ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

Department of Industrial Engineering

Second Cycle Degree in
AEROSPACE ENGINEERING

Design and experimental validation of advanced guidance and navigation algorithms for Unmanned Aerial Vehicles

Academic Tutor
Prof. Fabrizio Giulietti

Student
Luca Linguerri

Hosting Institution
CIRI Aerospaziale-Aerospace

Academic Year 2024/2025

Una volta che avrete conosciuto il volo,
camminerete sulla terra guardando il cielo,
perché là siete stati e là desiderete tornare.
(Leonardo Da Vinci)

Contents

List of Figures

List of Tables

Acronyms

UAV	Unmanned Aerial Vehicle
GNSS	Global Navigation Satellite System
RTH	Return To Home
VO	Visual Odometry
SLAM	Simultaneous Localization And Mapping
RGB-D	Red Green Blue - Depth
ROS	Robot Operating System
PID	Proportional Integral Derivative
EKF	Extended Kalman Filter
ARF	Almost Ready to Fly
ESC	Electronic Speed Controller

IPS	Indoor Positioning System
IMU	Inertial Measurement Unit
GPS	Global Positioning System
UAS	Unmanned Aerial System
OS	Operative System
UDP	User Datagram Protocol

Chapter 1

Introduction

Navigation has been a fundamental challenge throughout the history of human exploration. From natural observation of the environment to early celestial navigation, from the first inertial systems to contemporary satellite-based solutions, humans have continuously and relentlessly searched for reliable methods to determine their position, orientation and motion within an environment. With the advent of (Unmanned Aerial Systems) UAVs, this challenge has reached new horizons and posed novel and increasingly complex problems: confined and crowded spaces, both outdoor and indoor, require robust and advanced algorithms to navigate safely and successfully complete missions.

Contemporary UAVs applications span from infrastructure inspection to precision agriculture, from environmental monitoring to autonomous delivery. In every scenario, to ensure correct and safe operations, the drone critically depends on accurate localization and reliable guidance strategies which, at the time of this dissertation, are mainly based on Global Navigation Satellite System (GNSS). Commercially available flight controller firmware, such as Ardupilot and PX4, require a position estimate to implement some of the most important flight algorithms such as:

- enabling the flight in "Position mode", which is the most user-friendly configuration,
- allowing the drone to enter the "RTH mode" in case of emergency,
- enabling collision prevention algorithms
- performing stable hovering and precise trajectory tracking

The reliance on GNSS for these core functionalities makes this system critical not only for mission success but, more importantly, for operational safety. As UAV operations expand into increasingly complex environments, traditional systems like inertial and satellite-based ones might not be available or sufficient anymore, as in GNSS-denied environment or in presence of obstacle not detected by satellites. The need for resilient and robust GNSS-aided or, more drastically, GNSS-independent solutions become essential, this is the reason why a fusion between traditional systems and non-traditional ones like optical or multispectral is becoming increasingly important in modern and future UAV applications.

The state of the art of visual navigation approaches in a GNSS-denied environment is represented by Visual Odometry and Simultaneous Localization and Mapping. The concept of Visual Odometry (VO) was introduced by Nistér et al. in 2004 [2], while a formal definition was given by Davide Scaramuzza et al. as "the process of estimating the egomotion (e.g., position and attitude) of an agent using only the input of a single or multiple cameras attached to it" and it aims at recovery path incrementally [3]. This last publication became the standard reference for VO and introduced the classification of Feature-based VO (methods that track features for different frames) and Direct VO (which use pixel intensity). Neistér, in its publication, presents the first schemes for Monocular and Stereo estimation. For the second, the steps are the following:

1. Match feature points between the left and right images of the stereo pair. Triangulate the observed matches into 3D points.

2. Track features for a certain number of frames.

3. Compute the pose of the stereo rig

Then it is shown how this setup provides acceptably results and precision by comparing the stereo visual odometry with GNSS and INS, testing the algorithms with a fixed-wing UAV. In Fig. 1.1 a clear illustration of monocular Feature-based VO is shown.

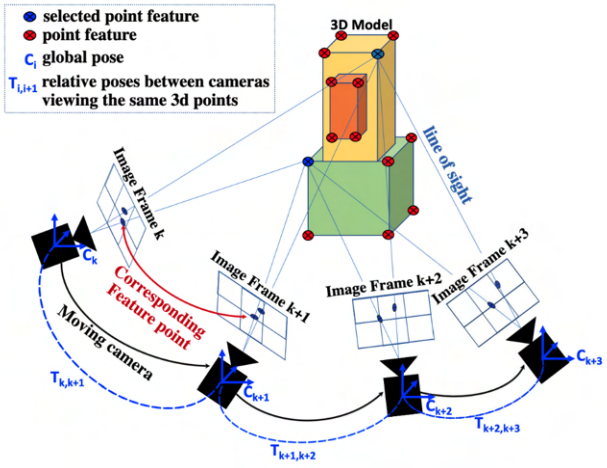


Figure 1.1: Illustration of Monocular Feature-based VO scheme [1]

Remarkable improvements for Monocular Visual Odometry were accomplished in 2014 by Forster et al. [4] developing a Semi-Direct Visual Odometry (SVO) that eliminated the need of costly feature extraction and robust matching and that, at the time of the dissertation, outperformed every other VO algorithm. In 2015 Engel et al. [5] developed a Direct Sparse Odometry (DSO) algorithm that combines the benefits of direct methods with the flexibility of sparse approaches and it was implemented into a monocular visual odometry algorithm showing its accuracy and robustness. In recent years, researchers are developing Learning-Based Visual Odometry such as NeRF-VO [6] that introduces a monocular visual odometry system that integrates

learning-based sparse visual odometry for low-latency camera tracking and a neural radiance scene representation.

The other visual navigation approach is represented by the SLAM algorithm, which stands for Simultaneous Localization And Mapping, and it is a technique for estimating simultaneously the sensor motion and reconstructing the geometrical structure of the visited area [7]. It is actually older than VO since it was firstly introduced in 1985 by Chatila and Laumond [8]. SLAM algorithm jointly estimates:

- Robot trajectory (its poses over time)
- Map of the environment

Those two elements are coupled since a better pose estimation results in a better map and viceversa. The SLAM algorithm can be approached with different type of sensors: cameras, LiDAR or a combination of the two.

1. Visual-SLAM is performed using cameras and can be decompose into:

   - Feature-Based SLAM. It is based on the detection and tracking of some important points in the image. The first Monocular approach (MonoSLAM) was proposed in 2003 [9] and further refined in 2007 [10]. At the time of writing this article the most used algorithm for SLAM is ORB-SLAM, which was introduced by Mur-Artal et al. in 2015 [11]. This approach is able to handle a vast variety of cameras like monocular, stereo or RGB-D (Red Green Blue - Depth) but it needs a high number of tuning parameter in order to correctly work in a given environment.
   - Direct SLAM. It is based on using the image without any feature detectors and descriptors. These methods are usually time consuming and require a GPU-based processing. It's interesting to notice how, generally, Visual Odometry can be considered a subset of SLAM algorithms. This is due to the use of VO for the Localization part of the SLAM problem, for example some algorithms use SVO or DSO, which were presented as purely VO methods.

   V-SLAM research is very rich and its algorithms provide very good results. Anyway, this approach is prone to errors due to its sensitivity of light changes or due to a low textured environment. Generally speaking, they perform well in indoor scenarios with a constant and well lit environment.

2. LiDAR based SLAM algorithms use a laser sensor to measure the distance from an object and are able to achieve a low drift motion estimation [12]. LiDAR sensors exist both as 2D or 3D versions and are often coupled with IMU in order to achieve better estimation results thanks to sensor fusion. Earlier algorithms used Bayes-based filter approaches, some significant implementations available as ROS packages are GMapping [13] and HectorSLAM [14]. The newer algorithms are graph-based methods and the major implementations are Cartographer [15] and KartoSLAM [16]. This last category of algorithms stores a graph of robot poses and features. Graph-based algorithms have to maintain only the pose-graph, which usually make them efficient in handling

resources, especially while building large scale maps [17]. The main solution for LiDAR-based SLAM is the scan-matching approach with graph optimization.

In this regard, this thesis firstly explores the implementation of non conventional positioning systems in order to enable indoor navigation. From an analysis of the current state of the art, a family of sensors has been selected, and ultimately one sensor was identified as the most suitable for this implementation. The drone and the sensor were modeled in the simulation environment Gazebo and consequently, extensive simulations were conducted in order to test ROS2 nodes and to analyze the SLAM algorithm performance before implement those in the real world.

This dissertation is organized as follows. Chapter 2 introduces the mathematical modelling of multirotors kinematics and dynamics, the PX4 Control Architecture and Collision Prevention algorithm and finally a description of the SLAM system. Chapter 3 presents the description of the system, it begins from the definition of UAV and UAS, it analyzes the Indoor Positioning Systems and then describes briefly Companion Computers. Chapter 4 presents the Numerical Validation, starting from the setup and workflow description and then the simulation results are shown and discussed. Chapter 5 presents the Experimental Validation and in Chapter 6 the overall results are discussed.

Chapter 2

Mathematical Model

This chapter presents the mathematical modeling of a multicopter kinematics and dynamics, and then its control system architecture using the PX4 flight control software. It is then introduced the Collision Prevention algorithm and, in conclusion, the mathematical model of SLAM is shown.

2.1 Multirotors kinematics and dynamics

In this section, the non-linear mathematical model of a multirotor is defined starting from general expressions for the kinematics and dynamics of a rigid body.

2.1.1 Reference Frames

Firstly, the following right-handed orthogonal frames of reference are introduced, these are essential in order to derive the mathematical model of the multirotor:

1. F^i - NED Frame (North - East - Down): this is an inertial frame, it is defined under the assumptions of flat and non-rotating Earth.

2. F^v - Vehicle Frame: this is a Local Vertical and Horizontal frame, its origin is located in the center of gravity of the multirotor and its axis are parallel to F^i

3. F^b - Body Frame: this frame has its origin located in the center of gravity. Its X-axis is oriented as the bow of the multicopter, the Y-axis to the right and the Z-axis points downwards.

The multirotor state is given by the following twelve state variables:

$$[\ x \quad y \quad z\ ]^T$$
$$[\ u \quad v \quad w\ ]^T$$
$$[\ \phi \quad \theta \quad \psi\ ]^T$$
$$[\ p \quad q \quad r\ ]^T$$

where, the first vector is the position of the multirotor in the inertial frame F^i and the other three are respectively its velocity, Euler angles and angular rates.

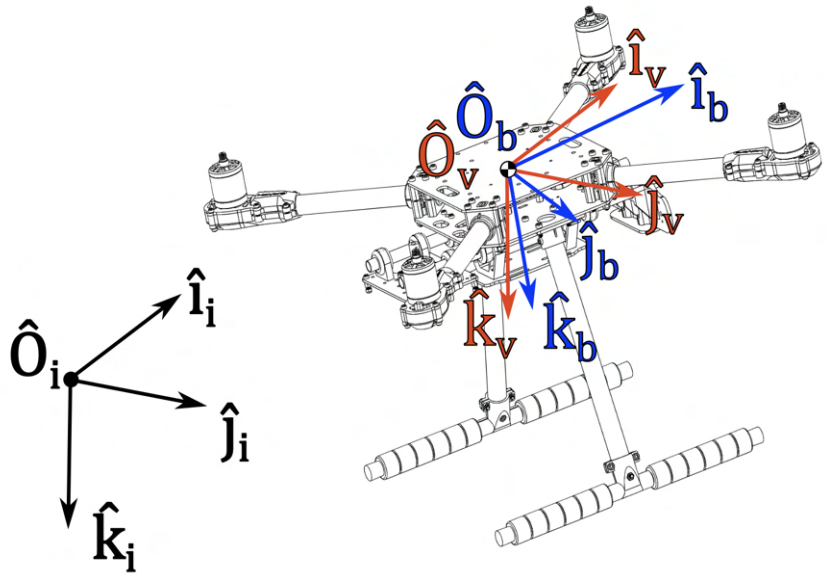


Figure 2.1: Reference Frames

2.1.2 Multirotor kinematics

The relationship existing between the position (defined in the Vehicle Frame F^v) and the velocity (defined in the Body Frame F^b) is given by the following expression:

$$\frac{d}{dt}\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R_b^v \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{2.1}$$

where R_b^v represents the rotation matrix to transform a vector from the Body Frame to the Vehicle Frame and, considering c, s as cos and sin, it's defined as follows:

$$R_b^v = \begin{bmatrix} c\theta\ c\psi & s\phi\ s\theta\ c\psi - c\phi\ s\psi & c\phi\ s\theta\ c\psi + s\phi\ s\psi \\ c\theta\ s\psi & s\phi\ s\phi\ s\psi + c\theta\ c\psi & c\phi\ s\theta\ s\psi - s\phi\ c\psi \\ -s\theta & s\phi\ c\theta & c\phi\ c\theta \end{bmatrix}$$

The relationship between the absolute angular rates $[\ \dot{\phi}\ \dot{\theta}\ \dot{\psi}\ ]$ and the Body Frame defined angular rates $[\ p\ q\ r\ ]$ is given by the following expression:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & s\phi\ t\theta & c\phi\ t\theta \\ 0 & c\phi & -s\phi \\ 0 & s\phi/c\theta & c\phi/c\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{2.2}$$

where the term t indicates the tangent.

2.1.3 Multirotor dynamics

Regarding translational motion, Newton's law can be expressed as:

$$m\ \frac{dV}{dt_i}\ =\ F$$

where m is the mass of the multirotor, V is its velocity vector, ${}^{d}/_{dt_i}$ is the time derivative in the Inertial Frame and F is the total force applied to the multirotor. This expression can be elaborated further using the Coriolis equation and it becomes:

$$m \frac{dV}{dt_i} = m \left(\frac{dV}{dt_b} + \Omega_{b/i} \times V \right) = F$$

in which ${}^{d}/_{dt_b}$ is the time derivative in the Body Fixed frame and $\Omega_{b/i} = [\, p \; q \; r \,]^T$ is the angular velocity of the multirotor with respect to the Inertial Frame F^i. This equation can be expressed as:

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} rv - qw \\ pw - ru \\ qu - pv \end{bmatrix} + \frac{1}{m} \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} \tag{2.3}$$

Regarding the rotational dynamic, Newton's law can be expressed as:

$$\frac{dh}{dt_i} = M$$

where h is the angular momentum and M is the torque applied to the multirotor. This expression can be elaborated further using the Coriolis equation and it becomes:

$$\frac{dh}{dt_i} = \frac{dh}{dt_b} + \Omega_{b/i} \times h = M$$

writing the angular momentum as $h = J\ \Omega_{b/i}$, it follows that:

$$J \frac{d\Omega_{b/i}}{dt_b} + \Omega_{b/i} \times (J\ \Omega_{b/i}) = M$$

$$\frac{d\Omega_{b/i}}{dt_b} = J^{-1} \left(M - \Omega_{b/i} \times (J\ \Omega_{b/i}) \right)$$

Assuming the multirotor as a symmetric body in all of its three axis, the inertia matrix J is diagonal and can be written as follow:

$$J = \begin{bmatrix} J_x & 0 & 0 \\ 0 & J_y & 0 \\ 0 & 0 & J_z \end{bmatrix}$$

Defining the total torque as $M = \begin{bmatrix} \tau_\phi & \tau_\theta & \tau_\psi \end{bmatrix}^T$, the last equation can be formulated in body coordinates as follows:

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \frac{J_y - J_z}{J_x}\ q\ r \\ \frac{J_z - J_x}{J_y}\ p\ r \\ \frac{J_x - J_y}{J_z}\ p\ q \end{bmatrix} + \begin{bmatrix} \frac{1}{J_x}\ \tau_\phi \\ \frac{1}{J_y}\ \tau_\theta \\ \frac{1}{J_z}\ \tau_\psi \end{bmatrix} \tag{2.4}$$

Finally, the six-degree of freedom model for the multirotor is defined for the twelve state variables and it's formulated by equations (2.1), (2.2), (2.3) and (2.4). Below, the four equations are reported again for clarity:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} c\theta\ c\psi & s\phi\ s\theta\ c\psi - c\phi\ s\psi & c\phi\ s\theta\ c\psi + s\phi\ s\psi \\ c\theta\ s\psi & s\phi\ s\phi\ s\psi + c\theta\ c\psi & c\phi\ s\theta\ s\psi - s\phi\ c\psi \\ -s\theta & s\phi\ c\theta & c\phi\ c\theta \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix}$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & s\phi\ t\theta & c\phi\ t\theta \\ 0 & c\phi & -s\phi \\ 0 & s\phi/c\theta & c\phi/c\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} rv - qw \\ pw - ru \\ qu - pv \end{bmatrix} + \frac{1}{m} \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix}$$

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \frac{J_y - J_z}{J_x}\ q\ r \\ \frac{J_z - J_x}{J_y}\ p\ r \\ \frac{J_x - J_y}{J_z}\ p\ q \end{bmatrix} + \begin{bmatrix} \frac{1}{J_x}\ \tau_\phi \\ \frac{1}{J_y}\ \tau_\theta \\ \frac{1}{J_z}\ \tau_\psi \end{bmatrix}$$

Where the forces are the sum of aerodynamical forces F_a, gravitational Force F_g and propulsive forces F_p. If the center of pressure is assumed to be coincident with the center of gravity, then the torques are only given by the propulsive forces created by the motors. As any other multirotor model, this is under-actuated, meaning that the translational speed in the horizontal plane of the NED frame must be controlled through the system dynamics by controlling its attitude.

2.2 PX4 Control Architecture

PX4 flight control algorithms rely on a standard cascaded control architecture, where the outer loop is the Position Controller and the inner one is the Attitude Controller, as shown in figure 2.2.

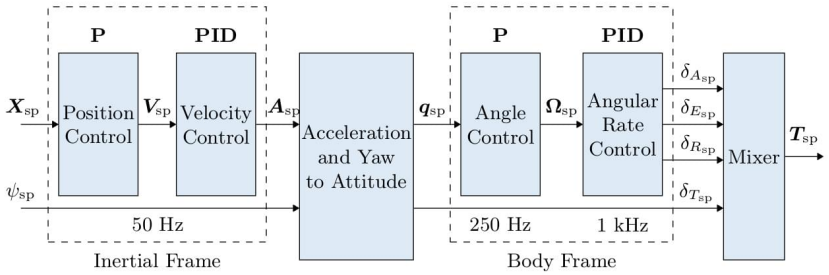


Figure 2.2: Multicopter control architecture

The core of the Attitude Controller is the Angular Rate Controller, which is composed of a Proportional Integral Derivative Controller and runs ad 1 KHz. The second most inner layer is the Angle Controller that runs at a fourth of the frequency of the first one and it's composed of a single proportional gain. The Attitude Controller is directly responsible for the control of the UAV during manual flight, where the angle setpoints are given by the pilot from the radio transmitter and are elaborated into a signal for the Motor Mixer, which provides the signal sent to the ESCs to actively control the motors angular velocity.

The outer loop is the Position Controller, which is composed of a proportional gain on the position and a PID Controller on the velocity setpoints.

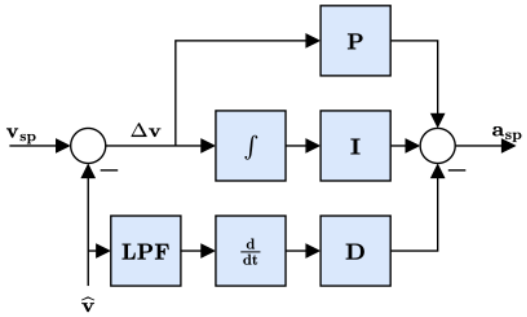


Figure 2.3: Multicopter velocity control architecture

Mathematically, the velocity Controller architecture shown in Fig. 2.3 can be modeled as follow:

$$a_{sp} \;=\; K_P\,\Delta v \;+\; K_I \int \Delta v \;+\; K_D \frac{d\hat{v}}{dt} \tag{2.5}$$

where Δv is the error between the velocity setpoint and the velocity provided by the Extended Kalman Filter (EKF) while K_P, K_I and K_D are the gains of the PID Controller. These are provided from the PX4 firmware but to improve the flight behavior they should be appropriately tuned in the first flights. It is important to notice that this architecture use the velocity error for the proportional and integral Controller, while it uses a low-pass filtered version of the measured velocity for the derivative Controller. The velocity setpoints can be provided from two sources: by the pilot via the radio Controller or by the outer loop of the position Controller, which is shown in Fig. 2.4 and can simply be modeled as:

$$v_{sp} \;=\; K_P\,\Delta r \tag{2.6}$$

where Δr is the error between the position setpoint and the position provided by the EKF $\hat{r}$. A saturator is needed in order to avoid physically unfeasible velocity setpoint when the position setpoint is at a great distance from the current UAV position.

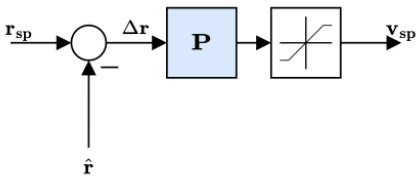


Figure 2.4: Multicopter position control architecture

It is important to note that the position reference has to come from a positioning system, that can be a satellite-based like GNSS or other non conventional systems. Moreover, from this brief description of PX4 control architecture, it is clear that the cascade structure is practical for UAVs applications, since the source of setpoints can either be the pilot, the above loop or even a flight companion computer.

2.3 Collision Prevention

Collision prevention is a natively supported feature of the PX4 flight control software. In order to enable it, the UAV must be flown in "Position Mode", where the pilot itself introduces position setpoints, and can use sensor distance data from an offboard companion computer, offboard rangefinders over MAVLink, a rangefinder attached to the flight Controller, or any combination of the above.

The vehicle restricts the current velocity in order to slow down as it gets closer to obstacles and adapts the acceleration setpoint in order to disallow collision trajectories. The data from the sensor are fused into an internal representation of 72 sectors around the vehicle, each containing distance data and timestamps. The acceleration setpoints provided by the pilot are constrained by a scale factor belonging to the interval $[-1; 1]$ as shown if Fig. 2.5 where, if the UAV is at a distance from an object greater than "Scale Distance" the acceleration is not constrain at all; if the distance is between "Scale Distance" and the user-defined parameter "CP_DIST" the acceleration is actively constrained up and if the distance is lower than "CP_DIST" the system provides a negative acceleration in order to keep the distance from obstacle.

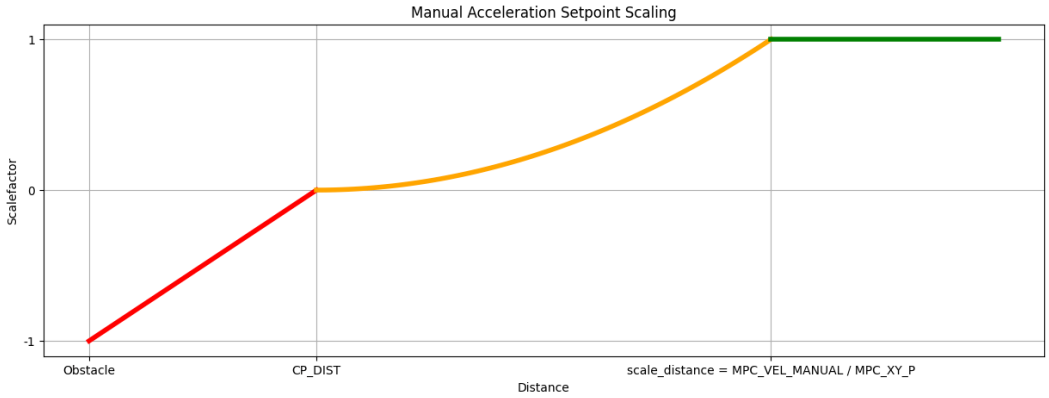


Figure 2.5: Acceleration scale factor

Mathematically, this algorithms can be described as follow, by indicating the distance from the object as x, the "Scale Distance" as μ and the parameter "CP _ DIST" as x_{min}:

$$\text{Scale Factor} = \begin{cases} 1 & \text{if } x \geq \mu \\ \frac{(x-x_{min})^2}{\mu^2} & \text{if } x_{min} < x < \mu \\ -1 + \frac{x}{x_{min}} & \text{if } x \leq x_{min} \end{cases} \tag{2.7}$$

2.4 SLAM algorithm

Formally, SLAM is described in probabilistic terminology [18]. Let us denote time by t and the robot location by x_t, the sequence of locations, or path, is given as

$$X = \{x_0,\ x_1,\ x_2,\ ..., x_T\}$$

where T is the terminal time (which could be ∞) and x_0 is the initial position, which is used as a point of reference for the estimation algorithm.

Odometry provides relative information between two consecutive locations, let us denote with u_t the odometry that characterize the motion between the time $t-1$ and t. The sequence

$$U = \{u_1,\ u_2,\ u_3,\ ..., u_T\}$$

characterize the motion of the robot. This would be sufficient to recover the poses from the initial location x_0 to the terminal location x_T, but odometry measurements are typically noisy and integration techniques diverge from the ground truth. For example, for drones the odometry is mainly given by IMUs (which are noisy) and in order to recover the position a double integration in time would be needed. This causes a large divergence in position estimate and makes flight unstable. Let denote the true map of the environment with m and let the sequence of measurement be

$$Z = \{z_1,\ z_2,\ z_3,\ ..., z_T\}$$

these are the information established by measurements between the features in m and the robot location x_t.

The SLAM problem essentially consists in recovering a model of the world m and the robot trajectory X. The literature distinguishes two main forms of the SLAM problem:

1. Full SLAM problem: $p(X, m \mid Z, U)$. It involves the estimation of the joint Ad Posteriori Probability over the entire robot path X together with the map m from the available data Z and U.

2. Online SLAM problem: $p(x_t, m \mid z_t, u_t)$. It recovers the present robot location instead of the entire path. This algorithms are usually called filters.

There exist three formulations of the SLAM problem: EKF, Particle filters and Graph-Based.

1. Extended Kalman Filter (EKF) formulation is the oldest one and propose the use of a single state vector to estimate the locations of the robot and a set of features in the environment, with an associated error covariance matrix representing the uncertainty in the estimates. It comes with a high computational cost that poses serious scaling limitations.

2. Particle filters formulation represent a posterior probability through a set of particles (which can be defined as a concrete guess). By collecting many guesses into a set of particles, the particle filter approximates the posterior distribution. This allows the SLAM to be computationally efficient but the number of necessary particles can grow quickly.

3. Graph-Based formulations solves the slam problem through nonlinear sparse optimization. Landmarks and robot locations can be thought as nodes in a graph: each consecutive pair of location is tied together by the information provided by odometry U and there exist also connections between the nodes that correspond to the locations and the landmarks. Considering to the

graph as a spring-mass model, computing the SLAM solution is equivalent to computing the state of minimal energy of the model. These methods have become the standard for building large-scale and incremental maps.

Chapter 3

System Description

This chapter presents the system architecture selected for the purpose of this thesis. The proposed platform consists of a Holybro X500 V2 quadcopter equipped with a Pixhawk 6C flight controller, a 2D LiDAR sensor and a Raspberry Pi serving as the onboard companion computer. In the following sections, each selected component will be presented and its main characteristics analyzed.

3.1 UAV

The chosen UAV platform is the Holybro X500 V2, which is an Almost Ready to Fly (ARF) quadcopter, lightweight and robust, thanks to a large use of carbon fiber both in its arms and main plates. A particularly interesting feature for this application is that the main plate of the frame has mounting holes for popular companion computers such as the Raspberry Pi and Jetson Nano, thus eliminating the need to develop a custom made support platform. The constituting components of the quadcopter are the following:

1. Frame - It provides structural strength and acts as the mounting platform for all the other components. It's made with full carbon fiber twill and it has carbon fiber tube arms supported by fiber-reinforced nylon connectors. The X500 main plate is modular and presents multiple holes that are useful to attach different elements and platforms.

2. Motors - They provide the angular velocity to the propellers in order to create lift and control the aircraft, and they are governed by the flight controller through the ESC. They were included in the ARF kit from Holybro and they are brushless triphase motor with a 920KV (namely 920 rotations per minute per Volt applied).

3. ESC - The Electronic Speed Controller is the element that generates a three-phase pulse-width-modulated voltage to regulate the current and thus control motor torque and speed.

4. Battery - It provides power to the system, both for the propulsion and the avionics systems.

5. Sensors - they can be included in the flight controller such as the IMU, barometer and compass or can be external like the GNSS receiver.

6. Data Link - there are two main data links mounted onboard, the pilot radio receiver and the telemetry receiver/transmitter. The first one receives radio signal from the pilot transmitter, while the second one is capable of transmitting and receiving data regarding the aircraft attitude, position and status.

3.1.1 Flight Controller

The Flight Controller is the core component of the UAV and thus it needs a deeper analysis. It's a small computer mounted onboard that manages all the flight-related aspects such as:

- Reading and filtering sensors data (IMU, GNSS, barometers and compass). In particular, sensor fusion is achieved thanks to an Extended Kalman Filter that takes as inputs all the different sensors readings and gives as outputs filtered and precise information about the aircraft attitude and position in the space.

- Managing flight tasks, from the simplest ones like maintaining the UAV position, to the most complicated ones such as performing autonomous missions.

- Managing input from external sources like radio-frequency transmitted data from the pilot and providing the pulse-width-modulated signal to the motor in order to control the UAV motion in the space.

The most famous and popular flight controllers are from the Pixhawk family which, together with the CubePilot family, hold a substantial market share. Pixhawk was born as a student project at ETH Zurich but it rapidly grew into the great project that it is today, influencing the whole drone industry by offering open source standards. The newer Pixhawk family is represented by the Pixhawk 6C and 6X and their main characteristics are shown in Table 3.1

	Pixhawk 6C	**Pixhawk 6X**
CPU	STM32H743	STM32H753
Clock Speed	480 MHz	480 MHz
IMU	Double Redundancy ICM-42688-P; BMI055	Triple Redundancy 3x ICM-45686
Barometer	No Redundancy MS5611	Double Redundancy ICP20100; BMP388
UART	8	7

Table 3.1: Pixhawk 6C and 6X characteristics

They both provide modularity and ease-of-use thanks to deep and well-written documentation, but it is clear that the Pixhawk 6X provides more redundancy for critical components and provides more processing power. However, the 6C is more cost effective since it's being sold at just 169€ against the 304€ of the 6X. For the purposes of this thesis, which is developing a reliable and robust indoor navigation system using non conventional sensors, the Pixhawk 6C was eventually selected due to its lower cost and lower INS precision. The latter implies a more challenging environment and thus the need for a greater effort in order to develop a reliable system.

3.2 UAS

Modern UAVs are not anymore just wooden aircrafts simply translating pilot inputs to surfaces deflections: they are becoming something more advanced and complex, to such an extent that it can be introduced the term Unmanned Aerial System (UAS), which comprehend not only the aircraft itself but also the Control and Monitoring Unit (CMU). This last component is described as the "equipment to control unmanned aircraft remotely" in the EU Regulation 2018/1139 [19] and comprehends "any instrument, equipment, mechanism, apparatus, appurtenance, software or accessory that is necessary for the safe operation of an unmanned aircraft, which is not a part, and which is not carried on board of that unmanned aircraft". The most common items included above are shown in Fig. 3.1 and are listed below:

1. UAV - the aircraft itself, as described in the previous section.

2. Radio transmitter - it's the source of pilot inputs, it translates potentiometers readings into radio waves at frequency usually of 2.4 GHz that are received by the UAV thanks to a linked receiver mounted on board and connected to the flight controller.

3. Ground station - it's the only device that provides a two-way communication with the UAV, it receives telemetry data like attitude, position and vehicle status and it is capable to send back commands via the protocol MAVLink. This communication relies on small transmitters and receivers hardware called "Telemetry Radio", one is connected to the ground station and one is mounted onboard, they need to operate at the same frequency which is dependent on the country regulations, for Italy they transmit and 433 MHz with a power output of 10mW. The Ground station is usually a portable device like a computer and uses a ground station software like QGControl or Mission Planner.

3.3 Flight Control Software

In order to manage all the hardware components mentioned above and the complex tasks that modern UAV are required to perform, it would be virtually impossible to rewrite the entire code each time even for the slightest difference in the configuration. It is for this reason that in the past decade, different open source solutions were developed but only two of those conquered almost entirely the space of academic

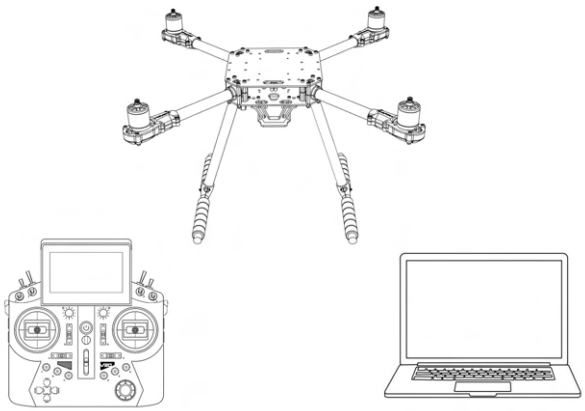


Figure 3.1: UAS items

research and commercial applications: PX4 and ArduPilot. From an user perspective they are highly comparable since they share a substantial number of functionalities and have similar performances, but a more in-depth analysis reveals that their architecture, design philosophy and community ecosystems are different. PX4 is distinguished by its modular and layered architecture: it separates sensor drivers, state estimation, control, and mission logic into discrete modules that communicate via well-defined publisher-subscriber middleware (uORB), enabling flexibility and easier maintenance. On the other hand, ArduPilot uses a more monolithic architecture where scheduling, control loops, and peripherals are tightly integrated within a central event-driven framework, therefore extensibility can be more challenging. Both support deep integration with the framework ROS2 and communication with companion computers. In this thesis, PX4 was selected thanks to its modular architecture and wide adoption, analyzing the ROS 2-PX4 architecture in detail, we can see in Fig.3.2 how it provides a deep integration between ROS 2 and PX4, allowing ROS 2 subscribers or publisher nodes to interface directly with PX4 uORB topics using the uXRCE-DDS middleware.

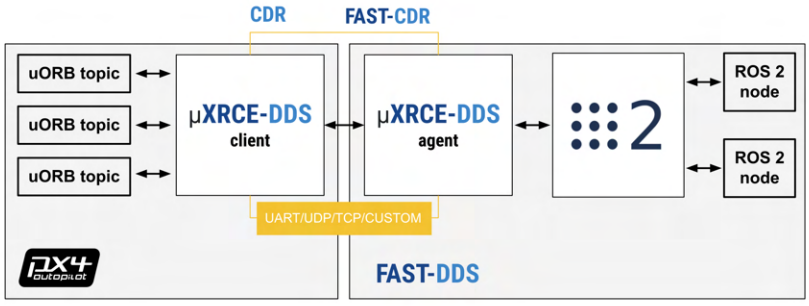


Figure 3.2: XRCE-DDS architecture

3.4 Indoor Positioning System

The state of the art of GNSS-free indoor positioning system is represented by a plethora of different solutions that can be classified based on three primary and orthogonal criteria [20].

1. Type of signal
    - Radio Frequency (RF) signals
    - Light (visible and infrared)
    - Sound (audible and infrared)
    - Magnetic Field
2. Configuration
    - Active, the portable device carried by the user generates a signal that is received by the infrastructure.
    - Passive, the mobile device receives the signal and calculates its position locally.
3. Presence of Intentionally Embedded Symbolic Information
    - With Embedded Information, it contains an intentionally embedded pattern of symbolic information, which is generated in the signal source and then reconstructed at the receiving end.
    - Without Embedded Information, does not contain pattern of symbolic information.

Since these three classification criteria are orthogonal it is possible to visualize the classification space as a cube. In the following sections some significant example are given to clarify this classification and to introduce the systems taken into account in this thesis.

3.4.1 Drone Deployable IPS

In order to choose which system to implement, it is important to deeply analyze all the advantages and disadvantages of the most common IPS functional for drones.

1. Inertial Technology. It's one of the oldest navigation methods and it's also called "dead reckoning". It estimates a future position given an initial one, a speed and a direction. The main problem of this technology is the accumulation of errors, a small error in direction for example, can be translated into a huge error as more distance is traveled. Modern systems use digital accelerometers and gyroscopes and combine this information with other sensors and then are filtered with a Kalman Filter to improve estimation. For example, the flight controller PixHawk 6X has three IMU, two barometers and a magnetometer. However, using only these integrated sensors is not sufficient to provide a reliable and stable position estimate due to the high drift errors that comes from the triple integration of a noisy acceleration measurement.

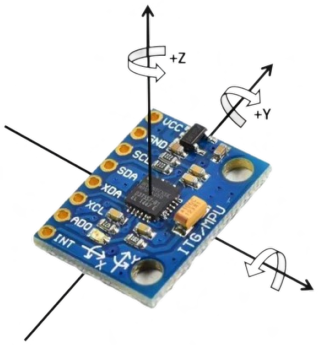


Figure 3.3: MPU6050 IMU

2. MoCap (Motion Capture). This system collects information from cameras and implements image processing techniques to track objects in a defined space. This is part of the "active" category because the infrastructure receives the light reflection of the object. Moreover, it does not use embedded information. This system is highly accurate, reliable and it does not add weight to the drone (just some infrared markers for some system) because the computations are done externally and it does not need processors onboard, which also increase the flying time. However, its use is not feasible in all scenarios, as it relies on a dedicated infrastructure equipped with calibrated cameras, this means that is not possible to be used, for example, for inspections of industrial facilities without a previous installations of the camera system and precludes a vast number of exploration mission. Another disadvantage is the high cost, but an article from 2024 [21] proposes a solution for this problem, developing and comparing a low-cost and open-source MoCap to different solutions.

3. Computer Vision. This is a passive system that uses cameras mounted directly onto the drone and can use the approaches of Visual Odometry or V-SLAM to update its position. VO goal is to only estimate the position while V-SLAM, which stands for Visual-SLAM, aims to obtain a global, consistent estimate of the robot path. This implies keeping a track of a map of the environment because it is needed to realize when the robot returns to a previously visited area. There are two main type of sensors: RGB and RGB-D, that can be used in a Stereo or Monocular Camera configurations. The first one uses two cameras mounted with an angle with respect to each other, this enables the possibility to retrieve information even in the depth direction and provides a three dimensional reconstruction. The second type uses a single camera and different algorithms are needed to reconstruct the 3D environment. It is possible to find cameras with IR projectors that emits a pseudo-random dot pattern that increase accuracy both in texture-less and low light environment. A great example of this technology is the Intel Real Sense D435 shown in Fig. 3.4

   Related works can be divided into three categories: feature-based methods,

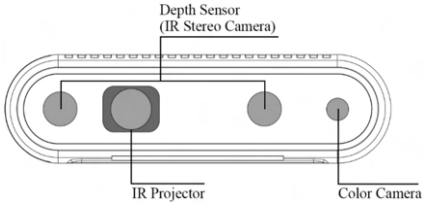


Figure 3.4: Intel Real Sense D435 Stereo Camera

appearance-based methods, and hybrid methods. Feature-based methods are based on salient and repeatable features that are tracked over the frames; appearance-based methods use the intensity information of all the pixels in the image or subregions of it; and hybrid methods use a combination of the previous two.

4. Optical Flow. It is a system composed of a downward facing camera and a distance sensor. Most of the products available on the market use the same sensors of gaming mice like the PMW3901 and the working principle is similar: the optical sensor takes photos at a fixed frequency and compares the difference between two adjacent frames, extrapolating the velocity based also on the vertical distance and the angle of the UAV. There are different solutions readily available on the market, spanning from low-cost to high-end options. In optimal conditions of light and floor patterns, the results of position estimate are quite promising for the low-cost sensor used [22]. However, the main disadvantage is that the floor needs to present irregular patterns and shapes to increase the reliability of this algorithm.

5. LiDAR. It is the acronym for Light Detection And Ranging and this sensors measures distances D by emitting laser pulses and recording the time it takes for the reflected light to return to the sensor [23].

$$D = \frac{C\,t}{2}$$

There are three main types: 1D, 2D or 3D. The first one detects the distance on a straight line, which is useful to increase the accuracy of vertical distance estimation, this sensors can be found readily available on the market at a highly affordable price and directly supported by the mains autopilot firmware. The second category uses a mono directional LiDAR sensor mounted on a rotating platform that allows to scan the distance along the plane perpendicular to its axis of rotation. The products present in the market spans from a hundred euros to much more, depending on the sample rate, rotation rate and resolution. Some worth mentioning products are the Slamtech RPLiDAR "A" family, which are triangulating low-cost LiDAR sensors with a vast choice of technical characteristics. In particular, the RPLidar A1 is a 2D omni directional sensor, with a sample rate up to 8000 samples per seconds, 12 meters of maximum range and a scan rate of 10 Hz. These characteristics are quite remarkable for the commercial price, which is about 100 euros. SLAM algorithms can be implemented using this sensors and the results are very promising.

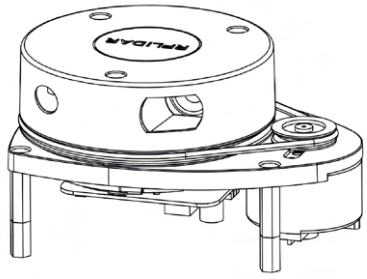


Figure 3.5: RPLiDAR A1M8

3.4.2 Assessment and Selection of Indoor Positioning Technologies

The assessment of candidate sensor was based on a set of requirements tailored to the operational constraints of the UAV and its intended application scenarios. In particular, the following selection criteria were defined:

1. Reliability in challenging and unknown environments. The system must maintain consistent performance even in adverse conditions such as partially flooded or with limited lighting environments. Since the possible applications of this UAV are exploration and inspection, special emphasis was placed on this criterion also due to the recent floods that hit the region Emilia-Romagna.

2. Cost effectiveness. In term of affordability it was thought that the need for a scalable and accessible solution was preferable, to ensure feasibility for a widespread deployment on the market.

3. Omnidirectionality. This is optional but preferable in order to improve the mapping and localization performance across the whole plane, independently of the navigation direction

Selection Criteria	**MoCap**	**Camera**	**Optical Flow**	**LiDAR**
1) Reliability	–	+	–	+
2) Cost effectiveness	–	+	+	+
3) Omnidirectionality	+	–	+	+

Table 3.2: Assessment of sensor based on the defined selection criteria

Each family of sensors has been subjected to these three criteria and the table 3.2 was compiled, showing that the LiDAR sensor family is the most suitable for the applications developed in this thesis. Among these, it was chosen to use the Slamtech RPLiDAR A1M8 sensor, since it was the one with the best cost-effectivness factor and the company directly provides the ROS and ROS2 directories to test and implement distance sensing, accelerating the first part of this work and allowing to focus more on the development of the positioning and navigation algorithms

This sensor is connected to a Raspberry Pi5, which is a compact but powerful computer that serves as Companion Computer to the PX4 software.

3.5 Companion Computer

Companion computers, are separate on-vehicle computers that are connected to the flight controller via fast serial or Ethernet link, typically communicating using uXRCE-DDS, they enable computationally expensive features. While the flight controller manages the core flight tasks and provides safety features, the companion computer, usually running Linux, is a much better platform for "general" software development and it comunicates to the ground station via a radio or a Wi-Fi link as shown in Fig.3.6.

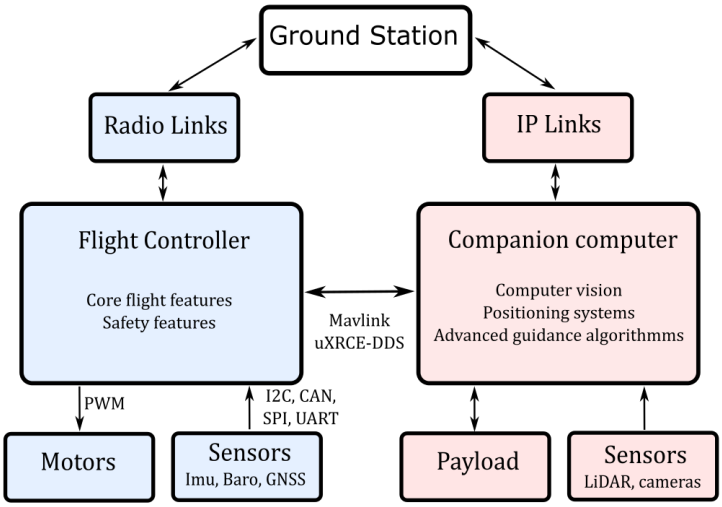


Figure 3.6: Companion Computer architecture

PX4 supports a high number of companion computer options, but they can be divided into two main categories:

1. Pixhawk autopilot bus carrier boards with companion computers, which integrates directly the companion computer into the flight controller board like the Pixhawk RPi CM4 shown in Fig. 3.7 or the Pixhawk Jetson board.

Figure 3.7: Pixhawk RPi CM4 flight controller

2. Separated companion computers, which need to be physically connected over a serial port or Ethernet and they are configured to communicate with MAVLink or uXRCE-DDS Middleware. The most famous high power example is the NVidia Jetson TX2, but the most popular is definitely the Raspberry Pi family, which is a much lower powered companion computer but with an affordable price. At the time of this dissertation, the newer product is the Raspberry Pi5 shown in Fig. 3.8. For this thesis the Pi5 was chosen over the Pi4 because it features a new CPU that make it three time faster with respect to the previous generation. Such a great performance leap is also due to the updated RAM, which arrives up to 16GB, even if the 8GB version was eventually chosen for this work.

Figure 3.8: Raspberry Pi5

Chapter 4

Simulation Environment

Simulation plays a fundamental role in the development and validation of advanced guidance and navigation algorithms, providing a controlled, repeatable, and safe environment in which system behavior can be analyzed under a wide range of operating conditions before deploying on hardware. The simulation environment enables the integration of the flight control stack, middleware and sensor models, allowing the complete system to be tested as a whole before experimental validation. It is possible to model and modify sensor characteristics such as noise, resolution, and update rate to replicate exactly the real behavior and analyze the robustness of the algorithms. Furthermore, simulations enable extensive repeatability and scalability. Identical scenarios can be executed multiple times to ensure consistent results, reducing development time and minimizing the risks associated with early-stage testing on physical platforms.

In this chapter, the setup will be firstly analyzed, then the workflow will be shown and the results of hover and navigation will be finally presented.

4.1 Setup

The simulations were performed on a PC using Ubuntu 22.04.5 LTS (Jammy Jellyfish) as the operative system. This is an open-source OS under the GNU General Public License, it is the standard and recommended system for PX4-ROS2 algorithms development.

4.1.1 Simulator

The simulator used is Gazebo, this is an open-source robotics simulator that began development in 2002. This environment allows the developed algorithms and the flight stack to interact with the physics and dynamics of the drone. Since it's open-source, there is a great work of the community and different drones, worlds and sensors models are ready to use. This thesis uses the following models:

- World model. Initially, a ready-to-use world called "walls" from gazebo was used. Then, a custom world was developed and it is shown in Fig. 4.1. It's made of three zones, each representing different challenges and tasks: the first one, zone A, is made of tight and crowded spaces, it was created in order to test

the algorithm in presence of multiple features and to analyze the computational cost in such environment. The second zone (B) represents a long and narrow aisle, testing position algorithm at the limit of the LiDAR range. The third zone (C) represents a large and empty room, and it is used to test navigation around right angles and to create the right conditions to initiate the loop closure feature.

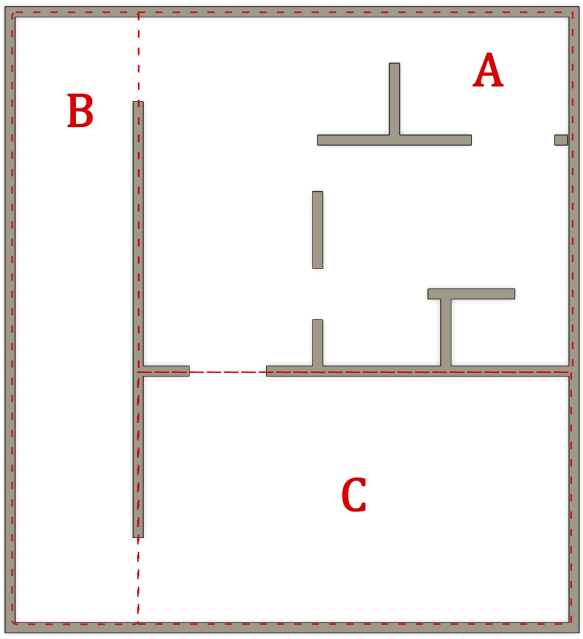


Figure 4.1: Custom made Gazebo world

- UAV model. The PX4-Gazebo interface already contains the Holybro X500 V2 model in the form of a SDF file, which is maintained by Farhang Naderi and perfectly replicates the real drone structure (Fig.4.2). Some differences can be found in the sensors, there is just one modeled IMU while the Pixhawk 6C has a double redundancy. In order to increase accuracy, the IMU noises of the model were changed to the exact one of the Pixhawk 6C. Similarly, it was done for the barometer and for the internal compass. Moreover, the model contains a GNSS module that was removed from the file to ensure that the position reference is only the one given by the SLAM algorithm.

- 2D LiDAR model. A generic 2D LiDAR sensor was already present in the PX4-Gazebo interface. This SDF file was taken and modified to match the characteristic of the SLamtech A1M8 sensor. In particular, the maximum range was set to 12 meters, the rotational speed was set to $10Hz$ and, given a sample rate of 4000 samples per second, the number of samples per rotation was set to 400.

- 1D LiDAR model. Similarly to what was done with the 2D LiDAR, a sensor model was already configured and its file was modified to match the character-

Figure 4.2: Holybro X500 V2 modeled in Gazebo

istic of the Benewake TF-mini LiDAR. Then, it was added to the X500 file setting the right position and orientation.

Finally, in Fig. 4.3 the modified drone, called "X500_lidar_2d_down", is shown flying in the custom made environment. To visualize the LiDAR data, a plugin was used and the blue points visible in the figure represent the samples taken by the sensor.

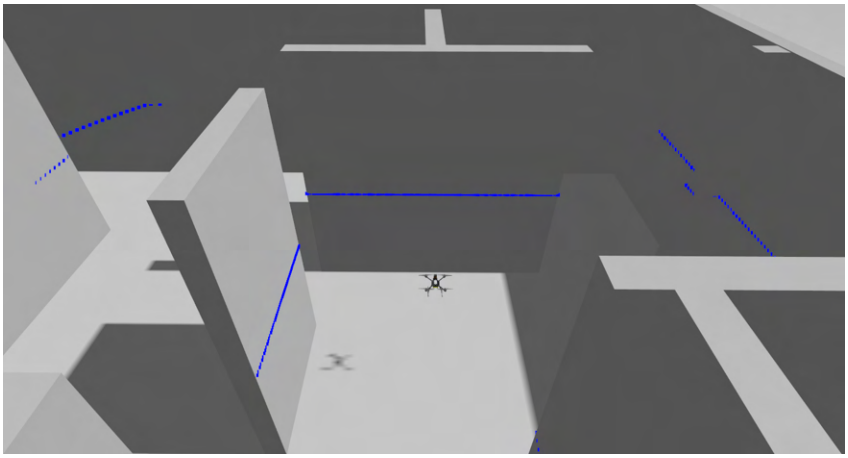


Figure 4.3: X500 modified model flying in the custom made world

4.1.2 PX4

PX4 setup was straightforward since the documentation is particularly deep and coherent. Firstly, it is necessary to clone locally the GitHub directory and then it is possible to start the simulations with the terminal command:

```
make px4_sitl gz_x500
```

Where `sitl` means Software In The Loop and it's the target, `gz` stands for Gazebo and it expresses the simulator version and `X500` expresses the UAV model. In order to simulate the custom world with the modified X500 drone the command becomes:

```
make px4_sitl gz_x500_lidar_2d_down_slam
```

where "slam" is the name given to the custom world model.

After starting the simulation, the UAV behave exactly like in reality, it is in fact possible to use a ground station software like QGroundControl to modify the PX4 parameters, analyze transmitted data and download flight logs. After removing the GNSS sensor from the model file, its related parameters had to be changed in order not to activate the startup failsafe and to set the EKF algorithm to not rely on those data. Then, the parameters related to the vertical distance sensor were activated, including its type, its range and the distance from the ground. Finally, the native obstacle avoidance algorithm was activated with the following parameters:

```
CP_DIST = 1
CP_GO_NO_DATA = disable
```

where the first parameter, `CP_DIST` indicates the minimum distance in meters from an obstacle that the UAV can achieve, as already examined in the collision prevention section 2.3. The second parameter doesn't allow the UAV to fly in the directions where the distance sensor is not active or it doesn't give coherent measurements, this is a safety features that avoid hitting objects if the LiDAR sensor is not completely 2D but have a limited visual angle.

4.1.3 ROS2

ROS2 is a middleware framework designed to support the development of modular, scalable and distributed robotic applications. It provides standardized communication mechanisms, including publish–subscribe messaging, services, and actions, that enable efficient data exchange between software components. In the context of this work, ROS 2 serves as the integration layer between the PX4 flight control stack, the LiDAR sensor, the SLAM algorithm and the Gazebo environment for the simulations. The communication between ROS2 and PX4 is done via the uXRCE-DDS protocol, where the client is PX4 and the agent is the Companion Computer, the Raspberry in this case. In simulations the client is started automatically with the `make` command, while the agent has to be manually started from the terminal command:

```
MicroXRCEAgent udp4 -p 8888
```

where `8888` is the UDP port of the local host.

This step allows the uORB messages to be displayed as ROS2 topics and they can be controlled using the ROS2 CLI or can be manipulated by ROS2 nodes. They are divided into input and output topics with respect to PX4 software. In this way if a ROS2 node needs to read the Odometry uORB message, it needs to subscribe to `/fmu/out/vehicle_odometry` topic. While if the node wants to publish a uORB message to PX4 it needs to publish in `/fmu/in/` ROS2 topic.

4.1.4 ROS2 frames

In ROS 2, frames are reference coordinate systems used to describe the position and orientation (pose) of sensors, robots, and objects in space. With the ROS REP105 "Coordinate Frames for Mobile Platforms" created in 2010, a convention for frames was developed and the frames used in this work are:

- map - This coordinate frame is a world fixed frame, with its Z-axis pointing upwards. The pose of a mobile platform, relative to the map frame, should not significantly drift over time. The map frame is not continuous, meaning the pose of a mobile platform in the map frame can change in discrete jumps at any time. In a typical setup, a localization component constantly recomputes the robot pose in the map frame based on sensor observations, therefore eliminating drift, but causing discrete jumps when new sensor information arrives. The map frame is useful as a long-term global reference.
- odom - This coordinate frame is a world-fixed frame. The pose of a mobile platform in the odom frame can drift over time, without any bounds. This drift makes the odom frame useless as a long-term global reference. However, the pose of a robot in the odom frame is guaranteed to be continuous, meaning that the pose of a mobile platform in the odom frame always evolves in a smooth way, without discrete jumps. In a typical setup the odom frame is computed based on an odometry source, such inertial measurement unit. The odom frame is useful as an accurate, short-term local reference, but drift makes it a poor frame for long-term reference.
- base_link - This coordinate frame is rigidly attached to the mobile robot base. The base_link can be attached to the base in any arbitrary position or orientation.
- laser_link - This coordinate frame is rigidly attached to the LiDAR sensor and consequently it moves with the base_link frame.

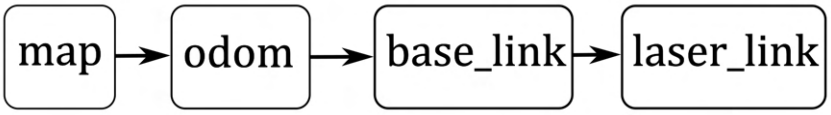


Figure 4.4: ROS2 frames

ROS2 manages relationships between frames using the TF2 library. TF2 maintains a time-stamped tree of coordinate transformations, allowing the system to convert data from one frame to another in a consistent and synchronized way. This is essential in robotic systems where multiple sensors, actuators, and estimation algorithms operate in different coordinate systems. For example, to rigidly connect the laser_link to the base_link the TF2 static transform publisher function was used, which defined the spatial distance and angles between a parent frame (base_link) and a child frame (laser_link).

It is also important to note that the PX4 and ROS2 frames conventions are different, as shown in table 4.1. For this reason, it is necessary to take care of frame conventions when publishing and subscribing to uORB or ROS2 topic. A conversion library is provided directly in PX4 firmware, assisting the development and avoiding mistakes.

Frame	PX4	ROS2
Body	FRD (X Forward, Y Right, Z Down)	FLU (X Forward, Y Left, Z Up)
World	NED (X North, Y East, Z Down)	ENU (X East, Y North, Z Up)

Table 4.1: PX4 and ROS2 frames convention

4.2 Workflow

The problem addressed in this work is the navigation in GNSS-denied environments, this means that the UAV is only relying on inertial measurements, and consequently it relates its pose with respect to the odom frame. This is totally fine for a manual flight, since the need for position reference is not required and the drone just control its angles and angular velocities. This situation can be represented with the following scheme in Fig. 4.5

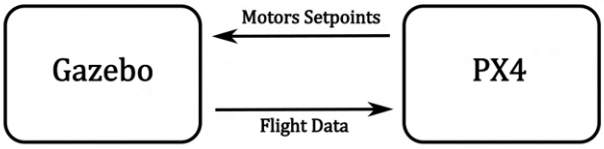


Figure 4.5: PX4-Gazebo environment

Where there is a simple exchange of information between Gazebo and PX4: Gazebo simulates the environment and gives to PX4 sensors data (IMU, barometer and magnetometer), while PX4 generates the motor signals, which are taken by the Gazebo motors model and the simulator computes the UAV physics.

In order to restore the map frame, the SLAM algorithm is needed.

4.2.1 SLAM node

It was chosen to use the ROS2 "slam_toolbox" developed by Steve Macenski, it is a 2D graph-based SLAM algorithm that is able to estimates the drone's pose while incrementally building an occupancy grid map of the environment. The package supports both online and offline mapping, loop closure detection, and map optimization, making it suitable for long-term and repeatable experiments. It also offers a RVIZ plugin that allow to save and reuse the resulting SLAM maps. The package generates a ROS node that subscribes to laser scan and odometry topics, and publishes map-to-odom transform and a map. The laser scan data has to be

published in the `/scan` topic, this information comes from Gazebo and are translated from a Gazebo topic to a ROS2 topic via a Gazebo-to-ROS2 bridge with the package ros_gz_bridge that allows data to be exchanged between the two systems. The odometry topic comes from PX4 and a custom node has been developed to translate PX4 odometry signal into a valid ROS2 tf topic, which will be analyzed later. The map-to-odom transform published by the SLAM node needs to be translated into a PX4 uORB message and hence one other custom made node has been developed.

4.2.2 dronepose node

The dronepose node is developed to convert odometry data produced by the PX4 flight control system into a ROS2–compatible message using the TF2 framework. The node subscribes to the `/fmu/out/vehicle_odometry topic`, which provides position and orientation estimates of the vehicle expressed in the PX4 reference frame. To ensure adaptability across different simulation and deployment scenarios, the parent and child frames of the transformation are defined as ROS2 parameters (odom and base_link), allowing the reference frame hierarchy to be reconfigured without modifying the source code, this parameter can be retrived and modified simply with the ROS2 CLI. Communication with the PX4 middleware is configured through customized Quality of Service (QoS) profiles, where best-effort reliability and volatile durability are selected for the subscription in order to prioritize low latency and compatibility with the PX4 message stream. Upon reception of a new odometry message, the node constructs a TransformStamped message containing both translational and rotational components of the vehicle pose. The timestamp of the transform is generated using the ROS 2 clock to ensure temporal consistency with other system components, particularly in simulation environments where simulated time may be employed. The position vector is extracted from the odometry message and modified to convert from PX4's North–East–Down (NED) coordinate convention to ROS2's East–North–Up (ENU) convention. Additionally, a constant offset is applied along the vertical axis to account for the physical displacement between the vehicle's reference point and the mounted LiDAR sensor. Orientation data, provided as a quaternion, is similarly transformed to maintain consistency between the two coordinate systems. Finally, the resulting transform is broadcast using the TF2 TransformBroadcaster, enabling other ROS2 nodes to correctly interpret the drone pose within the odom reference frame.

4.2.3 pose2odom node

The pose2odom node acts as a critical interface between ROS2 localization and the PX4 state estimation pipeline, it is designed to convert pose estimates produced within the ROS2 SLAM algorithm into a format compatible with the PX4 flight control system, enabling the use of external localization sources for vehicle state estimation. In particular, the node subscribes to a `PoseWithCovarianceStamped` message published on the `/pose` topic. This information is then repackaged and published as a `VehicleOdometry` message on the `fmu/in/vehicle_visual_odometry` topic, which PX4 uses as an input for its estimator. It was used this topic, even if it's not a Visual Odometry algorithm, because PX4 firmware does not differentiate between the pose input and this was the only available topic.

Communication parameters are explicitly configured through custom Quality of Service (QoS) profiles to ensure compatibility with PX4. The node operates at a fixed publishing frequency of 30 Hz defined by a ROS2 timer.

The node maintains the most recent pose estimate in an internal state variable, which is updated asynchronously upon reception of new messages from the `/pose` topic. At each timer callback, a new `VehicleOdometry` message is constructed. The message timestamp is generated using the ROS2 clock and converted to microseconds, as required by the PX4 message definition, ensuring temporal consistency within the PX4 estimator.

Position and orientation data are extracted from the received pose message and are initially expressed in the ROS2 East–North–Up (ENU) coordinate convention. Since PX4 uses a North–East–Down (NED) reference frame, coordinate transformations are applied. The transformed position and orientation are then assigned to the appropriate fields of the `VehicleOdometry` message.

In addition to pose information, the node propagates uncertainty information from the SLAM output to PX4. The covariance matrix contained in the `PoseWithCovarianceStamped` message, which is a $[36 \times 36]$ matrix, is used to extract positional variances along the three principal axes. These values are mapped directly to the `position\_variance` field of the PX4 odometry message. This step is particularly important for estimator fusion, as it enables PX4 to weight the external pose measurements according to their estimated reliability. It is also possible to define noise characteristics as PX4 parameters, even if it showed acceptable results, it is better to use the SLAM-derived variance. Moreover, velocity and angular velocity fields are set to zero, reflecting the absence of direct velocity estimates from the SLAM algorithm.

The overall workflow can be visualized as in Fig. 4.6

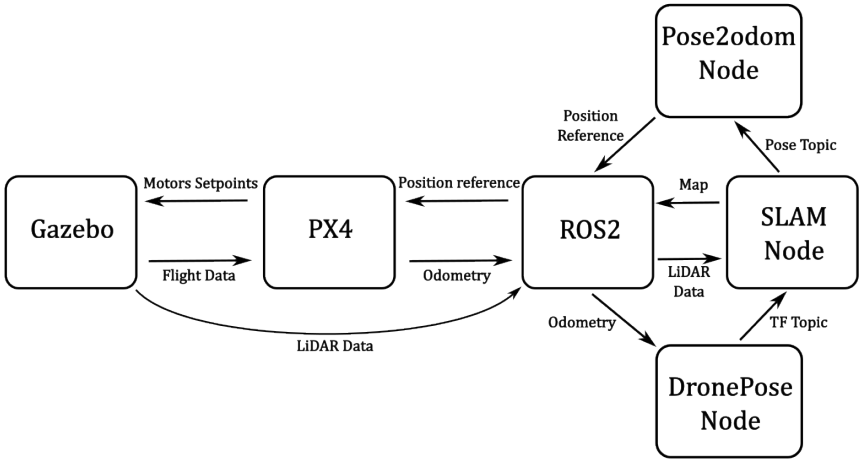


Figure 4.6: Overall simulation environment workflow

4.3 Hover result

In this section, the hover results are shown and analyzed. The three main parameters that are possible to tune in the ROS2 environment are the following:

- LiDAR Sample Rate
- SLAM mapping resolution
- SLAM optimization mode

Consequently, there are different combinations of parameters that need to be analyzed and studied. The first simulated flight was performed with:

- Sample Rate: 400 $Samples/rev$
- Resolution: 0.05
- Mode: Localization

Which resulted in a crash. From Fig. 4.7 is it possible to see the overall behavior during the flight.

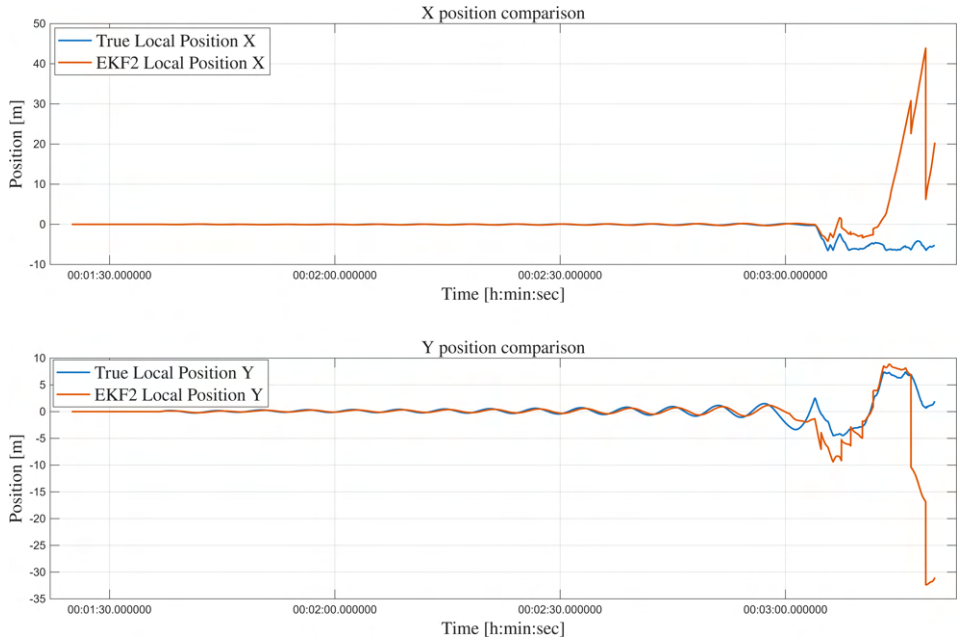
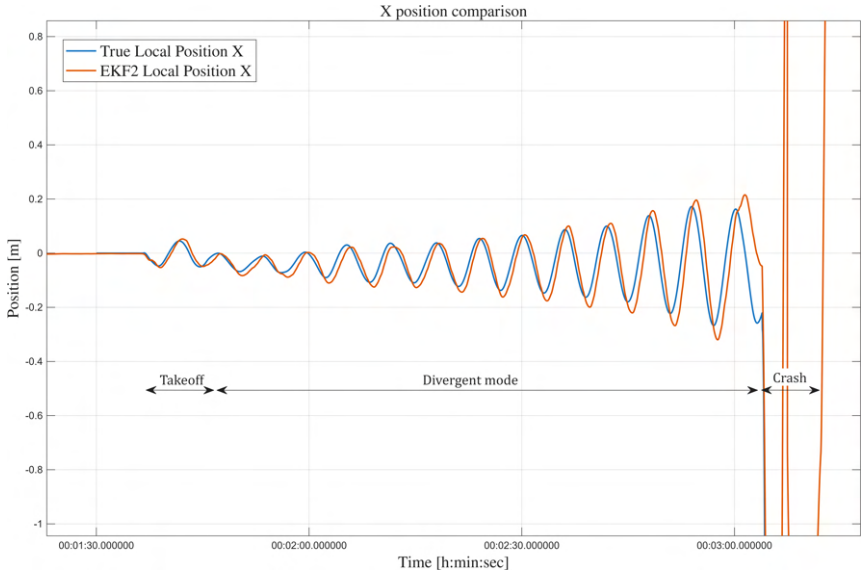


Figure 4.7: First simulated flight

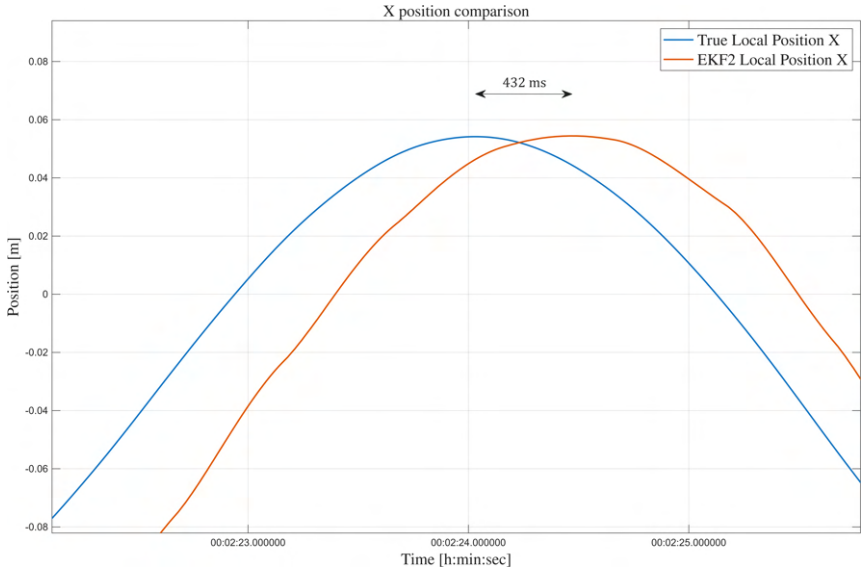
By analyzing deeper the flight section prior to the crash as in Fig. 4.8a, it is possible to characterize three different sections: the takeoff, the so called "toilet bowling" and the crash initiation.

It is particularly interesting to analyze the center section, as said before it is informally called "toilet bowling" and it is a divergent and uncontrolled mode that can happen during hover flight due to malfunctioning or discrepancies between IMU and compass or, as in this case, due to a non-correctly tuned positioning algorithm.

In particular, here it is due to a delay between the true position and the EKF estimation, this delay can be calculated by taking two near peaks of the two signals and making the difference between the timestamps as in Fig.4.8b and in this case the delay is 432 ms.



(a) Analysis of the different flight sections



(b) Analysis of the delay

Figure 4.8: Detail of X position comparison

The whole flight can be seen in Fig. 4.9, where the three flight sections are visible in the three-dimensional space in the NED frame (the Z direction has been inverted so that it points up). The drone does not exactly start at the ground because it center of origin is in the center of gravity which is raised from the ground. It is clear why it is informally called "toilet bowling" since it has an elliptical and divergent

motion that resembles water flow in the toilet.

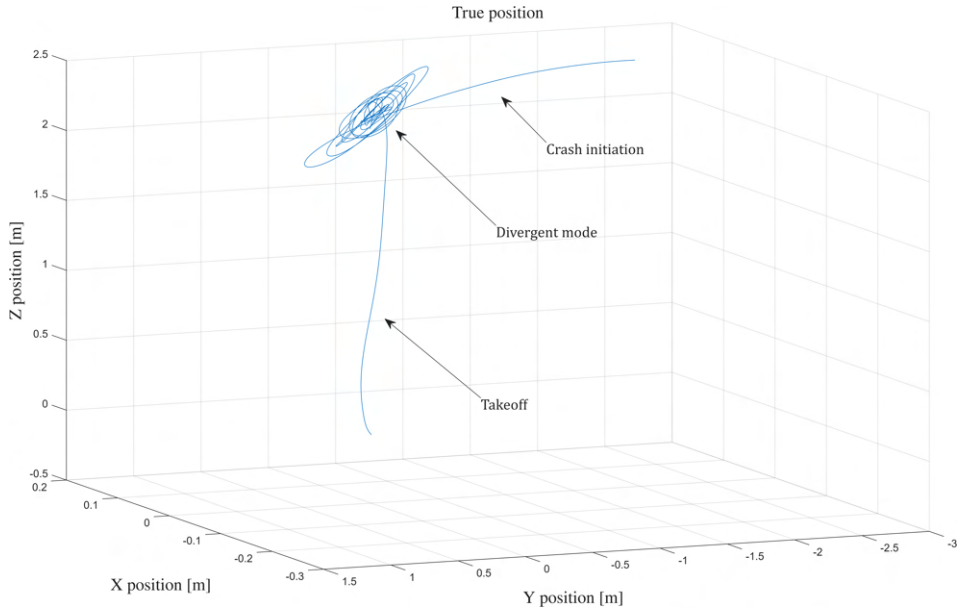


Figure 4.9: Analysis of the flight sections

Similar results were obtained by simulating the following parameters combination:

a) Sample Rate: 400 S/rev, Resolution: 0.05, Mode: Localization, delay: 432 ms

b) Sample Rate: 400 S/rev, Resolution: 0.01, Mode: Localization, delay: 528 ms

c) Sample Rate: 400 S/rev, Resolution: 0.05, Mode: Mapping, delay: 980 ms

d) Sample Rate: 200 S/rev, Resolution: 0.05, Mode: Localization, delay: 496 ms

e) Sample Rate: 200 S/rev, Resolution: 0.01, Mode: Localization, delay: 712 ms

f) Sample Rate: 200 S/rev, Resolution: 0.05, Mode: Mapping, delay: 932 ms

g) Sample Rate: 800 S/rev, Resolution: 0.05, Mode: Localization, delay: 584 ms

h) Sample Rate: 800 S/rev, Resolution: 0.01, Mode: Localization, delay: $n.d.$

i) Sample Rate: 800 S/rev, Resolution: 0.05, Mode: Mapping, delay: 2292 ms

In fig.4.10, it is possible to notice every time the same exact sequence: the takeoff, the formation of the divergent mode and the consequent crash. This makes the delay the first and most important parameter to tune in order to not result in crash, even for the real world application. It is also important to note that the combination of the localization mode and higher resolution value gives the lowest delay for each sample rate.

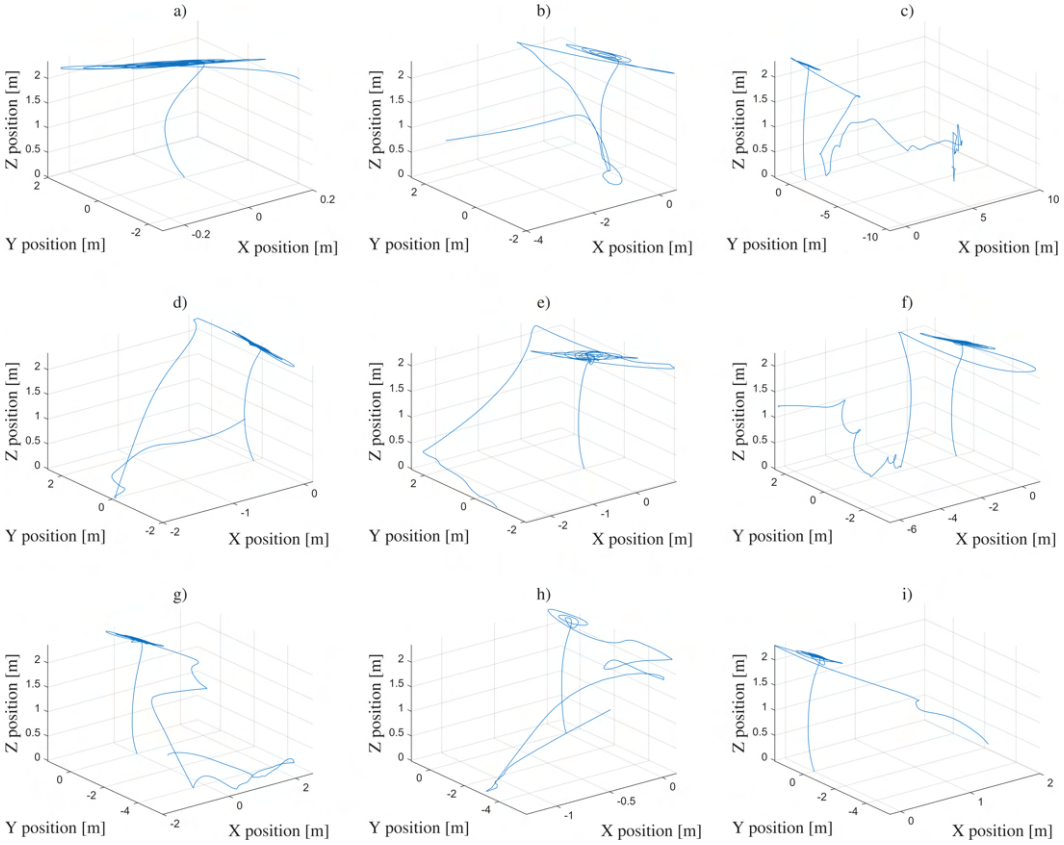
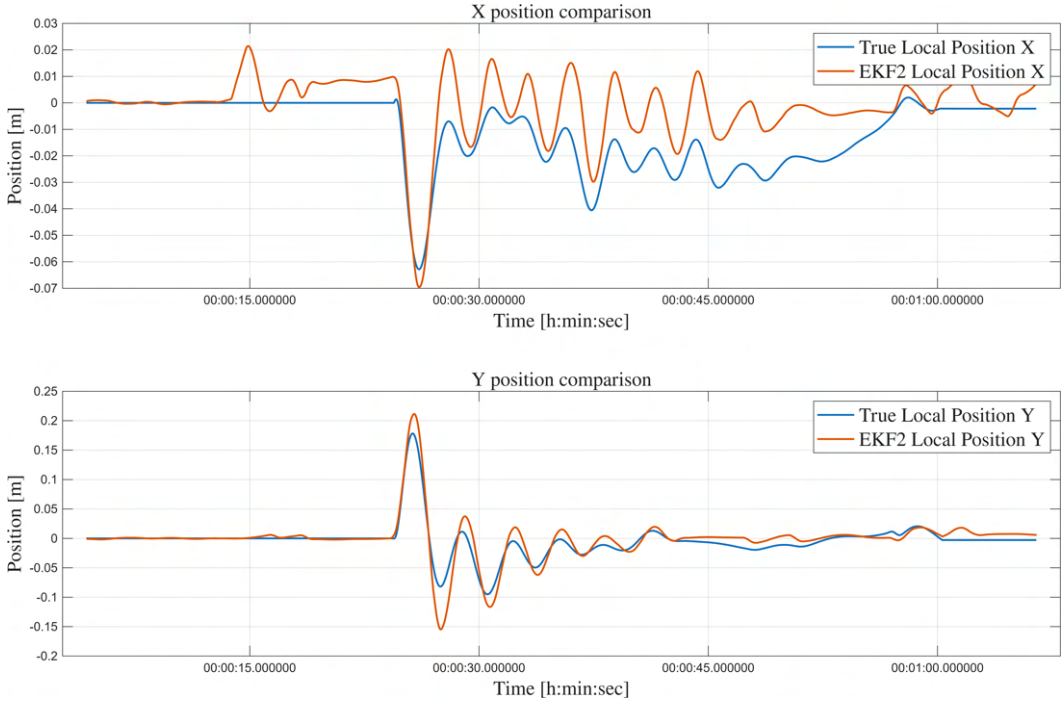


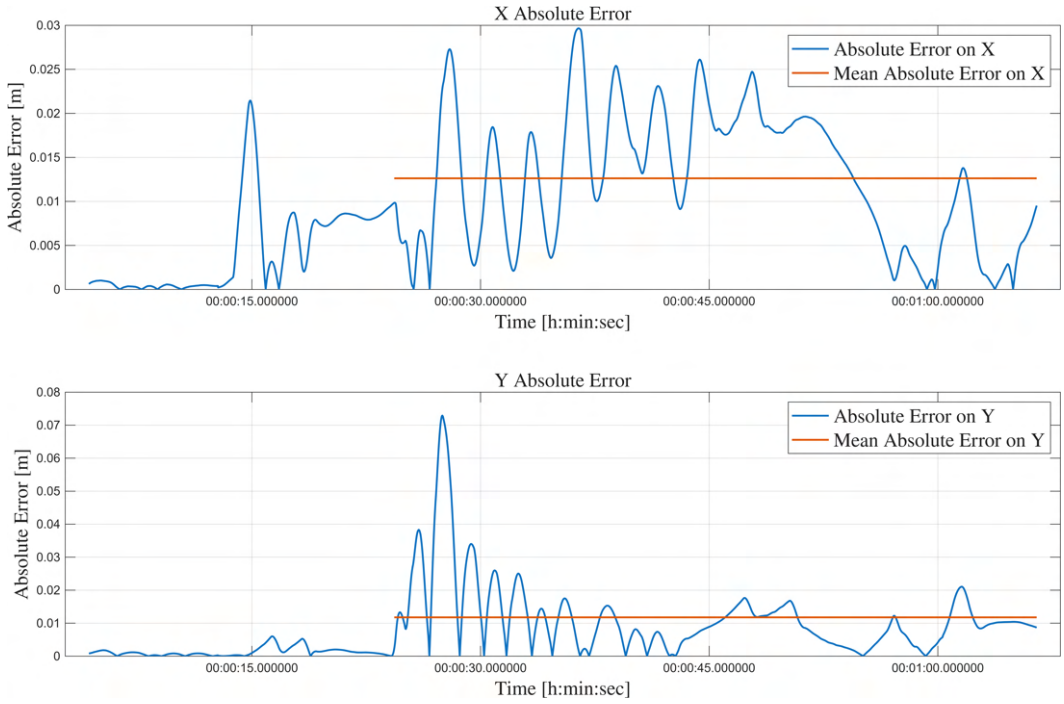
Figure 4.10: Simulated hover flights without delay correction

4.3.1 Delay Correction

Fortunately, PX4 software already takes this problem into account. In the parameters regarding EKF fusion with external vision, there is the parameter `EKF2_EV_DELAY` which has to be tuned accordingly to the system characteristics. Firstly, the configuration *a*) has been taken into account and the delay parameter has been set to `EKF2_EV_DELAY = 0.432`. A warning from PX4 appears because this value is above the threshold considered safe from the flight control firmware, by changing the parameter that defines this limit `EKF2_DELAY_MAX` to 0.6 the warning no longer appears. The results are shown in Fig. 4.11a, where there is the comparison between the true position and the one estimated by the PX4 EKF for both X and Y. By just changing the PX4 delay parameter, the hover maneuver does not result anymore in a crash and the peaks visible are not anymore shifted in time but they are aligned, this is done exclusively by the flight control firmware. Moreover, the precision of this simulated flight is more than satisfactory, the absolute error and its mean for X and Y coordinates are shown in Fig.4.11b. The maximum absolute error is equal to 0.272 m for X and 0.269 m for Y, in both cases it happens in correspondence of the takeoff sequence due to sudden change in attitude. The mean of the absolute error was taken from the start of the takeoff to the land touchdown to avoid misleading values of the mean itself. This is the reason why the signal does not start at the beginning of the time domain, its value is 0.013 m for X and 0.012 m for Y.



(a) Simulated hover with delay correction



(b) Absolute error analysis of the simulated flight

Figure 4.11: Position and absolute error analysis of the hover with combination *a*)

Moreover, the three dimensional plot of the flight is also reported in Fig.4.12. It is possible to notice that there is an initial substantial drift due to the aforementioned effects of the takeoff, then the UAV stabilizes itself in the position origin and shows more than acceptable positioning results for the whole duration of this flight segment, which is approximately one minute. Finally, it starts the landing sequence and touches the ground almost exactly where it started.

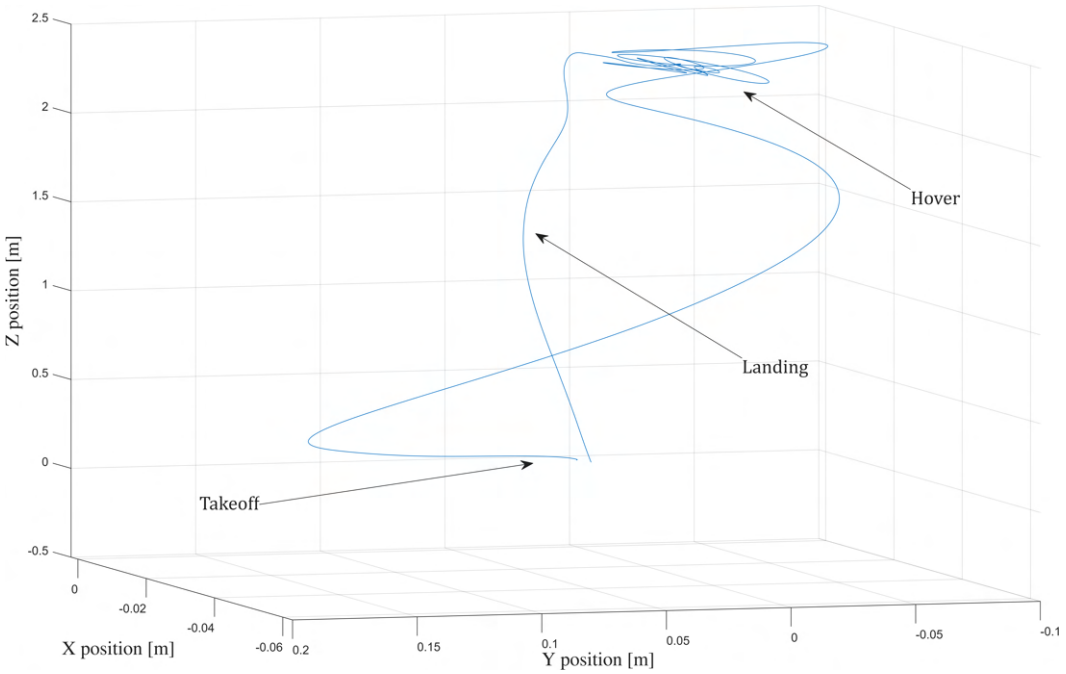


Figure 4.12: Three dimensional representation of the true position

The localization section of the SLAM algorithm has been discussed and analyzed, but in addition to that, the algorithm also consists of the mapping part, which creates a map of the environment while the UAV is localizing itself. The result is shown in Fig.4.13, where the whole distance sensing radius of 12 m has been used to create the shown map. The most important parameters for this topic are the maximum radius and the resolution. Usually, the maximum sensing distance of the sensor is taken as the maximum radius for mapping in order to give the pilot the most clear map as possible to avoid obstacle, but since this sensor is 2D, small attitude angle can give considerable distance estimation errors at long range. For this application, at 12 m the errors are appreciable but do not invalidate the algorithm and thus there is no need to restrict the maximum range even if it could be done simply by changing the `slam_toolbox` parameter `max_distance`. The second parameter, resolution, is fundamental in order to obtain a clear and correct map of the environment. This will be further discussed in the navigation section but essentially a lower value reproduces the map with a higher fidelity and smaller details are more appreciable. Unfortunately, resolution also effect computational cost and a high-fidelity map can be so expensive that the algorithm is not able to elaborate LiDAR data fast enough to be reliable and to fly safely, which can result in the crash of the UAV.

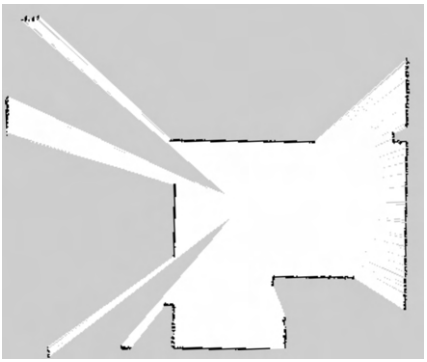


Figure 4.13: SLAM algorithm map output

4.3.2 Hover with 200 Samples/rev

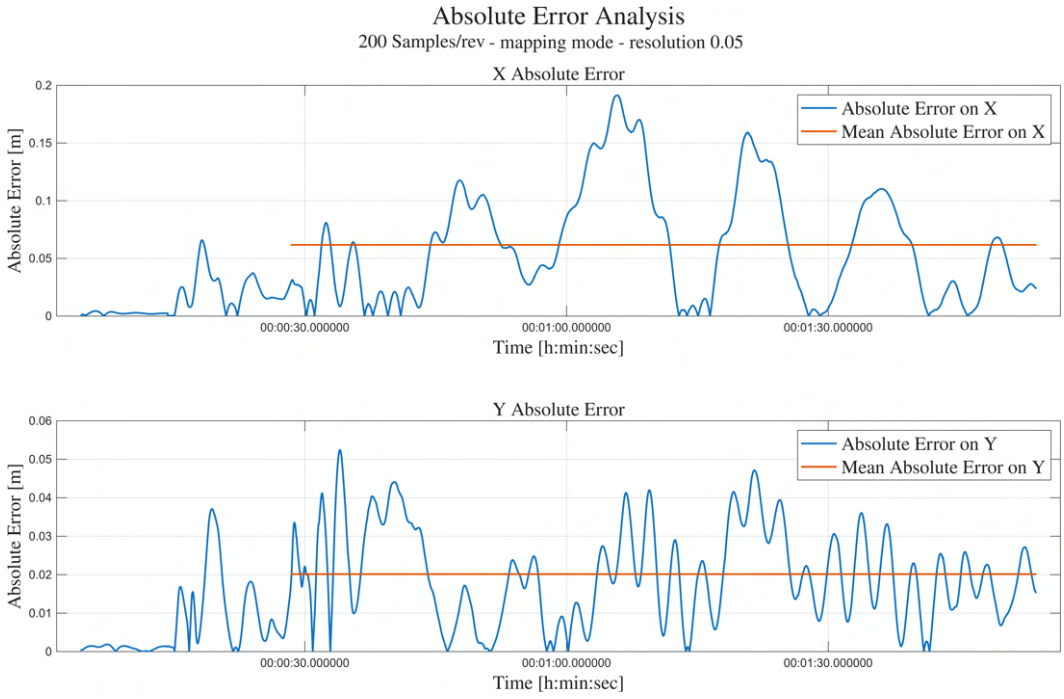
In order to understand the behavior of the algorithm with different settings, the simulations were divided firstly by sample rate. By imposing a sample rate equal to 200 in the LiDAR Gazebo file, the following results expressed in the table were obtained by changing the optimization mode and the delay parameter.

	Mode	Res.	`EKF2_EV_DELAY`	$\overline{\lvert e_x \rvert}$ $[m]$	$\overline{\lvert e_y \rvert}$ $[m]$	**Fig.**
1)	Mapping	0.05	932 420	*n.d.* 0.062	*n.d.* 0.020	4.14
2)	Mapping	0.01	420	0.027	0.018	4.15
3)	Localization	0.05	496	0.019	0.020	4.16
4)	Localization	0.01	712 520	*n.d.* 0.035	*n.d.* 0.039	4.17

Table 4.2: Hover result with 200 *Samples/rev*

Note that *n.d.* stands for "not determined" because those particular configurations were note able to fly with that delay parameter and thus it had to be decreased manually and experimentally in order to achieve a safe flight. The results are also reported in the bar chart 4.18 for clarity.

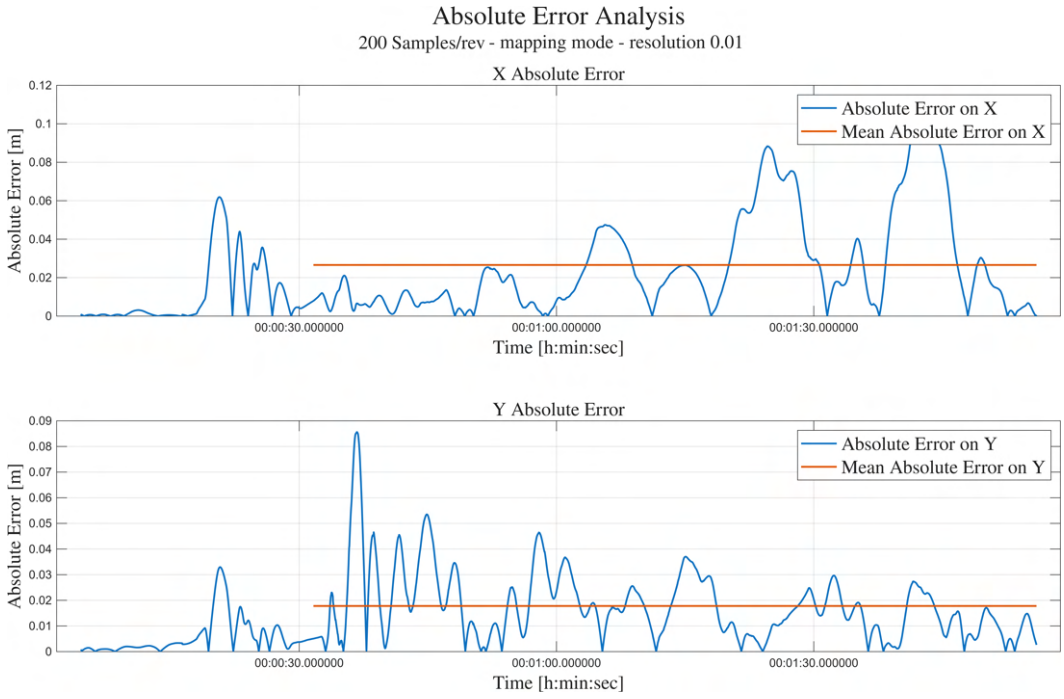
(a) Position comparison for configuration 1) with 200 $Samples/rev$



(b) Absolute error analysis for configuration 1) with 200 $Samples/rev$

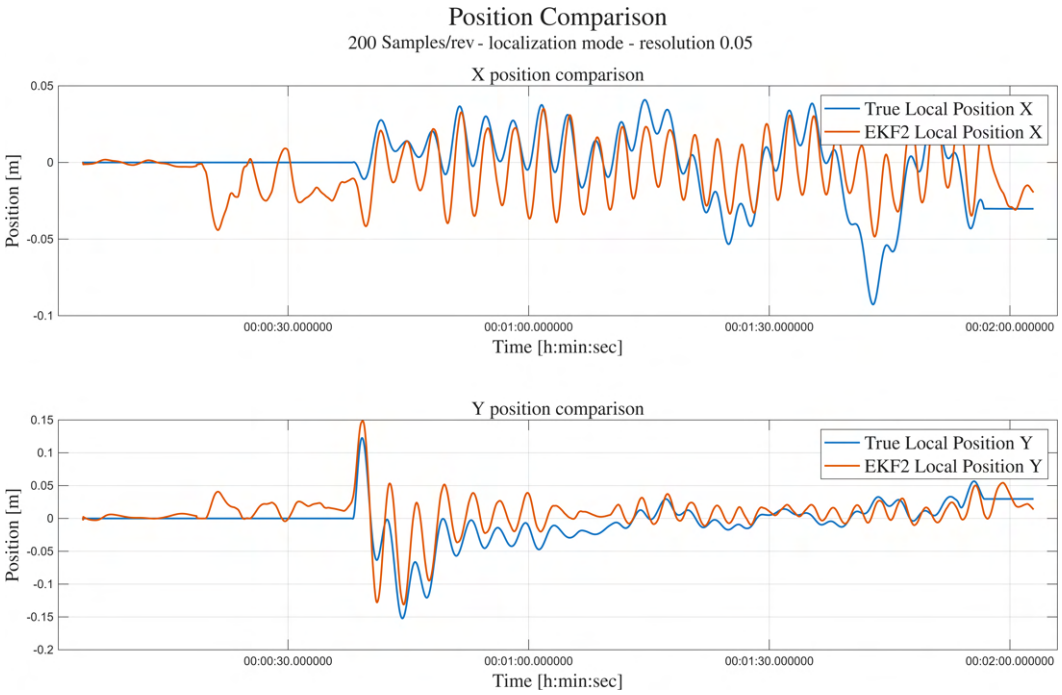
Figure 4.14: Position and absolute error analysis of configuration 1) with 200 $Samples/rev$

(a) Position comparison for configuration 2) with 200 $Samples/rev$

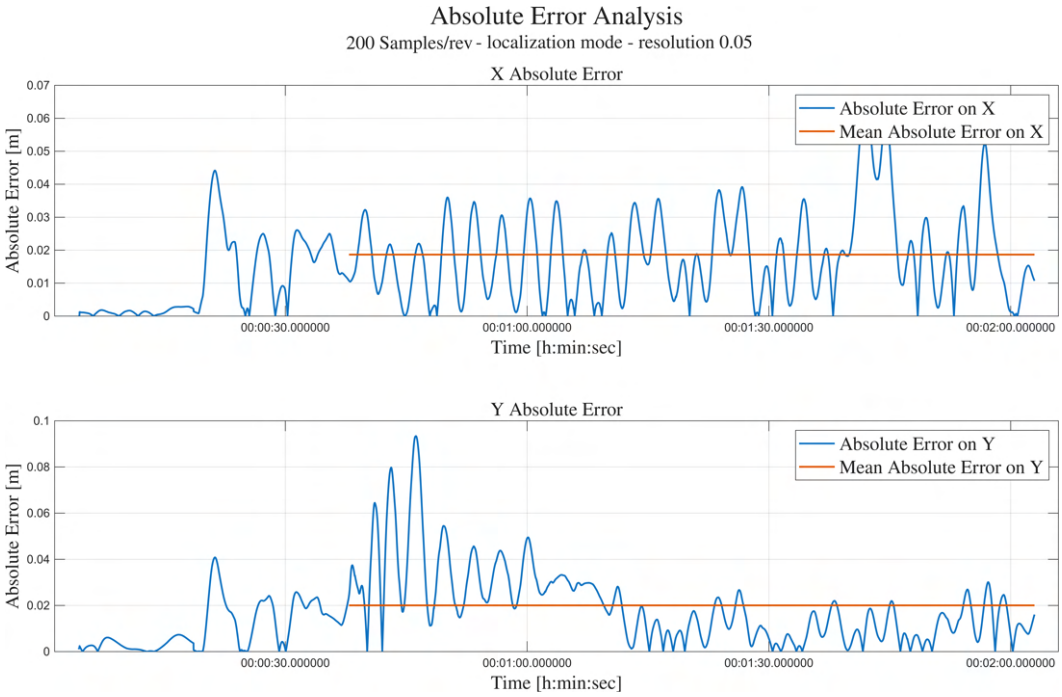


(b) Absolute error analysis for configuration 2) with 200 $Samples/rev$

Figure 4.15: Position and absolute error analysis of configuration 2) with 200 $Samples/rev$

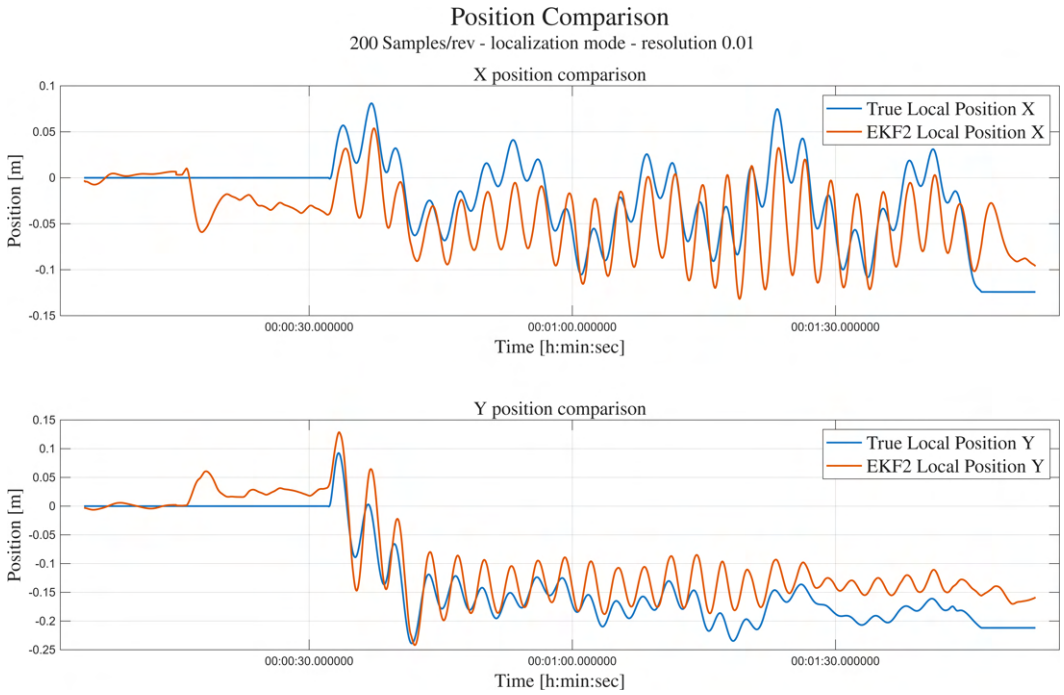


(a) Position comparison for configuration 3) with 200 $Samples/rev$

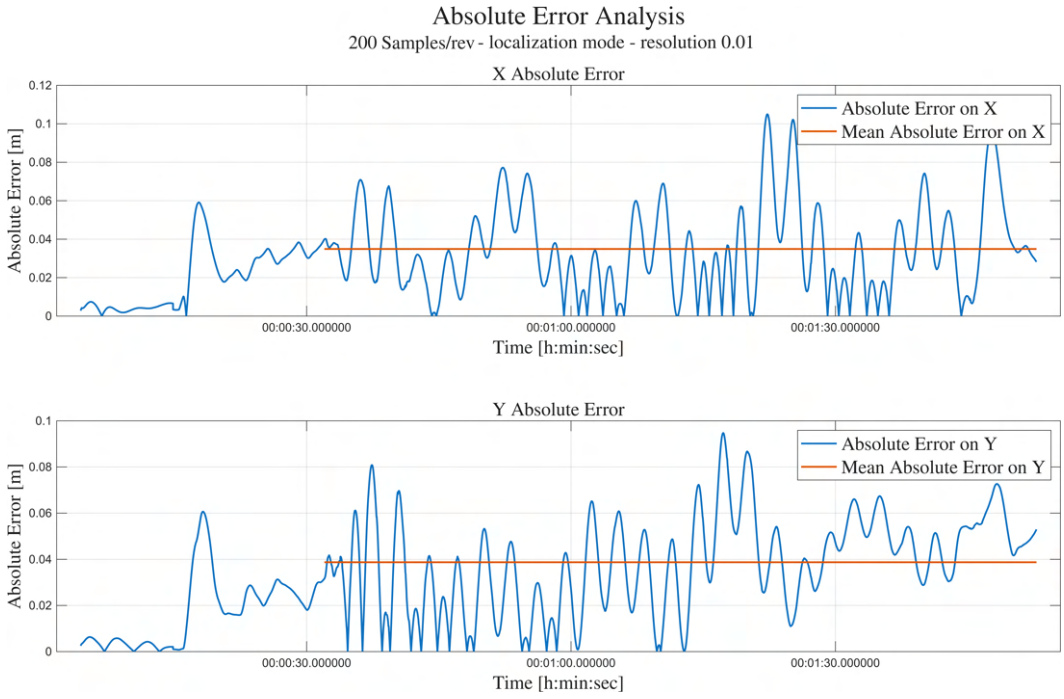


(b) Absolute error analysis for configuration 3) with 200 $Samples/rev$

Figure 4.16: Position and absolute error analysis of configuration 3) with 200 $Samples/rev$



(a) Position comparison for configuration 4) with 200 $Samples/rev$



(b) Absolute error analysis for configuration 4) with 200 $Samples/rev$

Figure 4.17: Position and absolute error analysis of configuration 4) with 200 $Samples/rev$

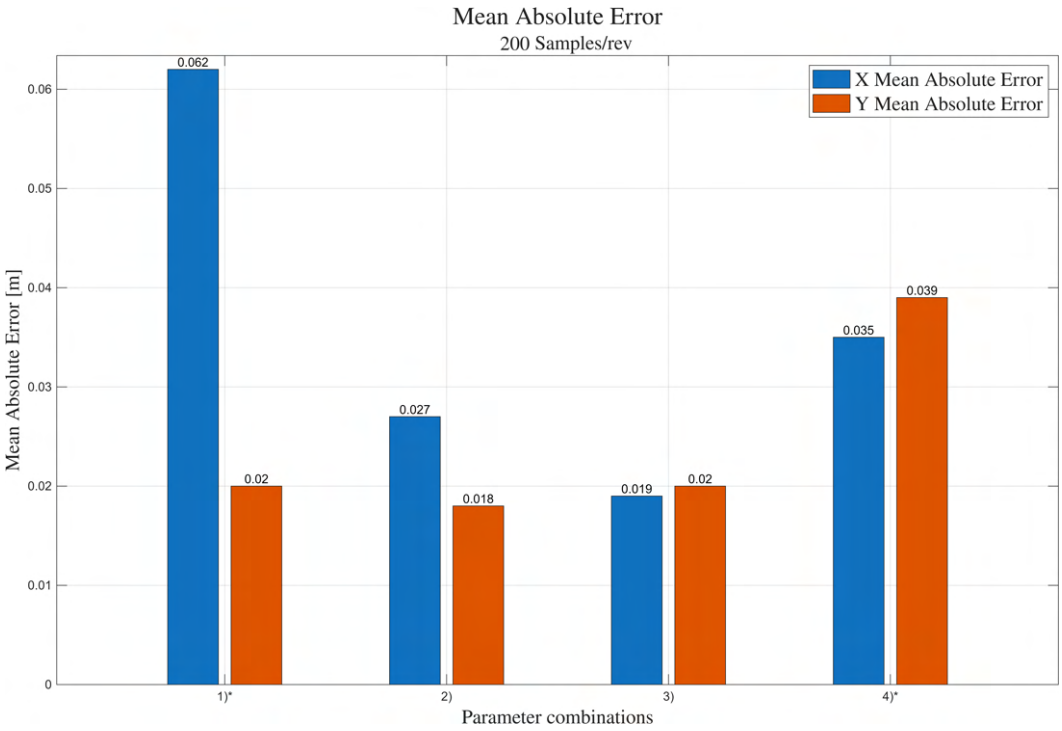


Figure 4.18: Mean Absolute Error for 200 *Samples/rev*

Configuration 1)* shows the highest mean absolute error in the x direction due to oscillations of the UAV. Since the delay was decreased from 932 ms to 420 ms (thus the apex), the oscillations could be dependent on the delay parameter itself and it could be tuned to achieve slightly better result. On the other hand, it is also possible that the oscillations are due to a low resolution mapping that could affect the positioning algorithm. The overall best result was achieved by the third configuration, which had a resolution of 0.05, it was in localization mode and achieved an x and y mean absolute error of 0.019 m and 0.020 m respectively. It is interesting to notice how applying a better resolution decreased the mean absolute error in the mapping mode, while for the localization mode it has the opposite effect.

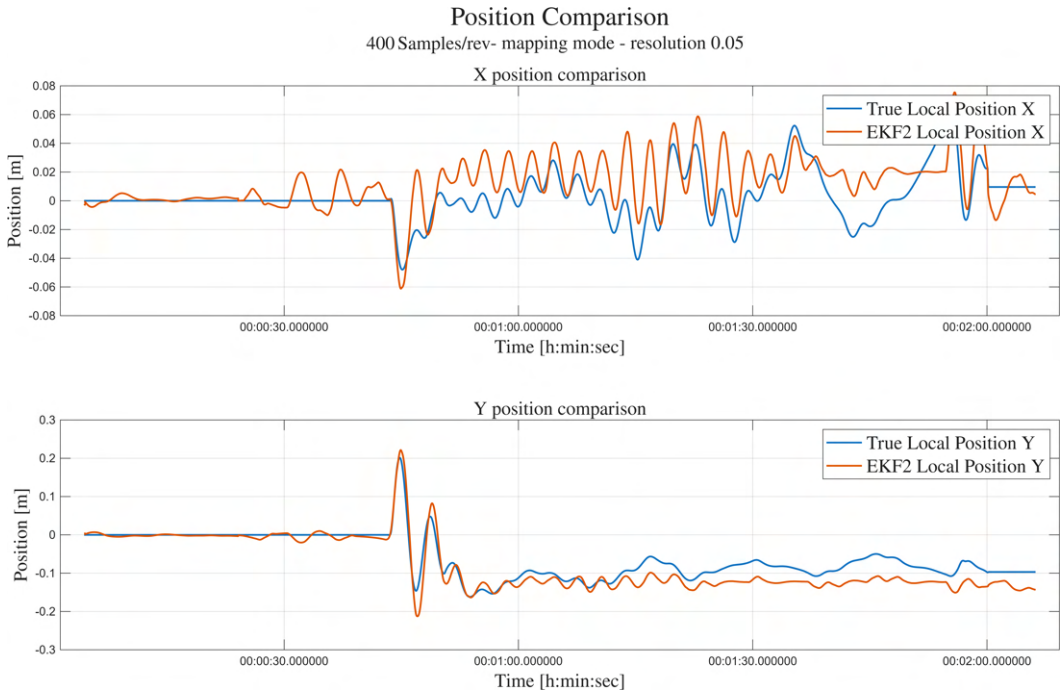
4.3.3 Hover with 400 Samples/rev

The same process was applied again changing the LiDAR Gazebo model and imposing a sample rate of 400 *Samples/rev* and the results are reported in the next table and bar chart. It is important to notice how useful are simulations: in this case, before implementing the actual algorithm on hardware, it is possible to analyze the behavior with different simulated hardware configurations by simply changing the LiDAR sample rate from its own model file.

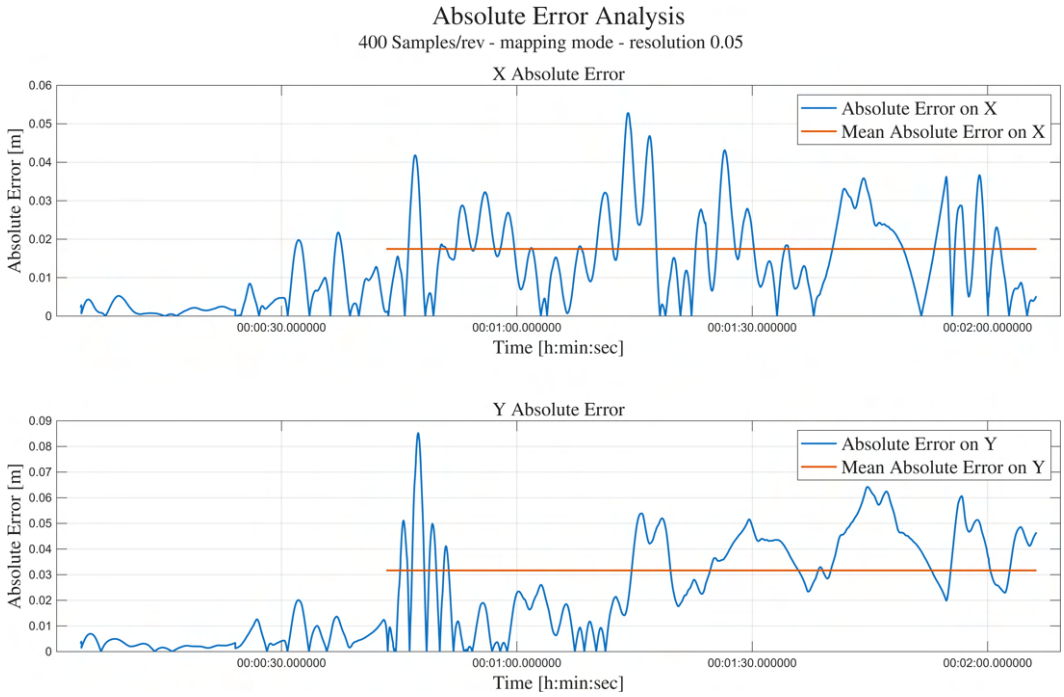
	Mode	Res.	`EKF2_EV_DELAY`	$\overline{\lvert e_x \rvert}$ $[m]$	$\overline{\lvert e_x \rvert}$ $[m]$	**Fig.**
1)	Mapping	0.05	980 420	*n.d.* 0.017	*n.d.* 0.032	4.19
2)	Mapping	0.01	420	0.024	0.020	4.20
3)	Localization	0.05	432	0.013	0.012	4.11a
4)	Localization	0.01	528	0.017	0.045	4.21

Table 4.3: Hover result with 400 *Samples/rev*

Each simulation is also reported in the following figures to offer the reader more clarity about the flight sequence and error trends.

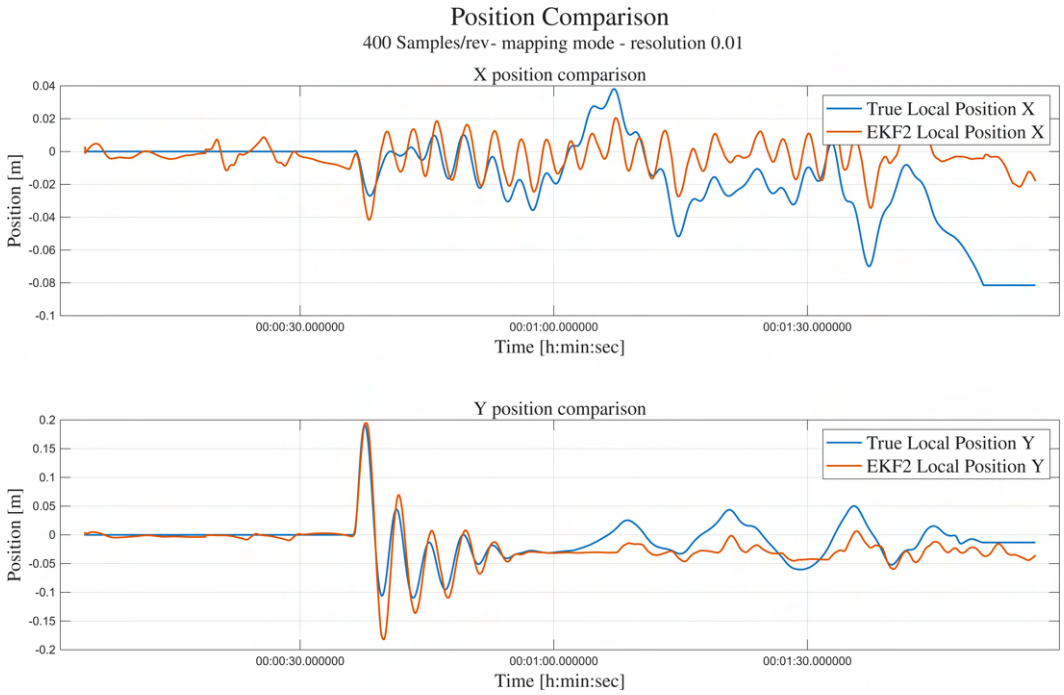


(a) Position comparison for configuration 1) with 400 $Samples/rev$

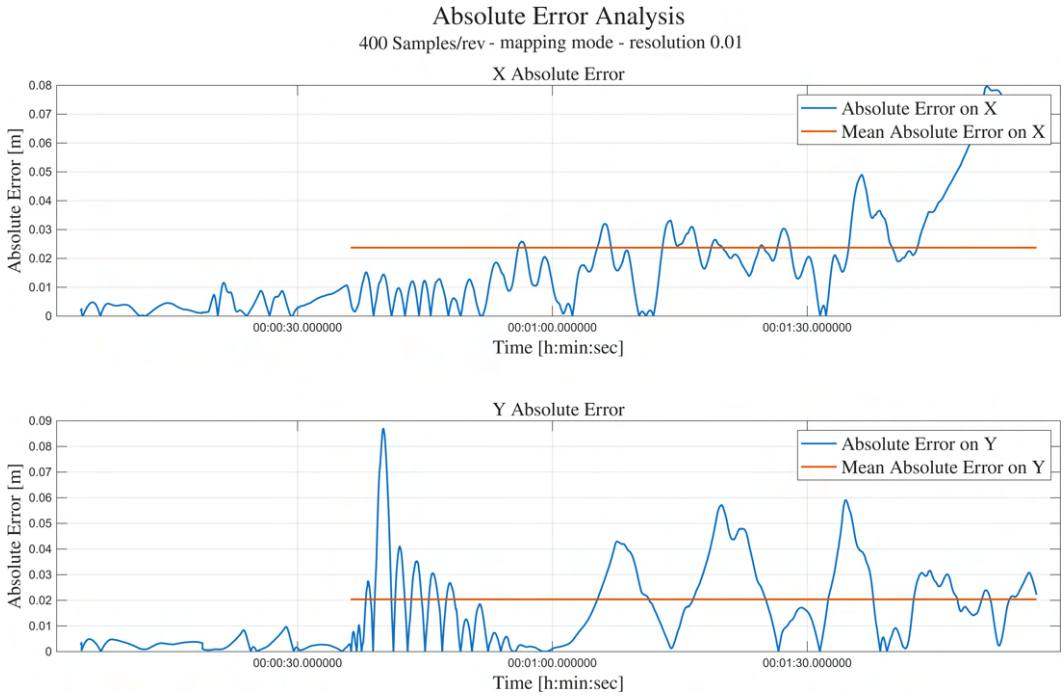


(b) Absolute error analysis for configuration 1) with 400 $Samples/rev$

Figure 4.19: Position and absolute error analysis of configuration 1) with 400 $Samples/rev$

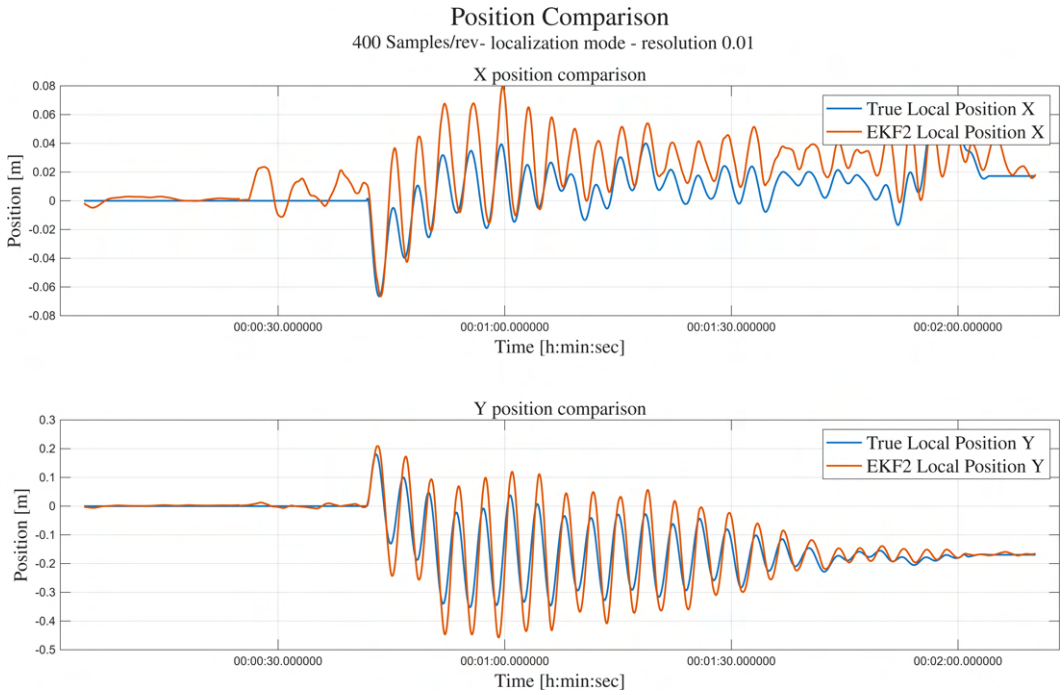


(a) Position comparison for configuration 2) with 400 $Samples/rev$

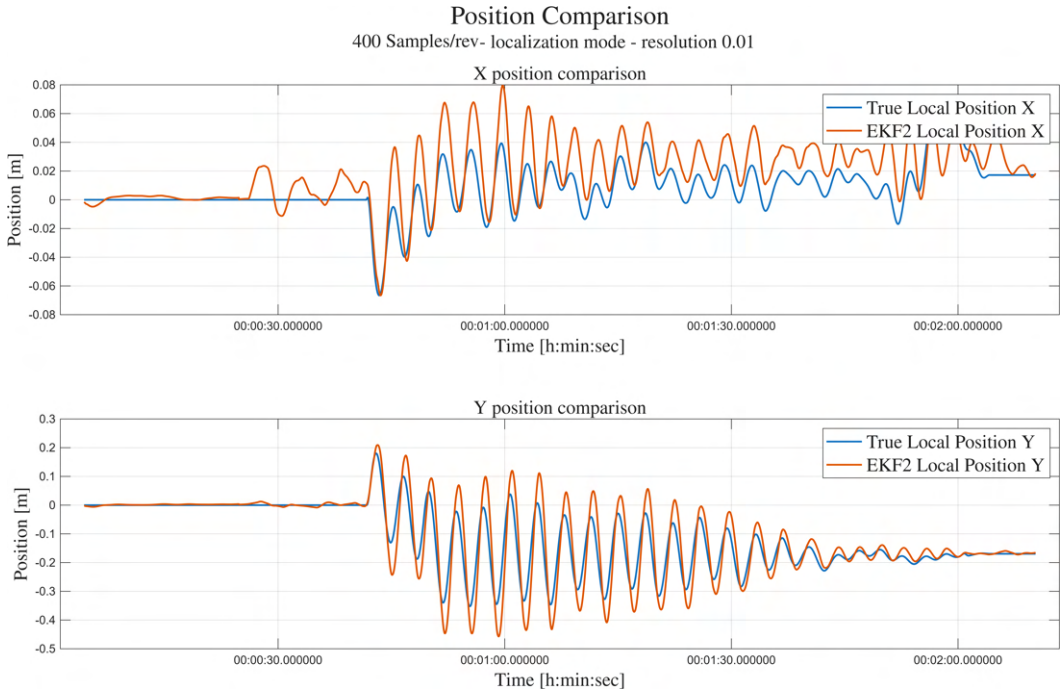


(b) Absolute error analysis for configuration 2) with 400 $Samples/rev$

Figure 4.20: Position and absolute error analysis of configuration 2) with 400 $Samples/rev$



(a) Position comparison for configuration 4) with 400 $Samples/rev$



(b) Absolute error analysis for configuration 4) with 400 $Samples/rev$

Figure 4.21: Position and absolute error analysis of configuration 4) with 400 $Samples/rev$

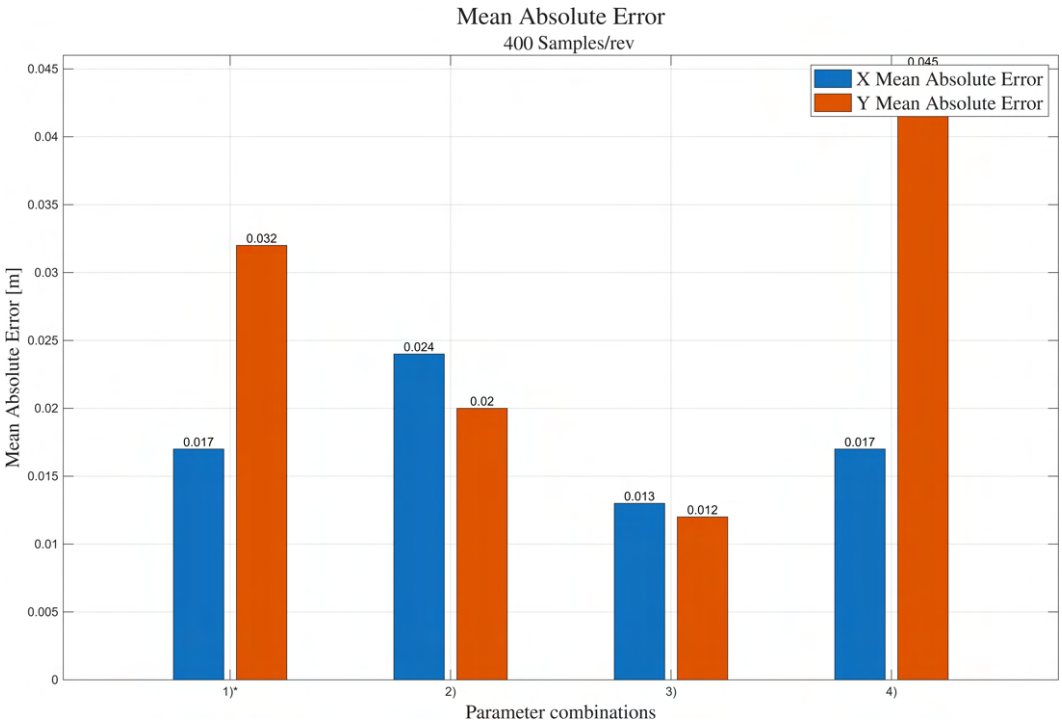


Figure 4.22: Mean Absolute Error for 400 *Samples/rev*

From this bar chart it is clear that the best results are achieved by the configuration with localization mode and resolution 0.05, the third one. It achieved an error of 0.013 m in the x direction and 0.012 m in the y direction. The closest configuration to this result is the second one, which was in mapping mode with a resolution parameter of 0.01. This simulation was able to achieve an error of 0.024 m in x and 0.020 m in y. The first and the last configurations show a high error again due to oscillations, which can be seen also in the figures above. Theoretically, by means of better and more precise tuning, it could be possible to damp these oscillations and hence increase the system accuracy.

4.3.4 Hover with 800 Samples/rev

The last set of hover simulations were performed using the maximum sample rate available from the LiDAR sensor: 800 *Samples/rev*. The results are reported in the following table. Moreover, with this setting the map creation becomes precisely detailed, particularly for corners, and the created map are shown below.

	Mode	Res.	`EKF2_EV_DELAY`	$\overline{\lvert e_x \rvert}$ $[m]$	$\overline{\lvert e_x \rvert}$ $[m]$	**Fig.**
1)	Mapping	0.05	2292	*n.d.*	*n.d.*	
			420	0.010	0.009	4.24
2)	Mapping	0.01	420	0.008	0.018	4.25
3)	Localization	0.05	584	*n.d.*	*n.d.*	
		0.05		0.009	0.013	4.26
4)	Localization	0.01	528	0.007	0.011	4.27

Table 4.4: Hover result with 800 *Samples/rev*

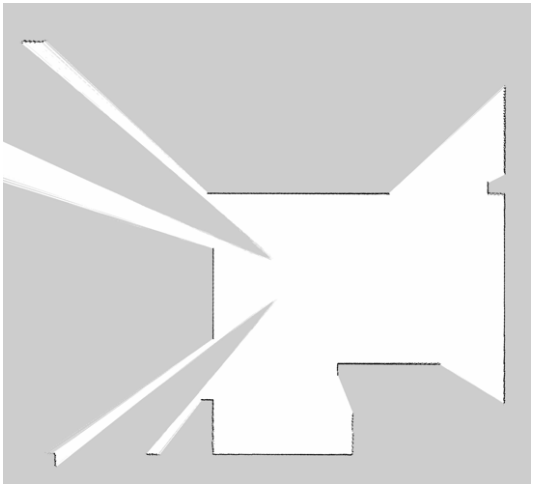
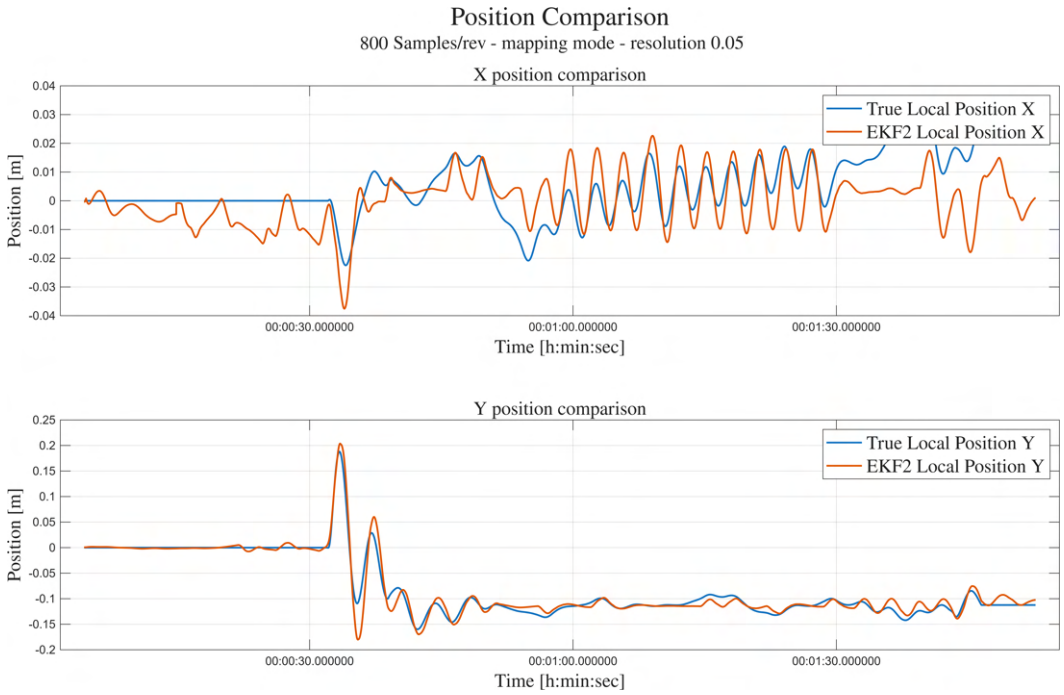


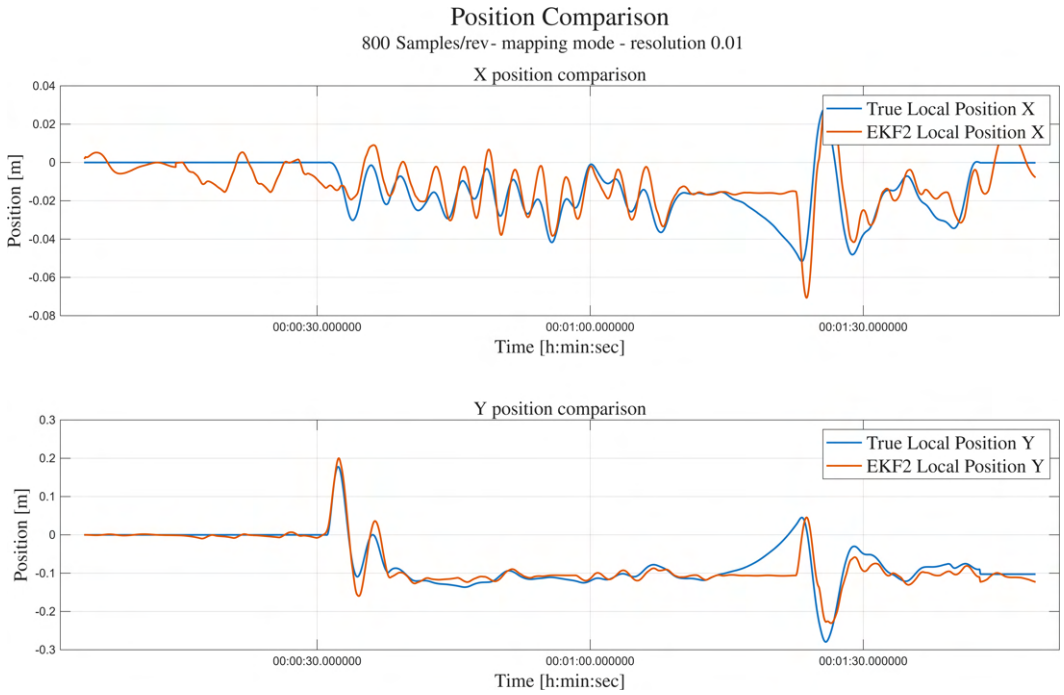
Figure 4.23: Map created by the SLAM algorithm with 800 *Samples/rev*, mapping mode and resolution 0.01



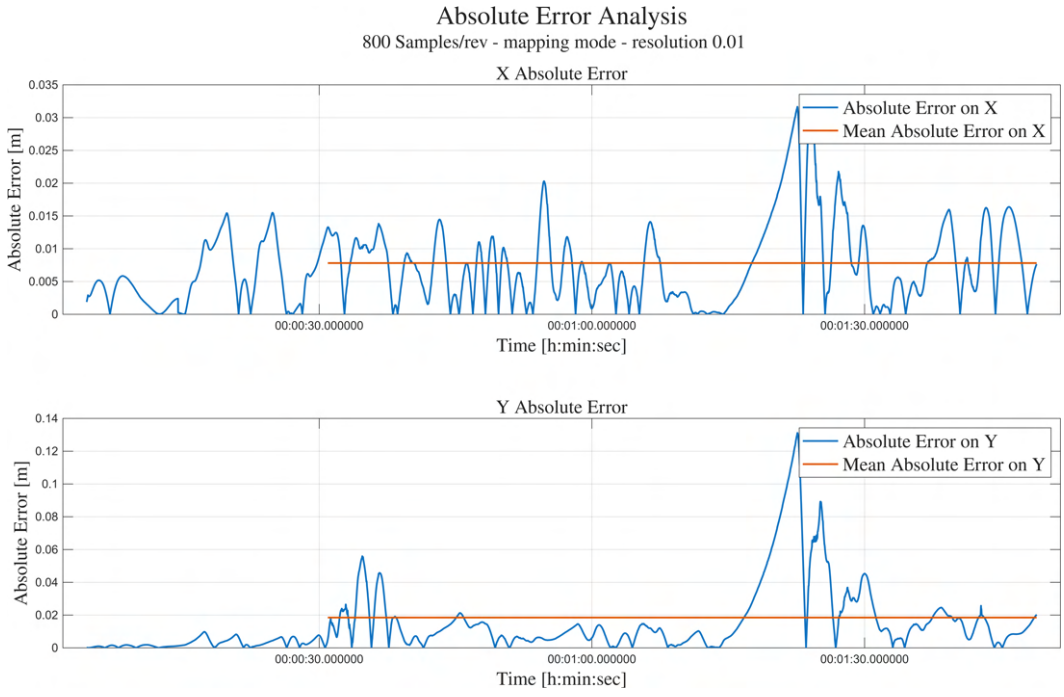
(a) Position comparison for configuration 1) with 800 *Samples/rev*

(b) Absolute error analysis for configuration 1) with 800 *Samples/rev*

Figure 4.24: Position and absolute error analysis of configuration 1) with 800 *Samples/rev*

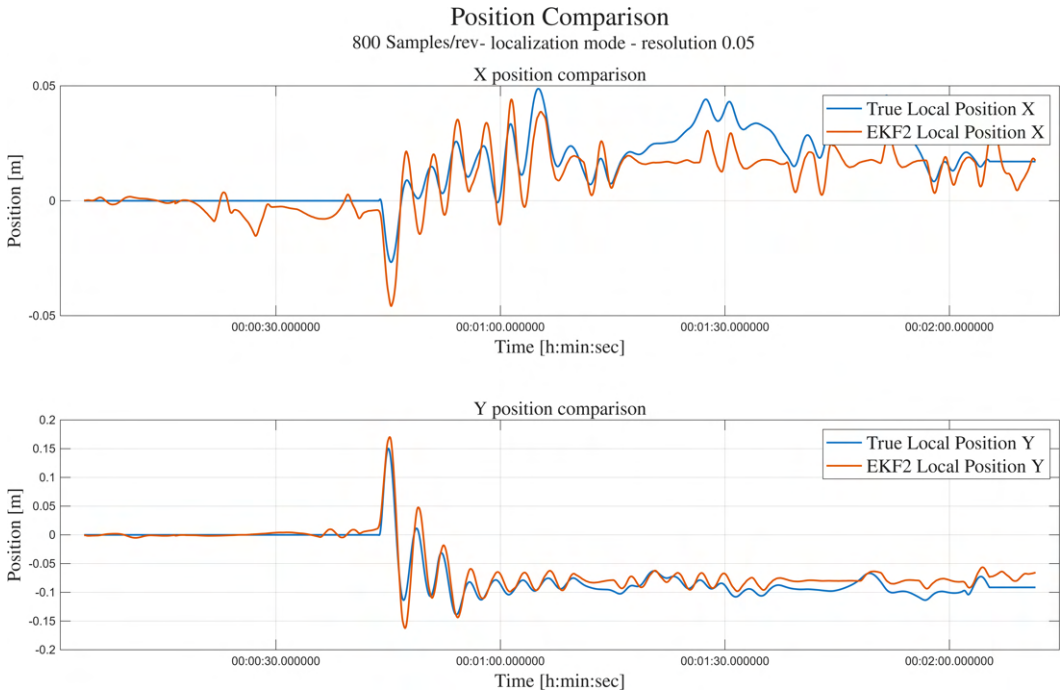


(a) Position comparison for configuration 2) with 800 *Samples/rev*

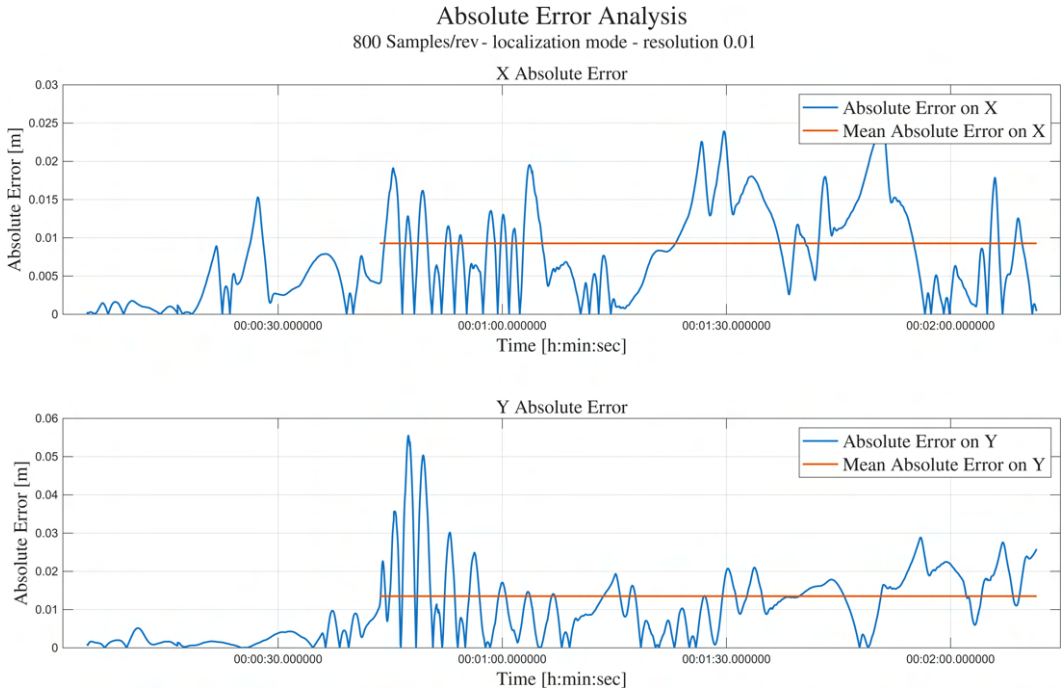


(b) Absolute error analysis for configuration 2) with 800 *Samples/rev*

Figure 4.25: Position and absolute error analysis of configuration 2) with 800 *Samples/rev*

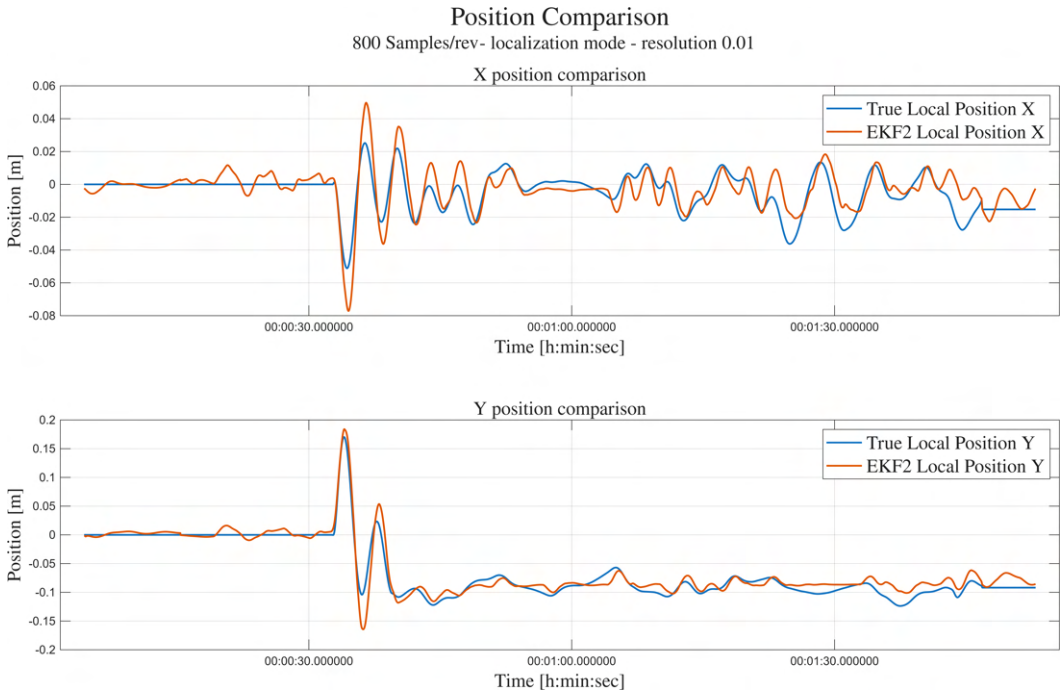


(a) Position comparison for configuration 3) with 800 *Samples/rev*

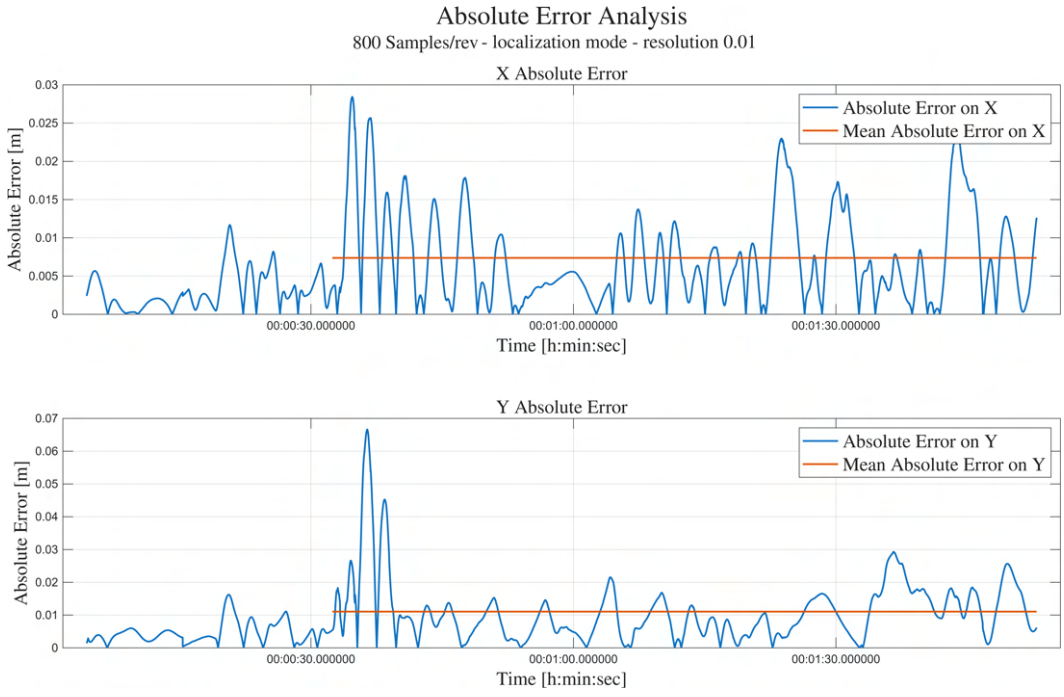


(b) Absolute error analysis for configuration 3) with 800 *Samples/rev*

Figure 4.26: Position and absolute error analysis of configuration 3) with 200 *Samples/rev*



(a) Position comparison for configuration 4) with 800 *Samples/rev*



(b) Absolute error analysis for configuration 4) with 800 *Samples/rev*

Figure 4.27: Position and absolute error analysis of configuration 4) with 800 *Samples/rev*

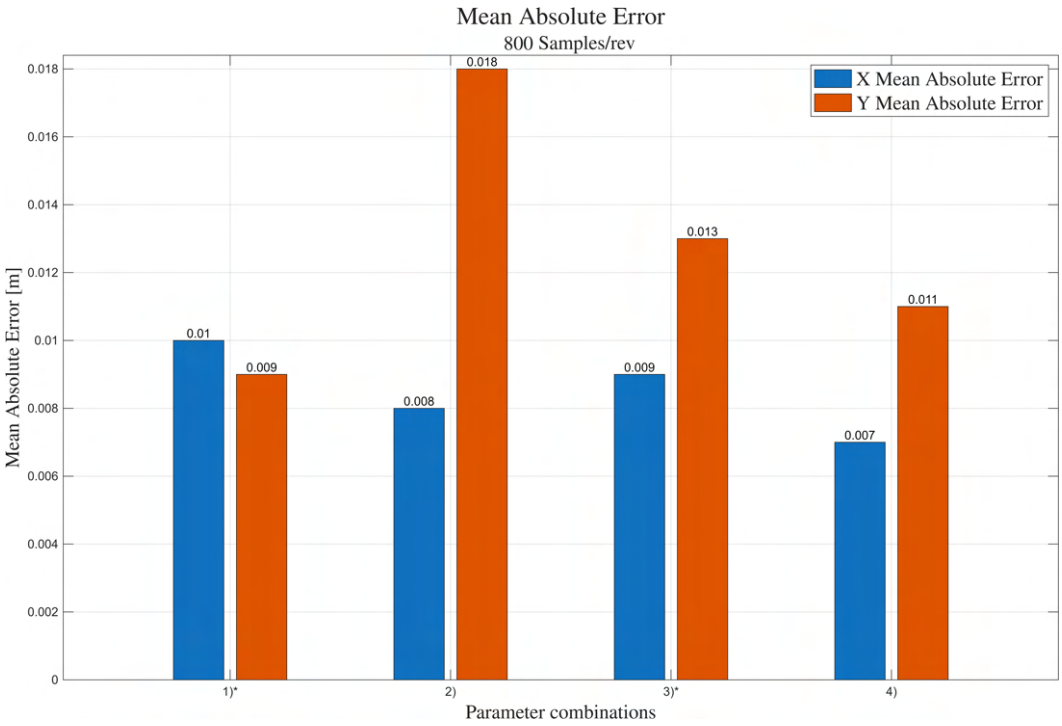


Figure 4.28: Mean Absolute Error for 800 *Samples/rev*

From this bar chart it is possible to notice that the best hover precision was achieved by the fourth configuration, the one in localization mode with a resolution parameter of 0.01. However, comparable precisions were also obtained with the other configurations except from the second one. Analyzing the second configuration in detail, it is possible to see a feature that was not present with lower samples rates: the algorithm lagging. In particular, around $1' 15''$ it can be seen how the EKF2 estimation shows a plateau while the UAV is drifting away. This happens due to hardware limitation because the higher the sample rate, the higher is the number of points that the algorithm has to deal with and thus it can be introduce this lagging error. It is not strange that it happened with the second configuration, since the mapping mode is more computationally expensive than localization mode.

4.3.5 Hover considerations

From the analysis of the hover flight, various and important considerations can be derived regarding parameter combinations and behavior consequences:

- Delay parameter is of fundamental importance for a safe precise flight. It has to be configured in the PX4 flight control firmware and the delay measured without correction can be taken as a first attempt value. After ensuring that the UAV can be safely flown without excessive oscillations, the delay parameter can be further refined by an empirical tuning process. Moreover, it has been discovered that applying values above 700 ms to the system does not allow the arming checks to be passed, since the PX4 firmware does not assume the position reference signal reliable enough. This is particularly interesting

because, even if the default maximum delay parameter is 300 ms, this could be theoretically brought up to 1000 ms from the firmware point of view.

- The hover precision achieved by this LiDAR SLAM system is impressive, even analyzing different setups with different parameter combination and tuning, the mean absolute error is in the order of tens of centimeters, which can be considered an outstanding result and could be, at least in simulations, compared to RTK GNSS systems.

- Localization mode is generally less computational expensive than mapping mode for the chosen parameter combinations. An increase of precision can be noticed by passing from mapping to localization mode with every sample rate, and this is due to the different settings that are applied to the Ceres solver. This is particularly evident with 800 $Samples/rev$ where the first signs of hardware limitation are visible, which lead to an increase of the absolute error between the true position and the estimated one from the Kalman Filter. However, it is necessary to further emphasize the impact of the delay parameter, since each combination needs a refined tuning in order to proper function.

- Map creation increases its accuracy and reproduces details better for higher samples rate. This is trivial, but it is still important to report that the best map accuracy is obtained in mapping mode, with the highest resolution parameter and highest samples rate.

4.4 Navigation results

Navigation is the process of determining the UAV position while flying in an unknown environment, and it is surely a more challenging task to perform than hover. While stationary flight simulations were employed to essentially determine the delay parameter and to check its mean absolute error, navigation simulations are needed in order to test the computationally cost of the SLAM algorithm and to assess its capability of locating the drone itself during a prolonged flight in presence of sensor noise and drift. The flights were performed using the "Position" flight mode and the pilot input needs to be sufficiently gentle to avoid excessive oscillations. In fact, if the attitude angle is too high, the LiDAR sensor readings do not represent anymore the real distance from the obstacles but there is a factor of $1/cos(\theta)$ or $1/cos(\phi)$ that can give erroneous results if this attitude angle is kept for a sufficiently high time. In particular, as possible to see in Fig. 4.29, the mathematical relation between the real distance and the measured distance is

$$d_{real} \;=\; d_{meas} \times \cos(\theta)$$

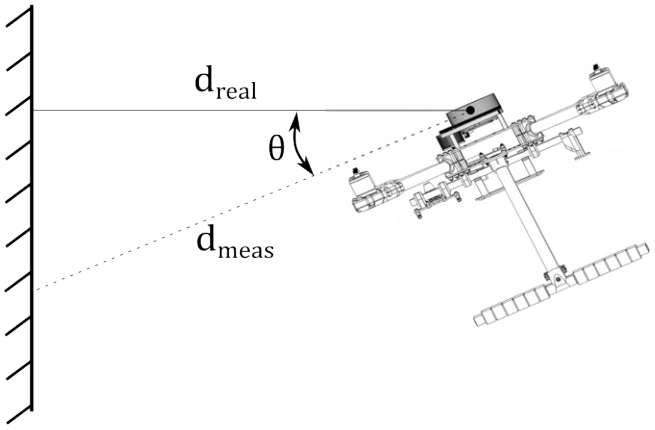


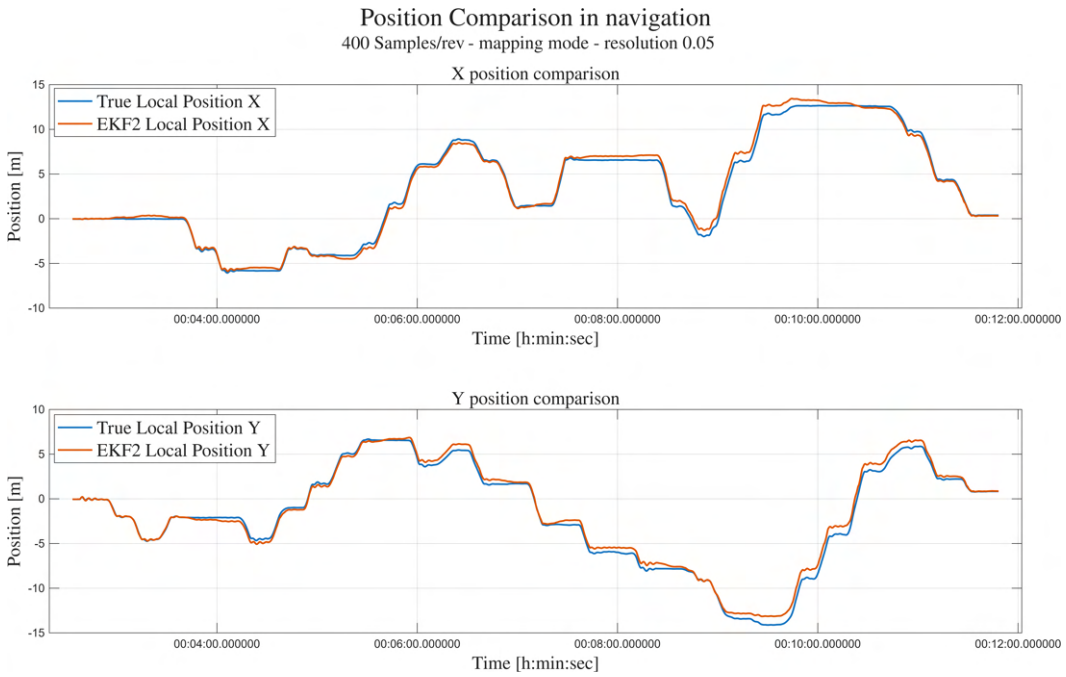
Figure 4.29: Object distance error for high attitude angle

Moreover, simulations with a sample rate of 200 $Samples/rev$ were not performed due to low map resolution, which yields results that are not satisfactory enough for our purposes. During flight the pilot can fly remotely, since there is the possibility to see the map creation and drone localization in real-time thanks to the visualizer RVIZ2 which is the standard software to graphically visualize ROS2 topics.

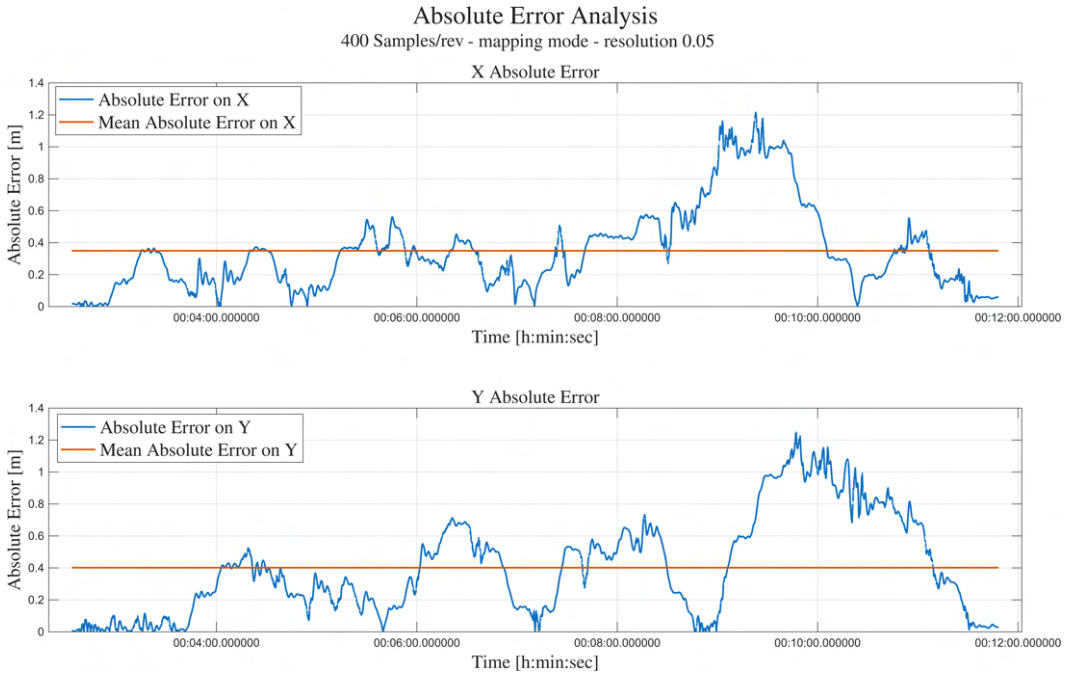
4.4.1 Navigation with 400 Samples/rev

In this subsection, the results of navigation simulations with LiDAR sample rate set to 400 $Samples/rev$ are reported. In addition, the created map is also shown.

1. Mapping mode, resolution 0.05



(a) Position comparison of mapping and resolution 0.05 with 400 *Samples/rev*



(b) Absolute error analysis of mapping and resolution 0.05 with 400 *Samples/rev*

Figure 4.30: Position and absolute error analysis of with mapping and resolution 0.05 with 400 *Samples/rev*

It is very interesting to notice that the position is tracked with satisfactory precision through the whole flight and the maximum error is approximatly 1.2 m for both x and y direction. There are two considerations to make by looking at the Fig.4.30. The first one is that the error tends to increase over time, this is due to the fact that the drone is localizing itself in a map made with some degree of uncertainties and thus positioning errors tends to build up. The second consideration is that at some points, for example at time $7' 10''$, errors tend to zero. This is not by accident, but it happens in correspondence of the UAV returning to already know and well mapped locations, particularly near the origin in the world zone "A". This is a pleasing feature of the SLAM algorithm and it can be used to lower the positioning errors in prolonged flights. In fact, during the landing phase at the end of the flight, which was performed near the takeoff point, the absolute error was 0.059 m in x and 0.041 in y. The mean absolute error during the whole flight was 0.349 m and 0.401 m on x and y respectevely. The trajectory of the drone during its navigation is reported in Fig.4.31

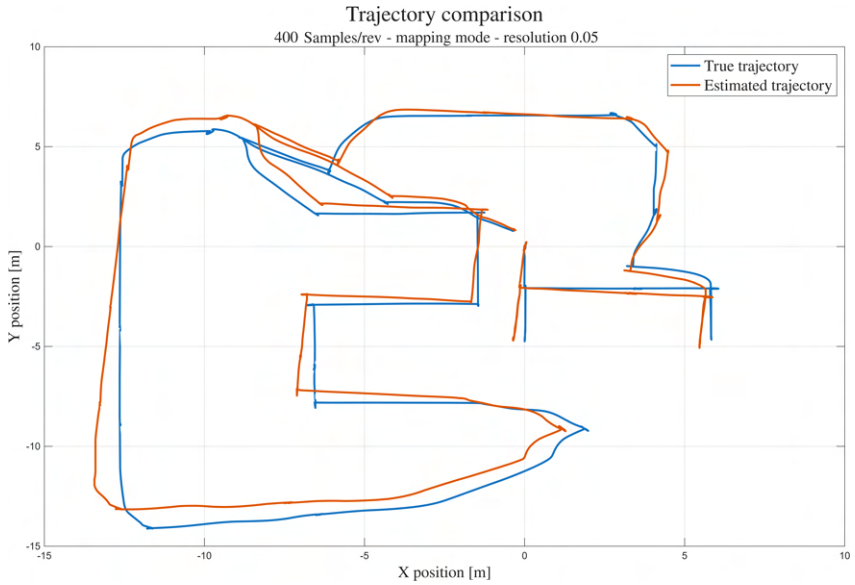


Figure 4.31: Trajectory comparison for navigation in mapping mode, resolution 0.05 and 400 *Samples/rev*

From this image it is clear that the maximum error was achieved in the left-down corner, which is also the most distant point from the origin and this was expected because the error built up during the navigation up until that point. Moreover, it is also visible the error reduction when the UAV return to known location, for example, after starting at the origin, the UAV goes backward and then flies forward for 2.5 m approximatevely, then it goes right and enters in the small room with only one entrance of the section "A". The error building here is increasing constantly but then, when the UAV exit the room and fly in the proximity of the origin at approximately coordinates (3,-1), the error decreases noticeably. After navigating and mapping the zone "A", it continues to the zone "B" and finally "C". In Fig.4.32 is reported the map

created by the SLAM node with the aforementioned parameter combination. As it possible to see from the picture, even for quite high maximum errors,

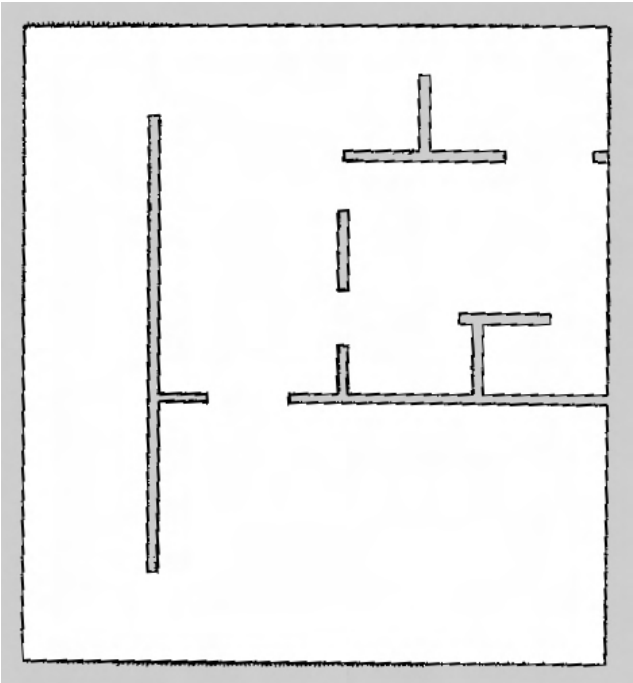


Figure 4.32: Output map from the SLAM algorithm

the map creation is not affected thanks to the so called "loop closure", which will be discussed in an other sections due to its importance. Small defects are visible in the most distance zones, where the thickness of the wall is greater and shows a "comb-like" texture facing outward. Since those defects are not inside the map environment, they do not pose serious consequences to the localization algorithm.

2. Mapping mode, resolution 0.01 - With this setup, the UAV was not able to perform a complete flight and resulted in a crash. The position comparison is reported below in Fig.4.33 and it is clear how to error increased and lead to the crash, which is represented by the spike in the last part of the flight. The simulation felt slower and the CPU consumption increased substantially. In fact, the reason behind the crash is what we called "Computational limitation" which is an effect due to limitation on simulation hardware computational capability. The SLAM algorithm in mapping mode adds constantly nodes to the graph and this increased the load on the CPU, arriving at the point where new scans are elaborated too slowly and the UAV just drifts away. This effect is clearer when looking in detail at the position comparison. In fact, as it is possible to see in Fig.4.34, at one point the EKF2 does not estimate anymore the position correctly and shows a plateau even if the true position is changing drastically over timel, resulting in a crash.

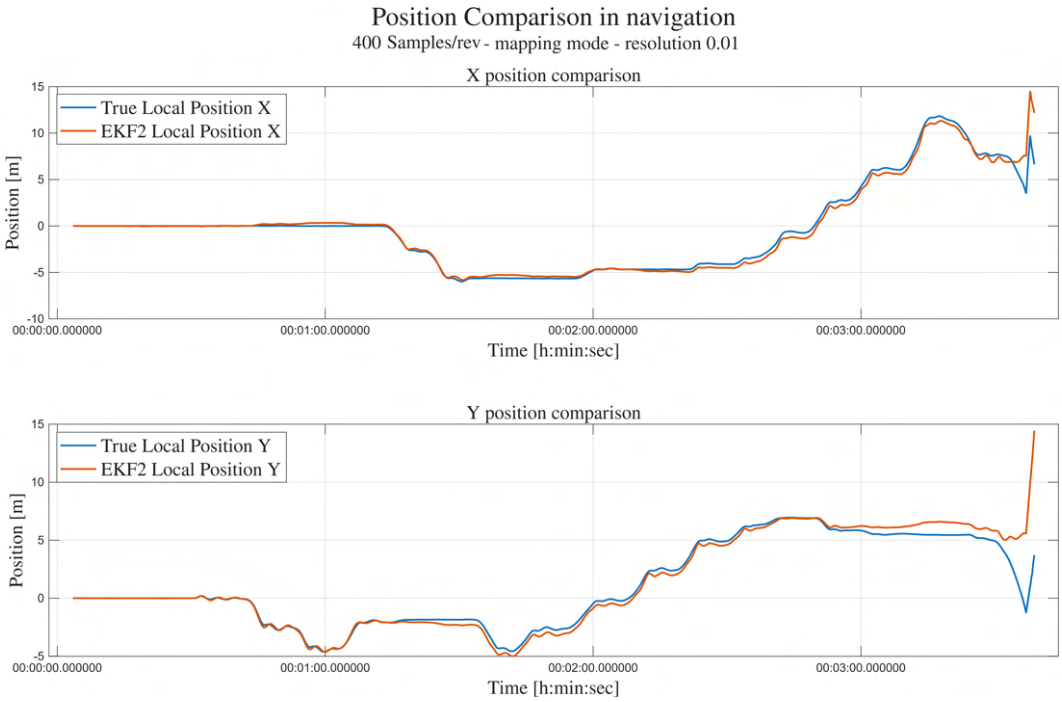


Figure 4.33: Position comparison of mapping and resolution 0.01 with 400 *Samples/rev*

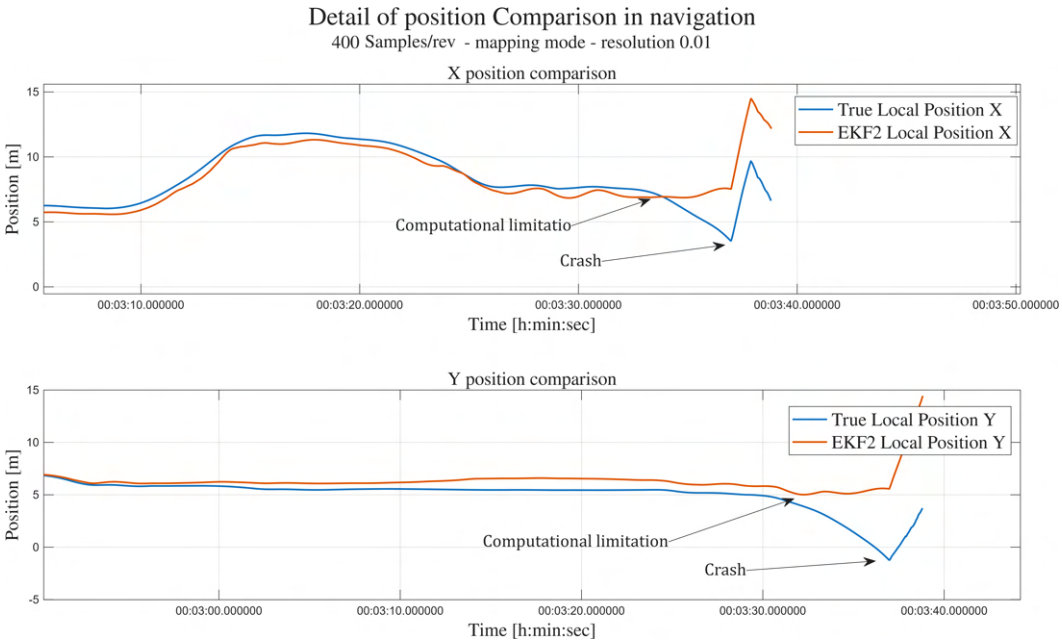
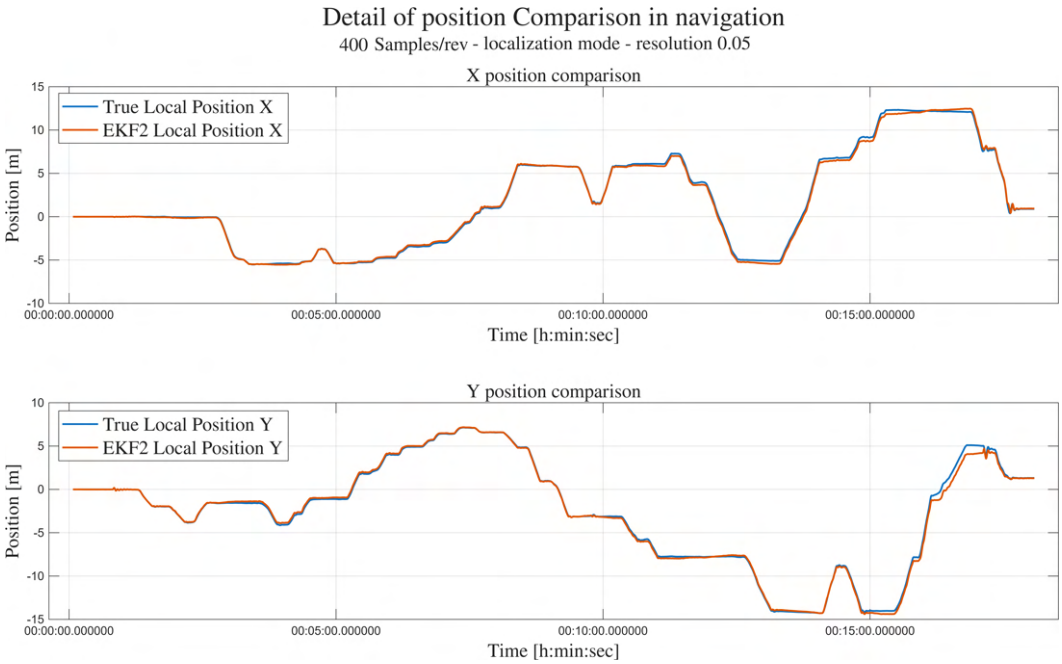
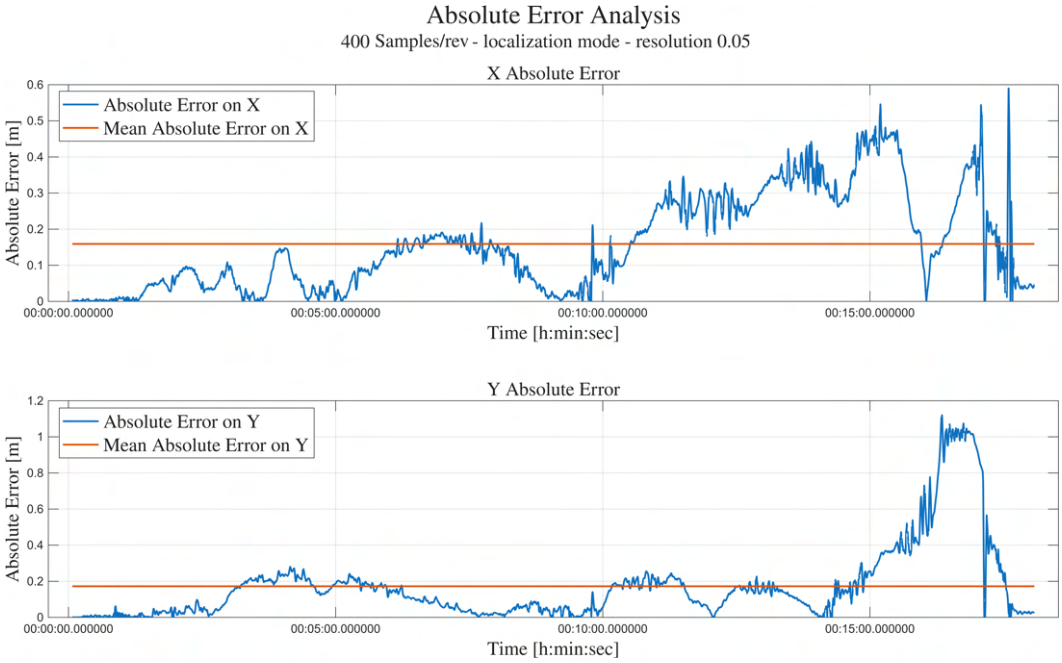


Figure 4.34: Detail of position comparison of mapping and resolution 0.01 with 400 *Samples/rev*

3. Localization mode, resolution 0.05



(a) Position comparison of localization and resolution 0.05 with 400 *Samples/rev*



(b) Absolute error analysis of localization and resolution 0.05 with 400 *Samples/rev*

Figure 4.35: Position and absolute error analysis of with localization and resolution 0.05 with 400 *Samples/rev*

This setup was able to perform a full flight and the result are shown in Fig.4.35. The position is tracked exceptional precision during the whole flight except in the zone "C". Here the aisle created some difficulties in the positioning system and it affected the absolute error. The system shows a mean absolute error of 0.159 in x and 0.172 in y, even if the error spike has a great impact. Here, it is clear that the feature of loop closure was activated and its result is the almost instantaneous reduction of the error. Thanks to its clarity, this will be taken as example in the loop closure section. Moroever, it is important to note that the map creation results of less quality and with more evident defects, it is reported below in Fig.4.36 and, even if defects are present, the map is in any case usable by the pilot. By comparing the map output between mapping and localization mode, it is clear that mapping creates a more detailed and less noisy map, but it is also interesting to notice how better the localization mode works with respect to the mapping one.

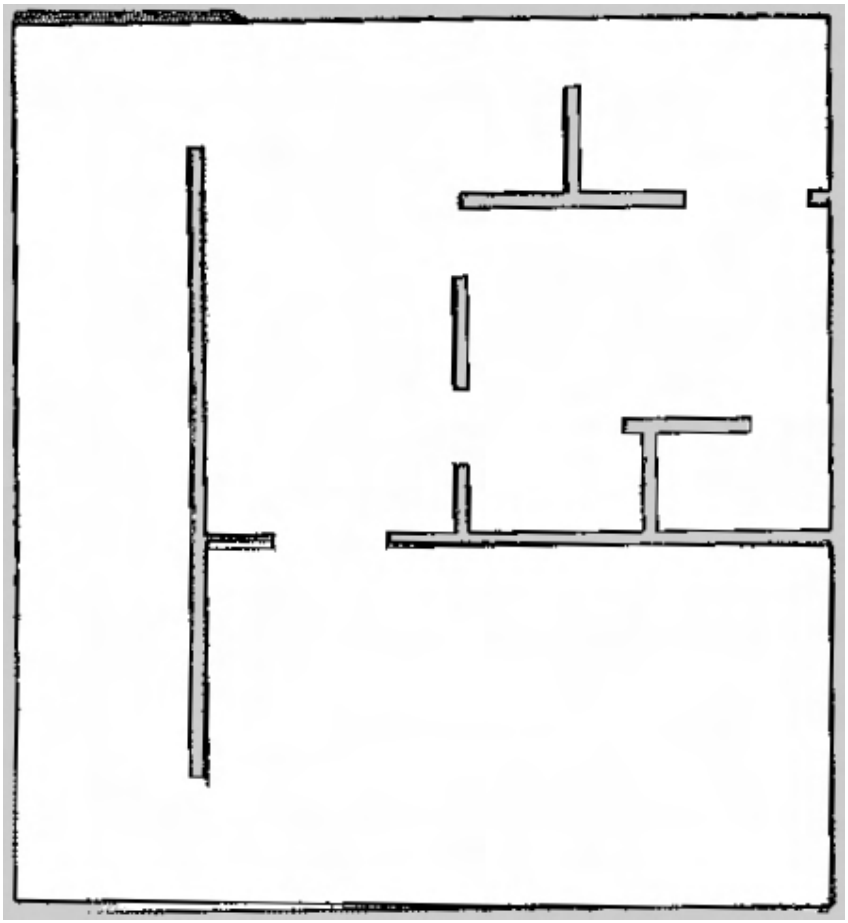


Figure 4.36: Output map from the SLAM node with localization mode and resolution 0.05 with 400 *Samples/rev*

4. Localization mode, resolution 0.01 - With this setup the UAV crashed after a very short flight again due to computational limitation of the simulating hardware. This is clear from the position analysis shown below where it is

possible to see the usual plateau that happens during this particular effect.

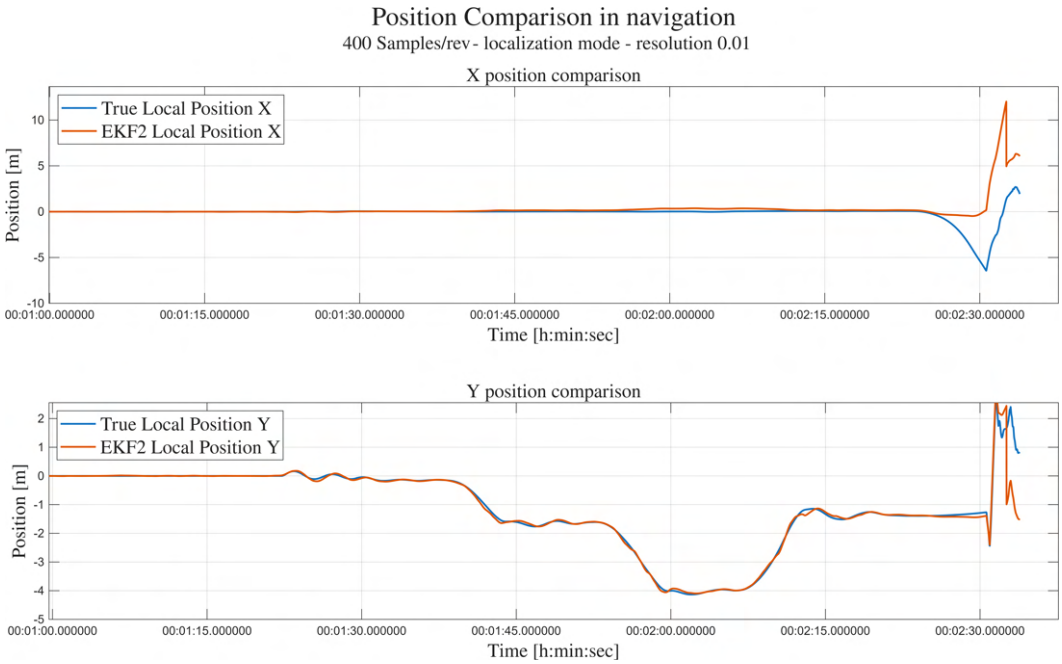


Figure 4.37: Position comparison of localization and resolution 0.01 with 400 *Samples/rev*

Concluding, in the case of a sample rate equal to 400 *Samples/rev*, flights with resolution of 0.05 can be performed either in localization or mapping mode and they yield very promising results, while with a better resolution the computational ccost becomes so high that the system starts to lag and then crashes.

4.4.2 Navigation with 800 Samples/rev

Similarly to what was done in the 400 *Samples/rev* case, in this subsection the navigation results with this sample rate are reported and analyzed.

1. Mapping mode, resolution 0.05 - with this setup, the UAV was able to flight and navigate around the first zone correctly. Then, while mapping the second zone, it showed signs of computation limitation as lagging in RVIZ2 and finally it crashed. The results of the flight are reported below. It is clear the plateau in correspondence of the computational limitation which caused the crash.

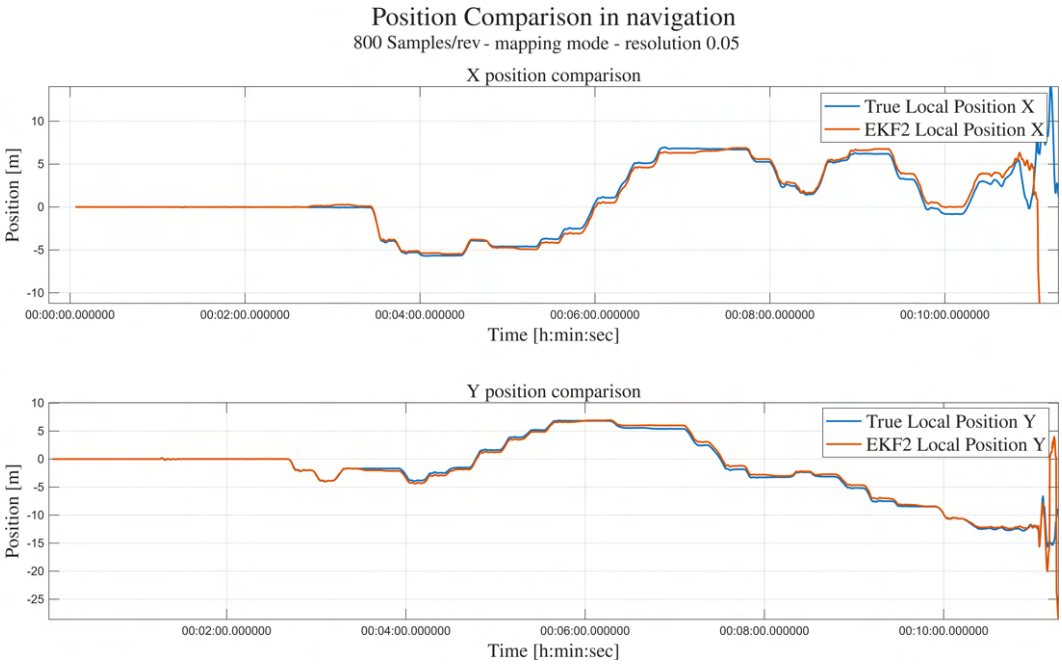


Figure 4.38: Position comparison of mapping and resolution 0.05 with 800 *Samples/rev*

Moreover, the CPU load is also reported as a screenshot from the system monitor that shows the computational cost of the SLAM algorithm. It is necessary to note that the computer not only has to run the SLAM node, but in addition it runs the simulator, the ground station and the other custom made nodes. The decrease of computational load visible at the end it is due to the stopping of the simulation.

Figure 4.39: CPU load during simmulation with mapping mode, resolution 0.05 and 800 *Samples/rev*

2. Mapping mode, resolution 0.01 - This setup is so computationally expensive that the UAV crashed after less than one minute of flight. RVIZ2 visualizer

started lagging during takeoff and after the first exploration maneuver it lost position reference and crashed. The results are shown below for completeness.

Figure 4.40: Position comparison of mapping and resolution 0.01 with 800 *Samples/rev*

3. Localization mode, resolution 0.05 - This simulation is very interesting because from its analysis various considerations can be deducted. The results are firstly presented in the following figure.

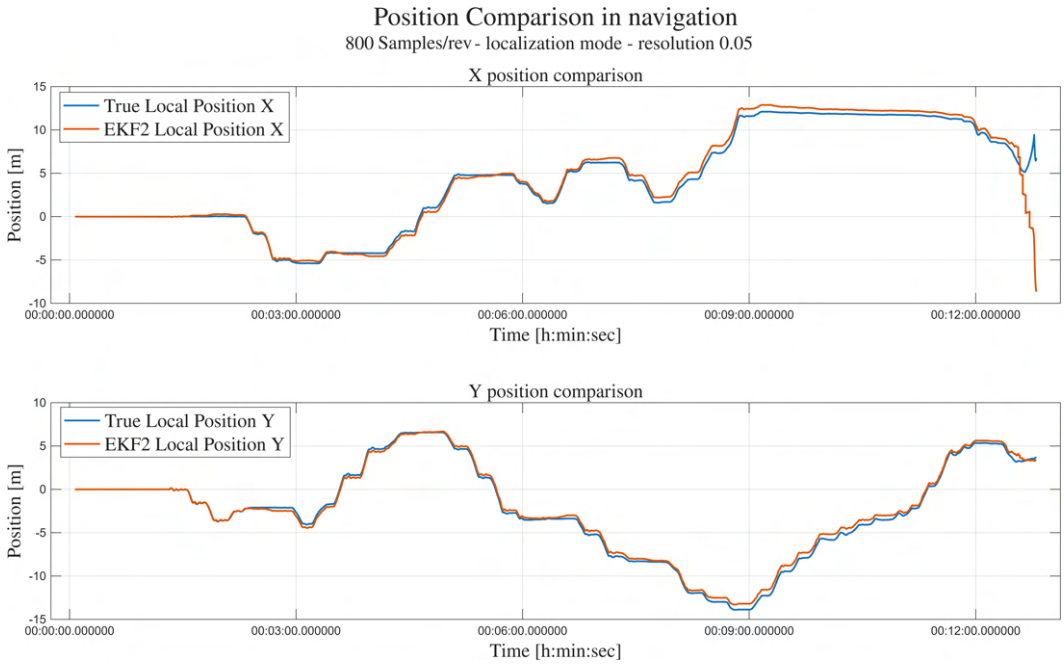


Figure 4.41: Position comparison of localization and resolution 0.05 with 800 *Samples/rev*

The first consideration is that this is the confirm that localization mode otpimizes the Ceres solver in such a way that it less computationally expensive, since this is the first setup with 800 *Samples/rev* that is almost able to complete a full flight without strong evidence of computational limitation. The second consideration is that loop closure can be a critical moment for the UAV. In fact, in this simulation it was possible to observe that, once the loop closure was completed, a strong discontinuity was applied in the positioning signal and that introduced high-amplitude oscillations that resulted in the crash. The actual map behavior will be analyzed in the loop closure section. The third consideration is that, even though the map was not created due to the crash, small defects could be found like in the 400 *Samples/rev* case in localization mode, which are typical of this optimization but do not inficiate the map usability. Anyway, this configuration can surely be used for simulation flights, perhaps with light tuning and parameter configurations.

4. Localization mode, resolution 0.01 - This setup simulation was able to takeoff and to perform the first exploration manouver, then it crashed due to computational limitation. Again, the crash could be anticipated by observing the lagging in RVIZ2.

4.4.3 Loop Closure

In SLAM algorithm, loop closure is the process of recognizing that the UAV has returned to a previously visited location and correcting accumulated pose error. Without loop closure, small odometry and scan-matching errors accumulate over time, causing map distortion and trajectory drift. Loop closure detection continuously compares the UAV LiDAR scans with previously stored observations to determine whether the present scene matches a past one. When a sufficiently strong match is found, the system deducts that the two poses correspond to the same physical place. This triggers the creation of a constraint in the pose graph, effectively "tying together" two distant points in time. The graph is then re-optimized so that the trajectory and map become globally consistent. Instead of only adjusting the current pose, the optimization redistributes the accumulated error across all poses back in time, shifting them to satisfy both odometry and loop constraints. The result is a corrected trajectory, reduced drift, and a map without large scale distortion. The map in Fig.4.42 is a coherent map locally, but not globally. The UAV can use it to avoid walls but clearly it doesn't match with the upper wall. This is what the pilot would see from RVIZ2, a two-dimensional map that shows the previous mapped features and the current laser scan, marked in red. Currently, RVIZ2 is visualizing just two topics, `/map` and `/laser_scan`, it would be possibile to visualize other topics such as odometry.

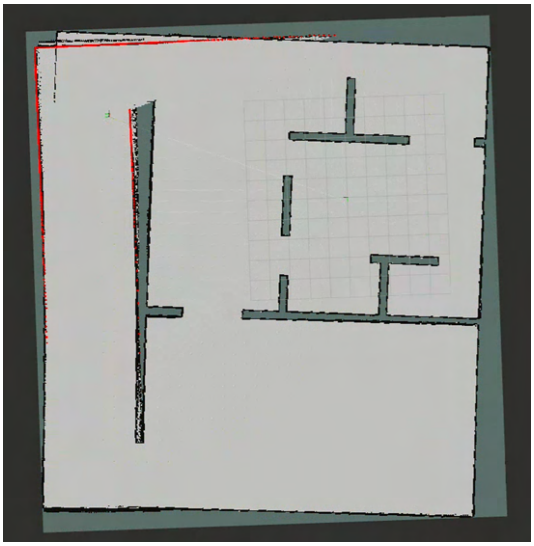


Figure 4.42: SLAM map at time 917s with localization mode, resolution 0.05 and 800 $Samples/rev$

At time 934s, just before the engagement of loop closure feauture, the map looks like 4.43, which is a totally inconsistent global map.

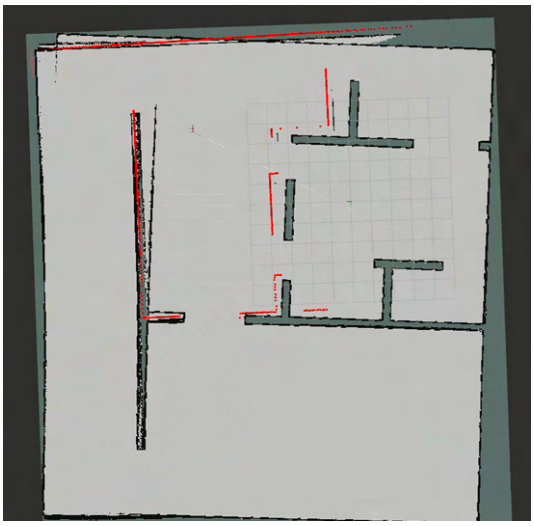


Figure 4.43: SLAM map at time 934s with localization mode, resolution 0.05 and 800 $Samples/rev$

At time 937s the loop closure feature activates and aligns the current scan match with known features and the created map is shown here 4.44. This almost istantaneous position change can be critical for the UAV, since it introduces strong oscillations and could be fatal for the flight, as happened in the simulation. This does not mean that each loop closure is fatal, in fact in other simulations, where the loop closure was not so evident like here, the process didn't result in crash but the UAV was still able to continue its mission.

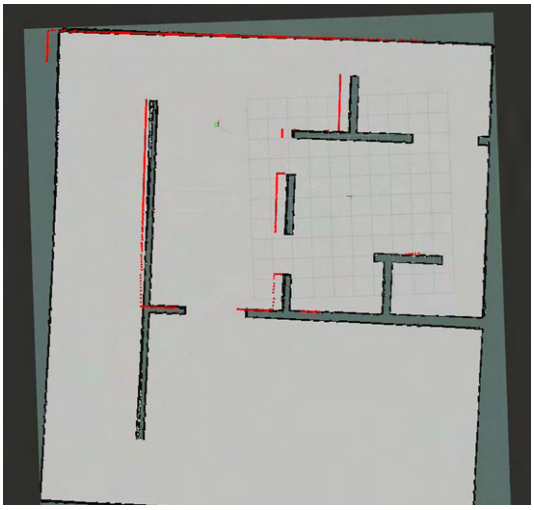


Figure 4.44: SLAM map at time 937s with localization mode, resolution 0.05 and 800 $Samples/rev$

4.4.4 Collision prevention

Collision Prevention (CP) is a native features of the PX4 flight control firmware and its mathematical model had been described before. It can be used during navigation within an environment and it allows the UAV to avoid touching obstacles and to keep a safe and user-definable distance by simply setting a parameter. The distance set in this numerical simulation was 1 m, thus the parameter is `CP_DIST = 1`. Moreover, the scale distance is measured as the ratio between two parameters, `MPC_VEL_MANUAL = 10` and `MPC_XY_P = 0.9`, which gives a value of 11.1 m. This a great distance and in fact the acceleration restriction is felt immedialetely and it is very clear from Fig.4.45, where both the original setpoints (the one imposed by the pilot via the radio transmitter or joysticks) and the adapted ones (after the Collision Prevention algorithm) are shown. It is possible to notice that the setpoints are immediately decreased since in both direction the obstacles are near the drone and the UAV is at a distance which is less than the Scale Distance. It is possible to note also that in few instances, the adapted setpoints had a negative value even if the setpoints were positive and quite high. This is because in those situation the UAV got closer to the wall than the imposed minimum distance of 1 m and thus the algorithm not only blocked the advance, but it also pushed the drone away from the obstacle. The other parameters to take into consideration are `CP_GO_NO_DATA`, which allows the drone to fly in directions were obstacle distances data are not present (this is not a problem for the selected 2D LiDAR sensor since it has a 360° visibility). The second parameter is `CP_GUIDE_ANGLE`, which indicates the angle (to both sides of the commanded direction) within which the vehicle may deviate if it finds fewer obstacles in that direction. This is a powerful features since it allows the drone to automatically deviate from an obstacle, it is particularly useful near doors or entrances, since it "pushes" the drone through the door without any effort from the pilot perspective.

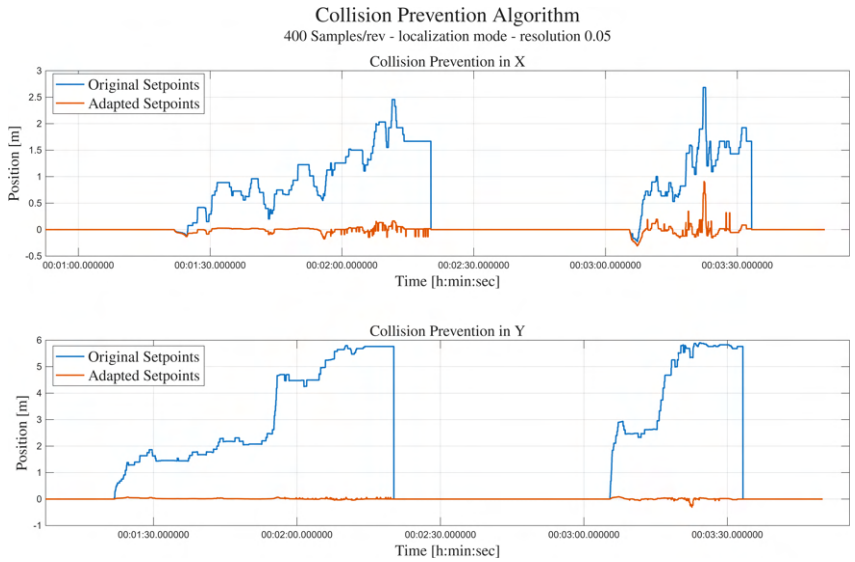


Figure 4.45: Collision Prevention algorithm original and adapted setpoints

4.4.5 Navigation consideration

Validating the system through nuemerical simulations is important and leads to a deeper understanding of the system behaviour. Navigation simulations in the custom-made Gazebo world highlighted several key phenomena and very important considerations can be deducted:

- Attitude has a great impact on the distance measurements. Due to this effect, navigation has to be gentle enough to avoid high attitude angle and mitigate this problem, for example it is important to not release the sticks instantly during a steady motion to avoid a sudden stop, which creates high angles.
- The best localization and mapping results are achieved by implementing a particular navigation strategy. It has been found that the best navigation strategy is applying carefully inputs to obtain a velocity of approximately 0.5 m/s (without creating high attitude angles) and keeping it for a small amount of time, then a few seconds hover is needed in order to allow the algorithm to correct the meaured distance of the walls.
- Computational limitation. High-resolution mapping, particularly when matched with high sample rates, triggered CPU load saturation. As the pose-graph grows, CPU consumption increases and can lead to lags and consequently crashes.
- Error accumulation and loop closure. During prolonged flights, positioning errors tended to build up due to uncertainties in the evolving map. However, the loop closure feature of SLAM significantly reduces these errors when the UAV returned to previously mapped locations. In one instance, a 1.2m accumulated error was corrected to just 0.05m upon returning to the takeoff point. This feature is surely useful but it has to be noted that introduces high amplitude oscillations and can be fatal for the drone.

- Native Collision Prevention Algorithm is particularly useful but it reduces drastically pilot's input in crowded and confined spaces where the obstacle is found at a distance inferior to the calculated scale distance.
- This 2D SLAM algorithm struggles in long and narrow aisles. In particular, when the UAV cannot sense anymore at least one wall in addition to the two aisle walls, it loses track of its position and results certainly in a crash. An anology to explain this situation can be formulated: let's imagine a person who enters in an aisle in the dark, only feeling the aisle with his hands but not able to look at the end of the aisle. By counting the steps it is easily possible to estimate the distance from the start of the aisle. If instead the floor is slippery (simulating IMU drifting), the estimation becomes more difficult and can lead to an erroneous result.

Chapter 5

Experimental Validation

Consequently to the validation of the proposed navigation architecture in the simulation environment, the following fundamental step consists in deploying the system on a real hardware platform and evaluating its performance under experimental conditions. While simulations provide a controlled and repeatable environment for algorithm development, real-world experiments introduce additional challenges like not-modeled sensor noise, communication delays, limited computational resources and environmental disturbances. For this reason, hardware deployment represents a crucial stage.

In this chapter, the algorithms previously developed and tested in simulation are integrated into the physical UAV platform, with the goal of verifying that the proposed SLAM-based navigation, together with the PX4–ROS2 architecture, can operate reliably on board the vehicle and provide a stable position estimate suitable for autonomous flight.

5.1 System Architecture

The experimental system architecture is similar to the one implemented in the numerical simulations and it is represented in Fig.5.1. In particular, here the system is divided based on which hardware platform the node or the algorithm is running. The Raspberry Pi5 board, represented in light blue, runs the ROS2 framework and its nodes: Sllidar for transforming data from the LiDAR sensor to a ROS2 topic, DronePose to transform PX4 Odometry uorb into a ROS2-compatible message, SLAM node that takes as input the `/tf` topic and the LiDAR data and by combining those two retrives the UAV position and the map of the environment and, finally, the Pose2odom node that takes the ROS2 topic `/pose` and translates it into a uorb message for PX4. The Pixhawk 6C flight controller board, represented in pink, runs the PX4 software. The external computers, which both run a Linux distribution, are represented in orange and are needed to obtain and save the MoCap data and to comunicate with the UAV and the ROS2 framework. From this perspective is easily understandable that the complexity of the experimental validation is much higher than the numerical validation. Infact, now different boards with different languages and settings need to cohexist in the same ecosystem and thus robust and reliable comunication protocols are needed.

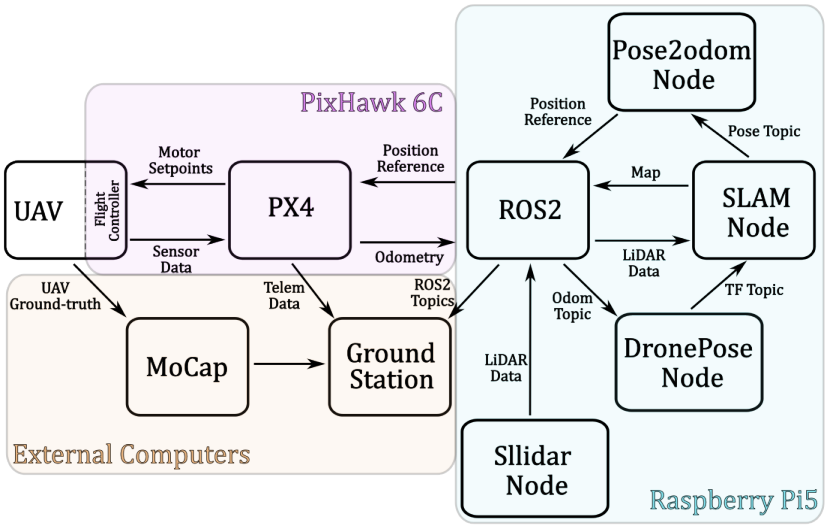


Figure 5.1: Experimental System Architecture

A more detailed discussion is needed for the Raspberry Pi5. The system was developed and tested in simulation with the Linux distribution Ubuntu 22.04. Unfortunately, during the set up phase of the board, it was discovered that since the Pi5 is equipped with a new and more powerfull processor, the operative system Ubuntu 22.04 is not supported and thus it was choosen to install a different OS, the one officially recommended is Ubuntu 24.04 LTS (Long-Term Support) "Noble Numbat". This OS does not support anymore the ROS2 version which was used in simulation, which is ROS2 "Humble", but only supports the ROS2 version "Jazzy Jalisco". In order to port the ROS2 packages from the older to the newer version, a GitHub repository called "Slam2PX4" was created and it will be publicly available in the next future for all the user. This repository was cloned into a local copy on the Raspberry Pi5 using the command:

```
git clone https://github.com/Lingua00/Slam2PX4.git
```

After that it needs to be built with the `colcon` command and then the workspace has to be sourced and the package can be utilized by the user. The aforementioned steps were perfomed while using a monitor and while powering the Raspberry Pi5 from a Power Supply. However, during flights, both this tools can not be used. This brings up two problem that were solved as follows:

1. Powering the system. Instead of using a power supply, it was decided to use the same battery that power the Pixhawk 6C and the motors to power the Raspberry, because the higher energy consumption was valued as less problematic than using a second battery just to power this board. In order to use the same battery, a BEC was needed and it was chosen accordingly to the recommended voltage and current characteristics of the Raspberry Pi5. This BEC was then soldered to the existing XT60 connector and its power lines were soldered into a USB-C cable, in order to use the already present

safety features of the Raspberry in case of over voltages or current spikes since powering the board from its GPIOs is possible but more hazardous.

2. Remote Access. Instead of using a monitor to visually use the Raspberry Ubuntu graphical interface, a protocol called "Secure SHell" (SSH) was used. Through the same WiFi connection, it allows the ground station to connect to the Raspberry and to use locally visualize the terminal of the Raspberry. It needs to be set up via IP address and thus it is recommended to assign to the Raspberry a static IP address, which can be easily done via the router configurator.

The last aspect to take into consideration is the communication between the Pixhawk and the Raspberry. Following the virtual guide it is possible to physically connect the two boards through a custom-made wire that connect the Pixhawk `telem2` port with the Raspberry GPIOs as follow

PX4 Telem2 pin	Raspberry Pi pin
UART5_TX (2)	RXD (GPIO 15 - pin10)
UART5_RX (3)	TXD (GPIO 14 - pin8)
GND (6)	Ground (pin 6)

Table 5.1: Pixhawk-Raspberry physical connection

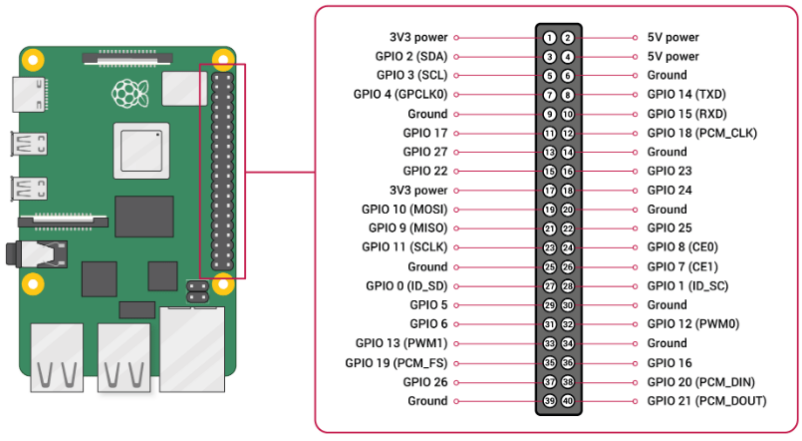


Figure 5.2: Raspberry Pi5 GPIO

5.2 Experimental Setup

The first experimental flight was performed in the CICLoPE facility in Predappio, in the hills above Forlì, where the once Caproni airplanes assembly lines were revalued into a university research facility with a wind tunnel and a newly-constructed drone cage. This last one is equipped with a Motion Capture system that allows the precise tracking of different drones via the software Motive. The drone cage and the cameras that constitutes the MoCap system are shown in Fig.5.3.

Figure 5.3: Drone Flight Arena and the MoCap system cameras

Before its use, the MoCap system needs to be calibrated with a "wand", which is a plastic stick with different markers mounted at different heights and angles, which allows the calibration process to be performed. After a quite fast calibration of few minutes, the precision of the system was in the order of less than 1 mm, which is outstanding. The data collected by the MoCap is not used as the position reference taken as input by the PX4 EKF, but rather it is used as a "ground-truth" to evaluate the performance of the SLAM positioning algorithm. The last step before perfoming the experimental flight is applying the markers to the UAV, the infrared markers are tracked by the MoCap cameras and they need to be fixed and "randomly" placed on the UAV. In order to provide a static mouting position, two elements were designed in Solidworks and later made with a Filament Deposition Method by using a carbon fiber reinforced filament and the Raise3D E2CF printer. Two markers were applied on a flat board also used as mounting platform for the LiDAR sensor and four markers were applied in the motor arms at random position and were locked in position with a custom made C-element. In order to offer to the SLAM algorithm some obstacles to detect and map, cardboard boxes were used, forming high and narrow objects (Fig.5.4).

Figure 5.4: Cardboard obstacles inside the drone flight arena with the UAV ready to fly

The first flights were succesfully performed with this configuration, but analyzing the results it was choosen to implement a cloth element $1.5m$ high in order to avoid the net interfere with the LiDAR sensor measurements (Fig.5.5).

Figure 5.5: Drone cage configuration with the cloth solution to avoid net interference

5.3 Experimental Results

The first validation flight was performed with a LiDAR sample rate of 800 $Samples/rev$, resolution of $0.05m$ and mapping mode. The UAV was flown in Position flight mode from takeoff to land and the results are shown in Fig. 5.6, while the maximum absolute error and the mean absolute error are reported in the table below:

Mapping mode $800S/rev$, res $0.05m$	X direction	Y direction
Maximum Absolute Error $[m]$	0.240	0.396
Mean Absolute Error $[m]$	0.083	0.273

Table 5.2: Maximum and Mean Absolute error for the first fligh - 800 $Samples/rev$, Mapping mode and resolution $0.05m$

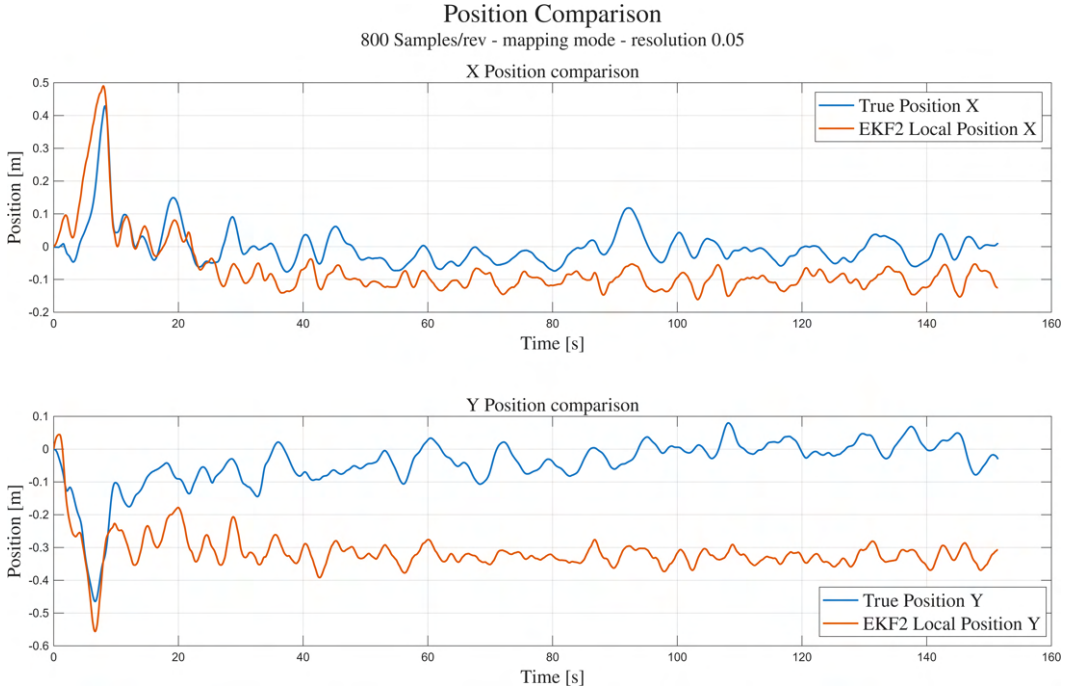
The mean absolute error in the forward direction is optimal and it is in the order of a RTK GNSS system with its centimetric precision of $8.3cm$. However, in the Y direction the results are not as good as in the X direction but are still acceptable. Infact, oscillations along that direction was observed during the flight and it firstly drifted from the origin but then it stabilized itself around $30cm$ from the origin.

The second flight was performed with the same configuration as before but in localization mode. The results are shown in Fig. 5.7 and the Maximum and Mean Absolute Error of the flight is reported in the following table:

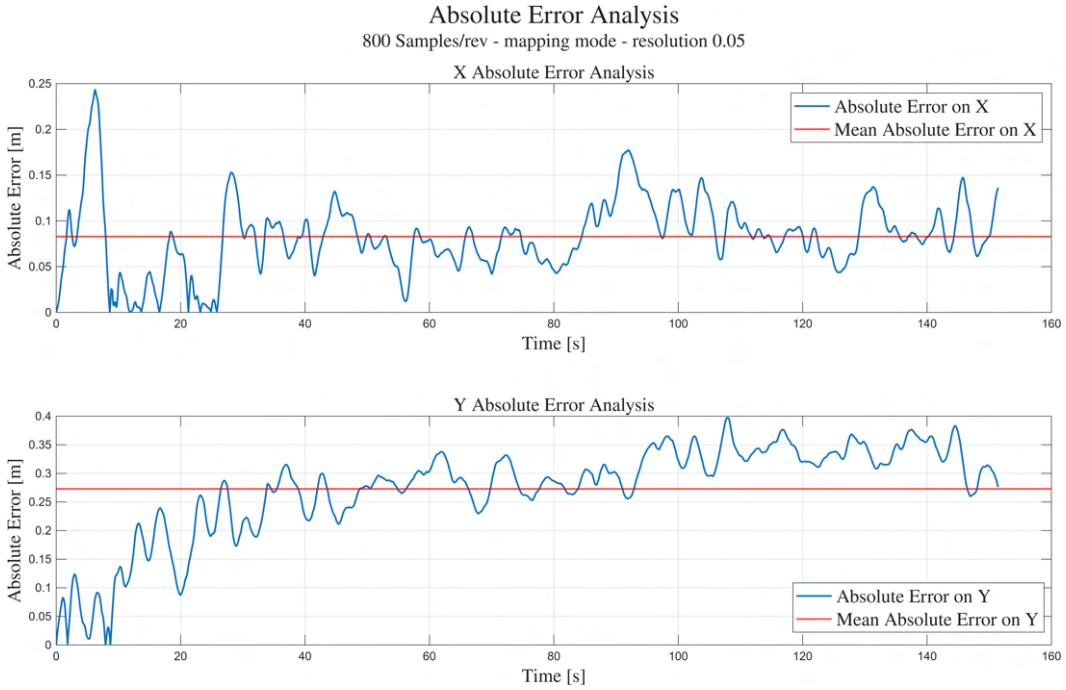
Localization mode $800S/rev$, res $0.05m$	X direction	Y direction
Maximum Absolute Error $[m]$	0.247	0.211
Mean Absolute Error $[m]$	0.051	0.105

Table 5.3: Maximum and Mean Absolute error for the first fligh - 800 $Samples/rev$, Localization mode and resolution $0.05m$

In this second flight the results are overall better than the first one. Infact, the Mean Absolute Error is decreased in both directions and it is respectively $5cm$ and $10cm$ in X and Y. This time both comparable to a RTK GNSS system precision.

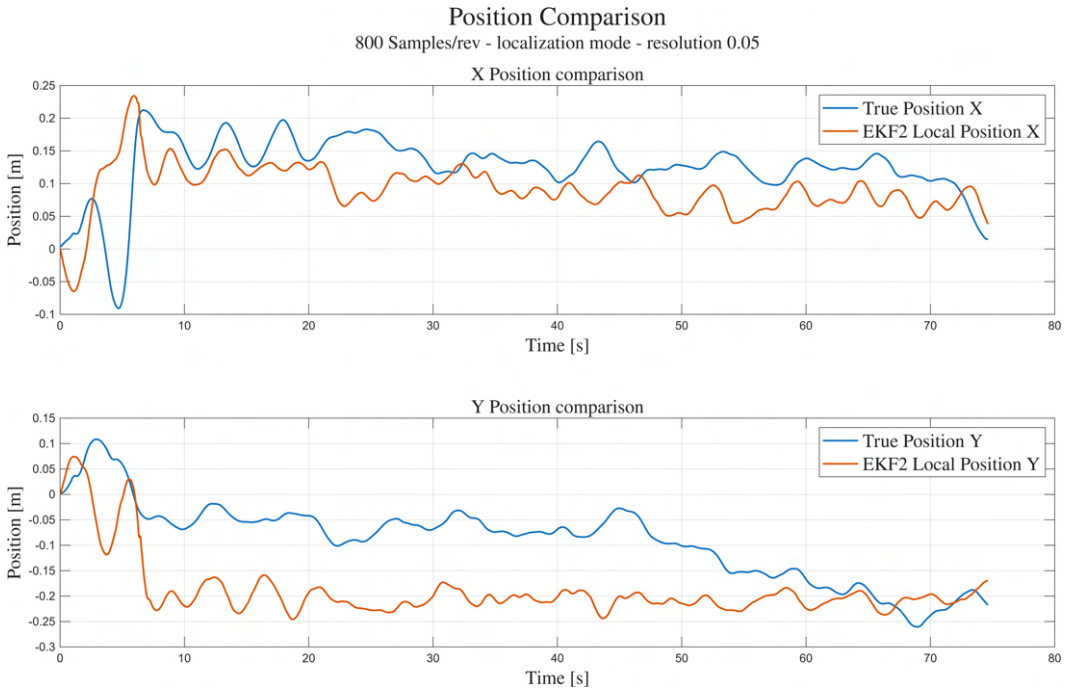


(a) Position comparison

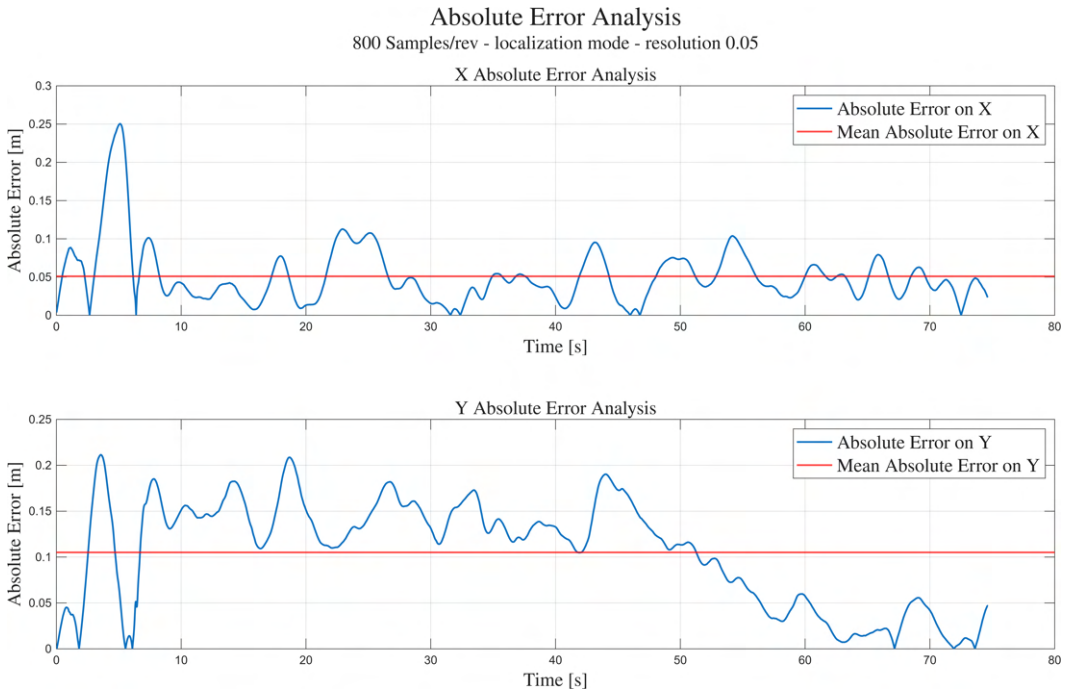


(b) Absolute error analysis

Figure 5.6: Position and absolute error analysis of the first flight with configuration 800 $Samples/rev$, mapping mode and resolution $0.05m$



(a) Position comparison



(b) Absolute error analysis

Figure 5.7: Position and absolute error analysis of the first flight with configuration 800 $Samples/rev$, localization mode and resolution $0.05m$

Following the first two flights, a new experimental campaign was carried out with a small change in the setup. From the map creation, it was possible to observe how the net had a strong interference at the boundaries of the map. The LiDAR sensor usually passed through the net but in some occasion it detected the strings and this created the boundary interference. To avoid this effect, a tightly woven cloth was added and locked to the net. To further improve the stability some weights were added in the bottom part to avoid oscillations due to wind created by the UAV propellers. This second flight campaign was performed aiming to test higher resolution settings, since the interference effects are avoided. The first flight was performed with 800 $Samples/rev$, mapping mode and resolution $0.005m$, the results are shown in Fig. 5.8 and the absolute errors are reported in the table.

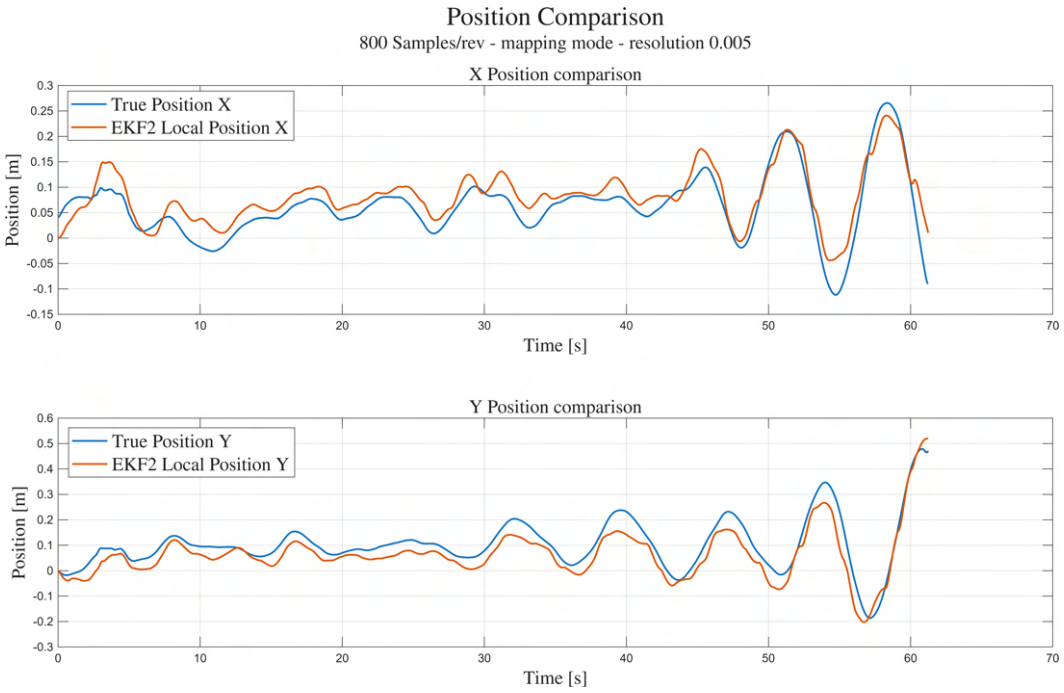
Mapping mode $800S/rev$, res $0.005m$	X direction	Y direction
Maximum Absolute Error $[m]$	0.107	0.144
Mean Absolute Error $[m]$	0.027	0.042

Table 5.4: Maximum and Mean Absolute error for the first fligh - 800 $Samples/rev$, Mapping mode and resolution $0.005m$

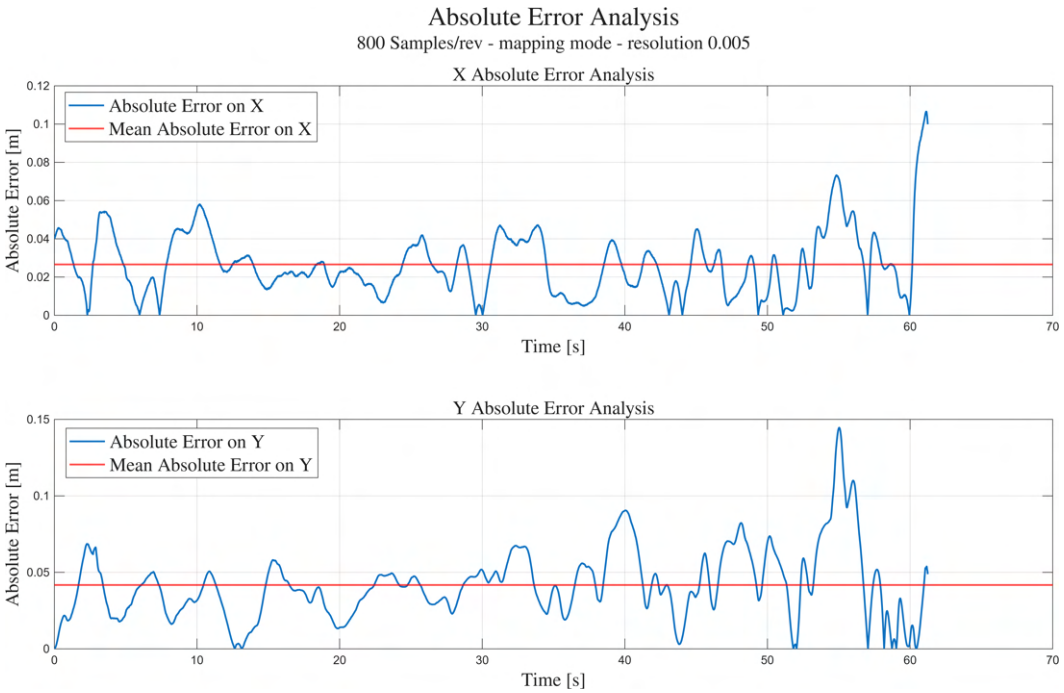
This results are impressive, the developed system is able to localize itself with a mean absolute error of $2.7cm$ in X and $4.2cm$ in Y. The same experiment was performed with the localization mode (Fig. 5.9) and the results are expressed in the following table.

Localization mode $800S/rev$, res $0.005m$	X direction	Y direction
Maximum Absolute Error $[m]$	0.085	0.074
Mean Absolute Error $[m]$	0.032	0.018

Table 5.5: Maximum and Mean Absolute error for the first fligh - 800 $Samples/rev$, Localization mode and resolution $0.005m$

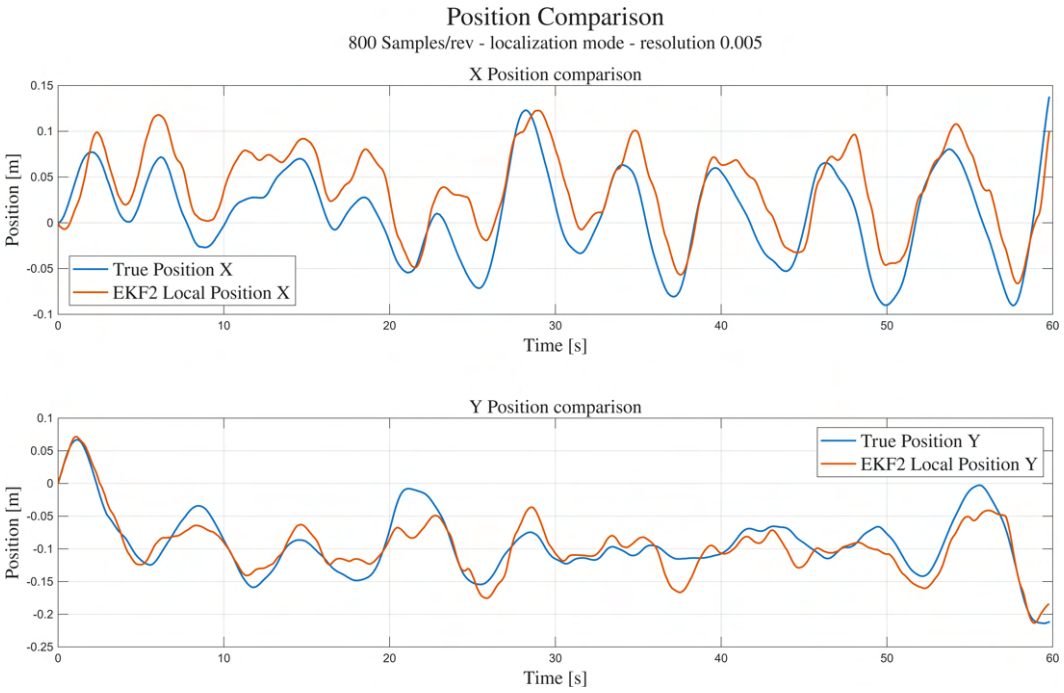


(a) Position comparison

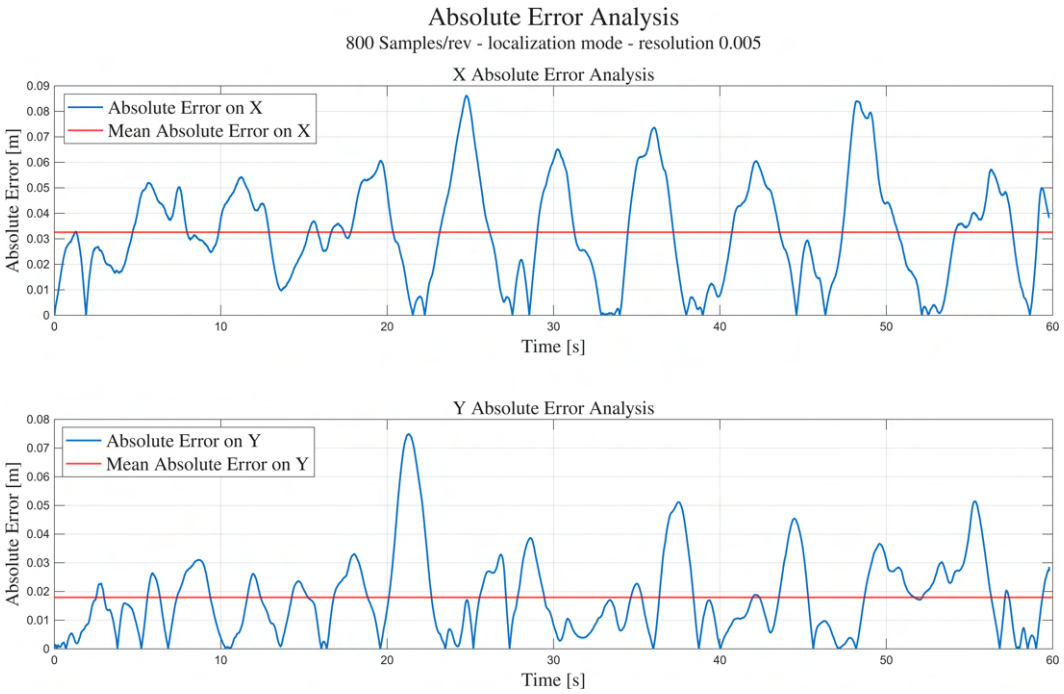


(b) Absolute error analysis

Figure 5.8: Position and absolute error analysis of the first flight with configuration 800 $Samples/rev$, mapping mode and resolution $0.005m$



(a) Position comparison



(b) Absolute error analysis

Figure 5.9: Position and absolute error analysis of the first flight with configuration 800 $Samples/rev$, localization mode and resolution $0.005m$

The two flights with higher resolution yield better results than the ones with lower resolution. In both optimization mode, the mean absolute errors are exceptionally low and in the Y direction of the localization mode it is observed the smallest mean absolute error, which is $1.8cm$.

5.4 Results Discussion

In this section, the results previously reported are discussed and analyzed. The considerations that are possible to deduct are the following:

1. The flights performed in the condition without the cloth demonstrated how the localization mode yields better results in terms of absolute error compared to the mapping mode. The mean absolute error showed a reduction of 38% on X and 61% on Y.

2. Since it was observed a strong interference of the net by looking at the map created by the SLAM node, it was chosen to apply a cloth in order to obtain clearer LiDAR measurements. Thanks to this solution it was possible to express the full potentiality of the SLAM algorithm by setting a higher resolution of $0.005m$, which gave exceptionally low absolute error values both in mapping and localization mode. In particular, with the localization mode was observed the smallest mean absolute error of just $1.8cm$. In Fig. 5.10

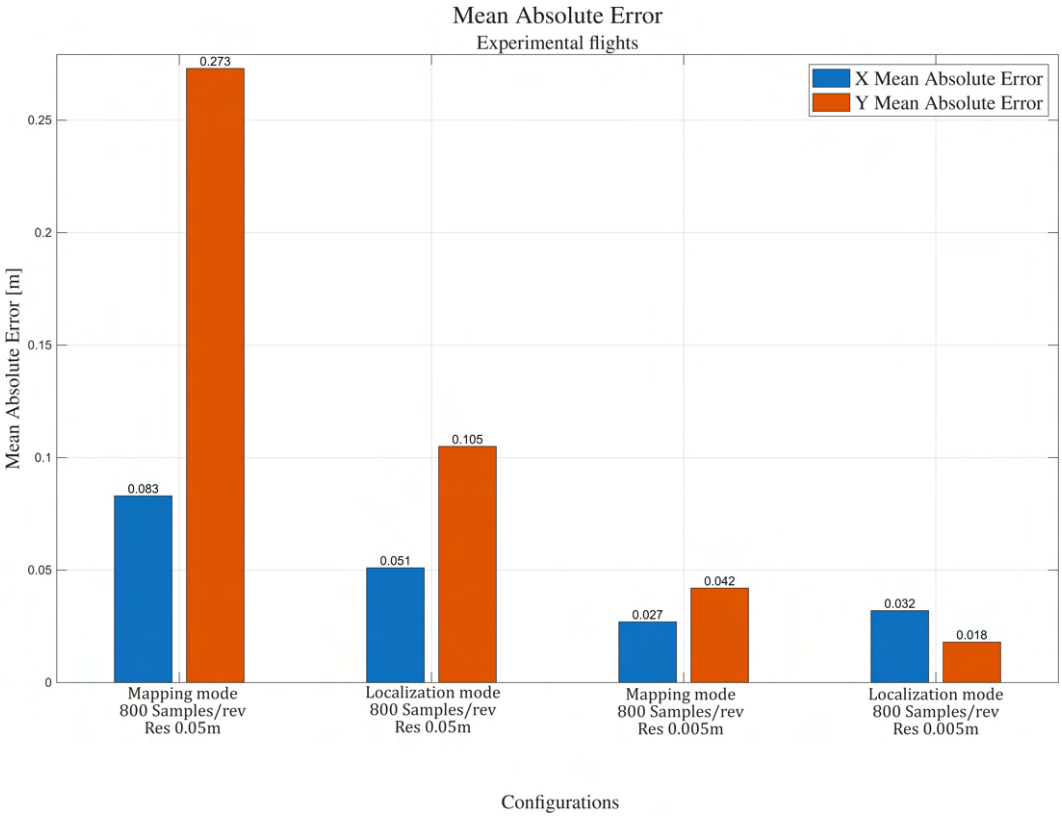


Figure 5.10: Mean Absolute Error for the experimental validation flights

3. The application of the cloth brought two noticeable effects that are important to take into consideration. The first one is beneficial and it is the reduction of the noise due to the net interference, which results in a much smoother position estimation which allows the UAV to reduce the small-amplitude high-frequency vibrations, that surely allows the system to increase its precision. The second effect is detrimental for system and it is the introduction of low-frequency oscillations, which are caused by the motion of the net due to the propellers effect. Those oscillations are immediately visible in both figures 5.8 and 5.9 and to prevent a crash it was choosen to land as soon as this effect started to appear. However, this behaviour is just related to the net motion and thus it will not be present in standard flight conditions. A detail of the cloth motion is shown in Fig. 5.11.

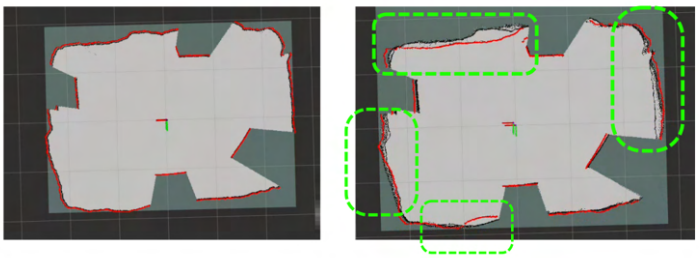


Figure 5.11: Detail of cloth motion during experimental flights

Where the left image is the map created by standing still not flying and the right one is the map during the hover. The green dotted boxes shows the worst-behaviour points where the black boundary is the map created by previous scanning and the red points are the current LiDAR measurements.

Chapter 6

Conclusions and Future Works

This thesis investigated the design and experimental validation of an indoor navigation framework for Unmanned Aerial Vehicles operating in GNSS-denied environments. The increasing diffusion of UAV applications in inspection, exploration and monitoring tasks requires robust localization and navigation capabilities even in environments where satellite-based positioning systems are unavailable or unreliable. For this reason, the present work focused on the integration of non-conventional sensors and advanced navigation algorithms with a commercial flight control firmware: PX4.

6.1 Conclusions

This work began with a bibliographic and commercial research of the most common non satellite-based navigation systems and algorithms. A comparative assessment of several indoor positioning technologies, including motion capture systems, computer vision approaches, optical flow sensors, and LiDAR, was performed. Based on reliability, cost effectiveness, and omnidirectional sensing capability, LiDAR technology was identified as the most suitable solution for the targeted application scenarios. Following this analysis, a complete UAV system architecture was designed. The selected platform consisted of a Holybro X500 V2 quadcopter equipped with a Pixhawk 6C flight controller, a Slamtec RPLiDAR A1M8 sensor, and a Raspberry Pi 5 used as a companion computer.

A key component of the work was the development of a simulation environment that allowed the integration of the flight controller, middleware, sensors, and navigation algorithms. Using Gazebo together with PX4 and ROS2, a realistic Software-In-The-Loop environment was created, including a custom world designed to reproduce different navigation challenges such as narrow corridors, large open spaces, and cluttered environments. The UAV model and sensors were carefully configured to match the characteristics of the real platform, enabling accurate performance evaluation prior to experimental deployment.

Extensive simulation campaigns were carried out to analyze the performance of the SLAM-based navigation system. Particular attention was given to the influence of LiDAR sampling density, map resolution, and algorithm optimization mode on the localization accuracy. Hover tests demonstrated that increasing the number of

samples per revolution improves the stability and precision of the position estimate, although it also increases computational load which can lead to computational limitation. Navigation experiments confirmed the ability of the SLAM algorithm to build consistent maps of the environment and to maintain a reliable pose estimate even in complex scenarios. Additionally, the integration with the PX4 collision prevention module allowed the UAV to adapt its motion in the presence of obstacles, increasing operational safety.

After validating the system in simulation, the algorithms were deployed on the real UAV platform for experimental testing. The experimental setup consisted of a controlled indoor flight area equipped with a motion capture system used as ground-truth reference. The results obtained during flight experiments confirmed the feasibility of the proposed architecture. The UAV was able to maintain stable flight and perform mapping and localization tasks using only onboard sensors and computation. The comparison between SLAM-based estimates and the reference measurements demonstrated exceptional levels of positioning accuracy for indoor navigation tasks during hover.

Overall, the work presented in this thesis demonstrates that low-cost LiDAR sensors combined with modern open-source robotics frameworks can provide a viable solution for GNSS-independent UAV navigation. The proposed system successfully integrates sensing, estimation, and control components within a unified architecture capable of supporting autonomous flight in indoor environments.

6.2 Future Works

Although the results obtained in this work are promising, several aspects could be further investigated to enhance the system capabilities, robustness and applicability in more complex operational scenarios. The following research directions represent the most relevant extensions of the present work.

1. Trigonometric correction for attitude–independent measurements. A first area of improvement might be the development of a trigonometric correction method aimed at reducing the dependency of LiDAR measurements on the UAV attitude configuration. During flight, variations in roll and pitch angles introduce distortions in the perceived geometry of the environment. By incorporating trigonometric corrections based on the vehicle's attitude estimate, it would be possible to compensate for these geometric distortions and obtain more consistent range measurements. This correction could significantly improve map consistency and localization accuracy.

2. Outdoor test navigation. Another important extension of this work is the validation of the proposed navigation framework in real-world environments outside the controlled motion capture cage. While the indoor setup allowed precise ground truth measurements and safe testing conditions, real environment introduces additional challenges such as irregular geometries, larger operational areas and environmental disturbances.

3. Real-world testing of PX4 collision prevention. The collision prevention module implemented in the PX4 flight stack was evaluated exclusively in simulation.

Future work should therefore focus on testing this functionality directly on the real UAV platform. Real-world testing would allow verification of the responsiveness and reliability of the collision avoidance behavior when interacting with actual obstacles. Such experiments would also help identify limitations related to sensor latency and to the UAV terminal velocity, improving the safety of autonomous flight operations.

4. Advanced guidance algorithms based on the pose-graph map. Another interesting direction involves the development of more advanced guidance and path planning algorithms that exploit the pose-graph map generated by the SLAM system. Algorithms such as Dijkstra-based shortest path planning or L1 guidance strategies could be implemented to allow the UAV to autonomously compute optimal trajectories between waypoints while considering the known map structure.

5. Improved collision avoidance strategies. While the current implementation relies mainly on the PX4 collision prevention functionality, more advanced collision avoidance methods could be explored. Techniques such as the potential field method could provide smoother and more efficient avoidance behaviors. These approaches could also enable the UAV to react more intelligently to moving obstacles or cluttered environments, improving both safety and navigation efficiency.

6. Extension of the framework to a 3D LiDAR sensor. Ultimately, an important future development is the application of the fully modular framework developed in this thesis to a three-dimensional LiDAR sensor. Integrating a 3D LiDAR would enable full volumetric mapping and significantly enhance the perception of complex environments with vertical structures. The ROS2 packages were implemented with a modular architecture and thus the transition to a 3D sensor should require limited structural changes while providing substantial improvements in environmental awareness and navigation capabilities.

Overall, these research directions would further increase the autonomy, robustness, and applicability of the proposed UAV navigation system, paving the way toward more advanced autonomous aerial operations in complex GNSS-denied environments.

Bibliography

[1] Y. Alkendi, L. Seneviratne, and Y. Zweiri, "State of the art in vision-based localization techniques for autonomous navigation systems," *IEEE*, 2021.

[2] D. Nistér, O. Naroditsky, and J. Bergen, "Visual odometry," *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2004.

[3] D. Scaramuzza and F. Fraundorfer, "Visual odometry [tutorial]," *IEEE Robot. Automat. Mag.*, 2011.

[4] C. Forster, M. Pizzoli, and D. Scaramuzza, "Svo: Fast semi-direct monocular visual odometry," *2014 IEEE International Conference on Robotics & Automation (ICRA)*, 2014.

[5] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *IEEE*, 2018.

[6] J. Naumann, B. Xu, S. Leutenegger, and X. Zuo, "Nerf-vo: Real-time sparse visual odometry with neural radiance fields," *IEEE Robotics and Automation Letters*, 2024.

[7] C. Debeunne and D. Vivet, "A review of visual-lidar fusion based simultaneous localization and mapping," *mdpi - Sensors*, 2020.

[8] R. Chatila and J.-P. Laumond, "Position referencing and consistent world modeling for mobile robots," *IEEE*, 1985.

[9] Davison, "Real-time simultaneous localisation and mapping with a single camera," *IEEE*, 2003.

[10] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "Monoslam: Real-time single camera slam," *IEEE*, 2007.

[11] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "Orb-slam: A versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, 2015.

[12] J. Zhang and S. Singhs, "Low-drift and real-time lidar odometry and mapping," *Auton Robot*, 2017.

[13] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE Transactions on Robotics*, 2007.

[14] S. Kohlbrecher, O. V. Stryk, J. Meyer, and U. Klingauf, "A flexible and scalable slam system with full 3d motion estimation," *IEEE*, 2011.

[15] W. Hess, D. Kohler, H. Rapp, and D. Ando, "Real-time loop closure in 2d lidar slam," *IEEE*, 2016.

[16] K. Konolige, G. Grisetti, R. Kümmerle, W. Burgard, B. Limketkai, and R. Vincent, "Efficient sparse pose adjustment for 2d mapping," *IEEE*, 2010.

[17] S. Macenski and I. Jambrecic, "Slam toolbox: Slam for the dynamic world," *JOSS*, 2021.

[18] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*. Springer, 2016.

[19] Council of European Union, "Council regulation 2018/1139 (article 3(32))," 2018.
`https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32018R1139`.

[20] R. Brena, JuanPabloGarcía-Vázquez, C. Galván-Tejada, D. Muñoz-Rodriguez, C. Vargas-Rosales, and J. Fangmeyer, "Evolution of indoor positioning technologies: A survey," *Journal of Sensors*, 2017.

[21] H. Wang, C. Chen, Y. He, S. Sun, L. Li, Y. Xu, and Y. B. Easy, "A low-cost and easy-to-use motion capture system for drones," *Drones*, 2024.

[22] M. Rodari, "Studio e sperimentazione di algoritmi di navigazione per velivoli multirotore in ambiente indoor," Master's thesis, University of Bologna, 2025.

[23] M. U. Khan, S. A. A. Zaidi, A. Ishtiaq, S. U. R. Bukhari, S. Samer, and A. Farman, "A comparative survey of lidar-slam and lidar based sensor technologies," *IEEE*, 2021.