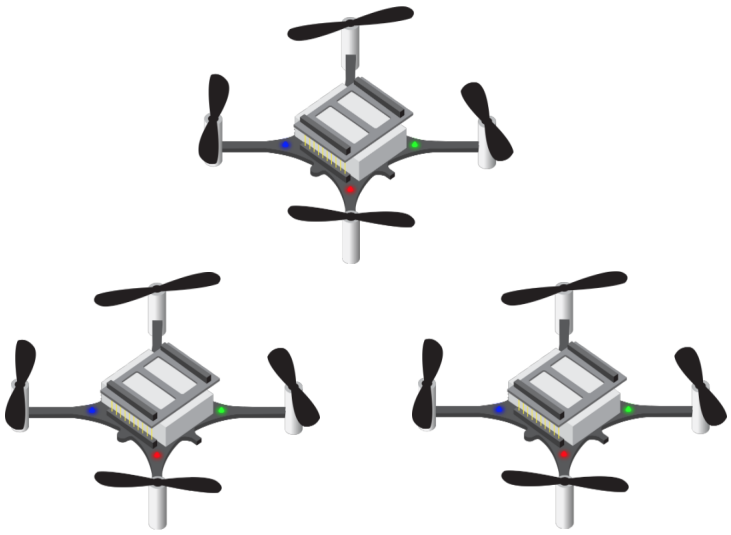
Distributed Control of Flying Quadrotors

Aren Karapetyan

University College

Fourth Year Project
University of Oxford



Supervised by **Prof. Antonis Papachristodoulou** and **Prof. Kostas Margellos**

Acknowledgements

I would like to express my sincere gratitude to Antonis Papachristodoulou and Kostas Margellos for their unlimited support and encouragement. I am also very grateful to everyone at the Oxford Control Group for their help and advice when I needed it. A special thank you to DPhil students Licio Romao, George Pantazis, Idris Kempf and Shuhao Yan for their support, and Prof. Paul Goulart for his help with the optimisation solver. I am thankful to my tutors Steve Collins, John Morton and Tom Povey for their help and support throughout my studies at Oxford.

Abbreviations

ADMM Alternating Direction Method of Multipliers.

IMU Inertial Measurement Unit.

LQR Linear-Quadratic Regulator.

MPC Model Predictive Control.

OSQP Operator Splitting Quadractic Program.

PI Proportional, Integral.

PID Proportional, Integral, Derivative.

PWM Pulse Width Modulation.

ROS Robot Operating System.

SMC Sliding Mode Control.

USB Universal Serial Bus.

Abstract

This research project presents the derivation of a linearised model for Crazyflie quadrotors and the implementation of a distributed, collision free trajectory tracking algorithm using the developed model. A cascade control law is implemented for the linearised model of the mini quadcopter. This structure enables the creation of two control modes, on-board and off-board. These are used depending on the requirements of the user. The quadrotors are localised using an optical flow based sensor mounted on each one of them. This makes the complete independence of the network from a central authority and a central positioning system, such as motion capture systems, possible. A Linear-Quadratic Regulator (LQR) optimal controller is implemented using the off-board configuration achieving stable flight in an indoors environment. A collision avoidance trajectory tracking algorithm is developed and solved for the Crazyflie developed dynamics. The nonconvex problem of collision avoidance is solved using a successive Taylor series linearisation around some nominal trajectories achieving convergence in milliseconds. The same problem is set up to be solved for in a distributed fashion using an algorithm based on the Alternating Direction Method of Multipliers (ADMM). Both the centralised and distributed problems are then solved using a receding horizon approach to introduce feedback into the system and allow an even faster computation due to the shorter horizon window size. The optimal inputs, obtained by solving these problems, generate collision free trajectories in simulation and are applied in an open loop fashion to the drones. The predicted and observed trajectories were compared to demonstrate the high accuracy of the derived model.

Contents

1 Introduction

1.1 Project Background and Motivation

Quadrotors or drones, as they are usually referred to, have been studied extensively for at least the last nine decades. It is almost undisputed that drones have their place in today's notions of the future day planet. In fact, even today we all witness small steps towards their integration in our mundane tasks with some of the most advanced camera equipped drones being used for search and rescue tasks [1] or heavy-sized ones used for irrigation and other agricultural purposes [2], to just name a few. With all this progress made so far, we start to wonder what else is possible. If we have a single drone now executing a delivery from a given origin to a destination in a completely free air space or another one irrigating a vast field on its own, then very soon this may change. The increase of flying machines means the airfield may soon be filled with an army of quadrotors, some with the same objective, some with completely different ones and possibly a few malfunctioning or noncooperative ones.

A particularly interesting example of this so called drone system is their use in urban environments for human and cargo transportation. Moreover, if we have traffic then we need a system in place to manage it. This becomes a control system task which can be tackled in a number of ways. Operating such a system in a centralised fashion may have several shortcomings, including it not being robust to attacks on or failures of the central system, its limited scalability, need for identification of every single agent and the disclosure of individual information, which may be private, to the entire network [3]. We thus turn to a solution requiring a complete independence of all of the agents, i.e. the quadrotors, from any centralised systems; in control terminology we are dealing with a distributed control problem.

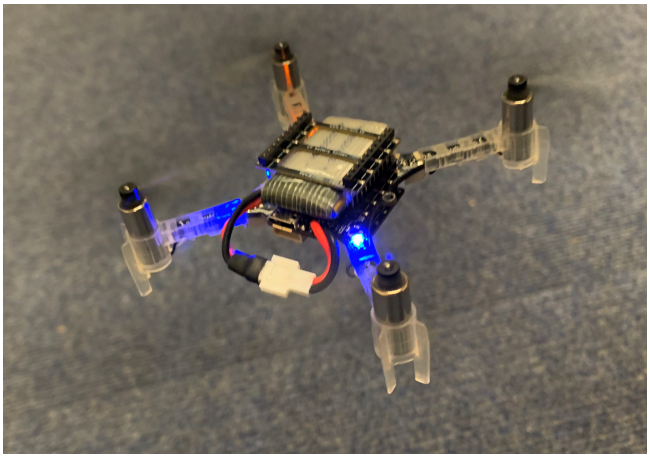


Figure 1.1: Crazyflie 2.0 with Flow Deck V2

The Crazyflie nano-quadcopters, shown in Figure 1.1 are developed by Bitcraze [4] and present an ideal test-bed for the development and testing of such control algorithms indoors. The open-source software [5] provides the opportunity of direct and low level control, and we aim to make use of this to derive a linearised model of these copters, develop centralised and distributed algorithms for their trajectory tracking and collision avoidance, and finally implement these in practice. The developed algorithms can then be implemented on a much larger scale for a number of applications,

as discussed.

1.2 Project Goals

As we established, the aim of the project is to build a distributed system independent of any central computations. This is to be achieved in steps, including the derivation of an accurate linear model for the Crazyflies, development of a centralised system that solves the required problems, and finally its distribution. These steps set the path for the rest of this report and are detailed below:

1. Develop a linearised mathematical model for the quadcopter and a controller designed to stabilise it locally. Use a camera based method to perform state estimation for the quadcopters and feed the data to the firmware at a high rate. The drone has to perform a stable flight along the reference (position or velocity) trajectory fed in by the user.

2. Develop a centralised controller to drive a group of Crazyflies to perform collective tasks, such as formation and trajectory tracking. The quadcopters should communicate at a reasonably high rate with the aggregator, i.e. a centralised authority supplying enough information (full state, position and/or velocity), such that collision avoidance and other local and coupled constraints are satisfied.

3. Replace the centralised algorithm by a decentralised one, distributing all the computation among the drones, eliminating the need of an aggregator. The collective tasks defined in the point above should still be achieved, specifically the collision avoidance constraint which should be enforced among all agents.

1.3 Report Layout

We provide a literature review in the next section, followed by the discussion of our technical setup in Section 3 explaining the details of the used sensors, the networked setup of the system, the technical details and justifications for the choices made. Following our defined goals we continue to the elaboration of a quadrotor's dynamics and control in Section 4. This section sets the cornerstone for the rest of the report by establishing the lower level of the control stabilising each drone for hover. After the linearised model of the Crazyflie is obtained, the multi-agent problem formulation and setup are presented in Section 5 and the best algorithm to fit our purposes is chosen. The centralised and distributed problems of collective trajectory tracking with collision avoidance, are solved in Sections 6 and 7, respectively. These are also implemented in a receding horizon implementation with Model

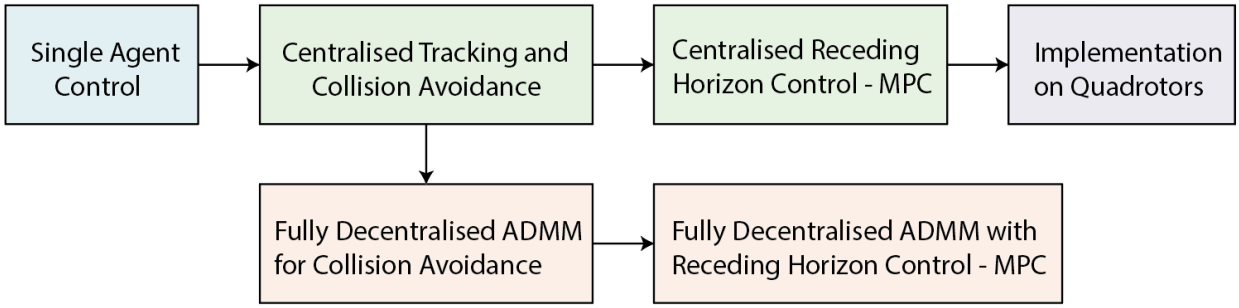


Figure 1.2: Project Structure

Predictive Control (MPC). Finally, in Section 8, the implementation of the algorithms, detailed in previous sections, in practice on the Crazyflies is described, with a conclusion in Section 9 to sum up the results. The development from single agent dynamics to centralised and distributed problems and finally to practical implementation is illustrated in Figure 1.2 above. Each preceding step acts as a building block for the later ones to be built upon.

2 Literature Review

Vertical take off and landing (VTOL) vehicle control has been studied extensively in the literature and a number of different methods have been developed [6] particularly for quadrotors, as they are becoming so increasingly more popular and applicable in daily lives. Probably the most widely used type of controller in the industry, the Proportional, Integral, Derivative (PID) controller has been applied to the quadrotor problem following linearisation of a drone model [7], achieving zero steady state error and small overshoot. Such linear models have also been used to develop optimal controllers, such as the Linear-Quadratic Regulator (LQR) [8, 9], achieving a stable flight by considering the attitude and position control problems separately. Lyapunov stability theory based non-linear control methods, such as Sliding Mode Control (SMC) [10] or integral (backstepping) control [11] have also been successfully applied driving the quadcopter to the required location at the required yaw angle.

Most of the developed techniques are also directly applicable to Crazyflie quadrotors. Their platform has been used for the design and implementation of various controllers, such as LQR low level controller [12], PID and Geometric controllers [13] and non-linear Lyapunov theory based approaches [14]. The recent advancements in computational power also gave rise to a number of learning and data based methods. These have been applied to quadrators and specifically to Crazyflies with great success, using deep neural networks [15], reinforcement learning [16] and data-based predictive control [17].

The problem of several agents in the same network solving a problem upon agreeing on some

parameters is known as the consensus problem and has been studied extensively [3, 18]. Concepts, such as flocking [19, 20] and formation control [21, 22] problems are directly applicable for our aim of controlling a swarm of drones. As it is discussed in Section 5.2, most of these feedback based approaches assume simple dynamics and constraints, making discrete iterative based decentralised [23], and distributed [24, 25, 26] algorithms a more applicable approach to the problem of collision avoidance trajectory tracking.

A number of projects are aimed at the networked control of Crazyflies, such as [27, 28]. For most of these projects the control is implemented from a central station and for collision avoidance potential methods are used. A centralised motion capture camera system is used to localise the agents, thus introducing a central aggregator and eliminating the possibility of fully distributed control. Others, such as [29], use two-way ranging based sensors which depend on carefully placed sets of anchors in the flying area. As discussed in Section 3, our proposed solution for the Crazyflie distributed collision avoidance trajectory tracking problem includes the application of on-board optical flow based sensors, independent of any other systems. These localise the agents with respect to their starting positions.

3 Technical Setup

To implement single agent, centralised and distributed algorithms in practice, we use the Crazyflie 2.0 quadrotors, just like the one shown in Figure 1.1. As discussed, their small size ($9\text{cm}^2, 27$grams [4]) allows us to perform our tests in an indoors confined environment. In this section we briefly describe how we set up the platform, including the used firmware and the localisation method used. Given that the reader has access to the developed software [30, 31, 32] and hardware [4] and has setup both the physical and computer environments as detailed below, the achieved results can be reproduced and possibly further improved.

3.1 Software And Data Collection

The high level structure of the Crazyflie system is simple. There are two microcontrollers onboard the copter [33], one carrying out tasks like controlling the power supply or USB port control and the other receiving and analysing sensors, actuating inputs to motors and reading and writing to the memory. While both of these chips contain editable firmware, we are mainly concerned with the latter, where the flight control takes place, and it will be referred to as firmware from this point on. As it is discussed in Section 4.3, depending on the type of control we use, there are always certain operations taking place in the quadcopter firmware and data needs to be communicated with the central computer, either to

close a loop or to receive references. This communication takes places through a radio module called CrazyRadio PA [34], which comes with its own firmware and features a 20dBm power amplifier, giving a range of up to 1km. It connects to the computer using a USB, operating at 2.4GHz frequency. The necessary libraries to be installed on the computer to connect and control the Crazyflie are provided by Bitcraze on their GitHub page [5].

The tests are carried out using a Linux Ubuntu 16.04 based central computer. We use the modified firmware [30] for the off-board control, discussed in Section 4.3.1 and the latest official firmware release, version ”2020.02” [36] for all the other tests. The details of the setup for swarm control are presented in Section 8.1. The inter-program communication and data collection from the Crazyflie quadrotor is supported using the Robot Operating System (ROS) framework, in particular the Kinetic Kame distribution [37].

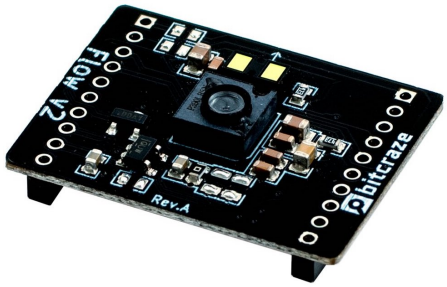


Figure 3.1: Flow Deck V2 [35]

3.2 Localisation

Distributed control requires complete independence from any central systems. Having a motion capture system to localise the drones will violate this condition. This is why we decided to equip all of the drones with an optical flow based sensor, called Flow Deck V2 [35], by Bitcraze. As shown in Figure 3.1, the sensor is mounted underneath each Crazyflie. The Kalman filter in the firmware of the Crazyflies makes use of the velocity gained through the photographs taken by the optical flow sensor and retrieves the position coordinates using its observer structure. For the height measurement, the Flow Deck is equipped with a light sensor that estimates the position by calculating the duration of the travel of a light beam. This in-situ implementation also has the advantage of having almost no communication delays and operating in the firmware.

4 Single Quadrotor Dynamics and Control

Controlling a network or a swarm of quadcopters is not possible without understanding the individual agent dynamics. In this section, we analyse the non-linear equations of motion for quadrotors and their linearisation. This model enables the development of a cascaded control law with different frequencies of operation in each loop.

Two control modes: on-board and off-board are introduced. In the first one, the low level control takes place on board, and the computer operates the high level controller generating some reference inputs to the flying system. The second approach provides a much lower level of control, where controllers can be developed to actuate the angular rates directly. The advantages and drawbacks of both types are discussed and a Linear-Quadratic Regulator (LQR) controller is implemented.

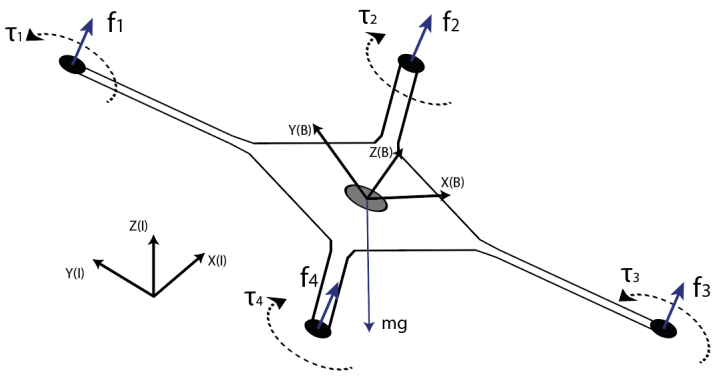


Figure 4.1: Forces acting on the Crazyflie

4.1 Equations of Motion

The dynamics of the drone, although simple, are still non-linear and need to be analyzed before a controller may be designed. Figure 4.1 above shows the forces acting on the copter, as well as the used inertial and body frames of reference, marked by (I) and (B) respectively. Following the derivation of ETH Zürich Automatic Control Lab's D-FALL project [12] the three equations of motion below accurately capture the evolution of a quadcopter under the influence of the four motor thrusts:

$$\ddot{\vec{p}} = \begin{bmatrix} \ddot{p}_x \\ \ddot{p}_y \\ \ddot{p}_z \end{bmatrix} = \frac{1}{m} \left(\boldsymbol{R}_{(I)}^{(B)}(\vec{\psi}) \begin{bmatrix} 0 \\ 0 \\ \sum_{i=1}^{4} f_i \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} \right), \tag{4.1a}$$

$$\ddot{\vec{\psi}} = \begin{bmatrix} \ddot{\gamma} \\ \ddot{\beta} \\ \ddot{\alpha} \end{bmatrix} = \boldsymbol{T}^{-1}(\vec{\psi})(\dot{\vec{\omega}} - \dot{\boldsymbol{T}}(\vec{\psi}, \dot{\vec{\psi}})\dot{\vec{\psi}}), \tag{4.1b}$$

$$\dot{\vec{\omega}} = \begin{bmatrix} \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} = \boldsymbol{J}^{-1} \left(\begin{bmatrix} \sum_{i=1}^{4} f_i y_i \\ \sum_{i=1}^{4} -f_i x_i \\ \sum_{i=1}^{4} f_i c_i \end{bmatrix} - (\vec{\omega} \times \boldsymbol{J}\vec{\omega}) \right), \tag{4.1c}$$

with the vector $\vec{p}$ being the position vector in inertial frame, f_i the force acting on the i^{th} motor, m the mass of the copter, $\boldsymbol{J}$ the moment of inertia matrix, x_i and y_i the horizontal and vertical distances between the center and the blades, c_i the linear coefficient of blade drag, $\vec{\omega}$ the body rates vector and $\vec{\psi}$ the Euler angles, of the form $\begin{bmatrix} \text{roll} & \text{pitch} & \text{yaw} \end{bmatrix}^T$. $\boldsymbol{R}_{(I)}^{(B)}(\vec{\psi})$ is a rotation matrix transforming from the body frame to the inertial one. The above are derived using Newton's second law for the translational and rotational cases respectively and the body rates and Euler angles are related by Equation (4.1b),

with $\boldsymbol{T}(\vec{\psi})$ being a transformation matrix defined below:

$$\boldsymbol{T}(\vec{\psi}) = \begin{bmatrix} 1 & 0 & -\sin(\beta) \\ 0 & \cos(\gamma) & \sin(\gamma)\cos(\beta) \\ 0 & -\sin(\gamma) & \cos(\gamma)\cos(\beta) \end{bmatrix}. \tag{4.2}$$

4.2 Linearised Models

4.2.1 Cascaded Structure

Before a controller can be designed for this system we have to retrieve its linearised version, and to do so we need to choose an equilibrium point around which that should be performed. For any system $\dot{x} = h(x, u)$, we can linearise around an equilibrium operating point (x^0, u^0) using first order Taylor series approximation and small deviations from this point,

$$\dot{x} \approx h(x,u) = h(x^0,u^0) + \left(\frac{\delta h}{\delta x}\right)^T\Bigg|_{x^0,u^0}(x-x^0) + \left(\frac{\delta h}{\delta u}\right)^T\Bigg|_{x^0,u^0}(u-u^0). \tag{4.3}$$

This can be then written in a matrix form, noting that the first term is 0 as defined by the equilibrium conditions, and by defining the deviations from this point by δx and δu for state and input respectively,

$$\dot{x} = \boldsymbol{A}\delta x + \boldsymbol{B}\delta u, \tag{4.4}$$

where $\boldsymbol{A}$ and $\boldsymbol{B}$ are the matrices arising from the second and third terms of Equation (4.3) respectively.

The most natural point of equilibrium around which to perform the linearisation of the system is the hovering position of the drone with $\dot{\vec{p}} = 0$, $\vec{\psi} = 0$ and $\dot{\vec{\psi}} = 0$. Now, defining f_{total} to be the total force acting on the drone, i.e. the sum of the four motors, and τ_x,τ_y and τ_z to be the torques acting on the three body axis as shown in Figure 4.1, we can represent a hover state in the following matrix form,

$$\begin{bmatrix} f_{total} \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ y_1 & y_2 & y_3 & y_4 \\ -x_1 & -x_2 & -x_3 & -x_4 \\ c_1 & c_2 & c_3 & c_4 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} = \begin{bmatrix} mg \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{4.5}$$

The above, indeed, satisfies the equilibrium state conditions as expected when plugged into Equations (4.1a) and (4.1c). For the nonlinear system described by the set of Equations (4.1), we define the state to be the position of the drone in the inertial frame of reference, its rate of change, the Euler angles in the body frame of reference and their rates of changes. The inputs are simply the four motor thrusts. Then, applying Equation (4.3) to the nonlinear system and following the derivation in the Appendix A of the ETH D-FALL project [12] we achieve a linear state space model as formed below,

$$\begin{bmatrix} \delta\dot{\vec{p}} \\ \delta\ddot{\vec{p}} \\ \delta\dot{\vec{\psi}} \\ \delta\ddot{\vec{\psi}} \end{bmatrix} = A_{hover} \begin{bmatrix} \delta\vec{p} \\ \delta\dot{\vec{p}} \\ \delta\vec{\psi} \\ \delta\dot{\vec{\psi}} \end{bmatrix} + B_{hover} \begin{bmatrix} \delta f_1 \\ \delta f_2 \\ \delta f_3 \\ \delta f_4 \end{bmatrix}. \tag{4.6}$$

In practice we have access to our state variables at different frequencies. For example, the body rates are received from the Inertial Measurement Unit (IMU) at $1000Hz$, while the position information is retrieved using the Flow Deck at $200Hz$. This means that a cascade controller has to be implemented. The controller is dividedinto two loops, the outer and the inner as shown in Figure 4.2 below. This division and cascade implementation can be justified by the assumption that the inner loop is infinitely fast, or in our case fast enough to close the loop before the next reference input from the outer loop is generated.

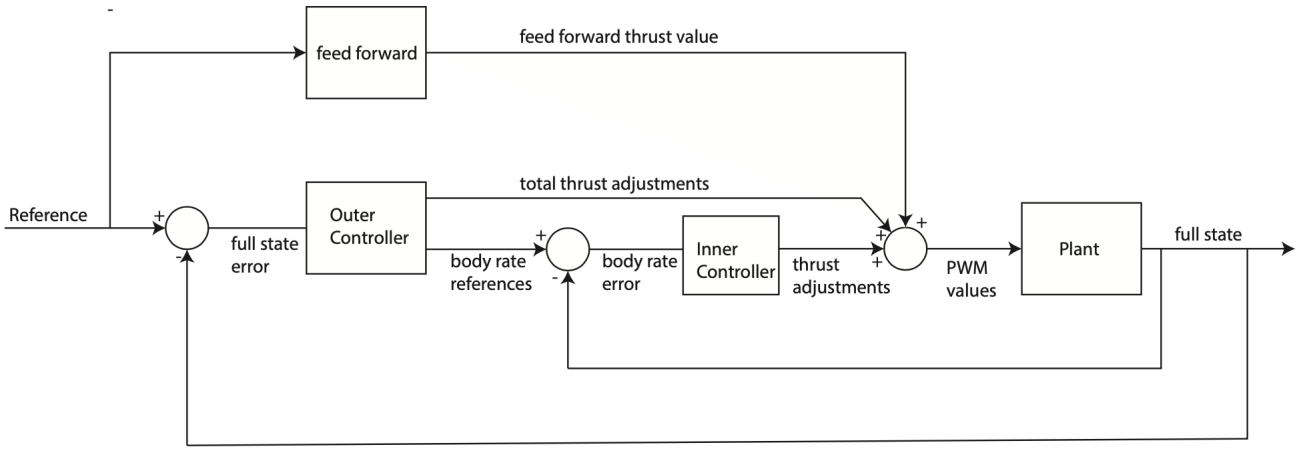


Figure 4.2: Crazyflie Cascade Controller Block Diagram

The outer loop has the quadcopter's position, velocity and the Euler angles as its state vector. The control inputs are then the total thrust, f_{total}, and the adjustments made to the three body rates around the body axis, formulated in (4.7a). The reference to the outer loop, theoretically, can be any of its states, as detailed in Section 4.3. The generated adjustments for the body rates by the outer loop are fed to the inner loop as reference signals, as shown in Figure 4.2. In this configuration, the states of this loop are defined to be these adjustment body rates and the control inputs are the four motor thrust adjustments of the torques around the three axis, as formulated in Equation (4.7b). Summed with the total and feed forward thrust adjustments these are transformed to voltage values using Pulse Width Modulation (PWM) and applied to the four motors.

$$\begin{bmatrix} \delta\dot{\vec{p}} \\ \delta\ddot{\vec{p}} \\ \delta\dot{\vec{\psi}} \end{bmatrix} = A_{outer} \begin{bmatrix} \delta\vec{p} \\ \delta\dot{\vec{p}} \\ \delta\vec{\psi} \end{bmatrix} + B_{outer} \begin{bmatrix} \delta f_{total} \\ \delta\omega_{x,ref} \\ \delta\omega_{y,ref} \\ \delta\omega_{z,ref} \end{bmatrix} \tag{4.7a}$$

$$\begin{bmatrix} \delta\dot{\vec{\omega}}_x \\ \delta\dot{\vec{\omega}}_y \\ \delta\dot{\vec{\omega}}_z \end{bmatrix} = A_{inner} \begin{bmatrix} \delta\vec{\omega}_x \\ \delta\vec{\omega}_y \\ \delta\vec{\omega}_z \end{bmatrix} + B_{inner} \begin{bmatrix} \delta\vec{\tau}_x \\ \delta\vec{\tau}_y \\ \delta\vec{\tau}_z \end{bmatrix} \tag{4.7b}$$

with,

$$A_{outer} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & g & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -g & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \qquad B_{outer} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.8a}$$

$$A_{inner} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad B_{inner} = \boldsymbol{J}^{-1}, \tag{4.8b}$$

where $\boldsymbol{J}$ is the mass moment of inertia matrix. Figure 4.2 also shows a feed-through term which is exactly the amount of total vertical force required to keep the copter hovering. This is naturally just the weight of the copter, as shown in Equation (4.5) and is precalculated. While it may change over the course of the flight due to the battery weight decrease, it has shown not to have any noticeable effects on the flight performance.

4.2.2 Full Linearised Model

Having a linearised model for the outer loop, an outer controller can be designed, as it is done in Section 4.3.1. However, as detailed in Section 4.3.2, aiming for a complete autonomy in the system, the low level controllers are moved to the drone firmware, on-board. To achieve this, we model and linearise the on-board Bitcraze implemented outer controller and join it with our outer level state space model (4.7a) to obtain the full linearised model of the drone. Thus, the outer state space representation (4.7a) will be used for off-board control in Section 4.3.1 and the full linearised model derived in this section will be used for the on-board control in Section 4.3.2 and later for the multi agent problem.

It can be seen from equations (4.7a) and (4.8a) that the controller structure can be decoupled into four separate controllers, namely height, yaw angle, x and y controllers. Moreover, as implemented by Bitcraze, the input signal to the controllers can be referenced by position, yaw angle or velocity. Below we discuss the designs of each of these controllers and our choice of references which will then become our control inputs for the controller integrated model. The blue lines in Figures 4.3 to 4.5 below show the error inputs to the controllers.

Figure 4.3 shows the outer controller block diagram for the height controller. The difference between the desired and the measured height from the Flow Deck Kalman estimator, $-\delta z = (z_0 - z)$, becomes the input to the controller. We choose the height instead of its rate of change to be the

input, as we plan to do most of the testing in two dimensions and we do not need a lower level input for this controller. However, this can easily be modified to a velocity input controller as it is done for the x and y controllers. There is a cascade of Proportional, Integral (PI) controllers. The first one is applied to the reference input to achieve the target vertical velocity. This is then compared with the state estimate of the current vertical velocity received from the IMU, $\dot{z}$, and is passed through the next PI controller and a constant gain $\boldsymbol{K_{th}}$ to get the total thrust force in motor command values. This is then transformed to Newtons using a quadratic equation [12] and added to the constant feed forward force value to generate f_{total}, the first input in the outer linearised model (4.7a).

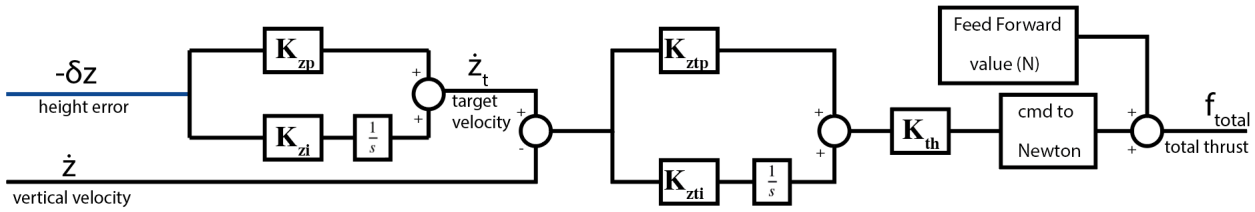


Figure 4.3: Outer Height Controller

In contrast to the height controller, reference velocities are used as inputs for the x and y ones as shown in Figure 4.4 below. This choice will allow a lower level and smoother control for the x and y directions. This has a similar structure to the height controller with a cascade of PI controllers. The first stage, called the velocity controller, receives the error between the reference and the estimated velocities (from the IMU), $-\delta\dot{x} = (\dot{x_0} - \dot{x})$ and $-\delta\dot{y} = (\dot{y_0} - \dot{y})$ and outputs the target pitch and roll angles, γ_t and β_t, respectively for the x and y controllers. Then the difference between the target and the estimated angles (obtained from the gyroscope) is passed through the next PI stage, called the attitude controller, outputting the pitch and roll body rates, ω_y and ω_x, respectively. It should be noted that there is a conversion happening from radians to degrees at the start of the stages and from degrees to radians at the end to match the units used on-board and in our calculations. Additionally, the pitch angles are pre and post multiplied by -1 to account for the different axis conventions used.

For the yaw angle controller, the desired yaw angle is simply set as reference and the input to the controller becomes the error between that and the measured angle obtained from the gyroscope, $-\delta\alpha = (\alpha_0 - \alpha)$, as shown in Figure 4.5 below. In this case we have a simple PID controller giving the required reference yaw body rate ω_z. Like in the previous controller, we perform angular unit conversions at the start and the end.

Summing up, we have four states that are used as references with this setup, specifically, the height z_0, the x and y velocities $\dot{x}_0$, $\dot{y}_0$ and the yaw angle α_0. The other inputs to the controller, such as the roll and pitch angles or the vertical velocity, are state measurements and we do not

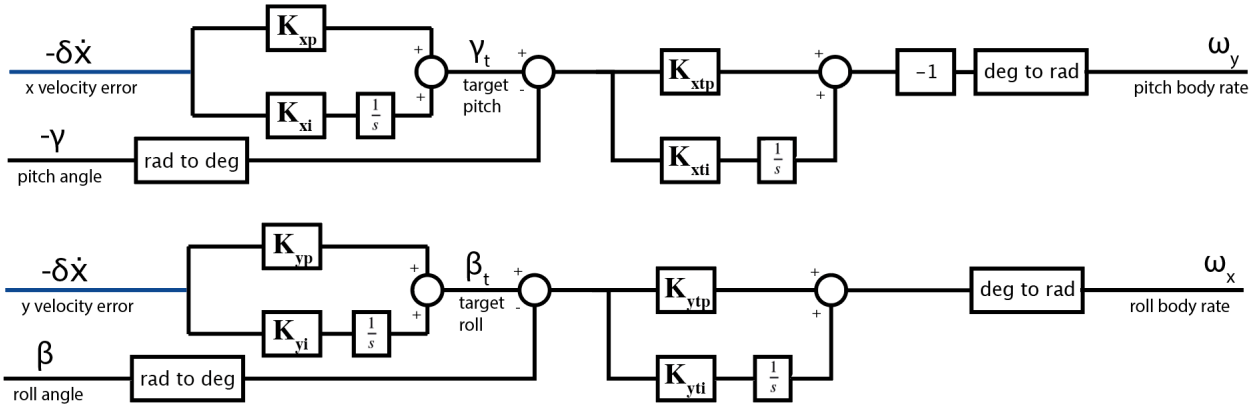


Figure 4.4: Outer X,Y Controller

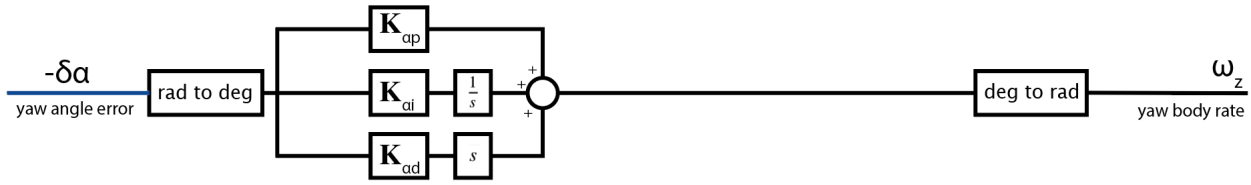


Figure 4.5: Outer Yaw Controller

directly reference those. We construct a SIMULINK model for the four controllers defined above and linearise it using MATLAB's "linmod" function. The resulting linearised state space representation of the controller is then joined with the outer level state space system (4.7a) in feedback using the "connect" function, just like it is shown in Figure 4.2. This finally gives us the full linearised model of the Crazyflie drone, represented by Equation (4.9). The state vector s_f comprises of 16 states for the system, with nine belonging to the outer state space model in (4.7a), and seven introduced by the controller. The control input vector u_f comprises of nine states, however only the referenced states, height, yaw and (x,y) velocities are controlled, while the others are set to zero to achieve negative feedback.

$$\begin{aligned} \dot{s_f} &= \boldsymbol{A}_f s_f + \boldsymbol{B}_f u_f \\ y_f &= \boldsymbol{C_f} s_f \end{aligned} \tag{4.9}$$

The resulting $\boldsymbol{A}_f$ and $\boldsymbol{B}_f$ matrices, thus have 16×16 and 16×9 sizes, and the matrix $\boldsymbol{C}_f$ is of dimension 9×16, with the first 9×9 being an identity matrix for the plant states and the rest a null matrix for the controller ones. We plug in the numerical values for the gains used in the four part controller by Bitcraze [36], shown in the Table 1 below, as well as the mass of the quadcopter for the plant model which we set to 29 grams with the Crazyflie and the flowdeck weighing 27 and 2 grams each. We note that the gains for the y controller are the same as for the x one, as expected due to symmetry. To prove the stability of the system, we check in the eigenvalues of the state space matrix $\boldsymbol{A}_f$, confirming they are all negative or zero. We have only two poles at the origin which appear when

K_{zp}	K_{zi}	K_{ztp}	K_{zti}	K_{xp}	K_{xi}	K_{xtp}	K_{xtp}	$K_{\alpha p}$	$K_{\alpha i}$	$K_{\alpha d}$
2	0.5	25	15	25	1	6	3	6	1	0.35

Table 1: Numerical Values for the Cascaded Outer Controller

we check the transfer function from the reference $\dot{x}_0$ and $\dot{y}_0$ to x and y positions respectively, which is expected as we anticipate an integral term from velocity to position.

4.3 Implementation

With an understanding of the dynamics of a quadrotor, a low level control architecture of Crazyflies and a linearised model in our hands we can now think about developing and implementing controllers in practice. At this point, using the nested structure of the Crazyflie model as shown in Figure 4.2 we distinguish two modes of controlling the Crazyflies, namely off-board and on-board and discuss both in detail below.

4.3.1 Off-Board Mode

In this mode of operation, also implemented by [12], the fast inner loop controller is left to run on-board, as implemented by Bitcraze in their firmware [36] and an outer controller is designed and ran off-board on a computer. With fast enough communication links in place, this way the outer loop is closed on the computer instead of the on board microcontroller. As the structure of the outer loop suggests, the controller running on the computer receives the current full state s_{fs}, as defined in Equation (4.10), from the drone and performs the computation to output the total thrust and the three reference body rate adjustments as input deviations, δu, defined in Equation (4.11). This is combined with the equilibrium feed forward input u^0 to form the control input u and is sent the drone, as shown in Figure 4.6 below. The terms $\boldsymbol{K_o}$, $\boldsymbol{K_i}$ and $\boldsymbol{G}$ in the figure represent the outer and inner controllers and the plant models respectively.

$$s_{fs} = \begin{bmatrix} \vec{p} \\ \dot{\vec{p}} \\ \vec{\psi} \end{bmatrix}, \qquad (4.10) \qquad\qquad \delta_u = \begin{bmatrix} \delta f_{total} \\ \delta\omega_{x,ref} \\ \delta\omega_{y,ref} \\ \delta\omega_{z,ref} \end{bmatrix}. \qquad (4.11)$$

As discussed in Section 3, we use the Flow Deck optical flow based sensor to localise the drone, which is mounted directly below the drone and the software for it is integrated right in the firmware. To control the Crazyflie with a low level actuation, such as the body rates and the total thrusts, the firmware needs to be reconfigured. Otherwise, the modes of control are high level, such as ”take-off” or ”land”, position, trajectory or velocity references. As mentioned before, with the assumption of an

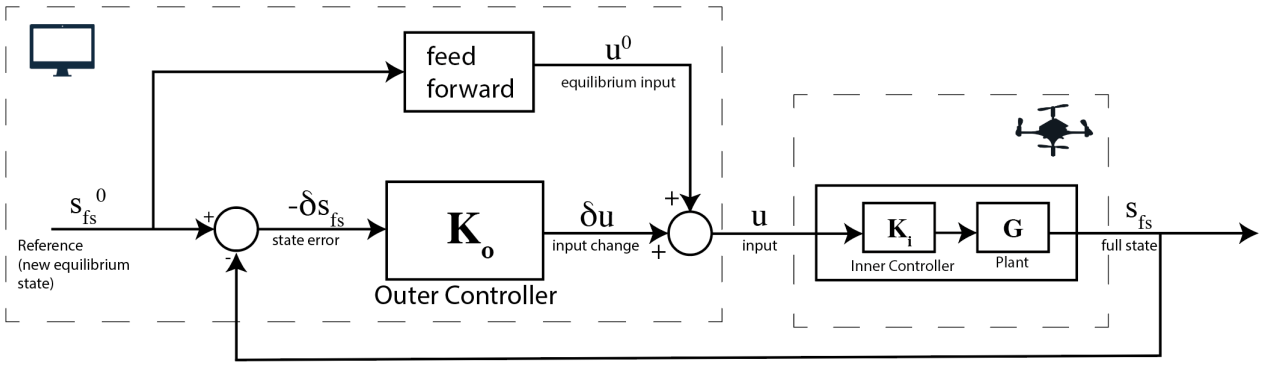


Figure 4.6: Off-Board Control Block Diagram

infinitely fast inner loop we can consider only the outer loop linearised state space representation of the model in (4.7a). Using this we can then develop a controller to satisfy user requirements, such as robustness or optimality. A LQR, for example, can be designed by solving the minimization problem in Equation (4.12) below and running the controller according to the described off board scheme. Each reference provides a new equilibrium state, s^0_{fs} as a reference for the model and the regulator drives the deviations from this state, i.e. $\delta s_{fs} = s_{fs} - s^0_{fs}$ to zero, achieving a tracking behaviour controlled by the design matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$.

$$\begin{aligned} \min_{\delta s_{fs}, \delta u} \; & \boldsymbol{J}(\delta s_{fs}, \delta u) = \frac{1}{2} \int_0^\infty (\delta s_{fs}(t)^T \boldsymbol{Q} \delta s_{fs}(t) + \delta u(t)^T \boldsymbol{R} \delta u(t)) dt \\ \text{subject to } & \dot{\delta s}_{fs} = \boldsymbol{A}_{outer} \delta s_{fs} + \boldsymbol{B}_{outer} \delta u \end{aligned} \tag{4.12}$$

Solving the Algebraic Riccati equation gives a feedback solution for the above problem with the law $\delta u = -\boldsymbol{K} \delta s_{fs}$, with $\boldsymbol{K} = \boldsymbol{R}^{-1} \boldsymbol{B}^T \boldsymbol{P}$ and $\boldsymbol{P}$ being a matrix dependent on the eigenvectors of the Hamiltonian Matrix. The Riccati equation for the linearised outer loop dynamics is solved and a ladder step response test with the off-board configuration in Figure 4.6 in place is performed. Figure 4.7 shows the evolution of the height of the drone over time under the step response, with the blue curve showing the reference, the red the expected response from the simulated non-linear dynamics and the orange curve the measured drone signal collected using the ROS system. We see a a close

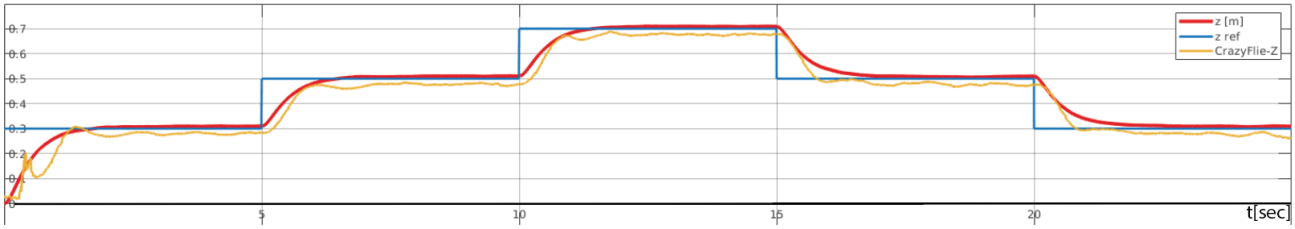


Figure 4.7: LQR Ladder Step Response under Off-board control Mode

match between the red and the orange curves proving the accuracy of the linearised model under simulated white noise. We see a staggered response at the start which is due to the error of the

Flow Deck sensor which exhibits such behaviour at heights under $20cm$ and more then $300cm$ [38]. The existence of the steady state error is a proof that there is no integral action involved as we are only including a state gain feedback. However, the observed maximum height error, a combination of steady state and sensor errors, was never more than $2cm$ for hovers at heights of $30cm$ or higher.

4.3.2 On-Board Mode

While the off board control mode is suitable for a single quadrotor control and for computations requiring larger memory sizes and faster processors, it fails when we add the requirement of distribution. With the final objective being the complete independence of the agents from each other and from any central authority, the off board control mode introduces a central computer and adds the need for fast communication. For these reasons, we introduce an on board approach, where we model the dynamics of the plant together with the inner and outer controllers of Bitcraze , as was done in Section 4.2.2 and leave the computation to be done on board the Crazyflie. We step a level higher and design a higher level controller which will generate optimal reference inputs u_f. This high level controller $\boldsymbol{K}_h$ is to bedesigned using the full linearised model in (4.9). Figure 4.8 below shows the

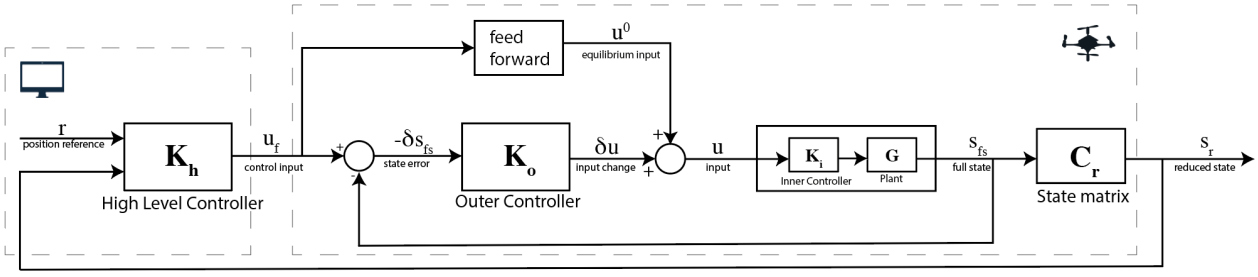


Figure 4.8: On-Board Control Block Diagram

on-board implementation diagram that we propose, where the outer and inner controllers are run on-board. The state matrix $\boldsymbol{C_r}$ outputs a reduced state vector s_r which is used to introduce a feedback loop in the design, for instance in a Model Predictive Control (MPC) fashion. We choose the higher level reference to be a position trajectory r, according to our defined goals in Section 1.2. The reference state s^0_{fs}, now becomes the control input u_f as defined in Equation (4.9), which is communicated to the drone from the high level controller running on a computer. This can also later be transferred on-board the Crazyflie microprocessor as discussed in Section 9.1.

In summary, this mode of control allows one to perform a high level control for the Crazyflies with a linearised model for a quadcopter's dynamics that integrates both the outer and inner controllers. We verify this full linearised model (4.9) and the on-board proposed control strategy for a swarm of drones in Section 8.

5 Cooperation and Coordination in Networked Multi-Agent Systems

Having derived a linearised model for the closed loop single agent system we can now think about possible methods for the distributed control of a swarm of drones. The field of multi agent coordination has been widely researched from different perspectives and for different purposes, such as the average consensus problem [18, 39, 3] , flocking [19, 20], formation control [21, 22] or collision avoidance [40, 24] to just name a few. Following the objectives defined in Section 1.2, the possible approaches that can be made use of to solve the collision avoidance and trajectory tracking problems for a group of Crazyflies are discussed to choose an algorithm best suitable for our purposes.

5.1 Networked Dynamical Systems and Consensus

Dynamical systems are concerned with modelling natural systems that have features evolving over time, such as the linearised model of a quadrotor we developed earlier. When we have a network of such systems, with some sort of communication topology in place we can develop control algorithms which will make use of this exchange of information to fulfill local or common goals. When dealing with networks the notation and properties developed in graph theory become very useful [41]. A graph $\boldsymbol{G}$ consists of a set of nodes $\boldsymbol{V} = \{1, \dots, M\}$ and a set of edges $\boldsymbol{\mathcal{E}} = \{(i,j), i,j \in \boldsymbol{V}\}$. Undirected and directed graphs are distinguished by the fact that for the former ones if there exists an edge (i,j), then there must be a (j,i) edge as well, while for the latter it is not a necessary condition. For our purposes we assume that all communications are reciprocal, thus we will only be concerned with undirected graphs, like in Figure 5.1, where the edges have no direction. We can define a number of important matrices for all graphs, such as the weighted adjacency matrix **A** capturing the weight of each of the edges.

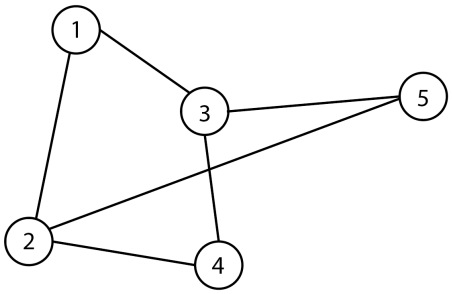


Figure 5.1: An Undirected Graph

$$\mathbf{A}_{ij} = \begin{cases} w_{ij}, & \text{if } (i,j) \in \boldsymbol{\mathcal{E}} \\ 0, & \text{otherwise} \end{cases} \tag{5.1}$$

We then define the degree matrix **D** of a graph, which is a diagonal $M \times M$ matrix with the number of edges entering or leaving the node i as the (i,i) entry. One of the most important matrices describing a graph, is called the Laplacian matrix, defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$, which is symmetric for undirected graphs. The Laplacian has a number of significant properties applicable for networked systems, such as:

- **L** is square and symmetric, therefore with real eigenvalues

- **L** is positive semi-definite, i.e. its eigenvalues are non-negative.
- $\mathbf{L}\mathbf{1}_M = 0$, i.e. 0 is an eigenvalue of **L** with eigenvector $\mathbf{1}_M$

This understanding of graph theoretical concepts and notation now allows us to represent our network of dynamical systems as a graph. In such a network each node in $\boldsymbol{V}$ represents an agent with dynamics and each edge represents a weighted connection between these agents, or its absence if the weight is zero. The idea of consensus algorithms is for the agents to agree on a variable they hold. For simplicity we discuss agents with dynamics $\dot{x}_i = u_i$, connected in some sort of undirected graph, trying to agree on the state value x. A simple algorithm to achieve this is for each agent to steer its dynamics towards the weighted average of its neighbours:

$$\dot{x}_i = \frac{1}{d_i + 1} \left(\sum_{j=1}^{d_i} \boldsymbol{A}_{ij}(x_j - x_i) \right), \tag{5.2}$$

where d_i is the number of neighbours of the agent i and $\boldsymbol{A}$ is the adjacency matrix of the graph. To understand the collective dynamics of the agents we have to consider the whole network as a dynamic system itself. The states x_i are concatenated into a vector x, and after transformations we see that the control law 5.2 can be represented in a matrix form as follows:

$$\dot{x} = -(\boldsymbol{D} + \boldsymbol{I})^{-1} \boldsymbol{L} x, \tag{5.3}$$

where **D** and **L** are the degree and Laplacian matrices of the graph respectively. $(\mathbf{D} + \mathbf{I})$ is a positive definite matrix and all the eigenvalues of **L** are negative with the exception of $\mathbf{L}\mathbf{1}_M = 0$. Therefore this control law achieves stability and all the agents converge to the same value of $x^\star$. Such consensus algorithms are explored further to incorporate new objectives such as collision avoidance between quadrotors, in the next section.

5.2 Flocking

Flocking is a phenomenon achieved in large animal, social or robotic groups where the interacting agents are trying to fulfill a common objective under certain individual and coupled constraints. The first visualisation of flocking behaviour and its three necessary rules were introduced by Reynolds [19]:

1. Cohesion: Agents follow a common objective for the flock, staying as close together as possible.
2. Collision Avoidance: Collisions between the agents in a flock are to be avoided.
3. Velocity Consensus: Agents match the velocities with their neighbours in the flock.

A flocking algorithm which implements all of the points above was introduced by Olfati-Saber [20].

Adding to the notations and concepts defined for graphs in Section 5.1, a set of neighbours of node(agent) i, $\mathcal{N}_i$, is defined based on the interaction range r of the agents, as formulated in Equation (5.4) and shown in Figure 5.2, where the sixth neighbour falls out of this defined radius r. This is the maximum range an agent is capable of transmitting and receiving data fast and reliable enough to stay safe and stable.

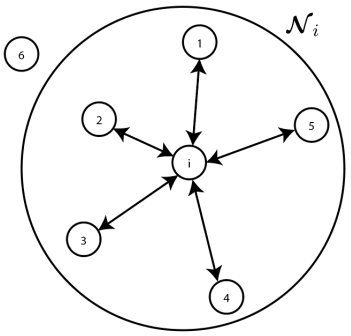


Figure 5.2: Spherical Neighbourhood

$$\mathcal{N}_i = \{j \in \boldsymbol{V} : \|x_j - x_i\| < r\} \tag{5.4}$$

The position of the agents is still denoted by x as in the previous section and $\|\cdot\|$ denotes the Euclidean norm throughout the report. In this algorithm the double integrator dynamics are considered, where the actuation is the acceleration. A distributed algorithm is then introduced [20] to achieve flocking whereby each agent's control input consists of the following three terms:

$$u_i = f_i^c + f_i^v + f_i^o, \tag{5.5}$$

where the first term is responsible for collision avoidance between the agents, the second one for velocity consensus and the last one for collective objective tracking. The velocity consensus term as expected will have a similar structure to the algorithm discussed in Section 5.1 with the velocity being the state the agents are trying to agree on. The collision avoidance, intuitively has to have a term involving the difference between the agents positions and the objective tracking a proportional gain controller trying to reduce the error between the measured and reference positions.

5.3 Limitations of Potential Field Approaches

The insight gained from consensus andn flocking algorithms is important for the clear formulation of our problem. However, there are a number of reasons that these approach of solving the collision avoidance, velocity consensus or formation problems, commonly referred to as potential field methods are often impractical. Firstly, most of the convergence and stability properties for such algorithms are proven for simple single or double integrator dynamical systems, while what we have is a larger linearised model with complex dynamics (4.9). While there is research conducted to solve the multi-agent problem, such as the consensus problem with general linear models [42], the main problem with such approaches is the difficulty to account for multiple objectives and integration of multiple local and coupling constraints, such as maximum allowable velocity constraints we might have on

the quadrotors or the hard constraints of collision avoidance. This is the reason that alongside the development of computational power, optimization-based coordination and control techniques are being developed. These often consider the discretized dynamics of the system and are implemented using receding horizon prediction techniques, such as MPC [43, 44]. In the next two sections we cover the two main classes of optimization-based iterative algorithms, in particular the Alternating Direction Method of Multipliers (ADMM) which is the main theoretical stronghold for the final algorithm we implement on our network of quadrotors.

5.4 Decentralised Algorithms

The need for inclusion of complex, local and coupling constraints in the control of complex networked systems makes the need of optimization based methods apparent. The class of decentralised algorithms considers the network structure where there exists a central authority or an aggregator. It communicates some tentative information with the agents, thus linking those, as opposed to distributed algorithms, discussed in Section 5.5. There this central authority does not exist and the agents communicate with each other according to some graph representing their neighbourhoods, as explained in Section 5.1. The problem of collective trajectory tracking with collision avoidance constraints, that we are concerned with, can be formulated as a constrained coupled problem of the following form,

$$\begin{aligned} \min_{x} \quad & \boldsymbol{F}(x), \\ \text{subject to:} \quad & \boldsymbol{A}x = b \\ & x \in X, \end{aligned} \tag{5.6}$$

In this type of problems the coupling between the agents can be set up using the matrix $\boldsymbol{A}$ and vector b and by a careful choice of the variable vector x, as it is done later in Section 6.3. A number of algorithms have been developed to solve both constrained and unconstrained problems in decentralised fashion, such as the regularised Jacobi, Gauss-Seidel and Augmented Lagrangian algorithms [23]. One of the most widely used decentralised algorithms that solves the problem (5.6) is called the Alternating Direction Method of Multipliers (ADMM), due to its relatively fast convergence and simplicity. Many algorithms are build upon it, specialising for various setups and purposes. As we are using a fully decentralised version of it in Section 7, we will review ADMM's basic structure in the next section.

5.4.1 Alternating Direction Method of Multipliers (ADMM)

When dealing with a network of cooperative agents parallelism and efficiency are crucial, so that we make use of the possibilities brought in by the network, and ADMM does exactly that. The general formulation of the algorithm given for the problem (5.6), is as follows,

$$\begin{aligned} \min_{x,z} \quad & \boldsymbol{F_1}(x) + \boldsymbol{F_2}(z), \\ \text{subject to:} \quad & x \in C_1,\ z \in C_2 \\ & \boldsymbol{A}x = z. \end{aligned} \tag{5.7}$$

The ADMM algorithm is partially parallelisable and iterative. It is given in three steps for each agent i and iteration k, as follows,

$$x(k+1) = arg \min_{x \in C_1} \boldsymbol{F_1}(x) + \lambda(k)^T(\boldsymbol{A}x - z(k)) + \frac{c}{2}\|\boldsymbol{A}x - z(k)\|^2 \tag{5.8a}$$

$$z(k+1) = arg \min_{z \in C_2} \boldsymbol{F_2}(z) + \lambda(k)^T(\boldsymbol{A}x(k+1) - z) + \frac{c}{2}\|\boldsymbol{A}x(k+1) - z\|^2 \tag{5.8b}$$

$$\lambda(k+1) = \lambda(k) + c(\boldsymbol{A}x(k+1) - z(k+1)), \tag{5.8c}$$

where $c > 0$ is a constant parameter, and λ is a Lagrangian variable. The algorithm is proven to converge if either the set C_1 is bounded or the matrix $\boldsymbol{A^T A}$ is invertible [23]. The convenient setup for our problem of networked quadrotors is the following constraint coupled problem for M agents [26, 23],

$$\begin{aligned} \min_{x_1,\ldots,x_M} \quad & \sum_{i=1}^{M} \boldsymbol{F}_i(x_i) \\ \text{subject to :} \quad & \sum_{i=1}^{M} \boldsymbol{A}_i x_i = \sum_{i=1}^{M} b_i = b \\ & x_i \in X_i \qquad i = 1, ..., M \end{aligned} \tag{5.9}$$

where $X_i \in \mathbb{R}^{n_i}$, $\boldsymbol{A_i} \in \mathbb{R}^{m \times n_i}$ and $b_i \in \mathbb{R}^m$. It is assumed that for all $i = 1, ..., M$, the function $\boldsymbol{F}_i$ is convex and the set X_i is convex and compact; and both the primal and dual problems have optimal solutions. To bring the problem to the ADMM format (5.7) we denote $\boldsymbol{A_i}x_i = z_i$ and then form the augmented Lagrangian problem as follows,

$$\boldsymbol{\mathcal{L}}(c, \lambda) = \sum_{i=1}^{M} (\boldsymbol{F}_i(x_i) + p_i^T(\boldsymbol{A}_i x_i - z_i) + \frac{c}{2}\|\boldsymbol{A}_i x_i - z_i\|^2), \tag{5.10}$$

where $p_i \in \mathbb{R}^m$ is the Lagrangian multiplier vector for agent i. Following (5.8), the first two steps of the iterative ADMM algorithm then yield,

$$x_i(k+1) = arg \min_{x_i \in X_i} \boldsymbol{F}_i(x_i) + p_i(k)^T \boldsymbol{A}_i x_i + \frac{c}{2}\|\boldsymbol{A}_i x_i - z_i(k)\|^2, \quad \forall i = 1, \ldots, M, \tag{5.11a}$$

$$z(k+1) = arg \min_{z: \sum_i^M z_i = b} -\sum_{i=1}^{M} p_i(k)^T z_i + \sum_{i=1}^{M} \frac{c}{2}\|\boldsymbol{A}_i x_i(k+1) - z_i\|^2, \tag{5.11b}$$

with the dual ascent equation for the Lagrangians p_i, given as,

$$p_i(k+1) = p_i(k) + c(\boldsymbol{A}_i x_i(k+1) - z_i(k+1)), \quad \forall i = 1, \ldots, M. \tag{5.12}$$

The iteration for (5.11b) can be solved analytically by introducing $\lambda \in R^m$ as Lagrange multipliers for

its constraint:

$$z(k+1) = arg\min_{z} -\sum_{i=1}^{M} p_i(k)^T z_i + \sum_{i=1}^{M} \frac{c}{2}\|\boldsymbol{A}_i x_i(k+1) - z_i\|^2 + \lambda^T \left(\sum_{i=1}^{M} z_i - b\right). \tag{5.13}$$

Solving the primal and dual problems of (5.13) for z and λ respectively, we find,

$$z_i(k+1) = \boldsymbol{A}_i x_i(k+1) + \frac{p_i(k) - \lambda(k+1)}{c}, \quad \forall i = 1, \ldots, M, \tag{5.14}$$

$$\lambda = \frac{1}{M}\sum_{i=1}^{M} p_i + \frac{c}{M}\left(\sum_{i=1}^{M} \boldsymbol{A}_i x_i(k+1) - b\right). \tag{5.15}$$

Comparing (5.14) with (5.12), we see that $p_i(k+1) = \lambda(k+1),\ \forall i = 1, \ldots, M$. Thus, substituting λ in (5.11-5.14) and eliminating $z_i(k)$ from (5.11a), we achieve the final form of the ADMM algorithm that generates an optimal primal solution and converges to optimal dual $\lambda^\star$ given that such exist [23],

$$x_i(k+1) = arg\ \min_{x_i \in X_i} \boldsymbol{F}_i(x_i) + \lambda(k)^T \boldsymbol{A}_i x_i + \frac{c}{2}\|\boldsymbol{A}_i x_i - \boldsymbol{A}_i x_i(k) + w(k)\|^2, \quad \forall i = 1, \ldots, M, \tag{5.16a}$$

$$\lambda(k+1) = \lambda(k) + cw(k+1). \tag{5.16b}$$

where,

$$w(k) = \frac{1}{M}\sum_{i=1}^{M}(\boldsymbol{A}_i x_i(k) - b_i). \tag{5.17}$$

It is obvious from the structure of the algorithm that a central authority is still required to process (5.16b) and (5.17).

5.5 Distributed Algorithms

As we aim to remove the necessity of an aggregator we next discuss distributed algorithms, in particular the tracking-ADMM [26], which builds on the discussed concepts and algorithms to fully distribute the computation among agents.

5.5.1 Tracking-ADMM

Tracking-ADMM considers the problem (5.9) and solves it without an aggregator [26]. A graph $\boldsymbol{G}$ is considered with vertices and edges $\boldsymbol{V}$ and $\boldsymbol{\mathcal{E}}$ as defined in Section 5.1. Moreover, the following assumptions are added to those mentioned in Section 5.4.1.

1. The graph representing the network of the M agents is undirected and connected, i.e. if there is a path from node i to node j, then the reverse has to be true as well.

2. For each edge there exists an associated weight $w_{i,j} \in [0, 1)$, such that $\sum_{i=1}^{M} \boldsymbol{A}_{ij} = 1$ for all

$j = 1, \ldots, M$ and $\sum_{i=1}^{M} \boldsymbol{A}_{ij} = 1$ for all $i = 1, \ldots, M$, where $\boldsymbol{A}_{ij}$ is the entry (i, j) of the adjacency matrix of the graph $\boldsymbol{G}$.

3. Define a consensus matrix $\boldsymbol{W} \in \mathbb{R}^{M \times M}$ whose $(i, j)^{th}$ entry is $\boldsymbol{A}_{ij}$. This matrix is symmetric, doubly stochastic and positive semidefinite.

The central unit's function in the decentralised ADMM case is updating the Lagrangian variable $\lambda(k)$ using the averaged term $w(k+1)$, calculated using the information obtained from all the agents, as defined in (5.16) and (5.17), respectively. To get an estimate of this averaged value and distribute $\lambda_i(k)$ and $w_i(k)$, tracking-ADMM makes use of a distributed dynamic average consensus mechanism by introducing the following iterative alogirithm for each agent i,

$$x_i(k+1) = arg \min_{x_i \in X_i} \boldsymbol{F}_i(x_i) + l_i(k)^T \boldsymbol{A}_i x_i + \frac{c}{2} \|\boldsymbol{A}_i x_i - \boldsymbol{A}_i x_i(k) + \omega_i(k)\|, \tag{5.18a}$$

$$w_i(k+1) = \omega_i(k) + \boldsymbol{A}_i x_i(k+1) - \boldsymbol{A}_i x_i(k), \tag{5.18b}$$

$$\lambda_i(k+1) = l_i(k) + c w_i(k+1), \tag{5.18c}$$

where, $\omega_i(k) = \sum_{j \in \boldsymbol{\mathcal{N}}_i} a_{ij} w_j(k)$, $l_i(k) = \sum_{j \in \boldsymbol{\mathcal{N}}_i} a_{ij} \lambda_j(k)$ and $\boldsymbol{\mathcal{N}}_i$ is the set of the neighbours of node i. For given initial conditions, the algorithm is initialised with $w_i(0) = \boldsymbol{A}_i x_i(0) - b_i$ for all agents. So, by communicating the values of $w_i(k)$ and $\lambda_i(k)$ only, [26] achieves full distribution of the ADMM algorithm (5.16) for problem (5.9). Given the mentioned assumptions, the iterative algorithm (5.18) achieves optimal values both for the primal and dual variables upon convergence.

To enforce collision avoidance with other agents Euclidean spheres with some safety radius Δ are placed around each agent. This constraint can be posed or each edge (i, j) in the graph $\boldsymbol{G}$ for a horizon window of N, as:

$$h_{ij}(x_i, x_j) = \min_{k=1,\ldots,N} \|x_i(k) - x_j(k)\|_2 - \Delta \geq 0 \quad \forall j \in \boldsymbol{\mathcal{N}}_i \text{ and } i \tag{5.19}$$

The set constraint clearly becomes nonconvex and hence violates the initial assumptions brought forward by the algorithms discussed earlier. The constraint needs to be linearised to make it convex. Hence, we turn to another distributed algorithm that makes use of ADMM and the introduced concepts for its setup and distribution but also incorporates the quadrotor's linearised dynamics, all discussed in the following section.

6 Centralised Problem Formulation

The discussion of multi-agent consensus problems leads us to an optimization-based distributed approach. This endows us with flexibility to choose local and collective constraints and guarantee convergence. The Fully Decentralized ADMM for Coordination and Collision Avoidance [24] uses the

ADMM algorithm to fully decentralize a collision avoidance and coordination optimisation problem using a communication topology. Before we set up and solve the distributed or the fully decentralised problem in Section 7, we first set the centralised trajectory tracking and collision avoidance problem and solve it both for general double integrator dynamics and for the full linearised model (4.9) of a Crazyflie quadrotor.

6.1 Modelling the Collision Avoidance Constraint

We consider a network of interconnected M agents which interact in some kind of graph topology as described in Section 5.1. We define N_i to be the number of neighbours of agent i, as formulated in Equation (5.4). For simplicity we consider r in (5.4) to be infinity such that we have a fully connected graph such that the number of agents in set $\boldsymbol{\mathcal{N}}_i$ as defined in Section 5.1 is $N_i = M - 1$, where each agent can communicate with all the rest. A constraint coupled problem is formulated, similar to (5.9),

$$\begin{aligned} \min_{p_i, v_i, a_i} \quad & \sum_{i=1}^{N} \boldsymbol{F}_i(p_i, a_i) \\ \text{subject to:} \quad & (p_i, v_i, a_i) \in \boldsymbol{S}_i \quad \forall i \\ & h_{ij}(p_i, p_j) \geq 0 \quad \forall j \in \boldsymbol{\mathcal{N}}_i \text{ and } i, \end{aligned} \tag{6.1}$$

where (p_i, v_i, a_i) are respectively the position, velocity and acceleration of agent i, $\boldsymbol{S}_i$ is a constraint set which incorporates the dynamics of each agent to be defined, and h_{ij} is a nonconvex collision avoidance constraint similar to the one defined in Equation (5.19).

We start by modelling simpler, discretized double integrator dynamics for each agent i as follows,

$$\begin{aligned} p_i(k+1) &= p_i(k) + T v_i(k) \\ v_i(k+1) &= v_i(k) + T a_i(k), \quad k = 0, 1, ..., N+1 \\ (p_i(k+2), v_i(k+1), a_i(k)) &\in (\boldsymbol{P}_i \times \boldsymbol{V}_i \times \boldsymbol{A}_i), \quad k = 1, ..., N, \end{aligned} \tag{6.2}$$

which is initialised with a current position $p_i(0)$, velocity $v_i(0)$ and acceleration $a_i(0)$ and where $T > 0$ is the discretization interval, $(\boldsymbol{P}_i \times \boldsymbol{V}_i \times \boldsymbol{A}_i)$ model any constraints in velocity, acceleration and position as mentioned earlier. The free decision variables, (p_i, v_i, a_i), are defined for some horizon window N, by vertically concatenating $p_i(k)$, $v_i(k)$ and $a_i(k)$, respectively. The set of these that satisfies the above constraints is denoted as $\boldsymbol{S}_i = \{(p_i, v_i, a_i) | \text{(6.2) holds}\}$. The acceleration a_i is the natural control input for these dynamics. The costs $\boldsymbol{F}_i$ are defined as a tracking LQR algorithm as follows,

$$\boldsymbol{F}_i(p_i, a_i) = (p_i - r_i)^T \boldsymbol{Q} (p_i - r_i) + (a_i)^T \boldsymbol{R}(a_i) \quad \forall i = 1, \ldots, M \tag{6.3}$$

where the position reference setpoints r_i, defined for a horizon window N, are tracked with $\boldsymbol{Q}$ and $\boldsymbol{R}$ design matrices. The choice of these matrices will affect how much the deviation from the reference states and control inputs will be penalised. The nonconvex collision avoidance constraint is modelled

as follows for a horizon window of N,

$$h_{ij}(p_i, p_j) = \min_{k=3,\dots,N+2} \|p_i(k) - p_j(k)\|_2 - \Delta \geq 0 \quad \forall j \in \boldsymbol{\mathcal{N}}_i \text{ and } i, , \tag{6.4}$$

where Δ is the minimum separation distance between the agents. To solve the problem of nonconvexity, following the strategy of [40], a first order Taylor series approximation is performed for agents i and j around some nominal trajectories $(\overline{p}_i, \overline{p}_j)$, defined later, as follows,

$$\begin{aligned} g_{ij}(k)\left(p_i(k), p_j(k)\right) &= \|\overline{p}_i(k) - \overline{p}_j(k)\| + \eta_{ij}^T(k)\left[(p_i(k) - p_j(k)) - (\overline{p}_i(k) - \overline{p}_j(k))\right] - \Delta, \\ \overline{h}_{ij}(p_i, p_j) &= \min_{k=3,\dots N+2} g_{ij}(k)(p_i(k), p_j(k)), \end{aligned} \tag{6.5}$$

where

$$\eta_{ij}(k) = \frac{(\overline{p}_i(k) - \overline{p}_j(k))}{\|\overline{p}_i(k) - \overline{p}_j(k)\|} \tag{6.6}$$

The above is illustrated in Figure 6.1 below, which shows the linear $g_{ij}(k)$ function for a particular time instant k and the safety circle with radius Δ around agent i at that time instant. The gray region shows the linearised safe area for the agent j to be. The problem (6.1) can now be reformulated with

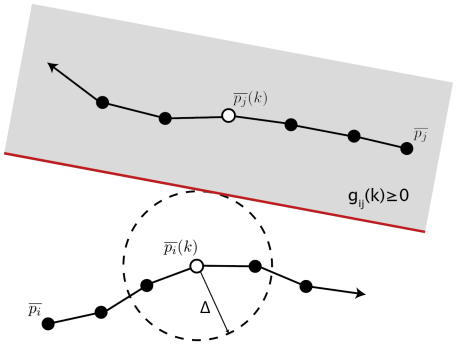
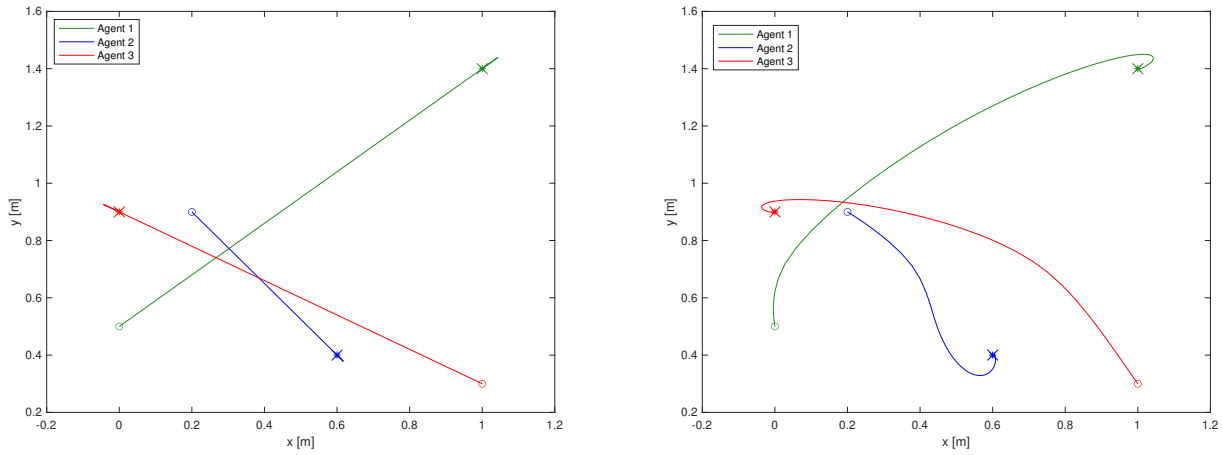


Figure 6.1: Linearization around nominal trajectories at time step k

the linearised $\overline{h}_{ij} \geq 0$ constraint making the problem convex, as follows,

$$\begin{aligned} \min_{p_i, v_i, a_i} \quad & \sum_{i=1}^{N} \boldsymbol{F}_i(p_i, a_i) \\ \text{subject to:} \quad & (p_i, v_i, a_i) \in \boldsymbol{S}_i \quad \forall i \\ & \overline{h}_{ij}(p_i, p_j) \geq 0 \quad \forall j \in \boldsymbol{\mathcal{N}}_i \text{ and } i \end{aligned} \tag{6.7}$$

The solution of the above linearised problem causes the trajectories to be more conservative than they would have been without the linearization. This conservatism can be mitigated by the careful choice of the nominal trajectories $(\overline{p}_i, \overline{p}_j)$. This is initialised by firstly solving the linearised problem (6.7) without the collision avoidance constraint. These are then updated by iteratively solving (6.7) to improve the linearisation accuracy [40]. The iterations are stopped when convergence for the objective value is achieved, i.e $|\sum_{i=1}^{M} \boldsymbol{F}_i^{\mu}(p_i, a_i) - \sum_{i=1}^{M} \boldsymbol{F}_i^{\mu-1}(p_i, a_i)| < \epsilon$, where ϵ is a tuning parameter and $\sum_{i=1}^{M} \boldsymbol{F}_i^{\mu}$ is the objective value for the current linearisation iteration μ.



(a) Collision avoidance constraint is not enforced (b) Collision avoidance is enforced with $\Delta = 0.35$m

Figure 6.2: Trajectory tracking of three agents with double integrator dynamics. The circles denote the starting positions and the crosses the targets.

The problem (6.7) is set up in MATLAB using the YALMIP optimisation toolbox [45] and the OSQP solver [46] with the double integrator dynamics in (6.2) and $T = 0.1$s. The only constraints are set for the final velocities and control inputs, i.e. at $k = N$, of all agents to be zero, so that those come to rest. For these and all future simulations the position is measured in meters (m), velocity in meters per second (m/s) and acceleration in meters per second squared (m/s^2). The design matrices, $\boldsymbol{Q}$ and $\boldsymbol{R}$ are set to be identity matrices with scalings of 14 and 5, penalising the deviations from the reference trajectory and the inputs, respectively and the horizon window N is set to 100. These design choices ensured the fast enough completion of the tracking under ten seconds. With this setup of three agents, convergence is achieved in only three iterations where we choose the tuning parameter ϵ to be the 0.5% of the current objective value. Figure 6.2 shows the results of the simulation with the above mentioned parameters, where we have three agents moving in two dimensions, starting positions denoted by circles and the targets for each denoted by crosses and by their respective colors. No collision avoidance constraints are imposed in Figure 6.2a and we observe that the agents fly too close to each other and achieve their targets in the shortest possible way, following a straight line. In contrast, in Figure 6.2b the linearised collision avoidance constraint $\overline{h}_{ij}$ is enforced with $\Delta = 0.35$m, i.e. creating a safe circle around all of the agents with radius Δ. In this case the agents now deviate from their shortest paths, following a safe trajectory.

6.2 Full Linearized Model Integration

In Section 4.2.2, a full linearised model for the Crazyflie quadrotors was developed to be used with the on board control scheme as introduced in Section 4.3.2. Before this model can be integrated

as a dynamical system in the network represented as a constraint set $\boldsymbol{S}_i$, it is reduced. First of all, we discretize the model (4.9) using the function "$c2d$" in MATLAB and a discretization interval of T. This results in a discretized state space model from reference inputs, height, (x, y) velocities and yaw angle to the full state output vector comprising of nine state variables, which can be represented in the following form and from now on referred to as the full discretized model,

$$\begin{aligned} s_f(k+1) &= \boldsymbol{A}_{df} s_f(k) + \boldsymbol{B}_{df} u_f(k) \\ y_f(k+1) &= \boldsymbol{C}_{df} s_f(k+1), \end{aligned} \tag{6.8}$$

where s_f, u_f and y_f are the state, input and output vectors as defined in Section 4.2.2, and $\boldsymbol{A}_{df}, \boldsymbol{B}_{df}$ and $\boldsymbol{C}_{df}$ are the discretized state matrices of the model (4.9). As discussed in Section 4, the sparse structure of the state matrices of the linearised model allows for the decoupling of the states and the development of four separate controllers. We make use of this to reduce the full discretized model to only the (x, y) velocity controllers to simulate and test the algorithms in two dimensions. Accounting for the numerical errors in the discretization and for the mentioned decoupling we are able to reduce the model (6.8) to have only $n_u = 2$ control inputs and $n_x = 6$ states. We call this the reduced discretized model and, for each agent i it takes the following discretized state space form,

$$\begin{aligned} s_i(k+1) &= \boldsymbol{A}_r s_i(k) + \boldsymbol{B}_r u_i(k) \\ y_i(k+1) &= \boldsymbol{C}_r s_i(k+1), \end{aligned} \tag{6.9}$$

where $s_i(k) \in \mathbb{R}^{n_x \times 1}$ is the reduced state vector containing only the (x, y) position and velocity, and the pitch and roll angles, as defined in Equation (6.10), where $p_i(k) = [x_i(k)\ y_i(k)]^T$ (for simplicity we use the same notation as in (6.2)) and $\psi(k) = [\gamma_i(k)\ \beta_i(k)]^T$. The control input vector $u_i(k) \in \mathbb{R}^{n_u \times 1}$ comprises of only (x, y) velocity control inputs, defined in Equation (6.11). The matrices $\boldsymbol{A}_r \in \mathbb{R}^{n_x \times n_x}$, $\boldsymbol{B}_r \in \mathbb{R}^{n_x \times n_u}$ and $\boldsymbol{C}_r \in \mathbb{R}^{n_x \times n_x}$ are the discretized and reduced state matrices derived from the full discretized model (6.8). And finally, the output vector is $y_i(k) \in \mathbb{R}^{n_x}$.

$$s_i(k) = \begin{bmatrix} p_i(k) \\ \dot{p}_i(k) \\ \psi(k) \end{bmatrix}, \tag{6.10}$$

$$u_i(k) = \begin{bmatrix} v_{xi}(k) \\ v_{yi}(k) \end{bmatrix}, \tag{6.11}$$

This reduced form can now be used to model the dynamics of the Crazyflie quadcopters in a two dimensional space by integrating it into the centralised collision avoidance problem (6.7) and then implement it in practice as discussed in Section 8. This problem is thus reformulated as follows,

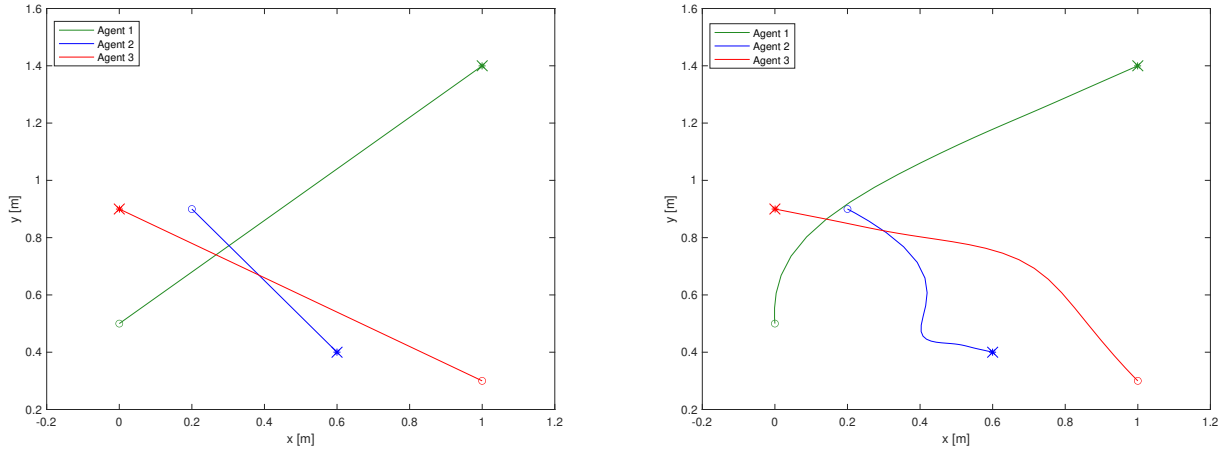
$$\begin{aligned} \min_{s_i, u_i} \quad & \sum_{i=1}^{M} \boldsymbol{F}_i(s_i, u_i) \\ \text{subject to:} \quad & (s_i, u_i) \in \boldsymbol{D}_i \quad \forall i \\ & \overline{h_{ij}}(p_i, p_j) \geq 0 \quad \forall j \in \boldsymbol{\mathcal{N}}_i \text{ and } i, \end{aligned} \tag{6.12}$$

where $\boldsymbol{D}_i$ represents the dynamics of the agent i, according to the reduced discretized model (6.9) and any constraints on the states and inputs. The state, input and position vectors, $s_i = [s_i^T(0)\ \dots\ s_i^T(N)]^T \in$

$\mathbb{R}^{n_x(N+1)}$, $u_i = [u_i^T(0) \ \dots \ u_i^T(N-1)]^T \in \mathbb{R}^{n_u N}$ and $p_i = [p_i^T(0) \ \dots \ p_i^T(N)]^T \in \mathbb{R}^{2(N+1)}$ are given by vertically concatenating the vector variables for each time step k. Defining $r_i = [r_i^T(0) \ \dots \ r_i^T(N)]^T \in \mathbb{R}^{n_x(N+1)}$ (the references for ψ are always set to zero), the objective function for i is given by,

$$\boldsymbol{F}_i(s_i, a_i) = (s_i - r_i)^T \boldsymbol{Q}(s_i - r_i) + (u_i)^T \boldsymbol{R}(u_i), \tag{6.13}$$

We thus assume that all of the agents, naturally, have the same dynamics.



(a) Collision avoidance constraint is not enforced

(b) Collision avoidance is enforced with $\Delta = 0.35$m

Figure 6.3: Trajectory tracking of three agents with Crazyflie dynamics. The circles denote the starting points and the crosses the targets.

The problem (6.12) is set up and solved with a discretization step of $T = 0.1$s for the dynamics and constraint set $\boldsymbol{D}_i$. We impose the constraint for the absolute values of the inputs to always be below 1m/s, which is the limit of the Crazyflies [47]. We set $\boldsymbol{Q}$ and $\boldsymbol{R}$ to be identity matrices, with weights of 10 and 13, penalising the position deviations and inputs, respectively, and the horizon window size, N, is set to 100. These choices proved to achieve fast enough trajectories for these dynamics in under ten seconds. Figure 6.3 above shows the trajectories of the agents with the same starting and target positions as for the double integrator case in Figure 6.2 discussed in Section 6.1. Figure 6.3a shows the trajectories without the collision avoidance, as opposed to Figure 6.3b for which those are enforced with a safety radius of $\Delta = 0.35$m. With the same choice of ϵ being the 0.5% of the objective value, convergence is, as before, achieved after only three iterations. Comparing Figures 6.3 and 6.2, we notice a significant difference in the shape of the trajectories and the lack of overshoot of the target for the agents with Crazyflie dynamics. In order to come to a stop, the agents with double integrator dynamics need to circle the target to lower their velocity to an eventual stop, while the more complex dynamics of the aerial vehicles allows them to reach the target by dissipating less energy following a shorter path. This means that the derived linearised model of the drones is a more accurate representation of the actual system than the other. This difference in trajectories was also

confirmed when the same design weights, as for the double integrator dynamics were used. However, as mentioned earlier, the change in the weights proved to result in a much smoother and fast enough flight.

6.3 OSQP Formulation

With the current setup of the collision avoidance problem it will be impossible to run a real time controller on quadcopters, whether in an off-board or an on-board mode. The reason is the time it takes for the linearized collision avoidance constraints to converge. For the tests we conducted with the YALMIP setup, the lowest time for one iteration was never lower than 2 seconds. Implementing such an algorithm in a MPC fashion, discussed later in Section 6.4 is impossible, as MPC requires successive optimizations to take place until the target is achieved. With the very fast dynamics of the Crazyflie (around 15Hz), we need a solver which will be faster, so that even when considering the communication time delays stability is not compromised. The Operator Splitting Quadractic Program (OSQP) [46] is an efficient and robust solver ideal for solving quadratic programs, such as (6.12). The program solves quadratic problems of the form,

$$\begin{aligned} \min_{x} \quad & \frac{1}{2} x^T \boldsymbol{P} x + q^T x \\ \text{subject to:} \quad & c_l \leq \boldsymbol{A}_q x \leq c_u \end{aligned}, \tag{6.14}$$

where $x \in \mathbb{R}^n$ is the optimization variable, $\boldsymbol{P} \in \mathbb{S}_+^n$ is a positive semidefinite matrix, $\boldsymbol{A}_q \in \mathbb{R}^{m \times n}$ is the matrix defining the linear constraints, vectors c_l and c_u are such that $c_l \in \mathbb{R} \cup \infty$ and $c_u \in \mathbb{R} \cup \infty$ and the linear term is the vector $q \in \mathbb{R}^n$. In this section we transform the centralised problem in (6.12) to the form in (6.14) and solve it to achieve the desired efficiency, robustness and speed.

In order to treat the multi-agent centralised problem in (6.12), we stack the states $s_i(k)$, defined in (6.10) and the control inputs $u_i(k)$, defined in (6.11), such that $\overline{s}(k) = [s_1^T(k)\ s_2^T(k)\ \dots\ s_M^T(k)]^T$, and $\overline{u}(k) = [u_1^T(k)\ u_2^T(k)\ \dots\ u_M^T(k)]^T$. The state matrices $\boldsymbol{A}_r$ and $\boldsymbol{B}_r$ are augmented for the network such that $\overline{\boldsymbol{A}}_r = \boldsymbol{I}^M \otimes \boldsymbol{A_r}$ and $\overline{\boldsymbol{B}}_r = \boldsymbol{I}^M \otimes \boldsymbol{B_r}$, where $\otimes$ is the Kronecker product and the form $\boldsymbol{I}^{\Xi}$ is an identity matrix of size Ξ. In this case, $\Xi = M$, the number of agents. We can then represent the network's dynamics as,

$$\underbrace{\begin{bmatrix} \overline{s}(0) \\ \overline{s}(1) \\ \overline{s}(2) \\ \vdots \\ \overline{s}(N) \end{bmatrix}}_{\widetilde{s}} = \underbrace{\begin{bmatrix} \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \overline{\boldsymbol{A}}_r & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \overline{\boldsymbol{A}}_r & \dots & \mathbf{0} \\ \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \dots & \overline{\boldsymbol{A}}_r & \mathbf{0} \end{bmatrix}}_{\widetilde{\boldsymbol{A}}_r} \underbrace{\begin{bmatrix} \overline{s}(0) \\ \overline{s}(1) \\ \overline{s}(2) \\ \vdots \\ \overline{s}(N) \end{bmatrix}}_{\widetilde{s}} + \underbrace{\begin{bmatrix} \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \overline{\boldsymbol{B}}_r & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \overline{\boldsymbol{B}}_r & \dots & \mathbf{0} \\ \vdots & \ddots & \vdots & \ddots \\ \mathbf{0} & \dots & \mathbf{0} & \overline{\boldsymbol{B}}_r \end{bmatrix}}_{\widetilde{\boldsymbol{B}}_r} \underbrace{\begin{bmatrix} \overline{u}(0) \\ \overline{u}(1) \\ \overline{u}(2) \\ \vdots \\ \overline{u}(N-1) \end{bmatrix}}_{\widetilde{u}} + \underbrace{\begin{bmatrix} \overline{s}(0) \\ \mathbf{0} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix}}_{\widetilde{s}_0}, \tag{6.15}$$

where $\widetilde{s}, \widetilde{s}_0 \in \mathbb{R}^{n_x(N+1)M}$, $\widetilde{\boldsymbol{A}}_r \in \mathbb{R}^{n_x(N+1)M \times n_x(N+1)M}$, $\widetilde{\boldsymbol{B}}_r \in \mathbb{R}^{n_x(N+1)M \times n_u(N)M}$, $\widetilde{u}, \in \mathbb{R}^{n_u(N)M}$ with n_x and n_u defined in the previous section, and the matrices $\mathbf{0}$ are null matrices to complete the $\widetilde{\boldsymbol{A}}_r$ and $\widetilde{\boldsymbol{B}}_r$. The above can now be rewritten compactly and simplified as follows,

$$\widetilde{s} = \widetilde{\boldsymbol{A}}_r \widetilde{s} + \widetilde{\boldsymbol{B}}_r \widetilde{u} + \widetilde{s}_0 \Rightarrow \tag{6.16a}$$

$$-\widetilde{s}_0 = (\widetilde{\boldsymbol{A}}_r - \boldsymbol{I})\widetilde{s} + \widetilde{\boldsymbol{B}}_r \widetilde{u} \Rightarrow \tag{6.16b}$$

$$-\widetilde{s}_0 = \begin{bmatrix} (\widetilde{\boldsymbol{A}}_r - \boldsymbol{I}) & \widetilde{\boldsymbol{B}}_r \end{bmatrix} \begin{bmatrix} \widetilde{s} \\ \widetilde{u} \end{bmatrix}. \tag{6.16c}$$

Equation (6.16c) above shows the collective dynamics of the networks and is a dynamical system itself. We group the state and input variables into a single vector $v \in \mathbb{R}^{n_x(N+1)M + n_u(N)M}$, given by,

$$v = \begin{bmatrix} \widetilde{s} \\ \widetilde{u} \end{bmatrix} \tag{6.17}$$

The objective function in (6.13) can be reformulated to the standard OSQP form in (6.14). By introducing augmented matrices $\overline{\boldsymbol{Q}} = \boldsymbol{I}^M \otimes \boldsymbol{Q}$ and $\overline{\boldsymbol{R}} = \boldsymbol{I}^M \otimes \boldsymbol{R}$, we can then formulate $\widetilde{\boldsymbol{Q}} = \boldsymbol{I}^{N+1} \otimes \overline{\boldsymbol{Q}}$, $\widetilde{\boldsymbol{R}} = \boldsymbol{I}^N \otimes \overline{\boldsymbol{R}}$ and $\widetilde{q} = [2(\overline{\boldsymbol{Q}} r(1))^T \ \dots \ 2(\overline{\boldsymbol{Q}} r(N+1))^T]^T$, where $r(k)$ is the reference state for timestep k. This allows us to formulate an objective $\boldsymbol{F}$ function for the whole system as,

$$\boldsymbol{F} = \widetilde{s}^T \widetilde{\boldsymbol{Q}} \widetilde{s} - \widetilde{q}^T \widetilde{s} + \widetilde{u}^T \widetilde{\boldsymbol{R}} \widetilde{u}. \tag{6.18}$$

The linearised constraint for collision avoidance is formulated in Equation (6.5). Before setting this in a matrix inequality form, it is reformulated below and its structure is then discussed in detail. For each timestep k we need that,

$$\|\overline{p}_i(k) - \overline{p}_j(k)\| + \eta_{ij}^T(k)[(p_i(k) - p_j(k)) - (\overline{p}_i(k) - \overline{p}_j(k))] - \Delta \geq 0 \quad \forall j \in \boldsymbol{\mathcal{N}}_i, \ i = 1, \dots, \ M \tag{6.19}$$

where $\eta_{ij}(k)$ is given by Equation (6.6). We gather the constants for the particular iteration to the right side and denote this value for each agent i and its neighbor j by $l_{ij}(k)$, as follows,

$$\eta_{ij}^T(k)(p_i(k) - p_j(k)) \geq \Delta + \eta_{ij}^T(k)(\overline{p}_i(k) - \overline{p}_j(k)) - \|\overline{p}_i(k) - \overline{p}_j(k)\| \triangleq l_{ij}(k). \tag{6.20}$$

The values $l_{ij}(k)$ are concatenated vertically into a vectors $l(k) \in \mathbb{R}^{MN_i}$. We then define a new vector $p_{i\Delta}(k) = [(p_i - p_1)^T(k) \ \ (p_i - p_2)^T(k) \ \dots \ (p_i - p_{N_i})^T(k)]^T$ for each k, as a constituent part of the condition above, formed by $p_{i\Delta}(k) = \boldsymbol{V}_i \overline{s}(k)$.

$$\boldsymbol{H} = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & & n_x \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \end{bmatrix} \end{array}, \tag{6.21}$$

$$\boldsymbol{K_i} = \begin{array}{c} \begin{matrix} 1 & 2 & & i & & M \end{matrix} \\ \begin{bmatrix} -1 & 0 & \dots & 1 & \dots & 0 \\ 0 & -1 & \dots & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & \dots & -1 \end{bmatrix} \end{array}. \tag{6.22}$$

To get the form of the matrix $\boldsymbol{V}_i \in \mathbb{R}^{2N_i \times n_x M}$, for the two dimensional case, we first define two matrices $\boldsymbol{H} \in \mathbb{R}^{2 \times n_x}$ and $\boldsymbol{K_i} \in \mathbb{R}^{N_i \times M}$ in (6.21) and (6.22), respectively. Matrix $\boldsymbol{H}$ has the property of extracting only p_i, the position of the agent, from the state. The matrix $\boldsymbol{K}_i$ is a negative identity matrix of size N_i with the columns from i to N_i shifted by one position to the right and the i^{th} column replaced by a vector of ones of size N_i. The transformation matrix is then given as $\boldsymbol{V}_i = \boldsymbol{K}_i \otimes \boldsymbol{H}$. This idea is then extended for the dynamics of the whole network. For this, the vector $p_\Delta(k) = [p_{1\Delta}^T(k) \ \dots \ p_{M\Delta}^T(k)]^T$ is defined and a matrix $\boldsymbol{K} \in \mathbb{R}^{MN_i \times M}$ is formed by concatenating the matrices $\boldsymbol{K}_i$ vertically for $i = 1, \dots, M$. Thus, putting everything together we can model the transformation for the whole network as $p_\Delta(k) = \boldsymbol{V}\bar{s}(k)$, with, $\boldsymbol{V} = \boldsymbol{K} \otimes \boldsymbol{H}$, where $\boldsymbol{V} \in \mathbb{R}^{2MN_i \times n_x M}$. A matrix $\boldsymbol{\eta}^M(k) \in \mathbb{R}^{MN_i \times 2MN_i}$ is then constructed for each k, as the linearisation term in the constraint, given in Equation (6.23) below.

$$\boldsymbol{\eta}^M(k) = \begin{bmatrix} \eta_{12}^T(k) & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \eta_{MN_i}^T(k) \end{bmatrix} \qquad (6.23) \qquad \boldsymbol{\eta}^M = \begin{bmatrix} \boldsymbol{\eta}^M(1) & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \boldsymbol{\eta}^M(N) \end{bmatrix} \qquad (6.24)$$

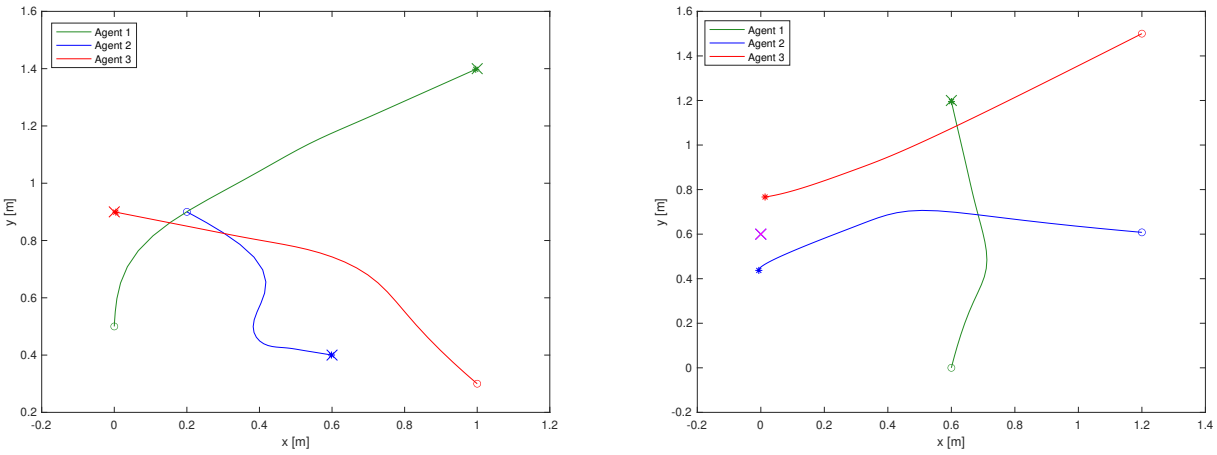
Defining (6.23) for all $k = 1, \dots, N$ we form the augmented matrix $\boldsymbol{\eta}^M \in \mathbb{R}^{M(N)N_i \times 2M(N)N_i}$ in (6.24). The vector $\widetilde{l} \in \mathbb{R}^{M(N)N_i}$ is formed by a vertical concatenation of $l(k)$ for all $k = 1, \dots, N$.

$$\boldsymbol{A}_{qs}\widetilde{s} \geq \widetilde{l} \qquad (6.25)$$

We then define the identity matrix $\boldsymbol{I}_N^{N+1} \in \mathbb{R}^{N \times (N+1)}$ whose first row is removed, so that we do not impose the constraints for the initial positions. The inequality matrix is thus formed as $\boldsymbol{A}_{qs} = \boldsymbol{\eta}_M(\boldsymbol{I}_N^{N+1} \otimes \boldsymbol{V})$, with $\boldsymbol{A}_{qs} \in \mathbb{R}^{M(N)N_i \times n_x(N+1)M}$ with the final constraint form defined in (6.25).

$$\begin{aligned} \min_v \ & \frac{1}{2} v^T \begin{bmatrix} \widetilde{\boldsymbol{Q}} & \boldsymbol{0} \\ \boldsymbol{0} & \widetilde{\boldsymbol{R}} \end{bmatrix} v - q^T v \\ \text{subject to: } & [(\widetilde{\boldsymbol{A}} - \boldsymbol{I}) \quad \widetilde{\boldsymbol{B}}]v = -v_0 \\ & \boldsymbol{A}_q v \geq \widetilde{l} \\ & u_{min} \leq \boldsymbol{A}_u v \leq u_{max} \end{aligned} \qquad (6.26)$$

As the OSQP form (6.14) suggests, everything is represented with a single vector v in (6.17). Thus, taking the objective function (6.18), the dynamics (6.16c) and the collision avoidance constraint (6.25); the optimisation problem in (6.12) is formulated in (6.26) above. The vector $q \in \mathbb{R}^{n_x(N+1)M + n_u(N)M}$ is the vector $\widetilde{q}$ concatenated with a vector of zeros to account for the control inputs $\widetilde{u}$, as it is done for $v_0 \in \mathbb{R}^{n_x(N+1)M + n_u(N)M}$. The matrix $\boldsymbol{A}_q \in \mathbb{R}^{M(N)N_i \times n_x(N+1)M + n_u(N)M}$ is an augmented matrix formed by horizontally concatenating $\boldsymbol{A}_{qs}$ and a null matrix, again to account for the augmented state with control inputs included. The matrix $\boldsymbol{A}_u \in \mathbb{R}^{M(N)n_u \times n_x(N+1)M + n_u(N)M}$ is an identity matrix for the control inputs of size N and a null matrix for the states. The vectors $u_{min}, u_{max} \in \mathbb{R}^{M(N)n_u}$ are the minimum and maximum allowable reference velocities, respectively.



(a) Configuration 1; $\Delta = 0.35$m

(b) Configuration 2; $\Delta = 0.33$m

Figure 6.4: Trajectory tracking of three agents with 2 different configurations and Δ-s. The circles denote the starting points and the crosses the targets.

The problem (6.26) is set up and solved with the same parameters $\boldsymbol{R}$, N and the same convergence stopping parameter ϵ as in Section 6.2. However, the design matrix $\boldsymbol{Q}$ is updated to penalise both the position deviations and the velocity with a weight of 10 for a slower and safer flight. Figure 6.4 above shows the two problems solved for two different configurations, i.e. different starting and reference positions. The configuration in Figure 6.4a is the same as in 6.3b, and we observe that the trajectories are almost exactly, accounting for the updated $\boldsymbol{Q}$. The second configuration in Figure 6.4b, differs from the other in the sense that we use a purple cross, meaning that the target position for both agent two and agent three is the same and they get as close as possible to the target maintaining the required $\Delta = 0.33$m distance. The algorithms again converge after only three iterations, however, in contrast to the setup in Section 6.2, the longest these iterations take was measured to be only 0.4 seconds. Thus, comparing this duration with the OSQP setup to the one set in the previous section, for which the lowest duration was 2 seconds, we achieve the same result about 5 times faster. This efficiency and gain in speed proves to be invaluable for the MPC setup of the problem discussed in the next section.

6.4 MPC Setup

Model predictive control is currently one of the most widely used techniques for controlling a number of systems. This is thanks to the great compromise between optimality and speed of computation that it provides [48]. Implemented in a receding horizon fashion, MPC provides a closed loop feedback control to the otherwise open loop implementation of an optimal control problem. The idea behind MPC is simple, we predict the future behaviour of the system over some horizon window using a model

and the estimates and measurements of the current state. Only the first input of this predictions is implemented and to introduce feedback, this process is repeated. The most important feature of the MPC for our system, is that it allows us to put constraints on states and inputs which is crucial for our requirements. In this section we discuss how MPC is implemented and then apply it to the centralised quadrotor collision avoidance problem (6.26).

6.4.1 MPC Overview

We consider a discrete state space representation of a linear model, with $x(k)$ and $u(k)$ being the state and input vectors of the system at time k,

$$x(k+1) = \boldsymbol{A}x(k) + \boldsymbol{B}u_k \tag{6.27}$$

For a predicted input and state sequence generated by simulating the model for a prediction horizon window of size N, The following vectors can be defined as stacked predicted sequences, similar to Equation (6.15),

$$\widetilde{u}(k) = \begin{bmatrix} u_0(k) \\ u_1(k) \\ \vdots \\ u_{N-1}(k) \end{bmatrix} \tag{6.28}$$

$$\widetilde{x}(k) = \begin{bmatrix} x_1(k) \\ x_2(k) \\ \vdots \\ x_N(k), \end{bmatrix} \tag{6.29}$$

where $x_i(k) = x(k+i)$ and $u_i(k) = u(k+i)$. These predicted sequences are usually generated by solving an optimization problem. We consider a quadratic objective function with linear constraints on states and inputs,

$$\begin{aligned} \min_{\widetilde{x}(k),\widetilde{u}(k)} \quad & \boldsymbol{J}(\widetilde{x}(k),\widetilde{u}(k)) = \sum_{i=0}^{N-1} (x_i(k)^T \boldsymbol{Q} x_i(k)^T + u_i(k)^T \boldsymbol{R} u_i(k)) + x_N(k)^T \boldsymbol{Q_N} x_N(k) \\ \text{subject to: } & x_{min} \leq \boldsymbol{G}\widetilde{x}(k) \leq x_{max} \\ & u_{min} \leq \boldsymbol{H}\widetilde{u}(k) \leq u_{max}, \end{aligned} \tag{6.30}$$

where $\boldsymbol{G}$ and $\boldsymbol{H}$ are the constraint matrices, $\boldsymbol{Q}, \boldsymbol{R}$ are weighting design matrices and $\boldsymbol{Q}_N$ is the terminal weighting matrix for the terminal state $x_N(k)$. The size of the horizon window N does not change. For the receding horizon implementation of the MPC, the above problem is solved at each time instant k and some optimal inputs $\widetilde{u}^\star(k)$ are generated.However, only the first predicted input $u_0(k)$ is input to the plant. For the infinite horizon window, if there are no model errors and/or disturbances to the system, then there is no difference between applying an optimal LQR controller, as in Section 4.3.1 and a receding horizon MPC controller for the unconstrained case. However, when we have a finite sized horizon window, then this MPC introduced closed loop system can go unstable. To solve this a dual mode of control is introduced [48], where the first $N-1$ inputs define the first mode and the control inputs are determined by optimisation and in the second mode for $i = N, N+1, \ldots$ the control

inputs are given by the stabilising feedback law, as $u_i(k) = \boldsymbol{K}x_i(k)$. The terminal matrix $\boldsymbol{Q}_N$ is the solution of the matrix Lyapunov equation [49] with the feedback gain $\boldsymbol{K} = -\boldsymbol{R}^{-1}\boldsymbol{B}^T\boldsymbol{Q}_N$, as follows,

$$(\boldsymbol{A}+\boldsymbol{B}\boldsymbol{K})^T\boldsymbol{Q}_N + \boldsymbol{Q}_N(\boldsymbol{A}+\boldsymbol{B}\boldsymbol{K}) = -\boldsymbol{Q} - \boldsymbol{K}^T\boldsymbol{R}\boldsymbol{K}, \tag{6.31}$$

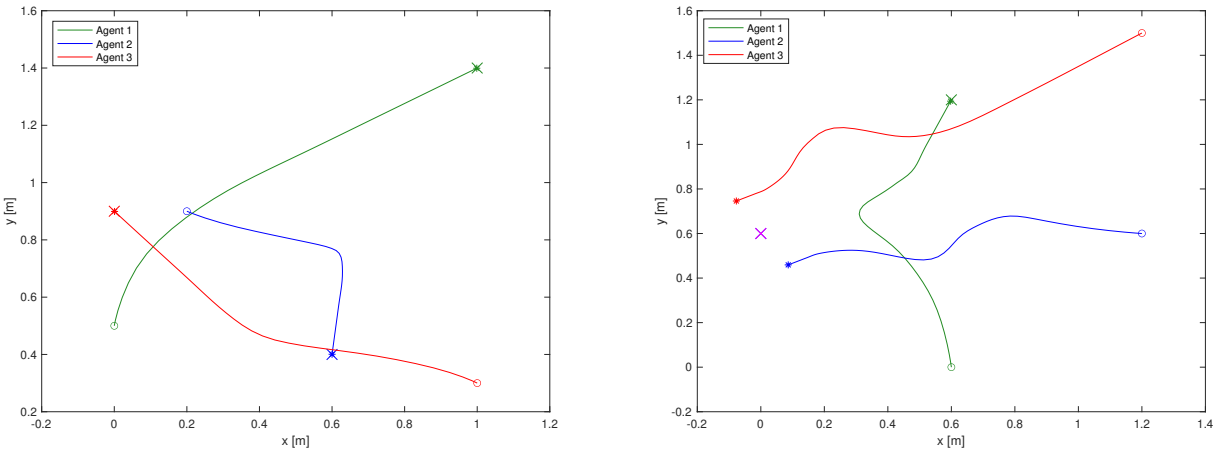
which is also the Algebraic Riccati Equation [50]. This choice of the terminal weighting matrix and feedback law is proven, using the Lyapunov stability theory [48], to represent the infinite horizon quadratic cost as,

$$\sum_{i=0}^{\infty}(x_i^T\boldsymbol{Q}x_i^T + u_i^T\boldsymbol{R}u_i) = x_0^T\boldsymbol{Q}_N x_0. \tag{6.32}$$

Thus, by choosing the terminal weighting matrix to be the solution of the Lyapunov Equation (6.31), the dual mode finite horizon control mode becomes equivalent to an infinite horizon window control. This means, that the stability of an unconstrained problem is guaranteed. For a constrained problem, as in (6.30), the stability of the system under MPC is guaranteed only if $\boldsymbol{Q}_N$ is a solution of (6.31), $\boldsymbol{Q}$ is positive definite and for time instant k the predicted inputs over a horizon N are all feasible, i.e., satisfy the constraints for all $k > 0$. The last condition, also referred to as recursive feasibility, ensures that the agents will not follow a path that will at some point bring them to an infeasibility; in our future solutions of the problem (6.26) we assume that this condition holds. The inclusion of Robust MPC algorithms for solving the quadrotor collision avoidance problem is discussed in Section 9.1.

6.4.2 Simulation Results

The advantages provided by MPC proved to be greatly applicable for our problem. It provides stability and convergence, while solving a constrained optimisation problem in a very short time. We set up



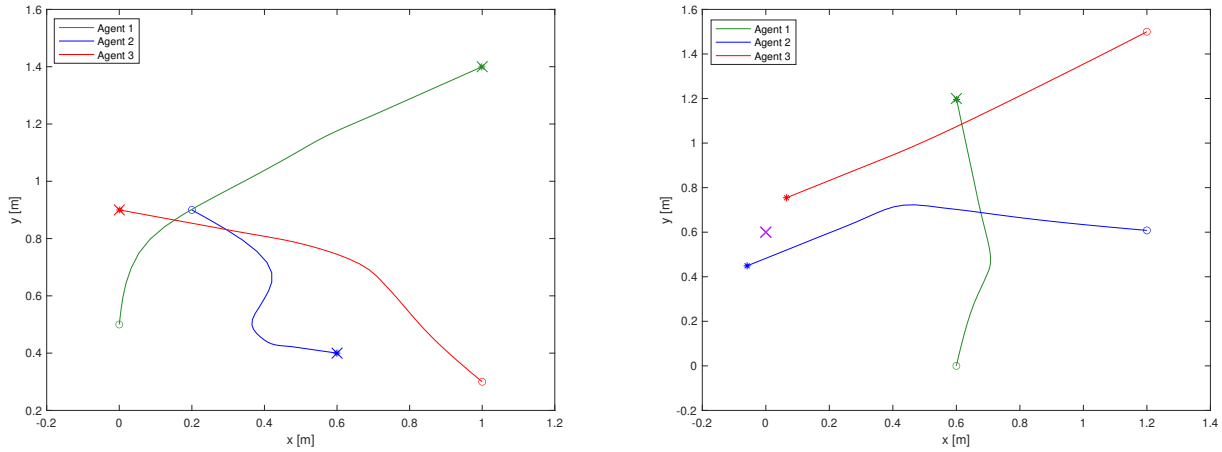
(a) Configuration 1; $\Delta = 0.35$m

(b) Configuration 2; $\Delta = 0.33$m

Figure 6.5: Trajectory tracking of three agents with 2 different configurations and Δ-s, generated under a receding horizon implementation with a horizon window size of $N = 10$. The circles denote the starting points and the crosses the targets.

the MPC framework for the already augmented centralised collision avoidance problem (6.26) with a horizon window of $N = 10$. We use the same structure of $\boldsymbol{R}$, as we used in Section 6.3; for the design $\boldsymbol{Q}$ matrix we use a weighting of 10 for the position deviation $(p - r)$ and the weight of for the velocity state $\dot{p}$ is updated to 15, so that the flights are slower than before, decreasing the probability of future infeasibilities. The augmented matrix is formed by the Kronecker product $\widetilde{\boldsymbol{Q}} = \boldsymbol{I}^N \otimes \overline{\boldsymbol{Q}}$, and the terminal matrix $\boldsymbol{Q}_N$ is set to be the solution of the Algebraic Riccati Equation (6.31) for the chosen $\boldsymbol{Q}$ and $\boldsymbol{R}$.

The receding horizon is implemented by solving the problem (6.26) iteratively, without changing the horizon window size N. At each iteration the constraints are updated. Specifically, in the first constraint specifying the dynamics v_0 is updated with the current state estimate. In simulation, this is done by iterating the model, and in implementation the data would be received from the Crazyflie sensors, as explained in Section 4.3.2. The matrix $\boldsymbol{A}_q$ is updated to integrate the latest nominal trajectories $(\overline{p}_i, \overline{p}_j)$. Inside the MPC loop there is also the linearisation of the nonconvex collision avoidance constraint taking place in a separate loop. This is again terminated according to the tuning parameter ϵ, as described in Section 6.1. The convergence for this is reached in just a couple of



(a) Configuration 1; $\Delta = 0.35$m

(b) Configuration 2; $\Delta = 0.33$m

Figure 6.6: Trajectory tracking of three agents with 2 different configurations and Δ-s, generated under a receding horizon implementation with a horizon window size of $N = 20$. The circles denote the starting points and the crosses the targets.

iterations, as we make use of the MPC calls (iterations) to start the next linearisation with a better approximation for the nominal trajectories. We set up up the MPC problem and solve it using OSQP for two different configurations as explained in previous sections. The results are shown in Figure 6.5. The trajectories look different as the horizon window is 10 times smaller than in the previous case in Section 6.3 and penalisation on velocity is increased. However, the constraints are still satisfied and optimal solutions for this window size are achieved. Each MPC call takes no more than 20 milliseconds

which is 50Hz and, thus perfect for using this with the real system online. This short duration is achieved thanks to the efficient structure of the OSQP solver and the short horizon window size.

Changing the horizon window size to $N = 20$ we observe that the trajectories, shown in Figure 6.6, are already much more closer to the solution in Figure 6.4. The duration for each call in this case is still very low, with the highest being around 30 milliseconds. Even though, this is fast enough to be implemented with the Crazyflie fast dynamics, we will still be using the shorter window of $N = 10$ for future tests so that the communication delay with a safety margin is accounted as well. The size N is subject to change and adaptation to different scenarios. The formulation of the centralised collision avoidance and trajectory tracking problem and its solution using fixed and receding horizon MPC methods, now allows us to move on to its distribution using ADMM, as discussed in the next section.

7 Distributed Problem Setup

In this section we continue the setup of the multi-agent problem by presenting how the centralised collision avoidance problem, formulated in (6.7) can be distributed to be solved among the quadrotor agents using a fully decentralised version [24] of the ADMM algorithm described in Section 5.4.1. This distributed or fully decentralised problem is then solved using OSQP for the quadrotor derived dynamics (6.9) and then in a receding horizon fashion with MPC to introduce closed loop feedback into the system.

7.1 Fully Decentralised ADMM

In ADMM, a decentralised algorithm, there is a need for a presence of an aggregator or a central authority, as it was discussed in Section 5.4. Depending on the type of the problem, this authority may hold different kinds of variables. However, it is always something that is common to all the agents. The Fully Decentralised ADMM algorithm for collision avoidance [24] similar to Tracking-ADMM [26], modifies the original ADMM to remove this aggregator dependence. Before we discuss how it is done and implement it, we first bring the centralised problem with simple, double integrator dynamics in (6.7) to the required ADMM form in (5.7).

$$\begin{aligned} \min_{p_i, w_i, v_i, a_i, w_{i\to j}} \quad & \sum_{i=1}^{M} \boldsymbol{F}_i(p_i, a_i) \\ \text{subject to:} \quad & (p_i, v_i, a_i) \in \boldsymbol{S}_i \qquad \forall i \\ & \overline{h_{ij}}(w_i, w_{i\to j}) \geq 0 \quad \forall j \in \boldsymbol{\mathcal{N}}_i \text{ and } i \\ & w_i = p_i; \;\; w_{i\to j} = p_j \;\forall j \in \boldsymbol{\mathcal{N}}_i \text{ and } i. \end{aligned} \tag{7.1}$$

The Equation (7.1) above is thus formed, where the set $\boldsymbol{S}_i$ incorporating the simple dynamics and the state and input constraints is defined in (6.2). The objective function $\boldsymbol{F}_i$ is defined in (6.3). The linearised collision avoidance constraint $\overline{h}_{ij} \geq 0$ is defined in (6.5), the variables (p_i, v_i, a_i) correspond to x and the newly introduced $(w_i, w_{i\to j})$ correspond to z in (5.7). The latter are duplicate variables, where w_i is the duplicate of p_i, and $w_{i\to j}$ that of p_j and is the so called proposed trajectory by agent i for its j^{th} neighbour. These are all formed by vertically concatenating the decision variables for each time step k over a horizon window of N, as was done Section 6.1 for the centralised case. Note that the collision avoidance constraint is no longer imposed on the trajectories but on their duplicates, w. The parallelisable iterative algorithm that solves this problem proposed by [24] follows the traditional ADMM steps (5.8), but introduces two communication steps in between to achieve full parallelisation. It is shown in Algorithm 1 and described in detail below.

Algorithm 1: Fully Decentralised ADMM algorithm [24] for each agent $i = 1, \ldots, M$

initialize $\lambda_i, w_i, \{\lambda_{i\to j}, \lambda_{j\to i}, w_{j\to i}\}_{j\in\boldsymbol{\mathcal{N}}_i}$;
repeat

1. *prediction*: update (p_i, v_i, a_i) with
$$\arg\min_{p_i,v_i,a_i\in\boldsymbol{S}_i} \boldsymbol{F_i}(p_i, a_i) + \lambda_i^T(p_i - w_i) + \frac{\rho}{2}\|p_i - w_i\|^2 + \sum_{j=1}^{N_i}(\lambda_{j\to i}^T(p_i - w_{j\to i}) + \frac{\rho}{2}\|p_i - w_{j\to i}\|^2)$$
2. *communication I*: send p_i to $j \in \boldsymbol{\mathcal{N}}_i$, and receive $\{p_j\}_{j\in\boldsymbol{\mathcal{N}}_i}$
3. *linearisation*: update $\{\overline{h}_{ij}\}_{j\in\boldsymbol{\mathcal{N}}_i}$ based on (6.5) and $p_i, \{p_j\}_{j\in\boldsymbol{\mathcal{N}}_i}$
4. *coordination*: update $w_i, \{w_{i\to j}\}_{j\in\boldsymbol{\mathcal{N}}_i}$ with
$$\arg\min_{w_i,\{w_{i\to j}\}_{j\in\boldsymbol{\mathcal{N}}_i}} \lambda_i^T(p_i - w_i) + \frac{\rho}{2}\|p_i - w_i\|^2 + \sum_{j=1}^{N_i}(\lambda_{i\to j}^T(p_j - w_{i\to j}) + \frac{\rho}{2}\|p_j - w_{i\to j}\|^2),$$
$$\text{subject to: } \overline{h}_{ij}(w_i, w_{i\to j}) \geq 0, \quad j \in \boldsymbol{\mathcal{N}}_i,$$
5. *mediation*: update $\lambda_i, \{\lambda_{i\to j}\}_{j\in\boldsymbol{\mathcal{N}}_i}\}$ with
$$\lambda_i \leftarrow \lambda_i + \rho(x_i - w_i)$$
$$\lambda_{i\to j} \leftarrow \lambda_{i\to j} + \rho(x_j - w_{i\to j}), \quad j \in \boldsymbol{\mathcal{N}}_i.$$
6. *communication II*: send $(\lambda_{i\to j}, w_{i\to j})$ to $j \in \boldsymbol{\mathcal{N}}_i$; receive $\{\lambda_{j\to i}, w_{j\to i}\}_{j\in\boldsymbol{\mathcal{N}}_i}$

until satisfaction of a stopping criterion

In the first step, following (5.8), the augmented Lagrangian of the problem is formed and minimised with respect to the x variables, which in this case are (p_i, v_i, a_i). The step is parallelisable; all the agents perform it at the same time in parallel. It is called *prediction* and is formulated below,

$$\arg\min_{p_i,v_i,a_i\in\boldsymbol{S}_i} \boldsymbol{F_i}(p_i, a_i) + \lambda_i^T(p_i - w_i) + \frac{\rho}{2}\|p_i - w_i\|^2 + \sum_{j=1}^{N_i}(\lambda_{j\to i}^T(p_i - w_{j\to i}) + \frac{\rho}{2}\|p_i - w_{j\to i}\|^2). \quad (7.2)$$

The Lagrangian variables λ_i and $\lambda_{j\to i}, j \in \boldsymbol{\mathcal{N}}_i$, as well as the collision free trajectories w and $w_{j\to i}, j \in$

$\mathcal{N}_i$ are initialised at the start locally for all the agents. The first term is the actual cost function, the next two are the augmented Lagrangian terms for the constraint $w_i = p_i$ with a constant ρ. In the final summation, the agent tries to stay close to the proposed trajectories for itself by its neighbours, in other words the augmented Lagrangian for the reversed constraint $w_{j\to i} = p_i, j \in \mathcal{N}_i$ is formed. As a whole, In this step the agents satisfy their own dynamics while staying close to the collision free trajectories w.

The second step is a *communication* step during which the current optimal position data p_i is shared by each agent with its neighbours. Each agent, thus receives the current estimate for $p_j, j \in \mathcal{N}_i$, the estimated trajectory of all of its neighbours.

During the third, *coordination*, step, following the classic ADMM structure, the augmented Lagrangian is minimised with respect to the z variables in (5.7), which correspond to $(w_i, w_{i\to j})$ in (7.1). As in the *prediction* step, *coordination* is also parallelisable and is formulated below for each agent i,

$$\arg \min_{w_i, \{w_{i\to j}\}_{j\in\mathcal{N}_i}} \lambda_i^T(p_i - w_i) + \frac{\rho}{2}\|p_i - w_i\|^2 + \sum_{j=1}^{N_i}(\lambda_{i\to j}^T(p_j - w_{i\to j}) + \frac{\rho}{2}\|p_j - w_{i\to j}\|^2), \tag{7.3a}$$

$$\text{subject to: } \overline{h}_{ij}(w_i, w_{i\to j}) \geq 0, \quad j \in \mathcal{N}_i, \tag{7.3b}$$

where the Lagrangian variables $\lambda_{i\to j}, j \in \mathcal{N}_i$ are initialised at the start in parallel for all agents. The first two terms, similar to the *prediction* step, are the augmented Lagrangian terms for the constraint $w_i = p_i$, and the summation is the augmented Lagrangian term for the constraints $w_{i\to j} = p_j, j \in \mathcal{N}_i$. The collision avoidance constraint is enforced for the duplicate variables, for which the nominal trajectories $(\overline{p}_i, \overline{p}_j)$ are used. These are received by the agents during the previous communication step and can be updated iteratively either at each MPC call, as in [44], or make use of the iterative nature of the ADMM and update at each ADMM step, as [24] proposes. With the latter we can only guarantee optimality of the solution if the iterations converge, however, this modification provides reduced conservatism and in most of the simulations it converges. The *coordination* step, thus, generates collision free trajectories w_i for each agent and proposed trajectories $w_{i\to j}$for their neighbours $j \in \mathcal{N}_i$. The the coupling linearised constraint is enforced on this, which is possible thanks to the communication step before.

During the next *mediation* step, following the ADMM in (5.7), the dual variables are updated, accumulating the gap between the predicted trajectories x and the collision free ones, w. This is given as,

$$\lambda_i \leftarrow \lambda_i + \rho(x_i - w_i) \tag{7.4a}$$

$$\lambda_{i\to j} \leftarrow \lambda_{i\to j} + \rho(x_j - w_{i\to j}), \quad j \in \mathcal{N}_i. \tag{7.4b}$$

The generated proposal trajectories by agent i for their neighbours, as well as the proposed Lagrangian variables are then communicated in the second *communication* step. More specifically each agent i sends its $w_{i\to j}$ and $\lambda_{i\to j}$ to its neighbours $j \in \mathcal{N}_i$, and receives from those $w_{j\to i}$ and $\lambda_{j\to i}$ respectively. After this final step, the algorithm is then repeated from the prediction step (7.2) until the iterations of all the agents converge.

The problem (7.1) is set up and solved using YALMIP [45] and the distributed algorithm [24] discussed above, exactly for the same configuration and design parameters as for the centralised solution shown in Figure 6.2 and discussed in Section 6.1. The Lagrangian variables and the collision free trajectories are all initialised with zeros and the constant ρ is set to 40. As discussed earlier, the nominal trajectories for the collision avoidance constraint are updated at each ADMM iteration for a higher potential for reducing conservatism. Thus, if it converges then the solution is optimal. We stop the algorithm after 35 iterations and the trajectories are plotted in the Figure 7.1 above. The objective residual between the distributed and centralised solutions for the current configuration is calculated to be around 5%, which is expected as the successive linearisation with the ADMM algorithm takes place differently. The trajectories, however, are similar to the centralised ones with the collision avoidance constraint implemented for the same Δ.

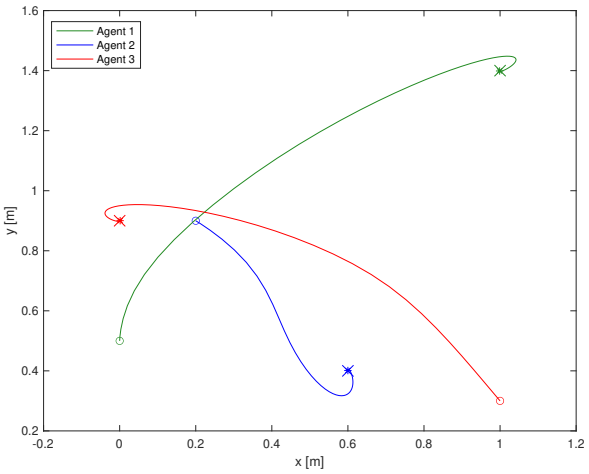


Figure 7.1: Trajectory tracking of three agents with double integrator dynamics, solved using the fully decentralised ADMM; $\Delta = 0.35$m

7.2 Full Linearized Model Integration

The fully decentralised ADMM algorithm discussed in the previous section was applied to the simple double integrator dynamics. Similar to Sections 6.2 and 6.3 for the centralised case, in this section we set up and apply the discussed algorithm to the problem (7.5) with the reduced quadrotor model in (6.9). As the OSQP solver proved to be very fast for the centralised problem, we set up everything to meet its format (6.14). The reformulated centralised collision avoidance problem, for the Crazyflie quadrotor dynamics $\boldsymbol{D}_i$ given in (6.9), is given as follows,

$$\begin{aligned} &\min_{s_i,w_i,u_i,w_{i\to j}} \sum_{i=1}^{M} \boldsymbol{F}_i(s_i,u_i) \\ &\text{subject to: } (s_i,u_i) \in \boldsymbol{D}_i \qquad \forall i \\ &\qquad \overline{h_{ij}}(w_i,w_{i\to j}) \geq 0 \qquad \forall j \in \boldsymbol{\mathcal{N}}_i \text{ and } i \\ &\qquad w_i = p_i; \;\; w_{i\to j} = p_j \;\; \forall j \in \boldsymbol{\mathcal{N}}_i \text{ and } i, \end{aligned} \tag{7.5}$$

where the state, input and position vectors $s_i \in \mathbb{R}^{n_x(N+1)}$, $u_i \in \mathbb{R}^{n_u N}$ and $p_i \in \mathbb{R}^{2(N+1)}$ are formed by the vertical concatenation of the variable vectors as explained in Section 6.2 and the w variables are just duplicate variables for p, naturally having the same dimensions.

We start with the *prediction* step (7.2) and formulate its augmented matrix version for each agent i for a horizon window size of N in Equation (7.6). The Lagrangian variable vectors are augmented, similarly to the decision variables, such that $\widetilde{\lambda}_i, \widetilde{\lambda}_{j\to i} \in \mathbb{R}^{2(N+1)} \; \forall j \in \boldsymbol{\mathcal{N}}_i$. The matrices $\widetilde{\boldsymbol{Q}}_i = \boldsymbol{Q} \otimes \boldsymbol{I}^{N+1} \in \mathbb{R}^{n_x(N+1)\times n_x(N+1)}$, $\widetilde{\boldsymbol{R}}_i = \boldsymbol{R} \otimes \boldsymbol{I}^{N+1} \in \mathbb{R}^{n_x(N+1)\times n_x(N+1)}$ are formed using the design matrices for each agent. The vector for the linear term is defined as $\widetilde{q}_i = [2(\boldsymbol{Q}r_i(1))^T \; \dots \; 2(\boldsymbol{Q}r_i(N+1))^T]^T \in \mathbb{R}^{n_x(N+1)}$ for the individual target states $r_i(k) \in \mathbb{R}^{n_x} \; \forall k$, and $\boldsymbol{H}^{N+1} = \boldsymbol{H} \otimes \boldsymbol{I}^{N+1} \in \mathbb{R}^{2(N+1)\times n_x(N+1)}$, with $\boldsymbol{H}$ defined in (6.21). The last matrix is introduced so that the state can be represented by just a single vector s_i.

$$\begin{aligned} &\arg\min_{s_i,u_i\in\boldsymbol{D}_i} s_i^T\widetilde{\boldsymbol{Q}}_i s_i + u_i^T\widetilde{\boldsymbol{R}}u_i \\ &\qquad - \widetilde{q}_i^T s_i + \widetilde{\lambda}_i^T p_i + \frac{M\rho}{2}({p_i}^T p_i) - \rho {w_i}^T p_i + \left(\sum_{j\in\boldsymbol{\mathcal{N}}_i} (\widetilde{\lambda}_{j\to i}^T - \rho w_{j\to i}^T)\right) p_i = \end{aligned} \tag{7.6a}$$

$$\begin{aligned} &= \arg\min_{s_i,u_i\in\boldsymbol{D}_i} s_i^T \left(\widetilde{\boldsymbol{Q}_i} + \frac{M\rho}{2}\boldsymbol{I}^{N+1}\right) s_i \\ &\qquad + \left(-\widetilde{q}_i^T + \left(\widetilde{\lambda}_i^T - \rho {w_i}^T + \sum_{j\in\boldsymbol{\mathcal{N}}_i} (\widetilde{\lambda}_{j\to i}^T - \rho w_{j\to i}^T)\right)\boldsymbol{H}^{N+1}\right) s_i + u_i^T\widetilde{\boldsymbol{R}}_i u_i \end{aligned} \tag{7.6b}$$

The state and input vectors can then be concatenated vertically into one, such that the problem can be represented with a single variable $v_i = [s_i \; u_i]^T$, similar to (6.17), and represented in an OSQP form of (6.14), as follows,

$$\begin{aligned} \min_{v_i} \;& \frac{1}{2} v_i^T \begin{bmatrix} \widetilde{\boldsymbol{Q}}_{id} & \boldsymbol{0} \\ \boldsymbol{0} & \widetilde{\boldsymbol{R}_i} \end{bmatrix} v - \widetilde{q}_{id}^T v_i \\ \text{subject to: } & [(\boldsymbol{A}_r - \boldsymbol{I}) \;\; \boldsymbol{B}_r] v_i = -v_{i0} \\ & u_{min} \leq \boldsymbol{A}_{iu} v_i \leq u_{max}, \end{aligned} \tag{7.7}$$

where the matrix $\widetilde{\boldsymbol{Q}}_{id} \in \mathbb{R}^{n_x(N+1)\times n_x(N+1)}$ and the vector $\widetilde{q}_{id} \in \mathbb{R}^{n_x(N+1)+n_u N}$ are the quadratic and linear terms in Equation (7.6), respectively. The vector $v_{i0} \in \mathbb{R}^{n_x(N+1)+n_u N}$ is the initial state of the agent i concatenated with zeros, as it was described for the centralised case in Section 6.3. Similar

to the centralised case, we set constraints on the velocity inputs as defined by the quadrotor limits. To do this we define the matrix $\boldsymbol{A}_{iu} \in \mathbb{R}^{n_u N \times n_x(N+1)+n_u N}$, where we have an identity matrix for the last $n_u N \times n_u N$ sized square matrix and zeros for the rest, i.e the states.

Using the above defined augmented states, inputs, vectors and matrices the *coordination* step (7.3) can now be reformulated as follows,

$$\begin{aligned} arg \min_{w_i, w_{i\to j}} \quad & \left(-\widetilde{\lambda}_i - \rho {p_i}^T\right) w_i + w_i \left(\frac{\rho}{2}\boldsymbol{I}^{2(N+1)}\right) w_i \\ & + \sum_{j \in \boldsymbol{\mathcal{N}}_i} \left(\left(-\widetilde{\lambda}_{i\to j}^T - \rho {p_j}^T\right) w_{i\to j} + w_{i\to j}^T \left(\frac{\rho}{2}\boldsymbol{I}^{2(N+1)}\right) w_{i\to j}\right) \end{aligned} \tag{7.8a}$$

$$\text{subject to:} \quad \widetilde{h}_{ij}(w_i, w_{i\to j}) \geq 0, \quad j \in \boldsymbol{\mathcal{N}}_i \tag{7.8b}$$

where the vectors $\widetilde{\lambda}_{i\to j} \in \mathbb{R}^{n_u(N+1)}$ and $w_{i\to j} \in \mathbb{R}^{2(N+1)}$ are formed as for the previous step and the constraint (7.8b) is the linearised collision avoidance constraint for the augmented states. Now, to formulate everything in an OSQP form, as was done (7.7), we first define a single augmented state for agent i, for a horizon window size of N, as $\widetilde{w}_{ij} \in \mathbb{R}^{2(N+1)M}$. This is given below, as,

$$\widetilde{w}_{ij} = [w_i(0) \ \dots \ w_i(N) \ \dots w_{i\to 1}(0) \ \dots \ w_{i\to 1}(N) \ \dots w_{i\to N_i}(0) \ \dots \ w_{i\to N_i}(N)]^T.$$

The matrix formulation for (7.8) is then given below, as,

$$\begin{aligned} \min_{\widetilde{w}_{ij}} \quad & \frac{1}{2}\widetilde{w}_{ij}^T \boldsymbol{P}_{ij} \widetilde{w}_{ij} - \widetilde{q}_{ij}^T \widetilde{w}_{ij} \\ \text{subject to:} \quad & \boldsymbol{A}_{iw}\widetilde{w}_{ij} \geq \widetilde{l}_{iw}, \end{aligned} \tag{7.9}$$

where $\boldsymbol{P}_{ij} = \boldsymbol{I}^M \otimes \frac{\rho}{2}\boldsymbol{I}^{2(N+1)} \in \mathbb{R}^{2(N+1)M \times 2(N+1)M}$ and $\widetilde{q}_{ij} \in \mathbb{R}^{2(N+1)M}$ are the quadratic and linear terms in (7.8), respectively. The collision avoidance constraint vector $\widetilde{l}_{iw} \in \mathbb{R}^{N_i N}$ is formed in Equation (7.10) by concatenating the vectors $l_{iw}(k) \in \mathbb{R}^{N_i}$ for all $k = 1, \dots, N$. These, in turn, are formed by the vertical concatenation of the values l_{ij}, defined in (6.20), but only for the agent i and its neighbourhood $j \in \boldsymbol{\mathcal{N}}_i$. Thes are linearised around the newly communicated position vectors $(\overline{p}_i, \overline{p}_j)$.

$$\widetilde{l}_{iw} = \begin{bmatrix} l_{i1}(1) & \dots & l_{iN_i}(1) & \dots & l_{i1}(N) & \dots & l_{iN_i}(N) \end{bmatrix}^T \tag{7.10}$$

$$\boldsymbol{\eta}_i^M(k) = \begin{bmatrix} \eta_{i1}^T(k) & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \eta_{iN_i}^T(k) \end{bmatrix} \tag{7.11}$$

$$\boldsymbol{\eta}_i^M = \begin{bmatrix} \boldsymbol{\eta}_i^M(1) & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \boldsymbol{\eta}_i^M(N) \end{bmatrix} \tag{7.12}$$

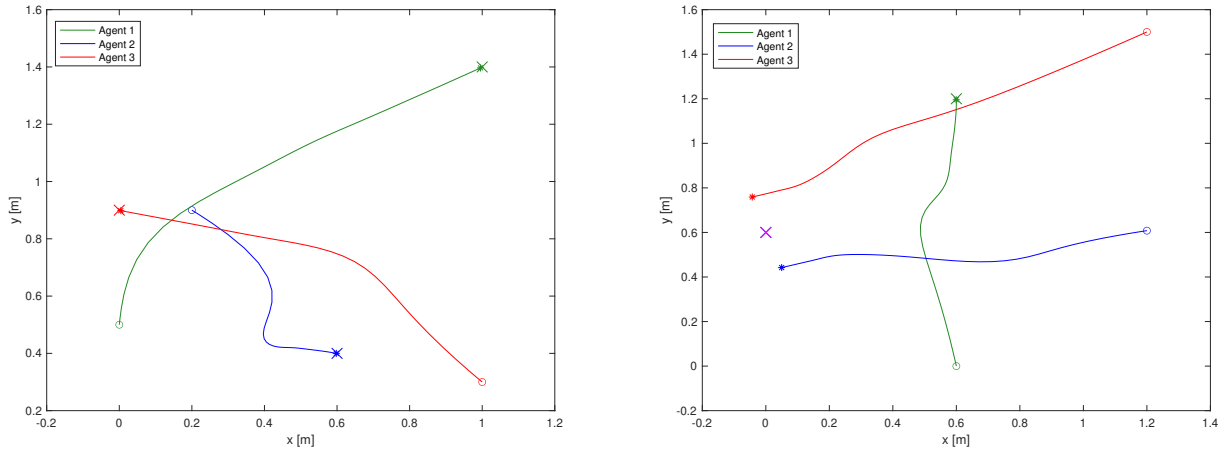
Similar to the formulations in Section 6.3, the matrix $\boldsymbol{\eta}_i(k) \in \mathbb{R}^{N_i \times 2N_i}$ is first formed in (7.11), defined for the timestep k. It is then augmented for all $k = 1, \dots, N$ to form $\boldsymbol{\eta}_i^M \in \mathbb{R}^{N_i N \times 2(N) N_i}$ in (7.12). Using this, the constraint matrix is formulated as $\boldsymbol{A}_{iw} = \boldsymbol{\eta}_i^M \boldsymbol{H}_{iw} \in \mathbb{R}^{N_i N \times 2(N+1)M}$. To form the matrix $\boldsymbol{H_{iw}} \in \mathbb{R}^{2(N)N_i \times 2(N+1)M}$, we first define the following to aid us,

$$\boldsymbol{H}_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ \vdots & \vdots \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{7.13}$$

$$h_j = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \begin{matrix} 1 \\ \vdots \\ j \\ \vdots \\ N_i \end{matrix}, \tag{7.14}$$

with $\boldsymbol{H}_1 \in \mathbb{R}^{2N_i \times 2}$ and $h_j \in \mathbb{R}^{N_i}$, which is a vector of zeros, except for its j^{th} element. Then, the horizontal concatenation of matrices $\boldsymbol{I}_N^{N+1} \otimes \boldsymbol{H}_1$ and $\boldsymbol{I}_N^{N+1} \otimes (-\boldsymbol{I}^2) \otimes h_j, \quad \forall j \in \boldsymbol{\mathcal{N}}_i$, gives the $\boldsymbol{H}_{iw}$ matrix for agent i, where $\boldsymbol{I}_N^{N+1} \in \mathbb{R}^{N \times (N+1)}$ is defined in Section 6.3.

The problem is set up following the algorithm and the OSQP formulations of its two optimisation problems (7.7) and (7.9) and solved for exactly the same design matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$, and the same horizon window size $N = 100$ as in Section 6.3. We use the same two configurations as in Figure 6.4; this allows us to calculate the objective residual between the centralised and the distributed solutions to get a quantitative estimate of the proximity for those. The results are shown in Figure 7.2, for the constant value of $\rho = 40$ and for 45 iterations. We can see that for configuration 1 Figures 7.2a and 6.4a hardly differ, with the collision avoidance, and input constraints still satisfied. For this configuration the objective residual is calculated to be only 0.38%. For the second configuration, for the distributed and centralised solutions in Figures 7.2b and 6.4b, respectively, the only discernible difference is the path that the agent one (in green) takes. The objective residual for this case is calculated to be just 0.5%, meaning that both of the solutions reach very close to the same optimal objective value, while having different solutions, as the trajectories are the solved optimal solutions of the problems. The duration of the algorithm is estimated to take about 500 milliseconds for each



(a) Configuration 1; $\Delta = 0.35$m

(b) Configuration 2; $\Delta = 0.33$m

Figure 7.2: Trajectory tracking of three agents with 2 different configurations and Δ-s, generated in a distributed fashion for a horizon window size of $N = 100$.

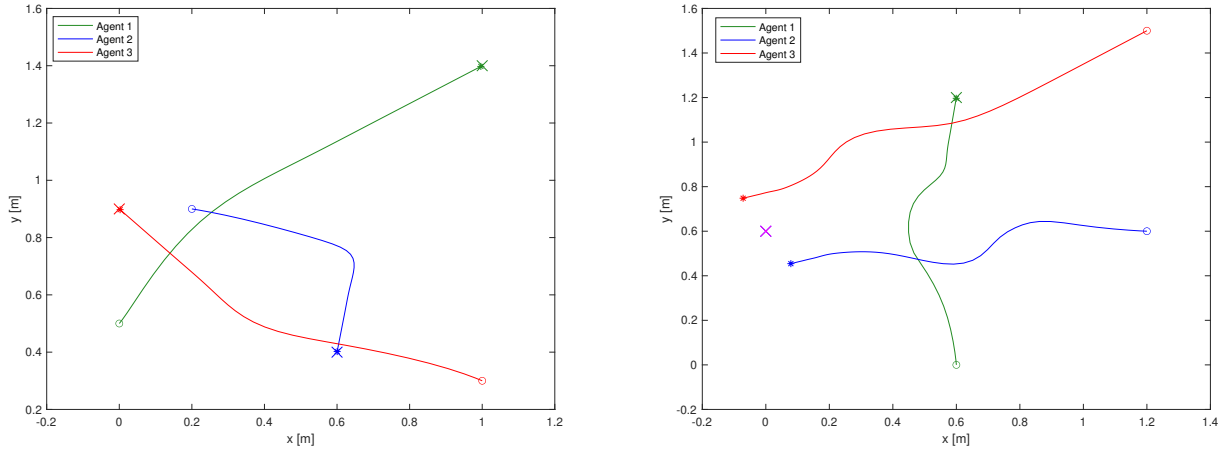
agent, however, this is for very large number of iterations and for a large horizon window size. In

practice, when implementing such an algorithm, MPC will be used to decrease the computation time and introduce closed loop feedback. This is discussed in the following section.

7.3 MPC Setup

The model predictive control was discussed in detail in Section 6.4 and applied to the centralised collision avoidance problem with Crazyflie quadrotor dynamics in 6.4.2. To ensure that the finite sized receding horizon implementation behaves as if it was an infinite one, a dual mode was introduced. In the first mode we have a finite sized horizon, and in the second the inputs are calculated by a feedback rule: $u = \boldsymbol{K}x$. As it was discussed in Section 6.4, this is equivalent of having a special terminal weight matrix $\boldsymbol{Q}_N$, which must be the solution of the Lyapunov Equation (6.31).

The fully decentralized algorithm [24] is set up with OSQP [46], discussed in detail in Sections 7.1 and 7.2 for the Crazyflie quadrotor dynamics. The design parameters $\boldsymbol{Q}$, $\boldsymbol{R}$ and N are kept the same as in the centralised MPC setup in Section 6.4.2, and the value of ρ is the same compared to the previous section, for direct comparison. After the ADMM iterations terminate, some optimal control inputs $u_i^\star(k) \in \mathbb{R}^{n_u N}$ are generated for the current timestep k. As the MPC algorithm suggests, the initial position v_{i0} is updated in (7.7) and the first control input $u_0^\star(k)$ is applied to the plant, or the model in simulation. We note the important point, that the update of the nominal trajectories for the linearisation of the collision avoidance constraint in (7.9), does not happen at this point, during the serial MPC call, but instead during the iterations of the ADMM loop, as proposed by Modification 1 in [24]. As we discussed earlier, this ensures optimality if the solutions converge and potentially provides less conservatism, thanks to the more frequent updates of these trajectories. The results for the two configurations used previously are shown in Figure 7.3 below.



(a) Configuration 1; $\Delta = 0.35$m

(b) Configuration 2; $\Delta = 0.33$m

Figure 7.3: Trajectory tracking of three agents with 2 different configurations and Δ-s, generated in a distributed, receding horizon fashion for a horizon window size of $N = 10$.

We can see the resemblance of the Figures to the centralised MPC implementation for the same configurations in Figure 6.5. The ADMM loop here is terminated after 15 iterations, and takes no more 100 milliseconds per agent per loop. This is much faster compared to the tests we conducted for MPC configuration without an OSQP setup.

8 Implementation on Quadrotors

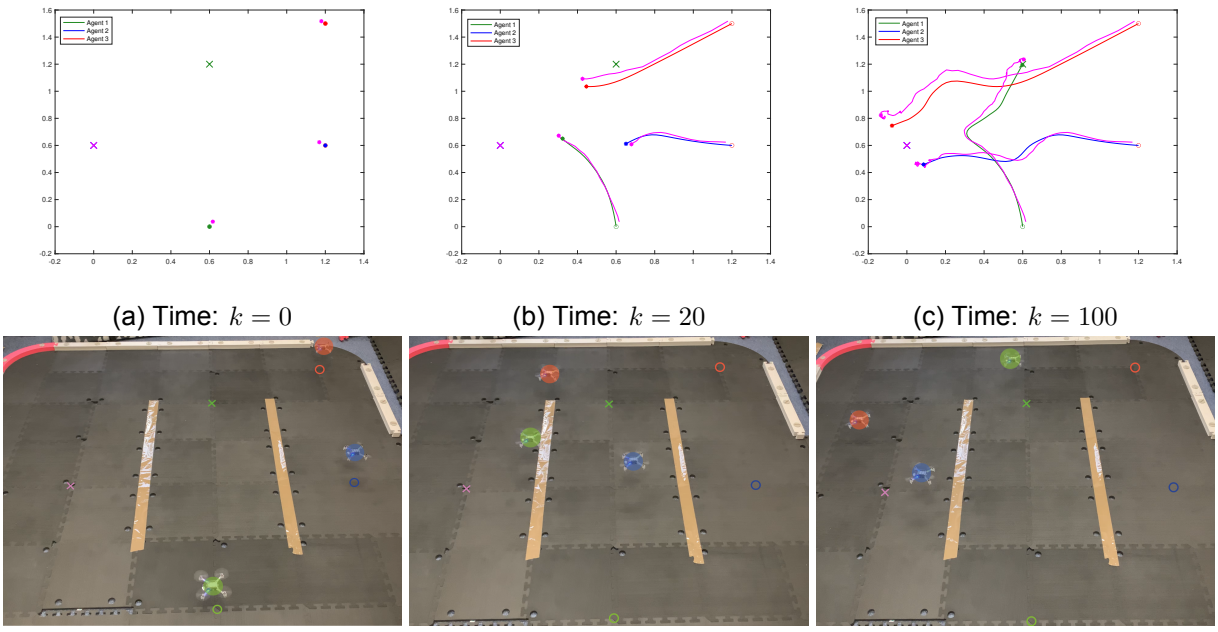
In the previous sections the multi-agent collision avoidance problem was simulated successfully both with centralised and distributed setups and in a receding horizon fashion for each, proving that in an ideal environment without any model errors and external or internal disturbances, the agents are able to track individual trajectories while staying in their velocity limits and avoiding collisions with neighbours. Moreover, we also showed in Sections 6.4.2 and 7.3, that given the problem is feasible for all future timesteps, then we also achieve closed loop stability. Following the Section 1.2 and the Figure 1.2, in this section, a central system using Python and MATLAB [32, 31] is set up to implement the generated optimal inputs from the MPC centralised simulations in Section 6.4.2. The full and reduced models discussed in Section 6.2 are also compared.

8.1 Swarm Setup and Optimal Control Implementation

The MPC setup in Section 6.4.2 provides optimal control inputs for a three agent collision avoidance trajectory tracking problem (6.26) simulated in a receding horizon fashion with a window size of $N = 10$. We then want to implement these optimal trajectories on three Crazyflies in an open loop fashion to test how accurate the model is, the reliability of the communication with the CrazyRadio PA and to set a practical arrangement for swarm control for later developments.

The hardware and software for the Crazyflies are set up according to Section 3 by equipping the quadrotors with the Flow Deck sensors and updating their firmware with latest version [36]. In addition to the prerequisites such as localisation and data acquisition, a common frame of reference has to be established as no central positioning system is used. To solve this, each drone is placed in a predefined starting position and initialised using the same coordinate system for the whole network. This is done at the startup by setting the initial positions of the Flow Deck Kalman filter in the Python script. In Section 4.3 two control schemes were introduced, and as discussed earlier, for the multi agent problem we use the on board mode, presented in detail in Section 4.3.2 and shown in Figure 4.8. In this mode the low level control for single quadrotor stabilisation is performed on board the quadrotors, in the firmware, achieving faster and more reliable dynamics and wider available band-

width. Hence, the optimisation based controller that generates optimal inputs by solving the problem (6.26) represents the high level controller $\boldsymbol{K}_h$ in Figure 4.8, with the exception that we are not closing the loop with the reduced state s_r, but applying the inputs for a certain horizon window in an open loop. The optimisation problem (6.26) is solved in MATLAB [31] and the optimal inputs, u_i are saved for each agent i. The Python software then reads these inputs, which are the reference velocity inputs in (x, y) directions, as described in 6.2, and sends these to the quadcopters to be applied every $T = 0.1$ seconds [32], as shown in Figure 4.8. One CrazyRadio PA is used for each drone, operating at 2MBit/s Bandwidth and with a separate, unique, radio channel. Then, using the ROS system, as described in Section 3.1, the position data of the drones is stored in individual rosbags (special data files) for analysis.

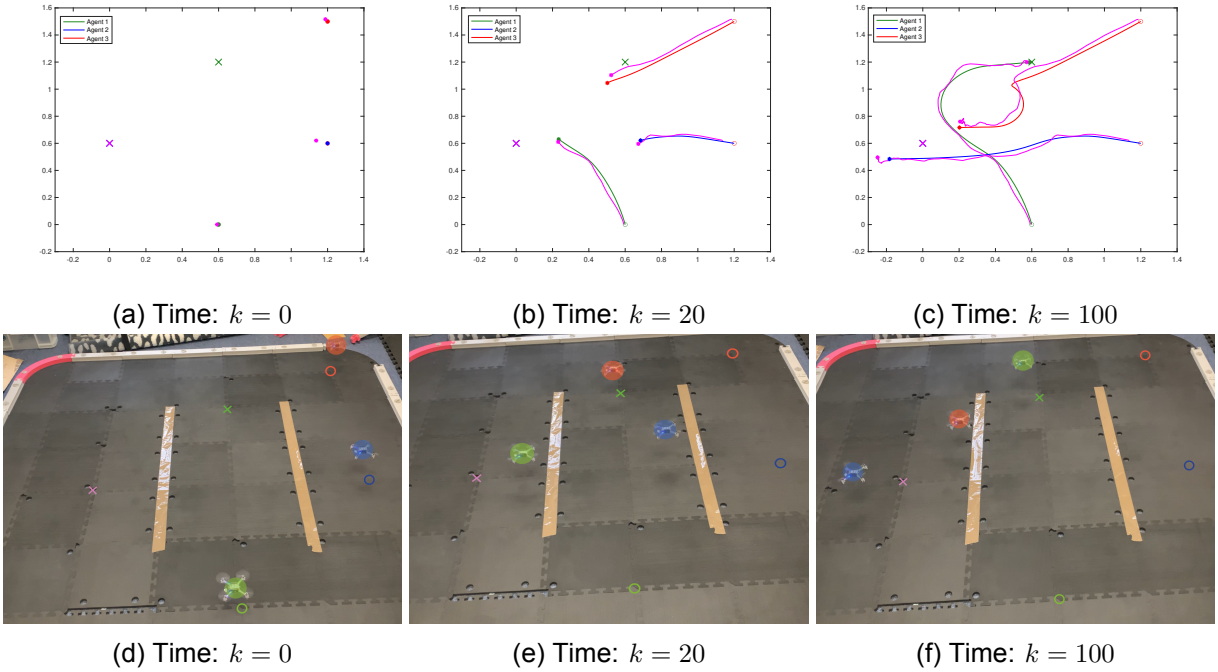


(a) Time: $k = 0$ (b) Time: $k = 20$ (c) Time: $k = 100$

(d) Time: $k = 0$ (e) Time: $k = 20$ (f) Time: $k = 100$

Figure 8.1: Trajectory tracking of three agents for $\Delta = 0.33$m, generated in a receding horizon fashion for a horizon window size of $N = 10$, at 3 different time steps. The purple trajectories in simulations (a)-(c) show the real quadrotor positions, the crosses denote the targets, and the circles the initial positions of the setup. The Figures (d) - (f) show the snapshots of the recorded flight.

The problem (6.26) is set up and solved exactly as in Section 6.4.2 with the same design parameters for the second configuration shown in Figure 6.5b. It is solved for two safety radii $\Delta = 0.33$m and $\Delta = 0.45$m and the optimal inputs are applied as described. The results are shown in Figures 8.1 and 8.2 for the two radii, respectively. The flight videos can be accessed online [51, 52]. The purple shadow trajectories in these figures denote the Crazyflie estimated trajectories saved by the ROS system. The first conclusion that can be drawn from the above tests is that there is a very close match between the measured trajectories of the Crazyflie quadrotors and the simulated, predicted ones, as verified in the figures. This is particularly pleasing, as it is implemented in an open loop



(a) Time: $k = 0$ (b) Time: $k = 20$ (c) Time: $k = 100$

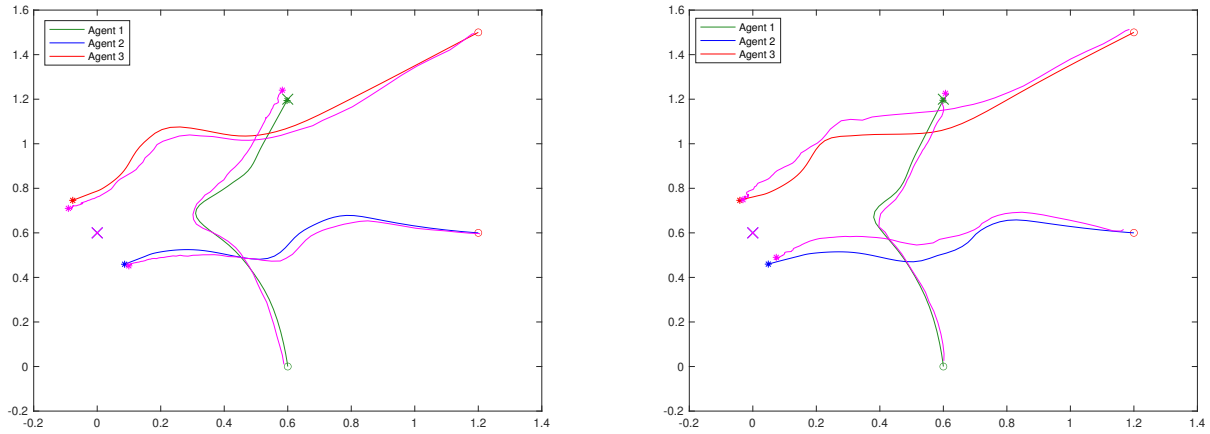
(d) Time: $k = 0$ (e) Time: $k = 20$ (f) Time: $k = 100$

Figure 8.2: Trajectory tracking of three agents for $\Delta = 0.45$m, generated in a receding horizon fashion for a horizon window size of $N = 10$, at 3 different time steps. The purple trajectories in simulations (a)-(c) show the real quadrotor positions, the crosses denote the targets, and the circles the initial positions of the setup. The Figures (d) - (f) show the snapshots of the recorded flight.

fashion. In the final states for both of the configurations in Figures 8.1c and 8.2c, we can observe that the quadrotor measured trajectories (in purple) at some points deviate from the predicted ones, although not much. We conjecture that this occurs due to a number of reasons. Firstly, as we can see from Figures 8.1a and 8.2a, the start positions for all of the agents are slightly shifted. This is due to two factors: the imperfect function of the Flow Deck sensor under 20cm, causing the drone to vibrate and shift slightly, and the corner effect of the flooring, which greatly affects the state estimation of the Flow Deck sensor, as mentioned in Section 4.3.1. This latter is particularly visible for the agent three, as it is the closest to the corner of the flooring. The second reason for such small deviations is the intermittent breakup or slowdown of the communication links due to packet drops and losses. This is verified by actuating the same inputs one by one and observing a much closer fit, as shown in Figure 8.3a, proving that a higher load on the communication bandwidth may cause sudden breakups and jitter. Finally, the Flow Deck depends on the flooring conditions to work perfectly; having an uneven floor may introduce vibrations and inaccurate readings. The first two of the mentioned problems can be solved to a great extent by introducing a closed loop on-line implementation using the developed MPC setup, as explained in Section 9.1. The last one is solved by using a flooring working perfectly with Flow Deck to fix the inaccuracies in the readings.

8.2 Full Model Comparison

In Section 6.2 we reduced the ten state system to a six state one, which has been used throughout the report for simulations, as well as practical implementations. We justified this simplification by arguing that the additional states we ignored are all stable and that their influence was very low. This is an indication that the coefficients for these states were non-zero only because of the numerical errors introduced by using the MATLAB ”linmod” function and the SIMULINK program. To get an understanding of the quality of our approximation, we repeat the test in the previous section with an MPC setup and an open-loop implementation for the full discretized model (6.8) by only reducing the inputs to be the (x, y) reference velocities. We then compare the predicted and measured trajectories as before. For the same setup with the same configuration and design specifications, with the only slightly different $\Delta = 0.3$m we get the trajectories shown in Figure 8.3b as compared to the simplified reduced model results in 8.3a. These are implemented for all agents in sequence to remove the effects of communication. Although the trajectories still match very well for an open loop implementation, this



(a) Simplified Model; $\Delta = 0.33$m

(b) Full Model; $\Delta = 0.3$m

Figure 8.3: Trajectory tracking of three agents, simulated using a simplified(a) and a full(b) model generated in a receding horizon fashion for a horizon window size of $N = 10$. The purple trajectories in simulations show the real quadrotor positions, the crosses denote the targets, and the circles the initial positions of the setup

full model match is worse than the one for the reduced one as shown in Figures 8.1 and 8.2. For all of the tests with different configurations that we performed, this pattern persisted with the match for the full model always being worse than the reduced one. This supports the fact that either the conjecture that low value state coefficients were generated by numerical errors is true or that the full model is too accurate and somehow overfitting takes place. In either of the cases, using the smaller model, which showed a very good match with the real system as verified earlier is the more logical choice for shorter computation time, simplicity and better match.

9 Conclusion

This project was aimed at achieving autonomous and stable flight for a swarm of Crazyflie quadrotors. We modelled and successfully executed the three constituent subtasks towards achieving this goal. The small size and the open-source software and firmware of the quadcopters allowed us to implement some of the simulated algorithms in practice and verify the accuracy of both the system models and the networked collision avoidance algorithms.

The control of a single quadrotor was considered by developing a linearised model and introducing off-board and on-board control modes with a local sensor, removing the need for motion capture systems. The tests in both modes under a LQR optimal control proved a near perfect match between the linearised model and the non-linear plant with a height error of no more than 2cm for flights at 30cm and higher. This linearised model was then simplified further and used for the multi agent problem.

A review of networked consensus and flocking problems was conducted. The iterative discrete algorithms for decentralised and distributed problems provide more flexibility than potential field approaches by allowing the incorporation of more complex dynamics and objective functions. A centralised collision avoidance and trajectory tracking problem was set up and then distributed using a variation of the ADMM algorithm. Linearising the nonconvex collision avoidance constraint, the problems were solved for the developed Crazyflie model using the OSQP solver [46] with a horizon window size of 100 and a discretization step of 0.1 seconds. The objective residual between the centralised and distributed problems was calculated to be less than 1%. For each setup a feedback loop was then introduced with a receding horizon implementation for a horizon window size of 10 and the same discretization. The Table 2 below shows the computation times for each of these setups. Note that the centralised MPC is faster than the Crazyflie dynamics, making on-line closed loop control possible. The distributed case is discussed in the next subsection. The generated optimal inputs for the centralised MPC problem were then implemented on Crazyflie quadrotors in an open loop fashion using the on-board control mode. In the absence of any model errors and disturbances, the results showed a very close match between the predicted and observed trajectories with only minor deviations from their optimal path. One of the keys towards achieving true autonomy is the independence from any

Setup	Centralised: (Section 6.3)	Centralised MPC (Section 6.4.2)	Distributed (Section 7.2)	Distributed MPC (Section 7.3)
Computation time per agent (*ms*)	400	20 (per MPC call)	500	100 (per MPC call)

Table 2: Completion times for the centralised and distributed problems. The problems with a MPC setup were solved for a horizon window size of $N = 10$, the others for $N = 100$

centralised localisation systems. Keeping this level of autonomy, we successfully set up a system to control both single and networked quadrotors that track trajectories and avoid collisions while logging and analysing the data. This allows future users to build and improve upon, integrating more features.

9.1 Future Work and Extensions

There are a few directions for the project's further development and improvement. We discuss some of these, explaining how those can be built on the current setup. Firstly, to achieve a closed loop implementation, at each MPC call the initial states of all the agents, whether in centralised or distributed setups, are updated with their current estimated states. This closes the outer feedback loop in Figure 4.8. By the time the quadcopters communicate their position, they actuate the current optimal input. This is possible if the agents implement only the previous computed inputs. Naturally, this computation time has to be less than the discretization step T. The distributed MPC problem, presented in Section 7.3 is updated to incorporate this delay and T is reduced to 0.05 seconds. It is solved in less than 45 milliseconds for a horizion window of $N = 20$ and eight ADMM iterations. Thus, it leaves some time to account for communication delays. The simulation is obviously, much smoother, and thanks to the faster model, the one timestep delay does not affect the stability, as shown in Figure 9.1. Making use of the already setup ROS system and callback functions, this MPC loop can then be closed on the computer. This can then be improved further by eliminating the computer for performing the optimisation, and programming everything directly in the firmware. The newly introduced peer to peer communication protocol [53] can enable the information exchange. This will greatly reduce the communication and operating system delays imposed by the CrazyRadio and the computer. Further areas of extension include a more detailed analysis of the reduction of the Crazyflie full model, and the possible improvement upon it. Furthermore, the robustness of the networked system can be considered either by using stochastic control techniques [25] or by a more traditional robust control and robust model predictive control analysis [48]. These would consider the effects of uncertainties in the model and the environment, such as winds, when flown outdoors, damaged propellers, motors, or sensor and battery failures.

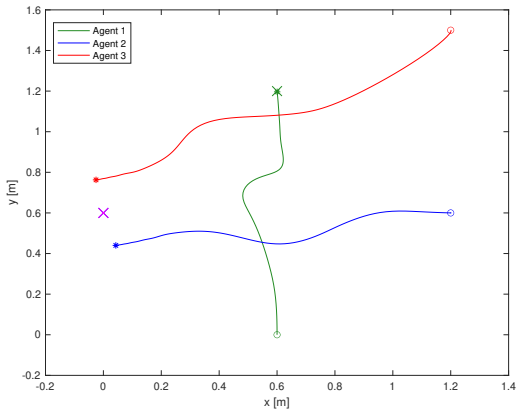


Figure 9.1: Distributed trajectory tracking of three agents, simulated in a receding horizon fashion for a horizon window size of $N = 20$, $\Delta = 0.33$m, discretization step $T = 0.05$ s and one timestep delay.

References

[1] M. A. Goodrich, B. S. Morse, D. Gerhardt, J. L. Cooper, M. Quigley, J. A. Adams, and C. Humphrey, "Supporting wilderness search and rescue using a camera-equipped mini uav," *Journal of Field Robotics*, vol. 25, no. 1-2, pp. 89–110, 2008.

[2] F. Veroustraete, "The rise of the drones in agriculture," *EC agriculture*, vol. 2, no. 2, pp. 325–327, 2015.

[3] S. S. Kia, B. Van Scoy, J. Cortes, R. A. Freeman, K. M. Lynch, and S. Martinez, "Tutorial on dynamic average consensus: The problem, its applications, and the algorithms," *IEEE Control Systems Magazine*, vol. 39, no. 3, pp. 40–72, 2019.

[4] BitCraze, "Crazyflie 2.1." `https://www.bitcraze.io/crazyflie-2-1/`, 2020. [Online; accessed 27-October-2019].

[5] BitCraze, "Bitcraze github page." `https://github.com/bitcraze`, 2020. [Online; accessed 27-January-2020].

[6] M.-D. Hua, T. Hamel, P. Morin, and C. Samson, "Introduction to feedback control of underactuated vtolvehicles: A review of basic control design ideas and principles," *IEEE Control systems magazine*, vol. 33, no. 1, pp. 61–75, 2013.

[7] J. Li and Y. Li, "Dynamic analysis and pid control for a quadrotor," in *2011 IEEE International Conference on Mechatronics and Automation*, pp. 573–578, 2011.

[8] P. Castillo, R. Lozano, and A. Dzul, "Stabilization of a mini-rotorcraft having four rotors," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, pp. 2693–2698 vol.3, 2004.

[9] E. Reyes-Valeria, R. Enriquez-Caldera, S. Camacho-Lara, and J. Guichard, "LQR control for a quadrotor using unit quaternions: Modeling and simulation," in *CONIELECOMP 2013, 23rd International Conference on Electronics, Communications and Computing*, pp. 172–178, IEEE, 2013.

[10] R. Xu and U. Ozguner, "Sliding mode control of a quadrotor helicopter," in *Proceedings of the 45th IEEE Conference on Decision and Control*, pp. 4957–4962, IEEE, 2006.

[11] S. Bouabdallah and R. Siegwart, "Full control of a quadrotor," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 153–158, Ieee, 2007.

[12] P. N. Beuchat, "N-rotor vehicles: modelling, control, and estimation," 2019-02-18.

[13] M. Greiff, "Modelling and control of the crazyflie quadrotor for aggressive and autonomous flight by optical flow driven state estimation," 2017.

[14] G. P. Subramanian, *Nonlinear control strategies for quadrotors and CubeSats*. PhD thesis, 2015.

[15] N. O. Lambert, D. S. Drew, J. Yaconelli, S. Levine, R. Calandra, and K. S. Pister, "Low-level control of a quadrotor with deep model-based reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4224–4230, 2019.

[16] A. Molchanov, T. Chen, W. Hönig, J. A. Preiss, N. Ayanian, and G. S. Sukhatme, "Sim-to-(multi)-real: Transfer of low-level robust control policies to multiple quadrotors," *arXiv preprint arXiv:1903.04628*, 2019.

[17] J. Coulson, J. Lygeros, and F. Dörfler, "Data-enabled predictive control: In the shallows of the deepc," in *2019 18th European Control Conference (ECC)*, pp. 307–312, IEEE, 2019.

[18] W. Ren, R. W. Beard, and E. M. Atkins, "Information consensus in multivehicle cooperative control," *IEEE Control systems magazine*, vol. 27, no. 2, pp. 71–82, 2007.

[19] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pp. 25–34, 1987.

[20] R. Olfati-Saber, "Flocking for multi-agent dynamic systems: Algorithms and theory," *IEEE Transactions on automatic control*, vol. 51, no. 3, pp. 401–420, 2006.

[21] Z. Lin, L. Wang, Z. Han, and M. Fu, "Distributed formation control of multi-agent systems using complex laplacian," *IEEE Transactions on Automatic Control*, vol. 59, no. 7, pp. 1765–1777, 2014.

[22] M. Porfiri, D. G. Roberson, and D. J. Stilwell, "Tracking and formation control of multiple autonomous agents: A two-level consensus approach," *Automatica*, vol. 43, no. 8, pp. 1318–1328, 2007.

[23] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and distributed computation: numerical methods*, vol. 23. Prentice hall Englewood Cliffs, NJ, 1989.

[24] F. Rey, Z. Pan, A. Hauswirth, and J. Lygeros, "Fully decentralized ADMM for coordination and collision avoidance," in *2018 European Control Conference (ECC)*, pp. 825–830, IEEE, 2018.

[25] K. Margellos, A. Falsone, S. Garatti, and M. Prandini, "Distributed constrained optimization and consensus in uncertain networks via proximal minimization," *IEEE Transactions on Automatic Control*, vol. 63, no. 5, pp. 1372–1387, 2017.

[26] A. Falsone, I. Notarnicola, G. Notarstefano, and M. Prandini, "Tracking-ADMM for distributed constraint-coupled optimization," *arXiv preprint arXiv:1907.10860*, 2019.

[27] J. A. Preiss, W. Honig, G. S. Sukhatme, and N. Ayanian, "Crazyswarm: A large nano-quadcopter swarm," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3299–3304, IEEE, 2017.

[28] S. Green and P. Månsson, "Autonomous control of unmanned aerial multi-agent networks in confined spaces," 2019.

[29] J. Jaramillo, T. Wieczorek, D. M. Tran, V. S. Dadi, T. Yucelen, and S. Chellappan, "Experimental validation of a distributed control approach based on multiplex networks on formations of unmanned aerial vehicles," in *AIAA Scitech 2019 Forum*, p. 1190, 2019.

[30] Aren Karapetyan, "Modified firmware." `https://github.com/akarapet/EUROP_Crazy`, 2020. [Online; accessed 6-April-2020].

[31] Aren Karapetyan, "Matlab code for optimisation." `https://github.com/akarapet/admm_collision_avoidance`, 2020. [Online; accessed 24-March-2020].

[32] Aren Karapetyan, "Python code for swarm control." `https://github.com/akarapet/dist_swarm`, 2020. [Online; accessed 24-March-2020].

[33] BitCraze, "Crazyflie system architecture." `https://www.bitcraze.io/2014/07/crazyflie-2-0-system-architecture/`, 2020. [Online; accessed 28-October-2020].

[34] BitCraze, "Crazyradio pa." `https://www.bitcraze.io/crazyradio-pa/`, 2020. [Online; accessed 27-January-2020].

[35] BitCraze, "Flow deck v2." `https://www.bitcraze.io/flow-deck-v2/`, 2020. [Online; accessed 27-January-2020].

[36] BitCraze, "Bitcraze github page." https://github.com/bitcraze/crazyflie-firmware, 2020. [Online; accessed 6-April-2020].

[37] Stanford Artificial Intelligence Laboratory et al., "Robotic operating system."

[38] BitCraze, "Bitcraze flow deck setup." https://www.bitcraze.io/documentation/tutorials/getting-started-with-flow-deck/, 2020. [Online; accessed 6-April-2020].

[39] W. Ren and Y. Cao, "Distributed coordination of multi-agent networks. communications and control engineering series," 2011.

[40] F. Augugliaro, A. P. Schoellig, and R. D'Andrea, "Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach," in *2012 IEEE/RSJ international conference on Intelligent Robots and Systems*, pp. 1917–1922, IEEE, 2012.

[41] F. Bullo, J. Cortes, and S. Martinez, *Distributed control of robotic networks: a mathematical approach to motion coordination algorithms*. Princeton University Press, 2009.

[42] W. Zhu, Z.-P. Jiang, and G. Feng, "Event-based consensus of multi-agent systems with general linear models," *Automatica*, vol. 50, no. 2, pp. 552–558, 2014.

[43] Y. Kuwata and J. P. How, "Cooperative distributed robust trajectory optimization using receding horizon milp," *IEEE Transactions on Control Systems Technology*, vol. 19, no. 2, pp. 423–431, 2010.

[44] H. Zheng, R. R. Negenborn, and G. Lodewijks, "Fast ADMM for distributed model predictive control of cooperative waterborne agvs," *IEEE Transactions on Control Systems Technology*, vol. 25, no. 4, pp. 1406–1413, 2016.

[45] J. Löfberg, "Yalmip: A toolbox for modeling and optimization in matlab," in *Proceedings of the CACSD Conference*, vol. 3, Taipei, Taiwan, 2004.

[46] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: An operator splitting solver for quadratic programs," *ArXiv e-prints*, Nov. 2017.

[47] BitCraze, "Crazyflie max velocity." https://forum.bitcraze.io/viewtopic.php?t=3296, 2020. [Online; accessed 24-March-2020].

[48] B. Kouvaritakis and M. Cannon, "Model predictive control," *Switzerland: Springer International Publishing*, 2016.

[49] A. S. Poznyak, *Advanced mathematical tools for automatic control engineers: Stochastic techniques*. Elsevier, 2009.

[50] R. E. Kalman *et al.*, "Contributions to the theory of optimal control," *Bol. soc. mat. mexicana*, vol. 5, no. 2, pp. 102–119, 1960.

[51] Aren Karapetyan, "Collision avoidance for 0.33m, video." https://youtu.be/tC_gWjCMjts, 2020. [Online; accessed 17-May-2020].

[52] Aren Karapetyan, "Collision avoidance for 0.45m, video." https://youtu.be/RO9rYhEk4Ic, 2020. [Online; accessed 17-May-2020].

[53] BitCraze, "Bitcraze peer to peer api." https://www.bitcraze.io/documentation/repository/crazyflie-firmware/2020.02/p2p_api/, 2020. [Online; accessed 28-April-2020].