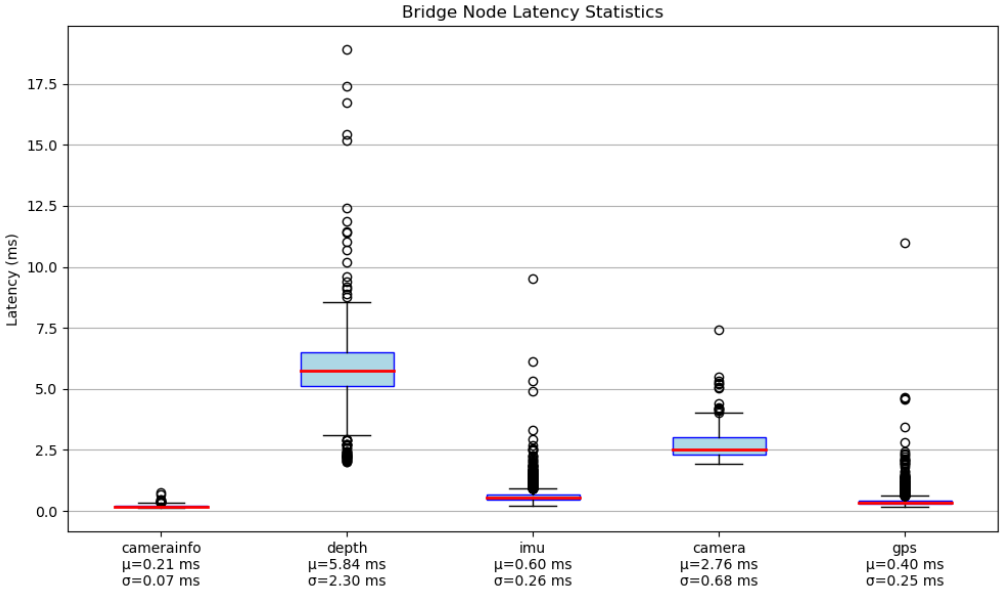


Figure 4.24: Jetson resource usage during the data acquisition pipeline



(a) *Input latency*



(b) *Bridge node latency*

Figure 4.25: ROS 2 latency for the data acquisition pipeline

4.5.3 Point Cloud Processing

To underline the resource demand of the ZED computations, the Jetson stats during the execution of the point cloud algorithm using `rosbag` data are recorded. Result in Figure 4.26 shows that the algorithm used is optimized for real time usage, thanks to C++ coding with CUDA accelerations. ZED processing was hence the bottleneck.

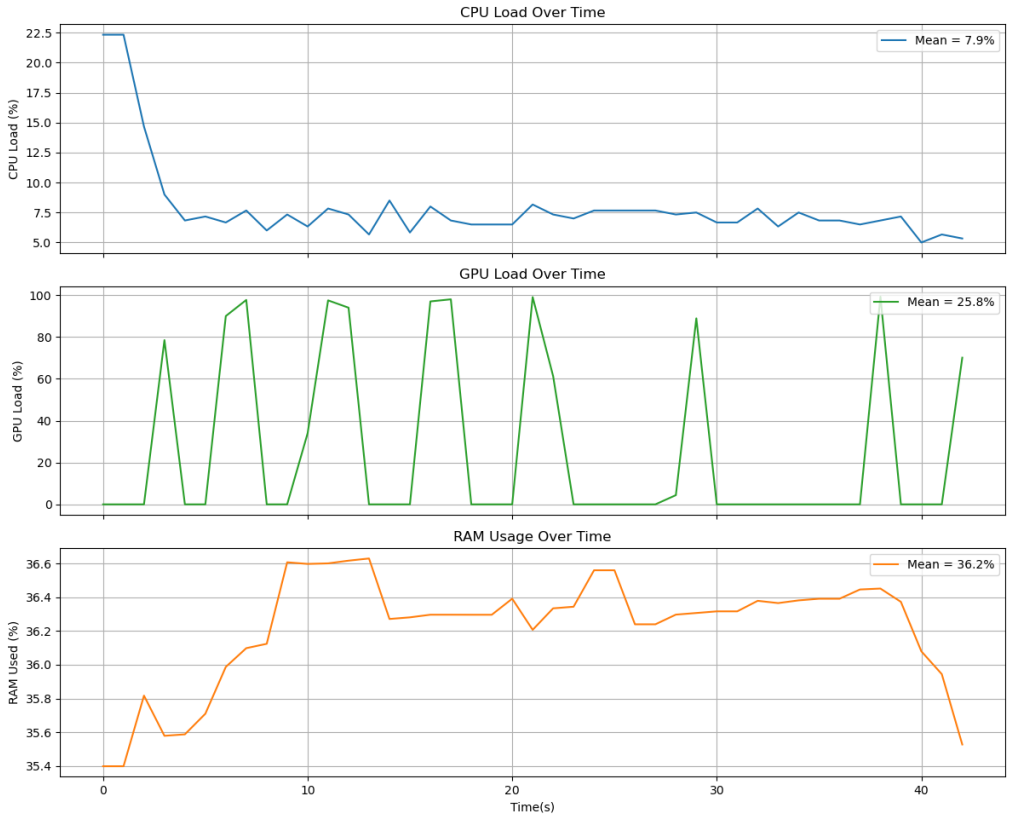


Figure 4.26: Jetson resource usage during the execution of the point cloud mapping algorithm only.

4.5.4 Full Mapping Pipeline

During onboard execution of the mapping pipeline, i.e. data acquisition and mapping algorithm, Jetson resource usage and ROS 2 latency were recorded to evaluate overall performance of the proposed mapping framework.

The Jetson resource usage in Figure 4.27 shows that, although the GPU is heavily utilized by the depth computation, it can still interleave additional kernels from the mapping process. CPU usage is higher than in previous stages, as expected. The average frame rate further decreases:

$$mean = 1.771\,Hz \qquad std = 0.532\,Hz$$

Inspecting the mapping algorithm logs, the mapping algorithm processing time per frame is:

$$mean = 48.92\,ms \qquad std = 3.66\,ms$$

indicating that the algorithm could process frames at up to about 20 Hz without introducing any delay.

Figure 4.28 shows ROS 2 latency for the full pipeline. As expected, bridge node latency is negligible compared to input latency. Interestingly, the largest input latency occurs for the `camerainfo` topic, but this won't affect since is taken only at initialization. RGB and depth topics have more than 300 ms delay, which could seems high. However, given the slow UAV speeds in powerline inspection, this delay can still be acceptable for real-time operation.

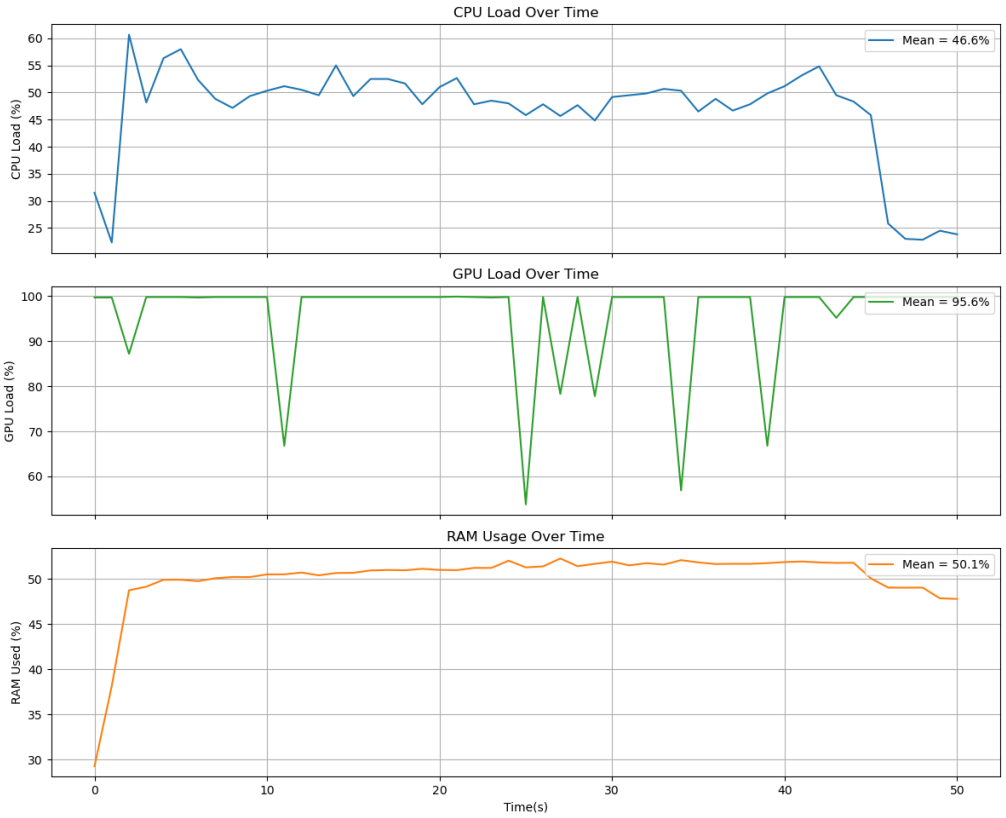
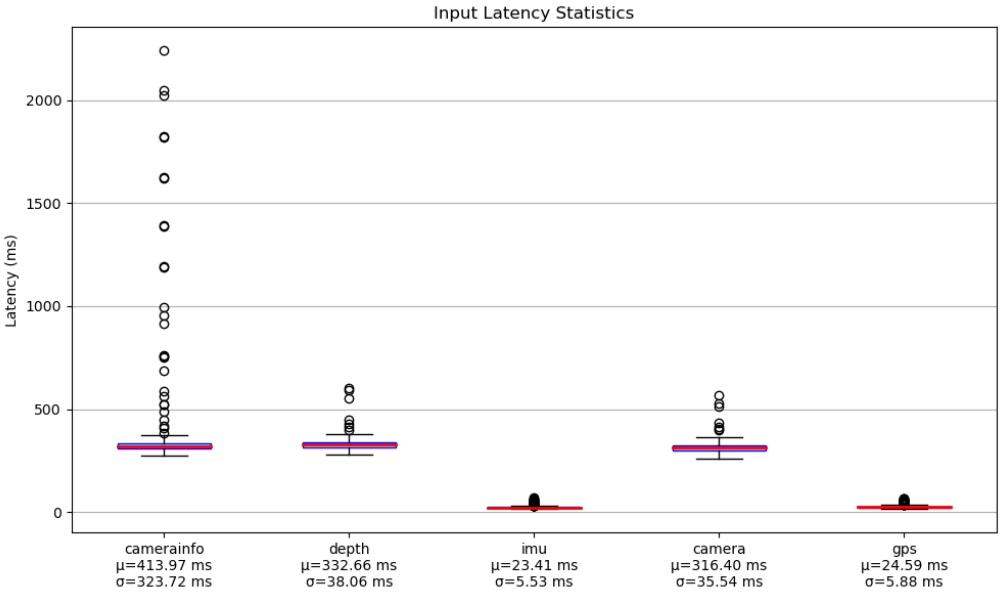
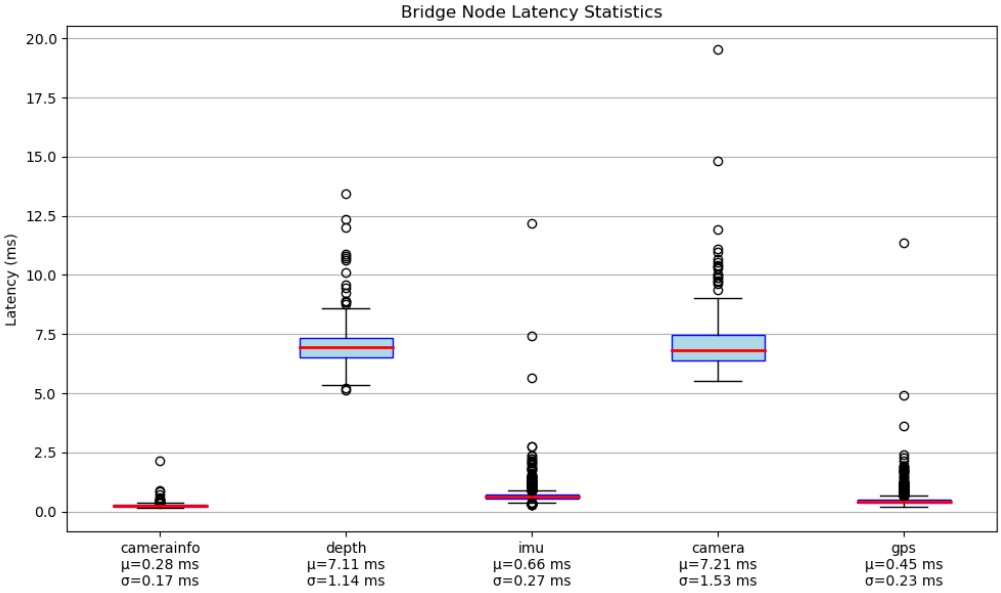


Figure 4.27: Jetson resource usage during execution of the mapping framework.



(a) *Input latency*



(b) *Bridge node latency*

Figure 4.28: ROS 2 latency for the mapping framework.

4.5.5 Real-Time Achievements

In general, a system can be considered real-time if is able to react to external stimuli within time intervals dictated by the environments. In this case study, real-time mapping means that the delay between input (perception) and output (control) is sufficiently small to ensure safe flight mission success. A practical question is: what is the maximum latency that allows the UAV to respond to critical situations? The answer depends on the speed of the drone and the map depth range. From previous results, a depth of 8 meters can be considered an acceptable range. Typical speed for this kind of application is lower than 5 m/s (i.e. 18 km/h). Considering a safety

distance buffer to account for all uncertainties and a safety distance margin, a simple mathematical inequality can be formulated:

$$\textit{reaction distance} + \textit{braking distance} \leq \textit{useful range}$$
$$\text{where } \textit{useful range} = \textit{maximum depth range} - \textit{safety buffer}$$

Assuming a typical drone braking deceleration of $3\,m/s^2$, the inequality can be formulated as:

$$vt + \frac{v^2}{2a} \leq R - s \tag{4.17}$$

where:

- v is the drone speed;
- a is the drone maximum deceleration, assumed $3\,m/s^2$;
- t is the system total latency;
- R is the maximum detection range, assumed $8\,m$;
- s is the safety buffer that can be set large to $2\,m$.

With a mapping latency of about 300 ms, an overall delay of 500 ms can be assumed. Solving the quadratic inequality:

$$v^2 + (2at)v - 2a(R - s) \leq 0$$

yields:

$$v_{max} = at + \sqrt{a^2t^2 + 2a(R - s)} \quad \rightarrow \quad v_{max}(t = 0.5s) = 4.7m/s$$

This can be considered an acceptable maximum velocity for UAV powerline inspection purposes. More generally, Figure 4.29 shows maximum UAV speed as a function of total system latency.

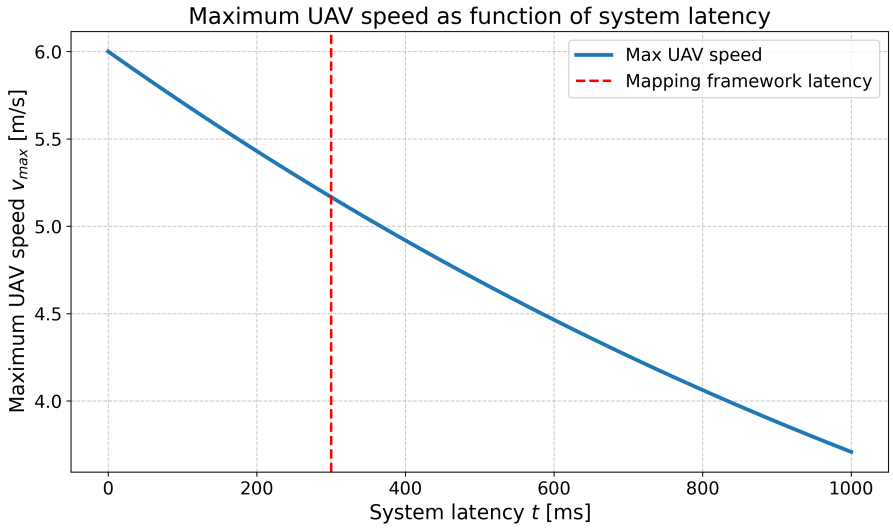


Figure 4.29: Maximum UAV velocity as a function of system latency to ensure real-time capabilities. Assuming a UAV deceleration of $3\,m/s^2$, depth range fo $8\,m$, and a safety distance buffer of $2\,m$.

Conclusions

This thesis presented a real-time mapping framework for UAV-based powerline inspection using a depth camera (ZED 2) and an RTK GNSS system. The objective was to develop a pipeline capable of generating accurate georeferenced 3D point clouds suitable for path planning and navigation. As part of a larger project on autonomous powerline inspection using UAVs, in collaboration with Fraunhofer Italia, the proposed framework was implemented on a drone mock-up and tested on transmission towers. It was validated through sensor and algorithm testing for accuracy and real-time performance. The results demonstrate that the proposed platform is capable of onboard real-time mapping. Considering the high computational demand of the ZED NN-based depth estimation algorithm, the measured latency of the entire pipeline is sufficiently low to allow safe flight at typical inspection speeds. Specifically, with a measured system latency of about $300\,ms$, the UAV can fly at speeds up to almost $5\,m/s$ without compromising safety, assuming a deceleration of $3\,m/s^2$ and a safety buffer of $2\,m$. However, a separate, faster obstacle avoidance system may still be required for real missions given the unstructured and dynamic nature of the environment. Regarding map accuracy, although a ground-truth dataset was not available, visual evaluation of the results, together with statistical analysis, demonstrated that good-quality mapping can be achieved from a distance of approximately $5\,m$. Since stereo depth accuracy decreases exponentially with distance, an upper limit for reliable mapping and obstacle detection can be established at about $8\,m$.

This work assessed the feasibility of achieving a complex task such as autonomous UAV powerline inspection using standard, commercially available technologies. Although most mapping approaches are based on active sensors, stereo cameras have proven capable and deserve further consideration. Moreover, using GNSS-derived pose information is uncommon in similar approaches due to its typical low precision, but the RTK pipeline demonstrated that it can significantly enhance performance and, together with the IMU, provide a pose as precise and smooth as standard odometry estimates. This makes it a strong advantage in the challenging conditions of outdoor environments. Laying the foundation for future research in this area, the proposed framework is modular enough to allow easy integration of additional features or improvements, thanks also to the ROS 2 ecosystem.

5.1 Lessons Learned

During the development of this thesis, several lessons were learned about stereo cameras, hardware constraints and UAV systems:

- **Depth estimation performance:** The ZED 2 demonstrated how a stereo camera can provide both RGB and depth data using a single passive sensor.

However, obtaining precise depth estimates is computationally demanding, and performance degrades exponentially with increasing distance. Thin structures such as transmission towers are particularly challenging to map, especially when a safety distance between objects must be maintained, requiring careful tuning of all relevant parameters.

- **Sensor fusion:** The georeferencing pipeline developed in this work required fusing data from three sensors: IMU, GNSS, and depth. Particular attention had to be paid to data alignment and synchronization. In this context, the ROS 2 middleware proved extremely advantageous, providing standardized tools for reference frame management, time synchronization, data communication, different HW and SW integration.

- **RTK GNSS:** The RTK GNSS system proved reliable and capable of providing smooth and precise pose information suitable for mapping applications. This accuracy is also due to the PX4 EKF, which filters the raw GNSS data. However, the critical component remains the communication link between the base and the onboard receiver: packet loss or communication interruptions quickly degrade performance to standard GNSS levels.

- **Hardware constraints:** When hardware components are involved, computational limitations immediately arise, particularly when real-time performance is required. Clearly, these constraints dictate design choices, requiring software strategies to mitigate them. In particular, the computationally expensive NN-based depth estimation algorithm made the implementation of a fast and efficient mapping algorithm essential. Other tasks are therefore likely to be offloaded to a ground station, balancing onboard processing load and communication bandwidth.

- **Autonomous UAV:** Understanding the challenges related to autonomous UAV missions is critical for design decisions, as many factors are closely interdependent. For example, using a stereo camera can reduce the overall system weight and thus increase flight duration. On the other hand, the computational power required for depth estimation can significantly impact energy consumption, potentially reducing flight time. Moreover, onboard computational resources are limited, meaning that additional tasks could be compromised. Another trade-off involves the real-time constraint: the inevitable latency of the system imposes a limit on UAV speed. At lower speed, mapping accuracy improves, but the mission duration increases, emphasizing the need to balance accuracy, safety and efficiency.

5.2 Future Works

The proposed framework can serve as a foundation for developing path planning and navigation algorithms to validate autonomous powerline inspection. However, several improvements and extensions could enhance both robustness and autonomy, particularly by extending testing to full powerline scenarios and integrating additional perception capabilities necessary for a fully autonomous system.

- **Depth estimation:** Although the ZED 2 parameters were tuned for challenging structures, powerline wires remain an unknown. Further testing on similar thin and reflective structures should be carried out to evaluate the detectability of cables with the stereo camera.
- **Optimize real-time performance:** The input latency of ZED data is difficult to reduce, as it mainly depends on the efficiency of the neural depth computation. However, minor improvements should be explored. For example, dynamic depth resolution could be implemented, reducing the resolution where not needed. Moreover, the use of a more powerful companion computer should be considered, as well as the use of a stereo camera with integrated GPU.
- **Robust wireless communication:** The framework relies on a stable link between the UAV and the ground station for the RTK GNSS pipeline and drone control. While a wired Ethernet connection was used in this work, real-world UAV operations require a robust wireless solution.
- **Enhance autonomy:** Several additional modules are essential to achieve full autonomy, such as:
  - **Object detection module:** automatic detection of objects of interest is needed for semantic environment understanding, enabling task-specific path planning and inspection operations.
  - **Obstacle avoidance module:** to ensure safety in complex and dynamic environments, a faster mapping and control loop to rapidly react to obstacles is mandatory. For this purpose, fast sensors such as range finders could be integrated.

Bibliography

[1] Laghari R.A. Nawaz H. Laghari A.A. Jumani A.K. "Unmanned aerial vehicles: A Review". In: *Cognitive Robotics* 3 (2023), pp. 8–22. DOI: https://doi.org/10.1016/j.cogr.2022.12.004.

[2] Keshari A. Ahmed F. Mohanta J.C. "Recent Advances in Unmanned Aerial Vehicles: A Review". In: *Arab J Sci Eng* 47 (2022), pp. 7963–7984. DOI: https://doi.org/10.1007/s13369-022-06738-0.

[3] S. Jordan et al. "State-of-the-art technologies for UAV inspections". In: *IET Radar Sonar Navig.* 12 (2018), pp. 151–164. DOI: https://doi.org/10.1049/iet-rsn.2017.0251.

[4] Kaabouch N. Boukabou I. "Electric and Magnetic Field Analysis of the Safety Distance for UAV Inspection around Extra-High Voltage Transmission Lines". In: *Drones* 8.2 (2024), p. 47. DOI: https://doi.org/10.3390/drones8020047.

[5] Jakob Simmerle. "Point Cloud Processing". Algorithm to build a point cloud in the world coordinate frame for CUDA deveices.

[6] Anibal Ollero et al. "Application of Intelligent Aerial Robots to the Inspection and Maintenance of Electrical Power Lines". In: *Now Publisher* (2024), pp. 179–201. DOI: 10.1561/9781638282839.ch8.

[7] Yang Tang et al. "Perception and Navigation in Autonomous Systems in the Era of Learning: A Survey". In: *IEEE Transactions on Neural Networks and Learning Systems* 34.12 (2023), pp. 9604–9624. DOI: 10.1109/TNNLS.2022.3167688.

[8] Lyubomyr Demkiv et al. "An Application of Stereo Thermal Vision for Preliminary Inspection of Electrical Power Lines by MAVs". In: *2021 Aerial Robotic Systems Physically Interacting with the Environment (AIRPHARO)*. 2021, pp. 1–8. DOI: 10.1109/AIRPHARO52252.2021.9571025.

[9] Nicolaj Haarhøj Malle, Frederik Falk Nyboe, and Emad Samuel Malki Ebeid. "Onboard Powerline Perception System for UAVs Using mmWave Radar and FPGA-Accelerated Vision". In: *IEEE Access* 10 (2022), pp. 113543–113559. DOI: 10.1109/ACCESS.2022.3217537.

[10] P. Ramon-Soria et al. "Autonomous landing on pipes using soft gripper for inspection and maintenance in outdoor environments". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2019, pp. 5832–5839. DOI: 10.1109/IROS40897.2019.8967850.

[11] K. Wang et al. "UAV-based simultaneous localization and mapping in outdoor environments: A systematic scoping review". In: *Journal of Field Robotics* 41 (2024), pp. 1617–1642. DOI: 10.1002/rob.22325. URL: https://doi.org/10.1002/rob.22325.

[12] Jan Bednář et al. "Deployment of Reliable Visual Inertial Odometry Approaches for Unmanned Aerial Vehicles in Real-world Environment". In: *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2022, pp. 167–176. DOI: 10.1109/ICUAS54217.2022.9836067.

[13] H. Fazeli, F. Samadzadegan, and F. Dadrasjavan. "Evaluating the Potential of RTK-UAV for Automatic Point Cloud Generation in 3D Rapid Mapping". In: *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* XLI-B6 (2016), pp. 221–226. DOI: 10.5194/isprs-archives-XLI-B6-221-2016. URL: https://doi.org/10.5194/isprs-archives-XLI-B6-221-2016.

[14] Junlin Li et al. “A Survey of Indoor UAV Obstacle Avoidance Research”. In: *IEEE Access* 11 (2023), pp. 51861–51891. DOI: 10.1109/ACCESS.2023.3262668.

[15] Dipraj Debnath et al. “A Review of UAV Path-Planning Algorithms and Obstacle Avoidance Methods for Remote Sensing Applications”. In: *Remote Sensing* 16.21 (2024). ISSN: 2072-4292. DOI: 10.3390/rs16214019. URL: https://www.mdpi.com/2072-4292/16/21/4019.

[16] Vladan Papić et al. “High-Resolution Image Transmission from UAV to Ground Station for Search and Rescue Missions Planning”. In: *Applied Sciences* 11.5 (2021), p. 2105. DOI: 10.3390/app11052105. URL: https://doi.org/10.3390/app11052105.

[17] Michael Jones, Soufiene Djahel, and Kristopher Welsh. “Path-Planning for Unmanned Aerial Vehicles with Environment Complexity Considerations: A Survey”. In: *ACM Comput. Surv.* 55.11 (Feb. 2023). ISSN: 0360-0300. DOI: 10.1145/3570723. URL: https://doi.org/10.1145/3570723.

[18] Devon Wanner et al. “UAV avionics safety, certification, accidents, redundancy, integrity, and reliability: a comprehensive review and future trends”. In: *Drone Systems and Applications* 12 (2024), pp. 1–23. DOI: 10.1139/dsa-2023-0091. URL: https://doi.org/10.1139/dsa-2023-0091.

[19] Nicolaj Haarhøj Malle, Frederik Falk Nyboe, and Emad Ebeid. “Survey and evaluation of sensors for overhead cable detection using UAVs”. In: *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE. 2021, pp. 361–370. DOI: 10.1109/ICUAS51884.2021.9476724.

[20] Geetesh Dubey, Ratnesh Madaan, and Sebastian Scherer. “DROAN - Disparity-Space Representation for Obstacle Avoidance: Enabling Wire Mapping and Avoidance”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018, pp. 6311–6318. DOI: 10.1109/IROS.2018.8593499.

[21] Haoxuan Sun and Taoyang Wang. “A Stereo Disparity Map Refinement Method Without Training Based on Monocular Segmentation and Surface Normal”. In: *Remote Sensing* 17.9 (2025), p. 1587. DOI: 10.3390/rs17091587. URL: https://doi.org/10.3390/rs17091587.

[22] Raja Hamzah and Yuhanis Yusof. “Literature Survey on Stereo Vision Disparity Map Algorithms”. In: *Journal of Sensors* 2016 (2016), pp. 1–23. DOI: 10.1155/2016/8742920.

[23] Stereolabs. *ZED 2 Data Sheet*. Data Sheet.

[24] Ahmed Abdelsalam et al. “Depth accuracy analysis of the ZED 2i Stereo Camera in an indoor environment”. In: *Robotics and Autonomous Systems* 179 (2024), p. 104753. DOI: 10.1016/j.robot.2024.104753.

[25] Rochak Bajpai et al. “Middleware Architecture - History and Adaptation with IEEE 802.11”. In: *Middleware Architecture*. Ed. by Mehdia Ajana El Khaddar. London: IntechOpen, 2021. Chap. 2. DOI: 10.5772/intechopen.97124. URL: https://doi.org/10.5772/intechopen.97124.

[26] Brian Gerkey, Richard Vaughan, and Andrew Howard. “The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems”. In: *Proceedings of the International Conference on Advanced Robotics* (Aug. 2003).

[27] Giorgio Metta, Paul Fitzpatrick, and Lorenzo Natale. “YARP: Yet another robot platform”. In: *International Journal of Advanced Robotic Systems* 3 (Mar. 2006). DOI: 10.5772/5761.

[28] Noriaki Ando et al. “RT-Component Object Model in RT-Middleware—Distributed Component Middleware for RT (Robot Technology)”. In: July 2005, pp. 457–462. ISBN: 0-7803-9355-4. DOI: 10.1109/CIRA.2005.1554319.

[29] Morgan Quigley et al. “ROS: an open-source Robot Operating System”. In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe. 2009, p. 5.

[30] Andrea Bonci et al. “Robot Operating System 2 (ROS2)-Based Frameworks for Increasing Robot Autonomy: A Survey”. In: *Applied Sciences* 13.23 (2023). ISSN: 2076-3417. DOI: 10.3390/app132312796. URL: https://www.mdpi.com/2076-3417/13/23/12796.

[31] Yuya Maruyama, Shinpei Kato, and Takuya Azumi. "Exploring the performance of ROS2". In: *Proceedings of the 13th International Conference on Embedded Software*. Association for Computing Machinery, 2016. DOI: 10.1145/2968478.2968502. URL: https://doi.org/10.1145/2968478.2968502.

[32] Tobias Kronauer et al. "Latency Analysis of ROS2 Multi-Node Systems". In: *2021 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*. 2021, pp. 1–7. DOI: 10.1109/MFI52462.2021.9591166.

[33] NVIDIA Development Team. *Jetson Orin NX Series Data Sheet*. Data Sheet.

[34] Richard B Langley. "Rtk gps". In: *Gps World* 9.9 (1998), pp. 70–76.

[35] Stereolabs. *ZED ROS 2 wrapper*. Offical Github.

[36] Open Robotics. *Gazebo Sim*. Official website.

[37] Open Robotics. *Robot Operating System*. ROS official website.

[38] PX4 Development Team. *PX4 Autopilot Documentation*. Offical documentaion.

[39] Anis Koubâa et al. "Micro Air Vehicle Link (MAVlink) in a Nutshell: A Survey". In: *IEEE Access* 7 (2019), pp. 87658–87680. DOI: 10.1109/ACCESS.2019.2924410.

[40] PX4 Development Team. *MAVLink Guide*. Offical documentaion.

[41] Aerial-Core Project. *Gazebo Models and Worlds*. Github Repo.

[42] ARK-Electronics. *ROS2 PX4 Offboard Example*. Github Repo.

[43] Gazebo Sim. *ROS 2 Gazebo Bridge*. Github Repo.

[44] Eldar Rubinov et al. "Review of GNSS formats for real-time positioning". In: *Africa GEO* 2011 (2011).

[45] SEMU Counsulting. *pyrtcm: Python library for parsing RTCM3 messages*. Official Github.

[46] Nick Van Oosterwyck. "Real Time Human Robot Interactions and Speed Control of a Robotic Arm for Collaborative Operations". PhD thesis. May 2018. DOI: 10.13140/RG.2.2.28723.53286.

[47] National Geospatial-Intelligence Agency. *The Universal Grids: Universal Transverse Mercator (UTM) and Universal Polar Stereographic (UPS)*. Tech. rep. National Geospatial-Intelligence Agency, 1989. URL: https://www.nga.mil/ProductsServices/GeospatialIntelligence/TechnicalDocuments/TechnicalReports/Pages/default.aspx.

[48] Jian S. Dai. "Euler–Rodrigues formula variations, quaternion conjugation and intrinsic connections". In: *Mechanism and Machine Theory* 92 (2015), pp. 144–152. ISSN: 0094-114X. DOI: https://doi.org/10.1016/j.mechmachtheory.2015.03.004. URL: https://www.sciencedirect.com/science/article/pii/S0094114X15000415.

List of Figures

List of Tables