Where is the Flapper?

An Evaluation Of The Lighthouse and Loco Positioning Systems on The Flapper Nimble+

Fleur van Teijlingen

Layout: typeset by the author using LaTeX.
Cover illustration: Flapper Drones
Further images (unless specified) are made by the author

Where is the Flapper?

An Evaluation Of The Lighthouse and Loco Positioning Systems on The Flapper Nimble+

Fleur van Teijlingen
15269477

Bachelor thesis
Credits: 18 EC

Bachelor *Kunstmatige Intelligentie*

University of Amsterdam
Faculty of Science
Science Park 900
1098 XH Amsterdam

Supervisor
Dr. A. Visser

Informatics Institute
Faculty of Science
University of Amsterdam
Science Park 900
1098 XH Amsterdam

Semester II, 2025-2026

Abstract

The Flapper Nimble+ is a bio-inspired flapping-wing flying robot that runs on similar firmware as the Crazyflie quadcopter and relies on external positioning systems for controlled flight. The company Bitcraze provides two lightweight and relatively low-cost positioning systems for Crazyflie: the Lighthouse Positioning System, based on infrared base stations, and the Loco Positioning System, based on Ultra-WideBand (UWB) radio. Although both systems are technically compatible with the Flapper, the Loco system is still considered experimental on this platform, challenged by the unique flapping-wing dynamics of the Flapper. This thesis aims to evaluate the performance of the Lighthouse and Loco systems on the Flapper by comparing the position estimates against a ground truth measured by the motion capture system OptiTrack. To determine the general accuracy, the Flapper performs a series of controlled experiments including static hover and more complex trajectories such as circles and helices. The 3D position error is computed using the Euclidean distance between the coordinates gathered by each system and OptiTrack. To limit the influence of the particular setups, the Crazyflie 2.1 Brushless is subjugated to the same series of experiments.

The results show that Lighthouse consistently outperforms the Loco system on both the Flapper and Crazyflie, with typical mean absolute 3D errors ranging between 10-20 centimetres, compared to an accuracy of approximately 30-40 centimetres for Loco system. The Lighthouse showed stable flight behaviour without noticeable drift, but its accuracy is substantially worse than the 2 cm reported in the literature, mainly due to imperfect time synchronisation with OptiTrack and challenging environmental conditions such as strong sunlight. The Loco system, in contrast, struggles with altitude control, likely caused by multipath effects. The comparison between the Flapper and the Crazyflie using the Loco system reveals no major performance gap, suggesting that the system has been sufficiently tuned to the Flapper's flight dynamics. Overall, the findings indicate that Lighthouse is currently the more suitable positioning system for accurate Flapper flight, while Loco can provide more rough positioning and requires improved altitude estimation and sensor fusion before it can support high-precision control.

Contents

Chapter 1

Introduction

The effortless flight of birds and insects has always fascinated mankind and has been an inspiration to many engineering marvels, including Flapping Wing Micro Aerial Vehicles (FWMAVs). These are small drones which can take flight due to artificial wings. An example of these FWMAVs is the Flapper Nimble+[1]. The Flapper Nimble+ is a tailless bio-inspired drone with soft flapping wings that imitates insect-flight. Originally the Flapper was called the Delfly, developed by the Technical University Delft in the Netherlands [11]. After commercialization it was introduced to the world as the Flapper.

The main motivation of developing FWMAVs is the potential they have to be safer than the larger drones [19]. Their size makes them more accessible and it creates the possibility to have the vehicles operate in swarms. The Flapper is not just safer because of its size, but also because of it specific hardware. It has soft wings instead of hard-spinning propellers as drones usually have, enabling the drone to handle environments where vulnerable objects like people or plants are involved. This improved safety was one of the reasons the project of the Delfly started [11].

Besides safety, the TU Delft set out to create a drone that imitates insect-flight to research these specific flight dynamics. Specifically the Flapper can accurately mimic the rapid escape manoeuvres of flies, making it more agile than most drones.

While the Flapper was developed by the TU Delft, its firmware was created by the company Bitcraze[2]. Bitcraze is known for their creation of the Crazyflie, a quadcopter: a drone with four motors and propellers. Their firmware is therefore first and foremost suited for quadcopters, which the Flapper evidently is not. Modifications have been made to suit the Flapper's needs, but the control and flight dynamics are very different than those of the Crazyflie, thus as the company states themself 'further contributions are needed to unlock the full potential of the Flapper' [12].

One of the areas in need of improvement is the Flapper's ability for autonomous flight. Autonomous flight is the capability to fly according to on-board control systems without the influence of a human pilot or remote control. For this to work, the drone is in need of knowledge of its own location and the surroundings, which is what positioning systems provide.

Positioning systems compute an estimate of the state of a drone. The positioning is the place of an object in a space, meaning the XYZ-coordinates. In combination with internal IMU sensors, this enables the drone to fly accurately.

The Crazyflie is compatible with several of these positioning systems: the Lighthouse Positioning System, the Loco Positioning System and the Motion Capture System. Both the Lighthouse and Loco systems compute their positioning on-board through a respectively-named deck directly attached to the drone, referencing the space using external beacons, which use either infrared light or radio waves. The Motion Capture System is an off-board positioning system, where cameras using infrared locate the drone and an external host computer calculates the position.

[1] `https://flapper-drones.eu/nimbleplus/`
[2] `https://www.bitcraze.io/`

Flapper drones [4] states that the Flapper is 'compatible with various positioning systems (Lighthouse, Optical Motion Capture, experimentally UWB) for precise computer controlled indoor flight', meaning the Loco Positioning system (which uses Ultra-Wide Band Radio) is open for improvement. While Bitcraze mentions all systems are equally easy to use due to their attachable decks, they do warn about weaknesses for the Flapper per system. The Loco Positioning System struggles with position estimation, and the Lighthouse and Motion Capture systems require line of sight, which can be blocked by either the wings or the tilts of the Flapper, decreasing performance.

To more effectively improve these positioning systems, it is vital to investigate their current weaknesses and compare one with another. Using the Motion Capture System OptiTrack as ground truth, the Lighthouse System and the Loco System are used to fly the drone and make it perform certain benchmarks, such as static flight or flight under a certain speed. The estimated coordinates will then be evaluated by the global coordinates obtained with the OptiTrack, using Mean Squared Error to evaluate their accuracy. Other weaknesses will also be explored, ranging from occlusion to inference.

Chapter 2

Theoretical Background

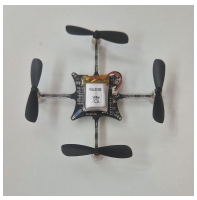
2.1 Micro Aerial Vehicles

Micro Aerial Vehicles (MAVs) are autonomous, unmanned air vehicles (UAV). They are essentially a smaller version of the typical drones. This reduction in size has generated interest since it makes it easier to deploy the drones in smaller or vulnerable spaces [15]. One of the main motivations to make MAVs is their increase in safety and their ability for swarm operation.

The term MAV was originally defined by the Defense Advanced Research Projects Agency (DARPA) as drone with a dimension of maximum 15 centimetres [18]. However, as the need increased for drones to be autonomous, so did the size. While it is technically possible for an autonomous MAV to fit DARPA's definition (e.g.the Black Widow [6]), making the MAV bigger creates more possibilities for add-ons. Autonomous drones like the Carolo, the first fixed-wing MAV [2], and the first flapping-wing MAV, the Delfly Explorer [3] broadened the definition. Nowadays MAVs are more likely categorised as lightweight drones, that can be launched by hand or fit in the palm, and can easily be operated in human-centric environments.

2.1.1 Crazyflie

One of the important MAVs for this research is the Crazyflie. Crazyflie is an umbrella term for a series of quadrotor helicopters (quadcopters), drones which derive lift from four separate rotors each oriented about a vertical axis[1], produced by the company Bitcraze. Each rotor is powered by its own thruster. With the first Crazyflie, the Crazyflie 1.0 (Figure 2.1a), manufactured in 2010 [1], Bitcraze AB company sought to create an open-source flying development platform 'made by developers for developers' [5]. The platform is now used for research and education in robotics and control engineering. The current versions in production are the Crazyflie 2.1+ and the Brushless.



(a) The Crazyflie 1.0

(b) The Crazyflie 2.0

(c) The Crazyflie 2.1 Brushless

Figure 2.1: The Crazyflie by Bitcraze AB

The Crazyflie falls under the category of Micro Aerial Vehicles due to its small size. The distance between motors ranges from 90mm to 92mm (depending on the version), making it easy to fit on

[1] `https://www.merriam-webster.com/dictionary/quadcopter`

the palm of your hand. The Crazyflie is lightweight with the most recent version, the Crazyflie 2.1 Brushless (Figure 2.1c), being only 32 grams (no accessories)[2].

Control

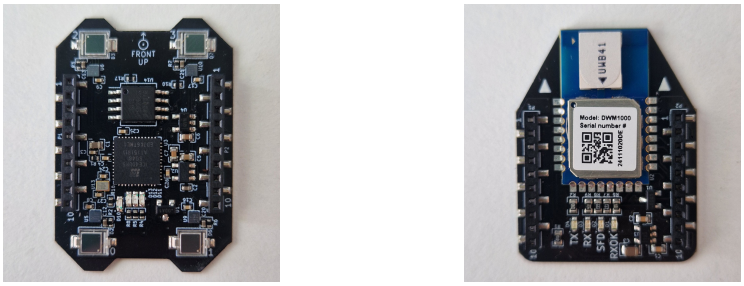
The Crazyflie is equipped with low-latency/long-range radio as well as Bluetooth LE. In combination with an USB-dongle, Crazyradio 2.0 (Figure B.2), it is possible to control the Crazyflie with a computer or a gamepad or use scripts to fly the drone autonomously. The drone can even be flown using a phone with Bitcraze's app, Crazyflie Client.

To successfully install the Crazyflie, Bitcraze's own GUI Application CFclient (Appendix D.1) is used, from which one can also control the drone. The latest firmware can be updated or the customers own software. Any insight to the parameters and fine-tuning can be done through this platform as well. Additionally, after installing a positioning system, the GUI also enables control for flying the drone. Positioning systems are used for absolute positioning in the coordinate system. For ordinary flight the Crazyflie uses Inertial Measurement Unit (IMU) sensors like an accelerometer and a gyroscope to stay in control. A complimentary filter uses the measurements to calculate its rotation and the flight controller uses this information to correct its flight, for example sending more power to one of the thrusters to compensate for uneven weight.

Decks

The Crazyflie is a do-it-yourself project, as you buy the loose parts and must assemble the drone yourself. It is completely wireless with a chargeable LiPo battery, giving the drone 10 minutes of flight when it's fully charged. One of the greatest strengths of Crazyflie is its compatibility with a lot of plug-and-play expansion decks[3]. It enables the Crazyflie with extra abilities. Decks can be combined by using multiple pins, although not all decks are compatible. Certain decks like the lighthouse and loco positioning systems use the same pins, requiring a different hardware setup before they can be used together. Additionally, there is no dedicated user interface for this setting; it must be configured manually in the drone firmware using Bitcraze's kbuild-based build system intended for developers.

One of Bitcraze's decks is the Flow deck, which provides state estimation for the drone by a small camera which can be attached to the bottom side. This is a simple form of positioning for the drone so it can see the distance to the floor, calculating its height, and giving the ability to fly on a controlled altitude. However, due to the vibrations caused by the flapping of the flappers wings, camera footage is unstable and highly unreliable for accurate positioning. For this research the Lighthouse Deck (Figure 2.2a) and Loco Deck (Figure 2.2b) are of importance, since they able the drone the use of the Lighthouse Positioning System (Chapter 2.2.1) and Loco Positioning System (Chapter 2.2.2). These decks need to be combined with an outside structure to help the Crazyflie estimate its position. All calculations are done inside the deck which is provided with firmware.



(a) Lighthouse Positioning Deck (Bitcraze) (b) Loco Positioning Deck (Bitcraze)

Figure 2.2: Positioning Decks by Bitcraze AB

[2] https://www.bitcraze.io/documentation/hardware/crazyflie_2_1_brushless/crazyflie_2_1_brushless-datasheet.pdf

[3] https://www.bitcraze.io/documentation/system/platform/cf2-expansiondecks/

2.1.2 Flapper Nimble+

The main MAV of this research is the The Flapper Nimble+ (Figure 2.3). The Flapper is a tailless flying robot. It is a Flapping Wing Micro Aerial Vehicle (FWMAV), a subcategory of MAVs. It distinguishes itself by its wings which enable flight, an attribute that also sets it apart as an ornithopter instead of a quadcopter like the Crazyflie. The design of the Flapper is bio-inspired, meaning it was inspired by the flight of birds and insects. Originally called the Delfly, it was brought to life at the TU Delft [11]. Afterwards the Flapper Nimble+ was created: a slightly bigger version of the Delfly which could be produced commercially [20]. The Flapper is interesting for the study of the flight mechanics of birds and insects. Flappers can also used for drone shows and can be flown in swarms of multiple Flappers.

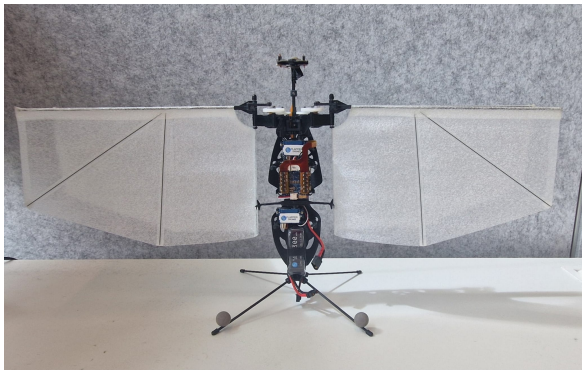


Figure 2.3: Flapper Nimble+

The Flapper has four wings - inspired by dragonflies - which improve stability. These wings flap back and forth horizontally 12 times every second [4]. The total wingspan is approximately 45 centimetres. The Flapper is about 25 centimetres tall. On-board there are gyroscopic sensors to keep the Flapper balanced, which update the wing motion parameters 500 times every second. The Flapper can fly in any direction, making it similar to a humming bird. The soft material of wings limits damage to surroundings during collision.
The Flapper works with the same firmware as Crazyflie, making the control similar. Just like the Crazyflie, the Flapper can be flown manually or with a pre-defined script. However, not not all functions and decks can be easily used, given the fact the flight dynamics differ much of the Crazyflie's. The specifics can be found in the Flapper's compatibility page[4]. Currently the company of Bitcraze has multiple decks available for the Flapper, but not all work as well as they should. Due to the flapping nature of the Flapper, Flappers can appear wobbly and unstable, making on-board camera work harder. Decks like the Flow deck are therefore impossible to use. The Lighthouse Deck is compatible, but only when an extra standard is added to the top of the Flapper. The Loco deck can also be used, but the position estimation is not yet optimal and is still experimental according to the company behind the Flapper[5].

[4]Flapper Drones compatibility: `https://docs.flapper-drones.eu/lib/exe/fetch.php?media=nimbleplus:flapper_nimble_deck_compatibility.pdf`

[5]`https://flapper-drones.eu/nimbleplus/`

2.2 Positioning Systems

Positioning systems are methods used to define the position of an object in a space. Drones are pushed to move more quickly and more efficiently. Control by humans can be limiting. The need for autonomy for drones increases, and for that to happen drones need to be self aware of their position. This is even more important for when drones need to interact with one another. Thus there is a great need for good positioning systems to make this happen.

For drones like the Flapper positioning is important, because awareness of its own location compared to the space it is located can significantly improve its flight.

Positioning systems make use of a variance of techniques. There are systems that make use of base stations sending and receiving signals in communication with objects as a way to find the position. There are also optical systems that use cameras to locate an object. Inertial Navigation Systems (INS) use environmental sensors or Inertial Measurement Units (IMUs), such as a gyroscope or accelerometers to measure the change of motion in an object [8, p.118]. Every system has its own accuracy and coverage, and additional advantages and disadvantages making it (un)suitable for different situations. There is a distinction made between indoor and outdoor positioning systems.

Outdoor positioning systems

Outdoor positioning systems are used to track objects or people outside. These systems are frequently used on a large scale, even globally. A very well-known outdoor positioning system is the USA's Global Positioning System, also known as GPS. This system falls under the Global Navigation Satellite Systems (GNSS). Other examples of GNSS are Europe's Galileo, Russia's GLONASS and the Chinese's BeiDou [14]. GNSS requires direct line-of-sight with the sky, since positioning calculation is done through signals send to satellites. This results in failings or decreases in accuracy indoor because of blockades like walls that interfere with the signal, making it unsuitable for indoor use.

More regional uses of outdoor positioning systems would be the tracking of phones through the signals send from and to cell towers.

Indoor positioning systems

Indoor positioning systems (IPS) are used when objects appear inside. Indoor positioning is on a smaller scale than outdoor, like positioning in one building. These systems make use of technologies like Bluetooth, Ultra Wide Band Radio (UWB Radio), Wi-Fi and Infrared to localize the objects. Local Positioning Systems are part of IPS, but can be meant for even a smaller scale, like one specific room. While Bluetooth has a high coverage and an accuracy up to ten centimetres, radio inference is a killer. UWB also has high coverage and high accuracy, but the signal can be distorted by liquid or metal surfaces.

For this research there will only be focus on indoor positioning systems due to the added difficulties of outside flight for the Flapper. The systems under review are the Lighthouse and Loco Positioning Systems by Bitcraze. Additionally, the Motion Capture System OptiTrack will be used for ground-truth comparison. Further information on the systems will be provided in their respective chapters.

2.2.1 Lighthouse Positioning System

The Lighthouse Positioning System (Appendix B.2) is an indoor local positioning system. This system created for drones (specifically the Crazyflie) was developed by the company Bitcraze AB. However, they did not develop the technology behind the system. The lighthouse technology, typically known as SteamVR tracking, is a 3D motion capture system, originally with applications for Virtual Reality [21].

Bitcraze's Lighthouse system consists of two parts. The first part is multiple external beacons for reference. These beacons are called base stations, which emit laser beams consisting of infrared light. The infrared beams are received by the second part: the Lighthouse deck. The lighthouse deck has four micro light-receivers (photo-diodes), which catch the light. The Lighthouse deck is attached to either the Crazyflie or the Flapper.

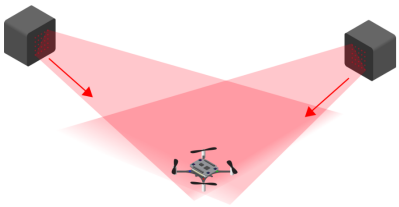


Figure 2.4: Lighthouse Positioning System [16]

Because this is an optical-based system, it is vital the lighthouse deck has a direct line-of-sight to the base stations. For the Crazyflie this means the deck can directly be attached to the top. For the Flapper the mounting board is only attached to one side, meaning there is need for an extra structural mount attached to the top of the Flapper. To connect the deck, one can use Bitcraze's extension cable specifically designed for the lighthouse deck.

Once the deck is connected to the board, the deck needs to be configured to the proper platform through the GUI CFclient. This step is important, because the deck is compatible with both platforms, and the settings are different for both of them. The Flapper won't show optimal performance if the settings are set to that of the Crazyflie.

For calibration of the hardware stations, the CFclient is also required. Bitcraze provides a step-by-step guide for this purpose. A plus for the lighthouse station is that the distance of the base stations doesn't need to be measured by hand. Providing few movements of the drone, the base stations are able to figure out their own location in the room referencing to the 0-coordinate provided by the user and can create a 3D-coordinate system for the drone to use. Beforehand, the base stations must first be configured and set to different channels. Afterwards the deck can distinguish the stations by their channels. The deck can calculate the position using the Angle of Arrival of the lasers.

There are two versions of base stations compatible with the Lighthouse deck. The earliest version was the Lighthouse Base Station V1. Bitcraze no longer offers these in their shop, but the system is still compatible. The system works with 1-2 stations, each with a range of approximately 6 metres. The front of the stations is flat and the stations have a 120 by 120 degrees view. The V1 station uses two rotating drums. Compared to the V1, the V2 base stations use only one rotating drum with two slighted planes on top. The front of the station is curved. This creates a horizontal angle view of 150 degrees, and a vertical view of 110 degrees. This system also supports typically supports a maximum of 4 stations, but it can be configured inside the firmware to accept up to 16 stations. While one base station should be sufficient, a minimum of two creates better performance. The versions of base stations are not compatible with one another. While the lab is in possession of the four V1 statins, Bitcraze currently only provides access to the V2 base stations, suggesting this is the improved version. Therefore for this thesis only V2 base stations are used.

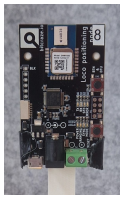
There are two positioning methods. One is the crossing beam method. This is where the light sweeps must intersect (or close to) to compute the position. The algorithm then calculates this position which is put into a Kalman estimator (Chapter 2.2.4), which further estimates the current state of the drone. This method requires at least two stations.

The second and preferred method is the raw sweep. The base geometry and the angle on which the deck received the laser are sent directly to a Kalman estimator. The advantage of this method is the platform needs only one station, not two.

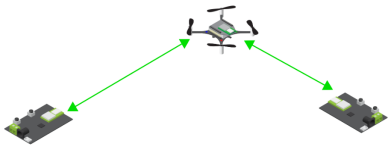
The Lighthouse Positioning System is compatible with both the Crazyflie and Flapper, and is claimed to have an accuracy in the millimetres [22]. According to Bitcraze (2026) the maximum area covered is 5x5x5, but environments with a lot of sunlight, glass or mirrors can limit this, with sunlight causing extra infrared and glass and mirrors reflecting the light.

2.2.2 Loco Positioning System

Bitcraze's Loco Positioning System (Figure 2.5b) is also an indoor local positioning system. Unlike the optically-based Lighthouse system, the Loco system estimates positions using radio waves. The system makes use of Ultra Wide Band (UWB) Radio. This is a type of communication system that requires two or more devices equipped with UWB chips to communicate with one another. For Bitcraze's Loco system, this system consists of two sorts of devices: a Loco Positioning Deck, also referred to as a Tag (Figure 2.5c), which is directly attached to the drone platform, and a set of Anchors (Figure 2.5a) which can be positioned all throughout the room that serve as a reference [17]. The anchors and the tags all have their own timer.



(a) Anchor



(b) Loco Positioning System (Bitcraze)

(c) Tag

All anchors must be configured using the LPS configuration tool and be given an unique ID. Once that is done, they can be placed inside a room in different locations. Depending on the chosen mode, the amount of anchors can differ between 4 and an unspecified amount, although a maximum of 8 anchors is consistently advised. To set up the coordinate system, the location of all anchors must be measured precisely by hand and each anchor must be provided with a power source. The anchors must each be provided in Bitcraze's CFclient with their 3D-distance from the appointed origin. Afterwards, the anchors will start to communicate with one another and the tag. The communication style is dependent on the method of use[6], but will usually depend on the Time of Flight (ToF), meaning the time it takes for a signal to travel through space.

Two-way Ranging

Two-way ranging (TWR) is a positioning method using a two-way communication protocol. There are four messages, two sent by the anchor and two sent by the tag[7]. By gathering the timestamps from both parties after receiving the signal, it allows the tag to measure its distance to an anchor. The system requires a minimum of four anchors for 3D state estimation, but six to eight anchors is better for adding redundancy and accuracy. This mode is described by Bitcraze as the most accurate, but the system is limited to using only one tag in the coordinate space, making swarm operation impossible. This mode does allow the tag to range outside of the anchor space.

Time Difference of Arrival

Another positioning method is the Time Difference of Arrival (TDoA). This mode is a one-way communication style, where packets are sent continuously by the anchors. The tag receives these packages, and can calculate the relative distance to two anchors by using the difference in the time of arrival. With this information, 3D calculation of the position becomes possible. The difference with TWR is that the tag listens only passively, making it possible to have more tags in the anchor space. However with the TDoA of Bitcraze, the accuracy of the state estimation decreases as the drone threads out of the anchor space.

There are two versions of TDoA[8]. TDoA2 is the first official version of TDoA, and this mode works best with eight anchors. With this configuration, the accuracy is comparable to TWR according to Bitcraze. TDoA3 is similar to TDoA2, only this mode allows for more anchors that also can be placed in multiple rooms. The main difference is that TDoA2 uses scheduled transmission with a master anchor, while TDoA3 anchors work with randomized transmissions to avoid that single dependency. TDoA3 can also be modified for a longer range, but the platform must be updated

[6] https://www.bitcraze.io/documentation/system/positioning/loco-positioning-system/

[7] https://www.bitcraze.io/documentation/repository/lps-node-firmware/master/protocols/twr-protocol/

[8] https://www.bitcraze.io/documentation/repository/lps-node-firmware/master/functional-areas/tdoa2-vs-tdoa3/

accordingly too.

Since the accuracy of the systems is practically the same, the difference is in how it is used. The space for the experiments in this thesis is not ideal, because it limits placement of anchors on the floor space. The advice for TDoA is that the anchors must be placed in the corners of the room for good coverage and geometry. For TWR this isn't necessary, which leaves more possibilities for anchor placement. Additionally, there is only the possibility for one flying platform at the same time due to limited decks/drones, thus multi-tag capability is irrelevant. TWR is also recommended as starting point, because configuration is easier. All those reasons is why for this thesis the anchors are configured to the TWR mode.

2.2.3 Motion Capture System

Motion capture systems use the technology motion capture (Mocap), which is a technology that records the movement of objects or people, thus 'capturing motion'. Mocap has all sorts of applications, such as virtual reality, sports science, biomechanics, animation, and robotics. These systems are typically divided into four categories.

Inertial systems
Inertial systems use Inertial Measurement Unit (IMU) sensors, which includes gyroscopes and accelerometers that measure rotation and acceleration respectively. These sensors are attached to the object or the person that performs the movement. An sensor-fusion algorithm computes each sensor's 3D orientation, negating the need for cameras. However, because of the lack of an external reference, the system tends to develop a drift in its position estimation. While the estimated rotation can remain accurate, the accuracy of absolute positioning typically decreases over time. Absolute positioning is crucial for the evaluation of the Lighthouse and Loco system, thus making the Inertial system inadequate as a ground truth. Adding a sensor to the drone would also be disadvantageous, given that every added weight is an extra strain on the battery.

Markerless systems
Markerless mocap systems use standard cameras which capture the movement of the subject. There are no markers on the subject or any other form of wearable hardware, creating a more natural state for the subject. The system estimates positioning with computer vision algorithms detecting the body keypoints and machine learning models computing the movement afterwards. This technology creates the ability to analyse movements of objects and people long after the videos were first recorded. Although, it is important to note that for more accurate (3D-)position estimation, the subject should be recorded from multiple angles.
The accuracy for position estimate is lower than for other types of mocap systems. Markerless systems perform significantly worse with self-occlusion or fast and complex movements. The movement of a drone is typically complex and fast, especially the Flapper, with rapid changes in positioning and orientation. It is hard for the system to accurately track its flight pattern. This flaw makes the markerless system unsuitable for this thesis.

Optical Passive
Optical passive mocap systems use cameras that can emit and detect infrared light. The cameras generate these infrared strobes in time with their shutters. These strobes of light are reflected by retro-reflective markers placed on the subject, allowing the system to track the subject's motion. These retro-reflective markers are typically small, lightweight spheres that are cased in reflective material. There are also reflective stickers, but they have as disadvantage that they are not 360 degrees visible. The spheres are placed on body key-points. To be able to calculate the rotation, they are usually placed asymmetrically. However, for lightweight drones like the Flapper, an unfair distribution of weight means the motors must compromise for this difference, which decreases the battery's life and can thus be unfavourable. An example of this type of mocap system is the OptiTrack (Figure 2.6).

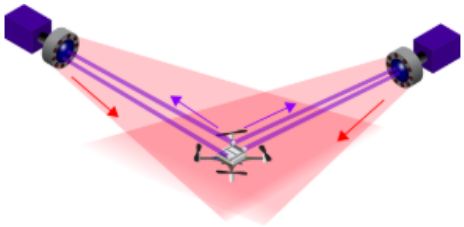


Figure 2.6: OptiTrack

Optical Active
Contrary to the passive optical system, there is also the active optical mocap system. Just like the passive system, this system uses cameras. However, these cameras are passive themselves, meaning they only receive infrared light, not emitting it. The light is generated by small LEDS placed on the subject. The LEDs are attached to the subject of the movement, and each broadcasts a specific signal, which enables the system to calculate position and rotation. While this leads to a more accurate positioning, the small LEDs themselves are an increase in weight and to work they also require an extra battery placed on the drone, which is another increase in weight. For drones like the Flapper every increase of weight is an instant decrease in the life of the battery.

As noted before, each system has its own unique flaws. To function the inertial and optical systems all require extra hardware to attach to the drone, adding weight to an already sensitive object. Contrarily, the markerless system does not need this, but this system has difficulty capturing fast movements. The optical systems have the highest accuracy. Although the passive optical system still requires extra weight, the spheres to be added are lightweight. In a trade off with the accuracy, this is the best option.

2.2.4 Kalman Filter

Positioning isn't merely a system looking at a subject and thereby localising it. What happens when there is a type of occlusion, what happens when the system can't see the object any more? The object doesn't seize to exist, thus the positioning must still be computed. And how do positioning systems calculate their positioning for autonomous flight? One way to do this is with a Kalman filter.

The Kalman filter is an algorithm used to estimate the state of a linear dynamic system from a series of measurements observed over time, including statistical noise. The state, in this case, refers to the position and rotation with the dynamic system being the drone. With the Kalman filter, the positioning of the drone is calculated, meaning the XYZ-coordinates, the yaw, the roll and the pitch. While in real life time is continuous, computers divide time into discrete steps, using the Discrete Kalman Filter.

The (Discrete) Kalman Filter was invented by R.E. Kalman back in 1960 [10]. His paper described an recursive solution to the discrete-data linear filtering problem. This referenced the difficulty of estimating a moving object's true state while only having noisy measurements. To solve this problem, Kalman came up with an equation which can essentially be reduced to two steps: prediction and update/correction [13].

The prediction is the *a priori* $\hat{x}_k^-$ estimate of the state at step$_k$ given the knowledge prior to step$_{k-1}$. The update is the *a posteriori* $\hat{x}_k$ estimate of the state at step$_k$, with measurement z$_k$ [25]. At each step, the filter first predicts the next state by using physics to compute the position. If the drone is now at this position in space, and it moves at this speed for a given duration, the drone will now be at this other position in space. To calculate this, the filter takes into account the pure mechanics of the drone (A_k) times the state (x_k), the control-input matrix (B_k) times the motion input (u_k) and the noise (w_k) as seen in the system(Equation 2.1).

$$x_{k+1} = A_k x_k + B_k u_k + w_k \tag{2.1}$$

Next the state generates a measurement through the sensors (Equation 2.2). z_k is the measurement, modelled by the measurement matrix (H_k) times the state (x_k) plus the measurement noise (v_k).

$$z_k = H_k x_k + v_k \tag{2.2}$$

Then the correction is performed by comparing the measurement and the predicted measurement to each other, which will lead to the new updated state (Equation 2.3).

$$\hat{x}_k = \hat{x}_k^- + K(z_k - H_k \hat{x}_k^-) \tag{2.3}$$

Extended Kalman Filter
The basic Kalman filter has two strong assumptions. The first being that the world is Gaussian. This means that the assumption is that all noise behaves under a normal distribution, thus it can be modelled as a bell-shaped curve. The second assumption is that models are linear. This means all variables behave as a straight line. However, in real life models are rarely linear. For a drone like the Flapper this is also not the case. To overcome this non-linearity, the Extended Kalman Filter (EKF) was invented. The non-linear dynamic is linearized by using Taylor series [13]. This filter is also used in the Crazyflie and the Flapper when positioning systems are attached. Without these systems the drones fly with a Bitcraze's Complementary filter, which essentially exists of the input of the IMU sensors.

Unscented Kalman Filter
While the EKF was an improvement towards the basis Kalman Filter, that filter struggles when the dynamic system becomes more complicated. If the assumptions of local linearity are violated, the EKF will start to drift, losing its accuracy in the state estimation. As a solution, the Unscented Kalman Filter (UKF) was invented [9]. Instead of linearizing an non-linear system, the inventors chose to approximate the probability distribution. Although this filter is technically available on the Crazyflie and the Flapper, Bitcraze mentions this filter is still in the 'experimental phase'[9], and so it needs to be manually enabled inside the code itself. Because of its experimental nature, it felt unfit to use the UKF for this thesis, given the fact one of the positioning systems itself is already deemed experimental.

[9] `https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/sensor-to-control/state_estimators/`

Chapter 3

Methodology

3.1 Research Questions

The purpose of this research is to evaluate the usage of the Lighthouse Positioning system and the Loco Positioning System on the Flapper Nimble+ using the OptiTrack as ground truth. This leads to the following questions:

1. How accurate are Bitcraze's Lighthouse Positioning System and Bitcraze's Loco Positioning System in positioning compared to the OptiTrack positioning?
   (a) Is there a presence of drift, meaning is there an influence of time on the accuracy?
   (b) How does an increased speed effect the accuracy of the positioning systems?
   (c) Is there a decrease in accuracy once a drone reaches the limits of the positioning geometry system?
2. What extreme conditions influence the accuracy, meaning what are the weaknesses? Occlusion, signal interference (like sunlight), distance, height, speed?

3.2 Experimental Setup

3.2.1 Location

All experiments are performed in the indoor robotics arena of the Intelligent Robotics Lab at the University of Amsterdam (Appendix C.3). The arena is situated within a larger room and is bordered by six structural support columns. The sides of the arena are enclosed by safety nets, while the relative south-side is open and the north-side is closed off by large windows. The arena itself consists of an artificial turf field resembling a small soccer pitch. The outlined soccer pitch measures approximately to a nine by six meter area. The ceiling is 3.59 metre high, with four overhead support beams running along the width of the room an height of 3.04 metre. These beams are equipped with electrical sockets, which are used to power the external positioning infrastructure.

3.2.2 Hardware

For the experiments, the platform used is the Flapper Nimble+ from Flapper Drones[1]. To enable a fair and meaningful comparison, the Crazyflie Brushless 2.1 executes the same set of flight trajectories as the Flapper using the same positioning systems. By evaluating both platforms under the same motion patterns and the same positioning setup, the resulting accuracy metrics provide a better insight in the Flapper's performance, less dependent of the specific setup configurations.

The positioning infrastructure consists of three systems: The motion capture system OptiTrack, Bitcraze's Lighthouse Positioning System and Bitcraze's Loco Positioning System. The OptiTrack system consists of eight cameras, distributed around the soccer field and pointed towards the centre

[1] `https://store.flapper-drones.eu/bundles/flapper-nimble-lighthouse-bundle-64.html`

of the field, creating a trackable space in the middle of the field, omitting the corners. Per test flights (Appendix E), this trackable space of the OptiTrack was defined to a field of 1.8 by 1.8 by 1.8. This seems to a space where the OptiTrack can track the object for most of the time, if the sun isn't too bright.

To use the OptiTrack system, four hollow retro-reflective balls are placed on the legs of the Flapper to evenly distribute the weight and keep the Flapper in balance, thus limiting the drain on the Flapper's battery. They are placed on the bottom to minimize interference with infrared signals received by Lighthouse deck mounted on top of the Flapper. For the Crazyflie, the retro-reflective balls are wedged inside the standing points belonging to the propeller guards.

The external optical beacons of the Lighthouse System are two Steam VR base stations 2.0, located diagonally from one another, mounted on the overhead beams closest to the middle of the field. The estimated geometry calculated by the base stations is shown in Table 3.1. The stations calculate the distance to the deck, causing the difference in height. The lighthouse deck is attached to the tops of both platforms. For the Crazyflie, the hardware automatically enables this. For the flapper an extra mount by design of Bitcraze is attached to the top to fit the deck. The deck is connected using a connection cable.

Type	**Station**	**X**	**Y**	**Z**	**Deck Height**
Flapper	1	1.489843	-1.713750	2.622439	± 0.255
	2	-1.833724	2.120603	2.626641	± 0.255
Crazyflie	1	1.471097	-1.723236	2.813884	± 0.035
	2	-1.796200	2.124095	2.819597	± 0.035

Table 3.1: Estimated Geometry (m) by Lighthouse Stations

The Loco Positioning system consists of eight Loco Positioning Nodes configured to UWB anchors, using Two Way-Ranging. To maintain the necessary distance of the walls, the anchors are placed on their positions raised by standards using the models provided by Bitcraze[2]. The Loco deck can directly be attached to both platforms without need for extra mounts. The Loco coordinate system can be found in Figure B.11.

3.2.3 Communication and software

All firmware and software is provided by Bitcraze AB. The Flapper is connected to an Acer PC operated by Windows using the Crazyradio 2.0. The Crazyradio was connected through the program Zadig. Through the GUI CFclient, the most recent firmware is updated to the platforms and decks. The lighthouse base stations have been configured with the same GUI, setting the base station channels respectively to 1 and 2. The Loco positioning nodes have been updated and set to UWB anchors using the LPS configuration tool provided by Bitcraze. Updating the anchors using the tool was only possible with Linux. However, configuration was available with using the tool with Windows. The anchors have been set to channels 0-7. They have been physically labelled 1-8, with 8 being aligned to channel 0. The rest of the anchors have the same channel as their label.

3.2.4 Coordinate System

All the positioning systems use their own separate coordinate systems. The OptiTrack is the most reliable system and therefore the base system. The Lighthouse positioning system and the Loco positioning system need to be transformed to match the OptiTrack coordinates. To do this, the drones are placed on three different locations: the origin, -1.5 metres on the X-axis and -1.5 metres on the Y-axis. This has to be repeated for all combinations of platforms and decks and for every day. The rotation and translation is computed by combining these three files and comparing the estimates of the positions and finding the best way to match them, resulting in a transformation matrix, applicable to the flight coordinates.

[2]`https://github.com/bitcraze/bitcraze-mechanics/blob/master/LPS-anchor-stand/anchor-stand.stl`

3.3 Experiments

To gain a diverse set of data points for evaluation, the drones are subjugated to multiple benchmarks regarding their flight. For a fair comparison between the positioning systems, the drones will take flights determined by pre-defined trajectories for both positioning systems. Each flight experiment will be repeated three times. Flights of which there is an extreme deviation of the average data points - e.g. two-hundred instead of three-thousand - will be redone. This lack of data points can occur when the OptiTrack loses track of the drone's markers, or the drone crashes mid-flight.

The first benchmark is a static hover of 30 seconds. While the system believes it remains on the same X and Y coordinates, the ground truth provided by the OptiTrack will prove whether the drone has a tendency to drift. The drone will fly on two different heights: 0.5 and 1.0 metres. While the ceilings height is 3.25 metres, it is important both the OptiTrack cameras and the Lighthouse stations can keep track of the drone. The OptiTrack has a limit of approximately 1.8 meters as discovered in the test flights (Appendix E). During test flights it also became evident the Loco Positioning System is unsteady and can't always steadily remain at a constant Z-coordinate. Choosing a maximum height of one metre leaves a buffer of 0.8 metre for the drone to drift.

The second benchmark will be a careful flight along the X-axis, the Y-axis, and the X-Y axis. This will be done on a height of 1.0 metres. The drones will fly to an X and Y of 1.5 metres. This is a distance that has been tested and seems to remain in the limits set by the OptiTrack system. These three flights will be performed with a two different speeds (0.2 and 0.5 m/s) to test control.

The third benchmark is three more complicated flight patterns (Figure 3.1). Inspiration of the flight patterns comes from [24]. The chosen flight patterns are the circle, the helix and the staircase.

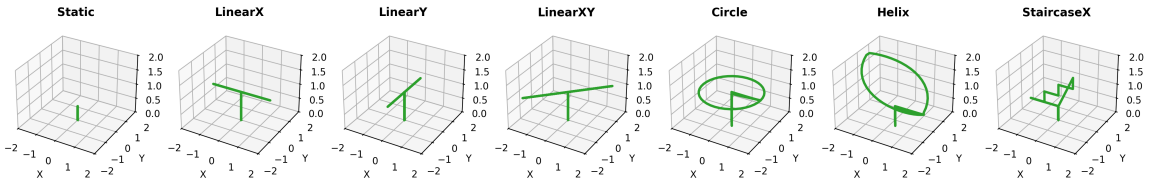


Figure 3.1: Flight Experiment Trajectories

All flights test a different aspect of the drone, as can be seen in Table 3.2. For example, the circle is a test for the reach of the XY-plane and the performance of the drone under constant yaw. The helix is a test for both the effects of the yaw and the height control under constant thrust. The staircase is a test for the XZ-plane and abrupt change in direction.

Flight	Test	Distance* (m)	(Max) Height (m)	Speed (m/s)	Total runs
Static	Endurance, Z-control	0	0.5, 1.0	0.2	2 x 3 reps
LinearX	Limits of field	1.5X	1.0	0.2, 0.5	2 x 3 reps
LinearY	Limits of field	1.5Y	1.0	0.2, 0.5	2 x 3 reps
LinearXY	Limits of field	1.5XY	1.0	0.2, 0.5	2 x 3 reps
Circle	Yaw	1.5 (radius)	1.0	0.2, 0.5	2 x 3 reps
Helix	Yaw + Thrust	1.5 (radius)	2.0	0.2, 0.5	2 x 3 reps
Staircase	(Z) control	3X	2.0	0.2, 0.5	2 x 3 reps

Table 3.2: Flight Experiment Parameters (* metre per axis)

,

3.4 Evaluation

3.4.1 Processing data

For evaluation, multiple parameters are gathered. For the drones, there are 34 parameters that are logged (Table A.1). These parameters consist of the state estimation (meaning the XYZ-coordinates and roll, pitch and yaw of the drone), the estimated speed, the drone's confidence of its estimations, the position targets, the battery and the connection to the outer hardware used for the positioning systems.

The OptiTrack gathers data of the state estimation and the confidence of this estimation per rigid body (Table A.2). The OptiTrack keeps track of the coordinates in its own separate file. To make them available for comparison, timestamps must be synchronized. The OptiTrack has its own separate time due to the base computer used instead of the computer used to direct the drone. To match the files, the start time in microseconds is noted and computed to the Unix Epoch time. Unix time system is a operating system commonly used for storing timestamps for files [23]. For each data point, the Unix Epoch Time is added to the timer in microseconds provided by the OptiTrack itself. For the files gathered by the drone, the Unix Epoch time is also added as a parameter. However, as noticed during test trials, while the Unix time is a general time for each computer, there could still be a gap up to 8 microseconds. To correct for this gap, the difference is calculated using the data from the static flights, finding the moment the drone takes off in both datafiles, and adding this difference to the Unix time of the OptiTrack. For each day this time difference is saved and used for the rest of the files of that day.

After this correction, the data points are matched by finding the closest OptiTrack time to the drone's own measured time.To match the coordinate systems, per day and for each combo of deck and platform, the transformation matrix is computed by using the three motionless files, which is then applied to the files matching the date, platform and deck. The parameters of the OptiTrack of the rotation are removed. Since the positioning systems only compute the coordinates and rotation of the drone is computed internally with IMU sensors, comparing the rotation does not add anything to this research specifically. Besides, due to the even distribution of the markers, the OptiTrack had no way of knowing the drone's real rotation, strengthening the decision of removing the rotation parameters. After removing these parameters, the rest of the file is cleared of lines with missing values. The first lines of the files before take off yet are also removed. This is checked by looking at the control target which is set to 0.0 at the start.

3.4.2 Computing accuracy

To evaluate the positioning accuracy, the positioning according to both the lighthouse system and loco system will respectively be compared to the ground-truth of the OptiTrack.
The position tracking error at time step k can be calculated with the Euclidean error:

- $e_{LH}(t) = |p_{LH}(t) - p_{opti}(t)|$ where p_{LH} is the XYZ-coordinates according to the Lighthouse System
- $e_{LP}(t) = |p_{LP}(t) - p_{opti}(t)|$ where p_{LP} is the XYZ-coordinates according to the Loco System

This formula provides the error per time stamp. p_{opti} is in both instances the XYZ-coordinates of the drone according to the OptiTrack system. For every flight and phase the following metrics are calculated:

- Mean Absolute Error (MAE): $\frac{1}{N}\sum_{i=1}^{N} e(t_i)$
- Root Mean Square Error (RMSE): $\sqrt{\frac{1}{N}\sum_{i=1}^{N} e(t_i)^2}$
- Maximum error: $|\max(e_*)|$
- Standard deviation error: $\sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(e(t_i) - \mu)^2}$

3.5 Model

For further analysis, a small XGBRegressor model will be trained on the gathered data. This is to evaluate the importance of other variables besides the time, position, date, platform and decks. The choice of model was inspired by [7]. This paper uses two models, a Random Forest and a XGBoost, for real-time error prediction. The model for this thesis is only used for feature importance analysis, thus one model seems sufficient. The choice for XGBoost was made, because of its robustness, easy use by GPU and former experience with the model. Starting from parameters as provided by [7], the model will be slightly tweaked until a sufficient R^2-score is achieved.

Chapter 4

Results

For this thesis there were four possible combinations of platforms and decks. Seven types of flight experiments have been executed for each. Each flight had two settings: the static flight experiments had a difference in height setting, and the other flight experiments had a difference in velocity setting. For reliability each experiment was repeated three times.

These different settings led to a total of 168 files, consisting of data points ranging between the thousand and six thousand depending on the duration of the flight. After removing all stationary data points (meaning the data points before take off), and removing all rows where the OptiTrack failed to capture the ground truth, the data set was left to 486.675 data points for all experiments combined. Each experiment can be visualised per three reps like Figure 4.1, where the blue line is how the drone thinks it has flown, and the black dotted line is how the drone has actually flown according to the OptiTrack data. For this example, both coordinates line up quite accurately. Neither system claims to have flown a perfect circle as the target values would suggest.

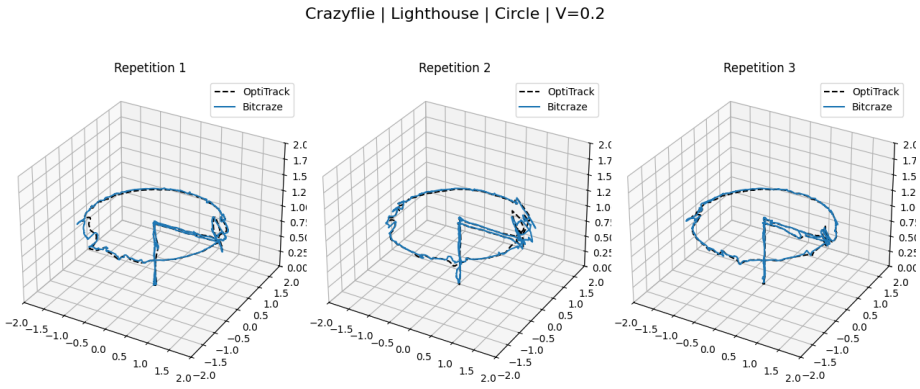


Figure 4.1: Coordinate plot - Experiment Crazyflie Lighthouse Circle 0.2 m/s

During the flight experiments, the operator made multiple optical observations, which provides some context on what can be observed in the recorded data:

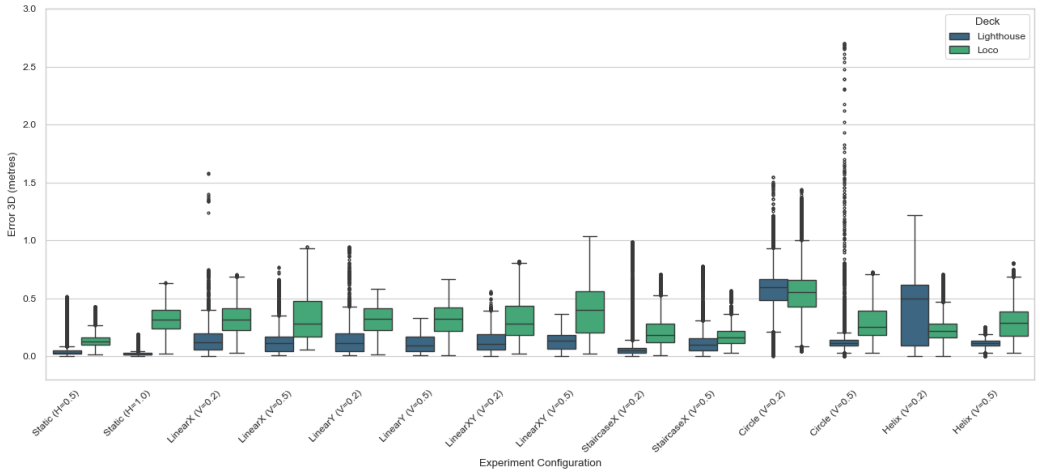
1. If the red light on the drone remains on, the battery is near empty and the drone will fall down mid-flight without any grace or control.
2. When velocity increased, the drone moved more quickly. For flight control this provided little issue, but during landing, the Flapper would tip over and the Crazyflie would bounce
3. The Crazyflie+Lighthouse showed a wiggle, especially on the negative Y-axism where the drone would temporarily lose altitude, in particular during the circle flights. The Flapper showed less of this behaviour, appearing more stable.
4. Both platforms in combination with the Loco system showed little control over the altitude, constantly changing height

4.1 General Flight Accuracy

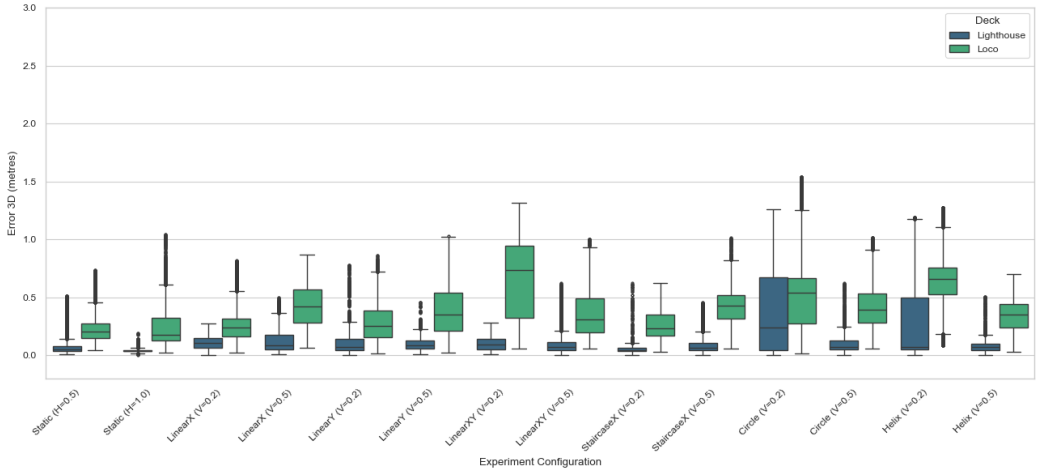
To test for accuracy under different circumstances, multiple flight patterns have been performed by both the Flapper and the Crazyflie. The prime indicator of positioning accuracy is the 3D-positioning error, taking in account axes X, Y and Z. In Figure 4.2 the range of this error has been visualised, segregated by platform, deck, experiment and specified height/velocity. Specific numbers can be found in Table A.4. For

The Crazyflie shows a slight variable mean 3D error per experiment for both decks. With the Lighthouse deck, the drone shows a consistently low error with a few spikes in the circle and helix experiments. However, eleven out of fourteen experiments show a high outlier density, as indicated by the thick black line at the top of the box plots. In comparison, the Loco system exhibits outliers in ten out of fourteen experiments, but these outliers are clustered more closely together. In general, the Lighthouse error distribution is more compact, but includes more extreme outliers, whereas the Loco system shows a more consistent error distribution with fewer very large deviations.

For the Flapper in combination with the Lighthouse deck the mean error is mostly consistent over multiple experiments, except for the circle and helix flights. These flights show a greater range in 3D errors. The mean error for the Loco deck has much more variance. For the Flapper both systems suffer outliers, and like the Crazyflie, the Lighthouse consistently shows more range in its outliers.



(a) Crazyflie



(b) Flapper

Figure 4.2: 3D error plots per experiment per platform

The data points of all experiments have been combined and the pre-defined metrics have been computed as can be seen in Table 4.1, showing the general accuracy of each setup combination. Beforehand, it should be noted that for the minimum error while some rows may show an error of 0.00, this error is never exactly 0, but it is lower than 0.001.

	Platform	**Deck**	**Lines**	**MAE**	**Std**	**Min**[1]	**25%**	**50%**	**75%**	**Max**	**RMSE**
Error_3D	Crazyflie	Lighthouse	121914	0.21	0.25	0.00	0.04	0.11	0.23	2.71	0.33
		Loco	112991	0.31	0.18	0.00	0.17	0.27	0.41	1.45	0.36
	Flapper	Lighthouse	128382	0.15	0.20	0.00	0.04	0 .07	0.14	1.26	0.25
		Loco	123388	0.40	0.25	0.02	0.20	0.34	0.57	1.54	0.47
Error_X	Crazyflie	Lighthouse	121914	0.13	0.21	0.00	0.01	0.04	0.14	2.01	0.24
		Loco	112991	0.12	0.12	0.00	0.04	0.09	0.16	1.20	0.17
	Flapper	Lighthouse	128382	0.07	0.16	0.00	0.01	0 .02	0.05	1.26	0.18
		Loco	123388	0.19	0.21	0.00	0.06	0.12	0.24	1.21	0.28
Error_Y	Crazyflie	Lighthouse	121914	0.10	0.14	0.00	0.01	0.03	0.11	1.38	0.17
		Loco	112991	0.11	0.11	0.00	0.03	0.08	0.15	0.86	0.16
	Flapper	Lighthouse	128382	0.06	0.12	0.00	0.01	0 .02	0.06	0.91	0.13
		Loco	123388	0.14	0.16	0.00	0.04	0.08	0.16	0.97	0.21
Error_Z	Crazyflie	Lighthouse	121914	0.06	0.12	0.00	0.01	0.02	0.06	1.58	0.14
		Loco	112991	0.21	0.17	0.00	0.08	0.18	0.31	1.43	0.27
	Flapper	Lighthouse	128382	0.07	0.10	0.00	0.03	0 .04	0.06	0.99	0.12
		Loco	123388	0.24	0.20	0.00	0.09	0.19	0.34	1.54	0.31

Table 4.1: Metrics (m) per setup combination

Comparing the difference of performance of the platforms in the smallest positioning error, this seems dependent on the positioning system with which the platform is combined. The Crazyflie combined with the Loco system has a general smaller error than the Flapper with the Loco system, while the Flapper combined with the Lighthouse system has a smaller error than the Crazyflie with the same system.

When dissecting the error scores of the decks, the Lighthouse system shows a (substantially) smaller positioning error on all axes compared to the Loco system when looking at the MAE, apart from the Crazyflie on the X-error. It should be noted this is only a different of 0.01 metre. The RMSE shows the same pattern, with an exception also being the Y-coordinate for the Crazyflie. The quadcopter combined with the Lighthouse system shows a higher RMSE than with the Loco system. However, taking a closer look to the Crazyflie+Lighthouse, it continuously shows a smaller error than the Crazyflie+Loco in the quartiles, with a bigger maximum error and standard deviation.

Speed

Looking at Table 4.2 which sets out 3D error over different speed settings, the table shows a decrease in positioning error by a higher speed regardless of platform or deck.

Platform	Deck	Speed	MAE (m)	RMSE (m)
Crazyflie	Lighthouse	V=0.2	0.3025	0.4168
		V=0.5	0.1261	0.1735
	Loco	V=0.2	0.3215	0.3736
		V=0.5	0.3086	0.3537
Flapper	Lighthouse	V=0.2	0.1926	0.3127
		V=0.5	0.1001	0.1303
	Loco	V=0.2	0.4550	0.5307
		V=0.5	0.3955	0.4370

Table 4.2: 3D error per speed setting

Due to the short lives of the drones' batteries and the increase of sun over the day, most experiments have been performed spread out over multiple mornings and an occasional afternoon. Experimental circumstances per day were different and can be found in Section C.3. The average accuracy of the different setups per day is shown in Table 4.3. This table includes the number of data lines gathered per day. Per day the average 3D error for the Lighthouse system was ±10-15 centimetres and for the Loco system ±30-35 centimetres, regardless of the platform used. Only the first day (04-06) and 08-06 are exceptions, showing a much higher error. On 04-06 the weather was cloudy, but the sunscreens refused to go down, nothing blocking the natural sunlight. On 08-06 the screens did go down, but this day was characterized by being very sunny. For both days, the error for the Lighthouse could result out of the interference by the sunlight. 08-06 shows for all combinations an error around 0.65. While 04-06 has a lower error, it is still a substantially higher error with the Crazyflie+Lighthouse having an error of 20 centimetres and the Flapper+Lighthouse an error of 52 centimetres For the explanation of the spike in error for the Loco system further research must be conducted.

Date	**Platform**	**Deck**	**Lines**	**MAE**	**Std**	**Min***	**25%**	**50%**	**75%**	**Max**	**RMSE**
04-06	Crazyflie	Lighthouse	2967	0.13	0.16	0.01	0.03	0.05	0.15	0.52	0.20
	Flapper	Lighthouse	15242	0.40	0.33	0.01	0.05	0.43	0.68	1.26	0.52
05-06	Crazyflie	Lighthouse	23469	0.13	0.10	0.00	0.05	0.12	0.19	1.58	0.16
		Loco	24624	0.28	0.14	0.02	0.15	0.27	0.39	0.71	0.31
	Flapper	Lighthouse	25150	0.11	0.08	0.00	0.06	0.10	0.15	0.78	0.14
		Loco	29009	0.27	0.16	0.02	0.15	0.22	0.33	1.04	0.31
08-06	Crazyflie	Lighthouse	27680	0.58	0.23	0.00	0.50	0.59	0.65	1.55	0.63
		Loco	10297	0.55	0.23	0.04	0.43	0.55	0.66	1.45	0.60
	Flapper	Lighthouse	6270	0.62	0.19	0.10	0.56	0.64	0.70	1.19	0.64
		Loco	35597	0.64	0.24	0.02	0.51	0.63	0.76	1.54	0.68
09-06	Crazyflie	Lighthouse	22283	0.07	0.12	0.00	0.02	0.03	0.06	0.99	0.13
		Loco	36802	0.26	0.14	0.00	0.16	0.24	0.33	0.82	0.29
	Flapper	Lighthouse	32228	0.07	0.06	0.00	0.04	0.05	0.08	0.63	0.09
		Loco	28462	0.32	0.18	0.03	0.18	0.27	0.42	1.03	0.36
12-06	Crazyflie	Lighthouse	32563	0.13	0.12	0.00	0.06	0.11	0.16	2.71	0.17
		Loco	32620	0.31	0.17	0.01	0.17	0.27	0.42	1.04	0.35
	Flapper	Lighthouse	34731	0.09	0.08	0.00	0.04	0.06	0.10	0.62	0.12
		Loco	30320	0.34	0.19	0.02	0.18	0.30	0.46	1.02	0.38
15-06	Crazyflie	Lighthouse	12952	0.08	0.05	0.01	0.03	0.07	0.11	0.32	0.09
		Loco	8648	0.32	0.16	0.02	0.20	0.29	0.44	0.82	0.36
	Flapper	Lighthouse	14761	0.06	0.04	0.00	0.04	0.05	0.07	0.33	0.07

Table 4.3: 3D error per setup combination per date

4.2 Drift

To check for drift, the influence of time on the error must be reviewed. The correlation between time and error has been visualized in Figure 4.3. This graph is a visualisation of all flights where the drone performed a static hover of thirty seconds. The graph includes take off and landing time. As can be seen in the figure, the 3D error does not increase over time. For the combination Flapper+Loco, the error actually decreases.

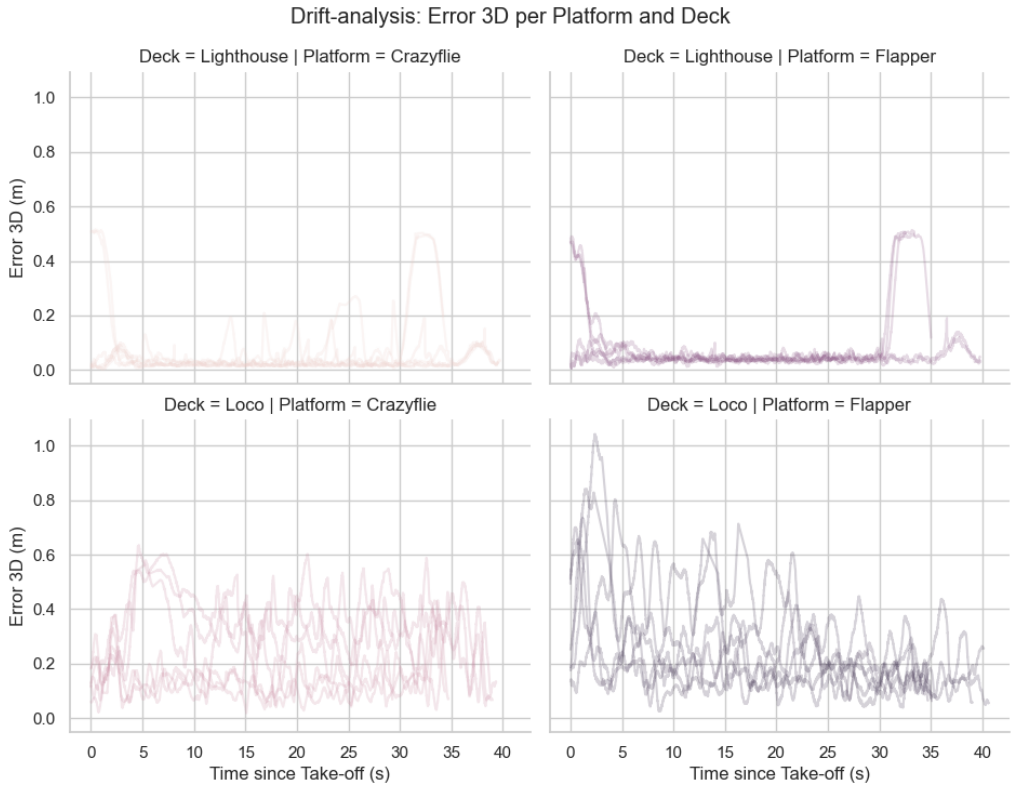


Figure 4.3: Drift analysis of static flights per setup

To turn this data numeric, a Pearson correlation was performed. It showed no strong relation between the time since take off and 3D error, except a slightly weak correlation of -0.45 for the Flapper+Loco. To check if this correlation upheld in all flights, another Pearson correlation was performed. However, the already weak correlation was diminished when subjugated to all flights, having an absolute score of 0.21.

	Platform	Deck	Mean Correlation
Static	Crazyflie	Lighthouse	0.12
		Loco	-0.02
	Flapper	Lighthouse	0.10
		Loco	-0.45
General	Crazyflie	Lighthouse	0.09
		Loco	0.10
	Flapper	Lighthouse	0.21
		Loco	-0.21

Table 4.4: Drift analysis: Pearson correlation 3D error over time

4.3 Position

Mapping out the 3D errors over the XYZ-plane led to Figure 4.4. This was an attempt to visualize spatial influence on the accuracy. This is a combination of every XYZ-coordinate where the drone has been, creating a complete map of all flight experiment trajectories together. For both drones the combination with the Lighthouse system shows an increase in error when the drone nears the 'limits' of the fly area. Especially the Flapper shows a higher error when the system nears the edges of two metres on all axes. For the combination with the Loco system, the figure doesn't show a strong correlation. The error seems to be general throughout the whole coordinate system.

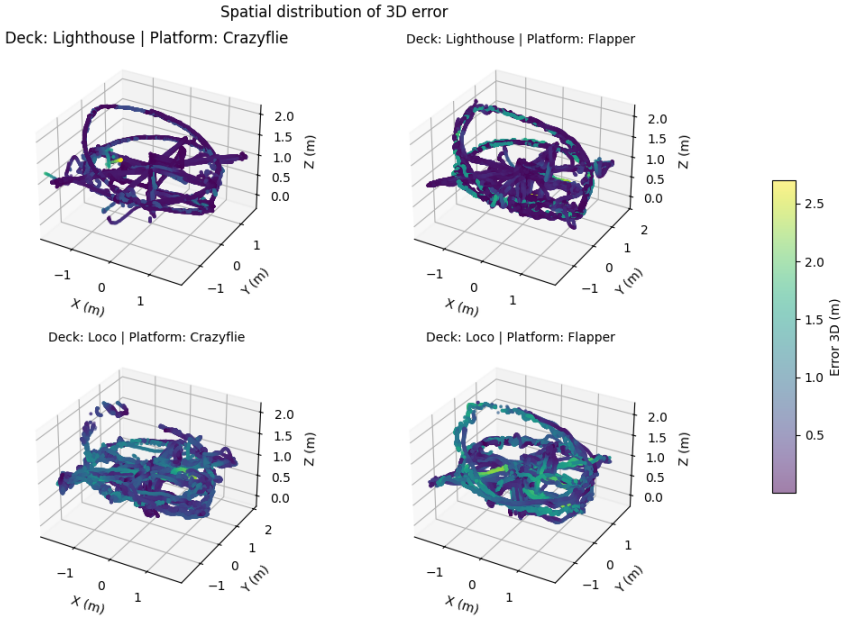


Figure 4.4: Spatial distribution of 3D error

A Pearson correlation performed on the absolute coordinates combined with the 3D error showed no strong correlation (Table 4.5). For the Crazyflie+Lighthouse was the strongest correlation to the XY-coordinates with 0.27 and 0.22 respectively, and for the Crazyflie+Loco was the strongest correlation of 0.34 with the Z-coordinates. For the Flapper+Lighthouse no coordinates seemed to have a remarkable correlation.

Platform	Deck	X	Y	Z
Crazyflie	Lighthouse	0.27	0.22	0.11
	Loco	0.15	0.16	0.34
Flapper	Lighthouse	0.17	0.15	0.09
	Loco	0.23	0.15	0.14

Table 4.5: Pearson correlation of 3D error over position

4.4 Extra factors

For a general analysis of other influencing factors, a small XGBoost model has been trained on the data grouped by platform and deck. The final parameters for training can be seen in Table 4.6a. The models' performance was measured by the R^2-score as shown in Table 4.6b. The model performed worse for the Lighthouse systems compared to the Loco system, but changing parameters did not increase the performance by much. For this research's purpose, being analysis instead of real-life application, the score was deemed sufficient.

Parameter	**Value**
Tree method	Hist
N estimators	100
Learning rate	0.05
Max Depth	8
Objective	reg:squarederror

(a) Model Parameters

System	Crazyflie	Flapper
Lighthouse	0.507	0.400
Loco	0.807	0.793

(b) Model R^2-score per setup

Table 4.6: Model XGBoost information

All variables shown in Table 4.7 were used in the model to predict the 3D error. The three parameters with the highest importance are marked bold in the table. To prevent data leakage, parameters regarding the coordinates or time period were eliminated.

For the Crazyflie+Lighthouse the battery seems of highest importance, closely followed by the Kalman variance of the velocity in the X-direction. The Crazyflie+Loco and both systems for the Flapper show that the Kalman variance of the Z-axis is an important feature for the error calculation. The loco systems also highlight the importance of its distance to the anchors. Comparably the Lighthouse system does show the signals received by the base stations are of importance, but other features show more promise for error prediction. For the Flapper+Lighthouse the pitch and yaw are also deemed important.

	Crazyflie		Flapper	
	Lighthouse	Loco	Lighthouse	Loco
Vbat	**0.17**	0.03	0.09	0.02
Roll	0.06	0.01	0.05	0.02
Pitch	0.08	0.02	**0.10**	0.07
Yaw	0.06	0.03	**0.10**	0.04
lhStatus	0.04	0.00	0.06	0.00
lhReceive	0.06	0.00	0.09	0.00
lcRng0	0.00	0.04	0.00	0.04
lcRng1	0.00	**0.08**	0.00	**0.08**
lcRng2	0.00	0.05	0.00	0.07
lcRng3	0.00	0.03	0.00	0.04
lcRng4	0.00	0.05	0.00	**0.12**
lcRng5	0.00	**0.12**	0.00	0.06
lcRng6	0.00	0.05	0.00	0.05
lcRng7	0.00	0.06	0.00	0.07
VarX	0.07	0.03	0.07	0.06
VarY	**0.09**	0.07	0.09	0.04
VarZ	0.07	**0.20**	**0.12**	**0.08**
VarPX	**0.14**	0.05	0.09	0.07
VarPY	0.08	0.05	0.07	0.06
VarPZ	0.07	0.02	0.07	0.02

Table 4.7: Feature Importance by model XGBoost

To further review factors, another Pearson correlation analysis was performed (Table 4.8). The reviewed parameters included the same parameters as used by the XGBoost model, with addition of the absolute values of the XYZ-coordinates. One variable that is consistently represented in the top 10 is the Z-coordinate, showing there is a general linear correlation between the Z-value and the 3D error.

For the Crazyflie+Loco the first three strongest correlations are the Z-coordinate, the Kalman variance of the velocity in the Z-direction and the Kalman variance of the Z-coordinate, showing a strong correlation with variables on the Z-axis.

The Lighthouse system shows favour to the Kalman variance of velocity in all directions. The Kalman variance of the X and Y coordinates show a correlation to the error in this system too.

Crazyflie	Lighthouse	Crazyflie	Loco	Flapper	Lighthouse	Flapper	Loco
VarPX	0.15	Z	0.34	VarPY	0.10	lcRng4	0.19
VarPZ	0.14	VarPZ	0.33	VarPX	0.09	X	0.17
Z	0.11	VarZ	0.30	Z	0.09	VX	0.16
Vbat	0.09	lcRng3	0.17	lhReceive	0.08	lcRng6	0.15
lhReceive	0.09	lcRng7	0.16	Yaw	0.07	Pitch	0.14
VarX	0.08	Y	0.13	VarPZ	0.07	Z	0.14
VarPY	0.08	lcRng5	0.11	VarZ	0.07	lcRng2	0.14
VarZ	0.07	Vbat	0.11	VarY	0.06	VarY	0.11
VarY	0.07	VY	0.09	VZ	0.05	lcRng0	0.11
X	0.07	VarY	0.09	lhStatus	0.05	lcRng5	0.10

Table 4.8: Top 10 variables for linear correlations to 3D error per setup

Chapter 5

Discussion

This study aimed to examine the performance of the Flapper in terms of positioning accuracy as provided by the Lighthouse and Loco Positioning Systems. For reliability of the data, all tests have been performed on the Crazyflie as well to reduce the influence of faulty setups. This creates a window for comparison of performance between the two drones.

The accuracy was measured by the positioning error, where the drone's own positioning was compared to a ground truth as provided by the motion capture system OptiTrack. For concise and controlled analysis, the main parameter looked at was the 3D error calculated by the Euclidean formula, while sometimes the position error on each separate axis was also taken into account.

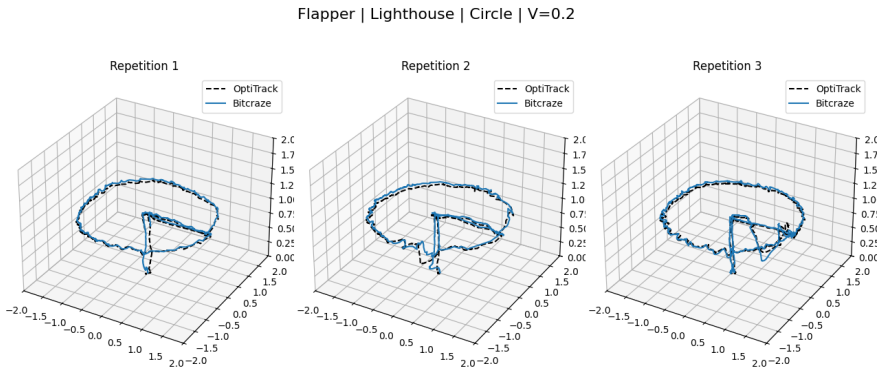
5.1 General Observations

5.1.1 Lighthouse

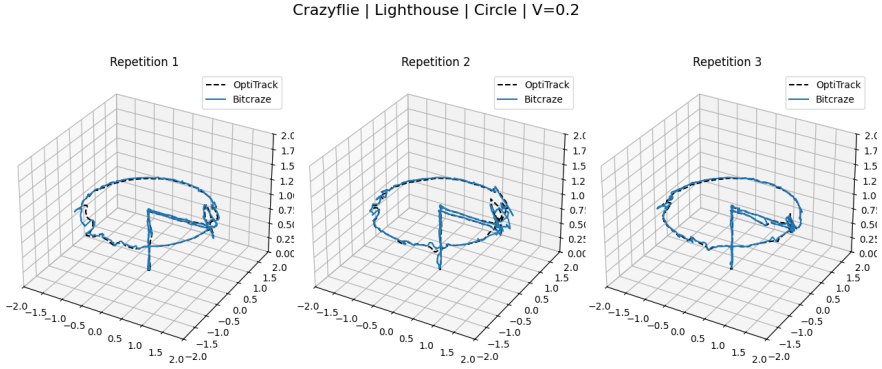
Overall the Flapper combined with the Lighthouse System exhibited smaller positioning errors than with the Loco positioning system, indicating a higher positioning accuracy. The Crazyflie shows the same pattern, except for the X-axis. In this case the larger X-error can be attributed to a larger set of outliers as reflected by a substantially larger max error and standard deviation. Across all experiments, the Flapper+Lighthouse achieved a mean positioning error of approximately 20 centimetres.

Separating the accuracy per day (Table 4.3) shows four out of six days achieve a mean absolute 3D error within ± 10 centimetres, while the two remaining days (04-06 and 08-06) exhibit an positioning error larger than 40 centimetres. Checking the accuracy scores per flight experiment, the experiments with the largest deviation of the standard positioning errors for the Flapper+Lighthouse were the circle and helix flight at a velocity of 0.2 m/s, for which the Crazyflie+Lighthouse also shows a larger deviation. The higher positioning error can be directly traced back to these experiments. The Flapper executed the circle flights on 04-06 and the circle and helix flight of the Crazyflie were performed on 08-06, explaining the spikes on those dates.

Visualizing these particular flights (Figure 5.1 and Figure A.4), the coordinates computed by the Lighthouse system and the OptiTrack seemingly correspond quite well, yet the error is high. While the coordinates seem to correspond, they do not. For visualisation the cause can be found by plotting the Z-coordinates over time (Figure 5.2). The ground truth was not properly synchronized to the time of the drone; the OptiTrack is always a few seconds ahead or behind, depending on the platform. This temporal offset led to a systematic difference in coordinates, setting a baseline for the error. As a result, part of the high error in these specific experiments is just a result of time misalignment rather than the intrinsic Lighthouse accuracy. Together with the unusual lighting conditions on these days, this suggests that the high daily errors are driven by a small number of problematic flights rather than by a general degradation of Lighthouse performance.

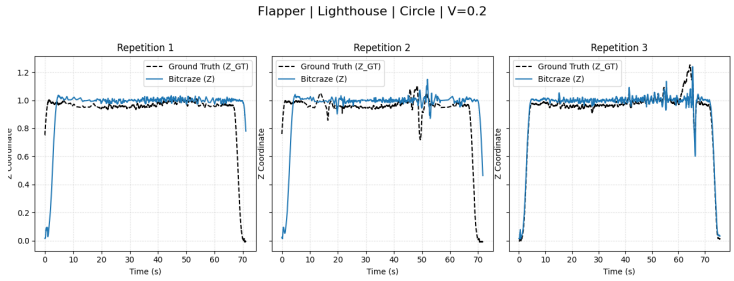


(a) Flapper - Lighthouse System

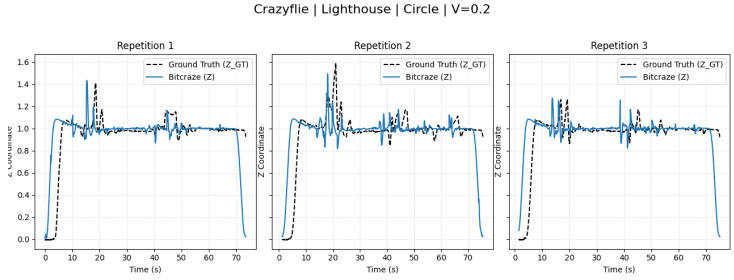


(b) Crazyflie - Lighthouse System

Figure 5.1: Visualised Circle experiment V=0.2 m/s



(a) Flapper - Lighthouse



(b) Crazyflie - Lighthouse

Figure 5.2: Circle flight: Z-coordinates over time

The Lighthouse System showed no signs of drifting over time. The error was not strongly influenced by its flight time. The position in the coordinate frame however did have some influence. As shown by plotting the errors in the 3D space, the errors tended to increase as the drone neared the edges of the tracked volume defined by the base-station's line of sight. For both platform this effect was visible on all axes, although the correlations were relatively weak.

The influence of the drone's velocity is slightly unclear. Table A.4 suggests that the 3D error decreases as speed increases. However, the slower flights are also known to consist of problematic flights, such as the circle experiments discussed above. The Pearson correlation showed a correlation for velocities on all axes, each being in the top ten. For the Crazyflie the velocity in X and Z direction was even the highest correlation, and for the Flapper the X and Y direction. However, these correlations remained modestly low, prohibiting the claim that velocity is a strong influence.

In Table 5.1 occurrences of the signals received by the Lighthouse deck are illustrated per experiment in percentages. There is a division between when the drone receives no signal, a signal from either base station, or both. For the Crazyflie the speed barely influences the signal received by deck. On the contrary, for the Flapper the occurrence of the deck receiving just one base station increases as velocity rises. This is likely related to the increased yaw and attitude changes required for faster flight, which may temporarily block the line of sight to one of the base stations.

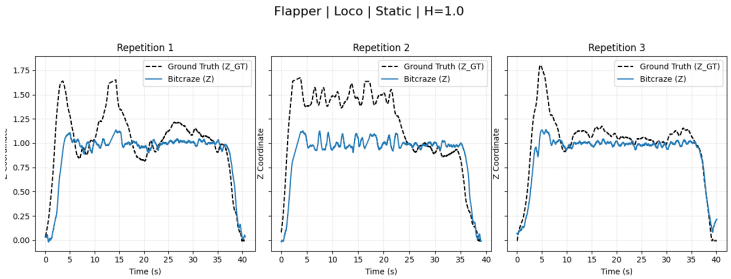
Platform	Experiment	Setting	None	BS 1	BS 2	Both
Crazyflie	Circle	V=0.2	0.05	0.1 5	0.12	0.68
		V=0.5	0.02	0.18	0.12	0.68
	Helix	V=0.2	0.04	0.14	0.11	0.72
		V=0.5	0.05	0.09	0.12	0.74
	LinearX	V=0.2	0.02	0.20	0.14	0.64
		V=0.5	0.04	0.20	0.11	0.65
	LinearXY	V=0.2	0.04	0.05	0.22	0.69
		V=0.5	0.07	0.06	0.24	0.63
	LinearY	V=0.2	0.02	0.08	0.10	0.80
		V=0.5	0.01	0.04	0.15	0.80
	StaircaseX	V=0.2	0.02	0.11	0.10	0.76
		V=0.5	0.03	0.13	0.08	0.75
	Static	H=0.5	0.02	0.02	0.35	0.60
		H=1.0	0.01	0.03	0.11	0.85
Flapper	Circle	V=0.2	0.03	0.18	0.02	0.77
		V=0.5	0.05	0.37	0.02	0.57
	Helix	V=0.2	0.07	0.29	0.05	0.59
		V=0.5	0.08	0.42	0.03	0.48
	LinearX	V=0.2	0.01	0.07	0.02	0.90
		V=0.5	0.03	0.23	0.02	0.72
	LinearXY	V=0.2	0.02	0.12	0.02	0.84
		V=0.5	0.06	0.19	0.04	0.71
	LinearY	V=0.2	0.04	0.17	0.02	0.77
		V=0.5	0.02	0.23	0.02	0.72
	StaircaseX	V=0.2	0.01	0.16	0.03	0.81
		V=0.5	0.03	0.19	0.03	0.74
	Static	H=0.5	0.01	0.05	0.04	0.91
		H=1.0	0.01	0.05	0.04	0.90

Table 5.1: Percentages for signals of Lighthouse base stations received per experiment

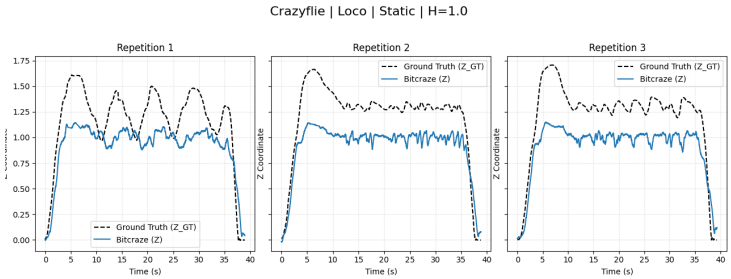
5.1.2 Loco System

For the Loco system, the Crazyflie exhibits a lower positioning error than the Flapper combined with the Loco deck, indicating a higher positioning accuracy (Table 4.1). Across all trajectories, the 3D RMSE typically ranges from 30 centimetres up to 60 centimetres for both platforms, which is considerably higher than the positioning error obtained by the Lighthouse system. This confirms that in its current configuration, the Loco system has less accuracy. The Flapper+Loco combination proves to be the least accurate of all four tested setups.

When decomposing the 3D error into its components (Figure A.2), the Z-axis clearly dominates the positioning error. The X- and Y-coordinates remain relatively low, but the estimation of the Z-coordinates shows larger deviations, which inflates the overall 3D-error. Looking at the static flights, the drone consistently 'believes' it is lower than it actually is, thus compensating by climbing and then overshooting the target. After some time, the Flapper occasionally realises its error and returns to the specified height. This can be seen in Figure 5.2, where the Z-coordinate is set out against the time. For the Flapper in static flight, the drone corrects itself (Figure 5.3a. The Crazyflie however has a tendency to remain at its presumed height, suggesting an overconfidence in its estimated position.



(a) Flapper + Loco



(b) Crazyflie + Loco

Figure 5.3: Static flight: Z-coordinate over time

The observed Z-dominance is consistent with underlying ranging data. Table 5.2 shows the standard deviations of Loco range measurements (Jitter noise) of the multiple anchors of the Loco System. This deviation is commonly between 0.30-1.00. While the averages of the jitter noise seem quite alike, anchors with numbers 1, 3, 5 and 7 show an extreme increase during the LinearXY flights. This indicates a lot of noise, meaning there is likely multipath interference. This interference can often translate into uncertainty especially in the vertical direction.

Platform	Experiment	Setting	lcRng0	lcRng1	lcRng2	lcRng3	lcRng4	lcRng5	lcRng6	lcRng7
Crazyflie	*Average*		0.53	0.64	0.50	0.71	0.54	0.71	0.51	0.66
	Circle	V=0.2	0.79	0.81	0.74	0.90	0.80	0.95	0.79	0.87
		V=0.5	0.91	0.85	0.88	0.92	0.92	0.97	0.90	0.89
	Helix	V=0.2	0.86	0.78	0.69	0.92	0.80	0.88	0.82	0.84
		V=0.5	0.64	0.70	0.62	0.92	0.64	0.83	0.66	0.82
	LinearX	V=0.2	0.42	0.38	0.55	0.48	0.67	0.79	0.59	0.37
		V=0.5	0.53	0.47	0.53	0.49	0.76	0.90	0.63	0.52
	LinearXY	V=0.2	0.43	1.20	0.24	1.31	0.31	1.20	0.24	1.26
		V=0.5	0.39	1.20	0.31	1.29	0.35	1.21	0.23	1.26
	LinearY	V=0.2	0.77	0.64	0.56	0.77	0.45	0.30	0.51	0.69
		V=0.5	0.79	0.72	0.62	0.81	0.51	0.41	0.58	0.76
	StaircaseX	V=0.2	0.32	0.50	0.56	0.35	0.57	0.61	0.54	0.31
		V=0.5	0.30	0.41	0.45	0.33	0.54	0.67	0.44	0.33
	Static	H=0.5	0.07	0.08	0.10	0.20	0.08	0.09	0.08	0.17
		H=1.0	0.16	0.17	0.17	0.25	0.13	0.10	0.07	0.12
Flapper	*Average*		0.61	0.66	0.57	0.69	0.57	0.68	0.52	0.67
	Circle	V=0.2	0.93	0.86	0.86	0.92	0.95	0.97	0.89	0.95
		V=0.5	0.99	0.85	1.08	0.98	1.04	0.96	1.01	0.93
	Helix	V=0.2	0.97	0.81	0.80	0.93	0.90	0.86	0.89	0.87
		V=0.5	0.86	0.83	0.77	0.93	0.74	0.81	0.74	0.90
	LinearX	V=0.2	0.47	0.41	0.45	0.46	0.58	0.65	0.48	0.40
		V=0.5	0.52	0.55	0.50	0.47	0.70	0.72	0.54	0.48
	LinearXY	V=0.2	0.32	0.93	0.24	0.94	0.27	0.93	0.20	0.99
		V=0.5	0.44	1.21	0.44	1.19	0.43	1.15	0.30	1.26
	LinearY	V=0.2	0.81	0.80	0.81	0.78	0.52	0.48	0.60	0.79
		V=0.5	0.76	0.69	0.77	0.73	0.51	0.39	0.59	0.75
	StaircaseX	V=0.2	0.37	0.53	0.42	0.34	0.52	0.60	0.41	0.35
		V=0.5	0.43	0.44	0.46	0.38	0.51	0.54	0.42	0.31
	Static	H=0.5	0.35	0.10	0.11	0.25	0.13	0.23	0.13	0.18
		H=1.0	0.27	0.19	0.20	0.30	0.15	0.16	0.11	0.18

Table 5.2: Standard deviation of the jitter noise per Loco anchor

Besides the anchor data, the XGBoost model that was trained on the data also identified parameters connected to the Z-axis such as the Kalman variance of the Z-axis. Additionally, the Pearson correlation highlighted this dominance of the Z-axis even more so.

After analysis the Loco system doesn't seem to be influenced by either time nor location, with an exception of the Z-axis. Figure 4.3 showed no signs of drift and instead showed a decrease in error for the Flapper. As stated before, the Flapper showed a slight correction in altitude.

The Flapper in combination with the Loco shows an ability to fly roughly the pre-defined flight trajectories, but to keep the drone from crashing, it is important there remains a buffer towards the ceiling in height.

5.2 Comparison with previous work

From the experiments, it was concluded the Lighthouse System had a general better performance in accuracy than the Loco System. This evidence is supported by the literature. The Lighthouse System is claimed by Bitcraze to have millimetre accuracy, and the Loco system is claimed to have centimetre accuracy. For the Lighthouse system, the procured accuracy is substantially worse than what the literature claims. [22] reported a mean Euclidean error of about 2 centimetres. The accuracy of this thesis' setup achieved a Mean 3D error of approximately 20 centimetres.

This difference can be attributed to multiple things. As stated before, a temporal error caused an increase in error in multiple flights by falsely synchronizing the coordinates. Additionally, the Lighthouse system is known to decrease in accuracy when exposed to glass or mirrors. While there is a distance between the base stations, one of the base stations has clear view of the large window that envelops the entire north side of the experimental area. The experimental area of the literature was a large inside space with no natural light. This thesis' experimental area is constantly subjugated to natural light with the sunscreens never completely blocking the light that shines through the window. Even if the sun isn't shining, infrared light consistently falls on the window. These factors can easily contribute to the decrease in accuracy.

While the Lighthouse system was originally developed for the Crazyflie, the Flapper has shown a slightly higher accuracy in the performed flights. During flight, the Crazyflie consistently showed a slight wobble when nearing the edges of the field, losing altitude and then quickly correcting itself. These volatile movements are harder for the OptiTrack to track, especially in combination with the placement of the OptiTrack markers, which were already very small too. While observing the mocap system's tracking, it was observed that this system lost track of the Crazyflie more often. The Crazyflie+Lighthouse deck also has slightly lower data points. As seen in the plots, this combination also showed more outliers, which were also partly caused by the same temporal error as mentioned before.

The Loco system has also shown a worse accuracy than reported in the literature. The accuracy in literature is supposedly 10 centimetres, but the experiments showed an accuracy ranging between 0.30 and 1.00 metres. The data gathered by the anchors do point to the chance the radio signals suffered from multipath interference. Despite the communication being done by radio waves, the system still prefers a line of sight. Possible obstructions could be the metal overhead beams on which the top anchors are placed.

While the system was developed for the Crazyflie and is deemed experimental for the Flapper, the difference in accuracy is very small. Breaking the 3D error down to its components and comparing the error per axis, the max difference in error is less than seven centimetres. For the variance in 3D error, this is also less than 10 centimetre. While there is definitely room for improvement for the system, especially for the altitude control, the system works fine for both platforms. The general robustness of the Loco system is even better than of the Lighthouse system, since it has less outliers.

5.3 Limitations

Experimental Area
For limitations during this research, the biggest setback was the large window at the north side of the experimental area. Even before the sun arrived at that side of the building, the light that shown through the window could interfere with the OptiTrack. Part of this was caused by reflections as well. When the sun hadn't reached its highest yet, the sunscreens were relatively equipped enough to block any infrared nuisance from outside. However, once the sun directly hit the window, anytime the drones reached the the side closest to the window, they would wobble or sometimes even crash when using the Lighthouse system. The OptiTrack system would also lose its track of the drone. Another limiting factor was that the sunscreens were sensitive to winds, meaning if there was wind, they would not go down as they were electrical. For future research if there is a window available, it would be beneficial to use black, thick curtains that can be manually closed. In combination with the short longevity of the batteries, the experiments were spread out over multiple days, leading to a number of different circumstances. Due to the amount of reps(=3), the data became very sensitive for outliers.

OptiTrack
The range of the OptiTrack was also limiting. Apart from its sensitivity to sunlight, the trackable volume was not as large as expected. The distances of the experiments were changed to fit this volume, when originally there had been an intention to test a larger area. Additionally, the placement of the markers was troubling. When the drones stood on the ground, the OptiTrack sometimes had difficulty registering them. The markers were very small, which was necessary for the duration of the battery, but it proved a smaller surface was harder to detect.

Unsynchronized time
As noted before, not all coordinates per experiment were perfectly synchronized in time due to the difference in Unix Epoch Time. A script computed the time difference per day, but the data suggests this difference in time changes over the day. There is a moment in which the computer resets its Unix time, and if that happens after the time difference has been computed, the computed time difference will not make the data correspond. This leads a higher error, despite the fact the systems might actually be correct. Some experiments like Figure 5.4 show almost perfect alignment, while others like Figure 5.5 consistently show a slight temporal offset.

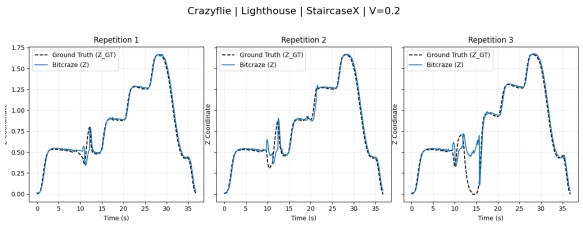


Figure 5.4: Staircase flight: Z-coordinate over time

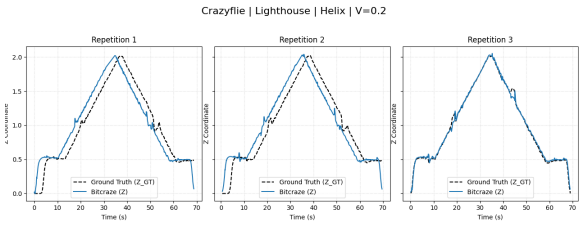


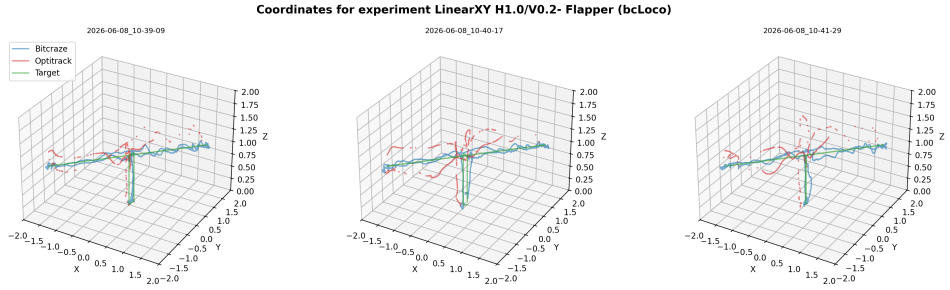
Figure 5.5: Helix flight: Z-coordinate over time

Detransforming data

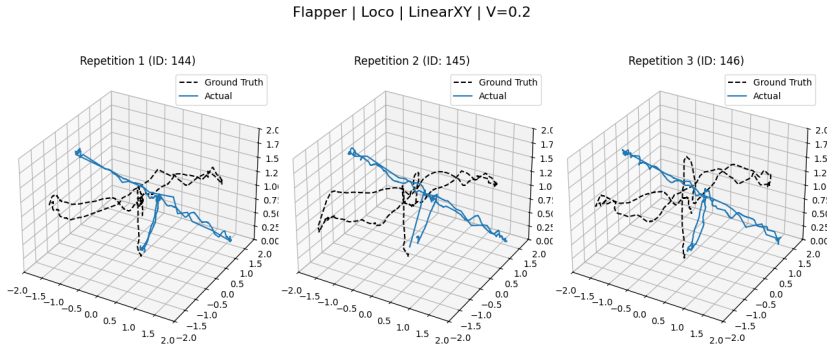
During analysis, it became evident the transformation matrix computed by the motionless data points was faulty and increased the positioning error instead of reducing it. Bitcraze's positioning systems were set up in a way to limit the difference to the OptiTrack's coordinate system, picking an origin as close to the OptiTrack's origin as possible. However, after visualizing the flight trajectories, it became evident some transformation matrices, especially the ones computed for the Loco system, had applied unnecessary rotations. An example of this phenomenon can be seen in figure 5.6 with its corresponding transformation matrix (Equation 5.1).

$$\begin{bmatrix} 0.972 & -0.142 & 0.183 & -0.0145 \\ 0.045 & 0.890 & 0.453 & -0.041 \\ -0.228 & -0.433 & 0.872 & 0.092 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{5.1}$$

Figure 5.6a shows the visualization of the coordinates before the transformation matrix was applied, and Figure 5.6b shows the effects after, showing a skewed coordinate system. Having witnessed the drone's flying direction which was evidently straight up, this is a clear mishap, resulting in applying the inverse matrices to all coordinates to undo the damage. While not all coordinates of every day had been affected, for a fair comparison all had to be returned to their original state. This action proved fruitful as the error immediately showed a large decrease in all areas. However, the relationships between error rates stayed relatively the same, which can be explained by the fact the error is of all axes, and the matrices applied only rotation and translation, which doesn't change the ratio between data points.



(a) Coordinates pre-transformation



(b) Twisted coordinates post-transformation

Figure 5.6: Influence of transformation matrix

For more robust transformation matrices, the time spent recording should be longer than a minimum of five seconds. Afterwards, it should be checked if the OptiTrack system also gathered enough data points. For this thesis, some days the motionless data consisted of just seventy files per placement, which is not enough data to do compute a proper transformation matrix, especially if there are outliers present. One should immediately perform static flights too, to account for the time difference in the systems if present.

5.4 Future work

Future experiments should implement a more reliable time synchronisation method between the OptiTrack and the onboard logs to avoid errors caused by temporal misalignment. A more reliable way of setting up a transformation matrix may also improve the results.

Either way, both the positioning systems should be examined more closely. The Lighthouse system has the potential for very high accuracy that shows a lot of promise in fine-control of drones. Its main limitation appears to be sensitivity to other sources of infrared light. As the OptiTrack shows, not all infrared light interferes with the system. It would be interesting to test to what extent the Lighthouse system can withstand interference of other sources of infrared light such as sunlight. This is consistent with the large errors observed on 04-06 and 08-06. During experiments, the drone did manage to fly without crashing even with sunlight sometimes interfering. The problem was eventually it does lose track of its state and won't be able to reposition itself in time. If there is a way to fine-tune that, the system can be incredibly valuable.

For the Loco system the data strongly suggests the key to unlocking the full potential of the system is in its altitude control. Mastering altitude estimation will likely result in a smaller positioning error thus increasing its overall accuracy. And while for this research only the mode Two Way Ranging was used, future works can repeat the experiment for modes TDOA2 and TDOA3, which have a different trade-off. Since this experiment showed there is not a big difference in accuracy by between combining the Loco system with either platform, with an accuracy varying between 0.30 centimetres and 0.40 centimetres, it would be recommended to test future Loco modes with the Flapper. This platform shows a steadier control of flight with less volatile motions, making it the safer choice in the experimental phase. The drone can crash or fly against objects and there is no damage done. For experiments this is an optimal condition.

Ideally, the decks should be combined. The robustness of Loco system combined with the high precision of the Lighthouse system can lead to an extremely promising way of positioning. Whether this will be attempted in the future, only time will tell. For now the major obstacle is the fact both decks use the same pins. For future reference, the hardware might have to be redesigned.

Chapter 6

Conclusion

This thesis investigated the performance of the Lighthouse and Loco positioning systems on the Flapper. For reference the quadcopter the Crazyflie performed the same experiments. The main goal was to evaluate both positioning systems, examining the difference in accuracy under different circumstances of controlled flight, testing which system was more suitable. Out of all experiments, the Lighthouse system consistently outperformed the Loco system on either platform. Typical Mean Absolute Errors for the Lighthouse were in the range of 10-20 centimetres, whereas the Loco system had errors of approximately 30-40 centimetres. The Lighthouse system showed a consistency over time without experiencing drift. The accuracy is however significantly lower than as described in the literature. These differences can be ascribed to experimental limitations, including imperfect time synchronization of the two coordinate systems and challenging environmental conditions such as sunlight. This led to variability in day-to-day accuracy, with a low performance specifically shown in the circle and helix flights flown on low speed.

The Loco system showed substantially larger errors, particularly on the vertical axis. After closer examination of errors spread over different axes, the anchor robustness and feature importance showed altitude estimation is the biggest challenge for the Loco system, with multipath interference most likely contributing to the errors. This system is suitable for rough control, but for highly accurate control this system needs improvement.

Comparing the Loco system on the Flapper with the Crazyflie, there was no big difference between 3D errors. With the Loco system being deemed 'experimental' on the Flapper by Flapper drones, a larger difference had been expected. This small difference shows that the system has been sufficiently attuned to the specific flight dynamics of the Flapper.

Overall, the results demonstrate the Lighthouse has high accuracy for position estimation of the Flapper, whereas the Loco can perform general accuracy, but requires an improvement in altitude estimation and control before high accuracy can be achieved. Future work should focus on testing the effects of environmental factors on both systems. An interesting road could be for sensor fusion of between the Lighthouse, Loco and IMU sensors, thus creating a more robust positioning system.

Bibliography

[1] *BitCraze — BitCraze*. URL: https://www.bitcraze.io/about/bitcraze/.

[2] Marco Buschmann et al. "Development of a Fully Autonomous Micro Aerial Vehicle (MAV) for Ground Traffic Surveillance". In: *IFAC Proceedings Volumes* 36.14 (Aug. 1, 2003), pp. 73–78. DOI: 10.1016/s1474-6670(17)32398-4. URL: https://doi.org/10.1016/s1474-6670(17)32398-4.

[3] C. De Wagter et al. "Autonomous flight of a 20-gram Flapping Wing MAV with a 4-gram onboard stereo vision system". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 2014, pp. 4982–4987. DOI: 10.1109/ICRA.2014.6907589.

[4] *Flapper Drones - Bioinspired flyig robots*. URL: https://flapper-drones.eu/nimbleplus/.

[5] Wojciech Giernacki et al. "Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering". In: *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*. 2017, pp. 37–42. DOI: 10.1109/MMAR.2017.8046794.

[6] Joel Grasmeyer and Matthew Keennon. "Development of the Black Widow Micro Air Vehicle". In: *39th Aerospace Sciences Meeting and Exhibit* (Jan. 8, 2001). DOI: 10.2514/6.2001-127. URL: https://doi.org/10.2514/6.2001-127.

[7] Shivali Gupta, Simran Kaur, and Ujwal Gupta. "A Comparative Study of Machine Learning and Deep Learning Models for Drone Object Detection". In: *2025 6th International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV)*. 2025, pp. 680–685. DOI: 10.1109/ICICV64824.2025.11085911.

[8] Christopher Jekeli. *Inertial Navigation Systems with Geodetic Applications*. Dec. 31, 2001. DOI: 10.1515/9783110800234. URL: https://doi.org/10.1515/9783110800234.

[9] Simon J. Julier and Jeffrey K. Uhlmann. "New extension of the Kalman filter to nonlinear systems". In: *Proceedings of SPIE, the International Society for Optical Engineering/Proceedings of SPIE* 3068 (July 28, 1997), p. 182. DOI: 10.1117/12.280797. URL: https://doi.org/10.1117/12.280797.

[10] R. E. Kalman. "A New Approach to Linear Filtering and Prediction Problems". In: *Journal of Basic Engineering* 82.1 (Mar. 1, 1960), pp. 35–45. DOI: 10.1115/1.3662552. URL: https://doi.org/10.1115/1.3662552.

[11] Matej Karasek et al. "A tailless aerial robotic flapper reveals that flies use torque coupling in rapid banked turns". English. In: *Science* 361.6407 (Sept. 2018), pp. 1089–1094. ISSN: 0036-8075. DOI: 10.1126/science.aat0350.

[12] Matěj Karásek. *flapperdrones — Bitcraze*. URL: https://www.bitcraze.io/tag/flapperdrones/.

[13] Masoud Khodarahmi and Vafa Maihami. "A Review on Kalman Filter Models". In: *Archives of Computational Methods in Engineering* 30.1 (Oct. 1, 2022), pp. 727–747. DOI: 10.1007/s11831-022-09815-7. URL: https://doi.org/10.1007/s11831-022-09815-7.

[14] Khomsin et al. "Accuracy Analysis of GNSS (GPS, GLONASS and BEIDOU) Obsevation for Positioning". English. In: *E3S Web of Conferences* 94 (May 2019). Publisher Copyright: © The Authors, published by EDP Sciences, 2019.; 2018 International Symposium on Global Navigation Satellite System, ISGNSS 2018 ; Conference date: 21-11-2018 Through 23-11-2018. ISSN: 2555-0403. DOI: 10.1051/e3sconf/20199401019.

[15] Yongsheng Lian et al. “The characterization of tandem and corrugated wings”. In: *Progress in Aerospace Sciences* 65 (2014), pp. 41–69. ISSN: 0376-0421. DOI: https://doi.org/10.1016/j.paerosci.2013.08.001. URL: https://www.sciencedirect.com/science/article/pii/S0376042113000742.

[16] *Lighthouse Positioning System — BitCraze*. URL: https://www.bitcraze.io/documentation/system/positioning/ligthouse-positioning-system/.

[17] *Loco Positioning system — Bitcraze*. URL: https://www.bitcraze.io/documentation/system/positioning/loco-positioning-system/.

[18] Robert C. Michelson. “Overview of Micro Air Vehicle System Design and Integration Issues”. In: *Encyclopedia of Aerospace Engineering* (Sept. 14, 2010). DOI: 10.1002/9780470686652.eae401. URL: https://doi.org/10.1002/9780470686652.eae401.

[19] Diana A. Olejnik et al. “A Tailless Flapping Wing MAV Performing Monocular Visual Servoing Tasks”. In: *Unmanned Systems* 08.04 (2020), pp. 287–294. DOI: 10.1142/S2301385020500235. eprint: https://doi.org/10.1142/S2301385020500235. URL: https://doi.org/10.1142/S2301385020500235.

[20] Ernesto Sanchez-Laulhe, Guido C.H.E de Croon, and Anibal Ollero. “Equilibrium State for a Tailless Flapping Wing Micro Air Vehicle in Forward Flight”. In: *IEEE Robotics and Automation Letters* 11.1 (2026), pp. 17–24. DOI: 10.1109/LRA.2025.3629992.

[21] Soumitra P. Sitole, Andrew K. LaPre, and Frank C. Sup. “Application and Evaluation of Lighthouse Technology for Precision Motion Capture”. In: *IEEE Sensors Journal* 20.15 (2020), pp. 8576–8585. DOI: 10.1109/JSEN.2020.2983933.

[22] Arnaud Taffanel et al. *Lighthouse Positioning System: Dataset, Accuracy, and Precision for UAV Research*. Apr. 2021. DOI: 10.48550/arXiv.2104.11523.

[23] K. Thompson. “UNIX time-sharing system: UNIX implementation”. In: *The Bell System Technical Journal* 57.6 (1978), pp. 1931–1946. DOI: 10.1002/j.1538-7305.1978.tb02137.x.

[24] Syed Izzat Ullah and Jose Baca. *NanoBench: A Multi-Task Benchmark Dataset for Nano-Quadrotor System Identification, Control, and State Estimation*. 2026. arXiv: 2603.09908 [cs.RO]. URL: https://arxiv.org/abs/2603.09908.

[25] Greg Welch and Gary Bishop. *An Introduction to the Kalman Filter*. Vol. 1. 4. Jan. 1, 1995, pp. 1–16. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.705.3475.

Appendix A

Data

A.1 Data information

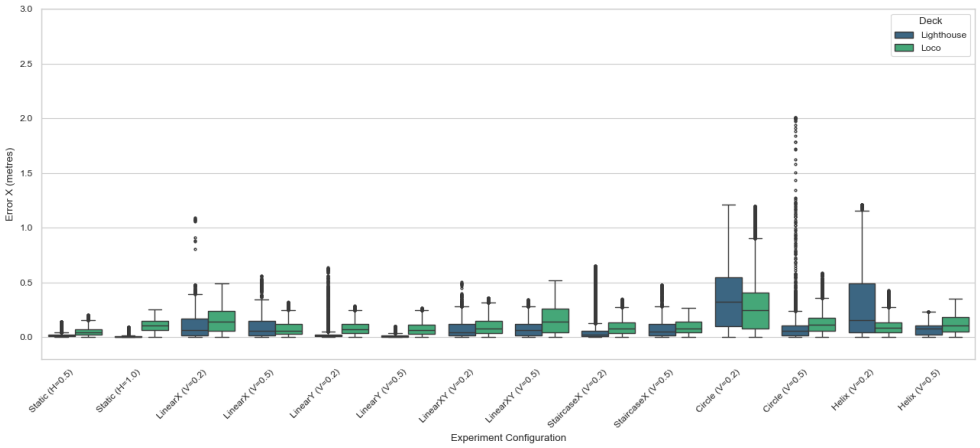
Parameters	**Meaning**	**Units**
Timestamp	Timer of Bitcraze	milliseconds
Time Unix	Timer of Unix	seconds
State estimation X	The estimation of the position on the x-axis	metre
State estimation Y	The estimation of the position on the y-axis	metre
State estimation Z	The estimation of the position on the z-axis	metre
State estimation Roll	The estimation of the rotation around the x-axis	degrees
State estimation Pitch	The estimation of the rotation around the y-axis	degrees
State estimation Yaw	The estimation of the rotation around the z-axis	degrees
State estimation VX	The estimation of the velocity on the x-axis	metre per second
State estimation VY	The estimation of the velocity on the y-axis	metre per second
State estimation VZ	The estimation of the velocity on the z-axis	metre per second
Kalman variance X	The uncertainty of the state estimation X	$(\text{metre per second})^2$
Kalman variance Y	The uncertainty of the state estimation Y	$(\text{metre per second})^2$
Kalman variance Z	The uncertainty of the state estimation Z	$(\text{metre per second})^2$
Kalman variance PX	The uncertainty of the state estimation VX	$(\text{metre per second})^2$
Kalman variance PY	The uncertainty of the state estimation VY	$(\text{metre per second})^2$
Kalman variance PZ	The uncertainty of the state estimation VZ	$(\text{metre per second})^2$
Control target X	Target position on the x-axis	metre
Control target Y	Target position on the y-axis	metre
Control target Z	Target position on the z-axis	metre
Lighthouse status	Number of active base stations	0/1/2
Lighthouse receive	Number of base stations currently received	0=None, 1=BS1, 2=BS2, 3=Both
Ranging distance n	Computed distance of drone to anchor n	metre

Table A.1: Parameters for experiment as gathered by Lighthouse and Loco system

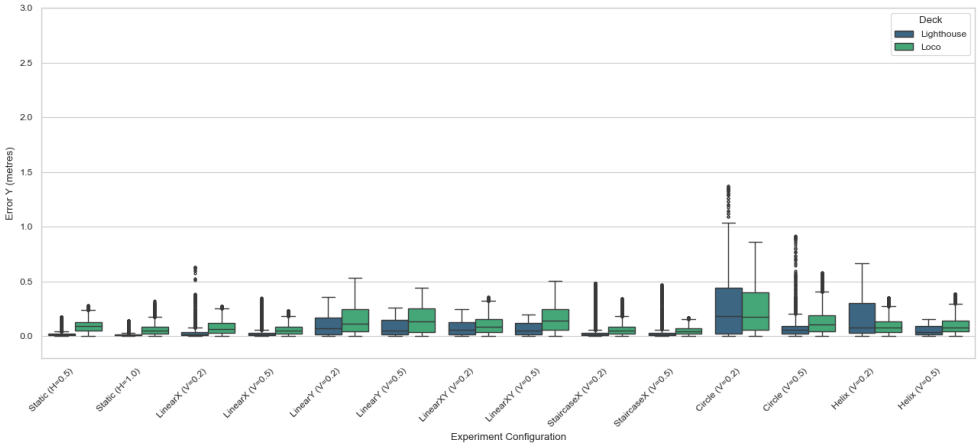
Parameters	Meaning	Units
Frame	Number of frame	int
Time	Time since begin tape	seconds
Rotation X	State estimation of the roll	metre
Rotation Y	State estimation of the pitch	metre
Rotation Z	State estimation of the yaw	metre
Position X	State estimation of the X-coordinate	metre
Position Y	State estimation of the Y-coordinate	metre
Position Z	State estimation of the Z-coordinate	metre
Error per marker	Confidence of state estimation of the rigid body	metre

Table A.2: Parameters for experiment as gathered by the Optitrack per rigid body

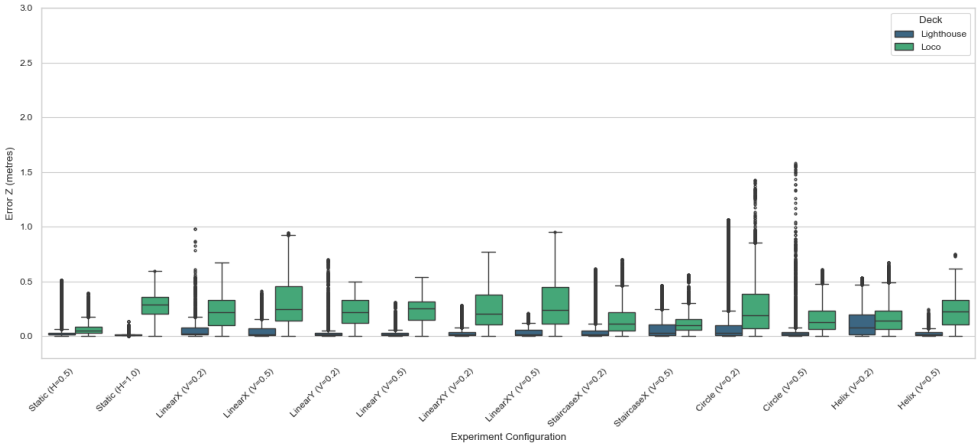
A.2 Results



(a) Error X

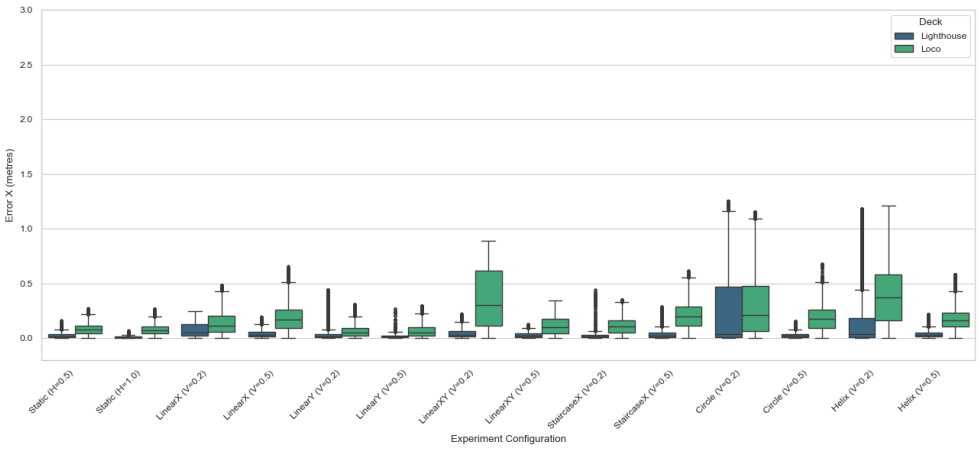


(b) Error Y

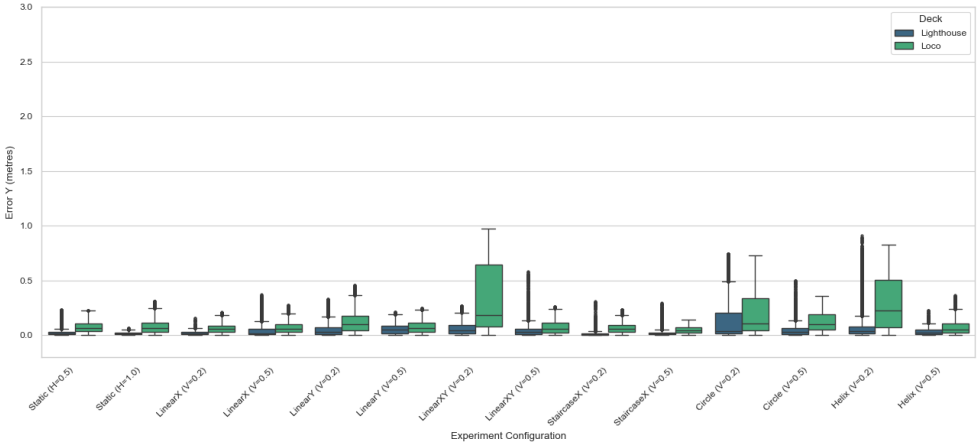


(c) Error Z

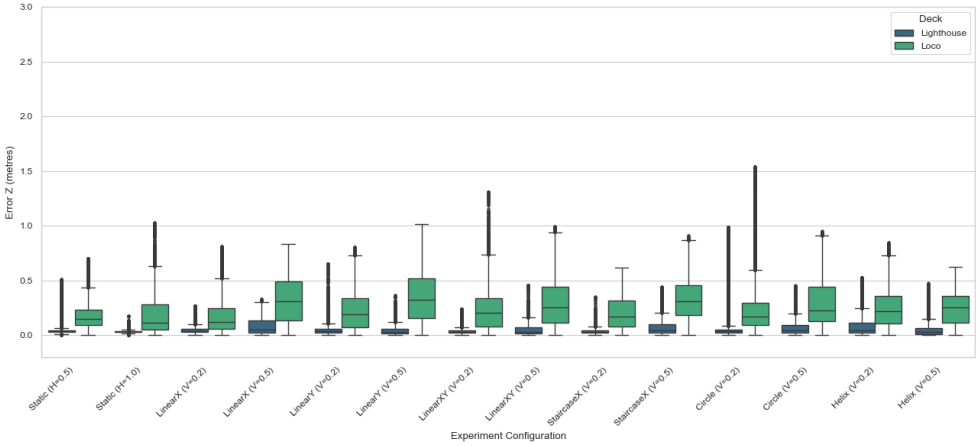
Figure A.1: Position Error per axis (Crazyflie)



(a) Error X



(b) Error Y



(c) Error Z

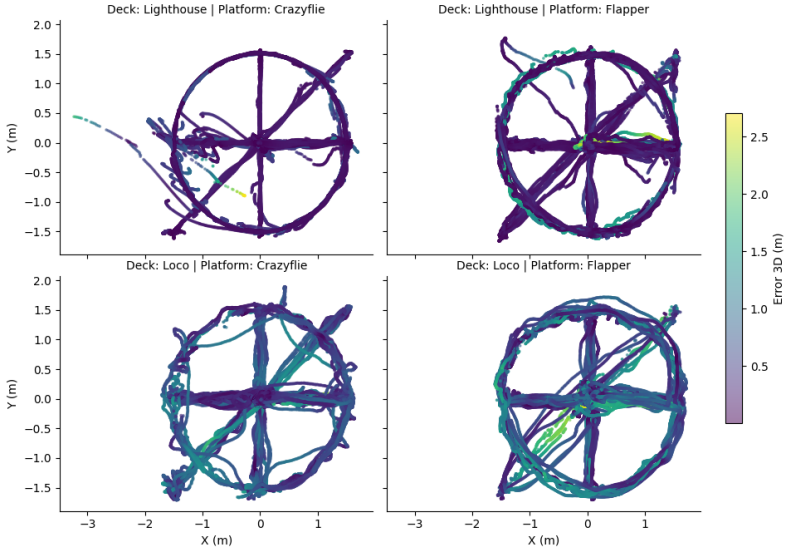
Figure A.2: Position error per axis (Flapper)

Platform	Deck	Experiment	Setting	06-04	06-05	06-08	06-09	06-12	06-15
Crazyflie	Lighthouse	Circle	V=0.2	0	0	3	0	0	0
			V=0.5	0	0	0	0	2	1
		Helix	V=0.2	0	0	2	0	0	1
			V=0.5	0	0	0	0	3	0
		LinearX	V=0.2	0	2	0	0	0	0
			V=0.5	0	0	0	1	2	0
		LinearXY	V=0.2	0	3	0	0	0	0
			V=0.5	0	0	0	0	3	0
		LinearY	V=0.2	0	3	0	0	0	0
			V=0.5	0	0	0	0	3	0
		StaircaseX	V=0.2	0	0	0	3	0	0
			V=0.5	0	0	0	0	3	0
		Static	H=0.5	2	0	0	0	0	1
			H=1.0	0	0	0	3	0	0
	Loco	Circle	V=0.2	0	0	3	0	0	0
			V=0.5	0	0	0	0	3	0
		Helix	V=0.2	0	0	0	3	0	0
			V=0.5	0	0	0	0	3	0
		LinearX	V=0.2	0	3	0	0	0	0
			V=0.5	0	0	0	0	3	0
		LinearXY	V=0.2	0	0	0	1	0	2
			V=0.5	0	0	0	0	3	0
		LinearY	V=0.2	0	3	0	0	0	0
			V=0.5	0	0	0	0	3	0
		StaircaseX	V=0.2	0	0	0	3	0	0
			V=0.5	0	0	0	0	3	0
		Static	H=0.5	0	3	0	0	0	0
			H=1.0	0	1	0	2	0	0
Flapper	Lighthouse	Circle	V=0.2	2	0	0	0	1	0
			V=0.5	0	0	0	0	3	0
		Helix	V=0.2	0	0	1	1	0	1
			V=0.5	0	0	0	0	3	0
		LinearX	V=0.2	0	3	0	0	0	0
			V=0.5	0	0	0	3	0	0
		LinearXY	V=0.2	0	3	0	0	0	0
			V=0.5	0	0	0	0	3	0
		LinearY	V=0.2	0	2	0	0	0	1
			V=0.5	0	0	0	3	0	0
		StaircaseX	V=0.2	0	0	0	3	0	0
			V=0.5	0	0	0	0	3	0
		Static	H=0.5	3	0	0	0	0	0
			H=1.0	0	0	0	2	0	1
	Loco	Circle	V=0.2	0	0	2	0	1	0
			V=0.5	0	0	0	0	3	0
		Helix	V=0.2	0	0	3	0	0	0
			V=0.5	0	0	0	0	3	0
		LinearX	V=0.2	0	3	0	0	0	0
			V=0.5	0	0	0	3	0	0
		LinearXY	V=0.2	0	0	3	0	0	0
			V=0.5	0	0	0	1	2	0
		LinearY	V=0.2	0	1	0	2	0	0
			V=0.5	0	0	0	3	0	0
		StaircaseX	V=0.2	0	0	0	3	0	0
			V=0.5	0	0	0	0	3	0
		Static	H=0.5	0	3	0	0	0	0
			H=1.0	0	2	0	0	1	0

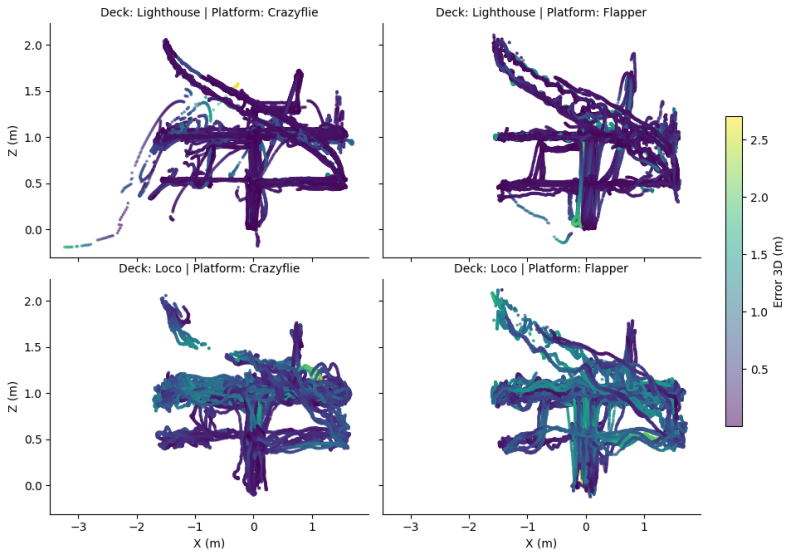
Table A.3: Experiments gathered per day

Platform	Deck	Experiment	H	V	D	MAE	RMSE
Crazyflie	Lighthouse	Circle	1.0	0.2	1.5	0.59	**0.64**
				0.5	1.5	0.14	**0.23**
		Helix	0.5	0.2	1.5	0.40	**0.49**
				0.5	1.5	0.11	0.12
		LinearX	1.0	0.2	1.5	0.14	0.19
				0.5	1.5	0.13	0.18
		LinearXY	1.0	0.2	1.5	0.12	0.15
				0.5	1.5	0.13	0.15
		LinearY	1.0	0.2	1.5	0.13	0.17
				0.5	1.5	0.11	0.13
		StaircaseX	0.5	0.2	1.5	0.09	0.17
				0.5	1.5	0.13	0.19
		Static	0.5	0.2	0.0	0.07	0.14
			1.0	0.2	0.0	0.03	0.04
	Loco	Circle	1.0	0.2	1.5	0.55	**0.60**
				0.5	1.5	0.30	0.33
		Helix	0.5	0.2	1.5	0.23	0.26
				0.5	1.5	0.29	0.33
		LinearX	1.0	0.2	1.5	0.32	0.35
				0.5	1.5	0.33	**0.38**
		LinearXY	1.0	0.2	1.5	0.32	0.36
				0.5	1.5	0.41	**0.47**
		LinearY	1.0	0.2	1.5	0.32	0.34
				0.5	1.5	0.32	0.35
		StaircaseX	0.5	0.2	1.5	0.22	0.26
				0.5	1.5	0.18	0.20
		Static	0.5	0.2	0.0	0.14	0.15
			1.0	0.2	0.0	0.32	0.34
Flapper	Lighthouse	Circle	1.0	0.2	1.5	0.37	**0.50**
				0.5	1.5	0.10	0.13
		Helix	0.5	0.2	1.5	0.24	**0.37**
				0.5	1.5	0.08	0.10
		LinearX	1.0	0.2	1.5	0.11	0.13
				0.5	1.5	0.12	0.16
		LinearXY	1.0	0.2	1.5	0.10	0.12
				0.5	1.5	0.10	0.14
		LinearY	1.0	0.2	1.5	0.10	0.14
				0.5	1.5	0.10	0.11
		StaircaseX	0.5	0.2	1.5	0.06	0.07
				0.5	1.5	0.10	0.13
		Static	0.5	0.2	0.0	0.10	**0.17**
			1.0	0.2	0.0	0.04	0.05
	Loco	Circle	1.0	0.2	1.5	0.52	0.58
				0.5	1.5	0.42	0.46
		Helix	0.5	0.2	1.5	0.65	**0.68**
				0.5	1.5	0.35	0.37
		LinearX	1.0	0.2	1.5	0.27	0.31
				0.5	1.5	0.44	0.47
		LinearXY	1.0	0.2	1.5	0.67	**0.76**
				0.5	1.5	0.36	0.41
		LinearY	1.0	0.2	1.5	0.29	0.33
				0.5	1.5	0.39	0.44
		StaircaseX	0.5	0.2	1.5	0.27	0.29
				0.5	1.5	0.43	0.46
		Static	0.5	0.2	0.0	0.23	0.25
			1.0	0.2	0.0	0.70	**0.74**

Table A.4: 3D MAE and RMSE (m) per flight experiment per setting



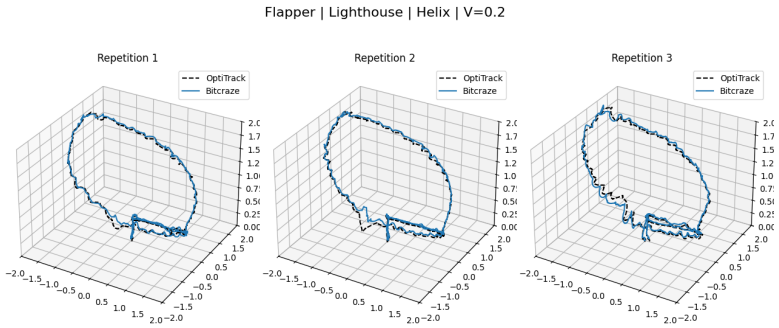
(a) 3D Error (m) - XY-plane



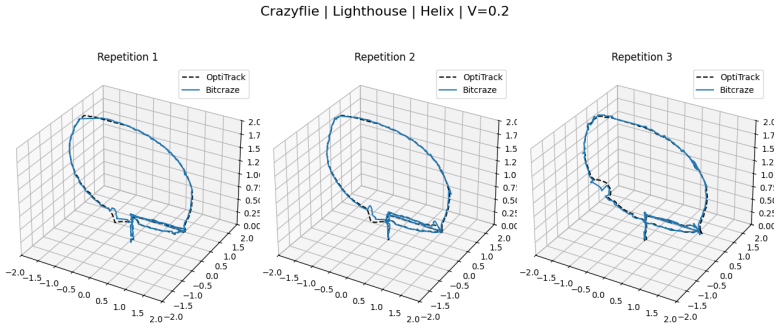
(b) 3D Error (m) - XZ-plane

Figure A.3: 3D errors on 2D planes

Figure A.4 shows flight trajectories of both the Crazyflie and the Flapper combined with the Lighthouse system



(a) Flapper - Lighthouse System



(b) Crazyflie - Lighthouse system

Figure A.4: Helix flight V=0.2 - Flight trajectory

Appendix B

Hardware

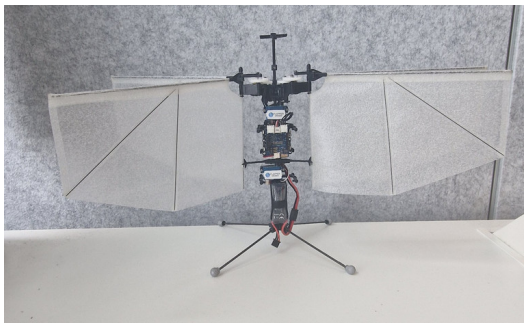
Product	Quantity	Store
Flapper Nimble+	1	Flapper Drones (Lighthouse Bundle)
Steam VR Base Station 2.0 (V2)	2	
Lighthouse Deck	1	
Crazyradio 2.0	1	
Crazyflie 2.1 Brushless	1	Bitcraze (Loco explorer Bundle - Crazyflie 2.1 Brushless)
Loco Positioning Nodes	8	
Loco Positioning Deck	1	
OptiTrack Cameras	8	Unknown

Table B.1: Specifics of the hardware

Part	Weight (gram)
Flapper + Lighthouse Mount	112
Crazyflie 2.1 Brushless	37
Body Panels	5-6
Lighthouse Deck	3
Lighthouse Deck + Cable	5
Loco Deck	3
OptiTrack Balls 4x (d=1.5cm)	1.5
OptiTrack Hollow Balls 4x (d=1.0cm)	0.5

Table B.2: Approximated weight of hardware

B.1 Platforms



(a) The Flapper Nimble+

(b) The Crazyflie 2.1 Brushless

Figure B.1: Flying platforms

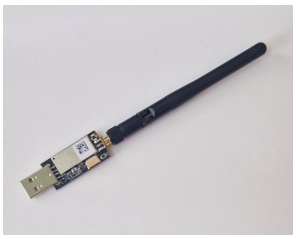
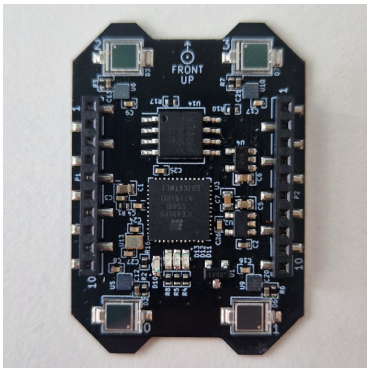


Figure B.2: Bitcraze's USB Dongle - the Crazyradio 2.0

B.2 Lighthouse Positioning System

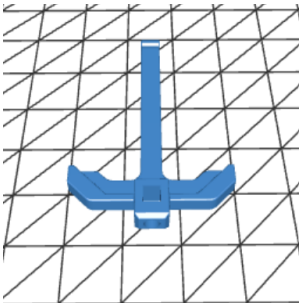
(a) Base station 1 (±115°)



(b) Lighthouse deck

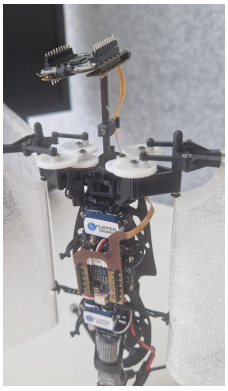
(c) Base station 2 (±120°)

Figure B.3: Lighthouse Positioning System Hardware



(a) 3D mount

(b) Physical mount



(c) Attached Lighthouse deck

Figure B.4: Lighthouse Deck on the Flapper

Figure B.5: Lighthouse deck on the Crazyflie

Geometry

Type	Station	X	Y	Z	Deck Height
Flapper	1	1.489843	-1.713750	2.622439	± 0.255
	2	-1.833724	2.120603	2.626641	± 0.255
Crazyflie	1	1.471097	-1.723236	2.813884	± 0.035
	2	-1.796200	2.124095	2.819597	± 0.035

Table B.3: Estimated Geometry (m) by Lighthouse Stations

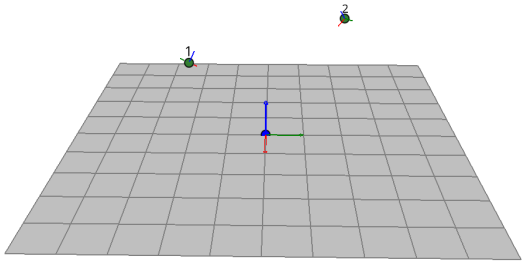
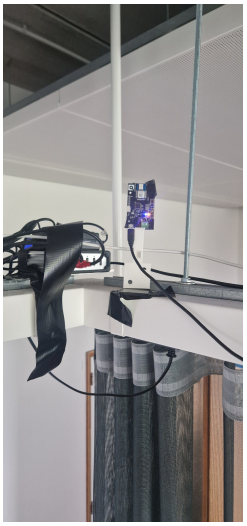


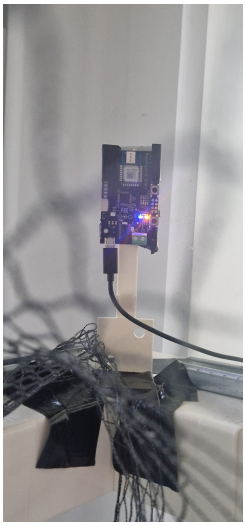
Figure B.6: Estimated Geometry Lighthouse Stations (CFclient)

B.3 Loco Positioning System

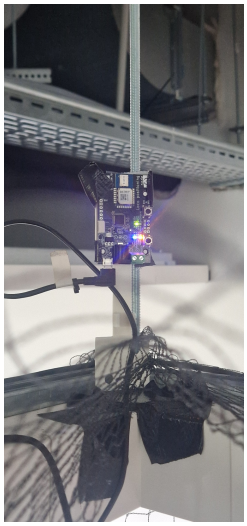


(a) Loco Node 0

(b) Loco Node 1

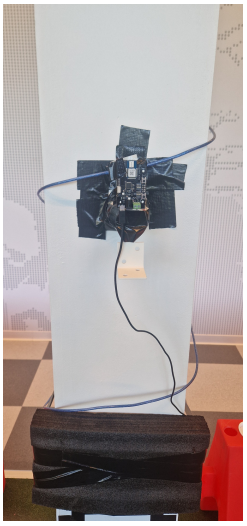


(c) Loco Node 2



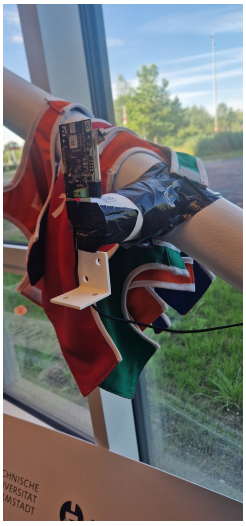
(d) Loco Node 3

Figure B.7: Upper Loco Positioning Nodes

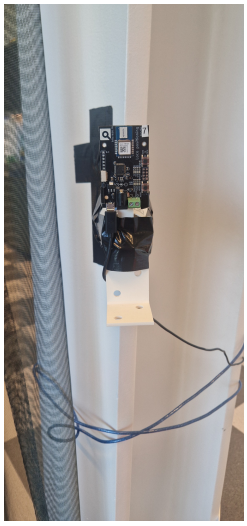


(a) Loco Node 4

(b) Loco Node 5

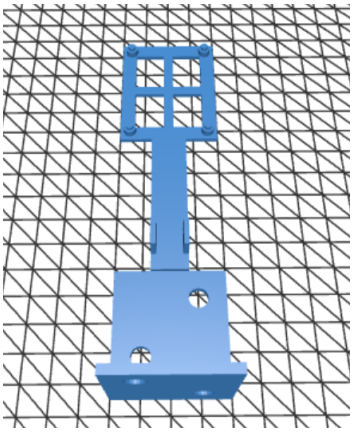


(c) Loco Node 6

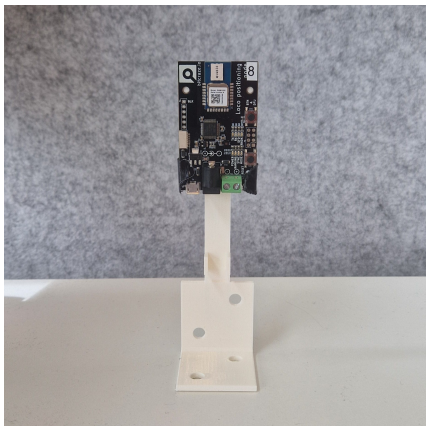


(d) Loco Node 7

Figure B.8: Lower Loco Positioning Nodes



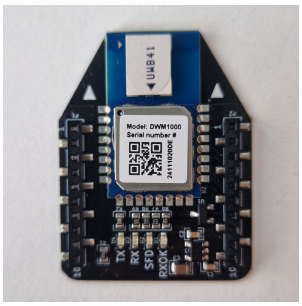
(a) 3D-model



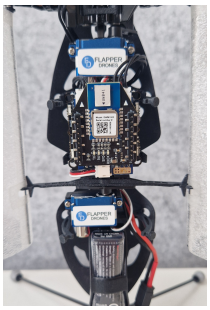
(b) 3D-printed

Figure B.9: Loco Anchor Stand provided by Bitcraze

(a) Deck on the Crazyflie



(b) Loco deck



(c) Deck on the Flapper

Figure B.10: Caption

Geometry

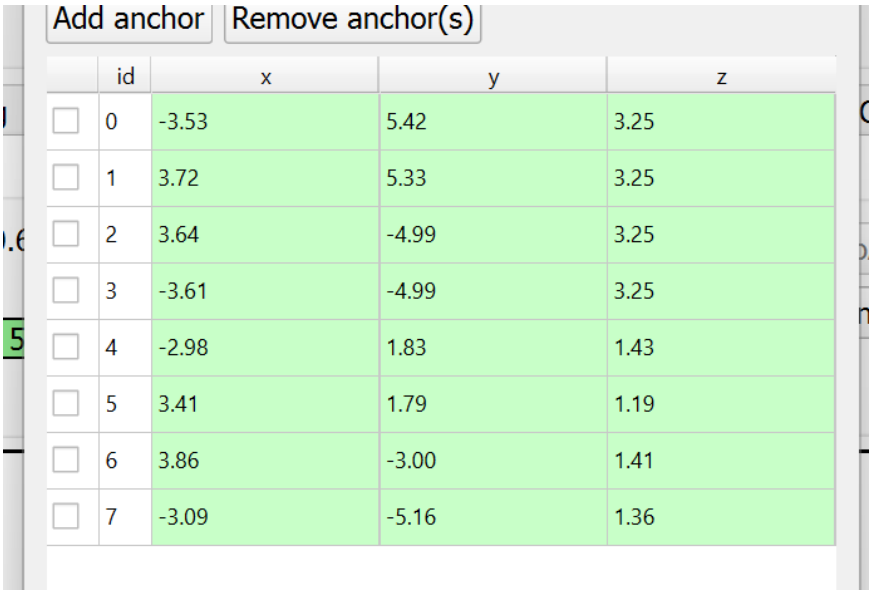


Figure B.11: Loco Positioning System: Measured Geometry (m)

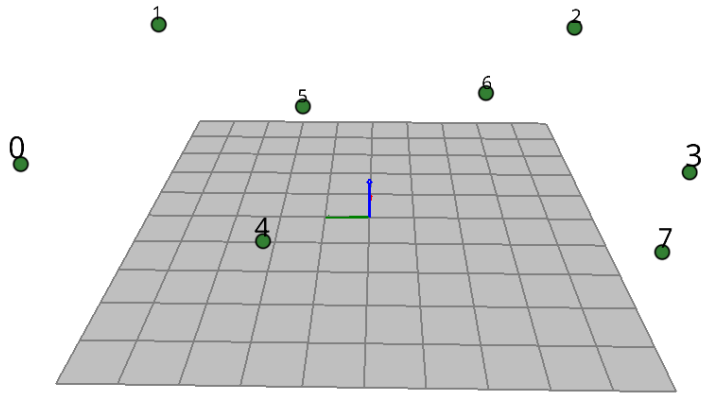
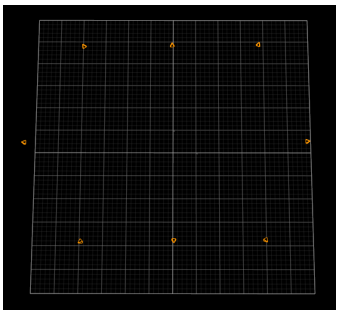


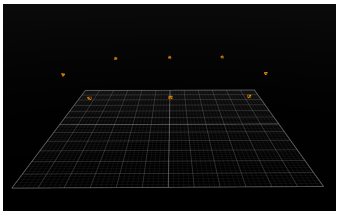
Figure B.12: Loco Positioning System - 3D Geometry (CFclient)

B.4 OptiTrack System



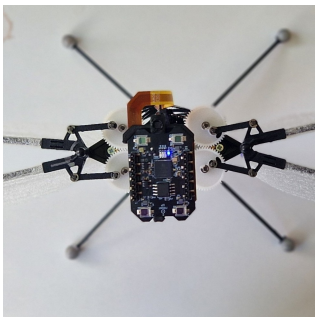
(a) Top View (Motive)

(b) Physical camera 1



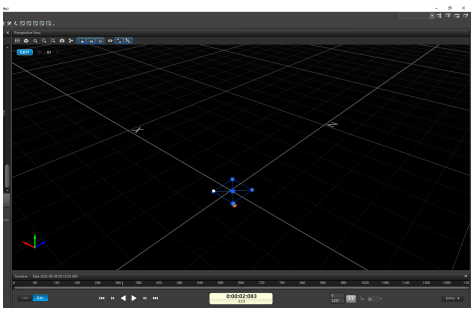
(c) Side View (Motive)

Figure B.13: OptiTrack cameras



(a) Top view

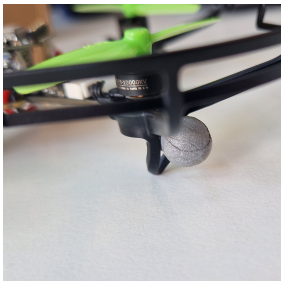
(b) Zoomed in



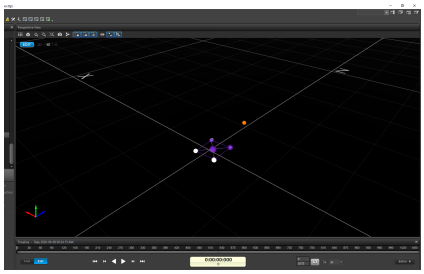
(c) Rigid Body (Motive)

Figure B.14: Placement of OptiTrack balls on the Flapper

(a) Top view



(b) Zoomed in



(c) Rigid Body (Motive)

Figure B.15: Placement of OptiTrack balls on the Crazyflie

Figure B.16: View from OptiTrack cameras

Appendix C

Experiments

C.1 Experimental Area

(a) Intelligent Robotics Lab - North side

(b) Intelligent Robotics Lab - South side

Figure C.1: Intelligent Robotics Lab

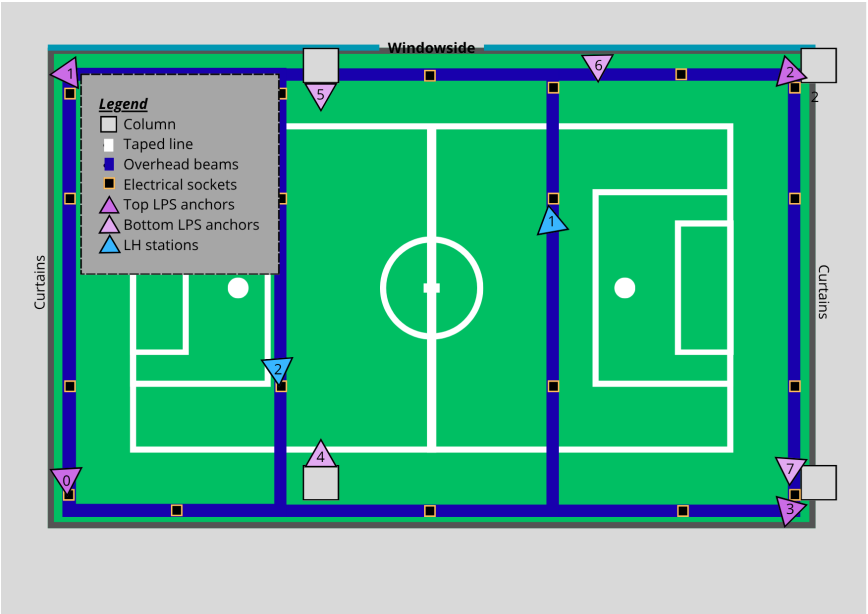


Figure C.2: Intelligent Robotics Lab - 2D field

C.2 Experimental Protocol

Phase 1: Hardware Setup

1. Power on the Lighthouse, Loco and OptiTrack system by flipping on the black switch
2. Check the positions of the Lighthouse and Loco system
3. Power on the OptiTrack Computer
4. Connect the CrazyRadio 2.0
5. Lower the sunscreens
6. Close the curtains for safety
7. Remove possible obstacles of the field
8. Power on the drone

Phase 2: Software setup

1. Recalibrate OptiTrack system using the L-calibration staff (daily)
2. Reconfigure drone with the proper deck through the CFclient
   If new combination on the day:
   (a) Create a new section in the OptiTrack
   (b) Record five seconds of static position at three different locations for recalculation of the translation matrix. $[[0.0.], [0, -0.75], [-0.75, 0]]$
3. Check the activity of the positioning systems

Phase 3: Experiment

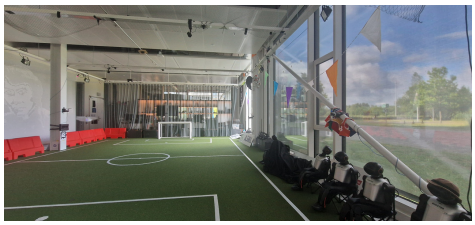
1. Place the drone at approximately the origin.
2. Check the YAML file for proper configuration
3. Start the run.py script
4. Start the countdown and the same time of the start of connection with the drone platform.
5. Stop the OptiTrack file when the scripts says stop

Phase 4: Data gathering

1. Remove take/file if drone failed to perform the experiment and redo.
2. Export the takes of the OptiTrack as .csv and use the USB-stick to save them on the laptop. Place all in correct folder

C.3 Experimental circumstances

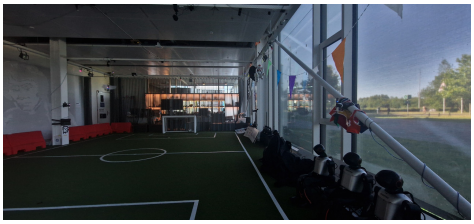
(a) 04-06 - 0/3 screens, sunlight, lights off



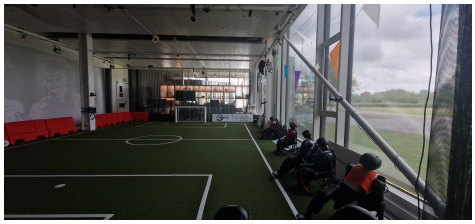
(b) 05-06 - 2/3 screens, cloudy, lights off

(c) Morning 08-06 - 3/3 screens, cloudy, lights on

(d) Midday 08-06 - 3/3 screens, sunny, lights on



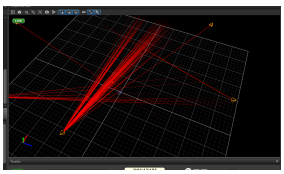
(e) 09-06: 3/3 screens, cloudy, lights off



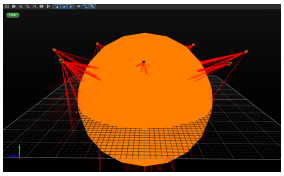
(f) 12-06: 3/3 screens, cloudy, lights off

(g) 15-06: 3/3 screens, cloudy, lights on

Figure C.3: Experimental Area - Circumstances



(a) 08-06: 3/3 screens, sunny, lights on



(b) 12-6: 0/3 screens, sunny, lights on

Figure C.4: OptiTrack circumstances: Afternoons

As seen in Figure C.4, when it was sunny outside, the OptiTrack saw sunlight as an object, making it unable to track the much smaller Optitrack markers. This is also why some experiments done on 04-06 (Figure C.3a) needed to be redone, because the ground truth was missing due to the interference.

Appendix D

Software

D.1 CFclient

CFclient is Bitcraze's graphic user interface to control and monitor their drones. A laptop using the Crazyradio can connect through radio. To fly you either need a motion controller connected, or a positioning system. The positioning systems can be configured through the GUI, and if they're properly the drone can be commanded by Command Based Flight Control becomes active. The Flapper is then auto-armed, meaning it's ready to fly. For the Crazyflie you have to first press the green box labelled 'arm' to activate the flight control. To fly using a controller, this is the same situation.

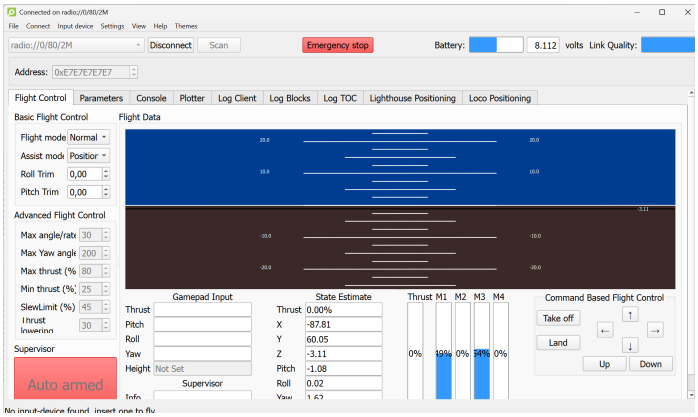


Figure D.1: GUI CFclient

Tab	Use
Flight control	State estimation and control settings for flight + commander
Parameters	Settings for parameters
Console	Output terminal for printouts while running Bitcraze's firmware
Plotter	Plotter for variables
Log Client	Output terminal for log messages
Log Blocks	Control for which configuration blocks to log/save
Log TOC	Additional information of all logging blocks
Lighthouse Positioning	View/configuration of Lighthouse system when present
Loco Positioning	View/configuration of Loco system when present

Table D.1: CFclient Tabs

D.2 Python Scripts

.YAML file

.yaml files were used to set the parameters for the flights. These defined which experiment was run on what platform and which deck. It also set the height, distance and speed to be flown.

```
systems:
  - "Flapper"
  - "Crazyflie"
decks:
  - "bcLighthouse4"
  - "bcLoco"
experiments:
  - "Motionless"
  - "Static"
  - "LinearX"
  - "LinearY"
  - "LinearXY"
  - "Circle"
  - "Helix"
  - "Staircase"


active_setup:
  platform: "Crazyflie"
  deck: "bcLighthouse4"
  experiment: "Circle"


matrix:
  distances: [1.5]
  heights: [1.0]
  speeds: [0.2, 0.5]
  repetitions: 3
```

Run.py

This is the script used to run the experiments, using the command 'python experiments/scripts/run.py experiments/setup/{experiment}.yaml', where experiment referred to the flightpattern.yaml file.

```
import logging
import sys
import time
import cflib.crtp
import sys
import yaml
import itertools
from datetime import datetime
from cflib.crazyflie import Crazyflie
from cflib.crazyflie.log import LogConfig
from cflib.crazyflie.syncCrazyflie import SyncCrazyflie
from cflib.utils import uri_helper
from mission_manager import MissionManager

URI = uri_helper.uri_from_env(default='radio://0/80/2M/E7E7E7E7E7')
logging.basicConfig(level=logging.ERROR)


def load_matrix_config():
    """
    Loads yaml file: one for each experiment.
    Has platform, deck, experiment, height, distance, speed as parameters
    """
    if len(sys.argv) < 2:
        print("Error: no YAML file")
        sys.exit(1)
    with open(sys.argv[1], 'r') as file:
        return yaml.safe_load(file)


if __name__ == '__main__':
    config = load_matrix_config()
    active = config['active_setup']
    PLATFORM = active['platform']
    DECK = active['deck']
    EXPERIMENT = active['experiment']

    matrix = config['matrix']
    distances = matrix['distances']
    heights = matrix['heights']
    speeds = matrix['speeds']
    repetitions = matrix['repetitions']

    # calculates all runs
    all_runs = list(itertools.product(distances, heights, speeds, range(1, repetitions + 1)))

    print(f"\n==========================================")
    print(f" Configuration loaded: {len(all_runs)} flights generated.")
    print(f" Experiment: {EXPERIMENT} op {PLATFORM} ({DECK})")
    print(f"==========================================")

    ans = input(f" Start with {len(all_runs)} flights? ")
    if ans.lower() not in ['y', 'yes', 'ja', 'j', '1']:
        sys.exit(1)

    LighthouseActive = (DECK == 'bcLighthouse4')
    LocoActive = (DECK == 'bcLoco')
    cflib.crtp.init_drivers()

    for dist, height, speed, rep in all_runs:
        print(f" \n Next Run: Dist={dist}m | Height={height}m | Speed={speed}m/s | Repetition={rep}/{repetitions}")

        ready = input(" Press enter to start or (q)uit\n ")
        if ready.lower() in ['q', 'quit']:
            break

        date = datetime.now()
        filedate = date.strftime("%Y-%m-%d_%H-%M-%S")

        print("================= START ==================")

        mission = MissionManager(
            platform=PLATFORM,
            deck=DECK,
            experiment=EXPERIMENT,
            time=filedate,
            default_distance=dist,
            default_height=height,
            default_speed=speed
        )

        # ms for loggings
        log_period = 10

        with SyncCrazyflie(URI, cf=Crazyflie(rw_cache='./cache')) as scf:
            try:
                time.sleep(1)

                # ---------------------- LOGCONFIGS ----------------------
                logconf_pos = LogConfig(name='Position', period_in_ms=log_period)
                logconf_pos.add_variable('stateEstimate.x', 'float')
                logconf_pos.add_variable('stateEstimate.y', 'float')
                logconf_pos.add_variable('stateEstimate.z', 'float')
                logconf_pos.add_variable('stateEstimate.roll', 'float')
                logconf_pos.add_variable('stateEstimate.pitch', 'float')
                logconf_pos.add_variable('stateEstimate.yaw', 'float')

                vbat_log = LogConfig(name='BatteryCheck', period_in_ms=100)
                vbat_log.add_variable('pm.vbat', 'float')

                logconf_kal = LogConfig(name='KalmanVariance', period_in_ms=log_period)
                logconf_kal.add_variable('kalman.varPX', 'float')
```

```
        logconf_kal.add_variable('kalman.varPY', 'float')
        logconf_kal.add_variable('kalman.varPZ', 'float')
        logconf_kal.add_variable('kalman.varX', 'float')
        logconf_kal.add_variable('kalman.varY', 'float')
        logconf_kal.add_variable('kalman.varZ', 'float')

        logconf_vel = LogConfig(name='Velocity', period_in_ms=log_period)
        logconf_vel.add_variable('stateEstimate.vx', 'float')
        logconf_vel.add_variable('stateEstimate.vy', 'float')
        logconf_vel.add_variable('stateEstimate.vz', 'float')
        logconf_vel.add_variable('ctrltarget.vx', 'float')
        logconf_vel.add_variable('ctrltarget.vy', 'float')
        logconf_vel.add_variable('ctrltarget.vz', 'float')

        logconf_cont = LogConfig(name='ControlVariables', period_in_ms=log_period)
        logconf_cont.add_variable('ctrltarget.x', 'float')
        logconf_cont.add_variable('ctrltarget.y', 'float')
        logconf_cont.add_variable('ctrltarget.z', 'float')


        # adds lighthouse and loco parameters
        if LighthouseActive:
            logconf_light = LogConfig(name='Lighthouse', period_in_ms=log_period)
            logconf_light.add_variable('lighthouse.status', 'float')
            logconf_light.add_variable('lighthouse.bsReceive', 'float')
        elif LocoActive:
            logconf_loco1 = LogConfig(name='Loco', period_in_ms=log_period)
            logconf_loco1.add_variable('ranging.distance0', 'float')
            logconf_loco1.add_variable('ranging.distance1', 'float')
            logconf_loco1.add_variable('ranging.distance2', 'float')
            logconf_loco1.add_variable('ranging.distance3', 'float')

            logconf_loco2 = LogConfig(name='Loco2', period_in_ms=log_period)
            logconf_loco2.add_variable('ranging.distance4', 'float')
            logconf_loco2.add_variable('ranging.distance5', 'float')
            logconf_loco2.add_variable('ranging.distance6', 'float')
            logconf_loco2.add_variable('ranging.distance7', 'float')


        # adds logging to configuration
        scf.cf.log.add_config(vbat_log)
        scf.cf.log.add_config(logconf_cont)
        scf.cf.log.add_config(logconf_pos)
        scf.cf.log.add_config(logconf_vel)
        scf.cf.log.add_config(logconf_kal)


        # asks for update
        if LighthouseActive:
            scf.cf.log.add_config(logconf_light)
            logconf_light.data_received_cb.add_callback(mission.log_lighthouse)

        elif LocoActive:
            scf.cf.log.add_config(logconf_loco1)
            scf.cf.log.add_config(logconf_loco2)
            logconf_loco1.data_received_cb.add_callback(mission.log_loco1)
            logconf_loco2.data_received_cb.add_callback(mission.log_loco2)

        logconf_kal.data_received_cb.add_callback(mission.log_kalman)
        logconf_cont.data_received_cb.add_callback(mission.log_control)
        vbat_log.data_received_cb.add_callback(mission.log_battery)
        logconf_pos.data_received_cb.add_callback(mission.log_position)
        logconf_vel.data_received_cb.add_callback(mission.log_velocity)

        # ---------------------------- HARDWARE CHECKUPS ----------------------------------
        vbat_log.start()

        # checks for deck
        scf.cf.param.add_update_callback(group="deck", name=mission.deck, cb=mission.param_deck)
        scf.cf.param.request_param_update(f"deck.{mission.deck}")
        start_wait = time.time()
        while not mission.deck_attached_event.is_set():
            time.sleep(0.1)
            if time.time() - start_wait > 3.0:
                print(f"ERROR: Correct deck ({mission.deck}) is NOT attached or not responding!")
                sys.exit(1)
        time.sleep(1)

        # forces kalman filter to reset
        try:
            scf.cf.param.set_value("kalman.resetEstimation", "1")
            time.sleep(2.0)
            print("Kalman filter reset")
        except Exception as ke:
            print(f"Kalman filter failed {ke}")

        # ---------------------------- START EXPERIMENT ----------------------------------
        scf.cf.platform.send_arming_request(False)
        time.sleep(1)
        mission.start_csv()

        print("Start logging")
        logconf_cont.start()
        logconf_pos.start()
        logconf_kal.start()
        logconf_vel.start()
        if LighthouseActive:
            logconf_light.start()
        elif LocoActive:
            logconf_loco1.start()
            logconf_loco2.start()

        print("Arming")
        scf.cf.platform.send_arming_request(True)
        time.sleep(0.5)

        print("Flying")
        mission.perform_flight(scf)

    except Exception as e:
        print(f"Error: {e}")
```

```
# ---------------------------- SHUTTING DOWN -----------------------------------
finally:
    # succes: flight was performed
    if mission.is_open:
        print("Disarming")
        scf.cf.platform.send_arming_request(False)

        print("Stop logging")
        logconf_pos.stop()
        vbat_log.stop()
        logconf_cont.stop()
        logconf_kal.stop()
        logconf_vel.stop()
        if LighthouseActive:
            logconf_light.stop()
        elif LocoActive:
            logconf_loco1.stop()
            logconf_loco2.stop()
        time.sleep(1)

        mission.close()
        print(f"Saved in {mission.file_path}")

        print("================= STOP ====================")
    else:
        print("Try again!")

    scf.close_link() # very important! Enables multiple consequent script runs
```

Flight_commander.py

This was the script that defined all experiments and was used to control the drones.

```
import time
import math

def motionless_flight(distance:int=0, height:int=0.0, speed:int=0.0):
    time.sleep(5)


def static_flight(pc, distance:int= 1.5, height:int=0.5, speed:int=0.2):
    """
    Static flight for 30 seconds at height metres
    """
    pc.go_to(0.0, 0.0, height)
    time.sleep(30)


def linearx_flight(pc, distance:int= 1.5, height:int=1.0, speed:int=0.2):
    """
    Dynamic flight: rise, fly backward, fly forward, land.
    """
    # RISE
    pc.go_to(0.0, 0.0, height)
    time.sleep(0.5)

    # GO BACK
    pc.go_to(-distance, 0.0, height)

    # GO FORWARD
    pc.go_to(distance, 0.0, height)

    # RETURN
    time.sleep(0.5)
    pc.go_to(0.0, 0.0, height)


def lineary_flight(pc, distance:int= 1.5, height:int=1.0, speed:int=0.2):
    """
    Dynamic flight: rise, fly left, fly right, land.
    """
    # RISE
    pc.go_to(0.0, 0.0, height)
    time.sleep(0.5)

    # GO FORWARD
    pc.go_to(0.0, distance, height)

    # GO BACK
    pc.go_to(0.0, -distance, height)

    # RETURN
    time.sleep(0.5)
    pc.go_to(0.0, 0.0, height)


def linearxy_flight(pc, distance:int= 1.5, height:int=1.0, speed:int=0.2):
    """
    Dynamic flight: rise, fly topleft fly bottomright, land.
    """
    # RISE
    pc.go_to(0.0, 0.0, height)
    time.sleep(0.5)

    # GO FORWARD
    pc.go_to(distance, distance, height)

    # GO BACK
    pc.go_to(-distance, -distance, height)

    # RETURN
    time.sleep(0.5)
    pc.go_to(0.0, 0.0, height)


def square_flight(pc, distance:int=1.5, height:int=1.0, speed:int=0.2):
    """
    Square flight: make a square
    """
    # rise up
    pc.go_to(distance, 0.0, height)
    time.sleep(0.5)

    # go to top right corner
    pc.go_to(distance, -distance, height)

    # go to bottom right corner
    pc.go_to(-distance, -distance, height)

    # go to bottom left corner
    pc.go_to(-distance, distance, height)

    # go to top left corner
    pc.go_to(distance, distance, height)

    # go to original position
    pc.go_to(distance, 0.0, height)

    # RETURN
    time.sleep(0.5)
    pc.go_to(0.0, 0.0, height)


def circle_flight(pc, distance: float = 1.5, height: float = 0.5, speed: float = 0.2):
    """
    Circle flight around (0,0) at a constant height.
    """
    rad = distance
    steps = 40
```

```
    # calculate needed time for circle for speed
    circumference = 2 * math.pi * rad
    total_time = circumference / speed
    time_per_step = total_time / steps

    pc.go_to(rad, 0.0, height)
    time.sleep(1.5)

    # move step for circle
    for i in range(steps + 1):
        angle = (2 * math.pi / steps) * i
        x = rad * math.cos(angle)
        y = rad * math.sin(angle)
        pc._cf.high_level_commander.go_to(x, y, height, 0, time_per_step, relative=False)
        time.sleep(time_per_step)

    pc.go_to(0.0, 0.0, height)
    time.sleep(1)


def helix_flight(pc, distance:float=1.5, height:float=0.5, speed:float=0.2):
    """
    Helix flight around (0,0).
    Enforces the exact YAML speed by utilizing the high_level_commander with calculated step durations.
    """
    rad = distance
    steps = 40

    # calculates time steps
    circumference = 2 * math.pi * rad
    total_time = circumference / speed
    time_step = total_time / steps

    # calculates height steps
    max_height = 2.0
    h_steps = (max_height - height) / (steps // 2)

    h = height
    pc.go_to(rad, 0.0, h)
    time.sleep(1)

    for i in range(steps + 1):
        # computes new z
        if i < (steps//2):
            h += h_steps
        elif i > (steps//2) and h > height:
            h -= h_steps

        # changes new x, y
        angle = (2 * math.pi / steps) * i
        x = rad * math.cos(angle)
        y = rad * math.sin(angle)

        pc._cf.high_level_commander.go_to(x, y, h, 0, time_step, relative=False)

        time.sleep(time_step)

    pc.go_to(0.0, 0.0, height)
    time.sleep(1)


def staircasex_flight(pc, distance:int= 1.5, height:int=0.5, speed:int=0.2):
    max_height = 2.0
    max_distance = distance * 2
    staircase_steps = 4

    # computes stepsize
    h_steps = (max_height - height) / staircase_steps
    d_steps = max_distance / staircase_steps

    pc.go_to(0.0, 0.0, height)
    time.sleep(0.5)

    # start staircase
    d = -distance
    pc.go_to(d, 0.0, height)

    d += d_steps
    pc.go_to(d, 0.0, height)

    height += h_steps
    pc.go_to(d, 0.0, height)

    d += d_steps
    pc.go_to(d, 0.0, height)

    height += h_steps
    pc.go_to(d, 0.0, height)

    d += d_steps
    pc.go_to(d, 0.0, height)

    height += h_steps
    pc.go_to(d, 0.0, height)

    time.sleep(0.5)

    # return
    pc.go_to(0, 0)
```

Mission_manager.py

This was the script to control the loggings and the making of the .CSV files per experiment run.

```
from pathlib import Path
import csv
from cflib.positioning.position_hl_commander import PositionHlCommander
import sys
import flight_patterns
from threading import Event
import time


class MissionManager:
    def __init__(self, platform: str, deck: str, experiment: str,
    time: str,  default_distance=1.5, default_height:int=0.5, default_speed=0.2):
        """
        Initializes current mission with platform, deck and experiment
        """
        self.platform = platform
        self.deck = deck
        self.time = time

        self.file_path = Path('data/bitcraze') / platform / deck[2:] / experiment / f"{platform[0]}{deck[2:5]}-{experiment}-D{
        self.experiment = experiment
        self.battery_printed = False
        self.current_vbat = 0.0
        self.is_open = False
        self.H = default_height
        self.V = default_speed
        self.D = default_distance
        self.deck_attached_event = Event()

        nan_val = float('nan')
        self.kx = nan_val
        self.ky = nan_val
        self.kz = nan_val
        self.kvx = nan_val
        self.kvy = nan_val
        self.kvz = nan_val
        self.vx = nan_val
        self.vy = nan_val
        self.vz = nan_val
        self.target_vx = nan_val
        self.target_vy = nan_val
        self.target_vz = nan_val
        self.lh_status = nan_val
        self.lh_receive = nan_val
        self.lc_rd0 = nan_val
        self.lc_rd1 = nan_val
        self.lc_rd2 = nan_val
        self.lc_rd3 = nan_val
        self.lc_rd4 = nan_val
        self.lc_rd5 = nan_val
        self.lc_rd6 = nan_val
        self.lc_rd7 = nan_val

    def start_csv(self):
        """
        Creates csv file for parameters
        """
        self.file_path.parent.mkdir(parents=True, exist_ok=True)
        self.csv_file = open(self.file_path, mode='w', newline='')
        self.writer = csv.writer(self.csv_file)
        self.writer.writerow([f"#Time:{self.time}, Platform:{self.platform}, Deck:{self.deck},
        Experiment:{self.experiment}, Parameters: [H:{self.H}, D:{self.D}, V:{self.V}]"])
        self.writer.writerow(['TimestampMS', 'TimeUnix', 'Vbat',
                              'X', 'Y', 'Z',
                              'VarX', 'VarY', 'VarZ',
                              'TargetX', 'TargetY', 'TargetZ',
                              'Roll', 'Pitch', 'Yaw',
                              'VX', 'VY', 'VZ',
                              'VarPX', 'VarPY', 'VarPZ',
                              'TargetVX', 'TargetVY', 'TargetVZ',
                              'lhStatus', 'lhReceive',
                              'lcRng0', 'lcRng1', 'lcRng2', 'lcRng3',
                              'lcRng4', 'lcRng5', 'lcRng6', 'lcRng7'])

        self.is_open = True

    def close(self):
        """
        Closes csv file
        """
        self.csv_file.close()
        self.is_open = False

    def param_deck(self, _, value_str):
        """
        Checks if correct deck is attached
        """
        value = int(value_str)
        if value:
            self.deck_attached_event.set()

    def log_battery(self, timestamp, data, logconf):
        """
        Checks if battery is full enough for a flight
        """
        voltage = data['pm.vbat']
        self.current_vbat = voltage

        if voltage == 0.0:
            return

        # different voltage for different platform
        if self.platform == 'Flapper':
            MIN_BATTERY_VOLTAGE = 6.5
        elif self.platform == 'Crazyflie':
```

```
        MIN_BATTERY_VOLTAGE = 3.5

    # checks battery
    if not self.battery_printed:
        if voltage >= MIN_BATTERY_VOLTAGE:
            print(f'Battery level is good: {voltage:.2f}V')
            self.battery_printed = True
        else:
            print(f'ERROR: Battery too low ({voltage:.2f}V)!')
            sys.exit(1)

def log_kalman(self, timestamp, data, logconf):
    """
    Logs Kalman variance for position and velocity
    """
    self.kx = data.get('kalman.varX', 0.0)
    self.ky = data.get('kalman.varY', 0.0)
    self.kz = data.get('kalman.varZ', 0.0)
    self.kvx = data.get('kalman.varPX', 0.0)
    self.kvy = data.get('kalman.varPY', 0.0)
    self.kvz = data.get('kalman.varPZ', 0.0)

def log_velocity(self, timestamp, data, logconf):
    """
    Logs velocity and target velocity
    """
    self.vx = data.get('stateEstimate.vx', 0.0)
    self.vy = data.get('stateEstimate.vy', 0.0)
    self.vz = data.get('stateEstimate.vz', 0.0)
    self.target_vx = data.get('ctrltarget.vx', 0.0)
    self.target_vy = data.get('ctrltarget.vy', 0.0)
    self.target_vz = data.get('ctrltarget.vz', 0.0)

def log_control(self, timestamp, data, logconf):
    """
    Logs target position
    """
    self.target_x = data.get('ctrltarget.x', 0.0)
    self.target_y = data.get('ctrltarget.y', 0.0)
    self.target_z = data.get('ctrltarget.z', 0.0)

def log_lighthouse(self, timestamp, data, logconf):
    """
    Logs lighthouse status
    """
    self.lh_status = data.get('lighthouse.status', 0.0)
    self.lh_receive = data.get('lighthouse.bsReceive', 0.0)

def log_loco1(self, timestamp, data, logconf):
    """
    Logs loco status top anchors
    """
    self.lc_rd0 = data.get('ranging.distance0', 0.0)
    self.lc_rd1 = data.get('ranging.distance1', 0.0)
    self.lc_rd2 = data.get('ranging.distance2', 0.0)
    self.lc_rd3 = data.get('ranging.distance3', 0.0)

def log_loco2(self, timestamp, data, logconf):
    """
    Logs loco status bottom anchors
    """
    self.lc_rd4 = data.get('ranging.distance4', 0.0)
    self.lc_rd5 = data.get('ranging.distance5', 0.0)
    self.lc_rd6 = data.get('ranging.distance6', 0.0)
    self.lc_rd7 = data.get('ranging.distance7', 0.0)

def log_position(self, timestamp, data, logconf):
    """
    Logs state estimation x, y and z, and roll, pitch and yaw.
    """
    self.x = data.get('stateEstimate.x', 0.0)
    self.y = data.get('stateEstimate.y', 0.0)
    self.z = data.get('stateEstimate.z', 0.0)
    self.roll = data.get('stateEstimate.roll', 0.0)
    self.pitch = data.get('stateEstimate.pitch', 0.0)
    self.yaw = data.get('stateEstimate.yaw', 0.0)

    logging_variables = [timestamp, time.time(), self.current_vbat,
                         self.x, self.y, self.z,
                         self.kx, self.ky, self.kz,
                         self.target_x, self.target_y, self.target_z,
                         self.roll, self.pitch, self.yaw,
                         self.vx, self.vy, self.vz,
                         self.kvx, self.kvy, self.kvz,
                         self.target_vx, self.target_vy, self.target_vz
                         ]
    nan_val = float('nan')
    if self.deck=='bcLighthouse4':
        logging_variables += [self.lh_status, self.lh_receive] + [nan_val for _ in range(8)]
    elif self.deck=='bcLoco':
        logging_variables += [nan_val, nan_val, self.lc_rd0, self.lc_rd1, self.lc_rd2, self.lc_rd3, self.lc_rd4,
            self.lc_rd5, self.lc_rd6, self.lc_rd7]

    self.writer.writerow(logging_variables)

def perform_flight(self, scf):
    """
    General function to call on experiment functions
    """
    # finds initial position through positioning system
    try:
        init_x = self.x
        init_y = self.y

    except Exception as e:
        # sets starting coordinate as origin
        print(f"Error: Set 0.0 automatically {e}")
        init_x = 0.0
        init_y = 0.0

    print(f"Executing {self.experiment}")
```

```
experiment_name = self.experiment.lower() + "_flight"

if experiment_name == 'motionless_flight':
    flight_patterns.motionless_flight()
else:
    # uses high level commander for flight control - more precise than motion commander
    with PositionHlCommander(scf, x=init_x, y=init_y, default_height=self.H, default_velocity=self.V,
        default_landing_height=0.05, controller=PositionHlCommander.CONTROLLER_PID) as pc:
        try:
            # finds flight pattern in flight_patterns.py
            flight_function = getattr(flight_patterns, experiment_name)
            flight_function(pc, distance=self.D, height=self.H, speed=self.V)
        except AttributeError:
            print(f"ERROR: {self.experiment} does not exist!")
            sys.exit(1)
```

Appendix E

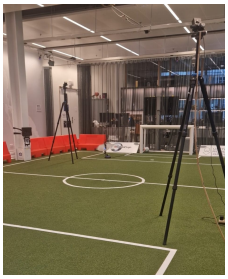
Test phase

E.1 Lighthouse Positioning System

These were the original test setups for the Lighthouse Positioning System. These were meant to test the range of the base stations.

(a) 90 degrees

(b) Diagonal (< 2x2m)



(c) Diagonal(±2x2m)

Figure E.1: Lighthouse System setups

Setup	**Station**	**X**	**Y**	**Z**
90 degrees	1	1.25	-1.81	1.91
	2	1.36	1.70	1.85
Diagonal (< 2x2m)	1	1.99	-2.35	1.85
	2	-1.95	2.21	1.85
Diagonal (> 2x2m)	1	2.49	2.18	2.24
	2	-2.26	-1.76	2.20

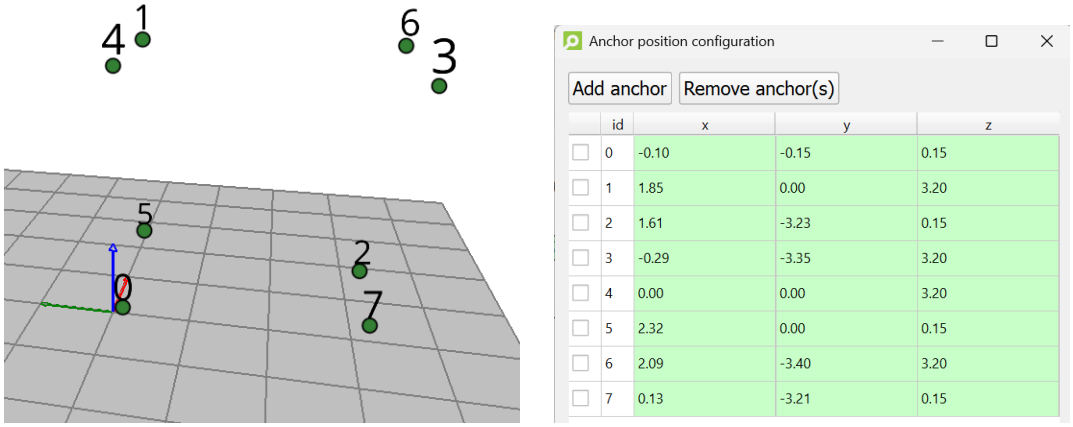
Table E.1: Estimated geometry per setup

E.2 Loco Positioning System

This was the original setup for the Loco Positioning System. Originally this was only tested on the Crazyflie. However, because of the other uses of the field, it became evident anchors close to the ground were not practical.

(a) Top Anchor (b) Ground anchor

Figure E.2: Loco System Setup Anchors



(a) 3D Geometry (CFclient) (b) Coordinates (CFclient)

Figure E.3: Loco System Original Setup

E.3 OptiTrack

Figure E.4 shows the original placement of the 1.5cm-OptiTrack balls, having placed two balls on the 'feet' of the Flapper, and one on its head. However, creating a triangle caused an imbalance, making the Flapper wobbly during flight. To stabilise this, two more balls were placed on the other legs. The Flapper remained wobbly and had more trouble taking flight, suggesting the Flapper was too heavy. Thus these balls were replaced by the hollow 0.5-balls.

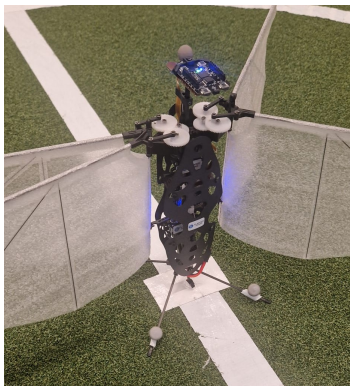
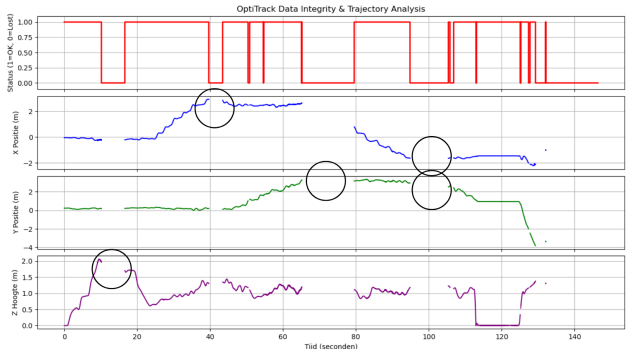
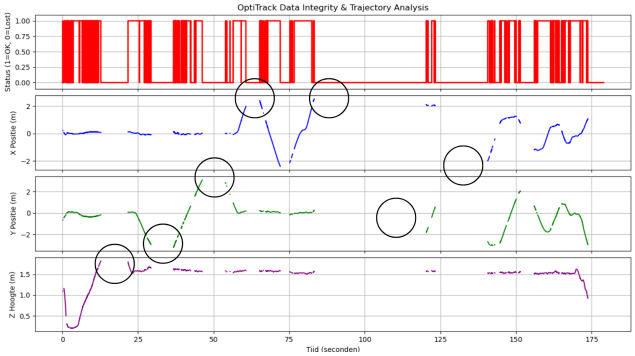


Figure E.4: Original placement of OptiTrack spheres (triangle)

Figure E.5 shows the range of the OptiTrack during a test flight with the Crazyflie and manual movement using a calibration staff. It showed a lack in data for any axis that had a distance greater than 1.8-2.0.



(a) Tracking crazyflie



(b) Tracking calibration staff

Figure E.5: Range of OptiTrack (limits circled with black)