UNIVERSIDAD
POLITÉCNICA
DE MADRID

Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Industriales

Máster Universitario en Automática y Robótica

Trabajo Fin de Máster

Mini-drone autonomous missions for indoor warehouse logistics

Autor: **Miguel Granero Ramos**

Director 1: **Pascual Campoy**

Director 2: **Pedro Arias**

Tutor Externo: **Jérôme Rutinowski**

December, 2022

A mis padres.

Acknowledgements

I would like to personally thank the *Computer Vision and Aerial Robotics (CVAR)* group of the *Universidad Politécnica de Madrid* and the *Lehrstuhl für Förder- und Lagerwesen (FLW)* of the *Technische Unversität Dortmund* for making this thesis possible as a collaboration between the two universities. It has been a very enriching experience both academically and professionally.

Special thanks to my main tutors in Madrid and Dortmund, Pedro Arias and Jérôme Rutinowski, who have guided and helped me greatly in the development of this thesis. And also to Pascual Campoy for offering me this great opportunity.

Resumen

Esta tésis trata sobre el desarrollo de una plataforma de control y comunicación para los mini-drones Crazyflie dentro del framework de navegación de UAVs, Aerostack 2, desarrollado actualmente por el grupo Computer Vision and Aerial Robotics (CVAR) del Centro de Automática y Robótica (CAR), para realización de misiones autónomas en el área de logística en interiores. Para ello es necesario que la plataforma desarrollada se integre con todas las funcionalidades existentes dentro de Aerostack 2, así como el desarrollo de nuevos módulos necesarios para su correcto funcionamiento.

Además, se propone un caso de uso de identificación de europallets mediante técnicas de visión por computador y aprendizaje profundo. El módulo de expansión de Crazyflie AI Deck incorpora una cámara RGB y conectividad Wifi que puede ser empleado para transmisión de imágenes en directo. Estas imágenes son posteriormente procesadas por un sistema de identificación de pallets basado en el trabajo aportado por la Universidad Técnica de Dortmund. En él se emplea un modelo YOLO para la detección de los bloques de soporte de europallets y una posterior identificación del pallet mediante una arquitectura basada en Part-based Convolutional Baseline (PCB).

Ya que la tésis es el resultado de una colaboración entre la Universidad Politécnica de Madrid y la Universidad Técnica de Dortmund, gran parte del desarrollo y pruebas se han realizado en las instalaciones de la TUD en Alemania.

Abstract

This thesis addresses the development of a control and communication platform for Crazyflie mini-drones within the UAV navigation framework Aerostack 2, currently being developed by the Computer Vision and Aerial Robotics (CVAR) group of the Center of Automation and Robotics (CAR), for autonomous missions in the field of indoor logistics. For this, it is necessary that the developed platform is integrated with all the existing functionalities within Aerostack 2, as well as the development of new modules necessary for its correct operation.

In addition, a use case for the identification of europallets using computer vision and deep learning techniques is proposed. The Crazyflie AI Deck expansion module incorporates an RGB camera and Wifi connectivity that can be used for live image transmission. These images are subsequently processed by a pallet identification system based on work contributed by the Technical University of Dortmund. A YOLO model is used for the detection of europallet support blocks and subsequent pallet identification using a Part-based Convolutional Baseline (PCB) architecture.

As the thesis is the result of a collaboration between the Polytechnic University of Madrid and the Technical University of Dortmund, most of the development and testing has been carried out at the TUD facilities in Germany.

Index

Chapter 1

Introduction

1.1 Background and Rationale

This Master thesis is the result of a collaboration between two universities. *Universidad Politécnica de Madrid (UPM)* and *Technische Universität Dortmund (TUD)*. The group of *Computer Vision and Aerial Robotics (CVAR)* from the Centro de Automática y Robótica (CAR) of the UPM and the chair for*Lehrstuhl für Förder- und Lagerwesen (FLW)* of the TUD started their contact at the end of 2021 with the intention of mixing the areas of expertise of both research groups into a common project, autonomous mini-drone missions for logistics in warehouses.

1.2 Motivation

Warehouse management is in fact one of the major areas of research in logistics, for which it is essential to have methods for recognising and locating products, usually grouped on pallets. The classic methods are dependent on physical **labels** that are attached to the surface of these pallets. Therefore, they are dependent on a supply of physical labels and on specific external systems for reading these codes.

For these reasons, the possibility of employing a **recognition system** that is not dependent on external tags is highly attractive in current research. More specifically, systems that are capable of recognising the pallet by its own characteristics and materials (type and peculiarities of the wood). To this end, the rise of **deep learning** has helped the development of new technologies applicable to this sector. The *FLW* has developed a deep learning model which, in a similar way to facial recognition models, is capable of extracting **pallet characteristics**, to create a database of products and be able to identify them later.

The potential application of these techniques with mobile systems in **autonomous**

missions is very attractive, but no similar applications or attempts have yet been found in the sector. **Drones** are the perfect match for this problem, their portability and agility make them ideal for this type of task, as they do not require major installations or additional infrastructure.

One of the drones gaining popularity in recent years is the **Crazyflie**. This is a mini drones with a modular and open source architecture and a research focus. They are fully programmable and customisable, so they can be integrated with existing software and applications. The *FLW* has experience with these drones and they are especially interesting for logistics applications, due to its small size and agility. The Crazyflie also has an expansion board with an RGB **camera** and **WiFi** connection, the *AI Deck*. This deck makes possible the use of Crazyflies for perception and computer vision applications.

As important as the hardware base is to have software capable of executing the actions necessary for the task to be solved. The standard in open-source robotics software is ROS. The first version of this software ends its useful life in 2025, but its successor **ROS 2** is already available to developers. This second version solves many shortcomings and problems of its predecessor and presents itself as the option of choice for the long term. Although solutions already exist in ROS 2 for controlling vehicles of other types, such as Nav2 [1] for land vehicles, there is no comparable software package for UAVs.

Aerostack 2 (AS2) is a framework for aerial vehicles being developed by *CVAR*. It should serve as an abstraction layer over the different hardware platforms as well as a complete suite of navigation, control and state estimation software, among others, for UAVs. An union between the Crazyflie and Aerostack 2 seems really interesting as a base for research in more high-level tasks.

This thesis tries to unify the three technologies mentioned above, thus becoming a **proof of concept** for autonomous missions for indoor warehouse. Therefore, the thesis has a strong research focus on a rather broad field of application, and not the development of a final, commercial product.

1.3 Objectives

Taking into account the above, the main objective of the thesis is an autonomous mini-drone mission for indoor warehouse logistics. It could be divided into the following parts:

1. Integration of the **Crazyflie** drone into the Aerostack 2 framework by means of a **software platform**.

2. Integration of the **Vicon** system with Aerostack 2 and the drones.

3. Set-up of the **AI Deck** expansion board with camera and Wifi connectivity and its use for image streaming

4. Processing of these images with the existing pallet **recognition model**.

5. Creation of **autonomous** pallet recognition **mission**.

Chapter 2

State of the Art

In this chapter, the actual state of the technologies implied in the thesis will be covered. First of all, an overview of the actual use-cases for mini-drones in industry and research. Secondly, an introduction to the selected drone for the tasks, as well as the current recommended high-level software. Lastly, the current techniques for pallet recognition in the industry.

2.1 Mini Drones

A rise in the popularity of mini drones has been seen in the last years. The low prices and capabilities of the new technology have led to an increase in the number of companies adventuring themselves in the world of mini drones. A lot of commercial models can already be found in the stores. Due to the small size and lift power, mini drones are mainly used in tasks such as surveillance, military, swarming and imaging as we can see in the main scientific papers' portals. These drones are the perfect tool for research in swarming algorithms and navigation in 3D spaces. They are inexpensive and small-sized, which enables investigating high densely populated spaces in real hardware instead of simulation. But what is exactly a mini drone? The concept of what mini means has also changed in the last years, according to [2] a drone is categorized as mini when it weighs less than 25kg and its range is smaller than 10km. But this is far from today's idea of a mini drone. To define, or at least narrow down, the meaning of mini drone, it is necessary to review the characteristics of the main commercial drones in use today.

All the drones in the Table 2.1 are announced as mini or small drones by their vendors. Usually they are accompanied by a programming SDK, which allows the user to program custom applications. These SDK, may vary in quality and level, since some of the drones have a completely open source background, and others are proprietary commercial products.

Drone	F. Time [min]	Range [m]	Mass [g]	Size [mm]	Connectivity	Price [€]
Crazyflie	7	100	27	92x92x29	Bluetooth + 2.4GHz Radio (+ Wifi)	200
DJI Tello	13	100	87	98x92.5x41	2.4 GHz Wifi	159
Parrot Mambo	9	60	62.9	180x180x39.8	Bluetooth + 2.4GHz Radio	160
DJI Mini 2	31	6K - 10K	249	245x289x56	2.4GHz - 5.8 GHz Radio + GPS, GLONASS, Galileo	459
DJI RoboMaster TT	8	100	87	98x92.5x41	Bluetooth + Wifi	260
Parrot Anafi	25	4K	320	175x240x65	2.4GHz - 5.8 GHz Wifi + GPS, GLONASS	450

Table 2.1 Drone Comparison table.

Within this table, two main groups can be seen. The DJI Mini 2 and Parrot Anafi are professional mini drones which offer a complete solution for navigation and imaging. This leads to an increased price and connectivity, since they both are the only ones that have GPS connectivity. Their size is also considerably bigger, even though they are still considered mini drones, which also allows them to have a bigger range and flight time capability. The rest of them have very similar capabilities, with a time of flight not bigger than 15 minutes and a range up to 100 metres. This is due to their size, around 10 cm per side in most cases. Most of these drones are not announced as professional tools, but as products for enthusiasts, education and research with less native capabilities.

Figure 2.1 DJI Mini 2. www.dji.com/de/mini-2

Therefore, within the category of mini drones, it is also possible to make a clear distinction between the general mini drones and the so called micro or nano drones. The latter are usually much smaller drones, with a size of up to 15cm per side and flight autonomy of around 10 minutes. These drones are usually not thought to be a professional commercial

product, but as a tool for research and education. Due to their smaller size, the inertia when crashing is reduced, which leads to minimal security risks when flying and experimenting with them. Furthermore, due to their low price, it is possible to have a larger number of drones, and thus to experiment with much more densely populated spaces.

2.1.1 Applications and Research

The literature on mini drones is not very extensive due to their recent popularity. Most of the publications are concerned with modelling and control of mini drones, which serves as a basis for higher level applications. Although this is not the focus of this thesis, it is important to consider the state of the art in this sector.

Drones are highly non-linear systems. The design of controllers for these kinds of dynamics can turn really complicated, since the classical **PID** approach may not be the most optimal option. In one of the most relevant publications on the subject, [3], a mathematical model for one of the reviewed drones, the **Parrot Mambo**, is developed. And on top of this model simulation, a model based controller is presented.

Figure 2.2 Parrot Mambo. www.willinternational.co.uk/products/parrot-mambo/

These kind of models are crucial for the correct and stable controlling of mini drones. In [4] the strengths of **nonlinear adaptive control strategies** are shown. Demonstrating that these approaches can yield much better results than classical techniques, increasing stability, performance indexes. For example, it was shown that the *overshoot* can be reduced to 0% , a difficult dynamic to achieve with, for example, a PID.

The improvements in the control of these systems, has led to the popularisation of other commercial tasks with mini drones. Their agility and reduced cost, make them the perfect solution,for example, for drone shows, which have become really popular in the last few years. An example of this is the company **Dronisos** [5]. These shows are based on a great number of drones that are able to coordinate themselves and make 3D formations for spectators. It may seem like a simple task, but the technology implied in the communication and coordination of the drones has been and still is a great field in the research panorama.

This technique of coordinating drones is what is commonly named as **swarming**. Swarming is also a major area of research today. This name is not tied to the world of

Figure 2.3 Drone light show in China. [6]

technology. Its most generic definition according to *Wordreference* [7] is:

1. *A body of honeybees that leave a hive and fly off together, accompanied by a queen, to start a new colony.*

2. *A great number of things or persons moving together*

This is not the exact meaning of swarm and swarming in robotics. According to [8], a swarm can be defined as:

"A robot swarm is a group of three or more robots that perform tasks cooperatively while receiving limited or no control from human operators."

Here, two key concepts are introduced: **cooperation and independence**. Cooperation does not necessarily mean that every single agent of the swarm has to communicate with each other, but that they all have to share a common task and way to complete it. It is also relevant, that the human control is involved, but it should only dictate the **general behaviours** of them, and not the specific commands for each of them.

Mini drones are also ideal for **indoor** tasks. **Logistics** and warehousing is an interesting field of application due to the space restrictions that are often present. In [9] a new system for inventory management using mini drones and based on *QR codes* is proposed. This opens the door to new challenges, such as indoor localization, reduced flight autonomy and navigation, but significant advantages in cost, security and time efficiency, which make the concept attractive to investors. The publication shows that the state of

the art is still far away from a commercial product, but also yields promising results in terms of accuracy and velocity.

2.2 Frameworks for drones autonomous missions

This section aims to give a general overview of the actual frameworks used for the development of unmanned aerial vehicles (**UAV**) missions, also often called Flight Stacks.

2.2.1 Autopilots

Autopilots are not exactly a framework, but they are worth mentioning is this section. They are software that covers the some low level functionality of drones, such as motors control and flight stability. For that it may responsible for the communication and coordination of the different components and sensors of the drone. It can be compared to the firmware of the system. Therefore, it is designed to be used in combination with more high level software on an upper layer of abstraction.

The **ArduPilot** software platform [10] offers the user a wide range of possibilities and applications. It is fully open-source with a GPL licence, which makes it one of the main options for both DIY *("do it yourself")* and commercial drones. This project started with *Arduino* hardware as a base (hence the name), but nowadays there is a wide range of hardware compatible with it. It is also not tied to a single type of UAV, but can be run on different types of vehicles, such as **multirotors** or **fixed-wing drones**. In its list of features we can find support for both manual control with all the most common needs, as well as for autonomous missions.

Another well known autopilot is **PX4**. It is, according to *Dronecode* [11],their own developers:

"PX4 is an open source flight control software for drones and other unmanned vehicles."

It is also an open source project based on a modular architecture, which supports a number of different hardware. PX4 was initially developed to support autopilots that follow the **Pixhawk standard** for drones [12]. These boards take care of the stability and low-level functionality of the UAV. The sensors and peripherals are the additional hardware that can be connected to the drone to provide extended functionality, this can comprise between a camera or power devices.

In order to develop new applications, PX4 offers several **APIs** with official support. **MAVSDK** is their recommended framework due to its simplicity, but it is restricted to the use of MAVLink services. An API based on **ROS** or ROS 2 is also supported, which might be the standard framework for the development of open-source robotic applications nowadays.

Figure 2.4 Pixhawk 4 [11]

These autopilots work seamlessly in combination with many **Ground Control Station (GCS)** programs. Their main use is for wireless **telemetry** and flight status monitoring, but it is also possible to use them to generate control commands and modify the flight mission. This kind of programs is not unique for ArduPilot and PX4, but a general concept in these flight frameworks. These programs normally run on a ground computer, which also allows for more computational capacity and to run software that does not require real-time capabilities.

The **QGroundControl** is the ground control software developed by *Dronecode*, which is, of course, the recommended software to use together with the PX4 stack. It allows, not only all the features already mentioned in the sections above, but also the direct on air flashing of a wide range of devices. With ArduPilot, the main *GCS* used and even recommended in their own webpage is **Mission Planner**.

2.2.2 AeroStack

Aerostack [13] is a software framework for aerial robotic systems developed by the *Universidad Politécnica de Madrid*. It is focused on building autonomous aerial systems with a flexible architecture and it is not platform-dependent. The user is able to program new modules to integrate the desired functionality of additional sensors and hardware, but with the idea to **reuse** the existing algorithms already encapsulated in AeroStack, which is also the basic pillar behind the software it is based on, **ROS**.

Although the main computational load runs off-board the aerial systems, it is also possible to embed part of its operation in on-board computers. The architecture of the software can be seen in the Figure 2.5:

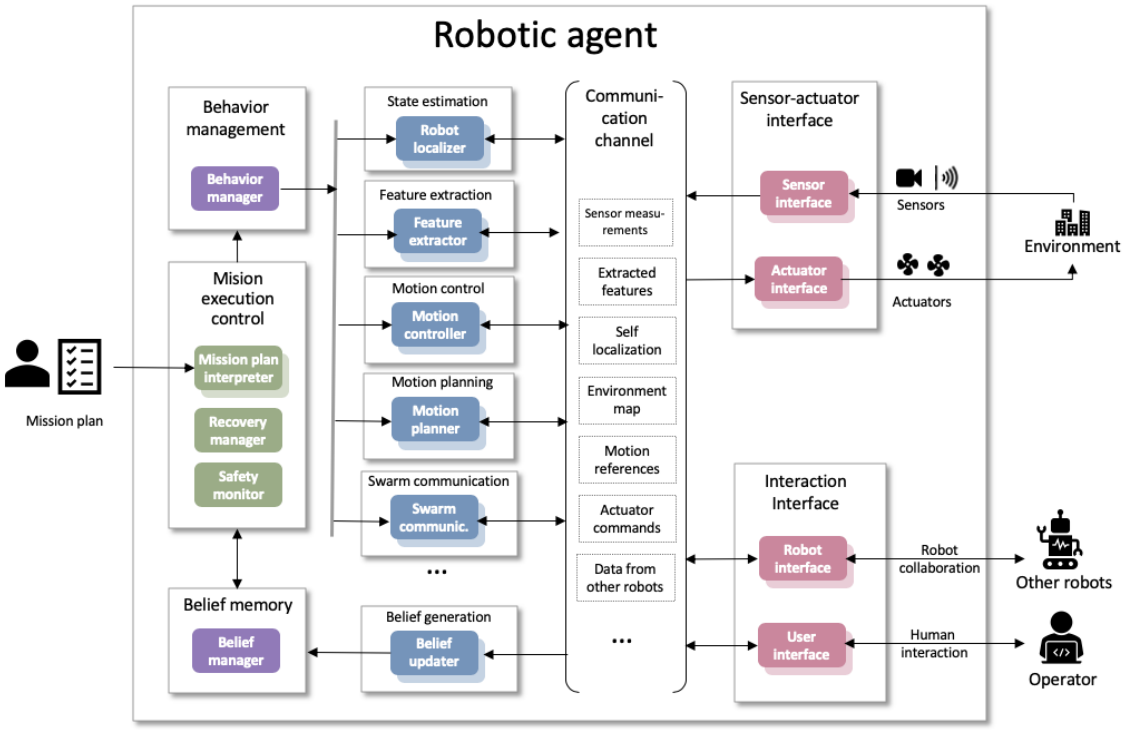


Figure 2.5 AeroStack architecture [13]

It is mainly based on four parts:

1. **Interfaces:** Communicate with the hardware to send commands and receive sensor measurements.

2. **Communication Channel:** Based on ROS communication, all processes share the message space so it is easier to expand and interoperate different modules.

3. **Behaviours:** This is the basis of Aerostack. These behaviours contain the main reusable logic block. They can implement controllers, computer vision algorithms, localization, etc.

4. **Mission control:** Executes the most high-level logic using mainly the python API it is incorporated or Behaviour Trees.

Currently, the second version of AeroStack is being developed, **AeroStack 2**. Just like its predecessor it is based on a modular ROS2 architecture. This is the framework that will be used in this thesis.

2.2.3 Crazyswarm

Crazyswarm [14] is a ROS-based swarming framework for the crazyflie drones, mentioned above in this chapter, and developed by a collaboration of *University of Southern California (USC)* and the *Technische Universität Berlin (TUB)*. It wraps all the main functionalities of the drones with a high-level user friendly API, which is intended to enable the user to focus on high-level functionality forgetting about the low-level programming SDK of the Crazyflie.

It is based on a **C++** API implementation for these drones, *crazyflie_cpp* library [15] plus other helper libraries to add additional functionality, such as external motion capture systems. The architecture of this software can be seen in the figure 2.6. Additionally it

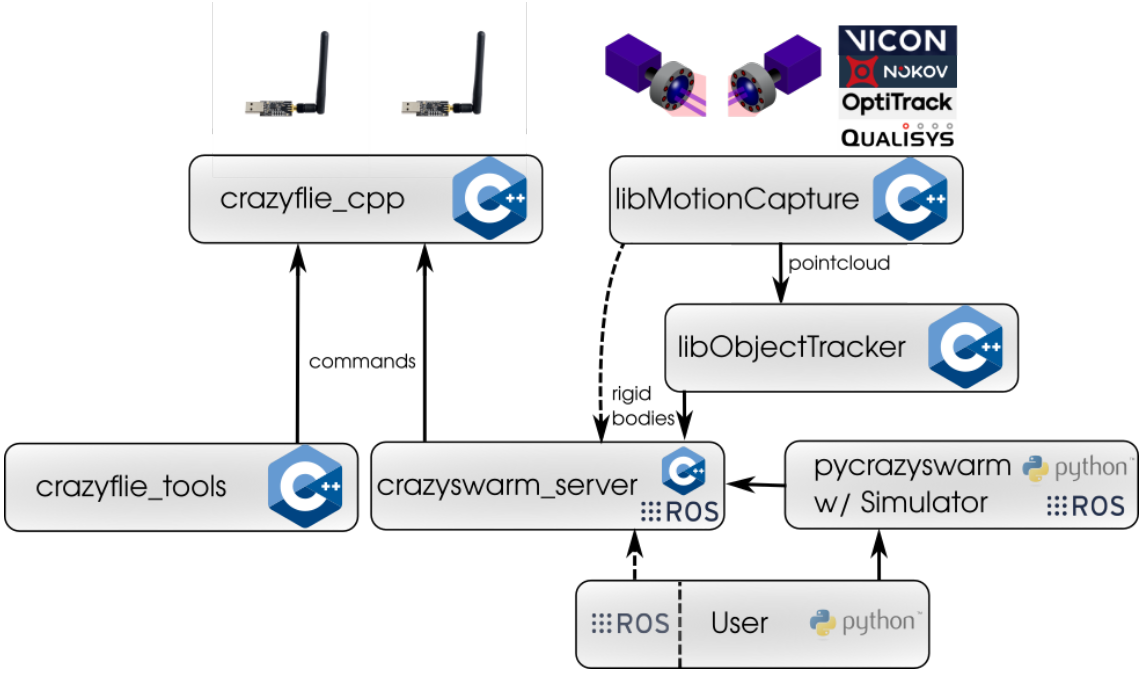


Figure 2.6 Crazyswarm

incorporates an external simulator to test the functionality before running it on the real hardware. All of these is conveniently wrapped up into a **Python interface** for the user, which also makes use of the ROS communication system to inter-operate with the rest of the architecture.

This framework focuses on the **swarming** capabilities of the crazyflies, easing the control and formation of several drones in a practical way. But it does not incorporate many high-level behaviours, as can be seen in the previous frameworks.

2.3 Pallet identification in logistics

Pallets are the main storage and transport unit for goods in logistics. Their use is extremely widespread due to their robustness and load capacity, as well as the ease of transport with the right equipment (forklift trucks for example). Although there is no single type of pallet, the most widespread in Europe is the wooden *Europallet* [16], which has a support surface of 1200x800 mm and a weight of 27kg. With these characteristics, they are capable of supporting loads of 1,500 kg in movement or up to 4,000 kg statically. In addition, it is necessary that they show distinguishing marks of origin and demarcation of Europallet characteristics.

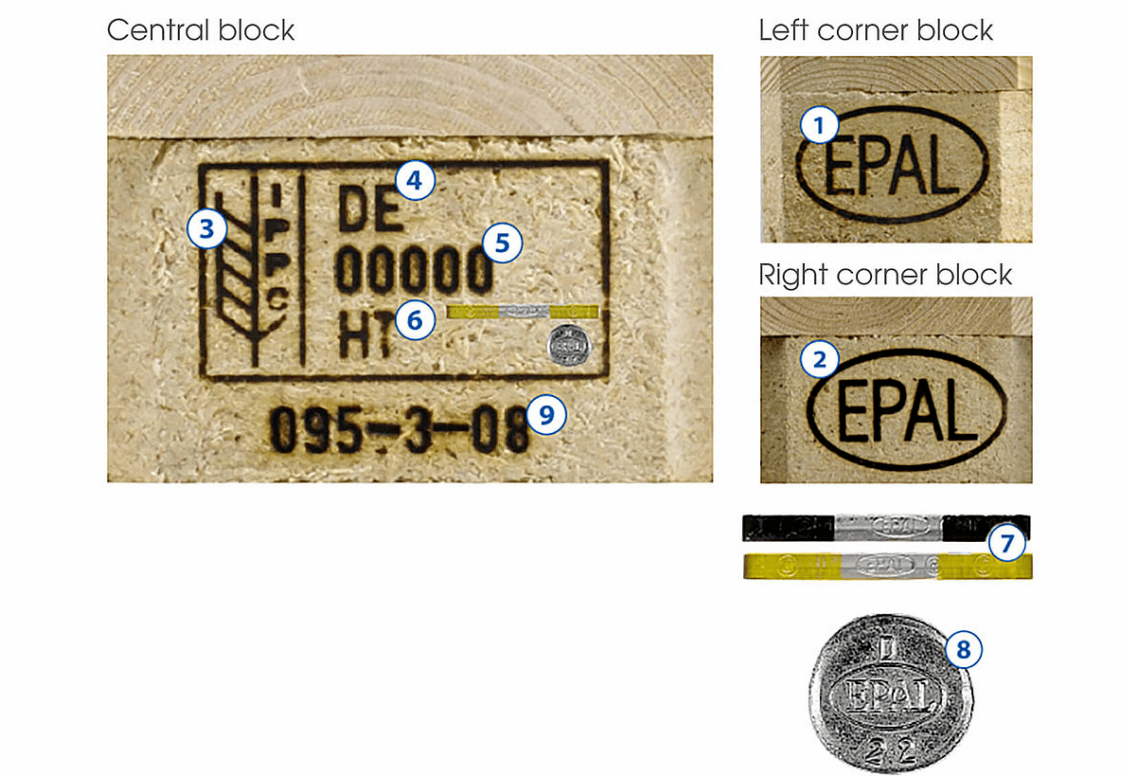


Figure 2.7 Europallet. More information about tags at [16]

However, not only pallets of this type can be found in industry. Depending on the type of load, other materials such as plastic or metal can be used to achieve different properties.

An essential task in logistics is to keep track of products. This can be a demanding job if the number of goods and products to manage is very high. Although pallet tracking

Figure 2.8 Pallet structure. logistiekonline.be

and idenfication systems are not wide spread, there have been some approaches to do this.

2.3.1 Optical

Barcodes are very familiar to everyone in shops and supermarkets. They are a robust and efficient method of product identification and also have a place in logistics and warehousing. They are based on one- or two-dimensional patterns that can be detected by optical methods, usually laser or camera, at a reduced cost.

This also has a major disadvantage, as a **direct line of sight** between the reader and the code is necessary for identification. This means having several codes per pallet if an automatic re-identification system needs to be set up, since in most cases it is not possible to ensure correct positioning of the pallets beforehand. In addition, having only one code can be a problem in the event that it is damaged. Pallets ID (identification) labels are available in different forms, shapes and colours, which aims to adapt to the necessities of each use case. Usually they contain a number which then has to be assigned to the adhered product via a warehouse management software. In 2D barcodes or **data matrices**, the information density is much bigger (being able to embed alphanumeric characters) and errors can be recovered even if the surface is up to 30% damaged. But these advantageous characteristics lead to a slower reading frequency and the need to use a different reader than the one-dimensional codes.

Another approach with this type of label is to print or **engrave** them directly onto the pallet surface. This eliminates the need for a constant supply of labels, but at the same time spoils the material itself. In addition, especially if the pallet is made of wood, this engraving can be easily erased due to rubbing or impacts, leaving the code unreadable.

(a) Barcode **(b)** 2D Barcode. Data Matrix

Figure 2.9 Optical label examples. barcode.tec-it.com

2.3.2 Radio Frequency Identification

Radio frequency identification or RFID has become very popular in the last 15 years. Instead of having an optical code to read, the id code is sent via radio frequency. This solves the obvious problem of barcode labels requiring direct line of sight to the reader. This allows reading even if the product is dirty or not perfectly positioned, which aids the automation of the task.

The RFID [17] tags have an antenna inside which sends the information to the reader. There are two main types of tags: passive and active.

1. **Passive:** They do not have a battery inside, which means that they do not have a power source from inside. Instead, they charge up from the electromagnetic field produced by the reader and then send their radio signal.

2. **Active:** They hold a battery inside, which lets them actuate faster without needing to charge before sending the information. They tend to be more expensive and have a limited lifespan, but they can also hold more information.

In addition, RFID tags can generally store more information than a barcode, which helps to improve product **traceability**. Although RFID tags cost more than optical codes, they can be reprogrammed to change the information they hold. This means they are much easier to reuse, but it also opens the door to security issues, as an external party may be able to modify the information. Additionally, multiple tags can be scanned at the same time and with a much greater speed, this makes the RFID technology the best option for fast paced or **automated systems**.

Chapter 3

Tools: Hardware and Software

In this chapter all tools used in the thesis will be briefly introduced.

3.1 Crazyflie

The *Crazyflie* [18] is a drone developed by *Bitcraze*, a company based in Sweden. Their software and hardware are completely open-sourced which enables the customisation of the drones and onboard firmware for special use-cases.

Figure 3.1 Crazyflie 2.1

The base drone has a take-off mass of only 27g with an onboard microcontroller, IMU and communication capabilities through the *Crazyradio*, a custom USB dongle with support for 2.4GHz radio communications based on its own protocol and up to 125 different

channels with 2Mbps, 1Mbps and 250Kps communication data-rate.

Figure 3.2 Crazyradio

A very important concept of the *Crazyflie* is the modularity. Therefore, there are multiple expansion decks available for the drone, adding new functionalities to the drone, such as SD Card logging, optical flow for state estimation and imaging with Wi-Fi connectivity.

3.1.1 Localisation

Localisation is, as mentioned in the previous Chapter, crucial for the precise control of the quadcopters. *Bitcraze* offers different modes of state estimation and localisation, just using the internal sensors and expansion decks, and using an external positioning systems. Internally the drone has the option to use an *Exteded Kalman Filter* (EKF), which fuses the information from different sensors and positioning systems. This modularity enables the customisation of the drone for specific use-cases. The external positioning systems supported are:

1. **Motion Capture:** The usage of commercial motion capture systems (mocap) such as VICON or OptiTrack is supported. These are usually based in passive markers which are recognized by an external system and subsequently multi-perspective algorithms for pose estimation (like PnP) are used.

2. **LOCO Positioning:** This system was developed by *Bitcraze* and it is based on Ultra Wide Band radio. It enables the drone to estimate the pose onboard with the aid of external *anchors* (stationary) which send messages to the *tags* (fixed to the object to track). The estimation is based on the time and latency (*Differential Time*

of Arrival (DToA)) of exchange of these messages. Therefore an expasion deck with the necessary hardware is needed. According to their own documentation [19]: *"It is in many ways similar to a miniature GPS system."*

3. **Lighthouse Positioning:** Based on the optical beacons of *SteamVR*, it enables the drone to localize itself onboard. These send laser beams through the room, which are then received by the drones. Therefore, a direct line of view between the beacons and the drones is needed, as well as light diodes to detect the light of the beams onboard.

In this thesis, the main positioning system used is external with a motion capture system.

3.1.2 CFClient & APIs

In order to test the functionalities of the drone, log variables and configure parameters, there is an already developed desktop application called *cfclient*.

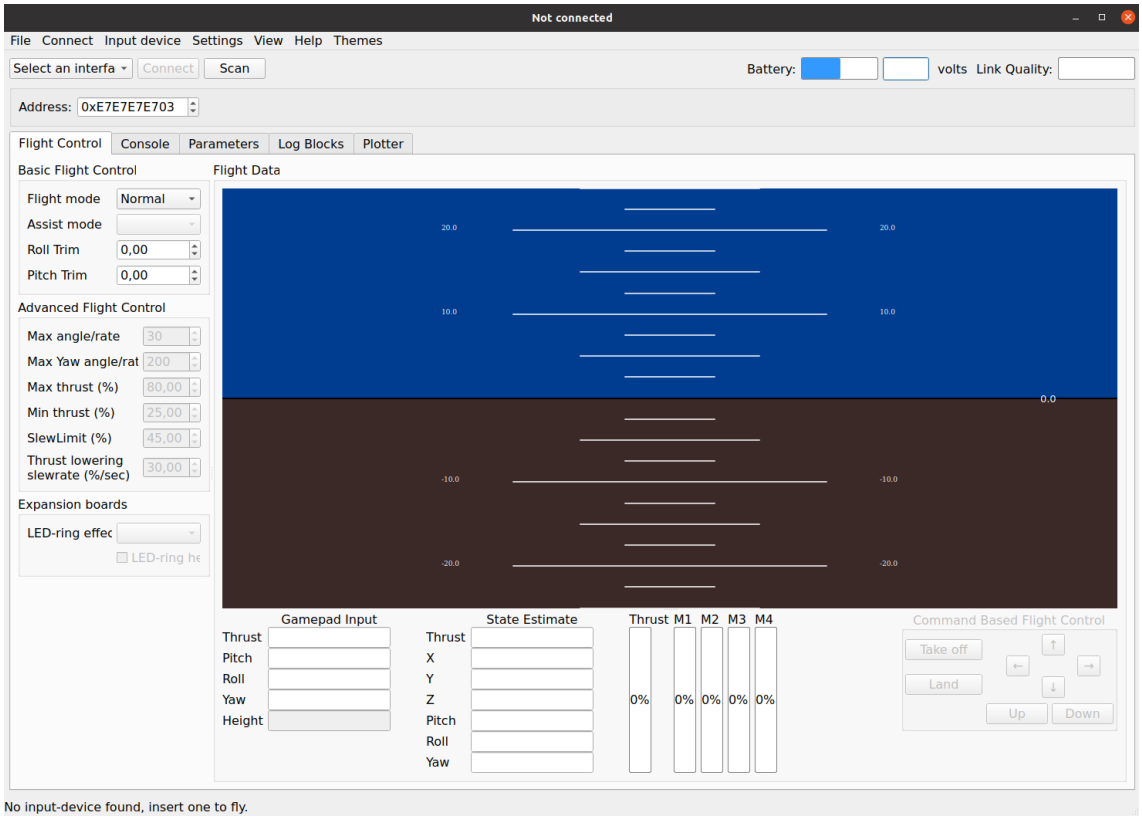


Figure 3.3 CFClient

This tools also enables the user to flash the drone with the official firmware and fly it manually with a controller (for example a XBOX controller). It is intended to be used as a debug tool for the drone, since there are no high-level capabilities included in the

client. The drone firmware used in this thesis is the version from May 2022, which can be downloaded automatically from the cfclient choosing the realese *2022.05 - firmware-cf2-2022.05.zip*.

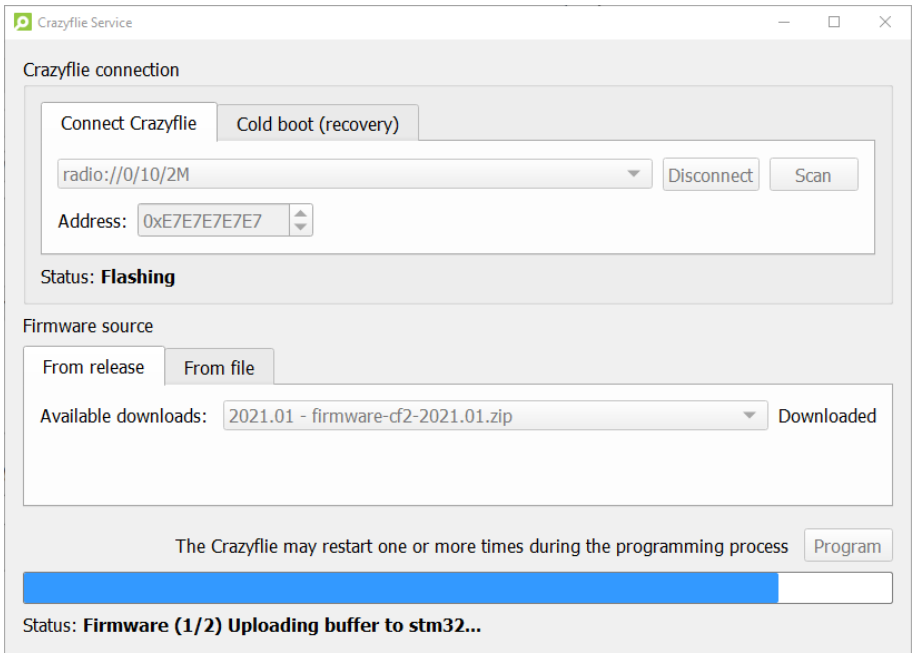


Figure 3.4 Firmware upgrade in cfclient [20].

If the user wants to create a custom application, there are two main options to communicate with the drones via the *crazyradio*.

1. **Python [21]:** *crazyflie-lib-python* incorporates the communication API with the drone as well as some swarming capabilities. It is actively updated at the time of writing this thesis.

2. **C++ [15]:** The older *crazyflie_cpp* repository contains the basic functionality to control the drones. Even though it is not as advanced and well documented as the python API, it is adequate to use if the user needs to integrate it with a bigger C++ software. This is the case of this thesis, and therefore it will be used.

It is important to emphasise that these libraries are not the drone's firmware, but tools for communication with the UAV. The firmware used is the official one provided by Bitcraze, whose functionalities can be configured with parameters. Some of these parameters will be tweaked in order to adapt the drone to the test environment.

3.1.3 AIDeck

The AIDeck [22] is the expansion board with **Wifi** connectivity and a built-in **camera** (RGB or greyscale). This expansion module can be used to provide the drone with the ability to stream images to an external system.

This board also incorporates a *GAP8 processor* [23] that extends the on-board computational capacity and allows the implementation of small image analysis and recognition programs. However, for the purposes of this thesis, this additional computational capacity is not sufficient, and will be used just for **image streaming.**

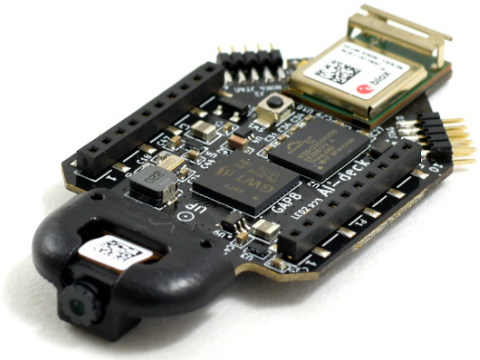


Figure 3.5 AIDeck

Even though there is no official information, the maximum framerate achieved by the streaming camera is around **12-15 FPS** (frames per second), and it is quite dependent on the quality of the wifi network. This is created via an *ESP32* chip also mounted on the expansion board. However, the images have to pass through the GAP8 first, so if image processing is done on board, it is also possible to drop below the above mentioned FPS. This expansion plate, like the others available, is easy to install. It is simply attached to the main body of the drone using the pin slots on the module.

3.2 Vicon

As mentioned in the previous section, the localization method used in this thesis is the external motion capture system. Specifically, a system from *Vicon* [24] will be used. Since localisation is not a focus of this work, the functionality and types of motion capture systems will not be discussed in depth.

Vicon is a well-known company for **mocap** systems. It is based on the acquisition of data by a multitude of cameras for a subsequent joint estimation of the pose of pre-introduced objects. These objects are marked with spherical markers reflecting infrared radiation. It is important that the objects registered in the system have distinctive marker configurations, otherwise estimation errors and even confusion between different objects may occur.

In the research hall of the *FLW*, a total of **52 cameras** are available. Most of them are the *Vero* model, but there are also some *Vantage* cameras. The user interfaces used are the Vicon *Nexus* and *Tracker* programs, which are simply used to add objects into

the tracking system and to visualize the whole system. Subsequently, it is possible to extract the estimated pose information directly from the server created by Vicon, it is only necessary to be on the same local network. No additional configuration or calibration was necessary for this thesis.

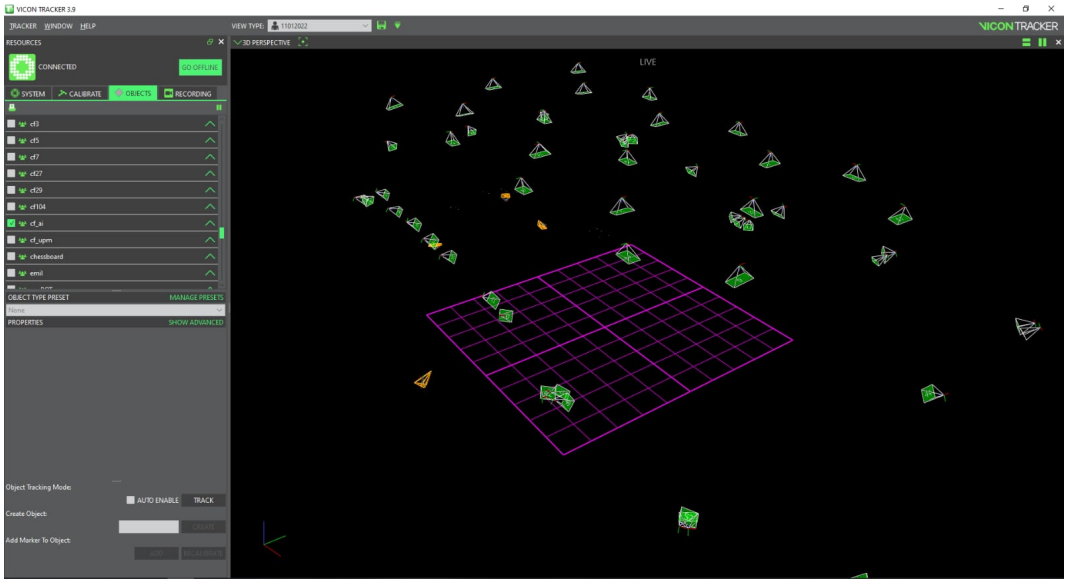


Figure 3.6 Vicon Tracker screenshot

3.3 Aerostack 2

AeroStack 2 [25] is the successor to Aerostack, mentioned on the previous section. Currently the version 2 is being developed at the *(UPM)* by the *Computer Vision and Aerial Robotics (CVAR)* group of the *Centre for Automation and Robotics (CAR)*.

It is based on **ROS 2** and focuses on modularity. Not only it aims to build an abstraction layer for different drone platforms, but to serve as a complete framework for the state estimation, control, navigation, etc. of UAVs. Its architecture could be divided into the following main modules:

1. **Platform**: Communicates the drone with Aerostack 2. To do so, it makes use of the libraries and communication methods specific to each specific hardware. It serves as an abstraction layer and hardware interface for the rest of the software.

2. **State estimator**: Fusion of the different sensors to obtain the state of the robot.

3. **Controller**: In charge of sending control signals to the platform depending on the given objective. It is a control loop external to the drone, the autopilot is in charge of the internal loop.

4. **Behaviour Handler**: Manages the different behaviours implemented for the platform (take-off, landing, trajectory tracking, etc). On the one hand it communicates with the controller, sending new targets, and on the other hand it communicates with the user interface. These behaviours are high-level abstractions and behaviours.

5. **User interface**: Layer of abstraction for the user. In charge of the higher level commands, calling the behaviours integrated in the previous block. Currently a Python and Behaviour Trees interface is supported.

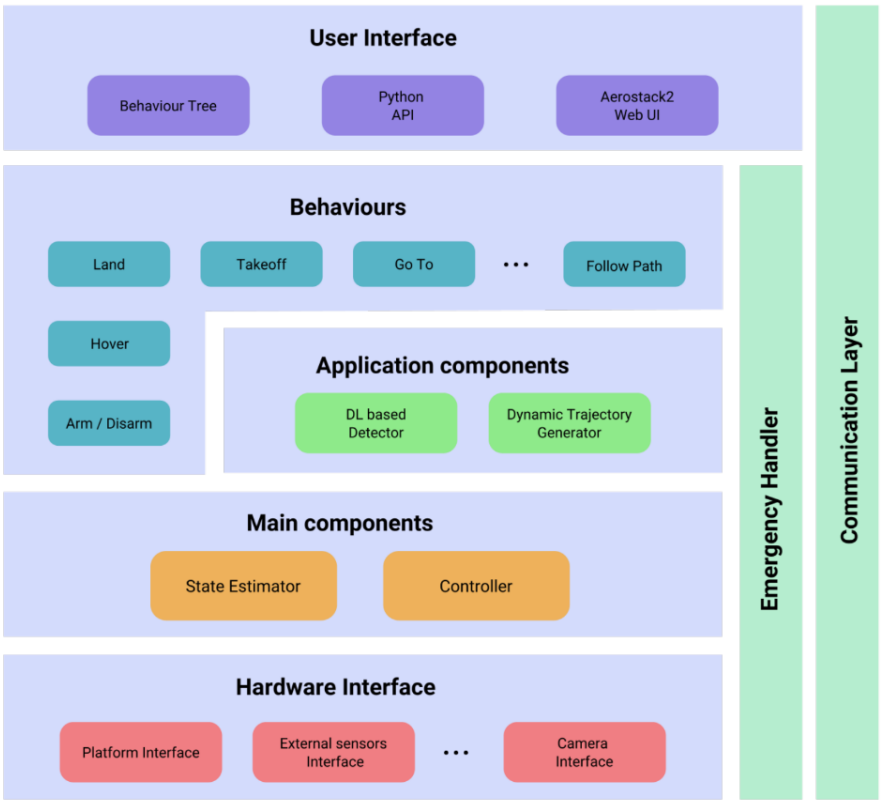


Figure 3.7 AeroStack 2 concept architecture.

With these main components, the user should be able to create custom applications using the existing modules, or even creating new ones with the provided architecture.

In addition, it is possible to use simulators to support the development of new features. It is simply necessary to create a fake hardware interface that communicates with the simulator as if it were a real drone. This can be very useful especially for testing the performance of new high-level behaviours.

The main simulator supported is Ignition Gazebo [26]. Unfortunately there is no realistic and functional model of the Crazyflie drone for this simulator, so it is not possible to test specific behaviours or tune up controllers with it.

The main version used in this thesis is the v0.2.0 Beta.

3.4 Pallet identification

As mentioned in the Introduction, the use of artificial intelligence and machine learning methods is very interesting for pallet recognition as it eliminates the need for external tags and additional infrastructure.

In [27] a new model for person identification, also called *person retrieval* or person *re-identification*, called *Part-based Convolutional Baseline (PCB)* is presented. From the data of a user-defined convolutional model (*backbone network*), local features are extracted from different parts of the image, *part-informed features*, and subsequently a feature vector that can be used for classification, resulting in an identification of the person present in the image. The results are quite favourable and prove to be a competitor to state-of-the-art techniques in this field.

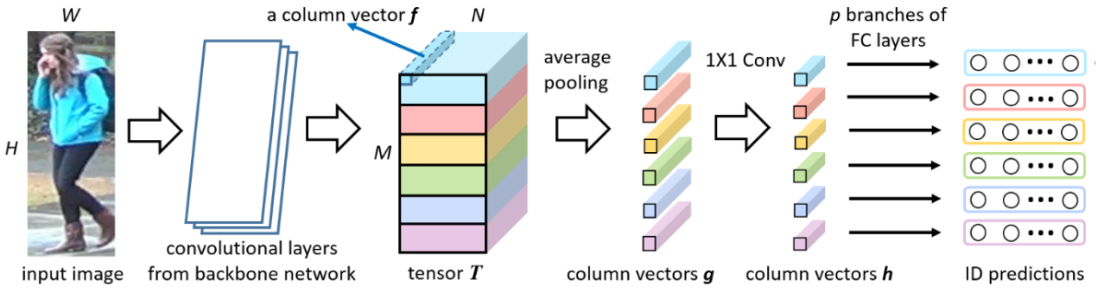


Figure 3.8 Part-based Convolutional Baseline Architecture. [27]

One of the main problems with deep learning models is the large amount of data needed to train the entire network. For this purpose, the *FLW* in collaboration with the *Fraunhofer Institute for Material Flow and Logistics* has created a **dataset** [28], [29] containing 10 images of 502 pallets, with a total of **5020 images**. Furthermore, they evaluate the quality of the generated dataset with the help of the architecture mentioned in the previous paragraph. More specifically, they use the **PCB_P4** model with the **ResNet50** backbone [30] for feature extraction, to which a **Yolov2** network [31] is added to detect the region of interest (ROI) in the image and perform acrop of the pallet block. This Yolov2 also adds the possibility of using the model on a video stream and not on images, as this added network will act as a pre-processor for the input of the PCB model. An example two dimensional representation of two of the features vectors' characteristics can be seen in Figure 3.9, where the features of multiple pictures from a pallet block are visualized. It can be seen that similar pictures, generate similar feature vectors and can be therefore classified. The results on the dataset data are quite favourable, reaching around 96% in both *mean average precision* and *rank-1-accuracy*.

The implementation of this particular architecture is done with the aid of *Torchreid* [32], a *Pytorch* based framework for person re-identification.

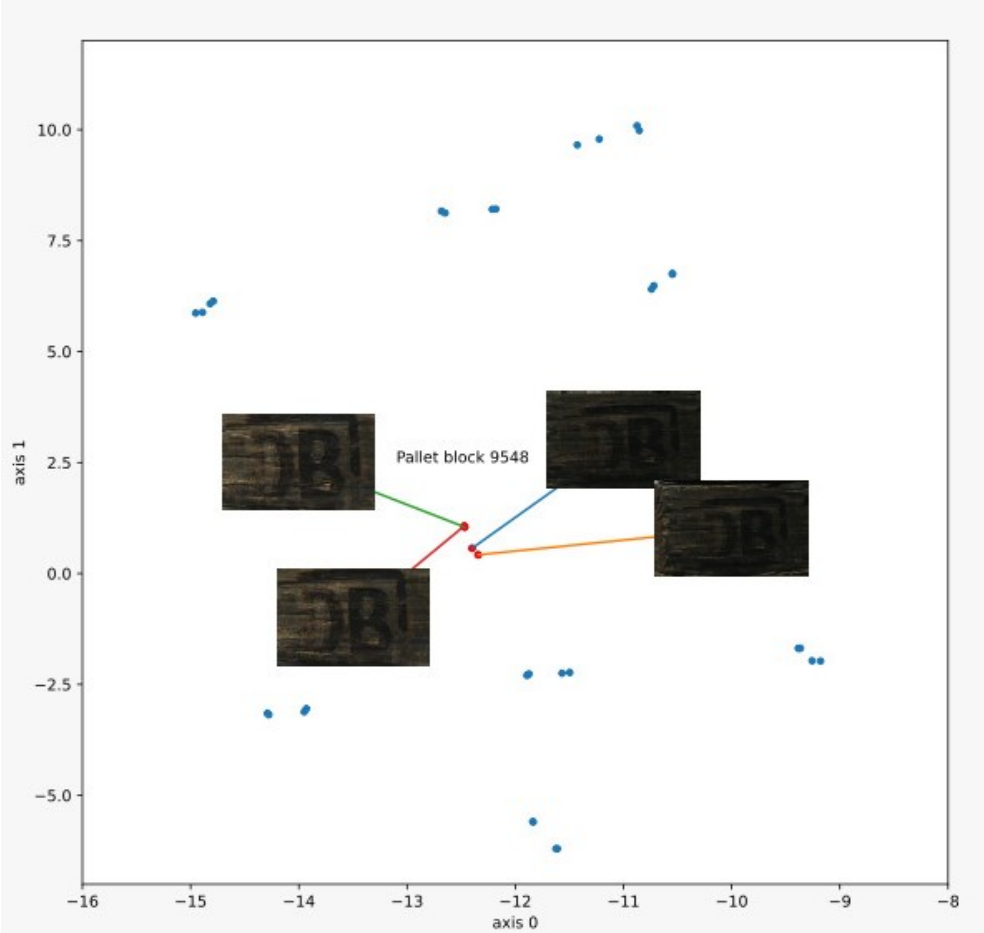


Figure 3.9 Pallet feature vector.

3.5 Work environment

In addition to the tools already mentioned, there are others that have also been actively employed in the development of the thesis and deserve to be mentioned. All software is developed under the Linux-based operating system, **Ubuntu 20.04 LTS**, as this is the recommended version for development in ROS 2. The main programming languages used are:

- **Python 3.9**: Secondary packages, image processing modules and user interface.
- **C++**: Hardware platform interface in Aerostack 2.

3.5.1 ROS 2 - Galactic

The version of ROS used in this work is *ROS 2 Galactic* [33], as it is the latest stable version of this software, with an End-of-Life (EOL) date in November 2022. However, the developed software is also compatible with the latest *ROS 2 Humble* distribution.

Distribution Name	Release Date	EOL Date
Humble Hawksbill	May 23rd, 2022	May 2027
Galatic Geochelone	May 23rd, 2021	November 2022
Foxy Fitzroy	June 5th, 2020	May 2023

Table 3.1 Latest ROS 2 Distributions release and EOL dates.

Although the work done is not completely based on ROS, it is important to be clear about the main concepts of ROS-based architectures, as AeroStack2 employs it. And to maintain the same modular philosophy, the modules developed must also adhere to this architecture. The design of the ROS 2 architecture, as in ROS 1, is based on graphs. The three main elements of these are:

- **Node:** The main entity in the ROS graph. These nodes can communicate with other nodes by sending messages through certain topics. They are also configurable by means of parameters. These nodes do not necessarily have to be on the same physical machine, as the ROS network can be extended across a network of computers.
- **Topic:** Main concept to communicate different Nodes. A topic can be published by multiple nodes, meaning that all these nodes can send messages to this topic. It can also be subscribed by several nodes, so when a message of this topic is sent, all subscribed nodes will receive them.
- **Message:** Data that is actually being transported between nodes.

The basic scheme of node - topic - message operation can be seen in Figure 3.10.

Apart from these, there are some more complex structures like services or actions. The latter will also be used in this thesis. Actions are ROS architectures intended for long running tasks. In them, one node (client) asks another node (server) to execute a task. Meanwhile the task is being executed, the client node also receives feedback of the execution status from the server node. They have three main parts, as it can be seen in the Figure 3.11:

1. Request for the service.
2. Execution and feedback.
3. Result.

ROS 2 also incorporates the *Colcon* compilation system [35], which will be used in packages developed in this language.

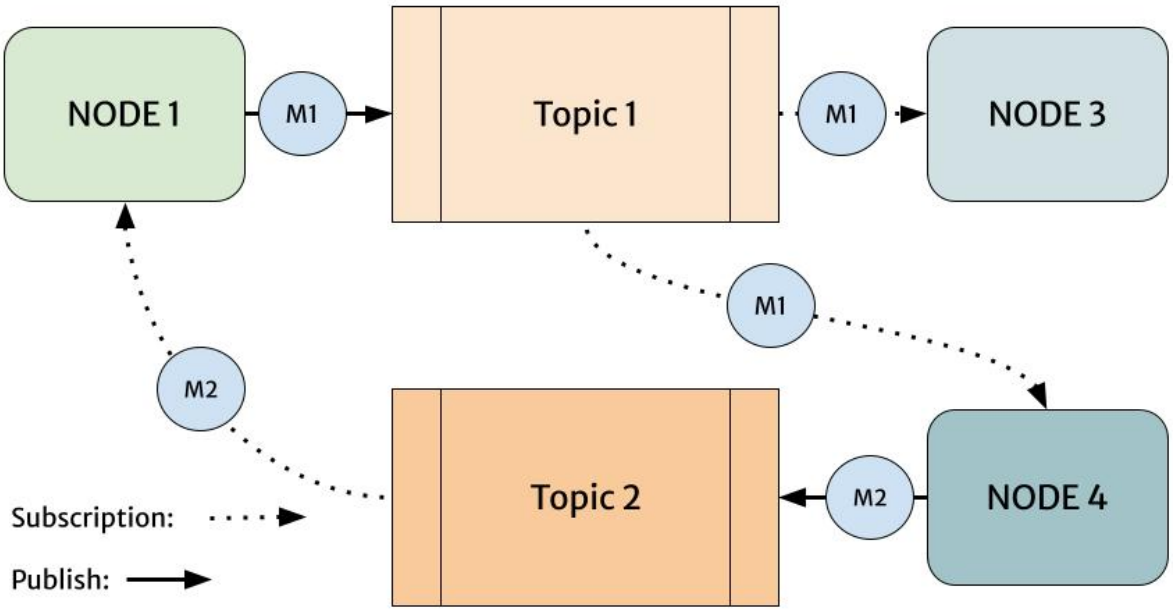


Figure 3.10 ROS node - topic - message concept.

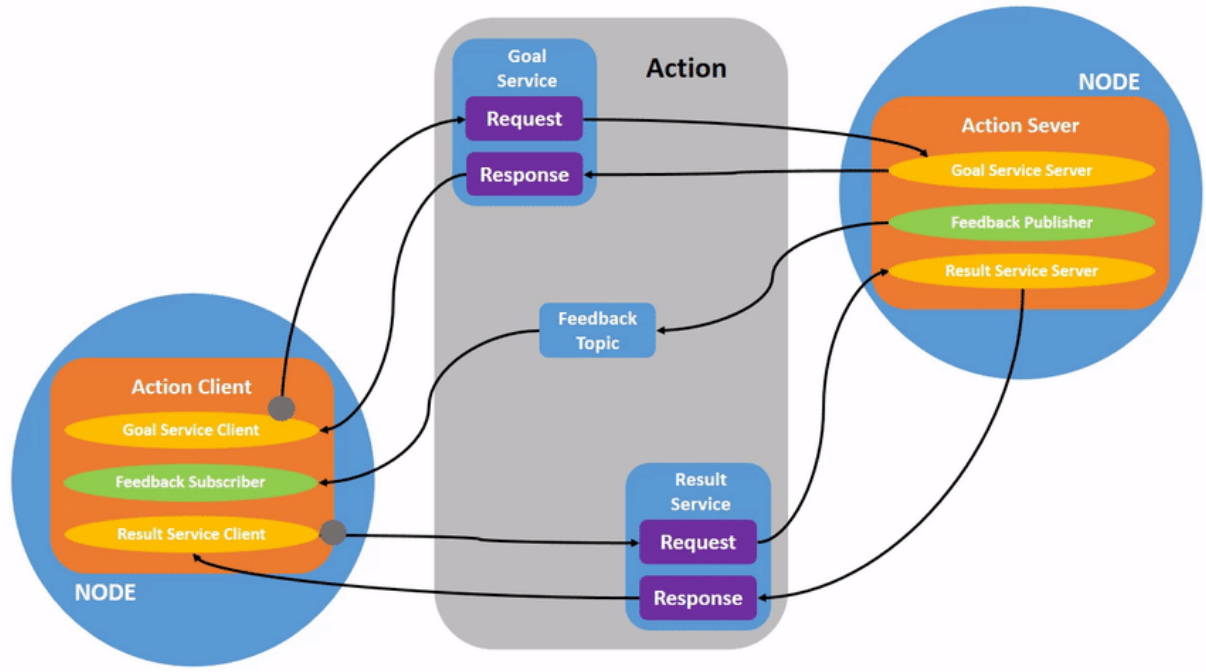


Figure 3.11 ROS 2 Actions arquitecture [34].

Chapter 4

Development

This section will show the development process of the final result of the thesis, divided into different sub-modules and their subsequent integration. The general architecture of the developed work can be seen in Figure 4.1.

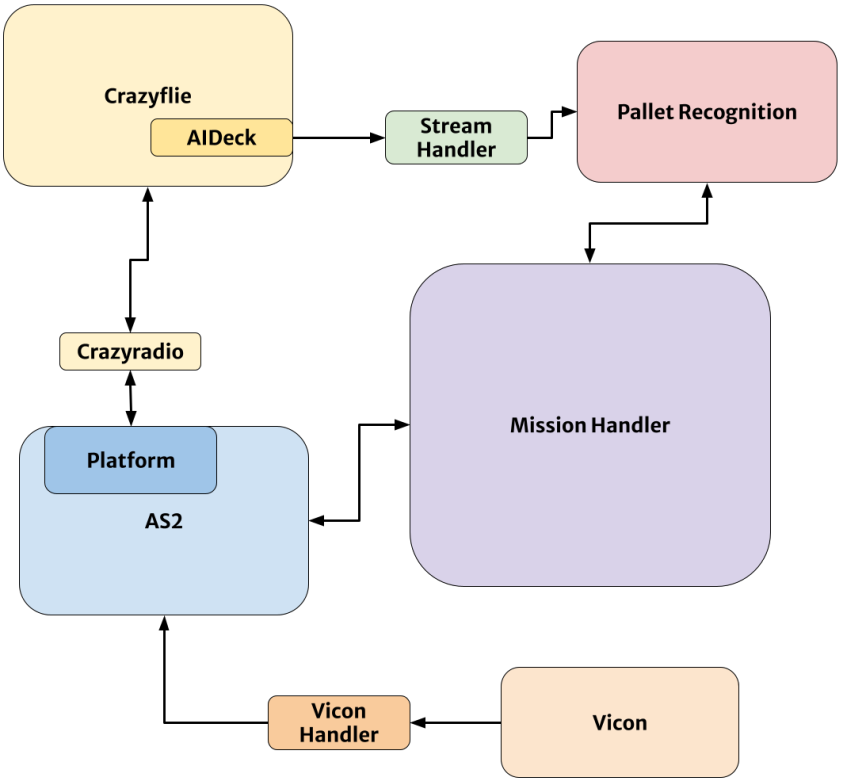


Figure 4.1 Thesis architecture

- **Aerostack 2:** Contains the Crazyflie platform, it is in charge of high-level control and behaviours. It communicates with the crazyflie drone through the platform and

this in turn through the crazyradio, connected by USB.

- **Crazyflie:** Hardware platform and firmware from the manufacturer. The AI Deck module is connected to the drone, but has a separate communication channel for the image stream.
- **Stream Handler:** In charge of creating the socket connection with the AI Deck via wifi and the pre-processing of the images necessary to transmit them to the pallet recognition AI.
- **Pallet Recognition:** Image processing for pallet recognition.
- **Mission Handler:** Global management of the mission with Python interfaces.
- **Vicon:** Motion capture system.
- **Vicon Handler:** Connects mocap estimates to Aerostack 2 via ROS 2 messages.

4.1 Crazyflie Platform

As mentioned in the previous chapter, an essential component in AeroStack 2 is the platform or hardware interface, which serves as a communication between the hardware and the rest of the Aerostack 2 software [25].

Aerostack 2 provides a platform template, called *AerialPlatform*, that must be completed with the peculiarities of the specific UAV, in this case the Crazyflie. It is also in charge of all the internal management of the platform in AS2, such as the drone's state machine. This class is itself derived from a ROS 2 Node, so this software will run independently from the rest.

Therefore, within the platform there are two distinct parts, the template given by Aerostack and the crazyflie specific implementation. The latter has to take care of configuring the drone sensors, logging the corresponding variables and parameters, as well as sending the control commands received from the Aerostack control module, for which, the crazyflie_cpp [15] library is used.

A simplified version of the functioning of hardware interfaces within Aerostack 2 can be seen in the Figure 4.2.

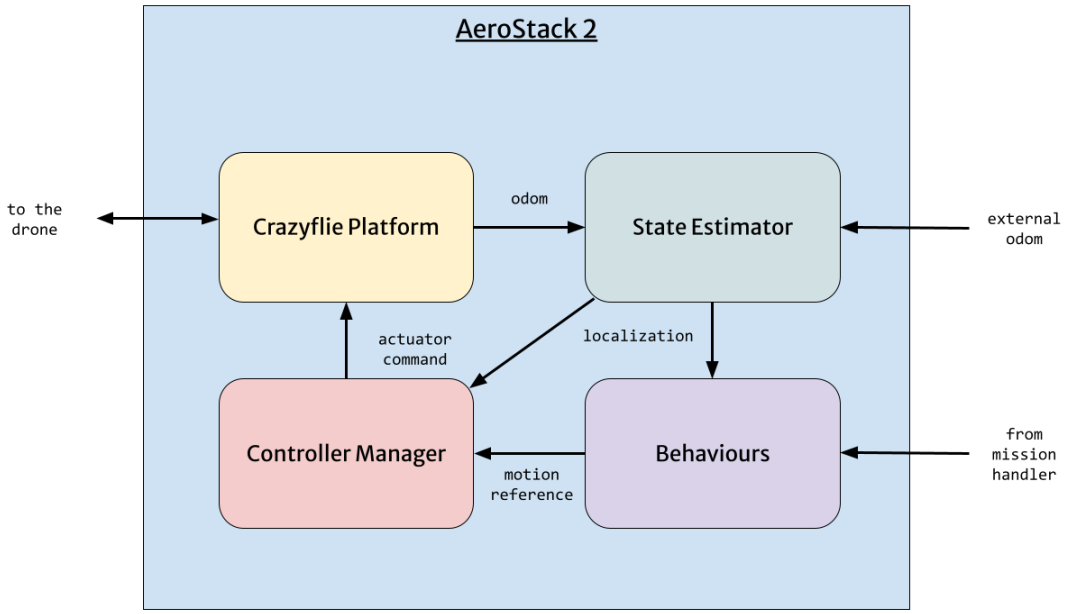


Figure 4.2 Crazyflie Platform within Aerostack 2. Simplified.

4.1.1 Initialisation and Logging

In the initialisation of the platform, the structures necessary to log variables from the drone and to connect to the UAV are set-up. This connection is programmed in a blocking way, meaning that the initialization will not continue and be completed, until a successful connection to the drone is done.

In order to connect to a Crazyflie via Crazyradio, the radio address of the drone, or *URI*, is necessary. This is given in the form *radio://0/82/2M/E7E7E7E703*. Here we can see four different components after the *radio://* mark separated by the / symbol:

1. Always **0**.
2. Radio **Channel**.
3. **Frequency**. It can be configured to be 250 Kbps (250K), 1 Mbps (1M) or 2 Mbps (2M).
4. Actual **address** of the drone.

Once the connection is established, a periodical connection check will be run, in order to detect unwanted disconnections. If that is the case, the same blocking process will be executed.

If the connection is done correctly, it is possible to start logging data from the drone. Crazyflie's variable nomenclature follows the form *group.name*. The variables available for logging are quite extensive [36], but only the following will be considered:

- **Odometry:** The internal state estimate, both pose and velocity. Present in the group *stateEstimate*.
- **IMU values:** Although not currently implemented, Aerostack 2 contemplates the possibility of performing low-level state estimation off-board. Groups *gyro* and *acc*.
- **Battery:** Current battery voltage level. Group *pm*.

The way to do this is through *LogBlocks* (logging blocks). These structures allow logging up to 26 bytes of information through the Crazyradio. Therefore, it is necessary to configure different *LogBlocks* to log all the desired variables. More specifically, 4 *LogBlocks* are used; 2 for odometry, 1 for IMU and 1 for battery.

It is also necessary to specify the maximum frequency of logging for these variables at the moment of starting the logBlock. Therefore, in order to fully configure and activate a logging block, two steps are necessary:

1. Instatiation of the LogBlock. Here the desired variables in the form *group.name* and the callback function are given.
2. Running the LogBlock. Here the desired logging frequency must be set.

An example of logBlock configuration for the IMU an be found in the code listing 1.

```
// ++ IMU ++

// Select the variables to log. "group.name"
std::vector<std::string> vars_imu = {"gyro.x", "gyro.y", "gyro.z",
                                     "acc.x", "acc.y", "acc.z"};

// Create the std::bind for the callback function. In this case "onLogIMU" function.
cb_imu_ = std::bind(&CrazyfliePlatform::onLogIMU, this,
                    std::placeholders::_1,
                    std::placeholders::_2,
                    std::placeholders::_3);

// Create the instance of the LogBlock
imu_logBlock_ = std::make_shared<LogBlockGeneric>(cf_.get(), vars_imu,
                                                  nullptr, cb_imu_);

// Run it with a logging period of 100ms
imu_logBlock_->start(100);
```

Listing 1 LogBlock configuration for the IMU

In order to integrate these sensors with the rest of the framework, Aerostack 2 provides a predefined sensor structure. These sensor can be created in the framework by the user,

which is meant to be done in the *configureSensors()* function. By doing this, Aerostack 2 can make use of the values of the sensor and make them availabe inside of Aerostack 2 for other applications, like state estimation. In this case, three sensors are created, coinciding with the logging groups afored mentioned; Odometry, IMU and Battery. When values associated with the *LogBlocks* are received, a callback function is executed. This is when the values are sent to the rest of the Aerostack 2 system via the registered sensor structures.

```
imu_sensor_ptr_ = std::make_unique<as2::sensors::Imu>("imu", this);
odom_estimate_ptr_ =
    std::make_unique<as2::sensors::Sensor<nav_msgs::msg::Odometry>>("odom", this);
```

Listing 2 Creation of three Aerostack 2 sensors. "imu" is predefined and "odom" is a custom type.

Another very important part of the initialisation is the configuration of the drone parameters [37]. These parameters are also organized in the form of *group.name* and can be easily configured on run-time. It is possible choose between different types of control implemented in the firmware, state estimation techniques, standard deviation of sensors and external estimations, etc. After experimenting, it was decided to leave the standard drone configuration with internal cascaded PID control [38] and Kalman estimation [39].

```
cf_->setParamByName<uint8_t>("stabilizer", "controller",
                             (uint8_t)(controller_type_));
```

Listing 3 Example of parameter configuration with the crazyflie_cpp's function *setParamByName*.

The initialisation of the crazyflie platform follows the structure in Figure 4.3.

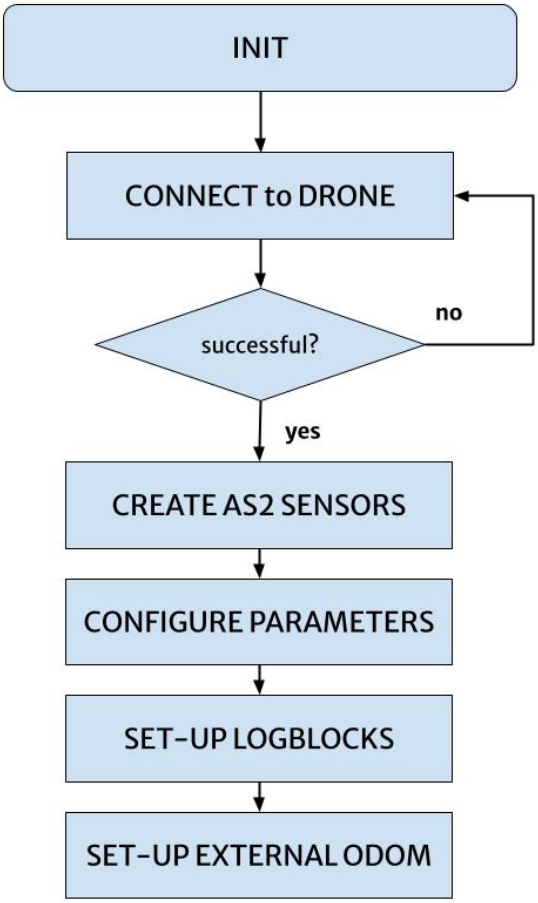


Figure 4.3 Flow diagram of the initialization of the Crazyflie Platform.

4.1.2 Drone Control

The platform receives control commands from the Aerostack 2 controller manager block, but it is responsible for sending these commands to the platform hardware. This is implemented in the AeroStack *AerialPlatform* function *ownSendCommand()*.

Depending on the supported control modes and the controller used in Aerostack 2, commands can be received in pose, twist (velocity) or thrust (low level command for motors). The available control modes have to be entered in a *.yaml* configuration file within the platform package. For crazyflie, the following control modes have been implemented, all in the ENU global reference system [40]:

- **Speed with yaw speed:** Twist commands in linear movement and rotation in yaw.
- **Speed in a plane:** Pose command for the height and twist command for planar movement and rotation in yaw.
- **Position in yaw angle:** Pose commands for position and orientation. This last

one is only compatible if using the onboard position controller, which will not be used in this thesis since it does not allow controlling the velocity of movement.

- **Unset:** This is the initial control mode, it should always be supported.

```
available_modes:
  - 0b00000000 # UNSET
  # - 0b00010000 # HOVER
  #- 0b00100100 # ACRO (p,q,r, Thrust)
  #- 0b00110001 # ATTITUDE with yaw ANGLE ( r,p,y , Thrust)
  # - 0b00110101 # ATTITUDE with yaw SPEED ( r,p, dy , Thrust)
  # - 0b01000000 # SPEED with yaw ANGLE in the LOCAL_FLU_FRAME
  #- 0b01000001 # SPEED with yaw ANGLE in the GLOBAL_ENU_FRAME
  # - 0b01000100 # SPEED with yaw SPEED in the LOCAL_FLU_FRAME
  - 0b01000101 # SPEED with yaw SPEED in the GLOBAL_ENU_FRAME
  # - 0b01010000 # SPEED_IN_A_PLANE with yaw ANGLE in the LOCAL_FLU_FRAME
  # - 0b01010001 # SPEED_IN_A_PLANE with yaw ANGLE in the GLOBAL_ENU_FRAME
  # - 0b01010100 # SPEED_IN_A_PLANE with yaw SPEED in the LOCAL_FLU_FRAME
  - 0b01010101 # SPEED_IN_A_PLANE with yaw SPEED in the GLOBAL_ENU_FRAME
  - 0b01100001 # POSITION with yaw ANGLE in the GLOBAL_ENU_FRAME
  # - 0b01100101 # POSITION with yaw SPEED in the GLOBAL_ENU_FRAME
  # - 0b01110001 # TRAJECTORY with yaw ANGLE in the GLOBAL_ENU_FRAME
  # - 0b01110101 # TRAJECTORY with yaw SPEED in the GLOBAL_ENU_FRAME
```

Listing 4 Control modes configuration file.

The controller manager can make use of plugins that implement different types of controllers. At the time of writing this thesis, only one speed controller [41] was available. This controller implements a PID in position and yaw orientation with output in velocity commands, so the drone will be using the *SPEED with yaw SPEED* mode, but the rest are still supported.

In addition to serving as an intermediary between the drone and the controller manager, the reference systems used in Aerostack 2 and internally in the hardware platform must be taken into account. As well as the units used in each. In this case the drone's local axes and directions of rotation are aligned and in accordance with the global system. Therefore, it is not necessary to apply any base transformation. However, a change of units does become necessary for orientation, as the crazyflie_cpp library is programmed to accept sexagesimal degrees, while Aerostack 2 calculates internally in radians.

The final reference system scheme can be seen in Figure 4.5. Earth refers to the global axes, this is common to all elements of the world. Map represents, as its name indicates, where the start of the UAV's working area is located, in this case it will be coincident with earth. Odom stands for the transformation given by the odometry of the drone, this is the one updated thanks to the Crazyflie logBlocks. The transformation between odom

and base_link indicates a possible discrepancy between the odometry pose and the current drone pose. In this case, the last two frames will also match.

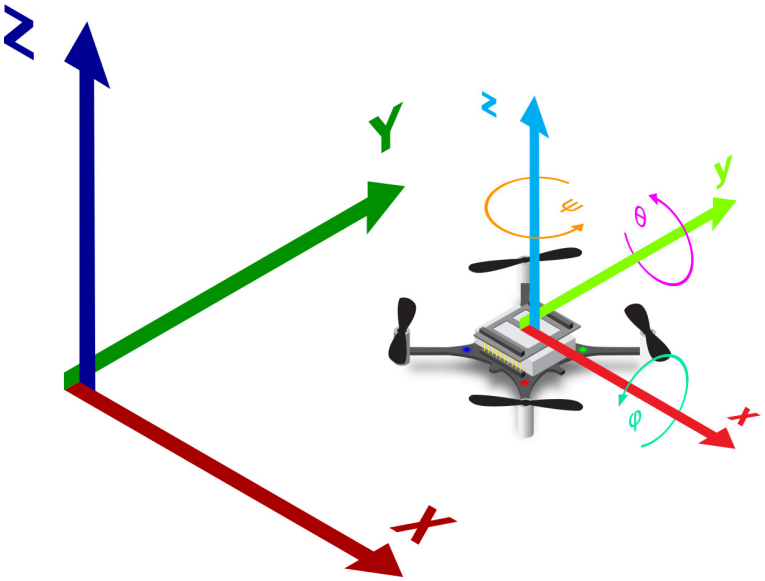


Figure 4.4 Reference frames of the Crazyflie. [18]

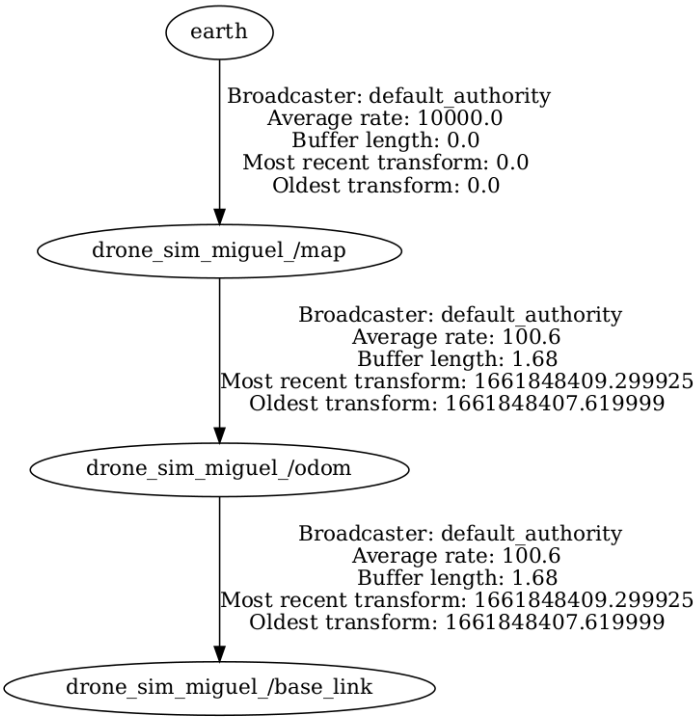


Figure 4.5 Transformations tree in Aerostack 2 for the Crazyflie.

4.1.3 External odometry

Both the hardware platform and the AeroStack's state estimator need the external odometry for proper operation. The Crazyflie can use this information to improve the estimation of the internal sensors that is used in the on-board controllers. In addition, Aerostack 2 uses this information to close the off-board control loops running on the ground base.

To extract the information from the Vicon server and make it compatible with the AeroStack ROS-based system, the Vicon Handler module is used. It uses the proprietary SDK of the mocap system to connect to its server and publish the estimates via topics in ROS. These topics are then readable by the Crazyflie Platform and the Aerostack 2 State Estimator. The platform is responsible for sending this information to the drone, in order to update the internal estimation.

Here it is also important to take into account the variance of the measurements of the *mocap* system in use, both in position and orientation. A very low variance leads to an excessive reliance on external measurements, which can destabilise the system as seen in the Figure 4.6. This can be easily configured through Crazyflie parameters from the group *locoSrv*, even allowing to change it while flying.

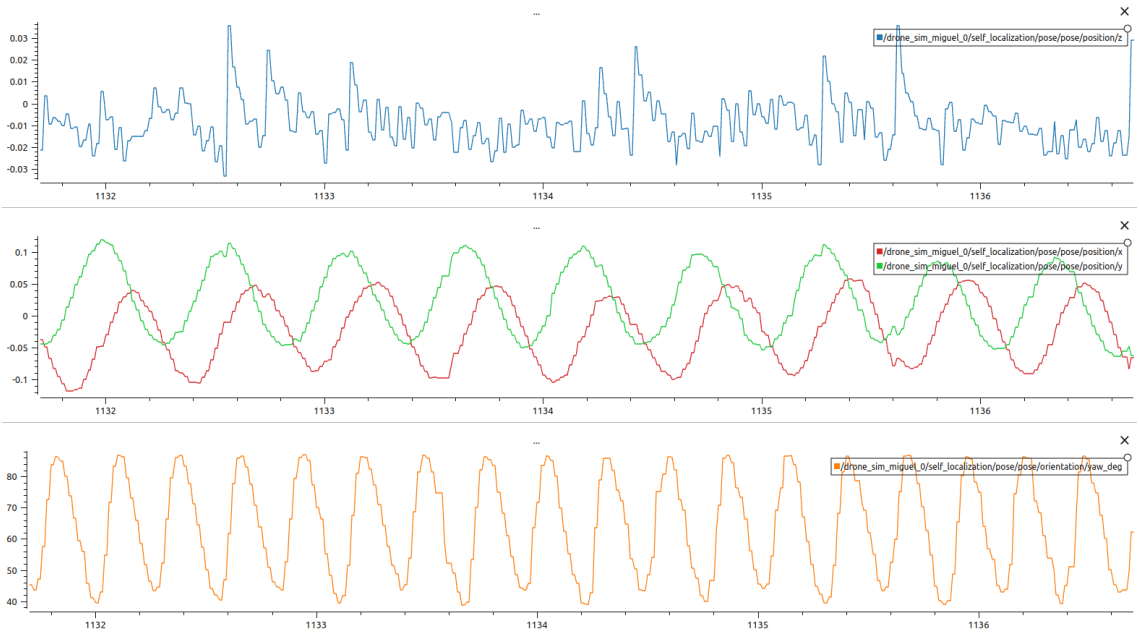


Figure 4.6 Instability caused by external odometry. Top to bottom: Height, position in horizontal plane, orientation in yaw.

Another factor to take into account is the markers on the drone. As explained in the Chapter 3, the Vicon system needs some markers forming a geometry to be recognized in the cameras. It is also important that these markers do not obstruct the air flow of the drone or create any disturbance on flight due to, for example, unbalanced weight. Although there is also an expansion deck availabe for the Crazyflie that makes it much

easier to adhere the markers to the drone, it was not used in this thesis. Instead of that, a custom solution was adopted. In the Figure 4.7 the final geometry adopted for the drone can be seen.

Figure 4.7 Marker geometry on the drone.

4.2 AI Deck and Image Processing

The AI Deck is, as mentioned in the Chapter 3, the expansion board for the Crazyflie, which provides Wifi connectivity with a camera module. In this section, we will look at how to configure the board for proper operation, as well as the ground-based system required to read images from the drone and use them for pallet recognition.

4.2.1 AI Deck Initial configuration

The AI Deck has two processors, a GAP 8 [23] and an ESP 32 [42]. The first one is in charge of image management and communication with the rest of the drone, while the second one is only in charge of Wifi connectivity.

The version of the expansion board used in this thesis is the C revision. With this version of the board it is not possible to perform updates and flash programs wirelessly by default. To solve this it is necessary to change the firmware of the GAP 8 controller on the board with a JTAG programmer [43]. To do this, a Docker image from the manufacturer is provided. This Docker image already has all the necessary dependencies installed for the native compilation of official Bitcraze software as well as the libraries to flash via cable with the JTAG. A more detailed explanation of all the necessary steps can be found at[43].

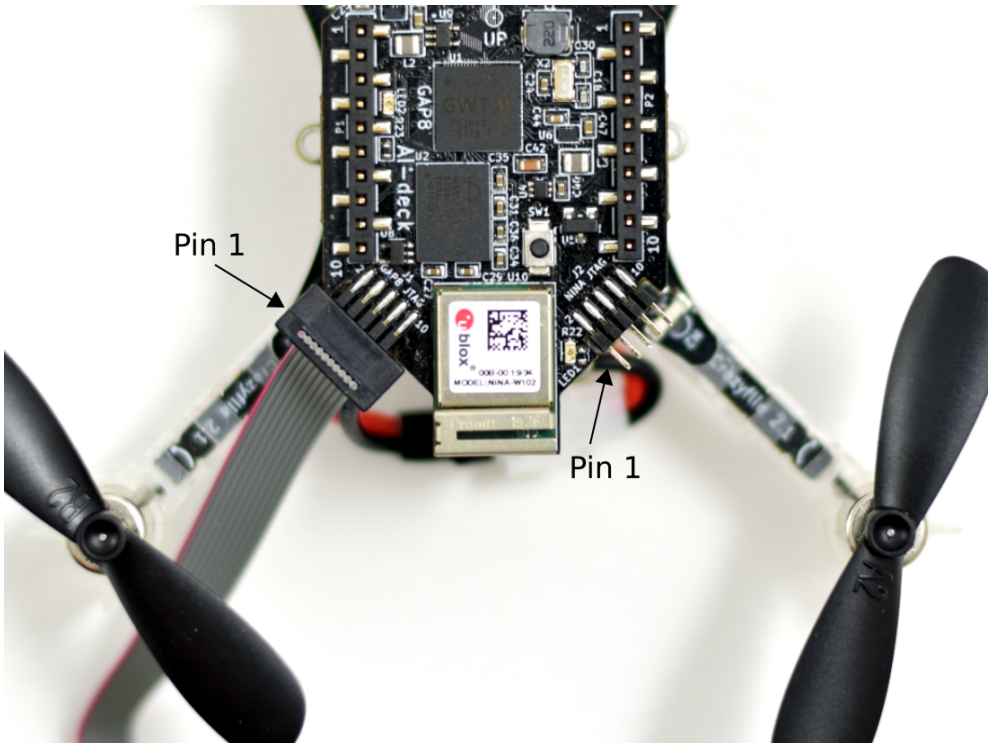


Figure 4.8 JTAG connection to the AI Deck. PIN 1 for GAP 8 and PIN 2 for ESP32.

Once the GAP 8 firmware has been updated, the drone should be able to be updated

and programmed wirelessly. However, at the time of writing this thesis, there is an incompatibility between the firmware of the crazyflie base drone and the AI Deck. This means that the expansion module cannot be configured correctly. As a solution to this we propose an update of the ESP 32 controller firmware [44], this time no JTAG is needed. To do this, a newer version of the firmware must be downloaded and compiled from the official repository [45] with the Docker image. Consequently it has to be flashed onto the drone, which can be done with the help of a tool programmed by Bitcraze called cfloader [46], which allows the flashing of new firmware wirelessly.

4.2.2 Wifi Connection configuration

Before using the expansion module to transmit images, it is necessary to configure the Wifi network connection. The board supports both the creation of its own access point to which external equipment can be connected, as well as the connection to an existing network. This configuration must be done in the firmware, so it is not possible to have a dynamic configuration. For this purpose, a tool in the crazyflie base firmware called KBuild [47] is used. This allows the configuration of the software build in a graphical way.

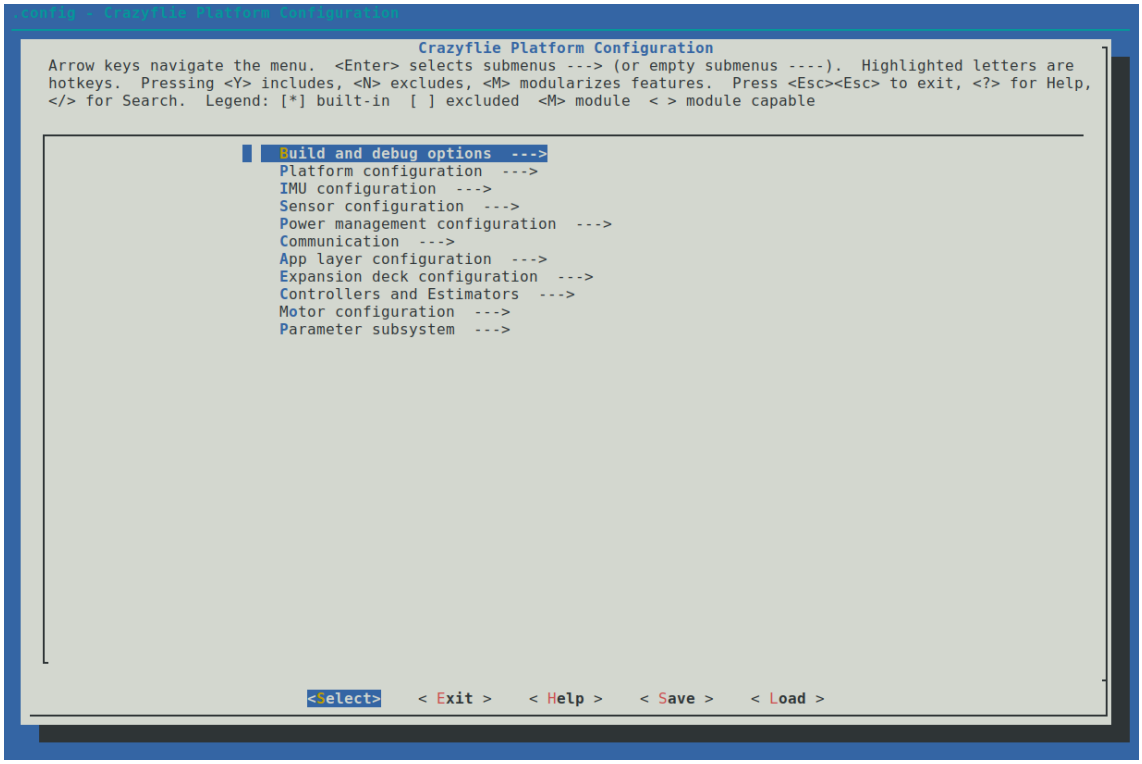


Figure 4.9 KBuild's main menu.

Within the menu of this tool, it is possible to configure different aspects of the firmware. In this case, an expansion deck, the AI Deck, has to be configured. Within this submenu are the configurations of different expansion modules. To change the Wifi connectivity mode, alter the Wifi setup at startup, selecting Connec to Wifi network. Subsequently, the corresponding credentials of the network to be used must be entered.

Once this is done, just compile the firmware in the same way as for the ESP32 and flash it into the drone.

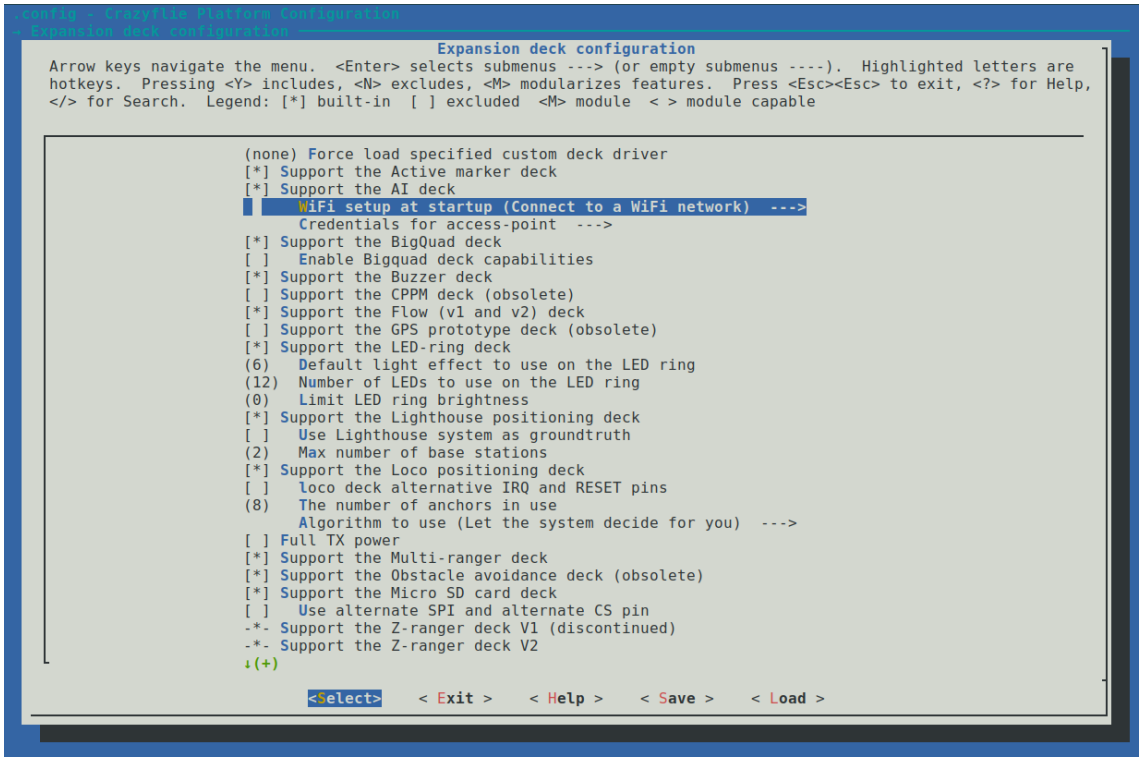


Figure 4.10 KBuild's Expansion Decks menu.

4.2.3 Wifi Streaming

Once the drone and AI Deck are fully configured, a custom program can be uploaded into the GAP 8 memory to be executed when launched. Fortunately, the manufacturer has an example program available that is very similar to what is required in this thesis [48]. The wifi-streamer is a simple software that allows the configuration of the camera and connectivity of the AI Deck and its use for live streaming. This program first takes care of setting up the Wifi connection according to the configuration indicated in the firmware and then takes an image and broadcasts it via Wifi.

However, it is necessary to modify this example slightly to provide it with automatic brightness adjustment capability. Otherwise, it is necessary to reboot the drone so that the camera adjusts to the brightness of the environment. Although no additional information has been provided by the manufacturer, it appears to be a problem with the camera hardware itself. The camera sensor has a resolution of 324x324 pixels, but the images taken for relaying 324x244 pixels to achieve better module performance. The proposed solution is to take a 324x324 image before the 324x244 image to be retransmitted to force the camera to clear the internal buffer. However, if this is done for every image, the final performance of the module would be much worse, so it is proposed to do it only once every fifteen times. This would allow a brightness adjustment of approximately once a second, without damaging the performance of the retransmission.

4.2.4 Image reception

Once the AI Deck module is fully functional and operational, it is necessary to receive the images at the ground base in some way. This is done via a socket connection to ESP 32. This requires that the ground base and the drone are connected to the same local network. In order to maintain a modular architecture based on ROS, a package has been developed that takes care of the connection to the AI Deck, reception and decoding of images and subsequent publication in a ROS 2 topic.

Figure 4.11 Image as received from the drone.

However, it can be observed that the image as received from the camera has a greenish look. This is because the sensor used in the camera is Bayer technology. These sensors have more surface area sensitive to green than to red and blue. Therefore, a raw image received directly from the camera does not represent the colours as they are perceived by humans.

One solution to this problem is to apply a simple colour correction. A simple but effective method attempts to equalise the histograms of the three channels on a white surface [49]:

1. Point the camera at a uniform white surface.
2. Take the pixel values of the image. The red (R), green (G) and blue (B) channel values should be in approximately the same proportion. This is represented in the histogram of the three channels, which should have similar shapes.
3. Pixels in the upper and lower 0.05 percentiles are discarded.

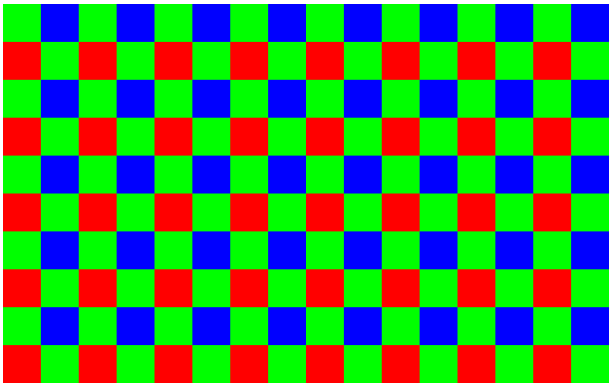


Figure 4.12 Bayer sensor's color matrix.

4. Stretch the resulting histogram to the full range with a multiplier.

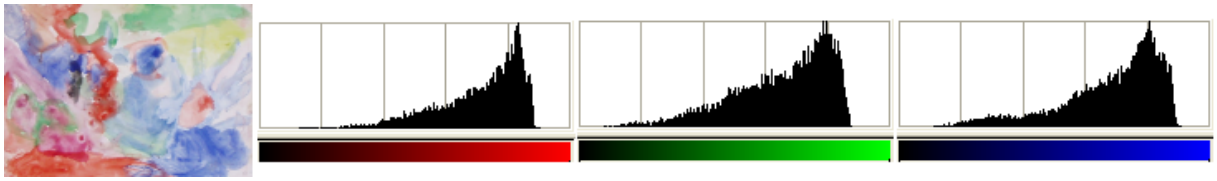
5. Use these multipliers on the rest of the images.

This method is not perfect and stretches each of the channels separately as seen in Figure 4.13, which may affect the overall brightness of the image, but the results are acceptable for this application.

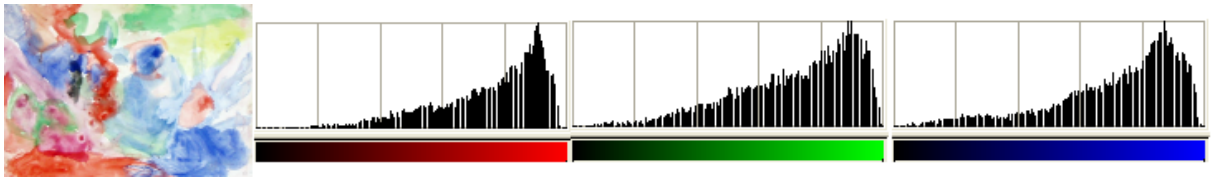
```
1  def colorBalanceStretch(img, perc = 0.05):
2      factor = [1,1,1]
3      img_out = img.copy()
4      for c in range(3):
5          channel = img[:,:,c]
6          mi, ma = (np.percentile(channel, perc), np.percentile(channel,100.0-perc))
7          factor[c] = 255.0/(ma-mi)
8          img_out[:,:,c] = np.clip(img_out[:,:,c]*factor[c],0,255)
9
10     print("Factors: {}".format(factor))
11     return img_out, factor
```

Listing 5 White Balance function in Python.

Once these images are corrected, they are published in ROS to be used in other modules. In the case of this thesis, these images must be read by the pallet identification block. This is done through a video stream device in Linux. Again, following a modular architecture, a ROS package has been realised that is able to read images from a given topic and create a capture device or virtual camera from them. This package and the stream receiver for the AI Deck make up the Stream Handler module in Figure 4.15.



(a) Previous to white balance



(b) After white balance

Figure 4.13 White balance's effect in the histogram [49].

Figure 4.14 Image after the color correction.

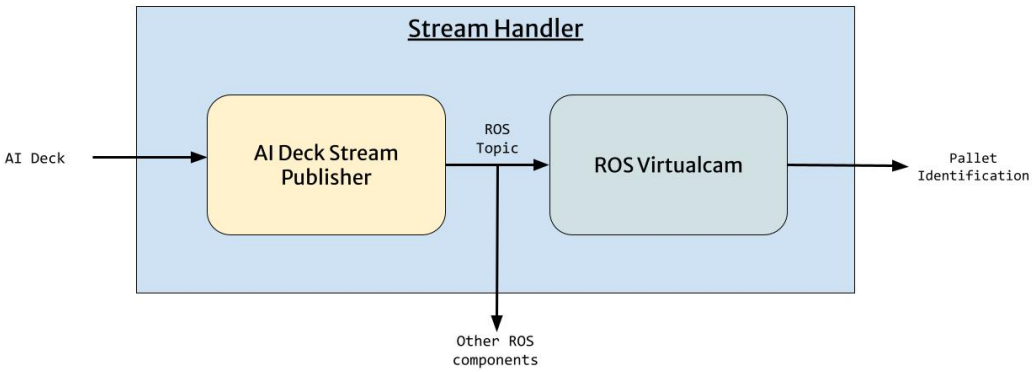


Figure 4.15 Stream handler's architecture.

4.2.5 Pallet Identification

The method used in this thesis is the model trained by FLW for pallet identification, composed of the YOLO V3 and PCB structures, that was presented in the Chapter 3. The classical structure of a europallet has three support blocks on the sides as seen in the Figure 2.8. The first step is to recognise each of these blocks in a given image sequence with the help of YOLO. The algorithm for detecting the three blocks can be found in Algorithm 1.

Algorithm 1 Block detection

```
Nblocks ← 0
blocksImage ← empty
blocksCrop ← empty
nextBlock ← false
while Nblocks < 3 do                          ▷ Repeat for the three support blocks
    im ← getImage()
    box ← YOLOdetect(im)
    if box is not empty then                  ▷ If something was detected
        distanceX ← abs(im.center.x − box.center.x)
        if box.center.x ≤ im.center.ix then   ▷ If the block is on the left
            nextBlock ← false
            if distanceX ≤ MAX_DISTANCE_LEFT then
                blockCrop ← im.crop(box)
            end if
        else if distanceX > MIN_DISTANCE_RIGHT then   ▷ Block is on the right
            if nextBlock is false and blockCrop is not empty then
                blocksImage(Nblocks) ← blockCrop
                nextBlock ← true
                Nblocks ← Nblocks + 1
            end if
        end if
    end if
end while
```

As a brief summary of the behaviour of this algorithm:

- Blocks are expected from the left to the right of frame in the image sequence.
- YOLO detects the block and a crop of the image is performed.
- The system waits until the block has gone on the right side to start looking for the next one.
- This is repeated for the three support groups.

The parameters *MAX_DISTANCE_LEFT* and *MIN_DISTANCE_RIGHT* deliniate the width of the central area used for the detection and cropping the image to the block size.

Everything can be visualized on real time by the user in an application window 4.16, where the actual position of the detected box and its confidence can be seen.

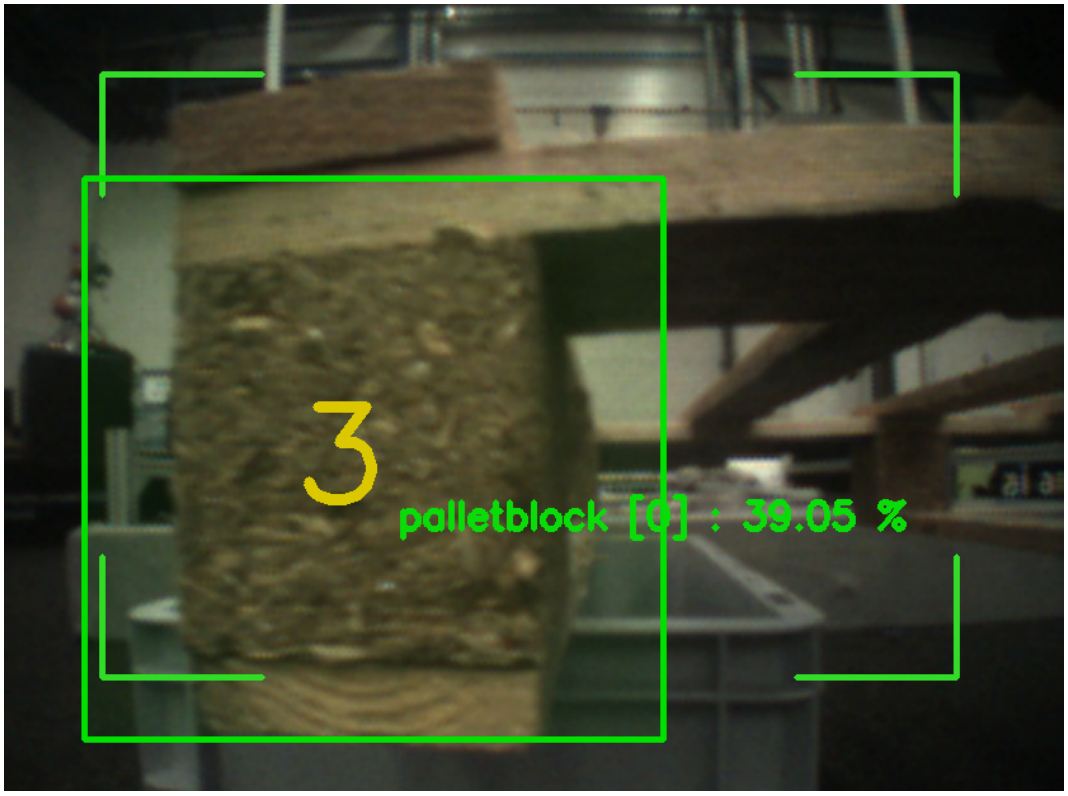


Figure 4.16 User interface at the block detection. Block 3 detected.

Once this algorithm has been carried out, three images are obtained as a result, corresponding to the three blocks of the pallet. Taking these three blocks and with the help of the PCB model, it is possible to obtain distinctive characteristics of the piece to create a database of pallets or recognise them, in a similar way to how it would be done with faces. Specifically, the model creates three characteristic vectors, one for each block. These three vectors are stored in a pallet database.

In order to recognize, the same process of extracting three feature vectors is repeated. Then, the three vectors are compared to the ones in the database. The way this is done is a one by one comparison with the pallets in the database. The first, second and third feature vectors from the new pallet are compared with the first, second and third feature vectors respectively of each pallet in the database, giving as a result three measurements of distance in the feature space. In the figure 4.17 the result of the identification in the user interface can be seen. The three block images that have been processed with the PCB model are present (stretched for the model input).

To get a final result of the best match for the new pallet to be recognized, the mean value of these three distances is calculated. Thus, the matching pallet from the database will be the one with the smallest mean distance value.

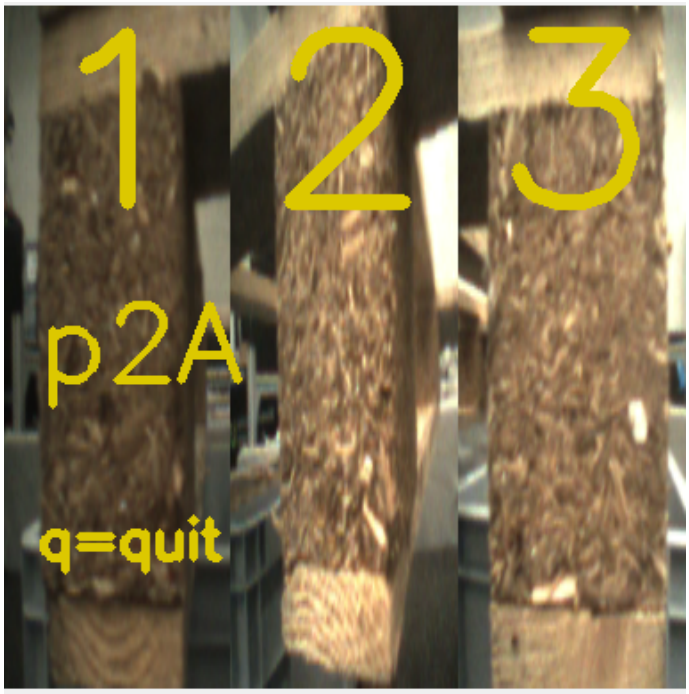


Figure 4.17 User interface at the identification. The pallet identified as pallet "p2A"

4.3 Autonomous Mission

Once the components are up and running, it is time to see how to integrate them into an autonomous mission for the Crazyflie. The chosen way to implement autonomous missions is through the Aerostack 2 *Python Interface*, which provides an API to access the most common elements of the framework. In Aerostack, applications that implement a specific use-case are called projects. Therefore, in order to group the developed operation, one of these projects is created on *GitHub* [50]] that encompasses all the high-level behaviour and configuration of Aerostack 2 for the autonomous mission, except for the components corresponding to the identification of pallets.

Before explaining the structure and components of the developed project, it is important to comment on some modifications that have been necessary in the Aerostack framework to include the necessary functionality for the pallet recognition mission.

As discussed in the Chapter 3, behaviours in Aerostack 2 implement tasks as *ROS2 action servers*, which can be called by clients by other ROS2 nodes (e.g. python interface node). Among these behaviours, at the time of doing this thesis, it is possible to find implementations for take off, landing, going to a position and following a path, which are accessible, for example, from the Python Interface through the *Drone Interface* functions *takeoff, land* or *go_to_point*. Therefore, the flow of control commands from the user to the hardware is as shown in Figure 4.18.

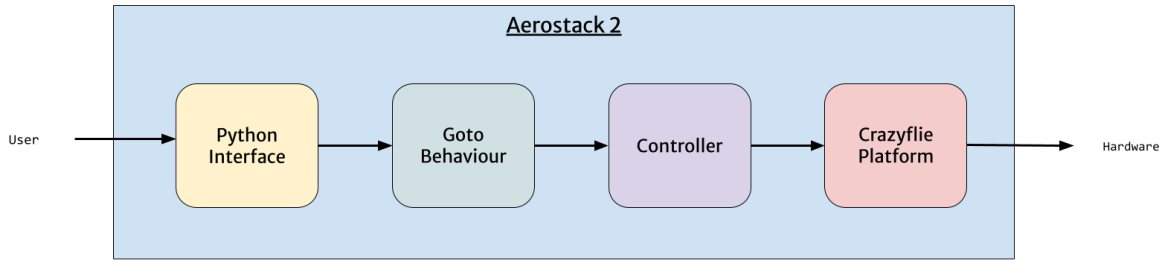


Figure 4.18 Commands' flow from user to the drone.

4.3.1 Yaw control

Goto Behaviour

In Aerostack 2 the go to behaviour's only completion condition was proximity in position. This does not implement a precision control of the **yaw orientation** of the drone, which is necessary for this thesis, as it is vital that the camera is pointed in the right direction. Therefore, it was decided to modify the structure of the goto_behaviour [51] to include this functionality. Within the repository of this behaviour there are two sub-packages:

- **Goto_behaviour**: Implements the ROS node necessary for its operation.
- **Goto_plugin_base:** As with the drivers, it is possible to have several different implementations of the same behaviour. This is done through plugins. This module contains the base plugin from which the rest of the implementations must be derived.

These implementations can be found in the repository [52]. It can be seen that there are three different plugins:

- **goto_plugin_position:** Communicates with the controller through position commands (POSITION with YAW ANGLE). Remember that the controller sends velocity commands to the drone. Therefore a position - velocity control is performed.
- **goto_plugin_velocity:** Orientation and velocity commands for position (SPEED with YAW ANGLE). It implements a velocity - velocity control, although it is possible to skip the controller and pass the commands directly to the drone by activating the controller bypass parameter.
- **goto_plugin_traj:** Performed with a trajectory generator, not used in this thesis.

The speed and position plugins have been modified, so that it is also possible to control the orientation of the drone. The behaviours have these three fundamental parts or functions:

1. **Service request and acceptance:** Implemented in the *onAccept* function. Here the validity of the given request and goal is checked.

2. **Execution:** Implementation of the behaviour itself. In addition, feedback is provided periodically during execution *onExecute* function.

3. **Cancellation:** Management of the behaviour cancellation request. Function *onCancel*.

The necessary changes to add the targeting functionality are found both in the request and in the execution. In the request contains the fields in Table 4.1:

Field	Type	Description
target_pose	geometry_msgs/Pose	Goal position of the movement. 3D Vector.
max_speed	float	Maximum speed.
ignore_pose_yaw	bool	A flag used to select the type of movement.
yaw_mode_flag	uint	To select the type of yaw control. Not used.
yaw_angle	float	Angle for the yaw control. Not used.

Table 4.1 Goto action fields.

```
# Go to waypoint action definition
# Request
geometry_msgs/Pose target_pose          # goal pose (m)
float32 max_speed                       # maximum speed (m/s)
bool ignore_pose_yaw                    # ignore yaw in pose

uint8 KEEP_YAW            = 0
uint8 PATH_FACING         = 1
uint8 FIXED_YAW           = 2

uint8 yaw_mode_flag                     # yaw mode
float32 yaw_angle                       # yaw (rad)
---
# Result
bool goto_success                       # false if failed
---
# Feedback
float32 actual_speed                    # actual speed (m/s)
float32 actual_distance_to_goal         # distance to goal (m)
```

Listing 6 GoToWaypoint.action at github.com/aerostack2/as2_msgs

In the original implementation there were only two types of yaw control:

1. **Path Facing:** The UAV was oriented in the forward direction of movement.

2. **Keep Yaw:** The initial orientation of the movement was maintained.

The control mode was selected by means of the boolean *ignore pose yaw* in the request, so the yaw mode and yaw angle fields were not used. The proposed changes are as follows:

1. Use the yaw mode field to select between three control modes: *Path Facing, Keep Yaw and Fixed Yaw.*

2. This last control mode uses the yaw angle field to indicate the desired orientation.

3. The Yaw Mode Flag will now be used to modify the completion condition of the behaviour. Originally this is done by means of a distance threshold in position. In this thesis another optional threshold is added for the drone orientation. It is possible to activate or deactivate this additional condition by means of this flag. If this flag is active, the orientation threshold condition will have no effect on the completion of the behaviour.

Once these changes have been made and the corresponding logic has been extended within the behaviour *goto plugins*, it is also necessary to check the interface with the adjoining layers 4.18. The communication with the controller does not need to be adapted, but the communication between the Python Interface and behaviour does.

Python Interface

The Python Interface [53] provides a class that groups all the functions available for UAVs, the DroneInterface. This class allows the management and access to the functionalities associated with a drone (in this case, the crazyflie), such as obtaining the current position of the drone, arming the engines or calling the landing or take off behaviour. The goto behaviour can also be called from the python interface and it is necessary to make some modifications to adapt it to the new version mentioned in the previous subsection. The function used in this thesis is go_to_point. In the original version the following parameters are accepted:

- Point: Goal.
- Speed: Desired movement speed.
- Ignore Yaw: Boolean.

These correspond to the action request fields of the goto behaviour 6. To these fields it is necessary to add those corresponding to yaw_mode and yaw_angle. In the case of yaw mode, it is important to keep the enumeration values, so that the choice of control mode is correct. This can be done by taking the values from python's own internal libraries

```
class goto_yaw_modes():
    PATH_FACING = GoToWaypoint.Goal.PATH_FACING
    FIXED_YAW = GoToWaypoint.Goal.FIXED_YAW
    KEEP_YAW = GoToWaypoint.Goal.KEEP_YAW
```

Listing 7 Yaw control modes

generated from the action's description by ROS, resulting in the wrapping class in Listing 7.

Once the internal structure of the goto in the Python Interface is adapted to the new behaviour,it can be called directly from the code as seen in 8.

```
# Original version
go_to_point(point: tuple, speed: float, ignore_yaw: bool)

# New version
go_to_point(point: tuple, speed: float, yaw_mode: int, yaw: float, ignore_yaw: bool)
```

Listing 8 Goto Behaviour in Python Interface. Original and new versions.

4.3.2 Project Structure

The project [50] should contain the following main components:

- Configuration files for Aerostack 2.
- Bash scripts to launch all the necessary ROS nodes and Aerostack 2 packages.
- Python scripts describing the behaviour in the task to be performed.

The configuration files describe the behaviour of some components, such as controllers or behaviours, by means of parameters. These can be selected when executing Aerostack 2 packages in the bash script. In the Listing 9 you can see the configuration of the employed controller in the controller handler, mentioned in the Crazyflie Platform section.

These files are usually written in YAML format. In this thesis, the following components are configured:

- Controller: Speed controller, the only one available at the time of writing this thesis. As well as its control frequency and proportional, differential and incremental gains for each degree of freedom.
- Goto Behaviour: The plugin is used in position.

```yaml
/**:
  ros__parameters:
    cmd_freq: 80.0  # Hz (default: 100.0)
    info_freq: 10.0  # Hz (default: 10.0)
    plugin_name: "controller_plugin_speed_controller"
    use_bypass: false
    plugin_config_file: 'drone_config/controller_config.yaml'
    plugin_available_modes_config_file: ""
```

Listing 9 Controller configuration.

- Takeoff Behaviour: Also in position.

Bash scripts are used to execute the necessary Aerostack 2 modules. An example of the minimum configuration required is Listing 10. In 10 tmux [54] is used to create different windows within the same session (macro new_window). The parameters and configurations for each node launched are parsed directly from the bash command. There, it is also possible to see other external dependencies apart from Aerostack 2. For example, the vicon_receiver package, which connects to the Vicon server in order to read the pose estimations.

In addition, if a simulator is used instead of a real UAV, it is also convenient to launch it in an analogous way by providing the corresponding configuration.

The Python script that makes use of the Python Interface to command the drone to perform the tasks is also a fundamental piece. It contains all the drone's movement logic, as well as other external components that need to be used for the specific mission. In this thesis, the developed autonomous missions all follow the same structure:

1. Connection to the corresponding Drone Interface via a unique id to the platform.
2. Activation of offboard control mode
3. Arming the drone
4. Execution of the required sequence of movements (mission) with the help of the modified behaviour goto.
5. Disarm the drone. In the specific case of the Crazyflie, arming the drone does not perform any particular action, because being such a small drone, the motors do not need to be armed beforehand.
6. Shut down the drone.

The step 4) is the one that contains the main logic and will change according to the mission or experiment to be performed, but all movements are based on the modified yaw

```
new_window 'vicon_pub' "ros2 launch vicon_receiver \ client_remap.launch.py \
    hostname:=192.168.2.221:801 \
    namespace:=vicon \
    remap_from:=/vicon/cf_ai/cf_ai/pose \
    remap_to:=${drone_namespace}/ground_truth/pose "

new_window 'crazyflie_interface' "ros2 launch crazyflie_platform \
    crazyflie_platform_launch.py \
    drone_id:=$drone_namespace \
    drone_URI:=radio:///0/82/2M/E7E7E7E703 \
    controller_type:=1 \
    external_odom:=True \
    external_odom_topic:=/${drone_namespace}/ground_truth/pose \
    estimator_type:=2 \
    mass:=0.040 "

new_window 'controller_manager' "ros2 launch controller_manager \
    controller_manager_launch.py \
    drone_id:=$drone_namespace  \
    use_bypass:=False \
    config:=drone_config/controller_plugin.yaml"

new_window 'traj_generator' "ros2 launch trajectory_generator \
    trajectory_generator_launch.py  \
    drone_id:=$drone_namespace "

new_window 'basic_behaviours' "ros2 launch as2_basic_behaviours \
    all_basic_behaviours_launch.py \
    drone_id:=$drone_namespace \
    config_goto:=drone_config/goto.yaml \
    config_takeoff:=drone_config/takeoff.yaml"

new_window 'state_estimator' "ros2 launch basic_state_estimator \
    basic_state_estimator_launch.py \
    drone_id:=$drone_namespace \
    ground_truth:=True \
    odom_only:=False "
```

Listing 10 Main launcher bash script.

control explained in the previous section. Also in step 4. the pallet recognition module is launched. In the next chapter, the experiments carried out will be discussed in more detail.

An simple mission example can be seen at the Listing 11. For more information about the actual implementation of the project, please refer to [50].

```
drone_id = "drone_miguel"
print("Connecting to: ",drone_id)

uav = DroneInterface(drone_id, True)
uav.offboard()
uav.arm()

uav.takeoff(1.0, 0.3)
sleep(4)

uav.go_to_point([1.0,1.0.1.0],speed=0.5,
                yaw_mode=DroneInterface.goto_yaw_modes.KEEP_YAW)
sleep(4)

di.land(0.5)
sleep(1)

di.disarm()
uav.shutdown()
print("FINISHED")
```

Listing 11 Simple mission with Python Interface.

Chapter 5

Experiments and Results

In this chapter we will see the different experiments carried out to check the functionality designed in the thesis. As well as a short discussion or commentary on the results obtained.

5.1 Pallet Identification

To evaluate the behaviour of the pallet identification system, the following experiment is carried out:

1. Enter 10 different pallets in the database. More specifically, 5 (1-5) pallets are chosen and both sides of the pallet (A-B) are entered into the database.

2. Perform three attempts to identify each of the same ten pallets. Therefore, 30 different identification results will be obtained.

3. Evaluate the accuracy of the system with the percentage of total successful matches.

To do this, the algorithm based on PCB and Yolo with implementation in Python shown in 4.2.5 is used. In addition, the results of comparison of each identification with the pallets in the database and the images used to extract the feature vectors are stored, so that the causes of identification failures can be analysed.

It can be seen in the Table 5.1 that a total of nine identification errors are obtained. This means that the system achieves 70% accuracy. This is a very positive result compared with those obtained in the paper [28], 96%, taking into account the characteristics of the hardware used.

However, it is worth looking at the robustness that the system can provide. In most cases of failure, it is observed that the final comparison measurement between several pallets is very close. For example, in the case of pallet 1B, the measurements in the Table 5.2 are obtained in the second identification.

Pallet ID	Identification 1	Identification 2	Identification 3
1A	1A	1A	1A
1B	1A	1B	1A
2A	2A	2B	2A
2B	2B	2A	2B
3A	3A	4A	3A
3B	3B	3B	3B
4A	4A	4A	2A
4B	4B	4A	4B
5A	5A	5A	5A
5B	2B	5B	4A

Table 5.1 Identification Results. Errors are marked in red.

Pallet ID	Block 1	Block 2	Block 3	Mean Score
1A	1.61	1.56	1.49	1.54
1B	1.38	1.56	1.60	1.51
2A	2.06	1.83	1.85	1.91
2B	1.83	1.83	1.82	1.83
3A	1.47	2.21	1.86	1.85
3B	1.68	1.53	1.64	1.62
4A	1.53	1.56	1.56	1.55
4B	1.30	1.59	1.93	1.60
5A	2.15	1.91	1.64	1.90
5B	1.96	1.78	1.78	1.84

Table 5.2 Results of the identification of 1B.

The pallet is identified as 1A, but with a minimal difference between pallets 1A and 1B, which does not make the result very reliable. In addition, we can see that other pallets also come dangerously close to the correct pallet score, such as 4A.

It is also possible to analyse the individual block identifications, since, although they are generally in a close range, it is possible to find cases with very unbalanced scores. This is the case of 3A. Figure 5.1 shows the images taken of the pallet 1B blocks and

Figure 5.2 shows the images of the pallet 3A in the database.

Figure 5.1 1B blocks used for identification.

Figure 5.2 3A blocks in the database.

Although the block scores do not directly represent the similarity between the images, it can be expected that features obtained from very different images are very distant in the resulting vector space, and therefore their distance score is much higher. In the case of 1B - 3A we can see that the first block has a relatively small score compared to the other two blocks, while the second one gives a rather distant measure. This is directly visible in the images, the first blocks contain similar engravings, while the second ones are completely different in texture, printings and even illumination.

This lack of certainty in identification again highlights the lack of resolution of the AI Deck camera, as well as the need to employ some more refined technique for the pallet comparison and selection process that can ensure greater certainty.

5.2 Vicon

First, a couple of tests are carried out to analyse the localisation behaviour using the Vicon mocap system and to be able to adjust the configuration parameters of the drone's Kalman Filter. To do this, data is first collected from the pose of the static drone.

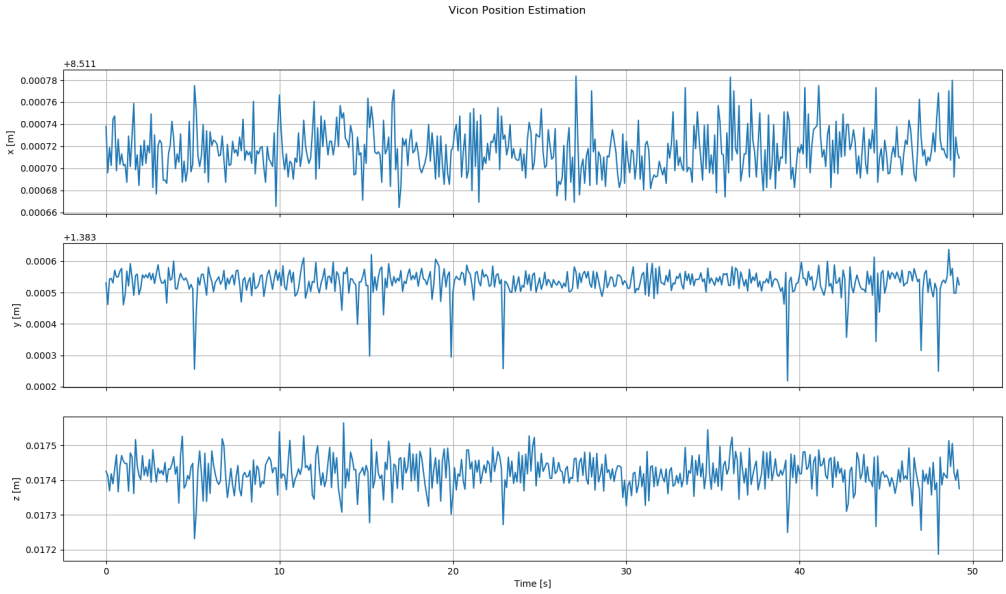


Figure 5.3 Position estimated by the Vicon system.

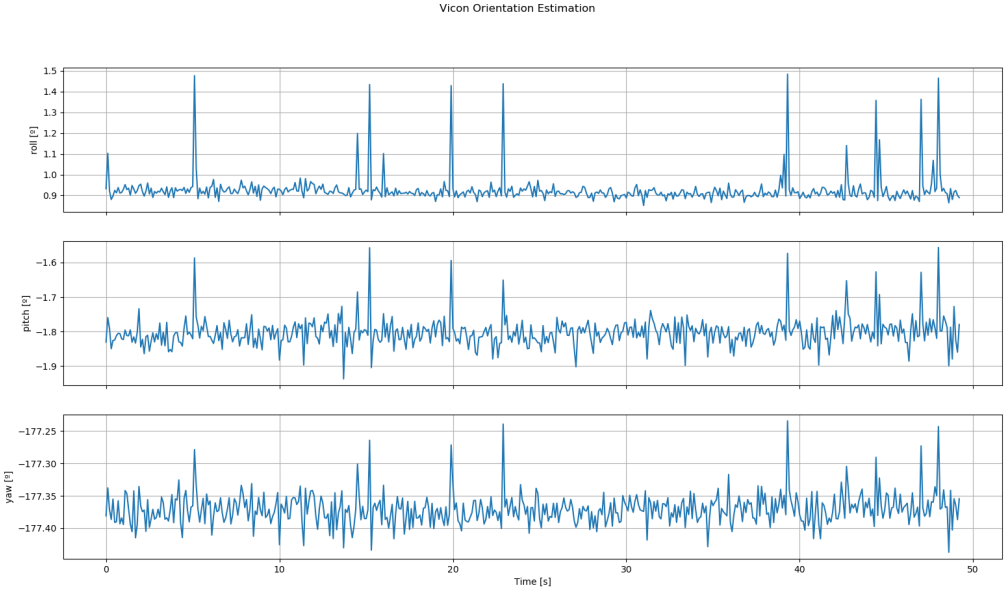


Figure 5.4 Orientation estimated by the Vicon system.

These graphs represent the measurements given by the vicon over almost 50 seconds of the drone's position and orientation. It is worth mentioning that the orientation is not given directly in roll, pitch and yaw (RPY), but quaternions are used. For representativeness, they have been converted to RPY angles.

In the position fairly stable measurements can be seen. The standard deviation of each of the axes is:

- X [m]: 2.16810^{-05}
- Y [m]: 4.51210^{-05}
- Z [m]: 4.78810^{-05}

With an average standard deviation of 3.82210^{-05}. This is in the order of a hundredth of a millimetre, more than sufficient for the tasks at hand.

The following standard deviations can be observed in the orientation:

- Roll [º]: 0.073
- Pitch [º]: 0.041
- Yaw [º]: 0.024

This means an average standard deviation of 0.046 º, which is also quite small for the use case. However, this measurement is not useful for setting the Kalman Filter parameters of the Crazyflie. Therefore, it is necessary to calculate the standard deviations of the quaternion. These have four unit-less components; x, y, z, w.

- X: 3.75210^{-04}
- Y: 6.34310^{-04}
- Z: 7.33810^{-06}
- W: 2.15610^{-04}

With an average standard deviation of 3.08110^{-04} (0.0003081). Which is higher than the default value of the *"locSrv.extQuatStdDev"* parameter (0.00025) that defines the variance of the external orientation quaternion updates, which can cause an incorrect functioning of the drone position estimation and consequently instability.

It is worth mentioning that the averages obtained may depend on the calibration of the Vicon system performed (it must be done periodically). It is also possible that the behaviour of the system is not the same in all areas of the estimation space (area covered by the Vicon system), so it is necessary to use somewhat higher values than those obtained to maintain a safety margin.

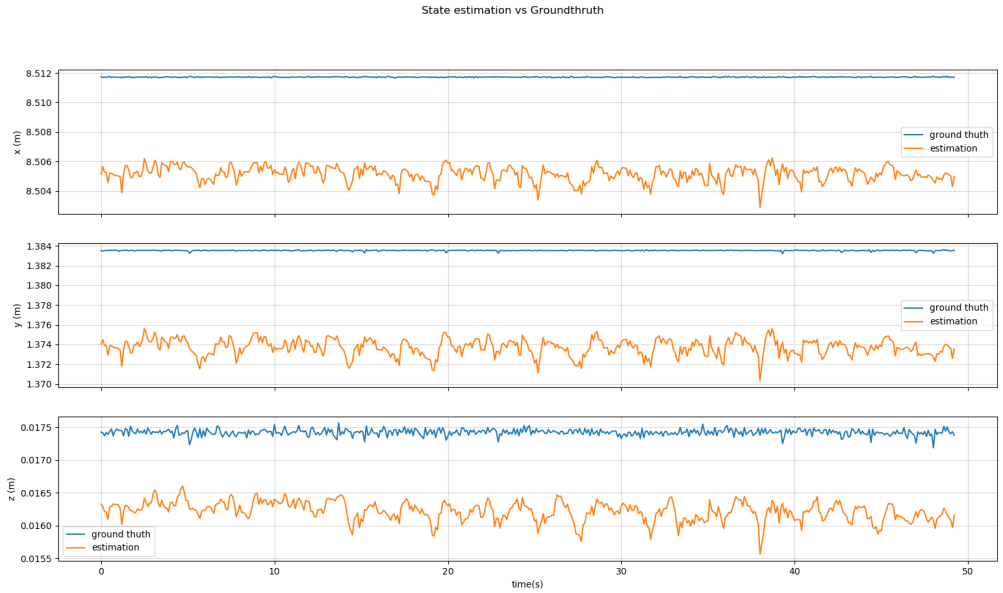


Figure 5.5 Position estimated by the Vicon system vs Kalman Filter.

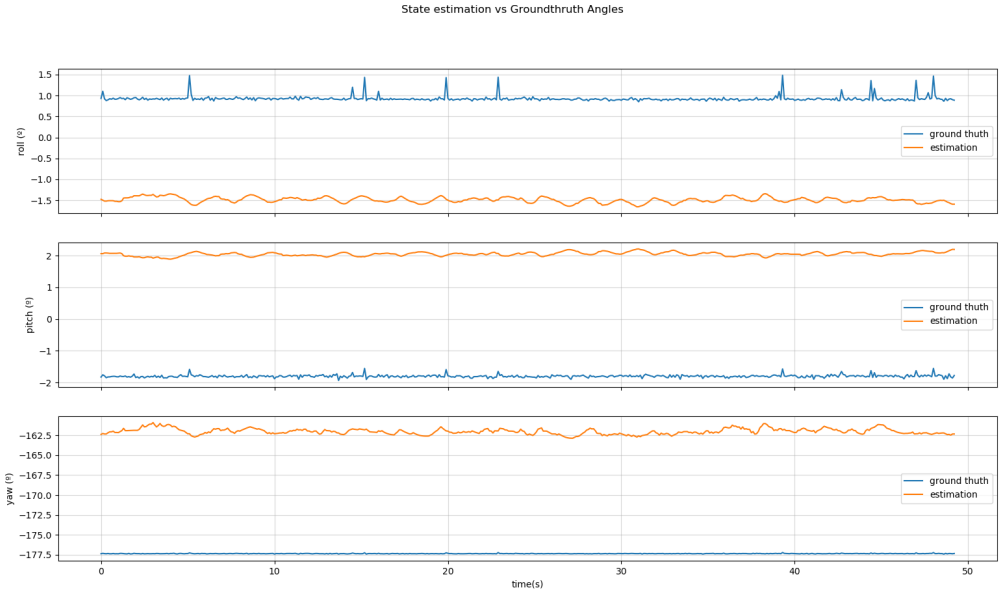


Figure 5.6 Orientation estimated by the Vicon system vs Kalman Filter.

It is also possible to compare this information with the estimate made internally in Crazyflie, as it can be seen in Figures 5.5 and 5.6.

In this case it can be seen that the internal Kalman filter estimate of the position is quite close to that of the Vicon, while that of the orientation (especially in yaw) is quite far away. It does not seem to converge to the Vicon value either, but has already reached its steady-state value. Unfortunately, no further convergence towards the Vicon has been

achieved without setting the standard deviations to zero. This would be an ideal case which is also undesirable, as the mocap system also has estimation failures.

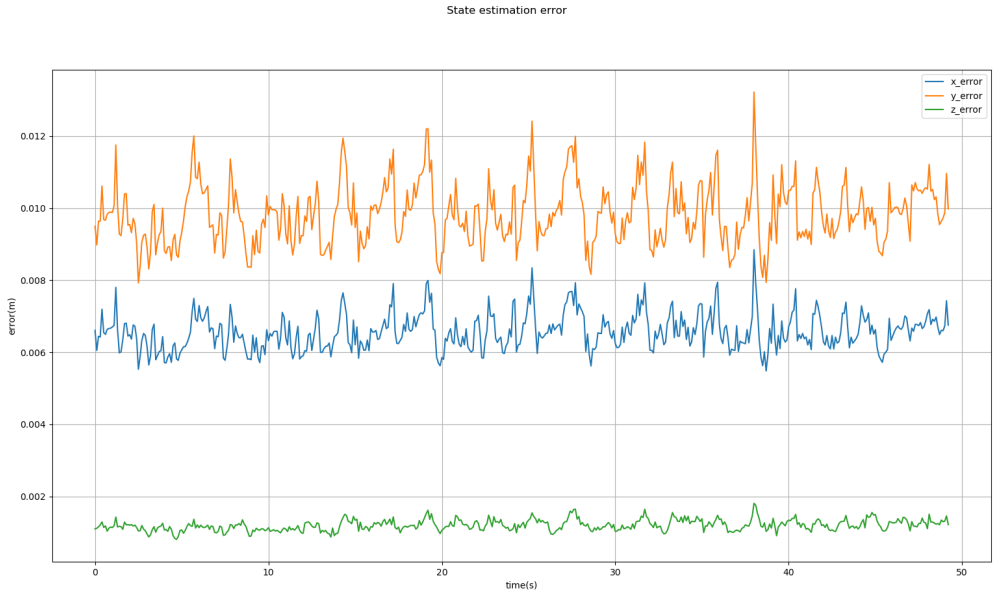


Figure 5.7 Error in position estimated by the Vicon system vs Kalman Filter.

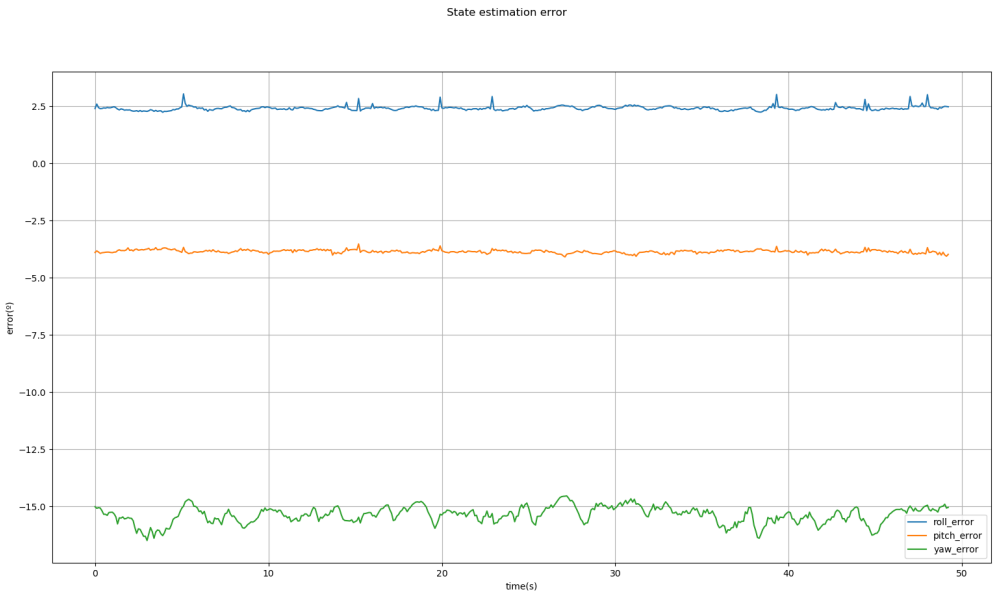


Figure 5.8 Error in orientation estimated by the Vicon system vs Kalman Filter.

However, once all the drone's parameters have been configured and the drone is in motion, this difference is less than 5º. This difference between the orientation obtained from the Vicon system and the orientation that the drone calculates internally with the Kalman filter is an important aspect to take into account. This difference in angle also

leads to differences in the Aerostack 2 control loop, closed with the Vicon estimates, and the drone's internal control loop. This makes it difficult to use high speeds, as there is a drift in the orientation of the movement, as well as the application of very precise movements.

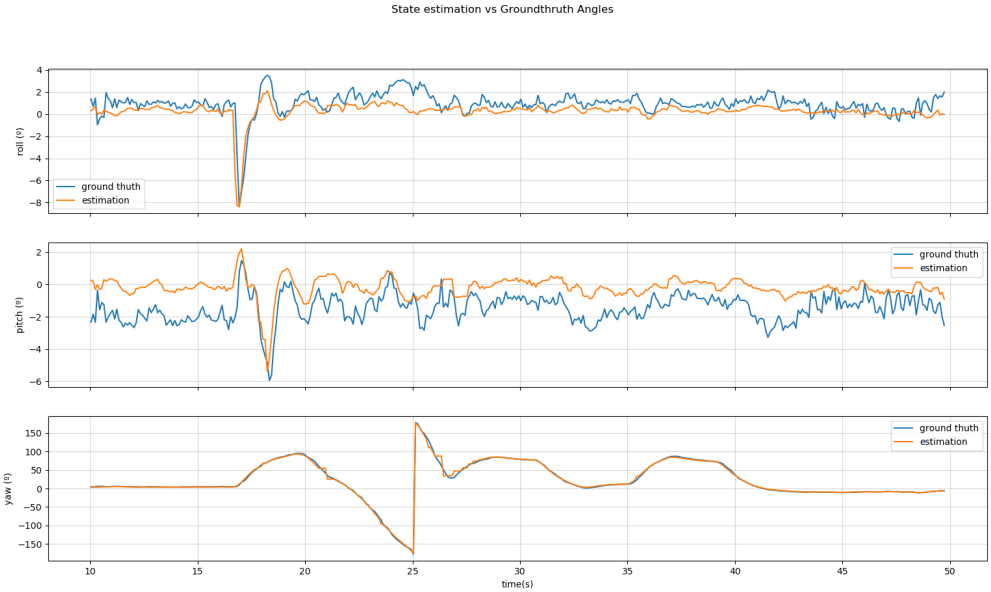


Figure 5.9 Orientation estimated by the Vicon system vs Kalman Filter. With parameters adjusted.

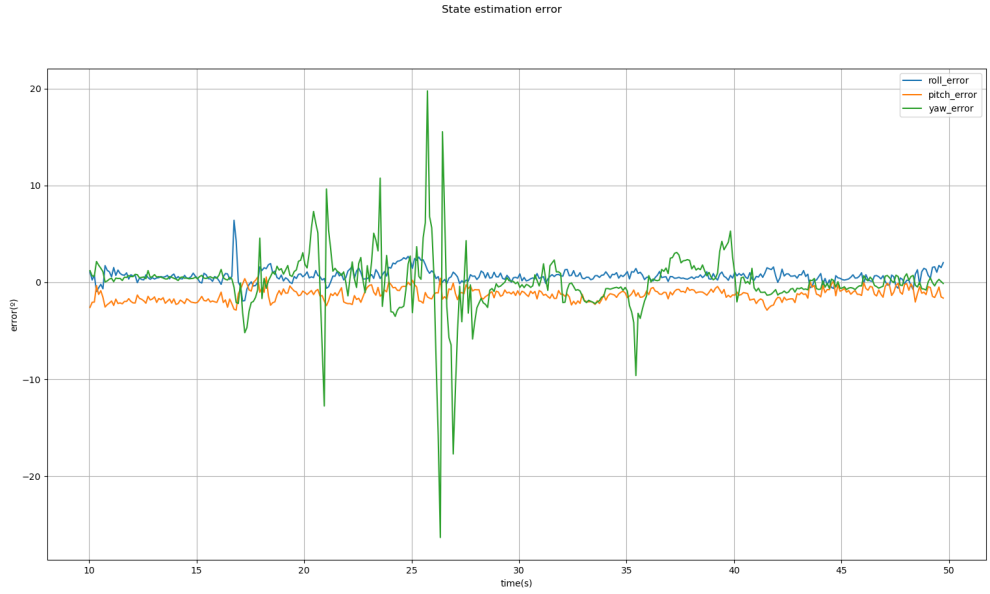


Figure 5.10 Error in orientation estimated by the Vicon system vs Kalman Filter. With parameters adjusted

It is worth mentioning that the jumps seen in the graph in orientation from -180º to +180º are due to the fact that the orientation values are constrained to these limits, but this has been taken into account when calculating the difference between reference and actual orientation for the motion completeness condition with the yaw threshold.

5.3 Drone

Once the data from the mocap system has been analysed and the internal state estimator has been adjusted, the behaviour obtained with the drone can be analysed.

5.3.1 Yaw Control

One of the essential functionalities added as part of this thesis is the orientation control (yaw angle) the crazyflie to orient the camera to the desired location. An autonomous mission with the structure presented in Section 4.3.1 is used to test this type of control. This mission contains the following steps:

1. Take-off.
2. Movement to predetermined position with yaw control in Path Facing mode and an angle threhsold of 15º.
3. Pivot twice in the same position at 90º angles (fixed yaw mode).
4. Landing.

It can be seen that the drone is able to follow the references without problems, both in path facing (approximately 17 to 27 seconds) and in fixed yaw, although the dynamics are somewhat slow due to the reduced speed used. This is due to the aforementioned problem of orientation difference between the Kalman and the Vicon. Nevertheless, the result is very satisfactory and more than sufficient for the required camera orientation task.

5.3.2 Position Control

Similarly, an experiment can be carried out to test position following. The developed mission has the following structure:

1. Take-off.
2. Taking the position after take-off as the initial position, calculate the positions of a square of 2 metres on each side in the XY plane.
3. Move to each of the points. Position threshold of 3 centimeters and 15º in angle.

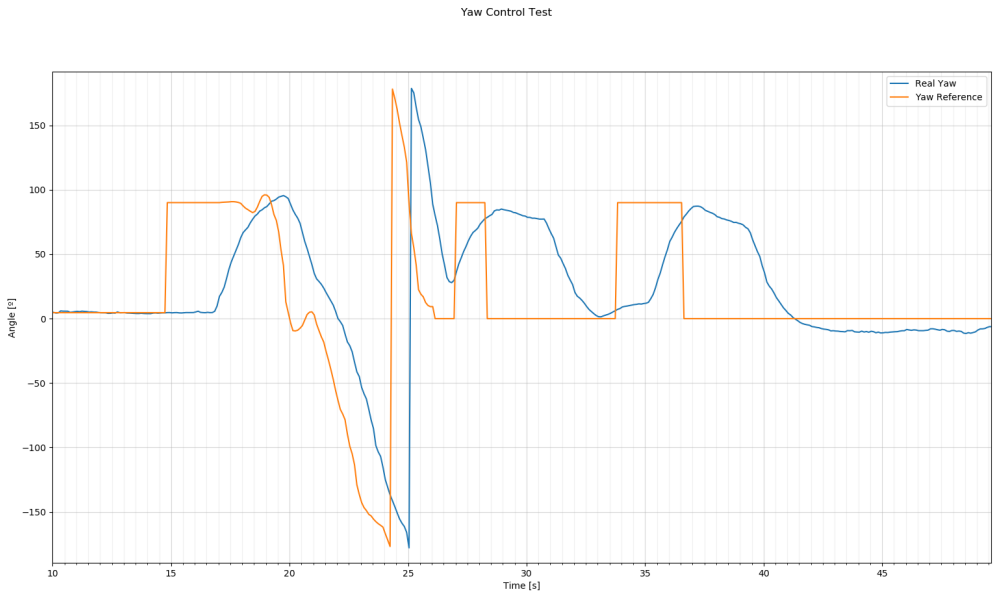


Figure 5.11 Orientation test

4. Landing.

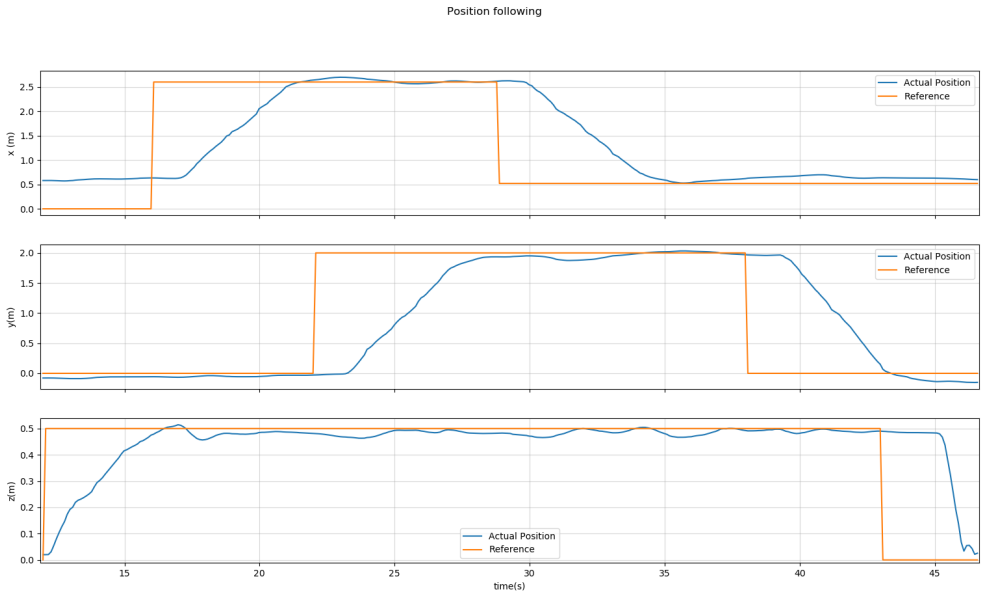


Figure 5.12 Position test. Position vs reference.

Again, it can be seen that the drone is able to follow the proposed references without any problems. The movement used in this mission is in yaw path facing mode. The response in orientation is also satisfactory, being able to reach the references with the given tolerance.

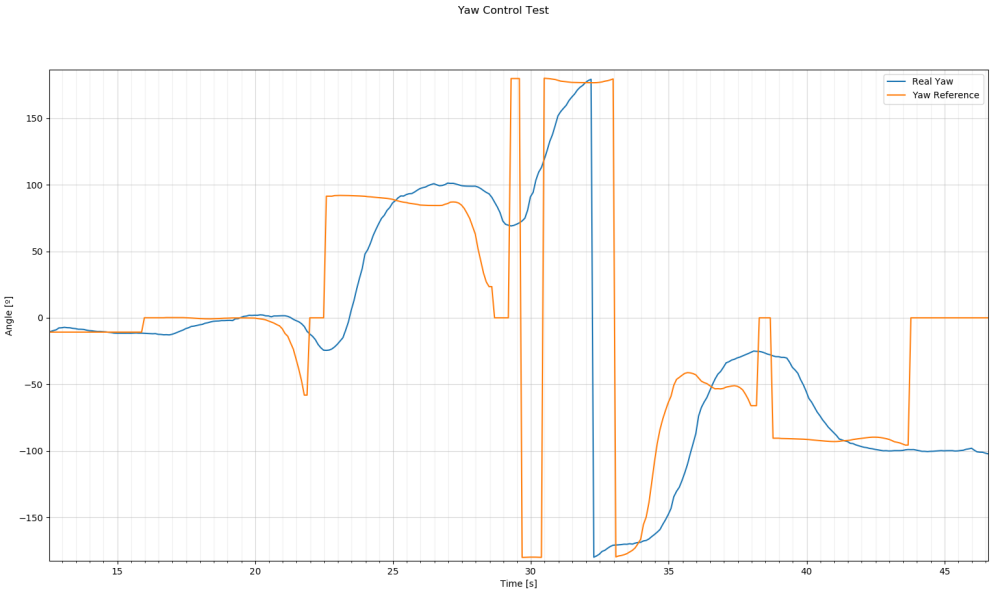


Figure 5.13 Position test. Yaw vs reference.

5.3.3 Precision Movements

The thresholds used in the above tests can be reduced, but if very small tolerances are used, it may take a long time for the drone's movement to stabilise around the reference. This may be primarily due to the angle difference between the Vicon system and the Crazyflie Kalman filter estimate. The velocity control commands are given in the global reference system, so this difference in orientation means that the direction of movement performed by the drone does not correspond accurately to the direction of the planned movement.

But on the other hand, there is also a problem with very subtle and precision movements with the Crazyflie. When the drone is very close to the target point and the threshold in position is less than 3 centimetres, it can be observed that the drone is close to the equilibrium position but does not stabilise. This should be due to a bad configuration of the PID controller used, but this behaviour is observed in all the tested configurations, with higher and lower values of the PID constants.

Furthermore, it can be seen in the figure 5.15 that the velocity command sent to the drone always indicates a movement towards the target in the case of the X and Y axes, but the drone does not get closer to the target and pursues with its oscillatory movement. This suggests that the problem may be in the low-level or hardware of the drone itself, unable to achieve the required accuracy, or due to perturbations.

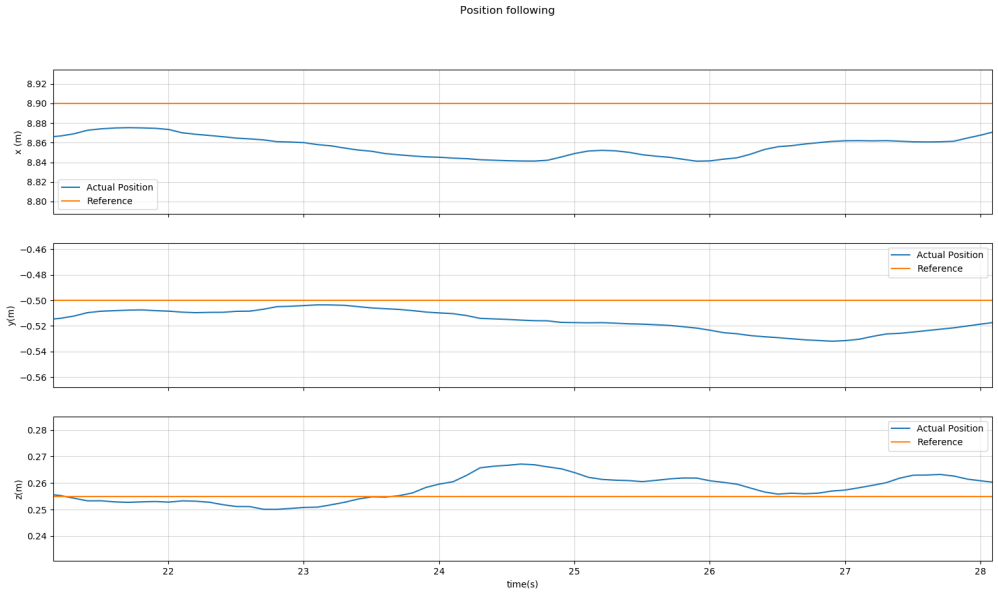


Figure 5.14 Position test. Close to the objective.

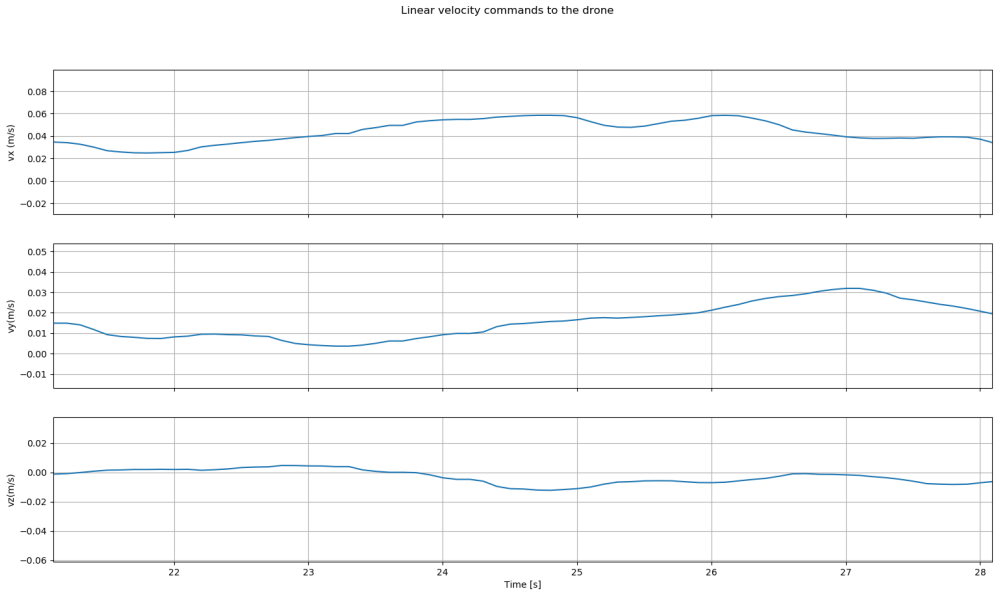


Figure 5.15 Velocity commands sent to the drone.

5.4 Pallet Identification Mission

In this mission, all the modules implemented in the thesis are put together to perform an autonomous pallet identification mission with the Crazyflie. The objective is to take the drone to a predefined position where a pallet is located. Subsequently, a scanning movement of the three blocks of the pallet is performed while transmitting video with the

AI Deck to the pallet identification system. To do this, it is necessary to have a pallet database where the pallet to be identified is registered. The aim of these tests is, therefore, to check if it is possible to detect the pallet blocks while the drone is in flight with an autonomous mission, but the accuracy of the identification system is not analysed.

The mission has the following structure:

1. Take-off.
2. Flight to the predefined position.
3. Orientation towards the pallet.
4. Scanning movement of the pallet blocks. In the predefined direction and with a fixed orientation towards the pallet. During this movement, the pallet perception system receives the images from the drone and executes the identification algorithm.
5. Return to initial position after take-off.
6. Landing.

Figures 5.16 represents the trajectory followed by the drone in position and orientation. Where the predefined position of the pallet is, in the global reference system:

- X [m]: 8.900
- Y [m]: -0.050
- Z [m]: 0.255

And the scanning movement is aligned with the Y axis. Furthermore, in order to avoid possible false positions along the scanning movement, this is not carried out with a single call to the behaviour goto, i.e. a single target, but the line of movement is taken and divided into three parts, in order to also have a checkpoint in the middle of the pallet. Again, a threshold is used in a position of 3 cm and in an orientation of 15°.

It is worth mentioning that the initial position of the drone is not on the ground (Z = 0m), but starts on a platform. This has been done to test different initial cases to those shown previously. Although the take-off behaviour is not used in this case, but a goto in position is used directly. It can be seen that the drone is able to perform the movement correctly. However, the identification system is not always successful. This is mainly due to a failure in the detection of the blocks, i.e. in the first phase of the identification system with YOLO. By not being able to detect the blocks correctly, it is not possible to take the necessary three images and subsequently extract features for identification. Which makes the combination of all system not very robust.

This failure in detection is caused by a poor positioning of the drone with respect to the pallet. Due to the low resolution of the AI Deck camera, it is necessary to bring the

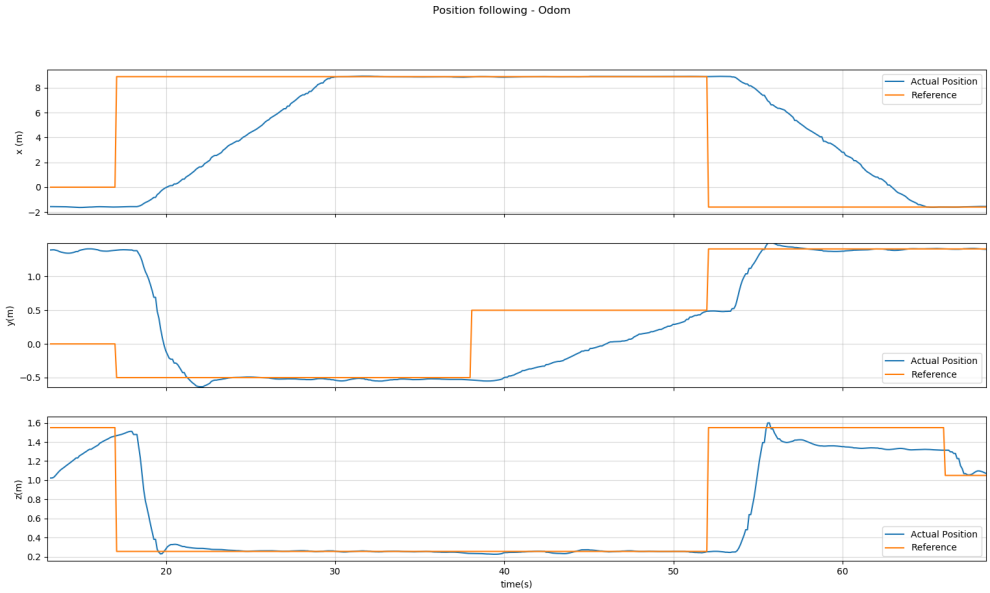


Figure 5.16 Position following

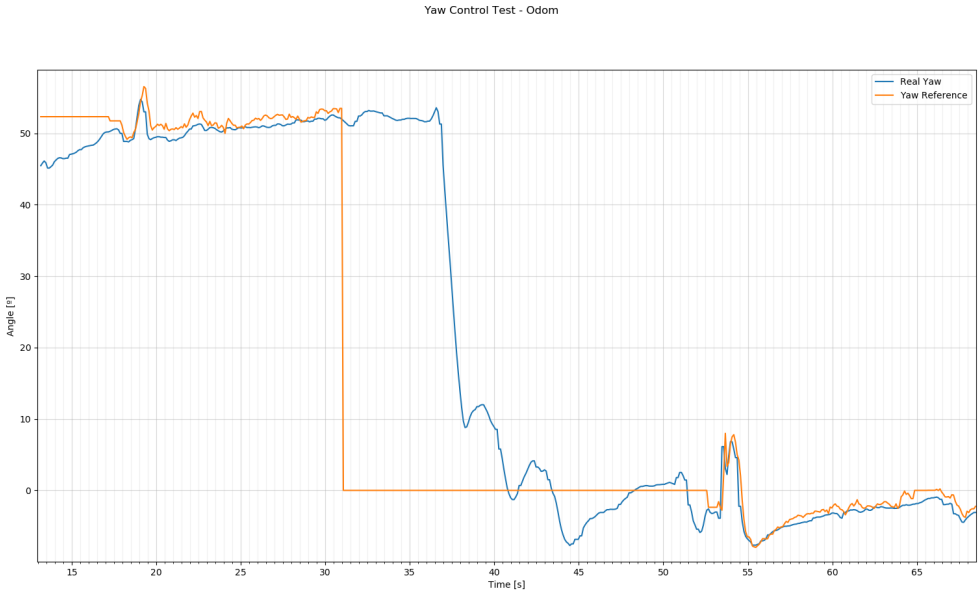


Figure 5.17 Orientation following.

drone quite close to the pallet and keep the blocks at a fairly small range. However, due to the control accuracy problem below 3cm, it is quite difficult to ensure correct positioning of the drone. This experiment has been performed 20 times, of which only 8 have resulted in correct detection of the blocks on the pallet.

```python
def palletRecMission(di: DroneInterface, vel=0.5, N=3):
    p0 = np.array((8.90,-0.5,0.255))
    pf = np.array((8.90,0.5,0.255))

    pts = np.linspace(p0,pf,N)
    scan_yaw = 0

    di.offboard()
    di.arm()
    print("Taking off")
    init_pos = di.get_position()
    takeoff_pos = shiftPos(di,np.array([0,0,0.30]),pos=init_pos)
    di.takeoff(0.5,vel)
    sleep(1)

    print("Going to: ", p0)
    di.go_to_point(p0,vel,yaw_mode=DroneInterface.goto_yaw_modes.KEEP_YAW, yaw=0)
    sleep(1)

    print("Starting scanning movement")
    print("Launch the identification software!!")
    pos = di.get_position()
    di.go_to_point(pos,vel,
        yaw_mode=DroneInterface.goto_yaw_modes.FIXED_YAW, yaw=scan_yaw)
    sleep(1)

    # Go to intermediate points.
    # Done so in order to make sure the drone does not fly away.
    for objective in pts[1:]:
        print("Slide {}".format(objective))
        di.go_to_point(objective,0.07,
            yaw_mode=DroneInterface.goto_yaw_modes.FIXED_YAW, yaw=scan_yaw)

    sleep(1)
    print("Scanning movement ended")

    di.go_to_point(point=takeoff_pos,speed=vel,
        yaw_mode=DroneInterface.goto_yaw_modes.KEEP_YAW)
    sleep(1)

    print("Landing")
    di.go_to_point(point=init_pos,speed=vel,
        yaw_mode=DroneInterface.goto_yaw_modes.KEEP_YAW)
    di.disarm()
    print("FINISHED")
```

Listing 12 Pallet Identification Mission.

Chapter 6

Conclussions

In general terms, the project has met all the initially proposed objectives. The software platform for integrating the Crazyflie drones into the Aerostack 2 operating framework has been realised, using the Vicon motion capture system as source of localization.

Furthermore, the AI Deck expansion board has been correctly configured for the transmission of live images to the stationary processing station. These images are correctly received and processed to prepare them for later use in pallet identification.

The pallet identification module, based on previous technologies developed in Dortmund, has been correctly integrated with the rest of the system. Moreover, the identification results are quite satisfactory considering the characteristics of the hardware used.

Finally, all modules have been used for the creation of an autonomous mission for the Crazyflie, in order to move the drone to the position of a pallet and perform its identification. Although the results of the final integration into the autonomous mission show that the proposed task is possible to perform with the selected hardware and software, the overall system still needs work to improve the robustness of the results.

Therefore, the final objective of the thesis, the proof of concept of the task, is achieved, however some limitations as well as possible improvements for future continuation projects of this thesis have to be discussed.

6.1 Limitations and future improvements

The Crazyflie is not an ideal drone for pallet identification. The main limitation is the low resolution of the camera. This means that the drone has to be very precisely positioned in front of the pallets for the identification algorithm to work properly. Additionally, an improvement in drone control may also be desirable. This may be realisable through a modification of the drone hardware, or even firmware that is better suited to the needs of the task, since the main focus of Crazyflie is swarming and not precise movement. This is

very visible in the libraries published by the developers, the ones currently maintained are swarming oriented. Therefore, the use of another drone with better imaging capabilities or an improved control of the Crazyflie is recommended.

As a possible improvement of the drone control, a reactive control for the scanning movement of the pallets could be implemented. Currently this movement is not coordinated between the pallet identification module and the Aerostack 2 control. A control that is capable of issuing commands to the drone depending on what is detected in the images could greatly increase the robustness of the system.

In addition, the use of a more recently updated drone communication library than the one used in this thesis (crazyflie_cpp) is recommended. Unfortunately, as mentioned in section 3.1.2, the currently supported library is written in Python. This is not directly compatible with Aerostack 2, as it relies on C++ programming of an AerialPlatform, as discussed in section 4.1. At the time of this thesis, a very early and still developing version of the software was used, which also hindered the development of the project.

However, it is also advised to use the most current and stable version of Aerostack 2 which may also already have implemented some of the mentioned changes. The most updated version can be found in the new repository at [55].

Chapter 7

References

[1] "Nav2 - navigation 2". (), [Online]. Available: https://navigation.ros.org/ (visited on 10/2022).

[2] H. Eisenbeiss, "A mini unmanned aerial vehicle (uav): System overview and image acquisition", *undefined*, 2004.

[3] D. Lee and D. H. Shim, "Development of mini-drones and feedback linearization based velocity control for outdoor autonomous swarming flights", *IFAC-PapersOnLine*, vol. 51, pp. 178–183, 22 Jan. 2018, ISSN: 24058963. DOI: 10.1016/J.IFACOL.2018.11.538.

[4] J. Chaoraingern, V. Tipsuwanporn, and A. Numsomran, "Modified adaptive sliding mode control for trajectory tracking of mini-drone quadcopter unmanned aerial vehicle", *International Journal of Intelligent Engineering and Systems*, vol. 13, pp. 145–158, 5 Oct. 2020, ISSN: 21853118. DOI: 10.22266/ijies2020.1031.14.

[5] "Dronisos - drone light shows — europe - usa - middle east". (), [Online]. Available: https://www.dronisos.com/ (visited on 10/2022).

[6] "Dronedj - light show". (), [Online]. Available: https://dronedj.com/2019/11/20/drone-light-shows-china/ (visited on 10/2022).

[7] "Wordreference - swarm". (), [Online]. Available: https://www.wordreference.com/definition/swarm (visited on 10/2022).

[8] R. Arnold, K. Carey, B. Abruzzo, and C. Korpela, "What is a robot swarm: A definition for swarming robotics", *2019 IEEE 10th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference, UEMCON 2019*, pp. 0074–0081, Oct. 2019. DOI: 10.1109/UEMCON47517.2019.8993024.

[9] D. Cristiani, F. Bottonelli, A. Trotta, and M. D. Felice, "Inventory management through mini-drones: Architecture and proof-of-concept implementation", *Proceedings - 21st IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, WoWMoM 2020*, pp. 317–322, Aug. 2020. DOI: 10.1109/WOWMOM49955.2020.00060.

[10] "Ardupilot - versatile, trusted, open". (), [Online]. Available: https://ardupilot.org/ (visited on 10/2022).

[11] "Open source autopilot for drones - px4 autopilot". (), [Online]. Available: https://px4.io/ (visited on 10/2022).

[12] "Pixhawk — the hardware standard for open-source autopilots". (), [Online]. Available: https://pixhawk.org/ (visited on 10/2022).

[13] "What is aerostack · cvar-upm/aerostack wiki". (), [Online]. Available: https://github.com/cvar-upm/aerostack/wiki/What-is-Aerostack (visited on 10/2022).

[14] J. A. Preiss, W. Honig, G. S. Sukhatme, and N. Ayanian, "Crazyswarm: A large nano-quadcopter swarm", *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 3299–3304, Jul. 2017, ISSN: 10504729. DOI: 10.1109/ICRA.2017.7989376.

[15] "C++ library to communicate with bitcraze crazyflie". (), [Online]. Available: https://github.com/whoenig/crazyflie_cpp (visited on 10/2022).

[16] "Epal euro pallet". (), [Online]. Available: https://www.epal-pallets.org/eu-en/load-carriers/epal-euro-pallet (visited on 10/2022).

[17] "Rfid tags: What are they and how do they work?" (), [Online]. Available: https://blog.nortechcontrol.com/rfid-tags (visited on 10/2022).

[18] "Crazyflie 2.1 — bitcraze". (), [Online]. Available: https://www.bitcraze.io/products/crazyflie-2-1/ (visited on 10/2022).

[19] "Positioning systems overview". (), [Online]. Available: https://www.bitcraze.io/documentation/system/positioning/ (visited on 11/2022).

[20] "Userguide cfclient gui — bitcraze". (), [Online]. Available: https://www.bitcraze.io/documentation/repository/crazyflie-clients-python/master/userguides/userguide_client/ (visited on 10/2022).

[21] "Cflib". (), [Online]. Available: https://www.bitcraze.io/documentation/repository/crazyflie-lib-python/master/ (visited on 10/2022).

[22] "Ai deck 1.1 — bitcraze". (), [Online]. Available: https://www.bitcraze.io/products/ai-deck/ (visited on 10/2022).

[23] "Ultra low power gap processors — greenwaves technologies". (), [Online]. Available: https://greenwaves-technologies.com/low-power-processor/ (visited on 10/2022).

[24] "Vicon". (), [Online]. Available: https://www.vicon.com/ (visited on 10/2022).

[25] "Aerostack 2 - github". (), [Online]. Available: https://github.com/aerostack2 (visited on 10/2022).

[26] "Ignition gazebo". (), [Online]. Available: https://gazebosim.org/docs/citadel (visited on 10/2022).

[27] Y. Sun, L. Zheng, Y. Yang, Q. Tian, and S. Wang, "Beyond part models: Person retrieval with refined part pooling (and a strong convolutional baseline)", *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11208 LNCS, pp. 501–518, Nov. 2017, ISSN: 16113349. DOI: 10.48550/arxiv.1711.09349. [Online]. Available: https://arxiv.org/abs/1711.09349v3.

[28] J. Rutinowski, C. Pionzewski, T. Chilla, C. Reining, and M. T. Hompel, "Towards re-identification for warehousing entities - a work-in-progress study", *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, vol. 2021-September, 2021, ISSN: 19460759. DOI: 10.1109/ETFA45728.2021.9613250.

[29] J. Rutinowski, T. Chilla, C. Pionzewski, C. Reining, and M. ten Hompel, "Pallet-block-502 – a chipwood re-identification dataset", Sep. 2021. DOI: 10.5281/ZENODO.6353714. [Online]. Available: https://zenodo.org/record/6353714.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-December, pp. 770–778, Dec. 2015, ISSN: 10636919. DOI: 10.48550/arxiv.1512.03385. [Online]. Available: https://arxiv.org/abs/1512.03385v1.

[31] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger", *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, vol. 2017-January, pp. 6517–6525, Nov. 2017. DOI: 10.1109/CVPR.2017.690.

[32] K. Zhou and T. Xiang, "Torchreid: A library for deep learning person re-identification in pytorch", Oct. 2019. [Online]. Available: http://arxiv.org/abs/1910.10093.

[33] "Ros 2 documentation — ros 2 documentation: Galactic documentation". (), [Online]. Available: https://docs.ros.org/en/galactic/index.html (visited on 11/2022).

[34] "Understanding ros 2 actions — ros 2 documentation: Dashing documentation". (), [Online]. Available: http://docs.ros.org.ros.informatik.uni-freiburg.de/en/dashing/Tutorials/Understanding-ROS2-Actions.html (visited on 11/2022).

[35] "Using colcon to build packages — ros 2 documentation: Foxy documentation". (), [Online]. Available: https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Colcon-Tutorial.html (visited on 11/2022).

[36] "Logging groups and variables — bitcraze". (), [Online]. Available: https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/api/logs/ (visited on 10/2022).

[37] "Parameter groups and variables — bitcraze". (), [Online]. Available: https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/api/params/ (visited on 10/2022).

[38] "Controllers in the crazyflie — bitcraze". (), [Online]. Available: https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/sensor-to-control/controllers/ (visited on 10/2022).

[39] "State estimation — bitcraze". (), [Online]. Available: https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/sensor-to-control/state_estimators/ (visited on 10/2022).

[40] "Reference frames and how they are used in inertial navigation · vectornav". (), [Online]. Available: https://www.vectornav.com/resources/inertial-navigation-primer/math-fundamentals/math-refframes (visited on 10/2022).

[41] "Aerostack2/controller_plugin_speed_controller". (), [Online]. Available: https://github.com/cvar-upm-archive/controller_plugin_speed_controller (visited on 12/2022).

[42] "Esp32 wi-fi and bluetooth mcu i espressif systems". (), [Online]. Available: https://www.espressif.com/en/products/socs/esp32 (visited on 11/2022).

[43] "Flashing — bitcraze". (), [Online]. Available: https://www.bitcraze.io/documentation/repository/aideck-gap8-examples/master/getting-started/flashing/ (visited on 10/2022).

[44] "Ai deck in client mode (connect to a wifi hotspot) bitcraze". (), [Online]. Available: https://github.com/orgs/bitcraze/discussions/151 (visited on 10/2022).

[45] "Ai deck esp32 firmware". (), [Online]. Available: https://github.com/bitcraze/aideck-esp-firmware (visited on 11/2022).

[46] "Bootload the crazyflie 2.x — bitcraze". (), [Online]. Available: https://www.bitcraze.io/documentation/repository/crazyflie-clients-python/master/functional-areas/cfloader/ (visited on 11/2022).

[47] "Customize firmware with kbuild — bitcraze". (), [Online]. Available: https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/development/kbuild/ (visited on 11/2022).

[48] "Wifi video streamer — bitcraze". (), [Online]. Available: https://www.bitcraze.io/documentation/repository/aideck-gap8-examples/master/test-functions/wifi-streamer/ (visited on 11/2022).

[49] "White balance". (), [Online]. Available: https://docs.gimp.org/en/gimp-layer-white-balance.html (visited on 11/2022).

[50] "Aerostack2/project_tfm_miguel_granero". (), [Online]. Available: https://github.com/cvar-upm-archive/project_tfm_miguel_granero (visited on 12/2022).

[51] "Aerostack2/goto_behaviour at migran-devel". (), [Online]. Available: https://github.com/cvar-upm-archive/goto_behaviour/tree/migran-devel (visited on 12/2022).

[52] "Aerostack2/goto_plugins at migran-devel". (), [Online]. Available: https://github.com/cvar-upm-archive/goto_plugins/tree/migran-devel (visited on 12/2022).

[53] "Aerostack2/python_interface at migran-devel". (), [Online]. Available: https://github.com/cvar-upm-archive/python_interface/tree/migran-devel (visited on 12/2022).

[54] "Tmux/tmux: Tmux source code". (), [Online]. Available: https://github.com/tmux/tmux (visited on 11/2022).

[55] "Aerostack 2". (), [Online]. Available: https://github.com/aerostack2/aerostack2 (visited on 12/2022).

Appendix A

Time plan

The thesis has been developed in two different locations. The beginning of the thesis, mainly documentation and programming of the AS2 Aerial Platform software were done in Madrid, Spain. Whereas the rest of the thesis was done in Dortmund, Germany. The Figure A.1 represents the Gantt chart of the time plan.

The work can be divided into the blocks or work packages in Table A.1

ID	Name	Start Date	End Date	Duration
1	First Meetings and Research	abr 21, 2022	may 09, 2022	19 days
2	Crazyflie Platform Programming	may 10, 2022	jun 09, 2022	31 days
6	AS2 Modifications	jun 10, 2022	jun 24, 2022	15 days
8	Total Integration	ago 06, 2022	ago 24, 2022	19 days
3	Use-case Research	jun 08, 2022	jun 29, 2022	22 days
7	AI Deck	jun 30, 2022	jul 15, 2022	16 days
5	Integration of AI model and AI Deck	jul 18, 2022	ago 05, 2022	19 days
9	Testing and debugging	ago 25, 2022	sept 16, 2022	23 days
10	Autonomous Mission programming and testing	sept 17, 2022	sept 25, 2022	9 days
11	Experiments	sept 26, 2022	oct 14, 2022	19 days
12	Thesis Document	oct 15, 2022	dic 13, 2022	60 days
13	Review	dic 14, 2022	dic 17, 2022	4 days

Table A.1 Time plan.

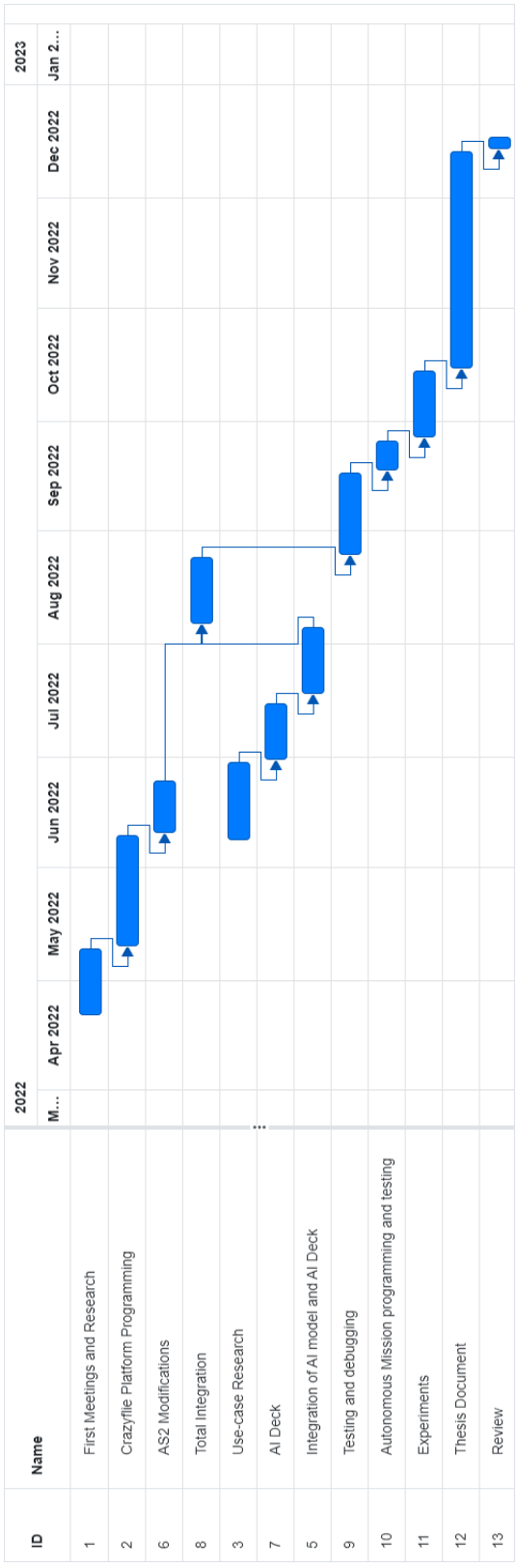


Figure A.1 Gantt diagram.

Appendix B

Budget

The approximated total cost of this project (excluding electricity and infrastructure maintenance) can be summarised as follows. Firstly, the engineering hours:

$$400h \times 23 \text{ €} /h = 8,280.00 \text{ €}$$

Secondly the amortisation cost of the equipment used for the thesis. In this case, just a laptop computer:

$$1100 \text{ €} /96 \text{ months} \times 6 \text{ moths used} = 68.75 \text{ €}$$

Name	Cost
Crazyflie 2.X	232.00 €
AI Deck	285.00 €
RGB Camera for AI Deck	21.00 €
Crazyradio	36.00 €
JTAG programmer	20.00 €
Battery	7.00 €
Batttery charger	10.00 €
Motion Capture Markers	10.00 €
PC	68.75 €
Egineering Hours	8,280.00 €
Total	8,969.75 €

Table B.1 Total costs.

The Table B.1 adds does not contain the costs of a Vicon or motion capture system. It can vary a lot depending on the custom installation needed, number of cameras, etc. It can be around 20,000.00€ on average.

Therefore the cost of the thesis is **8,969.75 €.**

Appendix C

Abbreviations

UPM Universidad Politécnica de Madrid

TUD Technische Universität Dortmund

CAR Centro de Automática y Robótica - Centre for Automation and Robotics

CVAR Computer Vision and Aerial Robotics

FLW Lehrstuhl für Förder- und Lagerwesen

RGB Red - Green - Blue

ROS Robot Operative System

AS2 Aerostack 2

UAV Unmanned Aerial Vehicle

SDK Software Development Kit

GPS Global Positioning System

PID Proportional - Integral - Derivative

WiFi Wireless Fidelity

API Application Programming Interface

ID Identification

RFID Radio Frequency Identification

IMU Inertial Measurement Unit

EKF Extended Kalman Filter

FPS Frames per second

PCB Part-based convolutional Baseline

RPY Roll - Pitch - Yaw

DToA Differential Time of Arrival

Figures Index

Tables Index