A Study on UWB-Aided Localization for Multi-UAV Systems in GNSS-Denied Environments

Master of Science Thesis
University of Turku
Department of Future Technologies
Turku Intelligent Embedded and Robotic Systems (TIERS) Lab
2020
Carmen Martínez Almansa

Supervisors:
Msc. Jorge Peña Queralta
Assoc. Prof. Tomi Westerlund

UNIVERSITY OF TURKU
Department of Future Technologies

CARMEN MARTÍNEZ ALMANSA: A Study on UWB-Aided Localization for Multi-UAV Systems in GNSS-Denied Environments

Master of Science Thesis, 54 p.
Turku Intelligent Embedded and Robotic Systems (TIERS) Lab
August 2020

Unmanned Aerial Vehicles (UAVs) have seen an increased penetration in industrial applications in recent years. Some of those applications have to be carried out in GNSS-denied environments. For this reason, several localization systems have emerged as an alternative to GNSS-based systems such as Lidar and Visual Odometry, Inertial Measurement Units (IMUs), and over the past years also UWB-based systems. UWB technology has increased its popularity in the robotics field due to its high accuracy distance estimation from ranging measurements of wireless signals, even in non-line-of-sight measurements. However, the applicability of most of the UWB-based localization systems is limited because they rely on a fixed set of nodes, named anchors, which requires prior calibration. In this thesis, we present a localization system based on UWB technology with a built-in collaborative algorithm for the online autocalibration of the anchors. This autocalibration method, enables the anchors to be movable and thus, to be used in ad-doc and dynamic deployments. The system is based on Decawave's DWM1001 UWB transceivers. Compared to Decawave's autopositioning algorithm we drastically reduce the calibration time while increasing accuracy. We provide both experimental measurements and simulation results to demonstrate the usability of this algorithm. We also present a comparison between our UWB-based and other non-GNSS localization systems for UAVs positioning in indoor environments.

Keywords: UWB, Localization, UAVs, GNSS-denied environments

Contents

List of Figures

List of Tables

List Of Acronyms

DoF	Degrees of Freedom
DS-TWR	Double-Sided Two-Way ranging
EKF	Extended Kalman Filter
IMU	Inertial Measurement Unit
GNSS	Global Navigation Satellite System
LSE	Least Squares Estimator
LOS	Line of Sight
NLOS	Non-Line of Sight
ROS	Robotic Operating System
RTLS	Real Time Location System
SLAM	Simultaneous Localization and Mapping
SoC	System on Chip
SS-TWR	Single-Sided Two-Way Ranging
TDoA	Time Difference of Arrival
ToF	Time of Flight
UAV	Unmanned Aerial Vehicle
UWB	Ultra-Wideband
VIO	Visual Intertial Odometry

1 Introduction

Autonomous robots, and particularly autonomous unmanned aerial vehicles (UAVs), have gained popularity over the past years. Localization is an essential aspect of the control of this type of robots at the same time as one of the most challenging. Usually, UAVs' navigation is based on GNSS sensors [1]. However, UAVs have high potential in new applications where GNSS signal is not available or is too weak to rely on it, for example in industrial or indoor environments. For this reason, many localization techniques have emerged as a replacement to GNSS-based systems in localization systems for drones. Some of those methods rely only on information acquired by onboard mobile agents, such as lidar odometry and visual odometry, and have positioned themselves as competitive alternatives to GNSS sensors due to their high accuracy and high sample rate. However, these methods present limitations as well, for instance, their low long-term autonomy. In addition to this, in the case of visual odometry methods, there is a lack of reliability in low-visibility scenarios or those with the presence of dust or smoke such as post-disaster scenarios and industrial environments.

Over the last few years, positioning systems based on ultra-wide band technology have appeared as a replacement to GNSS-sensors in robotics thanks to its high accuracy [2]. These systems provide position estimations with an accuracy of the order of tens of centimeters which, in application scenarios where higher accuracy is not necessary, represents a suitable and inexpensive alternative to high-accuracy motion capture systems [3]. UWB also enables longer operations and tighter control over the behavior of

mobile robots due to its long-term autonomy.

Nonetheless, the main drawback of UWB positioning systems is that they require a predefined set of beacons, often called anchors, to be placed in known positions in the operational environment [4]. In order to localize the anchors, accurate calibration is carried out before starting the actual localization of the mobile node, known as tag. Usually, that calibration consists in introducing manually the position of each anchor one by one. Once the anchors have been calibrated, their position is fixed, which significantly reduces the applicability of this kind of system in dynamic or ad-hoc deployments. The calibration of the anchors is required to use them as a reference point to estimate the tag's position through multilateration methods. For this, a minimum of three anchors are required to measure their distance from the tag and calculate the tag's position [5].

Adding an autocalibration feature would enable the system to update the anchors' coordinates during operation if they changed. Thus, the anchors could be movable and be used in dynamic deployments. However, to the best of our knowledge, the previously developed autocalibration methods require too tight conditions and the other autocalibration algorithm currently available for Decawave's transceivers, the autopositioning firmware provided by the manufacturer, has been proven in our experiments to be overly slow and inaccurate to be suitable for mobile settings.

1.1 Significance and Motivation

Motivated by the aforementioned limited applicability of UWB technology, we have developed a localization system based on UWB technology with a built-in automatic calibration algorithm that allows the anchors to be mobile and hence to be used in dynamic localization systems. Other technologies have also been studied and used in the experimental part of this thesis in order to carry out an exhaustive comparison of different localization methods for UAVs in indoor environments. However, the focus of the project

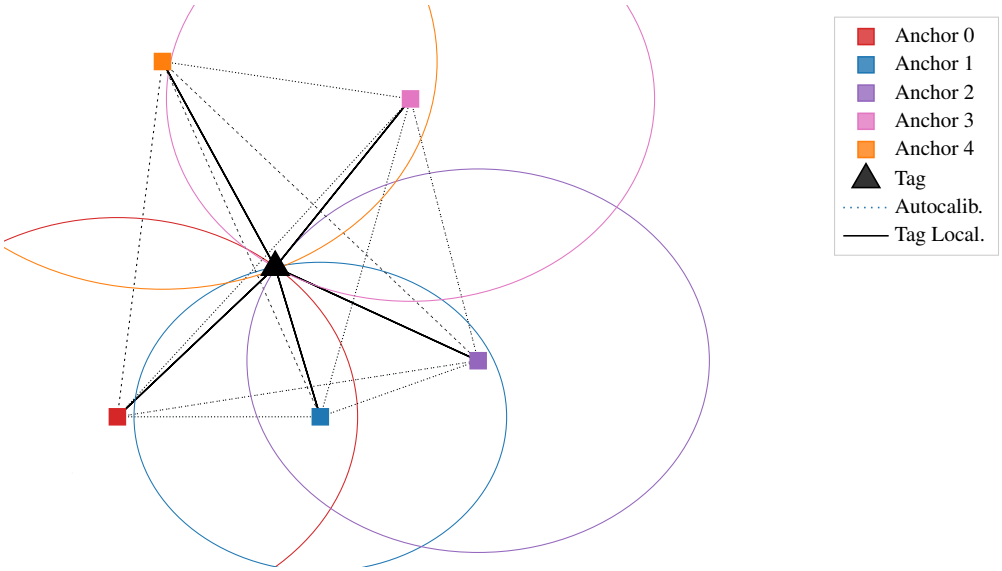


Figure 1.1: UWB localization and autocalibration concept. The circles are defined by the UWB-measured range between each of the anchors and the tag. The dotted lines represent the inter-anchor measurements taken during the autocalibration process. The number of tags and anchors is arbitrary, and only one tag is shown here for illustrative purposes. Illustration from our previous work [6].

remains on the UWB-based localization system and the built-in autocalibration algorithm developed for it.

As we described in [6], the typical procedure to estimate the position of a mobile tag based on the position of fixed anchors is depicted in Figure. 1.1, where the radius of each circle is defined by the distance to the tag estimated through UWB ranging. The tag can locate itself by estimating the individual distances to each of the anchors (solid line), while inter-anchor distances (dotted lines) can be utilized by the anchors themselves to calibrate their positions.

The novelty of the UWB-based system is that the developed automatic calibration requires less strict assumptions than the other autocalibration algorithms found in the literature and also that it takes a really short amount of time which is a key requirement to enable the system to be mobile.

In summary, our main objective is the design and development of a mobile UWB-based localization system for UAVs in indoor GNSS-denied environments. The DWM1001 UWB transceiver from Decawave has been utilized to implement it. Other systems based on different technologies such as visual odometry or lidar has been studied as well in order to have an educated overview of the possibilities in this type of scenarios. This document presents the results yielded in those experiments as well as the experimental measurements and simulation results to demonstrate the usability of the autocalibration algorithm developed as part of wider UWB experiments reported in [5]. The code is made publicly available in our GitHub repository[1], where we have released an initial version of the autocalibration firmware for Decawave's DWM1001 development board. This thesis, therefore, focuses on the results of those experiments to assess the viability and usability of the proposed system.

1.2 Related works

Autonomous navigation of UAVs is challenging, especially in situations where GNSS signal is not available or cannot be trusted such as indoor environments. Different technologies have been used in previous works to localize unmanned aerial vehicles in indoor environments from visual-inertial systems [7] to a fusion of different methods, for instance, landmark recognition and IMUs [8]. UWB ranging has also been used in aerial robots localization or navigation. In the remainder of the section, we will discuss previous works about UWB-based localization systems for aerial robots and then, we will review existing autocalibration methods for such systems and analyze in more detail the autocalibration method provided by Decawave.

[1]TIERS UWB Dataset: https://github.com/tiers/uwb_drone_dataset

1.2.1 UWB in Aerial Robots Navigation

In most existing literature, UWB-based localization systems are dependent on a set of beacons, usually named anchors, situated at fixed and well-known positions [4]. Although this fixed configuration has been used for different applications, we will focus on those related to UAVs.

UWB has been utilized to aid aerial robots' flight in different ways and environments. In [9], an UWB-based positioning system was used to assists unmanned aerial systems auto-landing, obtaining an accuracy of the order of tens of centimeters. While the proposed UWB-based system presented in [10], was used to localize several UAVs simultaneously in GNSS-denied environments. It has also been studied the applicability of UWB positioning and communication systems in multi-UAVs close formation [11].

Another common approach is to use UWB as an aiding system to other localization techniques, for instance, odometry-based estimations, when the position uncertainty is too high or to fuse its data with the data from other sensors to bound its drift. We found also many works where UWB is combined with IMU to obtain robust localization systems. IMU improves tracking results, especially for dynamic quantities, and makes orientation observable, in addition to enabling a higher sample rate to update the estimations.[12]

1.2.2 UWB Autocalibration Methods

The literature of autocalibrated systems is scarce. There are just a few papers related to this topic.

One of the first autocalibration approaches for UWB devices in mobile robots was presented by K. C. Cheok *et al.* [13]. The solution proposed in this paper estimates the positions of four anchors through multilateration after calculating the distance between each pair of them. The assumptions in which the calculation of the positions is based are the following: The anchors must follow a predefined order such as which Anchor 0 is situated at the origin of coordinates. The positive x-axis direction is defined by the line

from Anchor 0 to Anchor 1. The first three anchors are the ones that define the plane x-y.

The automatic calibration for UWB radios for multi-robots localization systems presented by M. Hamer *et al.* relies on stricter assumptions [14] than the aforementioned work. In this algorithm, in addition to the listed conditions above, it is also assumed that Anchor 2 lies on the positive y-direction, Anchor 3 on the positive z-axis and that all anchors are at fixed positions. Moreover, the localization is based on time difference of arrival (TDoA) which implies that the system needs to be accurately synchronized.

Several other works have presented onboard localization systems based on UWB technology for either one target [15], [16], or multiple targets [17]. In these papers, the anchors are situated on a mobile platform. The relative position of the tag, which is mounted on the target robot or person, is estimated from the distances between itself and the anchors.

In addition to these methods, exists also an auto-positioning system designed specifically for Decawave's UWB modules. It was developed by Decawave and it is used through their own mobile app, the same used to configure and set the positions of the anchors. This auto-positioning process is offered as part of Decawave's real-time localization system (DRTLS). Before starting the anchors, these have to be situated meeting a list of conditions in order for the algorithm to work properly. The number of anchors has to be four and they have to be placed at the same height and forming a rectangle in counter-clockwise order. On top of the restrictive requirements, our experiments have shown that the calculation time of this algorithm is around 40 s and the error above 1 m in deployments where the inter-anchor distance was less than 20 m. These results imply that the algorithm is overly slow and inaccurate to be suitable for mobile settings. In fact, the lack of accuracy is warned in the app itself, where it is recommended to measure and introduce the anchors' positions manually since the autocalibration feature makes the positioning less precise. Decawave devices are some of the most widely used UWB ranging modules [18], and thus there is an evident need for faster and more accurate autocalibration methods to enable faster ad-hoc and even mobile deployments.

1.3 Contributions

The main contribution of this thesis is the design of a self-calibration method for UWB anchors with:

- custom firmware developed for the DWM1001 device, both for real-time tag localization as well as anchor autocalibration; and
- a set of ROS nodes to interface with the devices in the different modes, as well as perform the position estimation of the tag nodes.

In addition, the work that has been carried out towards the results presented in this thesis includes:

- the design of an algorithm and development of a corresponding ROS node for localizing a UAV in a 3D lidar point cloud by fusing the lidar data with UWB-based positioning;
- an analysis on the accuracy of UWB-based ranging with the DWM1001 in line-of-sight conditions;
- the comparison of UWB and lidar-based localization with visual odometry localization with an UAV in GNSS-denied environments; and
- the measurement of the energy consumption of the UWB devices in different configurations.

1.4 Structure

The aim of this document is to detail the actual implementation of a UWB-based localization system as well as the autocalibration algorithm developed to augment its applicability. In order to ease the understanding of the method, it will be explained the theory

behind the UWB technology used for localization purposes and the approach followed in the software design along with the experimental results obtained while testing the system and the conclusions drawn from them. This document is divided into four main parts:

- Chapter 2 introduces the different methods for ranging measurements between UWB devices.
- In chapter 3, we describe in detail the proposed UWB autocalibration and localization algorithms and the communication methods used by the different sensors.
- In chapter 4, the implementation of the experimental platform used is described.
- In chapter 5, the results obtained from the different experiments are reported.
- Finally, chapter 6 concludes this work and outlines future work directions.

2 Background

The main part of the project is the developed localization system based on UWB, hence this technology will be explained more into detail in this section. Then the system as a whole will be presented distinguishing between the software and the hardware parts.

2.1 Ultra Wideband Technology

Ultra-Wideband technology is a wireless communication protocol, based on radio waves, which operates at higher frequencies than other wireless technologies such as Bluetooth and Wi-Fi. A wireless transmission scheme is classified as ultra-wideband when it occupies a bandwidth of more than 25 % of a center frequency, or more than 1.5 GHz [19]. UWB for ranging operates in the frequency bands from 3.1 to 10.6 GHz. In recent years, UWB-based localization systems have seen wider adoption in the robotics domain. Besides, owing to its precise accuracy and the capability of penetrating walls, ultra-wideband technology has gained popularity for indoor navigation.

The modus operandi followed to use Ultra-Wideband devices for localization purposes usually consists in measuring the time it takes a signal to travel from one UWB radio to another. This time is called time of flight (ToF) and, since radio waves travel at the speed of light ($c = 299\ 792\ 458\ m/s$), we can obtain distance from ToF by dividing it by c. This formula makes clear the need to measure time in a very accurate way.

From Heisenberg's uncertainty principle we can draw that, in order to achieve accurate timing, we need narrow pulses which implies that we need large bandwidth. In order

to not take all the bandwidth available the transmission power is restricted to be really low in UWB systems. This restriction leads to weak pulses, which means that a single pulse is usually below the noise level when it reaches the receiver making the reception impossible. To avoid this, instead of a single pulse, the transmitter sends a train of pulses for each bit of information. The receiver accumulates the received pulses creating a resulting pulse with enough power to exceed the noise level. Thanks to the sharp UWB pulses, UWB is able to provide measurements within a centimeter-level accuracy.

2.1.1 Ranging Measurements Methods

The two main techniques for ranging measurements in wireless ranging technologies, including UWB, consist in measuring either Time of Flight (ToF) or Time Difference of Arrival (TDoA).

ToF is the simplest and most widely used method for estimating distances between pairs of UWB nodes. These pairs are formed by a transceiver, which sends an initial message or ranging request, and a receiver, which replies to that message. The two modalities that exist within the ToF method, Single-Sided Two-Way Ranging (SS-TWR) and Double-Sided Two-Way ranging (DS-TWR) can be seen in Figure 2.1. In the former, When the response message reaches the initiator (node which sent the request), this node calculates the elapsed time, between its request and the response, and multiplies it by the speed of light to estimate the distance to the responder. In this method, the antenna delays and the message processing time must be known to calibrate the modules and obtain an accurate estimation. On the other hand, in DS-TWR, the process explained above is followed, with the difference that, it is done at both nodes. So the same distance is calculated by the two of them and this way the calibration of the devices is not needed. Minimum three anchors are needed to be able to calculate the distance through multilateration.

TDoA is also widely used but a bit more complex as it requires the anchors to be accurately synchronized. This technique consists in comparing the arrival times of a

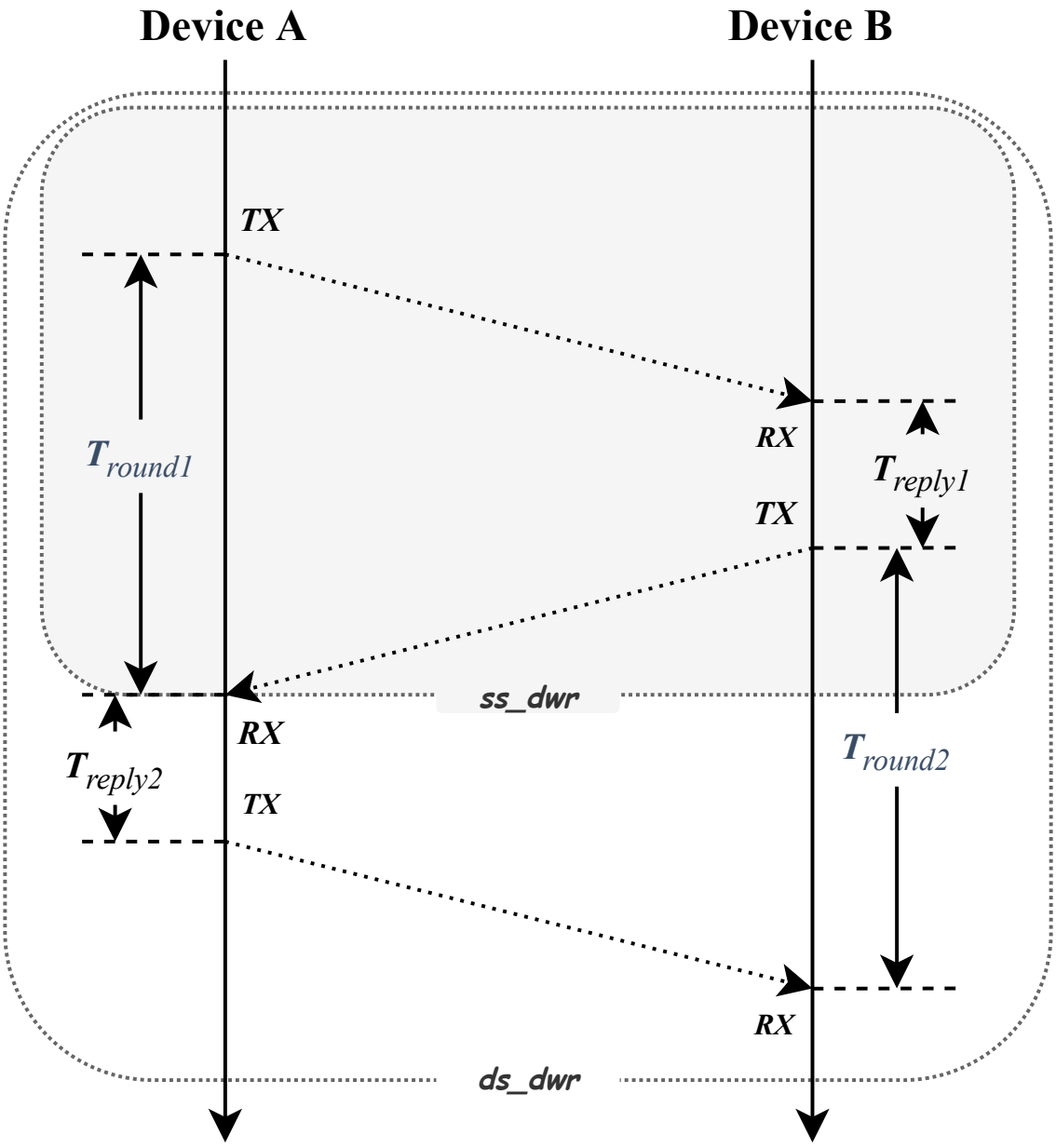


Figure 2.1: Illustration of the Single-Sided and Double-Sided Two-Way Ranging concepts (SS-TWR and DS-TWR). Depicted are the signal transmission steps and the delays involved in the process. This figure was included in our previous work, a survey about UWB-based localization for multi-UAV systems [20]

broadcast message sent by a mobile tag to the anchors. Among the anchors, there is one configured as the master anchor. When the broadcast blink has been received by all the anchors, the difference between the reception times at the master anchor and all other anchors is calculated and a hyperbola branch is drawn. The intersection point for all hyperbolas will be taken as the estimated tag's position. In this method at least four anchors are necessary.

2.2 Hardware Components

In this section, we describe the elements that form the hardware part of our experimental platform and the different technologies on which their functioning is based.

2.2.1 3D Lidar

Lidar is a technology used to calculate distances based on ranging. It uses laser beams that reflect on the surface of the surrounding environment. By measuring the time it took each pulse to reach the emitter it can estimate very precisely the distance to the object to which the ray bounced. There are different types of Lidar, while some can create an accurate 3D map of their working space, the simplest ones measure only one dimension.

3D lidars are nowadays widely used for autonomous robots navigation and mapping. This is because a LIDAR makes millions of measurements of depth information in all directions simultaneously creating an extensive collection of data points. This type of data are Point Clouds and are very convenient for creating maps in real-time. Point clouds can be easily visualized in 3D visualization tools such as RVIZ in ROS as shown in Figure 2.2.

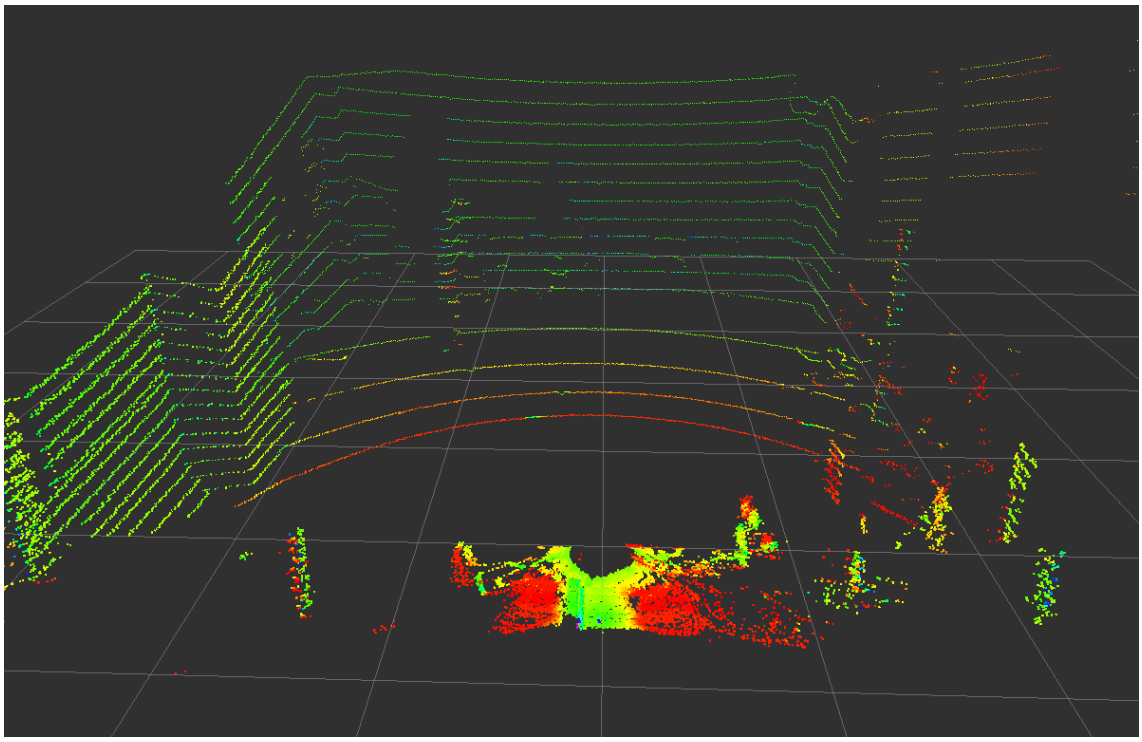


Figure 2.2: Visualization in RVIZ of the point cloud map of the experimental environment generated by the 3D Lidar.

The library used for 3D point clouds processing the Point Cloud Library (PCL). This open-source contains all the necessary algorithms for point cloud processing including visualization, filtering, model fitting, etc. It is written in C++ and split into modular libraries to facilitate the development [21]. It is also fully integrated with ROS.

There are two types of point clouds, organized and unorganized. In addition to the data array where the points are stored, point clouds contain two data fields to indicate their width and height. Width represents different magnitudes for each type of cloud. In organized point clouds, width is the number of points per row, whereas in unorganized point clouds width is the number of points in total because they are arranged as a 1D array. For this reason, height is always set to 1 in unorganized point clouds, while in the organized type height represents the number of rows.

The organized point clouds shared their characteristics with the data set generated by

stereo cameras and Time of Flight cameras where the data is organized as a 2D array. Hence, the points can be accessed through the column and row indexes. In the organized ones, the number of points is related to the frame size and therefore they remain unchanged while the frame size doesn't change. Taking into consideration this relationship, this type of point clouds is advantageous as the relationship between pixels, and consequently between neighboring points, can be used to process the point cloud more efficiently and thus, to reduce the computational costs of those algorithms.

K-d trees are used to organize a certain number of points in a space with K dimensions. For this reason, it is a common way to process the data from unorganized point clouds, especially for range and neighbor searches. A k-d tree is a binary search tree, i.e. the points are split along one specific dimension at each level. Initially, a dimension is chosen and all the points are divided into two branches, or sub-trees, depending on the value of their coordinate in that dimension. This is called a hyperplane, which is perpendicular to the corresponding axis. At the following level, the division is made looking at the coordinates in the second dimension and the process is repeated along every dimension and then starts again from the first one until the sub-trees have a single element so they cannot be partitioned anymore. Once the cloud is arranged in form of a k-d tree it is easy to find a point's neighbors within a certain radius or its nearest neighbor. The Fast Library for Approximate Nearest Neighbors (FLANN) is used for the k-nearest neighbor search operations in our system since it contains optimized algorithms for these operations even in large datasets [22].

2.2.2 DWM1001 module

All the UWB nodes use the same hardware, the Decawave's module DWM1001 shown in Figure 2.3. This module integrates a UWB transceiver DW1000 with a three-axis motion sensor and a Bluetooth microprocessor. The DW1000 transceiver includes a UWB antenna. This antenna is a Channel 5 printed PCB antenna which provides 6.8 Mbps data

Figure 2.3: Decawave's DWM1001 Development Board. Hardware used for anchors and tags of both UWB-based localization systems utilized in the experiments.

rate and up to 30 m range. The individual update rate can be set to 10 Hz. The motion sensor included in the module is a 3-axis linear accelerometer. Finally, the microprocessor is the Nordic Semiconductor nRF52832, an ultra-low power system on chip (SoC) that comprises the nRF52 Series 2.4 GHz transceiver and an ARM Cortex-M4 CPU with 512 kB flash memory and 64 kB RAM. One of the reasons to choose this technology was its low current consumption on sleep mode, which is under 15 μA.

2.2.3 Tracking Camera

In our experimental platform we included the Intel RealSense Tracking Camera T265 [23]. The new generation of Intel tracking cameras provides Simultaneous Localization and Mapping (SLAM) and Visual Intertial Odometry (VIO). Intel has named this combination V-SLAM.

SLAM is a computational procedure that consists in creating a map of an unknown environment and simultaneously, keeping track of the position of an agent within it. The calculation of both the map of the environment and the agent's position is based on the data captured with the sensors. SLAM algorithm is an iterative process. The estimation will improve with the movement of the agent as it will take more information which will be taken into account in the following iteration. The data sources might be visual, for

example cameras, or non-visual such as lidar, radar or IMUs.

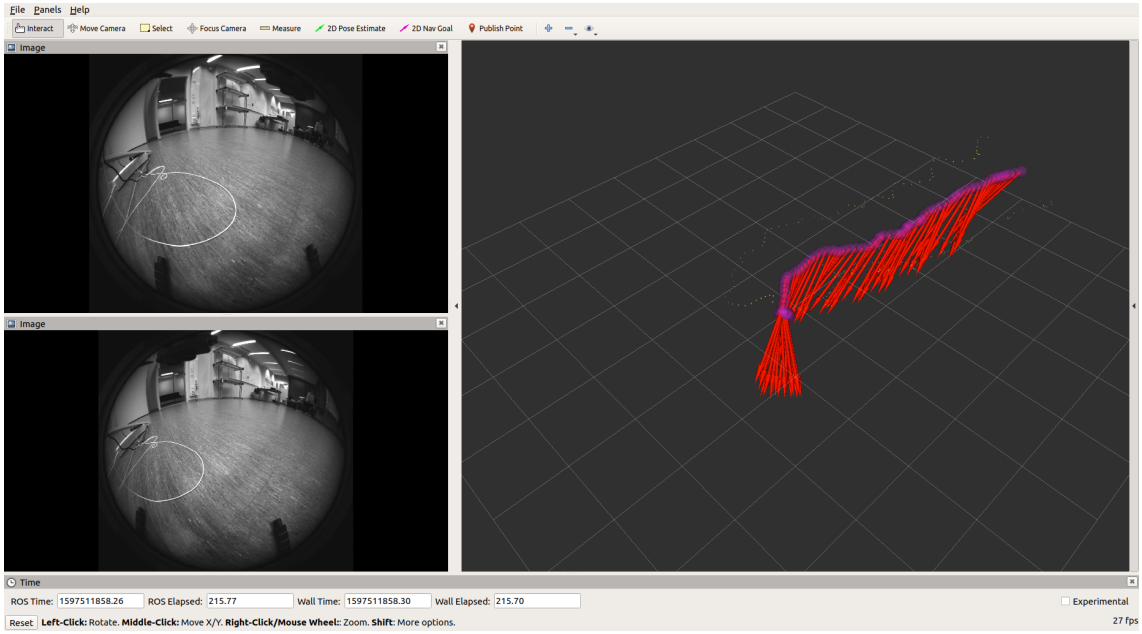


Figure 2.4: RVIZ visualization of the data from the tracking camera. On the left are shown the images captured by the fish-eye cameras 1 and 2. On the right side the odometry data is depicted with points and arrows to show the position an orientation of the device respectively.

Visual Inertial Odometry is based on the data generated by the camera and its built-in inertial sensors such as accelerometers and gyroscopes. From the fusion of this data, the system is capable of estimating the position and orientation of the device. With the combination of camera and IMU, the limitation of single cameras to capture absolute scales and the propensity of IMUs to drift are under control. Thus, VIO achieves accurate state estimation [24].

The data from the camera can be visualized in RVIZ. In 2.4 it can be seen the visualization in this tool of the images captured by the fish-eye cameras and the odometry data depicted as red arrows.

2.2.4 Inertial Measurement Unit

The IMU is a single device that comprises accelerometers and gyroscopes. Usually, it consists of a 3-axis accelerometer and a 3-axis gyroscope which provides the system with six degrees of freedom (DoF), although there are models with fewer DoF. In Figure 2.5 the DoF of an IMU with the usual configuration can be seen. The rotational speed of the device the IMU is mounted on is measured by the gyroscope while the external acceleration is given by the accelerometer. The samples of both sensors are synchronized by the hardware. IMUs are frequently found as aiding component in other localization devices such as tracking cameras.

IMU can be used to estimate the position of the device by using dead reckoning. Thus, the orientation of the device would be obtained after integrating the measurement from the gyroscope and with this, the accelerations from the accelerometer, after subtracting gravity, could be rotated. The position would be obtained after double integrating those converted accelerations. This position will be added to the previous obtained position or the initial reference given to the sensor. Integration makes error accumulate really fast over time so it is easy to see why this method will quickly drift.

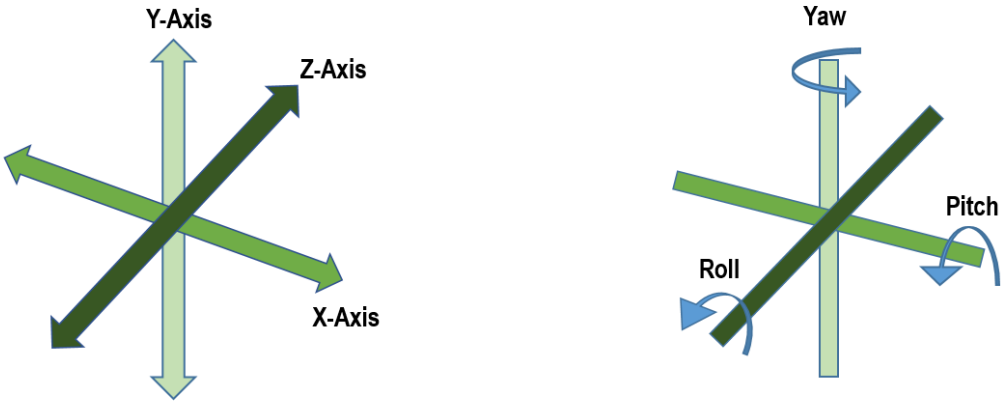


Figure 2.5: Inertial Measurement Unit Degrees of Freedom. Represents three degrees of translation and three of rotation movements.

2.3 Software Components

In this section, we overview the firmware used in the ultra-wideband nodes and the Robot Operating System used to interface all the parts in our system.

2.3.1 UWB nodes firmware

The firmware embedded in the DWM1001 module used in this project includes the firmware Positioning and Networking stack (PANS) library and the PANS Application Programming Interface (API) which are used for on-board development and communication with the module respectively. There are three user APIs included C, BLE, SPI and UART.

C API allows the user to insert application-specific code which will run on top of the PANS library and the main functionality of the module. To do this it is required to use a toolchain that consists of Segger Embedded Studio (SES) and the GNU ARM Embedded Toolchain 5.4 2016q3. The code is edited, compiled and debugged in SES and then, the code can be flashed on the target module via a J-Link tool (J-Flash Lite).

The UART interface will be used to read the messages from the tag, or the target device, and in our code also to send the start and end messages to the initiator. It can be accessed through USB.

2.3.2 Robot Operating System

The Robot Operating System (ROS) is used as the central component of our system, interfacing all individual parts comprising the setup in a flexible and modular way.

ROS, contrary to what its name might suggest, is not a real operating system as it actually goes on top of Linux Ubuntu. ROS is an open-source set of software libraries and tools used in robotics applications [25]. One of the main goals of ROS is to support code reuse in robotics development. To that end, ROS is designed as a distributed framework of processes, also called nodes, which enables executables to be individually designed and

loosely coupled at runtime. Grouping these processes into Packages and Stacks makes ROS code easily shared and distributed. ROS allows to abstract the software from the hardware, which implies that first, you don't need to change the entire software if you change the robot and secondly that the programmers don't need to know much about hardware.

For our development, the idea was to implement a ROS node per task so the configuration stayed modular. Most of our nodes were implemented in Python, however, some of them were in C++ to use some specific libraries, for instance, to process point clouds, and these could still subscribe and read from topics published by Python nodes because in ROS, nodes written in different programming languages can communicate.

3 Methodology

The objective of our work is to study indoor positioning systems for UAVs based on different technologies and on the combination of some of those, to compare their limitations, applicability, accuracy, performance, etc. The main focus of our project, however, remains on UWB technology, based on which we have developed a localization system with built-in autocalibration.

Portions of text and figures of this chapter are reproduced from our previous work [6].

3.1 Customized UWB Autocalibration Algorithm

The aim of the autocalibration algorithm we have developed is to enable the anchors in a UWB network to discover their own position in a fast and easy way, which at the same time allows them to be used in dynamic deployments. Since UWB is a ranging technology, the position of each of the anchors in the network is obtained from the inter-anchor distances measured by them. In order to localize the system in the space our system relies on a series of assumptions for the first measurement. These initial assumptions are similar to those in the related works described in section 1.2:

- The first anchor (Anchor 0) is situated at the origin of coordinates.
- The direction from Anchor 0 to Anchor 1 defines the positive x-axis.
- All other anchors lie in the half-plane with positive y-coordinate.

Table 3.1: Latency of the Autopositioning method from Decawave's DRTLS compared to our self-calibration method for anchors.

	Latency
RTLS Autopositioning	$40\,s\ \pm 5\,s$
Custom Calibration (x50)	$2.5\,s \pm 0.1\,s$
Custom Calibration (x5)	$0.9\,s \pm 0.05\,s$

The initial estimation of the anchors' positions, taking the assumptions listed above into consideration, is based on the distances to Anchor0 and Anchor 1, i.e. the anchor at the origin and the one that defines the positive x-axis direction. Then, a Least Square Estimator (LSE) is used to adjust the position of all anchors. The adjusted position is the one that minimizes the error between the inter-anchor distances and the distances measured by UWB. Once the first calibration is done, the only remaining assumption is that Anchor 0 defines the origin. These restrictions might seem relaxed when compared to previous works [14] and [13]. However, we studied the rotational error and our experiments shown it to be negligible. Therefore, we can be more flexible when situating our x and y axes.

Throughout the autocalibration process, all the anchors behave as initiator and responder. They do it taking turns as there can be only one initiator in the network. The first anchor to be designated initiator is the anchor at the origin of coordinates, henceforth referred to as Anchor 0. This first initiator is the one in charge of starting the autocalibration process. But first, a start command is sent to Anchor 0 via UART signaling that it has to initiate the process.

A counter-clockwise order has been established in order to keep an organized communication between the anchors. So once Anchor 0 receives the start command, it sends a message to the next anchor according to that order and calculates the distance between them based on the time of flight (TOF) method explained in 2.1.1. In order to remove

Table 3.2: Accuracy of the Autopositioning method from Decawave's DRTLS compared to our self-calibration method for anchors.

Covered	**RTLS Autopositioning**		**Our Autocalibration**	
Area	Min. Err.	Max. Err.	Min. Err.	Max. Err.
$1\,m^2$	9 cm	52 cm	4 cm	39 cm
$9\,m^2$	4 cm	28 cm	5 cm	24 cm
$144\,m^2$	163 cm	219 cm	135 cm	182 cm

noise and misleading measurements from the actual measurement, the mean of consecutive estimated distances is calculated and stored. The number of measurements taken to calculate the mean is defined by the programmer beforehand. Only after the initiator has received the required amount of messages from the first responder, has calculated and stored the mean and has broadcasted it to the rest of the network, it starts the communication with the following responder. In Table 3.1, we show the latency when we take 5 or 50 measurements for each pair of anchors.

Once the initiator has gathered the distance values to every other anchor, it sends a message to the following one, according to the counter-clockwise order established to signal that its turn has finished and becomes responder. The recipient of the message changes its mode to initiator and starts the cycle again. This cycle will continue until the last anchor in the network finishes its turn. Then, it will communicate it to Anchor 0, which will become initiator again and await the next start trigger. Calibrations should occur either periodically or whenever the inter-calibration positioning error at the anchors exceeds a certain error threshold. The inter-calibration positioning can be done with other onboard methods, such as visual or lidar odometry. Table 3.2 shows the difference in calibration accuracy between our firmware and Decawave's DRTLS autopositioning system, the latter being a process that is triggered through the mobile application.

The implementation of the autocalibration method described above has been done on top of the firmware for Decawave's DWM1001 Development board in C, using Segger Embedded Studio. The algorithm itself expressed in pseudocode has been included in Algorithm 1 to illustrate the previous description.

Algorithm 1: Autocalibration of the anchors.

Initiator:

- **foreach** $anchor_i \in AnchorsList$ **do**
  - **becomesInitiator**$(anchor_i)$;
  - **foreach** $anchor_j \in AnchorsList$ **do**
    - **sendMsgToResponder**$(anchor_j, anchor_i)$;
    - $message =$ **getMsgFromResponder**$(anchor_i, anchor_j)$;
    - $d_{ij} =$ **calculatesDistance**$()$;
    - **storeDistance**(d_{ij});
  - **sendEndMessage**$(AnchorsList)$;
  - **becomesResponder**$(anchor_i)$;
- **awaitForStartTrigger**$()$;

Responder:

- **while** $message.received == false$ **do**
  - $message =$ **rangingNetwork**$()$;
- **if** $message.destination == responder.id$ **then**
  - **sendTimeStamp**$(message.Source)$;
- **if** *message.type* == *end* **then**
  - **storeDistances**$(message.Distances)$;

3.2 UWB-based Localization Systems

Localization systems based on UWB technology consist of one or more mobile tags and at least three anchors since three is the minimum number of anchors required to estimate the tag's position through multilateration. However, in our experiments we always used at least four anchors as adding a fourth one makes the system more robust. In UWB positioning systems the hardware composition used for anchors and tags is the same, the Decawave's DWM1001 transceiver. The difference between the two components lies in their software configuration.

3.2.1 Customized Real-Time Localization System

In the Customized Real-Time Localization System (CRTLS), the system developed during this project, the tag is programmed to be the initiator in the network, i.e. it starts the communication tag-anchor by sending requests. To have further control over the network, the initiator only starts the process when it receives a signal from our system, in this case, a start command ('S') sent over UART. It will also finish when an end command ('F') is received through the same communication channel. The tag-anchor communication is between the tag and a single anchor so the tag sends an individual message, gets the response from that specific anchor, estimates the distance between them and stores that information in a matrix, only then it asks the next one. This implies that the requests are sent in order and the tag will not ask the next anchor until it has gotten enough valid messages from the current anchor to calculate the mean and the standard deviation of the measurement. However, this could lead to deadlocks, to avoid these situations, it is necessary to implement a timeout. This timeout will indicate when an anchor is taking too long to respond and it is necessary to move on to the next anchor, even without having received the information from the previous one. This way we ensure that the network keeps working even if there is an anchor disconnected or broken.

On the other hand, the anchors are programmed as responders, which means that they wait for the tag's request to arrive and just reply to it. Before replying they check that indeed it was the recipient of the message. The responder includes in the response message its id, the tag's id and a timestamp that consists of the sum of the programming time and the antenna delay. This antenna delay is determined during the calibration of the transceivers and it is important to know it in order to be able to subtract it for the calculations. Finally, the distance will be calculated by the tag from that timestamp sent by the responder and the one it included in the initial requests. There is no inter-anchor communication during the localization process, unlike during autocalibration.

Customizable Messages

The modules are flashed with our code, still make use of the lower layers of Decawave's firmware and our application will run on top of them. In this customized code for localization, the format of the messages is different from that of the manufacturer's default code, however, some frames have to remain unchanged for the devices to understand.

The frames used are Decawave specific ranging frames, complying with the IEEE 802.15.4 standard data frame encoding. There are two types of frames, a poll message sent by the initiator to trigger the ranging exchange and a response message sent by the responder with the information required by the initiator to calculate the time of flight estimation. There are some bytes common to both frames:

- Byte 0/1: frame control. In our case, data frames use 16-bit addressing, so these bytes are set to 0x8841.
- Byte 2: sequence number. Keeps count of the number of frames.
- Byte 3/4: PAN ID, in this case 0xDECA.
- Byte 5/6: destination address. Our customized address consists of a letter 'I' or 'R' to denote the type of device receiving the message and that device's ID.

- Byte 7/8: source address. Same as in byte 5/6. 'I' or 'R' and the ID of the device sending the message.
- Byte 9: function code. In our autocalibration algorithm, poll messages have this field set to either 'S' or 'F' to denote whether the iteration has finished or not. While in our localization algorithm this field is ignored.
- Byte 10 to 13: poll message reception timestamp.
- Byte 14 to 17: response message transmission timestamp.
- Last two bytes: reserved for a 2-byte checksum automatically set by DW1000.

While respecting this structure, the user can add application-specific fields to the messages. In our case, we use them to send the calculated mean and standard deviation, so we extend the message four bytes.

3.2.2 Decawave Real-Time Localization System

The firmware of the Decawave Real-Time Localization System (DRTLS) developed by Decawave to estimate the position of a mobile tag is not disclosed. They provide an app, which can be also used for the autopositioning of the anchors, through which the coordinates of each of the anchors can be set as well as the configuration of both the anchors and the tag.

The tag can be configured as active or passive. In active mode, the tag ranges in the network while if set to passive the tag is used as a listener. In our experiments, we used an active tag. A listener or passive tag could have been used to receive the active tag's messages on a different computer, but we connected it to the drone's onboard computer directly. The anchors have to be configured as active and one of them as initiator. For our experiments, except for those in which the objective was the comparison of the autocalibration methods, we set the anchor's coordinates manually. The network can consist of

multiple tags to be localized but only four anchors.

Decawave's Messages

The communication with the nodes using the localization firmware developed by Decawave can be done over SPI and UART. The UART shell can be configured in different modes depending on which information and in which format the user wants to receive it. In our project, we configured it to show the distances to ranging anchors and the estimated position in CSV format. The message format contains the keywords DIST and POS to denote the type of measurement following. The general format for a network with X anchors would be:

[DIST, X, AN0, AN0_ID, $AN0_x$, $AN0_y$, $AN0_z$, d_{T0},

AN1, AN1_ID, $AN1_x$, $AN1_y$, $AN1_z$, d_{T1},

AN2, AN2_ID, $AN2_x$, $AN2_y$, $AN2_z$, d_{T2},

AN3, AN3_ID, $AN3_x$, $AN3_y$, $AN3_z$, d_{T3},

{...},

ANX, ANX_ID, ANX_x, ANX_y, ANX_z, d_{TX},

POS, T_x, T_y, T_z, pqf]

The second byte of the message represents the number of anchors in the network. The measured distance to each of the anchors is preceded by two bytes to identify the anchor and three more to show its position. The first of those bytes is ANX where X is the number corresponding to the anchors' place in the network, i.e. AN0, AN1, etc. The second byte contains the unique ID of the anchor and the following three bytes correspond to the three coordinates of the position of the anchor. pqf represents the position quality factor in percent and d_{TX} the distance from the tag to the Anchor X. Finally,

3.2.3 ROS Messages

The communication in ROS takes place through publishers and subscribers. These nodes publish or subscribe to ROS topics where messages are transmitted. The type of message used to exchange information depends on the information itself. A ROS message contains two different parts: fields and constants. The data sent in the message is stored in fields while constants are used to define certain values that can be useful to interpret those fields. The structure of each message as well as the meaning of each field listed below can be found in ROS wiki page [26].

Throughout the whole process, the ROS nodes in our system exchange many different types of messages depending on what they are transmitting, i.e. pose, odometry, a 3D point, a Point Cloud, etc. There are many others but in this document only the fields contained by the messages used during our process will be explained.

Both UWB systems share their estimation of the drone's position via a type of message under the namespace of geometry_msgs named Pose, which is comprised of two other geometry_msgs: Point and Quaternion representing the position and the orientation respectively and those, in turn, contain their own fields. This structure of messages is listed below for better understanding.

```
class geometry_msgs::Pose{
    geometry_msgs::Point position;
    geometry_msgs::Quaternion orientation;
};
class geometry_msgs::Point{
//Represents the position of a point in free space
    Float64 x;
    Float64 y;
    Float64 z;
};
class geometry_msgs::Quaternion{
//Represents an orientation in free space in quaternion form.
```

```
    Float64 x;
    Float64 y;
    Float64 z;
    Float64 w;
};
```

The tracking camera, for its part, since it calculates the velocity of the device besides the position, sends its estimations via Odometry messages. This is a message type contained in nav_msgs. Its fields are header, child_frame_id and pose and twist with covariance. The pose in this message should be specified in the coordinate frame given by the field frame_id inside header while the twist should be specified in the coordinate frame given by the child_frame_id.

```
class nav_msgs::Odometry{
//Represents an estimate of a position and velocity in free space.
    std_msgs::Header header;
    string child_frame_id;
    geometry_msgs::PoseWithCovariance pose;
    geometry_msgs::TwistWithCovariance twist;
};
class std_msgs::Header{
//Used to communicate timestamped data in a particular coordinate frame.
    uint32 seq;
    time stamp;
    string frame_id;
}
class geometry_msgs::PoseWithCovariance{
//Represents a pose in free space with uncertainty.
    geometry_msgs::Pose pose;
    float64[36] covariance;
};
class geometry_msgs::TwistWithCovariance{
//Expresses velocity in free space with uncertainty.
```

```
    geometry_msgs::Twist twist;
    float64[36] covariance;
};
class geometry_msgs::Twist{
//Expresses velocity in free space broken into its linear and angular parts.
    geometry_msgs::Vector3 linear;
    geometry_msgs::Vector3 angular;
};
class geometry_msgs::Vector3{
//Represents a vector in free space.
    Float64 x;
    Float64 y;
    Float64 z;
};
```

The TFmini lidar sends Range messages, which are under the namespace sensor_msgs. This type messages contains some constants in addition to the fields necessary to transmit its measurements.

```
class sensor_msgs::Range{
//Single range reading from an active ranger that emits energy
//and reports one range reading that is valid
//along an arc at the distance measured
    uint8 ULTRASOUND=0
    uint8 INFRARED=1
    std_msgs::Header header
    uint8 radiation_type
    float32 field_of_view
    float32 min_range
    float32 max_range
    float32 range
};
```

The type of messages via which the 3D-Lidar communicates is PointCloud2 and is

also included in the namespace sensor_msgs. In PointCloud2 the layout of the point data, which is stored as a binary blob, is described by the contents of the $fields$ array. The cloud is defined by its height and width. If the cloud is unordered, height is 1 and width is the length of the point cloud.

```
class sensor_msgs::PointCloud2{
//This message holds a collection of N-dimensional points, which may
//contain additional information such as normals, intensity, etc.
    std_msgs::Header header
    uint32 height
    uint32 width
    sensor_msgs::PointField[] fields
    bool is_bigendian
    uint32 point_step    //length of a point in bytes
    uint32 row_step      //length of a row in bytes
    uint8[] data
    bool is_dense
};
```

It is important to know the type of messages in case of necessary conversions and also how the messages' fields are nested to access them. For instance, when receiving a message of the type Odometry, named odom, from the tracking camera, if we wanted to access its estimation for the x-axis, we would have to do $odom.pose.pose.position.x$. All the information presented above can be found in the documentation pages provided by ROS.

4 Implementation

In our the experiments, the mobile target of the localization systems was the drone depicted in 4.1. It was equipped with one tag flashed with Decawave's firmware, one with our own localization algorithm, a tracking camera and a 1D lidar, besides a computer. A 3D-printed mount was attached to the drone in order to screw all the components to it in a stable manner so the drone could still fly. All these parts and their role in the system will be explained in this section.

4.1 System Characteristics

In our deployment, each UWB localization system consisted of four anchors and a tag. The devices in one of the UWB network kept the default Decawave's embedded firmware, which provides real-time location system, henceforth referred to as DRTLS. The devices in the other UWB network used the customized software explained in section 3.2, so we named it CRTLS. Neither of these UWB localization systems provides accurate results for the Z-axis measurements. For this reason, it was decided to combine their results for axes X and Y with a downward-facing 1D lidar, to get the drone's height more precisely. There is only one 1D lidar on the drone on which both UWB systems rely.

The other on-board localization system is the tracking camera, which is based on visual odometry and provides both position and orientation of the drone. The last system is a 3D lidar. It is based on lidar odometry and was placed in the operating environment as a localization system. It was also used to obtain the UWB anchors' positions for their

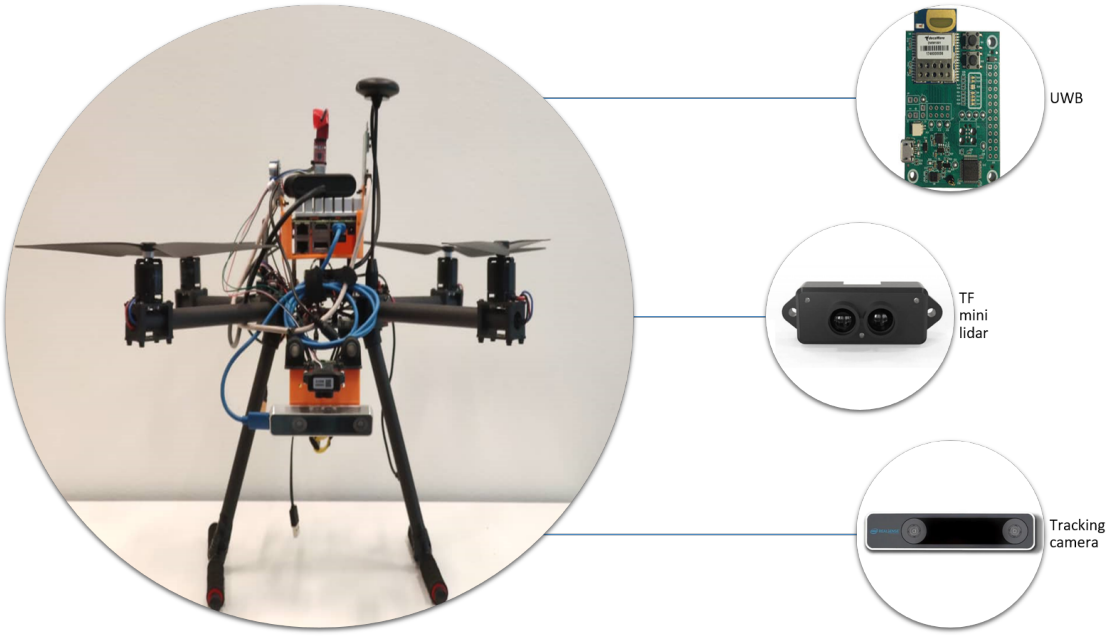


Figure 4.1: Experimental platform. Drone equipped with TF mini lidar, UWB tag and RealSense Tracking Camera

calibration before starting the experiment.

In order to keep the code modular and reusable, one ROS node was created to carry out each task. A list of the ROS nodes used and their functionality is included below.

1. Distance publisher. Reads the serial port to which the tag programmed with our software is connected to get the distances from it, decodes the messages and creates two topics per anchor where it publishes the distance to the tag and the standard deviation of the measurement.

2. Position calculator. Subscribes to the distance topics published by Node 1, estimates the position of the tag relying on each pair of anchors and calculates and publishes the mean of those estimations. It should subscribe to the anchors' positions published after autocalibration but at this moment these positions are entered manually.

3. LSE calculator. Subscribes to the topic where the mean position is published by

Algorithm 2: Extracting drone's points from the point cloud map.

Subscribe to:

- $/lslidar\_point\_cloud\_map$;
- $/UWB/tag/position$;
- $/tfmini\_ros\_node/TFmini$;

Publish to:

- $/drone/pointcloud$;

Process

- $SearchPoint = p_{sp} =$ **fuse_sensors**$(uwb\_x, uwb\_y, tfmini)$;
- **foreach** $p \in PointCloud_{map}$ **do**
  - $NearestPointSearch : p_{sn} =$ **nearestKSearch**(K, p_{sp}) ;
  - $SquaredDistance :$ **computeDistance**(p_{sp}, p_{sn});
- $SearchPoint : p_{sp} = p_{sn}$;
- **foreach** $p \in PointCloud_{map}$ **do**
  - $Radius : r = 0.3$;
  - $radiusPointSearch : p_{sr} =$ **radiusSearch**(r, p_{sp}) ;
  - $SquaredDistance :$ **computeDistance**(p_{sp}, p_{sr});
  - $dronePointCloud :$ **addPoint2PointCloud**$(droneCloud, p_{sr})$;

node 2 and calculates the point around that position that minimizes the error by the Least Squares Estimator method. In order to do this, this node needs to be subscribed to the distance topics as well. Finally, it publishes the estimated position.

4. Decawave's Position Publisher. Reads the serial port to which the tag flashed with the original positioning software from Decawave is connected. Decodes the messages and publishes the estimated position.

5. Drone Point Cloud Extractor. The node subscribes to the lidar's pointcloud topic, to

both uwb systems' position topics and to the topic where the mini 1D lidar publishes the Z-axis value. First of all, the format of the data coming from the lidar needs to be transformed from a type of ros message to a PointCloud2. The "plc conversions" ROS library was used for this. Once we have the data in the desired format, the position estimated by either of the UWB is taken as Search Point. This node creates a k-D tree and then finds the nearest neighbors of this point within a given radius. The resulting neighboring points are grouped in a new point cloud and published to be visualized in RVIZ. The algorithm's structure is presented in pseudocode in Algorithm 2.

6. TFmini Lidar publisher. Publishes the ranging measurements of the 1D lidar.

7. CSV writer. Reads the topics of UWB estimations, camera odom sample and lidar estimation and format the data so it can be stored in a CSV file for later analysis and plotting.

8. 3D plotter. Reads the CSV created by the CSV writer node and uses matplotlib to make graphs from the data contained in it.

The 3D lidar was connected to a different computer outside the drone. The computer on the drone was used to run the nodes which published the sensor data, i.e. distance publisher for UWB and the ones for the tracking camera and the 1D lidar. The external computer was in charge of running all the other nodes, getting 3D-lidar data, doing the calculations and PCL conversions and storing the data. The communication between this computer and the on-board computer was done via an SSH connection.

5 Experimental Results

In this chapter, we present the results of the experiments conducted for both the autocalibration and the localization algorithms.

5.1 Autocalibration of the Anchors

Portions of text and figures of this chapter are reproduced from our previous works [6] and [5].

We report two different types of results to study our autocalibration algorithm in terms of accuracy and usability. First, the accuracy of the DWM1001 transceiver was measured along with its maximum error. The results have been shown in Table 3.2. In the second place, the localization accuracy was studied in a simulation in which the data extracted from the previous experiment was used to characterized the devices. This simulated environment was a mobile deployment with multiple anchors and tags.

The network used in the first test where we studied the latency and accuracy of the DWM1001 development board flashed with our autocalibration algorithm consisted of four anchors. All the anchors were placed in fixed positions except from one which was moved to different positions, always in line of sight, at distances ranging from 0.5 m to 22 m. The distances measured by the UWB modules during this experiment are depicted in Figure. 5.1. The results from this experiment served to characterize the modules' error.

The UWB network used in the simulation was also formed by four anchors. Besides, three tags were situated within the shape formed by the anchors to be localized. The move-

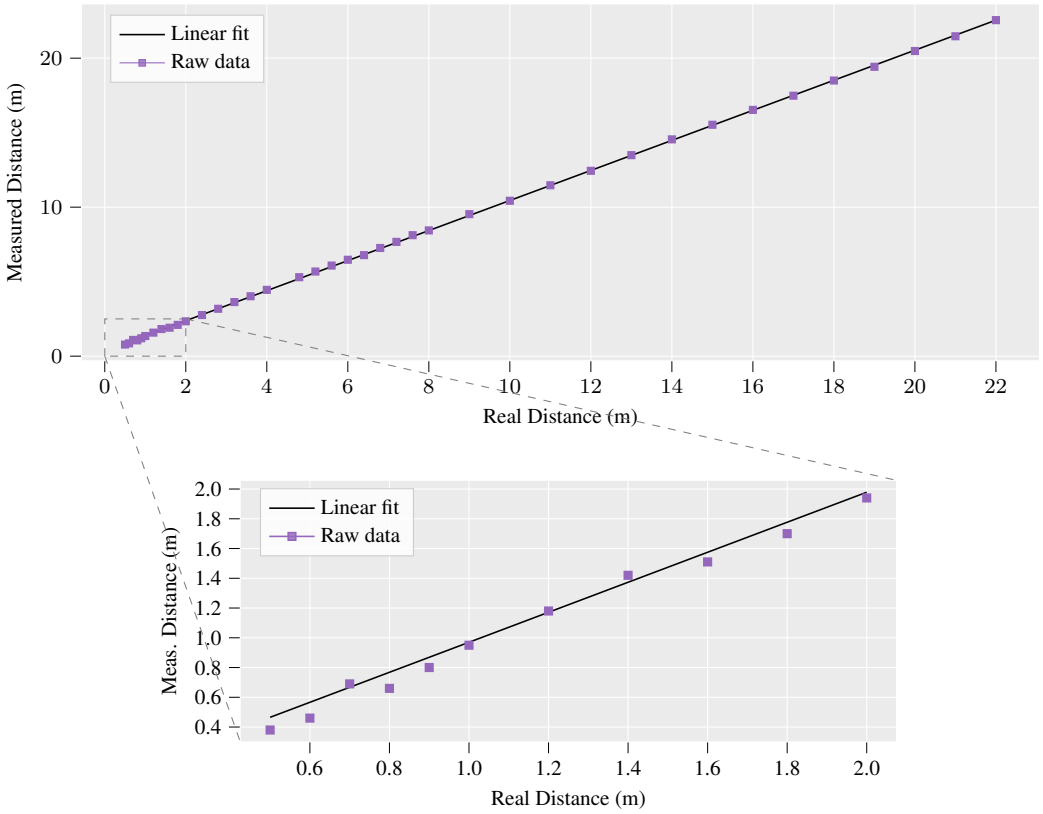
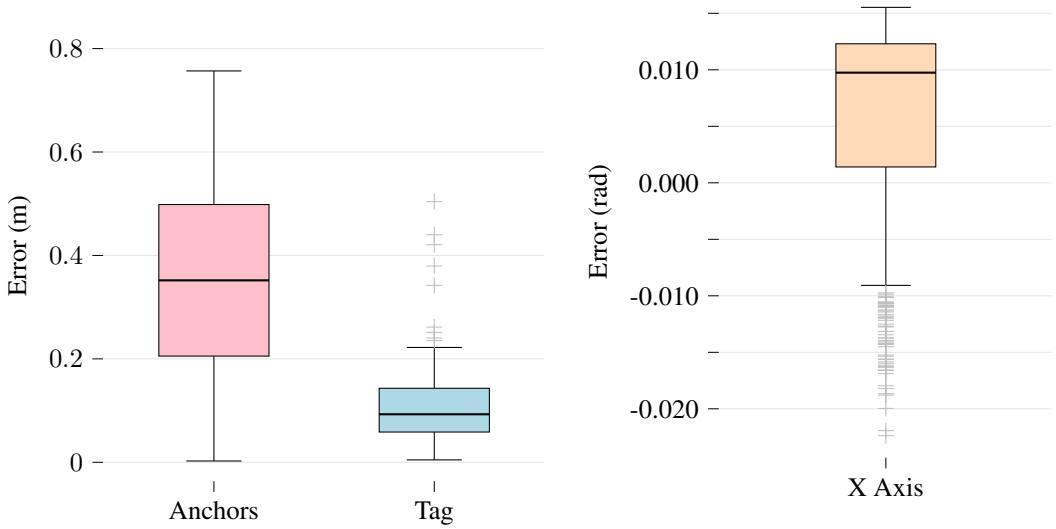


Figure 5.1: Experimental measurements and fitted line utilized for the simulations. The offset created by the responder delay has been already adjusted in the raw data measurements.

ment of the anchors and the tags was generated following a constant direction with added random Gaussian noise. In every step, a random value in the interval $(-0.1\,m,\ +0.1\,m)$ was added to each anchor's position, representing the error of the on-board position estimation utilized between calibrations. This range of values was chosen in order to have a significant error accumulated between calibrations and test the ability of the autocalibration process to bring the error down. The anchors' calibration was performed every ten steps in the simulation. Both the calibration of the anchor positions and the positioning of the tags are done utilizing a least squares estimator, except for the initial positioning step before the movement starts.

The results of our simulation are shown in Figures 5.2 and 5.4. Figure 5.2 shows the

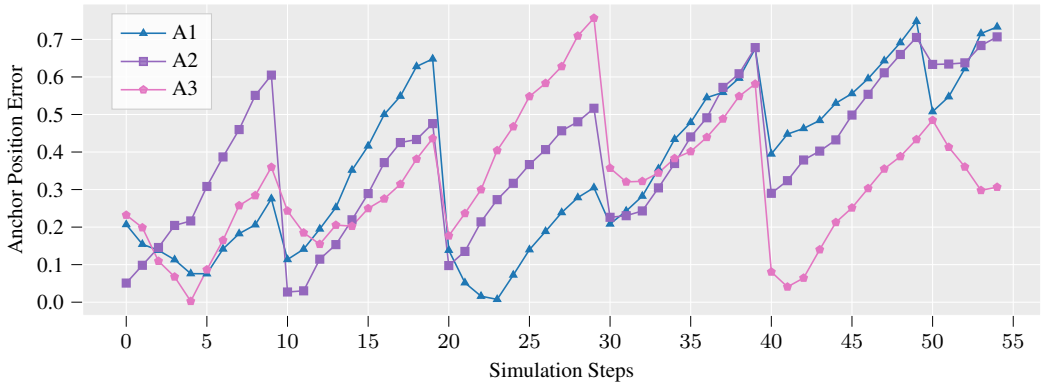


(a) Translation Error

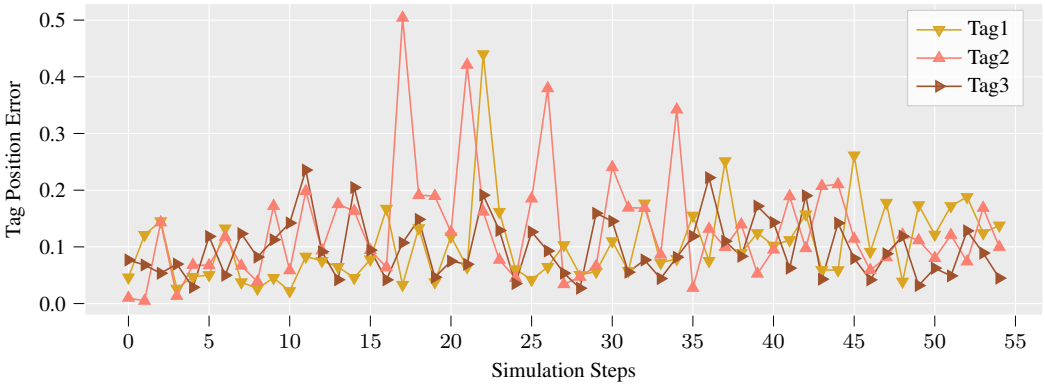
(b) Rotation Error

Figure 5.2: Translation and rotation error distribution for the anchors and tags. The rotation error refers only to the X-axis, defined as the direction between the origin anchor and the first one in the counter-clockwise direction.

distribution of translation and rotation errors. The translation error was calculated for both anchors and tags and is illustrated in subfigure 5.2a. The rotation error in subfigure 5.2b shows the error in the angle calculated between the x-axis and the line crossing the origin and Anchor 1. Note that Anchor 1 does not necessarily lie in the x-axis after the movement starts. As shown in subfigure 5.2b, in cases where the distance between these two anchors is enough, this error is small. Therefore, the assumption that Anchor 1 defines the x-axis is only needed as an initial condition and not afterward. Finally, Subfigures 5.3a and 5.3b show the error in anchors and tags positioning over a simulation of 55 steps, respectively. It can be observed how calibration, performed every 10 steps, reduces significantly the anchors' positional error. The number of steps shown in this figure is reduced for visualization purposes. We have carried out over 20 simulations with up to 1000 steps and observed the same behavior.



(a) Error in the estimated position of anchors. The UWB calibration happens every ten simulation steps.

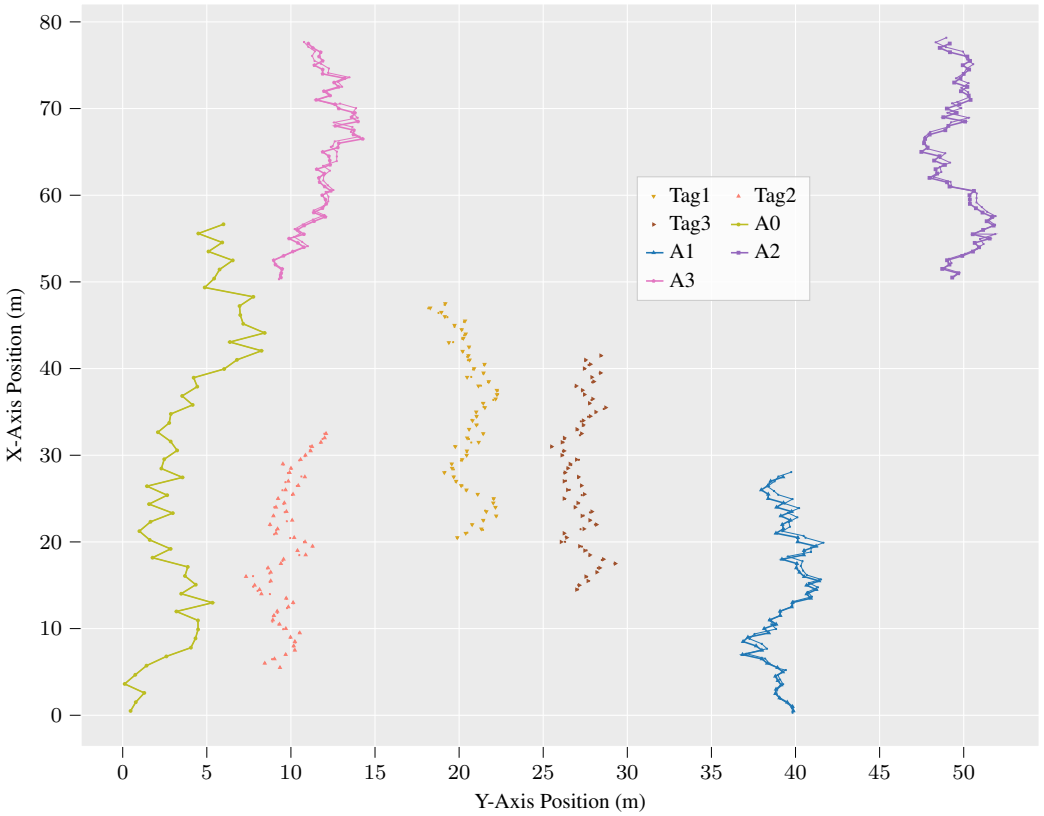


(b) Error in the estimated position of the tag during the simulation. The position of the tag is always calculated from the anchor positions based on UWB ranging.

Figure 5.3: The figures show the error over time of the four anchors and three tags that formed the simulated system. Anchor 0 does not appear because it is taken as reference and hence, has no error. The position estimation of the anchors between calibrations are defined with a random error at each simulation step.

5.2 UAV Localization Systems

To evaluate the performance of the different localization systems, we set up an experimental environment inside an office room. Due to the reduced space available, it wasn't possible to fly the drone inside the experimental room, so we moved it manually replicating a real drone behavior. The anchors of both UWB-based systems were placed on the



(a) Paths followed by the anchors and tags over the simulation. The paths have individual random components.

Figure 5.4: Simulation results for a system with four anchors and three tags. The figures show the paths over time of the anchors and tags. The movement of the nodes was generated in a constant direction with added random Gaussian noise.

walls of the room, the 3D lidar was on a table and the tracking camera, the UWB tags and the 1D lidar were attached to the drone.

Initially, the idea was to compare the four systems (3D-Lidar, tracking camera, Decawave Real-Time Localization System (DRTLS) and our Customized Real-Time Localization System (CRTLS)) at the same time. However, it was observed that interference between our CRTLS and Decawave's RTLS was significantly deteriorating the performance of the latter. Hence, it was impossible to test them simultaneously so we conducted the same experiment twice and used once UWB-based system at a time.

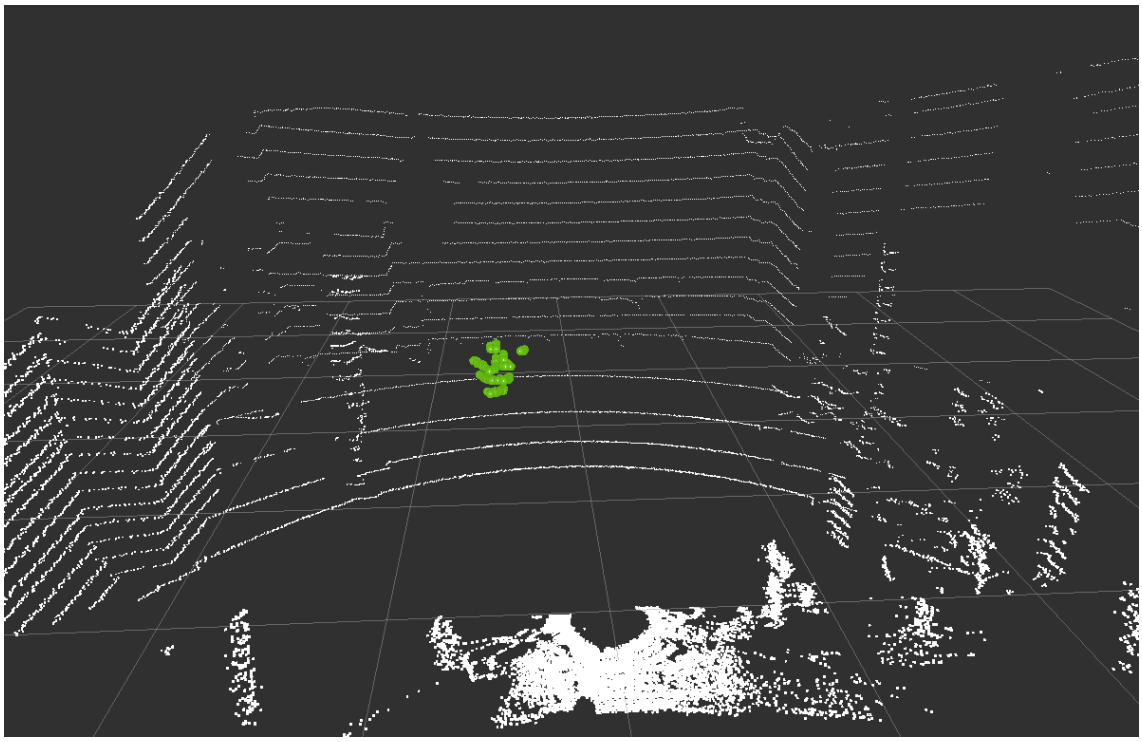


Figure 5.5: RVIZ visualization of the point cloud map generated by the 3D-lidar in white and of the point cloud resulting from the neighbor search around the drone position given by our UWB- based Customized Real-Time Localization System in green.

The experiment consisted in moving the drone around in different directions and at different heights within the scanning area of the Lidar. The on-board systems were calculating the position of the drone and all the data was recorded. Then, the estimation given by the UWB system, either our CRTLS or DRTLS, was fed to Algorithm 2 for it to detect all the points of the drone in the point cloud generated by the lidar. This new point cloud representing the drone was visualized in RVIZ as shown in Figures 5.5 and 5.6. Both UWB-based localization systems provided good estimations and the shape of the lidar can be seen in RVIZ clearly.

The estimations from all the systems are represented in a 3D graph in Figure 5.7. It is important to note that the point shown as the lidar estimation is actually the mean of all the points in the point cloud that the lidar detected as belonging to the drone. In addition

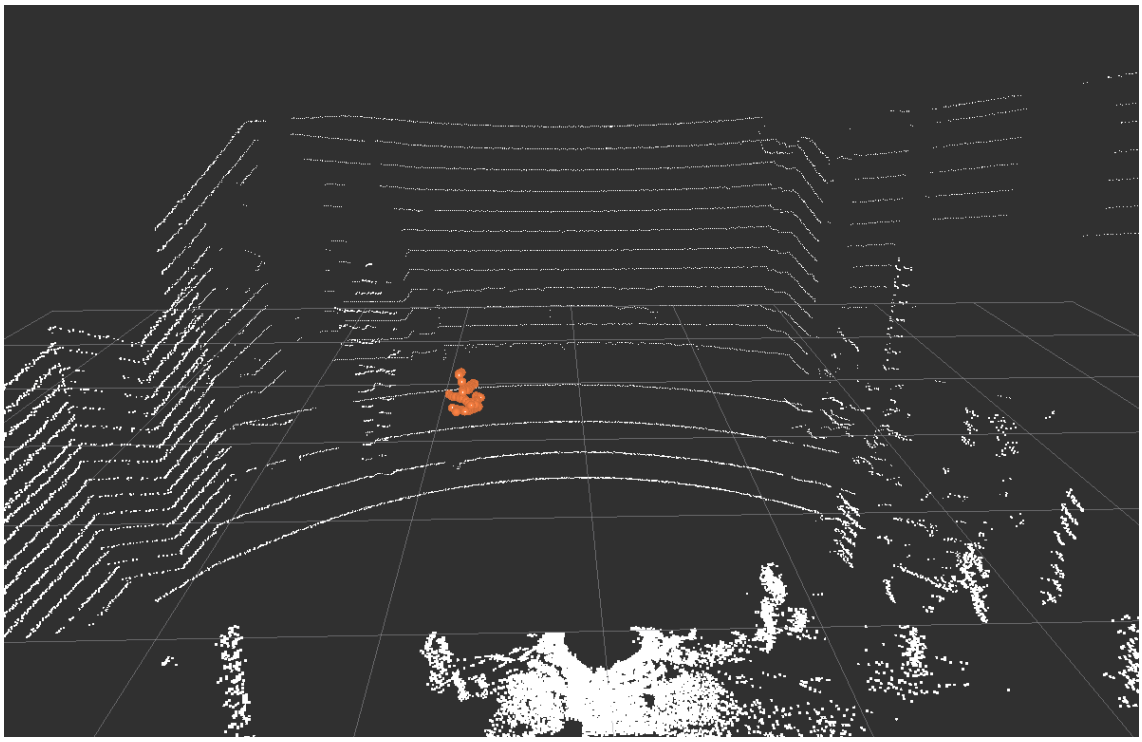


Figure 5.6: RVIZ visualization of the point cloud map generated by the 3D-lidar in white and of the point cloud resulting from the neighbor search around the drone position given by UWB Decawave's Real-Time Localization System in orange.

to this, since the cloud was generated by searching the neighboring points of the position given by the UWB-based system, this lidar estimation cannot be taken as ground-truth as the error from the UWB is carried over.

The 3D lidar and both UWB systems were placed in such a way that their coordinates systems coincide but the tracking camera bases its estimations in dead reckoning and takes as the origin of coordinates the point at which it starts the localization process. In order to translate this point to the global coordinate system to be able to compare the four estimations, the initial position calculated by the corresponding UWB system was added to the position estimation of the tracking camera.

To ease the comparison of the systems, in addition to the 3D plot shown in Figure 5.7, each axis was plotted individually. Two plots per axis are included as the UWB systems were tested separately.

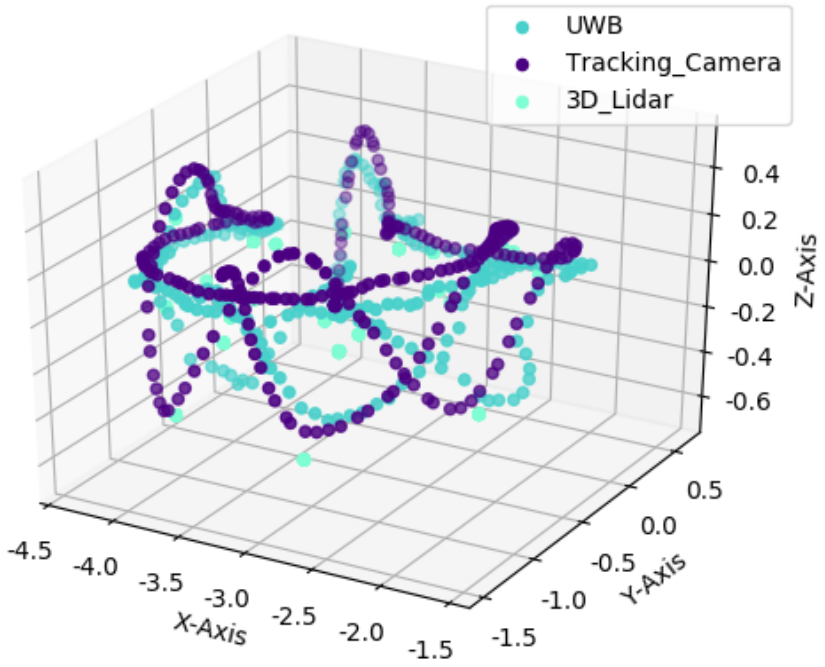
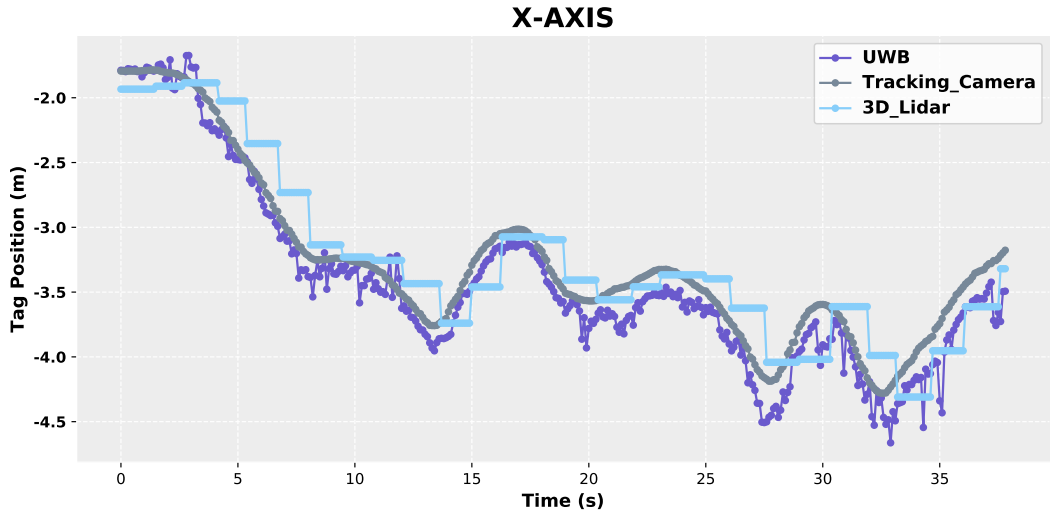


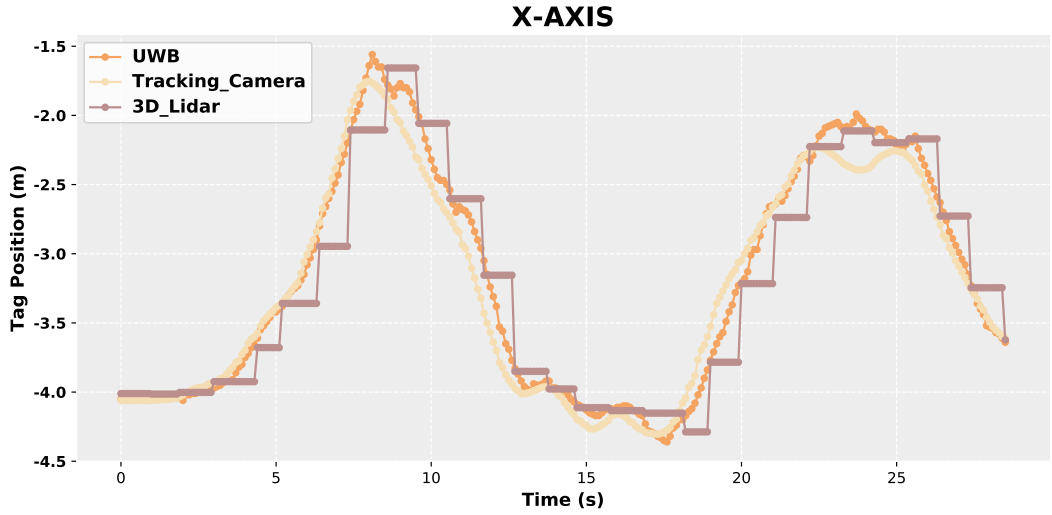
Figure 5.7: 3D representation of the drone's position estimations provided by the UWB-based system, the tracking camera and the 3D Lidar. The point taken as the 3D lidar estimation is the mean of the cluster formed by the points detected by the neighbor search algorithm within a predefined radius around the UWB estimation.

5.2.1 Qualitative Analysis

A qualitative analysis of all the localization systems, based on the results from the conducted experiments, was carried out to compare them. The criteria used in this analysis along with the results of each system are presented in a more visual way in Table 5.1.

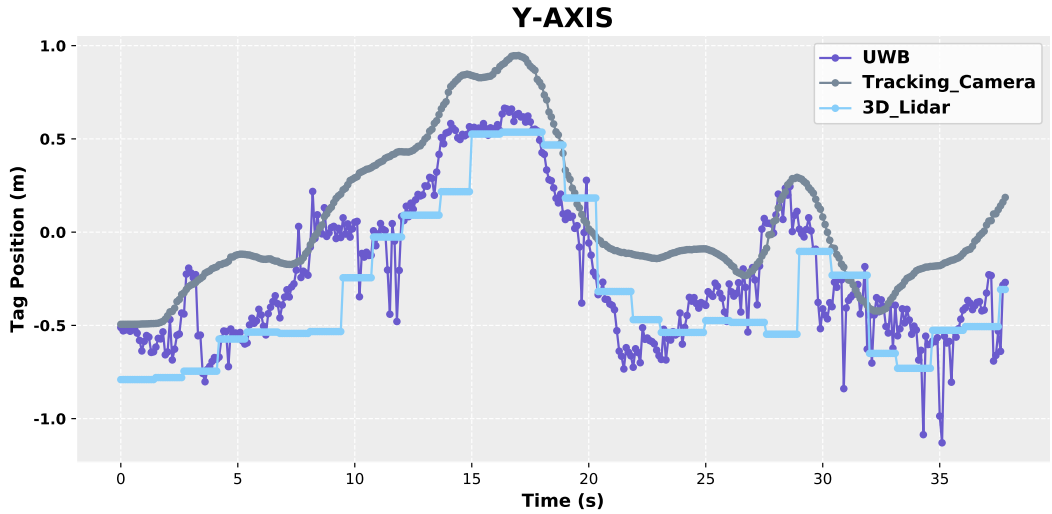


(a) Resulting X coordinate estimations given by our Customized UWB RTLS, the 3D lidar and the tracking camera.

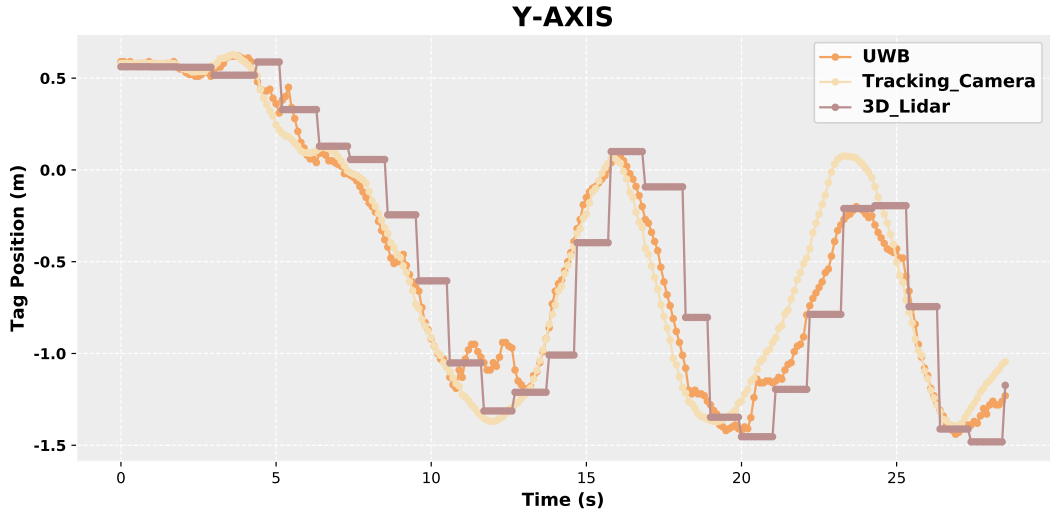


(b) Resulting X coordinate estimations given by Decawave's UWB RTLS, the 3D lidar and the tracking camera.

Figure 5.8: Estimations of the drone's position in the X-Axis provided by the four localization systems of our platform: the tracking camera, the 3D Lidar and both UWB-based Real-Time Localization Systems. The UWB-based systems were tested one at a time with the two other systems.

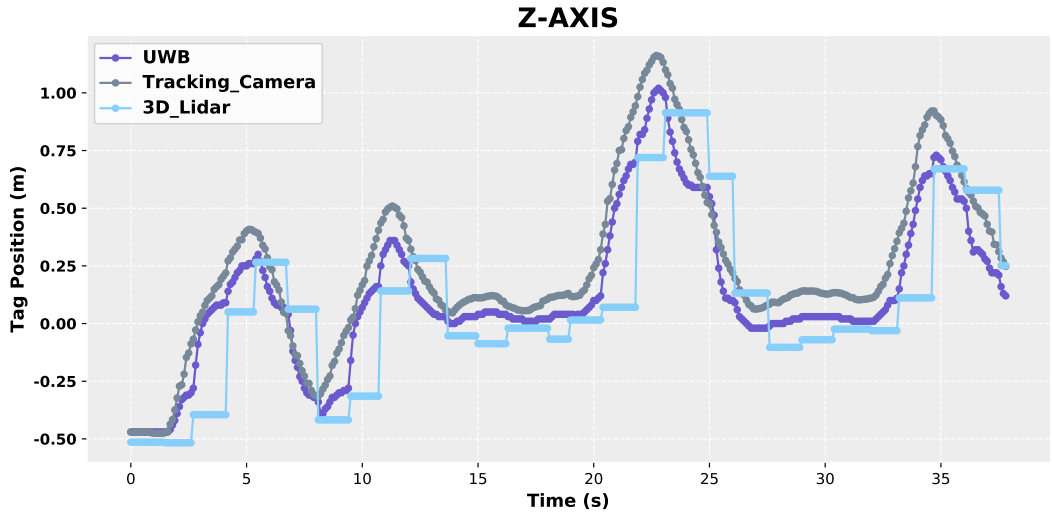


(a) Resulting Y coordinate estimations given by our Customized UWB RTLS, the 3D lidar and the tracking camera.

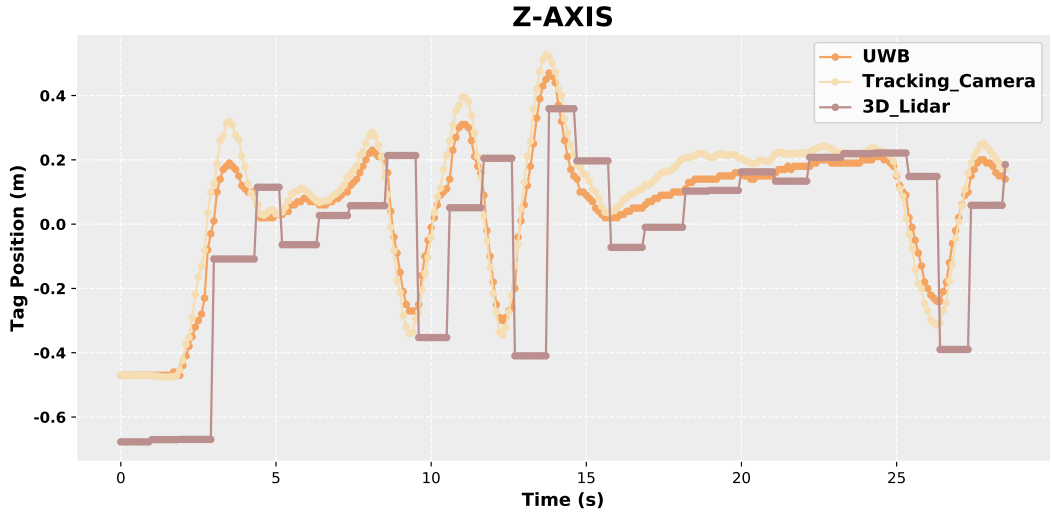


(b) Resulting Y coordinate estimations given by Decawave's UWB RTLS, the 3D lidar and the tracking camera.

Figure 5.9: Estimations of the drone's position in the Y-Axis provided by the four localization systems of our platform: the tracking camera, the 3D Lidar and both UWB-based Real-Time Localization Systems. The UWB-based systems were tested one at a time with the two other systems.



(a) Resulting Z coordinate estimations given by our Customized UWB RTLS, the 3D lidar and the tracking camera.



(b) Resulting Z coordinate estimations given by Decawave's UWB RTLS, the 3D lidar and the tracking camera.

Figure 5.10: Estimations of the drone's position in the Z-Axis provided by the four localization systems of our platform: the tracking camera, the 3D Lidar and both UWB-based Real-Time Localization Systems. The UWB-based systems were tested one at a time with the two other systems.

Tracking Camera

The tracking camera, was observed to provide precise estimations and fast response to the drone's motion. However, it needs the ideal conditions to work, that is good lighting, dust-free environments, etc. In addition to this, visual odometry systems are not reliable in the long term because they are prone to drift. Besides, they are not suitable for "extreme" behaviors, when the drone makes abrupt movements since they lost track when there is a sudden change on the velocity of the device.

3D Lidar

3D Lidars in combination with Arena Motion Capture (MOCAP) provide precise spatial data, which is perfect for localization and tracking. However, due to their elevated cost we didn't have a MOCAP available for our experiments and hence, we didn't achieve such a high level of precision. For this reason, we couldn't rely on the 3D Lidar as a reference system. Nonetheless, the 3D Lidar provided quite good results both in short and long-term operations and is suitable for mobile deployments.

Decawave's RTLS

The main drawback of Decawave's Real-Time Localization System, that it shares with many other wireless localization systems based on active beacons, has been already mentioned and is the need for a set of fixed anchors to rely on, and the fact that they have to be accurately and manually calibrated beforehand. Other disadvantages of this system are that is not very stable in the short-term and it provides a quite low sample rate. However, it is rather reliable in long-term runs, which makes it a good candidate to be fused with a system with higher short-term accuracy and sample rate.

Customized UWB RTLS

Our customized localization system with built-in autocalibration provides a great advantage over DRTLS because it is movable. In addition to this, the sample rate can be adjusted so high rates could be achieved in our system. In terms of accuracy, UWB-based systems enable localization with an accuracy of the order of tens of centimeters, which is not very precise, but on the other hand, they are reliable in long-term operations because they don't accumulate error, which is a key requirement if the system is going to be used for long periods of time. They are also an inexpensive solution, especially compared to high-accuracy motion capture systems used for navigation.

Table 5.1: Comparison of the localization systems used in our experiments: 3D Lidar, Intel T265 Tracking Camera, Decawave's UWB-based Real-Time Localization System and our Customized UWB-based Real-Time Localization System. A MOCAP system has been included in the comparison to highlight the differences but it was not used during our experiments.

	Movable	High Accuracy	Long-Term Autonomy	High Sample Rate	Price ($-$$$$$)
MOCAP	✗	✓	✓	✓	$$$$$
3D Lidar	✓	∼	∼	✗	$$$
Tracking Camera	✓	✓	✗	✓	$$
Decawave RTLS	✗	∼	✓	✗	$
Customized RTLS.	✓	∼	✓	✓	$

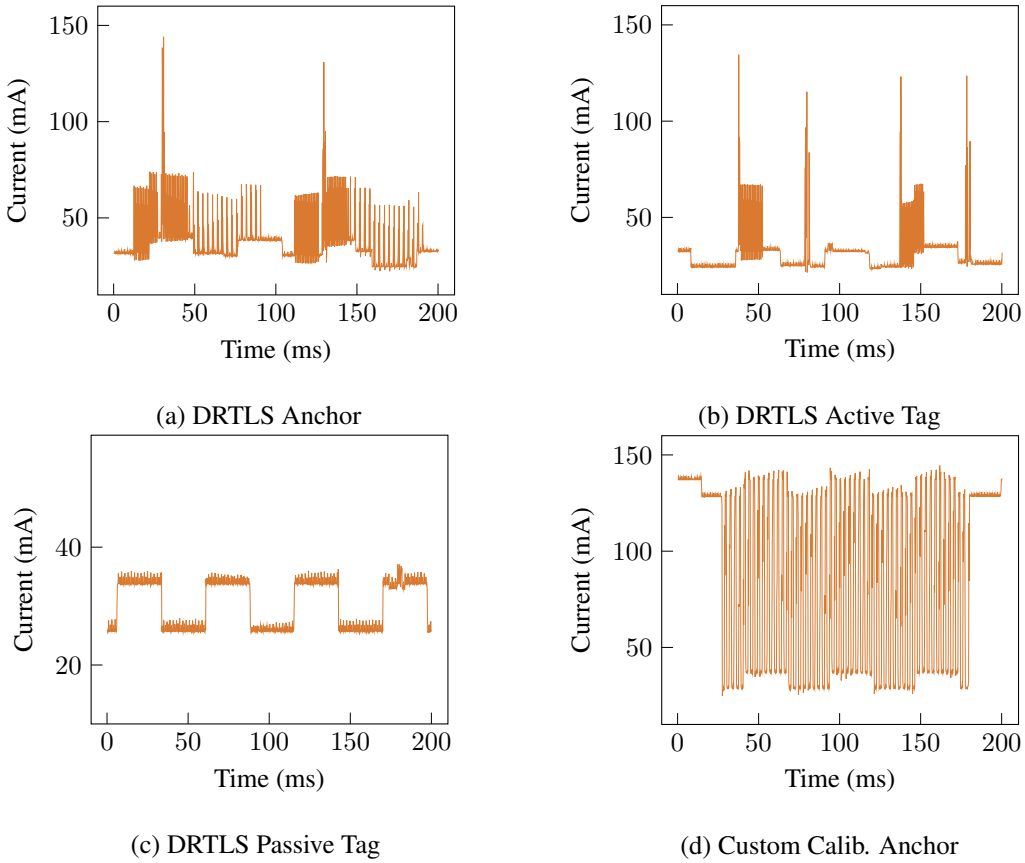
5.3 Energy efficiency

The power consumption was also analyzed for different operating modes of the UWB transceivers since this can be a key aspect in some applications with strict power constraints. The power consumption was monitored with the Monsoon's High Voltage (HV) power monitor.

In Figure 5.11, it can be clearly seen that the power consumption of the anchor flashed with our customized code is greater than that of Decawave's RTLS setup. This is caused because, in our system, communication is maintained constantly, even while it is calculating, which rises the consumption markedly. The devices programmed with Decawave's firmware present a significant increase in the power consumption when configured as anchor or active tag compared to the passive tags due to the higher amount of data transmissions.

We also wanted to study the behavior of this quantity under different supply voltages. Table 5.2 shows the measured average and maximum power consumption for every operating mode when powering the device with an input voltage of 3.7V̇ and 5V̇.

The relevance of these experiments depends on the application under development because, while for UAVs the impact on total energy expenditure might be insignificant during flight, it might be crucial in mobile settings, for instance, in systems with ground robots that move only from time to time, since having UWB nodes always active might affect long-term operation.



(a) DRTLS Anchor

(b) DRTLS Active Tag

(c) DRTLS Passive Tag

(d) Custom Calib. Anchor

Figure 5.11: Power consumption of the UWB anchors and tags in different modes. The tags were powered through USB in all these measurements.

Table 5.2: Power consumption of UWB tags and anchors in different modes and with different input voltages.

	Power @ 5V		Power @ 3.7V	
	Avg. (mW)	Max. (mW)	Avg. (mW)	Max. (mW)
Anchor	171	699	129	545
Active Tag	161	687	115	543
Passive Tag	155	189	114	503
Custom Init.	440	726	341	554
Custom Resp.	523	731	358	557

6 Conclusion

Motivated by the need for reliable navigation systems for UAVs in indoor applications, we have presented a localization system based on UWB technology with a built-in autocalibration that allows it to be mobile and thus, to be used in dynamic deployments. The novelty of this proposed system is the periodic built-in self autocalibration of anchor positions, which represents a key advantage since allows for the localization error to stay within a certain tolerance even if the anchors are moving. We have also presented a comparison between our developed system and other localization systems based on different technologies.

6.1 Future works

In future work, we will experiment with localization systems resulting from the fusion of ultra-wideband systems and other sensors. The UWB-based localization system is accurate and doesn't have a high error accumulation over time. However, it doesn't provide orientation and the estimations are not stable in the short-term to rely only on this method. The fusion of the system with another sensor with those characteristics might improve the overall performance.

Actually, we started to design a system based on that idea. It was a fusion of UWB and IMU sensors' data. An IMU with six degrees of freedom provides orientation and it has a really high data acquisition rate. The drawback of using IMU is its accumulative error. The objective of this fusion would be to provide our system with the short-term accuracy

of the IMU and the long-term reliability of the UWB. To fuse these two techniques a Kalman Filter algorithm would be used. Thus, the estimation phase of the filter would be based on the IMU estimations while the correction step would take into account the UWB measurements, bounding the drift of the IMU sensor. In future work, we will continue this development and test it.

We will also study the performance of our autocalibration in real multi-robot systems and provide a more exhaustive analysis of the usability of the proposed system in complex scenarios.

References

[1] W. Stempfhuber and M. Buchholz, "A precise, low-cost rtk gnss system for uav applications", *Unmanned Aerial Vehicle in Geomatics*, 2011.

[2] N. Macoir, J. Bauwens, B. Jooris, B. Van Herbruggen, J. Rossey, J. Hoebekeand, and E. De Poorter, "Uwb localization with battery-powered wireless backbone for drone-based inventory management", *Sensors*, vol. 19, 2019.

[3] J. S. Furtado, "Comparative analysis of optitrack motion capture systems", in *Advances in Motion Sensing and Control for Robotic Applications*, Springer, 2019.

[4] F. Zafari, A. Gkelias, and K. K. Leung, "A survey of indoor localization systems and technologies", *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2568–2599, 2019.

[5] J. Peña Queralta, C. Martínez Almansa, F. Schiano, D. Floreano, and T. Westerlund, "Uwb-based system for uav localization in gnss-denied environments: Characterization and dataset", *arXiv preprint arXiv:2003.04380*, 2020.

[6] C. Martínez Almansa, W. Shule, J. Peña Queralta, and T. Westerlund, *Autocalibration of a mobile uwb localization system for ad-hoc multi-robot deployments in gnss-denied environments*, 2020. arXiv: 2004.06762 [cs.RO].

[7] F. Fanin and J. Hong, "Visual inertial navigation for a small uav using sparse and dense optical flow", in *2019 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED UAS)*, 2019, pp. 206–212.

[8] A. R. Kuroswiski, N. M. F. de Oliveira, and É. H. Shiguemori, "Autonomous long-range navigation in gnss-denied environment with low-cost uav platform", in *2018 Annual IEEE International Systems Conference (SysCon)*, 2018, pp. 1–6.

[9] E. Kim and D. Choi, "A uwb positioning network enabling unmanned aircraft systems auto land", *Aerospace Science and Technology*, vol. 58, pp. 418–426, 2016.

[10] T. M. Nguyen, A. H. Zaini, K. Guo, and L. Xie, "An ultra-wideband-based multi-uav localization system in gps-denied environments", in *2016 International Micro Air Vehicles Conference*, 2016.

[11] S. Cao, Y. Zhou, D. Yin, and J. Lai, "Uwb based integrated communication and positioning system for multi-uavs close formation", in *2018 International Conference on Mechanical, Electronic, Control and Automation Engineering (MECAE 2018)*, Atlantis Press, 2018/03, ISBN: 978-94-6252-493-4. DOI: https://doi.org/10.2991/mecae-18.2018.98. [Online]. Available: https://doi.org/10.2991/mecae-18.2018.98.

[12] J. D. Hol, F. Dijkstra, H. Luinge, and T. B. Schon, "Tightly coupled uwb/imu pose estimation", in *2009 IEEE International Conference on Ultra-Wideband*, 2009, pp. 688–692.

[13] K. C. Cheok, M. Radovnikovich, P. Vempaty, G. R. Hudas, J. L. Overholt, and P. Fleck, "Uwb tracking of mobile robots", in *21st Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, 2010, pp. 2615–2620.

[14] M. Hamer and R. D'Andrea, "Self-calibrating ultra-wideband network supporting multi-robot localization", *IEEE Access*, vol. 6, pp. 22 292–22 304, 2018.

[15] S. Güler, J. Jiang, A. A. Alghamdi, R. I. Masoud, and J. S. Shamma, "Real time onboard ultrawideband localization scheme for an autonomous two-robot system",

in *2018 IEEE Conference on Control Technology and Applications (CCTA)*, 2018, pp. 1151–1158.

[16] B. Hepp, T. Nägeli, and O. Hilliges, "Omni-directional person tracking on a flying robot using occlusion-robust ultra-wideband signals", in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 189–194.

[17] S. Güler, M. Abdelkader, and J. S. Shamma, "Infrastructure-free multi-robot localization with ultrawideband sensors", in *2019 American Control Conference (ACC)*, 2019, pp. 13–18.

[18] P. Dawei and Y. Yunhua, "Design of indoor position system based on DWM1000 modules", vol. 585, p. 012 067, Aug. 2019.

[19] J. Foerster, E. Green, S. Somayazulu, D. Leeper, I. A. Labs, I. A. Labs, I. Corp, and I. Corp, "Ultra-wideband technology for short-or medium-range wireless communications", *Intel Technology Journal*, vol. 2, p. 2001,

[20] W.Shule, C. Martinez Almansa, J. Peña Queralta, Z. Zuo, and T. Westerlund, "Uwb-based localization for multi-uav systems and collaborative heterogeneous multi-robot systems: A survey", in *The 15th International Conference on Future Networks and Communications , Elsevier*, 2020.

[21] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)", in *2011 IEEE international conference on robotics and automation*, IEEE, 2011, pp. 1–4.

[22] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration.",

[23] *Tracking camera t265 – intel realsense depth and tracking cameras*, May 2020. [Online]. Available: `https://www.intelrealsense.com/tracking-camera-t265/`.

[24] D. Scaramuzza and Z. Zhang, *Visual-inertial odometry of aerial robots*, 2019. arXiv: `1906.03289 [cs.RO]`.

[25] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: An open-source robot operating system", in *ICRA workshop on open source software*, Kobe, Japan, vol. 3, 2009, p. 5.

[26] *Ros wiki*. [Online]. Available: `http://wiki.ros.org/common_msgs?distro=melodic`.