Signal Processing Systems
Mekelweg 4,
2628 CD Delft
The Netherlands
https://sps.ewi.tudelft.nl/

SPS-2025-4908058

M.Sc. Thesis

Distributed Affine Formation Control with Quadcopters

Shaan Hossain

Abstract

The usage of robots replacing human tasks has become more prevalent. Controlling multiple of these robots can be useful in applications such as disaster response, surveillance and exploration. This form of control is often achieved by using geometric patterns, such as triangles, squares, etc. Drones can employ this concept of formation control to fly and maneuver through environments and obstacles.

In this work a distributed affine formation control algorithm is implemented onto Crazyflie drones from Bitcraze. Ultra-wideband is used for positioning and communication between the drones. The implementation of the affine formation control algorithm is optimised such that it is only executed when new information is available, to prevent the onboard microcontroller from bottlenecking. This resulted in the drones flying in formation successfully with an accuracy of approximately 6.80 cm from its expected position.

Additionally, this algorithm is extended to manage cases where unexpected missing drones could comprise the stability of the formation. The implementation uses the CMSIS library that is optimised for matrix operations. This resulted in the drones flying in formation successfully even in the case of an observation loss with an accuracy of approximately 17.69 cm.

This work not only provides empirical data of experiments with an affine formation control algorithm, but also provides a baseline implementation for future research in the field of affine formation control, which can potentially lead to noise analysis or the application affine formation controls under different circumstances.

Distributed Affine Formation Control with Quadcopters

Thesis

submitted in partial fulfillment of the
requirements for the degree of

Master of Science

in

Computer and Embedded System Engineering

by

Shaan Hossain
born in The Hague, The Netherlands

This work was performed in:

Signal Processing Systems Group
Department of Microelectronics
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Delft University of Technology

Copyright © 2025 Signal Processing Systems Group
All rights reserved.

Delft University of Technology
Department of
Microelectronics

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **"Distributed Affine Formation Control with Quadcopters"** by **Shaan Hossain** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: 23 January 2025

Chairman: _____

dr.ir. R.T. Rajan

Advisor: _____

ir. Z. Li

Committee Members: _____

dr.ir. C.J.M. Verhoeven

Abstract

The usage of robots replacing human tasks has become more prevalent. Controlling multiple of these robots can be useful in applications such as disaster response, surveillance and exploration. This form of control is often achieved by using geometric patterns, such as triangles, squares, etc. Drones can employ this concept of formation control to fly and maneuver through environments and obstacles.

In this work a distributed affine formation control algorithm is implemented onto Crazyflie drones from Bitcraze. Ultra-wideband is used for positioning and communication between the drones. The implementation of the affine formation control algorithm is optimised such that it is only executed when new information is available, to prevent the onboard microcontroller from bottlenecking. This resulted in the drones flying in formation successfully with an accuracy of approximately 6.80 cm from its expected position.

Additionally, this algorithm is extended to manage cases where unexpected missing drones could comprise the stability of the formation. The implementation uses the CMSIS library that is optimised for matrix operations. This resulted in the drones flying in formation successfully even in the case of an observation loss with an accuracy of approximately 17.69 cm.

This work not only provides empirical data of experiments with an affine formation control algorithm, but also provides a baseline implementation for future research in the field of affine formation control, which can potentially lead to noise analysis or the application affine formation controls under different circumstances.

Acknowledgments

First and foremost I would like to thank my parents who granted me this opportunity to attend the top technical university in The Netherlands. What was once my childhood dream, has been made reality because of them. My mother in particular whom I have lived with my whole life and has always been supportive, patient and caring throughout all our ups and downs. My dad, who once dreamt of attending this very university but was not able to due to financial limitations now being able to see his son fulfil his own dreams. Now putting an end to my academic career, and starting my industrial career, I can finally give back the support and care they have given to me all my life.

Secondly, I would like to thank my supervisor dr.ir. Raj Rajan and my daily supervisor ir. Zhonggang Li. Despite his very busy schedules, Raj was able to give very helpful guidance and advice whilst always staying cheerful and motivational. Zhonggang always managed to listen to my questions and frustrations on-demand and has been an incredible help. Honestly, I could not have wished for better mentors.

A special thanks goes to the Swarming Lab at the Science Centre of TU Delft. In particular, the lab managers Jelmer, Lukas and Ruben who provided me the hardware, technical support and the environment to fly the drones during my thesis research.

Forever grateful to all my friends who have been a part of my life. Imad and Enel who have been as such for over 10 years, and with whom I could not have imagined what my personal development would have looked like without them. Micha for interchanging struggles and sharing laughters. Cheyenne who has shown me a different side of life and a fresh perspective on problems I had never even considered. A particular group of friends that go by the names of Soufyan, Aboubakr, Ibrahim, Kevin, Mohamed, Rene, Tarik, Diyar and too many more names that are not mentioned. Sharing great memories ranging from going on vacations to playing online games together, ultimately always uplifting the mood. And of course friends I have made during my MSc degree, Dion, Lucy, Karolis and Zhanhe just to name a few. Often visiting me in my lab whenever I needed a much needed break and organising social activities to escape the reality of our MSc degree.

My ex-manager and colleagues at Deloitte and Chipsoft whom I still (try to) keep in touch with. As well as many international people I met at the university for too short and wish to see when I visit their home country.

Last but definitely not least, my best friend who passed away too soon. He who I share the best and worst memories with during our academic career. He whom I promised I would walk out of here with a master's degree. This one is for you Valencio.

I did it

Shaan Hossain
Delft, The Netherlands
December 2024

Contents

List of Figures

List of Tables

Nomenclature

a	Scalar
$\mathbf{a}$	Column vector
$\mathbf{A}$	Matrix
$\mathcal{A}$	Set
A_{ij}	Entries of the i-th row and j-th column of matrix A
$\mathbb{R}^N$	Set of real vectors of length N
$\mathbb{R}^{N \times M}$	Set of real matrices of size N by M
$\lvert\cdot\rvert$	Cardinality of a set

1 Introduction

Over the past decade the usage of drones has been increasingly popular [1, 2, 3, 4, 5]. With applications varying from the logistics sector [6] to agriculture [7] and disaster response [8]. Types of drones have also evolved varying in size, weight, types of propulsion and level of automation [9]. Each variation of a drone can be applicable for different use cases. The software for drones has also undergone significant advancements over time. This evolution is evident from more robust controllers [10], to communicating over different media [11]. The enhancement in connectivity has enabled simultaneous real-time control and command of multiple drones, thereby facilitating the potential for swarm operations in a systematic fashion [12]. Swarm operations are not limited to drones and can apply to any kind of agent. This concept of multiple agents working independently to achieve a common goal is considered a distributed autonomous system.

1.1 Distributed autonomous systems

With robots and AI becoming more prevalent in replacing human tasks a primary way of achieving this is if these robots, or agents, work together. With distributed autonomous systems each agent behaves independently from other agents and can adapt to changing environments without human interventions [13] e.g., self-driving cars are systems equipped with sensors which are used to navigate roads, detect obstacles and make decisions without human intervention.

This form of automation can be applied in a variety of industrial and social sectors. Replacing human tasks is potentially more time-efficient, energy-efficient and safer. e.g., Using milking machines in the livestock to autonomously milk cows [14]. Or, Using automated guided vehicles are used to transport up to 100,000 kg in a warehouses which reduces the cost of labour, energy and increases safety of the human workers [15].

The distributed property of autonomous systems brings significant pros for handling tasks. Firstly, it is less common to fail on a single node of the network. In the case of a failing agent the system can still continue to operate [16]. Secondly, the system can be easily scalable by adding more agents without the system needing a complete overhaul [13]. For example, a swarm of flying drones can still operate even if one single drone fails. Additionally, the swarm can also easily be expanded by adding more drone to the swarm. A group of autonomous systems is more conveniently controlled in geometric patterns than as individual operating agents [17]. For example, a swarm of drones flying over crop fields in one straight line such that seeds or fertilizers are distributed evenly [18]. This naturally introduces the problem of formation control.

1.2 Formation Control

Coordinating and controlling a swarm of agents effectively requires a system or algorithm. For complex tasks, agents are more effective when working together. This can be used for example in disaster response [19], exploration [20] and inspection of buildings and vehicles [21, 22]. To achieve them working coherent and systematically they operate in a certain formation. Conventional formation control consist of certain key properties. First, controlling a swarm can be done for example by a global coordinator. This coordinator collects data from all agents, makes decisions based on this information, and then sends coordination commands to the agents [23]. For example, a laptop or PC, operated by a human or running an algorithm, sends instructions to a swarm of drones on where to fly.

Additionally, formation control usually consists of two subtasks. Firstly, formation shape control, which steers the agents to form a desired geometric pattern from any initial position. Secondly, formation maneuver control, which steers agents as a whole such that the centroid, orientation, scale, etc. can be changed. Different types of formation control range from position-based to displacement-based and leader-follower-based formation control [23, 24]. On average, depending on the type, formation control with drones is accurate approximately between 5 to 30 cm [25, 26]. This means that there is an average variance of 5 to 30 cm between the actual and expected position of the drones.

Some key limitations in formation control are scalability, robustness, and environmental adaptability [27]. In the case of increasing number of agents, the complexity of coordinating and controlling the agents grows as well [28]. Robustness challenges such as hardware failures, sensor noise, and environmental conditions cannot be completely prevented. Instead of trying to stop them from occurring, it is more effective to develop strategies for appropriately dealing with these scenarios [29].

Given these challenges, maintaining geometric consistency in formations is crucial. An affine formation is defined as a formation that is able to maintain its geometric consistency with affine transformations applied to it. For example, a formation that rotates but maintains the same relatives distances betweens the agents throughout the rotation. Affine formation control is particularly useful in use cases where a formation requires transformation such as avoiding obstacles [30] or avoiding collisions in changing environments [31].

A key characteristic of affine formation control is that the drones mainly depend on communicating with each other in order to stay in formation. In other words, the position of drones are required to stay observable for other drones. This implies that if one of the drones suddenly falls out of formation, the whole formation may collapse. Functioning and accurate localization is thus important to prevent this case of observation loss. The effectiveness and efficiency of affine formation control depend not only on the algorithms but also on the software and hardware used to implement them.

1.3 Bitcraze

For the purposes of this research, drones from Bitcraze are utilised, called Crazyflie [32]. The Crazyflie 2.1, shown in Figure 1.1, is small, modular and open-source, making it suitable for research applications.

1.3.1 Key components

The Crazyflie consist of many components such as, motors, wires and sensors. Some key components relevant for this research are the micro-controller, radio chip and the ultra-wideband (UWB) chip. A full list of properties and hardware specifications for the Crazyflie can be found in appendix A.

Micro-controller

The STM32F405 is developed and manufactured by STMicroelectronics [33]. This particular version of the micro-controller contains a Cortex-M4 core. It has a maximum clock frequency of 168MHz and contains 192kb of SRAM. Note that the SRAM operates faster than conventional DRAM but it requires power to retain its data [34]. Once no more power is supplied all data stored on the SRAM is lost. Additionally, the microcontroller contains 1Mb flash storage that is used to store the firmware of the Crazyflie.

Radio chip

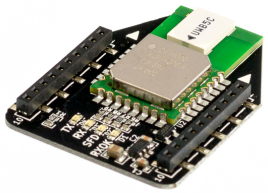
The NRF51822 is a System-on-Chip (SoC) developed and manufactured by Nordic Semiconductors [35]. This radio chip allows for connectivity over Bluetooth Low Energy (BLE) and 2.4GHz radio frequency. The SoC is also equipped with a Cortex-M0 core, running at a maximum clock frequency of 32MHz. Additionally, it has a 16Kb SRAM and a 128Kb of flash storage for the firmware that handles sending and receiving signals. It supports data rates up to 2Mbps and is mainly responsible for the communication between the drone and the external commander. For example a joystick, a laptop or a smartphone.

UWB chip

The DWM1000 is a UWB transceiver Integrated Chip (IC) developed and manufactured by Decawave (now acquired by Qorvo) [36]. It supports bands between 3.5GHz and 6.5GHz with data rates up to 6.8Mbps. According to the manufacturer, the application of position estimation with this transceiver is measured to be accurate up to 10 cm in standard deviation[1]. This means that the measured position is expected to be within 10 cm from its actual position.

[1]The 10 cm standard deviation is not further specified by the manufacturer. Therefore the usage of this 10 cm as a baseline when comparing the results of experiments throughout this report should be disregarded.

(a) Crazyflie 2.1 [32]



(b) UWB deck for the Crazyflie [37]

Figure 1.1: Crazyflie 2.1 drone developed by BitCraze

1.3.2 Bitcraze stock firmware

The firmware on the Crazyflie is open-source, developed and maintained by Bitcraze. This firmware is responsible for a variety of critical tasks, including Generating PWM signals for motor control, stabilizing the drone during flight through a PID controller, and sensor fusion, among other functions. All these tasks are managed by a real-time operating system, FreeRTOS, which handles task scheduling, interrupt handling and memory management [38]. FreeRTOS operates with a tick rate of 1000Hz, meaning it checks 1000 times per second for new tasks to execute. However, the actual execution speed of these tasks depends on the CPU clock frequency, which in this case, is a maximum of 168MHz.

One of the positioning systems present on the Crazyflie is the Loco Positioning System (LPS). The LPS uses an algorithm to estimate the position of the drone by receiving information over UWB. This algorithm proposed by [39] effectively combines the principles of Time Difference of Arrival (TDOA) and Two-Way Ranging (TWR) to enhance positioning accuracy. In this system, the drones receive packets from UWB devices which are fixed in place, called anchors. Simultaneously, the anchors also communicate with other anchors, exchanging packets and applying TWR to determine the time offsets between their clocks, which is then communicated with the drones. This approach eliminates the necessity for the synchronisation of clocks among the anchors before transmitting packets to the drone. The UWB chip accuracy of 10 cm must be taken into consideration when flying the drones and analysing results.

1.4 Goals and Overview

Combining all the aforementioned concepts, the aim of this research is to fly drones in formation in a decentralised fashion, where each drone can determine its own direction of flight that will engage formation. Current research has applied affine formation control for different types of applications in simulation [40, 41]. This research aims to explore the gap of applying affine formation control onto actual hardware by reporting results and bringing attention to challenges that may have been previously unaddressed. The objectives for this research can be summarised as the following

- How can an affine formation control algorithm, with observation losses, be implemented onto hardware?
- What accuracies can be achieved with the affine formation control?
- What challenges arise with affine formation control in the real-world?

For these objectives, this report is structured as the following. Chapter 2 introduces the notation and definitions regarding positioning which is used throughout this research. Chapter 3 provides a detailed explanation of the concept of affine formation control, along with the algorithm used. This is followed by the implementation of the algorithm onto hardware and an analysis of its performance based on various experiments. Chapter 4 addresses the scenario where a drone might suddenly lose connection and fall out of formation. It describes an algorithm capable of handling such a situation, followed by its implementation on hardware and an analysis of its performance based on various experiments. The final chapter evaluates the analyses of experiments and presents the conclusions. The future work of this thesis is also discussed in this chapter.

2 Position estimation

This chapter discusses how positions of agents can be determined and what is necessary to achieve this. Various techniques, algorithms and hardware used in positioning are explored and discussed. Finally, the chapter provides a detailed description of the positioning system used in all experiments throughout this research, along with an analysis of its performance.

2.1 Position

In multi-agent systems, precise positioning of agents is fundamental for effective coordination and task execution [42]. Accurate positioning can be important in various applications such as robotics, autonomous vehicles, and sensor networks. Here performance and safety of the system depend on the ability of agents to determine their locations relative to one another [43]. Positioning involves determining the exact location of each agent.Positions of agent i is defined as $\mathbf{z}_i \in \mathbb{R}^d$. Consequently, the displacement between agents i and j is defined by

$$\mathbf{z}_{ij} = \mathbf{z}_j - \mathbf{z}_i = [z_{j1} - z_{i1}, z_{j2} - z_{i2}, \ldots, z_{jd} - z_{id}]^T \tag{2.1}$$

The distance between these agents is defined as the Euclidean distance by the following equation

$$d_{ij} = \|\mathbf{z}_j - \mathbf{z}_i\| = \sqrt{\sum_{k=1}^{d} (z_{jk} - z_{ik})^2}. \tag{2.2}$$

2.2 Types of ranging

The devices that ensure connectivity create an infrastructure in which information can be exchanged between transmitters and receivers. This infrastructure can be utilised to estimate distances between the, so called, transmitter and receiver. By mounting such a device onto the agent and placing some devices in fixed positions (called anchors), different algorithms and techniques can be utilised to estimate the position of the agent. These ranging techniques are considered essential in multi-agent systems [13]. Different types of ranging can be categorised into One-Way-Ranging (OWR) and Two-Way-Ranging (TWR).

With OWR the communication between anchors and agents is unidirectional. Conventionally the anchor functions as the transmitter and the agent functions as the receiver. Some algorithms that operate one-way are Time of Flight (ToF), Time of Arrival (ToA), Received Signal Strength (RSS) and Time Difference Of Arrival (TDOA). For most OWR algorithms is required that the clocks on transmitters and receivers are

synchronised with each other. Since with unsynchronised clocks the receiver is unable to correctly determine how long a packet has travelled for from its transmitter [44].

TWR involves an exchange of signals between two devices. Because of this exchange there is no need for synchronised clocks. TWR can be further categorised in One-Sided TWR and Two-Sided TWR. One-Sided TWR requires only one round-trip exchange of information. Whereas Two-Sided TWR requires two round-trip exchanges. This provides higher accuracy but will consume more time and computation [45]. Some examples of TWR algorithms are Angle Of Arrival (AOA) and Round-Trip Time of Flight (RToF).

Using additional techniques and heuristics can further improve positioning accuracy. Kalman filtering and sensor fusion are often employed to enhance positioning accuracy [46]. Additionally, a high rate of information exchange between transmitters and receivers can also improve position accuracy. Transmitting packets at a higher rate means executing the ranging algorithm more frequently which results in more position estimations over time.

2.3 Connectivity

Communication and connectivity is key in estimating positions and in commanding agents. Communication is done over a particular media which can be different depending on the use case. For example, wifi is widely used for indoor positioning since it is widely available and offers relatively high accuracy. UWB provides precise, short-range positioning, which is ideal for applications where high accuracy is required. RFID offers high accuracy, scalability and no need of line-of-sight, often used in logistics and inventory management. Using Bluetooth strikes a balance between range and accuracy. Therefore, this is suitable for both indoor and outdoor positioning. Infrared is used for short-range, line-of-sight positioning, often in controlled environments. These technologies are just a limited selection of all that allow to determine positions of agents and facilitate coordination and navigation [47, 48]

For connectivity to function, devices must be capable of acting as both transmitters and receivers. Each medium of connectivity requires specific types of devices. For instance, UWB uses a UWB chip, wifi relies on specific wifi modules, and RFID operates using tags.

2.4 TDOA

TDOA is used in several real-world use cases such as, cellular networks and GPS systems. With TDOA, arrival times of packets from pairs or transmitters are compared with each other. Assume a scenario with anchors i and j where $i, j \in \mathcal{A}$, and drone $r \in \mathcal{V}$ in $d = 3$ dimensions. Anchor $\mathbf{z}_i = [x_i, y_i, z_i]$ and $\mathbf{z}_j = [x_j, y_j, z_j]$ are transmitting packets to drone $\mathbf{z}_r = [x_r, y_r, z_r]$. The drone calculates the difference in time of packets arriving from both anchors. This is TDOA generally defined as

$$\Delta t = \Delta t_{rx} - \Delta t_{tx}. \tag{2.3}$$

Here Δt_{rx} is the difference in arrival times between two packets. And Δt_{tx} is the difference in transmission times between two packets. With this, the following equation can be derived to estimate the position of drone r with anchor i and j

$$||\mathbf{z}_r - \mathbf{z}_i|| - ||\mathbf{z}_r - \mathbf{z}_j|| = c\Delta t, \tag{2.4}$$

Where c is the speed of propagation of the signal in its medium, which is generally simplified to the speed of light.

Since $[x_r, y_r, z_r]$ are unknown, plotting this equation results in a hyperbola between anchors i and j, also called a TDOA curve. When this process is repeated for different pairs of anchors, the intersection of the multiple curves results into the estimated position of the drone. Figure 2.1 illustrates the TDOA curves between the pairs of anchors and the estimated position at the intersection of these curves. In order to achieve an accurate position estimation the following must hold $|\mathcal{A}| \geq d + 1$. The number of anchors determines how many TDOA curves can be plotted. In particular, $|\mathcal{A}|$ anchors results in $\frac{|\mathcal{A}|(|\mathcal{A}|-1)}{2}$ TDOA curves. In the case of less then $d + 1$ anchors, there are not enough curves to form an intersection in $\mathbb{R}^d$ [49, 50].

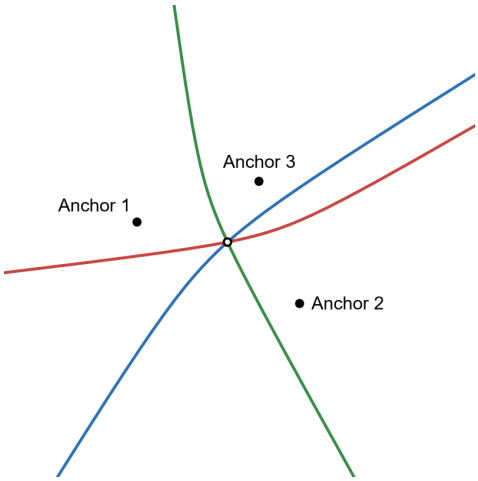


Figure 2.1: TDOA curves between 3 pairs of anchors

2.5 Loco Positioning System

The Loco Positioning System (LPS) is the positioning system integrated into the firmware of the Crazyflie. The LPS uses UWB decks and UWB anchors for positioning, seen in Figures 2.2a and 2.2b respectively. The algorithm used in LPS proposed by [39] effectively combines the principles of TDOA and TWR to mitigate clock drift. Figure 2.3a illustrates how these two principles are combined in an example scenario with 2 anchors and 1 receiving drone. Anchors i and j act as the transmitters while drone r acts as the receiver. Simultaneously, packets are also exchanged between the two anchors. Lets consider the last 3 packets (P1, P2, P3) received by the drone shown in Figure 2.3a. For TDOA, obtaining the Δt_{rx} is trivial since the drone can use its own clock to determine the arrival time of the packets. The transmission time of the

(a) Crazyflie 2.1 with an UWB deck

(b) UWB Anchor

Figure 2.2: Hardware required for the LPS

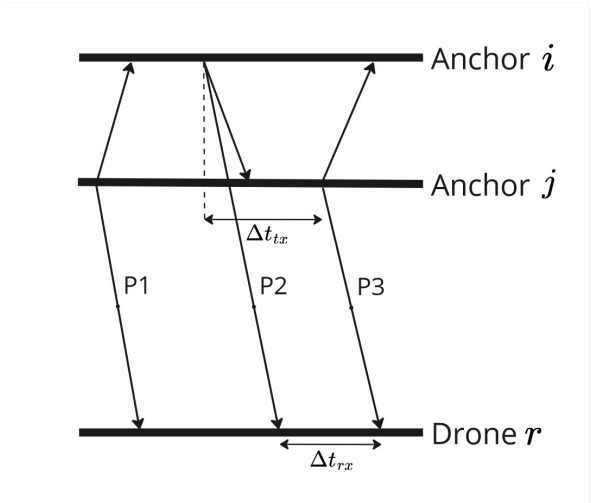
packets can also be determined since it is included in the packet data. However, since the clocks of the anchors might not be synchronised, the transmission time has to be corrected for this. Figure 2.3b illustrates TWR between the anchors and how the clock difference Δt_{ij} can be determined. With this information, anchor j can determine at what time packet P2 was transmitted from anchor i according to the clock of anchor i. This information is used to determine at which time packet P2 was sent according to the clock of anchor j. This results in drone r being able to determine at what times all the packets were transmitted. Now with the Δt_{rx} and Δt_{tx} being known, Δt can be determined. However, this equation does not account for clock drift that might have occurred. Therefore, the Δt_{tx1} and Δt_{rx1} are used to determine clock drift γ_1, as seen in Figure 2.3c. Here γ_1 is defined as $\frac{\Delta t_{rx1}}{\Delta t_{tx1}}$ and the final TDOA equation that accounts for clock drift results in the following

$$\Delta t = \Delta t_{rx} - (\gamma_1 \Delta t_{tx}). \tag{2.5}$$

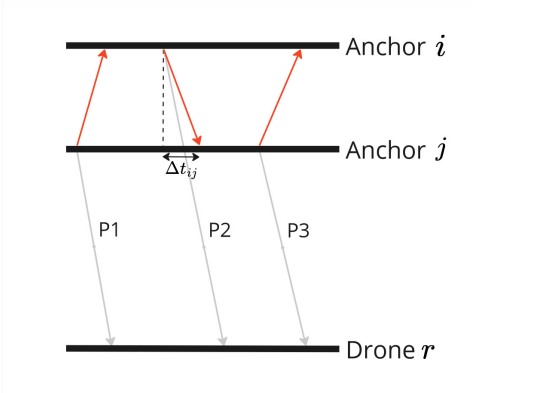
This Δt can be used in (2.4) to determine the TDOA curve. When this process is repeated for different pairs of anchors, the intersection of the multiple curves result into the estimated position of the drone.

The LPS relies on transmitting packets containing data. Figure 2.4 illustrates how a LPS packets is constructed. The key components that the LPS consist of are: a source address, which is the ID of the drone transmitting the packet, and a payload containing a unique packet ID and a timestamp of when the packet is transmitted.

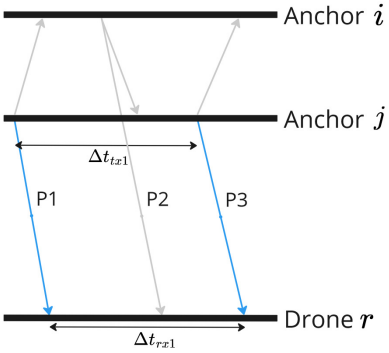
Figure 2.4: Layout of the LPS packet



(a) Transmissions between 2 anchors and 1 drone.



(b) TWR among the two anchors shown in red.



(c) Determining clock drift of anchor j shown in blue.

Figure 2.3: Example scenario of the LPS with last 3 packets received by drone r

2.6 Experiments with LPS

The accuracy of the LPS was tested under certain circumstances and its performance was analysed. Recall from Section 1.3.1 that the UWB chip used on the drone and anchors is measured to have a position accuracy of 10 cm. The experiments were executed with hybrid mode enabled, which will be further described in the next chapter. No other computation or algorithm is executed during the experiment on the drones, other that the positioning estimation algorithm. This experiment only considers 2 dimensions instead of 3 dimensions. This simplification will act as a baseline for the more complex case of 3 dimensions. The accuracy of drones were tested by placing them on fixed positions on the ground. The drones would remain stationary for 50 seconds while their position was polled by a nearby laptop. The expected positions, also defined as nominal positions, are defined as $\mathbf{p}_i \in \mathbb{R}^2$ and the actual positions are defined as $\mathbf{z}_i \in \mathbb{R}^2$. The metric used to measure the accuracy between the nominal and

actual position is defined as the Rooted Squared Error (RSE) by the following equation

$$||\mathbf{p}_i - \mathbf{z}_i|| = \sqrt{\sum_{k=1}^{d}(p_{ik} - z_{ik})^2}. \tag{2.6}$$

This is also commonly referred as the euclidean distance. The RSE is derived from Rooted Mean Square Error (RMSE) which is used to determine the mean RSE over the duration of an experiment and is defined as

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}||\mathbf{p}_i - \mathbf{z}_i||}. \tag{2.7}$$

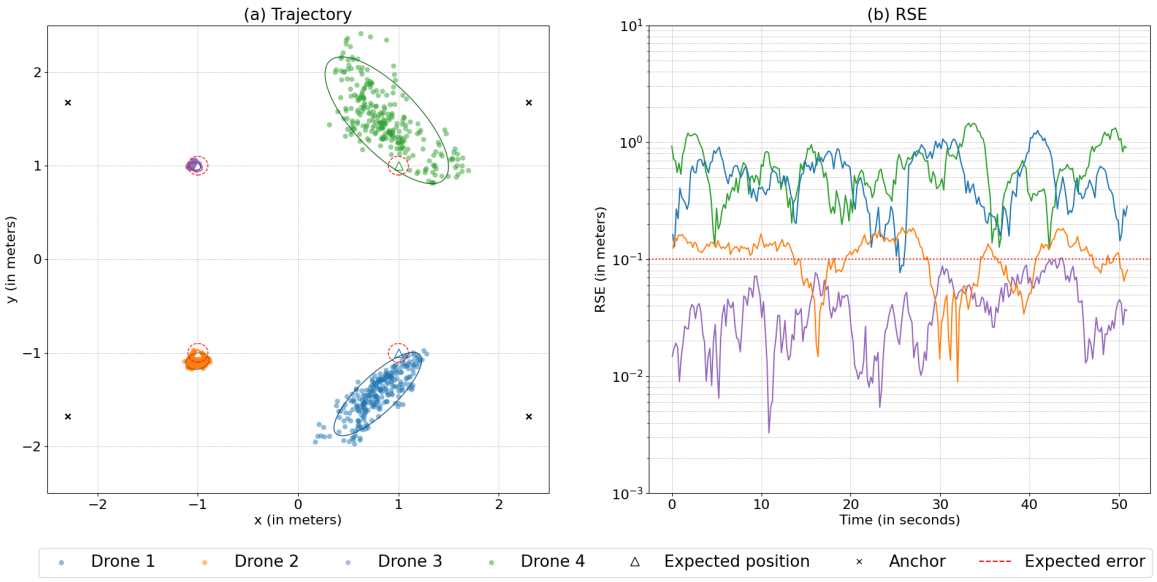


Figure 2.5: Position estimation scatterplot and RSE of 4 stationary drones using Hybrid mode

Each drone in the trajectory plot features a confidence ellipse around its respective position estimations. The ellipse is determined based on a 95% confidence interval assuming the position estimations is distributed normally. This results in far outliers not being taken into account for the confidence ellipse. Ultimately, the ellipse illustrates the density and position of the estimations. The red-dotted circle represents the expected 10 cm error radius around each nominal position. Therefore, the confidence ellipse can be visually compared to this error circle to illustrate the accuracy of the positioning. In the Figure 2.5, the ellipses representing drones 1 and 4 are significantly larger than the expected error circle, indicating higher uncertainty in their position estimations. Conversely, drone 3 has a small ellipse which falls within the error circle, demonstrating accurate positioning. Meanwhile, drone 2's ellipse is small but lies partly outside the error circle, suggesting a slight offset and minor inaccuracy in its positioning.

Table 2.1: RMSE of LPS position accuracy

Drone ID	Hybrid mode (m)
Drone 1	0.5185
Drone 2	0.1084
Drone 3	0.0402
Drone 4	0.6136

The observation from ellipses can be confirmed with data from the RSE plot. Table 2.1 shows the RMSE from this plot for each drone. Here it shows that drones 1 and 4 show an RMSE of 0.5185 and 0.6136 meters respectively, which is significantly more than the expected 10 cm. Drones 2 and 3 show a RMSE of 0.1084 and 0.0402 meters respectively.

Potential reasons for this inaccuracy could be measurement noise and systematic errors. Measurement noise are mostly outliers caused by multipath and signal propagation from not being in line of sight [51]. Since the drones were all in line of sight with the anchors, only multipath is considered as potential measurement noise. The edge along the positive x-axis was parallel to a solid a wall. This makes it very likely that the UWB signal from the anchors reflected from the wall to the drones 1 and 4, and ultimately causing the outliers [52]. Since the other sides are not surrounded by a solid wall this explains why drones 2 and 3 are not experiencing multipath and thus are more accurate. Systematic errors, in this experiment, are mainly software-related issues. For example, the onboard micro-processor of the drone might not be able to process the incoming UWB packets in time since it might be busy performing other calculations or because there are more UWB packets incoming than it can process over time. A more detailed experiment regarding software execution times will be discussed in the next chapter. Additionally, since the environment of the experiments could not be adjusted as trivially, an attempt was made to improve the software that handled the UWB. This improved version of the firmware and results of its respective experiments are discussed in the next chapter.

2.7 Summary

In this chapter, the concept of positioning is defined and discussed. Different types of ranging methods, algorithms and devices are mentioned. An description of how TDOA estimates positions is provided, along with an explanation on how the LPS on the Crazyflie works. The main results can be summarised as

- The LPS is a TDOA algorithm used for positioning and incorporates TWR among anchors to correct for unsynchronised clocks.
- The observed accuracy of the LPS is approximately between 4 to 50 cm (with hybrid mode enabled) and is worse than the promised accuracy of 10 cm by the manufacturer of the UWB chip.

3 Affine Formation Control

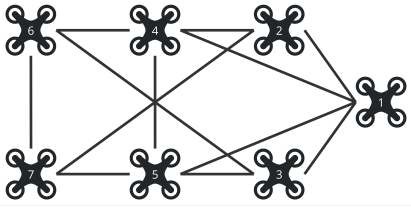
With positioning explained in the previous chapter it can be deployed on systems and algorithms that rely on positioning. In this chapter, the principle of affine formations are described along with an affine formation control algorithm. The implementation of this algorithm consist of two versions: Hybrid mode and P2P, with one being an improvement over the other. With the improved implementation, several experiments are done to analyse the improved accuracy of the positioning system and the performance of the affine formation control algorithm.

3.1 Graph Theory

In graph theory a formation can be defined by a undirected graph $\mathcal{G}$. This graph $\mathcal{G}$ is represented by a set of vertices $\mathcal{V}$ and a set of weighted edges $\mathcal{E}$. The set of vertices $\mathcal{V}$ represents a set of N drones where $\mathcal{V} = [1, ..., N]$. An edge is defined by a pair of vertices (i, j) where $i, j \in \mathcal{V}$. Since graph $\mathcal{G}$ is undirected, for each $(i, j) \in \mathcal{E}$, i can exchange information with j and vice-versa, the vertices i and j considered neighbours. Each edge in $\mathcal{E}$ carries weight l_{ij} and represents the stress between vertices i and j. The weight l_{ij} can be determined by either convex optimization or solving a feasibility problem of a linear matrix inequality [53]. All the weights are represented in a stress matrix $\mathbf{\Omega} \in \mathbb{R}^{\mathbf{N}\times\mathbf{N}}$ as follows,

$$\omega_{ij} = \begin{cases} 0, & \text{if } i \neq j \text{ and } (i,j) \notin \mathcal{E} \\ -l_{ij}, & \text{if } i \neq j \text{ and } (i,j) \in \mathcal{E} \\ \sum_{j \in N_i} l_{ij}, & \text{if } i = j \end{cases} \tag{3.1}$$

Figure 3.1 shows an example graph of a formation with 7 drones and its respective stress matrix.



(a) Graph structure

$$\begin{bmatrix} 0.2741 & -0.2741 & -0.2741 & 0.1370 & 0.1370 & 0.0000 & 0.0000 \\ -0.2741 & 0.6852 & 0.0000 & -0.5482 & 0.0000 & 0.0000 & 0.1370 \\ -0.2741 & 0.0000 & 0.6852 & 0.0000 & -0.5482 & 0.1370 & 0.0000 \\ 0.1370 & -0.5482 & 0.0000 & 0.7537 & -0.0685 & -0.2741 & 0.0000 \\ 0.1370 & 0.0000 & -0.5482 & -0.0685 & 0.7537 & 0.0000 & -0.2741 \\ 0.0000 & 0.0000 & 0.1370 & -0.2741 & 0.0000 & 0.2741 & -0.1370 \\ 0.0000 & 0.1370 & 0.0000 & 0.0000 & -0.2741 & -0.1370 & 0.2741 \end{bmatrix}$$

(b) Stress matrix

Figure 3.1: Example of a 7 drone formation

3.2 Affine Formation Control

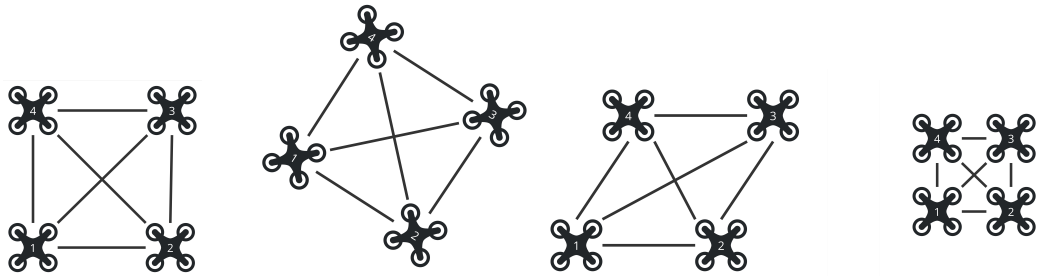
Affine formation control is a decentralised approach since communication with only neighbours is required. This mitigates the necessity of a global coordinator and reduces the impact of individual failures [54]. Additionally, affine formations allow its formation to transform whereas conventional formations are usually only able to translate [55]. Therefore, it can be particularly useful in use cases where agents have to operate in a dynamic and unpredictable environment [56].

A formation can defined as a combination of a geometric pattern of agents and an associated graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$. In formation control, the associated graph of a nominal formation is defined as $\mathcal{G}(\mathcal{V}, \mathbf{p})$, where $\mathbf{p}$ represents the positions of drones in a specified pattern in which the agents are expected to maintain. Figure 3.2a illustrates a formation with 4 drones starting at random positions. Eventually, these are expected to go into their respective nominal position, shown in Figure 3.2b. After reaching the nominal position, the drones can reach a target formation. This target formation is a transformation from its nominal formation which might be desired in changing environments. In order to achieve any target formation, a form of mapping is required. An affine formation is a collection of states where the agents can reach this target formation by applying affine transformations. This implies that collinearity and the ratios of distances are preserved [41]. Which means that agents on a line will still remain on a line after the transformation, and also the ratios of distances between the agents will be maintained. Some examples of affine transformations are: translation, scaling, rotation and shearing, shown in Figure 3.3. Furthermore, it can be achieved by manoeuvring only a subset of the agents. This subset of agents are considered leaders whereas the remaining agents of the formation are followers. The followers will follow the leaders while manoeuvring and do not need any knowledge about its target formation [57].

(a) Initial positions (b) Nominal formation

Figure 3.2: Example of a 4 drone formation

For affine formation control, universal rigidity is necessary to stabilize and converge into a nominal formation as well as a target formation [41]. In the case that the structure of a nominal formation is not universally rigid, an affine transformation applied to it will not guarantee a stable target formation. Consequently, if an agent falls out of formation, for example due to loss of connection, the formation may lose its universal rigidity, resulting in an unstable formation and thus fail to converge.



(a) Nominal formation (b) Rotated formation (c) Sheared formation (d) Scaled formation

Figure 3.3: Transformations of a 4 drone formation

3.3 Algorithm

The algorithm proposed by [53] allows agents to converge into a predefined formation without the need for a global coordinator. Assume the formation with $\mathcal{G}(\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of drones and the set of neighbours of drone i is defined as $\mathcal{N}_i$. The set of edges is defined as $\mathcal{E}$ with its weights defined in stress matrix $\mathbf{\Omega}$. Recall that the position of drone i is defined as $\mathbf{z}_i \in \mathbb{R}^2$, and that this is 2-dimensional which will serve as a baseline before considering more complex 3 dimensions. The algorithm is modelled as the following continuous system

$$\dot{\mathbf{z}}_i = \sum_{j \in \mathcal{N}_i} \omega_{ij}(\mathbf{z}_i - \mathbf{z}_j). \tag{3.2}$$

The algorithm is executed on each drone i individually. The difference in position between agent i and its neighbour j is multiplied with its corresponding weight ω_{ij}. This is done for all neighbours $j \in \mathcal{N}_i$ and summed together. The result $\dot{\mathbf{z}}_i$ is the velocity that must be applied to agent i to reach its nominal position. For the algorithm to work as expected, certain properties and prerequisites must be met. Firstly, The algorithm relies on the leader-follower strategy. This means that the target formation is dictated by the leaders and that only the followers will employ the algorithm. For leader selection, the selected leaders must span $\mathbb{R}^d$, where d is the dimension of the formation [53]. This thus implies that there must be at least $d + 1$ leaders. Secondly, each drone must be able to determine the position of its neighbours $\mathbf{z}_j \in \mathcal{N}_i$. This implies that each drone must be able to communicate with each other in order to exchange information about their position. Lastly, the structure of the nominal formation must be universally rigid as mentioned in Section 3.2.

The digital control on the drone acts as a discrete time system and executes everything periodically. Thus, to match the properties of a continuous system which the algorithm was originally designed for, the algorithm must be executed as often as possible. Since the algorithm is designed for synchronous execution, it assumes that all information is readily available simultaneously. However, in real-world applications synchronous execution is unrealistic. The main cause for this is that all information that is necessary for this algorithm may arrive at different times. If synchronous execution is desired it will be likely that the drone has to wait for all information to

arrive. Once the drone has received positions from all its neighbours it can execute the algorithm, which is then considered synchronous. Waiting for information is undesired and therefore the execution of the algorithm in this report is done asynchronously where the most recent information available is used. Conclusively, the asynchronous implementation of the algorithm will still allow convergence of drone into formation [58].

3.4 Implementation

The implementation of the algorithm is an extension of the original firmware that is described in Section 1.3.2. The original firmware did not provide sufficient functionalities for the algorithm to work. The ability of drones knowing the position of their neighbour was not present and had to be implemented. This required a systemic method of communication between the drones exchanging information on their current positions.

At first, several constants must be stored on the drone locally in arrays and matrices, which are defined in lines 2-6 of Algorithm 1. The weights for the stress matrix is done by either convex optimisation or by solving a feasibility problem of a linear matrix inequality as mentioned in Section 3.1 prior to the implementation. Similarly, the adjacency matrix, which defines the neighbouring drones, is defined prior to the implementation. Secondly, (3.2) needs to be implemented along with an additional variable for storing intermediate results. The input that is needed for the algorithm, such as current and neighbour positions must be requested and handled prior to algorithm execution. Lastly, the output from the algorithm, which is a velocity vector, must be send to the internal flight commander of the drone, which handles all movement of the drone. Additional functions that are called in this algorithm are handled by the firmware are described in Table 3.1.

Algorithm 1 Affine formation control algorithm

1: **Initialization**
2: Define number of drones $N = |\mathcal{V}|$
3: Define dimension of formation $d = 2$
4: Allocate and fill adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$
5: Allocate and fill position matrix $\mathbf{Z} \in \mathbb{R}^{N \times d}$
6: Allocate and fill stress matrix $\mathbf{\Omega} \in \mathbb{R}^{\mathbf{N} \times \mathbf{N}}$
7: **procedure** AFC(i)
8: Define $\mathbf{z}_i \leftarrow$ getCurrentPosition()
9: Define $\dot{\mathbf{z}}_i \leftarrow \mathbf{0} \in \mathbb{R}^d$
10: **for** $j \in \mathcal{N}_i$ **do**
11: Compute $\dot{\mathbf{z}}_i = \dot{\mathbf{z}}_i + \omega_{ij}(\mathbf{z}_i - \mathbf{z}_j)$ ▷ Control law (3.2)
12: **end for**
13: Execute setVelocity($\dot{\mathbf{z}}_i$)
14: **end procedure**

Table 3.1: Functions used in Algorithm 1

Function	Parameters	Description
GetCurrentPosition	-	Return current position of drone
SetVelocity	$\dot{\mathbf{z}}_i$: velocity vector	Send velocity to flight commander

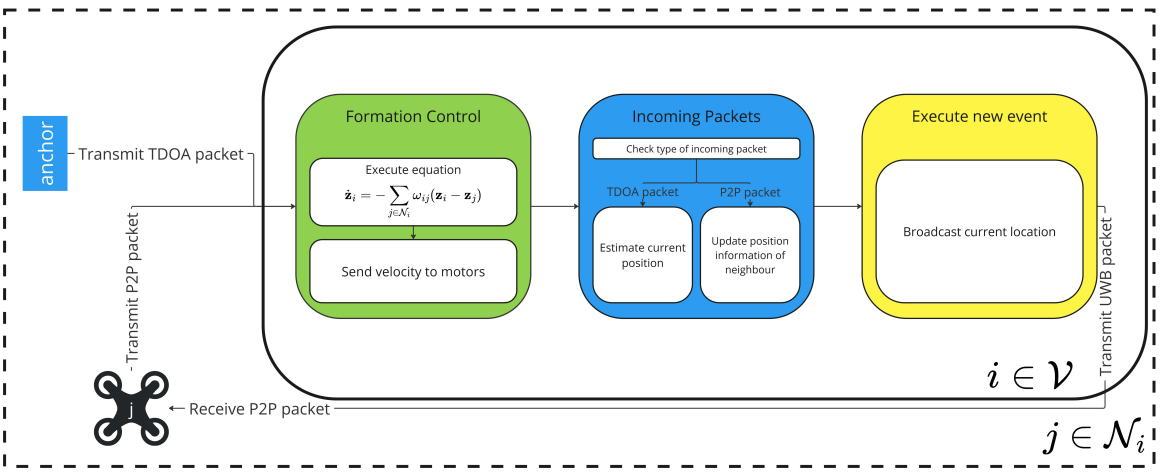


Figure 3.4: Order of execution for Hybrid Mode

3.4.1 Hybrid mode

An experimental feature in the firmware called *Hybrid Mode* has the integrated feature of communicating with other drones. It allows the drones to send packets over UWB to each other containing information about their current location. Position information that is received from neighbours with hybrid mode is stored locally on the drone. Hybrid mode also provides additional features such as sending packets for TWR and clock correction. However, these features do not provide any benefit to the purpose of this research.

Consequently, the implementation of Algorithm 1 must be triggered by an event in order to be executed. For every type of UWB activity, an interrupt is triggered. This interrupt calls the *UWBEventHandler* to process the UWB activity accordingly. Algorithm 2 shows when the execution of Algorithm 1 takes place, namely at the start of the *UWBEventHandler*. This means that, for every type of UWB activity, the algorithm is executed with the data that was present at that moment of time. After the algorithm is finished processing, the event-handler will start processing the UWB event. Additional functions that are called in this algorithm are handled by the firmware and are described in Table 3.1. Figure 3.4 gives an overview of the drone executing the algorithm and its communication with other drones and anchors.

Algorithm 2 Affine formation control with Hybrid Mode

```
1: procedure UWBEventHandler(eventType)
2:     Execute AFC(i)                                   ▷ Algorithm 1
3:     if eventType = packetReceived then               ▷ Check incoming packets
4:         Execute packetHandler(packet)
5:     end if
6:     Execute hybridMode()                             ▷ Send position to peers
7: end procedure
```

Table 3.2: Functions used in Algorithm 2

Function	Parameters	Description
AFC	i: Drone ID	Run Algorithm 1
PacketHandler	Packet: UWB packet	Process data of UWB packet
HybridMode	-	Transmit position to other drones

3.4.2 P2P mode

Implementation of hybrid mode showed some shortcomings as shown in Section 2.6. In order to address these, it was necessary to either rework or completely disable the hybrid mode. Given that the hybrid mode provides additional features that serve no purpose to this research, a more efficient solution was to develop the functionality of communicating with neighbours from the ground up and disable the hybrid mode.

3.4.2.1 P2P

Peer-to-Peer (P2P) communication is a concept where, in this research, drones send and receive information to and from other drones. The type transmission that is done to send information to other peers is done by broadcasting. Here the transmitting drone sends its UWB signal into its surrounding area without it being dedicated to a particular receiver. The custom P2P packet developed for this research is integrated within the existing LPS packet. This design choice ensured that the protocol used to handle UWB packets required only minor adjustments, rather than a complete redesign, to appropriately process the P2P packet. Figure 3.5 shows that the P2P packets contains a header which can be used to distinguish this packet as a P2P packet, and it contains coordinates of the position of the drone.

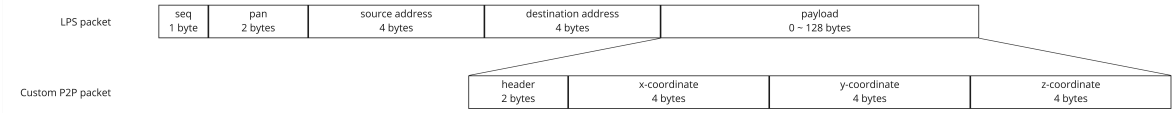


Figure 3.5: Layout of the P2P packet

This version of the implementation with consist of 2 key changes compared to its previous version. Firstly, the exchange of Hybrid Mode for the custom P2P mode. Secondly, the execution of Algorithm 1 moved from the start of the *UWBEventHandler* to after the *PacketHandler*, as can been seen in Algorithm 3. This means that the Algorithm 1 is executed only after a new P2P packet has arrived and is handled. This ensures that the execution of this algorithm happens only when there is new position information available about one of its neighbours. Figure 3.6 gives an overview of the drone executing the algorithm and its communication with other drones and anchors.

Algorithm 3 Affine formation control with custom P2P packets

```
1: procedure UWBEventHandler(eventType)
2:     if eventType = packetReceived then                ▷ Check incoming packets
3:         Execute packetHandler(packet)
4:         if packet = P2PPacket then
5:             Execute AFC(i)                                          ▷ Algorithm 1
6:         end if
7:     end if
8:     Define z_i ← getCurrentPosition(i)
9:     Execute sendP2PPacket(z_i)                        ▷ Send custom P2P packets
10: end procedure
```

Table 3.3: Functions used in Algorithm 3

Function	Parameters	Description
PacketHandler	Packet: UWB packet	Process data of UWB packet
AFC	i: Drone ID	Run Algorithm 1
GetCurrentPosition	-	Return current position of drone
SendP2PPacket	$\mathbf{z}_i$: Position of drone i	Transmit position to other drones

3.5 Experimental setup

For analysing the performance of the algorithm a sequence of tests were carried out. This required a hardware and software to be set up. First, an enclosed environment in which the drone can fly safely, shown in Figure 3.7a. Secondly, each of the drones require an UWB deck for the positioning system and P2P communication. Additionally, the positioning system also requires UWB anchors which act as reference points. Two anchors are placed on each vertical pole of the cage 0.25 m and 2.0 m from the ground respectively. Figure 3.7b shows a schematic drawing of the cage, the anchors are marked in blue.

After implementation the code needs to be compiled, build and flashed onto the drones. Each drone will then also be assigned an unique ID. The drones are connected with a 300mAh battery, switched on and placed at random positions on the ground inside the cage. Commanding the drone is done by using a laptop executing a python

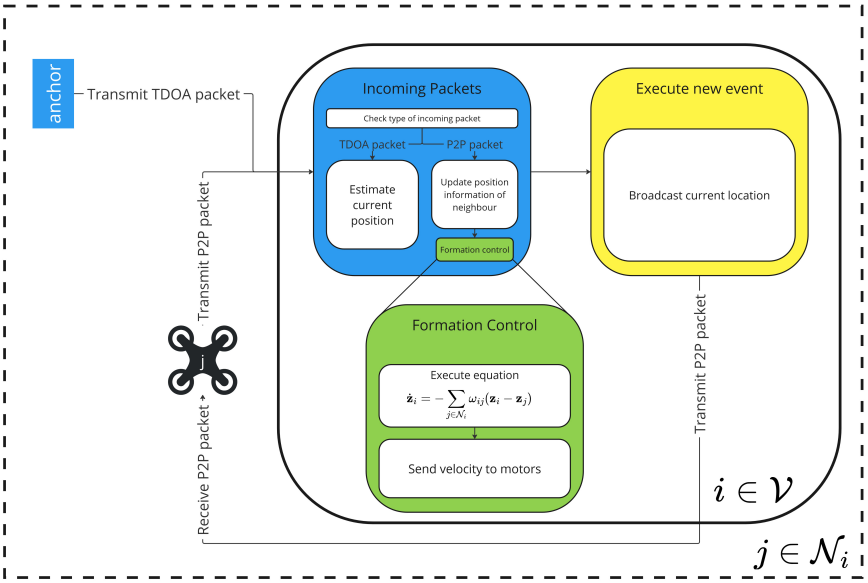
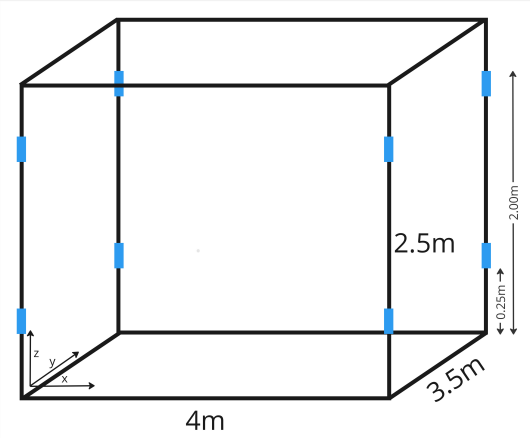


Figure 3.6: Order of execution with P2P

(a) Image of cage



(b) Schematic drawing of cage

Figure 3.7: Set up of cage

script. This script sends instructions and commands over 2.4GHz radio to the drones. Figure 3.8 illustrates the different types of communication that occur during an experiment. The anchors transmit TDOA packets to the drones for position estimation over UWB. The drones transmit and receive position data from each other over UWB. The laptop transmits instructions and commands to the drones over 2.4GHz radio.

In order to test the performance of the algorithm, the 4 drone formation seen in Figure 3.3a is used. The drone are placed on the ground inside the cage at random positions. Afterwards, all drones are commanded to take off and hover in the air. A command is then sent to all drones to engage formation, either by using the algorithm for the followers or by flying to their respective nominal positions for the leaders.

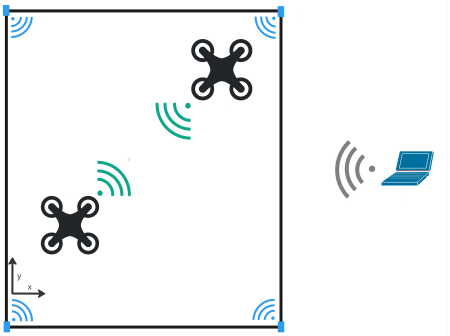


Figure 3.8: Schematic of the experimental setup with TDOA (blue), P2P (green), and radio (grey) signals

3.6 Results

3.6.1 LPS accuracy

The experiment conducted in Section 2.6 was done again but with the P2P implementation. However, here the drones were mirrored in position along the y-axis. Thus, compared to the original experiment drone 1 and drone 2 swapped positions, and drone 3 and drone 4 swapped positions. This was done to rule out the hypothesis that faulty hardware may had been the cause for the poor position accuracy during previous experiment. Recall that RSE, defined in (2.6), is the difference in current position and nominal position. A significant reduction in RSE is noticeable from results in Figure 3.9 compared to the previous experiment. The ellipses of the drones are all similar sizes to the circle of expected error. Table 3.4 shows the RMSE of the experiment with P2P. The table also contains the RMSE from the previous experiment using hybrid mode. When comparing these values, it is clear from that the P2P implementation offers a significant improvement in position accuracy. The drones that previously showed poor position accuracy, drones 2 and 3 in this experiment, showed enhancements of approximately 87% and 93% respectively. The remaining drones did not show an as significant improvement since these were already accurate within 10 cm. Drone 1 improved 24% in position accuracy whereas drone 4 actually decreased in accuracy by 1.24%. However, this is negligible, since 1.24% here corresponds to 0.0005 meters. In conclusion, the P2P implementation performs better than the hybrid mode in terms of position estimation accuracy. Therefore, it will be used throughout the remainder of this research.

Both hybrid mode and P2P were performed in a similar environment, implying that the measurement error is expected to be comparable. The improvement is primarily due to changes in the software, which address systematic errors. In the following section, results from an experiment can be found that tested various inner software components and identified improvements.

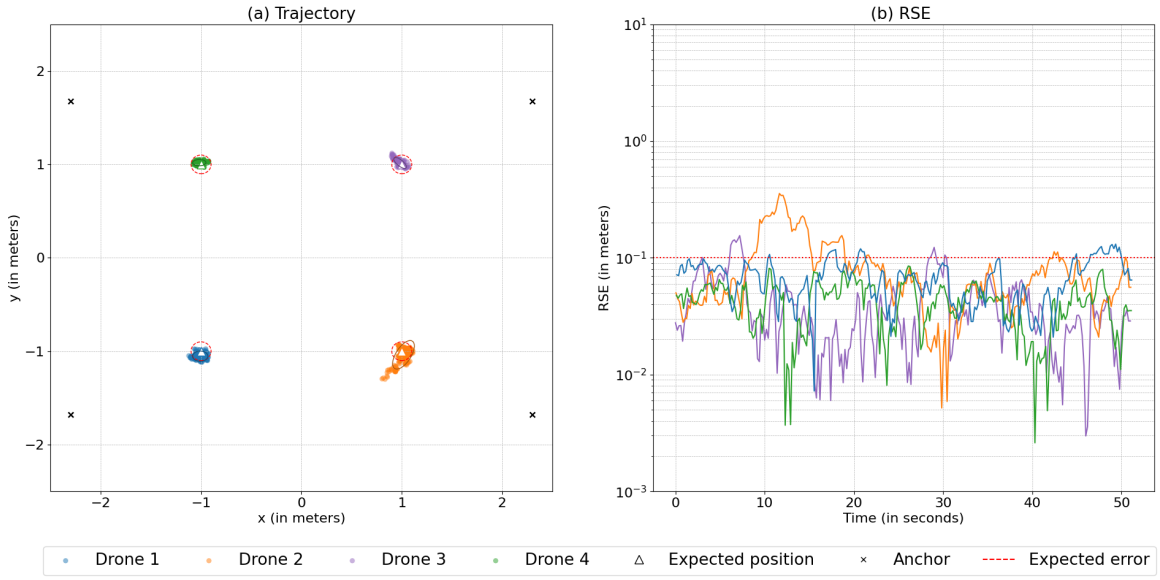


Figure 3.9: Position estimation scatterplot and RSE of 4 stationary drones using P2P communication

Table 3.4: RMSE of LPS position accuracy

Drone ID	Hybrid mode (m)	P2P (m)	Improvement (%)
Drone 1	0.1084	0.0830	23.43
Drone 2	0.5185	0.0673	87.02
Drone 3	0.6136	0.0423	93.10
Drone 4	0.0402	0.0407	-1.24

3.6.2 Control rate

Recall from Section 3.4.1 that Hybrid Mode does not only sends packets containing position, but also sends additional packets used for TWR and clock correction. These additional packets may bottleneck the UWB bandwidth and the operating system which results in less TDOA packets being received and handled and thus a less accurate position estimation. To test this hypothesis and compare it to the P2P implementation, experiments were conducted to track the execution rates of several function calls used in the software.

The number of drones influences the number of events that occur regarding UWB. Since, more drones relates to more UWB traffic this also implies there is more computation involved for each drone. Thus, more UWB traffic results in the affine formation control algorithm being called more frequent since more information is being received. Additionally, recall from the hybrid mode implementation in Algorithm 2, that the affine formation algorithm was executed whenever a UWB event occurred, regardless of the type of event. UWB events include receiving a UWB packet, transmitting a

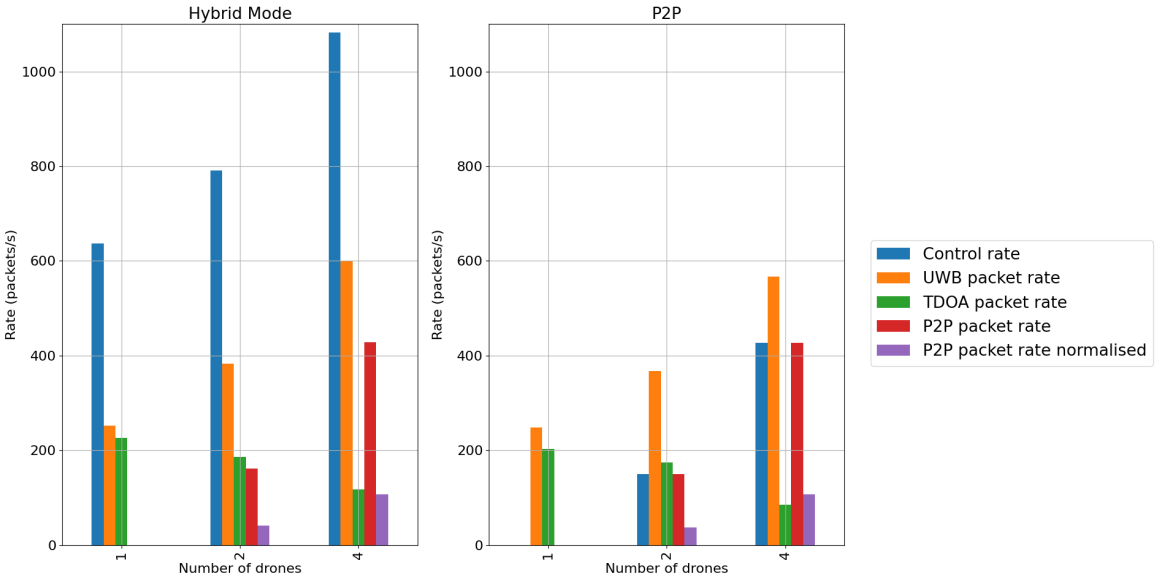


Figure 3.10: Rates of execution and packet

UWB packet, failing to transmit a UWB packet, and similar activities. This algorithm could potentially be executed without actually receiving any new relevant information regarding the neighbours of the drone. Since no new information is being received, the algorithm executions do not produce a different output and are therefore considered redundant. The P2P implementation optimised this such that the algorithm is only executed upon receiving new information about neighbour positions. This results in no redundant algorithm executions and a less congested CPU overall.

Figure 3.10 shows the result of an experiment in which the rate of receiving packets and the rate of algorithm execution are measured on stationary drones. The drones were placed inside the cage and commanded to run the formation control algorithm and the position estimation algorithm. Simultaneously, a laptop was monitoring the execution rates of these algorithms and tracking the frequency of UWB packet receptions. This experiment was conducted using a total of 1, 2, and 4 drones simultaneously, corresponding to having 0, 1, and 3 neighbours respectively. Both the Hybrid mode implementation and the P2P implementation were tested under these conditions separately. The UWB packet rate here is defined as all the UWB packets received by the drone per second. These incoming UWB packets can be split into P2P packets received from neighbours, and TDOA packets received from anchors used for positioning estimation. Since the P2P packets are received from possibly multiple neighbours, the P2P packet rate must be normalised. If P2P packet rate is defined as p and N is the number of drones in the experiment, then the P2P packet rate is normalised as follows

$$p' = \frac{p}{N} \tag{3.3}$$

Here p' represents how many P2P packets are sent per neighbour per second.

Since P2P communication only happens when more than 1 drone is present, the P2P packet rate and the normalised P2P packet rate are both 0 for the case where only

Table 3.5: Comparison in implementation between Hybrid mode and P2P

Version	Key characteristics	Pros (+) & Cons (−)
Hybrid mode	• Stock (experimental) feature • Formation control at the start of each UWB event	+ Ease of implementation − Overloading UWB − Redundant computation
P2P Mode	• Custom packet type for P2P • Custom packet handling • Formation control only after P2P packet	+ No redundant computation + Less UWB traffic

1 drone is present in the experiment. Additionally, recall from Section 3.4.2 that with the P2P implementation, algorithm execution happens immediately after a P2P packet is received. Therefore, in the case only 1 drone is present, the control rate is also 0. More precisely, for the P2P version, the control rate is always equal to the P2P packet rate. In the presence of 4 drones the control rate decreased from approximately 1100 executions per second in Hybrid mode to approximately 425 executions per second in the P2P version. This concludes that the P2P version indeed performs less redundant computation and thus is more efficient. The amount of UWB traffic remained similar between Hybrid mode and P2P decreasing from approximately 600 packets/s to 550 packets/s. In addition to this experiment, Appendix C contains a technical note accompanying datasets providing time-varying positioning data and execution time data of the Crazyflie under different conditions. A overview of key characteristics between the hybrid mode implementation and the P2P implementation can be found in Table 3.5.

3.6.3 Convergence

In this experiment drones 1 (blue), 2 (orange) and 4 (green) are leaders, and drone 3 (purple) is a follower. The anchors used for positioning are marked in as black crosses. Firstly, a simulation was done to estimate the trajectory and error of all drones. Figure 3.11 a shows how the drones reach their respective nominal positions. As expected, the RSE for all drones approaches zero, since the simulation does not factor in noise. A drone is considered to have converged in its nominal position once its RSE consistently remains below the expected error of 10 cm. This data is used to compare against the empirical data of this experiment which is shown in Figure 3.12.

Different stages of the experiment are shown in the following figures. Figure 3.13 shows the drones are on the ground ready for take-off at $t = 0$ seconds. This also shows where the anchors are located in the cage and which of the drones are leaders and follower. Figure 3.14 shows the scenario at $t = 2$ where the drones are in the air moments before applying affine formation control and converging. Here the white lines indicate the position in which the drones are expected to converge to. Figure 3.15 shows that the drones have converged into their expected position successfully at approximately $t = 12$.

Comparing the simulation and experiment shows that the leaders and follower con-

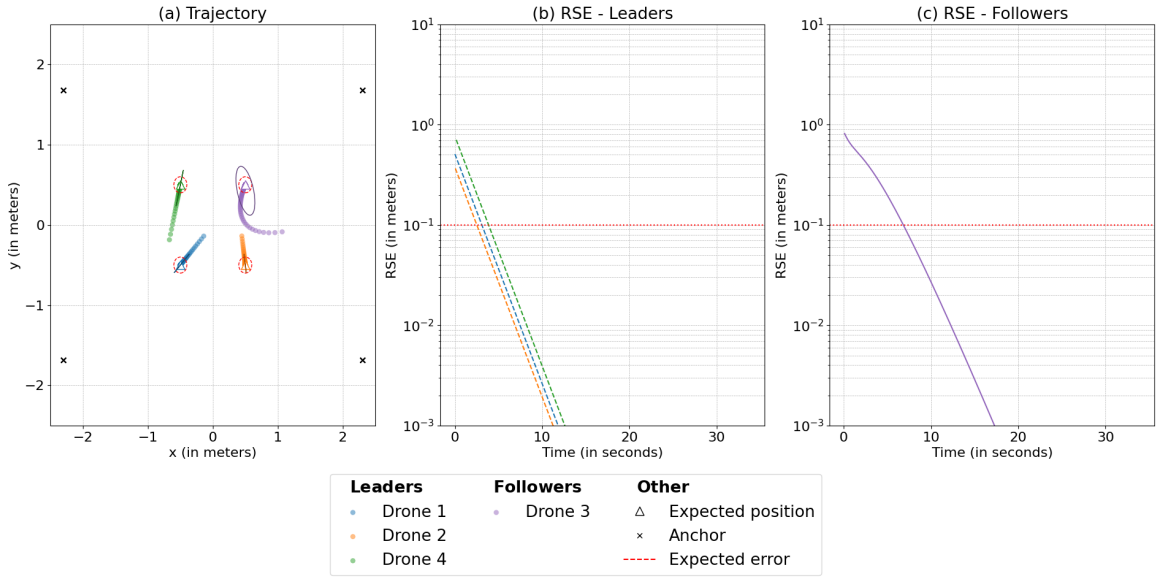


Figure 3.11: Simulation data on trajectories and RSE of 4 drones using the affine formation control algorithm

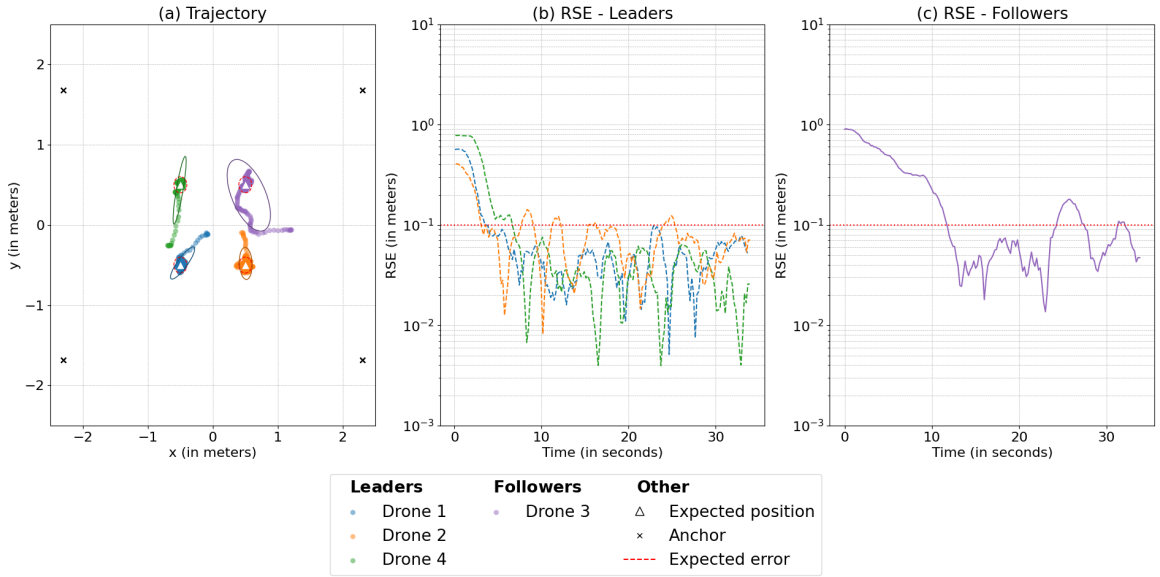


Figure 3.12: Empirical data on trajectories and RSE of 4 drones using the Affine Formation Control algorithm

verge consistently without any large outliers. However, the convergence happens much more slowly in the experiment compared to the simulation. The follower, drone 3, converged into nominal position at approximately $t = 12$ whereas in the simulation it did so at $t = 8$. The leaders also showed a slower convergence but since this difference is so small it is considered negligible. The main reason for a slower convergence is

Table 3.6: RMSE comparison between simulation and experiment

Drone ID	$0 \leq t \leq 50$		$t \geq 12$	
	Simulation (m)	Experiment (m)	Simulation (m)	Experiment (m)
Leaders				
Drone 1	0.0300	0.0827	0.000	0.0447
Drone 2	0.0215	0.0871	0.000	0.0632
Drone 4	0.0417	0.1116	0.000	0.0315
Average	0.0311	0.0938	0.000	0.0465
Followers				
Drone 3	0.0804	0.2387	0.000	0.0680

the presence of noise in the experiments. The expected positioning accuracy of 10 cm influences the speed of convergence. Despite not reaching an RMSE of 0 m, as in the simulation, the drones maintain an RMSE within the expected margin of 10 cm.

Table 3.6 presents the RMSE per drone for the entire duration of the experiment (50 seconds), as well as for the period after convergence, which occurs at $t = 12$. The RMSE is expected to be higher for the duration of the whole experiment compared to just after convergence. During $0 \leq t \leq 50$ the leaders and followers show a higher RMSE for the experiment than the simulation. However, due to the different initial positions of each respective drone, the RMSE varies between drones. Therefore, data where $t \geq 12$ is more relevant since it only considers the positions of the drones after convergence. The simulation showed the drones having a rounded RMSE of 0 m, which is unrealistic as it does not account for noise. The RMSE for all drones are below the expected 10 cm. This result can also be compared to Table 2.1 where the drones showed a similar RMSE. In conclusion the leaders show an average RMSE of 0.0465 m and the follower 0.0680 m, both of which fall within the acceptable margin of error of 10 cm.

3.7 Summary

This chapter delves into the affine formation control algorithm, including how it works and its implementation on drones. Experiments were also conducted to test its performance. The main points can be summarised as follows:

- The P2P implementation transmits custom packets to other neighbours and only executes the affine formation control if new information is received
- The P2P implementation prevents overloading the positioning system algorithm, thereby improving in position estimation by approximately 90% for drones that previously performed poorly with hybrid mode.
- The drones successfully converge into formation with an RMSE of 4.65 cm for leaders and 6.80 cm for the follower.

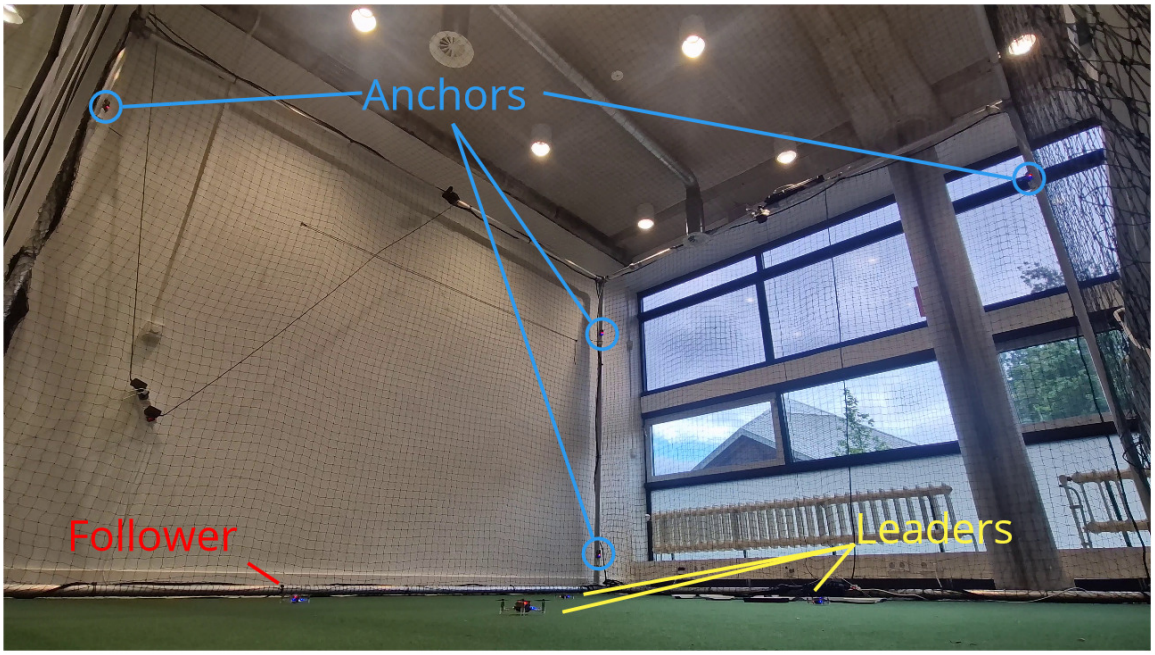


Figure 3.13: AFC Experiment with square formation at $t = 0$ before taking off

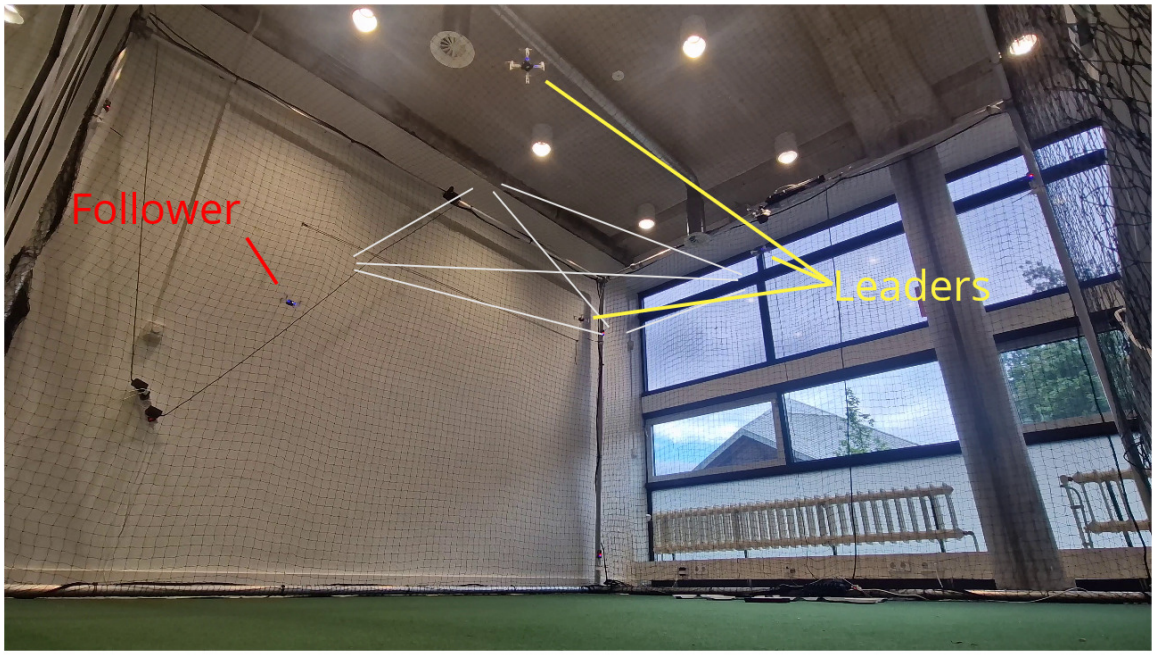


Figure 3.14: AFC Experiment with square formation at $t = 2$ after taking off

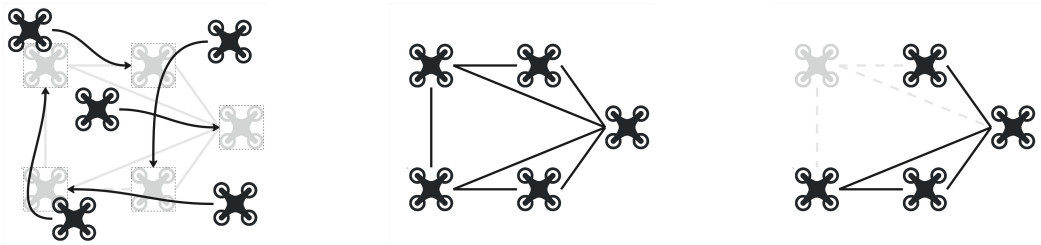
Figure 3.15: AFC Experiment with square formation at $t = 12$ in formation

4 Relative Affine Localisation

The previous chapter discussed formations in which all drones are present and provide information to their neighbours accordingly. As mentioned in Section 3.2, in the case of an suddenly absent neighbour the formation may not converge and thus fall apart. This chapter describes an addition to the formation control algorithm called Relative Affine Localisation (RAL) which allows to keep the formation stable even in the case of observation loss of a drone.

4.1 Algorithm

As mentioned in Section 3.2, in the case of a missing agent the formation most likely will not converge. However, in some cases the position of this missing agent can be inferred, as if the drone is still in its nominal position. The previous chapter only considered two stages, the initial formation and the nominal formation. This chapter introduces a third stage where a drone is missing and lost from observation, shown in Figure 4.1c. The missing drone that is lost from observation has no edges with its neighbours and thus is not able to communicate with them anymore.



(a) Initial position (b) Nominal formation (c) Missing drone

Figure 4.1: Three different stages of a 5 drone formation from initial positions to nominal formation to a missing drone

The RAL algorithm, proposed by [56] is an extension from the original algorithm defined in Section 3.3. This means that the original affine formation algorithm will execute as usual, but only in the case of a missing neighbour, the RAL will be executed to infer the position of this missing neighbour. Lets assume formation $\mathcal{G}(\mathcal{V}, \mathcal{E})$ where the set of neighbours of agent i, $\mathcal{N}_i$ can be divided into $\mathcal{N}_i = (\mathcal{N}_i^k, \mathcal{N}_i^m)$. Where $\mathcal{N}_i^k$ are known neighbours and $\mathcal{N}_i^m$ are missing neighbours of agent i.

The RAL has certain limitations for operating effectively. First, the limit on missing agents is defined by $|\mathcal{N}_i^k| \geq d$ for all $i \in \mathcal{V}$. Secondly, $\mathcal{N}_i^k$ must span $\mathbb{R}^d$ for all $i \in \mathcal{V}$ [57]. Figure 4.2 shows 3 different configurations of agents. In this example configuration (a) is not feasible since $|\mathcal{N}_i^k| < d$. Configuration (b) is also not feasible since $\mathcal{N}_i^k$ does not

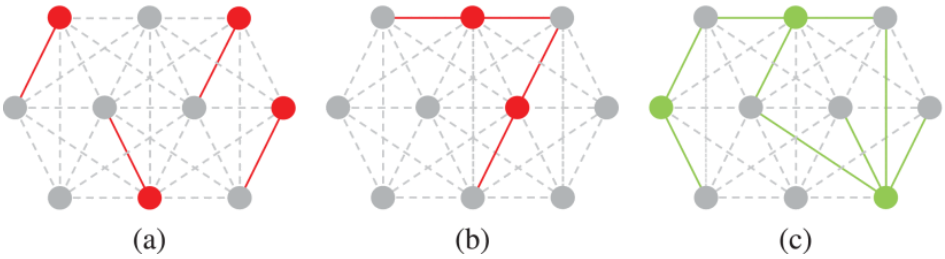


Figure 4.2: Example of feasible and not feasible configurations of agents in a formation of $\mathbb{R}^2$ [57]

span $\mathbb{R}^2$.

The relative position of the missing drone can be estimated by using only observations. These observations include the relative positions of drones that still maintain a connection with their neighbours. Assume a formation of dimension d at time t. With this we can construct the following matrices from these observations. $\mathbf{H}_i \in \mathbb{R}^{d\times|\mathcal{N}_i|}$ and $\mathbf{X}_i \in \mathbb{R}^{d\times|\mathcal{N}_i|}$. Recall that the nominal position of drone i is defined as $\mathbf{p}_i \in \mathbb{R}^d$ and the actual position is defined as $\mathbf{z}_i \in \mathbb{R}^d$. The construction for $\mathbf{H}_i$ is done by the following equation

$$\mathbf{p}_{ij} = \mathbf{p}_j - \mathbf{p}_i \text{ for all } j \in \mathcal{N}_i^k, \tag{4.1}$$

which is the displacement of nominal positions between drone i and all its known neighbours. $\mathbf{X}_i$ is constructed by the following equation

$$\mathbf{z}_{ij} = \mathbf{z}_j - \mathbf{z}_i \text{ for all } j \in \mathcal{N}_i^k, \tag{4.2}$$

which is the displacement of actual positions between drone i and all its known neighbours. With these definitions it is now possible to retrieve $\mathbf{z}_{ij}$ for $j \in \mathcal{N}_i^m$ with the following equation proposed by [56]

$$\mathbf{\Theta_i(t)^T} = \left(\mathbf{H_i(t)^T H_i(t)}\right)^{-1} \mathbf{H_i(t)^T X_i(t)}. \tag{4.3}$$

The matrix $\mathbf{\Theta_i(t)} \in \mathbb{R}^{\mathbf{d}\times\mathbf{d}}$ is here considered as the global affine transformation matrix and is used to infer the displacement of its missing neighbour with the following equation

$$\mathbf{z}_{ij}(t) = \mathbf{\Theta_i(t) p_{ij}} \text{ where } \mathbf{j} \in \mathcal{N}_\mathbf{i}^\mathbf{m}. \tag{4.4}$$

The final vector $\mathbf{z}_{ij}(t)$ is the inferred displacement of missing neighbour j.

4.2 Implementation

Since the RAL is an addition to the original formation control algorithm, no major changes are needed to the original implementation. However, there is some additional information required such as, nominal positions of the formation as well as an array that keeps track of missing drones.

The RAL algorithm introduces a new type of computation that was not present in the previous implementations, which is matrix calculations. Matrix multiplication and

inversion are computationally intensive operations on embedded hardware. Because of this, these operations must be executed as efficient as possible to prevent bottlenecking the microcontroller in time and space. All variables, arrays, and matrices used as input, output and intermediate results for the RAL are allocated at compile time, as can be seen at lines 2-13 in Algorithm 4. This static memory allocation method prevents issues related to running out of memory at runtime, which can occur with dynamic memory allocation.

4.2.0.1 DSP CMSIS

For the computation of (4.3) and (4.4), the CMSIS DSP software library is used to execute matrix transposes, multiplication and inversion [59]. This library is specifically optimised for the Cortex-M family of microcontrollers which is also used on the drone. There are 2 main characteristics of this library that allow faster computation. First, the processor addresses memory as a flat linear address space. Because of this, the library uses the First In First Out principle to store and shift new incoming data. This means that less instructions are needed for handling data movement. This also mitigates the overhead of checking the index every time data is accessed [60]. Secondly, the algorithms present in CMSIS used for arithmetic operations allow for loop unrolling [61]. This means that multiple iterations of a loop are executed at once. These benefits enhance the computation used for all matrix operations in RAL. Unlike using this library, manually implementing these matrix operations is error-prone and misses out on the advantages provided by the DSP CMSIS library. Lastly, a common method to improving computation time is by converting floating-point data into fixed-point data. Fixed-point calculations are generally faster than floating-point calculations, however comes at the cost of precision. The Cortex-M4 contains a Floating-Point Unit (FPU) which is a processing unit with special instructions optimised for floating-point calculations. Utilising the FPU can result in fast computation while keeping the benefit of precision from the floating-point numbers. Therefore, converting the floating-point numbers into fixed-point is in this case undesired.

Lines 15-21 in Algorithm 4 show the function for matrix transpose, multiplication and inversion. Note that for this implementation it is assumed that each drone is notified immediately when a drone is considered missing. Additional functions that are called in this algorithm are handled by the firmware are described in 4.1 and a full list of functions used in this research can be found in Appendix B. The source code for the RAL implementation can be found in Appendix D.2.

Table 4.1: Functions used in Algorithm 4

Function	Parameters	Description
Mat_Trans	**A**: Matrix **A**	Transpose matrix **A**
Mat_Mult	**A**, **B**: Matrix **A** and **B**	Multiply matrix **A** with matrix **B**
Mat_Inv	**A**: Matrix **A**	Invert matrix **A** or return -1 if invertible

Recall Algorithm 1 which is the the affine formation control algorithm without RAL. With the introduction of RAL, it must now be included in the original algorithm.

Algorithm 4 Relative affine localization algorithm

Initialization
  Define number of non-missing neighbours $m = |\mathcal{N}_i^k|$
  Allocate and fill displacement of nominal position $\mathbf{p}_{ij} \in \mathbb{R}^d$
  Allocate and fill $\mathbf{H}_i \in \mathbb{R}^{m \times d}$ ▷ Based on (4.1)
  Allocate and fill $\mathbf{X}_i \in \mathbb{R}^{m \times d}$ ▷ Based on (4.2)
  Allocate $\mathbf{H}_i^T \in \mathbb{R}^{m \times d}$, $\mathbf{H}_i^T \mathbf{H}_i \in \mathbb{R}^{d \times d}$, $\mathbf{H}_i^T \mathbf{X}_i \in \mathbb{R}^{d \times d}$, $(\mathbf{H}_i^T \mathbf{H}_i)^{-1} \in \mathbb{R}^{d \times d}$
  Allocate $\mathbf{\Theta_i} \in \mathbb{R}^{\mathbf{d} \times \mathbf{d}}$, $\mathbf{\Theta_i^T} \in \mathbb{R}^{\mathbf{d} \times \mathbf{d}}$, $\mathbf{\Theta_i^T p_{ij}} \in \mathbb{R}^{\mathbf{d}}$ and $\mathbf{z}_{ij} \in \mathbb{R}^d$
procedure RAL(i)
  Compute $\mathbf{H}_i^T$ by MAT_TRANS($\mathbf{H}_i$)
  Compute $\mathbf{H}_i^T \mathbf{H}_i$ by MAT_MULT($\mathbf{H}_i^T$, $\mathbf{H}_i$)
  Compute $\mathbf{H}_i^T \mathbf{X}_i$ by MAT_MULT($\mathbf{H}_i^T$, $\mathbf{X}_i$)
  Compute $(\mathbf{H}_i^T \mathbf{H}_i)^{-1}$ by MAT_INV($\mathbf{H}_i^T \mathbf{H}_i$)
  Compute $\mathbf{\Theta_i^T}$ by MAT_MULT($(\mathbf{H}_i^T \mathbf{H}_i)^{-1}$, $\mathbf{H}_i^T \mathbf{X}_i$) ▷ Based on (4.3)
  Compute $\mathbf{\Theta_i}$ by MAT_TRANS($\mathbf{\Theta_i^T}$)
  Compute $\mathbf{z}_{ij} = \mathbf{\Theta_i p_{ij}}$ by MAT_MULT($\mathbf{\Theta_i^T}$, $\mathbf{p}_{ij}$) ▷ Based on (4.4)
  return $\mathbf{z}_j = \mathbf{z}_{ij} - \mathbf{z}_i$ ▷ Position of missing drone j
end procedure

Algorithm 5 illustrates how the RAL is incorporated within the affine formation control algorithm. The main changes are present in lines 10-12, where the positions of all missing drones are inferred using RAL. After this, the algorithm continues as usual by executing (3.2).

Algorithm 5 Affine formation control algorithm with RAL

Initialization
  Define number of drones $N = |\mathcal{V}|$
  Define dimension of formation $d = 2$
  Allocate and fill adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$
  Allocate and fill position matrix $\mathbf{Z} \in \mathbb{R}^{N \times d}$
  Allocate and fill stress matrix $\mathbf{\Omega} \in \mathbb{R}^{\mathbf{N} \times \mathbf{N}}$
procedure AFC(i)
  Define $\mathbf{z}_i \leftarrow$ GETCURRENTPOSITION()
  Define $\dot{\mathbf{z}}_i \leftarrow \mathbf{0} \in \mathbb{R}^d$
  for $r \in \mathcal{N}_i^m$ **do**
    Define $\mathbf{z}_r \leftarrow$ Execute RAL(i) ▷ Algorithm 4
  end for
  for $j \in \mathcal{N}_i$ **do**
    Compute $\dot{\mathbf{z}}_j = \dot{\mathbf{z}}_i + \omega_{ij}(\mathbf{z}_i - \mathbf{z}_j)$ ▷ Control law (3.2)
  end for
  Execute SETVELOCITY($\dot{\mathbf{z}}_j$)
end procedure

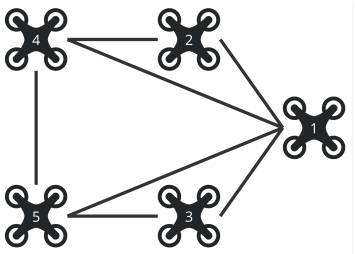


Figure 4.3: Formation with 5 drones

4.3 Experimental Setup

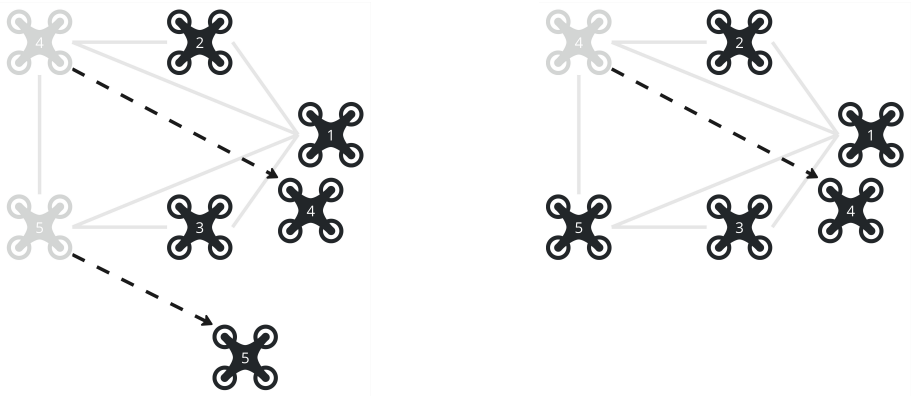
For testing the RAL, the original experimental setup from the previous chapter must be slightly modified. The formation consists of 5 drones as oppose to 4 drones which was used in the previous chapter. The drone formation shown in Figure 4.3 consists of 3 leaders and 2 followers. Drones 1, 2 and 3 will be leaders, and drones 4 and 5 will be followers. The need for 2 followers is because one must simulate being the missing drone and the other will use the RAL algorithm to infer the missing drone's position. Simulating a missing drone can be done by for example drifting away from its nominal position into certain direction the with constant velocity.

In this particular setup, drone 4 will drift off and be considered the missing drone, while drone 5 will employ the RAL algorithm to infer drone 4's position. From a trivial perspective, it may seem obvious to simply turn off drone 4 and then enable RAL on drone 5 to test and analyse the effectiveness of the RAL algorithm. However this will not properly test the RAL algorithm. Recall from Section 3.3 that the most recent information is used for the affine formation control algorithm. If drone 4 is turned off at its nominal position, drone 5 will assume drone 4 is still at its nominal position since that was the most recent information it received. Therefore, the position of drone 4 does not need to be inferred. A proper test requires drone 4 to actively fly away from its nominal position and communicate its position with drone 5. Figure 4.4a illustrates this experiment without RAL enabled, where drone 4 deviates from its position while neighbouring drone 5 attempts to follow, as expected with affine formation control algorithm. Figure 4.4b illustrates the same experiment but with the RAL enabled on drone 5, which infers the position of drone 4.

4.4 Result

4.4.1 Baseline Experiment

In the baseline experiment, the followers behaved as shown in Figure 4.4a. In Figure 4.5 drone 4 actively flies away from its nominal position, and drone 5 follows accordingly. Recall that the anchors used for positioning are marked as black crosses. At $t = 0$ all drones take off and will engage formation. Until $t = 37$, the drones stay in their nominal position. After $t = 37$, drone 4 is simulating a loss in connection by flying away from its nominal position and drone 5 follows this movement accordingly. The RSE for the leaders fluctuates around the expected 10 cm bound. Despite some outliers above 10



(a) Baseline outcome without RAL (b) Desired outcome with RAL

Figure 4.4: Schematic drawing of RAL experiment

cm for the leaders, the RMSE is still below 10 cm and therefore considered acceptable. Both followers showed a significantly higher RSE than its leaders as expected. However, this was not as linear as was expected. Where a trajectory shown in Figure 4.4a was expected, the experiment showed significant outliers. This can be confirmed by width of the confidence ellipses. A potential cause for this could be an increase in noise. Recall from the LPS experiment in Section 2.6 that the side along the positive x-axis is parallel to a wall and possibly causing issues regarding multipath. However, since in this experiment the noise is present on both sides of the x-axis, it can be ruled out that multipath is the cause of these outliers. Consequently, a probable cause are the presence of systematic errors. In particular, the software not being able to process the amount of the packets that are exchanged over UWB and thus resulting in poor position estimations. The absence of sufficient empirical data prevents concluding the exact impact of these systematic errors. Thus, it is left as future work to analyse the nature and impact of these systematic errors. Furthermore, research has shown that systematic errors have complex behaviour and in a similar setup have an accuracy that is typically larger than 15 cm [62].

4.4.2 RAL Experiment

Figure 4.6 shows that in the RAL experiment the drones behaved as expected in Figure 4.4b. At $t = 0$ the drones are stationary on the ground, seen in Figure 4.7. Here it is also shown were the anchors are positioned, and which drones are leaders and followers. At $t = 10$ all drones have taken off and are all flying in nominal position, shown in Figure 4.8. From $t = 37$ drone 4 is commanded to fly away from it nominal position. Simultaneously, drone 5 is notified of drone 4 losing connection and the RAL is enabled on drone 5. At $t = 45$ drone 4 is away from its nominal position and drone 5 successfully inferred the position of drone 4. Figure 4.9 illustrates which edges of the formation are lost because of the missing drone and how drone 5 has remained in its nominal position by inferring the position of drone 4.

The trajectory in Figure 4.5a shows that drone 4 flies in a much more controlled manner compared to baseline experiment. The confidence ellipse is not as large in the

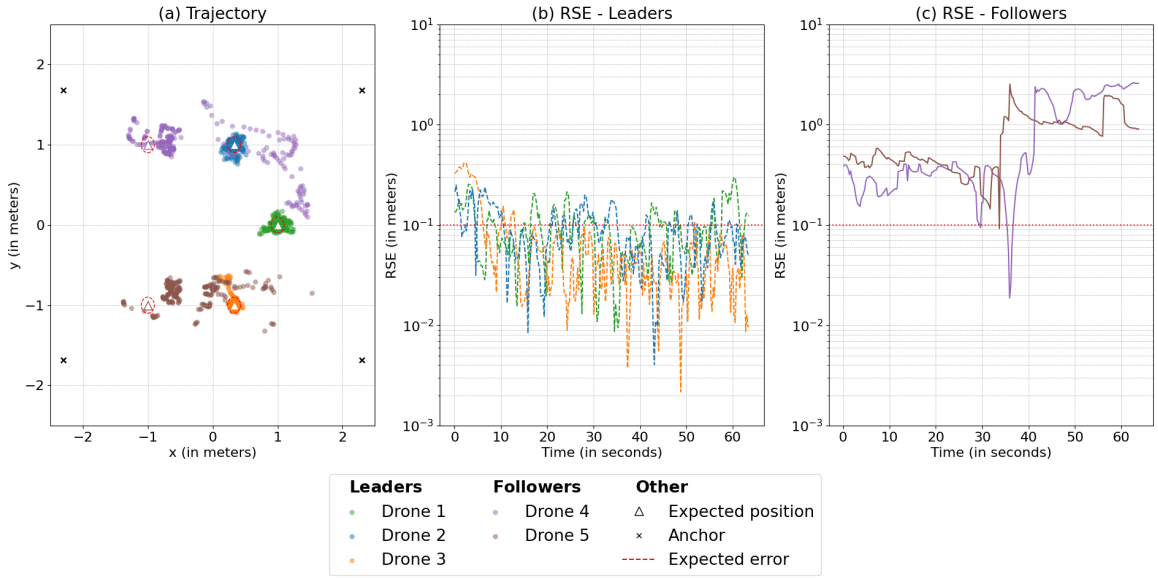


Figure 4.5: Empirical data on trajectories and RSE of 5 drones, without using RAL

Figure 4.6: Empirical data on trajectories and RSE of 5 drones, using RAL

experiment as in the baseline. As drone 4 flies away from its nominal position, drone 5 remained at its own nominal position since it inferred the nominal position of drone 4. This trajectory is similar to the expected trajectory in the schematic drawing in Figure 4.4b. The RSE for the leaders are similar to that of the baseline experiment. The RSE of the followers are however improved compared to the baseline experiment. Drone 4 steadily increases in error from $t = 37$, as expected which relates to drifting away from its nominal position. Furthermore, drone 5 does stays in its nominal position and does

not follow drone 4, which implies that the RAL works as expected.

Table 4.2: RMSE comparison between baseline and experiment

Drone ID	$0 \leq t \leq 60$		$t \geq 37$	
	Baseline (m)	Experiment (m)	Baseline (m)	Experiment (m)
Leaders				
Drone 1	0.0944	0.0768	0.0950	0.0674
Drone 2	0.0790	0.0868	0.0671	0.0927
Drone 3	0.0415	0.0837	0.0372	0.1058
Average	0.0716	0.0824	0.0664	0.0886
Followers				
Drone 4	1.1039	0.8914	1.6492	1.4686
Drone 5	0.8655	0.1566	1.1996	0.1769

Table 4.2 contains the RMSE from each drone from both baseline and RAL experiment. The data is divided into two intervals: the entire experiment, $0 \leq t \leq 60$, and post-convergence, $t \geq 37$. For the duration of the whole experiment the leaders perform as expected within the margin of 10 cm. Drone 4 is not considered relevant for performance since it is expected to have a high RSE. The RMSE of drone 5 improved from 0.8655 m in the baseline experiment to 0.1566 m in the RAL experiment which is 70% in terms of RMSE for the duration of the whole experiment. However, only the data after convergence allows for an equal comparison, as the drones are expected to be at their nominal positions post-convergence. Post-convergence, the leaders perform as expected within the margin of 10 cm. Additionally, Drone 5 improved in accuracy from 1.1996 m to 0.1769 m, an 91% improvement by employing RAL. Recall from Section 3.6.3 that the RMSE for the follower for the 4 drone formation was 6.80 cm. Despite the 91% improvement in this 5 drone experiment, the RMSE is higher than 6.80 cm from the 4 drone experiment and also higher than the expected accuracy of 10 cm. A potential cause for the decrease in position accuracy is the usage of RAL in combination with more drones being present. In particular, the software of the drone not able to handle the RAL computation and the UWB packet handling simultaneously. Conclusively, the RAL algorithm work as expected by maintaining the formation in the case of an observation loss. However, there are still challenges regarding noise caused by the environment and potential computational overhead.

4.5 Summary

In this chapter the case of observation loss in affine formation control is discussed. An algorithm, RAL, was introduced which is able to infer the position of missing drones. The RAL is implemented with the DSP CMSIS library, which provides faster computation of matrix operations. Lastly, experiments were done to analyse the performance of RAL. The main results can be summarised as follows:

- The RAL allows the drones to successfully converge in formation in the case of

observation losses.

- The RMSE increased from 6.80 cm in Table 3.6 for the 4 drone experiment to 17.69 cm in Table 4.2 for the 5 drone RAL experiment.
- Potential causes of increased position inaccuracy using RAL includes computational overhead and more UWB traffic. Other research in similar conditions with systematic errors have shown an accuracy of larger than 15 cm which aligns with the 17.69 cm that was measured in the RAL experiment.

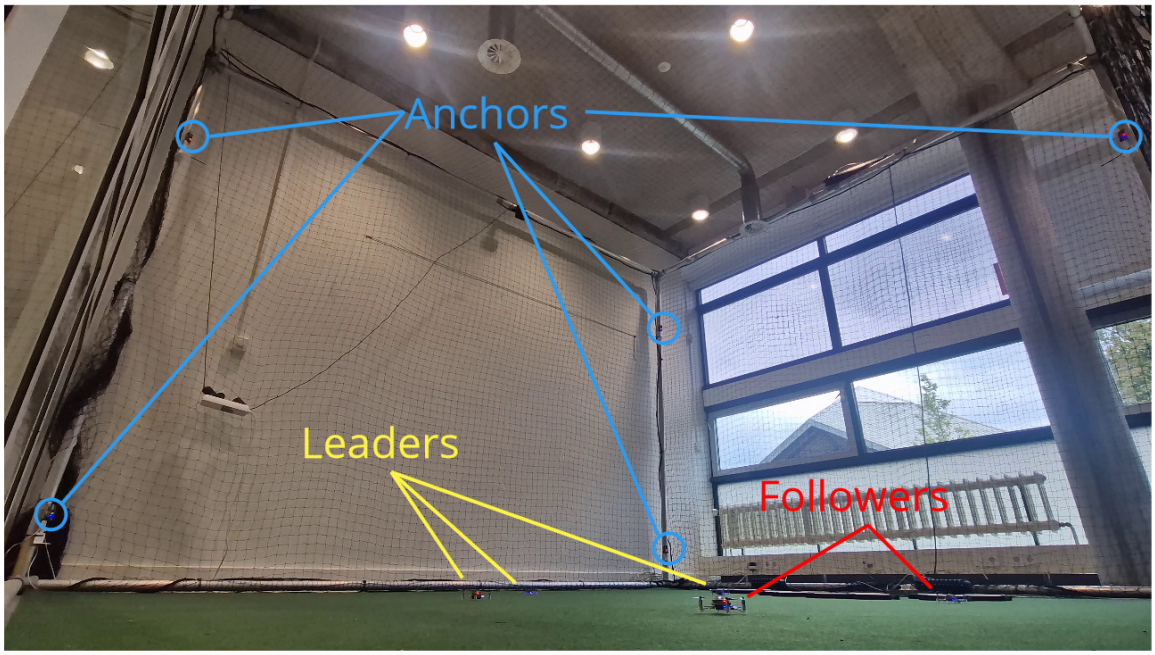


Figure 4.7: RAL Experiment with 5-drone formation at $t = 0$ before taking off

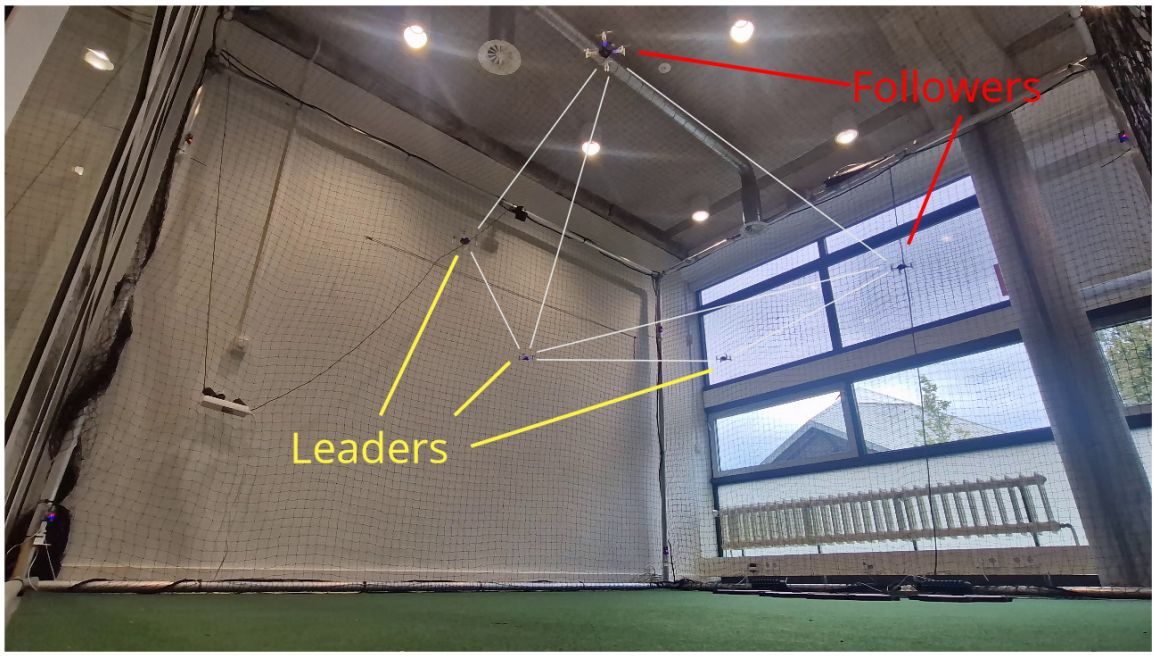


Figure 4.8: RAL Experiment with 5-drone formation at $t = 10$ in formation

Figure 4.9: RAL Experiment with 5-drone formation at $t = 45$ after missing a drone

5 Conclusion

This research focussed on implementing an affine formation control algorithm on drones and handle the case of observation losses. The positioning principle used on the drones is TDOA combined with TWR and is expected to be accurate up to 10 cm on average by hardware specification. The objectives mentioned in Section 1.4 are summarised as follows.

- The implementation of affine formation control required changing the execution order of handling UWB packets to prevent the microcontroller from redundant computation. Additionally a custom implementation of P2P communication was designed. In the case of observation losses the RAL algorithm is used to infer the position of the missing drones. The implementation of the RAL introduced challenges with matrix operations for the CPU. The usage of the DSP CMSIS library ensured a more error-prone implementation and more optimal execution since this library was specifically designed for the microcontroller present on the drones. Experiments with RAL showed that the formation stays intact in the case of a missing drone.

- The affine formation control algorithm showed an accuracy of approximately 6.80 cm, which is within the expected 10 cm. In the case of observation loss the RAL algorithm showed an accuracy of approximately 17.69 cm. Despite this being an improvement than without RAL, it is still a lower accuracy than the expected 10 cm.

- The usage of the affine formation control algorithm with P2P communication introduces UWB traffic. This is approximately 550 packet/s for a 4 drone formation, and increases as the formation grows in number of drones. Combining this with the execution of RAL equations, the computational load increases on the drones. These factors contribute to systemic errors, along with measurement noise such as signal propagation and multipath effects.

5.1 Future Work

Despite this research successfully implementing and demonstrating affine formation control and observation losses, there are several key areas that can be improved to enhance the findings.

A limitation of this research was the assumption that missing drones were manually indicated by the operator to all other drones. Ideally, the drones would be able to detect on their own if a neighbouring drone is missing. Additionally, the experiments were conducted a confined area. The limited size of the cage made it challenging to fly

more than five drones without risking collisions with the border of cage or with each other.

With the algorithm now implemented on hardware, it can serve as a baseline for future research into affine formation control. This includes noise analysis to understand where the noise originates from and how this is distributed. Furthermore, the speed and accuracy of convergence can be studied under different circumstances, such as outdoor environments or various number of drones. Lastly, while this research only considers two-dimensional formations, the application of three-dimensional formations remains empirically untested.

The accuracy of the positioning system is approximately 10 cm, which, depending on the use case, can be considered not accurate enough. An improvement of the positioning system can be considered by using alternative position estimation methods, such as TWR, TOA or an improved version of TDOA. Additionally, replacing the existing UWB chip, DWM1000, with a newer version, DWM3000, could also further improve the accuracy.

The firmware on the drones has been modified and fine-tuned for this research. However, this can still be further improved. The formation control algorithm and the RAL have been optimized for execution time, but the implementation's structure could still be enhanced such that it allows for scalability. In particular, being able to change formation or re-assign leaders during flight. Secondly, the RAL is currently manually triggered by the operator. Ideally, a missing drone detection feature that enables RAL automatically would be desired. Lastly, the software lacks collision avoidance measures. While the affine formation algorithm suggests collisions should not occur, implementing collision avoidance could serve as a valid safety measure in case the drones fail to maintain formation.

During the experiments, it became clear that the durability of some parts of the drones were insufficient. The fragile propellers often broke during causing the drones to crash during experiments. Additionally, the wires connecting the battery to the drone's motherboard occasionally failed due to poor quality wires or poor quality soldering. This causes the drones to suddenly switch off mid-flight cause the wires delivering power from the battery would disconnect. These weaknesses often led to crashes, causing the entire formation to collapse. Using more robust parts will increase the longevity of the drone and lower the amount crashes.

Bibliography

[1] K. W. Chan, U. Nirmal, and W. G. Cheaw, "Progress on drone technology and their applications: A comprehensive review," *AIP Conference Proceedings*, vol. 2030, no. 1, p. 020 308, Nov. 2018, ISSN: 0094-243X. DOI: 10.1063/1.5066949.

[2] K. Telli *et al.*, "A comprehensive review of recent research trends on unmanned aerial vehicles (uavs)," *Systems*, vol. 11, no. 8, 2023, ISSN: 2079-8954. DOI: 10.3390/systems11080400.

[3] S. A. H. Mohsan, N. Q. H. Othman, Y. Li, M. H. Alsharif, and M. A. Khan, "Unmanned aerial vehicles (uavs): Practical aspects, applications, open challenges, security issues, and future trends," *Intelligent Service Robotics*, vol. 16, no. 1, pp. 109137, Mar. 2023, ISSN: 1861-2784. DOI: 10.1007/s11370-022-00452-4.

[4] T. Gautam and R. Johari, "Drone: A systematicreview ofuavtechnologies," in *Proceedings of Fourth International Conference on Computing, Communications, and Cyber-Security*, S. Tanwar, S. T. Wierzchon, P. K. Singh, M. Ganzha, and G. Epiphaniou, Eds., Singapore: Springer Nature Singapore, 2023, pp. 147158, ISBN: 978-981-99-1479-1.

[5] F. Ahmed, J. C. Mohanta, A. Keshari, and P. S. Yadav, "Recent advances in unmanned aerial vehicles: A review," *Arabian Journal for Science and Engineering*, vol. 47, no. 7, pp. 79637984, Jul. 2022, ISSN: 2191-4281. DOI: 10.1007/s13369-022-06738-0.

[6] A. Rejeb, S. J. S. Rejeb, and H. Treiblmaier, "Drones for supply chain management and logistics: A review and research agenda," *International Journal of Logistics Research and Applications*, vol. 26, no. 6, pp. 708731, 2023. DOI: 10.1080/13675567.2021.1981273.

[7] A. Rejeb, A. Abdollahi, K. Rejeb, and H. Treiblmaier, "Drones in agriculture: A review and bibliometric analysis," *Computers and electronics in agriculture*, vol. 198, p. 107 017, 2022.

[8] A. Restas, "Drone applications for supporting disaster management," *World Journal of Engineering and Technology*, vol. 3, pp. 316321, 2015. DOI: 10.4236/wjet.2015.33C047.

[9] M. Hassanalian and A. Abdelkefi, "Classifications, applications, and design challenges of drones: A review," *Progress in Aerospace Sciences*, vol. 91, pp. 99131, 2017, ISSN: 0376-0421. DOI: 10.1016/j.paerosci.2017.04.003.

[10] M. Okasha, J. Kralev, and M. Islam, "Design and experimental comparison of pid, lqr and mpc stabilizing controllers for parrot mambo mini-drone," *Aerospace*, vol. 9, no. 6, 2022, ISSN: 2226-4310. DOI: 10.3390/aerospace9060298.

[11] P. Boccadoro, D. Striccoli, and L. A. Grieco, "An extensive survey on the internet of drones," *Ad Hoc Networks*, vol. 122, p. 102 600, 2021, ISSN: 1570-8705. DOI: 10.1016/j.adhoc.2021.102600.

[12] J. Tang, H. Duan, and S. Lao, "Swarm intelligence algorithms for multiple unmanned aerial vehicles collaboration: A comprehensive review," *Artificial Intelligence Review*, vol. 56, no. 5, pp. 42954327, May 2023, ISSN: 1573-7462. DOI: 10.1007/s10462-022-10281-7.

[13] J. Ota, "Multi-agent robot systems as distributed autonomous systems," *Advanced Engineering Informatics*, vol. 20, no. 1, pp. 5970, 2006, ISSN: 1474-0346. DOI: 10.1016/j.aei.2005.06.002.

[14] L. N. Duong *et al.*, "A review of robotics and autonomous systems in the food industry: From the supply chains perspective," *Trends in Food Science & Technology*, vol. 106, pp. 355364, 2020, ISSN: 0924-2244. DOI: 10.1016/j.tifs.2020.10.028.

[15] I. Karabegovi, E. Karabegovi, M. Mahmi, and E. Husak, "The application of service robots for logistics in manufacturing processes.," *Advances in Production Engineering & Management*, vol. 10, no. 4, 2015.

[16] J. Bourgeois *et al.*, *Distributed autonomous robotic systems*. Jan. 2024. DOI: 10.1007/978-3-031-51497-5.

[17] T. Halsted, O. Shorinwa, J. Yu, and M. Schwager, "A survey of distributed optimization methods for multi-robot systems," *CoRR*, vol. abs/2103.12840, 2021. DOI: 10.48550/arXiv.2103.12840.

[18] S. Ahirwar, R. Swarnkar, S. Bhukya, and G. Namwade, "Application of drone in agriculture," *International Journal of Current Microbiology and Applied Sciences*, vol. 8, no. 01, pp. 25002505, Jan. 10, 2019. DOI: 10.20546/ijcmas.2019.801.264. [Online]. Available: 10.20546/ijcmas.2019.801.264.

[19] S. M. S. Mohd Daud *et al.*, "Applications of drone in disaster management: A scoping review," *Science Justice*, vol. 62, no. 1, pp. 3042, 2022, ISSN: 1355-0306. DOI: 10.1016/j.scijus.2021.11.002.

[20] S. Qamar, S. H. Khan, M. A. Arshad, M. Qamar, J. Gwak, and A. Khan, "Autonomous drone swarm navigation and multitarget tracking with island policy-based optimization framework," *IEEE Access*, vol. 10, pp. 91 07391 091, 2022. DOI: 10.1109/ACCESS.2022.3202208.

[21] T. Uzakov, T. P. Nascimento, and M. Saska, "Uav vision-based nonlinear formation control applied to inspection of electrical power lines," in *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2020, pp. 13011308. DOI: 10.1109/ICUAS48674.2020.9213967.

[22] R. H. Jacobsen *et al.*, "Design of an autonomous cooperative drone swarm for inspections of safety critical infrastructure," *Applied Sciences*, vol. 13, no. 3, 2023, ISSN: 2076-3417. DOI: 10.3390/app13031256.

[23] K.-K. Oh, M.-C. Park, and H.-S. Ahn, "A survey of multi-agent formation control," *Automatica*, vol. 53, pp. 424440, 2015, ISSN: 0005-1098. DOI: 10.1016/j.automatica.2014.10.022.

[24] Y. Liu and R. Bucknall, "A survey of formation control and motion planning of multiple unmanned vehicles," *Robotica*, vol. 36, no. 7, pp. 10191047, 2018. DOI: 10.1017/S0263574718000218.

[25] P. Zhang, G. Chen, Y. Li, and W. Dong, "Agile formation control of drone flocking enhanced with active vision-based relative localization," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 63596366, 2022. DOI: 10.1109/LRA.2022.3171096.

[26] K. M. Kabore and S. Gler, "Distributed formation control of drones with onboard perception," *IEEE/ASME Transactions on Mechatronics*, vol. 27, no. 5, pp. 3121 3131, 2022. DOI: 10.1109/TMECH.2021.3110660.

[27] Y. Bu, Y. Yan, and Y. Yang, "Advancement challenges in uav swarm formation control: A comprehensive review," *Drones*, vol. 8, no. 7, 2024, ISSN: 2504-446X. DOI: 10.3390/drones8070320.

[28] Y. Ding, X. Wang, Y. Cong, and H. Li, "Scalability analysis of algebraic graph-based multi-uavs formation control," *IEEE Access*, vol. 7, pp. 129 719129 733, 2019. DOI: 10.1109/ACCESS.2019.2938991.

[29] F. Saffre, H. Hildmann, and H. Karvonen, "The design challenges of drone swarm control," in *Engineering Psychology and Cognitive Ergonomics*, D. Harris and W.-C. Li, Eds., Cham: Springer International Publishing, 2021, pp. 408426, ISBN: 978-3-030-77932-0.

[30] P. Peng, W. Dong, G. Chen, and X. Zhu, "Obstacle avoidance of resilient uav swarm formation with active sensing system in the dense environment," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022, pp. 10 52910 535. DOI: 10.1109/IROS47612.2022.9981858.

[31] Y. Liu, X. Lin, Y. Liu, A. Jiang, and C. Zhang, "Affine formation maneuver control of underactuated surface vessels: Guaranteed safety under moving obstacles in narrow channels," *Ocean Engineering*, vol. 303, p. 117 721, 2024, ISSN: 0029-8018. DOI: 10.1016/j.oceaneng.2024.117721.

[32] *CrazyFlie 2.1 | BitCraze*. [Online]. Available: https://www.bitcraze.io/products/old-products/crazyflie-2-1/.

[33] *STM32F405/415*, Ver. 9, STMicroElectronics, 2020. [Online]. Available: https://www.st.com/resource/en/datasheet/stm32f415rg.pdf.

[34] S. Mittal and J. S. Vetter, "A survey of techniques for architecting dram caches," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 6, pp. 1852 1863, 2016. DOI: 10.1109/TPDS.2015.2461155.

[35] *nRF51822*, Ver. 3.4, Nordic semiconductor, 2023. [Online]. Available: https://docs.nordicsemi.com/bundle/nRF51-Series/resource/nRF51822_PS_v3.4.pdf.

[36] *DWM1000*, Rev. 1.8, Qorvo, 2016. [Online]. Available: https://www.qorvo.com/products/d/da007948.

[37] *Loco Positioning deck | BitCraze*. [Online]. Available: https://www.bitcraze.io/products/loco-positioning-deck/.

[38] Amazon Web Services, *FreeRTOS Real-Time Operating System*, version 10.4.0, https://www.freertos.org, 2020.

[39] A. Ledergerber, M. Hamer, and R. D'Andrea, "A robot self-localization system using one-way ultra-wideband communication," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 31313137. DOI: 10.1109/IROS.2015.7353810.

[40] H. Zhi, L. Chen, C. Li, and Y. Guo, "Leaderfollower affine formation control of second-order nonlinear uncertain multi-agent systems," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 12, pp. 35473551, 2021. DOI: 10.1109/TCSII.2021.3072652.

[41] Z. Lin, L. Wang, Z. Chen, M. Fu, and Z. Han, "Necessary and sufficient graphical conditions for affine formation control," *IEEE Transactions on Automatic Control*, vol. 61, no. 10, pp. 28772891, 2016. DOI: 10.1109/TAC.2015.2504265.

[42] C. Pearanda, V. Julian, J. Palanca, and V. Botti, "A multi-agent system to improve mobile robot localization," in *Hybrid Artificial Intelligent Systems*, F. J. Martnez de Pisn, R. Urraca, H. Quintin, and E. Corchado, Eds., Cham: Springer International Publishing, 2017, pp. 471482, ISBN: 978-3-319-59650-1.

[43] R. F. Brena, J. P. Garca-Vzquez, C. E. Galvn-Tejada, D. Muoz-Rodriguez, C. Vargas-Rosales, and J. Fangmeyer Jr., "Evolution of indoor positioning technologies: A survey," *Journal of Sensors*, vol. 2017, no. 1, p. 2 630 413, 2017. DOI: 10.1155/2017/2630413.

[44] L. Mainetti, L. Patrono, and I. Sergi, "A survey on indoor positioning systems," in *2014 22nd International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, 2014, pp. 111120. DOI: 10.1109/SOFTCOM.2014.7039067.

[45] M. Kwak and J. Chong, "A new double two-way ranging algorithm for ranging system," in *2010 2nd IEEE InternationalConference on Network Infrastructure and Digital Content*, 2010, pp. 470473. DOI: 10.1109/ICNIDC.2010.5657814.

[46] S.-H. Lee, I.-K. Lim, and J.-K. Lee, "Method for improving indoor positioning accuracy using extended kalman filter," *Mobile Information Systems*, vol. 2016, no. 1, p. 2 369 103, 2016. DOI: 10.1155/2016/2369103.

[47] T. G. Hailu, X. Guo, H. Si, L. Li, and Y. Zhang, "Theories and methods for indoor positioning systems: A comparative analysis, challenges, and prospective measures," *Sensors*, vol. 24, no. 21, 2024, ISSN: 1424-8220. DOI: 10.3390/s24216876.

[48] S. (Zekavat *et al.*, "An overview on position location: Past, present, future," *International Journal of Wireless Information Networks*, vol. 28, no. 1, pp. 4576, Mar. 2021, ISSN: 1572-8129. DOI: 10.1007/s10776-021-00504-z.

[49] T. Sathyan, A. Sinha, and T. Kirubarajan, "Passive geolocation and tracking of an unknown number of emitters," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 42, no. 2, pp. 740750, 2006. DOI: 10.1109/TAES.2006.1642587.

[50] J. Smith and J. Abel, "Closed-form least-squares source location estimation from range-difference measurements," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 35, no. 12, pp. 16611669, 1987. DOI: 10.1109/TASSP.1987.1165089.

[51] W. Zhao, J. Panerati, and A. P. Schoellig, *Learning-based bias correction for time difference of arrival ultra-wideband localization of resource-constrained mobile robots*, 2021. DOI: 10.48550/arXiv.2103.01885.

[52] W. Zhao, A. Goudar, and A. P. Schoellig, "Finding the right place: Sensor placement for uwb time difference of arrival localization in cluttered indoor environments," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 60756082, Jul. 2022, ISSN: 2377-3774. DOI: 10.1109/lra.2022.3165181.

[53] S. Zhao, "Affine formation maneuver control of multiagent systems," *IEEE Transactions on Automatic Control*, vol. 63, no. 12, pp. 41404155, 2018. DOI: 10.1109/TAC.2018.2798805.

[54] J. Wang, X. Ding, C. Wang, Z. Zuo, and Z. Ding, "Affine formation control of general linear multi-agent systems with delays," *Unmanned Systems*, vol. 11, no. 02, pp. 123132, 2023. DOI: 10.1142/S2301385023410017.

[55] Y. Li and W. Dong, *A flexible and resilient formation approach based on hierarchical reorganization*, 2024. DOI: 10.48550/arXiv.2406.11219.

[56] Z. Li and R. Rajan, *Robust affine formation control of multiagent systems*, Jul. 2024. DOI: 10.48550/arXiv.2407.03911.

[57] Z. Li and R. T. Rajan, "Geometry-aware distributed kalman filtering for affine formation control under observation losses," in *2023 26th International Conference on Information Fusion (FUSION)*, 2023, pp. 17. DOI: 10.23919/FUSION52260.2023.10224101.

[58] F. Molinari and J. Raisch, "Efficient consensus-based formation control with discrete-time broadcast updates," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, 2019, pp. 41724177. DOI: 10.1109/CDC40024.2019.9029346.

[59] ARM-software, *Cmsis-dsp: Cmsis-dsp embedded compute library for cortex-m and cortex-a*, GitHub, 2022.

[60] L. T, *The dsp capabilities of arm cortex-m4 and cortex-m7 processors*, ARM, 2016. [Online]. Available: https://community.arm.com/cfs-file/__key/communityserver-blogs-components-weblogfiles/00-00-00-21-42/7563.ARM-white-paper-_2D00_-DSP-capabilities-of-Cortex_2D00_M4-and-Cortex_2D00_M7.pdf.

[61] J. Huang and T. Leng, "Generalized loop-unrolling: A method for program speedup," in *Proceedings 1999 IEEE Symposium on Application-Specific Systems and Software Engineering and Technology. ASSET'99 (Cat. No.PR00122)*, 1999, pp. 244248. DOI: 10.1109/ASSET.1999.756775.

[62] A. Masiero, F. Fissore, and A. Vettore, "A low cost uwb based solution for direct georeferencing uav photogrammetry," *Remote Sensing*, vol. 9, no. 5, 2017, ISSN: 2072-4292. DOI: 10.3390/rs9050414.

Crazyflie 2.1

A

A.1 Hardware specifications

The following specifications in Table A.1 are taken from the Bitcraze website [32].

Table A.1: Hardware specification of Crazyflie 2.1

Type	Properties	Note	Image
Weight	27 grams		
Battery	300mAh	±7min flying ±40min charging	
Microcontroller	Clock frequency: 168MHz RAM: 192Kb SRAM Storage: 1Mb flash	Cortex M4 / STM32F405	
Accelerometer	3-axis gyroscope	BMPI088	
Pressure sensor		BMP388	
Storage	8KB EEPROM		
Radio	2.4GHz ISM band	nRF51822	

A.1.1 Additional Decks

The crazyflie's functionality can be extended by assembling so-called expansions decks. Table A.2 shows some decks the Crazyflie supports and their functionality. All information is taken from the Bitcraze website [32]

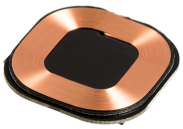
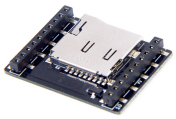
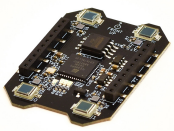
A.1.2 Positioning systems

The Crazyflie has multiple options to use for positioning

Lighthouse

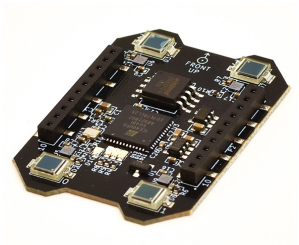
So called lighthouse beacons emit infrared laser scans which are detected by the infrared sensor on the Crazyflie. The beacons are fixed in-place and require a direct line of sight with the drone to estimate the drones position accurately.

Table A.2: Additional deck used on the Crazyflie

Type	Properties	Image
AI deck	Processor for machine learning	
LED-ring deck	12 downward facing leds	
Qi 1.2 charger deck	Wireless charging	
MicroSD Card deck	Logging data	
Flow deck	Optical flow sensor for ground movement	
Loco deck	UWB for positioning	
Lighthouse deck	Infrared sensor for positioning	

Loco Positioning System

A combination of reference anchors at fixed positions and an anchor on the Crazyflie. These anchors communicate over UWB and apply the TDOA principle to estimate its current position.

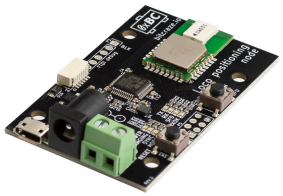


(a) Lighthouse deck for receiving infrared signals

(b) Infrared base station used as reference point

Figure A.1: Hardware required for the lighthouse positioning system

(a) UWB deck used for receiving UWB signals



(b) UWB Anchor used as reference point

Figure A.2: Hardware required for the LPS

Motion Capture Positioning

A (third-party) motion capture camera uses markers on the drone to detect its position in a global reference frame. The cameras use infrared flashes and detect reflections from the markers.

Table of Functions

B

Table B.1: Table of all functions used in this research

Function	Parameters	Description
Main functions		
AFC	i: Drone ID	Run Algorithm 1
RAL	i: Drone ID	Run Algorithm 4
Helper functions		
GetCurrentPosition	-	Return current position of drone
HybridMode	-	Transmit position to other drones
PacketHandler	Packet: UWB packet	Process data of UWB packet
Mat_Inv	$\mathbf{A}$: Matrix $\mathbf{A}$	Invert matrix $\mathbf{A}$ or return -1 if invertible
Mat_Mult	$\mathbf{A}, \mathbf{B}$: Matrix $\mathbf{A}$ and $\mathbf{B}$	Multiply matrix $\mathbf{A}$ with matrix $\mathbf{B}$
Mat_Trans	$\mathbf{A}$: Matrix $\mathbf{A}$	Transpose matrix $\mathbf{A}$
SendP2PPacket	$\mathbf{z}_i$: Position of drone i	Transmit position to other drones
SetVelocity	$\dot{\mathbf{z}}_i$: velocity vector	Send velocity to motors

Technical note on positioning dataset

C

UWB experiments with Crazyflie quadcopters

Shaan Hossain

***Abstract*—This technical note accompanies datasets providing positions and UWB communication transmission frequency of the Crazyflie quadcopter. This note explains the context, creation, format, and potential uses of the dataset. The code used to create these data is publicly available.**

I. Context

The Crazyflie is a nano quadcopter developed by Bitcraze for educational research and hobbyist purposes. Bitcraze offers a variety of tools and hardware that can be utilised in conjunction with the quadcopter, including positioning systems, upgraded motors, and additional stabilisers. One of the positioning systems compatible with the Crazyflie is the Loco Positioning System (LPS), which incorporates Ultra Wide Band (UWB) expansion boards on the quadcopter and communicates with designated UWB anchors [1]. These UWB anchors have fixed positions and act as a reference to the quadcopters. By transmitting packets and calculating the Time Difference Of Arrival (TDOA) between the quadcopter and the anchors, the system can estimate the quadcopter's position [2].

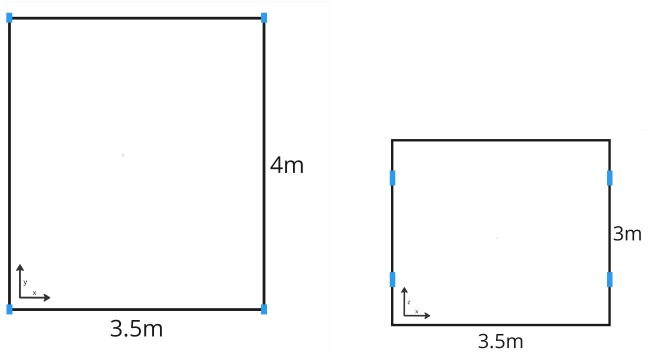
Experiments were executed with two goals in mind. Firstly, determining the accuracy of the position estimated by the LPS under various conditions. Here the position of the quadcopters are constantly polled to measure This data allows to draws conclusions on how number of quadcopters affect the LPS.

Secondly, another set of experiments was done to measure at what rate UWB packets are received on the quadcopter. Each quadcopter is broadcasting their current position to other quadcopters. Totally, These include packets sent by UWB-anchors as well as other quadcopters. Packets from the anchors are used for TDOA and ultimately position estimation. Packets from other quadcopters are used to run a formation control algorithm.

II. Setup

All the experiments were done inside a cage with dimensions $4\times3.5\times3$ meters in length, width and height respectively. Each vertical pole of the cage has two UWB anchors placed, 0.3m and 2.3m from the ground respectively, these are marked blue in figure 1. The position of each anchor is fixed and known to all quadcopters.

The quadcopters are running a modified version of the stock firmware. This modification allows the quadcopters to communicate with other quadcopters alongside the UWB anchors. This is necessary to run the formation control algorithm. Additionally, a python script is used to command the quadcopters to take off, engage formation and land.



(a) Top view (b) Front view

Fig. 1: Schematic of cage setup

III. Dataset

All relevant data and additional instructions can be find on Github[1]

A. Tick rate

The goal of these experiments was to measure the performance of UWB communication with varying numbers of quadcopters. Table I provides a description of the data fields collected during these experiments. Data was collected from a single quadcopter for the duration of 30 seconds with a polling interval of 10ms. This quadcopter is placed at the centre of the cage, which is considered the origin. In the case of other quadcopters, these were placed randomly and evenly spaced from each other inside the cage. The experiment also varies in number of anchors that are present. These anchors allow us to estimate the position of each quadcopter. Note that at least 2 or more anchors are required to estimate positions. Figure 2 shows a schematic scenario of an experiment with 2 anchors and 2 quadcopters. The anchors send TDOA packets to the quadcopters (depicted in blue), and the quadcopters are communicating amongst each other (depicted in green).

The data files follow a specific naming convention: tickLocoRate-x-y.csv. Here, x represents the number of activated anchors, and y represents the number of quadcopters present. For instance, a file named tickLocoRate-4-6.csv would contain data from an experiment with 4 anchors and 6 quadcopters. For this set of experiments, data of each scenario was collected once.

B. Localisation

The goal of the localisation experiments was to measure the accuracy of the LPS under different conditions. These

[1]https://github.com/shossains/dollaFlie/tree/master/dollaFlie/logs

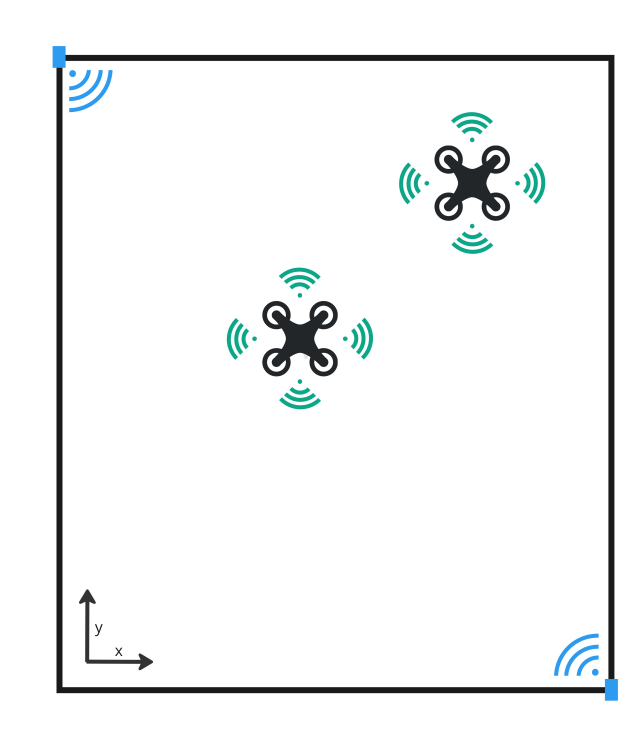


Fig. 2: Example scenario with 2 anchors and 2 quadcopters

TABLE I: Headers of data files for tickrate experiments

header	description	unit
rxRate	All packets that are received over UWB	packets/s
posUpdateRate	Packets that are used for localisation	packets/s
P2PRate	Packets from other quadcopters	packets/s
controlRate	Formation control algorithm execution	executions/s

conditions involve changing the number of quadcopters and transitioning between stationary on the ground, stationary in the air, and flying at a constant speed. The logging of positions occurred concurrently for all quadcopters during each polling instance. Table II provides a summary of the data collected for each experiment. The naming convention for the file names of these experiments are explained in the readme file.

1) Experiment 3: During this set of experiments the quadcopters were placed on the ground. The number of quadcopters used for these experiments were 1, 2, 3, 4 and 5 quadcopter(s) at fixed locations on the ground. Using UWB constantly polling current location.

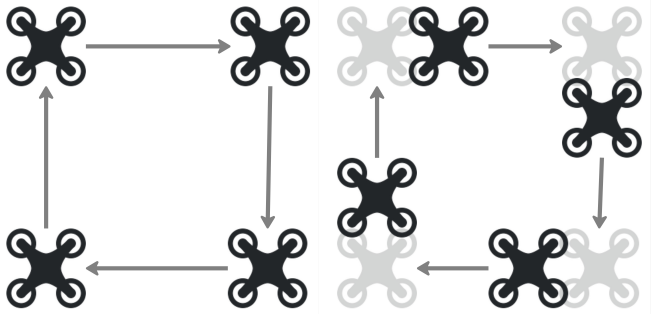
2) Experiment 4: During this set of experiments quadcopters were flying in a square formation. Each quadcopter flew at a fixed position at one of the corners of the formation.

TABLE II: Headers of data files for localisation experiments

header	description	unit
x	x coordinate of quadcopter	meter
y	y coordinate of quadcopter	meter
z	z coordinate of quadcopter	meter
uri	Id of the drone	-

The positions of the quadcopters were polled for 30 seconds. The number of quadcopters used for these experiments were 1, 2, 3 and 4 quadcopters.

3) Experiment 5: During this set of experiments 4 quadcopters were flying in a square formation. Each quadcopter started at one of the corners of the formation. They moved at a constant speed from their starting position to the next corner in the clockwise direction, see figure 4. When arrived at the next corner the quadcopters land on the ground.



(a) Situation at $t = 0$ (b) Situation at $t > 0$

Fig. 4: Top-down view of experiment 5

Figure 3 shows some example plots of positioning data of the quadcopters flying at a speed of 0.2m/s for 10 seconds.

IV. Comments

For certain experiments or iterations of experiments, the quadcopters may have been swapped from their positions. For instance, an experiment using four drones in a square formation may have been rearranged when the experiment was repeated. This was done so anomalous behaviour could be differentiated between a faulty quadcopter or environmental influences.

References

[1] *Loco Positioning system — Bitcraze*. [Online]. Available: https : / / www . bitcraze . io / documentation / system / positioning/loco-positioning-system/.

[2] *TDoA principles — Bitcraze*. [Online]. Available: https://www.bitcraze.io/documentation/repository/lps-node-firmware/master/functional-areas/tdoa_principles/.

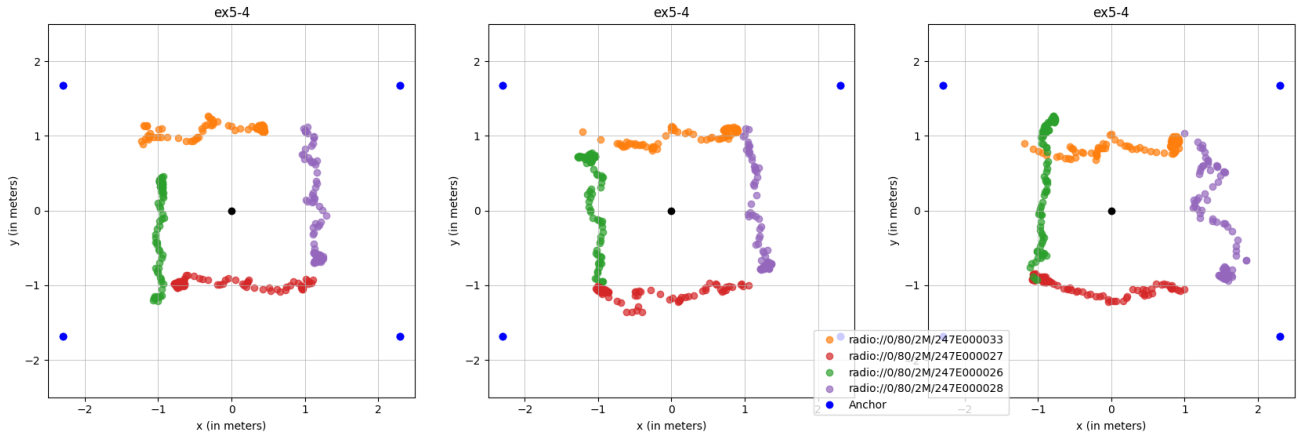


Fig. 3: Trajectory plots of files YYYYmmdd-HHMMSS-ex5-4-0.5-4.csv

Source code

D

D.1 Affine Formation Control implementation in C

```
1  //                3 ------ 2
2  //                | \    / |
3  //                |   /\   |                3 ----- 0 ----- 2 ----- 1
4  //                | /    \ |
5  //                0 ------ 1
6
7  #define numberOfDrones 4
8  uint8_t missing = 255;
9  uint8_t droneIds[numberOfDrones] = {0x18, 0x19, 0x20, 0x26};
10
11 float weights[numberOfDrones][numberOfDrones] = {
12     {1.0, -1.0, 1.0, -1.0},
13     {-1.0, 1.0, -1.0, 1.0},
14     {1.0, -1.0, 1.0, -1.0},
15     {-1.0, 1.0, -1.0, 1.0}};
16
17 neighbours_t adjacency[numberOfDrones] = {
18     {(uint8_t[]){1, 2, 3}, 3},
19     {(uint8_t[]){0, 2, 3}, 3},
20     {(uint8_t[]){0, 1, 3}, 3},
21     {(uint8_t[]){0, 1, 2}, 3}};
22
23 pos_t positions[numberOfDrones] = {
24     {0.0, 0.0, 0.0},
25     {0.0, 0.0, 0.0},
26     {0.0, 0.0, 0.0},
27     {0.0, 0.0, 0.0}};
28
29 uint8_t getNormalisedDroneId(uint8_t droneId)
30 {
31     for (int i = 0; i < numberOfDrones; i++)
32     {
33         if (droneIds[i] == droneId)
34         {
35             return i;
36         }
37     }
38
39     return 255;
40 }
41
42 // Update position of drone
```

```
43 void updatePositions(uint8_t droneId, float x, float y, float z)
44 {
45     uint8_t id = getNormalisedDroneId(droneId);
46 
47     if (id != 255 && id != missing)
48     {
49         positions[id].x = x;
50         positions[id].y = y;
51         positions[id].z = z;
52     }
53 }
54 
55 // Send target velocity to commander
56 void setVelocitySetpoint(setpoint_t *setpoint, float vx, float vy, float
       z, float yawrate)
57 {
58     setpoint->mode.x = modeVelocity;
59     setpoint->mode.y = modeVelocity;
60     setpoint->mode.z = modeAbs;
61     setpoint->mode.yaw = modeVelocity;
62     setpoint->velocity.x = vx;
63     setpoint->velocity.y = vy;
64     setpoint->position.z = z;
65     setpoint->attitudeRate.yaw = yawrate;
66 
67     setpoint->velocity_body = true;
68 
69     commanderSetSetpoint(setpoint, 2);
70 }
71 
72 // Send target position to commander
73 void setPositionSetpoint(setpoint_t *setpoint, float x, float y, float z,
        float yawrate)
74 {
75     setpoint->mode.x = modeAbs;
76     setpoint->mode.y = modeAbs;
77     setpoint->mode.z = modeAbs;
78     setpoint->mode.yaw = modeAbs;
79 
80     setpoint->position.x = x;
81     setpoint->position.y = y;
82     setpoint->position.z = z;
83     setpoint->attitudeRate.yaw = yawrate;
84 
85     setpoint->velocity_body = false;
86 
87     commanderSetSetpoint(setpoint, 2);
88 }
89 
90 // Main equation
91 void calcVelocity(uint8_t droneId)
92 {
93     // Get most recent known position of self
```

```
94      pos_t curPos;
95      curPos.x = logGetFloat(logGetVarId("stateEstimate", "x"));
96      curPos.y = logGetFloat(logGetVarId("stateEstimate", "y"));
97      // curPos.z = logGetFloat(logGetVarId("stateEstimate", "z"));
98
99      uint8_t id = getNormalisedDroneId(droneId);
100     if (id != 255)
101     {
102         uint8_t numberOfNeighbours = adjacency[id].numberOfNeighbours;
103         pos_t sum = {0.0, 0.0, 0.0};
104         uint8_t i;
105
106         for (i = 0; i < numberOfNeighbours; i++)
107         {
108             uint8_t curNeighbour = adjacency[id].neighbours[i];
109
110             pos_t neighbourPosition = positions[curNeighbour];
111             sum.x = sum.x + (weights[id][curNeighbour] * (curPos.x -
                    neighbourPosition.x));
112             sum.y = sum.y + (weights[id][curNeighbour] * (curPos.y -
                    neighbourPosition.y));
113             // sum.z = sum.z + (weights[id][curNeighbour] * (curPos.z -
                    neighbourPosition.z));
114         }
115
116         setVelocitySetpoint(&setpoint, sum.x, sum.y, 1.5, 0);
117     }
118 }
119
120 // Main loop
121 void formationControlLoop(uint8_t droneId)
122 {
123     // Leaders fly to dedicated position
124     switch (droneId)
125     {
126     case 0x17:
127         setPositionSetpoint(&setpoint, 1.0, 0.0, 1.5, 0);
128         break;
129     case 0x18:
130         setPositionSetpoint(&setpoint, 0.33, 1, 1.5, 0);
131         break;
132     case 0x19:
133         setPositionSetpoint(&setpoint, 0.33, -1, 1.5, 0);
134         break;
135     default: // Followers
136         calcVelocity(droneId);
137         break;
138     }
139
140     return;
141 }
```

Listing D.1: Implementaion of Formation Control algorithm in C

D.2 RAL implementation in C

```
1  void RAL(uint8_t i)
2  {
3      uint8_t numberOfNeighbours = adjacency[i].numberOfNeighbours;
4      uint8_t numberOfMissingDrones = 1; // Number of missing neighbours
5      uint8_t M = numberOfNeighbours - numberOfMissingDrones; // Number of
           known neighbours
6      uint8_t N = 2; // Dimension of formation
7
8      float32_t vp_ij[N][1];
9      float32_t vH_i[M][N];
10     float32_t vX_i[M][N];
11     float32_t vH_i_T[N * M];
12     float32_t vH_i_T_H_i[N * N];
13     float32_t vH_i_T_X_i[N * N];
14     float32_t vH_i_T_H_i_inv[N * N];
15     float32_t vtheta_i[N * N];
16     float32_t vtheta_i_T[N * N];
17     float32_t vtheta_i_T_p_ij[N * 1];
18
19     // Populate p_ij = p[i] - p[missing]
20     pos_t p_i = p[i];
21     pos_t p_missing = p[missing];
22     vp_ij[0][0] = p_i.x - p_missing.x;
23     vp_ij[1][0] = p_i.y - p_missing.y;
24     vp_ij[2][0] = p_i.z - p_missing.z;
25
26     pos_t curPos;
27     curPos.x = logGetFloat(logGetVarId("stateEstimate", "x"));
28     curPos.y = logGetFloat(logGetVarId("stateEstimate", "y"));
29     curPos.z = logGetFloat(logGetVarId("stateEstimate", "z"));
30
31     // Populate H_i and X_i
32     uint8_t j = 0;
33     uint8_t write = 0;
34     for (j = 0; j < numberOfNeighbours; j++)
35     {
36         uint8_t curNeighbour = adjacency[i].neighbours[j];
37         if (curNeighbour != missing)
38         {
39             // Populate H_i = formation displacement of known neighbours
                   (p_ij)
40             pos_t p_neighbour = p[curNeighbour];
41             vH_i[write][0] = p_i.x - p_neighbour.x;
42             vH_i[write][1] = p_i.y - p_neighbour.y;
43             vH_i[write][2] = p_i.z - p_neighbour.z;
44
45             // Populate X_i = actual positions of known neighbours (z_ij)
46             pos_t neighbourPosition = positions[curNeighbour];
47             vX_i[write][0] = curPos.x - neighbourPosition.x;
48             vX_i[write][1] = curPos.y - neighbourPosition.y;
```

```
49              vX_i[write][2] = curPos.z - neighbourPosition.z;
50
51              write++;
52          }
53      }
54
55      // Initialize the ARM matrix structures
56      arm_matrix_instance_f32 p_ij = {N, 1, (float32_t *)vp_ij};
57      arm_matrix_instance_f32 H_i = {M, N, (float32_t *)vH_i};
58      arm_matrix_instance_f32 X_i = {M, N, (float32_t *)vX_i};
59      arm_matrix_instance_f32 H_i_T = {N, M, vH_i_T};
60      arm_matrix_instance_f32 H_i_T_H_i = {N, N, vH_i_T_H_i};
61      arm_matrix_instance_f32 H_i_T_X_i = {N, N, vH_i_T_X_i};
62      arm_matrix_instance_f32 H_i_T_H_i_inv = {N, N, vH_i_T_H_i_inv};
63      arm_matrix_instance_f32 theta_i = {N, N, vtheta_i};
64      arm_matrix_instance_f32 theta_i_T = {N, N, vtheta_i_T};
65      arm_matrix_instance_f32 theta_i_T_p_ij = {N, 1, vtheta_i_T_p_ij};
66
67      // Transpose H_i
68      mat_trans(&H_i, &H_i_T);
69
70      // Calculate H_i.T @ H_i
71      mat_mult(&H_i_T, &H_i, &H_i_T_H_i);
72
73      // Calculate H_i.T @ X_i
74      mat_mult(&H_i_T, &X_i, &H_i_T_X_i);
75
76      // Calculate inverse of H_i.T @ H_i
77      mat_inv(&H_i_T_H_i, &H_i_T_H_i_inv);
78
79      // Calculate theta_i
80      mat_mult(&H_i_T_H_i_inv, &H_i_T_X_i, &theta_i);
81
82      // Transpose theta_i
83      mat_trans(&theta_i, &theta_i_T);
84
85      // Calcutate z_ij
86      mat_mult(&theta_i_T, &p_ij, &theta_i_T_p_ij);
87
88      // Update position of missing drone
89      positions[missing].x = curPos.x - theta_i_T_p_ij.pData[0];
90      positions[missing].y = curPos.y - theta_i_T_p_ij.pData[1];
91      positions[missing].z = curPos.z - theta_i_T_p_ij.pData[2];
92  }
```

Listing D.2: Implementaion of RAL algorithm in C