Master of Science Thesis in Electrical Engineering
Department of Electrical Engineering, Linköping University, 2022

# Intelligent drone swarms

## Motion planning and safe collision avoidance control of autonomous drone swarms

**Adam Åsbrink and Hilding Gunnarsson**

LINKÖPING UNIVERSITY

Master of Science Thesis in Electrical Engineering

**Intelligent drone swarms: Motion planning and safe collision avoidance control of autonomous drone swarms**

Adam Åsbrink and Hilding Gunnarsson

LiTH-ISY-EX--22/5474--SE

Supervisor: **Daniel Arnström**
ISY, Linköpings universitet
**Markus Andersson**
Saab Dynamics AB

Examiner: **Daniel Axehill**
ISY, Linköpings universitet

*Division of Automatic Control*
*Department of Electrical Engineering*
*Linköping University*
*SE-581 83 Linköping, Sweden*

## Abstract

The use of unmanned aerial vehicles (UAV), so-called drones, has been growing rapidly in the last decade. Today, they are used for, among other things, monitoring missions and inspections of places that are difficult for people to access. To efficiently and robustly execute these types of missions, a swarm of drones may be used, i.e., a collection of drones that coordinate together. However, this introduces new requirements on what solutions are used for control and navigation. Two important aspects of autonomous navigation of drone swarms are formation control and collision avoidance.

To manage these problems, we propose four different solution algorithms. Two of them use leader-follower control to keep formation, Artificial Potential Field (APF) for path planning and Control Barrier Function (CBF)/Exponential Control Barrier Function (ECBF) to guarantee that the control signal is safe i.e. the drones keep the desired safety distance. The other two solutions use an optimal control problem formulation of a motion planning problem to either generate open-loop or closed-loop trajectories with a linear quadratic regulator (LQR) controller for trajectory following. The trajectories are optimized in terms of time and formation keeping. Two different controllers are used in the solutions. One of which uses cascade PID control, and the other uses a combination of cascade PID control and LQR control.

As a way to test our solutions, a scenario is created that can show the utility of the presented algorithms. The scenario consists of two drone swarms that will take on different missions executed in the same environment, where the drone swarms will be on a direct collision course with each other. The implemented solutions should keep the desired formation while smoothly avoiding collisions and deadlocks. The tests are conducted on real UAVs, using the open source flying development platform Crazyflie 2.1 from Bitcraze AB. The resulting trajectories are evaluated in terms of time, path length, formation error, smoothness and safety.

The obtained results show that generating trajectories from an optimal control problem is superior compared to using APF+leader-follower+CBF/ECBF. However, one major advantage of the last-mentioned algorithms is that decision making is done at every time step making these solutions more robust to disturbances and changes in the environment.

## Acknowledgments

We would like to give our warmest appreciation to our supervisor Daniel Arnström who's insight and knowledge into the subject matter steered us every week through this project. Markus Andersson, for providing support, valuable reviews and most importantly, making us feel welcomed at Saab Dynamics. We are grateful to all employees at GNC for fun coffee and lunch breaks and a special thanks to Thorbjörn Crona for letting us do our master's thesis at Saab Dynamics and Kristoffer Bergman, the originator of the project. Finally, a big thanks to our examiner, Daniel Axehill, for helping us with all practicalities regarding our thesis.

*Linköping, May 2022*
*Adam Åsbrink and Hilding Gunnarsson*

# Contents

# Notation

## Abbreviations

| Abbreviation | Meaning |
|---|---|
| APF | Artificial Potential Field |
| CBF | Control Barrier Function |
| ECBF | Exponential Control Barrier Function |
| LF-control | Leader Follower control |
| LQR | Linear Quadratic Regulation |
| MAV | Micro Aerial Vehicle |
| MPC | Model Predictive Control |
| MMSE | Minimum Mean Square Error |
| MSE | Mean Squared Error |
| PID controller | Proportional Integral Derivative controller |
| QP-problem | Quadratic Programming problem |
| ROS | Robot Operating System |
| TCAS | Traffic Alert and Collision Avoidance Systems |
| UAV | Unmanned Aerial Vehicle |
| UWB | Ultra-Wideband |

# 1 Introduction

The aim for this thesis is to implement formation control with efficient guidance while guaranteeing safety for a swarm of drones. The master thesis is a collaboration between Linköping University and Saab Dynamics AB. This chapter introduces the problem and provides an outline of the thesis.

## 1.1 Background

The application of unmanned aerial vehicles (UAVs), so-called drones, has grown rapidly in the last decade. Today, they are used for, among other things, monitoring missions and inspections of places that are difficult for people to access. To efficiently and robustly execute these types of missions, swarms of drones can be used. However, it places new demands on what solutions are used for control and navigation of the individual drones. The particular drone used in this project is the Crazyflie 2.1 from Bitcraze AB [6] which is a very small and lightweight UAV also known as a micro aerial vehicle (MAV). The advantages of using MAVs are their agility and ability to move through narrow spaces. There are also several advantages of using a MAV instead of a UAV in research projects because of easier testing and portability [1].

Formation control of drone swarms are typically divided into three different approaches: leader-follower approaches, behavior based approaches and virtual structures. This project will focus on a leader-follower approach which means that one drone in the swarm are designated as the leader and the other drones are followers. A well performing formation control is when the relative displacement of each drone do not change, i.e., the formation keeps a desired formation. By controlling the movement of the leader, the whole formation will follow suit [27].

To effectively avoid obstacles and reach a desired target, guidance or path planning algorithms need to be implemented. One common approach for UAVs is to use so called artificial potential fields (APFs), which are widely used because of their simplicity, computational efficiency and smooth trajectory generation [23]. The APF algorithm can be likened to a magnetic field where each neighbouring drone can be seen as a repulsive magnet and the desired target can be seen as an attracting magnet. The resultant force from the magnets guides each drone to a desired target while avoiding other drones.

The last step to a well performing system is to guarantee safety for each drone in every time step, i.e., that the drones do not collide with each other. For this thesis a control barrier function (CBF) was used. From formation control and guidance, each drone gets a nominal control signal and in the last step a quadratic programming problem is formulated that minimizes the error between the control signal and the nominal control signal that satisfies safety constraints from the control barrier function. CBF is a mathematical way of ensuring that a set of states is forward invariant, which means that if a safe set of states is defined, CBF ensures that the drones always remains within this safe set [28]. An extension to CBF called exponential control barrier function (ECBF) is also used.

This thesis will begin with an overview of the system including hardware, software as well as a model of the system in Chapter 2. Secondly, the theory and implementations for control, safety and guidance are presented in their respective chapters, namely Chapter 3, 4, and 5. Four different solutions to the problem formulation will be presented in Chapter 6 and in Chapter 7, the results will be presented. Lastly in Chapter 8, the results will be discussed and in Chapter 9, conclusions and potential improvements are presented.

## 1.2 Problem formulation

This thesis aims to investigate existing solutions and develop algorithms that can be used for collision-safe control and motion planning of drone swarms. As a way to rationalize the aims of this thesis, a scenario is created that can show the utilization of the presented algorithms. The scenario consists of two drone swarms that will take on different missions executed in the same environment, where the drone swarms will be on a direct collision course with each other. This should be implemented and tested on the quadcopter Crazyflie 2.1. To simplify the problem and avoid turbulent air beneath each drone, the motion of the drones is constrained to the horizontal plane. The performance of the solutions will be evaluated on five aspects

- Time - How fast do the swarms finish their respective missions?
- Path length - How long are the resulting paths?

- Smoothness - Do the collision avoidance algorithms and control of the drone result in smooth and smart trajectories?
- Safety - Do the drones maintain a safe distance from each other?
- Formation keeping - Is the desired formation kept during the mission?

# 2 System Overview

This chapter provides an overview of the hardware, software and a model of the Crazyflie.

## 2.1 Hardware

The platform is a Crazyflie 2.1, shown in Figure 2.1, from the Swedish company Bitcraze.

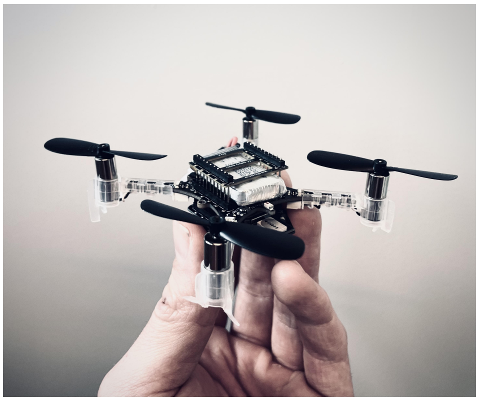


***Figure 2.1:*** *Crazyflie 2.1 quadcopter from Bitcraze.*

This is an open source quadcopter that weighs only 27 grams and is therefore well suited for research projects on drone swarms. Lots of research has been made using the platform and thorough documentation makes it easy to get started. Extensions to the quadcopter can be added such as a positioning system as well as decks to improve the versatility and performance. In this work the ultra-wideband (UWB) positioning system called Loco and the Flow deck was used [6]. The Flow deck measures distance to the ground and velocity which improves the position and velocity estimation significantly compared to only using Loco. The precision of combining Loco and a Flow deck is in the range of centimeters [6]. For communication between a computer and Crazyflies, Bitcraze provides the "Crazyradio PA" which is a long range USB based radio dongle [7]. The computer used in this thesis is the Dell Latitude 5520 with processor Intel core i7 and RAM 16GB.

### 2.1.1 UWB positioning system

UWB is a quite new radio technology that have several applications, including positioning. The UWB localization works similarly to GPS, but with shorter range and better precision [2]. The setup used in this thesis is eight UWB anchors placed in the room at known positions, forming a global reference frame. Each crazyflie has a UWB node onboard which communicates with the anchors and estimates its position with respect to the global frame [6]. An example setup with two anchors is shown in Figure 2.2.

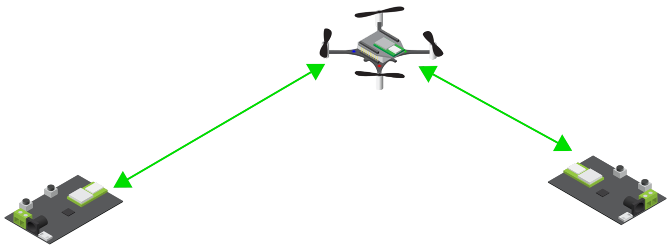


***Figure 2.2:*** *Loco positioning system.*

### 2.1.2 Flow deck

The Flow deck measures both the distance to the ground and the velocity in its body frame. The distance measurement is based on a time of flight ranging sensor which measures the round trip time of an artificial light signal, i.e. a laser [11]. The optical flow sensor measures movement relative to the ground, which is used to estimate the velocity in its body frame [6].

## 2.2 Software

The software platform chosen for this project was Crazyswarm [19]. The other option was to use Bitcraze Python API. However, one major advantage with Crazyswarm was the simple but helpful simulation environment. One other difference compared to Bitcraze Python API is that Crazyswarm is built upon ROS (Robot Operating System) with a Python API as a thin wrapper around the ROS interface.

Included in Crazyswarm is communication with the drones, position estimation as well as a Python API for among other things sending commands and getting the position estimation of the drones. During the project there was also a need for solving optimal control problems and for this the software tool CasADi was used. CasADi is an open-source software tool for solving optimization problems and optimal control problems [3].

### 2.2.1 Robot Operating System (ROS)

On a high level, "ROS is an open-source, meta-operating system for your robot" [24]. It includes functionalities as message passing between processes, tools and libraries for writing code as well as implementation for commonly used functionalities. ROS is a framework for distributed processes and makes it easy to design, integrate and communicate between the individual processes. Therefore it is suitable for large systems such as swarm applications.

## 2.3 Control architecture

Two different control architectures were used to control the drones. Schematics of the architectures are shown in Figure 2.3, and Figure 2.4. In both cases, there is a Kalman filter on board the drone which estimates position, velocity, acceleration, attitude and attitude rate of the drone based on sensor measurements. The estimations are used both locally in the feedback loops on the drones, and for navigation and control on the PC. All communication between the drones and the PC are by radio. The second control architecture has a linear quadratic regulator (LQR) controller implemented to output desired attitude commands to improve upon the PID controller in the first control architecture. A more detailed description of the controller is given in Chapter 3.

### 2.3.1 Control architecture 1

In the first control architecture, velocity commands are sent to the drones, and position estimates are collected from the drones. This control architecture is used for its simplicity in solution "APF+Leader-follower+PID+CBF", described in Chapter 7.

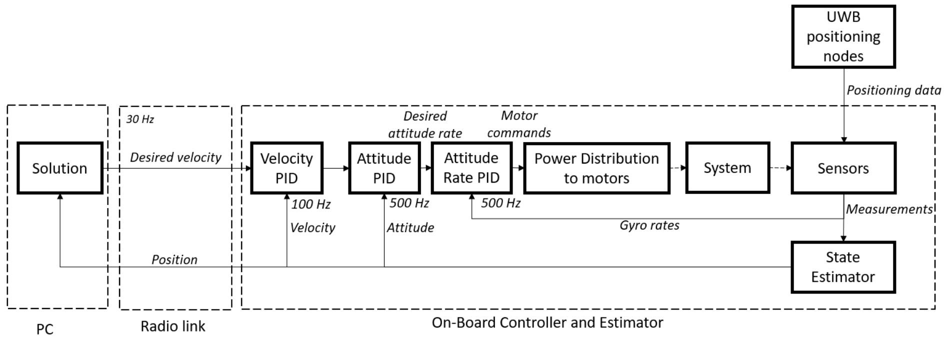


***Figure 2.3:*** *Schematics of control architecture 1. Velocity commands are sent to the drones, and position estimates are collected from the drones. In the figure, "Solution" refers to the solution APF+Leader-follower+PID+CBF, described in Chapter 7.*

### 2.3.2 Control architecture 2

This slightly more advanced control architecture uses attitude and thrust commands to the drones, and collects an estimate of the position, velocity and attitude. This control architecture is used in solution "APF+Leader-follower+LQR+ECBF" as well as the motion planning solutions, described in Chapter 7. Compared to control architecture 1, the velocity PID on the onboard controller is bypassed and more information about the state of a drone is collected.

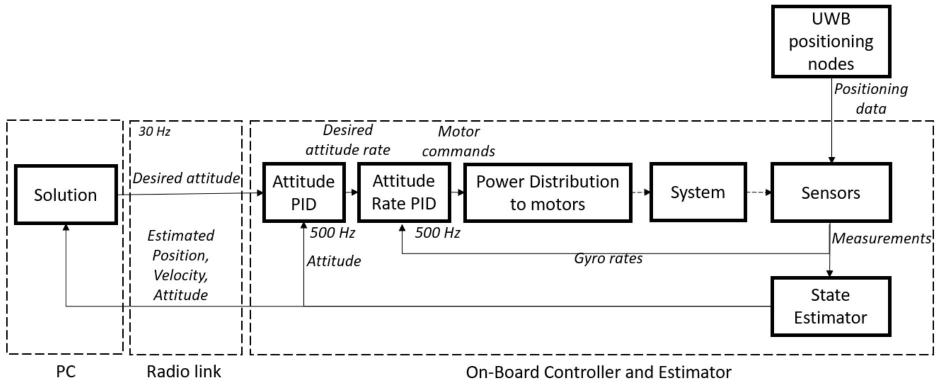


***Figure 2.4:*** *Schematics of control architecture 2. Attitude and thrust commands are sent to the drones, and an estimate of the position, velocity and attitude are collected from the drones. In the figure, "Solution" refers to either the solution APF+Leader-follower+LQR+ECBF or one of the two motion planning solutions, described in Chapter 7.*

## 2.4 Model of the system

The solutions to the problem formulation will build upon two different linear motion models. The first model is the single integrator model, used for its simplicity. The second model is a quadcopter model which capture more of the true system dynamics, at the expense of being more complex. Since the problem is simplified to two dimensions, only movement along the $x$ and $y$ axes is considered in the models.

### 2.4.1 Single integrator model

For the first control architecture, a single integrator model was used where it is assumed that the drone velocity can be controlled instantly with velocity commands. As the Crazyflies were solely controlled by cascaded PIDs the model was only needed to formulate a control barrier function. The control architecture can be seen in Figure 2.3 and the state-space model is:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} u, \; x = \begin{bmatrix} P_x \\ P_y \end{bmatrix}, \; u = \begin{bmatrix} V_x \\ V_y \end{bmatrix}, \tag{2.1}$$

where $P$ is the position of a drone and $V$ is the velocity of a drone.

### 2.4.2 Quadcopter model

In the second control architecture from Figure 2.4, where pitch, roll and thrust were the control signals, a more complex model was used that also took into account the dynamics of the system. The dynamics in $z$ (vertical axis) was not modeled, as the thrust was controlled by a PID controller. However, as the pitch and roll commands were controlled by a LQR controller, the model had to be sufficiently precise. The model that was used was inspired from [18] where they used a similar model to control a quadcopter with linear MPC. Some slight changes were made to the model to better fit the implementation in this thesis. The model can be seen below.

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \\ \dot{x}_6 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -\lambda & 0 & g & 0 \\ 0 & 0 & 0 & -\lambda & 0 & -g \\ 0 & 0 & 0 & 0 & -\frac{1}{\tau_\phi} & 0 \\ 0 & 0 & 0 & 0 & 0 & -\frac{1}{\tau_\theta} \end{bmatrix} x + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \frac{K_\phi}{\tau_\phi} & 0 \\ 0 & \frac{K_\theta}{\tau_\theta} \end{bmatrix} u, \; x = \begin{bmatrix} P_x \\ P_y \\ V_x \\ V_y \\ \phi \\ \theta \end{bmatrix}, \; u = \begin{bmatrix} \phi_{cmd} \\ \theta_{cmd} \end{bmatrix} \tag{2.2}$$

The derivative of $P_x$ and $P_y$ is self explanatory from the state-space model. The derivative of $V_x$ and $V_y$ is under three assumptions. The first is that the acceleration in $z$ is zero, in other words that the Crazyflie is hovering. This applies to our system since the Crazyflies have a constant $z$ position. The second assumption is that the yaw angle is zero which means that the body frame is in line with the global frame according to Figure 2.5.

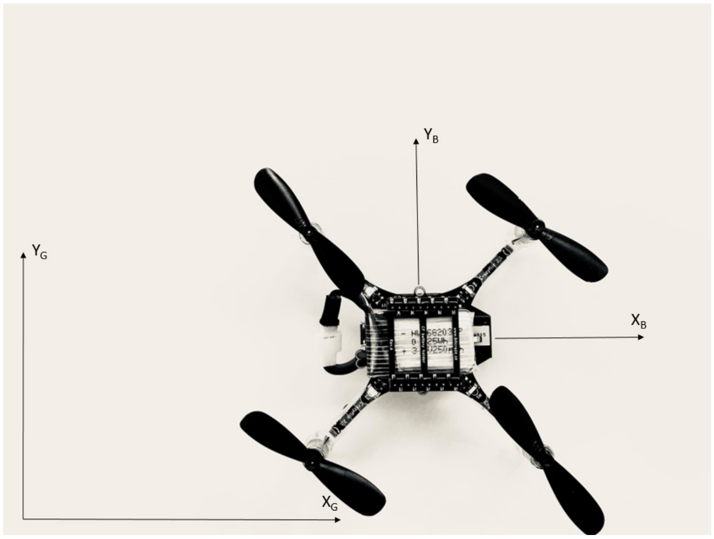


***Figure 2.5:*** *The global coordinate system G and the body coordinate system B is assumed to be aligned.*

To achieve this alignment two things are done. First the Crazyflie is lined up with the global coordinate system as close as possible before start; secondly the true angle between the global coordinate frame and the body coordinate frame is estimated during flight and accounted for before sending the attitude commands. The estimation of the angle, called global yaw angle, is described in Section 2.5.4. The third assumption is that pitch and roll are small enough that cosines of the angles can be approximated as one and sines of the angles can be approximated as the angles in radians. This is the well-known small angle approximation. The derivative of $V_x$ and $V_y$ was found by modeling the forces on the Crazyflie during flight and deriving the equilibrium equations in $x$, $y$ and $z$.

From Figure 2.6 the equilibrium equations in $x$ and $z$ are:

$$F_z = F_t\cos(\phi) - mg \approx F_t - mg = 0 \implies F_t = mg \tag{2.3}$$

$$F_x = F_t\sin(\phi) - F_d \approx F_t\phi - \lambda v_x \tag{2.4}$$

The equations are under the assumptions of no acceleration in $z$, small angles and that the body coordinate system is aligned with the global coordinate system. Since $F_x = ma_x$, the acceleration in $x$ is:

$$F_x = ma_x = mg\phi - \lambda v_x \implies a_x = g\phi - \frac{\lambda v_x}{m} \implies a_x = g\phi - \lambda v_x. \tag{2.5}$$

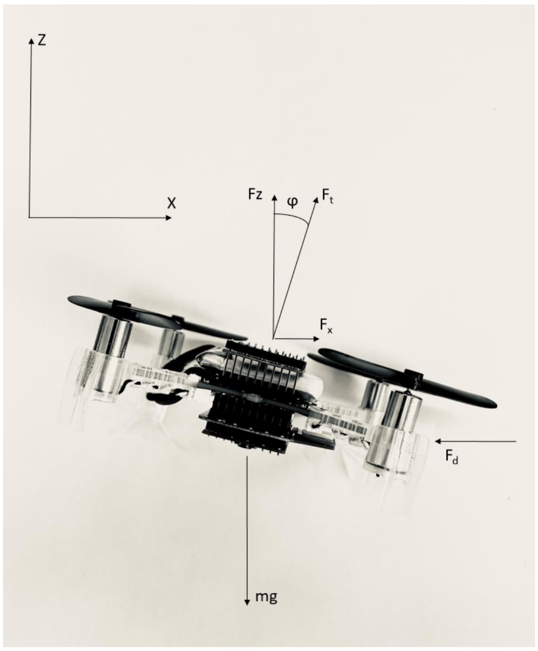


***Figure 2.6:*** *The modeled forces on the crazyflie during flight. $F_t$ is the total force from the rotors, $F_d$ is the drag force and $mg$ is the gravitational force.*

Because both $\lambda$ and $m$ are constants, $\frac{\lambda}{m}$ is seen as one constant, $\lambda$. Since the Crazyflie is symmetrical, the same equations for $x$ also apply in $y$ by exchanging the pitch angle to the roll angle. This yields

$$a_y = g\theta - \lambda v_y. \tag{2.6}$$

$\lambda$ is a parameter to describe air resistance. From aerodynamics it is known that the air resistance correlates to the velocity squared, but since a linear model is considered this is not applicable. Modelling the air resistance as $\lambda v$ might result in a significant model error, therefore the influence $\lambda$ has on the model is explored in Section 2.5. Lastly, the derivative of pitch and roll are estimated by fitting a first order system to a step response from commanded pitch/roll to actual pitch/roll. This was in contrast to [18] who suggested a second order system to describe the dynamics between actual pitch/roll and commanded pitch/roll. See Section 2.5 for more details regarding why a first order system was chosen and how the drag coefficient $\lambda$, the first order gains $K_{\phi/\theta}$ and the first order time constants $\tau_{\phi/\theta}$ were found.

## 2.5 System identification

In this section the unknown parameters from the model in (2.2) are estimated. Another parameter that is estimated, which is not part of the model, is the time delay between sent commands and received commands.

### 2.5.1 Identification of $\tau_\phi$,$\tau_\theta$ $K_\phi$ and $K_\theta$

In this section the parameters $\tau_\phi$,$\tau_\theta$ $K_\phi$ and $K_\theta$ in the state space model (2.2) are estimated. Because of symmetry it can be assumed that $\tau_\phi = \tau_\theta$ and $K_\phi = K_\theta$.

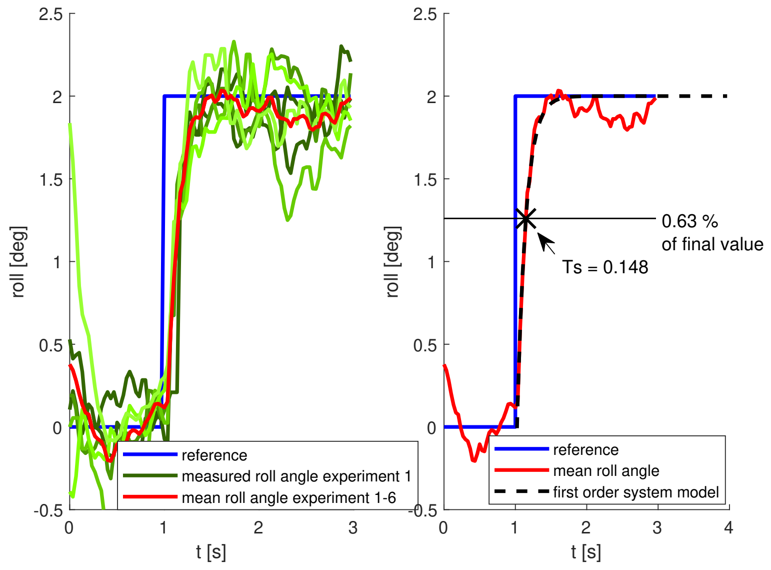


***Figure 2.7:*** *Step response for identification of the time constant $T_s$. The model is a first order system where the time constant is the time to reach 63% of the final value. The green lines in the figure are the measured pitch angle for each individual experiment 1-6.*

The unit step response from Figure 2.7 is used to estimate the parameters $\tau_\phi$

and $K_\phi$ in (2.2). Row 5 in (2.2) yields

$$\dot{\phi} = -\frac{1}{\tau_\phi}\phi + \frac{K_\phi}{\tau_\phi}\phi_{cmd}, \tag{2.7}$$

which can be rewritten in the Laplace-domain as

$$\phi s = -\frac{1}{\tau_\phi}\phi + \frac{K_\phi}{\tau_\phi}\phi_{cmd} \Leftrightarrow \phi = \frac{K_\phi}{1+\tau_\phi s}\phi_{cmd}. \tag{2.8}$$

This is a first order system, with the step response

$$\phi(t) = K_\phi(1 - e^{-\frac{1}{\tau_\phi}t}). \tag{2.9}$$

Evaluating as $t \to \infty$ gives

$$\lim_{t\to\infty} \phi(t) = \lim_{t\to\infty} K_\phi(1 - e^{-t})\phi_{cmd}(t) = \lim_{t\to\infty} K_\phi \phi_{cmd}(t). \tag{2.10}$$

Hence we can identify $K_\phi$ as the final value in Figure 2.7, which is estimated to be the same as the reference value. Thus $K_\phi = 1$.

Evaluating in $t = \tau_\phi$ gives

$$\phi(\tau_\phi) = (1 - e^{-1})\phi_{cmd}(\tau_\phi) \approx 0.63\phi_{cmd}(\tau_\phi). \tag{2.11}$$

Hence we can identify $\tau_\phi$ from Figure 2.7 as the time to reach 63% of the final value. Thus $\tau_\phi \approx 0.148\ s$.

### 2.5.2 Identification of $\lambda$

In this section, the parameter $\lambda$ from the state space model (2.2) is estimated.

The third equation in (2.2) yields

$$\dot{x}_3 = -\lambda V_x + g\phi, \tag{2.12}$$

where $\dot{x}_3$ is the acceleration in the x-direction, denoted $a_x$. The term $-\lambda V_x$ should capture the dynamics of air drag. This can be compared with well known equations for modelling air drag, where Newton's second law

$$F = ma \tag{2.13}$$

and the drag

$$F_D = \frac{1}{2}C_d\rho A V^2, \tag{2.14}$$

gives the deceleration term

$$a_D = \frac{1}{m}C_d\rho A V^2 = CV^2, \tag{2.15}$$

where $C$ is a constant. By comparing the $-\lambda V$ term from (2.12) with $a_D$ from (2.15), $\lambda$ is expected to have the following behaviour

$$\lambda(V) = CV, \tag{2.16}$$

which means that $\lambda$ will vary linearly with the velocity. This relation will now be investigated using measured flight data.

Flight data was generated using a sine wave reference trajectory with increasing frequency, shown in Figure 2.8. The trajectory is designed to capture the dynamics at both low and high velocities.

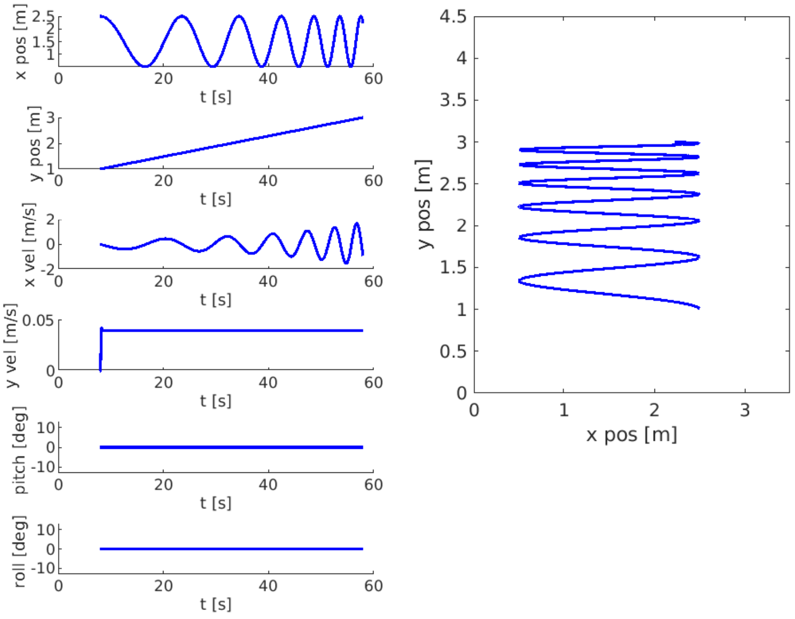


*Figure 2.8: Reference trajectory for $\lambda$ identification. Sine wave with increasing frequency and slow constant velocity in the y-direction.*

Using the model in (2.2) and flight data measurements of the velocity $V_k$ and pitch $\phi_k$, a prediction of the velocity for the next time step $\hat{V}_{k+1}(V_k, \phi_k, \lambda)$ can be made. The quality of the prediction can be evaluated using the mean squared error (MSE)

$$\text{MSE} = \frac{1}{n}\sum_{k=1}^{n}(V_k - \hat{V}_k)^2, \tag{2.17}$$

which is the average squared difference between the estimated values and the actual value. The MSE evaluated for different $\lambda$ is shown in Figure 2.9.

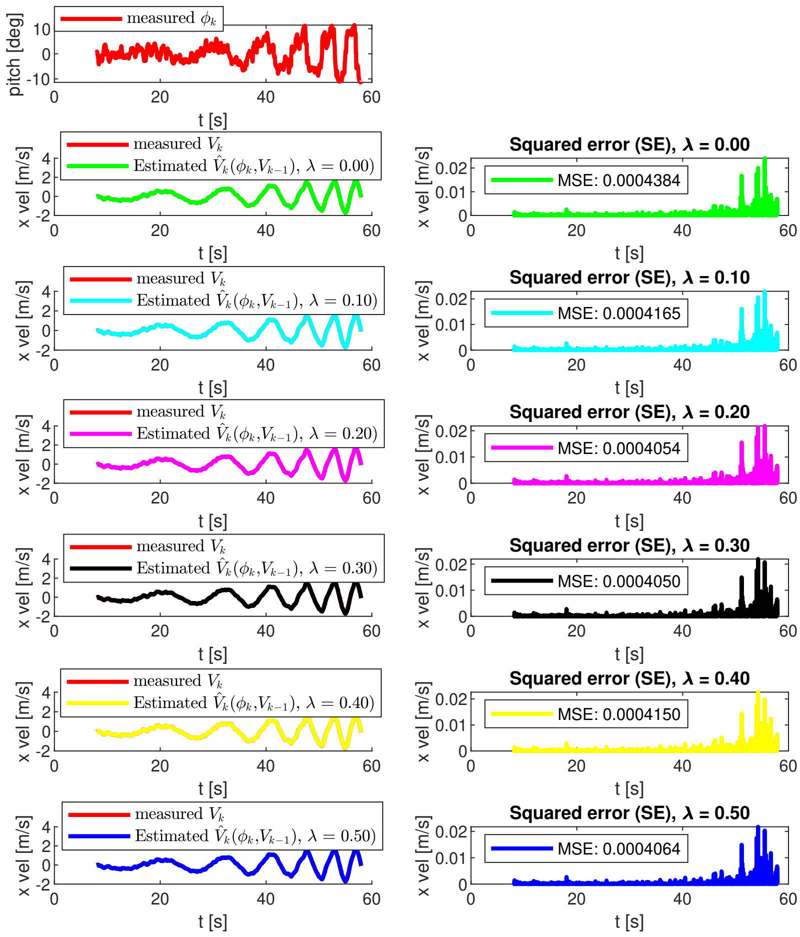


***Figure 2.9:*** *Predicted velocity* $\hat{V}_{k+1}(V_k, \phi_k, \lambda)$*, for* $\lambda = 0.0, 0.1, ..., 0.5.$

Now finding the best constant $\lambda$ can be formulated as finding the argument that minimizes the MSE

$$\lambda^* = \arg\min_{\lambda} \text{MSE}. \tag{2.18}$$

Resulting in a minimum mean square error (MMSE) estimator $\hat{V}_{k+1}(V_k, \phi_k, \lambda^*)$. In Figure 2.10 a minimum is seen for $\lambda = 0.25$.

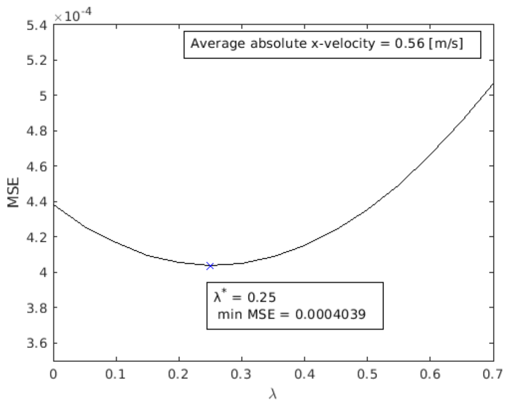


***Figure 2.10:*** *Finding the optimal constant $\lambda^*$ for the given trajectory.*

The trajectory has an average absolute x-velocity $|\bar{V}_x|$ of 0.56 $m/s$. Since the velocity varies over the trajectory, the minimization can be made on a shorter time span of the trajectory, resulting in the optimal $\lambda$ for the average velocity of that time span, $\lambda^*(V)$. The optimal $\lambda$ for 11 different time spans for the trajectory shown in Figure 2.8 are plotted as blue crosses in Figure 2.11.

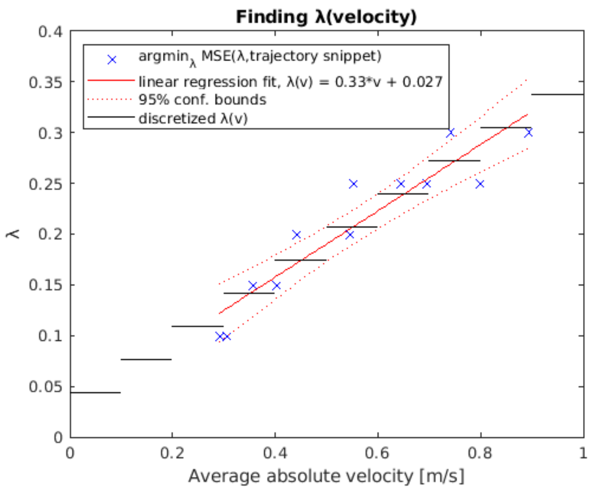


***Figure 2.11:*** *Finding the optimal $\lambda$ on shorter time spans of the trajectory with different average velocities. A linear model $\lambda(V)$ (red) can be fitted to the data points (blue), and discretized (black).*

As expected from (2.16), $\lambda$ can be represented as a linear function of the velocity:

$$\lambda(V) = 0.33V + 0.027\ . \tag{2.19}$$

Since the model is used to calculate the LQR gain matrix $K$, described in Section 3.3, the best solution would be to evaluate $\lambda(V)$ in every iteration and then calculate $K$. However, to reduce the complexity of the implementation, $\lambda(V)$ can be discretized for some set of velocities, and a corresponding set of gain matrices $K(V)$ can be calculated in advance. The discretized $\lambda(V)$, shown in Figure 2.11, is rounded up or down to the nearest discretized step, and saturated for V > 0.9, described mathematically as

$$\lambda_d(V) = \begin{cases} \lambda(0.1k + 0.05),\ \ 0.1k < V \leq 0.1(k+1) & ,k = 0,\ \dots\ ,\ 9 \\ \lambda(0.95),\ \ V > 0.9\ . \end{cases} \tag{2.20}$$

For each velocity there will be a corresponding LQR gain matrix

$$K = K(\lambda_d(V)). \tag{2.21}$$

In the quadcopter model, the states of the x and y directions are independent, which means that the first and second row of the $K$ matrix can be separated. The first row of the $K$ matrix will be based on the x-velocity, and the second row will be based on the y-velocity

$$K(V) = \begin{bmatrix} K_{(1,*)}(V_x) \\ K_{(2,*)}(V_y) \end{bmatrix}. \tag{2.22}$$

In Table 2.1 the quality of some predictors $\hat{V}$ using different implementations of $\lambda$ is compared. The $\lambda_d(V_{MA})$ predictor uses the moving average (MA) velocity of the last 7 seconds, acting as a low-pass filter on the velocity.

***Table 2.1:*** *Comparison of predictors $\hat{V}$ using different selections of $\lambda$.*

| $\lambda$ | MSE $10^{-3}$ | Relative MSE change compared to MSE($\lambda = 0.25$) |
|---|---|---|
| $\lambda = 0.0$ | 0.4384 | +8.54 % |
| $\lambda = 0.5$ | 0.4355 | +7.82 % |
| $\lambda = 0.1$ | 0.4165 | +3.12 % |
| $\lambda = 0.4$ | 0.4150 | +2.74 % |
| $\lambda_d(V)$ | 0.4065 | +0.64 % |
| $\lambda = 0.2$ | 0.4054 | +0.37 % |
| $\lambda = 0.3$ | 0.4050 | +0.27 % |
| $\lambda = 0.25$ | 0.4039 | 0.0 % |
| $\lambda_d(V_{MA})$ | 0.4036 | -0.07 % |

The quality of the predictors is relatively similar for fixed $\lambda$ between 0.2 and 0.3, and the functions $\lambda(V)$, with a relative difference of less than 1%. But a large deviation in $\lambda$ from the optimal value results in severe prediction errors. Using a fixed $\lambda$ of 0.25 is arguably good enough for most trajectories, but using the function $\lambda(V)$ could be more accurate where the average velocity is very high or low, where $\lambda = 0.25$ is deviating much from the optimal $\lambda$ value. Using $\lambda_d(V_{MA})$ the performance is marginally better compared to $\lambda_d(V)$. Since the optimal lambda was generated using the average velocity of a small part of the trajectory, the model is slightly better fitted to the moving average than the instantaneous velocity.

### 2.5.3 Time delay

There are two perspectives on what the time delay of the system is. The first is the time it takes from sending commands until the drone is receiving the commands, and the second one is the time it takes from the drone sending its current states until the drone is receiving the commands. In the perspective of the controller, the second one is more interesting since the control signal from the state feedback controller is based on the latest update of the state. If there is a significant time delay between the time the drone sends its current state until the time the drone receives the commands, the current state would be different in comparison to the state that the control signal is based on. However, this time delay is difficult to measure, which leads us to use the first time delay mentioned. This time delay can be known by measuring the time delay from sent commands until the system reacts. This is done by a simple step response which can be found above in Figure 2.7. A close up of the step response shows that this time delay is approximately 33 ms.

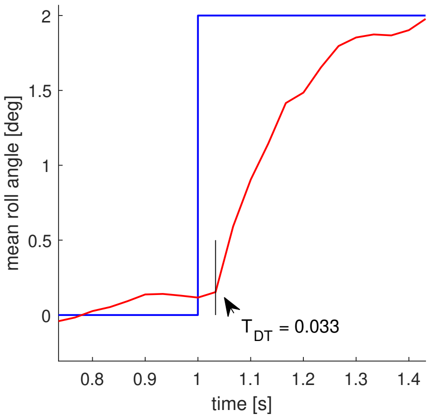


***Figure 2.12:*** *Identification of time delay $T_{DT}$*

### 2.5.4 Global yaw angle

As stated in Section 2.4 an assumption about the system is that the body frame is aligned with the global frame. Even though each drone is aligned with the global frame before start a small error can easily occur. This error in angle is, in this thesis, called global yaw angle and is estimated during the flight. If the drone is aligned with the global frame, then the global acceleration vector of the drone and the body attitude vector should align. However, if not, the angle between global acceleration vector and the body attitude vector is the measured global yaw angle.

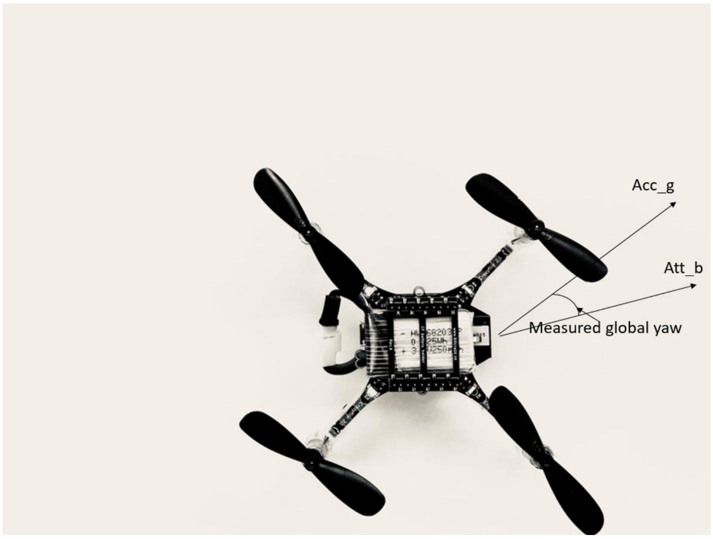


***Figure 2.13:*** *Measurement of global yaw angle. Acc_g is the acceleration vector in the global frame and att_b is the attitude vector in the body frame.*

The estimated global yaw angle is updated with the error in measurement times a constant $K$ as

$$yaw_{estimated} = yaw_{estimated} + K(yaw_{measured} - yaw_{estimated}). \tag{2.23}$$

If the estimation of the global yaw angle was critical, a Kalman filter would provide a better estimation. Because of simplicity this filter was used, with a conservative $K$. The global yaw angle is used to rotate the desired attitude command vector with the rotation matrix:

$$\begin{bmatrix} \cos(yaw) & \text{-}\sin(yaw) \\ \sin(yaw) & \cos(yaw) \end{bmatrix}.$$

# 3 Control

As described in Section 2.3, the onboard controller of the drone is using cascade control with PIDs that either goes from global velocities to motor speed or from attitude and thrust in its body frame to motor speed. In the second control architecture an LQR controller is implemented that outputs desired attitude in the body frame. The LQR controller replaces one PID controller from the first control architecture. This section will begin by describing how a PID controller as well as cascade control works and secondly, in detail, present the implemented LQR controller. Finally the implemented formation control is described.

## 3.1 PID

A PID is the most common controller. Its advantages lies in that there is no need of any knowledge of the system to implement a PID controller, even though it can be advantageous when, for example, analyzing stability. A PID controller has three parts: a proportional (P) part, an integral (I) part and a derivative (D) part. The mathematical expression for the resulting control law is

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{d}{dt} e(t), \tag{3.1}$$

where the error $e(t)$ is the difference between a measured output signal and a reference signal, and $u(t)$ is the control signal used as input to the system. A PID controller aims to make the error $e(t)$ as small as possible. The proportional part is the base of the PID controller and higher values of $K_p$ lead to a faster response to an error however, sometimes this can lead to oscillatory behaviour. Therefore a derivative part is added that aims to slow down the rate of change of the error. The integral part is added to remove steady-state errors [10].

## 3.2 Cascade control

Cascade control is simply having several controllers in a serial connection where the reference signal to each controller is the output signal from the previous controller. This is used to maximize the information about the system and is used in the onboard controller on the Crazyflie (see Figures 2.3 and 2.4). Instead of having one PID controller to go from desired global velocities to power distribution to each motor, there is a cascade of PIDs which utilizes information about estimated velocities, attitude and attitude rate, which otherwise would not have been used. The idea is that since more information about the system is used, the controller will be better [10].

## 3.3 Linear Quadratic Regulator (LQR)

A linear quadratic regulator (LQR) is an optimal controller based on state feedback. The control signal $u(t)$ depends on the current states $x$ and a gain matrix $K$ that describes how each state contributes to the control signal. In other words the control law is given by

$$u(t) = -Kx. \tag{3.2}$$

In the more common case when there is a state reference, the control signal is given by

$$u(t) = K(x_{ref} - x) = Ke. \tag{3.3}$$

The LQR controller is based upon the following optimization problem:

$$\begin{aligned} \min_{u} \quad & \frac{1}{2}\int_{0}^{\infty} e^T Q e + u^T R u \\ \text{s.t.} \quad & \dot{x} = Ax + Bu, \end{aligned} \tag{3.4}$$

where $Q$ is a real positive-definite or positive-semidefinite symmetric matrix and $R$ is a positive-definite symmetric matrix. $A$ and $B$ describe the linear dynamics of the system. The gain matrix $K$ is given by

$$K = R^{-1} B^T P, \tag{3.5}$$

where P is the real symmetric matrix solution to the the Riccati equation

$$A^T P + PA - PBR^{-1} B^T P + Q = 0. \tag{3.6}$$

LQR gives an optimal gain matrix based on a model of the system and a cost function. The cost function is determined by the state error as well as the size of the control signals subjected to weight matrices given by the user. The first weight matrix $Q$ determines how much impact the state error will have on the cost function and the second weight matrix $R$ determines how much impact the size of the control signal will have on the cost function. For example, to avoid large control signals the user can increase the element values of the $R$-matrix.

Tuning of the weight matrices $Q$ and $R$ are described in Section 3.3.2. A precise model of the system is important to achieving good results from the LQR problem.

## 3.3.1 LQR with time delay

To compensate for the time delay, from the time the commands are sent to the time the commands are received, an extension to LQR that is based on Smith prediction theory can be implemented. With time delay compensation the control signal is given by [29]

$$u_k = -K(F^d x_k + \sum_{i=1}^{d} F^{i-1} G u_{k-1}), \tag{3.7}$$

where $K$ is the gain matrix obtained from the LQR problem and $F$,$G$ is the matrices that describes the system in discrete time as

$$x_{k+1} = F x_k + G u_k. \tag{3.8}$$

To go from continuous time to discrete time the F and G matrices are given by [9]

$$\begin{aligned} F &= e^{AT_s} \\ G &= \int_0^{T_s} e^{At} B dt, \end{aligned} \tag{3.9}$$

where $A$ and $B$ describe a continuous time system and $T_s$ is the sampling time. By using a first order Taylor expansion the following discrete time matrices were found:

$$\begin{aligned} F &= I + AT_s \\ G &= (IT_s + \frac{AT_s^2}{2})B. \end{aligned} \tag{3.10}$$

More details regarding the time delay of the system can be found in Section 2.5.

## 3.3.2 Choosing weight matrices

The weight matrices of the LQR controller, $Q$ and $R$, where tuned with a trial and error approach, where the aim is to design an LQR controller to perform an efficient control with a minimum error in desired position.

The LQR state feedback controller is designed to stabilize the system and minimize the cost function from (3.4) where $Q$ and $R$ can be defined as diagonal matrices

$$Q = \begin{bmatrix} q_1 & 0 & \dots & 0 \\ 0 & q_2 & 0 & 0 \\ \vdots & 0 & \ddots & 0 \\ 0 & 0 & 0 & q_6 \end{bmatrix},$$

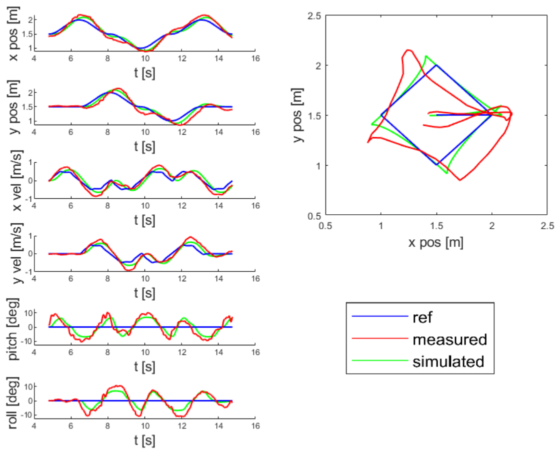
$$R = \begin{bmatrix} r_1 & 0 \\ 0 & r_2 \end{bmatrix}.$$

Here $q_1$, ... , $q_6$ corresponds to the cost of deviation from the desired reference state $x_{ref1}$, ... , $x_{ref6}$, and $r_1$, $r_2$ corresponds to the cost of large attitude control signals $u_1$ and $u_2$.

To achieve low error in desired position, an initial approach is to use large $q_1$ and $q_2$ to minimize the position error, and small $r_1$ and $r_2$ to not restrict attitude controls. This works very well in ideal conditions, but experimentally, noise in the position estimate results in jerky behaviour as small deviations in position results in large attitude controls. To handle this, $r_1$, $r_2$ are increased until the jerk is at an acceptable level. For trajectories with fast maneuvers, it can be beneficial to use the velocity profile of the reference, i.e., increasing $q_3$, $q_4$. The difference of introducing cost on the velocity reference can be seen by comparing Figure 3.1 and Figure 3.2. However, for relatively slow trajectories, introducing cost on the velocity does not make any noticeable difference and the tracking errors are dominated by disturbances, see Figure 3.3 and Figure 3.4. The final values of the tuning parameters are:

$$Q = \begin{bmatrix} 10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 & 0 & 0 \\ 0 & 0 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 9 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1\text{e-}10 & 0 \\ 0 & 0 & 0 & & 0 & 1\text{e-}10 \end{bmatrix},$$

$$R = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}.$$



0

***Figure 3.1:*** *A fast trajectory reference with cost on position errors, using tuning parameters* $q_1 = 10$, $q_2 = 10$, $r_1 = 10$, $r_2 = 10$ *. Fast reference changes results in overshoot.*

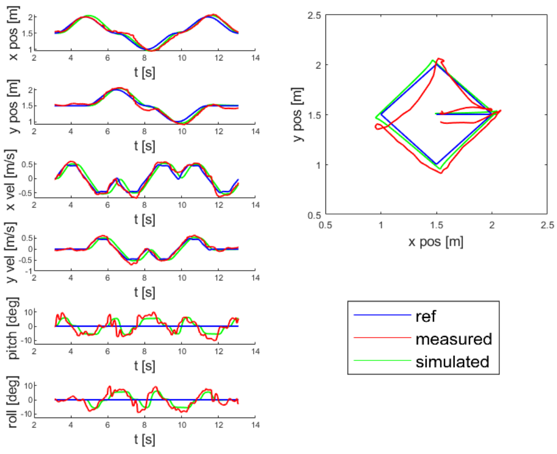


***Figure 3.2:*** *Fast trajectory with cost on position and velocity errors, using* $q_1 = 10$, $q_2 = 10$, $q_3 = 9$, $q_4 = 9$, $r_1 = 10$, $r_2 = 10$. *The velocity reference reduces the overshoot compared to only position reference.*

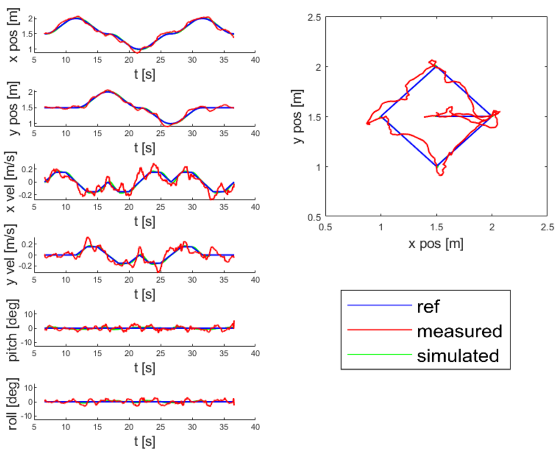


*Figure 3.3: Slow trajectory with cost on position errors, using $q_1 = 10$, $q_2 = 10$, $r_1 = 10$, $r_2 = 10$. Errors due to disturbances are predominant.*

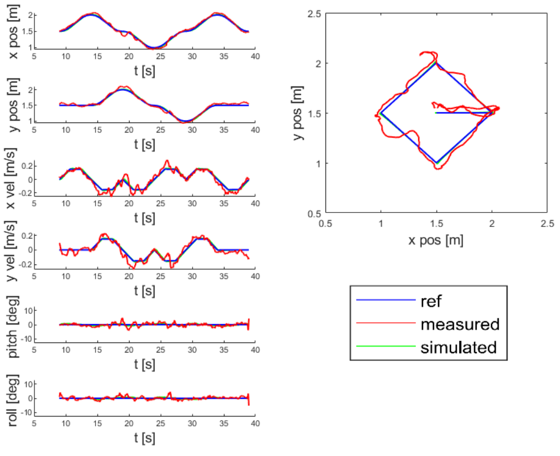


*Figure 3.4: Slow trajectory with cost on position and velocity errors, using $q_1 = 10$, $q_2 = 10$, $q_3 = 9$, $q_4 = 9$, $r_1 = 10$, $r_2 = 10$. The introduced cost on velocity does not improve the position error for a slow trajectory. Errors due to disturbances are predominant.*

## 3.4 Formation control

Formation control of drone swarms are typically divided into three approaches: leader-follower approaches, behavior-based approaches and virtual structures. The leader follower method designates one drone as the leader which will fly along a predefined trajectory and the other drones as followers which tries to maintain their relative position and velocity to the leader and the neighbouring drones. Simplicity in analysis and scalability are the main pros, while a limitation is that the formation is dependent on a single drone [27] [22]. This thesis will focus on this approach.

Behavior-based methods weigh behaviors such as keeping a certain formation, obstacle avoidance and trajectory following to determine a final control of each drone. This approach was developed with inspiration from nature where flocking can be beneficial to certain animals if it is to avoid predators or find prey. It can also be seen in birds during long flights to minimize the air resistance. One advantage is that the formation control is decentralized and communication between drones can be kept to a minimal. However, it is difficult to analyse convergence of the formation mathematically [4] [16].

The last method is called virtual structure which sees the drone swarm as a single entity. In rigid body motion, all points maintain their exact relative position due to fixed constraints. This concept is used in virtual structure methods to force the drones to maintain these fixed points in the structure. Each drone follows a virtual leader to keep a fixed formation with high precision. However the virtual leader can not make any decisions by itself which is the main disadvantage [25] [20].

### 3.4.1 Leader follower displacement control

A typical displacement-based formation control imposes the following requirements on agents:

- Each agent needs a local coordinate system which is aligned to a global coordinate system and sensing capability of the relative displacements of their neighbors with respect to the global coordinate system.
- A topology graph that defines the desired displacements from any agent to the other, which can be characterized by either connectedness or existence of a spanning tree. [17]

#### Control law using a single integrator model

Consider the single integrator model where the position $p \in \mathbb{R}^n$ of drone $i$ is controlled via the control signal $u \in \mathbb{R}^n$:

$$\dot{p}_i = u_i, \quad i = 1, ..., N. \tag{3.11}$$

If the desired relative displacements $p_i^* - p_j^*$ between agent i and j are given, and the objective of the agents is to satisfy the constraint

$$p_i - p_j = p_i^* - p_j^*, \tag{3.12}$$

then the formation control can be solved by implementing the control law [17]

$$u_i = k_p \sum_{j \in \mathcal{N}_i} (p_j - p_i - (p_j^* - p_i^*)). \tag{3.13}$$

## Implementation

The desired formation is defined by specifying the displacement of each drone to an origin, as illustrated in Figure 3.5. The origin can be chosen arbitrarily as the coordinates are only used for calculating the relative displacement between two drones.

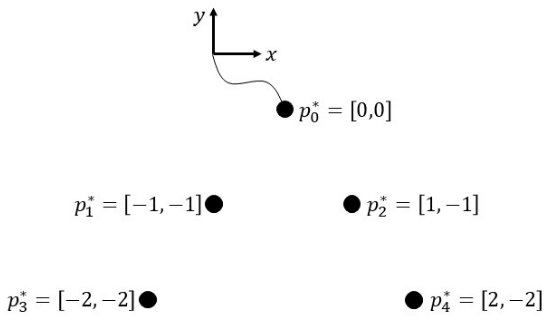


***Figure 3.5:*** *The desired formation is defined by specifying the displacement of each drone to an origin.*

The leader of the formation will guide the formation using an external control signal $u_{ref}$, and the followers will use the displacement control law in (3.13) with feed-forward of the leader control, resulting in the total control

$$u_i = k_p \sum_{j \in \mathcal{N}_i} (p_j - p_i - (p_j^* - p_i^*)) + u_{ref}. \tag{3.14}$$

The feed-forward term keeps the movement of the followers synchronized with the leader, and the displacement control will minimize the formation displacement errors that occur due to initial offsets and disturbances.

# 4

# Safety

Given a nominal control signal, the last step before sending commands to the drones is to ensure the safety of the control signal. This is done by implementing a control barrier function (CBF). As stated previously, two different control signals were used. The drones were either controlled by velocity commands or by attitude commands. For the latter case an extension to CBF were used called exponential control barrier function (ECBF).

## 4.1 Control Barrier Function (CBF)

This section will begin by introducing the general definition of a control barrier function and secondly show the implementation that was used in this work.

### 4.1.1 Definition

Control barrier function are used to guarantee safety for a system. The general idea is to create a safe set of states and use CBF to ensure that the set is forward invariant. That is, if the system starts in the safe set, it will always stay within the safe set. To give a more precise definition of a CBF, consider a general dynamical system on control affine form

$$\dot{x} = f(x) + g(x)u, \tag{4.1}$$

where the state $x \in \mathbb{R}^n$ and $u \in U$, $f$ and $g$ are locally Lipschitz continous. A locally Lipschitz continous function is a more strict form of a continuous function which also is limited to how fast it can change. Now, consider a continuously differentiable safety function $h : \mathbb{R}^n \to \mathbb{R}$ : defining a safe set $S$ as [21]

$$S = \{x \in \mathbb{R}^n \ : \ h(x) \geq 0\}. \tag{4.2}$$

There is one more definition before a control barrier function can be formally defined. That is a continuous function $\kappa : (-b, a) \rightarrow \mathbb{R}$ for some $a, b > 0$. This is called an extended class-$K$ function if it is strictly increasing and $\kappa(0) = 0$.

**Definition 4.1.** $h$ is a CBF if there exists a extended class-$K$ function $\kappa$ such that

$$\sup_{u \in U} \dot{h} + \kappa(h) \geq 0$$

This will ensure forward invariance of the set $S$ in (4.2) [26]. The derivative of $h$ is given by [8][26]

$$\dot{h} = \frac{dh}{dt} = \frac{\partial h}{\partial x}\dot{x} = \frac{\partial h}{\partial x}(f(x) + g(x)u). \tag{4.3}$$

## 4.1.2 Implementation

A control barrier function can be used when using velocity commands, since the derivative of the $h$-function is dependent on the control signal, see (4.5). The model of the dynamics of the system was a simple single-integrator model which can be seen in (2.1).

The safe set $S$ was set to be all states where the distance to neighbouring drones was greater or equal to a safety distance $D_s$ in the x-y plane. A CBF was implemented to make sure that if a drone starts in the safe set, i.e., if the distance to neighbouring drones is greater or equal to the safety distance $D_s$, the drone will always remain in the safe set. A CBF is zero at the boundary of the set $S$ and greater than zero for all safe states. Therefore an intuitive $h$ is

$$h = (x - x_n)^2 + (y - y_n)^2 - D_s^2, \tag{4.4}$$

where $x$, $y$ is the global position coordinates of a drone and $x_n$, $y_n$ is the global position coordinates of a neighbouring drone. This function will be equal to zero when the drones are exactly at the safety distance $D_s$ from each other and greater than zero when the distance is greater than $D_s$. From (4.3) the time dependent derivative of $h$ is:

$$\dot{h} = 2(x - x_n)v_x + 2(y - y_n)v_y, \tag{4.5}$$

where $v_x$ and $v_y$ are the control signals. The extended class-$K$ function $\kappa(h)$ was set as:

$$\kappa(h) = \alpha h, \tag{4.6}$$

which follows the definition of an extended class-$K$ function stated above. A lower value of $\alpha$ gives a more strict control signal to the point that a value of zero means that any negative derivative of the $h$-function is forbidden. With $\alpha$ set to zero the CBF does not allow the drones to move toward each other, which obviously affects the performance greatly. The value of $\alpha$ was chosen considering both safety and performance.

To implement the CBF, a quadratic programming problem was formulated with the inequality constraints from Definition 4.1. The penalty function was to minimize the quadratic error between a nominal control signal $u_{nom}$ and the control signal $u$. The nominal control signal was based on algorithms for formation control and guidance/path planning, described in Chapter 5. The quadratic programming problem was formulated as

$$\begin{gathered} u_c = \underset{u}{\operatorname{argmin}} (u_{nom} - u)^2 \\ \text{subject to: } \dot{h} + \alpha h \geq 0. \end{gathered}$$

## 4.2 Exponential Control Barrier Function (ECBF)

This section introduces the general definition of an exponential control barrier function and shows the implementation that was used in this work.

### 4.2.1 Definition

The definition of ECBF is similar to the definition of CBFs. However, CBFs can not be used when $\dot{h}$ does not depend on the control signal. An ECBF can be used when the second derivative is dependent on the control signal. Consider the same dynamical system, the same safe set $S$ and the same function $h$ as for the CBF.

**Definition 4.2.** The function $h$ is a ECBF if there exists a $K \in \mathbb{R}^2$ that places the poles of $\ddot{h} + K^T \begin{bmatrix} h \\ \dot{h} \end{bmatrix} = 0$ and there exists a control signal $u \in U$ that satisfies the inequality

$$\ddot{h}(t,x) + K[h(t,x)\ \dot{h}(t,x)]^T \geq 0,\ x \in S. \tag{4.7}$$

This ensures forward invariance of set S which means that the system will be safe [28].

### 4.2.2 Implementation

An ECBF was implemented when the drones was controlled by LQR. The control signal from the state feedback is attitude commands, which correlates to the acceleration of the drone. Since the derivative of $h$ does not depend on the control signal/acceleration, an ECBF was implemented. The implementation of an ECBF is approximately the same as for CBF. The same $h$ as in (4.4) and the same method to construct the problem as a quadratic programming problem were used. The differences are that another model of the system is used and that a second derivative of $h$ needs to be calculated. The state-space model from (2.2) was used with minor changes. The $h$-function depends on the position of the drones which means that the derivative will depend on the velocity of the drones and the second derivative will depend on the acceleration of the drones. In other words using an ECBF with this $h$-function requires the acceleration of the drones to be dependent on the control signal, which is not the case in the model in (2.2). A simplified model based on (2.2) was therefore created:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -\lambda & 0 \\ 0 & 0 & 0 & -\lambda \end{bmatrix} x + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ g & 0 \\ 0 & -g \end{bmatrix} u, \; x = \begin{bmatrix} P_x \\ P_y \\ V_x \\ V_y \end{bmatrix}, \; u = \begin{bmatrix} \phi_{cmd} \\ \theta_{cmd} \end{bmatrix}. \tag{4.8}$$

This model assumes that the commanded pitch and roll angles are the actual attitude for the drone. This assumption neglects the dynamics from commanded attitude to actual attitude which can be seen in the step response in Figure 2.7. This is a necessary assumption to make the second derivative of the $h$-function depend on the control signal. The derivative of the position is still assumed to be the measured velocity, which means that the derivative of $h$ is the same as from (4.5). The second time derivative of $h$ is:

$$\ddot{h} = 2((x - x_n)(-\lambda v_x + g\phi_{cmd}) + (y - y_n)(-\lambda v_y - g\theta_{cmd}) + v_x^2 + v_y^2). \tag{4.9}$$

The inequality constraints in the QP-problem is given by (4.7) with the first and second derivative of the $h$-function given above.

# 5 Guidance

This chapter describes the theory and implementations for collision avoidance and trajectory planning, which lies at the core for UAV collaborative tasks. In the civil aviation field, this is referred to as Traffic Alert and Collision Avoidance Systems (TCAS). In recent years, there has been research on collision avoidance on UAVs, such as Artificial Potential Field algorithm, ant colony algorithm, genetic algorithm, geometric optimization, colored petri net, Markov Decision Progress and the combination and optimization of these algorithms [23]. Two different solutions to the guidance problem have been explored in this thesis: Artificial Potential Field (APF) and an optimal control formulation.

## 5.1 Artificial potential fields

Artificial potential field algorithms can be used for path planning where the movement of each agent is determined by two artificial forces. The attractive force pulls the agent towards the goal, and the repulsive force pushes the agent away from obstacles in the environment, resulting in the total force

$$F_{APF} = F_{att} + F_{rep}\ . \tag{5.1}$$

### 5.1.1 Theory

The attractive potential from the goal and the repulsive potential from the obstacles introduced by Khatib [14] is given by

$$U^{att} = \frac{1}{2}\xi((x - x_g)^2 + (y - y_g)^2), \tag{5.2}$$

where $x, y$ and $x_g, y_g$ are the x- and y-coordinate of the object and the goal, respectively, and $\xi$ is the potential attraction constant. The repulsive potential from each obstacle is

$$U^{rep} = \begin{cases} \frac{1}{2}k_r(\frac{1}{\rho_O} - \frac{1}{r_O})^2 & \text{if } \rho_O \leq r_O \\ 0 & \text{if } \rho_O > r_O \end{cases} \tag{5.3}$$

where $k_r$ is the potential repulsive constant, $r_O$ is the distance from the radius of the object to the maximum radius of influence from the repulsion force, and $\rho_O$ is the distance between the agent and the obstacle, shown in Figure 5.1.

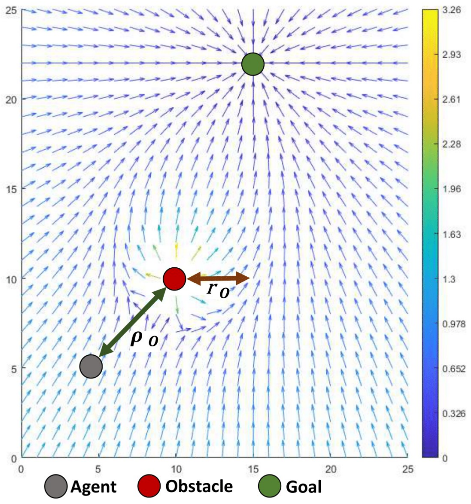


***Figure 5.1:*** *The artificial potential force field. The arrows and colors represent the direction and magnitude of the resultant force vector exerted by the field for a given point in the horizontal plane. The force vector is given by* (5.7)*,* (5.8)*,* (5.9) *and* (5.10)*.*

The force in each point in the potential field is given by the negative gradient of the potential [13]. This yields

$$F_O^{rep} = -\nabla U_{rep}(x, y) \tag{5.4}$$

with components in $x$ and $y$ that are given by

$$F_x^{rep}(x, y) = \frac{\partial U_{rep}(x, y)}{\partial x}, \tag{5.5}$$

$$F_y^{rep}(x, y) = \frac{\partial U_{rep}(x, y)}{\partial y}. \tag{5.6}$$

Inserting (5.3) into (5.5) and (5.6) gives the additional force from each obstacle

$$F_x^{rep} = \begin{cases} -k_r(1 - \frac{\rho_O}{r_O})\frac{x_{or}}{\rho_O^3} & \text{if } \rho_O \leq r_O \\ 0 & \text{if } \rho_O > r_O \end{cases} \tag{5.7}$$

$$F_y^{rep} = \begin{cases} -k_r(1 - \frac{\rho_O}{r_O})\frac{y_{or}}{\rho_O^3} & \text{if } \rho_O \leq r_O \\ 0 & \text{if } \rho_O > r_O \end{cases}. \tag{5.8}$$

The same calculation can be made for the attraction force which gives the additional force from the goal as

$$F_x^{att} = \xi(x - x_g) \tag{5.9}$$

$$F_y^{att} = \xi(y - y_g). \tag{5.10}$$

The implementation of APFs, according to [13] makes the drone move in the same direction as the resultant force vector. In other words, the resultant force vector in the potential field is used as the desired velocity for each drone.

$$v = F^{att} + \sum_{j=1}^{n} F_j^{rep} \tag{5.11}$$

where $F^{att}$ is the attractive force and $F_j^{rep}$ is the repulsive force from each obstacle.

## 5.2 Optimal motion planning

The meaning of the term *optimal motion planning* can vary depending on application and field of subject [15]. In this thesis the term is used to describe the problem of finding a feasible and optimal trajectory for one or several swarms of drones with a known initial and final state. A feasible solution means that the trajectory does not violate the system dynamics and avoids collision with obstacles. For each time step, each drone also needs to be in a safe state. A state is safe if the distance to every other drone is greater than or equal to a specified safety distance.

### 5.2.1 Problem formulation

First consider a linear dynamic model of the system

$$x_{k+1} = f(x_k, u_k) = Fx_k + Gu_k, \tag{5.12}$$

where $x_k \in \mathbb{R}^n$ and $u_k \in \mathbb{R}^m$ is the state and control input respectively. These are subject to constraints

$$x_k \in X \subseteq \mathbb{R}^n, \quad u_k \in U \subseteq \mathbb{R}^m. \tag{5.13}$$

The control signal will usually be bounded by a maximum and minimum value. The constraints on the states are usually defined by the system dynamics, initial/final constraints and collision avoidance with either static or dynamic obstacles. The states which are occupied by obstacles can be defined as

$$X_{obst}(k) = X_{obst}^s \cup X_{obst}^d(k) \subset \mathbb{R}^n, \tag{5.14}$$

where $X^s_{obst}$ are states occupied by static obstacles and $X^d_{obst}$ are states occupied by dynamic obstacles. From this definition the free state space can be written as

$$X_{free}(k) = X \setminus X_{obst}(k). \tag{5.15}$$

From these definitions, the problem formulation can be stated as an optimal control problem, i.e., compute a sequence of feasible states and control inputs $\{x_k\}_{k=0}^{L-1}$, $\{u_k\}_{k=0}^{L-1}$ that moves the system from an initial state $x_i$, to a final state $x_f$, which minimizes the performance measure $J_l$ [5]:

$$\begin{aligned} \min_{x_k{}_{k=0}^{L-1}, u_k{}_{k=0}^{L-1}} \quad & J_l \\ \text{s.t.} \quad & x_0 = x_i \\ & x_L = x_f \\ & x_{k+1} = f(x_k, u_k), k = 0, ..., L-1 \\ & x_k \in X_{free}(k), k = 0, ..., L-1 \\ & u_k \in U, k = 0, ...L-1 \end{aligned} \tag{5.16}$$

Formulating the motion planning problem as an optimal control problem makes it easy to put constraints on the state and control input as well as switching dynamical model and performance measure [5]. There exists several techniques to solve this optimal control problem, which have been implemented in several softwares. The details of the solution techniques will not be outlined in this thesis. Instead, the next section will discuss how the motion planning problem was solved using existing software.

### 5.2.2 Implementation

To solve an optimal control problem in the form (5.16), the software library CasADi was used with the IPOPT solver. In CasADi an optimal control problem can be formulated and saved as a function to minimize the computational time for each time the problem is solved. Input to the function was the initial and final state, and output was state and control signals for each trajectory. The outputted states were used as reference to the state feedback for each drone.

The dynamical constraints in (5.16) are given by (5.12). The same $F$ and $G$ matrices as in (3.10) were used which is the discretized form of the state space model from (2.2). The constraints for the state input are defined by $X_{free}(k)$, which is the free state space where the drones are allowed to go. The drones are restricted by the fixed constraints of the room as well as constraints regarding collision avoidance. i.e., the drones are not allowed to fly into walls and to be closer than 0.5 meters from each other. The constraints on the control input are defined by $U$ which is a lower and upper bound of the pitch and roll angles for each drone. This is necessary to avoid overly aggressive or even unstable maneuvers for the drones. The initial and final state are defined before each flight as the current position as well as the desired final position of each drone.

Next consider the cost function $J_l$ that defines the performance measurement of the trajectory. This was a crossroad for two different implementations of the motion planning problem. However, one thing in common for both implementations was the inclusion of a cost for formation error. This meant that the computed trajectories kept the formation error to a minimum. Apart from the formation error, a cost for the final time is included. This yields the cost function:

$$J_l = WT_f + \sum_{j=1}^{L} \sum_{d=1}^{N} (P_{leader} - P_d + D_d)^2, \tag{5.17}$$

where $W$ is a weight, $T_f$ is the final time, $L$ is number of control steps in the trajectory, $N$ is number of drones, $P_{leader}$ is the position of leader, $P_d$ is the position of a drone and $D_d$ is the desired displacement of drone $d$.

While this worked well, two drawbacks were found using this method. The first drawback is that the sample time for the computed trajectory is dependent on the number of control steps, $L$, as well as the final time $T_f$. If the state output from the trajectory should be used as state reference to the drones, it is important that the sample time for the trajectory is the same as the sample time for updating commands to the drones. This could, however, be fixed by resampling the outputted state and control signal to the correct sample time. Another drawback by having the final time as a variable was that the computational time was quite long (up to several seconds). If the trajectories were to be updated in real time the computational time had to be lower and therefore another motion planning problem was formulated where the final time as well as the sample time was fixed. In this case the cost function was formulated to minimize the distance to the final state in each time step. With a fixed sample time the computational time was lowered significantly, approximately by a factor of 10. This gave the cost function:

$$J_l = \sum_{j=1}^{L} (W(P_{leader} - P_{goal})^2 + \sum_{d=1}^{N} (P_{leader} - P_d + D_d)^2) \tag{5.18}$$

with the same variables as (5.17) and $P_{goal}$ is the position of the goal state.

# 6 Solutions to problem formulation

This chapter provides four different solutions that incorporate the algorithms and implementations stated in previous chapters to solve the scenario outlined in Chapter 1. The scenario consists of two drone swarms that will take on different missions executed in the same environment, where the drone swarms will be on a direct collision course with each other. This scenario is chosen because of the simplicity and the inclusion of the fundamental problems in safe collision avoidance control and motion planning of drone swarms.

## 6.1 APF+Leader-follower+PID+CBF

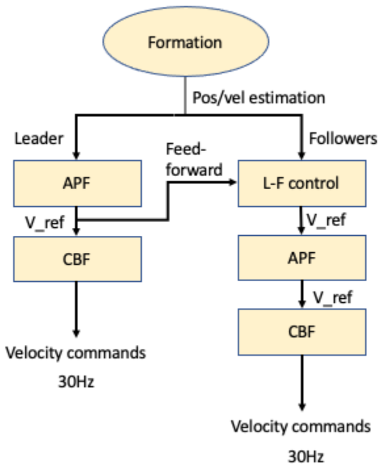


***Figure 6.1:*** *APF is used for collision free guidance. The formation is maintained through leader-follower (L-F) control. Lastly the control barrier function (CBF) guarantees safe control.*

The first solution uses the algorithms from leader-follower control, described in Section 3.4, to keep formation and the algorithms from APF, described in Section 5.1, for collision avoidance. This gives a nominal control signal describing the desired velocity for each drone. A control barrier function makes sure that the control signal is safe, i.e., guarantees that the drones will not collide. In each formation there is one leader and the rest are followers. The leader is the one controlling where the formation will go and the followers try to keep their relative position to the leader. This is the reason only the followers have a block for leader follower control in Figure 6.1. Important to note is that the followers do not have an attraction contribution to their resulting desired velocity from the APF. However, the desired velocity from the leader-follower control can be seen as that attraction contribution. The velocity contribution from leader follower control and APF are added before sending that nominal control signal to the block for control barrier function to guarantee safety. In Figure 6.1 an illustration of the solution is presented.

A drawback by using velocity commands as control signal in combination with a control barrier function is during deadlocks when two drones are directly in front of each other. The closest control signal to the nominal control signal that is safe will be zero. However if the drone keeps receiving zeros as the velocity command it will eventually start to drift and potentially crash into another drone. This was solved by controlling the position of the drone when the control signal, desired velocity, was zero. Whenever the control signal was greater than zero, the drone was once again controlled by the desired velocity.

Another important aspect is how to solve deadlocks. Deadlocks occur when another drone is blocking the way and the algorithms do not find a way forward past that drone. The simple way of dealing with this problem is to identify this scenario and perturb the nominal control signal. With a sufficient amount of perturbation, the drones will be able to break the deadlock.

## 6.2 APF+Leader-follower+LQR+ECBF

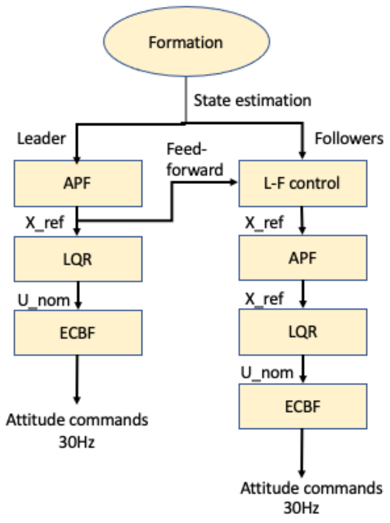


***Figure 6.2:*** *APF is used for collision free guidance. The formation is maintained through leader-follower (L-F) control. An exponential control barrier function (ECBF) guarantees safe control and, lastly, an LQR controller is implemented with attitude as control signals.*

The second solution, shown in Figure 6.2, is the same as the first except for a different control scheme. In this case an LQR controller, described in Section 3.3, is implemented, which outputs attitude commands, and a PID controller is used to control thrust to the drone. As state feedback is used, the output from the leader-follower control and APF are now used as the state reference. The output of those algorithms is velocity which means that the position reference is given by integrating the velocity reference. Another difference is that ECBF is used instead of CBF since the control signal now does not affect the first derivative of the $h$-function. This is explained in more detail in Chapter 4.

## 6.3 Open-loop motion planning+LQR

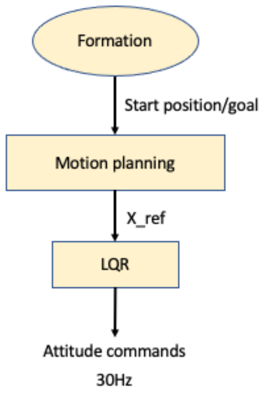


***Figure 6.3:*** *Open-loop motion planning with a LQR controller for trajectory following.*

In the third solution a trajectory is calculated prior to take-off, as described in Section 5.2, for all drones based on their starting position. The trajectory is optimized for safety, time and formation control, which means that the algorithms from formation control, collision avoidance and safety are not included. The trajectory given by solving the motion planning problem (5.16) contains the desired states in each time step which are used as state reference to the state feedback. After take-off the drones tries to get as close to their starting position as possible in the x-y plane before following the trajectory to minimize any initial error. Initially, an ECBF was intended to be a part of this solution however since the ECBF was interfering with the trajectory following it had to be left out. The safety distance to other drones is still implemented as constraints in the motion planning problem. However this solution is not particularly robust with respect to changes in the environment and disturbances. Figure 6.3 provides an illustration of the solution.

## 6.4 Closed loop Motion planning+LQR

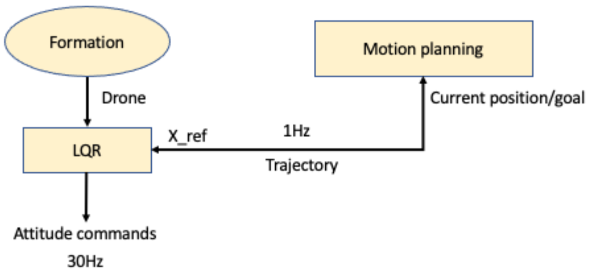


***Figure 6.4:*** *Closed-loop motion planning with a LQR controller for trajectory following.*

Finally, the fourth solution also uses trajectories solving the motion planning problem in (5.16). However, the trajectories are now updated in real time during the flight. The updated trajectory will be adjusted to the current position of the drone, which is beneficial in case a drone gets off course from the initial trajectory. To run several processes on the computer, one that controls the drones and one which updates the trajectories, ROS is used. One of the major features in ROS is to run several processes at the same time and communicate between them. Because of the computational time of calculating optimal trajectories, when a new trajectory has been found, the drones most likely will not be at the initial point of the trajectory. To contend with that problem, whenever the trajectory is updated, the point on the trajectory closest to the drone is used as starting point. Figure 6.4 provides an illustration of the solution.

# 7 Results

In this chapter presents the results from the implementation of the solutions from Chapter 6 on the hardware presented in Chapter 2. Five identical tests for each solution have been performed. The tests were done with four drones in two formations and the mission for each leader was to go to the initial position of the other leader, as illustrated in Figure 7.1.

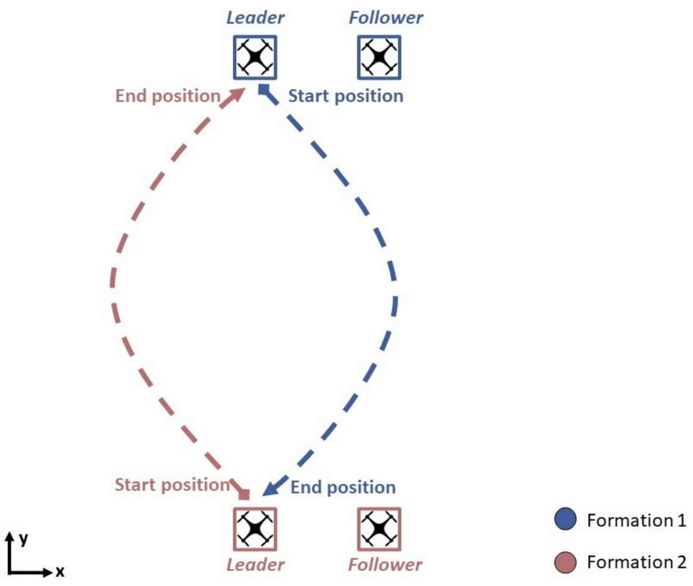


***Figure 7.1:*** *For each solution presented in Chapter 6, the tests were done with four drones in two formations, where the mission for each leader was to go to the initial position of the other leader.*

To reduce the number of figures, one or two typical tests from each solution will be presented in detail with plots for paths, formation error and value of control barrier function. The other test results are presented in tables. The chapter ends with a comparison of the solutions based on their performance.

The performance of each test is based on five metrics relating to the desired performance outlined in the problem formulation. The five metrics are:

- Time to finish the mission.
- Distance traveled
- Smoothness of trajectory
- Average formation error
- Unsafe time

Distance traveled is the average distance traveled by one drone. Smoothness of trajectory is measured as the average angle difference between two adjacent vectors of the trajectory. Given two adjacent vectors $u$ and $v$ the angle difference is

$$\Delta angle = |\arctan(\frac{u_y}{u_x}) - \arctan(\frac{v_y}{v_x})|. \tag{7.1}$$

$u$ is the position vector from point $i$ to point $i+1$ on the trajectory and $v$ is the position vector from point $i+1$ to point $i+2$ on the trajectory. A completely straight and smooth trajectory would correspond to $\Delta angle = 0$. Smoothness of the trajectory is defined as the average $\Delta angle$ taken over the entire trajectory. This measurement is inspired from [12] where a similar measurement for smoothness is used. The average formation error is the average leader-follower displacement error. Unsafe time is the total time that the distance from one drone to other drones is less than the safety distance.

## 7.1 APF+Leader-follower+PID+CBF

The first solution uses APF for guidance, leader follower control to keep formation and CBF for safety, see details in Section 6.1. In the next section, a typical test run is presented more thoroughly to give the reader a good intuition of how the drones behave with this implementation, after which the statistics of the five test runs are presented.

### 7.1.1 Detailed presentation of a typical test run

The tests show similar behaviour and metrics, and Test 3 is a good representation of the most typical behaviour out of the five tests. The resulting path of the drones are shown in Figure 7.2; Figure 7.3 shows the safety constraint of the CBF, which should always be positive to satisfy the safety distance between two drones; Figure 7.4 shows the formation error.

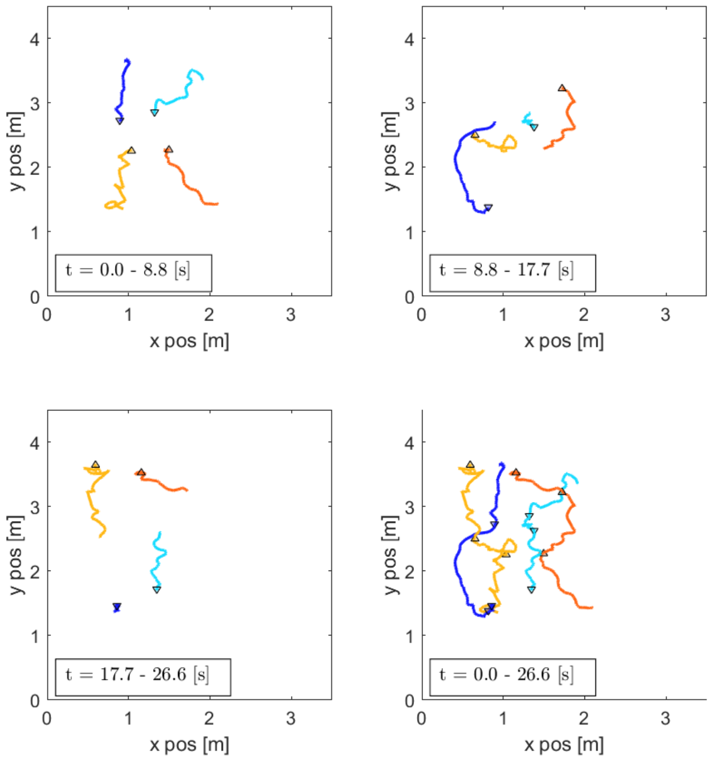


***Figure 7.2:*** *Drone paths for Test 3. The tests show similar behaviour and metrics, and Test 3 is a good representation of the most typical behaviour out of the five tests. The yellow drone is the leader of formation yellow/red, and the blue drone is the leader of formation blue/cyan. The mission is for each leader to go to the initial position of the other leader. Solution is built on the single integrator model and APF, CBF, leader follower control for maintaining formation and avoiding collisions.*

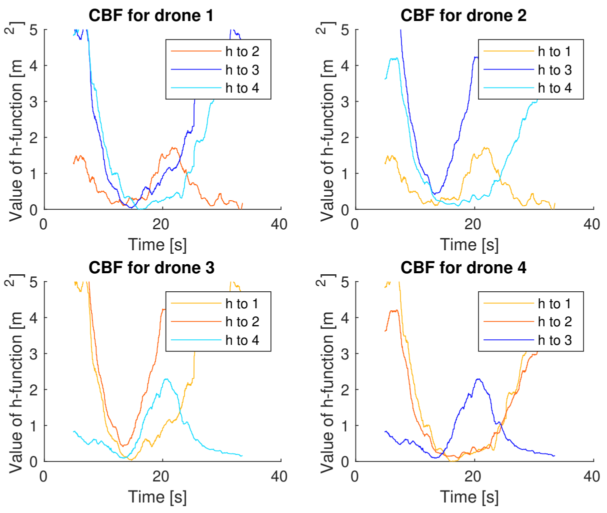


*Figure 7.3:* *Value of h-function, defined in (4.4), from Test 3. Theoretically, the CBF guarantees that the h-function is always positive. "h to x" means value of h-function to drone x.*

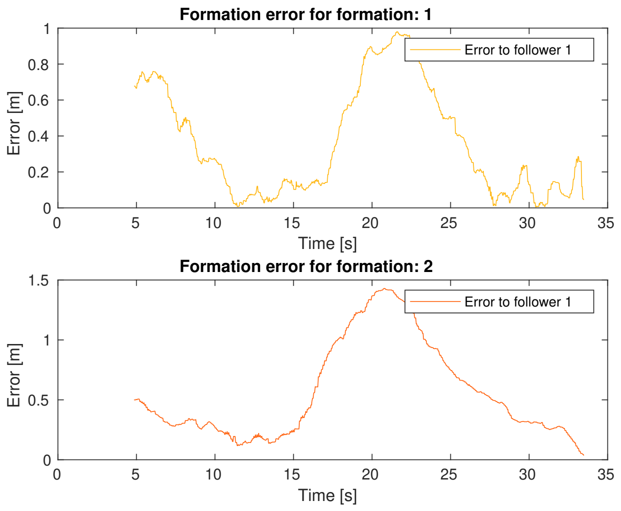


*Figure 7.4:* *Value of formation error from Test 3. The formation error is the leader-follower displacement error.*

### 7.1.2 Statistics from all test runs

In this section, tables are presented of the the performance metrics on all test runs and a total average score.

***Table 7.1:*** *Time to finish the mission.*

| Time [s] | | | | | |
|---|---|---|---|---|---|
| Test 1 | Test 2 | Test 3 | Test 4 | Test 5 | Average |
| 26.6 | 23.3 | 26.6 | 36.3 | 26.7 | 27.9 |

***Table 7.2:*** *Average distance traveled for one drone.*

| Distance traveled [m] | | | | | |
|---|---|---|---|---|---|
| Test 1 | Test 2 | Test 3 | Test 4 | Test 5 | Average |
| 4.13 | 3.90 | 3.81 | 5.40 | 4.43 | 4.33 |

***Table 7.3:*** *Smoothness of the trajectory is the average $\Delta angle$ for the entire trajectory.*

| Smoothness [°] | | | | | |
|---|---|---|---|---|---|
| Test 1 | Test 2 | Test 3 | Test 4 | Test 5 | Average |
| 44.8 | 38.3 | 41.6 | 38.4 | 40.9 | 40.8 |

***Table 7.4:*** *Unsafe time is the total time that the distance between two drones is less than the safety distance.*

| Unsafe time [s] | | | | | | |
|---|---|---|---|---|---|---|
| Drone | Test 1 | Test 2 | Test 3 | Test 4 | Test 5 | |
| 1 | 0.0 | 0.0 | 0.533 | 0.633 | 0.0 | |
| 2 | 0.0 | 0.0 | 0.433 | 0.0 | 0.0 | |
| 3 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 4 | 0.0 | 0.0 | 0.1 | 0.633 | 0.0 | |
| Average | 0.0 | 0.0 | 0.267 | 0.317 | 0.0 | 0.117 |

***Table 7.5:*** *Average formation error is the average leader-follower displacement error.*

| Average formation error [m] | | | | | | |
|---|---|---|---|---|---|---|
| Form ation | Test 1 | Test 2 | Test 3 | Test 4 | Test 5 | |
| 1 | 0.357 | 0.345 | 0.365 | 0.418 | 0.368 | |
| 2 | 0.466 | 0.495 | 0.566 | 0.724 | 0.471 | |
| Average | 0.412 | 0.42 | 0.466 | 0.571 | 0.42 | 0.458 |

## 7.2 APF+Leader-follower+LQR+ECBF

The second solution uses APF for guidance, leader follower control to keep formation, ECBF for safety and a LQR controller to send attitude commands to the Crazyflie, see details in Section 6.2. Test number 2 of 5 will be presented more thoroughly.

### 7.2.1 Detailed presentation of a typical test run

The tests show similar behaviour and metrics, and Test 2 is a good representation of the most typical behaviour out of the five tests. The resulting path of the drones are shown in Figure 7.5; Figure 7.6 shows the safety constraint of the CBF, which should always be positive to satisfy the safety distance between two drones; Figure 7.7 shows the formation error.

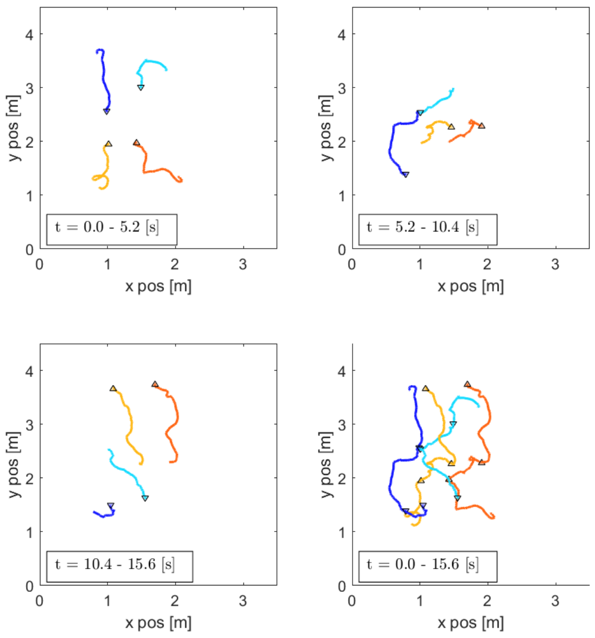


*Figure 7.5: Drone paths for Test 2. The tests show similar behaviour and metrics, and Test 2 is a good representation of the most typical behaviour out of the five tests. The yellow drone is the leader of formation yellow/red, and the blue drone is the leader of formation blue/cyan. The mission is for each leader to go to the initial position of the other leader. Solution is built on the quadcopter model and APF, ECBF, leader follower control for maintaining formation and avoiding collisions.*

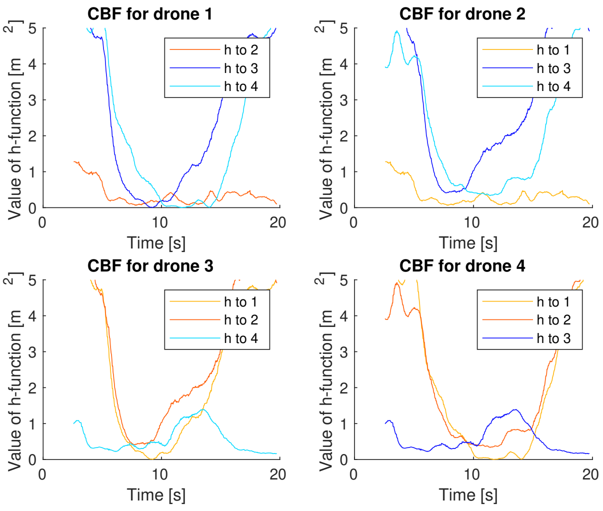


***Figure 7.6:*** *Value of $h$-function, defined in (4.4), from Test 2. Theoretically, the ECBF guarantees that the $h$-function is always positive. "h to x" means value of $h$-function to drone x.*

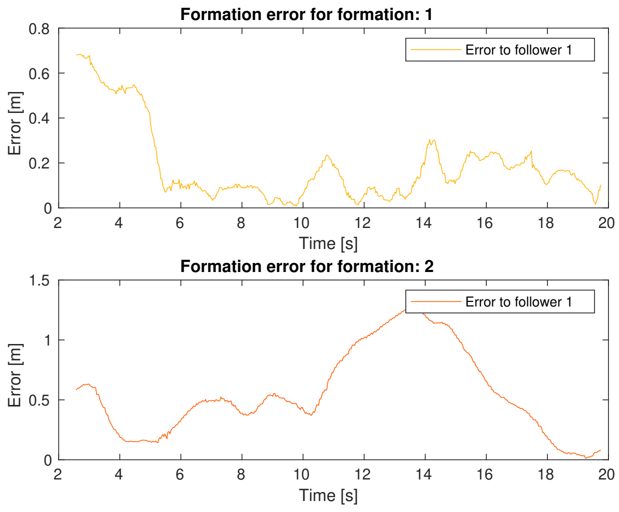


***Figure 7.7:*** *Value of formation error from Test 2. The formation error is the leader-follower displacement error.*

### 7.2.2 Statistics from all test runs

***Table 7.6:*** *Time to finish the mission.*

| Time [s] | | | | | |
|---|---|---|---|---|---|
| Test 1 | Test 2 | Test 3 | Test 4 | Test 5 | Average |
| 16.0 | 15.6 | 16.3 | 20.6 | 19.0 | 17.5 |

***Table 7.7:*** *Average distance traveled for one drone.*

| Distance traveled [m] | | | | | |
|---|---|---|---|---|---|
| Test 1 | Test 2 | Test 3 | Test 4 | Test 5 | Average |
| 4.19 | 3.99 | 4.09 | 5.43 | 5.01 | 4.54 |

***Table 7.8:*** *The smoothness of the trajectory is the average $\Delta angle$ for the entire trajectory.*

| Smoothness [°] | | | | | |
|---|---|---|---|---|---|
| Test 1 | Test 2 | Test 3 | Test 4 | Test 5 | Average |
| 34.7 | 38.5 | 28.8 | 35.0 | 34.5 | 34.3 |

***Table 7.9:*** *Unsafe time is the total time that the distance between two drones is less than the safety distance.*

| Unsafe time | | | | | | |
|---|---|---|---|---|---|---|
| Drone | Test 1 | Test 2 | Test 3 | Test 4 | Test 5 | |
| 1 | 0.067 | 0.0 | 0.367 | 1.4 | 0.833 | |
| 2 | 0.067 | 0.0 | 0.333 | 1.067 | 0.2 | |
| 3 | 0.0 | 0.0 | 0.0 | 0.233 | 0.133 | |
| 4 | 0.0 | 0.0 | 0.033 | 1.3 | 0.9 | |
| Average | 0.034 | 0.0 | 0.183 | 1.0 | 0.517 | 0.347 |

***Table 7.10:*** *Average formation error. The formation error is the leader-follower displacement error.*

| Formation error [m] | | | | | | |
|---|---|---|---|---|---|---|
| Form ation | Test 1 | Test 2 | Test 3 | Test 4 | Test 5 | |
| 1 | 0.217 | 0.186 | 0.223 | 0.533 | 0.370 | |
| 2 | 0.549 | 0.559 | 0.493 | 0.462 | 0.519 | |
| Average | 0.383 | 0.372 | 0.358 | 0.498 | 0.444 | 0.411 |

## 7.3 Open-loop motion planning + LQR

The third solution solves the motion planning problem from (5.16) which gives an optimal trajectory for the mission. The trajectory is followed using the LQR controller, see details in Section 6.3. Test number 1 will be presented in detail.

### 7.3.1 Detailed presentation of a typical test run

The tests show similar behaviour and metrics, and Test 3 is a good representation of the most typical behaviour out of the five tests. The resulting paths of the drones are shown in Figure 7.8; Figure 7.9 shows the safety constraint of the CBF, which should always be positive to satisfy the safety distance between two drones; Figure 7.10 shows the formation error.

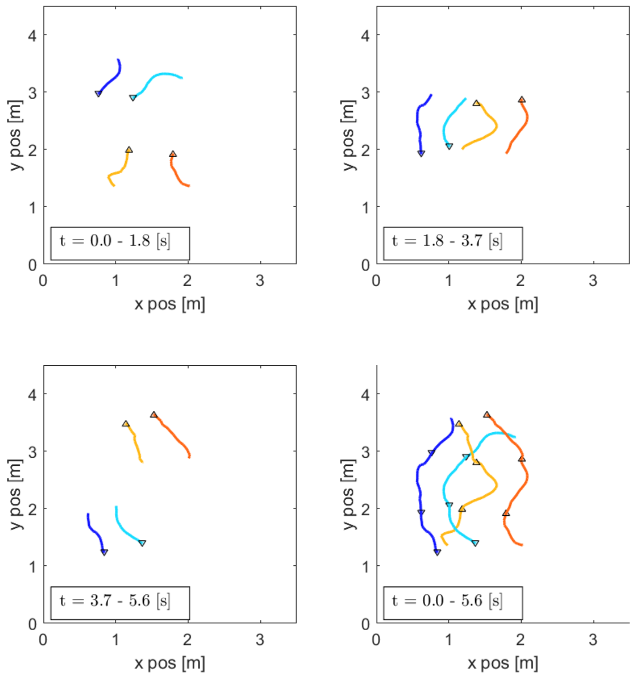


***Figure 7.8:*** *Drone paths for Test 1. The tests show similar behaviour and metrics, and Test 1 is a good representation of the most typical behaviour out of the five tests. The yellow drone is the leader of formation yellow/red, and the blue drone is the leader of formation blue/cyan. The mission is for each leader to go to the initial position of the other leader. Solution is built on the quadcopter model and motion planning for maintaining formation and avoiding collisions.*

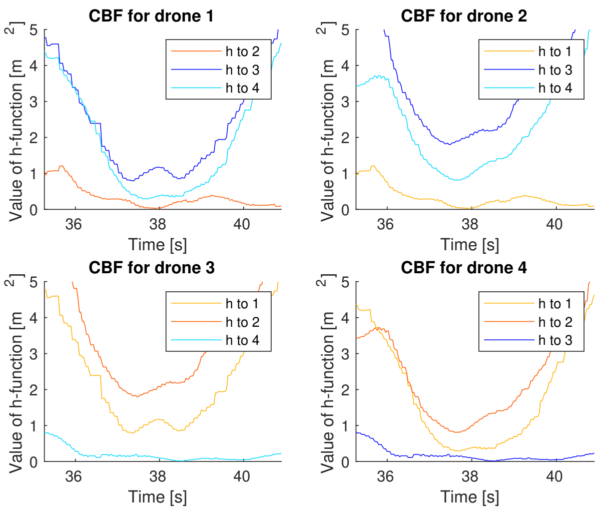


***Figure 7.9:*** *Value of h-function, defined in (4.4), from Test 1. For Open-loop motion planning+LQR, the CBF algorithm is not implemented instead the safety distance is defined as a constraint in the motion planning problem. "h to x" means value of h-function to drone x.*

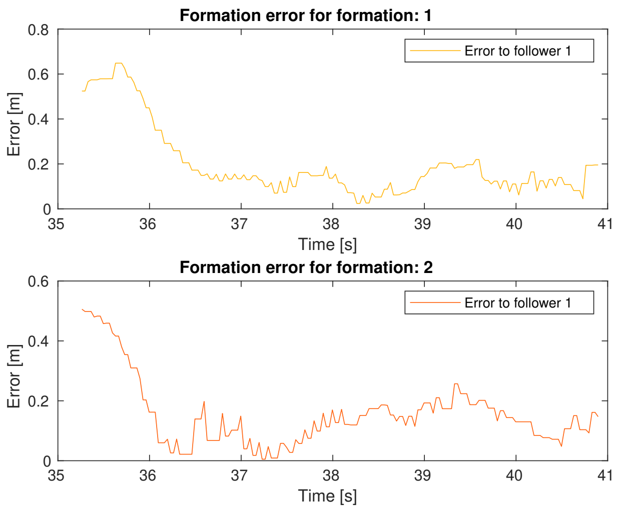


***Figure 7.10:*** *Value of formation error from Test 1. The formation error is the leader-follower displacement error.*

### 7.3.2 Statistics from all test runs

***Table 7.11:*** *Time to finish the mission.*

| Time [s] | | | | | |
|---|---|---|---|---|---|
| Test 1 | Test 2 | Test 3 | Test 4 | Test 5 | Average |
| 5.6 | 5.6 | 5.6 | 5.6 | 5.6 | 5.6 |

***Table 7.12:*** *Average distance traveled for one drone.*

| Distance traveled [m] | | | | | |
|---|---|---|---|---|---|
| Test 1 | Test 2 | Test 3 | Test 4 | Test 5 | Average |
| 2.69 | 2.63 | 2.51 | 2.60 | 2.66 | 2.62 |

***Table 7.13:*** *The smoothness of the trajectory is the average $\Delta angle$ for the entire trajectory.*

| Smoothness [°] | | | | | |
|---|---|---|---|---|---|
| Test 1 | Test 2 | Test 3 | Test 4 | Test 5 | Average |
| 14.5 | 13.2 | 13.8 | 15.4 | 11.6 | 13.7 |

***Table 7.14:*** *Unsafe time is the total time that the distance between two drones is less than the safety distance.*

| Unsafe time [s] | | | | | | |
|---|---|---|---|---|---|---|
| Drone | Test 1 | Test 2 | Test 3 | Test 4 | Test 5 | |
| 1 | 0.0 | 0.0 | 0.0 | 0.0 | 0.167 | |
| 2 | 0.0 | 0.0 | 0.0 | 0.0 | 0.167 | |
| 3 | 0.0 | 0.0 | 0.6 | 0.0 | 1.1 | |
| 4 | 0.0 | 0.0 | 0.6 | 0.0 | 1.1 | |
| Average | 0.0 | 0.0 | 0.3 | 0.0 | 0.63 | 0.186 |

***Table 7.15:*** *Average formation error. The formation error is the leader-follower displacement error.*

| Formation error [m] | | | | | | |
|---|---|---|---|---|---|---|
| Form-ation | Test 1 | Test 2 | Test 3 | Test 4 | Test 5 | |
| 1 | 0.198 | 0.182 | 0.158 | 0.175 | 0.213 | |
| 2 | 0.155 | 0.135 | 0.212 | 0.148 | 0.221 | |
| Average | 0.177 | 0.159 | 0.185 | 0.162 | 0.217 | 0.18 |

## 7.4 Closed-loop motion planning+LQR

The fourth solution is similar to the prior solution however this time the trajectories are updated in real time, see details in Section 6.4. Here, two typical behaviours are observed. The first behaviour is that the formations go around each other, and the second behaviour is that the formations take a more direct path, utilizing the space between two drones. Test number 1 and 4 are presented more thoroughly to showcase two different paths the drones chose to finish their mission.

### 7.4.1 Detailed presentation of two typical test runs

The tests show similar behaviour and metrics, and Test 3 is a good representation of the most typical behaviour out of the five tests. The resulting path of the drones are shown in Figures 7.11 and 7.12; Figures 7.13 and 7.14 show the safety constraint of the CBF, which should always be positive to satisfy the safety distance between two drones; Figures 7.15 and 7.16 show the formation error.

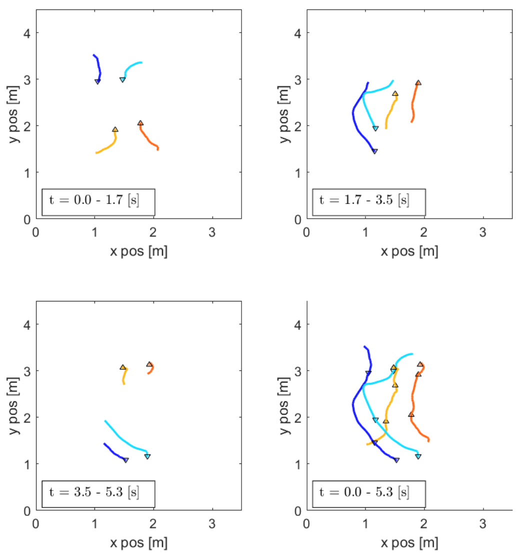


***Figure 7.11:*** *Trajectory for Test 1. Out of the five tests, two typical behaviours are observed. The first behaviour is that the formations go around each other, shown here for Test 1. The yellow drone is the leader of formation yellow/red, and the blue drone is the leader of formation blue/cyan. The mission is for each leader to go to the initial position of the other leader. The solution is built on the quadcopter model and real-time motion planning for maintaining formation and avoiding collisions.*

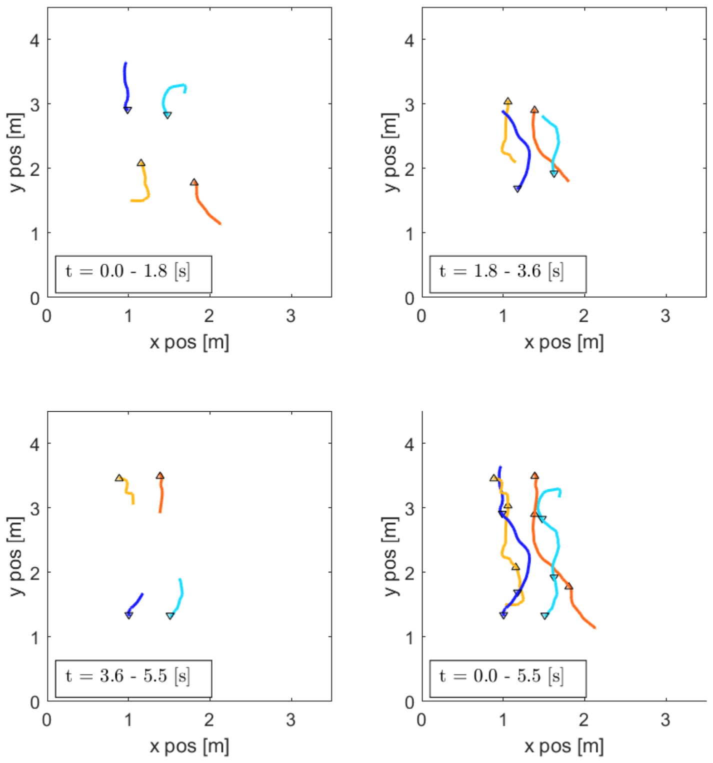


***Figure 7.12:*** *Out of the five tests, two typical behaviours are observed. The second behaviour is that the formations take a more direct path, utilizing the space between two drones, shown here for Test 4.*

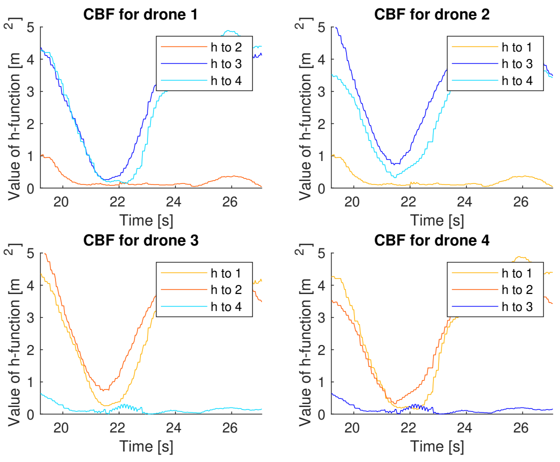


*Figure 7.13: Value of h-function, defined in (4.4), from Test 1. For Closed-loop motion planning+LQR, the CBF algorithm is not implemented instead the safety distance is defined as a constraint in the motion planning problem. "h to x" means value of h-function to drone x.*

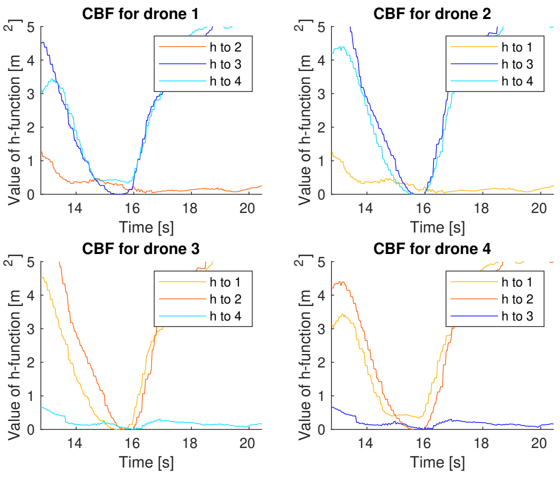


*Figure 7.14: Value of h-function, defined in (4.4), from Test 4. For Closed-loop motion planning+LQR, the CBF algorithm is not implemented instead the safety distance is defined as a constraint in the motion planning problem. "h to x" means value of h-function to drone x.*

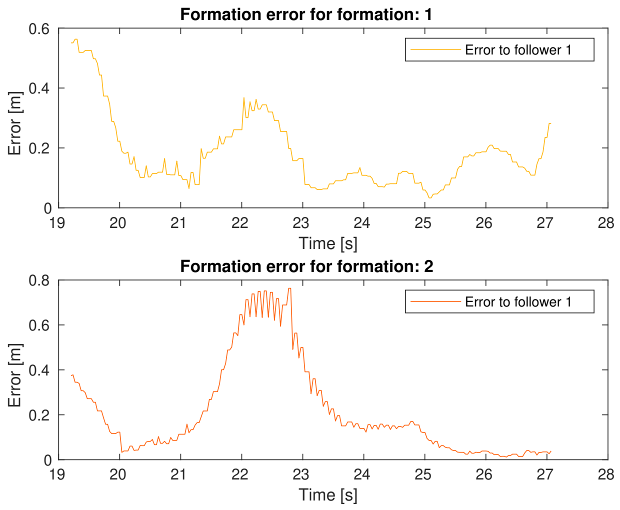


***Figure 7.15:*** *Value of formation error from Test 1. The formation error is the leader-follower displacement error.*

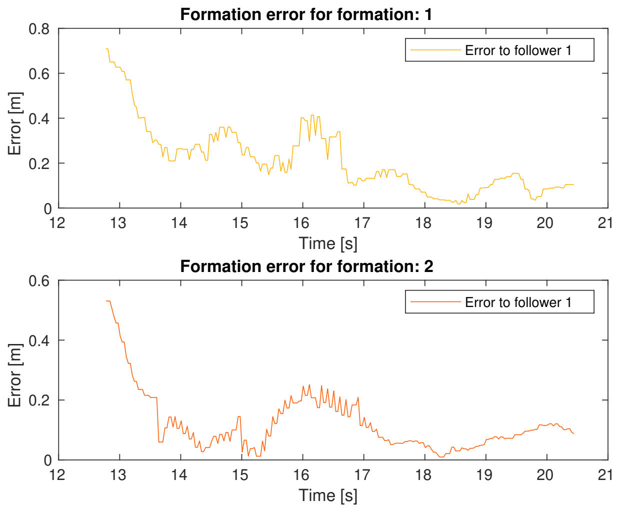


***Figure 7.16:*** *Value of formation error from Test 4. The formation error is the leader-follower displacement error.*

### 7.4.2 Statistics from all test runs

***Table 7.16:*** *Time to finish the mission.*

| Time [s] | | | | | |
|---|---|---|---|---|---|
| Test 1 | Test 2 | Test 3 | Test 4 | Test 5 | Average |
| 5.3 | 6.0 | 5.3 | 5.5 | 6.0 | 5.6 |

***Table 7.17:*** *Average distance traveled for one drone.*

| Distance traveled [m] | | | | | |
|---|---|---|---|---|---|
| Test 1 | Test 2 | Test 3 | Test 4 | Test 5 | Average |
| 2.47 | 2.69 | 2.56 | 2.51 | 2.75 | 2.60 |

***Table 7.18:*** *The smoothness of the trajectory is the average $\Delta angle$ for the entire trajectory.*

| Smoothness [°] | | | | | |
|---|---|---|---|---|---|
| Test 1 | Test 2 | Test 3 | Test 4 | Test 5 | Average |
| 29.5 | 27.6 | 23.1 | 26.4 | 19.8 | 25.3 |

***Table 7.19:*** *Unsafe time is the total time that the distance between two drones is less than the safety distance.*

| Unsafe time [s] | | | | | | |
|---|---|---|---|---|---|---|
| Drone | Test 1 | Test 2 | Test 3 | Test 4 | Test 5 | |
| 1 | 0.0 | 0.0 | 0.0 | 0.1 | 1.0 | |
| 2 | 0.0 | 0.0 | 0.333 | 0.4 | 0.3 | |
| 3 | 0.033 | 2.067 | 0.0 | 0.467 | 0.167 | |
| 4 | 0.033 | 2.067 | 0.333 | 0.4 | 0.7 | |
| Average | 0.017 | 1.033 | 0.167 | 0.342 | 0.542 | 0.42 |

***Table 7.20:*** *Average formation error. The formation error is the leader-follower displacement error.*

| Formation error [m] | | | | | | |
|---|---|---|---|---|---|---|
| Form-ation | Test 1 | Test 2 | Test 3 | Test 4 | Test 5 | |
| 1 | 0.182 | 0.213 | 0.158 | 0.213 | 0.207 | |
| 2 | 0.215 | 0.272 | 0.256 | 0.123 | 0.288 | |
| Average | 0.199 | 0.242 | 0.207 | 0.168 | 0.248 | 0.213 |

## 7.5 Summary of results

The statistics from all test runs are presented in Table 7.21.
APF+Leader-follower+PID+CBF is the slowest to finish the mission with a high formation error and poor smoothness, however it also has the least amount of unsafe time for each drone. APF+Leader-follower+LQR+ECBF is similar to the first solution although noticeable differences is the faster time to finish the mission and more unsafe time. Open-loop Motion planning has the best result in all aspects except the unsafe time where only the first solution surpasses. Closed-loop Motion planning performs similarly to the third solution, however it has an inferior performance in regards to smoothness and unsafe time. The results are discussed in Chapter 8.

***Table 7.21:*** *Summary of results for all solutions. The average value of the 5 tests are presented.*

| Summary of results | | | | |
|---|---|---|---|---|
| - | Solution 1 | Solution 2 | Solution 3 | Solution 4 |
| Time [s] | 27.9 | 17.5 | 5.6 | 5.6 |
| Distance [m] | 4.33 | 4.54 | 2.62 | 2.60 |
| Smoothness [°] | 40.8 | 34.3 | 13.7 | 25.3 |
| Formation error [m] | 0.458 | 0.411 | 0.18 | 0.213 |
| Unsafe time [s] | 0.117 | 0.347 | 0.186 | 0.42 |

# 8 Discussion

In this chapter the results from Chapter 7, limiting factors to the performance of the system and potential improvements will be discussed.

## 8.1 Results

The only difference between the first and second solution is the controller which is illustrated in Figures 2.3 and 2.4. A comparison between these two solutions is therefore equivalent to evaluating the two different control architectures. A noticeable difference is the much faster time to finish the mission using the LQR controller which indicates a more aggressive controller compared to the default controller on the Crazyflie firmware. From the time and trajectory it is clear that a less aggressive controller results in more time to solve deadlocks. A deadlock occurs when two drones face each other with the desired velocity directed towards each other. The control barrier function mitigate collision, which results in drones getting stuck on their path to the goal. During deadlocks the impact of control barrier function is high, which leads to small reference signals. In this case an aggressive controller is more likely to give a control signal with sufficient magnitude to resolve deadlocks. Since formation-keeping is neglected during deadlocks, the formation error is also slightly higher using a PID controller. Interestingly, with a more aggressive controller the trajectories are smoother. This is probably caused by the indecisiveness during deadlocks causing the desired velocity to change direction many times. A less aggressive controller is however better with regards to safety, where the first solution outperforms all other solutions.

The differences between open-loop motion planning and closed-loop motion planning is more unsafe time for closed-loop motion planning, higher formation error as well as inferior smoothness. This is probably due to the dynamics of changing trajectory during the flight. As described in Section 6.4 the computational time of computing a trajectory results in a position error of the drone relative to the initial point of the trajectory. Even though this is compensated for by finding the closest point on the new trajectory to the drone, the closest point is not guaranteed to be close to the drone. This disturbance to the reference trajectory is why closed-loop motion planning performs worse compared to open-loop motion planning. The advantage with closed loop motion planning is, in theory, higher robustness to changes in the environment. For example if a formation of drones do not know how an other formation will fly, a closed-loop motion planning could, in real-time, change to a safe trajectory while an open-loop motion planning would lead to collision. High robustness is, however, dependent on that the motion planning problem from (5.16) can be solved quickly. Less computational time to solve the motion planning problem would not only increase the robustness, but also enhance the performance in regards to the performance metrics.

The motion planning solutions outperforms the solutions using APF+Leader-follower+CBF. However, since decision making is done in real time for every time step, these solutions are much more robust to disturbances and changes in the environment. One idea prior to testing was to include an ECBF to the motion planning solutions to guarantee that the control signal was safe. However the trajectory following was affected negatively and a decision was made to exclude any real time safety measures. The main reason that the motion planning solutions outperform the solutions using APF+Leader-follower+CBF is that deadlocks are avoided. The algorithms for APF only looks at each time step separately and therefore a long-term plan can not be achieved. Solving the motion planning problem from (5.16) gives a planned trajectory for several seconds in the future, which allows for smooth avoidance of deadlocks and collisions.

Open-loop motion planning is applicable for a known and controlled environment with centralized control for all drones. Closed-loop motion planning is more robust to changes in the environment however still requires centralized control of all drones. The solutions using APF+Leader follower+CBF/ECBF do not require centralized control of all drones neither do they require a controlled environment. Since decision making is done in real time these solutions are applicable for unknown and changing environments. Another important aspect is scalability. Since the computational time of solving the motion planning problem increases with more drones, the closed-loop motion planning solution does not scale well. Four drones was probably near the limit with the implementation in this thesis. The other solutions scale well.

## 8.2 Limiting factors to performance

There are three important limiting factors to the performance of the system. The first factor is the positioning system which has a precision in the range of centimeters. Disturbances and errors to the position and velocity estimation will have a negative affect on the performance of the algorithms.

The second factor, only limiting to the closed-loop motion planning solution, is computational power. The computational power affects the time to calculate each trajectory which limits the update frequency of new trajectories. A low update frequency, results in trajectories that are based on the wrong position of the drones since the drones have moved during the computation process. To decrease the computational time for solving optimal control problems it is possible for some solvers in CasADi to generate C code. Unfortunately this is not possible using the IPOPT which was used to solve the non-convex problem in (5.16). In the open-loop motion planning solution the trajectory is computed prior to take off which means that the computational time is not a limiting factor, at the cost of not being able to change trajectory to account for a changing environment.

The third limiting factor to the system is the communication between the computer and the Crazyflies. The bandwidth of the radio limits the update frequency of position/velocity/attitude estimations and the frequency of sending commands to each Crazyflie. Flying several drones at the same time lowers the bandwidth of the communication which has a negative affect on performance. With four drones the update frequency of estimations is lowered compared to one or two drones, although no decrease in performance is observable. However, this can become a limiting factor if more drones are added to the swarms.

## 8.3 Potential improvements

Listed below are potential improvements to the work in this thesis.

- A more precise positioning system would lead to significant improvements. Since the position estimate is used in almost every part of the solutions, any error will lead to reduced performance.

- To enhance the performance of solution 4 the computational time of computing trajectories needs to be lowered. One solution is to generate C code to solve the motion planning problem however as for now this is not supported for the IPOPT solver which means that another solver needs to be used. Another option is to use a more powerful computer that solves the motion planning problem faster. The last option is to only define linear constraints to the motion planning problem, which requires less computation.

- The quadcopter model is used for the LQR controller, ECBF and to describe dynamics in the motion planning problem which means that a more precise model would improve all of those aspects. An interesting way to improve the model by using data driven system identification.

# 9 Conclusions and further work

This chapter will answer the problem formulation, derive conclusions and present further work on the subject.

## 9.1 Answers to the problem formulation

To solve the problem of collision-safe control and motion planning of drone swarms, four different solutions is proposed

- APF+Leader-follower+PID+CBF
- APF+Leader-follower+LQR+ECBF
- Open-loop Motion planning+LQR
- Closed-loop Motion planning+LQR

The algorithms were implemented on the hardware Crazyflie 2.1 and evaluated using the five metrics

- Time to finish the mission
- Distance traveled
- Smoothness of trajectory
- Average formation error
- Unsafe time

The Open-loop Motion planning+LQR had the best performance on almost every metric, although, requires planning of the trajectory in advance. The Closed-loop Motion planning+LQR has similar control performance but can run in real-time updating the trajectory at a frequency of 1 Hz. Lower computational resources is required for the APF+Leader-follower+PID+CBF and the APF+Leader-follower+LQR+ECBF, where the former is the least complex solution, and the latter is faster due to the LQR control enabled by more complex system dynamics modelling.

## 9.2 Conclusions

Collision-safe control and motion planning of drone swarms requires a collection of solutions from a widely researched area of motion planning and control. Four different solutions were implemented and tested on the Crazyflie 2.1 quadcopter platform. The solutions were divided into three categories, guidance, safety and control. The solutions based on trajectory generation performed better than the solutions based on APF for guidance. Quadcopter modelling and LQR control performed better than the simple single integrator model and PID control, justifying the added complexity to the control structure.

## 9.3 Further work

The solutions in this thesis use a centralized control structure with reliable and comprehensive information about the states of the drones. In most aerial swarm applications, reliable global positions and guidance from a centralized controller will not be available at all times. Therefore it would be useful to solve the problem locally on each drone, using relative positions between the drones.

Another aspect is to investigate if the controller can be improved by implementing a model predictive controller (MPC). Especially for solution 3 and 4 since the state reference trajectory is known for the whole mission.

Finally, it would be interesting to extend the algorithms to allow for movement along three axes, instead of two. This might require modelling of the turbulent air caused by the rotors, for the drones to be able to fly beneath each other.

# Bibliography

[1] Mohamed Abdelkader, Samet Güler, Hassan Jaleel, and Jeff S Shamma. Aerial swarms: Recent applications and challenges. *Current Robotics Reports*, 2(3):309–320, 2021.

[2] Abdulrahman Alarifi, AbdulMalik Al-Salman, Mansour Alsaleh, Ahmad Alnafessah, Suheer Al-Hadhrami, Mai A Al-Ammar, and Hend S Al-Khalifa. Ultra wideband indoor positioning technologies: Analysis and recent advances. *Sensors*, 16(5):707, 2016.

[3] Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, In Press, 2018.

[4] Tucker Balch and Ronald C Arkin. Behavior-based formation control for multirobot teams. *IEEE transactions on robotics and automation*, 14(6):926–939, 1998.

[5] Kristoffer Bergman. *Exploiting Direct Optimal Control for Motion Planning in Unstructured Environments*. PhD thesis, Linköping University Electronic Press, 2021.

[6] Bitcraze. Crazyflie 2.1, . URL `https://www.bitcraze.io/products/crazyflie-2-1/`.

[7] Bitcraze. Crazyradio pa, . URL `https://www.bitcraze.io/products/crazyradio-pa/`.

[8] Urs Borrmann, Li Wang, Aaron D Ames, and Magnus Egerstedt. Control barrier certificates for safe swarm behavior. *IFAC-PapersOnLine*, 48(27):68–73, 2015.

[9] Gunnarsson Svante Engqvist Martin Lindskog Peter Löfberg Johan m.fl Glad Torkel, Ljung Lennart. *Industriell reglereteknik Kurskompendium*. Institution for automatic control at university of Linköping, 2014.

[10] Ljung Lennart Glad Torkel. *Reglerteknik Grundläggande teori*. Studentlitteratur AB, 2006.

[11] Radu Horaud, Miles Hansard, Georgios Evangelidis, and Clément Ménier. An overview of depth cameras and range scanners based on time-of-flight technologies. *Machine vision and applications*, 27(7):1005–1020, 2016.

[12] Ibraheem Kasim Ibraheem and Fatin Hassan Ajeil. Multi-objective path planning of an autonomous mobile robot in static and dynamic environments using a hybrid PSO-MFB optimisation algorithm. *arXiv preprint arXiv:1805.00224*, 2018.

[13] I Iswanto, A Ma'arif, O Wahyunggoro, and A Imam. Artificial potential field algorithm implementation for quadrotor path planning. *Int. J. Adv. Comput. Sci. Appl*, 10(8):575–585, 2019.

[14] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous robot vehicles*, pages 396–404. Springer, 1986.

[15] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[16] Jonathan RT Lawton, Randal W Beard, and Brett J Young. A decentralized approach to formation maneuvers. *IEEE transactions on robotics and automation*, 19(6):933–941, 2003.

[17] Kwang-Kyo Oh, Myoung-Chul Park, and Hyo-Sung Ahn. A survey of multi-agent formation control. *Automatica*, 53:424–440, 2015.

[18] Linnea Persson. *Model Predictive Control for Cooperative Rendezvous of Autonomous Unmanned Vehicles*. PhD thesis, KTH Royal Institute of Technology, 2021.

[19] James A. Preiss*, Wolfgang Hönig*, Gaurav S. Sukhatme, and Nora Ayanian. Crazyswarm: A large nano-quadcopter swarm. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3299–3304. IEEE, 2017. doi: 10.1109/ICRA.2017.7989376. URL `https://doi.org/10.1109/ICRA.2017.7989376`. Software available at `https://github.com/USC-ACTLab/crazyswarm`.

[20] Wei Ren and Randal W Beard. Decentralized scheme for spacecraft formation flying via the virtual structure approach. *Journal of Guidance, Control, and Dynamics*, 27(1):73–82, 2004.

[21] Andrew Singletary, Karl Klingebiel, Joseph Bourne, Andrew Browning, Phil Tokumaru, and Aaron Ames. Comparative analysis of control barrier functions and artificial potential fields for obstacle avoidance. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8129–8136. IEEE, 2020.

[22] Nahid Soltani, Aref Shahmansoorian, and Mohammad A Khosravi. Robust distance-angle leader-follower formation control of non-holonomic mobile robots. In *2014 Second RSI/ISM International Conference on Robotics and Mechatronics (ICRoM)*, pages 024–028. IEEE, 2014.

[23] Jiayi Sun, Jun Tang, and Songyang Lao. Collision avoidance for cooperative uavs with optimized artificial potential field algorithm. *IEEE Access*, 5: 18382–18390, 2017.

[24] Robot Operating System. Ros/introduction. URL `http://wiki.ros.org/ROS/Introduction`.

[25] Kar-Han Tan and M Anthony Lewis. Virtual structures for high-precision cooperative mobile robotic control. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS'96*, volume 1, pages 132–139. IEEE, 1996.

[26] Li Wang, Aaron D Ames, and Magnus Egerstedt. Safety barrier certificates for collisions-free multirobot systems. *IEEE Transactions on Robotics*, 33(3): 661–674, 2017.

[27] Falin Wu, Jiemin Chen, and Yuan Liang. Leader-follower formation control for quadrotors. In *IOP Conference Series: Materials Science and Engineering*, volume 187, page 012016. IOP Publishing, 2017.

[28] Bin Xu and Koushil Sreenath. Safe teleoperation of dynamic uavs through control barrier functions. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7848–7855. IEEE, 2018.

[29] Huanshui Zhang, Lin Li, Juanjuan Xu, and Minyue Fu. Linear quadratic regulation and stabilization of discrete-time systems with delay and multiplicative noise. *IEEE Transactions on Automatic Control*, 60(10):2599–2613, 2015.