POLITECNICO DI TORINO

Master's degree programme
MECHATRONIC ENGINEERING

Master's Thesis

Setup and configuration of an autonomous UAV swarm for indoor coverage path planning

Supervisors
Prof. Giorgio GUGLIERI
Dr. Stefano PRIMATESTA
Dr. Enrico FERRERA

Candidate
Ronald Cristian DUTU
student ID: 290185

Academic Year 2022-2023

Abstract

Nowadays, there is excitement among professionals over the incoming robotics technologies that are paving the way to make a more efficient, sustainable and comfortable working environment. In particular, in the last decades the UAVs (Unmanned Aerial Vehicles) are being exploited in several civil sectors such as agriculture, photography/videography, construction, environmental monitoring and even transport. This list can potentially grow further, thanks to the intrinsic versatility of drones and their capability of operating autonomously. However, the bottleneck of these devices relies on the methods used to localize them in space. Outdoor, satellite systems like GNSS (Global Navigation Satellite System) and GPS (Global Positioning System) allow drone localization with acceptable precision. Nevertheless, this consideration cannot be applied to indoor spaces, where It isn't possible to rely on such systems to get an accurate estimate of the drone position. Hence, the goal of this master thesis project is to configure an autonomous mission for a swarm of drones based on VICON localization system, while using state-of-the-art software such as ROS2 Humble and PX4 Autopilot v1.14. Specifically, the tests were conducted inside a cage of size 7x3x3 meters located in the Collaborative and Service Robotics Lab at LINKS Foundation. In the beginning, several experiments were performed to try and configure correctly the drones to fly. This thesis reports in detail how the setup of both software and hardware was conducted, highlighting the problems faced and the solutions found. The next step was to enable a fleet of multiple drones to perform an autonomous mission, to test the estimate system and the configuration done. The chosen task was mCPP (Multi Drone Coverage Path Planning) using existing algorithms: a STC (Spanning Tree Coverage) algorithm and DARP (Divide Areas based on Robot's initial Positions) algorithm. Moreover, the mCPP task was an offline one, meaning the environment in which the drones executed the mission was fully a priori known. A detailed analysis of the state-of-the-art of such a task and on how the above mentioned algorithms were implemented, along with the performance results is shown in this document. At the end, some considerations on further work and suggestions for future experimentation will be given.

Contents

List of Tables

List of Figures

List of Acronyms

CCU Clock Control Unit. 25

CPP Coverage Path Planning. 54, 55

DARP ivide Areas based on Robot's initial Positions. 65

DFS Depth First Search. 56, 63

ekf Estimated Kalman Filter. 41, 42, 43, 46, 48, 50

ESC Electronic Speed Controller. 14, 15, 19, 20

FC Flight Controller. 14, 17, 19, 20, 22, 23, 25, 26, 27, 28, 30, 31, 32, 36, 37, 41, 43, 45, 67

GCS Ground Control Station. 11, 14, 17, 19, 22, 24, 43, 46, 48, 59, 67

GPS Global Positioning System. 11, 12

IMU Inertial Measurement Unit. 17, 28

mCPP Multi-drone Coverage Path Planning. 11, 12, 13, 54, 62

MoCap Motion Capture. 12, 17

MST Minimum Spanning Tree. 56, 63

QGC QGroundControl. 6, 33, 50

RC Radio Control. 14, 15, 28, 32, 35, 39, 45, 46

ROI Region of Interest. 63

ROS Robotic Operative System. 12

ROS2 Robotic Operative System 2. 12, 14, 17, 18, 24, 25, 35, 38

SDK Software Development Kit. 30, 41

STC Spanning Tree Coverage. 55, 67

UAV Unmanned Aerial Vehicle. 11, 15, 17, 18, 19, 20, 37, 39, 40, 41, 45, 54, 58, 59

Part I

First Part

Chapter 1

Introduction

1.1 Thesis Goal

Over the last years, both researchers and companies focused on developing new solutions regarding the deployment of a UAV swarm to substitute humans in risky tasks such as maintenance, inspection or rescue. The majority of drone applications nowadays are thought for outdoor where the GPS sensor can guarantee a reliable localization.

Instead, the option of drones usage inside a building is not quite implemented, considering the numerous challenges that arise in an indoor environment: high localization imprecision and issues related to human safety, just to cite a few. The difficulties rise significantly if multi-drone tasks are considered into the equation.

That said, the objective of this thesis project is to correctly setup, configure and program a multi-drone environment consisting of multiple agents that are capable of performing a task autonomously, by receiving the commands from a script installed on the companion board of each drone and started by the GCS.

In particular, the task to accomplish is a mCPP of the cage area designed in such a way that each drone collaborate with each other by subdividing the coverage area among all UAVs.

1.2 Context of work

As declared in the Thesis Goal section, a set of drones that operate in an indoor location cannot use the GPS sensor information to locate itself. Instead, they have to rely on the internal sensors position estimate which in the scenario of this thesis does not guarantee enough localization precision, since the mission that the drones have to perform requires that they operate very closed to each other and so high precision position estimate is required.

Therefore, to ensure that no drone composing the swarm will collide with each other, a VICON MoCap system is mounted on top of the beams that constitute the cage. It is composed by six high precision MoCap cameras that are capable of recording and transmitting the precise position of each drone. During the mission, every drone gets that info in real time so it can update its internal localization and avoid crashing against its peers.

1.3 Novelty contribution

This master's thesis project is done in collaboration with another colleague: Dr. Davide Morazzo. We shared our struggles into correctly configure and setup the hardware components and software resources in order to make two drones fly at same time in a cage inside the lab. This common effort is what constitutes the first part of this thesis while the second part is describing the specific task that was assign to me, hence It's fully composed by personal contribution. The goal is to implement a mCPP both in simulation and in the real setting. The element of novelty that arises from this work lays under the fact that few research is done on localization and mission planning of a swarm in indoor environment, where the challenge is that there is no GPS availability. Moreover, others issues arise while dealing with such a framework like high instability due to the wall/ceiling effect and the nearby flight of other drones that could cause noises and wrong mission execution. Hence, studying and implementing such a system is critical also in the context of research and could unlock further investigations to perform several services such as inspection of critical structures and automation of task that are either trivial or unsafe for humans.

In addition to that, talking about the software required to implement such a setup, most of the projects found in the literature were done based on ROS and only a small percentage is based upon the new middle-ware ROS2. Therefore, our efforts in implementing the swarm system in a state-of-the-art software stack could be vital to enhance those new technologies and benefit the research in the area of indoor swarm development.

1.4 Thesis Organization

The overall thesis contents are organized in two main parts: a detailed configuration representation both of hardware and software in simulation and in lab and an in depth analysis of the mCPP mission, highlighting the series of steps to perform it and the issues found along the way.

The first part starts with the general introduction to whom this section also belongs, then with the explanation of all components that coexist inside an individual drone. This is done by starting with the description of the entire system and after that, explaining each component singularly in depth. Next comes the chapter describing separately the tests done both in simulation and in the real setting. The sequence of tests are presented in their chronological order, showing for each one what were the issues found and how they were solved.

The second part is composed by a first chapter that details the state of the art for mCPP following the motivation behind the choosing of Spanning Tree Approach, among all the possibilities. Then, it comes the chapter that refers to explaining the testing process of the mCPP task, distinguishing simulation from experimentation in the real setting and focusing on presenting the most important problems encountered and their solutions.

Finally, a few comments on the results gained and some advice regarding the further work that can be conducted as a follow-up of this thesis project.

Chapter 2

Hardware configuration

2.1 Overview

As stated in the Introduction section, this thesis project focuses on implementing a mission for a swarm composed of two drones. Those are custom made, so an analysis of their components and how they communicate with each other is vital.

First of all, regardless from the fact that a drone is a standard one available on the market or a custom made one, both must have on board two components in order for them to fly: the FC (or Autopilot) and the ESC. The first one is basically the "internal brain" entitled in calculating the parameters that will allow the movement of the drone, following the commands given by the GCS. The second one, directly welded to the actuators, is responsible for translating the commands coming from the Autopilot in speed values for the motors. Externally, the drone can be piloted using an RC system or using an autonomous flight mode like Offboard Mode, the one used in this project for the task to be accomplished. The former allows to manually control the drone using several flight modes like Stabilized and Position modes, while the latter can be activated using a ROS2 algorithm either by the GCS or locally by a Companion board, which is a component that enables drones to fly autonomously. This device resides on the drone itself and comes with an operative system, usually Linux based, where all the necessary software is installed to allow communications between drone and GCS. When the Companion board Agent is correctly linked to the Client on the FC and with the GCS, the algorithm launching the autonomous mission can be started from the GCS itself and sent through the Companion Board Agent directly to the FC Client, guaranteeing a correct transmission of the mission commands with low latency since GCS-Agent link is UDP based whereas the Agent-Client link is serial.

That said, our goal with this master's thesis project is to operate a mission autonomously, relying heavily on the use of Companion Boards and using RC manual flight modes only to test to initial correct setup of each drone.

The following sections will focus on each of the aforementioned components individually, giving more details on their configuration and implementation. The final section will briefly recap the configuration and introduce the topics covered in the next chapter.

2.2 Flight Controller

The Flight Controller is an essential component of a UAV. Its role is to make the bridge between high level behavioural commands (i.e Takeoff, Hover, Landing etc) coming from either the RC or Offboard algorithms, and the low level system where the ESC needs to know the correct speed and roll, pitch, yaw values to operate on the actuators.

The RC that we have at our disposal is called BetaFlight Omnibus F4 Pro v3, an image displaying the product and another one displaying the pinout can be seen below (Figure: 2.1, 2.2).

This Flight Controller has the following specifications:

- Processor: STM32F405 168 MHz ARM Cortex M4 with single-precision FPU
- RAM: 192 KB SRAM,
- Flash Memory: 1 MB
- MicroSD port
- Sensors: InvenSense MPU6000 IMU (accel, gyro)
- BMP280 barometer
- UARTS ports
- PWM outputs
- RC input PWM/PPM, SBUS
- I2C port for external compass
- USB port
- Built-in OSD

It has a good speed in terms of processor but limited memory, just 1 MB which will create problems highlighted in the following sections.

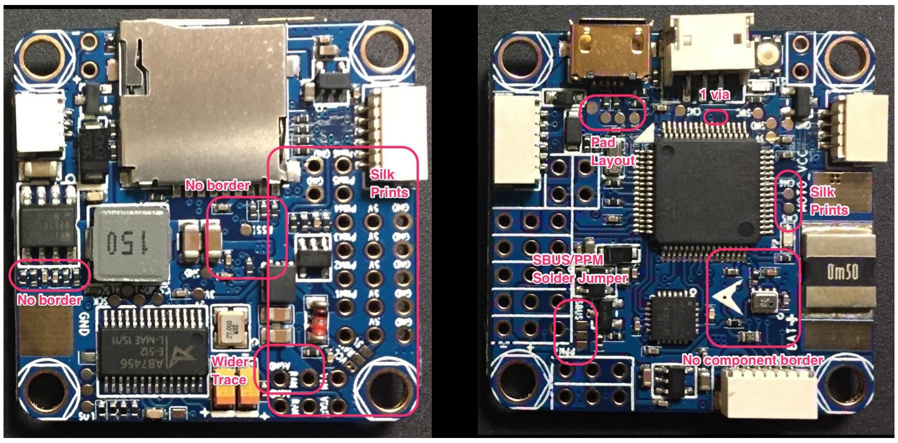


Figure 2.1. Representation of the Flight Controller model showing the back (right side) and the front (left side)

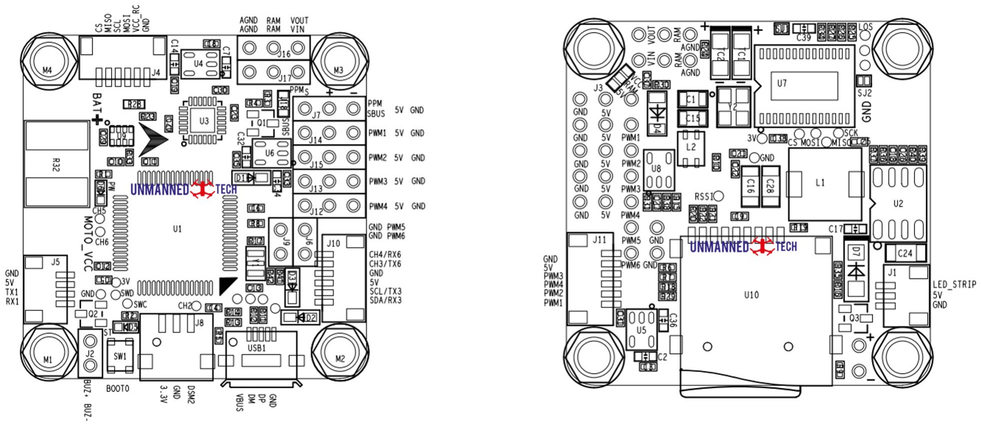


Figure 2.2. Representation of the Flight Controller pinout showing the back (right side) and the front (left side)

2.2.1 Choosing a Software Flight Stack

No matter the model type of Flight Controller used, it needs a Software Flight Stack in order to perform flights in different modes, handle emergency situation and to combine info coming from sensors. On market, there are mainly two types available both open source: Ardupilot and PX4. Considering that, in the lab where we conducted all the experimentation, PX4 was already employed for other projects, we decided to go for this one. On the following lines a brief summary of the PX4 Flight Stack is given, highlighting the important concepts that are needed to understand the further steps in developing the drone autonomous behaviours.

2.2.2 PX4 architecture definition

The PX4 Software stack is subdivided in mainly two parts: the Flight Stack and the Middleware, each one having its purpose separately described below.

As can be seen from the picture 2.3, the Flight Stack is composed of different sub-blocks, each of them having a task to complete, one after the other. The entire architecture can be summarized as organized in three main blocks: estimator, navigator and mixer. The estimator duty is to combine the information, coming from sensors both internally like IMU and externally like MoCap, in order to get a correct estimation of the drone state. In addition to that, the controller gets the estimation state and the setpoint state and updates the output state in order for it to reach the required setpoint. Finally, the mixer is responsible for translating the commands into values understandable by the actuators.

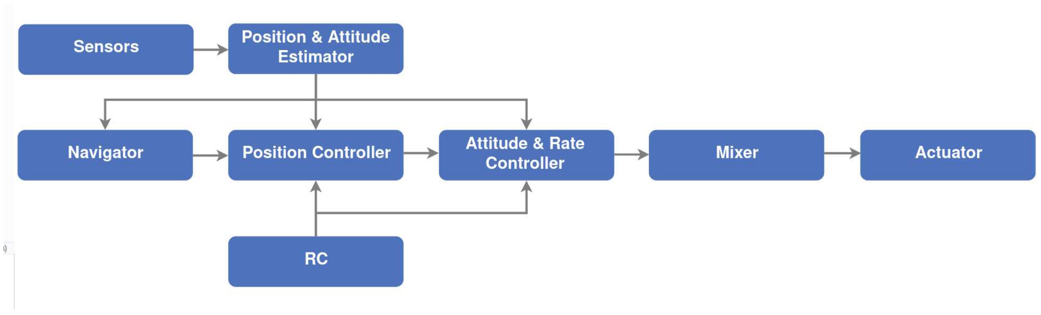


Figure 2.3. All the sub-blocks composing the Flight stack are shown in sequence

The Middleware is the other part of the PX4 software stack. Its main functionality is to enable the UAV to communicate both internally inside the Flight Controller, using uORB messages of publisher/subscriber type and externally with other devices such as GCS or Companion Boards. With these latter components the communication is handled by translating the uORB messages in ROS2 messages, that are the core of the software that runs the autonomous missions. Below is attached an image 2.4 showing the internal structure of the uXRCE-DDS middleware, the one used for this project. As highlighted also in the Introduction chapter, the middleware is composed of a Client, running directly on the FC that automatically runs on startup and an Agent that allows the drone to be connected with the Companion Board, the GCS and therefore to communicate eventually with other drones that have the same middleware installed. The important feature of uXRCE-DDS is that it translates the FC uORB commands into ROS2 commands and vice versa, enabling an easy configuration of the mission through an algorithm via the ROS2 API.

Following this overview of the flight stack architecture, the next thing to address

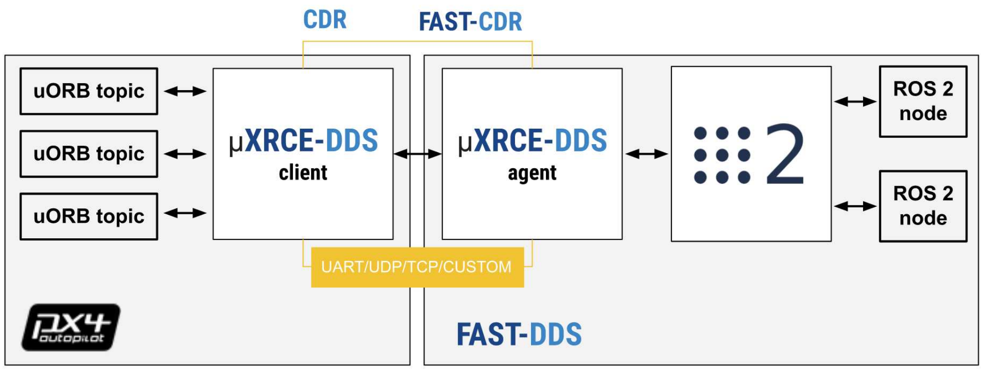


Figure 2.4. Figure showing the Client and Agent communication highlighting also the translation between uORB topics and ROS2 topics

is the reason why a particular version of px4 software was used and the implications of that also on the other software picks, related to the autonomy of the UAV.

2.2.3 Autopilot software configuration

Our goal again is to perform a task in form of autonomous mission for a set of two drones. That said, upon defining the requirements for this thesis project, a constraint arose: the mission has to be defined using the latest forms of software available open-source and in particular the ROS2 set of libraries. At that time, the newest version was ROS2 Humble that runs on Ubuntu 22.04 LTS. Defined that the next step was to identify a working version of the autopilot software that supported ROS2 Humble. From the autopilot side, the picking was tougher since Omnibus F4 v3 Flight Controller is discontinued, meaning is not commercially available anymore, hence the px4 development team doesn't give support to its software configuration. Therefore, some tests were performed using different px4 versions. At the end, the beta version of px4 autopilot software, future version 1.14, was used. Nonetheless, some manual corrections on modules and source code scripts were required to guarantee compatibility with the board. Considering also, as mentioned in the Flight Controller description, that the flash memory was only 1 MB, only the necessary modules were kept.

On the following lines the software setup used can be summarized:

- ROS2 Humble
- PX4 beta version (future v.1.14)
- Ubuntu 22.04 LTS

- Python 3.10.12
- QGroundControl 4.2

QGroundControl functionalities and usage will be explained in detail in its relative section. In short, is a software used to check if all the sensors mounted on the drone are correctly calibrated and to configure a set of parameters that allow to tailor the FC modules to the specific model of board.

That said, on the following lines the high level steps used to correctly configure the Autopilot are shown, for more details see [7]:

1. Flash the Bootloader: a defined .hex file has to be run in order to load inside Omnibus Pro v3 the boot functions so that it starts. To do this kind of operation, the Betaflight Configurator software was used, after connecting the board to the GCS using a microUSB-USB cable.

2. Flash the PX4 Firmware: with the starting program flashed, the next step is to load and install the PX4 firmware. To do that another program was used: QGroundControl. This piece of software is a must have in the "drone world" since it allows to configure the parameters of the UAV and to calibrate its internal sensors, just by using the Front End interface.

3. Setup the Firmware: additional setup was done including changes to some parameters in QGroundControl in order for the drone to be armed correctly. In particular, the motors ordering defined by the PX4 Airframe didn't match the one on the real drone, resulting in a wrong assignment of motors position. To fix that, table 3 shows the set of parameters that were changed:

Parameter Name	Value	Comment
param set PWM_MAIN_FUNC1	101	PWM1 Output to Motor 1
param set PWM_MAIN_FUNC2	103	PWM2 Output to Motor 3
param set PWM_MAIN_FUNC3	104	PWM3 Output to Motor 4
param set PWM_MAIN_FUNC4	102	PWM4 Output to Motor 2

Table 2.1. Table representing PX4 parameters change in order to switch motor ordering.

In addition to that, below it can be seen a figure 2.5 showing the difference between the PX4 airframe ordering and the actual ordering after the parameter changes. In this way, the mixer has no wrong ordering anymore and the drone can be armed and can takeoff without issues.

At this point, the FC is configured up and running, so that's all concerning this component. Next section will describe in detail the configuration of the ESC.

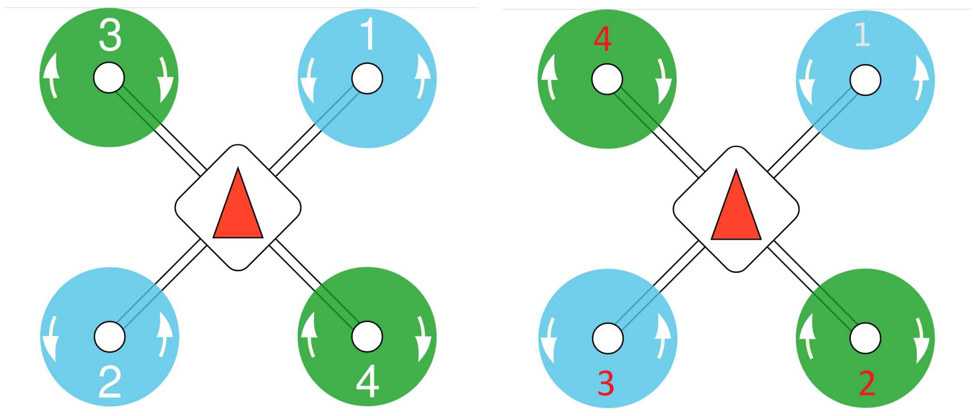


Figure 2.5. Picture showing the PX4 airframe wrong motor Ordering on the left and the ordering after each actuator is correctly assign on the right

2.3 Electronic Speed Controller

The ESC can be described as the heart of a UAV and, as already mentioned previously multiple times, is responsible of delivering the motor actions coming from the FC to every single actuator. Figures showing the model and pinout are displayed below (Figure 2.6, 2.7).

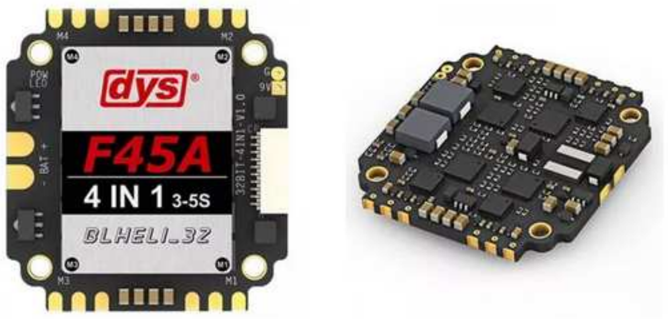


Figure 2.6. Electronic Speed Controller model showing the back (right side) and the front (left side).

For this reason, it needs to be welded to every single motor. As it can be seen from the picture describing the pinout, each actuators has three cables that need to be welded to three pins on the ESC while the battery has to be welded using red cable for the voltage and the black one for the ground. The female connector on the left side of the component, is used to connect it to the FC. In the following picture it can be seen how that is done, on the Omnibus side.

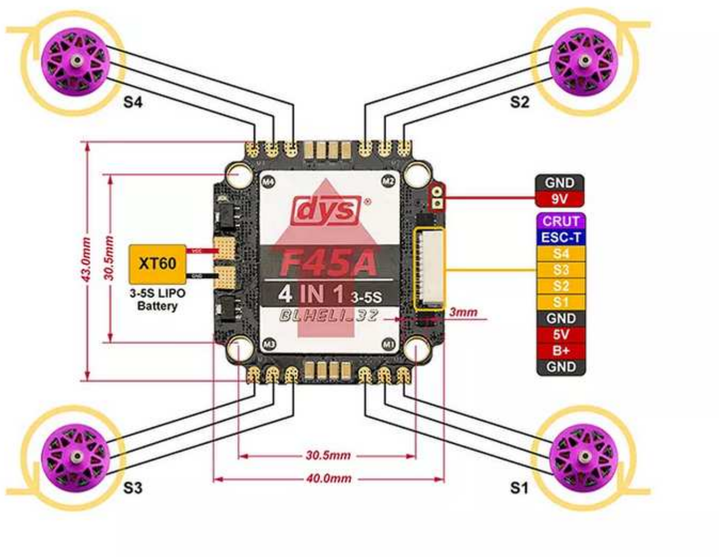


Figure 2.7. Electronic Speed Controller showing the location of every single port.

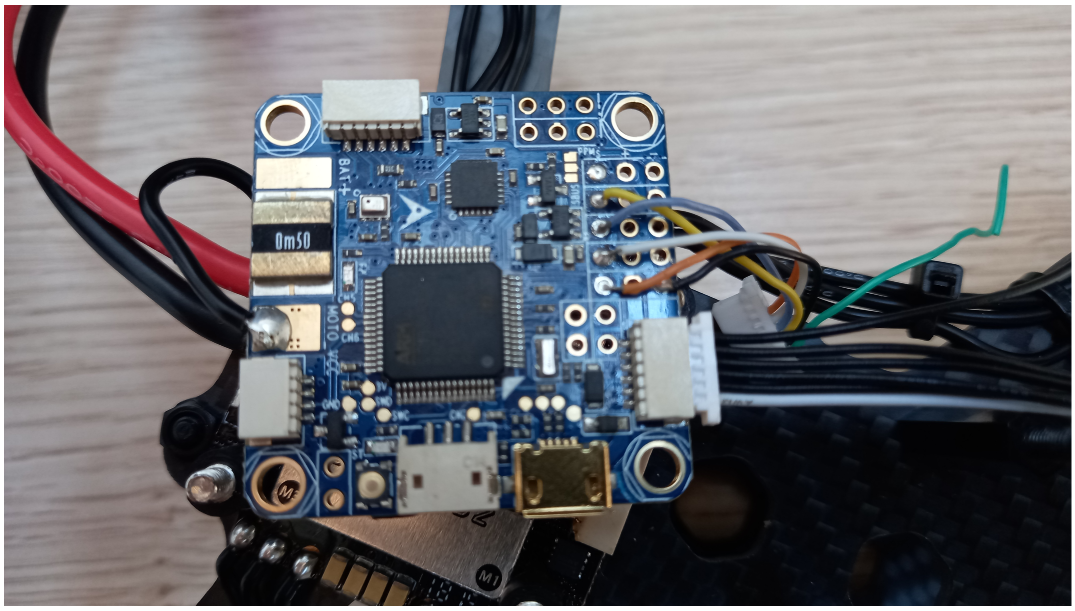


Figure 2.8. Picture displaying the wiring with the other components.

As can be seen, in the left side there is the black cable used to connect the omnibus to the battery pins, while on the top right side, four cables are attached to the PWM ports on the Omnibus directly connected to the 10 pin port to the ESC accordingly. This link is the direct responsible of transmitting the motor commands. On the bottom right side, is shown another port that was used to connect the FC to the Companion Board. Details on how that is done will be discussed in the following section.

Next another component will be described already mentioned several times, the Companion Board which duty is to enable communication between drone and the rest of the world, allowing autonomous mission to be possible.

2.4 NanoPi Companion Board

The model of Companion board at our disposal is the NanoPi Neo Air, figures displaying the chipset along with the pinout are attached below (Figure 2.9, 2.10).

On the following lines a list of the main specifications is presented:

- Processor: Allwinner H3, Quad-core Cortex-A7 Up to 1.2GHz
- RAM: 512MB DDR3 RAM
- Storage: 8GB eMMC
- WiFi: 802.11b/g/n
- MicroUSB: OTG and power input
- MicroSD Slot x 1
- GPIO1: 2.54mm spacing 24pin,It includes UART,SPI,I2C,GPIO
- GPIO2: 2.54mm spacing 12pin,It includes USBx2,IR,SPDIF,I2S
- OS/Software: u-boot, UbuntuCore, eflasher

This chipset has numerous features that are handy such as Wifi capability, MicroUSB, MicroSD and Linux as Operative system but limited RAM. Hence, the swarm mission has to be developed considering that last aspect. Moreover, the drone has to communicate directly with the external pose estimation system in order to acquire its precise pose, therefore requiring some computation also for that. Consequently, it isn't possible to decentralize the mission algorithm, meaning the drones cannot calculate their trajectories on their own. Instead, the GCS was used as the main brain for the computation of the set of spatial coordinates that constitute each drone trajectory. Next, the GCS sends the set of waypoints to be

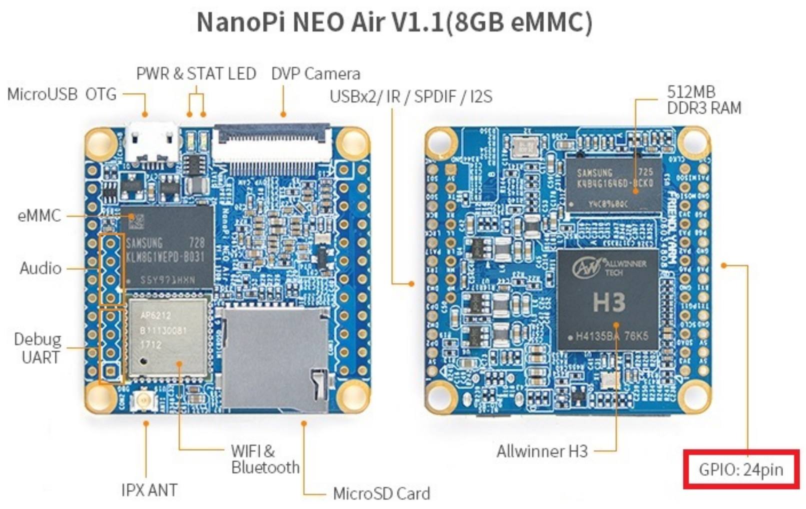


Figure 2.9. Companion Board model showing the back (right side) and the front (left side). In the figure, all the specific modules locations are highlighted.

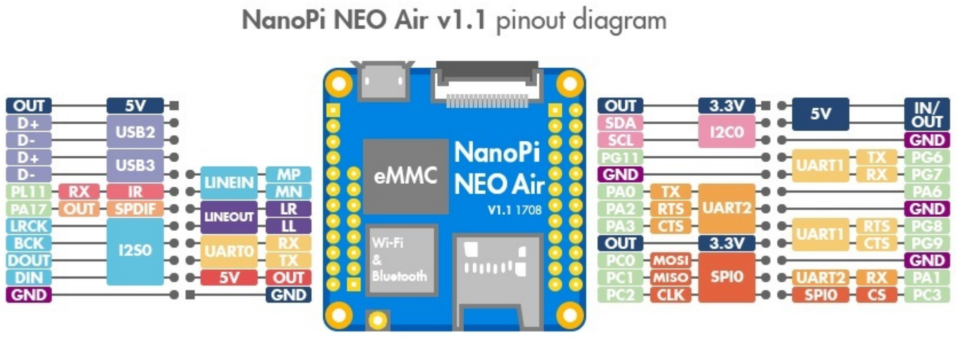


Figure 2.10. Companion Board pinout showing the location of each individual port.

achieved to the Companion Board Agent that conveys the messages directly to the FC Client.

As already mentioned, the fact that the board has just 512 MB of RAM produces several challenges such as the installation the software needed for the board to work.

The first step to accomplish was to correctly setup the WiFi module mounted on the NanoPi. An external antenna was mounted through the IPX ANT pin that can be seen in the Figure 2.9. Second step was to correctly install the Linux operative system inside the board and to correctly configure the WiFi module in order for the user to connect to the board using a terminal via ssh protocol. This allows to establish a secure and reliable connection with a remote device, in our case with the Companion Board. Each one has its own IP address assigned, so that it can be accessed individually by typing the address when the ssh linking is performed.

Next, another challenge was to install the ROS2 software and the micro-XRCE Agent on the Companion Board. The problem was that the processor type of the board is Armhf, therefore for what regards ROS2 only a source type of installation was possible. Since source type install is more demanding with respect that the binary one , all the attempts from the Companion Board to compile the packages kept failing, probably due to the scarce RAM availability. The only solution was to exploit cross-compilation, hence the software source code was compiled on another machine with higher power (in our case directly on the GCS) so that a binary file is produced and can be executed directly on the Companion Board. In this way, it was possible to successfully install ROS2 Humble on the Companion Board.

For what regards the micro-XRCE Agent, after failed attempts to install from source for same reasons of RAM limitation discussed above, the only solution found was to install it via snap package, a lighter version of the Agent, directly on the Companion Board without requiring cross-compilation.

The final step performed was to correctly wire the NanoPi to the Omnibus and establish the connection via terminal automatically, so that when the drone is powered both Client and Agent are exchanging topics. Below, a picture is displaying the wiring done in order to link the two (Figure 2.11).

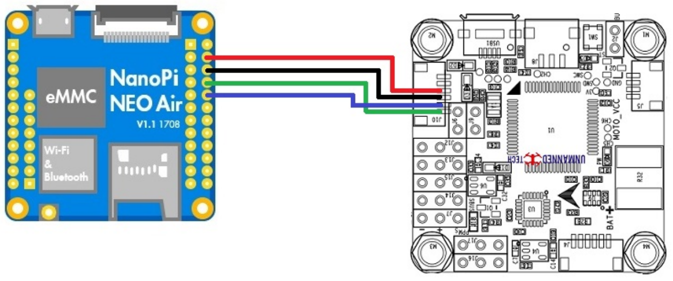


Figure 2.11. Wiring between NanoPi and Omnibus to establish an Agent-Client communication.

At this point, on both Omnibus and NanoPi startup specific lines of code has to be type in each device terminal in order to perform the connection, as shown below.

- $uxrce\_dds\_client\ start\ -t\ serial\ -d\ dev/ttyS2\ -b\ 1500000$

- $micro - xrce - dds - agent\ serial\ \ -D\ \ /dev/ttyS1\ \ -b\ 1500000$

First, the type of connection needs to be specified. In this case, Its serial since the linking is performed using a cable. Second, the port namespace assigned by each device needs to be told. Finally, the baud rate is another parameter to assign. This factor allows to define the speed at which the two devices are allowed to communicate and depends on the hardware capability. According to the PX4 documentation [2], the FC is capable of reaching 921600 baud/s, which is also the suggested minimum one for our autonomous mission goal. However, nothing is said concerning the maximum possible value. On the opposite, the data available on the Companion Board allowed to calculate the maximum baud rate value that it can achieve. By looking at the NanoPi processor data sheet, the CCU sets a frequency of 24MHz. This unit is the responsible of defining the main Clock frequency that all ports and chips use to time them self. That said, the peripheral clock of the NanoPi, is 24MHz/16 = 1.5 MHz of base clock = 1.5Mbaud/s. This value is the maximum speed allowed for all the ports, such as the UART port used for the Agent-Client communication, therefore it sets the maximum value that can be used. On internet, several possible values for the baud rate can be found but not all can be used since the rule is that the percentage change between the peripheral clock value and the baud rate value has to be below 2%. Acquired that, a test was performed in order to check if that speed value was also supported by the Client FC. Indeed, the test was positive and the communication was established correctly at that rate, allowing every ROS2 topic to be visible to each device.

All that said, the NanoPi Companion Board is now fully configured. The next analyzed component is a tentative of exploiting a different Board at our disposal, to solve the issues of RAM and processor speed that we were having with the Omnibus FC and the NanoPi Companion Board.

2.5 Modal AI VOXL Companion Board

As an alternative to the NanoPi Companion Board, the LINKS Foundation company had given to us the possibility of configuring another type of Companion Board, the Modal AI VOXL. Since this component had the advantage of having better specification w.r.t. the one of Modal AI, It was considered in the first phases as possible replacement having a Snapdragon 2.15 GHz instead and 4 GB of RAM compared to 1.2 GHz of processor power and 512 MB of RAM of the NanoPi. Moreover, the processor type was an ARM 64 bit, erasing all the problems with ROS2 configuration, at least at first glance.

In reality, even though the mounting and WiFi configuration was way easier than the NanoPi one, at some point regarding the installations of the tools needed, major issues were encountered. First, even if the producers company declares that the Modal AI VOXL uses Linux, It was discovered that was true just in part since It didn't contain a Debian version, rather a custom one: Linux Yocto Jethro. This version didn't have the apt package manager installed but opkg, therefore It was not possible to install ROS2 with binary since that utilizes apt. A workaround was found by installing Docker on the VOXL machine and running a pre-built image of ROS2 Humble on it. Unfortunately, when trying to install microXRCE Agent and configuring the communication between the Agent and the Client residing on the FC, It was not possible to find a way of linking the two. That was due to the fact that the VOXL Modal AI didn't allow to manage each port in the terminal and this fact stopped the possibility to assign the VOXL UART port to the Client FC. That meant the VOXL couldn't be used for our overall drone configuration and It was decided to keep using the NanoPi, despite its limits.

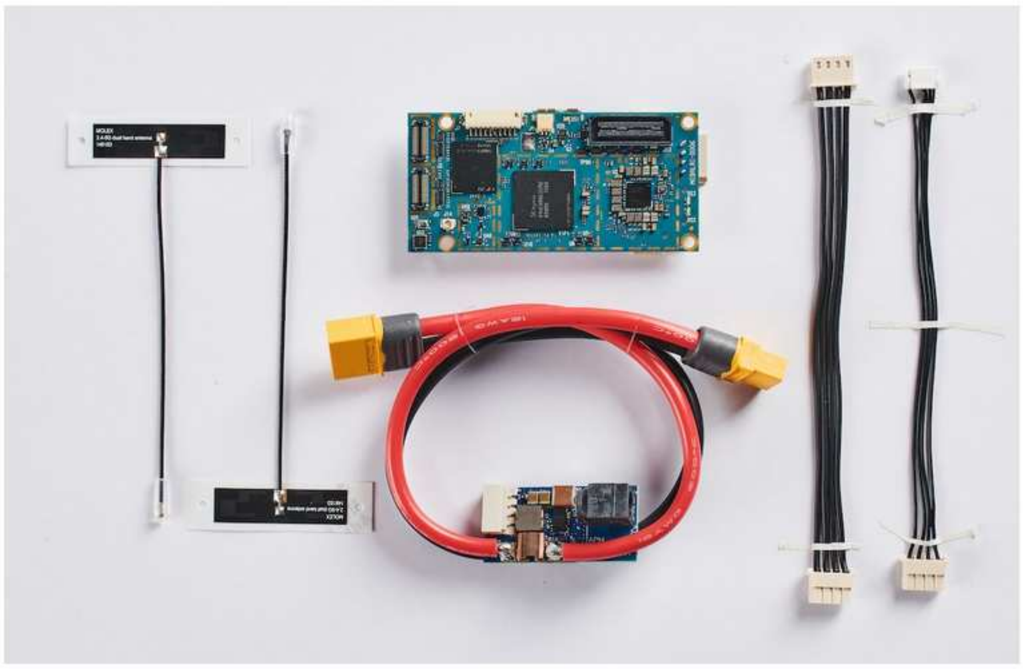


Figure 2.12. Representation of VOXL development kit comprising of all the components needed in order to configure the Modal AI Companion Board.

The next analyzed component is the Radio Controller, responsible for manual drone control in the initial phases of this drone setup and configuration.

2.6 Radio Controller

Using several type of flight modes (i.e Stabilized, Hold, Land, Position) the Radio Controller is a device that allows to manually fly the drone. During the project, it was exclusively used to test each drone fly stability individually during the first tests, while during the autonomous trials both the Radio Controller system had been shut down, otherwise the VICON system and the Radio Controller system would disturb each other. This issue will be discussed extensively afterwards, in the Testing section of the thesis. Moreover, in order for the radio controller to be connected to a drone, a FrSky 2.4GHz ACCST R-XSR radio receiver was mounted on each drone and linked with the FC through a cable. Nevertheless, Figure 2.13 and 2.14 show both transmitter and receiver models along with their respective pinouts.

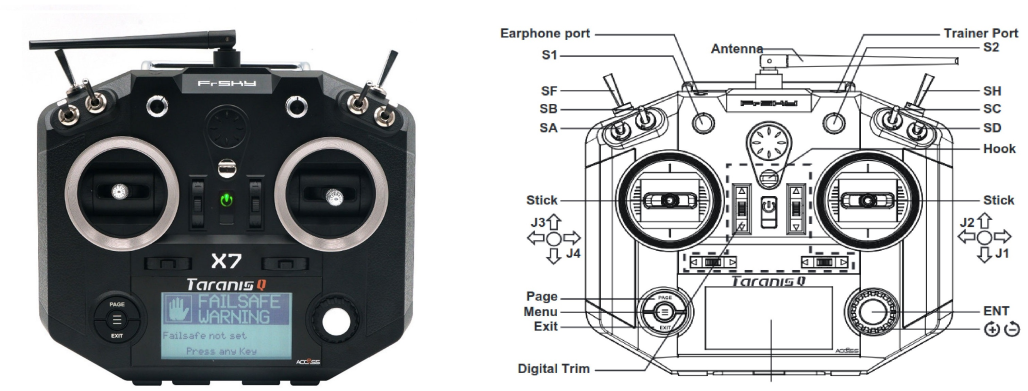


Figure 2.13. Representation of the Radio Controller. A description of the sticks to control the direction of the drone and the one to switch mode can be seen.

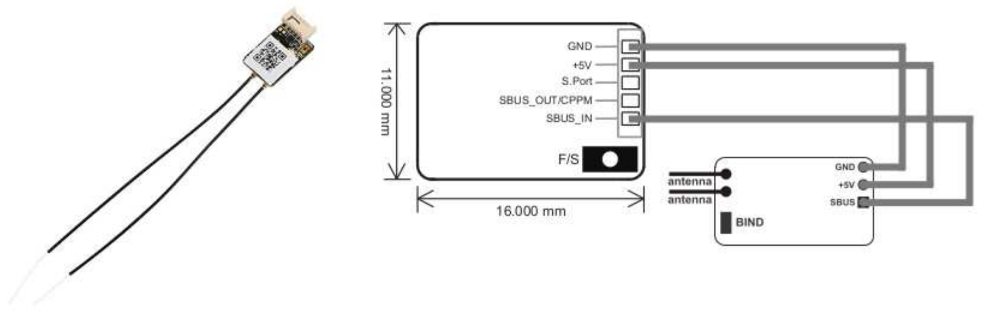


Figure 2.14. Representation of the Radio transmitter on the left and its pinout on the right.

As can be seen from the right side of the 2.13 picture, there are several levers and sticks that can be used to manually control each drone. The stick on the left side of the RC is responsible for controlling the throttle using up/down movements and the yaw using left/right movements. Instead, the right stick controls the roll using it up/down and the pitch using it left/right. For what regards the levers, SA was configured to allow arming with lever down and disarming with lever up. Next, SB was used to set the flight modes: lever up for Stabilized, lever center for Position and lever down for Land. The last lever used was the SH in which the Flight Termination Kill switch was configured. This command was used for emergency purpose as a last resort, since when turning the lever down, the drone stops all the actuators immediately.

2.7 External pose estimate system

As explained in the previous sections, the indoor cage contains a set of six motion capture cameras of type Vicon Vero v2.2, which purpose is to give to the Extended Kalman Filter module inside the FC a more precise pose estimate fused with the internal IMU information, in order to guarantee a precise localization. Pictures showing the laboratory cage and a single VICON camera are placed below (Figure 2.15 2.16).

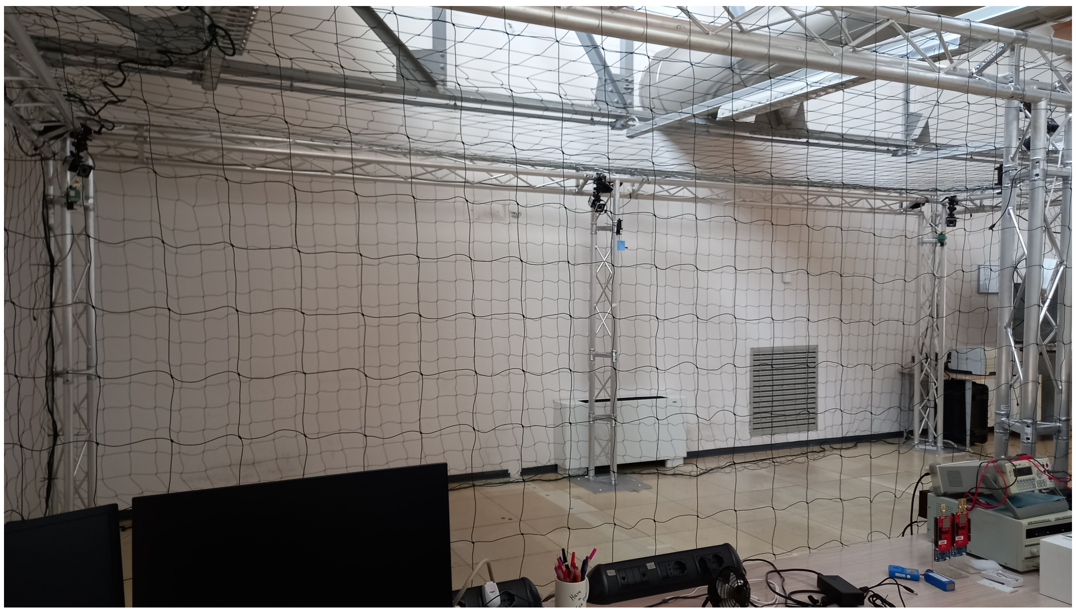


Figure 2.15. Picture showing the indoor cage used for testing.

Figure 2.16. Display of the external pose estimation motion capture camera (Vicon Vero v2.2) used in the lab.

According to specifications, the cameras have a maximum frame rate of 330 Hz and a latency of 3.6 ms, which is great for the kind of applications this master thesis is performing. In our case, the bottleneck was the capability of the NanoPi Neo Air Companion Board on acquiring the information of pose estimation in time. After several tests, it was agreed to reduce the VICON transmission frame rate to 100 Hz otherwise the Companion Board couldn't keep up with the incoming messages and several packages would have been lost in the process.

For what regards the configuration of the VICON system, each camera is linked through an Ethernet port to the main switch (Figure 2.17) that is connected also using Ethernet to the desktop computer used to handle the localization interface. Finally, the desktop computer is connected to a Wifi router (Figure 2.18) through another Ethernet cable, ensuring in this way that the connection is stable and reducing the possible lost messages to the minimum, when performing the experiments with autonomous drones.

The software used for our applications is called VICON Tracker v3.9. It acts as a Front End interface, comprising a large set of features. Among all, It lets to configure the camera system and all drones used as rigid bodies, setup the parameters such as frame rate and monitor the status of the tracking by plotting the 3D coordinates of each drone in real time. Below, an image displaying the software interface is shown (Figure 2.19).

In the following lines, there will be a description of the high level steps done in order to correctly setup the system and calibrate it so that it can be ready to give a precise localization of each drone.

1. Connected every Vero camera to an Ethernet port inside the switch system and connected the switch to a desktop computer having the Vicon Tracker installed. Moreover, inside Windows Network and Internet application, the IP address: 192.168.50.56 was added and the Subnet Mask: 255.255.255.0,

Figure 2.17. Representation of the switch used to link every VICON camera together with the main GCS

so that the cameras could be linked.

2. Calibrated the cameras using the Tracker menu options with a specific Wand, used again afterwards to set the origin of the localization system.

3. Added the drones as a rigid bodies by firstly mounting some markers on them.

Will all the steps performed, the VICON system was up and ready to transmit data as UDP packets at the IP address written above. The next and final step was to configure the receiver system, i.e the Companion Board, inside each drone in order for them to precisely retrieve the VICON system pose estimate and deliver it to the FC, ready to be fused.

The VICON Tracker environment is equipped with an SDK that allows developers to define a custom C++ script in which It's possible to connect with the Client, in this case the VICON system, and transform the pose estimate directly in ROS2 messages that can be correctly interpreted by the FC. On the following lines a brief overview of what the script does.

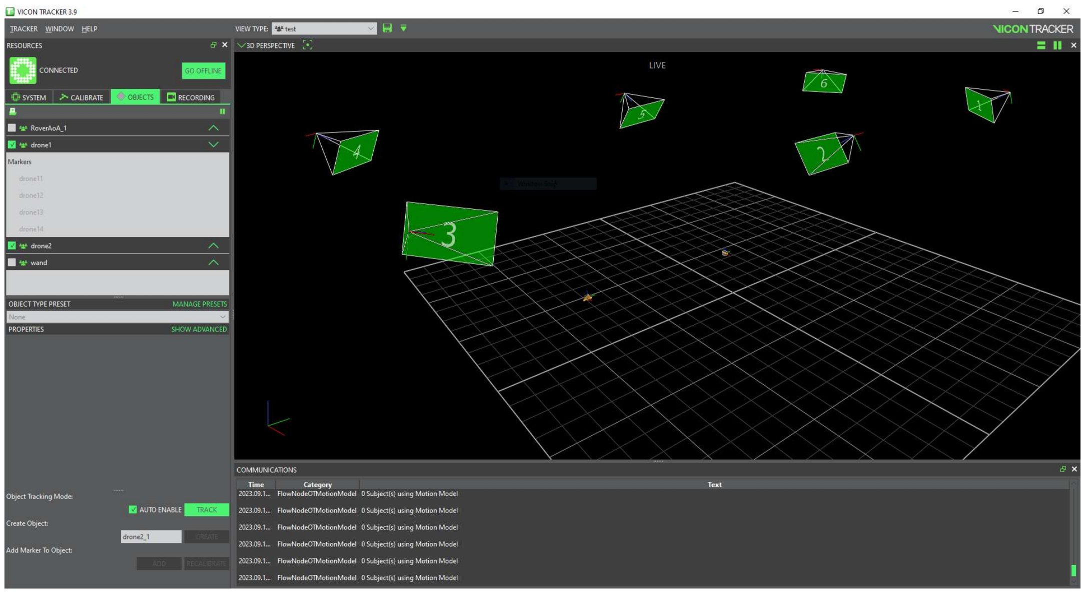
1. Connect to the Client at the IP Address: 192.168.50.56 and get each drone object name.

Figure 2.18. Model of the WiFi router used to connect the CGS via the VICON system and with the drones

2. For each object get the translation coordinates in form of x,y,z and the rotation coordinates in form of quaternion. For the former, the VICON system delivers this info in millimeters while PX4 accepts position estimate in meters, an appropriate transformation is required. For the latter, the PX4 frame convention needs to be specified for an appropriate acquisition of the quaternion rotation, therefore it necessary to specify to VICON to use the FRD (Forward Right Down) convention.

3. Transform the VICON message into ROS2 message and publish the result in the Vehicle Odometry message format in order to to be acquired directly to the Companion Board Agent.

4. After linking Agent with Client via serial cable, the Agent in real time sends the acquired messages to the FC so that, the Extended Kalman Filter module inside each drone, can fuse the information.



Time	Category	Text
2023.09.1...	FlowNodeOTMotionModel	0 Subject(s) using Motion Model
2023.09.1...	FlowNodeOTMotionModel	0 Subject(s) using Motion Model
2023.09.1...	FlowNodeOTMotionModel	0 Subject(s) using Motion Model
2023.09.1...	FlowNodeOTMotionModel	0 Subject(s) using Motion Model
2023.09.1...	FlowNodeOTMotionModel	0 Subject(s) using Motion Model
2023.09.1...	FlowNodeOTMotionModel	0 Subject(s) using Motion Model

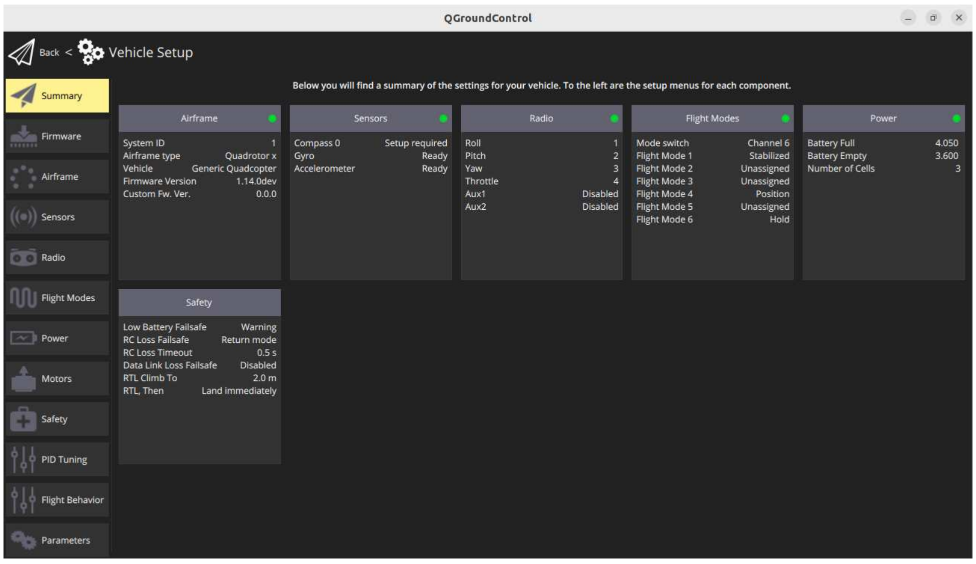
Figure 2.19. Image showing the VICON Tracker interface that was used in lab.

2.8 QGroundControl

This piece of software provides an easy interface (Figure 2.20) that enables to perform all the necessary steps in order for the drone to fly, once every component is up and running and correctly wired. Precisely, It gives the capability to calibrate the internal sensors, to setup the RC and the FC. In the context of this thesis, It was used for the initial configuration and calibration of each drone, for the assignment of specific parameters to allow the drone to be armed Offboard and to setup the system to accept external pose estimation. It was use extensively both during simulation and experimentation, to check the correct toggling of Flight Modes and mission execution.

2.9 Wrap Up

Once the Flight Controller, the Electronic Speed Controller, the Companion Board, the Radio Controller and the External pose estimate system setup is done, following the step highlighted in each above section, the next step was to perform several tests in order to check the correctness of the configuration in some practical examples. Therefore, in the next chapter, all the testing performed along with the problems and solutions, will be discussed in detail. Also, the focus will be to present the differences between the simulation and experimentation.



Below you will find a summary of the settings for your vehicle. To the left are the setup menus for each component.

Airframe ●

System ID	1
Airframe type	Quadrotor x
Vehicle	Generic Quadcopter
Firmware Version	1.14.0dev
Custom Fw. Ver.	0.0.0

Sensors ●

Compass 0	Setup required
Gyro	Ready
Accelerometer	Ready

Radio ●

Roll	1
Pitch	2
Yaw	3
Throttle	4
Aux1	Disabled
Aux2	Disabled

Flight Modes ●

Mode switch	Channel 6
Flight Mode 1	Stabilized
Flight Mode 2	Unassigned
Flight Mode 3	Unassigned
Flight Mode 4	Position
Flight Mode 5	Unassigned
Flight Mode 6	Hold

Power ●

Battery Full	4.050
Battery Empty	3.600
Number of Cells	3

Safety

Low Battery Failsafe	Warning
RC Loss Failsafe	Return mode
RC Loss Timeout	0.5 s
Data Link Loss Failsafe	Disabled
RTL Climb To	2.0 m
RTL, Then	Land immediately

Figure 2.20. Image showing the QGC interface overview of the internal sensors and components status.

Chapter 3

Testing the setup

3.1 Overview

Now that all the components have been mounted and calibrated, the next step is to test one drone, first in the simulation environment than in lab. In the following sections both topics are addressed, highlighting for each the problem encountered and how they were solved. At the end of the chapter, a few words of recap and an intro to the task that defines this master's thesis project.

3.2 Simulation Setting

For what regards the simulation environment, several time was spent at first to understand the version of px4 to use, since the Omnibus F4 is deprecated starting from px4 1.12. Nevertheless, the intention of this thesis was to try a configuration where all the software components are as updated as possible. Therefore, despite deprecation, we successfully managed to run the software stack both in simulation and in lab with real hardware using px4 v1.14 beta version, that at time of writing, represents the most updated version of Autopilot. Since in general, the software stack allows to install sets of modules defined for the specific hardware, in simulation tests are very close to the real setting scenario. In the following lines a recap of all the software used, as stated in the Hardware Description section:

- PX4 beta 1.14
- ROS2 Humble
- Micro-XRCE Agent
- Gazebo Garden

This software setup represents the most updated configuration available and allowed us to experiments the latest functionalities in terms of ROS2 middle-ware and Gazebo simulation environment.

3.2.1 First basic test

In first phases of the simulation tests, a Python script was written to allow a single drone to takeoff at 5 meters altitude, hold position for 8 seconds and then land. To write the code, [6] was used as reference. The entire basic mission is constructed and viewed as a state-machine, composed of actions expressed as commands for the drone to perform. This kind of "state-machine approach" is maintained from now on for all the upcoming missions, that rise in complexity, following this basic test.

According to the official documentation in [4], first the drone needs to change flight mode from "Stabilized Mode", which is the standard manual flight mode, to "Offboard Mode", which allows the drone to receive commands coming from an external script and not from the RC. Next, it needs to receive the position to reach, in this case to reach 5 meters in altitude and maintain position on x and y axes. To achieve both of this conditions, the algorithm that was created, publishes the required messages on two distinct ROS2 topics called "OffboardControlMode" and "TrajectorySetpoint".

Last step before the testing could be performed, was to do a series of frame transformations, since ROS2 and PX4 have different frame conventions, as stated in [3]. In particular, for what regards the body frame, PX4 uses FRD (X Forward, Y Right, Z Down) while ROS2 uses FLU (X Forward, Y Left, Z Up), whereas in the global frame PX4 uses NED and ROS2 uses ENU. The difference in frame reference can be easily viewed in figure 3.1. That said, if we want to send position commands as a form of waypoint to PX4 a transformation from FLU to FRD is required. On the contrary, to visualize the drone trajectory, since Rviz uses as ROS2 a FLU frame conversion for the body reference frame, the message coming from PX4 regarding the drone position needs to be transformed from FRD to FLU. If we are not doing so, of course everything will be messed up and the drone will not reach the desired positions.

In this basic test, since the drone performs just a change in attitude, just by adding a minus before the z coordinate will solve the issue due to the fact that in this case the difference between FLU and FRD is just a sign. Of course this doesn't apply on the general case where the entire x,y,z coordinate changes from waypoint to waypoint, hence some transformations have to be performed. Given the triplet of x,y,z coordinates, to perform a frame conversion from FLU to FRD just a rotation around X-axis of 180° Degrees is required. The same transformation applies to frame conversion from FRD to FLU. Therefore, the vector of coordinates

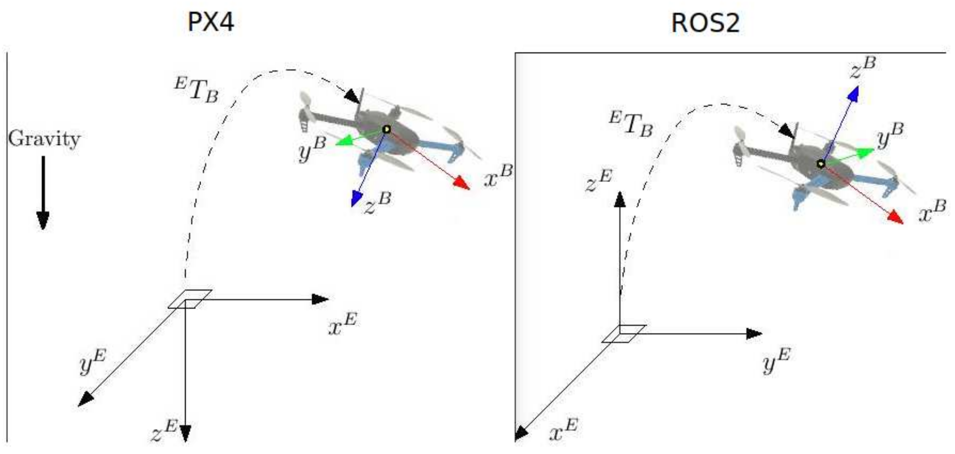


Figure 3.1. On left side PX4 FRD/NED reference frame, on the right side ROS2 FLU/ENU reference frame

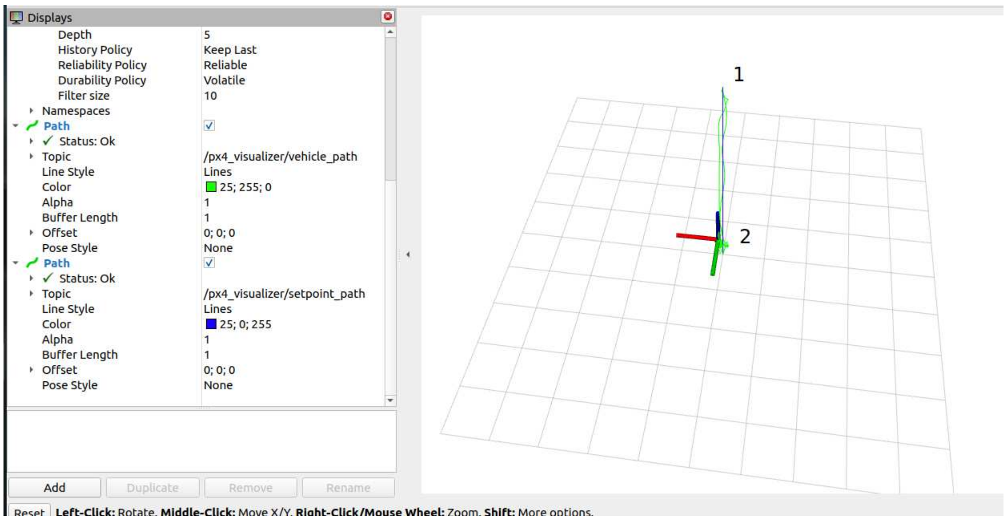
representing the position of the drone needs to multiplied by the following rotations matrix.

$$Rot(x,180°) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) \\ 0 & sin(\theta) & cos(\theta) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{3.1}$$

After all of this is done, the simulation runs smoothly and as can be seen from the following picture, the drone performs the test successfully. For more details on the implementation refer to [8], take from the official GitHub repository for this master's thesis project. One additional note is that, as can be seen from the picture, Rviz shows actually two trajectories. One is the actual one in green, the other is the setpoint ideal one, in blue.

Following this basic test, other ones were performed to assure that the frame convention was working correctly and that the drone was capable of reaching the specified sequence of waypoints. Below there is a picture highlighting one of these tests.

While performing these tests, one problem was that the drone didn't automatically disarm after landing. From Gazebo, the drone was clearly landing since reached the ground but from the FC Client perspective, the drone was seen as flying, hence the disarming cannot happen. We solved this issue by realizing that in order for the system to recognize the landing, "OffboardControlMode" and "TrajectorySetpoint" messages need to be shutdown before the drone reaches the ground.



Displays

Depth	5
History Policy	Keep Last
Reliability Policy	Reliable
Durability Policy	Volatile
Filter size	10
Namespaces	
Path	✓
✓ Status: Ok	
Topic	/px4_visualizer/vehicle_path
Line Style	Lines
Color	🟩 25; 255; 0
Alpha	1
Buffer Length	1
Offset	0; 0; 0
Pose Style	None
Path	✓
✓ Status: Ok	
Topic	/px4_visualizer/setpoint_path
Line Style	Lines
Color	🟦 25; 0; 255
Alpha	1
Buffer Length	1
Offset	0; 0; 0
Pose Style	None

Reset **Left-Click:** Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:** Zoom. **Shift:** More options.

Figure 3.2. Picture showing that the drone walked for the entire mission

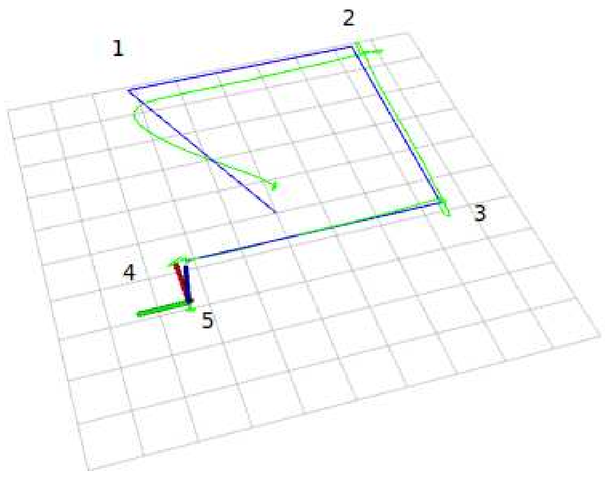


Figure 3.3. On left side PX4 FRD reference frame, on the right side ROS2 FLU reference frame

In this way, the UAV FC Client understands that the mission is completed because no more waypoints are received and even the request to stay Offboard ended, therefore It can recognize landing, shutdown the motors and disarm automatically.

3.2.2 Testing with multiple UAVs

Since the tests using one drone were all completed successfully, the next step was to implement a mission involving multi-drone offboarding. The PX4 software is equipped with the tools needed to perform multi-vehicle simulation. Each drone has its own instance, meaning a separate launch command has to be executed in order to have the two drones in the same simulation environment. This modularity allows to assign a name for each, for example "drone1" for the first drone and "drone2" for the second one, so that the topics assigned to each drone have as suffix the relative namespace. In this way, each drone has its own set of ROS2 topics easily identifiable and their operations can be commanded separately.

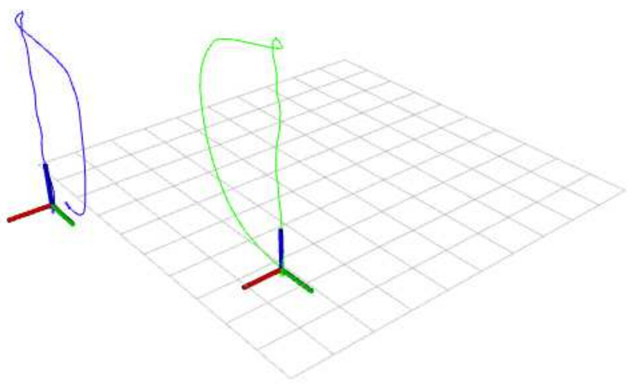


Figure 3.4. Picture showing that the drone walked for the entire mission

In the mission, one drone was spawned at x = 0 and y = 3, according to PX4 reference frame and the other at x = 0 y = 6. Moreover, each drone got a difference instance name, allowing to differentiate between the two. Mission was planned so that the two drones takeoff simultaneously, keeping their x and y start positions and just reach the altitude of 5 meters. Next, they keep their y and z position while departing from one another on the x axis positions. Lastly, they land on the current position and disarm. The entire mission planning trajectory execution is shown below in figure 3.4 .

Finally, the simulation environment is configured and ready for more complex tasks as the one described in the second part of the thesis. Now, the lab environment has to be configured and set accordingly, in order for the drones to performed the same basic tests done in simulation, with the appropriate changes to account for errors in localization, loss of data and more.

3.3 Real Setting

For what concern the lab experiments, there are several issues to analyze since the real drone flying configuration was not as smooth as in the simulation scenario. Hence, some considerations will be discussed regarding the problems of arming, of manual control and of linking with the VICON system, showing also the solution adopted to overcome them.

3.3.1 Arming the drone

At start, the first step was to correctly arm the drone. At the beginning, all the drone propellers were removed and using the RC, the drone was told to arm by moving the corresponding stick. The drone was also connected to the GCS via a microUSB-USB cable in order to link it to QGroundControl. The first problem encountered was that, upon trying to arm the UAV, the motors weren't moving, even if in the QGroundControl interface the display showed that the drone was in arming mode. After checking the log, the message showing the arming was toggled on, so the solution of the problem was to find elsewhere. After checking some forums and documentation on the internet, where people had the same issues of arming, It was discovered that some additional parameters had to be inserted inside the QGroundControl Parameters panel and hence inside the drone. In the following lines those parameters are shown in the following 3.1:

Upon changing those parameters, the drone motors started spinning accordingly, when the RC SA switch assigned to arming was used and if the throttle lever was put down, as mentioned in the Radio Controller section of Hardware Configuration and in figure 2.13. Therefore, the UAV was now ready for the first manual flights.

3.3.2 Manual control tests

After performing the first flights with the drone, some problems arose. First of all, the drone itself was not very easy to handle and guide, even if "Stabilized Mode"

Parameter Name	Value	Comment
COM_CPU_MAX	-1	Maximum allowed CPU load to still arm
SYS_MC_EST_GROUP	2	Set multicopter estimator group to EKF2
SYS_HAS_GPS	0	No GPS availability
COM_ARM_WO_GPS	1	Allowing to arm without GPS
SYS_HAS_MAG	0	No magnetometer availability

Table 3.1. Table representing PX4 parameters changed to allow arming.

was used, which should be a relatively easy mode to fly the drone with. Moreover, after disarming the drone motors where all more or less hot upon touching, signaling that something wrong is going on. In fact, after checking the logs of the flight, It was discovered that the "Yaw Angular Rate" and "Roll Angular Rate" quantities are to far away from their respective setpoints, whereas "Pitch Angular Rate" is acceptable. In addition to that, as can be seen from figure 3.5, there are high irregular spikes that are affecting the system. This anomalous behaviour is responsible of the high heat coming from the motors and on the fact that the UAV piloting was hard to execute. In order to solve that, It was necessary to do two additional steps. First, an ESC Calibration using QGroundControl menu interface by plugging the drone with the gcs. Second, a simple PID tuning was done by setting the multiplier to almost zero. After performing such operations, a great improvement happened in terms of maneuverability of the UAV during flight and motors didn't heat so much. Picture 3.6 shows that both quantities are now acceptable compared to the previous state, the peak-to-peak is lower and the overall trend is better following the one of the setpoint, indicating that now the drone can be piloted smoothly.

Now that the drone is capable of flying with manual control without major issues, the next step is enable autonomous flight by linking it with the VICON localization system.

3.3.3 Tests using the VICON system

As described in the Hardware Configuration subsection, the VICON is a technology of Motion Capture characterized by a set of cameras that are pointing to the inside of the laboratory cage, allowing to localize precisely where the drone is, at any time. This architecture is mandatory since autonomous mode can be powered on only if an external estimation system is at place, as stated in [1].

First of all, in order for the drone to receive the position information from the VICON system, a dedicated C++ script had to be produced using the VICON

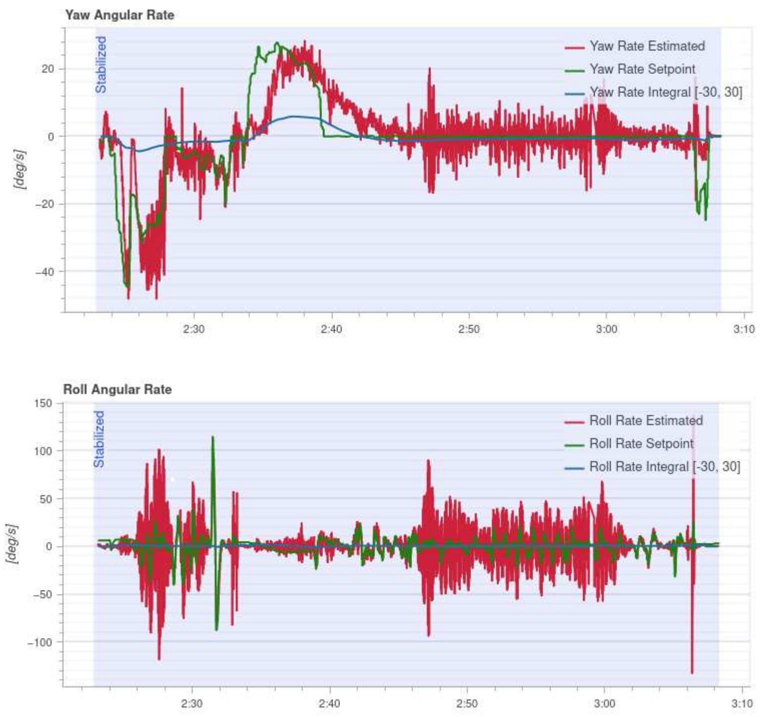


Figure 3.5. Picture highlighting a major issue with yaw and roll speed variation w.r.t. their setpoints

SDK. The script essentially connects to the cameras, extract the position info and transforms it into PX4 language by publishing a specific topic called "vehicle_visual_odometry". The Agent inside the Companion Board mounted on the drone transmits the topic information to the Client Flight Controller and therefore It can be fused with the IMU internal position estimate by the Estimated Kalman Filter.

Second of all, a list of FC internal parameters had to be changed to set the drone for external position estimate. On table 3.2, those parameters are described briefly. With that done, the drone is capable of receiving the external vision data to fuse inside the ekf.

At this point, the mission script created previously to perform the basic tests can also potentially be utilized to do the autonomous real tests. However, to check if the drone ekf fuses correctly the position information coming from the VICON system and to pilot the drone in a safe way, It was decided first to flight with manual control and not directly in autonomy.

Indeed after several trials, even if the VICON system was connected to the UAV, this one didn't receive correctly the pose information but, as can be seen

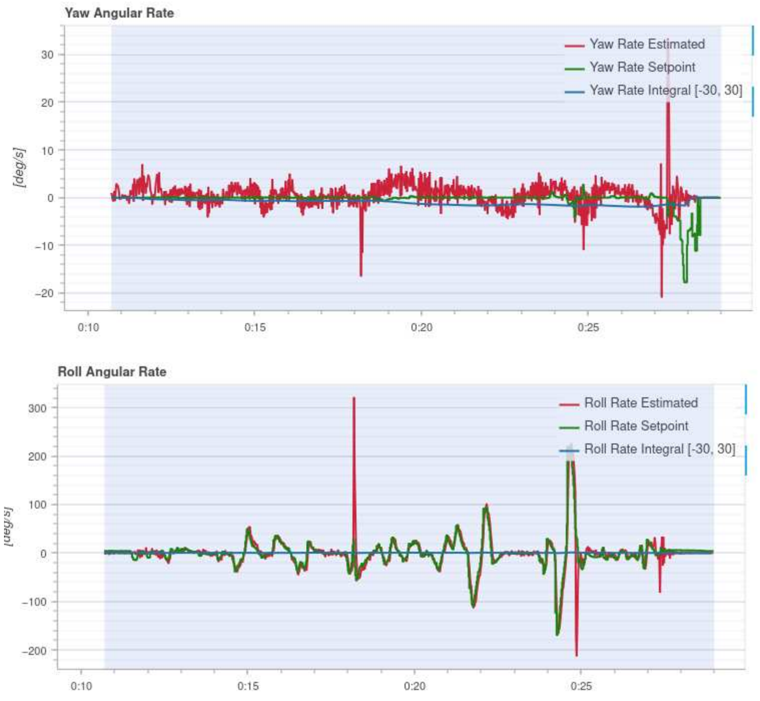


Figure 3.6. Picture showing yaw and roll angular rate after solving the issues

Parameter Name	Value	Comment
EKF2_MULTI_IMU	0	Running a single IMU and EKF
EKF2_IMU_CTRL	1	Accounts for the Acceleration Bias inside the IMU
EKF2_EV_CTRL	11	Enables yaw fusion
EKF2_EV_DELAY	50	Delay in ms between VICON and IMU timestamps
EKF2_HGT_REF	3	Sets external vision as source for height data

Table 3.2. Table describing PX4 parameters that were changed to allow external position estimate.

from the picture 3.7, the x,y,z positions, plotted w.r.t time, have multiple areas where no info is received for about a second, highlighted in the figure by the fact that the graphs are all composed by dashed lines, instead of continuous ones. This translates, from the drone perspective figure 3.8, in a erroneous fusion of the VICON data by the ekf and therefore there are still dashed lines in the position coordinates. As can be easily noticed, this situation is extremely dangerous since the drone in some parts of the flight looses information of its position which can be produce anomalous behaviours. Moreover, another proof that something is

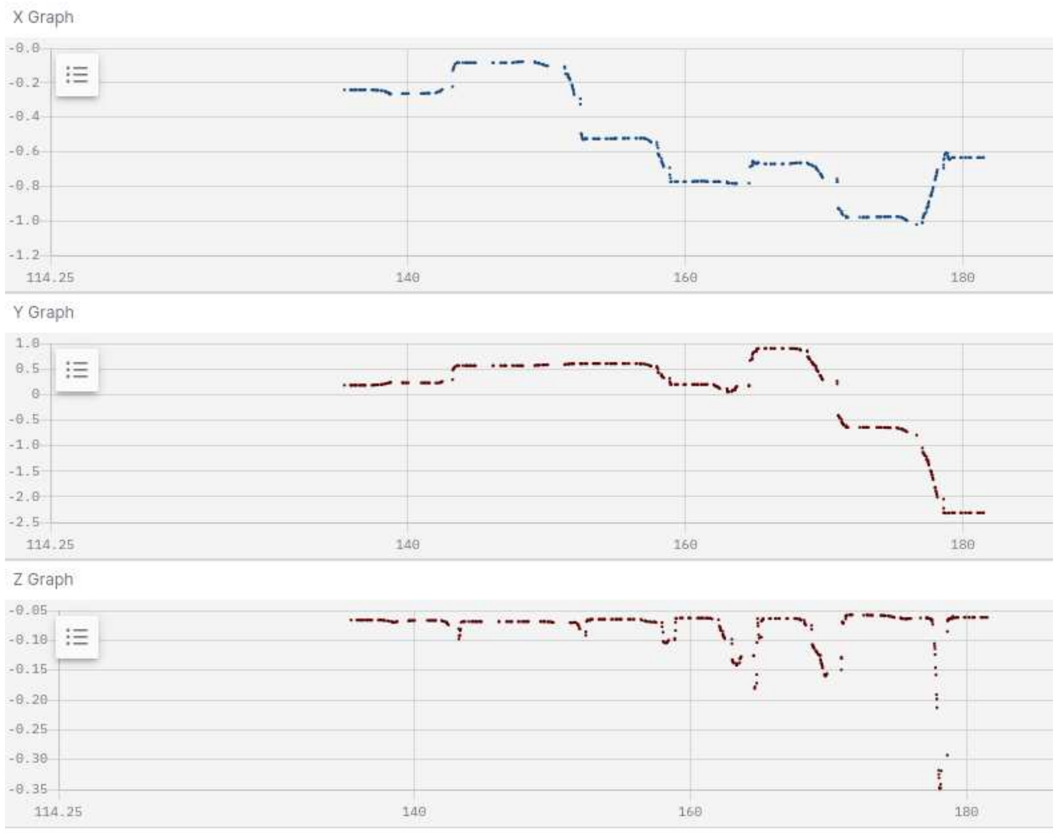


Figure 3.7. Picture showing the position data coming in from VICON from the Flight Controller perspective.

going wrong is that It can be seen from logs that internally, the FC ekf is indeed losing a lot of packages regarding the "vehicle_visual_odometry" message. A last indicator of a major loss in data packages, reside in the "reset counter" parameter of PX4. This quantity identifies how many times the ekf had to reset its fusion algorithm due to erroneous data. Indeed, picture 3.10 shows that during the flight 189 packages between x,y and z coordinates had been lost.

Nevertheless from the VICON only perspective, as can be seen from figure 3.9, the graphs are continuous and smooth, also there is no presence of dashed lines, indicating that the Motion Capture cameras actually was sending the data correctly. This suggests, that the problems reside only in the communication between VICON, the Companion Board and the GCS. At first, It was hypothesize a problem of the C++ script used to receive and send the VICON position estimate data but, after several trials in which some parameters where changed shown in table 3.3, the same behaviours described were occurring with only a slightly

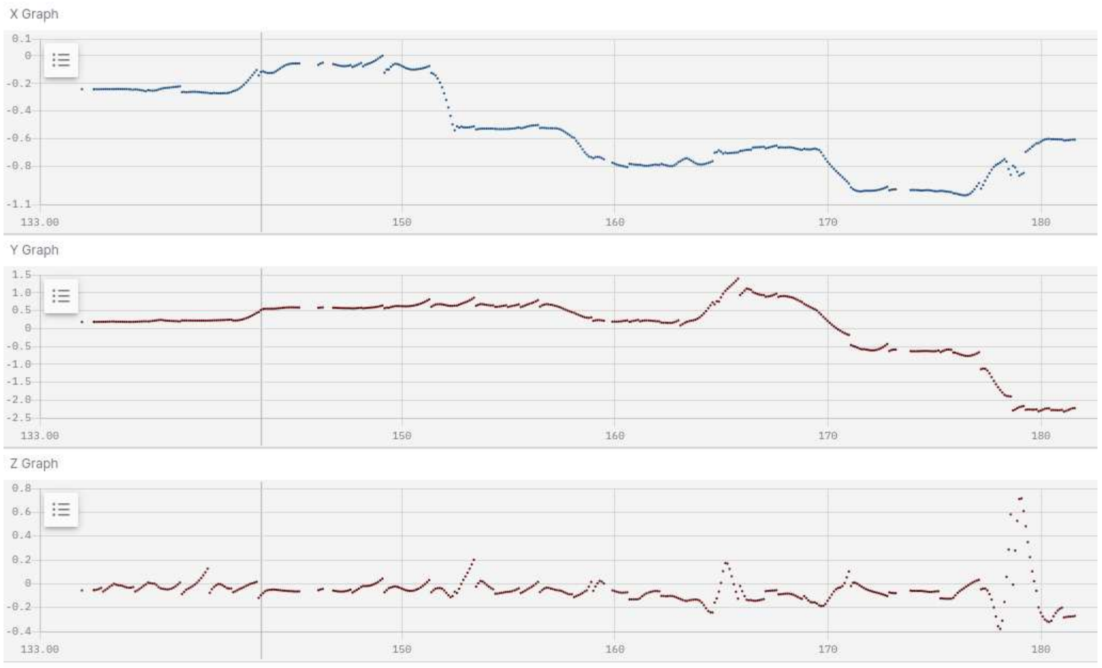


Figure 3.8. Picture showing the position data after EKF2 fusion with the VICON data.

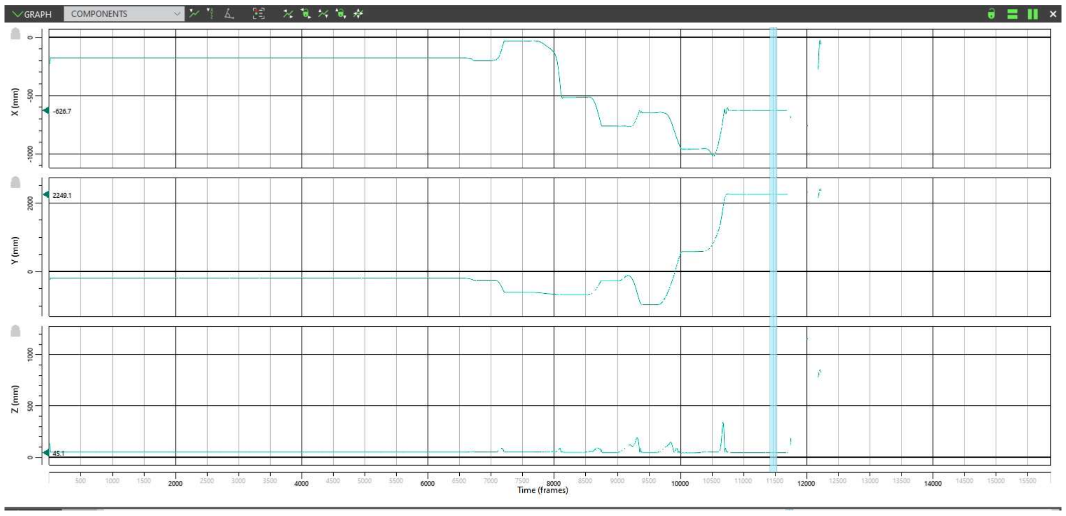


Figure 3.9. Picture showing the position data going out of the VICON system from the VICON perspective.

improvement.

Parameter Name	Value	Comment
EKF2_BARO_CTRL	1	Allowing Barometer for height estimation
EKF2_ACC_NOISE	0.8	Making EKF2 more resistant to vibrations
EKF2_BARO_NOISE	6	Threshold noise in barometric measurements
EKF2_GYR_NOISE	0.05	Threshold noise in gyroscope measurements
EKF2_EV_NOISE_MD	0	Taking noise info from vision message
IMU_ACCEL_CUTOFF	30.0	Low pass filter cutoff frequency for acceleration

Table 3.3. Table showing PX4 parameters that were changed in order to try and solve the dashed line error.

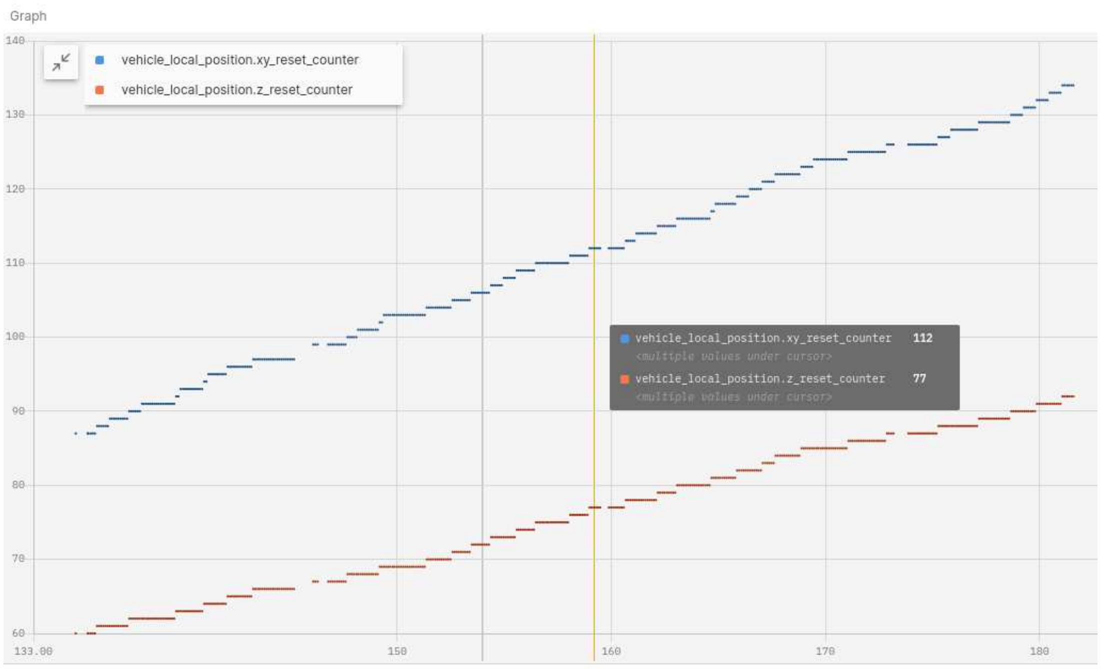
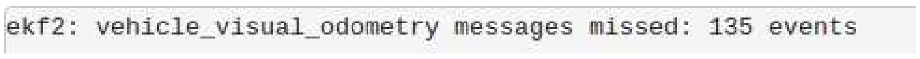


Figure 3.10. Picture representing the reset counter of position data by the EKF2 module inside the Flight Controller.

After a while, It was found out that the RC and its receiver operate at a bandwidth of 2.4 GHz, same as the WiFi module inside the NanoPi Neo Air Companion Board. Hence, considering that both components have the same bandwidth, they interfere with each other, disturbing the VICON data transmission to the FC.

Therefore, the solution was to unplug the receiver mounted on the drone and power off the RC to allow the UAV to receive correctly the external position



```
ekf2: vehicle_visual_odometry messages missed: 135 events
```

Figure 3.11. Picture from logs showing how many data packages have been lost.

estimate and to enable it to fly in Offboard mode. Indeed, another possible solution was to buy a receiver with a different bandwidth, i.e 800 MHz, so that Receiver/Radio Controller and Companion Board couldn't disturb each other, while maintaining the ability to take manual control if something wrong during Offboard happened. Since It was not possible to find in time a replacing receiver, It was decided to adopt the first solution, therefore to fly the drone in Offboard Mode without manual control as a backup.

That said, before proceeding with offboard tests, a form of emergency handling was needed, in case something went wrong during the flight there had to be a way of powering off the devices without incurring in component damages. The solution found was to exploit a feature of the PX4 Autopilot system, which is the "Flight Termination Kill". This behaviour allows to switch off the drone motors instantly and It was implemented inside the RC in the upper right switch. Since as mentioned above, the Radio Controller is not usable during Offboard, this feature was implemented as a ROS2 command and could be activated when needed from a terminal window on the GCS.

Finally, the drone was capable of flying offboard and had also a backup stopping if anything bad occurred. The next step was to perform some tests using first one drone, then two drone at the same time.

3.3.4 Tests in Offboard Mode

At this point, everything was ready for autonomous flights. After powering up the drone, the VICON system and checking that the two were communicating while having a terminal window with the "Flight Termination Kill" command ready, It was time to do a first test. Indeed as can be seen from the pictures 3.13 there was some improvement w.r.t. the tests with manual control involved. This flight had almost half of the reset counters in comparison with the previous one, allowing for a more stable and acceptable autonomous flight. Still, as can be seen from 3.12, in the graphs involving the "vehicle_visual_odometry" position messages, there are multiple spikes of anomalous data that are fed inside the ekf. Fortunately, the latter internal system is capable of interpreting the data and comparing it with the internal sensors. Considering that the spikes were very short in time and too much different w.r.t. the IMU estimation, the ekf discarded didn't consider them as plausible position estimate.

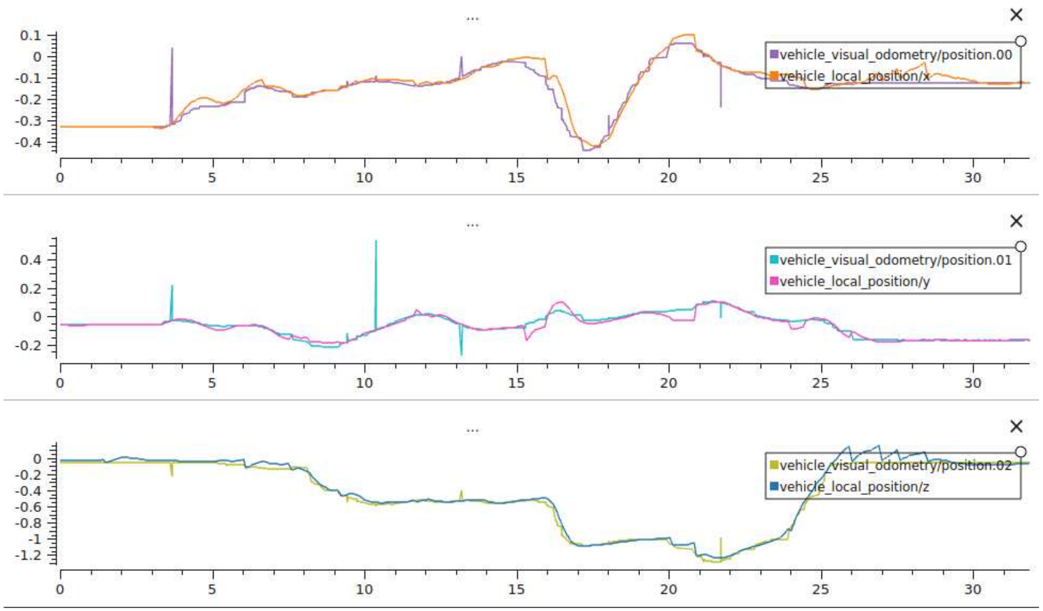


Figure 3.12. Picture representing position data of the FC compared with the info coming from the external position system.

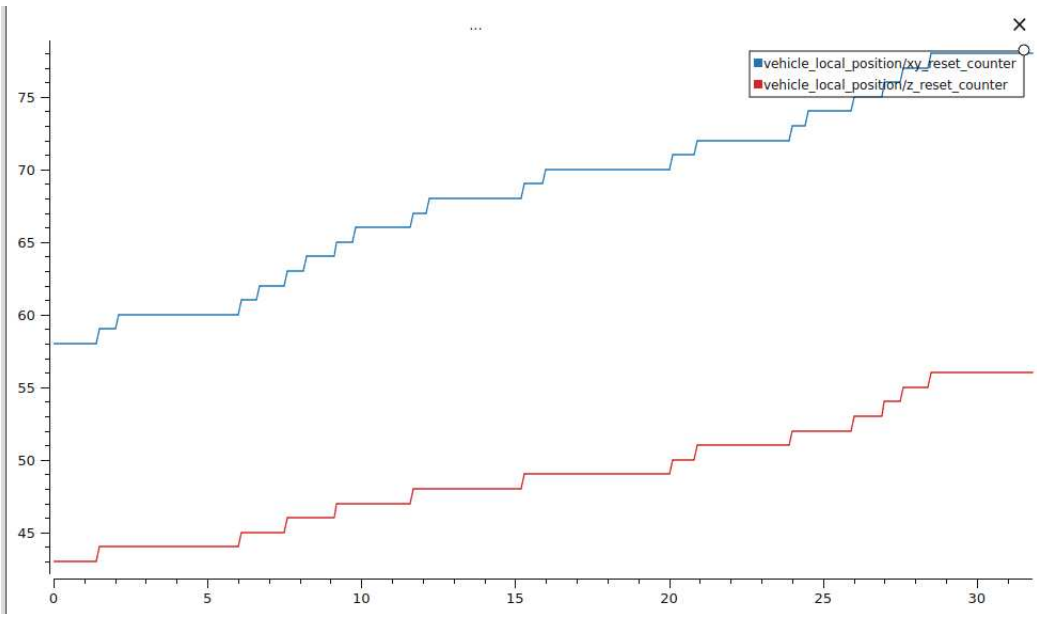


Figure 3.13. Picture showing the reset counter of position data by the EKF2 module inside the Flight Controller during Offboard Mode.

This suggests that some additional effort was still needed in order to zero out the reset counters and allowing to have a smooth flight, with no lost packages or multiple anomalous spikes in the position data.

During the time where we were trying to investigate the problem of huge package loss, one solution found to reduce them was to use a WiFi router inside the laboratory totally dedicated to the drone tests. In this way, we could be sure that all the bandwidth used was between the VICON system, the NanoPi and the GCS and everything was configured so that both external pose estimation system and the GCS were connected to the router via Ethernet, ensuring the maximum speed and accuracy of communication, leaving just the NanoPi to connect to the router using UDP protocol. Indeed, the use of a dedicated router had an huge impact in performances as can be seen from the picture 3.15, no more resets happened with the ekf which means no more package loss. The effect of this can be seen in figure 3.14 where, It can be noticed, there are no more spikes coming from the external position, the graphs are smooth resulting in the drone finally following correctly the desired trajectories.

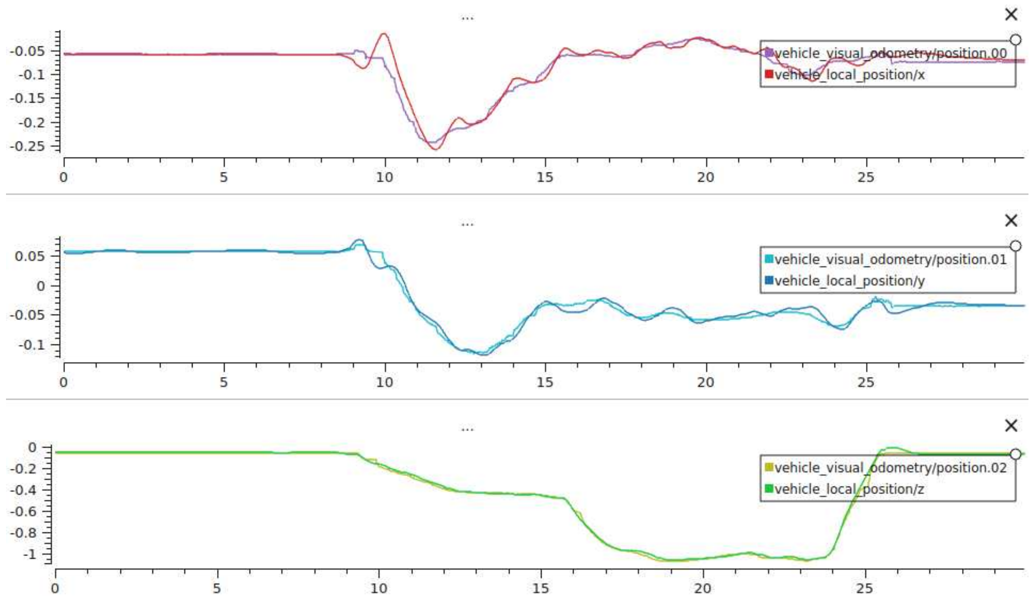


Figure 3.14. Picture representing position data of the FC compared with the info coming from the external position system, coming from a single drone simulated mission.

With the problem of localization solved, everything was ready for a test using two drones at ones. For this type of tests, the same procedure applicable to one drone was used, adding just the instance of the second drone in the algorithms that translate the VICON position into PX4 ROS2 message, adding a few more lines

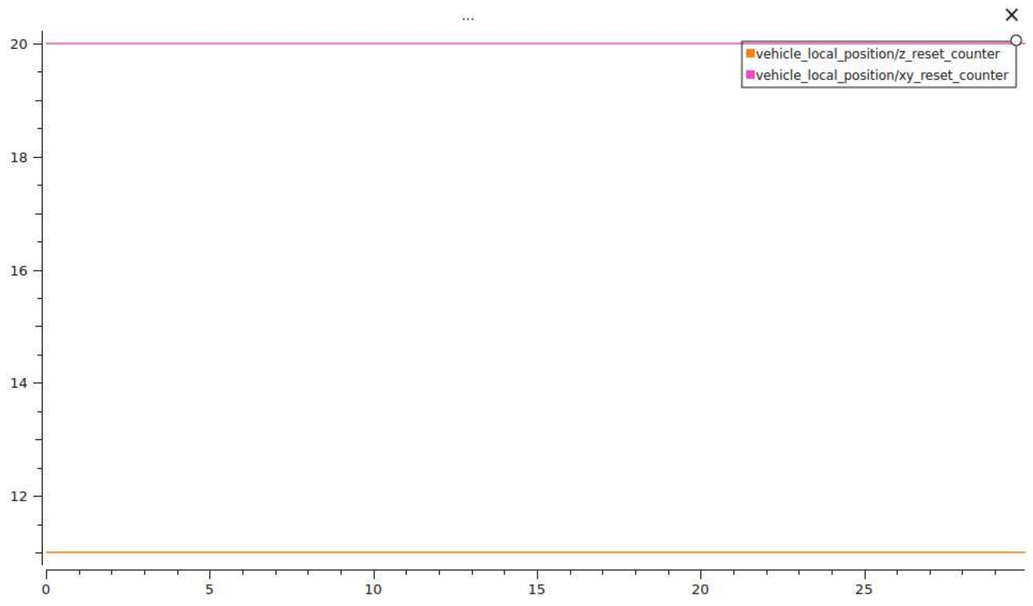


Figure 3.15. Picture the reset counter of position data by the EKF2 module during Offboard Mode after using exclusively the dedicated router, coming from a single drone simulated mission.

to assign a set of waypoints to reach also to the second drone and an additional terminal window was dedicated for the "Flight Termination Kill" of the second drone. As expected, the flight was neat and both drones performed the assigned mission correctly. Pictures 3.16, 3.17, 3.18, 3.19 show just that. Notice that the x,y reset counter for both drones resets just once, having no influence whatsoever on the trajectories of both drones.

With all that said, all the tests, performed considering some basic missions with the goal to configure the system, resulted in a clean autonomous flight configuration and hence everything was setup for some more advanced mission tasks. On the following section, just a recap will be presented, summarizing briefly the problems and solution and introducing the next chapter focused entirely on the final task to be performed by configured pair of drones.

3.4 Wrap up

This chapter was focused on presenting the struggles and solutions in order to setup and configure a pair of two drones, both in simulation than in the real setting. In the former, despite some initial trouble regarding the not recognized landing of one drone, everything else ran pretty smoothly. The same thing cannot be said for

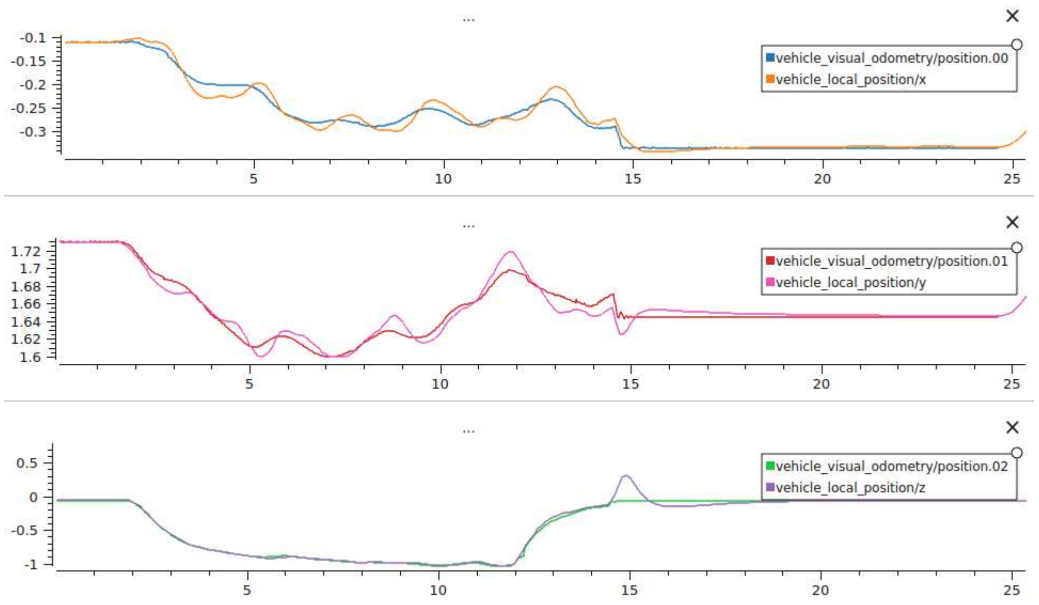


Figure 3.16. Picture representing position data of the drone 1 FC compared with the info coming from the external position system, coming from a two drones simulated mission.

the tests in real setting, since several issues were faced in order to make first one than two drones work in the same mission. First, the arming problem was solved by configuring some parameters inside the QGC interface.

Second, the problem of package loss was solved by realizing that the problem was the Receiver/Radio Controller and the NanoPi acting on the same bandwidth. Therefore, by unplugging the first, the problem was solved partially.

Since, some huge spike were present still and some resets with the ekf counters were happening, just by using a dedicated router connected only the mission system, the package loss problem was solved, allowing to perform neat missions, first with just one drone than with a pair of two simultaneously.

That said, the next chapter will characterized in detail the not so trivial task of Coverage Path Planning, first by introducing the state-of-the-art and then by showing the tests performed both in simulation and in the real setting, with the same approach used for this chapter.

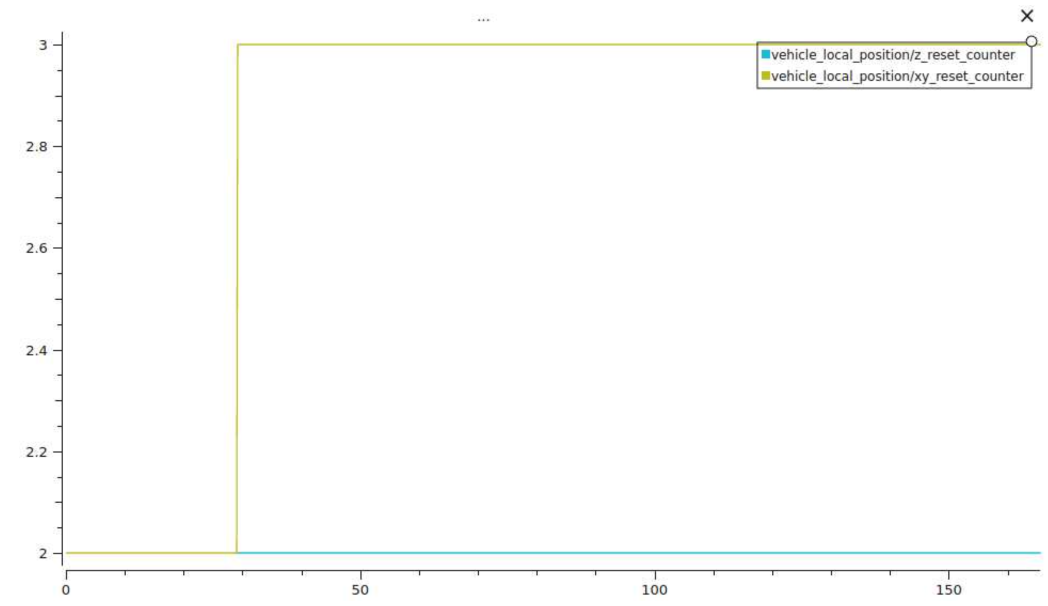


Figure 3.17. Picture the reset counter of position data for drone 1 by the EKF2 module inside the Flight Controller during Offboard Mode, coming from a two drones simulated mission.

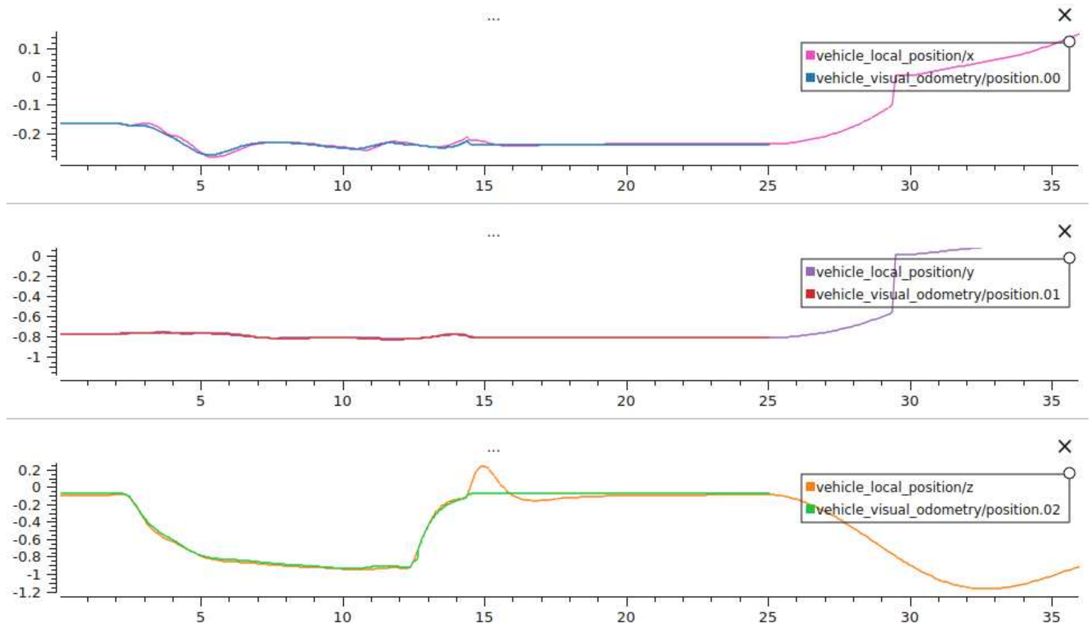


Figure 3.18. Picture representing position data of drone 2 the FC compared with the info coming from the external position system, coming from a two drones simulated mission.

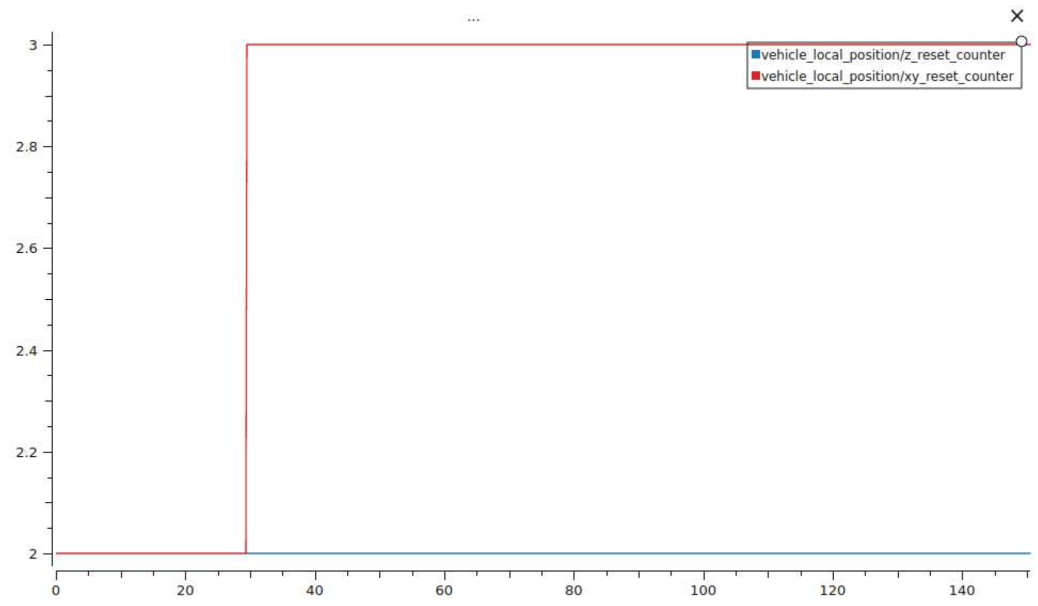


Figure 3.19. Picture the reset counter of position data for drone 2 by the EKF2 module inside the Flight Controller during Offboard Mode, coming from a two drones simulated mission.

Part II

Second Part

Chapter 4

The coverage path planning problem

4.1 Task and state-of-the-art description

As mentioned in the Thesis Goal section, after doing the configuration and setup of a multi-drone system, the aim is to perform a mCPP to map the indoor cage floor. To accomplish such an objective, a proper state-of-the-art research has been conducted to categorize the different algorithms that can be used. Before diving into that, first some context has to be given in order to understand better what we are talking about.

According to [10], the act of coverage means to find a path for the UAVs to follow, in order to completely map an environment while staying collision free and eventually avoiding obstacles that they may find along the way. Moreover, the Coverage Path Planning problem derives from the Travelling salesman problem where the goal is to define a path that visits a set of points once and to return to the starting point. In addition to that, the covering path problem addresses also the presence of obstacles and therefore finds a path collision and obstacle free for all the robots involved.

As stated in [9][5] the steps to perform the task are:

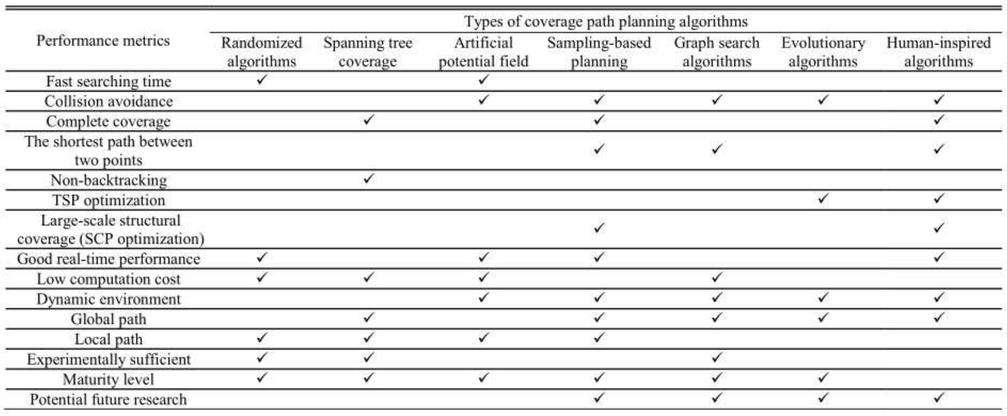
1. Define the coverage area by either getting the occupancy grid map or by specifying the vertices of the polygon that represents the area. The setting in which the multi-drone system operates can be:
    (a) Offline: the setting is perfectly known and location of static obstacles is given, typical of indoor environments.
    (b) Online: no prior info is given, hence the drones have to use sensors to locate themselves and map the environment around them properly,

typical of outdoor environments.

2. Divide the area in cells that correspond to the size of each drone. This technique is called cellular decomposition. Two types exist:
   (a) Approximate: the decomposition is done on a surface which is irregular so that the area cannot be 100% covered.
   (b) Exact: the target area surface is such that can be fully covered.
3. Split the area assigning each sector to a drone.
4. Each drone covers its subarea using a specified algorithm.

That said, methods that leverage exact cellular decomposition in an offline setting, can be considered to accomplish our task since the laboratory environment is a rectangle 7×3 meters, of course fully explorable and known.

Once the coverage area is defined and each drone has its subarea assigned, the final step is to specify an algorithm for the coverage. Again in [10], an analysis of the algorithm for CPP is done, emphasizing the advantages and disadvantages of each technique. The following picture 4.1, taken from the aforementioned article, summarises these concepts.



Performance metrics	Types of coverage path planning algorithms						
	Randomized algorithms	Spanning tree coverage	Artificial potential field	Sampling-based planning	Graph search algorithms	Evolutionary algorithms	Human-inspired algorithms
Fast searching time	✓		✓				
Collision avoidance			✓	✓	✓	✓	✓
Complete coverage		✓		✓			✓
The shortest path between two points				✓	✓		✓
Non-backtracking		✓					
TSP optimization						✓	✓
Large-scale structural coverage (SCP optimization)				✓			✓
Good real-time performance	✓		✓	✓			✓
Low computation cost	✓	✓	✓		✓		
Dynamic environment			✓	✓	✓	✓	✓
Global path		✓		✓	✓	✓	✓
Local path	✓	✓	✓	✓			
Experimentally sufficient	✓	✓			✓		
Maturity level	✓	✓	✓	✓	✓	✓	
Potential future research				✓	✓	✓	✓

Figure 4.1. List of CPP algorithms each with specified performance metrics.

Among all, for this master's thesis the STC approach has been chosen. The main reason is its non-backtracking feature, that guarantees a closed path coverage in the least amount of time since it explores every cell just once. Another important feature is the simplicity to implement it since is a well-known problem in the field and in the literature.

First, the coverage path problem was considered starting with the setup of just one drone, than It was extended to two drones. We will consider each of such cases separately, focusing on highlighting the issues and the solutions found.

4.2 Single drone coverage path planning

For a single drone, the aforementioned steps are valid but some additional considerations have to be made.

1. Construct a 2Dx2D sized Grid Map and assign to the drone a starting cell. Each cell central point represents a node.
2. Link each cell with Its neighbouring cells, defining edges therefore a graph structure.
3. Apply DFS algorithm to find one of the MSTs associated with that graph.
4. Divide each cell into four sub cells of size D and identify the starting sub cell central point from where the drone will start the coverage.
5. Circumnavigate the MST with a counterclockwise trajectory covering each sub cell once.

Figure 4.2 summarizes briefly what had been just described.

That said, two remarks are needed in order to clarify certain aspects. First, since in simulation the drone at disposal as approximately a size of 50x10x50 It was decided to assign D = 50, hence a sub cell has a size of 50x50 centimeters while a cell is 100x100 centimeters. Second, in general the DFS algorithm cannot guarantee that the exploration with generated an MST but, in this case, since all the weights associated to each edges are equal to 1 meter, the result cannot be other than that. Moreover, since a starting point has been specified, just one possible spanning tree will always be produced, even though in general to the graph multiple spanning trees are possible, for example generated using Kruskal algorithm instead of DFS.

Now, a separate describing will be given for simulation testing and for real testing, giving an idea of the issues and solution found along the way for each.

4.3 Simulation testing

As mentioned in the previous section, the simulated model of the drone has size 50x10x50 which is different w.r.t. the real drone that is of size 17x7x15. Considering that for the coverage mission It was considered as covered an area that's as

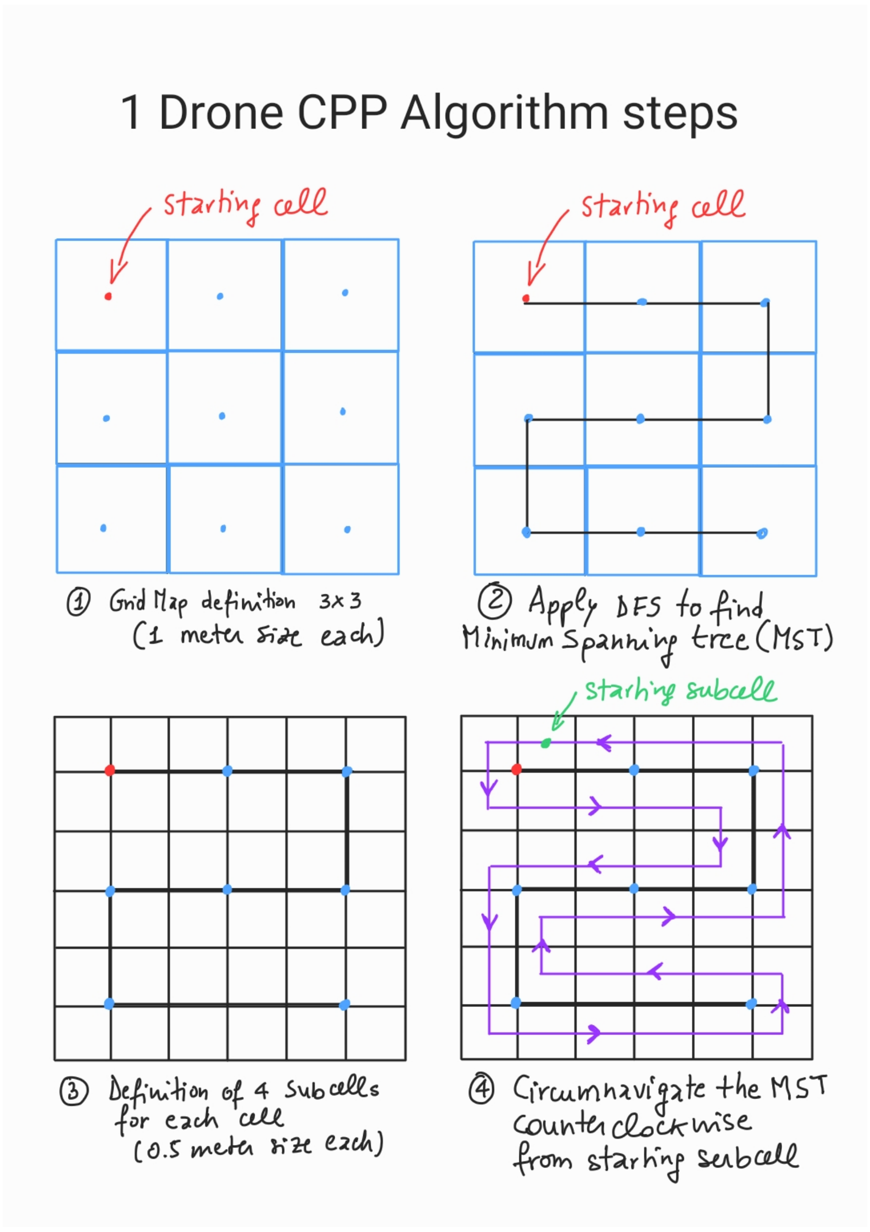

1 Drone CPP Algorithm steps
starting cell
starting cell
① Grid Map definition 3x3 (1 meter size each)
② Apply DFS to find Minimum Spanning tree (MST)
starting subcell
③ Definition of 4 Subcells for each cell (0.5 meter size each)
④ Circumnavigate the MST counterclockwise from starting subcell

Figure 4.2. Info-graphic showing the logic behind the algorithm used to perform single CPP.

big as its drone size, this means that different grid maps will be considered for simulation and real experimentation. In simulated environment, the grid map is 3x3 with each sub cell having D = 50 centimeters. The results are shown in the figure 4.3.

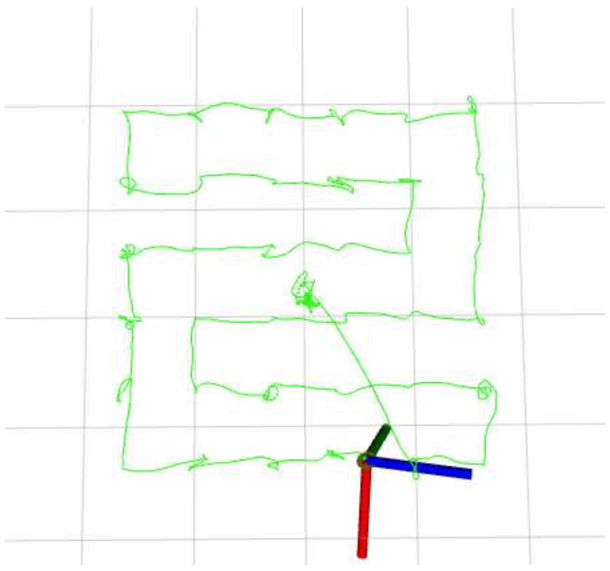


Figure 4.3. Representation of the Rviz2 trajectory capturing the coverage path planning inside Gazebo.

Considering that in the real setting some robustness in the algorithm is needed to ensure its efficacy, some of that was already implemented and tested in simulation. This means the mission still kept a state machine like structure and inside each state a check was performed and the state switch to the next one only if the test was passed. First, the drone mode is changed in Offboard and a test is performed to ensure that. Second, the drone is armed and until the command is fully acquired by the drone and executed no further advance in the state machine is allowed. Once arming is done, the drone gets its first waypoint to reach that belongs to the coverage path planning trajectory. Here, a test comparing the setpoint waypoint coordinates and the actual drone components guarantees that the drone move to the next sub cell only if the difference between its position and the setpoint isn't higher than five centimeters. This test is crucial to ensure that the coverage act is done with precision. Finally, for landing a specific command is given and the drone automatically lands and disarms after arriving on the final sub cell composing the coverage path planning trajectory.

4.4 Experimental testing

With respect to the simulation tests, since the real drone had a size of 17x7x15 cm, hence much smaller than the gz x500 model, the coverage area was changed accordingly, in order for the UAV to cover cells that have more or less its size. At first, the real drone had to cover an area of 11x3 cells of size D = 30 cm each. The area picking was done considering all the available space inside the laboratory cage, excluding squares that were too close to walls and net cage, since that would've put the drone at risk of severe damaging.

Before actually testing the drone in laboratory, a trial was conducted using the Gazebo Garden simulator, to check if everything was working correctly and in fact, the drone performed the coverage as expected, in accordance with the previous tests.

Nevertheless, when a first trial was performed on the real drone inside the laboratory, the mission in part was accomplished as far as the UAV stayed in the central area but when the drone was reaching a more edgy waypoint provided by the coverage algorithm, It lost the connection with VICON localization system and It was behaving erroneously. Prepared for these kind of emergency situations the command "Flight Kill Termination" was sent from the GCS and the drone landed and stopped, without major issues, even though It took a significant hit. As can be seen from the picture 4.4 in the first 15 seconds the VICON localization worked but after that It sent anomalous peaks of data, indicating that the drone was at 5 meters altitude, which was obviously wrong. Consequently, the internal local position was changed based on that but the result was not as intended.

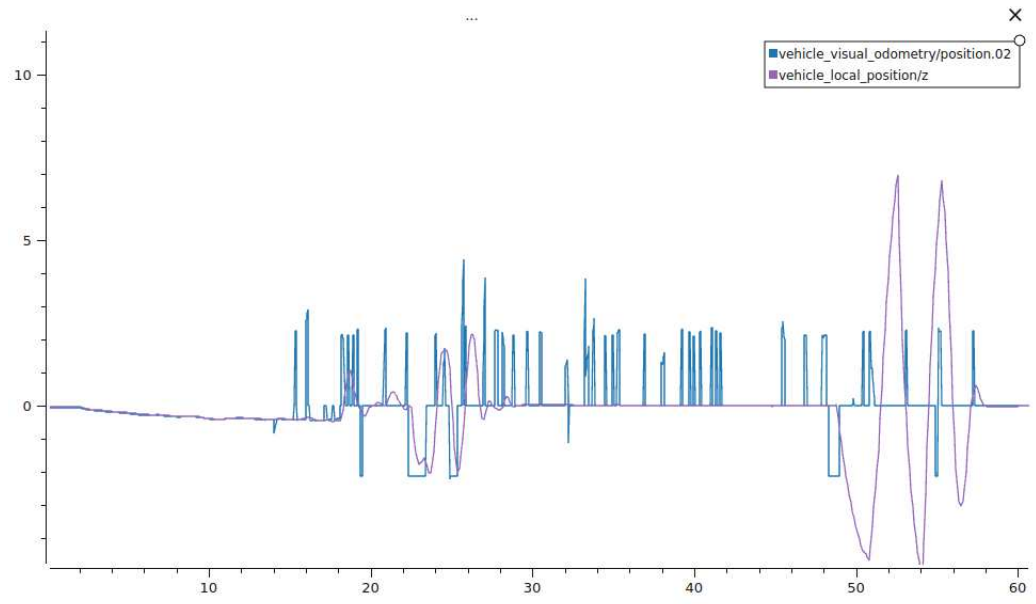


Figure 4.4. Figure taken from the drone logs. It can be seen that the VICON message is totally misleading and wrong.

After that episode, some investigations on the VICON setup itself showed that the VICON cameras lights were producing reflections that were erroneously interpreted inside the VICON system as markers of potentially other drones. Moreover, performing some tests even in the corners of the cage using a rover, showed that even in that scenario the VICON transmission was not good at all, ending in a misleading transmission of position data.

Fortunately, the VICON Tracker interface, that is the Front End software used to calibrate the system of Motion Capture Cameras had the option to mask some areas inside the cage, in this case the reflections, leading to a cut of wrongs markers appearance and overall in a more stable flying environment. Nevertheless, since the problems with wrong position data at the edge of the area was not fully solved, It was decided to restrict the coverage area to just a block of 60 cm squared cells at the center. Therefore a 4x2 region was used to perform the Coverage Path Planning task, as can be seen from the picture 4.5.

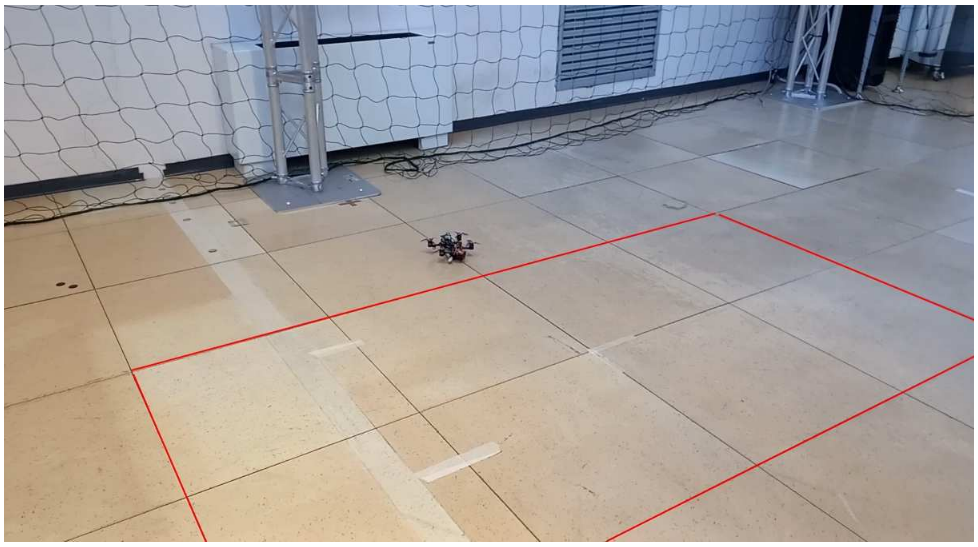


Figure 4.5. Figure representing the area covered by the drone used in laboratory.

With that accounted for, then the further trials of coverage ended successfully. Nevertheless, upon doing some more experimentation, It was necessary to change aspects of the algorithm with respect to the simulated scenario. Those changes are described briefly above:

- Relax the thresholds used to define if the drone was arriving to the setpoint sub cell. With a thresholds seated to 10 cm, the coverage was executed but It was raking too long, since the drone in flight was not very stable and It was taking too much time to recharge the required precision. Instead, he thresholds have been set to 15 cm on the x,y,z axis positions, ending in a much smoother and faster coverage.

- The PX4 landing command in version 1.14 was bugged and It didn't function properly. It was a well known issue but at the time of testing, It was not fully fixed by the development team that maintain the autopilot. Hence, as already mentioned in previous sections, a flight termination kill command was used. Of course, since that command would power off the drone entirely, when the drone finished to cover every point of the chosen area, It was given a setpoint to reach near the ground, same x and y and z equal to zero. After reaching almost the ground, then the flight terminal command was executed, allowing the drone to "land".

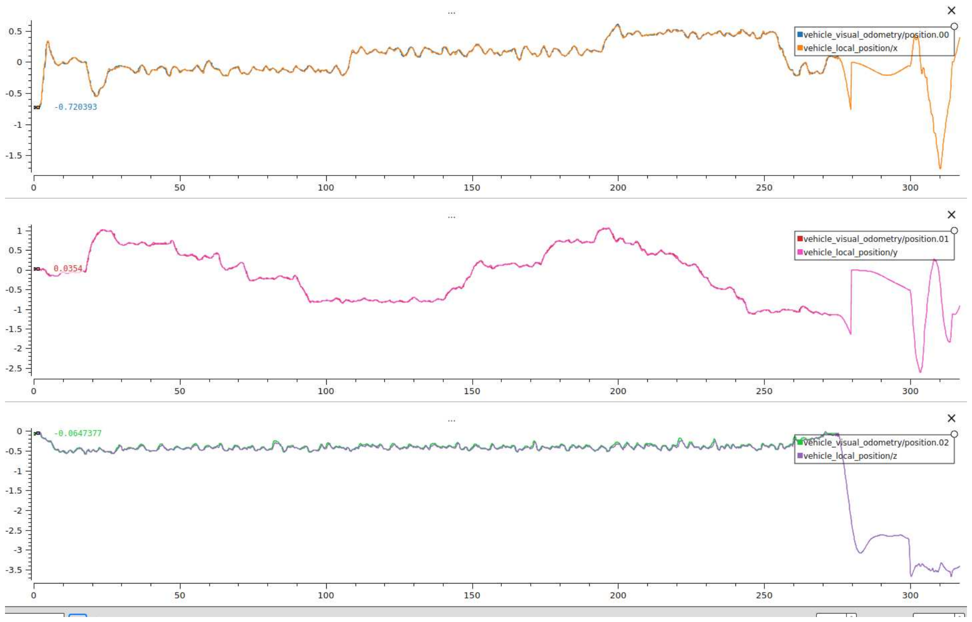


Figure 4.6. Figure describing the coverage path planning drone position w.r.t. VICON data. The flight ended after 270 sec.

As can be seen from the picture 4.6, the local position on all axes of the drone followed the VICON data precisely and the data was actually correct but the flight last too much, for the reasons described already above. At 270 sec, after almost 5 minutes of flying the drone landed automatically due to the fact that the battery almost discharged considering the long flight. That meant the drone only covering more or less 70% of the total area that was assign to.

After lessening the thresholds, picture 4.7 reveals a complete Coverage Path Planning of the assigned area in just 110 sec, not even 2 minutes of flight. With

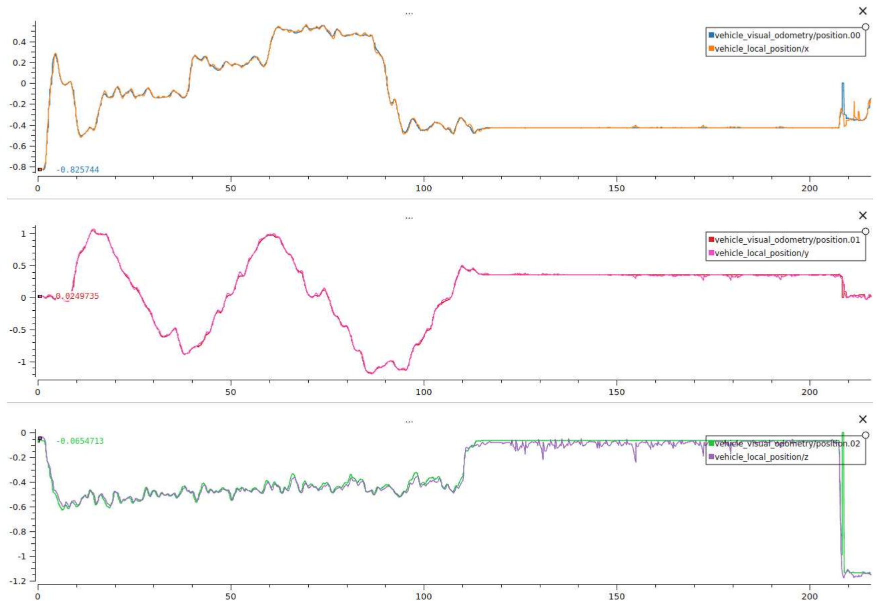


Figure 4.7. Figure representing the area covered by the drone used in laboratory. The flight ended after 110 sec.

that result, for what regards the single coverage path planning, the mission was accomplished. The Next section will address the efforts in extending the problem to two drones and how It was possible to perform a working simulation of that.

4.5 Pair of drones coverage path planning

In the case of mCPP, additional considerations must be done in order to assign each drone its portion of area to cover. In the following lines the updated algorithm will be described briefly:

1. Construct a Grid Map of size DxD cells and assign to each robot an initial position.
2. Calculate for each drone its Evaluation matrix (E), quantifying for each cell Its Euclidean distance from the k-drone initial position.
3. Use the E matrix to calculate the Assignment matrix (A), which allocates each cell to be covered the nearest drone. Moreover, in the cases where both drones have equal distances from certain cells, those were assigned half to one drone and the other half to the other in order to have, at the end, the same number of cell assigned to the drones.

4. Use the A matrix to define a Grid Map for each drone, composed of all the cells assigned.

5. Use each subarea defined by each drone Grid Map to construct an MST in each drone restricted ROI. Each MST computation is performed using the same DFS algorithm used in the single drone coverage path planning problem.

6. Use the MST as a pathway in order for each drone to circumnavigate their respective areas only.

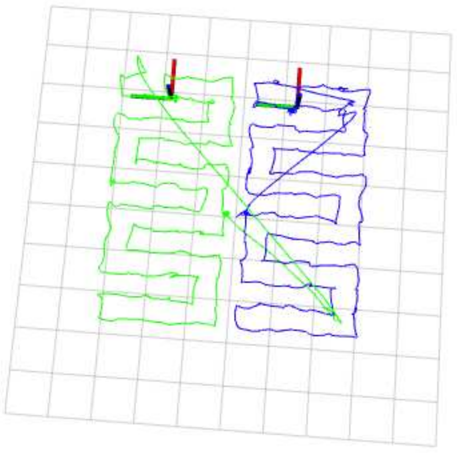


Figure 4.8. Representation of the Rviz2 trajectory capturing the coverage path planning of the two drones inside Gazebo.

Picture 4.9 summarizes the steps analyzed above. An important notice is that, when performing the tests in simulation, an odd situation was registered. By choosing as initial positions the top left corner and the bottom right one instead, the algorithm performed well until the last step. While attempting circumnavigation, the algorithm entered a dead end scenario in which It couldn't find a trajectory that was circumnavigating all the MST in its totality. Numerous attempts were performed in order to address the issue but none of them worked and It was decided to change initial positions and retry. Indeed, after choosing as initial positions the top left corner and the top right corner the algorithm worked according to the expectation and in simulation the theoretical behaviour was reproduced, as can be seen from picture 4.8.

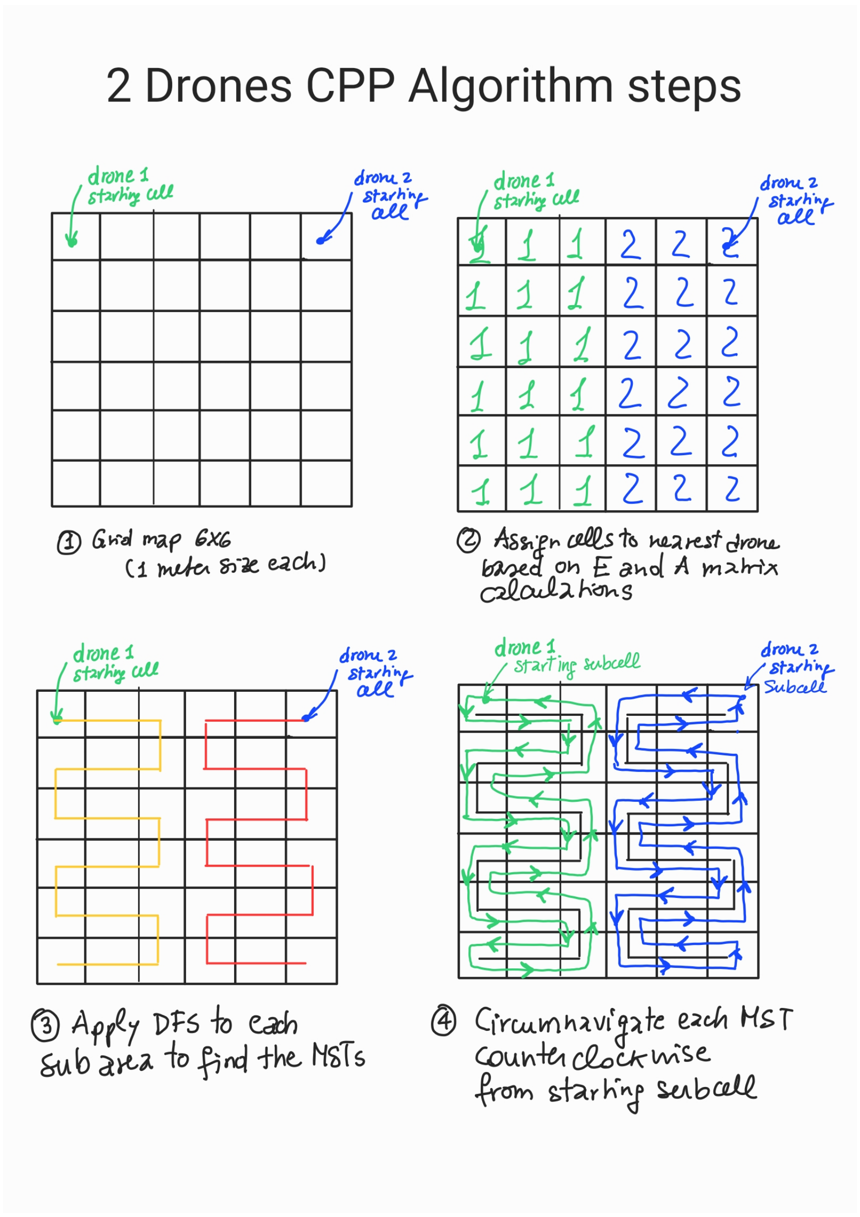

2 Drones CPP Algorithm steps
drone 1 starting cell
drone 2 starting cell
① Grid map 6X6 (1 meter size each)
drone 1 starting cell
drone 2 starting cell
② Assign cells to nearest drone based on E and A matrix calculations
drone 1 starting cell
drone 2 starting cell
③ Apply DFS to each sub area to find the MSTs
drone 1 starting subcell
drone 2 starting subcell
④ Circumnavigate each MST counterclockwise from starting subcell

Figure 4.9. Steps to perform in order to execute a mCPP mission composed of two drones.

4.6 Wrap up

That was all regarding the description of the Coverage Path Planning performed in the single and paired case.

In the former, since the drone size differ in the simulation with respect to reality, the assigned area to cover changed accordingly. In the real scenario, a loss of localization data was experienced at the borders of the laboratory cage, hence some reflection masking correction was done in order to try and solve the issue. However, since the problem didn't disappear even after that, It was decided to assign a smaller portion to cover at the center, to have always a precise localization. Indeed, the coverage path planning mission was accomplished, even though some relaxation of thresholds on the reaching of the setpoints was necessary to reduce the coverage time.

In the latter, performed exclusively in simulation, some effort was allocated to define a simplified version of the DARP algorithm. Each drone had Its area assigned based on calculation considering their initial distance with respect to each cell. After that, another operation allowed to assign each cell to the nearest drone. However, during simulation tests in the algorithm a dead end was find, indicating that It doesn't work for every possible scenario, hence some further investigation can be done in order to solve the issue, as follow-up work to this master's thesis project.

Part III

Conclusions

In the following document, a detailed and exhaustive analysis of a master's thesis project regarding the configuration of multiple drones, both in hardware and in software, was described. With all the work done in the first part, drones that have the same components as the one proposed can be setupped successfully and equipped with the autonomy capability. However, as highlighted in several occasions, the setup had some limitations mainly focused in low RAM and low processor speed of the NanoPi Companion Board. This device represents the bottleneck that in many ways limits the capability of autonomy. One important limitation that was not addressed in this project since It was not a major concern, is the fact that ROS2 software cannot run nodes inside the NanoPi Companion Board. For this reason, the task of coverage path planning couldn't be decentralized, meaning It wasn't possible to run the algorithm for each drone inside the Companion Board itself but It was necessary to use a GCS that was computing the algorithm and was sending to the drone just the set of waypoints, one at the time.

To try and solve the aforementioned limitation a possible work can be the one of exploiting the Modal AI VOXL Companion Board instead, which has very good specifications in comparison with the NanoPi board. Indeed, some effort have to be put in order to setup correctly the communications between the Client on the FC and the Agent in the VOXL Companion Board.

Since the tests of the paired coverage path planning problem were just conducted on simulation, the testing in the real setting with real hardware can be done also as a follow-up step to this project. The algorithm itself is already equipped with all the internal checks needed to run a smooth flight testing. The only issue with the testing of such a setting is the problem of wrong localization, if the drones reach the edges of the cage. Therefore, a careful analysis have to be done in order to choose an area, as central as possible, to be sure that the drones never lose the external position estimate. Indeed, in conjunction to that, a deep analysis of the VICON system can also be perform in order to better calibrate the Motion Capture system and maybe solve the issue with wrong localization on corners area.

Finally, several other algorithms can be tested, for example some others mentioned in [10], and an analysis of their performance can be done, comparing the results with the one coming from the STC algorithm used in this project. Moreover, some obstacles can also be put as a part of the equation and see how the various algorithms perform in such a scenario.

Bibliography

[1] DroneCode Fundation. External position estimation, 2023. URL https://docs.px4.io/main/en/ros/external_position_estimation.html.

[2] DroneCode Fundation. Omnibus f4 sd, 2023. URL https://docs.px4.io/main/en/flight_controller/omnibus_f4_sd.html.

[3] DroneCode Fundation. Px4-ros2 frame conventions, 2023. URL https://docs.px4.io/main/en/ros/ros2_comm.html#ros-2-px4-frame-conventions.

[4] DroneCode Fundation. Offboard mode, 2023. URL https://docs.px4.io/main/en/flight_modes/offboard.html.

[5] Yoav Gabriely and Elon Rimon. Spanning-tree based coverage of continuous areas by a mobile robot. *Kluwer Academic Publishers*, 2001.

[6] Jaeyoung Lim. px4-offboard, April 2023. URL https://github.com/Jaeyoung-Lim/px4-offboard.

[7] Davide Morazzo LINKS Fundation, Ronald Cristian Dutu. Configuration of flight controller, 2023. URL https://github.com/links-cosero/swarm_uav_config/blob/main/drone_setup_docs/config_v1.14.md.

[8] Davide Morazzo LINKS Fundation, Ronald Cristian Dutu. Ros 2 offboard control example for 1 and 2 drones, 2023. URL https://github.com/links-cosero/swarm_uav_config/blob/main/offboard_ws/src/px4_offboard/README.md.

[9] Javier Muñoz, Blanca López, Fernando Quevedo, Concepción A. Monje, Santiago Garrido, and Luis E. Moreno. Multi uav coverage path planning in urban environments. *MDPI*, 2021.

[10] Chee Sheng Tan, Rosmiwati Mohd-Mokhtar, and Mohd Rizal Arshad. A comprehensive review of coverage path planning in robotics using classical and heuristic algorithms. *IEEE Access*, 2021.