Trajectory planning and control for drone replacement during formation flight

Marta Isabel da Silva Marques

Thesis to obtain the Master of Science Degree in Aerospace Engineering

Supervisors: Prof. Bruno João Nogueira Guerreiro
Prof. Rita Maria Mendes de Almeida Correia da Cunha

Examination Committee
Chairperson: Prof. José Fernando Alves da Silva
Supervisor: Prof. Bruno João Nogueira Guerreiro
Member of the Committee: Prof. João Manuel Lage de Miranda Lemos

November 2018
Acknowledgments

I would like to thank my thesis advisors, Prof. Bruno Guerreiro and Prof. Rita Cunha, for their support throughout the duration of this thesis. I would specially like to thank Prof. Bruno Guerreiro for his guidance and suggestions during our weekly meetings and for always being available to answer my questions. I would also like to thank them for lending me a computer so that I could test my algorithms.

Furthermore, I would like to thank my friends for their support and for motivating me in this last five years. And, finally, I would like to thank my family for always believing in me and for being a constant support system.

This work was partially funded by the FCT projects LOTUS (PTDC/EEIAUT/5048/2014), REPLACE (LISBOA-01-0145-FEDER-032107), and LARSys (UID/EEA/50009/2013), as well as by the EU’s Horizon 2020 programme (grant No 731667, MULTIDRONE).
Resumo

Esta tese aborda o estudo de técnicas de planeamento e de controlo para múltiplos drones, de modo a permitir manobras de substituição de drones de forma segura e eficiente.

Numa primeira parte, é apresentado um método que permite gerar trajetórias discretas óptimas, baseado na formulação de um problema de optimização não linear, que considera um conjunto de restrições relacionadas com as limitações físicas dos veículos, com a formação e com o espaço aéreo circundante. Adicionalmente, são incluídas restrições não lineares para evitar colisões e garantir boa visibilidade durante a troca de posições na substituição. Para solucionar o problema de optimização, é implementado um algoritmo iterativo baseado em Programação Convexa Sequencial, que permite resolver um problema de optimização convexo aproximado em cada iteração. Também é proposto um controlador para seguir as trajetórias geradas, composto por uma malha responsável pela estabilização da atitude e por outra malha de retroacção externa para o seguimento da posição.

Numa segunda fase, é implementada uma estratégia de planeamento e de controlo que permite gerar trajetórias óptimas e comandos em tempo real. Para tal, é formulado um problema de Controlo Preditivo baseado em Modelos não linear, baseado na dinâmica do sistema, numa função de custo quadrática e num conjunto de restrições, considerando novamente um conjunto de restrições para evitar colisões e garantir boa visibilidade durante a substituição. Todos os algoritmos apresentados são validados com simulações que incluem o software do piloto automático.

Palavras-chave: Drone, Voo em formação, Substituição, Optimização de trajetórias, Controlo de seguimento, Controlo preditivo baseado em modelos
Abstract

This thesis addresses the design of trajectory planning and control techniques for a team of drones, so that a vehicle in formation flight can be safely and efficiently replaced.

In a first stage, an optimal trajectory generation method is presented, where a nonlinear optimization problem is formulated taking into account constraints imposed by the physical limitations of the vehicle, the formation and the surrounding environment to generate discrete trajectories. Additionally, nonlinear constraints are included to impose collision avoidance and clear visibility during the exchange of positions in replacement. To solve the nonlinear optimization problem, an iterative algorithm based on Sequential Convex Programming is implemented, thus, solving an approximate convex problem at each iteration. Furthermore, a tracking controller is proposed to follow the generated reference trajectories, that is composed by an inner loop responsible for attitude stabilization and an outer loop related to position tracking.

In a second stage, an online planning and control strategy is implemented, that is able to generate optimal positions and control inputs in real time. To this end, a Nonlinear Model Predictive Controller is formulated, based on the dynamic model of the system, a quadratic objective function, and a set of constraints, that again include collision avoidance and clear visibility. All of the algorithms presented are validated in simulations that include the autopilot software.

**Keywords:** Drone, Formation flight, Replacement, Optimal trajectory, Tracking control, Model Predictive Control
# Contents

Acknowledgments ........................................ iii
Resumo ........................................ v
Abstract ........................................ vii
List of Tables ..................................... xi
List of Figures .................................... xiii
List of Symbols .................................... xv
Acronyms ........................................ xix

1 Introduction ........................................ 1
   1.1 Motivation ................................... 1
   1.2 Objectives .................................. 2
   1.3 Problem Statement ........................... 3
   1.4 Thesis Outline ............................... 5

2 Related Work ...................................... 7
   2.1 Optimal trajectory generation ............... 7
   2.2 Trajectory tracking control ................. 8
   2.3 Real time trajectory generation and tracking ........................................ 9

3 Background Theory ................................ 11
   3.1 Optimization Overview ..................... 11
      3.1.1 Convexity .............................. 12
      3.1.2 Optimization problems ............... 14
   3.2 Model Predictive Control .................. 14
      3.2.1 Stability ................................ 16
   3.3 Rotations and Quaternions ................. 18
      3.3.1 Coordinate systems and Euler angles ... 18
      3.3.2 Quaternions ............................ 19
      3.3.3 Comparison of methods ............... 20

4 Modelling and Control ............................. 21
   4.1 System dynamics ............................ 22
4.1.1 Controller ................................................................. 22
4.1.2 Simulation results ..................................................... 25

5 Optimal Trajectory Generation ........................................ 29
  5.1 Trajectory dynamics .................................................. 29
  5.2 Objective function .................................................... 31
  5.3 Constraints ............................................................ 32
  5.4 Sequential Convex Programming ................................. 37
  5.5 Analysis of the optimization problem ......................... 38
  5.6 Simulation test results .............................................. 39

6 Online Planning and Control .......................................... 45
  6.1 System dynamics ..................................................... 46
  6.2 Perception objectives .............................................. 47
  6.3 NMPC algorithm ..................................................... 48
    6.3.1 Objective function ........................................... 48
    6.3.2 Constraints .................................................... 49
    6.3.3 ACADO implementation ................................. 50
    6.3.4 Stability analysis .......................................... 51
  6.4 Simulation results .................................................. 52

7 SITL simulations ....................................................... 61
  7.1 Tests with pre-planned trajectories .......................... 62
  7.2 Tests with online trajectories ................................ 64

8 Conclusions .................................................................. 69
  8.1 Future Work ............................................................ 70

Bibliography .................................................................... 71

A Basic concepts of stability .......................................... 75

B Alternative planning approaches .................................. 77
  B.1 Problem with mixed-integer constraints .................. 77
    B.1.1 Trajectory dynamics ........................................... 77
    B.1.2 Optimization variable ........................................ 78
    B.1.3 Objective function ........................................... 78
    B.1.4 Constraints ...................................................... 81
    B.1.5 Analysis of the optimization problem ................ 86
    B.1.6 Problems and limitations .................................. 86
  B.2 Combined approach ................................................. 89
    B.2.1 Simulation test results ........................................ 89
List of Tables

4.1 Controller gains ................................................................. 26

5.1 Formation flight positions with 2 drones ................................. 39
5.2 Differences in acceleration, velocity and position between solutions for scenario 1....... 41
5.3 Computation time and objective function obtained in each method for scenario 1 ......... 41
5.4 Formation flight positions with 3 drones ........................................ 41
5.5 Differences in acceleration, velocity and position between results in scenario 2 .......... 43
5.6 Computation time and objective function obtained with each method for scenario 2 ....... 43

6.1 Formation flight positions with 2 drones considered in the MPC algorithm ................. 53
6.2 Scaling factor considered in the cost matrices ..................................... 53
6.3 Formation flight positions with 3 drones considered in the MPC algorithm .................. 56

7.1 Formation flight positions with 3 drones considered in the SITL simulations ............... 62
7.2 Formation flight positions with 2 drones considered in the SITL simulations ............... 65
# List of Figures

1.1 MULTIDRONE Project ................................................................. 2
1.2 REPLACE Project ................................................................. 2
1.3 Representation of a general position $j$ in formation flight obtained from the parameters $\epsilon_{xzj}, \epsilon_{yj}$ and $\epsilon_{zj}$ ......................................................... 4
1.4 Flight plan diagrams ............................................................... 4

3.1 Representation of the coordinate frames in a quadrotor ....................... 18

4.1 Block diagram in *Simulink* of the system dynamics .......................... 23
4.2 Representation of the inner and outer loops of the controller .................. 23
4.3 Block diagram in *Simulink* of the overall control system .................... 23
4.4 Position response to the tracking controller for a simple step reference trajectory ......................................................... 26
4.5 Yaw angle response to the tracking controller for a simple step reference trajectory ......................................................... 27
4.6 Position response to the tracking controller for a complex reference trajectory ......................................................... 27
4.7 Velocity response to the tracking controller for a complex reference trajectory ......................................................... 27
4.8 Yaw angle response to the tracking controller for a complex reference trajectory ......................................................... 28

5.1 Representation of the target and world frames .................................. 33
5.2 Illustration of the clear visibility constraint ....................................... 36
5.3 Trajectories generated for scenario 1 ............................................ 40
5.4 Distance between drones in scenario 1 .......................................... 40
5.5 Clear visibility constraint in scenario 1 ........................................... 40
5.6 Trajectories generated for scenario 2 ............................................. 42
5.7 Distance between drones in scenario 2 .......................................... 42
5.8 Clear visibility constraint in scenario 2 ........................................... 42

6.1 Diagram of the MPC controlled system .......................................... 51
6.2 Trajectories obtained with MPC for scenario 1 .................................. 53
6.3 Control inputs obtained with MPC for scenario 1 ................................ 54
6.4 Perception parameters obtained with MPC for scenario 1 .................... 54
6.5 Distance between drones in scenario 1 .......................................... 55
6.6 Clear visibility constraint in scenario 1 ........................................... 55
List of Symbols

Greek symbols

$\omega_B$  Angular velocity in the body frame.

$\omega_i$  Angular velocity of rotor $i$.

$\alpha_{i j}$  Angle between the target and drone $i$ w.r.t. the camera in drone $j$.

$\alpha_{cam}$  Camera's field of view angle.

$\chi$  Optimization variable.

$\epsilon_{ti}, \epsilon_{xi}, \epsilon_{yi}, \epsilon_{zi}$  Formation flight parameters for position $i$.

$\phi, \theta, \psi$  Roll, pitch and yaw Euler angles.

$\sigma$  Polynomial trajectory.

Mathematical Notation

$\text{dim}$  Dimension of array.

$\hat{x}$  $[3 \times 3]$ Skew matrix of $x$.

$\Lambda x$  $[4 \times 4]$ Skew matrix of $x$.

$\nabla f(x)$  Gradient of function $f$.

$\nabla^2 f(x)$  Hessian of function $f$.

$\odot$  Element-wise matrix multiplication.

$\times$  Cross product.

$\otimes$  Quaternion multiplication operator.

$R_q(q)$  Quaternion rotation operator.

Roman symbols

$a q_b$  Quaternion for rotation of $b$ described in frame $a$.

$a R_b$  Rotation matrix of $b$ described in frame $a$. 
$U, X, X_f$ Input, state and terminal state sets.

$\mathbf{a}$ Acceleration in the world frame.

$b_{eq}$ Linear equality constraint vector.

$b_{in}$ Linear inequality constraint vector.

$F_{des}$ Desired force vector of the controller.

$j$ Jerk in the world frame.

$p_T$ Target position in the world frame.

$p_{form}$ Formation position in the world frame.

$p$ Position in the world frame.

$U$ Control inputs sequence.

$u$ Control input.

$v$ Velocity in the world frame.

$X$ States sequence.

$x$ System state.

$Z$ Perception objectives sequence.

$z$ Perception objectives.

$B$ Body frame.

$C$ Camera frame.

$I$ Moment of inertia.

$K_f$ Terminal control law.

$L, L_f$ Stage and terminal cost function.

$Q_x$ State cost matrix.

$Q_z$ Perception cost matrix.

$R$ Input cost matrix.

$W$ World frame.

$\mu_i$ Weight of drone $i$ in quadratic function.

$\tilde{u}_1$ Mass-normalized thrust.

$A_{eq}$ Linear equality constraint matrix.
$A_{in}$  Linear inequality constraint matrix.

d_{ij}  Distance between drones $i$ and $j$.

d_{saf}  Safety distance.

e_{\omega}, e_R  Angular velocity and Rotation errors.

e_p, e_v  Position and velocity errors.

$F_i$  Individual force in rotor $i$.

$f_x, f_y$  Camera's focal lengths.

g  Gravity constant.

$h$  Discretization time.

$H, f$  Components of a quadratic function.

$H_{min}$  Minimum height.

$K$  Number of time steps.

$k_F, k_M$  Quadrotor constants.

$K_p, K_v, K_\omega, K_R$  Controller gain matrices.

$L$  Length of quadrotor arm.

$m$  Mass.

$M_i$  Individual moment in rotor $i$.

$N$  Length of MPC sequence.

$N_P(T_i)$  Number of time steps in interval $T_i$.

$p, q, r$  Angular velocity in body frame components.

$Q$  Number of drones.

$s_x, s_y$  Camera's image plane coordinates.

$T$  Interval.

$T_h$  Time horizon.

$T_s$  Sampling time.

$u_1$  Quadrotor's net body force.

$u_2, u_3, u_4$  Quadrotor's body moments in the $x$, $y$ and $z$ axis.

$u_\tau$  Quadrotor's body moments.
$V_N$  Objective function in MPC.

**Subscripts and Superscripts**

*  Conjugate.

∨  Vee map.

$B$  Body frame.

$C$  Camera frame.

$des$  Desired.

$f$  Terminal.

$max$  Maximum value allowed.

$min$  Minimum value allowed.

$ref$  Reference.

$T$  Transpose.

$W$  World frame.

$x, y, z$  Cartesian components.
### Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D</td>
<td>Two-Dimensional</td>
</tr>
<tr>
<td>3D</td>
<td>Three-Dimensional</td>
</tr>
<tr>
<td>DMC</td>
<td>Dynamic Matrix Control</td>
</tr>
<tr>
<td>EKF</td>
<td>Extended Kalman Filter</td>
</tr>
<tr>
<td>GPC</td>
<td>Generalized Predictive Control</td>
</tr>
<tr>
<td>LQR</td>
<td>Linear-Quadratic Regulator</td>
</tr>
<tr>
<td>MAV</td>
<td>Micro Aerial Vehicle</td>
</tr>
<tr>
<td>MILP</td>
<td>Mixed-Integer Linear Problem</td>
</tr>
<tr>
<td>MIQP</td>
<td>Mixed-Integer Quadratic Problem</td>
</tr>
<tr>
<td>MPC</td>
<td>Model Predictive Control</td>
</tr>
<tr>
<td>NMPC</td>
<td>Nonlinear Model Predictive Control</td>
</tr>
<tr>
<td>RHC</td>
<td>Receding Horizon Control</td>
</tr>
<tr>
<td>ROS</td>
<td>Robot Operating System</td>
</tr>
<tr>
<td>RUAV</td>
<td>Rotary-Wing Unmanned Aerial Vehicles</td>
</tr>
<tr>
<td>SCP</td>
<td>Sequential Convex Programming</td>
</tr>
<tr>
<td>SITL</td>
<td>Software In The Loop</td>
</tr>
<tr>
<td>UAV</td>
<td>Unmanned Aerial Vehicle</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Over the past years, there have been significant technological developments in automotive, sensor, and computer industries, which empowered the development of rotary-wing unmanned aerial vehicles (RUAVs), or drones, as highly versatile tools for industrial and consumer applications. Multiple studies have been successful in defining strategies for the motion control of each of these vehicles in free flight. Nonetheless, new challenges may occur when multiple drones are required to share the same airspace or to work together in order to complete a certain task. In these cases, it is necessary to not only control each drone individually, but also to study how they interact in the overall task.

1.1 Motivation

Even though drones are versatile and can be used in different applications, there is still a major limitation that has to be considered every time these vehicles are used: their lack of autonomy. Depending on their size and battery capacity, they have a limited flight time available. This limitation is specially relevant when considering activities involving multiple drones.

This is one of the topics of discussion in the project MULTIDRONE [1] (Figure 1.1), which aims to develop an innovative intelligent multidrone team platform for media production to cover outdoor events that are typically distributed over large expanses. They focus on increasing the drone’s decisional autonomy and improving multiple drone robustness and safety mechanisms. When using multiple drones to automatically track points of interest or moving targets, they may be required to cover large distances over extended periods of time, exceeding the flight time of a single drone. In order to face this problem, replacement strategies can be implemented, in which vehicles running out of battery are replaced by others while guaranteeing collision avoidance and without interfering with the overall task.

Other project involving activities with multiple drones is the project REPLACE [2] (Figure 1.2). In this case, the goal is to develop a delivery service in urban environments using drones. The main idea is to use multiple drones executing strategical manoeuvres and cooperating with each other so that the delivery service is fast, efficient and reliable. When the distance to reach the final destination goes beyond a single drone’s range, additional drones may be used so that when one begins to run out of
battery, another will replace its position in carrying the package to be delivered.

The motivation for this thesis arises from the importance of devising strategies for the replacement of energy depleted drones during formation flight, in such a way as to minimize disruption of the formation and tracking, while ensuring the safety of every drone as well as the surrounding structures and people.

### 1.2 Objectives

This thesis focuses on the design of trajectory planning and control techniques for a small group of vehicles, so that a vehicle in formation flight can be safely and efficiently replaced. Different methods for trajectory planning and control will be implemented and compared, in order to solve an optimization problem with constraints imposed by the physical limitations of the vehicle, the formation, and the surrounding environment.

There are two main goals for this thesis:

- Generate optimal trajectories that are able to solve the problem of drone replacement;
- Implement a controller capable of tracking such trajectories.

In a first stage, we will tackle both goals separately, i.e. develop planning strategies to generate optimal trajectories by solving linear and/or nonlinear constrained optimization problems, and then implement a controller to track such trajectories.

Afterwards, in a second stage, the idea is to evolve from offboard trajectory generation approaches to online planning. In this case, planning and control will be implemented simultaneously and in real-time, relying on concepts of Model Predictive Control (MPC). In this way, the vehicles will be able to follow optimal trajectories and perform an efficient replacement process without interfering with the overall task.
1.3 Problem Statement

Since the replacement manoeuvres are central to the problem definition in this thesis, further details on the type of manoeuvres considered throughout the thesis are presented in this section.

Consider a scenario where one or more drones are monitoring a target that is following a path. When one of the drones begins to run out of battery, it should be replaced by another one. This means that a new drone must take off and join the formation flight. After a while, they will exchange positions, while guaranteeing that the replacement does not interfere with the overall task. Only then will the drone with low battery be allowed to exit the task and land in an intermediate position.

To solve the problem described, it is necessary to define a flight plan divided into 12 intervals, for which two approaches are considered:

- **Scenario with 2 drones** - consider that, in the beginning, drone 1 is following the target by staying behind it within a time interval $\epsilon_{t1}$ and separated by the distances $\epsilon_{x1}$, $\epsilon_{y1}$ and $\epsilon_{z1}$ (position 1 in formation flight), as represented in Figure 1.3. The replacement process begins when drone 1 receives a first low-battery warning. At that moment, drone 2 takes off and joins formation flight by staying behind the target with different $\epsilon_{t2}$, $\epsilon_{x2}$, $\epsilon_{y2}$ and $\epsilon_{z2}$ values (position 2 in formation flight). The next step is for the vehicles to exchange positions without interfering with the overall task, during which drone 2 goes to position 1 in formation flight and drone 1 to position 2. Afterwards, drone 1 is able to leave the formation flight and land in another intermediate position where it stays until the end of the task. Meanwhile, drone 2 keeps following the target, until the task is completed. In the end, it lands in its final position. The diagram in Figure 1.4a represents the flight plan described.

- **Scenario with Q drones** - consider that in the beginning $Q - 1$ drones are following the target in different formation flight positions defined by $\epsilon_{tj}$, $\epsilon_{xj}$, $\epsilon_{yj}$ and $\epsilon_{zj}$, with $j = 1, ..., Q - 1$, as in Figure 1.3. When drone $i$, with $i = 1, ..., Q - 1$, that is in position $i$ in formation flight, receives a first low-battery warning, another drone (drone $q$) takes off and joins the formation flight in the last position of the formation flight (position $Q$). The next step is for drones $i$ and $q$ to exchange positions, i.e. drone $i$ goes to position $Q$ and drones $q$ to position $i$ without being seen by the other drones. Afterwards, drone $i$ is able to leave the formation flight and land in its final position, where it stays until the end of the task. Meanwhile, the other $Q - 1$ drones keep following the target, until the task is complete, after which they go to their final positions. The diagram in Figure 1.4b represents the flight plan described.

This thesis aims to solve an optimization problem that is able to generate trajectories capable of executing the manoeuvres described while considering the UAV's physical limitations and ensuring collision avoidance between vehicles, which will, then, be used as references in a tracking controller. Afterwards, in a second stage, the problem will be reformulated as a Model Predictive Control algorithm that handles planning and control simultaneously and in real-time.
Chapter 1. Introduction

Figure 1.3: Representation of a general position \( j \) in formation flight obtained from the parameters \( \epsilon_{xj}, \epsilon_{yj} \) and \( \epsilon_{zj} \).

Figure 1.4: Flight plan diagrams

The contribution of this work will be to implement and adapt a planning method and, later an MPC strategy, to solve the problem of drone replacement in formation flight for a multidrone situation, by including additional constraints that account for the formation and for the vehicles’ field of view during flight.
1.4 Thesis Outline

The remainder of this thesis is structured as follows:

- Chapter 2 gives a brief explanation of other studies that have been conducted in the same research field, when it comes to planning techniques with optimal trajectory generation, tracking controllers and real-time planning and control.

- Chapter 3 describes important concepts and properties that are related with optimization, Model Predictive Control and rotations and quaternions.

- In Chapter 4, a mathematical model is presented for a quadrotor. A complementary tracking controller is also proposed in order to follow reference trajectories, together with some simulation results, the respective interpretation and a brief discussion.

- Chapter 5 describes a strategy to generate optimal trajectories to follow a target in formation flight. The problem is formulated as a non-convex optimization problem and solved with a Sequential Convex Programming (SCP) algorithm. In the end, some simulation results are shown and a performance analysis is made in order to test the stability and overall performance of the algorithm.

- Chapter 6 is related to the second stage of this thesis: the online trajectory planning and control, where the problem is reformulated as a Nonlinear Model Predictive Control (NMPC). Some simulation results are also shown.

- Chapter 7 presents Software In The Loop (SITL) simulation results to validate the planning method described in Chapter 5 and the MPC algorithm from Chapter 6.

- Finally, Chapter 8 summarises the goals achieved in this dissertation and discusses what can still be done as future work.
Chapter 2

Related Work

The development of Unmanned aerial vehicles (UAVs) has increased significantly over the past years, which led to plenty of studies in the field of trajectory generation for collision avoidance, specially when it comes to formation flight in UAVs [3, 4]. Multiple strategies and algorithms have been developed, which will be briefly discussed in this chapter.

Trajectory planning can be achieved by solving an optimization problem, in which it is necessary to define an objective function and a set of constraints [5]. The complexity of the optimization problem depends on the way it is formulated. Convex optimization problems are usually associated with reliable and efficient algorithms [6]. Model Predictive Control is a special case of an optimization problem, in which smaller optimization problems are solved in real-time for a short time horizon [7, 8]. Further details are presented in Chapter 3.

2.1 Optimal trajectory generation

Several studies have focused on optimal trajectory planning, that differ mostly on the type of trajectories generated (discrete, polynomial, etc) and on the types of constraints used (nonlinear, with mixed integers, linear, etc).

Some studies have successfully formulated nonlinear optimization problems. It is the case of [9], where the authors present an algorithm based on SCP that generates discretized trajectories in 3D for multiple vehicles. The collision avoidance constraints are formulated as nonlinear constraints, that will then be linearised with a local optimization method. A similiar case occurs in [10], where a nonlinear optimization problem is solved using an Interior Point algorithm and transforming the nonlinear equations into disjunction equations.

Other studies are able to use mixed-integer linear constraints instead, resulting in mixed-integer optimization problems. The method in [11] generates optimal polynomial trajectories that smoothly transition through waypoints at given times. The problem is then formulated as a Mixed-Integer Quadratic Problem (MIQP), where the collision avoidance constraints are written as linear mixed-integer inequalities. Another algorithm that uses mixed-integer constraints is presented in [12], where an optimization problem
is formulated for aircrafts that fly at constant altitude as Mixed-Integer Linear Problem (MILP), resulting in 2D trajectories.

In [13], an algorithm that combines both mixed-integer and nonlinear constraints is implemented. At first the problem is relaxed (without binary variables). The solution is obtained by applying a block coordinate descent, i.e. decoupling the problem into blocks and solving one at a time, and by using sequential programming techniques.

Other studies have used search-based algorithms to generate trajectories, with reachability algorithms [14], Rapid-exploring Random Trees [15], Generic Algorithms [16] and iterative heuristic searches [17]. However, these methods can be impractical when dealing with a high dimensional search space. The authors in [18] are able to avoid this issue by converting an optimal control problem into graph search-based planning method that explores a map using a set of short-duration motion primitives. The proposed approach is able to generate safe, dynamically feasibility trajectories efficiently by exploiting the explicit solution of a Linear Quadratic Minimum Time problem.

Note that even though most of the referenced studies may already account for multidrone trajectory generation and include collision avoidance constraints, they do not explicitly solve the problem of drone replacement in formation flight, nor do they consider restrictions related to the camera’s visibility.

2.2 Trajectory tracking control

As for modelling and trajectory tracking control for UAVs, different types of controllers have been developed. The work presented in [19] proposes a geometric tracking controller directly on the special Euclidean group, in which four inputs are used to track position and heading. This controller is proven to be stable and have a good performance. Then, in [20], a similar tracking controller is proposed for a differentially flat quadrotor, that considers the errors in position, velocity, rotation and angular velocity at each time-step.

Other studies implement MPC techniques to track reference trajectories, which can be used as a transition point to online planning and control. It is the case of [15], in which the control horizon varies according to the curvature of the trajectory. It also accounts for obstacles that may appear after the reference trajectory has been generated. In [21], two types of MPC are implemented: a linear one, in which the nonlinear dynamic equations of the quadrotor are linearised around a desired operating point; and a nonlinear one that uses state-dependent coefficient factorization to obtain pseudo-linear system matrices. In the end, the authors conclude that the nonlinear MPC has a better performance. Another example is presented in [22] that implements a parameterised MPC. The solution is obtained by finding feasible trajectories that are the closest to a reference trajectory. An iterative learning scheme based on a Kalman Filter is also used to remove repeatable disturbances. Finally, the authors of [23] compare two methods for trajectory tracking in Micro Aerial Vehicles (MAVs): a linear MPC and a nonlinear MPC, where they conclude that the NMPC has an overall better performance. External disturbances are also estimated with an augmented state Extended Kalman Filter (EKF).
2.3 Real time trajectory generation and tracking

Several studies have developed algorithms that deal with real time planning and control and they usually apply MPC techniques. It is the case of [24], that presents a two-layer MPC system that implements real-time planning and control for a formation of UAVs. It is composed by a linear MPC that handles the vehicles stabilization and reference tracking; and a hybrid decentralized MPC that is used to generate obstacle and collision-free paths. The optimization problem is formulated as a Mixed-Integer Quadratic Problem. Another example is [25] that proposes a robust MPC for multivehicle guidance. The authors present two versions of the algorithm: a centralized version, that uses constraint tightening that changes according to the external disturbances, an invariant set to ensure safety and a cost-to-go function (distance to the target) to generate obstacle-free trajectories for a short time horizon; and a distributed version, in which each vehicle optimizes its own trajectory by solving a smaller subproblem and then exchanging data with the other surrounding vehicles. There is, however, a common missing feature in both of these methods: they do not consider the drone’s field of view.

Then, there are studies that deal with the vehicle’s perception, but only consider situations with one vehicle. It is the case of [26], which addresses the problem of controlling the motion of a quadrotor using an onboard camera by proposing a minimum time trajectory planning method that guarantees visibility of the image features while the vehicle undertakes aggressive motions. The problem is formulated with differential flatness and B-splines parameters and solved with a SCP algorithm. However, the system is not suited for real-time control and the algorithm is only validated with simulations. Other example is [27], that proposes a vision-based navigation method, allowing for a quadrotor to reach a goal pose in the environment while constantly facing a target. After computing a goal pose, planning and control are executed in two consecutive stages: generation of an optimal trajectory by solving a nonlinear optimization problem, that takes into account dynamic and perception constraints; and a Receding Horizon NMPC controller to follow such trajectory. Additionally, the authors of [28] suggest a hybrid visual servoing technique applicable to differential flat and underactuated systems, in which an optimal control problem is formulated with an image-based cost in order to generate a trajectory to navigate to a goal position previously computed. There is, however, a problem with the methods described: they do not take into account the motion of the vehicle.

The authors in [29] are able to solve this issue by proposing a method that couples both objectives (motion and perception) and generates trajectories in real-time with applications in aerial videography, taking into consideration visibility under occlusion and collision avoidance. The problem is formulated as a Receding Horizon optimal Control problem with nonlinear constraints. Other method is presented in [30], where a perception-aware MPC algorithm is implemented for a quadrotor. In this case, the optimization problem is formulated with action and perception objectives and takes into account the system dynamics and its saturations. Additionally, it optimizes perception objectives by maximizing the visibility of a point of interest in the image and minimizing the velocity of its projection.

The contribution of this thesis at this stage will be to find an MPC strategy for a multidrone situation that implements formation flight for target following and considers the vehicles’ field of view during flight.
Chapter 3

Background Theory

In this chapter, we present some basic concepts and properties that are necessary to consider throughout this dissertation.

3.1 Optimization Overview

This section gives an overview on optimization and is based on seminal works on the subject [5, 6]. A simple formulation of an optimization problem implies the definition of the following components:

- An optimization variable \( \chi = [x_1, ..., x_n]^T \in \mathbb{R}^n \);
- An objective function \( f_0(\chi) : \mathbb{R}^n \Rightarrow \mathbb{R} \), which we want to minimize or maximize;
- A set of constraints \( c_i(\chi) : \mathbb{R}^n \Rightarrow \mathbb{R} \), which can be linear or nonlinear functions that represent equalities or inequalities and depend on \( \chi \).

Then, the optimization problem can be written as:

\[
\begin{aligned}
&\text{minimize } f_0(\chi) \\
&\text{subject to } c_i(\chi) = 0, \ i \in \mathcal{E}, \\
&\quad c_i(\chi) \leq 0, \ i \in \mathcal{I}
\end{aligned}
\]  

(3.1)

where \( \mathcal{E} \) represents the set of equality constraints and \( \mathcal{I} \) the set of inequality constraints.

The optimization variable \( \chi \) can take on any type of number. In discrete optimization problems, \( \chi \) can have integer variables (integer programming) or even a mixture of integer and continuous variables (mixed-integer programming). In these cases, the values of the objective functions and constraints may change significantly when moving between feasible points. By contrast, in continuous problems, since \( \chi \) can take on real numbers, this results in simpler optimization problems as it is possible to deduce the values of the objective function and constraints around a specific point in \( \chi \).

Generally, we want to find the global optimal solution of the optimization problem, i.e. the point with the lowest objective function value among all feasible points. However, this can be difficult to determine,
Chapter 3. Background Theory

specially when it comes to nonlinear problems or even to problems with a large number of variables. As an alternative, many algorithms determine local optimal solutions around nearby points. For convex programming problems, this solution is valid, since local solutions are also global solutions. However, for general nonlinear problems, this is no longer true. Therefore, the goal should be to formulate the optimization problem with convex objective functions and constraints whenever possible.

3.1.1 Convexity

A set \( C \in \mathbb{R}^n \) is an **affine set** if the line between any two points in \( C \) lies inside \( C \). Therefore, for any two points \( x \in C \) and \( y \in C \),

\[
\alpha x + (1 - \alpha)y \in S, \text{ for all } \alpha \in \mathbb{R}.
\]

This condition can be generalized for \( K \) points \( x_1, ..., x_k \). In this case, if \( C \) is an affine set, \( x_1, ..., x_k \in C \) and \( \alpha_1 + ... + \alpha_k = 1 \), then

\[
\alpha_1 x_1 + ... + \alpha_k x_k \in C,
\]

where \( \alpha_1 x_1 + ... + \alpha_k x_k \) is an affine combination.

A **convex set** can be defined in a similar way as the affine sets, differing only in \( \alpha \) \((0 \leq \alpha \leq 1)\). A set \( S \in \mathbb{R}^n \) is a convex set, if the straight line segment connecting any points in \( S \) lies inside \( S \). This can be formally described for a set of points \( x_1, ..., x_k \), where if \( S \) is a convex set, \( x_1, ..., x_k \in S \), \( \alpha_1 + ... + \alpha_k = 1 \) and \( \alpha_i \geq 0, i = 1, ..., k \), then

\[
\alpha_1 x_1 + ... + \alpha_k x_k \in S,
\]

where \( \alpha_1 x_1 + ... + \alpha_k x_k \) is a convex combination. Note that if a set is affine, it is also convex, but the opposite is not necessarily true.

There are several examples of sets that have already been proven to be convex and include the following:

- **Hyperplane** - \( \{x | a^T x = b\} \), also affine;
- **Halfspace** - \( \{x | a^T x \leq b\} \);
- **Norm ball with center** \( x_c \) and radius \( r \) - \( \{x | \|x - x_c\| \leq r\} \);
- **Norm cone** - \( \{(x, t) | \|x\| \leq t\} \);
- **Ellipsoid** - \( \{x | (x - x_c)^T P^{-1} (x - x_c) \leq 1\} \), where \( P \in S^n_{++} \) is a symmetric positive definite matrix;
- **Polyhedra** - a combination of linear inequalities, \( Ax \leq b \), and equalities, \( Cx = d \), also equivalent to a combination of hyperplanes and halfspaces;
- **Positive semidefinite cone** - if \( S^n_+ \) is a set of positive semidefinite symmetric \( n \times n \) matrices, such that \( X \in S^n_+ \iff z^T X z \geq 0 \), for all \( z \), then \( S^n_+ \) is a convex cone.
In order to establish the convexity of a set $S$, one can apply the definition directly or use certain operations that are known to preserve convexity, and include the following:

- Intersection - the intersection of any convex set is also convex;
- Affine function - considering that $f$ is an affine function, which means $f(x) = Ax + b$, with $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and $x \in \mathbb{R}^n$, the image of a convex set under $f$ is convex, which is equivalent to

$$S \in \mathbb{R}^n \text{ convex} \rightarrow f(S) = \{f(x)|x \in S\} \text{ convex.}$$

This is also valid for the inverse image $f^{-1}(S)$ of a convex set under $f$, thus,

$$S \in \mathbb{R}^m \text{ convex} \rightarrow f^{-1}(S) = \{x \in \mathbb{R}^n|f(x) \in S\} \text{ convex.}$$

Some examples of such functions are scaling, translation, projection and solution sets of linear matrix inequalities.

- Perspective function - $P : \mathbb{R}^{n+1} \Rightarrow \mathbb{R}^n$ is a perspective function if $P(x,t) = \frac{x}{t}$, with $t > 0$. The images and inverse images of convex sets under perspective functions are also convex.

- Linear-fractional function - $f : \mathbb{R}^n \Rightarrow \mathbb{R}^m$ is a linear-fractional function if $f(x,t) = \frac{Ax + b}{c^T x + d}$, with $c^T x + d > 0$. The images and inverse images of convex sets under linear-fractional functions are also convex.

A function $f : \mathbb{R}^n \Rightarrow \mathbb{R}$ is a **convex function** if $S$ is a convex set and if for any two points $x \in S$ and $y \in S$,

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha)f(y), \text{ for } 0 \leq \alpha \leq 1. \quad (3.2)$$

$f$ can also be identified as strictly convex if

$$f(\alpha x + (1-\alpha)y) < \alpha f(x) + (1-\alpha)f(y), \text{ for } 0 < \alpha < 1. \quad (3.3)$$

Additionally, $f$ is concave if $-f$ is convex. There are two important properties associated with convex functions:

- **First-order condition** - suppose that $f$ is differentiable with a convex domain and it has the following gradient

$$\nabla f(x) = \left(\frac{df(x)}{dx_1}, ..., \frac{df(x)}{dx_n}\right),$$

then, $f$ is convex if and only if

$$f(y) \geq f(x) + \nabla f(x)^T(y-x), \text{ for all } x, y \in \text{ dom } f. \quad (3.4)$$

- **Second-order condition** - Suppose that $f$ is twice differentiable with a convex domain and the
Chapter 3. Background Theory

Hessian $\nabla^2 f(x) \in S^n$ is defined as

$$\nabla^2 f(x)_{ij} = \frac{d^2 f(x)}{dx_i dx_j}$$

then, $f$ is convex if and only if

$$\nabla^2 f(x) \succeq 0, \text{ for all } x \in \text{dom } f.$$ (3.5)

3.1.2 Optimization problems

The formulation of an optimization problem determines the problem’s complexity and the type of algorithm that should be implemented. The goal should be to formulate the optimization problem as a convex problem, as it guarantees that if we are able to find a local optimal solution, then this solution is also unique and global. Furthermore, convex problems are usually less complex and simpler to prove convergence. For an optimization problem to be identified as convex, the objective function should be convex, the equality constraints linear and the inequalities defined by convex sets.

Optimization algorithms are iterative and usually take into account the values of the objective function $f$, the constraint functions $c_i$, and possibly the first and second derivatives of these functions to compute a solution. They should consider the following properties:

- Robustness - to perform well on a wide variety of problems;
- Efficiency - to not require excessive computer time or storage;
- Accuracy - to be able to identify a solution with precision.

3.2 Model Predictive Control

In this section, we will introduce some basic concepts of Model Predictive Control, which are based on [7, 8]. MPC is the method of using a dynamic model to predict a system’s behaviour and to optimize its performance by computing an optimal control input at the current time step. It uses past record measurements to determine the most likely initial state of the system. It may also be known as Receding Horizon Control (RHC), Dynamic Matrix Control (DMC) or Generalized Predictive Control (GPC).

Thus, MPC is a strategy capable of implementing planning and control simultaneously, in which, at each time step, an optimization problem is numerically solved for a prediction horizon and is subject to a set of constraints, resulting in a sequence of inputs and a sequence of state predictions. The system dynamics will then use the first value computed in the sequence of inputs and determine the next system state.
To formulate an MPC algorithm, one must define:

1. **A dynamic model** - which is composed by a set of differential states and control inputs and, for a generic representation, can be written as

\[
\frac{dx}{dt} = f(x, u, t) \\
y = h(x, u, t) \\
x(t_0) = x_0,
\] (3.6)

where \(x \in \mathbb{R}^n\) is the state, \(u \in \mathbb{R}^m\) is the input, \(y \in \mathbb{R}^p\) is the output and \(t \in \mathbb{R}\) is the current time of the system. In some cases, it is possible to formulate the dynamic model in (3.6) as a linear state space model. In this way, we can represent a general linear state space time-varying model as

\[
\frac{dx}{dt} = A(t)x + B(t)u \\
y = X(t)x + D(t)u \\
x(t_0) = x_0,
\] (3.7)

where \(A(t) \in \mathbb{R}^{n \times n}\) is the state transition matrix, \(B(t) \in \mathbb{R}^{n \times m}\) is the input matrix, \(C(t) \in \mathbb{R}^{p \times n}\) is the output matrix and \(D(t) \in \mathbb{R}^{p \times m}\) is a matrix to define the coupling between \(u\) and \(y\) (usually \(D = 0\)). If the system is time-invariant, then the matrices \(A, B, C\) and \(D\) are constant. The advantage of using linear models is their ease of solution and analysis.

2. **A set of constraints** - that can be used to limit the inputs \(u \in U\), states \(x \in X\) or even to impose conditions defined by a function \(g_i(x, u, t)\) (path constraints). The constraints related to the input values are usually *hard constraints* that represent the system physical bounds. In contrast, the state and path constraints are usually desirables and are characterised as soft constraints.

3. **An objective function** - that we want to minimize and is usually the error between \(x\) and \(u\) and their respective reference.

**Example of an MPC Regulator**

Let’s consider an example of a deterministic time-invariant linear system, for which we want to design a controller to force the state to the origin, that will be implemented using a Linear Quadratic Regulator (LQR). The system model can be formulated in discrete time as

\[
x_{k+1} = Ax_k + Bu_k \\
y_k = Cx_k,
\] (3.8)

where \(x_k\) and \(y_k\) are the state and output at time step \(k\) and \(x_{k+1}\) is the state at time step \(k + 1\).

We can predict how the state evolves given any set of inputs. Considering \(N\) time steps into the future, we can have an input sequence given by \(U\), such that \(U = [u_0, u_1, ..., u_{N-1}]\). This sequence can be subject to some constraints, distinguishing MPC from the standard LQR.
Chapter 3. Background Theory

The objective function can then be defined as the sum of weighted squares, such that

\[
V(x_0, u) = \frac{1}{2} \sum_{k=0}^{N-1} \left[ x_k^T Q_k x_k + u_k^T R_k u_k \right] + \frac{1}{2} x_N^T Q_N x_N, \tag{3.9}
\]

subject to \( x_{k+1} = A x_k + B u_k \), and where \( Q_k \) and \( R_k \) are the stage cost matrices related to \( x \) and \( u \) for the first \( N - 1 \) time steps respectively, and \( Q_N \) is the terminal stage cost matrix related to \( x \) for the last time step of the MPC’s time horizon \( T_h \). Note that the initial state is available from the measurements and that the remainder of the state trajectory, with \( x_k, k = 1, ..., N \), is determined by the model and the input sequence. In this way, the optimal control problem \( P_N \) is defined as

\[
P_N(x) : \min_U V(x_0, U). \tag{3.10}
\]

In order for a solution in this problem to exist and be unique, \( Q_k, Q_N \) and \( R_k \) should be real and symmetric, and in particular, \( Q_k \) and \( Q_N \) should be positive semidefinite and \( R_k \) positive definite. These matrices can be tuned for each situation in order to obtain an optimal solution.

3.2.1 Stability

To study the stability of an MPC algorithm, one has to take into account the Lyapunov Stability Theory properties summarized in Appendix A. In this section, we present a brief overview on how to establish stability for an MPC algorithm, which was based on Chapter 2 from [7] and Chapter 4 from [8].

Let \( V \) be a Lyapunov function, \( X \) a positive invariant set, \( \alpha_1 \) and \( \alpha_2 \) be two functions \( \alpha : \mathbb{R} \Rightarrow \mathbb{R}_{\geq 0} \) that are continuous, strictly increasing, zero to zero and unbounded, and \( \gamma \) be a positive definite function, i.e. \( \gamma : [0, \infty) \Rightarrow [0, \infty) \) is continuous, \( \gamma(t) > 0 \), for all \( t > 0 \) and \( \lim_{t \to \infty} \gamma(t) = \infty \). Then, the origin is asymptotically stable with a region of attraction \( X \) for a system \( x_{k+1} = f(x_k) \) if the following conditions are satisfied

\[
\alpha_1(|x|) \leq V(x) \leq \alpha_2(|x|) \tag{3.11}
\]

\[
V(f(x)) \leq V(x) - \gamma(|x|)
\]

Thus, the goal is to find a Lyapunov function \( V \) that satisfies these properties for an MPC system \( x_{k+1} = f(x_k, K_f(x_k)) \), where \( K_f \) is the terminal control law, which is usually the objective function of the MPC. Note that, with this, we are considering an optimization problem for a finite horizon control problem, yet stability must hold over an infinite horizon. A solution for this problem can be to assign a terminal cost in the objective function as to account for the impact of events that may lie beyond the finite horizon. We can provide sufficient conditions for stability by presenting the following:

1. A terminal set \( X_f \) that is invariant under the terminal control law;

2. A feasible terminal control law \( K_f \) that holds the terminal constraint set;

3. A terminal state cost \( \mathcal{L}_f \) for a finite horizon optimization problem.
Consider the following MPC problem:

\[
P_N(x) : \quad V_N^{OPT} := \min V_N(x_0, u) \\
\text{s.t.} \quad x_{k+1} = f(x_k, u_k), \quad \text{for } k = 0, ..., N - 1 \\
\quad u_k \in \mathbb{U}, \quad \text{for } k = 0, ..., N - 1 \\
\quad x_k \in \mathbb{X}, \quad \text{for } k = 0, ..., N \\
\quad x_N \in \mathbb{X}_f \subset \mathbb{X}
\]

with \( V_N(x_0, u) = \mathcal{L}_f(x_N) + \sum_{k=0}^{N-1} \mathcal{L}(x_k, u_k) \), for which we want to prove asymptotic or exponential stability of the origin, with the following assumptions:

a) The stage cost \( \mathcal{L}(x_k, u_k) \) satisfies \( \mathcal{L}(0, 0) = 0 \) and \( \mathcal{L}(x_k, u_k) \geq \gamma \) for all \( x \in \mathbb{S}_N, u \in \mathbb{U} \), where \( \mathbb{S}_N \subset \mathbb{X} \) is the set of initial feasible states;

b) The terminal cost \( \mathcal{L}_f \) satisfies \( \mathcal{L}_f(0) = 0, \mathcal{L}_f(x_k) \geq 0 \) for all \( x_k \in \mathbb{X}_f \) and there exists a control law \( K_f : \mathbb{X}_f \rightarrow \mathbb{U} \) such that \( \mathcal{L}_f(f(x_k, K_f(x_k))) - \mathcal{L}_f(x_k) \leq -\mathcal{L}(x_k, K_f(x_k)) \) for all \( x_k \in \mathbb{X}_f \);

c) The set \( \mathbb{X}_f \) is positive invariant under \( K_f(x_k) \), which is equivalent to \( \mathcal{L}_f(f(x_k, K_f(x_k))) \in \mathbb{X}_f \) for all \( x_k \in \mathbb{X}_f \);

d) The terminal control \( K_f(x_k) \) satisfies \( K_f(x_k) \in \mathbb{U} \) for all \( x_k \in \mathbb{X}_f \);

e) The sets \( \mathbb{X}_f \) and \( \mathbb{U} \) contain the origin.

Taking these assumptions into account, we can then formulate the theorem for stability of Receding Horizon Control, which is proven in [8] (pages 98-100).

**Theorem:** consider a closed loop system for a system \( x_{k+1} = f(x_k, u_k) \) controlled by an MPC algorithm with state and control constraints, then

1. The set \( \mathbb{S}_N \) of feasible positive initial states is positive invariant for the closed loop system;

2. The origin is globally attractive in \( \mathbb{S}_N \);

3. If \( 0 \in \text{int} \mathbb{S}_N \) and \( V_N^{OPT} \) is continuous on some neighbourhood of the origin, then the origin is asymptotically stable in \( \mathbb{S}_N \);

4. If \( 0 \in \text{int} \mathbb{X}_f, \mathbb{S}_N \) is compact, \( \gamma(t) \geq a t^\nu, \mathcal{L}_f(x) \leq b \|x\|^\sigma \) for all \( x \in \mathbb{X}_f \), where \( a > 0, \nu > 0 \) and \( \sigma > 0 \) are real constants and \( V_N^{OPT} \) is continuous in \( \mathbb{S}_N \), then the origin is exponentially stable in \( \mathbb{S}_N \).
Chapter 3. Background Theory

3.3 Rotations and Quaternions

In this section, we will give a brief overview on coordinate systems, Euler angles and Rotation matrices in 3D for a body (such as a UAV). We will also introduce the concept of quaternions and explain how to convert between Euler angles, quaternions, and Rotation matrices. The topics presented in this Section are based on the seminal works in [20, 30, 31].

3.3.1 Coordinate systems and Euler angles

Throughout this thesis, we consider three different coordinate systems, which are represented in Figure 3.1: the world frame $W$ with a fixed origin on the ground; the body frame $B$, with the origin in the center of mass of the UAV; and the camera frame $C$ with the origin fixed in the center of the lens of the camera. The rotation from world to body frame depends on the rotation motion around the three main axis $x$, $y$ and $z$, which results in the following Euler angles: roll $\phi$, pitch $\theta$ and yaw $\psi$. The Rotation matrix for each angle is described in (3.13).

The final Rotation matrix is the multiplication of these 3 matrices and, therefore, depends on the order of rotation. For instance, for an axis sequence of $Z \rightarrow X \rightarrow Y$, the resulting Rotation matrix from body to world axis $^W R_B$ is given by

$$^W R_B = R_z R_x R_y = \begin{bmatrix}
\cos(\psi) \cos(\theta) - \sin(\psi) \sin(\theta) & \sin(\psi) \cos(\phi) & \cos(\psi) \sin(\theta) + \sin(\psi) \sin(\phi) \cos(\theta) \\
\sin(\psi) \cos(\theta) + \cos(\psi) \sin(\phi) \sin(\theta) & \cos(\psi) \cos(\phi) & \sin(\psi) \sin(\theta) - \cos(\psi) \sin(\phi) \cos(\theta) \\
-\cos(\phi) \sin(\theta) & \sin(\phi) \cos(\theta) & \cos(\phi) \cos(\theta)
\end{bmatrix}. \quad (3.14)$$

Figure 3.1: Representation of the coordinate frames in a quadroto

An important property of these matrices is that they are orthogonal square matrices $[3 \times 3]$ with real entries, which means that $R^T = R^{-1}$, and that their determinant is 1 ($\det R = 1$).
3.3.2 Quaternions

Quaternions are another way to describe rotations and correspond to 4-tuple of real numbers defined in the 3-Sphere $S^3 = \{ q \in \mathbb{R}^4 : \| q \| = 1 \}$ and are given by

$$ q = q_w + q_x i + q_y j + q_z k, \quad (3.15) $$

where $q_w$, $q_x$, $q_y$ and $q_z$ are real scalar numbers and $i$, $j$ and $k$ are complex numbers. They arise from the Euler’s rotation theorem that states that any rotation or sequence of rotations of a rigid body or coordinate system about a fixed point is equivalent to a single rotation given by an angle $\alpha$ about a fixed axis, represented by a unit vector $u = u_x i + u_y j + u_z k$, that runs through a fixed point. The quaternion is obtained using an extension of Euler’s formula, such as

$$ q = e^{\frac{\alpha}{2} (u_x i + u_y j + u_z k)} = \cos \frac{\alpha}{2} + (u_x i + u_y j + u_z k) \sin \frac{\alpha}{2}. \quad (3.16) $$

The rotation of a vector $v$ with quaternions is achieved with the following operation

$$ v' = R_q(q)v = q^* v q, \quad (3.17) $$

where $R_q(q)$ represents the quaternion rotation operator and $q^* = q_w - q_x i - q_y j - q_z k$ is the conjugate of $q$. The previous expression can then be expanded in order to obtain an equation of the type $v' = Qv$, where $Q$ is defined as

$$ Q = \begin{bmatrix}
1 - 2q_y^2 - 2q_z^2 & 2(q_x q_y + q_w q_z) & 2(q_x q_z - q_w q_y) \\
2(q_x q_y - q_w q_z) & 1 - 2q_x^2 - 2q_z^2 & 2(q_y q_z + q_w q_x) \\
2(q_x q_z + q_w q_y) & 2(q_y q_z - q_w q_x) & 1 - 2q_x^2 - 2q_y^2
\end{bmatrix}. \quad (3.18) $$

Thus, with this representation, we can conclude that $Q$ is in fact the Rotation matrix known before as $R$ that transforms the world frame coordinates into body frame, but, in this case, it is defined by quaternions instead of Euler angles. The inverse operation is also possible, i.e. obtaining the quaternion components from the Rotation matrix. Considering that the entrances of $R$ are defined by $r_{ij}$, where $i$ is the line and $j$ is the column of $R$, then the components of the quaternion can be written as

$$ q_w^2 = \frac{1}{4}(1 + r_{11} + r_{22} + r_{33}), $$
$$ q_x^2 = \frac{1}{4}(1 + r_{11} - r_{22} - r_{33}), $$
$$ q_y^2 = \frac{1}{4}(1 - r_{11} + r_{22} - r_{33}), $$
$$ q_z^2 = \frac{1}{4}(1 - r_{11} - r_{22} + r_{33}). \quad (3.19) $$

To obtain the scalars, one could simply take the square root of each equation. However, this introduces an ambiguity, as it is not possible to determine the sign of each component. In order to attempt to solve
Chapter 3. Background Theory

this issue, we can consider the 6 additional equations:

\begin{align*}
q_0 q_1 &= \frac{1}{4}(r_{32} - r_{23}) & q_0 q_2 &= \frac{1}{4}(r_{13} - r_{31}) & q_0 q_3 &= \frac{1}{4}(r_{21} - r_{12}) \\
q_1 q_2 &= \frac{1}{4}(r_{12} + r_{21}) & q_1 q_3 &= \frac{1}{4}(r_{13} + r_{31}) & q_2 q_3 &= \frac{1}{4}(r_{23} + r_{32})
\end{align*}

(3.20)

Therefore, the procedure to convert a Rotation matrix in-to a quaternion should be, at first, to find the largest \( q_i^2 \) from the equations in (3.19), to avoid division by small numbers, and take its square root, and then, to use three of the equations in (3.20) to obtain the other three scalars. With this method, there is still the issue of determining the sign of the first component computed, which is a common ambiguity when using quaternions. This means that one Rotation matrix can result into two quaternions.

3.3.3 Comparison of methods

When describing the state of the system, it is necessary to include some components that describe the rotation of the body, which can have three possible representations:

- **Euler Angles** \((\phi, \theta \text{ and } \psi)\) - these may cause problems due to the discontinuities of the angles \((-\frac{\pi}{2} < \phi < \frac{\pi}{2}, -\frac{\pi}{2} < \theta < \frac{\pi}{2} \text{ and } -\pi < \psi < \pi)\), but have the advantage of requiring less storage memory;

- **Quaternions** \((q_w, q_x, q_y \text{ and } q_z)\) - these may not have issues with singularities, but have the ambiguity that two different quaternions result in the same Rotation matrix. They also require more storage memory than Euler angles and can lead to more complex mathematical manipulations. Nevertheless, they are still faster and more compact than Rotation matrices;

- **Rotation matrices** - these may be simpler to manipulate and do not have issues with singularities or ambiguities. However, storing a full matrix in a state space may be inefficient.

In the end, the method chosen should take into consideration which characteristics, such as timing constraints, memory management, mathematical manipulation, etc., are more relevant for the problem at hand.
Chapter 4

Modelling and Control

As mentioned before, one of the goals of this dissertation is to implement a controller that is able to follow certain desired trajectories. To do so, we propose a dynamic model of a quadrotor and a tracking controller, which are based on the research work presented in [20, 32].

Two coordinate systems, world frame $W$ and body frame $B$, are considered, as already represented in Figure 3.1. The rotation from world to body frame is denoted by the rotation matrix $W R_B$ in (3.14) and the angular velocity of the body, denoted by $\omega_B = [p, q, r]^T$, is related to the derivatives of $\phi$, $\theta$ and $\psi$, such as

$$\omega_B = \begin{bmatrix} \cos(\theta) & 0 & -\cos(\phi)\sin(\theta) \\ 0 & 1 & \sin(\theta) \\ \sin(\theta) & 0 & \cos(\phi)\cos(\theta) \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}. \quad (4.1)$$

Each rotor has an angular speed $\omega_i$ and produces a force $F_i$ and moment $M_i$, according to

$$F_i = k_F \omega_i^2, \quad M_i = k_M \omega_i^2, \quad (4.2)$$

where $k_F$ and $k_M$ are system constants that account for the relation between the speed of the motors and the generated lift and drag forces on each propeller. These can be used to compute the four controller inputs $u$: $u_1$, that represents the sum of the forces of the 4 rotors, and $u_r = [u_2, u_3, u_4]^T$, that denotes the body moments in each body axis. If we consider that $u'$ represents the control inputs for a body frame $B'$ where the $x$ and $y$ axis are aligned with the quadrotors arms, then the relation between $u'$ and the rotor speeds $\omega_i$ can be written as

$$u' = \begin{bmatrix} k_F & k_F & k_F & k_F \\ 0 & k_F L & 0 & -k_F L \\ -k_F L & 0 & k_F L & 0 \\ k_M & -k_M & k_M & -k_M \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix}, \quad (4.3)$$

where $L$ is the distance between the axis of the rotors and the center of the quadrotor. The control inputs...
\( u \) are then given by \( u = B^{R_{B'}} u' \), where \( B^{R_{B'}} \) represents the rotation from \( B' \) to \( B \).

Finally, the system is described by a state \( x \) at a given time step, which is composed of the vehicle’s position \( p = [p_x, p_y, p_z]^T \) and linear velocity \( v = [v_x, v_y, v_z]^T \) of the body frame described in the world frame, as well as its angular velocity \( \omega_B \) and Rotation matrix \( W^{R_B} \), as shown in (4.4). Note that we are using the Rotation matrix directly as a state component, as it will make the mathematical computations simpler and ensure that there are no singularities in the system.

\[
\begin{align*}
\dot{x} &= [p, v, \omega_B, W^{R_B}]^T \tag{4.4}
\end{align*}
\]

### 4.1 System dynamics

Given the components of the system state and the control inputs described before, the motion of the quadrotor can be described by the following equations:

\[
\begin{align*}
\dot{p} &= v, \\
\dot{v} &= -g z_W + \frac{u_1}{m} z_B, \\
\dot{\omega}_B &= \mathcal{I}^{-1} \left( -\hat{\omega}_B \mathcal{I} \omega_B + u_\tau \right), \\
W^{R_B} \dot{R}_B &= W^{R_B} \hat{\omega}_B,
\end{align*}
\] (4.5)

where \( g = 9.81 \text{m/s}^2 \) is the gravity constant, \( z_W = [0, 0, 1]^T \) is the \( z \) axis in the world frame, \( z_B = W^{R_B} z_W \) is the \( z \) axis in the body frame, \( \mathcal{I} \) is the moment of inertia and \( \hat{\omega}_B \) is the skew matrix obtained from \( \omega_B \), defined as

\[
\hat{\omega}_B = \begin{bmatrix}
0 & -r & q \\
 r & 0 & -p \\
- q & p & 0
\end{bmatrix}.
\] (4.6)

With these equations, we are able to obtain an updated state of the system from the control inputs and the previous Rotation matrix and angular velocities. The dynamics described were implemented in Simulink and are represented in the block diagram in Figure 4.1.

#### 4.1.1 Controller

The controller design described in this Section was based on [20, 32], where a trajectory tracking controller is implemented for a differentially flat system such as a quadrotor. This controller is composed by two main loops represented in Figure 4.2: an inner loop for attitude stabilization that regulates rotation and angular velocity components, and an outer loop for position tracking that controls the position and velocity. At each time step, new inputs \( u \) are computed from the previous state \( x \), the reference trajectory position \( p_{\text{ref}} \) and the derivatives \( v_{\text{ref}}, a_{\text{ref}} \) and \( \dot{a}_{\text{ref}} \), as well as the reference yaw \( \psi_{\text{ref}} \) and its derivative \( \dot{\psi}_{\text{ref}} \) as represented in the diagram in Figure 4.3.
4.1. System dynamics

Figure 4.1: Block diagram in *Simulink* of the system dynamics

Position controller

Figure 4.2: Representation of the inner and outer loops of the controller

Figure 4.3: Block diagram in *Simulink* of the overall control system

Generally a controller regulates the behaviour of a system by forcing its input variables to zero. Therefore, in order to follow certain trajectories, we must provide the errors between the references and
the actual values of the variables as inputs of the controller, so that when these go to zero, we know the system is following the references. To do so, we can compute the errors of position and velocity as

\[ e_p = p - p_{\text{ref}}, \]  
\[ e_v = v - v_{\text{ref}}. \]  

From these, we obtain the desired force vector as

\[ F_{\text{des}} = -K_p e_p - K_v e_v + m g z_W + m a_{\text{ref}}, \]  

where \( K_p \) and \( K_v \) are positive definite matrices known as the position and velocity controller gains. The controller input \( u_1 \) is then defined as the projection of \( F_{\text{des}} \) onto \( z_B \) and can be written as

\[ u_1 = F_{\text{des}} \cdot z_B. \]  

The other three inputs are related to the rotation errors, that depend on the desired Rotation matrix and angular velocity. To obtain the desired Rotation matrix \( ^W R_{\text{Bdes}} \), we need to compute the components \( x_{\text{Bdes}}, y_{\text{Bdes}} \) and \( z_{\text{Bdes}} \) individually. At first, we consider that \( z_{\text{Bdes}} \) corresponds to the direction of \( F_{\text{des}} \), which means that

\[ z_{\text{Bdes}} = \frac{F_{\text{des}}}{\|F_{\text{des}}\|}, \]  

assuming \( \|F_{\text{des}}\| \neq 0 \). Then, considering that the intermediate coordinate frame \( A \) is the transformation of the world frame after a yaw rotation, the component \( x_{\text{Ades}} \) is given by

\[ x_{\text{Ades}} = [\cos(\psi_{\text{ref}}), \sin(\psi_{\text{ref}}), 0]^T. \]  

From this, we can compute \( y_{\text{Bdes}} \) and \( x_{\text{Bdes}} \), assuming that \( \|z_{\text{Bdes}} \times x_{\text{Ades}}\| \neq 0 \), as

\[ y_{\text{Bdes}} = \frac{z_{\text{Bdes}} \times x_{\text{Ades}}}{\|z_{\text{Bdes}} \times x_{\text{Ades}}\|}, \]  
\[ x_{\text{Bdes}} = y_{\text{Bdes}} \times z_{\text{Bdes}}. \]  

Thus, \( ^W R_{\text{Bdes}} = [x_{\text{Bdes}}, y_{\text{Bdes}}, z_{\text{Bdes}}] \) and the rotation error is given by

\[ e_R = \frac{1}{2}(^W R_{\text{Bdes}}^T ^W R_B - ^W R_B^T ^W R_{\text{Bdes}})^\vee, \]  

where \( ^\vee \) represents the vee map, which is the inverse operation of the skew matrix.

As for the desired angular velocity \( \omega_{\text{Bdes}} \), it is necessary to manipulate the expression of the derivative of the acceleration \( \dot{a}_{\text{ref}} \), which can be written as

\[ m \ddot{a}_{\text{ref}} = \dot{u}_1 z_{\text{Bdes}} + ^W R_{\text{Bdes}} \omega_{\text{Bdes}} \times u_1 z_{\text{Bdes}}. \]
4.1. System dynamics

Similarly to \( u_1 \), \( \dot{u}_1 \) is the projection \( \dot{u}_1 = m\dot{a}_{ref} \cdot z_{Bdes} \). If we consider that \( h_{\omega des} \) is given by

\[
h_{\omega des} = W R_{Bdes} \omega_{Bdes} \times u_1 z_{Bdes} = \frac{m}{u_1} (\dot{a}_{ref} - (z_{Bdes} \cdot \dot{a}_{ref})z_{Bdes}),
\]

(4.17)

then \( p_{des} \) and \( q_{des} \) can be expressed by the following equations:

\[
p_{des} = -h_{\omega des} \cdot y_{Bdes}, \tag{4.18}
\]

\[
q_{des} = h_{\omega des} \cdot x_{Bdes}. \tag{4.19}
\]

To find \( r_{des} \), it is necessary to solve the system of equations with 3 unknowns \((\dot{\phi}_{des}, \dot{\theta}_{des} \text{ and } r_{des})\) in (4.20). Afterwards, we finally obtain \( \omega_{Bdes} = [p_{des}, q_{des}, r_{des}]^T \) and compute the error for the angular velocity as in (4.21).

\[
W R_{Bdes} \omega_{Bdes} = \begin{bmatrix} x_{Cdes} & y_{Bdes} & z_W \end{bmatrix} \begin{bmatrix} \dot{\phi}_{des} \\ \dot{\theta}_{des} \\ \dot{\psi}_{des} \end{bmatrix} \Leftrightarrow W R_{Bdes} \begin{bmatrix} p_{des} \\ q_{des} \\ r_{des} \end{bmatrix} = \begin{bmatrix} x_{Cdes} & y_{Bdes} & z_W \end{bmatrix} \begin{bmatrix} \dot{\phi}_{des} \\ \dot{\theta}_{des} \\ \dot{\psi}_{des} \end{bmatrix} \tag{4.20}
\]

\[
e_\omega = \omega_B - \omega_{Bdes}. \tag{4.21}
\]

In the end, we can determine the other control inputs as

\[
u_T = -K_R e_R - K_\omega e_\omega, \tag{4.22}
\]

where \( K_R \) and \( K_\omega \) are diagonal gain matrices.

Note that even though no proof of stabilibility is given in this thesis, we can still affirm that this controller is stable, as stability and convergence proofs are already provided for a similar controller in [19]. Further stability analysis fall out of scope of this thesis.

4.1.2 Simulation results

The dynamic model and controller were implemented in Simulink, where several tests were made with different types of trajectories generated in different optimization problems. At first, it was necessary to tune the controller gains in order to maximize the controller’s performance. To do so, we simulated a simple trajectory reference, where the position in each coordinate is given by a step reference that at \( t = 2 \) seconds goes from \( 0 \) to \( 5 \) meters, the yaw angle is also a step reference that goes from \( 0 \) to \( \frac{\pi}{3} \) rad at the same time and the other references are set to zero. Through empirical trials, we choose the gains by considering that:

- \( K_{pos} \) - leads to a faster response but with possible overshoot;
• $K_{vel}$ - attenuates the overshoot but the system becomes slower;

• $K_{R}$ - attenuates the oscillations;

• $K_{\omega}$ - leads to more oscillations.

We adjusted, at first, the gains for the inner loop ($K_{R}$ and $K_{\omega}$) to obtain a stable response without oscillations, and afterwards we tuned the gains for the outer loop ($K_{p}$ and $K_{v}$) to ensure that the controller is able to follow the references without any overshoot. Note that in the method for optimal trajectory generation to be described in Chapter 5, we only compute optimal values of position and do not explicitly consider how the yaw angle should behave. Therefore, in this controller we prioritized the response in position over the yaw angle. In the end, we obtained the gains shown in Table 4.1.

**Table 4.1: Controller gains**

<table>
<thead>
<tr>
<th></th>
<th>$K_{pos}$</th>
<th>$K_{vel}$</th>
<th>$K_{R}$</th>
<th>$K_{\omega}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\begin{bmatrix} 15 &amp; 0 &amp; 0 \ 0 &amp; 15 &amp; 0 \ 0 &amp; 0 &amp; 15 \end{bmatrix}$</td>
<td>$\begin{bmatrix} 10 &amp; 0 &amp; 0 \ 0 &amp; 10 &amp; 0 \ 0 &amp; 0 &amp; 10 \end{bmatrix}$</td>
<td>$\begin{bmatrix} 20 &amp; 0 &amp; 0 \ 0 &amp; 20 &amp; 0 \ 0 &amp; 0 &amp; 20 \end{bmatrix}$</td>
<td>$\begin{bmatrix} 2 &amp; 0 &amp; 0 \ 0 &amp; 2 &amp; 0 \ 0 &amp; 0 &amp; 2 \end{bmatrix}$</td>
</tr>
</tbody>
</table>

The position response to the controlled system with such gains for the simple step trajectory is shown in Figure 4.4. From the plots we can see that the system is not as fast as one might expect, as the settling time is around 2 seconds, but it does not have any overshoot or oscillations and is able to follow the reference. By readjusting the gains it would be possible to have a faster response, but at the expense of increasing the overshoot and oscillations.

![Position in x over time](image1.png) ![Position in y over time](image2.png) ![Position in z over time](image3.png)

**Figure 4.4: Position response to the tracking controller for a simple step reference trajectory**

Additionally, we also plotted the system response for the yaw angle, as represented in Figure 4.5. Here we can see that the system is able to follow the reference and that the response in faster than before, with a settling time of around 1 second, but with some overshoot.

With the tuned gains, we can now test the controller with a more complex trajectory, that was generated with the method in Chapter 5. This trajectory lasts 34 seconds and consists of references of position, velocity, acceleration and jerk, resulting in stair-case functions with a fixed time step ($h = 0.25$ seconds). Note that this method does not account for the yaw angle, which we assumed in this case to be zero. The position response is displayed in Figure 4.6, where we can see that the controller is able
4.1. System dynamics

Figure 4.5: Yaw angle response to the tracking controller for a simple step reference trajectory

(a) Position in x over time  
(b) Position in y over time  
(c) Position in z over time

Figure 4.6: Position response to the tracking controller for a complex reference trajectory

to follow the reference position as desired. Since the discretization time step is smaller than the settling time obtained in the previous test ($T_{set} = 2$ seconds), the controller cannot fully outline each step of the reference, but instead goes to the final position (of each step) through a smooth curve. These errors of discretization are even more noticeable when plotting the velocity response as shown in Figure 4.7. In this case, the reference in velocities has some spikes and steep curves that may not be feasible in reality. The controller is then able avoid them by smoothing the response, resulting in continuous feasible velocity profiles.

In addition, we can plot the system response for the yaw angle (Figure 4.8). Note that even though the response appears to be oscillatory, the scale of the oscillations is small (around $1 \times 10^{-4}$ rad), reaching maximum values of $6.3 \times 10^{-4}$ rad during takeoff at $t = 3$ sec and during the replacement process at $t = 16$ sec. One possible explanation for this is related to numerical errors.

Therefore, we can conclude that the controller is efficient and performs well for the desired task (trajectory tracking). The system dynamic response is able to follow a reference through a smooth and
safe trajectory, attenuating possible discontinuities in the reference.

Figure 4.8: Yaw angle response to the tracking controller for a complex reference trajectory
Chapter 5

Optimal Trajectory Generation

This chapter attempts to solve the problem stated in Section 1.3 with a pre-planned and centralized strategy. An optimization problem capable of generating collision free discrete trajectories for multiple vehicles is formulated using non-trivial constraints necessary for drone replacement, resulting in a non-linear optimization problem. The method presented in this section is based on the work presented in [9], with the necessary changes to the problem in terms of constraints and problem structure.

The algorithm was implemented using Matlab's optimization toolbox and Cplex [33], and was solved with the Cplex function cplexmiqp when there were only linear constraints and the Matlab function fmincon function when there were additional nonlinear constraints.

Alternative strategies were also investigated and are presented in Appendix B, where instead of using nonlinear constraints, we formulated the optimization problem with mixed-integer linear constraints, resulting in Mixed-Integer Quadratic Problems. In the end, we concluded that these were not as efficient as the strategy presented in this chapter, as they were not always able to follow a typical flight plan as described in Section 1.3.

5.1 Trajectory dynamics

As explained in Section 3.1, to solve an optimization problem, one must define several components, such as an optimization variable, an objective function, and linear and nonlinear constraints. To do so, it is necessary to define the motion dynamics for a generic vehicle.

Considering that, for $Q$ vehicles and $K$ time steps, $p_{ij}[k]$, $v_{ij}[k]$ and $a_{ij}[k]$ are the position, velocity and acceleration, respectively, of a vehicle $i$, with $i = 1, ..., Q$, in the direction axis $j \in \{x, y, z\}$ at the time step $k = 1, ..., K$ and $h$ is the discretization time. If we simplify the system dynamics in (4.5) so that $\dot{p} = v$ and $\ddot{v} = a$, and assume that the acceleration is constant in between discretization time steps, i.e. $a(t) = a[k], \forall t \in [t_k, t_k + h]$, with $k = 0, ..., K - 1$, then we may have the following discrete system
dynamics:

\[ v_{ij}[k + 1] = v_{ij}[k] + h a_{ij}[k], \]
\[ p_{ij}[k + 1] = p_{ij}[k] + \frac{h^2}{2} a_{ij}[k]. \]  

The optimization variable chosen in this method, \( \chi \in \mathbb{R}^{3QK} \), is given by the vehicles’ accelerations at each time step and can be written in the form

\[ \chi = [a_{1x}[1], \ldots, a_{1x}[K], a_{1y}[1], \ldots, a_{1y}[K], a_{1z}[1], \ldots, a_{1z}[K], \ldots, a_{Qx}[1], \ldots, a_{Qx}[K], a_{Qy}[1], \ldots, a_{Qy}[K], a_{Qz}[1], \ldots, a_{Qz}[K]]^T. \]

In order to obtain expressions for the position and velocity that only depend on the optimization variable, the equations in (5.1) can be modified according to

\[ v_{ij}[k] = v_{ij}[1] + h(a_{ij}[1] + a_{ij}[2] + \cdots + a_{ij}[k - 1]), \]
\[ p_{ij}[k] = p_{ij}[1] + h(k - 1)v_{ij}[1] + \frac{h^2}{2}((2k - 3)a_{ij}[1] + (2k - 5)a_{ij}[2] + \cdots a_{ij}[k - 1]). \]

Additionally, these trajectories can be expressed in a matrix form. If we consider a vehicle \( i \), with \( i = 1, \ldots, Q \), and in the direction axis \( j \in \{x, y, z\} \), the trajectory dynamics for the velocity and position can be written as

\[
\begin{bmatrix}
v_{ij}[1] \\
v_{ij}[2] \\
v_{ij}[3] \\
\vdots \\
v_{ij}[K]
\end{bmatrix} = 
\begin{bmatrix}
0 & 0 & \cdots & 0 \\
1 & 0 & \cdots & 0 \\
a_{ij}[1] & a_{ij}[2] & \cdots & a_{ij}[K] \\
\vdots & \vdots & \ddots & \vdots \\
a_{ij}[1] & a_{ij}[2] & \cdots & a_{ij}[K]
\end{bmatrix}
\begin{bmatrix}
v_{ij}[1] \\
v_{ij}[2] \\
v_{ij}[3] \\
\vdots \\
v_{ij}[K]
\end{bmatrix} +
\begin{bmatrix}
1 & 1 & \cdots & 1 \\
0 & 0 & \cdots & 0 \\
0 & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 0
\end{bmatrix}
\begin{bmatrix}
v_{ij}[1] \\
v_{ij}[2] \\
v_{ij}[3] \\
\vdots \\
v_{ij}[K]
\end{bmatrix} \iff v_{ij} = H_1 a_{ij} + v_{ij0}. \tag{5.4}
\]

\[
\begin{bmatrix}
p_{ij}[1] \\
p_{ij}[2] \\
p_{ij}[3] \\
\vdots \\
p_{ij}[K]
\end{bmatrix} = 
\begin{bmatrix}
0 & 0 & \cdots & 0 \\
1 & 0 & \cdots & 0 \\
\frac{h^2}{2} & 3 & 1 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
2K - 3 & 2(K - 1) - 3 & \cdots & 1 & 0
\end{bmatrix}
\begin{bmatrix}
a_{ij}[1] \\
a_{ij}[2] \\
a_{ij}[3] \\
\vdots \\
a_{ij}[K]
\end{bmatrix} +
\begin{bmatrix}
1 & 1 & \cdots & 1 \\
0 & 0 & \cdots & 0 \\
0 & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 0
\end{bmatrix}
\begin{bmatrix}
v_{ij}[1] \\
v_{ij}[2] \\
v_{ij}[3] \\
\vdots \\
v_{ij}[K]
\end{bmatrix} \iff p_{ij} = \frac{h^2}{2} H_2 a_{ij} + h H_3 \odot v_{ij0} + p_{ij0}. \tag{5.5}
\]

where \( a_{ij} \in \mathbb{R}^K \), \( v_{ij} \in \mathbb{R}^K \), \( p_{ij} \in \mathbb{R}^K \), \( H_1 \in \mathbb{R}^{K \times K} \), \( H_2 \in \mathbb{R}^{K \times K} \), \( H_3 \in \mathbb{R}^K \), \( v_{ij0} \in \mathbb{R}^K \) and \( p_{ij0} \in \mathbb{R}^K \), and \( \odot \) represents an element-wise matrix multiplication. From these matrices, it is possible to characterize the full trajectory dynamics in matrix form by repeating the equations for all direction axis \( (x, y \text{ and } z) \) and
all \( Q \) vehicles, such as

\[
\begin{bmatrix}
v_{1x} \\
v_{1y} \\
v_{1z} \\
\vdots \\
v_{Qx} \\
v_{Qy} \\
v_{Qz}
\end{bmatrix}
= 
\begin{bmatrix}
H_1 & 0 & 0 & \ldots & 0 & 0 & 0 \\
0 & H_1 & 0 & \ldots & 0 & 0 & 0 \\
0 & 0 & H_1 & \ldots & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & \ldots & H_1 & 0 & 0 \\
0 & 0 & 0 & \ldots & 0 & H_1 & 0 \\
0 & 0 & 0 & \ldots & 0 & 0 & H_1
\end{bmatrix}
\begin{bmatrix}
a_{1x} \\
a_{1y} \\
a_{1z} \\
\vdots \\
a_{Qx} \\
a_{Qy} \\
a_{Qz}
\end{bmatrix}
+ 
\begin{bmatrix}
v_{01x} \\
v_{01y} \\
v_{01z} \\
\vdots \\
v_{0Qx} \\
v_{0Qy} \\
v_{0Qz}
\end{bmatrix}
\implies \mathbf{v} = H_{11} \mathbf{\chi} + \mathbf{v}_0. \quad (5.6)
\]

\[
\begin{bmatrix}
p_{1x} \\
p_{1y} \\
p_{1z} \\
\vdots \\
p_{Qx} \\
p_{Qy} \\
p_{Qz}
\end{bmatrix}
= \frac{h^2}{2}
\begin{bmatrix}
H_2 & 0 & 0 & \ldots & 0 & 0 & 0 \\
0 & H_2 & 0 & \ldots & 0 & 0 & 0 \\
0 & 0 & H_2 & \ldots & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & \ldots & H_2 & 0 & 0 \\
0 & 0 & 0 & \ldots & 0 & H_2 & 0 \\
0 & 0 & 0 & \ldots & 0 & 0 & H_2
\end{bmatrix}
\begin{bmatrix}
a_{1x} \\
a_{1y} \\
a_{1z} \\
\vdots \\
a_{Qx} \\
a_{Qy} \\
a_{Qz}
\end{bmatrix}
+ \begin{bmatrix}
H_3 \\
H_3 \\
H_3 \\
\vdots \\
H_3 \\
H_3 \\
H_3
\end{bmatrix}
\circ \begin{bmatrix}
v_{01x} \\
v_{01y} \\
v_{01z} \\
\vdots \\
v_{0Qx} \\
v_{0Qy} \\
v_{0Qz}
\end{bmatrix}
\implies \mathbf{p} = \frac{h^2}{2} H_{22} \mathbf{\chi} + h H_{33} \circ \mathbf{v}_0 + \mathbf{p}_0. \quad (5.7)
\]

where \( \mathbf{v} \in \mathbb{R}^{3QK} \), \( \mathbf{p} \in \mathbb{R}^{3QK} \), \( H_{11} \in \mathbb{R}^{3QK \times 3QK} \), \( H_{22} \in \mathbb{R}^{3QK \times 3QK} \), \( H_{33} \in \mathbb{R}^{3QK} \), \( \mathbf{v}_0 \in \mathbb{R}^{3QK} \) and \( \mathbf{p}_0 \in \mathbb{R}^{3QK} \).

### 5.2 Objective function

The objective function is defined as the sum of the total mass-normalized thrust at each time step. Considering that \( m \mathbf{a} = -mgz_\mathbf{W} + \mathbf{F} \iff \mathbf{F}/m = \mathbf{a} + gz_\mathbf{W} \), where \( \mathbf{F} \) is the total thrust, then the objective function can be formulated as

\[
f_0 = \sum_{i=1}^{Q} \sum_{k=1}^{K} \|a_{i}[k] + g z_\mathbf{W}\|^2, \quad (5.8)
\]

where \( a_{i}[k] = [a_{ix}[k], a_{iy}[k], a_{iz}[k]] \) is the acceleration of a vehicle \( i \), with \( i = 1, \ldots, Q \), at the time step \( k \) in \( [1, \ldots, K] \), \( g = 9.81 m/s^2 \) is the gravity constant and \( z_\mathbf{W} = [0, 0, 1]^T \) is the \( z \) axis in the world frame. To obtain an objective function in a quadratic form, the expression (5.8) can be modified according to

\[
f_0(\mathbf{\chi}) = \mathbf{\chi}^T P \mathbf{\chi} + q^T \mathbf{\chi} + r, \quad (5.9)
\]

with
Chapter 5. Optimal Trajectory Generation

\[ P = I_{3QK} \in \mathbb{R}^{3QK \times 3QK}, \quad q = \begin{bmatrix} 0_{K,1} \\ 0_{K,1} \\ 2g \cdot 1_{K,1} \\ \vdots \\ 0_{K,1} \\ 0_{K,1} \\ 2g \cdot 1_{K,1} \end{bmatrix} \in \mathbb{R}^{3QK}, \quad r = g^2 \in \mathbb{R}. \]

Note that \( I_{3QK} \) is an identity matrix with dimension \([3QK \times 3QK]\), \( 0_{K,1} \) is a \([K \times 1]\) column-vector of zeros and \( 1_{K,1} \) is a \([K \times 1]\) column-vector of ones. When solving the optimization problem directly with nonlinear constraints, it might be useful to provide the gradient of the objective function as an additional input, which, in this case, is given by

\[ \nabla f_0 = \chi^T (P + P^T) + q^T. \quad (5.10) \]

5.3 Constraints

Three different types of constraints are considered in the formulation of the optimization problem, which include linear equalities, linear inequalities and nonlinear inequalities, as described in the following pages.

The linear equality constraints represent equations of the form

\[ A_{eq} \chi = b_{eq} \quad (5.11) \]

and include the following:

1. Initial and final states - to define the initial and final positions, velocities and accelerations of each vehicle. Suppose that \( k_I = 1, ..., K_I \) are the time indexes for which the initial state of the vehicles remains the same and \( k_F = K_F, ..., K \) are the time indexes for which their final state remains the same. Then, for a vehicle \( i \), with \( i = 1, ..., Q \), and in the direction axis \( j \in \{x, y, z\} \), the constraints are:

\[ a_{ij}[k_i] = a_{ij0}, \quad (5.12) \quad a_{ij}[k_f] = a_{ijK}, \quad (5.15) \]

\[ v_{ij}[k_i] = v_{ij0}, \quad (5.13) \quad v_{ij}[k_f] = v_{ijK}, \quad (5.16) \]

\[ p_{ij}[k_i] = p_{ij0}, \quad (5.14) \quad p_{ij}[k_f] = p_{ijK}. \quad (5.17) \]

Note that the initial position and velocity in the first time step of the trajectory \((k_I = 1)\) are already included in the trajectory dynamics in (5.6) and (5.7) with the terms \( p_0 \) and \( v_0 \). Therefore, we do not need to add the constraints (5.13) and (5.14) in this time step.
5.3. Constraints

The number of constraints depends on the number of time steps and the number of vehicles. If we consider a general scenario with \( Q \) drones as described in Figure 1.4b, the initial state constraints (5.12), (5.13) and (5.14) should be imposed during the interval \( T_1 \) for the first \( Q - 1 \) drones and during \( T_1 \) and \( T_2 \) for the last drone (drone \( q \)). As for the final state constraints, (5.15), (5.16) and (5.17) should be applied during \( T_{10}, T_{11} \) and \( T_{12} \) for the drone that was replaced (drone \( i \)) and during \( T_{11} \) and \( T_{12} \) for the other \( Q - 1 \) drones. Suppose that \( N_P(T_j) \) is the number of time steps inside an interval \( T_j \). Then, the number of constraints is

\[
3(Q - 1) \left[ 3N_P(T_1) - 2 \right] + 3 \left[ 2 \sum_{j=1}^{12} N_P(T_j) - 2 \right] + 9 \sum_{j=10}^{12} N_P(T_j) + 9(Q - 1) \sum_{j=11}^{12} N_P(T_j).
\]

2. Formation flight for target following - as explained in Section 1.3, the formation flight is achieved by assigning specific parameters (\( \epsilon_{xj}, \epsilon_{yj}, \epsilon_{zj} \), and \( \epsilon_{zj} \), with \( j = 1, \ldots, Q \)) to different positions as represented in Figure 1.3, so that the vehicles are distanced in time and space in relation to the target’s position and orientation.

Considering an additional coordinate system: the target frame, \( T \), as represented in Figure 5.1, the Rotation matrix denoted by \( W \) \( R_T[k] = [x_T, y_T, z_T] \), that describes the orientation of the target in the world frame in a time step \( k = 1, \ldots, K \), can be determined as:

\[
\begin{align*}
\mathbf{x}_T &= \frac{\mathbf{p}_T[k] - \mathbf{p}_T[k - 1]}{||\mathbf{p}_T[k] - \mathbf{p}_T[k - 1]||}, \\
\alpha &= \arctan \frac{x_{Ty}}{x_{Tx}}, \\
\beta &= \arcsin x_{Tz}, \\
\mathbf{y}_T &= [-\sin(\alpha)\cos(\beta), \cos(\alpha)\cos(\beta), -\sin(\beta)]^T, \\
\mathbf{z}_T &= [\sin(\alpha)\sin(\beta), -\cos(\alpha)\sin(\beta), -\cos(\beta)]^T,
\end{align*}
\]

(5.18)

where \( \mathbf{p}_T \) is the position of the target and \( k \) and \( k - 1 \) are consecutive time steps in \( [1, \ldots, K] \).
general position \( j \) in formation flight described in the world frame is given by

\[
P_{\text{form}}[k] = P_T[k - \frac{\epsilon_{ij}}{h}] + W R_T[k - \frac{\epsilon_{ij}}{h}] \epsilon_{pj}.
\] (5.19)

where \( \epsilon_{pj} = [\epsilon_{xj}, \epsilon_{yj}, \epsilon_{zj}]^T \) is a vector with the formation positions and \( h \) is the discretization step.

To assure that a vehicle \( i \), with \( i = 1, ..., Q \), follows the target in position \( j \) in formation flight, the following constraint must be implemented:

\[
P_i[k] = P_{\text{form}}[k].
\] (5.20)

The number of constraints also depends on the number of time steps and on the number of vehicles. For the general scenario with \( Q \) drones described in Figure 1.4b, drone \( i \) should be in \( p_{\text{form}} \) during the intervals \( T_2, T_4 \) and \( T_4 \) and in \( p_{\text{form}} \) during \( T_8 \); then drone \( q \) should be in \( p_{\text{form}} \) during the intervals \( T_4 \) and in \( p_{\text{form}} \) during \( T_8, T_9 \) and \( T_{10} \); and finally the other \( Q - 2 \) drones should be their respective positions in formation flight during \( T_2 - T_{10} \). Thus, the number of constraints is

\[
3 \sum_{j=2}^{4} N_P(T_j) + 3N_P(T_k) + 3N_P(T_k) + 3 \sum_{j=8}^{10} N_P(T_j) + (Q - 2)3 \sum_{j=2}^{10} N_P(T_j).
\]

Then, the linear inequality constraints are defined as

\[
A_{in} \chi \leq b_{in},
\] (5.21)

where \( \leq \) represents an element-wise inequality, and include the following:

1. **Dynamics limitation** - to define the maximum and minimum values allowed for the position, velocity, acceleration and jerk. For a vehicle \( i \), with \( i = 1, ..., Q \), and in any time step \( k \) in \([1, ..., K]\), we have the following constraints:

\[
P_{\min} \leq P_i[k] \leq P_{\max},
\] (5.22)

\[
V_{\min} \leq V_i[k] \leq V_{\max},
\] (5.23)

\[
a_{\min} \leq a_i[k] \leq a_{\max},
\] (5.24)

\[
J_{\min} \leq J_i[k] \leq J_{\max},
\] (5.25)

where \( J \) represents the jerk, which measures the variation of acceleration and can be computed as

\[
J[k] = \frac{a[k] - a[k-1]}{h}.
\] (5.26)

For each expression in (5.22), (5.23) and (5.24) there are \( 2 \times 3Q \) inequalities and for the jerk in (5.25) there are \( 2 \times 3Q(K-1) \). Note that the factor 2 is related to having a minimum and a maximum limit for each variable, and that the factor of \( K - 1 \) is because jerk depends on a previous value of
5.3. Constraints

the acceleration. Thus, the number of constraints of this type is

\[ 18QK + 6Q(K - 1). \]

2. Minimum height during flight - to guarantee that the vehicles are always above a minimum height \((H_{\text{min}})\) from the ground as a safety measure. For a vehicle \(i\), with \(i = 1, ..., Q\), and in a time step \(k\), this constraint can be written as

\[ p_{iz}[k] \geq H_{\text{min}}. \]  \hspace{1cm} (5.27)

For a general scenario with \(Q\) drones, this constraint should be imposed during the intervals \(T_2 - T_8\) for drone \(i\), during \(T_4 - T_{10}\) for drone \(q\) and during \(T_2 - T_{10}\) for the other \(Q - 2\) drones. Therefore, the number of constraints of this type is

\[ \sum_{j=2}^{8} N_P(T_j) + \sum_{j=4}^{10} N_P(T_j) + (Q - 2) \sum_{j=2}^{10} N_P(T_j). \]

Finally, the nonlinear constraints are defined as nonlinear functions \((c)\) that depend on the optimization variable \(\chi\), such that

\[ c(\chi) \leq 0 \]  \hspace{1cm} (5.28)

The following constraints were considered.

1. Collision Avoidance - assures that the vehicles do not collide with each other during flight by imposing that the distance between them is larger than a certain safety distance. For any two drones \(i\) and \(j\), with \(i = 1, ..., Q, j = 1, ..., Q\) and \(i \neq j\), this constraint can be written as

\[ \|p_i[k] - p_j[k]\| \geq d_{\text{saf}}, \]  \hspace{1cm} (5.29)

where \(d_{\text{saf}}\) is the minimum distance between vehicles and \(k\) is any time step in \([1, ..., K]\). Thus, the nonlinear function that satisfies the condition in (5.28) is given by

\[ c_1[k] = d_{\text{saf}} - \|p_i[k] - p_j[k]\| \]  \hspace{1cm} (5.30)

and the number of constraints of this type is \(3Q(Q - 1)K\).

Similar to the objective function, it may be useful to provide the gradient when solving the nonlinear optimization problem directly. Given that

\[ \|p_i[k] - p_j[k]\| = \sqrt{(p_i[k] - p_j[k])^T(p_i[k] - p_j[k])} = \sqrt{p_i[k]^T p_i[k] - 2p_i[k]^T p_j[k] + p_j[k]^T p_j[k]}, \]
the gradient of (5.30) is computed as
\[
\frac{d}{dt} c_1[k] = - \frac{1}{\|p_i[k] - p_j[k]\|} \left( p_i[k] \frac{dp_i[k]}{dt} - p_i[k] \frac{dp_j[k]}{dt} - p_j[k] \frac{dp_i[k]}{dt} + p_j[k] \frac{dp_j[k]}{dt} \right). \tag{5.31}
\]

2. Clear visibility during replacement - When drones \(i\) and \(q\) are exchanging positions, they should not be detected by any of the cameras of the other drones in flight. Considering the situation where drone \(q\) cannot be seen in drone \(i\), we assumed that drone \(i\)'s camera is pointed at the target in the direction of the vector \(p_{Ti} = p_T - p_i\) and that it has a field of view of \(150^\circ\), which is equivalent to \(\alpha_{cam} = 75^\circ\) relative to \(p_{Ti}\). So that drone \(q\) is not seen in drone \(i\)'s camera, the former should be behind the view angle, which is equivalent to saying that the angle, \(\alpha_{iq}^i\), between \(p_{Ti}\) and the vector \(p_{qi} = p_q - p_i\) should be greater than \(\alpha_{cam}\). This constraint is illustrated in Figure 5.2 where the camera’s field of view is represented as a forbidden zone for any other drone.

Taking advantage of the inner product properties, we can define this constraint as
\[
p_T[k]^T p_{qi}[k] - \|p_T[k]\|\|p_{qi}[k]\| \cos(\alpha_{cam}) \leq 0. \tag{5.32}
\]

Thus, the nonlinear function \(c_2\) can be written as \(c_2(\chi) = p_{Ti}[k]^T p_{qi}[k] - \|p_{Ti}[k]\|\|p_{qi}[k]\| \cos(\alpha_{cam})\) and its gradient is given by
\[
\frac{d}{dt} c_2 = p_{qi}[k]^T \frac{dp_{Ti}[k]}{dt} + p_{Ti}[k]^T \frac{dp_{qi}[k]}{dt} - \frac{\|p_{Ti}[k]\|\|p_{qi}[k]\|\|p_{qi}[k]\|}{\|p_{qi}[k]\|\|p_{Ti}[k]\|} p_{Ti}[k]^T \frac{dp_{Ti}[k]}{dt} \cos(\alpha_{cam})
\]
\[
\quad - \frac{\|p_{Ti}[k]\|}{\|p_{qi}[k]\|} p_{qi}[k]^T \frac{dp_{qi}[k]}{dt} \cos(\alpha_{cam}). \tag{5.33}
\]

This constraint should only be imposed during the exchange of positions as described in Figure 1.4b. For a general scenario with \(Q\) drones where drone \(i\) is replaced by drone \(q\), drone \(q\) should
not be seen in the other $Q - 1$ drones (including drone $i$) during the intervals $T_5$ and $T_6$ and drone $i$ should not be seen in the other $Q - 1$ drones (including drone $q$) during $T_6$ and $T_7$. Therefore, the number of constraints of this type is

$$(Q - 1) \sum_{j=5}^{6} N_P(T_j) + (Q - 1) \sum_{j=6}^{7} N_P(T_j).$$

5.4 Sequential Convex Programming

Working directly with nonlinear constraints can be inefficient as they increase the complexity of the problem. As an alternative, we present an algorithm that replaces these constraints with linear approximations around a previous solution. This method solves a local optimization problem iteratively until a stopping condition is satisfied.

At iteration $s + 1$, nonlinear constraints can be linearised around the previous solution $x^s$ using a first order Taylor expansion. Considering that $f(x, y)$ is a generic nonlinear function and that $(x^s, y^s)$ is a known solution, then the expression for the Taylor’s first order expansion is given by

$$\tilde{f}(x, y) = f(x^s, y^s) + \frac{df(x, y)}{dx} \bigg|_{(x, y) = (x^s, y^s)} (x - x^s) + \frac{df(x, y)}{dy} \bigg|_{(x, y) = (x^s, y^s)} (y - y^s). \quad (5.34)$$

We can apply this formula to the nonlinear constraints described before and obtain the following approximated linear constraints:

1. Collision avoidance - assuming that $x^s = p_{i^s}[k] - p_{j^s}[k]$ and $f_1(x) = ||x||$, with $x = p_{i^s}[k] - p_{j^s}[k]$ (and $y = 0$), $\tilde{f}_1(x)$ is computed from (5.34) and is given by

$$\tilde{f}_1(x) = ||x^s|| + \frac{x^s}{||x^s||} T (x - x^s). \quad (5.35)$$

This results in the following constraint

$$\tilde{f}_1(x) \geq d_{saf}. \quad (5.36)$$

2. Clear visibility - assuming that $x^s = p_T[k] - p_{i^s}[k]$, $y^s = p_q^s[k] - p_{i^s}[k]$ and $f_2(x, y) = x^T y - ||x|| ||y|| \cos(\alpha_{cam})$, with $x = p_T[k] - p_{i^s}[k]$ and $y = p_q^s[k] - p_{i^s}[k]$, $\tilde{f}_2(x, y)$ is computed from (5.34) and given by

$$\tilde{f}_2(x, y) = \left( x^s T y^s - ||x^s|| ||y^s|| \cos(\alpha_{cam}) \right) + \left( y^s - ||y^s|| \cos(\alpha_{cam}) \frac{x^s}{||x^s||} \right)^T (x - x^s)
+ \left( x^s - ||x^s|| \cos(\alpha_{cam}) \frac{y^s}{||y^s||} \right)^T (y - y^s). \quad (5.37)$$

This results in the following constraint

$$\tilde{f}_2(x, y) \leq 0. \quad (5.38)$$

With these approximations, we can formulate the problem as a quadratic optimization problem, with
a quadratic objective function and affine constraints. The SCP algorithm is described as follows:

1. Choose a starting point, which can be the solution of the optimization problem without the nonlinear constraints.

2. From the previous point, compute the new $A_{in}$ and $b_{in}$ by adding the approximated constraints in (5.36) and (5.38).

3. Solve the quadratic optimization problem and obtain a solution $\chi^*$.

4. Check if the stopping condition is satisfied, which only happens if:
   - $\chi^*$ fulfils the nonlinear constraints.
   - convergence of the objective value is achieved, which occurs when $|f_0(\chi^{s-1}) - f_0(\chi^*)| < \delta$, with $\delta$ being a threshold parameter.

If this is not satisfied, go back to step 2 and use the computed solution to linearise the next iteration.

In the end, we obtain an optimal solution from a linearised convex optimization that satisfies all of the initial constraints, and is similar to the solution obtained using the nonlinear constraints directly.

5.5 Analysis of the optimization problem

As explained in Section 3.1, when formulating an optimization problem as a convex problem, there is a guarantee that if we can find a local optimal solution, then this solution is also unique and global. Furthermore, convex problems are usually less complex and simpler to prove convergence. For an optimization problem to be identified as convex, the objective function should be convex, the equality constraints linear and the inequalities defined by convex sets.

We can prove the convexity of the objective function with the second-order condition explained in Subsection 3.1.1. The Hessian matrix is obtained by deriving the gradient in (5.10), resulting in

$$\nabla^2 f_0 = P + P^T.$$  \hspace{1cm} (5.39)

The condition states that $f$ is convex if $\nabla^2 f(x) \succeq 0$, which only happens if $P \succeq 0$, i.e. $P$ is a symmetric semi-positive definite matrix, which is indeed the case.

As for the constraints, the combination of linear equalities and inequalities will yield a convex polyhedron, as described in Subsection 3.1.1 as a known example of a convex set. This is the case with the presented linear constraints, as they only represent upper and lower bounds on the trajectory and its derivatives. The nonlinear constraints are, however, not convex, which means that when using these constraints directly, the optimization problem becomes non-convex and the solution found is only locally optimal.

The nonlinearity can be avoided by applying the SCP algorithm previously described. In this way, the nonlinear constraints become linear for each iteration and the problem can be solved as a quadratic
5.6 Simulation test results

Several simulation tests were made, varying the number of drones, trajectories, positions in formation flight and even duration of the trajectories. In the end, we chose to present simulation tests for two different scenarios: (1) two drones following a target in a circular path, and (2) three drones following a target in a straight path. The tests were conducted on a Lenovo Laptop running Ubuntu 16.04 and equipped with an Intel Core i7-7700HQ CPU @2.80GHz and 16.00GB of RAM, where Matlab R2018a for Linux was running.

**Scenario 1:** two drones following a target in a circular path.

The formation flight positions were set with the parameter values presented in the Table 5.1. We planned a trajectory of 52 seconds with a fixed time step of $h = 0.5$ seconds, which was split into 12 intervals (according to Figure 1.4a) with the following durations:

<table>
<thead>
<tr>
<th>$T_1$</th>
<th>$T_2$</th>
<th>$T_3$</th>
<th>$T_4$</th>
<th>$T_5$</th>
<th>$T_6$</th>
<th>$T_7$</th>
<th>$T_8$</th>
<th>$T_9$</th>
<th>$T_{10}$</th>
<th>$T_{11}$</th>
<th>$T_{12}$</th>
<th>$t(s)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>13</td>
<td>16</td>
<td>22</td>
<td>24</td>
<td>26</td>
<td>28</td>
<td>34</td>
<td>37</td>
<td>47</td>
<td>50</td>
<td>52</td>
</tr>
</tbody>
</table>

| 40 |

Table 5.1: Formation flight positions with 2 drones

<table>
<thead>
<tr>
<th></th>
<th>$\epsilon_x$</th>
<th>$\epsilon_y$</th>
<th>$\epsilon_z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_1$</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$d_2$</td>
<td>4</td>
<td>0</td>
<td>-1</td>
</tr>
</tbody>
</table>

The problem was solved with 2 different methods: **(a)** applying the nonlinear constraints directly and **(b)** linearising the nonlinear constraints with the SCP algorithm. In the end, we obtained the trajectories shown in Figure 5.3, which were able to satisfy all of the linear and nonlinear constraints, as well as execute the manoeuvres described initially in Section 1.3.

To verify the satisfiability of the collision avoidance constraint, we plotted the distance between the 2 drones along the trajectory as represented in Figure 5.4. Here, we can see that the distance is never below the minimum safety distance allowed, which means that there is no risk of collision. Additionally, to verify if the clear visibility constraints are satisfied during the exchange of positions, we computed, for the time steps in $T_5$ and $T_6$, the angle between the directions $p_{T1} = p_T - p_1$ and $p_{21} = p_2 - p_1$ and, for the time steps in $T_6$ and $T_7$, the angle between the directions $p_{T2} = p_T - p_2$ and $p_{12} = p_1 - p_2$. These are represented in Figure 5.5, where we can see that they are always above the camera's field of view angle, which means that there is indeed clear visibility during the mentioned intervals.
Figure 5.3: Trajectories generated for scenario 1. Note that the circles are the initial positions and the stars the final positions.

Figure 5.4: Distance between drones in scenario 1

Figure 5.5: Clear visibility constraint in scenario 1
Table 5.2: Differences in acceleration, velocity and position between solutions for scenario 1

<table>
<thead>
<tr>
<th>max ( \Delta \text{accel} )</th>
<th>max ( \Delta \text{vel} )</th>
<th>max ( \Delta \text{pos} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2489%</td>
<td>0.4730%</td>
<td>0.00372%</td>
</tr>
</tbody>
</table>

Table 5.3: Computation time and objective function obtained in each method for scenario 1

<table>
<thead>
<tr>
<th>Methods</th>
<th>Computation time(s)</th>
<th>Objective function value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>8.0931</td>
<td>117.1298</td>
</tr>
<tr>
<td>(b)</td>
<td>2.2896</td>
<td>20.8938</td>
</tr>
</tbody>
</table>

From the two methods (a) and (b), we obtained two identical solutions. In Table 5.2, we present the maximum difference in acceleration, velocity and position computed throughout the test. Since the percentages obtained were small (< 1%), we can say that the solutions were in fact the same and that this error might be related to numerical uncertainties in the computation.

Then, to obtain the computation time necessary to solve the problem with each method, both algorithms were tested 100 times. Table 5.3 presents the mean of the computation time as well as the value of the objective function obtained. We can see that the computation time in method (a) is about 4 times the computation time in (b) and that the value for the objective function in (a) is about 5.6 times the value obtained in (b). Thus, we can conclude that even though both methods are able to give an optimal solution, the SCP algorithm was more efficient.

**Scenario 2:** three drones following a target in a straight path.

In this case, the formation flight positions were set with the parameters presented in Table 5.4. The duration of the trajectory and of its intervals were the same as before. The problem was solved with the same two methods (a) and (b), where we obtained the trajectories in Figure 5.6. Here, we can see that these are able to solve the problem of drones’ replacement, as they satisfy all of the linear and nonlinear constraints and execute the necessary manoeuvres.

Table 5.4: Formation flight positions with 3 drones

<table>
<thead>
<tr>
<th>( \epsilon_x )</th>
<th>( \epsilon_y )</th>
<th>( \epsilon_z )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( d_1 )</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>( d_2 )</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>( d_3 )</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

To prove that the nonlinear constraints are being satisfied, we computed the distances between any two drones in the system (1 and 2, 1 and 3 and 2 and 3) and plotted them in Figure 5.7, where we observe that these are always above the minimum safety distance, thus ensuring collision avoidance. As for the clear visibility condition, to check if drone 3 is not detected during \( T_5 \) and \( T_6 \), we computed the angles between \( p_{T1} \) and \( p_{a1} \) and between \( p_{T2} \) and \( p_{a2} \), and then, to check if drone 1 is not detected during \( T_6 \) and \( T_7 \), we computed the angles between \( p_{T2} \) and \( p_{12} \) and between \( p_{T3} \) and \( p_{13} \). These are represented in 5.8, where we can see that they are always above the camera’s field of view angle, which
means that this constraint is satisfied.

![Trajectories in 3D](image)

(a) Trajectories in 3D

![Projection of the trajectory in XY plane](image)

(b) Projection of the trajectories in the plane XY

Figure 5.6: Trajectories generated for scenario 2. Note that the circles are the initial positions and the stars the final positions.

![Distance between drones in scenario 2](image)

Figure 5.7: Distance between drones in scenario 2

![Angular data](image)

Figure 5.8: Clear visibility constraint in scenario 2

Furthermore, we computed the maximum differences between solutions obtained in acceleration,
velocity and position, shown in Table 5.5. As these are small enough (< 1.2%), we can infer that the solutions are the same. Then, Table 5.6 presents the computation times and the values of the objective function obtained after running the algorithm 100 times. From these, we can say that the problem becomes more complex as more vehicles are added to the system, and therefore, requires more computation time to reach a solution. We also verify that with the solution obtained in (a), the computation time is about 6.4 times more than in (b) and the value of the objective function 1.6 more. Hence, we can conclude that the SCP algorithm in (b) is more efficient. Note that the differences in computation time are also related to the tools being used in each method, as Cplex is usually faster than Matlab. In method (a), we use the Matlab's fmincon function to solve the problem with nonlinear constraints, whereas in method (b), we only use Cplex’s cplexmiqp function since the problem is quadratic and only has linear constraints.

Table 5.5: Differences in acceleration, velocity and position between results in scenario 2

<table>
<thead>
<tr>
<th>max $\Delta$accel</th>
<th>max $\Delta$vel</th>
<th>max $\Delta$pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1602%</td>
<td>1.2042%</td>
<td>0.0699%</td>
</tr>
</tbody>
</table>

Table 5.6: Computation time and objective function obtained with each method for scenario 2

<table>
<thead>
<tr>
<th>Methods</th>
<th>Computation time (s)</th>
<th>Objective function value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>27.1526</td>
<td>258.1988</td>
</tr>
<tr>
<td>(b)</td>
<td>4.2484</td>
<td>161.9633</td>
</tr>
</tbody>
</table>
Chapter 6

Online Planning and Control

This chapter is dedicated to the second stage of this thesis, where we address the problem described in Section 1.3 with a real time trajectory planning and control algorithm. To do so, a Nonlinear Model Predictive Controller is implemented, which is able to simultaneously generate trajectories for a short time horizon and control the system in real time. The method implemented was based on the work presented in [30], adapted to the problem at hand as explained below.

As explained in Section 3.2, MPC is essentially a method that at each time step solves an optimization problem for a limited time horizon, resulting in a sequence of inputs and state predictions. The first input given by the sequence can then be applied to the system, where we obtain a new updated state, which, consequently, will be used to update the initial condition of the optimization problem to be solved in the next iteration of the MPC.

The optimization problem is similar to one implemented in Chapter 5, where we generated optimal trajectories taking into consideration the systems’ physical constraints, collision avoidance and clear visibility when needed, as well as guaranteeing that each vehicle is able to follow the target in formation flight. The major difference of the optimization problem used in this chapter is that we are explicitly considering the orientation of the vehicle and the position and orientation of the camera. Additionally, we are also including perception objectives in the objective function that are related to how the camera of the UAV is looking at the target. For this purpose, it was necessary to consider the three coordinate systems represented in Figure 3.1: world frame $W$, body frame $B$, and camera frame $C$.

This algorithm was implemented in Matlab using a Matlab interface for the ACADO toolbox [34], which is able to generate C/C++ code and setup the NMPC problem. The solution was then computed using a qpOASES solver [35].
6.1 System dynamics

In order for the problem to be able to run in real time, it is important that the computation time at each time step is smaller than the sampling time, which can be achieved by reducing the number of state variables and simplifying the system dynamics. For this reason, we introduce some modifications to the system dynamics described in Section 4.1, which included:

- using quaternions \( q = q_w + q_x i + q_y j + q_z k \) instead of Rotation matrices, which can also be represented by a vector \( q \in \mathbb{R}^4 \) in the unit sphere \( S^3 \in \mathbb{R}^4 \), with the quaternion multiplication operator denoted as \( \otimes \).

- considering as control variables the mass-normalized thrust \( \tilde{u}_i = \frac{F_1 + F_2 + F_3 + F_4}{m} \), where \( F_i \) is the thrust generated by the \( i^{th} \) motor and \( m \) is the mass, and the angular velocities \( \omega_B = [p, q, r]^T \) expressed in the body frame, instead of net thrust and body moments.

Thus, considering a system of \( Q \) vehicles, the state of the system is given by

\[
\mathbf{x} = [\mathbf{p}_1, \ldots, \mathbf{p}_Q, \mathbf{v}_1, \ldots, \mathbf{v}_Q, \mathbf{w}_q B_1, \ldots, \mathbf{w}_q B_Q]^T, \tag{6.1}
\]

where \( \mathbf{p}_i = [p_{ix}, p_{iy}, p_{iz}]^T \) and \( \mathbf{v}_i = [v_{ix}, v_{iy}, v_{iz}]^T \) represent the position and velocity in the world frame and \( \mathbf{w}_q B_i \) represents the quaternion of the body described in the world frame of a drone \( i \), with \( i = 1, \ldots, Q \), and the vector of control inputs is

\[
\mathbf{u} = [\tilde{u}_{11}, \ldots, \tilde{u}_{1Q}, \omega_{B1}, \ldots, \omega_{BQ}]^T. \tag{6.2}
\]

The system dynamics can then be formulated, for each drone \( i \), as

\[
\begin{align*}
\dot{\mathbf{p}}_i &= \mathbf{v}_i, \\
\dot{\mathbf{v}}_i &= -g \mathbf{w}_z + \mathbf{R}_q (\mathbf{w}_q B_i) \mathbf{w}_z \mathbf{w}_z \\
\mathbf{w}_q B_i &= \frac{1}{2} \Lambda(\mathbf{w}_B_i)^W \mathbf{w}_q B_i,
\end{align*} \tag{6.3}
\]

where \( g = 9.81 \text{ m/s}^2 \) is the gravity constant, \( \mathbf{w}_z = [0, 0, 1]^T \) is the direction of the \( z \) axis in the world frame, \( \mathbf{R}_q (\mathbf{w}_q B_i) \) is the quaternion rotation operation presented in (3.17) and \( \Lambda(\mathbf{w}_B_i) \) is the skew-symmetric matrix in 4D of \( \mathbf{w}_B_i \) given by

\[
\Lambda(\mathbf{w}_B_i) = \begin{bmatrix}
0 & -p_i & -q_i & -r_i \\
p_i & 0 & r_i & -q_i \\
q_i & -r_i & 0 & p_i \\
r_i & q_i & -p_i & 0
\end{bmatrix}. \tag{6.4}
\]
6.2 Perception objectives

As mentioned before, we are including in this chapter additional perception objectives that we aim to minimize. These are related to the deviation (and its derivative) of the center of the camera to the target, in order to guarantee that the UAV is looking at the target during formation flight and without oscillations. Note that in this case we are assuming that the position and orientation of the camera with respect to the body is fixed throughout the flight.

Suppose that $p_i, v_i$ and $^Wq_{iB}$ are the position, velocity and orientation of drone $i$’s body in the world frame, $^B_ip_{Ci}$ and $^Biq_{Ci}$ are the position and orientation of its camera with respect to the body frame and $p_T$ and $v_T$ are the position and velocity of the target in the world frame. Then, the position of the target in the camera frame $^Ci p_T$ is given by

$$^Ci p_T = R_q(^Wq_{Bi} \otimes ^Biq_{Ci})^{-1}[p_T - (R_q(^Wq_{Bi})^Bip_Ci + p_i)].$$

(6.5)

The projection of $^Ci p_T$ in the image plane coordinates $(s_x, s_y)$ can be computed according to the classical pinhole camera model. Considering the focal lengths $f_x$ and $f_y$ for pixel rows and columns, these can be expressed as

$$s_x = f_x \frac{^Ci p_T}{^Ci p_Tz}, \quad s_y = f_y \frac{^Ci p_T}{^Ci p_Tz}.$$ 

(6.6)

To further reduce the offset and oscillations of the camera view in respect to the target, we can also minimize the velocity of the projection onto the image plane. Thus, deriving (6.6), we obtain

$$\dot{s}_x = f_x \frac{^Ci \dot{p}_T}{^Ci p_Tz} \frac{^Ci p_Tx}{^Ci p_Tz}, \quad \dot{s}_y = f_y \frac{^Ci \dot{p}_T}{^Ci p_Tz} \frac{^Ci p_Ty}{^Ci p_Tz},$$

(6.7)

where $^Ci \dot{p}_T$ is the derivative of (6.5) and can be computed as

$$^Ci \dot{p}_T = R_q \left( -\frac{1}{2} \Lambda (R_q(\dot{\omega}_{Bi})^Biq_{Ci}) ^Ci p_T \right) + R_q \left( ^Wq_{Bi} \otimes ^Biq_{Ci} \right)^{-1}[v_T - (R_q(\frac{1}{2} \Lambda (\omega_{Bi})^Wq_{Bi})^Bip_Ci + v_i)].$$

(6.8)

Finally, we can obtain the perception parameters

$$z_i = [s_x, s_y, \dot{s}_x, \dot{s}_y]^T$$

(6.9)

for each drone, with $i = 1, \ldots, Q$, which will be added in the objective function of the MPC algorithm.
6.3 NMPC algorithm

As described in 3.2, MPC is a method of iteratively solving an optimization problem for a finite time horizon that can be generally formulated as

\[
\min_{U} V_N(X, U, Z),
\]

subject to

\[
\begin{align*}
 r(X, U, Z) &= 0, \\
h(X, U, Z) &\leq 0,
\end{align*}
\]  

(6.10)

where \( X = [x_0, ..., x_N] \), \( U = [u_0, ..., u_{N-1}] \), and \( Z = [z_0, ..., z_{N-1}] \) are state, input and perception sequences for the time horizon with the components denoted in (6.1), (6.2) and (6.9) respectively, \( V_N(X, U, Z) \) is the objective function and \( r(X, U, Z) \) and \( h(X, U, Z) \) represent equality and inequality constraints that the system must obey. To solve this optimization problem, it is necessary to discretize the system dynamics with a sampling time \( T_s \) and to consider a time horizon \( T_h \), so that at each iteration, we obtain a sequence of \( N \) inputs \( U \) and sequence of \( N + 1 \) state predictions \( X \), with \( N = T_h/T_s \). The components of the algorithm are described below.

6.3.1 Objective function

The objective function is formulated as a Linear-Quadratic Regulator that is applied to the sequence of parameters that we want to minimize and is composed by two main costs: the state, perception and input stage costs for the \( N \) time steps, and the final state cost for the time step in \( N + 1 \). These can be written as

\[
V_N(X, U, Z) = (x_N - x_N^{ref})^T Q_{x_N}(x_N - x_N^{ref}) + \sum_{i=0}^{N-1} \left( (x_i - x_i^{ref})^T Q_{x_i}(x_i - x_i^{ref}) + (z_i - z_i^{ref})^T Q_{z_i}(z_i - z_i^{ref}) + (u_i - u_i^{ref})^T R_i(u_i - u_i^{ref}) \right),
\]  

(6.11)

where:

- \( Q_{x_i}, Q_{z_i} \) and \( R_i \), with \( i = 0, ..., N - 1 \), are the time-varying state, perception and input cost matrices and \( Q_{x_N} \) is the final state cost penalty. These matrices can be tuned during flight in order to adjust the contribution of the state, perception and inputs reference and to obtain a desirable response;

- \( x_i^{ref} \), with \( i = 1, ..., N \), and \( z_i^{ref} \) and \( u_i^{ref} \), with \( i = 0, ..., N - 1 \), are the reference of state, perception and inputs that may also vary throughout the time horizon.

The state references \( x_i^{ref} \) have the purpose of describing the drones manoeuvres so that they follow the flight plans in Figure 1.4, which include starting at an initial position, taking off and joining formation flight, going to a different position in formation flight during replacement, then continuing formation flight and finally landing at the final position. These reference trajectories are defined for each drone and serve as a baseline for the trajectories to be generated by the MPC, which will also take into consideration the system’s physical constraints, collision avoidance and clear visibility during formation flight.
As for the perception references \( z_{i}^{\text{ref}} \), these should be zero in this particular case, so that the camera center is aligned with the target (offset equal to zero) and the image is as static as possible (derivative of the projection equal to zero). Note that, in some cases, we might want the target to be aligned with a different coordinate \((s_x, s_y)\) in the image plane, which is achieved by using that point as a perception reference. We can also assume that the reference values for the inputs are zero, in order to economize the drones batteries and reduce their oscillations during flight.

### 6.3.2 Constraints

To formulate the NMPC problem, it was necessary to add several types of constraints, that included the following:

- **System dynamics** - the equations in (6.3) were used as constraints of the MPC problem. These were then discretized using multiple shooting as a transcript method and a Runge Kutta integration scheme with a Gauss-Legendre integrator of order 4, which were executed with ACADO’s [36] generation code functions. Further details are provided in [37].

  Note that in the ACADO toolkit there is not an available integration scheme adapted to quaternions, thus, the integration is implemented with approximation errors. In [38], the authors compare Runge Kutta and Crouch-Grossman Lie Group (adapted to quaternions) integration methods to integrate quaternions and conclude that as the integration time step decreases, the errors associated with the Runge Kutta algorithms also decrease, resulting in valid solutions. We also noted this after implementing the algorithm for the flight plans described, where the quaternion normalization errors \( e_{\|q\|} = |1 - \|q\|| \) obtained were small (around \( 1e10^{-12} \)).

- **Control bounds** - we imposed limits on the control inputs, in order to avoid the saturation of the actuators, which, for a vehicle \( i \), with \( i = 1, \ldots, Q \), can be expressed as

  \[
  -\tilde{u}_{1\max} \leq \tilde{u}_i \leq \tilde{u}_{1\max},
  \]

  \[
  -\omega_{\max} \leq \omega_{B_i} \leq \omega_{\max}.
  \]  

- **State bounds** - we imposed limits on the position and velocity to guarantee that the solution gives feasible results, which can be written as

  \[
  p_{\min} \leq p_i \leq p_{\max},
  \]

  \[
  -v_{\max} \leq v_i \leq v_{\max}.
  \]  

- **Collision constraint** - A similar equation to (5.29) was used to ensure collision avoidance between any two vehicles. In this case, if \( p_i \) and \( p_j \) are the position of two vehicles in the world frame, such that \( i = 1, \ldots, Q, j = 1, \ldots, Q, i \neq j \), then the distance between them \( d_{ij} \) is

  \[
  d_{ij} = \|p_i - p_j\|.
  \]
This constraint imposes that $d_{ij}$ has to be greater than a safety distance $d_{saf}$, which means

$$d_{ij} \geq d_{saf}.$$  \hfill (6.15)

- **Clear visibility** - A similar equation to (5.32) is also implemented to guarantee that the drones in replacement are not seen in each others cameras. In this case however, instead of looking at the angle between the target and the other drone’s position with respect to its own position, this constraint uses the angle with respect to the position of its camera.

Considering a scenario where drone $i$ is being replaced by drone $q$, then the vector between the position of the target in the world frame, $p_T$, and the position of drone $i$’s camera is

$$p_{Ci_T} = p_T - \left( R_q(Wq_{Bi})^B_i p_{Ci} + p_i \right),$$  \hfill (6.16)

and the vector between the position of drone $q$ in the world frame, $p_q$, and the position of drone $i$’s camera is

$$p_{Ci_q} = p_q - \left( R_q(Wq_{Bi})^B_i p_{Ci} + p_i \right).$$  \hfill (6.17)

From these, we can compute the angle between the vectors $p_{Ci_T}$ and $p_{Ci_q}$, denoted by $\alpha_{ij}^q$, according to

$$\alpha_{ij}^q = \frac{p_{Ci_T}^T p_{Ci_q}}{||p_{Ci_T}|| ||p_{Ci_q}||}.$$  \hfill (6.18)

To ensure that drone $q$ is not seen by drone $i$, the angle $\alpha_{ij}^q$ should be greater than the cameras field of view angle $\alpha_{cam}$, as represented in Figure 5.2. Thus, the constraint can be written as

$$\alpha_{ij}^q \geq \alpha_{cam} \iff p_{Ci_T}^T p_{Ci_q} - ||p_{Ci_T}|| ||p_{Ci_q}|| \cos(\alpha_{cam}) \leq 0.$$  \hfill (6.19)

Note that this condition should only be applied during the replacement portion of the flight plan, i.e. if drones 1 and 2 are exchanging positions, then during the intervals $T_5$ and $T_6$ a constraint should be imposed with $i = 1$ and $q = 2$ and during $T_6$ and $T_7$, a similar constraint should be implemented but with $i = 2$ and $q = 1$. In the other intervals, it should be omitted to avoid unnecessary constraints to the MPC problem.

### 6.3.3 ACADO implementation

To formulate the NMPC problem, we used the **ACADO** toolbox, which is a software environment and algorithm collection able to generate C/C++ code in order to solve automatic control and dynamic optimization problems, such as an MPC problem [36]. The algorithm was implemented in a Matlab interface for **ACADO** that allows to formulate the problem in Matlab with a special **ACADO** syntax [34], which is then solved with a qpOASES solver [35]. In this way, we are able to use this toolkit and still take advantage of other Matlab functions.
6.3. NMPC algorithm

To formulate the problem in ACADO, the following steps had to be executed:

- Provide a set of differential states, \( x \), and control inputs, \( u \), and for this particular problem, a set of online data variables, such as the position and velocity of the target that may vary for each iteration;
- Provide a dynamic model \( \dot{x} = f(x, u, t) \), such as (6.3);
- Define MPC parameters, that include the time horizon and sampling time, the objective function (6.11) and constraints (as described before).

Code was then generated in 2 steps: one to represent a system simulator that integrates the system dynamics in (6.3), and other to solve the NMPC problem. Besides C/C++ code, ACADO also generates executable mexa64 files that can be used to test the algorithm either in a Matlab script or in a Simulink diagram. To do so, one has to define for each iteration the state, perception and input references, the cost matrices \( Q_x \), \( Q_z \) and \( R \), the bounds for the constraints and online data for the next \( N \) time steps. Then, we can solve the current NMPC problem by using the corresponding executable files, which results in sequences of inputs and predicted states. Afterwards, the first input \( u_0 \) can be applied to the system dynamics, from which we obtain a new updated state that will be used in the next MPC iteration. Figure 6.1 shows a block diagram that represents the MPC controlled system.

![Diagram of the MPC controlled system](image)

6.3.4 Stability analysis

Studying the stability of our MPC algorithm is a challenging task, as we have a nonlinear dynamic system with quaternion operations subject to state, control and nonlinear path constraints (collision avoidance and clear visibility).

In Subsection 3.2.1, we presented a theorem that proves the stability of a Receding Horizon Control system with control and state constraints for a scalar state and control input, where instead of following the reference, the controller forces the solution to the origin. Note that even though this theorem was formulated with scalar variables, its results can be generalized for a case with vectorial state and control inputs. Therefore, if we consider a simplified problem, where we exclude the nonlinear constraints and the perception objectives in the cost function, and consider that the optimization variable is given by the error between the original state and control inputs and their respective references, then, in this case, we can verify the assumptions a-e by choosing the terminal set \( \mathcal{X}_f \), the terminal control law \( \mathcal{K}_f \) and terminal state cost \( \mathcal{L}_f \) appropriately. By applying the theorem, we would conclude that the origin is globally
attractive in the set of feasible positive initial states $S_N$ and if the optimal objective function $V_N^{\text{OPT}}$ is continuous around a neighbourhood of the origin, then the origin would be asymptotically stable in $S_N$.

For the nonlinear problem in hand, proving such statements is more challenging, falling out of the scope of this thesis. Nonetheless, based on the presented concepts to achieve stability, we can discuss techniques to increase the practical stability of our problem.

One way to is to define higher weights for the terminal cost matrix $Q_{xN}$ in comparison with the weights in the stage cost matrices $Q_{xk}$ and $Q_{zk}$, with $k = 0, ..., N - 1$. We could also adjust the weight contributions in the cost functions for the state and perception parameters according to the different flight phases. In addition, we should make sure that the flight plan considered is feasible by guaranteeing that the distances in formation flight are greater than the minimum safety distance allowed in collision avoidance and by giving enough time between flight transitions. Another suggestion is to decrease the duration of the interval $T_6$, where a double clear visibility constraint is imposed so that both drones in replacement are not seen in the other cameras, as it may be too restrictive and lead to brief periods of infeasible solutions.

### 6.4 Simulation results

Several simulation tests were made, varying the number of drones and trajectories in formation flight, as well as MPC parameters such as sampling time and weights of the cost matrices. In this section, we present simulation test results for similar scenarios to the ones discussed in Section 5.6. The tests were conducted on the same Lenovo Laptop running Ubuntu 16.04 and equipped with an Intel Core i7-7700HQ CPU @2.80GHz and 16,00GB of RAM, where Matlab R2018a for Linux was running.

**Scenario 1:** two drones following a target in a circular path

We defined the NMPC problem by setting the sampling time as $T_s = 0.3$ seconds and the time horizon as $T_h = 6$ seconds, which means that $N = 20$. We used the formation flight positions presented in Table 6.1 and considered the same trajectory plan described in Section 5.6.

As for the cost matrices, we assigned scaling factors $y_{\text{scal}}$ to each component of the objective function (Table 6.2) so that smaller scaling factors led to higher weight contributions in the cost function, and then proceeded to compute the cost matrices by considering that these are diagonal and that, for the stage costs, its diagonal entries are $1/y_{\text{scal}}^2$ and for the final cost they are $25/y_{\text{scal}}^2$. This means that the terminal cost is always 25 times more than the stage cost for $x$. Note that it can be useful to change the contributions of each parameter throughout the flight. For instance, before takeoff, the camera does not have to be pointed at the target. Therefore, it is not necessary to consider the perception objectives during these intervals, which means $Q_z$ should be zero.
Figure 6.2: Trajectories obtained with MPC for scenario 1. Note that dotted line corresponds to the trajectory reference and the full line to the trajectory generated; the circles are the initial positions and the stars the final positions.

Table 6.1: Formation flight positions with 2 drones considered in the MPC algorithm

<table>
<thead>
<tr>
<th></th>
<th>ε_t</th>
<th>ε_x</th>
<th>ε_y</th>
<th>ε_z</th>
</tr>
</thead>
<tbody>
<tr>
<td>d1</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>d2</td>
<td>4</td>
<td>0</td>
<td>-1</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 6.2: Scaling factor considered in the cost matrices

<table>
<thead>
<tr>
<th>p</th>
<th>v</th>
<th>q</th>
<th>z</th>
<th>(\tilde{u}_1)</th>
<th>(\omega_B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>5</td>
<td>10</td>
<td>5</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>

In the end, we obtained the trajectories represented in Figure 6.2, where we can see that both drones were able to execute the manoeuvres planned and satisfy at least the collision avoidance constraint by temporarily moving away from the reference. In Figure 6.3, the control inputs of the mass-normalized thrust \(\tilde{u}_1\) and angular velocities \(\omega_B\) of each drone along the trajectory are represented. Here, we can verify that these are never saturated, which means that the drones are able to go to their desired state.

Additionally, we plotted the perception parameters \(z = [s_x, s_y, \dot{s}_x, \dot{s}_y]\) to see if these were minimized during the test. In Figure 6.4, we notice that these values are small but not zero. In order to further minimize them, we could have increased their contribution in the cost matrices, but this would compromise other factors, such as following the reference position.

Then, in order to prove that the collision avoidance constraint was satisfied, we plotted the distance between drones during the simulation (Figure 6.5) and verified that this is always above the minimum safety distance \(d_{saf}\). We also tested the clear visibility constraint, by plotting the angle between drone 2 and the target w.r.t. the camera of drone 1 during the intervals \(T_5\) and \(T_6\), as well as the angle between drone 1 and the target w.r.t. the camera of drone 2 during the intervals \(T_6\) and \(T_7\) (Figure 6.6) and verified that these are always above the field view angle of the camera \(\alpha_{cam}\), which means that they are
not detected during the replacement process.

Figure 6.3: Control inputs obtained with MPC for scenario 1

Figure 6.4: Perception parameters obtained with MPC for scenario 1
6.4. Simulation results

Finally, we can analyse the computation time that it takes to compute each iteration. In Figure 6.7, we represented the sampling time $T_s$, the computation time that it took for the MPC to compute the solution at each time step (RTI time), as well as the computation time that it took to fully complete the iteration which includes data initialization, solving the MPC and obtaining a new state from the system dynamics (iter time). Since the computation times are significantly smaller than the $T_s$, we can then conclude that the MPC can solve the problem in real time.

**Scenario 2**: three drones following a target in a straight path.

For this scenario, we used the same sampling time of $T_s = 0.3$ seconds and time horizon of $T_h = 6$ seconds, thus $N = 20$. We also used the formation flight positions defined in Table 6.3 and considered the same trajectory plan and cost matrices as before.
Chapter 6. Online Planning and Control

Figure 6.7: Computation time obtained with MPC in scenario 1

Table 6.3: Formation flight positions with 3 drones considered in the MPC algorithm

<table>
<thead>
<tr>
<th>d</th>
<th>ϵ_x</th>
<th>ϵ_y</th>
<th>ϵ_z</th>
</tr>
</thead>
<tbody>
<tr>
<td>d1</td>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>d2</td>
<td>4</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>d3</td>
<td>4</td>
<td>-2</td>
<td>0</td>
</tr>
</tbody>
</table>

In the end, we obtained the trajectories represented in Figure 6.8, where we can then see that all drones are able to follow the flight plan and still satisfy the collision avoidance constraint by temporarily moving away from the reference. Then, in Figure 6.9, we analysed the control inputs by plotting how they evolve along the trajectory. We can then see that there is a larger demand during takeoff, landing and exchange of position, but these never exceed the actuators saturation.

Figure 6.8: Trajectories obtained with MPC for scenario 2. Note that doted line corresponds to the trajectory reference and the full line to the trajectory generated; the circles are the initial positions and the stars the final positions.
6.4. Simulation results

We also plotted the perception parameters \( \mathbf{z} = [s_x, s_y, \dot{s}_x, \dot{s}_y] \) (Figure 6.10), where we note that in some cases it was not possible for these to be minimized (example of \( \dot{s}_{x1} \) and \( \dot{s}_{y1} \) for drone 1), due to the increased complexity of the problem. However, this issue does not jeopardise the success of the task, since it only happens after the replacement of drone 1 by drone 3, after which drone 1 becomes inactive and eventually lands.

Furthermore, we checked the performance of the collision avoidance constraint, for which we plotted the distance between each drone along the trajectory (Figure 6.11) and verified that this is always above the minimum safety distance \( d_{saf} \). We also analysed the clear visibility constraint by plotting the angles between drone 3 and the target w.r.t. the camera of drone 1 and 2 during the intervals \( T_5 \) and \( T_6 \), as well as the angles between drone 1 and the target w.r.t. the camera of drone 2 and 3 during the intervals \( T_5 \) and \( T_6 \). From these, we can also see that the angles are always above the minimum required (\( \alpha_{cam} \)), thus ensuring clear visibility during the replacement.

Finally, we analysed the computation time necessary by plotting in Figure 6.13 the sampling time \( T_s \), the computation time to reach an MPC solution (\( RTI \) time) and the computation time of each iteration (\( iter \) time) and comparing these with the values obtained in Figure 6.7. We can see that both computation times \( RTI \) time and \( iter \) time are around 3 times more than before, but they still do not exceed the maximum limit \( T_s \). The significant increase in computation time is related to the increase of complexity of the problem, as by adding a new vehicle, we are increasing the size of the state, control inputs and amount of constraints. In order to obtain smaller computation times, we could have increased the sampling time or decreased the time horizon, in order to have smaller sequences of \( N \) elements.
Chapter 6. Online Planning and Control

Figure 6.10: Perception parameters obtained with MPC for scenario 2

Figure 6.11: Distance between drones in scenario 2
6.4. Simulation results

Figure 6.12: Clear visibility constraint in scenario 2

Figure 6.13: Computation time obtained with MPC in scenario 2
Chapter 7

SITL simulations

In order to verify if the trajectories generated for the pre-planned stage in Chapter 5 and for the online planning stage in Chapter 6 are feasible and able to execute the manoeuvres described initially in Section 1.3, these are tested in Software-In-The-Loop simulations, which are made in a Gazebo simulation environment with the MULTIDRONE simulator [1], represented in Figure 7.1. The UAVs are then setup with a PX4 controller that communicates with ROS (Robot Operating System) through the MAVROS protocol [39].

All of the simulations presented in this Chapter were conducted on the same Lenovo Laptop running Ubuntu 16.04 and equipped with an Intel Core i7-7700HQ CPU @2.80GHz and 16.00GB of RAM. Note that the results obtained in these simulations should closely resemble the ones obtained with real experiments, as these employ the same algorithms and software for the autopilot and the physics engine faithfully reproduces the vehicle’s dynamics. We were able to record videos showcasing the tests presented in this Chapter, which are available at: https://fenix.tecnico.ulisboa.pt/homepage/ist178289/trajectory-planning-and-control-for-drone-replacement-during-formation-flight.

Figure 7.1: Gazebo environment with a MULTIDRONE simulator
7.1 Tests with pre-planned trajectories

At first, we tested the trajectories generated with the method discussed in Chapter 5. To do so, we had to make some changes in the code in order to adjust certain parameters such as the target trajectory (assumed to be a straight path with constant velocity) and formation positions. After running the code, we obtained a list of positions and velocities that represented the trajectories generated for each drone.

Afterwards, we made a ROS node for each drone, integrated in the MULTIDRONE simulator, that sends commands to a UAV so that it follows the desired trajectory. The test is then composed by two parts: a preparation part, in which the UAVs go to their initial position, \( p_{init} = [x_{init}, y_{init}, 0] \), by taking off to a certain takeoff height, \( h_{TO} \), flying to an intermediate position, \( p_{interm} = [x_{init}, y_{init}, h_{TO}] \), and landing at \( P_{init} \); and an execution part, in which velocity commands based on the position error between the desired and current position and on the desired velocity at each time step are sent to the UAVs so that they follow a moving car with a predefined trajectory. At the same time, the orientation of the drone’s camera is rectified so that the camera is always looking at the target during the test.

In this section, we present a simulation test made for a scenario with 3 drones, where we considered that drones 1 and 2 would start following the target at the beginning of the test. Then, at a certain point, drone 1 would run out of battery and thus would be replaced by drone 3 and leave the formation flight, while the others would proceed with the task. We considered a trajectory plan of 52 seconds similar to the one described in Section 5.6 and used the formation flight positions indicated in Table 7.1.

<table>
<thead>
<tr>
<th>( \epsilon_t )</th>
<th>( \epsilon_x )</th>
<th>( \epsilon_y )</th>
<th>( \epsilon_z )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( d1 )</td>
<td>0</td>
<td>-5</td>
<td>1</td>
</tr>
<tr>
<td>( d2 )</td>
<td>0</td>
<td>-5</td>
<td>-1</td>
</tr>
<tr>
<td>( d3 )</td>
<td>0</td>
<td>-5</td>
<td>3</td>
</tr>
</tbody>
</table>

In the end, we obtained the simulation test showcased in the Waypoint Trajectories video, where we can see that the UAVs were able to follow the flight plan previously described without any risk of collision and without being seen by their peers in formation flight. This can also be verified in the plots from Figure 7.2, where we represent the experimental and reference trajectories and note that the real trajectories are not as smooth, especially in the flight phase transitions. Additionally, we also examined the position error between the reference and the experimental trajectory (Figure 7.3) and noticed small deviations between positions that might be related to uncertainties in the measurements as well as the input demands during the flight transitions.

We also analysed the collision and clear visibility constraints by plotting in the Figures 7.4 and 7.5 the distances and angles described in the other simulation tests with 3 drones. The plot of distances shows that this constraint was always satisfied (distances above \( d_{saf} \)). As for the angles, we notice that after \( t = 25 \) seconds (end of \( T_6 \)) both angles between drone 3 and the target w.r.t the cameras in drone 1 and 2 fall below the supposed camera’s field of view angle. However, as these infringements were small, they did not compromise the task and drone 3 still managed to not be seen by the others. We can thus
7.1. Tests with pre-planned trajectories

conclude that the actual angle of view of the drones’ cameras is lower than the one considered in the
constraint.

(a) Trajectories in 3D

(b) Projection of the trajectories in the plane XY

Figure 7.2: Experimental trajectories from waypoint references for with 3 drones. Note that doted line
 corresponds to the trajectory reference and the full line to the actual positions of the UAVs; the circles
 are the initial positions and the stars the final positions.

Figure 7.3: Position error for waypoint trajectories

Figure 7.4: Distance between drones for waypoint trajectories with 3 drones
In a second stage, we tested the feasibility of the NMPC algorithm described in Chapter 6, which involved creating another ROS node, integrated in the MULTIDRONE simulator, that would use the C/C++ code files generated with ACADO to solve the MPC problem at each time step.

To do so, we also made some modifications to the previous algorithm, which besides adjusting some parameters in the MPC problem, the duration of trajectories and the formation positions, also included removing the perception objectives explained in Section 6.2, since the orientation of the camera is no longer fixed (as previously assumed) and changes according to the relative position of the target. With this restriction, the algorithm would try to optimize the orientation of the drone so that the camera would stay pointed at the target and without oscillations, but since the camera is not fixed, this would not have the desired effect, and was, therefore, omitted.

Another modification was made for the type of control inputs to be applied. As described in Chapter 6, the MPC computes a sequence of inputs (\( \tilde{\mathbf{u}} \) and \( \omega_B \)) and a sequence of predicted states (position, velocity and quaternion) at every iteration. In this simulation, we still wanted to use the PX4 controller as a low level controller that was able to stabilize the vehicle. Therefore, instead of applying the control inputs directly, we decided to send the predicted velocities as control commands.

The MPC algorithm was implemented in a single ROS node that at each time step solves the optimization problem taking into account the current states of each drone, the current position and velocity of the target, and the flight plan previously described, resulting in a sequence of inputs and predicted states. From these, we obtained the desired velocities for each drone, which were then simultaneously
sent to the respective vehicle through velocity commands. In order for the algorithm to be able to run in real time, it is important that the MPC solver and the tracking controller run in parallel and that the computation of each solution does not exceed the sampling time.

In this section, we present a test made for a scenario with 2 drones, considering a similar trajectory plan of 52 seconds and the formation flight parameters presented in the Table 7.2. We also considered a sampling time of \( T_s = 0.4 \) seconds and a time horizon of \( T_h = 8 \) seconds, resulting in sequences of \( N = 20 \) elements. As for the cost matrices, we used the same scaling factors shown in Table 6.2 (without the \( z \) parameters) and computed the matrices as described in Section 6.4.

Table 7.2: Formation flight positions with 2 drones considered in the SITL simulations

<table>
<thead>
<tr>
<th></th>
<th>( \epsilon_t )</th>
<th>( \epsilon_x )</th>
<th>( \epsilon_y )</th>
<th>( \epsilon_z )</th>
</tr>
</thead>
<tbody>
<tr>
<td>d1</td>
<td>0</td>
<td>-5</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>d2</td>
<td>0</td>
<td>-5</td>
<td>-2</td>
<td>4</td>
</tr>
</tbody>
</table>

We obtained the simulation test showcased in the MPC trajectories video. From this, we can see that the simulation was successful, as feasible trajectories were computed in real time. The UAVs managed to follow the planned flight path and to exchange positions without colliding or being seen. These trajectories are plotted in Figure 7.6, where it is shown the reference trajectory for the MPC algorithm (dotted lines), the computed trajectory (dashed lines) and the actual trajectory of the drones (full lines). Here, we observe that the trajectories are not as smooth as the ones obtained in the previous test, especially during the transition flight phases and they do not follow the MPC’s initial reference as accurately as in the Matlab simulations.

![Figure 7.6: Trajectories obtained with the MPC SITL simulation. Note that the doted line corresponds to the MPC’s trajectory reference, the dashed lines to the computed desired trajectory and the full line to the real trajectory; additionally, the circles are the initial positions and the stars the final positions.](image)

In addition, we plotted the error of position between the desired and the real positions (Figure 7.7) and noticed that even though there are some fluctuations throughout the test, the errors are small, which means that the UAVs are able to go to the desired positions. Note that the errors increased during the...
flight transitions.

Furthermore, we checked if the constraints for collision avoidance and clear visibility were satisfied by plotting the distance and the angles in Figures 7.8 and 7.9. Once again, the distance between the drones was always above the minimum allowed distance, proving that there was no risk of collision. As for clear visibility, we noticed that at $t = 24$ seconds, the angle in the drone 1’s camera falls below the supposed field of view angle. Nevertheless, from the video, we can see that in spite of this, the camera did not see the other drone during these intervals.

![Figure 7.7: Position error in the MPC SITL simulation](image_url)

![Figure 7.8: Distance between drones in the MPC SITL simulation](image_url)
Finally, we analysed the computation time that it took to solve the optimization problem for each iteration in order to verify that this does not exceed the sampling time of the controller. Figure 7.10 showcases the sampling time $T_s$, the computation time to reach an MPC solution ($RTI$ time) and the computation time of each iteration ($iter$ time), where we notice that the computation times are always lower than $T_s$, making it possible to have a new desired state already computed at the beginning of every iteration.
Chapter 8

Conclusions

The goal of this dissertation was to design a trajectory planning and control strategy to solve the problem of replacement of drones that are following a target in formation flight. This was achieved in two ways: with a pre-planned trajectory generation method and with an online trajectory planning and control algorithm based on MPC.

In a first stage, we tackled the problem of planning and control separately, where we wanted to implement a method that was able to generate optimized trajectories and then to use these as references in a trajectory tracking controller. In Chapter 4, a tracking controller was presented to follow the desired trajectories, by receiving as references the position and yaw angle, as well as, their respective derivatives, and computing the errors in position, linear and angular velocity, and rotation in order to obtain a set of control inputs. At the end of the chapter, we presented some simulation tests and concluded that, despite being slow, the controller managed to follow the desired trajectories and to smooth out possible spikes and steep peaks in the reference.

Then, in Chapter 5, we implemented a strategy for trajectory planning, where a nonlinear optimization problem was formulated with non-convex constraints that imposed collision avoidance and clear visibility during the replacement process. The trajectories generated were already discretized with a fixed time step. We then presented two ways of solving the nonlinear problem: solving it directly with the Matlab’s fmincon or using an iterative algorithm based on SCP that would linearize the nonlinear equations around a previous solution at each iteration and solve the linearized problem with Cplex (which is faster than Matlab). Since the problem was non-convex, it was only possible to obtain local optimal solutions. In the end, after doing some simulation tests, we concluded that the solutions obtained with the SCP algorithm were more efficient as they took less computational time and had a lower value for the cost function. Additionally, in Chapter 7, we tested these trajectories in SITL simulations, where position and velocity waypoints were used to compute velocity commands that were sent to UAVs equipped with a PX4 controller. In the end, we concluded that by following these trajectories, the drones could execute the manoeuvres initially planned without any risk of collision and without being seen during the replacement process.

In a second stage, we looked into online planning and control approaches based on MPC, where
the idea was to implement an algorithm capable of generating desired positions and control inputs in real time. To do so, we implemented a Nonlinear Model Predictive Controller in the Matlab interface for ACADO, where we had to specify certain MPC parameters, such as sampling time and time horizon, and define a dynamic model of the system, as well as an objective function and a set of constraints. In this case, as it was possible to obtain the orientation of the drone and its camera throughout the flight, we made some changes in the optimization problem in order to include additional perception objective parameters that would optimize the position and orientation of the drone so that its camera would always be pointed at the target. This algorithm was tested in Matlab using the files generated by the ACADO toolkit, where we concluded that the algorithm performed well.

In addition, we implemented the NMPC algorithm in a ROS node, where additional modifications had to be made in order to test the algorithm in a SITL simulation, such as changing the control inputs to velocity commands and removing the perception objectives from the cost function. In the end, we verified that the drones were able to follow the flight plan and execute the necessary manoeuvres without colliding or being seen by their peers. However, the trajectories were oscillatory and slower than expected when transitioning between flight phases. Thus, we can conclude that the algorithm performs well, but needs to be improved in order to increase its robustness and to obtain smoother trajectories.

8.1 Future Work

In terms of future work, there are several lines of research that could be pursued.

On one hand, we could further explore the stability of the controller presented in Chapter 4 by doing a complete stability analysis, as well as, implementing it on an autopilot, which could then be tested with SITL simulations and with experimental trials in real UAVs. As for planning, we could investigate alternative convex constraints to the nonlinear collision avoidance and clear visibility constraints, in order to obtain optimal global solutions from a convex optimization problem.

On the other hand, we could improve the robustness of our MPC algorithm by looking into more precise integration schemes that account for quaternion rotations, by decreasing the reaction time during flight transitions or even by modifying the algorithm so that it implements path following, i.e. follow the target without time constraints. We could also explore decentralized MPC approaches, where each drone would solve its own MPC problem onboard and exchange data with others, thus increasing the number of drones that the algorithms could tackle without performance degradation.

In terms of experimental results, further work could be dedicated to testing the scenarios presented in Chapter 7 for the generated waypoint trajectories and the MPC algorithm in real flight experiments with 2 and/or 3 drones and obtain experimental results that validate the methods described, which, due to lack of time and unforeseen integration challenges in the necessary drones of the MULTIDRONE project, was not yet possible. Then if these are successful, the next step would be to use the MPC as a low level controller by sending commands of thrust and angular velocities directly to the autopilot.
Bibliography


Appendix A

Basic concepts of stability

In this Appendix, we summarize some basic concepts on stability and particularly on the Lyapunov Stability Theory, which were based on [8]. Let $S \subseteq \mathbb{R}^n$ be a set that contains the origin and $f : \mathbb{R}^n \to \mathbb{R}^n$ be a function such that $f(S) \subseteq S$. Considering the system

$$x^+ = f(x), \quad (A.1)$$

with $x \in \mathbb{R}^n$, there is an equilibrium point, $x_0$, at the origin for which $f(0) = 0$. Then, let $x_0 \in S$ and \{x_i\} $\subseteq S$ with $i \geq 0$ be the time step sequence that satisfies the model. The following properties may apply for an equilibrium point:

1. **Lyapunov stable in $S$** - if for any $\epsilon > 0$, there exists $\delta > 0$, such that

$$x_0 \in S \text{ and } \|x_0\| < \delta \rightarrow \|x_i\| < \epsilon \text{ for all } i \geq 0.$$

2. **Attractiveness in $S$** - if there exists $\eta > 0$ such that

$$x_0 \in S \text{ and } \|x_0\| < \eta \rightarrow \lim_{i \rightarrow \infty} x_i = 0.$$

3. **Globally attractive in $S$** - if

$$x_0 \in S \rightarrow \lim_{i \rightarrow \infty} x_i = 0.$$

4. **Asymptotically stable in $S$** - if it is Lyapunov stable and attractive in $S$.

5. **Exponentially stable in $S$** - if there are constants $\theta > 0$ and $\rho \in (0, 1)$ such that

$$x_0 \in S \rightarrow \|x_i\| \leq \theta \|x_0\|\rho^i \text{ for all } i \geq 0.$$

There are 3 theorems that may be useful when proving stability:

1. **Theorem**: Attractivity in $S$ - which states that if there exists a Lyapunov function $V : S \Rightarrow [0, \infty)$ that satisfies the following conditions:
Appendix A. Basic concepts of stability

a. $V$ decreases along the trajectories in Equation (A.1), i.e. there exists a continuous function \( \gamma : [0, \infty) \Rightarrow [0, \infty), \gamma(t) > 0, \forall t > 0 \) such that

\[
V(f(x)) - V(x) \leq -\gamma(||x||), \forall x \in S;
\]

b. For every unbounded sequence \( \{y_i\} \subset S \), there exists \( j \) such that

\[
\lim_{i \to \infty} \sup V(y_i) > V(y_j).
\]

Then, we can conclude that \( \forall x_0 \in S \), the resulting sequence \( \{x_i\} \), with \( i \geq 0 \), from Equation (A.1) is such that \( \lim_{i \to \infty} x_i = 0 \), which means that if \( 0 \in S \), then the origin is globally attractive in \( S \).

2. **Theorem:** Lyapunov Stability - which states that if \( S \in \mathbb{R}^n \) contains an open neighbourhood of the origin \( N_\eta(0) = \{x \in \mathbb{R}^n : ||x|| < \eta\} \) and if there exists a Lyapunov function \( V : S \Rightarrow [0, \infty), V(0) = 0 \) that satisfies the following conditions:

a. \( V \) is continuous in \( N_\eta(0) \);

b. If \( \{y_k\} \subset S \) is such that \( \lim_{k \to \infty} V(y_k) = 0 \), then \( \lim_{k \to \infty} y_k = 0 \);

c. \( V(f(x)) - V(x) \leq 0, \forall x \in N_\eta(0) \).

Then, the origin is a Lyapunov stable equilibrium point of (A.1).

3. **Theorem:** Exponential Stability - which states that if \( S \in \mathbb{R}^n \) contains a nonzero element and if there exists a Lyapunov function \( V : S \Rightarrow \mathbb{R} \) and positive constants \( a, b, c \) and \( \sigma \) that satisfy the following conditions:

a. \( a ||x||^\sigma \leq V(x) \leq b ||x||^\sigma, \forall x \in S \);

b. \( V(f(x)) - V(x) \leq -x ||x||^\sigma, \forall x \in S \).

Then, if \( 0 \in S \), the origin is exponentially stable in \( S \).

Therefore, the goal should be to find a Lyapunov function with the properties described in the theorems either for asymptotic stability or exponential stability. In an MPC algorithm, this function is usually defined as the objective function of the receding horizon optimization problem.
Appendix B

Alternative planning approaches

In this chapter, we will present other planning methods that were also implemented in order to solve the same replacement problem. The first method is built on the works in [11, 20, 32], in which the collision avoidance constraints are implemented with mixed-integer variables: continuous and binary, making it possible for the problem to only have linear constraints. The trajectories generated are polynomials with a pre-defined degree, resulting in smooth trajectories.

The second method is a combined approach, in which the method described in Chapter 5 is used to generate trajectories, but the collision avoidance and clear visibility constraints are implemented with linear mixed-integer variables. This means that the trajectory dynamics, objective function and linear equalities and inequalities are the same as the ones described in Chapter 5, but the collision avoidance and clear visibility constraints are replaced by the ones in the second method.

Since both methods are formulated as Mixed-integer Quadratic optimization Problems, these will be solved with the Cplex’s cplexmiqp function.

B.1 Problem with mixed-integer constraints

This method solves the problem of trajectory optimization for multiple vehicles using polynomial trajectories and mixed-integer constraints.

B.1.1 Trajectory dynamics

The trajectories \( \sigma \) are characterized by 4 components: the position in the world frame axis \( p_x, p_y \) and \( p_z \) and the yaw angle \( \psi \), which are written as piecewise polynomial functions of order \( N \) over \( M \) time intervals, such that

\[
\sigma(t) = \begin{cases} 
\sum_{n=0}^{N} \sigma_n t^n, & t_0 \leq t < t_1 , \\
\sum_{n=0}^{N} \sigma_n t^n, & t_1 \leq t < t_2 , \\
\vdots \\
\sum_{n=0}^{N} \sigma_n t^n, & t_{M-1} \leq t \leq t_M . 
\end{cases}
\]  (B.1)
Appendix B. Alternative planning approaches

This system of equations must be repeated for each component of the trajectory ($p_x$, $p_y$, $p_z$ and $\psi$), therefore defining the trajectory of 1 vehicle over a set of $M$ time intervals. When there is more than 1 drone in the system, each should have its own system of equations with $p_x$, $p_y$, $p_z$ and $\psi$.

Note that some numerical issues may occur when dealing with complex trajectories or when these are extended over a long period of time, due to the terms $t^n$. These problems will be discussed in Subsection B.1.6.

B.1.2 Optimization variable

The optimization variable, $\chi$, is a vector of mixed-integer variables: continuous and binary. The continuous variables ($\chi_{\text{cont}}$) are related to the polynomial coefficients of each of the vehicles. Considering that $q$ is one of the $Q$ vehicles in the system, $n$ is an intermediate degree of the polynomial of $N^{th}$ degree with $0 \leq n \leq N$ and $m$ is one interval of a trajectory of $M$ intervals such that $1 \leq m \leq M$, we may have the following coefficients

$$
\sigma_{nm}^q = [p_{xnm}^q, p_{ynm}^q, p_{znm}^q, \psi_{nm}^q]^T.
$$

Thus, for the full trajectory dynamics, there will be $4(N + 1)MQ$ continuous variables, where $N$ is the degree of the polynomials, $M$ is the number of intervals and $Q$ is the number of drones. As for the binary variables, these are used to ensure collision avoidance ($\chi_{\text{coll}}$), which will be explained in Subsection B.1.4.

B.1.3 Objective function

The objective function consists on the integral of the $k_p^{th}$ derivative of the position squared and the $k_\psi^{th}$ derivative of the yaw angle squared, such that

$$
f_0 = \int_{t_0}^{t_M} \mu_p \left\| \frac{d^k_p \mathbf{P}}{dt^k_p} \right\|^2 dt + \mu_\psi \frac{d^k_\psi \psi}{dt^k_\psi}^2 dt. \quad (B.2)
$$

The constants $k_p$ and $k_\psi$ are related to the tracking controller described in Chapter 4, where we saw that the control inputs depended of the $4^{th}$ derivative of the position and on the $2^{nd}$ derivative of the yaw angle. Therefore, we can choose $k_p = 4$ and $k_\psi = 2$.

In order to formulate the problem as a quadratic program and to solve it directly using the same tools as before (Matlab and Cplex), the previous objective function can be rewritten in the quadratic form

$$
f_0 = \mathbf{c}^T \mathbf{H} \mathbf{c} + \mathbf{f}^T \mathbf{c}. \quad (B.3)
$$

Taking into account the trajectory dynamics in (B.1), the objective function can be manipulated as
demonstrated below.

$$f_0 = \int_{t_0}^{t_M} \mu_p \left| \frac{d^k p}{dt^k} \right|^2 + \mu_\psi \frac{d^k \psi}{dt^k} \, dt$$

$$= \int_{t_0}^{t_1} \mu_p \left| \frac{d^k P_1}{dt^k} \right|^2 + \mu_\psi \frac{d^k \psi_1}{dt^k} \, dt + \int_{t_1}^{t_2} \mu_p \left| \frac{d^k P_2}{dt^k} \right|^2 + \mu_\psi \frac{d^k \psi_2}{dt^k} \, dt$$

$$+ \ldots + \int_{t_{M-1}}^{t_M} \mu_p \left| \frac{d^k P_M}{dt^k} \right|^2 + \mu_\psi \frac{d^k \psi_M}{dt^k} \, dt$$

$$= I_1 + I_2 + \ldots + I_M.$$

Let $m$ be an interval such that $1 \leq m \leq M$, then

$$I_m = \int_{t_{m-1}}^{t_m} \mu_p \left| \frac{d^k p_{xm}}{dt^k} \right|^2 + \mu_\psi \frac{d^k \psi_{xm}}{dt^k} \, dt$$

$$= \int_{t_{m-1}}^{t_m} \mu_p \left| \frac{d^k p_{xm}^2}{dt^k} \right|^2 + \mu_\psi \frac{d^k \psi_{xm}^2}{dt^k} \, dt$$

$$= \mu_p \left( \int_{t_{m-1}}^{t_m} \frac{d^k p_{xm}^2}{dt^k} \, dt + \int_{t_{m-1}}^{t_m} \frac{d^k \psi_{xm}^2}{dt^k} \, dt \right)$$

Replacing $k_r$ by its numerical value and $p_x$ by its polynomial expression, we obtain

$$I_{mp_x} = \int_{t_{m-1}}^{t_m} \frac{d^4 p_{xm}}{dt^4} \, dt$$

$$= \int_{t_{m-1}}^{t_m} \frac{d^4 \sum_{n=0}^{N} p_{xm}^n}{dt^4} \, dt$$

$$= \int_{t_{m-1}}^{t_m} \left( \sum_{n=0}^{N} \frac{d^4 p_{xm}^n}{dt^4} \right) \, dt$$

$$= \int_{t_{m-1}}^{t_m} \left( \sum_{n=4}^{N} \frac{d^4 p_{xm}^n}{dt^4} \right) \, dt$$

$$= \int_{t_{m-1}}^{t_m} \sum_{n=4}^{N} \left( n(n-1)(n-2)(n-3)p_{xm}^n t^{n-4} \right) \, dt$$

$$= \int_{t_{m-1}}^{t_m} \sum_{n=4}^{N} \left( n(n-1)(n-2)(n-3)p_{xm}^n t^{n-4} \right) \, dt$$

$$+ 2 \sum_{n=4}^{N} \sum_{j=4}^{n-4} n(n-1)(n-2)(n-3)p_{xm}^n t^{n-4} j(j-1)(j-2)(j-3) p_{xj} t^{j-4} \, dt$$

$$= \sum_{n=4}^{N} \int_{t_{m-1}}^{t_m} \left( n(n-1)(n-2)(n-3)p_{xm}^n t^{n-4} \right) \, dt$$

$$+ 2 \sum_{n=4}^{N} \sum_{j=4}^{n-4} \int_{t_{m-1}}^{t_m} n(n-1)(n-2)(n-3)p_{xm}^n t^{n-4} j(j-1)(j-2)(j-3) p_{xj} t^{j-4} \, dt$$

$$= \sum_{n=4}^{N} n^2(n-1)^2(n-2)^2(n-3)^2 p_{xm}^n t^{2n-7} \left| t_m \right| t_m$$

$$+ \sum_{n=4}^{N} \sum_{j=4}^{n-4} n(n-1)(n-2)(n-3)p_{xm}^n t^{n+j-7} \left| t_m \right| t_m.$$
Similar expressions can be derived for the integrals $I_{mp_1}$ and $I_{mp_2}$:

$$I_{mp_1} = \sum_{n=4}^{N} n^2(n-1)^2(n-2)^2(n-3)^2 p_{ymn}^2 \int_{t_{m-1}}^{t_m} \frac{t^{2n-7}}{2n-7} dt$$

$$+ 2 \sum_{n=4}^{N} \sum_{j=4}^{n-1} n(n-1)(n-2)(n-3)p_{ymn} j(j-1)(j-2)(j-3)p_{ymj} \int_{t_{m-1}}^{t_m} \frac{t^{n+j-7}}{n+j-7} dt,$$

$$I_{mp_2} = \sum_{n=4}^{N} n^2(n-1)^2(n-2)^2(n-3)^2 p_{zmn}^2 \int_{t_{m-1}}^{t_m} \frac{t^{2n-7}}{2n-7} dt$$

$$+ 2 \sum_{n=4}^{N} \sum_{j=4}^{n-1} n(n-1)(n-2)(n-3)p_{zmn} j(j-1)(j-2)(j-3)p_{zjn} \int_{t_{m-1}}^{t_m} \frac{t^{n+j-7}}{n+j-7} dt.$$

As for the yaw angle, replacing $k_\psi$ by its numerical value and $\psi$ by its polynomial expression, we obtain

$$I_{m\psi} = \int_{t_{m-1}}^{t_m} \frac{d^2 \psi_m}{dt^2}^2 dt$$

$$= \int_{t_{m-1}}^{t_m} \frac{\sum_{n=0}^{N} \psi_{nm}^2}{dt^2} dt$$

$$= \int_{t_{m-1}}^{t_m} (\sum_{n=0}^{N} \frac{d^2 \psi_{nm}}{dt^2})^2 dt$$

$$= \int_{t_{m-1}}^{t_m} \left( \sum_{n=2}^{n-1} n(n-1)\psi_{nm}^{n-2} \right)^2 dt$$

$$= \int_{t_{m-1}}^{t_m} \left( \sum_{n=2}^{n-1} (n(n-1)\psi_{nm}^{n-2})^2 + 2 \sum_{n=2}^{n-1} \sum_{j=2}^{n-1} n(n-1)\psi_{nm}^{n-2} j(j-1)\psi_{jm}^{j-2} dt \right.$$}

$$= \sum_{n=2}^{n-1} \int_{t_{m-1}}^{t_m} (n(n-1)\psi_{nm}^{n-2})^2 dt + 2 \sum_{n=2}^{n-1} \sum_{j=2}^{n-1} \int_{t_{m-1}}^{t_m} n(n-1)\psi_{nm}^{n-2} j(j-1)\psi_{jm}^{j-2} dt$$

$$= \sum_{n=2}^{n-1} n^2(n-1)^2\psi_{nm}^{2n-3} \int_{t_{m-1}}^{t_m} + 2 \sum_{n=2}^{n-1} \sum_{j=2}^{n-1} n(n-1)\psi_{nm} j(j-1)\psi_{jm} \int_{t_{m-1}}^{t_m} \frac{t^{n+j-3}}{n+j-3} dt.$$

By placing the final expressions of $I_{mp_1}$, $I_{mp_2}$, $I_{mp_3}$ and $I_{m\psi}$ into the equation (B.5), we get an expression for $I_m$. If we do this for all $M$ intervals and place them in (B.4), we finally obtain a quadratic formula for the objective function.

In case there is more than 1 vehicle in the system, the objective function must be expanded. Since we can have different drones (with different sizes and masses), different costs may be assigned to the vehicles, so that bigger and heavier drones have a higher cost and smaller drones have a lower cost. In this way, considering that $\mu_i$ and $H_i$ are the cost and the individual quadratic cost matrix for drone $i$, with $i = 1, ..., Q$, and that there are $Q$ drones in the system, the new quadratic cost matrix $H$ can be written as:

$$H = \text{diag}(\mu_1 H_1, \mu_2 H_2, ..., \mu_Q H_Q).$$

(B.10)
B.1. Problem with mixed-integer constraints

Note that:

• The dimensions of $H$ and $f$ are: $H \in \mathbb{R}^{\text{dim}(\chi) \times \text{dim}(\chi)}$ and $f \in \mathbb{R}^{\text{dim}(\chi)}$;

• All terms are quadratic, which means that $f$ is a zero matrix;

• the terms associated with the binary variables are zero.

B.1.4 Constraints

The optimization problem contains three types of constraints: linear equalities, linear inequalities and mixed-integer inequalities, that are described in the following pages.

The **linear equalities** are defined by the matrices $A_{eq}$ and $b_{eq}$, and include the following:

1. Continuity between intervals - the trajectories need to be continuous in between intervals. To do so, we must enforce continuity of the position and yaw angle, as well as of their first $4^{th}$ and $2^{nd}$ derivatives respectively. For the coordinate $p_x$, considering that $q$ is one of the vehicles in a system of $Q$ vehicles and $m$ is one time interval such that $2 \leq m \leq M$, these constraints can be formulated as

   \[
   p_{q}^{x}(t_{m-1}) = p_{q}^{x}(t_{m-1}) \Leftrightarrow \sum_{n=0}^{N} p_{q}^{x}(t_{m-1}) t_{m-1}^{n-1} = \sum_{n=0}^{N} p_{q}^{x}(t_{m-1}) t_{m-1}^{n-1} \\
   \Leftrightarrow \sum_{n=1}^{N} t_{m-1}^{n-1} (p_{q}^{x} t_{m-1}^{n-1} - x_{q}^{n}(m-1)) = 0, \tag{B.11}
   \]

   \[
   \frac{d}{dt} p_{q}^{x}(t_{m-1}) = \frac{d}{dt} p_{q}^{x}(t_{m-1}) \Rightarrow \sum_{n=1}^{N} n p_{q}^{x} t_{m-1}^{n-2} = \sum_{n=1}^{N} n p_{q}^{x} t_{m-1}^{n-2} \\
   \Rightarrow \sum_{n=0}^{N} n t_{m-1}^{n-1} (p_{q}^{x} t_{m-1}^{n-1} - p_{q}^{x}(m-1)) = 0, \tag{B.12}
   \]

   \[
   \frac{d^2}{dt^2} p_{q}^{x}(t_{m-1}) = \frac{d^2}{dt^2} p_{q}^{x}(t_{m-1}) \Rightarrow \sum_{n=2}^{N} n(n-1) p_{q}^{x} t_{m-1}^{n-3} = \sum_{n=2}^{N} n(n-1) p_{q}^{x}(m-1) t_{m-1}^{n-3} \\
   \Rightarrow \sum_{n=2}^{N} n(n-1) t_{m-1}^{n-2} (p_{q}^{x} t_{m-1}^{n-2} - x_{q}^{n}(m-1)) = 0, \tag{B.13}
   \]

   \[
   \frac{d^3}{dt^3} p_{q}^{x}(t_{m-1}) = \frac{d^3}{dt^3} p_{q}^{x}(t_{m-1}) \Rightarrow \sum_{n=3}^{N} n(n-1)(n-2) p_{q}^{x} t_{m-1}^{n-4} = \sum_{n=3}^{N} n(n-1)(n-2) p_{q}^{x}(m-1) t_{m-1}^{n-4} \\
   \Rightarrow \sum_{n=3}^{N} n(n-1)(n-2) t_{m-1}^{n-3} (p_{q}^{x} t_{m-1}^{n-3} - p_{q}^{x}(m-1)) = 0, \tag{B.14}
   \]
\[
\frac{d^4}{dt^4} p_{xn}^q(t_{m-1}) = \frac{d^4}{dt^4} p_{x(m-1)}^q(t_{m-1})
\]
\[
\Leftrightarrow \sum_{n=4}^{N} n(n-1)(n-2)(n-3)p_{ynm}^q t_{n-4}^{n-4} = \sum_{n=4}^{N} n(n-1)(n-2)(n-3)p_{znm}^q t_{n-4}^{n-4} \quad (B.15)
\]
\[
\Leftrightarrow \sum_{n=4}^{N} n(n-1)(n-2)(n-3)t_{n-4}^{n-4}(p_{xn}^q - p_{xznm}^q) = 0.
\]

Similar constraints are also imposed for the other coordinates \( p_y \) and \( p_z \), where the terms \( p_{ynm} \) and \( p_{xznm} \) are replaced by \( p_{ynm} \) and \( p_{ynm} \), and \( p_{znm}^q \) and \( p_{znm}^q \). As for the yaw angle, we can apply the first 3 constraints \( (B.11), (B.12) \) and \( (B.13) \), where the terms \( p_{ynm} \) and \( p_{xznm} \) are replaced by \( \psi_{ym} \) and \( \psi_{n}(n-1) \). In the end, the number of these type of constraints is \( 5(M-1)Q \) for each direction axis and \( 3(M-1)Q \) for the yaw angle.

2. Initial and final derivatives - in order to assure that all vehicles begin at a resting position, we impose that the initial and final derivatives of the trajectories are zero, which translates into the following equations:

\[
\frac{d}{dt} p_x^q(t_0) = 0, \quad (B.16) \quad \frac{d}{dt} p_x^q(t_M) = 0, \quad (B.20)
\]
\[
\frac{d}{dt} p_y^q(t_0) = 0, \quad (B.17) \quad \frac{d}{dt} p_y^q(t_M) = 0, \quad (B.21)
\]
\[
\frac{d}{dt} p_z^q(t_0) = 0, \quad (B.18) \quad \frac{d}{dt} p_z^q(t_M) = 0, \quad (B.22)
\]
\[
\frac{d}{dt} \psi^q(t_0) = 0, \quad (B.19) \quad \frac{d}{dt} \psi^q(t_M) = 0. \quad (B.23)
\]

The number of constraints of this type is simply \( 4 \times 2Q \).

3. Initial and final states - in the flight plans described in Figure 1.4, it was stated that the vehicles should stay in their initial and final positions (and yaw angles) during the intervals before takeoff \((T_f)\) and after landing \((T_e)\). To do so, we can divide those intervals into smaller time steps, considering that \( t_i \), with \( t_0 \leq t_i \leq t_f \), is one of the intermediate time steps before takeoff and \( t_f \), with \( t_F \leq t_f \leq t_M \), is another intermediate time step after landing. We can then formulate the following constraints:

\[
p_x^q(t_i) = p_x^q(t_0), \quad (B.24) \quad p_x^q(t_f) = p_x^q(t_M), \quad (B.28)
\]
\[
p_y^q(t_i) = p_y^q(t_0), \quad (B.25) \quad p_y^q(t_f) = p_y^q(t_M), \quad (B.29)
\]
\[
p_z^q(t_i) = p_z^q(t_0), \quad (B.26) \quad p_z^q(t_f) = p_z^q(t_M), \quad (B.30)
\]
\[
\psi^q(t_i) = \psi^q(t_0), \quad (B.27) \quad \psi^q(t_f) = \psi^q(t_M). \quad (B.31)
\]

where \( p_x^q, p_y^q, p_z^q, \psi^q, p_x^q, p_y^q, p_z^q, \psi^q \), and \( \psi^q \) are the desired initial and final positions and yaw angles.
The number of constraints of this type depends on the duration of the intervals and the number of vehicles. Considering a general scenario with $Q$ drones and a flight plan with 12 intervals ($M = 12$), the initial state constraints (B.24), (B.25), (B.26) and (B.27) should be imposed during the interval $T_1$ ($t_0 \leq t_i \leq t_1$) for the first $Q - 1$ drones and during $T_1$ and $T_2$ ($t_0 \leq t_i \leq t_2$) for the last drone (drone $q$). As for the final state constraints, equations (B.28), (B.29), (B.30) and (B.31) should be applied during $T_{10}$, $T_{11}$ and $T_{12}$ ($t_9 \leq t_f \leq t_{12}$) for the drone that was replaced (drone $i$) and during $T_{11}$ and $T_{12}$ ($t_{10} \leq t_f \leq t_{12}$) for the other $Q - 1$ drones. Suppose that $N_P(T_j)$ is the number of time steps inside an interval $T_j$. Then, the number of constraints is

$$(Q - 1)4N_P(T_1) + 4 \sum_{j=1}^{2} N_P(T_j) + 4 \sum_{j=10}^{12} N_P(T_j) + (Q - 1)4 \sum_{j=11}^{12} N_P(T_j).$$

4. **Formation flight following target** - we also want to ensure that the drones follow the target in formation flight. Therefore, it is necessary to divide the corresponding intervals into smaller time steps and to define the formation positions with $\epsilon_{ij}$, $\epsilon_{xj}$, $\epsilon_{yj}$, and $\epsilon_{zj}$, with $j = 1, ..., Q$ as in Figure 1.3. Considering that $p_T$ is the position of the target, $W R_T(t_k)$ is the Rotation matrix of the target described in the world frame as previously defined in (5.18), and $t_k$ is an intermediate time interval in interval $T_m$, such that $1 \leq m \leq M$, then a generic position $j$ of the formation flight is given by

$$p_{\text{form}_j}(t_k) = p_T(t_k - \epsilon_{ij}) + W R_T(t_k - \epsilon_{ij})e_{pj}.$$  \hspace{1cm} (B.32)

where $e_{pj} = [\epsilon_{xj}, \epsilon_{yj}, \epsilon_{zj}]^T$ is a vector with the formation positions.

To ensure that a vehicle $i$, with $i = 1, ..., Q$, follows the target in position $j$ in formation flight, the following constraint must be implemented:

$$p_i(t_k) = p_{\text{form}_j}(t_k).$$  \hspace{1cm} (B.33)

The number of constraints also depends on the number of time steps and on the number of vehicles. For the general scenario with $Q$ drones described in Figure 1.4b, drone $i$ should be in $p_{\text{form}_i}$ during the intervals $T_2$, $T_3$ and $T_4$ ($t_1 \leq t_k \leq t_4$) and in $p_{\text{form}_q}$ during $T_8$ ($t_7 \leq t_k \leq t_8$); then drone $q$ should be in $p_{\text{form}_q}$ during the intervals $T_4$ ($t_3 \leq t_k \leq t_4$) and in $p_{\text{form}_i}$ during $T_8$, $T_9$ and $T_{10}$ ($t_7 \leq t_k \leq t_{10}$); and finally the other $Q - 2$ drones should be their respective positions in formation flight during $T_2 - T_{10}$ ($t_1 \leq t_k \leq t_{10}$). Thus, the number of constraints is

$$\left[3 \sum_{j=2}^{4} N_P(T_j) + 3N_P(T_8)\right] + \left[3N_P(T_4) + 3 \sum_{j=8}^{10} N_P(T_j)\right] + (Q - 2)3 \sum_{j=2}^{10} N_P(T_j).$$

Then, the **linear inequalities** are characterized by the matrices $A_{in}$ and $b_{in}$ and include the following:

1. **Constraint for the floor** - it imposes that the vehicles are always on or above the ground and can be written as

$$p_{z_i} \geq 0,$$  \hspace{1cm} (B.34)
Appendix B. Alternative planning approaches

with \( i = 1, ..., Q \). This results in \( Q \sum_{i=1}^{M} N_p(T_j) \) constraints. In Section 5.3, this constraint was included in the dynamics limitation constraint, where we imposed the maximum and minimum values allowed for the position, velocity, acceleration and jerk. These constraints could also be implemented with this method, if necessary.

2. **Minimum height during flight** - also implemented in Section 5.3 and it guarantees that during flight the vehicles are always above a minimum height \( H_{\text{min}} \) from the ground. For any vehicle \( i \), with \( i = 1, ..., Q \), and in the intermediate interval \( t_k \), with \( t_0 \leq t_k \leq t_M \), this constraint can be formulated as

\[
p_i z(t_k) \geq H_{\text{min}} \tag{B.35}
\]

For a general scenario with \( Q \) drones, this constraint should be imposed during the intervals \( T_2 - T_8 \) \((t_1 \leq t_k \leq t_8)\) for drone \( i \), during \( T_4 - T_{10} \) \((t_3 \leq t_k \leq t_{10})\) for drone \( q \) and during \( T_2 - T_{10} \) \((t_1 \leq t_k \leq t_{10})\) for the other \( Q - 2 \) drones. Therefore, the number of constraints of this type is

\[
8 \sum_{j=2}^{8} N_p(T_j) + 10 \sum_{j=4}^{10} N_p(T_j) + (Q - 2) \sum_{j=2}^{10} N_p(T_j).
\]

3. **Clear visibility** - in order to ensure that the problem is still linear, the nonlinear constraint in (5.32) was replaced by an approximate linear condition, where we made the assumption that each vehicle has the same orientation as the target and the orientation of the camera is based on the formation parameters \( \epsilon_{tj}, \epsilon_{xj}, \epsilon_{yj}, \text{and} \epsilon_{zj} \), with \( j = 1, ..., Q \). Therefore, if \( W_{RT}(t_k) \) is the Rotation matrix of the target described in the world frame for an intermediate interval \( t_k \), defined in (5.18), and \( W_{RB}(t_k) \) is the Rotation matrix of the body described in the world frame for \( t_k \), which we assume that \( W_{RB}(t_k) = W_{RT}(t_k) \), then, the Rotation matrix of the camera described in the body frame, denoted by \( W_{RC} = [x_C, y_C, z_C] \), can be determined as

\[
\begin{align*}
x_C &= \frac{\epsilon_{pj}}{||\epsilon_{pj}||}, \\
\alpha &= \arctan x_{Cy} \quad x_{Cx}, \\
\beta &= \arcsin x_{Cz}, \\
z_C &= [-\cos(\alpha) \sin(\beta), -\sin(\alpha) \sin(\beta), -\cos(\beta)], \\
y_C &= z_C \times x_C
\end{align*}
\tag{B.36}
\]

where \( \times \) denotes the cross product.

Considering the situation where drone \( q \) cannot be seen by drone \( i \), this constraint imposes that the former must be behind the camera of the latter and it does so by comparing the vector to which the camera is pointed \( (a) \) and the vector between the position of drone \( q \) and the position of the
B.1. Problem with mixed-integer constraints

drone $i$’s camera described in its camera frame (b). These vectors can be written as

$$a = [1, 0, 0]^T$$

$$b = \left( W_{Bi}(t_k - \epsilon_{ti})^{Bi} R_{Ci} \right)^T \left( p_q(t_k) - (p_i(t_k) + W_{Bi}(t_k - \epsilon_{ti})^{Bi} p_{Ci}) \right),$$

(B.37)

where $p_q$ and $p_i$ are the positions of drone $i$ and $j$ in the world frame and $^{Bi} p_{Ci}$ is the position of the camera of drone $i$ in its body frame. To have clear visibility, we impose that the angle between the vectors $a$ and $b$ should be greater than $90^\circ$, which is equivalent to

$$a^T b \leq 0.$$  

(B.38)

The number of constraints of this type is $(Q - 1) \sum_{i=5}^{6} N_P(T_i) + (Q - 1) \sum_{i=6}^{7} N_P(T_i)$. Note that this constraint is simply an approximation of the nonlinear one in (5.32), as there are no guarantees that the target is in the field of view pre-defined by the camera nor that the orientation of the vehicle is the same as the target. These were mere assumptions that had to be made in order to obtain linear conditions.

Finally, we considered mixed-integer constraints, that were used to ensure collision avoidance between vehicles as an alternative to the nonlinear constraint described in (5.3). This constraint imposes that the vehicles are separated by a minimum safety distance in at least one of its direction axis. Thus, for any drones $i$ and $q$ in a system of $Q$ drones with $i = 1, ..., Q$, $j = 1, ..., Q$ and $i \neq q$, the following conditions should be satisfied for each time step $t_k \in [t_0, t_M]$:

$$p^I_j(t_k) - p^Q_j(t_k) \geq d_{x_iq}$$

or

$$p^Q_j(t_k) - p^I_j(t_k) \geq d_{x_qi}$$

or

$$p^I_y(t_k) - p^Q_y(t_k) \geq d_{y_iq}$$

or

$$p^Q_y(t_k) - p^I_y(t_k) \geq d_{y_qi}$$

or

$$p^I_z(t_k) - p^Q_z(t_k) \geq d_{z_iq}$$

or

$$p^Q_z(t_k) - p^I_z(t_k) \geq d_{z_qi},$$

(B.39)

where $d_{x_iq}, d_{x_qi}, d_{y_iq}, d_{y_qi}, d_{z_iq}, d_{z_qi}$ are the safety distances.

If drones $i$ and $q$ are similar drones (in size and weight), then $d_{j_iq} = d_{j_qi}$, where $j$ is the direction axis $j \in \{x, y, z\}$. However, if, for instance, drone $i$ is bigger than drone $q$, then $d_{ziq} > d_{zqi}$. In [11], it is stated that quadrotors must avoid flying in the downwash of similar-sized or larger quadrotors because of the decrease in tracking performance and possible instability. This means that drone $i$ could fly under drone
Appendix B. Alternative planning approaches

$q$ but not the opposite. This constraint can then be formulated with mixed-integer variables, such as

$$x^i(t_k) - x^q(t_k) + V b_{xiqk} \geq d_{xiq},$$
$$x^q(t_k) - x^i(t_k) + V b_{xqik} \geq d_{xqi},$$
$$y^i(t_k) - y^q(t_k) + V b_{yiqk} \geq d_{yiq},$$
$$y^q(t_k) - y^i(t_k) + V b_{yiqk} \geq d_{yqi},$$
$$z^i(t_k) - z^q(t_k) + V b_{ziqk} \geq d_{ziq},$$
$$z^q(t_k) - z^i(t_k) + V b_{zqik} \geq d_{zqi},$$
$$b_{jiqk} = 0 \text{ or } 1, \forall j = x, y, z,$$
$$b_{jqik} = 0 \text{ or } 1, \forall j = x, y, z,$$
$$\sum_{i=1}^{6} b_{jiqk} \leq 5,$$

where $t_k$ is an intermediate interval, $b_{jiqk}$ is a binary variable associated with the collision avoidance constraint between drones $i$ and $q$ in the direction axis $j$ at $t_k$ and $V$ is a large number.

This should be repeated for all $Q$ vehicles in the system and for all of the $K$ intermediate time intervals. Thus, the number of binary variables necessary is $3Q(Q-1)K$ and the total number of constraints is $3Q(Q-1)K + 3Q(Q-1)K/6$.

### B.1.5 Analysis of the optimization problem

As mentioned before, we want the problem to be convex, which means that the objective function should be convex, the equality constraints linear and the inequality constraints should represent convex sets. Similarly to the method in Chapter 5, the objective function is also a quadratic function ($f_0 = c^THc + f^Tc$). Therefore, to preserve convexity, $H$ should be a symmetric semi-positive definite matrix, which is indeed the case. As for the set of constraints, all of the equalities and inequalities previously described were linear, thus, forming a convex polyhedron. Hence, we can conclude that this optimization problem is convex, which means that the trajectories generated consist of global optimal solutions.

### B.1.6 Problems and limitations

After doing several tests and experimenting with different trajectories and parameters, we can conclude that this method is not adequate for our problem, as it presents several limitations related with numerical issues. Since the trajectories depend directly on the term $t^n$, the problem becomes unsolvable when using polynomials of high degrees and for trajectories extended for long periods of time, as it exceeds the precision of the computer.

One solution for this problem, suggested in [20], is to scale it in time and space with two parameters $\alpha$ and $\beta$. To do so, the variables $p_x(t)$, $p_y(t)$, $p_z(t)$ and $\psi(t)$ are transformed into the nondimensional
variables $\tilde{p}_x(\tau), \tilde{p}_y(\tau), \tilde{p}_z(\tau)$ and $\tilde{\psi}(\tau)$ with $\tau$ being the nondimensional time, such that

\begin{align*}
t &= \alpha \tau \\
p_x(t) &= p_x(\alpha \tau) = \beta_{r1} + \beta_{r2} \tilde{p}_x(\tau), \quad (B.41) \\
p_y(t) &= p_y(\alpha \tau) = \beta_{r1} + \beta_{r2} \tilde{p}_y(\tau), \\
p_z(t) &= p_z(\alpha \tau) = \beta_{r1} + \beta_{r2} \tilde{p}_z(\tau), \\
\psi(t) &= \psi(\alpha \tau) = \beta_{\psi1} + \beta_{\psi2} \tilde{\psi}(\tau).
\end{align*}

The reformulation of the optimization problem does not change significantly. The quadratic cost function $H$ and the constraints are formulated in the same way. The only difference is that all of the parameters that depend on time and space need to be scaled to their nondimensional form in the beginning with $\alpha$ and $\beta$ and then converted in the end to their original scale.

The scaling solution makes it possible to have longer trajectories. However, these cannot be too complex, since the time steps have to be at least 1 second. This solution is not efficient, because as we are scaling the components, the values may become close to zero, exceeding again the precision of the computer.

Because of these limitations, it was not possible to obtain optimal feasible solutions capable of solving the problem at hand. Nevertheless, we were able to obtain a solution with numerical issues for a scenario with two drones following a straight path by making multiple compromises when it comes to the duration of the intervals, degree of the polynomials, scaling factors and other parameters. To do so, we used the same formation flight positions defined in Table 5.1, but had to consider a shorter trajectory with the following intervals:

\begin{center}
\begin{tabular}{cccccccccccc}
T_1 & T_2 & T_3 & T_4 & T_5 & T_6 & T_7 & T_8 & T_9 & T_{10} & T_{11} & T_{12} \\
0 & 2 & 7 & 9 & 14 & 16 & 18 & 21 & 26 & 28 & 33 & 35 & 36
\end{tabular}
\end{center}

The test was conducted on the same Lenovo Laptop running Ubuntu 16.04 and equipped with an Intel Core i7-7700HQ CPU @2.80GHz and 16,00GB of RAM, where Matlab R2018a for Linux was running. In the end, we obtained the trajectories represented in Figure B.1, where we can see that the vehicles are able to follow the target in formation flight. However, during the exchange of positions they do not appear to respect the collision avoidance and clear visibility constraints. To further verify if these conditions are satisfied, we plotted the distance between drones along the trajectory (Figure B.2) and the angles between the target and the other drones as explained in the other simulation tests (Figure B.3). In both situations, we notice that during the replacement ($15s < t < 20s$) the values fall below the minimum required, thus not satisfying the conditions. Additionally, the computational time necessary to obtain this solution was 0.9045 seconds, which is lower than the values obtained in Section 5.6 (around 2 seconds).
Appendix B. Alternative planning approaches

Figure B.1: Solution with numerical issues obtained with polynomial trajectories. Note that the circles are the initial positions and the stars the final positions.

Figure B.2: Distance between drones with polynomial trajectories

Figure B.3: Clear visibility constraint with polynomial trajectories
B.2 Combined approach

Due to insufficient results in the previous method, we were not able to validate the mixed-integer collision avoidance constraint, nor the approximate linear clear visibility constraint, as there were numerical issues associated with having polynomial trajectories. To avoid these issues, we implemented a combined strategy, in which these constraints along with others described in Chapter 5 would be used to generate the discrete trajectories defined in Chapter 5, in order to solve a Mixed-Integer Quadratic optimization Problem.

From the method in Chapter 5, we used the trajectory dynamics formulated in (5.6) and (5.7) given by

\[
\begin{align*}
\mathbf{v} &= H_{11} \mathbf{\chi} + \mathbf{v}_0, \\
\mathbf{p} &= \frac{h^2}{2} H_{22} \mathbf{\chi} + h H_{33} \odot \mathbf{v}_0 + \mathbf{p}_0,
\end{align*}
\]

(B.42)

where \( \mathbf{\chi} \) is the optimization variable and represents a vector of accelerations. The objective function is also the same (convex) quadratic function (5.8) \( f_0 = \sum_{i=1}^{Q} \sum_{k=1}^{K} ||a_i[k] + g z_W||^2 \). In terms of constraints, we implemented the same linear equalities and inequalities described in Section 5.3, which included:

1. Initial and final states of each drone in terms of position, velocity and acceleration;
2. Formation flight positions for target following;
3. Dynamic limitations for position, velocity, acceleration and jerk;
4. Minimum height during flight.

Then, from the method in Section B.1, we used the linear approximated clear visibility constraint and the mixed-integer collision avoidance constraint.

With this strategy we can avoid the numerical problems associated with using polynomials over long periods of time or for complex trajectories, and, at the same time, we are able to formulate the problem as a convex Mixed-Integer Quadratic optimization Problem, which means that the solution obtained is a unique global optimal solution. However, note that the linear clear visibility constraint is simply an approximation of the nonlinear constraint considered in (5.32), which means that the solution might not be satisfactory in solving our problem.

B.2.1 Simulation test results

Several simulation tests were made for different trajectories and using different parameters. In this section, we will analyse the results obtained for the same two scenarios described in 5.6. The tests were conducted on the same Lenovo Laptop running Ubuntu 16.04 and equipped with an Intel Core i7-7700HQ CPU @2.80GHz and 16.00GB of RAM, where Matlab R2018a for Linux was running.
**Scenario 1**: two drones following a target in a circular trajectory.

We used the same trajectory parameters defined in the Subsection B.1.6, such as the duration of intervals and formation flight positions (Table 5.1). In the end, we obtained the trajectories represented in Figure B.4, where we can see that the UAVs are able to execute the manoeuvres initially planned.

Then, to check the satisfiability of the collision avoidance and clear visibility constraints, we plotted the same distances and angles described in the other tests (Figures B.5 and B.6) and noticed that the solution satisfies the constraints imposed for collision avoidance and approximate clear visibility. However, it does not always satisfy the nonlinear condition for clear visibility in (5.32), which means that if we consider that the camera of drone 1 is pointed at the target (as assumed in (5.32)), then drone 2 will be seen after \( t = 16.75 \) seconds.

Additionally, we computed the mean computation time after running the algorithm 100 times and obtained 0.7781 seconds, which is lower than any case computed in Section 5.6 (2.2896 seconds for the SCP). Also, the value of the objective obtained was 75.7977.

![Trajectories in 3D](image1)

![Projection of the trajectory in XY plane](image2)

Figure B.4: Solution obtained with the combined approach for scenario 1. Note that the circles are the initial positions and the stars the final positions.

![Distance between drones 1 and 2](image3)

Figure B.5: Distance between drones with combined approach for scenario 1
Scenario 2: three drones following a target in a straight path.

In this scenario, the trajectory parameters were the same as the ones described in Section 5.6. In the end, we obtained the trajectories represented in Figure B.7. Although these appear to be distorted, specially during the exchange of positions, they are complying with the constraints imposed (both collision avoidance and approximate clear visibility), which denotes that this approximate formulation might not work for certain trajectories.

We also plotted the distances and angles described in the other simulation tests to verify the conditions of collision avoidance and clear visibility (Figures B.8 and B.9), and noticed that while the collision avoidance and approximate constraints are always satisfied, the angle computed with the nonlinear conditions between the target and drone 1 w.r.t. drone 3 is below the \( \alpha_{\text{cam}} \) during 24 \( \leq t \leq 26 \) seconds.

We also computed the mean computation time after running the algorithm 100 times and obtained 13.1005 seconds, which is higher than the SCP solution in Section 5.6 (4.2484 seconds), and the value of the objective obtained was 1038.91.

Finally, we can conclude that although this method is convex, which means that local optimal solutions are also global and unique, the solution obtained might not be sufficient in solving the problem of replacement described in Section 1.3, specially due to the necessary approximations in the clear visibility condition. Thus, we chose to use the method in Chapter 5, because despite not being convex, there is a guarantee that the solution provided satisfies all of the necessary constraints and is able to execute the replacement manoeuvres.
Appendix B. Alternative planning approaches

(a) Trajectories in 3D

(b) Projection of the trajectory in XY plane

Figure B.7: Solution obtained with combined approach for scenario 2. Note that the circles are the initial positions and the stars the final positions.

Figure B.8: Distance between drones with combined approach for scenario 2

Figure B.9: Clear visibility constraint with combined approach for scenario 2