Autonomous control of unmanned aerial multi-agent networks in confined spaces

Sebastian Green

Pontus Månsson

LUND UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6084
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Tryckeriet i E-huset
Lund 2019

Abstract

In this thesis, the functionality of existing Crazyflie models is extended to include current implementations of attitude and velocity control. The models are validated both as individual agents and as a swarm through comparisons of swarm behaviour during step, ramp and frequency response testing. The models are found to accurately replicate the dynamics of the real system, but additional research on measurement noise and accuracy would be necessary to ensure that the model remains accurate in less than ideal conditions. Several suggestions for further improvements are presented. Additionally, two separate swarm controllers are implemented and tested in a simulated environment as well as the real world to demonstrate the capabilities of the model and to evaluate the controller usability in a number of practical use cases. Both controllers are found to behave well on the Crazyflie system, and demonstrate the practicality of the Crazyflie platform as well as the controllers implemented.

Acknowledgements

Firstly we would like to thank Simon Yngve at Combine Control Systems for enabling us to do this Master's Thesis, not only by the support of hardware but also through helpful advice and clever ideas. Further gratitude goes out to Anton Holm and Pablo Correa Gómez at the same office for general help and feedback on our ideas and work. It has been a true pleasure working at Combine.

We would also like to thank the Department of Automatic Control, Faculty of Engineering, Lund University, more specifically Anders Robertsson for your never-ending enthusiastic support and supervision as well as guiding insights. Our swarm specialist, Zhiyong Sun, also deserves special thanks for your brilliant and very applicable Ph.D. thesis, your control theory and general UAV inputs and not to mention the great feedback. We are also thankful to Marcus Greiff for his Crazyflie models on which we based much of our work.

Lastly, a thank you goes out to Bitcraze for providing a great and very promising UAV platform, for quick support when we have had issues and for allowing us early access to the high precision Lighthouse deck.

Contents

1. Introduction **9**
- 1.1 Problem Statement . . . 9
- 1.2 Goal and purpose . . . 9
- 1.3 Thesis outline . . . 10
- 1.4 Division of labour . . . 10

2. Background **12**
- 2.1 Possible applications . . . 12
- 2.2 Previous research . . . 13
- 2.3 Trajectory control . . . 15

3. Modelling **17**
- 3.1 Rigid body dynamics . . . 18
- 3.2 Attitude controller . . . 22
- 3.3 Velocity controller . . . 24
- 3.4 Swarm modelling . . . 30

4. Real world implementation **31**
- 4.1 Bill of materials . . . 31
- 4.2 Lighthouse positioning . . . 31
- 4.3 Crazyflie firmware . . . 32
- 4.4 Swarm Client Software . . . 34

5. Swarm control **38**
- 5.1 Method . . . 38
- 5.2 Analysis . . . 43
- 5.3 Distance-based swarm controller . . . 44
- 5.4 Flocking-based swarm controller . . . 46
- 5.5 Results . . . 49
- 5.6 Swarm model validation . . . 58

6. Discussion **62**
- 6.1 Crazyflie firmware modelling . . . 62
- 6.2 Swarm modelling . . . 63

6.3 Python implementation 63
6.4 Methodology and analysis 64
6.5 Controller comparison 65
6.6 Further work 66
Bibliography **68**

1

Introduction

1.1 Problem Statement

The usage of drones has been increasing much lately with applications such as search and rescue or warehouses. Many of the possible applications profit from the use of multiple drones, however, the problem is centered on the fact that it is very hard to avoid collisions if each drone is controlled individually. We would therefore want to find and test two control processes which control the swarm's common position and speed, while each individual drone follows this position as precisely as possible while not colliding with other drones. Swarm in the context of this thesis pertains to a group of UAVs cooperating to achieve a common goal or to simplify their individual tasks. The number of drones should not impact the difficulty of operating them, but the drones should still be able to react to disturbances such as if a vessel is pushed towards the rest of the swarm.

Further, we aim to implement the controllers on a swarm of Crazyflie-drones to analyze how these systems are affected by physical limitations such as time delays, vibrations and calibration issues. Practical applications such as the formation and separation of swarms as well as the systems ability to handle different kinds of reference changes will be the primary focus.

1.2 Goal and purpose

- Develop a simulation model of a Crazyflie swarm.
  - Extend existing Crazyflie models to incorporate digital controllers and delays attributed to communication and execution.
  - Verification of the individual Crazyflie model as well as the whole swarm through comparison with real systems.
- Implement and test swarm control systems on the Crazyflie platform.
  - Implement multiple centralized controller variants.

  - Investigate possible limitations and issues connected to the platform.
  - Evaluate viability for flight in confined spaces.

1.3 Thesis outline

In Chapter 3 a model of the Crazyflie 2.1 quadcopter is derived through a combination of previous work and investigations into the current Crazyflie firmware implementation. The extended model is also validated through step analysis and frequency response testing. The Matlab implementations are available on Github [Green and Månsson, 2019b].

Chapter 4 contains an overview of our real world implementation as well as motivations for the chosen layout. The implementation uses the Crazyflie Python library and is available on Github [Green and Månsson, 2019a].

The swarm controllers implemented and tested are presented in Chapter 5. All conducted experiments are explained, and analysis as well as results from those experiments are presented. Additionally, the extended Crazyflie model is further validated through comparison with a real process in a swarm scenario.

Discussion in Chapter 6 is centered around the methodology chosen, performance and usability of the controllers as well as suggestions on further improvements and research.

1.4 Division of labour

- Attitude controller model
  - Sebastian
- Velocity controller model
  - Sebastian
  - Pontus
- Experiment design
  - Sebastian
  - Pontus
- Python framework
  - Sebastian
- Python controller implementation
  - Sebastian

  - Pontus
- Flocking controller
  - Sebastian
- Distance controller
  - Pontus
- Model validation
  - Pontus
  - Sebastian
- Data gathering
  - Sebastian
  - Pontus
- Results analysis
  - Sebastian
  - Pontus

2

Background

Unmanned aerial vehicles (UAVs/drones) have been the target of increasing research and development during the last few years due to decreasing manufacturing and development costs. This has led to multi-rotor vehicles in particular being used more for both professional and amateur applications. Most of the drones available for purchase at the time of writing utilize some form of driver assistance, usually in the form of onboard sensors and controllers to stabilize the quadcopter. This is to allow the operator to focus on positioning and the task at hand while the electronics control how much thrust should be generated by each motor to ensure the drone does not spiral out of control. This kind of assistance works well when the operator only has a very limited number of vehicles to control; however, as the number of vehicles increases, the task of manually operating them quickly becomes unmanageable. An example of a situation where controlling many drones would be beneficial is when flying in a confined space where external position references such as the global positioning system (GPS) are not available. This thesis aims to explore ideas for autonomous control of a swarm of drones in an enclosed space such as a cave or a building. By using multiple drones to continually scout ahead and set up new local positioning references, the swarm as a whole could be able to navigate unmapped areas efficiently and quickly by a single operator.

This chapter will start out by listing possible applications for a drone swarm, followed by previous research and a few different methods of controlling that swarm. Different previous swarm implementations using the Crazyflie platform will then be brought up and then a brief mention of some trajectory control methods.

2.1 Possible applications

The project aims to investigate possible applications of a drone swarm, and what advantages these will bring to the table. When it comes to open space, search and rescue would be a typical application since the drone swarm can cover a large area in very little time when time is important. A more specific scenario would be to use them to efficiently find and remove weeds on a field, improving crops and saving

time. Looking at enclosed spaces, the swarm could be used to map caves or buildings as mentioned above. These would generally be areas too dangerous for humans to enter, either for geographic exploration or for the above search and rescue case.

2.2 Previous research

The thesis will be based around the Crazyflie platform. Modelling and control of the individual drones will be based on a master's thesis done by [Greiff, 2017]. Greiff has done thorough models and tests of different control systems for individual drones. He has in his research improved the handling capabilities of the Crazyflie significantly and shown what levels of accuracy and precision are achievable with the platform currently. The models developed by Greiff have shown good accuracy in simulations and will be used as a starting point for simulating the multi-agent system. The dynamics of the Crazyflie 2.0 have also been studied and documented in [Förster, 2015] for use in simulations, however the dynamics used by Greiff were derived by [Luukkonen, 2011]. Numerous studies have been done to develop models and control systems for drone swarms, many of which study different theoretical systems to be able to control swarms in simulations and outside environments [Vásárhelyi et al., 2018],[Engebråten et al., 2018],[Oh et al., 2015]. This study aims to build upon these models and to implement them in hardware. Two major control approaches when dealing with multi-UAV formations or swarms outlined by [Sun, 2017],[Olfati-Saber, 2006] are displacement-based and distance-based control schemes. Each approach has distinct properties that make them suitable for different applications.

Displacement-based control

By assigning a distinct position to each individual agent in the system a formation can be achieved where each drone is essentially independent of the system as a whole. This approach requires less computational power in a centralized system. Because a single powerful unit can compute the desired positions of each drone, the onboard computations can be reduced to a minimum. Intercommunication between the drones is also not required provided sufficient safety margins are used. One potential drawback is that the system is limited by the measurement error and responsiveness of the individual agents. Limited positional accuracy and slow controller responsiveness may lead to unnecessarily large spacing in the swarm. Also, this method requires some kind of global reference frame such as GPS or an alternative for confined spaces. Another potential issue is that of disturbances or malfunctions in individual agents, as a collision with surrounding units is a risk.

Distance-based control

By allowing each unit to have a dynamic positioning based on proximity to other nearby agents, a more fluid and adapting structure can be achieved. This may also

reduce computational requirements of a central controller as it may be possible for individual drones to compute their desired positions. Another advantage is that a global or common reference frame among the drones is not necessary. However, these controllers are often non-linear, and due to the set rigid formation shapes it may for example be difficult to pass through a narrow passage.

Flocking behaviour

As is common in academia, nature can provide plenty of inspiration for efficient solutions. One such example is studied closely by [Reynolds, 1987] in an attempt to replicate the behaviour of flocks, herds and schools of fish for animation purposes. A visually accurate model of the decentralized control exhibited by flocks of birds was achieved through a combination of three basic desires of each agent in the system:

- Collision avoidance: avoid collisions with nearby flockmates.
- Velocity matching: attempt to match velocity with nearby flockmates.
- Flock centering: attempt to stay close to nearby flockmates.

Velocity in this context is described as a vector property consisting of heading and speed. Altering the influence based on the distance between two agents proved to be a very important for achieving lifelike behaviour, as localized knowledge allows for gradual differences in velocity throughout the flock. This is part of the reason flocks exhibit wave-like "shimmering" visuals. Originally Reynolds used the linear distance between agents as the base for error calculation, but changing the relationship to the inverse cube of the distance made for much more lifelike simulations. One potential difficulty when trying to implement such a behaviour is computational complexity. If each agent must be accounted for by each other one, a centralized controller implementation could easily result in $O(n^2)$ complexity without optimization [Olfati-Saber, 2006].

Hardware implementations

Swarms of Crazyflies have been successfully implemented in the past using motion capture system to determine positioning to a high degree of precision. With a positional error of less than 2 cm, accurate and precise trajectory planning of 49 drones was achieved [Preiss et al., 2017]. Bitcraze has also successfully flown multiple drones using time-difference-of-arrival (TDOA) for position measurements [Richardsson, 2017]. TDOA however suffers from significantly worse precision, with errors in the range of 10 cm. Accuracy increased significantly with an increased number of anchors. Positioning also seemed more stable within a space enclosed by anchors. Bitcraze's latest positioning system utilizes the lighthouse platform developed for SteamVR and used in the HTC Vive virtual reality systems

[Taffanel, 2019]. Lighthouse uses laser ranging technology to achieve decentralized sub-centimeter precision in three dimensions [Corporation, 2019].

Control structures designed in ideal simulation environments often work significantly worse in real implementations as many assumptions made in simulations do not hold up very well to reality. Communication irregularities or delays, unpredictable physical disturbances and the like can quickly make a seemingly stable system highly volatile when applied on real agents [Vásárhelyi et al., 2018]. For large formations, fast control and high relative velocities and accelerations increase the risk of a collision. Therefore a slower but more careful control is often desired to ensure system performance and integrity.

Crazyswarm

A framework for communicating with and setting paths for multiple Crazyflies already exists under the name Crazyswarm [Preiss et al., 2017]. This framework is built around ROS and allows for simple communication with many different Crazyflies. Currently the software is useful for connecting to and starting many drones, as well as setting up pre-determined patterns and routines based on MOCAP feedback. A trajectory generator is implemented together with a closed loop position controller [Chung et al., 2018].

2.3 Trajectory control

Trajectory control is a way of optimizing the route from point A to B while also taking for example obstacles into account, and is commonly used in robotics.

Field Potential Method (FPM)

FPM basically generates a potential field by making objects repulsive and the target point attractive. It is seen as easy to implement and needs little computational power, however it suffers from the non-reachable target problem where the agent may find a local minimum and get stuck. One case where this might happen is when the attraction from a target and the repulsion from an object are equal in a certain point. There are however examples where the repulsive function is modified slightly, taking the distance between the agent and target into consideration, with successful results [Mac et al., 2018].

Fuzzy Logic (FL)

Fuzzy logic makes use of a more human-like type of logic. It uses a number of rules, more specifically if-statements, with variables such as speed, height and slope as inputs. Before this, the variables are “fuzzified” through the so called membership functions, where for example the height is split into several intervals such as low, medium and high so that the if-statements can be defined. These if-statements can then once again generate a crisp output value to control e.g. the speed of the vehicle.

Ant Colony Optimization (ACO)

ACO takes on the principle of Swarm Intelligence (SI), where each agent makes its own decisions, without the use of centralized control. ACO is inspired by ants, which use the substance pheromone for orientation. Wherever an ant goes, it lays down a trail of pheromone. If this trail is found by another ant, there is a probability that it will follow the trail. This probability is increased for every ant which has travelled along it, i.e., by the amount of pheromone. This can be applied to a swarm of drones as well; by using positive feedback, paths done by several drones should get a "stronger" trail and therefore a higher probability of being chosen by an approaching drone choosing between two paths. Reasonably, a strong trail will correspond to a short path.

Particle Swarm Optimization (PSO)

The PSO method also uses SI, but with a different approach. The agents are said to be sharing a belief space, or search space, which may be modified by each agent according to three factors: exploratory, cognitive and social. Exploratory is about the knowledge of the environment, cognitive the state history for each agent while the social factor takes the state history of the agent's closest neighbours into account.

3

Modelling

To keep the number of drone crashes to a minimum, a simulation environment was set up to enable quick and consistent testing. Much previous work has been done on the modelling and system identification of quadcopters. To enable further improvements and adaptations, a modular simulation environment was designed where as much of the Crazyflie architecture and ecosystem as possible would be present. The desired entry point of the python swarm controller for simplified testing was determined to be the onboard velocity controller. Access to the velocity controller provides a good balance between the abstraction and ease of use provided by higher-level controllers, and the speed and customization available through lower level

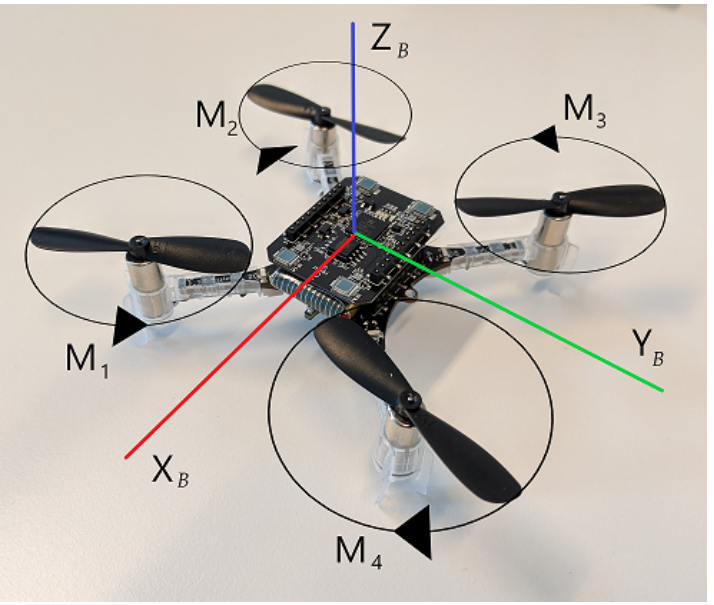

Figure 3.1 Crazyflie body coordinate system and rotor configuration

control. In order to mimic the actual physical plant as closely as possible, the whole chain of controllers between the desired velocity controller and the rigid body dynamics were included. An overview of the simulation layout chosen can be seen in Fig. 3.2.

This chapter will cover the models developed in order to simulate a single drone as well as a complete swarm. The rigid body and rotor dynamics are based on [Greiff, 2017], while the attitude and velocity controllers were developed during this thesis and based on the Crazyflie firmware [Bitcraze, 2019c].

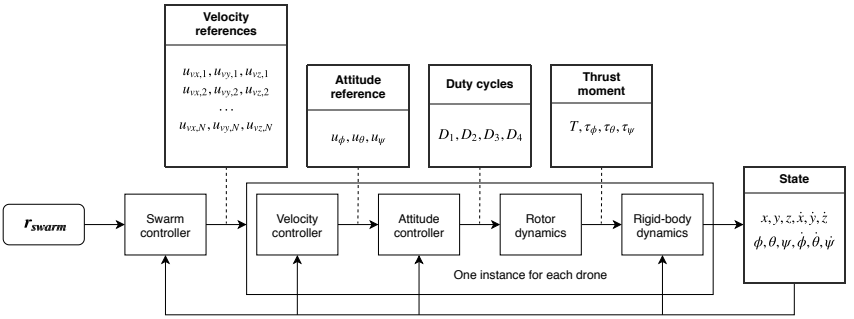

Figure 3.2 Simulation overview. Simulation blocks represented by rectangles, interfacing and signals represented by titled rectangles connected by dashed lines.

3.1 Rigid body dynamics

In order to simulate the behaviour of the Crazyflie quadcopter, a mathematical model of the drones' rigid-body behaviour is required. Such a model must include a description of the chosen coordinate system convention, as there are a number of possible configurations when operating in a three-dimensional space. Much previous work has been done on the rigid-body dynamics of quadcopters using Tait-Bryan angles. For the purposes of this thesis, the ZYX convention was employed in accordance with previous work [Greiff, 2017],[Luukkonen, 2011] to allow for the use of previously developed models. The three main coordinate systems will be denoted by sub-indexes $.G$, $.I$ and $.B$ for global, inertial and body respectively. Within the global system, the location of the drone center of mass will be denoted $\boldsymbol{p}$ [m]. The center of the inertial coordinate system is placed at the drone center of mass. We describe the body orientation by rotating the inertial coordinate system by the Tait-Bryan ZYX angles denoted $\boldsymbol{\eta}$ [rad]. The angular velocity of the body with respect to the global coordinate system is denoted $\boldsymbol{\omega}$ [rad/s], and the rotational

velocity of motor n is denoted Ω_n [rad/s].

$$\boldsymbol{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \qquad \boldsymbol{\eta} = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} \qquad \boldsymbol{\omega} = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \qquad \boldsymbol{\Omega} = \begin{bmatrix} \Omega_1 \\ \Omega_2 \\ \Omega_3 \\ \Omega_4 \end{bmatrix} \tag{3.1}$$

In accordance with the chosen Tait-Bryan representation, the complete rotation can be derived by applying three separate rotations in the specified ZYX order. Rotations about three orthogonal axes can be described by three separate rotational matrices where $c_i = \cos(i)$ and $s_i = \sin(i)$

$$\boldsymbol{R}(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\phi & -s_\phi \\ 0 & s_\phi & c_\phi \end{bmatrix}, \boldsymbol{R}(\theta) = \begin{bmatrix} c_\theta & 0 & s_\theta \\ 0 & 1 & 0 \\ -s_\theta & 0 & c_\theta \end{bmatrix}, \boldsymbol{R}(\psi) = \begin{bmatrix} c_\phi & s_\phi & 0 \\ -s_\phi & c_\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

resulting in the complete rotational matrix sub-indexed $_{GB}$ to indicate rotation from the global to the body coordinates.

$$\boldsymbol{R}_{GB} = \begin{bmatrix} c_\phi c_\psi & s_\psi c_\phi & -s_\theta \\ c_\psi s_\theta s_\phi - s_\psi c_\phi & s_\psi s_\theta s_\phi + c_\phi c_\psi & c_\theta s_\phi \\ c_\psi s_\theta c_\phi + s_\phi s_\psi & s_\psi s_\theta c_\phi - c_\psi s_\phi & c_\phi c_\theta \end{bmatrix} \tag{3.3}$$

To find the rotations from the body to the global frame $\boldsymbol{R}_{BG}$, we may take the inverse of $\boldsymbol{R}_{GB}$. As the inverse of a rotational matrix is also its transpose, the following equation gives us $\boldsymbol{R}_{BG}$.

$$\boldsymbol{R}_{BG} = \boldsymbol{R}_{GB}^{-1} = \boldsymbol{R}_{GB}^{T} \tag{3.4}$$

Rotor dynamics

Rotor transfer function The nonlinear rotor model of each individual rotor n as derived in [Greiff, 2017] is described by the single-input single-output system seen in Eqs. (3.5) and (3.6) where state vector $\boldsymbol{x}_n^r = \begin{bmatrix} \mu_n(t) & \dot{\mu}_n(t) & i_n(t) \end{bmatrix}^T$ containing rotor position $\mu_n(t)$, rotational velocity $\dot{\mu}_n(t) = \Omega_n(t)$ and current $i_n(t)$ is calculated from input voltage u_n^r.

$$\begin{aligned} \dot{\boldsymbol{x}}_n^r(t) &= \boldsymbol{A}_n^r \boldsymbol{x}_n^r(t) + \boldsymbol{B}_n^r u_n^r(t) \\ \boldsymbol{y}_n^r(t) &= \boldsymbol{C}_n^r \boldsymbol{x}_n^r(t) \end{aligned} \tag{3.5}$$

$$\boldsymbol{A}_n^r = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -b/J^{\pm} & K_t/J^{\pm} \\ 0 & -K_e/L & -R/L \end{bmatrix}, \quad \boldsymbol{B}_n^r = \begin{bmatrix} 0 \\ 0 \\ 1/L \end{bmatrix}, \quad \boldsymbol{C}_n^r = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}^T \tag{3.6}$$

K_t, K_e, b, $J^{\pm}$, R and L correspond to rotor parameters relating to motor current draw, resistance and inductance as well as friction, inertia and drag of the motor-rotor combination. Inertia $J^{\pm}$ depends on whether $|\Omega|$ is increasing or decreasing

according to Eqn. (3.7). Identification of the parameters has been done previously by [Greiff, 2017],[Luukkonen, 2011],[Förster, 2015]. The resulting parameters can be seen in Table 3.1.

$$J = \begin{cases} J^{+} & if \quad sign(\dot{\mu}(t) \times \ddot{\mu}(t)) > 0 \\ J^{-} & if \quad sign(\dot{\mu}(t) \times \ddot{\mu}(t)) < 0 \end{cases} \tag{3.7}$$

Table 3.1 Parameters of the rotor model

	Rotor parameter						
	K_t	K_e	b	J^+	J^-	R	L
Value	580	0.0011	0.10	0.031	0.13	2.3	0.12

Battery compensation The Crazyflie firmware implements a battery compensation feature in an effort to keep the thrust generated by the rotors similar throughout the full battery voltage range. The compensation algorithm also incorporates a mapping of the desired duty cycle to thrust. The mapping for battery voltage 3.7 V can be seen in Fig. 3.3. The compensated duty cycle ratio $D_{compensated}$ is calculated according to Eqn. (3.8) where D_{raw} denotes desired duty cycle ratio, U_{bat} denotes actual battery voltage and m_{cap} is the thrust mapping factor.

$$\begin{gathered} D_{compensated} = \frac{k_1(D_{raw}m_{cap})^2 + k_2(D_{raw}m_{cap})}{U_{bat}} \\ m_{cap} = 60.0, \quad k_1 = -0.0006239, \quad k_2 = 0.088 \end{gathered} \tag{3.8}$$

Thrust and moment The total thrust and moment about the center axes of an individual rotor n are denoted f_n and τ_n and are computed based on the assumption that the thrust and moment generated by each rotor is proportional to the rotor speed squared and acting parallel to the $\hat{\boldsymbol{z}}_B$ axis.

$$f_n = k_n \Omega_n^2, \quad \tau_n = b_n \Omega_n^2 \tag{3.9}$$

The total force acting on the drone body $\boldsymbol{T}_B$ is the sum of all rotor force contributions directed along the $\hat{\boldsymbol{z}}_B$ axis.

$$\boldsymbol{T}_B = T\hat{\boldsymbol{z}}_B = \hat{\boldsymbol{z}}_B \sum_{n=1}^{4} f_n \tag{3.10}$$

Equations of motion

Calculation of the angular momentum generated by the rotors is dependent on the configuration of the rotors in relation to the body coordinate system. Provided the cross-configuration depicted in Fig. 3.1 is used, the angular momentum in the body

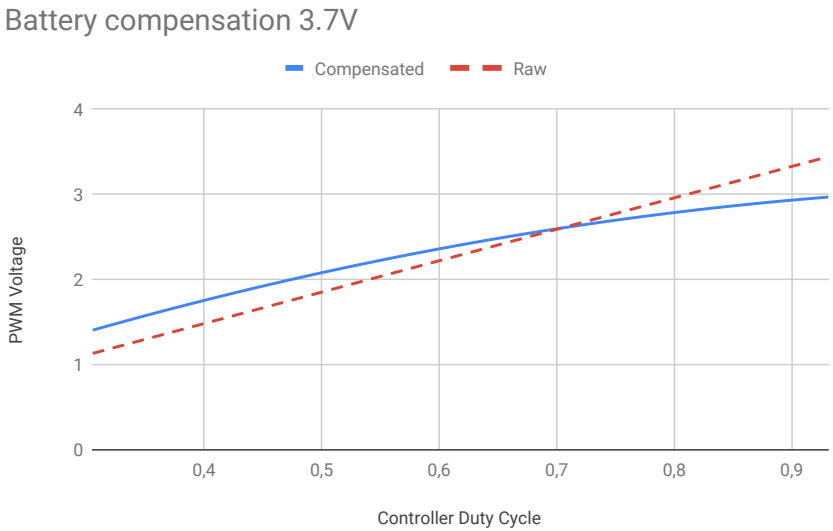

Figure 3.3 Battery compensation at 3.7V

coordinate system can be computed according to Eqn. (3.11) where each motor is mounted l m from the drone center of mass.

$$\boldsymbol{\tau}_B = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} \frac{l}{\sqrt{2}}(-f_1 - f_2 + f_3 + f_4) \\ \frac{l}{\sqrt{2}}(-f_1 + f_2 + f_3 - f_4) \\ \sum_{n=1}^{4} \tau_n \end{bmatrix} \tag{3.11}$$

The body inertial and drag matrices are defined as

$$\boldsymbol{I}_B = \begin{bmatrix} I_{11} & 0 & 0 \\ 0 & I_{22} & 0 \\ 0 & 0 & I_{33} \end{bmatrix}, \quad \boldsymbol{D}_B = \begin{bmatrix} D_{11} & 0 & 0 \\ 0 & D_{22} & 0 \\ 0 & 0 & D_{33} \end{bmatrix} \tag{3.12}$$

where the air resistance is assumed to be a reactionary force proportional to the velocity $\dot{\boldsymbol{p}}_B$. The above definitions can be summarized to the complete equations of motion in the Tait-Bryan angle representation given by Eqn. (3.13) where the full derivations of $\boldsymbol{C}$ and $\boldsymbol{J}$ can be found in [Greiff, 2017].

$$\begin{aligned} \ddot{\boldsymbol{p}} &= -g\hat{\boldsymbol{z}}_G + \tfrac{1}{m}\boldsymbol{R}_{BG}\boldsymbol{T}_B - \tfrac{1}{m}\boldsymbol{D}_B\dot{\boldsymbol{p}} \\ \ddot{\boldsymbol{\eta}} &= \boldsymbol{J}^{-1}(\boldsymbol{\eta})(\boldsymbol{\tau}_B - \boldsymbol{C}(\boldsymbol{\eta}, \dot{\boldsymbol{\eta}})\dot{\boldsymbol{\eta}}) \end{aligned} \tag{3.13}$$

3.2 Attitude controller

In an effort to focus on the dynamics of swarming rather than that of any single drone, the onboard orientation controller is used. The firmware contains three cascading controllers each regulating the attitude and angular velocity of roll, pitch and yaw respectively. An overview of the control structure, including names of the components for future reference, can be seen in Fig. 3.4. Control parameters of all the controllers can be seen in Table 3.2. The angle controllers run at 250 Hz and receive feedback from the onboard IMU. The angle controller output is then used as reference for the faster rate controllers operating at 500 Hz. This reference is compared to the angular velocity of the drone with respect to the body frame as measured by the gyroscope. All six individual PID controllers are implementations of the discrete parallel controller seen in Eqn. (3.14).

$$\begin{aligned} u(k) &= K_p \times e(k) + K_i \times I(k) + K_d \times D(k) \\ I(k) &= I(k-1) + e(k) \times h, \quad I \in [-I_{sat}, I_{sat}] \\ D(k) &= \frac{e(k)-e(k-1)}{h} \end{aligned} \tag{3.14}$$

The outputs of the rate controllers are used to set the 16 bit unsigned integers corresponding to the duty cycle of each motor, similarly to how differential drive works in a two-wheeled robot. The base thrust u_{thrust} is either set manually, or computed by the velocity controller described in Sec. 3.3. Conversion from controller output to motor voltage is done by dividing the controller output by the maximum value possible for unsigned 16 bit integers, 65535. In the cross configuration, the duty cycle D_n of each motor is calculated from the controller output $\boldsymbol{u}_{ob}$ according to Eqn. (3.15) before being remapped according to the thrust compensation mentioned in Sec. 3.1.

$$\begin{bmatrix} D_1 \\ D_2 \\ D_3 \\ D_4 \end{bmatrix} = \frac{1}{65535} \times \left(\begin{bmatrix} u_{thrust} \\ u_{thrust} \\ u_{thrust} \\ u_{thrust} \end{bmatrix} + \begin{bmatrix} -0.5 & 0.5 & 1 \\ -0.5 & -0.5 & -1 \\ 0.5 & -0.5 & 1 \\ 0.5 & 0.5 & -1 \end{bmatrix} \times \boldsymbol{u}_{ob} \right), \boldsymbol{u}_{ob} = \begin{bmatrix} u_\phi \\ u_\theta \\ u_\psi \end{bmatrix} \tag{3.15}$$

Implementing the controllers in Simulink proved difficult as there were slight differences in behaviour between the real and simulated system. Using the controller parameters of the real firmware (as seen in Table 3.2) resulted in an unstable system showing both oscillations and diverging amplitude. The simulation parameters were instead tuned to comply with the dynamics of the actual system, resulting in the attitude control parameters shown in Table 3.3. A comparison between the roll and yaw responses of the real and simulated systems using the re-tuned parameters can be seen in Figs. 3.5 and 3.6 respectively.

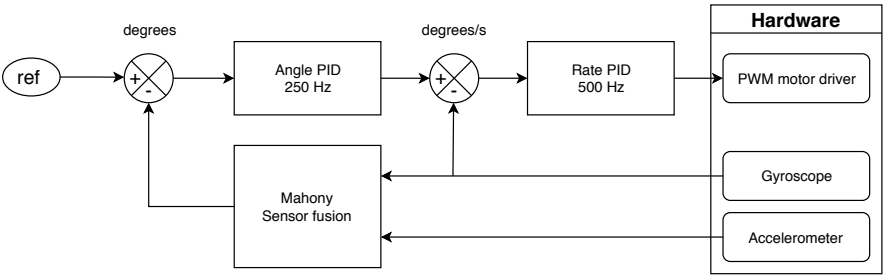

Figure 3.4 Onboard attitude controller layout

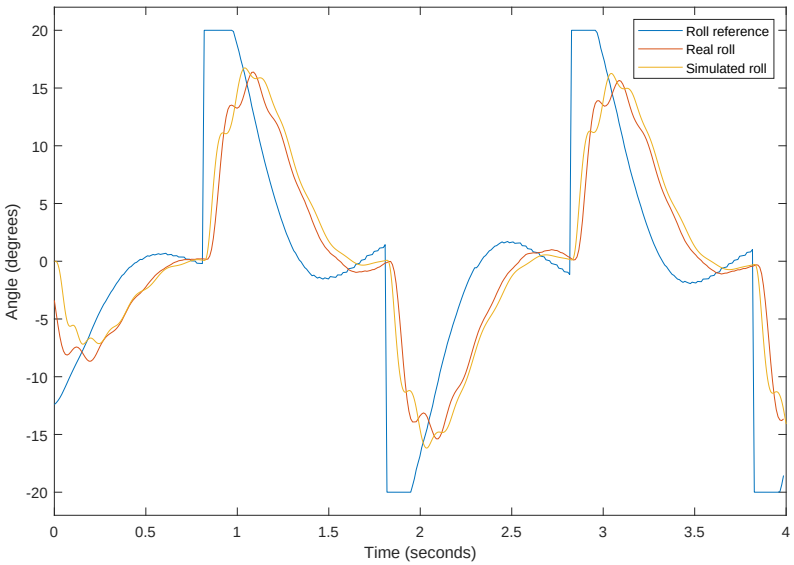

Figure 3.5 Real and simulated roll response

Table 3.2 Parameters of the onboard attitude controllers

	Angle				Rate			
	K_p	K_i	K_d	I_{sat}	K_p	K_i	K_d	I_{sat}
ϕ	6	3	0	20	250	500	2.5	33.3
θ	6	3	0	20	250	500	2.5	33.3
ψ	6	1	0.35	360	120	16.7	0	166.7

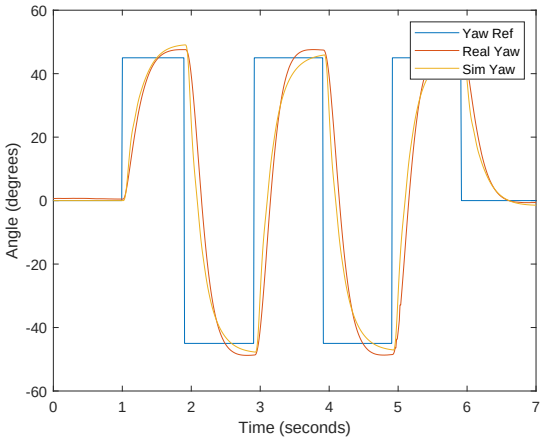

Figure 3.6 Real and simulated yaw response

Table 3.3 Parameters of the simulated onboard attitude controllers

	Angle				Rate			
	K_p	K_i	K_d	I_{sat}	K_p	K_i	K_d	I_{sat}
ϕ	6	3	1	20	250	500	5	33.3
θ	6	3	1	20	250	500	5	33.3
ψ	6	1	0.35	360	60	8.35	2	166.7

3.3 Velocity controller

The Crazyflie firmware contains three PI controllers for regulating the drone velocity in the global coordinate frame. Two of the controllers together map the velocity error in $\hat{\boldsymbol{x}}_G$ and $\hat{\boldsymbol{y}}_G$ to desired roll and pitch angles $u_{\phi,raw}$ and $u_{\theta,raw}$ which are then remapped and fed as references to the attitude controllers described in Sec. 3.2. As an increased angle in either roll or pitch will result in more thrust in the specified direction, the system can be seen as a set of non-linear single integrators provided the cascading attitude controllers are capable of following the setpoints accurately. Output references u_ϕ and u_θ depend on ψ and $u_{\phi,raw}$ and $u_{\theta,raw}$ according to Eqn. (3.16) as the desired angles need to be adjusted depending on the current rotation of the drone as $\hat{\boldsymbol{x}}_B$ and $\hat{\boldsymbol{y}}_B$ are not necessarily aligned with $\hat{\boldsymbol{x}}_G$ and $\hat{\boldsymbol{y}}_G$.

$$\begin{bmatrix} u_\phi \\ u_\theta \end{bmatrix} = \begin{bmatrix} -\cos(\psi) & -\sin(\psi) \\ \sin(\psi) & -\cos(\psi) \end{bmatrix} \times \begin{bmatrix} u_{\phi,raw} \\ u_{\theta,raw} \end{bmatrix} \tag{3.16}$$

The parameters used in the firmware and simulation can be seen in Tables 3.4 and 3.5 respectively. The simulation parameters have been slightly modified to better match the real drone behaviour. The $\hat{\boldsymbol{z}}_G$ velocity controller also encompasses a feed-forward term to counteract the constant gravitational force acting on the drone. This term consists of a base thrust of 36000 as well as a static gain of 1000 acting on the output of the controller as seen in Eqn. (3.17).

$$u_{thrust} = 36000 + 1000 \times u_z \tag{3.17}$$

The thrust calculated through this formula is fed as the baseline duty cycle to the attitude control seen in Eqn. (3.15). To keep the drone in a stable and well-behaved range, a number of saturations are implemented in the velocity controller. As the output of the controller is fed directly to the attitude controllers, the angle references have to be limited to make sure the drone has enough power so as to not lose altitude control. In the case where roll and pitch are both 0, all of the thrust generated by the motors will be directed upwards along $\hat{\boldsymbol{z}}_G$. However, as the roll or pitch angles tend toward ± 90 degrees, the thrust along $\hat{\boldsymbol{z}}_G$ decreases until reaching 0 as either ϕ or θ reaches ± 90 degrees. To counteract this, the angle reference output is limited from -20 to 20 degrees in roll and pitch respectively. This is visible in Fig. 3.5 where the blue roll reference plot does not go above 20 degrees or below -20 degrees even though the velocity reference step used in that run looks to go beyond the saturation. Another important thing to note here is that because the rotations are decoupled, it is possible for the attitude of the drone to exceed the 20 absolute degree limit about certain axes. As a result, larger accelerations are possible along directions other than the body-centered $\hat{\boldsymbol{x}}_B$ and $\hat{\boldsymbol{y}}_B$ axes at the cost of limited elevation control. As ϕ and θ tend towards their saturation points, the angle of the diagonal exceeds the saturation. Saturation in this way allows for easier approximation and tuning provided the axle decoupling persists through the control system. It does however limit the performance of the quadcopter somewhat, as the potential acceleration is not the same for all directions in the global $\boldsymbol{xy}_G$ plane. An alternative implementation where the saturation angles of ϕ and θ are determined dynamically based on the direction of the desired acceleration would allow for higher accelerations along $\hat{\boldsymbol{x}}_G$ and $\hat{\boldsymbol{y}}_G$, however such an implementation could prove more difficult to design and tune as conversion between Cartesian and polar coordinates might be necessary. Trigonometric calculations such as sine and cosine require significantly more computational power than addition, subtraction, multiplication and even division. As the velocity controller runs on the drone microcontrollers, performance is of high importance.

Frequency response

In order to verify the translational behaviour of the model, including the velocity controller, attitude controller and the drone dynamics, a frequency response test was run. The drone was fed with a sine wave velocity reference with constant am-

Table 3.4 Parameters of the onboard velocity controllers

	Velocity		
	K_p	K_i	K_d
$\dot{x}$	25	1	0
$\dot{y}$	25	1	0
$\dot{z}$	25	15	0

Table 3.5 Parameters of the simulated velocity controllers

	Velocity		
	K_p	K_i	K_d
$\dot{x}$	25	1	0
$\dot{y}$	25	1	0
$\dot{z}$	22	15	0

plitude, 1 m/s along $\hat{\boldsymbol{y}}_G$ and 0.5 m/s along $\hat{\boldsymbol{z}}_G$, and a frequency from 0.1 Hz to about 3 Hz, depending on the direction. Only the $\hat{\boldsymbol{y}}_G$ and $\hat{\boldsymbol{z}}_G$ axes were analyzed since the drone behaves similarly in the $\hat{\boldsymbol{x}}_G$ and $\hat{\boldsymbol{y}}_G$ axes. These references were then run in the model. A selection of the results can be seen in Figs. 3.7 and 3.8, with reference sine waves of 1 Hz along $\hat{\boldsymbol{y}}_G$ and 2 Hz along $\hat{\boldsymbol{z}}_G$, respectively. In the $\hat{\boldsymbol{y}}_G$ direction, the velocities match each other very nicely, with the simulation being just slightly quicker. However, looking at $\hat{\boldsymbol{z}}_G$ the simulation is noticeably quicker while also having a lower amplification. The frequency results were then used for a Bode diagram, see Figs. 3.9 and 3.10, and by looking at these one can get a bigger picture. The phase shift in $\hat{\boldsymbol{y}}_G$ is very similar but the amplification differs. At lower frequencies the simulation has a higher amplification which shifts at around 1.1 Hz, but overall the model is accurate. The same cannot be said for the response in $\hat{\boldsymbol{z}}_G$, where the amplification is off by a big margin between 1.1 and 2 Hz. Looking at the phase shift, the simulation is much slower after 1 Hz.

Step response

Following the frequency response test, step response was tested by giving the drone a velocity reference of 0.5 meters per second for a select time in each direction. As with the frequency response, the tests were only run in the $\hat{\boldsymbol{y}}_G$ and $\hat{\boldsymbol{z}}_G$ directions. Results can be seen in Fig. 3.11 and 3.12. For $\hat{\boldsymbol{y}}_G$, the results are quite similar in both speed and behaviour, although the model over- and undershoots more. Looking at $\hat{\boldsymbol{z}}_G$, there seems to be a substantial inaccuracy in the model when gravity is taken into account. Going upwards, the model is slower and has a different overshooting behaviour, followed by a failure at getting back to the reference. In negative $\hat{\boldsymbol{z}}_G$, the model resembles the real drone quite well putting aside the fact that it is too quick.

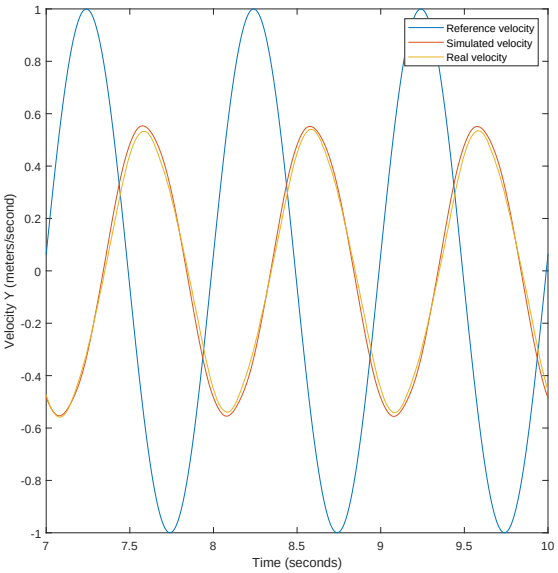

Figure 3.7 Velocity frequency response in $\hat{\boldsymbol{y}}_G$ of physical and simulated plant with 1 Hz sine wave

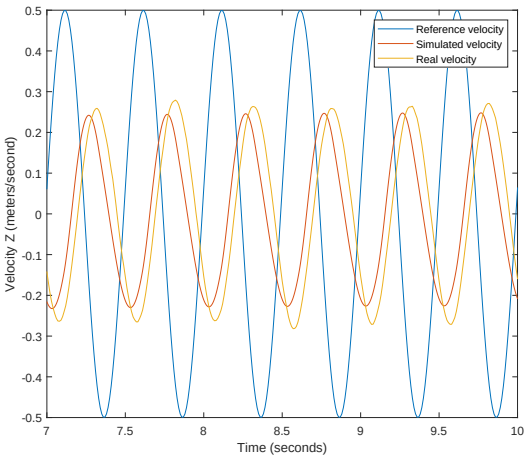

Figure 3.8 Velocity frequency response in $\hat{\boldsymbol{z}}_G$ of physical and simulated plant with 2 Hz sine wave

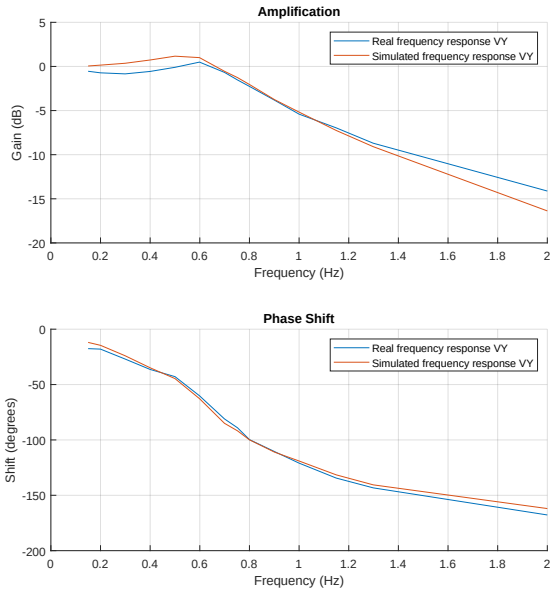

Figure 3.9 Bode diagram of physical and simulated plant in $\hat{\boldsymbol{y}}_G$

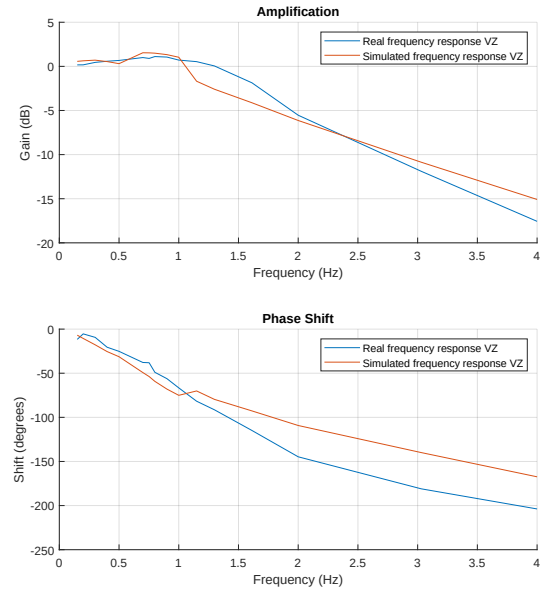

Figure 3.10 Bode diagram of physical and simulated plant in $\hat{\boldsymbol{z}}_G$

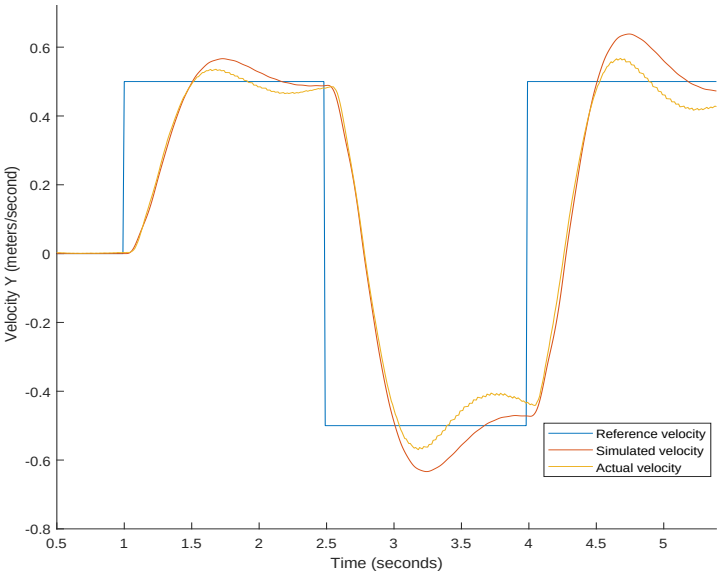

Figure 3.11 Velocity step response in $\hat{\boldsymbol{y}}_G$ of physical and simulated plant

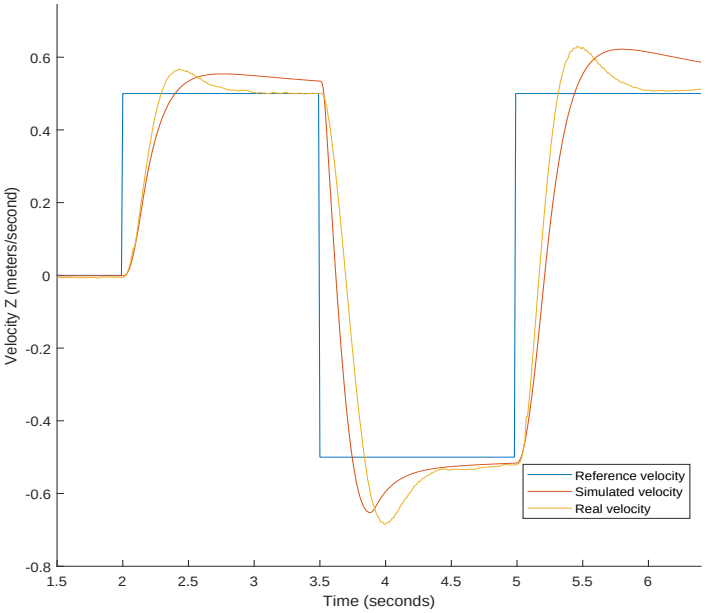

Figure 3.12 Velocity step response in $\hat{\boldsymbol{z}}_G$ of physical and simulated plant

3.4 Swarm modelling

Transitioning from a single drone to a swarm of N agents was done by simply multiplying the model structure described above N times. The resulting swarm model takes the desired reference angles of all agents individually and outputs the complete swarm state consisting of $\boldsymbol{p}_n$, $\dot{\boldsymbol{p}}_n$, $\boldsymbol{\eta}_n$ and $\dot{\boldsymbol{\eta}}_n$ for each drone n. The number of active agents is defined by the dimensions of the initial state matrix. Validation of the swarm model can be seen in Sec. 5.6.

4

Real world implementation

4.1 Bill of materials

The complete bill of materials for the swarm is summarized in Table 4.1. Five drones were found to be an adequate number to form a proper swarm while still being easy to control. Extra batteries were bought to be able to fly for longer than the seven minutes that one battery lasts, together with some spare parts due to the drones' tendency to crash during testing.

Table 4.1 Bill of materials for the swarm

Part	Quantity	Unit price (SEK excl VAT)	Sum (SEK excl VAT)
Crazyflie 2.1	5	1850	9250
Crazyradio PA	1	280	280
Battery + charger	10	80	800
Spare parts	1	170	170
Lighthouse deck	5	750	3750
HTC Vive	1	5200	5200
		Total	**19450**

4.2 Lighthouse positioning

Lighthouse deck and Vive base stations

The Lighthouse deck is an expansion deck for mounting on top of the Crazyflie. Using the laser and LED light emitted by the HTC Vive base stations and the four receivers on the deck, the Crazyflie can by itself estimate its position in the room with millimeter precision.

The base stations consist of a matrix of LED's and two rotating laser emitters, one horizontal and one vertical. The main concept is that the LED's flash at a frequency of 60 Hz, and each time they are followed by an alternating laser sweep, i.e. each laser sweeps every other flash. The point of the diodes is to start a counter on the receiver until it detects the laser sweep. Using this time in combination with the

base station positions and the relative positioning of the receivers, the exact position of the drone can be calculated.

Note that at the time of writing the deck is still in early access meaning there are some limitations. For example, orientation of the drone cannot be computed and due to the receivers being horizontal the drones have to be kept at least 40 cm below the base stations. Two base stations are required but later on a single one will be enough.

Setup

The Lighthouse positioning requires a HTC Vive-set with two base stations V1, as well as SteamVR, openVR and Python3. It is possible to set up the system without the head mounted display (HMD), although that is a more complex installation.

Using the HMD and with the two base stations mounted, the standard installation is run through SteamVR in order to define the room and origin. The origin ends up wherever the HMD is positioned when the area is calibrated, with positive $\hat{\boldsymbol{x}}_G$ in the direction the HMD is facing. In this case, the HMD had to be lifted up about 2 cm in the front to get a somewhat vertical $\hat{\boldsymbol{z}}_G$-axis which otherwise was angled. When this is done the HMD is no longer needed, except when repositioning the base stations.

Following the calibration, a script from the Crazyflie firmware is to be run which extracts the position of the base stations. This position is then pasted into the lighthouse deck driver and flashed onto the Crazyflie. For a more detailed setup instruction, see [Bitcraze, 2019e].

4.3 Crazyflie firmware

Communication

Communication with the individual Crazyflies happens through a custom radio chip called the Crazyradio which is connected to the client computer through USB [Bitcraze, 2019f]. The protocol uses 32 byte packets consisting of one byte of header information and up to 31 bytes of data [Bitcraze, 2019b]. The size limit of these packages limit how much information can be sent in each instance. As all positional information is calculated locally on the agent firmware, the quality and consistency of the radio link impacts the quality of the sensor information utilized by the centralized swarm controller. Testing of the radio communication quality was done by monitoring timestamps of when the data was received by the python client and measuring time intervals between received packets. A desired sample rate of 100Hz was used for initial communication testing and the resulting distribution of interval time can be seen in Fig. 4.1. The results showed that the sampling time is relatively consistent about the desired 10ms as 79.7% of samples were received within 8 and 12ms of the previous update.

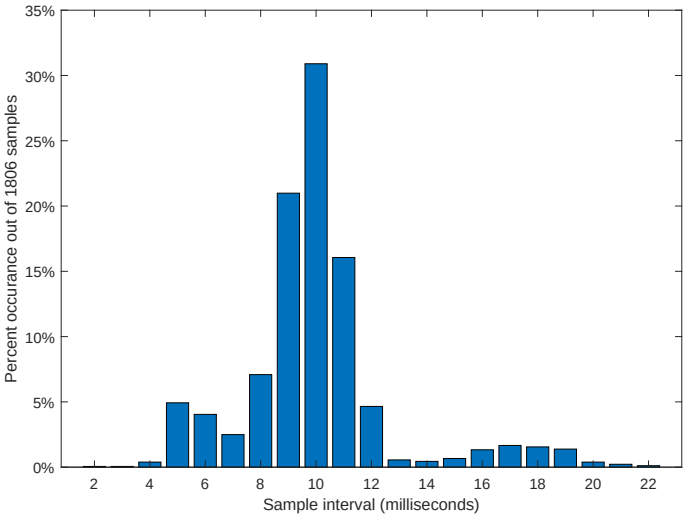

Figure 4.1 Radio consistency testing, distribution of time between packets received for single agent with a target period of 10ms

When using all five quadcopters, discrepancies were noticed between the timestamps sent by the Crazyflies as part of the communication protocol and timestamps recorded on the swarm client system time. The timestamps seemed to drift further and further apart during flights, resulting in polling updates arriving up to two whole seconds late after ten seconds of flight with five agents at 100Hz polling rate. This was attributed to rate limitations in the Crazyradio, and by limiting the polling rate to 20Hz the problem was eliminated and package delays and drops were minimized. This result means that the throughput of the radio is a major bottleneck in the physical implementation, and the performance could possibly be improved through the use of more radios or more efficient multi-agent communication and broadcasting techniques rather than the single-agent structure used in the standard Crazyflie Python library. The library does support the use of multiple radios for faster multi-agent communication, but the polling rate of 20Hz was deemed sufficient for the purposes of this thesis.

Logging

Information about the quadcopter state is sent using the logging framework implemented in the Crazyflie firmware as well as the Crazyflie Python library. The framework allows for custom log packages to be constructed and transmitted au-

tonomously at regular intervals. Each quadcopter can have multiple different log packets configured, and all communication through the logging framework occurs between the individual agent and the centralized commander radio connected to a pc. Whenever a connection is established between the Crazyradio and a Crazyflie, the variables available for logging are transmitted to create a synchronized table of contents. The creation of packets is then done by specifying which variables from the table of contents should be transmitted, and at what interval. Due to limitations in bandwidth and artificial limitations of the Crazyradio outlined in Sec. 4.3, a single package was constructed containing position $\boldsymbol{p} = [x, y, z]$ and velocity data $\dot{\boldsymbol{p}} = [\dot{x}, \dot{y}, \dot{z}]$ from the onboard Kalman estimator. Each variable is a 4 byte float, resulting in a total package payload of 24 bytes allowing for transmission of all relevant state information in a single package.

4.4 Swarm Client Software

Python library

All communication with the Crazyflie quadcopters occurs through the Crazyflie Python library available as open source through the Bitcraze GitHub account [Bitcraze, 2019d]. The library contains all structures necessary to establish a connection with individual Crazyflies through the Crazyradio as well as the *Commander* and *MotionCommander* classes used for sending setpoints to the onboard attitude and velocity controllers outlined in Secs. 3.2 and 3.3. The *Commander* class contains three functions which allow for the sending of attitude references, velocity references or position references through the Crazyradio using CRTP. The attitude and position setpoint functions were used for testing and verification of the model as well as more precise control of individual agents for testing the behaviour of the swarm during maneuvers that require individual drone control. The Python library also contains a *Swarm* class with basic functionality for connecting to and interacting with multiple drones at once. This class is useful for sending commands to all agents of the swarm in parallel, and much of the software implementation builds on the basic functions of this class.

Python implementation

The software developed for testing and flying a swarm of drones is designed around a main thread running a sequence of operations supported by a controller thread and a logging thread for managing data gathering and controller computations. Communication links and all swarm interactions are handled by an extended version of the *Swarm* class featured in the Crazyflie Python library called *AsyncSwarm*. An *AsyncSwarm* instance keeps track of all active agents and reacts to callbacks from the Crazyflie library whenever a new log record has been received by the Crazyradio. *AsyncSwarm* also provides functions for starting and stopping the swarm as

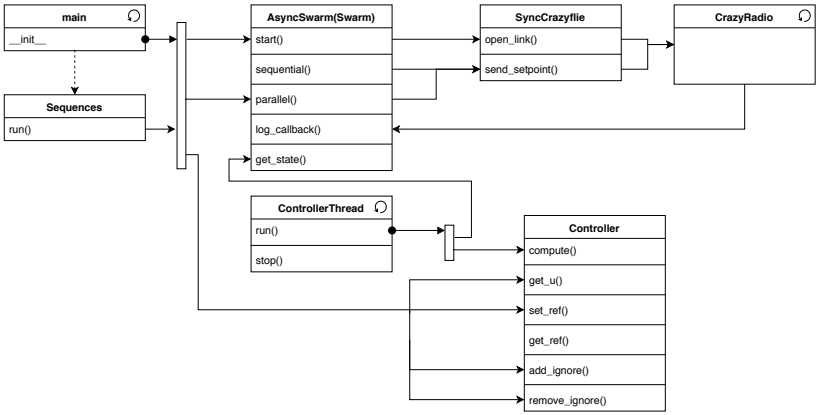

Figure 4.2 Overview of Python software implementation

well as adding or removing drones. An overview of the software structure can be seen in Fig. 4.2.

AsyncSwarm Main instance for establishing and maintaining connection with the Crazyflies. It contains a list of all active drones as well as the last known state of all drones. By extending the Crazyflie python library class *Swarm* this class can use the functions *Swarm.parallel* and *Swarm.sequential* which executes a specified function on all Crazyflies in the swarm either in parallel or in sequence before joining and continuing execution. The structure of the sequential and parallel functions requires that the function passed as the argument to be called must take a *SyncCrazyflie* instance as the first argument and any other optional parameters contained in a dictionary keyed by the Crazyflie URI. Because of this structure, a number of static support functions are available in the class *CFUtil* described more in detail in Sec. 4.4. *AsyncSwarm* also contains a function *follow_controller* that takes a valid controller as parameter and sends the current output of the controller to all Crazyflies in parallel.

Initial connection to the Crazyflies is done by attempting to open a link to all drones in sequence. While the chance of successfully connecting to a single drone is rather high, there exists a bug that causes the Crazyflie to continuously send log packets even after the connection has been terminated. One possible fix is to restart the drone, but another more user friendly fix when dealing with multiple drones is to send a stop_logging command to the drone to ensure no old log configurations are still active. This is done by calling the *CFUtil.ext_open_link_cf* function. This function will try to connect to the agent, but terminate and send a hard log reset package to the agent if no connection was established within a timeout period

before trying to connect again again. This generates an error message as the connection sequence is interrupted unexpectedly, but the connection will succeed on the subsequent connection attempt. Following a successful connection, the program will wait for the logging table of contents to be transferred as that information will otherwise clog up the radio during the first few seconds of flight.

Controller The swarm controllers are implemented as individual classes in the *Controllers* file. Each controller must supply the following functions in order to function with the *Sequences* class described in Sec. 4.4:

- *compute(dict: state)*
- *get_u()*
- *get_u_list()*
- *set_ref(iterable: ref)*
- *reset()*
- *add_ignore(iterable: uris)*
- *remove_ignore(iterable: uris)*

compute is the main function used to calculate the new velocity setpoints to send to all the agents based on the state specified in the parameter. All reference points are recalculated at the same time, based on the period specified in the *ControllerThread* instance. For the purposes of testing the interactions involved with new drones entering and leaving the swarm, an *_ignore_list* is implemented that allows the connection management and state tracking of all drones to remain in the *AsyncSwarm* instance while at the same time enabling the controller to ignore sending references to a selection of the connected drones. The controller is called periodically by the *ControllerThread* thread described in Sec. 4.4.

LogManager and Log In-flight data collection is achieved by periodically polling all variables of interest. All collected data can be saved to a Matlab compatible .mat file at any time during execution. An encompassing *LogManager* is created at program startup, all parameters to be polled can be added by attaching a caller to the *LogManager* instance with the method *add_caller*. *LogManager.add_caller* takes a name, a period, a start flag, and a function to be called at regular intervals defined by the period. If the start flag is enabled, the *Log* will start a periodic thread that calls the function and appends the returned data to a list. This structure allows for polling at regular intervals, however an additional call to *Log.push_data* must be made if the changed value is to be registered exactly at execution time rather than at the next logging interval. If the start flag is not enabled, the *Log* instance does not append any data automatically, but does still allow for manual data collection through *Log.push_data*. Timestamps of all data entries as well as start times are recorded to allow for synchronization of data after execution.

CFUtil CFUtil contains a number of help functions to simplify control through the *Swarm.parallel* and *Swarm.sequential* functions, as well as some additional functions related to managing state and setup of the *SyncCrazyflie* instances. Each function related to sending commands to drones through *CFUtil* must take a *SyncCrazyflie* instance as the first parameter, followed by any other parameters contained in the keyed argument dictionary specified in Sec. 4.4. *CFUtil* also contains some default static attributes such as the URIs of all available Crazyflies and a few complete argument dictionaries *POS_HOVER* (pictured in Fig. 5.1) and *POS_LAND* containing formations used for initializing the swarm and making safe take-offs and landings.

Sequences For testing the stability and performance of the swarm, a number of predetermined sequences explained in detail in Sec. 5.1 were predefined and stored in the *Sequences* object. *Sequences* has a single *run* function that takes the *AsyncSwarm* and *Controller* instances as well as a static key chosen from a selection available in the *Sequences* and performs the chosen sequence on the swarm. Sequences can be chained together by simply calling *run* multiple times, and provides a workable base for repeating tests and measurements on the swarm. The *Sequences* class can be extended with more predefined sequences, and will on initiation check to make sure no key collisions are present.

ControllerThread To ensure controller output is continuously updated regardless of flight state or communication errors, the *ControllerThread* instance calls the function passed as parameter *controller_func* at regular intervals specified by the *period_ms* argument. What function to be called is decided on initiation, however the thread will retrieve the current state of the swarm through *AsyncSwarm.get_state* and pass to the function. The intended use is the *compute* function of whichever controller is currently active.

Sampling

The sampling structure used was chosen with the intention of allowing for much larger swarm formations without changing the characteristics of the controller. Logger initiation timings and instability in the communication makes callback-based controller updates cumbersome and unreliable, whereby the decision was made to decouple the sampling and controller cycles. The resulting system updates the swarm state continuously as new log packets are received by the radio, and runs the controller computation in a separate thread at regular intervals. As mentioned in Sec. 4.3 the polling and update rate was chosen to be 20 Hz. This means each independent agent sends a packet containing its current state every 50 ms, and this information is retained until that agent sends another state update. The controller also executes every 50 ms, and uses the latest available state information of all active drones. In the ideal case, this corresponds to the controller updating once for every complete swarm state update, however with a maximum delay of 50 ms not including transport delays during transmission of the control signal to the agents.

5

Swarm control

Moving on to the control of multiple drones, two different swarm algorithms were implemented to be able to choose the best performing one according to the earlier specified requirements. More specifically, the distance and flocking controllers were chosen since they seemed simple to implement, appeared relevant to the project as well as only requiring relative positioning. The upcoming sections will cover these as well as methods of testing and comparing the algorithms. Positions and references will be denoted by $\boldsymbol{p_i}$ and $\boldsymbol{r_i}$ respectively where i corresponds to dn for agent n or *swarm* for swarm center positions.

5.1 Method

For practical applications, certain maneuvers are critical for usability and stability of the swarm. Micro-management of individual drones quickly becomes unfeasible as the number of involved agents increases. Thus, the swarm controller must be able to handle several commonly occurring scenarios without requiring external input from one or more human operators. Five maneuvers were established to provide practical scenarios to analyse the performance of the swarm controllers: hover, step, ramp, merge and leave.

Hover

Swarm behaviour in a steady state is examined by looking at stability and propagation of low-frequency disturbances and measurement noise throughout a prolonged period with a persistent swarm center reference. Interesting factors to consider are long term stability as well as the effects of increasing or decreasing the number of active agents, whereby three test cases were constructed where the number of active drones ranges from 1 to 3 to 5. Each test consisted of a swarm being initiated from a predetermined formation defined by individual drone positions $\boldsymbol{p_{0,di}}$ to follow a common swarm reference $\boldsymbol{r_{swarm}}$ according to Table 5.1.

Table 5.1 Hover case

	Position		
	x_G	y_G	z_G
$\boldsymbol{p_{0,d1}}$	0	0	1
$\boldsymbol{p_{0,d2}}$	0.5	0	1
$\boldsymbol{p_{0,d3}}$	0	0.5	1
$\boldsymbol{p_{0,d4}}$	−0.5	0	1
$\boldsymbol{p_{0,d5}}$	0	−0.5	1
$\boldsymbol{r_{swarm}}$	0	0	1

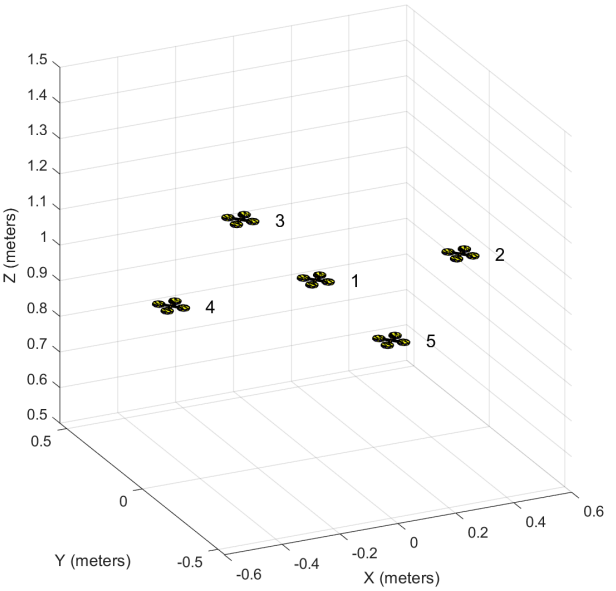

Figure 5.1 Positions of drone 1 through 5 before swarm controller is engaged

Step

A step is achieved by changing the position reference $\boldsymbol{r_{0,swarm}}$ of the swarm to another point $\boldsymbol{r_{1,swarm}}$ in the three-dimensional space. Two sets each consisting of three test cases were analyzed to determine step stability and responsiveness of the swarm controller. Step set one is contrived of 1 meter steps along positive and negative $\hat{\boldsymbol{z}}_{\boldsymbol{G}}$ as well as positive $\hat{\boldsymbol{y}}_{\boldsymbol{G}}$, whereas step set two consists of the same steps performed by a swarm of 3 agents. Due to the symmetry of $\hat{\boldsymbol{y}}_{\boldsymbol{G}}$, $-\hat{\boldsymbol{y}}_{\boldsymbol{G}}$, $\hat{\boldsymbol{x}}_{\boldsymbol{G}}$ and $-\hat{\boldsymbol{x}}_{\boldsymbol{G}}$, only one of those cases was analyzed. Analysis along positive and negative $\hat{\boldsymbol{z}}_{\boldsymbol{G}}$ may however

prove useful as the constant outside force resulting from gravity may change impact the response. All test case positions can be seen in Table 5.2.

Table 5.2 Step & ramp case

	$\hat{z}_G$			$-\hat{z}_G$			$\hat{y}_G$		
	x_G	y_G	z_G	x_G	y_G	z_G	x_G	y_G	z_G
$\boldsymbol{r_{0,swarm}}$	0	0	0.5	0	0	1.5	0	0	1
$\boldsymbol{r_{1,swarm}}$	0	0	1.5	0	0	0.5	0	1	1

Ramp

Similarly to Sec. 5.1, ramp responses were analyzed by linearly changing the swarm center reference $\boldsymbol{r_{swarm}}$ of a single agent as well as a swarm of three agents from $\boldsymbol{r_{0,swarm}}$ to $\boldsymbol{r_{1,swarm}}$. Using the positions in Table 5.2 with a desired velocity of $1m/s$ was achieved by interpolating value between $\boldsymbol{r_{0,swarm}}$ and $\boldsymbol{r_{1,swarm}}$ over 1 second.

Merge

Merging in this context pertains to the grouping of two previously independent agents or groups of agents. This action is critical to forming the initial swarm formation, and may be used in other scenarios for simplifying trajectory planning and route planning by grouping many agents with similar desired paths. One particular use case could be in a warehouse, where the limited space available could be optimized by forcing agents to travel in formations along predefined "highways" to ensure safe distance from humans and other machinery operating in the space. Five tests were created for the purpose of testing this use-case. Merge case one is a scenario where two agents starting from stationary positions $\boldsymbol{p_{0,d1}}$ and $\boldsymbol{p_{0,d2}}$ two meters apart are instructed to create a formation centered in a point $\boldsymbol{r_{swarm}}$ exactly halfway between $\boldsymbol{p_{0,d1}}$ and $\boldsymbol{p_{0,d2}}$. Starting positions and swarm position reference can be seen in Table 5.3.

Table 5.3 Merge case 1

	Position		
	x_G	y_G	z_G
$\boldsymbol{p_{0,d1}}$	0	-1	1
$\boldsymbol{p_{0,d2}}$	0	1	1
$\boldsymbol{r_{swarm}}$	0	0	1

Merge case two is where two drones hovering in $\boldsymbol{p_{0,d1}}$ and $\boldsymbol{p_{0,d2}}$ one meter apart join to a swarm reference $\boldsymbol{r_{swarm}}$ coinciding with the hover point of agent one, $\boldsymbol{p_{0,d1}}$.

In this case, the ability of the drones to adapt to each others relative velocity and inertia is critical. Starting positions and swarm position reference can be seen in Table 5.4.

Table 5.4 Merge case 2

	Position		
	x_G	y_G	z_G
$\boldsymbol{p_{0,d1}}$	0	0	1
$\boldsymbol{p_{0,d2}}$	0	1	1
$\boldsymbol{r_{swarm}} = \boldsymbol{p_{0,d1}}$	0	0	1

Similarly to merge case one, merge case three is a scenario where one agent and a swarm of three agents starting from stationary positions $\boldsymbol{p_{0,d1}}$ and $\boldsymbol{p_{0,swarm}}$ two meters apart are instructed to create a formation centered in a point $\boldsymbol{r_{1,swarm}}$ exactly halfway between them. In this case, the three agents already in formation will initially be acting as a swarm following a common reference $\boldsymbol{r_{0,swarm}}$. Starting positions and swarm position reference can be seen in Table 5.5.

Table 5.5 Merge case 3

	Position		
	x_G	y_G	z_G
$\boldsymbol{p_{0,d1}}$	0	-1	1
$\boldsymbol{p_{0,swarm}} \approx \boldsymbol{r_{0,swarm}}$	0	1	1
$\boldsymbol{r_{1,swarm}}$	0	0	1

Similarly to merge case two, merge case four is a scenario where one agent and a swarm of three agents starting from stationary positions $\boldsymbol{p_{0,d1}}$ and $\boldsymbol{p_{0,swarm}}$ one meter apart are instructed to join to a swarm reference $\boldsymbol{r_{1,swarm}}$ coinciding with the hover point of the lone agent, $\boldsymbol{p_{0,d1}}$. In this case, the three agents already in formation will initially be acting as a swarm following a common reference $\boldsymbol{r_{0,swarm}}$. Starting positions and swarm position reference can be seen in Table 5.6.

Merge case five is the reverse of merge case four, where one agent and a swarm of three agents starting from stationary positions $\boldsymbol{p_{0,d1}}$ and $\boldsymbol{p_{0,swarm}}$ one meter apart are instructed to join to a swarm reference $\boldsymbol{r_{1,swarm}}$ coinciding with the initial swarm reference, $\boldsymbol{r_{0,swarm}}$. In this case, the three agents already in formation will initially be acting as a swarm following a common reference $\boldsymbol{r_{0,swarm}}$. Starting positions and swarm position reference can be seen in Table 5.7.

Table 5.6 Merge case 4

	Position		
	x_G	y_G	z_G
$\boldsymbol{p_{0,d1}}$	0	0	1
$\boldsymbol{p_{0,swarm}} \approx \boldsymbol{r_{0,swarm}}$	0	1	1
$\boldsymbol{r_{1,swarm}} = \boldsymbol{p_{0,d1}}$	0	0	1

Table 5.7 Merge case 5

	Position		
	x_G	y_G	z_G
$\boldsymbol{p_{0,d1}}$	0	1	1
$\boldsymbol{p_{0,swarm}} \approx \boldsymbol{r_{0,swarm}}$	0	0	1
$\boldsymbol{r_{1,swarm}} = \boldsymbol{r_{0,swarm}}$	0	0	1

Leave

Branching off from the formation is an important function for any swarm implementation as the ability to send agents on individual missions greatly increases the versatility of the swarm. Periodically sending out individual drones to scout certain areas or retrieve/deliver payloads such as a package or some information may increase the utility of the swarm substantially. One major advantage of branching is that it allows for tasks to progress in parallel. Instead of requiring the entire swarm to wait for a data transfer to occur between a provider and a single swarm agent, the swarm may continue with other tasks as a single agent handles whatever objective it was assigned. To test this function, two test cases were implemented where a single agent was instructed to leave a configuration consisting of five agents centered around $\boldsymbol{p_{swarm}}$ by following a separate reference in point $\boldsymbol{p_{branch}}$. In all leave tests, determining which drone to send off was done by selecting the drone closest to $\boldsymbol{p_{branch}}$ when projected on a straight line through the swarm center $\boldsymbol{p_{swarm}}$ and the branch reference point $\boldsymbol{p_{branch}}$. Leave case one consists of a single agent leaving a stationary swarm hovering about a reference $\boldsymbol{r_{swarm}}$ in point $\boldsymbol{p_{swarm}}$. $\boldsymbol{p_{branch}}$ is positioned 1.5 meters away from $\boldsymbol{p_{swarm}}$ according to Table 5.8.

Table 5.8 Leave case 1

	Position		
	x_G	y_G	z_G
$\boldsymbol{p_{swarm}} \approx \boldsymbol{r_{swarm}}$	-0.5	0	1
$\boldsymbol{p_{branch}}$	1	0	1

Leave case two involves detaching a single drone from a moving swarm. Moving the swarm is accomplished by sending a ramp position reference altering the y_G component of swarm reference $\boldsymbol{r_{swarm}}$ from -1 at $t = t_0 = 0$ seconds to 1 at $t = t_2 = 2$ seconds. This corresponds to a 1 m/s change in swarm position reference. The single agent is detached at $t = t_1 = 1$ seconds by assigning $\boldsymbol{p_{branch}}$ as the target. References and positions can be seen in Table 5.9.

Table 5.9 Leave case 2

	Position		
	x_G	y_G	z_G
$\boldsymbol{r_{t_0,swarm}}$	0	-1	1
$\boldsymbol{r_{t_2,swarm}}$	0	1	1
$\boldsymbol{p_{branch}}$	1	0	1

5.2 Analysis

Stability analysis is of major interest when determining the performance of a swarm. Factors to consider are the tendency of the swarm to converge to a stable formation, as well as the density and collision risk of that formation. Convergence in this case can be measured by investigating the sum kinetic energy of all agents in the swarm as a low kinetic energy when hovering is equivalent to all drones having stabilized. The kinetic energy E_k of a swarm containing N agents is determined according to Eqn. (5.1).

$$E_k = \sum_{i=1}^{N} \frac{m \times v_i^2}{2}, v_i = \|\dot{\boldsymbol{p}}_i\| \tag{5.1}$$

Analysis of swarm efficiency and density can be achieved by looking at the distances between agents in the swarm. Looking at the mean distances between all agents may prove misleading, as a large number of agents may lead to overlapping formations where the distance between agents may not be the minimum possible as no stable formation is achievable where every agent is the same distance away from another. Four is the maximum possible number of drones where every agent can maintain the same distance to every other agent in a stable formation, due to the fact that four is the minimum number of nodes for a rigid three-dimensional structure. A different way of investigating size and density is to look at the minimum distance d_i^{min} between each agent i and its closest neighbour; d_i^{min} is calculated according to Eqn. (5.2).

$$d_i^{min} = \min\left\{f(\boldsymbol{p}_j) : \boldsymbol{p}_j = \boldsymbol{p}_1, ..., \boldsymbol{p}_N\right\}, f(\boldsymbol{p}_j) = \left\|\boldsymbol{p}_i - \boldsymbol{p}_j\right\| \tag{5.2}$$

This analysis provides more insight into the safety margins of each agent, but is also susceptible to misleading results as no indication is available as to the rigidity of the formation. A line of drones may show promising results when looking at distance to closest neighbour even though the formation may not take full advantage of the available three-dimensional space. Formation density may instead be investigated by looking at the volume of a sphere encapsulating the entirety of the swarm. Such as sphere may be derived by finding the agent furthest from the swarm center of mass and taking the distance from that agent to the swarm center as the sphere radius. The swarm center is calculated according to Eqn. (5.3). The encapsulating sphere radius r_{enc} is calculated according to Eqn. (5.4).

$$\boldsymbol{p_{swarm}} = \begin{bmatrix} x_{swarm} \\ y_{swarm} \\ z_{swarm} \end{bmatrix} = \sum_{i=1}^{N} \frac{\boldsymbol{p}_i}{N} \tag{5.3}$$

$$r_{enc} = \max\{f(\boldsymbol{p}_i) : \boldsymbol{p}_i = \boldsymbol{p}_1, ..., \boldsymbol{p}_N\}, f(\boldsymbol{p}_i) = \|\boldsymbol{p}_{swarm} - \boldsymbol{p}_i\| \tag{5.4}$$

Controller performance is also closely linked to the swarms ability to follow references accurately and quickly. By mapping the absolute positional error e_{swarm} against time a graph of the swarms ability to follow a setpoint can be generated. e_{swarm} is calculated according to Eqn. (5.5).

$$e_{swarm} = \|\boldsymbol{e_{swarm}}\| = \|\boldsymbol{r_{swarm}} - \boldsymbol{p_{swarm}}\| \tag{5.5}$$

Time analysis is done by examining rise time and settling time during a step response. Rise time is determined to be the time during which the error e_{swarm} is between 90% and 10% of the step distance $\|\boldsymbol{p_1} - \boldsymbol{p_0}\|$. Settling time is determined to be the time required for the error e_{swarm} to go from 95% of the step distance to the moment it enters and remains within 5% of the step distance.

5.3 Distance-based swarm controller

As explained in Sec. 2.2 the distance based approach to swarm control relies on having a set distance between each drone in the swarm. This has the advantage of being able to define a specific formation, although with the downside of not having very dynamic individual drone movements. The algorithm generates a velocity reference for each drone, which is a sum of the following three factors:

1. Keeping formation
2. Following global position reference
3. Avoiding disturbances

Table 5.10 Parameters of the simulated distance position control

Position		
K_p	K_i	K_d
1.5	0	0.1

Table 5.11 Parameters of the real distance position control

Position		
K_p	K_i	K_d
1.2	0	0.1

Formation

For the swarm control, a gradient based controller was chosen. In order to achieve the target distance d_{ij} between agent i and j, the velocity of agent i is defined as Eqn. (5.6)

$$\dot{\boldsymbol{p}}_{\boldsymbol{i}} = -\sum_{j=1}^{N} \frac{(\boldsymbol{p_i} - \boldsymbol{p_j})}{\|\boldsymbol{p_i} - \boldsymbol{p_j}\|} (\|\boldsymbol{p_i} - \boldsymbol{p_j}\| - d_{ij}) \tag{5.6}$$

where d_{ij} is the distance between agent i and j, defined in the distance matrix, which also defines the formation. Note that this merely controls the agents' relative distance to each other and not their actual positioning in the global coordinate system. Due to this, the swarm can rotate in any direction while still keeping the formation. This controller originates from the work of [Anderson et al., 2007]. According to [Sun, 2017], using Eqn. (5.6) causes the distance error to exponentially converge to zero. Merge and leave operations as mentioned in later segments provide a non-trivial challenge in rigid formations, and the formation of the new graph will depend on a number of environmental factors outlined in [Anderson et al., 2008].

Global positioning

In order to be able to regulate the global position of the swarm, a position controller had to be implemented. It is a PID controller which regulates the center position of the swarm rather than that of each individual drone, in order to move the swarm without interfering with the formation. This corresponds to the parallel reference contribution seen in right part of Fig. 5.3. This simply adds its signal on top of the one from formation control. See Tables 5.10 and 5.11 for parameters, which were found through trial and error. The simulation parameters were a bit too aggressive for the real drones and were hence lowered slightly.

Disturbances

The disturbance controller is used to enable obstacle avoidance. Since the drones can only "see" things connected to the Lighthouse positioning system, and not e.g.

walls, other drones are used as obstacles. Due to the nature of the distance controller, the disturbance controller is also needed to be able to interact and avoid collisions with detached (ignored) drones. The controller uses a non-linear curve with cut-off to make sure that a drone is not impacted by the disturbance unless coming close enough. The output signal is defined by Eqn. (5.7) where d_{in} is the distance between drone i at position $\boldsymbol{p}_i$ and obstacle n at position $\boldsymbol{p}_n$, and k_1, k_2 and k_3 correspond to tuning values.

$$\dot{\boldsymbol{p}}_{\boldsymbol{i}} = max(0, k_1 d_{in} + \frac{k_2}{d_{in}} + k_3)(\boldsymbol{p}_n - \boldsymbol{p}_i) \tag{5.7}$$

k_1, k_2 and k_3 are constants, in this case set as -0.3, 0.9 and 0, respectively. These values were iterated in the simulation and then empirically verified in the real swarm to give a smooth and stable reaction on the drones. The max function makes sure that the agents are never drawn to the obstacle. The output is basically a repelling force, rapidly increasing with decreasing space.

5.4 Flocking-based swarm controller

The flocking controller implemented is a variant of the flocking behaviour derived from the basic rules outlined in [Reynolds, 1987] and [Olfati-Saber, 2006]. Each trait corresponds to a goal of the individual drone, and as such the controller is completely dynamic in the sense that no formation or leader is predetermined. The final output of the controller for each drone is derived by taking the sum of all contributing factors. All parameters used in the flocking controller will be sub-indexed by $_{fc}$. The three main factors are:

1. Follow reference
2. Avoid other agents
3. Match speed of neighbours

These three factors combine to define the basic desires of each individual drone to ensure the swarm is controllable, suitably spaced, and stable. By not defining specific positions or formations the swarm should hopefully be able to stabilize dynamically, and should adapt automatically to positional or structural changes triggered by external interactions such as setpoint changes, changed parameters as well as addition or removal of agents or groups of agents. Equation (5.8) shows the notation of each contribution to the final velocity reference $\boldsymbol{u}_{\boldsymbol{v,i}}$ of drone i.

$$\boldsymbol{u}_{v,i} = \begin{bmatrix} u_{vx,i} \\ u_{vy,i} \\ u_{vz,i} \end{bmatrix} = \boldsymbol{u}_{v,i}^{ref} + \boldsymbol{u}_{v,i}^{avoid} + \boldsymbol{u}_{v,i}^{match} = \begin{bmatrix} u_{vx,i}^{ref} + u_{vx,i}^{avoid} + u_{vx,i}^{match} \\ u_{vy,i}^{ref} + u_{vy,i}^{avoid} + u_{vy,i}^{match} \\ u_{vz,i}^{ref} + u_{vz,i}^{avoid} + u_{vz,i}^{match} \end{bmatrix} \tag{5.8}$$

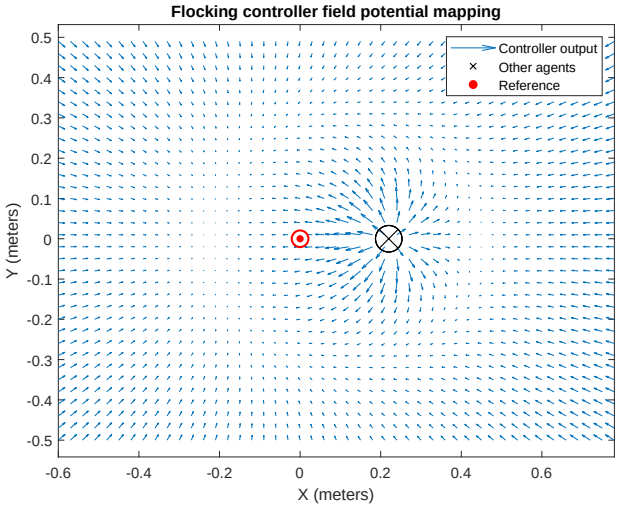

Figure 5.2 Mapping of flocking controller in the plane (z=0) containing one other agent where $k_{fc,ref} = 1$ and $k_{fc,avoid} = 0.1$. Arrows scaled to fit figure. Note the unstable local minimum at $(x = 0.45, y = 0)$ resulting from the other agent shadowing the reference.

This structure is a version of field potential path planning as each real and virtual object in the 3d-space contributes either an attracting or repelling force on the agents. The system can be visualized and analysed by mapping the output of the flocking controller $\boldsymbol{u_v}$ over a range containing a reference and one agent. Such an illustration is visible in Fig. 5.2.

Follow reference

Positional control of the swarm is achieved by providing all agents with a common reference point $\boldsymbol{ref}$ to head towards. The purpose of this reference is partly to provide external control of the drone positions, but also to group the swarm about a common point. This component acts as three proportional controllers for each individual drone i according to Eqn. (5.9). The reference $\boldsymbol{ref}$ is designed to be the point of entry for external control of the swarm, and the center of the swarm should tend towards this point over time to maintain controllability. An important distinction to make here is that the attraction force is converging as opposed to parallel as it is aimed from each individual agent to the swarm reference, meaning all drones are attracted to the same point as opposed to the swarm moving together in parallel

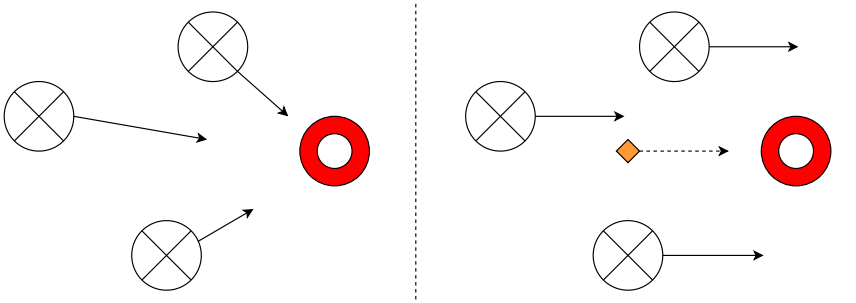

Figure 5.3 Illustration of converging (left) and parallel (right) reference contributions for swarm of three agents following reference in red

towards a target. An illustration of the difference can be seen in Fig. 5.3.

$$\boldsymbol{u}_{v,i}^{ref} = \begin{bmatrix} u_{vx,i}^{ref} \\ u_{vy,i}^{ref} \\ u_{vz,i}^{ref} \end{bmatrix} = k_{fc,ref} \times (\boldsymbol{r} - \boldsymbol{p}_i) = k_{fc,ref} \times \begin{bmatrix} r_x - x_i \\ r_y - y_i \\ r_z - z_i \end{bmatrix} \tag{5.9}$$

Avoid other agents

Avoidance of other agents in the swarm is one of the two components pertaining to avoiding collisions among the drones. Spacing and orientation is defined by the balance between this term and the reference attraction term $\boldsymbol{u}_{v,i}^{ref}$ outlined in Sec. 5.4. $\boldsymbol{u}_{v,i}^{avoid}$ is a repulsive force acting on drone i and is the sum of all reactions between agent i and the N other agents according to Eqn. (5.10). Each contribution $\boldsymbol{u}_{i,j}^{avoid}$ between agent i and j acts in opposite directions on both agents and is calculated according to Eqn. (5.11). By scaling the force by the inverse of the distance, drones are more heavily impacted by other agents that are nearby. This limits the impact of drones far away and allows for more even distribution in larger formations as each agent is primarily affected by its closest neighbours.

$$\boldsymbol{u}_{v,i}^{avoid} = \sum_{j=1}^{N} \boldsymbol{u}_{i,j}^{avoid} \tag{5.10}$$

$$\boldsymbol{u}_{i,j}^{avoid} = -\boldsymbol{u}_{j,i}^{avoid} = \frac{k_{fc,avoid}}{\left\| \boldsymbol{p}_i - \boldsymbol{p}_j \right\|} \times \frac{\boldsymbol{p}_i - \boldsymbol{p}_j}{\left\| \boldsymbol{p}_i - \boldsymbol{p}_j \right\|} \tag{5.11}$$

Match speed of neighbours

An important observation made in [Reynolds, 1987] is that matching the velocity of nearby agents may be of benefit to the behaviour of the swarm. From a control

perspective, adding a term proportional to velocity in a position controller is similar to adding a derivative term to act as a stabilizer and minimize overshoot and low frequency oscillations in the system. Intuitively, maintaining a common velocity with neighboring agents would contribute to avoiding collisions and minimizing erratic movement when in close proximity to other agents. It may also prove beneficial for avoidance of external interference as it would encourage the swarm to move as a unit, propagating the disturbance contribution through the swarm. The match contribution $\boldsymbol{u}_{v,i}^{match}$ is commonly known as velocity consensus [Olfati-Saber, 2006] and can be calculated similarly to the relative positional term outlined in Sec. 5.4. Eqs. (5.12) and (5.13) show the calculations used. Note that the velocity subtraction is reversed in Eqn. (5.13) compared to Eqn. (5.11) to follow traditional error derivative notation.

$$\boldsymbol{u}_{v,i}^{match} = \sum_{j=1}^{N} \boldsymbol{u}_{i,j}^{match} \tag{5.12}$$

$$\boldsymbol{u}_{i,j}^{match} = -\boldsymbol{u}_{j,i}^{match} = \frac{k_{fc,match}}{\left\|\boldsymbol{p}_i - \boldsymbol{p}_j\right\|} \times (\dot{\boldsymbol{p}}_j - \dot{\boldsymbol{p}}_i) \tag{5.13}$$

5.5 Results

Hover

Distance Using the distance controller, a stable hover was achieved in all three cases with one, three and five agents. Swarm center error was kept below 4 cm after stabilizing with a median just above 1 cm.

Fig. 5.4 shows the mean error of the reference distance and the actual distance for each drone. With three agents flying they are mostly around 1 cm off from their set distance, while five agents generate a completely different result with four of the drones with a mean error of about 12 cm. Note that drone one has a mean error of around 7 cm.

Fig. 5.5 shows that the swarm is never completely still, though it mostly shows the one drone being stuck in turbulence. In Fig. 5.6, the five agent swarm quickly stabilizes around 51-52 cm radius, while the three agents take a few seconds longer to end up at 34 cm.

Flocking Using the flocking controller, stable hovering was achieved over a 60 s period with one, three and five drones respectively. The results showed the absolute swarm center error remained below 3 cm for the entire 60 s flight. The distance to the closest neighbour did however not remain constant throughout the flight, as one agent initially stabilizes further away from the rest but converges to a closer position over the 60 s duration as seen in Fig. 5.7. Note that the lines in the figures may overlap as two drones could have each other as closest neighbours thus resulting in identical closest neighbour distances. The minimum distance to neighbours

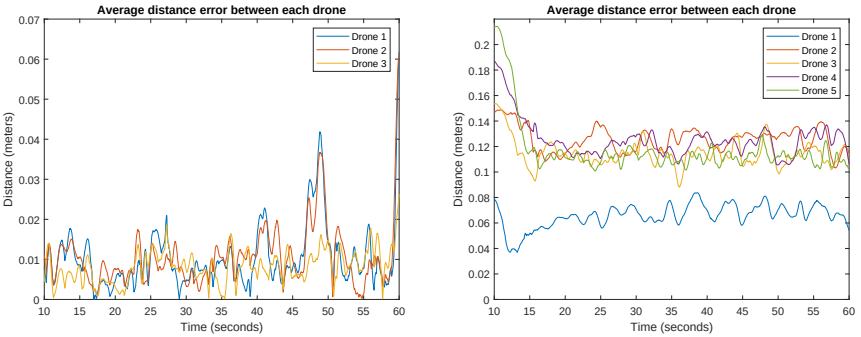

Figure 5.4 Average distance error for all agents during three (left) and five (right) agent flight with distance controller

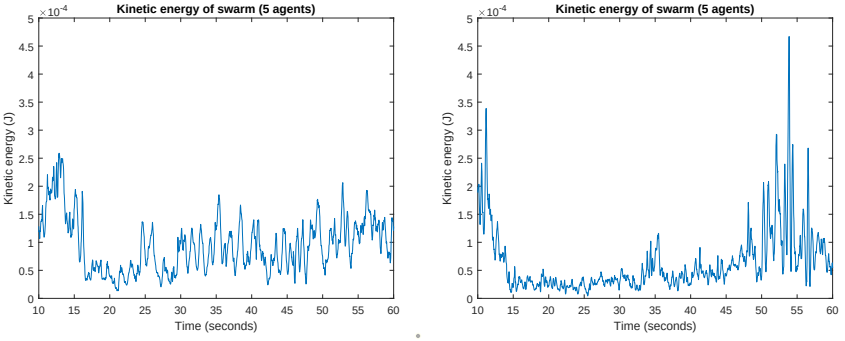

Figure 5.5 Kinetic energy of all agents during five agent hover with distance (left) and flocking (right) controllers

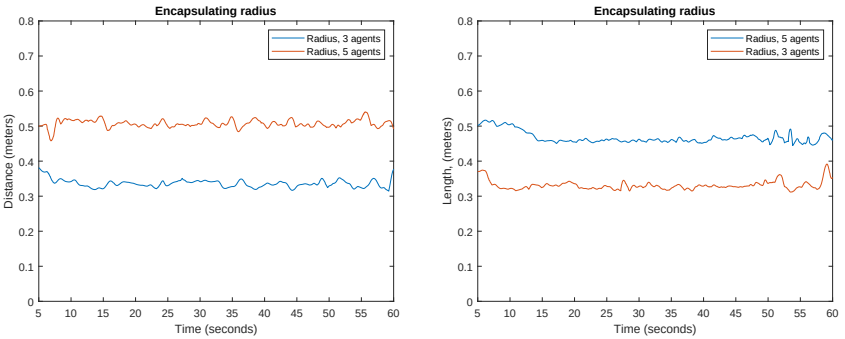

Figure 5.6 Encapsulating radius of three and five agents hovering with distance (left) and flocking (right) controllers

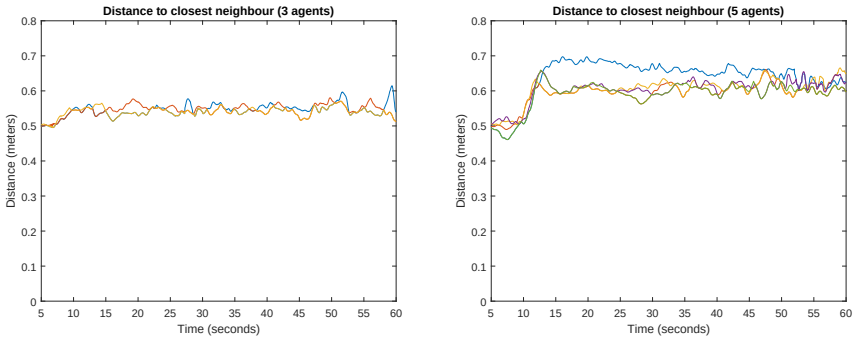

Figure 5.7 Distance to closest neighbour for all agents during three (left) and five (right) agent hover with flocking controller

during stable hovering also seems to be slightly higher in the five agent case than with three. The mean minimum neighbour distance for five agents was 63 cm, as opposed to 54 cm for three agents. During the last 10 s of the five agent flight, one of the drones experienced significant turbulence due to positioning directly underneath another agent. While the flocking controller managed to maintain stability, significant disturbances can be seen in Figs. 5.7 and 5.5 showing the combined kinetic energy of the swarm when hovering. The three agent formation stabilized to an encapsulating radius of 32 cm within 3 s of entering formation, however the five agent formation required 9 s of flight before stabilizing to a radius of 46 cm as seen in Fig. 5.6.

Step

Distance The distance controller performed well in all step cases with reasonable rise and settling times, with the exception of a single agent in the y-step, as well as little to no overshoot; see Table 5.12. An interesting detail is that in every case three agents performed better than a single one. Figure 5.8 shows the average distance error for each drone. Note the change in error around 6 seconds in where the step begins, meaning the position controller has a significant impact on the formation controller, and drone 1 takes several seconds to recover to the same error as the other two agents of just above 2 cm. It consistently has a larger distance error than the other two. Whether this has to do with their individual behaviour and hardware or something else is unclear.

Flocking The swarm behaviour of the flocking controller was not significantly different when flying three drones as compared to one. The resulting rise and settling times of all flight scenarios can be seen in Table 5.13. All rise times were slightly lower in the three agent configuration than in the single agent one. The settling time of the y-step with three agents is significantly higher than that of the single agent, however that is the result of an overshoot exceeding the settling thresh-

Table 5.12 Step response analysis of distance controller

	Rise time (s)		Settling time (s)		Overshoot (m)	
	$N=1$	$N=3$	$N=1$	$N=3$	$N=1$	$N=3$
$\hat{z}_G$	1.40	1.26	2.25	1.98	0	0
$-\hat{z}_G$	1.20	1.09	1.88	1.83	0	0
$\hat{y}_G$	1.49	1.40	4.02	2.07	5.61%	4.61%

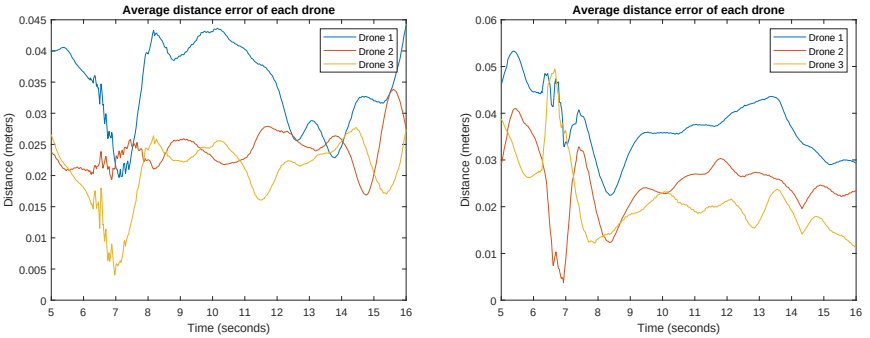

Figure 5.8 Average distance error for each drone in y-step (left) and positive z-step (right) using distance controller

old of 5% used in calculations. None of the step sequences analyzed showed any significant oscillation. The experiment did highlight an interesting aspect of the converging reference structure explained in Sec. 5.4, which is that the closest neighbour distances fluctuate slightly when the step is initiated and completed as the balance between the $\boldsymbol{u}_{v,i}^{\boldsymbol{ref}}$ and $\boldsymbol{u}_{v,i}^{\boldsymbol{avoid}}$ contributions is altered as the direction of the error vector changes. The closest neighbour plot during the step can be seen in Fig. 5.9.

Table 5.13 Step response analysis of flocking controller

	Rise time (s)		Settling time (s)		Overshoot (m)	
	$N=1$	$N=3$	$N=1$	$N=3$	$N=1$	$N=3$
$\hat{z}_G$	1.87	1.58	2.94	2.41	0	0
$-\hat{z}_G$	1.50	1.40	2.30	2.31	0	0
$\hat{y}_G$	1.63	1.53	2.30	4.09	3.27%	5.61%

Ramp

Distance As can be seen in Figs. 5.10 and 5.11 the difference in behaviour between single drone and swarm is very little. The y-ramp results in a slight overshoot,

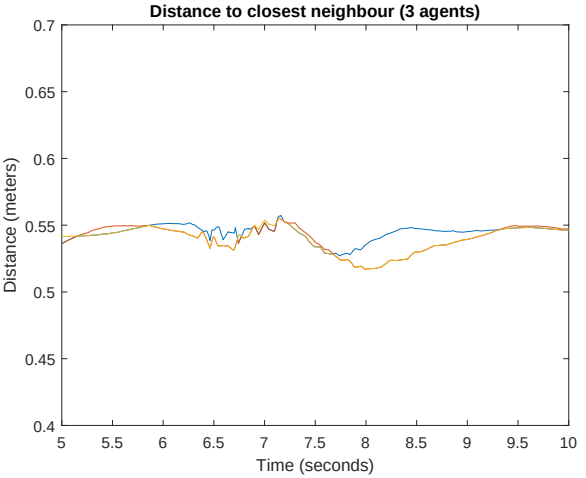

Figure 5.9 Closest neighbour fluctuation with three agent flocking controller. Step is initiated at $t = 6$ seconds

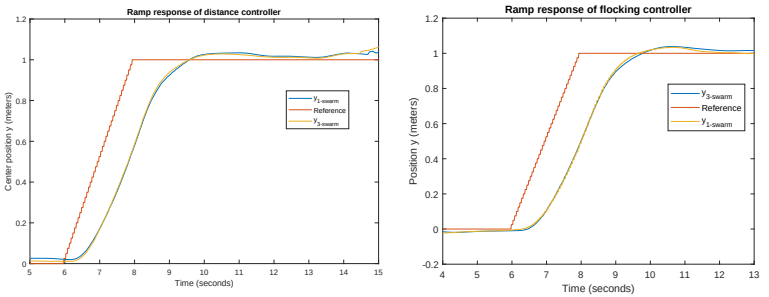

Figure 5.10 Reference and response of y ramp with 1 and 3 agents using distance (left) and flocking (right) controllers

otherwise fine, while the z-ramp instead is struggling with reaching the reference.

Flocking The flocking controller displayed very similar results when operating on one versus three agents. Similarly to the step scenario, the relative positions of the agents fluctuate during and slightly after the ramp. The encapsulating radius of the swarm increased somewhat during the execution of the ramp, however decreased and stabilized again after the ramp was completed as seen in Fig. 5.12.

Merge

Distance The distance controller managed to complete all the five merge cases without any issues. Figure 5.13 displays the distance to the closest neighbour during a merge of a lone agent and a swarm of three agents showing that it takes about

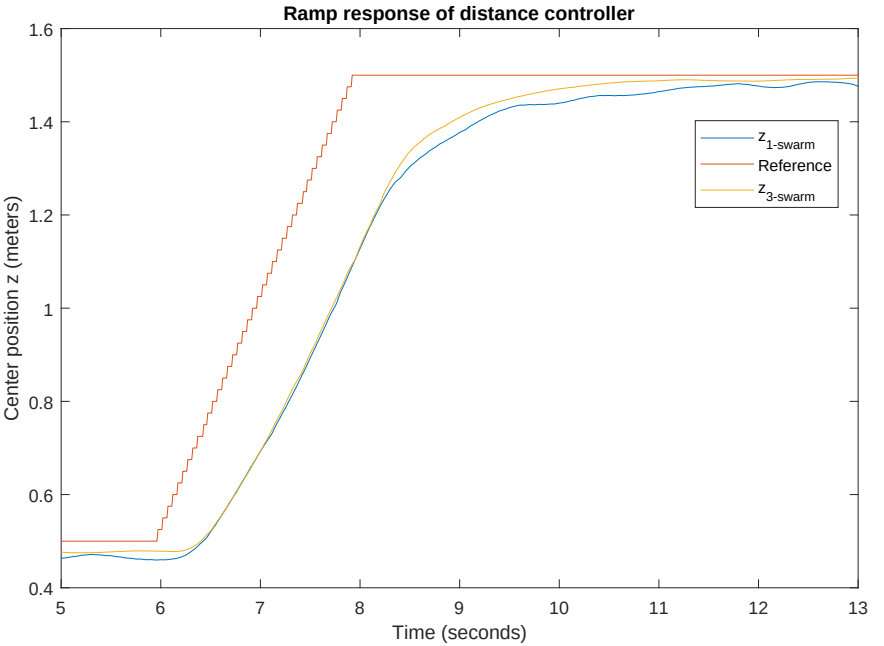

Figure 5.11 Ramp response with 1 and 3 agents in positive z using distance controller

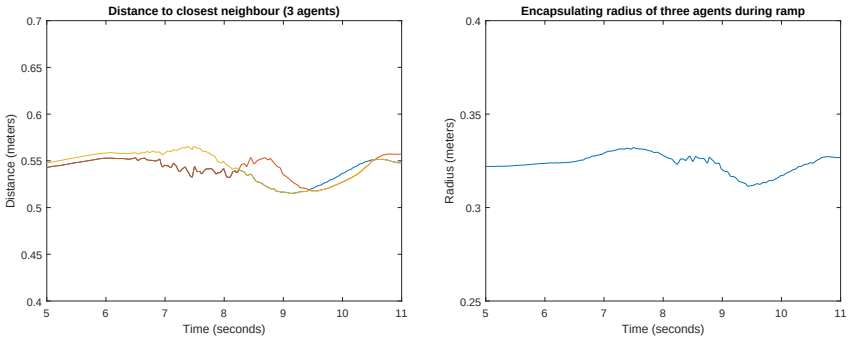

Figure 5.12 Closest neighbour fluctuation (left) and encapsulating radius (right) during three agent ramp with flocking controller. Ramp is initiated at $t = 6$ seconds and completed at $t = 8$ seconds.

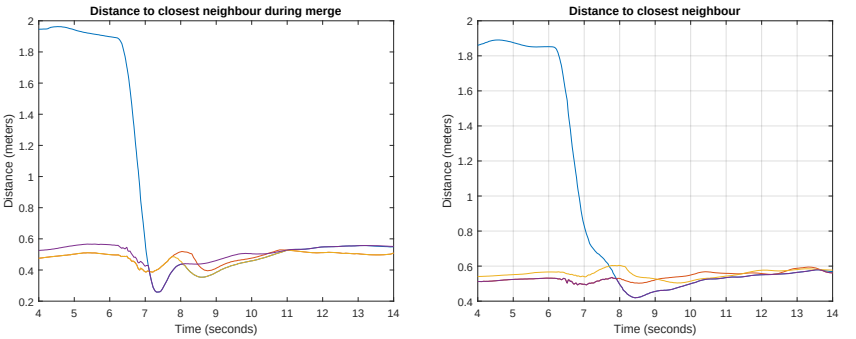

Figure 5.13 Distance to closest neighbour during merge of one and three agents with distance (left) and flocking (right) controllers.

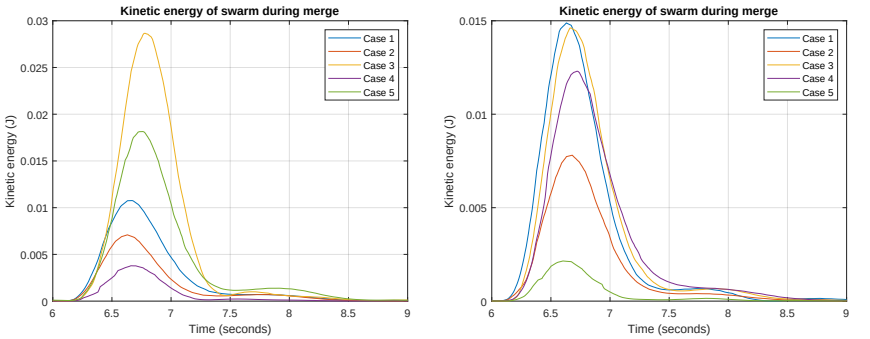

Figure 5.14 Mean kinetic energy of each drone during merge cases one through five with distance (left) and flocking (right) controllers. Merge is initiated at $t = 6$ seconds.

one second for the agent to join the swarm, however it does that with a bit of overshoot. Figure 5.14 draws the kinetic energy of the complete swarm, naturally being highest in the cases where the swarm of three agents are moving. In all cases the swarm settles after around 1.5 seconds after the initial step and is completely still 2.5 seconds in.

Flocking Testing of the merge cases showed that the flocking controller can handle all mentioned scenarios without resulting in a crash, and the minimum closest neighbour distance recorded during testing was 41.7 cm. That occurred during merge case three where a swarm of three agents and a single agent hovering 2 m apart join at a common point halfway between them. The closest neighbour distances recorded in that case can be seen in Fig. 5.13 which shows the lone agent rapidly approaching the swarm and swinging slightly closer to the other agents before stabilizing over a longer time period. The kinetic energy observed during the

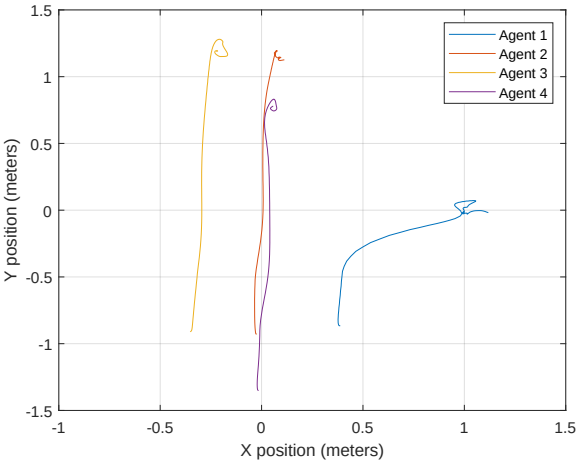

Figure 5.15 Path of the swarm when executing the leave-ramp case with distance controller

tests shown in Fig. 5.14 reveals that the swarm remained in motion longer for cases where more agents were required to move as compared to case five where a single drone joins to a stable swarm of three agents. Directly after the merge is initiated all agents rapidly converge towards a common point, but then remain in motion for up to one additional second before converging to a stable configuration.

Leave

Distance Just like the merge scenarios, the leave cases went smoothly and without any crashes or unstable behaviour. One can see the path taken by each drone from above in the leave-ramp scenario in Fig. 5.15, looking just as expected.

Flocking Leave testing of the flocking controller resulted in stable continuous movement throughout both leave scenarios. A top down view of the path taken by all agents during the ramp leave scenario can be seen in Fig. 5.16. The reference had moved halfway at the time of agent one leaving the swarm, however as is apparent in Fig. 5.10 the flock lags slightly behind the moving reference whereby the leave occurs after the swarm center has moved 40 cm. The closest neighbour results visible in Fig. 5.17 shows that the closest neighbour distances of the remaining swarm changes similarly to the results of the regular ramp cases. It is worth noting that the single agent is sent to a position slightly ahead and to the right of the swarm as can be seen in Fig. 5.16 whereby the distance decreases somewhat at $t = 8$ s as the swarm catches up.

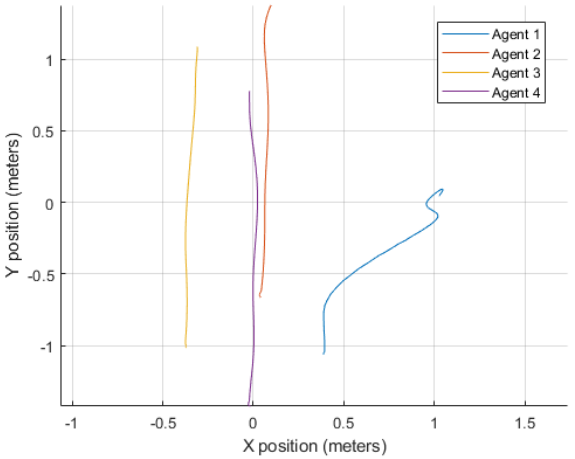

Figure 5.16 Top down view of one agent leaving a formation of four agents during a ramp maneuver. Agent one leaves halfway through the ramp reference change, however the swarm has not yet reached the halfway point.

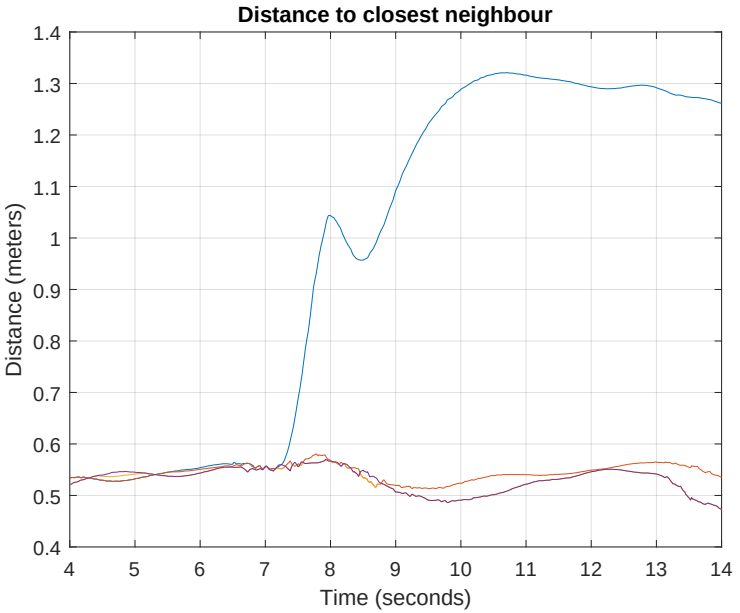

Figure 5.17 Distance to closest neighbour of leave during a ramp maneuver. The lone agent moves to a position slightly ahead of and beside the swarm, whereby the distance decreases slightly at $t = 8$ s as the swarm catches up and moves past

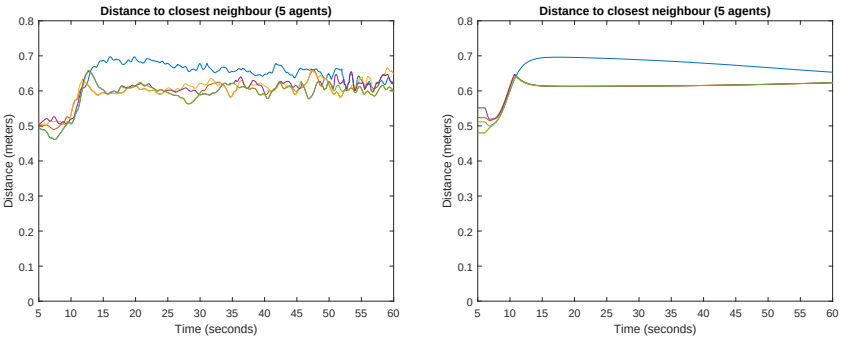

Figure 5.18 Closest neighbour distance during 60 s hover with five agents of real (left) and simulated (right) swarm.

5.6 Swarm model validation

A subset of the scenarios mentioned in Sec. 5.1 were used to investigate the validity of the swarm model. Programming constraints limited the feasibility of certain tests, however results were gathered for the hover, step, ramp and merge test cases. The flocking controller was used for all model validation tests, and all control parameters were identical in the real and simulated tests.

Hover

Comparing the hover performance of the swarm model to the real process certain similarities are immediately obvious when looking at the closest neighbour distance in Fig. 5.18. The result mentioned in Sec. 5.5 pertaining to the lone agent slowly converging is very apparent in the simulation as well, despite the model providing perfect state information. This suggests that the phenomenon is a direct result of the flocking algorithm. The model did however behave very similarly to the real swarm in this scenario. One additional interesting thing to note is that the real process converged quicker than the simulation, possibly due to the measurement noise introducing some randomness thus avoiding the local minimum illustrated in Fig. 5.2.

Step

Figure 5.19 shows a real and simulated swarm of three agents following a step change in $\hat{z}_G$. The simulated swarm moves very similarly to the real swarm, albeit slightly more continuously as the graph of the real swarm can be seen to cross the graph of the simulated swarm several times.

Ramp

The ramp response visible in Fig. 5.20 shows larger discrepancies than for the step response. The real process resulted in overshoot, whereas the simulation managed

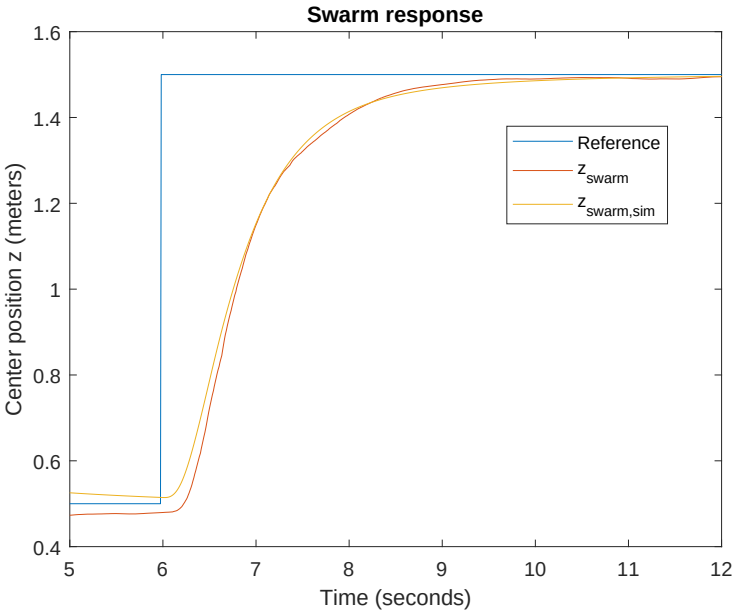

Figure 5.19 Swarm response to positive $\hat{\boldsymbol{z}}_G$ step of real and simulated swarms following flocking controller

to converge to the final reference in a more controlled manner. It also appears the simulation is slightly faster to respond to the changing reference, as the simulation trails somewhat less than the real process.

Merge

The model behaves similarly to the real process when merging, as can be seen in Fig. 5.21. It was however apparent that the simulation managed the merge case in a more controlled manner, as the closest neighbour overshoot is smaller. The kinetic energy analysis visible in Fig. 5.22 also shows much similarity. It should be noted that although the peak kinetic energy is similar, the simulated swarm stabilizes faster and maintains a lower kinetic energy during the stabilization phase. This may be a result of the measurement noise giving way to added oscillations.

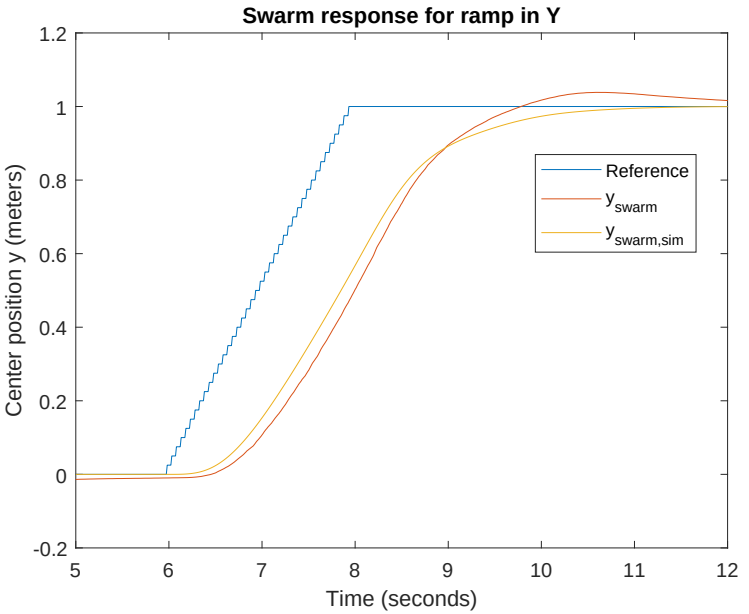

Figure 5.20 Swarm response to positive $\hat{\boldsymbol{y}}_G$ ramp of real and simulated swarms following flocking controller

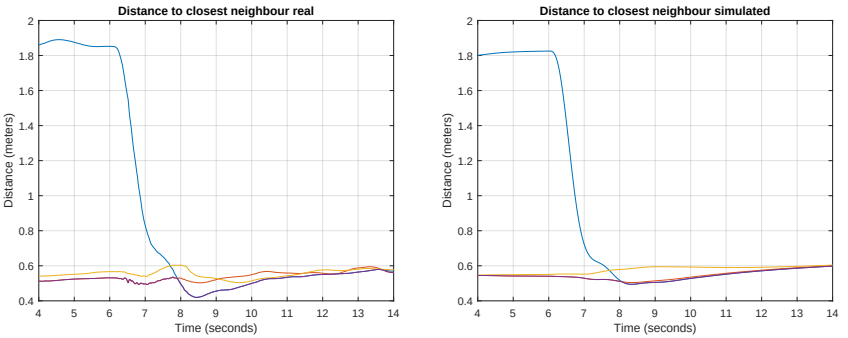

Figure 5.21 Distance to closest neighbour during merge of one and three agents with real (left) and simulated (right) swarm.

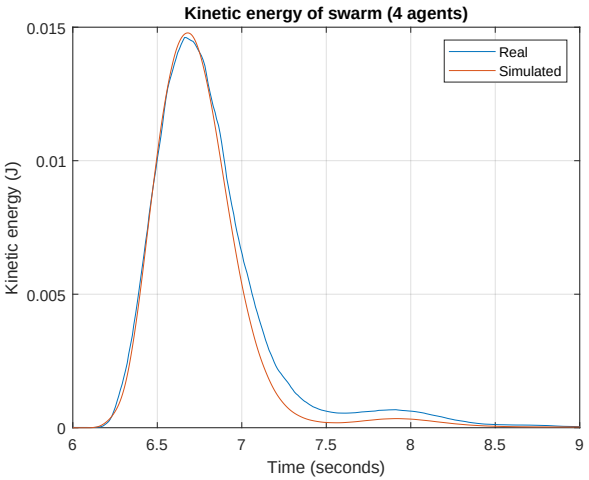

Figure 5.22 Kinetic energy of real and simulated swarm during merge case three agents.

6

Discussion

6.1 Crazyflie firmware modelling

The previous work on the rigid body and rotor dynamics of the Crazyflie provided a very good foundation to work from, and as can be seen in Secs. 3.2 and 3.3 our implementation of the cascading attitude and velocity controllers provided a very accurate model of the drone behaviour in the roll and pitch directions. There are however discrepancies in the tuning of the controllers, most notably in the yaw attitude controllers and the $\hat{\boldsymbol{z}}_G$ velocity controller. The tuning of the yaw controllers (seen in Tables 3.2 and 3.3) differs significantly. The simulated controller tuning required halving the gain as well as adding a large derivative action not present in the firmware, indicating an error somewhere in the model. As a result, the yaw comparison seen in Fig. 3.6 shows that the model will follow the reference similarly to how the real process does, but the characteristics of the step differ in terms of rise time and overshoot. For this reason, the model shows promising results in scenarios where the yaw behaviour is not of critical importance. As none of the experiments and sequences implemented in the thesis require aggressive yaw references the response was deemed good enough, but could definitely use more work. Complete decoupling of the different attitude controllers may not be the best solution for this physical plant, as all changes in rotor speed brought on by the attitude controllers will have an effect on the other states as well. This is because of the rotor configuration.

The simulated $\hat{\boldsymbol{z}}_G$ velocity controller behaved significantly different when compared to the real process as can be seen in Fig. 3.12. The cause of this discrepancy was not discovered, however once again the behaviour was similar enough to allow for use in simulations. By limiting the test scenarios to motion mostly in the $\boldsymbol{xy}_G$ plane the impact of the $\hat{\boldsymbol{z}}_G$ velocity discrepancy was mitigated.

Another area that could use more investigation is the accuracy of the lighthouse positioning system. Due to time constraints we were unable to focus on implementing an accurate emulation of the extended Kalman filter used in the firmware, whereby the impact of inaccurate position and velocity measurements could not be properly examined. As mentioned in Sec. 4.2 the lighthouse positioning system

provides sub-cm precision even when in motion which was the main motivation for choosing that system.

6.2 Swarm modelling

The simulated swarm showed good accuracy in terms of dynamics and behaved very similarly to the real system when testing step and ramp responsiveness. Because we chose to implement each cascading controller completely instead of simplifying the plant through approximated transfer functions, the simulation is quite heavy to run when many drones are active. Implementing it this way does however allow for more insight into the movement of the quadcopters, and factors such as drag and measurement noise can be studied at any point in the controller chain. One area which needs more work is the implementation of turbulence simulation. As mentioned in the results Sec. 5.5 the system experiences major instability when a drone happens to position underneath another, and the swarm controller could definitely be improved by incorporating a feature to avoid this situation. In the current implementations, no effort is made to avoid turbulence as the system seemed capable of handling the disturbance provided the distance between the drones affected is large enough. Possible fixes for this flaw include adding a term to nudge any drone experiencing turbulence out of the affected area, or changing the currently symmetric formations to another which favours distributing the drones in the $\boldsymbol{xy}_G$ plane rather than along $\hat{\boldsymbol{z}}_G$. Such an addition would be highly recommended if the system is to be used in situations where a very close formation is required. For simulations of higher numbers of active agents, it is important to keep in mind the limitations of the communications protocol used. The limit of 20 Hz polling rate used in this thesis may only be applicable for systems of five agents or below using the CRTP protocol. Interference may also become a problem as the throughput to each agent is already severely restricted by the single Crazyradio used in the experiments.

6.3 Python implementation

Although the current Python implementation is sufficient to conduct sequential experiments and rapid testing, certain improvements could be made to allow more general use in a broader range of applications. The connection procedure and logging systems have a rather high chance of successfully initiating and stabilizing, however there are still situations where a restart of one or more Crazyflies is required. Currently, an incorrectly closed connection will produce an exception when attempting to reconnect and although the execution may continue successfully regardless, it is not clear what effects if any this has on the execution. One major area that could possibly see improvement is the sampling. As mentioned in Sec. 4.4 the sampling and controller execution is currently decoupled as a result of the controller updates requiring the state of the entire swarm to compute correctly. By minimizing

the time between new information arriving and a controller output being computed, the stability and responsiveness of the system may be improved. One possibility is to alter the controller implementation to be dynamically updated, however this may require more computational power from the centralized computer. Alternatively, the required information could be contained to perhaps only the state of the closest neighbours, which could improve performance and scalability substantially. A limitation in the current Crazyflie implementation is the absence of a global yaw correction. The lighthouse framework has the potential to include such a feature in the future, but in the current implementation the Crazyflie must be aligned with the external positioning system at startup and the yaw estimation will drift over time.

6.4 Methodology and analysis

The methods for analysing the swarm stability and performance outlined in Sec. 5.2 were found to be a decent measure of the controller performance, however some aspects were difficult to evaluate due to the high degree of freedom inherit to this kind of physical process. How efficient the space was used is difficult to quantify as there are a plethora of use-cases where a simple spherical swarm formation may not be desired, such as when traversing narrow spaces or obstacle-filled environments. Closest neighbour distance shows some measure of collision safety in terms of allowing analysis of overshoot when performing operations such as merging or changing reference, but provides little information as to the rigidity and flexibility of the formation. The differing nature of the two swarm controllers investigated also provides some challenge when constructing experiments, as the controllers may be suited for different applications. Precise relative control such as the one used in the distance controller may be very useful for applications where the number of engaged drones stays the same throughout the operation, whereas the flexibility of the flocking controller makes adding and removing agents from the swarm very easy at the cost of not having the same absolute positional control. Time constraints also resulted in a lot of time being spent developing and preparing the platforms used for the experiments. Energy analysis such as the one used here provided rather good insight into the consistency of the formation, and also indirectly highlighted the effects of turbulence seen in the hover experiments. With more time, a more thorough investigation into the turbulence effects as well as measurement accuracy could greatly improve the swarm model as well as the performance of the two controllers.

The *LogManager* and *Log* classes used for data gathering allowed for very easy setup of new logging parameters, but once again the decoupling may be a possible source of faulty information as the slight delay introduced by the regular polling intervals causes the log entry to be registered slightly after the actual value has changed. With the log polling rate set to 100 Hz, the error caused by this was deemed too small to be of concern considering the state updates and controller computations occurred at 20 Hz.

6.5 Controller comparison

Firstly, it is difficult to compare the two controllers in a reasonable way since they are quite different and the level of relevance varies. For example, distance to the closest neighbour is interesting in a flocking perspective, while it is generally not for the distance controller since the absolute distance is a reference making the distance error a more relevant parameter. Because of this, some cases are not intended to be comparable but were rather done to make sure the controller could complete the task without any obvious instability issues.

With that said, some cases are more easily comparable. Looking at Tables 5.12 and 5.13, one can see that the distance controller performs better in all steps except the lone agent in $\hat{\boldsymbol{y}}_G$, although the differences are minor. With the ramp in $\hat{\boldsymbol{y}}_G$-direction in 5.10 it is hard to tell the controllers apart, with very similar results.

Regarding stability, the flocking controller seems to be much more steady when observing the kinetic energy and radius in Fig. 5.5 and 5.6. Even so, this could be part of the recurring issue of turbulence.

Comparing merges, Fig. 5.13 displays interesting behaviour. The lone agent joins the swarm quickly but also too aggressively using the distance controller, overshooting and coming dangerously close with a minimum distance of about 25 cm. The flocking controller does a much smoother merge, slowing down the drone before merging with only a slight overshoot.

An important note is that none of the controllers parameters have been optimized due to time limitations and simulation differences and have merely been chosen for stability while not necessarily having the highest performance. An attempt at optimizing the position part of the distance controller was made in the simulation, but the resulting parameters were found to be too aggressive in the real world making the drones very unstable. Furthermore a scalar of 0.5 had to be added to the formation control, otherwise the drones would fall into a heavy oscillation resulting in a crash.

An interesting subject is decentralization, and how well suited the controllers are for that. This would mean moving the velocity reference computation from the centralized computer to the drones. Most importantly this would lower the amount of communication issues and delays, enabling a faster controller as that is the most significant bottle neck of the system. An immediate issue with the current implementation of the distance controller is that all the drones would have to communicate with each other to keep the formation, whereas with flocking a drone would only need to communicate with its neighbours. Less data and computation is of high importance because of the drones' low computational power, although the controllers' computational cost has not been measured.

6.6 Further work

Decentralization

As mentioned in Sec. 6.5, a decentralized implementation of the controllers tested could significantly reduce the complications resulting from transport delays and communication errors. Due to current restrictions in the Crazyflie firmware, agent to agent communication is limited at best and a lot of work would be required to support inter-agent information sharing. Such an implementation could allow for much more aggressive control due to the closer link between controller and the individual drone hardware. Additionally, more information could perhaps be shared and processed in clusters as opposed to the whole swarm as with the centralized controller and state register. One interesting area of research is the implications on swarm performance of limiting state information available to the controller. An option would be to only allow communication within a certain range of each agent, to further improve the viability of decentralized control.

Mobile base station

If the controller is to remain centralized, the implementation could see great improvement from being mobile rather than stationary. Operation in known areas is well suited for centralized control, such as in a warehouse or for surveillance purposes, however for exploratory purposes it may prove beneficial to allow the centralized controller to follow the swarm as it moves through unknown space. With some modifications, the lighthouse positioning system used in the thesis could be mounted on a mobile ground based platform and provide continuous referencing through unknown spaces. One of the major difficulties in autonomous swarm control is the absence of an absolute position reference, but this problem could potentially be solved by periodically returning the drones to a known space provided by the base station. For search and rescue operations, the search space could branch out from the base station to cover a much larger area than what would be possible with only a ground based vehicle or even a single aerial drone.

Heterogeneous swarm

Investigation into the combination of multiple different agent types in a single swarm could provide similar benefits as those mentioned in Sec. 6.6. The small Crazyflie may be well suited for scouting in enclosed areas, but by adding other types of vehicles a more versatile swarm could be achieved. Adapting the controllers to allow for differing dynamics may prove challenging, both due to complex dynamics but also because of varying communication protocols.

Obstacle avoidance

For the controllers to behave truly dynamically, an obstacle avoidance algorithm could be implemented to simplify path planning and manual control. Dynamic

avoidance of obstacles could significantly improve the usability and stability of the swarm, and could perhaps also incorporate a way to avoid the turbulence problems encountered during experiments. Although unlikely, collisions have occurred during testing where a single agent has lost contact with the swarm and drifted into the path of another. A local avoidance algorithm could counter that and other such cases where control of individual agents was lost.

Other positioning systems

The experiments conducted in this thesis all implement the lighthouse positioning system described in Sec. 4.2, which has relatively good accuracy when used in combination with the onboard Kalman filter. Other positioning systems exist in the Crazyflie ecosystem, such as the Loco Positioning System and the Flow deck [Bitcraze, 2019a]. The effects of increased measurement noise and inaccuracy have not been examined in this thesis, but could be of interest in use cases with significant noise and disturbance in the environment. Although the controllers behaved well with good positional accuracy, the performance and stability may be significantly impacted by communication errors and measurement noise.

Bibliography

Anderson, B. D. O., C. Yu, S. Dasgupta, and A. Morse (2007). "Control of a three-coleader formation in the plane". *Systems & Control Letters* **56**:9, pp. 573–578.

Anderson, B. D. O., C. Yu, B. Fidan, and J. M. Hendrickx (2008). "Rigid graph control architectures for autonomous formations". *IEEE Control Systems Magazine* **28**:6, pp. 48–63.

Bitcraze (2019a). *Bitcraze official home page*. `https://www.bitcraze.io/`.

Bitcraze (2019b). *Crazyflie crazy realtime protocol*. `https://wiki.bitcraze.io/projects:crazyflie:firmware:comm_protocol`.

Bitcraze (2019c). *Crazyflie firmware github*. `https://github.com/bitcraze/crazyflie-firmware`.

Bitcraze (2019d). *Crazyflie python library github*. `https://github.com/bitcraze/crazyflie-lib-python/tree/master/cflib`.

Bitcraze (2019e). *Crazyradio*. `https://wiki.bitcraze.io/doc:lighthouse:setup`.

Bitcraze (2019f). *Crazyradio*. `https://wiki.bitcraze.io/projects:crazyradiopa:index`.

Chung, S.-J., A. A. Paranjape, P. Dames, S. Shen, and V. Kumar (2018). "A survey on aerial swarm robotics". *IEEE Transaction on Robotics* **34**:4, pp. 837–855.

Corporation, V. (2019). *Steamvr tracking*. `https://partner.steamgames.com/vrlicensing\#Tracking`.

Engebråten, S., K. Glette, and O. Yakimenko (2018). "Field-testing of high-level decentralized controllers for a multi-function drone swarm". In: IEEE 14th International Conference on Control and Automation. Anchorage, Alaska, USA, pp. 379–386.

Förster, J. (2015). *System Identification of the Crazyflie 2.0 Nano Quadcopter*. Institute for Dynamic Systems and Control Swiss Federal Institute of Technology, Zurich, Switzerland.

Green, S. and P. Månsson (2019a). *Crazyflie-swarm-python github repository*. `https://github.com/greensebastian/crazyflie-swarm-python`.

Green, S. and P. Månsson (2019b). *Crazyflie-swarm-simulink github repository*. `https://github.com/greensebastian/crazyflie-swarm-simulink`.

Greiff, M. (2017). *Modelling and Control of the Crazyflie Quadrotor for Aggressive and Autonomous Flight by Optical Flow Driven State Estimation*. ISSN 0280-5316. Dept of Automatic Control, Lund University, Lund, Sweden.

Luukkonen, T. (2011). *Modelling and Control of Quadcopter*. Independent research project in applied mathematics, Aalto University School of Science, Espoo, Switzerland.

Mac, T. T., C. Copot, R. D. Keyser, and C. M. Ionescu (2018). "The development of an autonomous navigation system with optimal control of a UAV in partly unknown indoor environment". *Mechatronics* **49**, pp. 187–196.

Oh, K.-K., M.-C. Park, and H.-S. Ahnb (2015). "A survey of multi-agent formation control". *Automatica* **53**:1, pp. 424–440.

Olfati-Saber, R. (2006). "Flocking for multi-agent dynamic systems: algorithms and theory". *IEEE Transactions on Automatic Control* **51**:3, pp. 401–420.

Preiss, J. A., W. Hönig, G. S. Sukhatme, and N. Ayanian (2017). "Crazyswarm: a large nano-quadcopter swarm". In: IEEE International Conference on Robotics and Automation. Singapore, May 29 - June 3, 2017, pp. 3299–3304.

Reynolds, C. W. (1987). "Flocks, herds, and schools: a distributed behavioral model". *Computer Graphics* **21**:4.

Richardsson, K. (2017). *TDOA swarm with the loco positioning system*. `https://www.bitcraze.io/2017/02/tdoa-swarm-with-the-loco-positioning-system/`.

Sun, Z. (2017). *Cooperative Coordination and Formation Control for Multi-agent Systems*. ISBN 978-3-319-74264-9. Australian National University, Canberra, Australia.

Taffanel, A. (2019). *Release of the lighthouse deck early access*. `https://www.bitcraze.io/2019/03/release-of-the-lighthouse-deck-early-access/`.

Vásárhelyi, G., C. Virágh, G. Somorjai, T. Nepusz, A. E. Eiben, and T. Vicsek (2018). "Optimized flocking of autonomous drones in confined environments". *Science Robotics* **20**:3. art. no. eaat3536.

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER'S THESIS

Date of issue
June 2019

Document Number
TFRT-6084

Author(s)
Sebastian Green
Pontus Månsson

Supervisor
Simon Yngve, Combine Control Systems
Zhiyong Sun, Dept. of Automatic Control, Lund University, Sweden
Anders Robertsson, Dept. of Automatic Control, Lund University, Sweden
Rolf Johansson, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle
Autonomous control of unmanned aerial multi-agent networks in confined spaces

Abstract

In this thesis, the functionality of existing Crazyflie models is extended to include current implementations of attitude and velocity control. The models are validated both as individual agents and as a swarm through comparisons of swarm behaviour during step, ramp and frequency response testing. The models are found to accurately replicate the dynamics of the real system, but additional research on measurement noise and accuracy would be necessary to ensure that the model remains accurate in less than ideal conditions. Several suggestions for further improvements are presented. Additionally, two separate swarm controllers are implemented and tested in a simulated environment as well as the real world to demonstrate the capabilities of the model and to evaluate the controller usability in a number of practical use cases. Both controllers are found to behave well on the Crazyflie system, and demonstrate the practicality of the Crazyflie platform as well as the controllers implemented.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-69

Recipient's notes

Security classification