UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

DEVELOPMENT AND SIMULATION OF A NOVEL GUIDANCE SYSTEM FOR QUADROTORS FLYING IN A CONTESTED ENVIRONMENT

A THESIS

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

MASTER OF SCIENCE

By

COLETON DOMANN

Norman, Oklahoma

2019

DEVELOPMENT AND SIMULATION OF A NOVEL GUIDANCE SYSTEM FOR QUADROTORS FLYING IN A CONTESTED ENVIRONMENT

A THESIS APPROVED FOR THE
SCHOOL OF AEROSPACE AND MECHANICAL ENGINEERING

BY

Acknowledgements

I would first like to thank my advisor, Dr. Andrea L'Afflitto, for bringing me into his lab. The last two and a half years have pushed me to to grow academically and as a person. He has provided me with several opportunities I would not have otherwise had and has taught me an immeasurable amount about optimization and control theory.

I would also like to thank Dr. Wei Sun and Dr. David Miller for serving on my thesis committee. Their feedback and questions push this work to be better. I would also like to express my appreciation to them for their patience with the thesis defense scheduing process.

Further, I would like to thank my coworkers at the Advanced Control Systems Laboratory: Karen Martinez Soto, Joshua Karinshak, Blake Anderson, Julius Marshall, John Paul Burke, and Timothy Blackford. Without their constant support and assistance, none of this could have been possible. Your companionship kept me sane when work piled up and got me to this point.

Lastly, I would like to thank my friends and family for their continuous love and support. When the tunnel didn't seem to have a light at the end, they were all my guiding light.

This thesis was partly supported by DARPA under grant no. D18AP00069. I wish to express my gratitude to DARPA for its continuous support

Contents

List of Figures

Abstract

Due to their accessibility, quadrotors are used as testbeds for guidance, navigation, and control systems and have been used for a wide variety of applications. These applications range from commercial aerial photography to military surveillance. In the latter case, the vehicle can be exposed to danger if opposing agents are present. The vehicle is incentivized to take cover among obstacles in order to protect it during surveillance operations. However, current research seeks only to avoid obstacles. This thesis develops a guidance system capable of generating tactical flight of a quadrotor. This guidance system consists of three complementary subsystems. The first is fast model predictive control (FMPC). This algorithm uses the dynamics of the system to plan several time steps into the future. Using this plan, the algorithm communicates the optimal action to take and repeats the process. FMPC is advantageous due to its fast computation subject to the system dynamics and customizable cost functions that can be used to enforce tactical behavior. However, it is not designed to work in non-convex environments and is vulnerable to locally optimal points that are not the goal position. The convexity issue is mitigated by using quadratic discrimination to find a locally convex region. Local optima are avoided by employing a global pathfinding algorithm built off of the motion primitive library. Flight test results demonstrating the capabilities of the resultant guidance system are presented.

Keywords: Quadrotor, model predictive control, guidance systems, constraint generation, contested environment

1. Introduction

Quadrotors are small, unmanned aerial vehicles (UAVs) that have grown in popularity as a testbed for guidance, navigation, and control systems due to their low cost, high maneuverability, and established dynamics. The research into these systems have opened the door to a diverse set of applications for quadrotors. Commercial applications include aerial photography, crop monitoring, package delivery, and search and rescue operations. Defense applications have significant overlap but tend to focus more heavily on scouting and surveillance. While these mission profiles differ, several of these applications incentivize similar flight characteristics. Namely, flight profiles that avoid obstacles and maneuver through the environment quickly are of particular interest.

However, this mission profile is not always ideal. In defense applications, the quadrotor will often operate in contested environments in which opposing agents are present. Since the quadrotor has no way of actively defending itself, it must either avoid being detected or avoid being targeted. In either case, it is advantageous to fly near and among obstacles. This is directly opposite of the profile desired by other guidance systems as flying near obstacles increases the chances of collision.

In this thesis, we seek build a novel autopilot consisting of guidance, navigation and control systems that steers the quadrotor near obstacles while traversing the environment. The control system, which has already been developed, is a model reference adaptive control chosen for its ability to reject disturbances due to aerodynamic effects near obstacles. The navigation system, still in development, uses a stereoscopic camera to map the environment and determine the state of the quadrotor. This thesis focuses on the guidance system where the vehicle is given directions on how to reach a goal in a tactical manner.

An outline of this thesis is as follows. Chapter 2 presents the history and theory behind the model predictive control algorithm. This algorithm is tasked with finding an optimal trajectory to a goal position. Further, we describe how the structure of the problem is exploited to more quickly find a solution to the problem, forming a fast model predictive control algorithm (FMPC). Then, we demonstrate the flexibility of this algorithm's cost function and describe how changes can be made to fit our mission profile.

Chapter 3 describes the equations of motion of a quadrotor. First, these equations are presented in full. However, the full equations of motion are highly nonlinear and thus cannot be directly applied with FMPC. So, the equations of motion are linearized and applied with the results of Chapter 2 to create simulations shown in Chapter 4.

Chapter 5 identifies that FMPC struggles in non-convex operating regions. To rectify this issue, constraint generation methods are developed that identify a locally convex region within which FMPC can operate. This is not a problem with a trivial solution, so instead an approximate solution is found by finding ellipsoids that separate the vehicle from surrounding obstacles at regular intervals while the vehicle traverses the environment. An affine hull based off of this ellipsoid is found.

In Chapter 6, we identify that FMPC is susceptible to finding locally optimal points and thus failing to find the goal position. This issue is fixed by identifying a global path planning algorithm based off of the motion primitive library. This path planner is updated to enable tactical flight. FMPC is used to avoid obstacles and refine the global plan. Chapter 7 brings all of these elements together and presents flight tests that demonstrate several capabilities of the guidance system.

2. Model Predictive Control

2.1 Background

Model Predictive Control (MPC) is a technique to find optimal control laws that optimize multi-input, multi-output (MIMO) systems subject to state and control constraints. According to this technique, at each time instance a sequence of control inputs, which minimize a cost function and verify the given constraints, is computed. The controller then applies the first control input of this sequence. Successively, the MPC algorithm calculates a new sequence of control inputs and the process is iterated until the system reaches its goal [2]. This approach offers several benefits. Deviations in the system behavior due to disturbances or uncertainties are inherently compensated for by the recalculation of the control plan. All the while, MPC ensures that constraints are verified at all time and the prediction is based on the system dynamics [2].

MPC was developed by Shell in 1978 for use in large scale industrial processes [3]. This original implementation relied on antiquated computers and was slow to execute. The dynamics of petroleum and chemical plants are slow and hence, MPC algorithms were suitable for these applications. MPC has advanced along with technology due to improved hardware, software, and algorithms, greatly increasing the number of MPC applications [4]. However, industrial applications continue to be the primary implementation of MPC. Adoption of MPC in systems with faster dynamics, such as those found in the aerospace industry, has been hampered by computational methods that require sampling times separated by several seconds or even minutes [5]. This kind of slow calculation would be disastrous outside of the current applications for which MPC is currently used.

Several methods have been introduced to reduce the computational cost of MPC. The first step for many of these methods is to limit the optimization problem to a certain class so that the structure of the problem can be exploited for efficiency. In this thesis, a linear dynamical model, polyhedral constraints, and a quadratic cost function are assumed so that the resulting optimization problem is a quadratic program (QP) [5]. Reducing the problem to a QP has the inherent advantage of ensuring that a solution exists to the optimization problem. In order to compute the solutions of QP problems underlying an MPC algorithm, Wang and Boyd proposed in [5] a new approach that has a relatively low computational cost. Specifically, the QP problem is rearranged so that the differential constraints involve tri-diagonal state transition matrices. In doing so, the time complexity of solving the MPC problem is reduced from cubic to linear [5]. Further reduction of the computational cost is possible by implementing a warm-start method, whereby the prediction from the previous iteration is used to create an initial guess for the solution of the next prediction. Specifically, since the solution at a future iteration is likely to be similar to the solution of the previous iteration, the number of steps needed to solve the problem is reduced drastically [6]. Lastly, the QP does not need to be solved to full accuracy to find a solution that is capable of controlling the system. This property is a consequence of the nature of MPC as a planner. Since only the first step of the plan is utilized, the plan does not need to be perfectly accurate in order to find a solution that stabilizes the system. In general, only 3 to 5 iterations are needed to provide high quality control [5]. For its reduced computational cost, Wang and Boyd's approach can be considered as a Fast Model Predictive Control (FMPC) algorithm. This algorithm is the foundation of the autopilot for multi-rotor UAVs presented in this thesis.

2.2 Model Predictive Control as a QP Problem

FMPC, as discussed previously, is an algorithm that seeks to control the discrete-time linear dynamical system

$$x(t+1) = Ax(t) + Bu(t) + w(t), \qquad x(0) = x_0, \qquad t = 0, 1, \ldots, T-1, \tag{2.1}$$

where t denotes a *time step*, $x(t) \in \mathbb{R}^n$ denotes the *state vector*, $u(t) \in \mathbb{R}^m$ denotes the *control vector*, $w(t) \in \mathbb{R}^n$ captures disturbances, $A \in \mathbb{R}^{n\times n}$, and $B \in \mathbb{R}^{n\times m}$. In the following, we assume that $w(\cdot)$ is independent identically distributed with known distributions for different values of $t = 0, \ldots, T-1$, and define $\bar{w} \triangleq \mathbf{E}w(t)$, $t = 0, \ldots, T-1$ as the *mean value* of $w(\cdot)$ [5].

The control input $u(\cdot)$ must minimize the objective function

$$J[x_0, u(\cdot)] = \lim_{T\to\infty} \frac{1}{T}\mathbf{E} \sum_{\tau=t}^{t+T-1} \ell(x(\tau), u(\tau)), \tag{2.2}$$

where

$$\ell(x,u) = \begin{bmatrix} x \\ u \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} Q & S \\ S^{\mathrm{T}} & R \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} + q^{\mathrm{T}}x + r^{\mathrm{T}}u, \qquad (x,u) \in \mathbb{R}^n \times \mathbb{R}^m \tag{2.3}$$

$Q = Q^T \in \mathbb{R}^{n\times n}$, $R = R^{\mathrm{T}} \in \mathbb{R}^{m\times m}$, $S \in \mathbb{R}^{n\times m}$, $q \in \mathbb{R}^n$, and $r \in \mathbb{R}^m$ are user-defined parameters. To guarantee the existence of a solution of the given QP problem, we assume that

$$\begin{bmatrix} Q & S \\ S^{\mathrm{T}} & R \end{bmatrix} \geq 0, \tag{2.4}$$

where $\geq$ denotes non-negative definiteness. In this thesis, the state and control constraints are separated, that is $S = 0$. Therefore, it is assumed that $Q \geq 0$ and $R \geq 0$. While not necessary to satisfy (2.4), it is typical to enforce $R > 0$ and $Q > 0$. Lastly, the control input $u(\cdot)$ must be chosen so that the dynamical model (2.1) verifies linear inequality constraints

$$F_x x(t) + F_u u(t) \;\leq\leq\; f, \qquad t = 0, 1, \ldots, T-1, \tag{2.5}$$

$$F_f x(t+T) \;\leq\leq\; f_f, \tag{2.6}$$

where $F_x \in \mathbb{R}^{l\times n}$, $F_u \in \mathbb{R}^{l\times m}$, $F_f \in \mathbb{R}^{k\times n}$, $f \in \mathbb{R}^l$, $f_f \in \mathbb{R}^k$, $T \in \overline{\mathbb{N}}$ denotes the time horizon, and $\leq\leq$ denotes the component-wise inequality. In general, the constraints are not assumed to be constant. Indeed, constraints can be modified to vary over time and space. However, doing so means that the affinity of the constraint set may not be retained. Therefore, we assume that the constraints are constant at each step of the FMPC algorithm and update them at each iteration, if necessary.

It follows from the dynamic programming principle [7] that the optimization problem captured by (2.1), (2.2), and (2.3) is equivalent to

$$\begin{aligned} \text{minimize:} \quad & J[x_0, u(\cdot)] = \ell_f(x(t+T)) + \sum_{\tau=t}^{t+T-1} \ell(x(\tau), u(\tau)), && (2.7)\\ \text{subject to:} \quad & F_x x(\tau) + F_u u(\tau) \leq\leq f, && (2.8)\\ & F_f x(t+T) \leq\leq f_f && (2.9)\\ & x(\tau+1) = Ax(\tau) + Bu(\tau) + \bar{w}, && (2.10)\\ & \tau = t, \ldots, t+T-1, && (2.11) \end{aligned}$$

where T denotes the time horizon, that is the number of time steps in the future that MPC uses to plan a control input, and

$$\ell_f(x) = x^{\mathrm{T}} Q_f x + q_f^{\mathrm{T}} x, \qquad x \in \mathbb{R}^n, \tag{2.12}$$

denotes the terminal cost function where $Q_f = Q_f^{\mathrm{T}} \geq 0$.

To illustrate how FMPC solves the problem captured by (2.7)-(2.11), a conceptual interpretation is presented here. At the beginning of the current time step t, the MPC controller takes the current state $x(t)$ and determines the optimal control $u^*(t)$ that minimizes the cost function $J[x_0, u(\cdot)]$ and verifies the constraints (2.7)-(2.11). Successively, the state vector at the next time step $t+1$ is calculated by applying $u^*(t)$ to (2.1). This process is repeated for each time step within the planning time horizon T and the planning process results in an optimal solution that is captured by $u^*(t), \ldots, u^*(t+T-1)$ and $x^*(t+1), \ldots, x^*(t+T)$. The control vectors can be interpreted as a plan of action for the next T time steps, while the state vectors can be interpreted as the predicted effect of the plan of action. The FMPC controller applies $u(t) = u^*(t)$, that is, only the first control vector of the plan of action. The system reacts to this input and the entire process is repeated. Note, this explanation is a purely conceptual illustration of the philosophy of the FMPC controller. Enforcing the system dynamics through equality constraints allows for all steps in the time horizon to be computed simultaneously instead of sequentially as described here.

2.3 Fast Solution of the QP Problem

With the overall structure of the QP established, the problem can be rearranged creating a more compact form that we will later exploit to reduce computation costs. First, we coalesce the state and control variables for each step of the time horizon into a single optimization variable of the form

$$z = [u^{\mathrm{T}}(t), x^{\mathrm{T}}(t+1), \ldots, u^{\mathrm{T}}(t+T-1), x^{\mathrm{T}}(t+T)]^{\mathrm{T}} \in \mathbb{R}^{T(m+n)}. \tag{2.13}$$

With this change, the QP problem (2.7)-(2.11) is equivalent to the following problem:

$$\begin{aligned} \text{minimize:} \quad & z^{\mathrm{T}} H z + g^{\mathrm{T}} z, && (2.14) \\ \text{subject to:} \quad & Pz \leq\leq h, && (2.15) \\ & Cz = b, && (2.16) \end{aligned}$$

where

$$H \triangleq \begin{bmatrix} R & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & Q & S & \dots & 0 & 0 & 0 \\ 0 & S^{\mathrm{T}} & R & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & Q & S & 0 \\ 0 & 0 & 0 & \dots & S^{\mathrm{T}} & R & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 & 0 \end{bmatrix}, \tag{2.17}$$

$$P \triangleq \begin{bmatrix} F_u & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & F_x & F_u & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & F_x & F_u & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 & F_f \end{bmatrix}, \tag{2.18}$$

$$C \triangleq \begin{bmatrix} -B & I & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & -A & -B & I & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & -A & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & I & 0 & 0 \\ 0 & 0 & 0 & 0 & \dots & -A & -B & I \end{bmatrix}, \tag{2.19}$$

$$g \triangleq \begin{bmatrix} r + 2S^{\mathrm{T}}x(t) \\ q \\ r \\ \vdots \\ q \\ r \end{bmatrix}, \tag{2.20}$$

$$h \triangleq \begin{bmatrix} f - F_x x(t) \\ f \\ \vdots \\ f \\ f_f \end{bmatrix}, \tag{2.21}$$

$$b \triangleq \begin{bmatrix} Ax(t) + \bar{w} \\ \bar{w} \\ \bar{w} \\ \vdots \\ \bar{w} \\ \bar{w} \end{bmatrix}. \tag{2.22}$$

Note that H, P, and C are block tridiagonal and, hence, large portions of the matrices are trivial when performing matrix multiplication. By recognizing the patterns in these matrices, we can use factorization and specialized linear algebra libraries such as Lapack [8] or Eigen [9] to efficiently perform matrix multiplications involving these matrices.

Wang and Boyd utilize an infeasible start primal barrier method [10] to solve the QP. To illustrate this method, we discuss the primal barrier method first. According to the primal barrier method, inequality constraints are replaced by a new term in the cost function so that the optimization problem given by (2.14)-(2.16) is equivalent to

$$\text{minimize} \quad z^{\mathrm{T}} H z + g^{\mathrm{T}} z + \kappa \phi(z), \tag{2.23}$$

$$\text{subject to} \quad Cz = b \tag{2.24}$$

where

$$\phi(z) \triangleq \sum_{i=1}^{lT+k} -\log(h_i - p_i z), \qquad z \in \mathbb{R}^{T(n+m)}, \tag{2.25}$$

denotes a *logarithmic barrier function* associated with the inequality constraints, $\kappa > 0$ denotes a weighting parameter on the barrier function, $p_1, \ldots, p_{lT+k}$ denote the rows of P in (2.18), and $h_1, \ldots, h_{lT+k}$ denote corresponding partitions h. The function $\phi(\cdot)$ introduces a constraint so that $Pz \leq\leq h$. Importantly, $\phi(\cdot)$ is a smooth, convex function within its domain. Since the objective function (2.14) is convex and the sum of convex functions is convex, (2.23) is also convex. Therefore, convex optimization solution methods can be applied to solve the constrained optimization problem given by (2.23) and (2.24).

In order to implement an infeasible start Newton method, the problem described by (2.23) and (2.24) is solved by finding $z \in \mathbb{R}^{T(m+n)}$ where the gradient of the equation is zero. To achieve this goal, a Lagrange multiplier $\nu \in \mathbb{R}^{Tn}$ associated with the equality constraint is added to the gradient. This yields the dual and primal residuals [11, pp. 215-273]

$$\begin{aligned} r_d(z) &= 2Hz + g + \nabla\kappa\phi(z) + C^{\mathrm{T}}\nu = 0, \qquad z \in \mathbb{R}^{T(m+n)}, & (2.26)\\ r_p(z) &= Cz - b = 0, & (2.27) \end{aligned}$$

where $\kappa\nabla\phi(z)$ denotes the gradient of the barrier function. It follows from (2.25) that

$$\kappa\nabla\phi_i(z) = \frac{\kappa p_i^{\mathrm{T}}}{h_i - p_i z}, \qquad z \in \mathbb{R}^{T(m+n)}, \tag{2.28}$$

and hence,

$$\kappa\nabla\phi(z) = \kappa \sum_{i=1}^{lT+k} \nabla\phi_i(z) = \kappa P^{\mathrm{T}} d(z), \tag{2.29}$$

where $d_i(z) \triangleq 1/(h_i - p_i z)$. Substituting (2.29) into (2.26) yields the final residual equations necessary to solve the problem in (2.23):

$$\begin{aligned} r_d(z) &= 2Hz + g + \kappa P^{\mathrm{T}} d + C^{\mathrm{T}}\nu = 0, \qquad z \in \mathbb{R}^{T(m+n)} & (2.30)\\ r_p(z) &= Cz - b = 0. & (2.31) \end{aligned}$$

The system is optimal when both the primal and dual residuals are equal to zero. This is checked by creating a vector $r \triangleq [r_d^{\mathrm{T}}, r_p^{\mathrm{T}}]^{\mathrm{T}}$ and computing $||r||_2$.

At the beginning of each iteration of FMPC, the residual is calculated with an initial z that satisfies the inequality constraints and any initial ν. If the norm of the residual is greater than a set tolerance value, then z and ν are modified by adding correction terms Δz and $\Delta\nu$, respectively. The correction terms are found by solving the linear approximation

$$\begin{bmatrix} \Phi(z) & C^{\mathrm{T}} \\ C & 0 \end{bmatrix} \begin{bmatrix} \Delta z \\ \Delta\nu \end{bmatrix} = - \begin{bmatrix} r_d(z) \\ r_p(z) \end{bmatrix}, \qquad z \in \mathbb{R}^{T(m+n)} \tag{2.32}$$

where

$$\Phi(z) = 2H + \kappa P^{\mathrm{T}} \mathrm{diag}(d(z))^2 P \tag{2.33}$$

and $\kappa P^{\mathrm{T}}\mathrm{diag}(d(z))^2 P$ is the Hessian of $\kappa\phi(z)$. Once Δz and $\Delta\nu$ are calculated, they are placed into a backtracking line search. In particular, this search seeks a step size $s \in (0, 1]$ such that the norm of the residual is below the tolerance value for $z := z + s\Delta z$ and $\nu := \nu + s\Delta\nu$. The initial value of s is equal to one and is multiplied by some $\alpha \in (0, 1)$ after each attempt so that the step size decreases until the residual reaches an acceptable value.

2.4 Soft Constraints in FMPC

As seen in the previous section, the constraints modeled in Wang and Boyd's FMPC algorithm are *hard constraints*. That is, the constraints cannot be violated under any circumstances. Ideally, MPC ensures that constraints are never violated by diverting the system from constraints detected within the planning horizon. However, there are cases where relying solely on hard constraints can be problematic. In particular, it is desirable to compute control inputs that allow reaction to constraints well before constraints are encountered.

The need to avoid constraints early on manifested itself during the development of the autopilot described in this thesis. For instance, in one of our first test cases, the constraints on the control vector were so strict that the quadcopter struggled to change its momentum. When combined with a short time horizon, the system was unable to avoid collisions in simulation forcing the program to exit. Evidently, a similar issue would have also occurred in actual flight tests. In particular, if a new obstacle was detected within the time horizon, the MPC's prediction would have struggled to divert the system.

The need to take control actions and avoid constraints early on is evident also in the case that the navigation system provides erroneous information about the environment and detects obstacles that do not exist. Violating non-existent constraints would cause the program to exit assuming that the quadrotor has crashed, although no physical collision actually occurred. Even if the program underlying the guidance system were rewritten to not exit when hard constraints are violated, the FMPC algorithm would have no information on the gradient of the cost function.

Introducing soft constraints helps alleviate all these issues. In particular, soft constraints allow the system to take corrective action well before the actual, hard constraints are met. This property is especially useful in implementing safety margins when defining constraints.

There are several methods to apply soft constraints. One approach is to introduce *slack variables* to transform inequality constraints into equality constraints [11, pp. 131]. This method comes with the downside of adding a new decision variable for every step in the time horizon [12]. It is possible to mitigate this issue by introducing a slack variable that captures the worst case violation [13]. However, the FMPC algorithm introduced by [5] relies on specific sparsity patterns in the QP and adding this slack variable may disrupt this pattern. To preserve the block-tridiagonal structure of the matrices $H, P,$ and C in (2.17)-(2.19), the authors of [12] implement soft constraint directly into the cost function of the QP, bypassing slack variables entirely.

The method presented in [12] to implement soft constraints involves the Kreisselmeister-Steinhauser (KS) function [14]. Given a set of k functions $g : \mathbb{R}^n \to \mathbb{R}^k$, the KS function is defined as

$$KS[g(x)] \triangleq \frac{1}{\rho}\left[\textstyle\sum_{j=1}^{k} e^{\rho g_j(x)}\right], \qquad x \in \mathbb{R}^n \tag{2.34}$$

where $\rho > 0$ is a user-defined parameter. The KS function is such that $KS[g(x)] \to \max_j g_j(x)$ as $\rho \to \infty$, where $g_j(\cdot)$ denotes the j^{th} component of $g(\cdot)$. Large values of ρ may cause the exponential to grow to the point of causing numerical issues. To prevent this problem from happening, a modified KS function can be employed and is defined by

$$KS[g(x)] = g_{\max}(x) + \frac{1}{\rho}\log\left[\textstyle\sum_{j=1}^{k} e^{\rho[g_j(x) - g_{\max}(x)]}\right], \tag{2.35}$$

where $g_{\max}(x) \triangleq \max_j g_j(x)$, $x \in \mathbb{R}^n$.

The modified KS function (2.35) can be applied to the soft constraints

$$\tilde{P}z \leq\leq \tilde{h}, \tag{2.36}$$

where $\tilde{P} \in \mathbb{R}^{\alpha \times T(m+n)}$, $\tilde{h} \in \mathbb{R}^{\alpha}$, and α is the number of soft constraints. Each soft constraint is given a penalty according to the equation

$$\theta_{\text{exact}}(\tilde{P}z - \tilde{h}) = \sum_{i=1}^{\alpha} \max\{0,\, \tilde{p}_i z - \tilde{h}_i\}, \qquad z \in \mathbb{R}^{T(m+n)}, \tag{2.37}$$

where $\tilde{p}_i$ denotes the i^{th} row of $\tilde{P}$ and $\tilde{h}_i$ denotes the i^{th} element of $\tilde{h}$. This penalty is equal to zero if the constraint is violated or verified with an equality, and increases as the constraint is violated. The penalty can be smoothed by employing the KS function for each $i = 1, \ldots, \alpha$. If the constraint is not violated, then $g_{\max} = 0$ and (2.37) can be captured as $KS[\tilde{P}z - \tilde{h}]$. Applying (2.35), the soft constraints (2.36) can be approximated by

$$\theta(\tilde{P}z - \tilde{h}) = \sum_{i=1}^{\alpha} \frac{1}{\rho} \log \left[e^{0} + e^{\rho(\tilde{p}_i z - \tilde{h}_i - 0)} \right], \qquad z \in \mathbb{R}^{T(m+n)}, \tag{2.38}$$

which simplifies to

$$\theta(\tilde{P}z - \tilde{h}) = \frac{1}{\rho} \sum_{i} \log \left[1 + e^{\rho(\tilde{p}_i z - \tilde{h}_i)} \right], \qquad z \in \mathbb{R}^{T(m+n)}. \tag{2.39}$$

The term $\theta(\tilde{P}z - \tilde{h})$, $z \in \mathbb{R}^{T(m+n)}$ is added to the cost function (2.23) so that the optimization problem considered in this thesis is now given by

$$\text{minimize:} \quad z^{\mathrm{T}} H z + g^{\mathrm{T}} z + \kappa \phi(z) + \theta(z), \tag{2.40}$$

$$\text{subject to:} \quad Cz = b, \tag{2.41}$$

where $\theta(z)$ denotes $\theta(\tilde{P}z - \tilde{h})$ for brevity.

Following the same procedure that is used in the previous section, the residuals and correction terms can be calculated. First, the gradient of the cost function is found. To avoid reworking the problem, the residual accounting for soft constraints is expressed in terms of the old residual terms:

$$\tilde{r}_d(z) \quad = \quad r_d(z) + \nabla\theta(z), \qquad z \in \mathbb{R}^{T(m+n)} \tag{2.42}$$

$$\tilde{r}_p(z) \quad = \quad r_p(z). \tag{2.43}$$

It follows from (2.39) that the gradient of the soft constraint penalty function is given by

$$\nabla\theta_i(z) = \frac{1}{\rho}\left[\rho\tilde{p}_i^{\mathrm{T}} e^{\rho(\tilde{p}_i z - \tilde{h}_i)} \frac{1}{1 + e^{\rho(\tilde{p}_i z - \tilde{h}_i)}}\right], \qquad z \in \mathbb{R}^{T(m+n)}, \qquad i = 1, \ldots, \alpha. \tag{2.44}$$

In order to avoid calculations involving large exponentials and find a more concise representation of (2.44), two exponential functions are defined. The first of these functions, which is used when the constraint is verified, is defined as

$$e_i^+(z) \triangleq e^{\rho(\tilde{p}_i z - \tilde{h}_i)}, \qquad z \in \mathbb{R}^{T(m+n)}, \tag{2.45}$$

while the second exponential function, which is used when the constraint is violated, is defined as

$$e_i^-(z) \triangleq e^{\rho(\tilde{h}_i - \tilde{p}_i z)} = \frac{1}{e_i^+(z)}. \tag{2.46}$$

It follows from (2.44) - (2.46) that

$$\nabla\theta_i(z) = \tilde{p}_i^{\mathrm{T}} \tilde{d}_i(z), \qquad z \in \mathbb{R}^{T(m+n)}, \qquad i = 1, \ldots, \alpha, \tag{2.47}$$

where

$$\tilde{d}_i(z) \triangleq \frac{e_i^+(z)}{1 + e_i^+(z)} = \frac{1}{1 + e_i^-(z)}, \tag{2.48}$$

and thus

$$\nabla\theta(z) = \sum_{i=1}^{\alpha} \nabla\theta(z) = \tilde{P}^{\mathrm{T}} \tilde{d}(z). \tag{2.49}$$

Further, the Hessian of the soft constraint penalty function is given by

$$\nabla^2\theta(z) = \rho \tilde{P}^{\mathrm{T}} \mathrm{diag}(\hat{d}(z)) \tilde{P}, \qquad z \in \mathbb{R}^{T(m+n)} \tag{2.50}$$

where

$$\hat{d}_i(z) = \frac{e_i^+(z)}{(1 + e_i^+(z))^2} = \frac{e_i^-(z)}{(1 + e_i^-(z))^2}. \tag{2.51}$$

With the Hessian established, the Newton step formula (2.32) modified to account for soft constraints can be established as

$$\begin{bmatrix} \tilde{\Phi}(z) & C^{\mathrm{T}} \\ C & \mathbf{0} \end{bmatrix} \begin{bmatrix} \Delta z \\ \Delta \nu \end{bmatrix} = - \begin{bmatrix} \tilde{r}_d(z) \\ \tilde{r}_p(z) \end{bmatrix}, \qquad z \in \mathbb{R}^{T(m+n)} \tag{2.52}$$

where

$$\tilde{\Phi}(z) = \Phi(z) + \rho \tilde{P}^{\mathrm{T}} \mathrm{diag}(\hat{d}(z)) \tilde{P}, \tag{2.53}$$

and

$$\tilde{r}_d(z) = r_d(z) + \tilde{P}^{\mathrm{T}} \tilde{d}(z). \tag{2.54}$$

It is improtant to remark that, in the residual equations, the soft constraint term $\tilde{P}^{\mathrm{T}}\tilde{d}(z)$ is of the same form as the term arising from the hard constraint barrier function $\kappa P^{\mathrm{T}} d(z)$. Likewise, the Hessian of the soft constraint penalty function used in computing the Newton step $\rho \tilde{P}^{\mathrm{T}} \mathrm{diag}(\hat{d}(z)) \tilde{P}$ is directly related to the Hessian of the hard constraint barrier function given by $\kappa P^{\mathrm{T}} \mathrm{diag}(d(z))^2 P$. By modeling the soft constraints in this manner, the block structure of $\Phi(z)$ is the same as the structure of $\tilde{\Phi}(z)$. Preserving this block structure allows the same solution process to be used when soft constraints are added to the QP.

3. Quadrotor Dynamics

3.1 Introduction

In this thesis, we consider unmanned Class 1 quadrotors, that is, aerial vehicles equipped with four coaxial propellers similar to the one schematically shown in Figure 3.1. In recent years, these platforms have seen an increase in popularity both in industrial and academic applications, since they are fairly cheap and easy to maintain. Additionally, the small size of quadrotors allows for indoor testing in safe and self-contained environments.

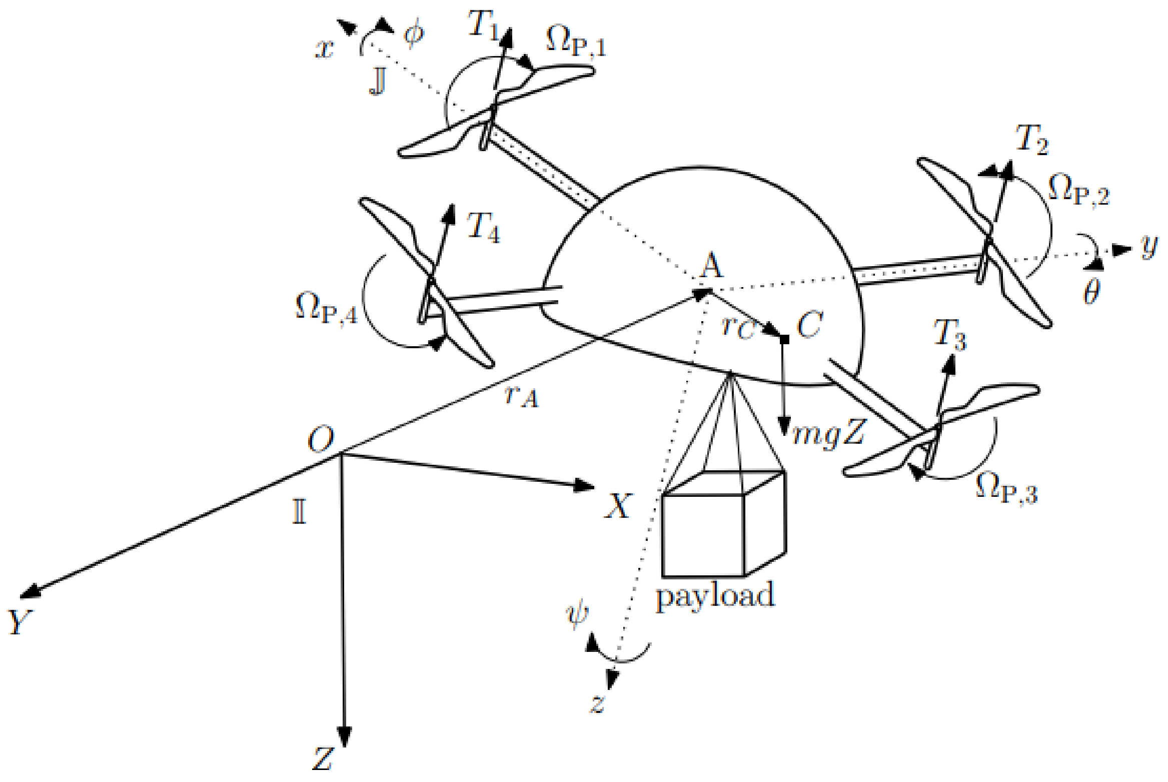


Figure 3.1: Kinematic diagram of a quadrotor in flight

In order to apply the FMPC algorithm presented in Chapter 2 and outline reference trajectories for automaton quadcopters, in this chapter we recall the continuous-time equations of motion of quadrotors. Successively, we provide a discrete-time dynamical model of these vehicles in a format that is suitable for FMPC.

3.2 Equations of Motion of a Quadcopter in Continuous Time

Figure 3.1 gives a pictorial representation of a quadrotor in flight [15]. position and orientation of this vehicle is captured in the inertial reference frame $\mathbb{I} = \{O; X, Y, Z\}$ that is centered at the point O and has orthonormal axes X, Y, and Z; note that the positive Z axis is collinear to the gravitational acceleration vector g. In particular, the position of the quadcopter is identified by the reference point A, whose distance from O is captured by $r_A : [s_0, \infty) \to \mathbb{R}^3$.

The reference frame $\mathbb{J} = \{A; x(s), y(s), z(s)\}$, $s \geq s_0$, is fixed with the quadcopter, and is such that $x(\cdot)$ and $y(\cdot)$ pass through two rotors. The i^{th} propeller's *spin rate* is denoted by $\Omega_{P,i} : [s_0, \infty) \to \mathbb{R}$, $i = 1, \ldots, 4$, and the corresponding thrust force is denoted by $T_i(s)$, $s \geq s_0$. Also, gravity acts on the vehicle's center of mass C. In general, there is no guarantee that the reference point A coincides with the center of mass C. The position of C with respect to A is denoted by vector $r_C : [s_0, \infty) \to \mathbb{R}^3$.

The attitude of the body reference frame $\mathbb{J}$ with respect to $\mathbb{I}$ is defined using a 3-2-1 Euler angle rotation sequence. According to this sequence, the rotation around the $z(\cdot)$ axis is captured by the *yaw* angle $\psi : [s_0, \infty) \to [0, 2\pi)$, the rotation about the $y(\cdot)$ axis is captured by the *pitch* angle $\theta : [s_0, \infty) \to (-\frac{\pi}{2}, \frac{\pi}{2})$, and the rotation about the $x(\cdot)$ axis is captured by the *roll* angle $\phi : [s_0, \infty) \to [0, 2\pi)$.

The equations of motion of a quadrotor are given by four interconnected differential equations. In particular, the *translational kinematic equation* is given by [15]

$$\dot{r}^{\mathbb{I}}_A(s) = R(\phi(s), \theta(s), \psi(s))v_A(s), \qquad r^{\mathbb{I}}_A(s_0) = r^{\mathbb{I}}_{A,0}, \qquad s \geq s_0, \tag{3.1}$$

where

$$R(\phi, \theta, \psi) \triangleq \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix},$$
$$(\phi, \theta, \psi) \in [0, 2\pi) \times \left(-\frac{\pi}{2}, \frac{\pi}{2}\right) \times [0, 2\pi), \tag{3.2}$$

and $v_A : [s_0, \infty) \to \mathbb{R}^3$ denotes the velocity of the reference point A with respect to O. Similarly, the *rotational kinematic equation* is given by [15]

$$\begin{bmatrix} \dot{\phi}(s) \\ \dot{\theta}(s) \\ \dot{\psi}(s) \end{bmatrix} = \Gamma(\phi(s), \theta(s))\omega(s), \qquad \begin{bmatrix} \phi(s_0) \\ \theta(s_0) \\ \psi(s_0) \end{bmatrix} = \begin{bmatrix} \phi_0 \\ \theta_0 \\ \psi_0 \end{bmatrix}, \qquad s \geq s_0, \tag{3.3}$$

where

$$\Gamma(\phi, \theta) \triangleq \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{bmatrix}, \qquad (\phi, \theta) \in [0, 2\pi] \times \left(-\frac{\pi}{2}, \frac{\pi}{2}\right), \tag{3.4}$$

and $\omega : [s_0, \infty) \in \mathbb{R}^3$ denotes the angular velocity of $\mathbb{J}$ with respect to $\mathbb{I}$ [16, p. 6].

Assuming that the mass of the quadrotor is constant for all time, the translational dynamic equation is given by [15]

$$\begin{aligned} F_g(\phi(s), \theta(s)) - F_T(s) + F(s) = m[&\dot{v}_A(s) + \omega^\times(s) v_A(s) + \ddot{r}_C(s) + \dot{\omega}^\times(s) r_C(s) \\ &+2\omega^\times(s)\dot{r}_C(s) + \omega^\times(s)\omega^\times(s) r_C(s)], \\ &v_A(s_0) = v_{A,0}, \qquad s \ge s_0, \end{aligned} \tag{3.5}$$

where

$$F_g(\phi, \theta) = mg[-\sin\theta, \cos\theta\sin\phi, \cos\theta\cos\phi]^{\mathrm{T}}, \qquad (\phi, \theta) \in [0, 2\pi) \times \left(-\frac{\pi}{2}, \frac{\pi}{2}\right), \tag{3.6}$$

denotes the weight of the quadrotor, $-F_T(s) = [0, 0, u_1(s)]^{\mathrm{T}}$ denotes the thrust force, and $F(s)$ denotes aerodynamic forces. Making the further assumptions that the quadrotor and payload joined together form a rigid body and the propellers are modeled as thin spinning discs, the rotational dynamic equation is given by [15]

$$\begin{aligned} M_T(s) + M_g(r_C(s), \phi(s), \theta(s)) + M(s) = mr_C^\times(s)[&\dot{v}_A(s) + \omega^\times(s) v_A(s)] + I\dot{\omega}(s) \\ &+\omega^\times(s) I\omega(s) + I_P \sum_{i=1}^{4} \begin{bmatrix} 0 \\ 0 \\ \dot{\Omega}_{P,i}(s) \end{bmatrix} + \omega^\times I_P \sum_{i=1}^{4} \begin{bmatrix} 0 \\ 0 \\ \Omega_{P,i}(s) \end{bmatrix}, \\ &\omega(s_0) = \omega_0, \qquad s \ge s_0, \end{aligned} \tag{3.7}$$

where $M_T(s) = [u_2(s),\ u_3(s),\ u_4(s)]^{\mathrm{T}}$ denotes the moment of forces generated by the propellers,

$$M_g(r_C,\phi,\theta) \triangleq r_C^\times F_g(\phi,\theta), \qquad (r_C,\phi,\theta)\in\mathbb{R}^3\times[0,2\pi)\times\left(-\frac{\pi}{2}\frac{\pi}{2}\right), \tag{3.8}$$

denotes the moment of the quadrotor's weight about A, $M : [s_0,\infty)\to\mathbb{R}^3$ denotes the moment of aerodynamic forces about A, $I\in\mathbb{R}^{3\times3}$ is the matrix of inertia of the quadrotor with respect to A, and $I_P\in\mathbb{R}^{3\times3}$ is the matrix of inertia of each propeller with respect to A. The terms $I_P\sum_{i=1}^{4}[0,\ 0,\ \dot{\Omega}_{P,i}(s)]$ and $\omega^\times(s)I_P\sum_{i=1}^{4}[0,\ 0,\ \Omega_{P,i}(s)]$ capture the *inertial counter-torque* and the *gyroscopic effect*, respectively.

From the equations of motion of the quadrotor, we can set a state vector that uniquely captures the position and rotation of the vehicle and a control vector can be identified. The state vector is given by

$$x = \begin{bmatrix} r_A^{\mathbb{I},\mathrm{T}} & \phi & \theta & \psi & v_A^\mathrm{T} & \omega^\mathrm{T}\end{bmatrix}^\mathrm{T}\in\mathbb{R}^{12}. \tag{3.9}$$

The relation between the thrust force generated by each propeller and the control inputs is given by

$$\begin{bmatrix} u_1(s)\\ u_2(s)\\ u_3(s)\\ u_4(s)\end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1\\ 0 & -l & 0 & l\\ l & 0 & -l & 0\\ -c_T & c_T & -c_T & c_T\end{bmatrix}\begin{bmatrix} T_1(s)\\ T_2(s)\\ T_3(s)\\ T_4(s)\end{bmatrix}, \qquad s\geq s_0, \tag{3.10}$$

where $l>0$ denotes the distance of the propellers from the vehicle's barycenter,

$c_T > 0$ denotes the propeller's drag coefficient, and the thrust generated by a propeller is related to its spin rate by

$$T_i(s) = k\Omega_{P,i}^2(s), \qquad i = 1, \ldots, 4, \qquad s \geq s_0, \tag{3.11}$$

where $k > 0$.

The equations of motion of the quadrotor described above can be simplified by making several common assumptions. One such assumption that applies to this thesis is that $r_C \approx 0$. Since the quadrotor is not carrying any payload, the reference point A can be set at the quadcopter's center of mass. While this placement is not guaranteed to be perfect, the vehicle's mass distribution is sufficiently well known to make this error sufficiently negligible. With this assumption, (3.5) simplifies to

$$F_g(\phi(s), \theta(s)) - F_T(s) + F(s) = m[\dot{v}_A(s) + \omega^\times(s) v_A(s)],$$
$$v_A(t_0) = v_{A,0}, \qquad s \geq s_0, \tag{3.12}$$

and (3.7) reduces to

$$M_T(s) + M(s) = I\dot{\omega}(s) + \omega^\times(s) I\omega(s) + I_P \sum_{i=1}^{4} \begin{bmatrix} 0 \\ 0 \\ \dot{\Omega}_{P,i}(s) \end{bmatrix} + \omega^\times I_P \sum_{i=1}^{4} \begin{bmatrix} 0 \\ 0 \\ \Omega_{P,i}(s) \end{bmatrix},$$
$$\omega(s_0) = \omega_0, \qquad s \geq s_0. \tag{3.13}$$

3.3 Linearization of Continuous-Time Equations of Motion

A further simplification must be made to employ the FMPC algorithm presented in Chapter 2. The FMPC algorithm requires the plant dynamics to be linear. However, the equations of motion of a quadrotor are highly nonlinear as can be seen in (3.1), (3.3), (3.12), and (3.13). To overcome this difficulty, the equations of motion of a quadcopter can be linearized around the equilibrium point

$$x_e = \begin{bmatrix} r_{A,e}^{\mathbb{I},\mathrm{T}} & \phi_e & \theta_e & \psi_e & v_{A,e}^{\mathrm{T}} & \omega_e^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}} \in \mathbb{R}^{12}, \tag{3.14}$$

$$u_e = \begin{bmatrix} u_{1,e} & u_{2,e} & u_{3,e} & u_{4,e} \end{bmatrix}^{\mathrm{T}} \in \mathbb{R}^{12}. \tag{3.15}$$

By definition, (X_e, u_e) are such that the kinematic and dynamic equations are equal to zero. Since $R(\cdot)$ is invertible, if follows from (3.1) that $\dot{r}_A^{\mathbb{I}}(s) = 0,\ s \geq s_0$, if and only if $v_{A,e} = 0$. Similarly, since $\Gamma(\cdot)$ is invertible, it follows from (3.3) that

$$\begin{bmatrix} \dot{\phi}(s) & \dot{\theta}(s) & \dot{\psi}(s) \end{bmatrix}^{\mathrm{T}} = 0, \qquad s \geq s_0, \tag{3.16}$$

if and only if $\omega_e = 0$. Since $v_A(s) = 0,\ s \geq s_0$, and $\omega(s) = 0$, the aerodynamic forces and moments acting on the vehicle are equal to zero. That is, $F(s) = 0$ and $M(s) = 0$. In this case, it follows from (3.5) that $\phi_e = \theta_e = 0$, and $u_{1,e} = -mg$.

Moreover, it follows from (3.7) that

$$\begin{bmatrix} u_{2,e} \\ u_{3,e} \\ u_{4,e} \end{bmatrix} = I_P \sum_{i=1}^{4} \begin{bmatrix} 0 \\ 0 \\ \dot{\Omega}_{P,i}(s) \end{bmatrix}, \qquad s \geq s_0. \tag{3.17}$$

Modeling propellers as thin discs implies that the inertia matrix is an invertible diagonal matrix and hence,

$$u_{2,e} = 0, \tag{3.18}$$

$$u_{3,e} = 0, \tag{3.19}$$

$$u_{4,e} = I_{P,z}^{-1} \sum_{i=1}^{4} \dot{\Omega}_{P,i}(s), \qquad s \geq s_0, \tag{3.20}$$

where $I_{P,z}$ denotes the vehicle's moment of inertia about its spin axis. It is common practice to set $u_{4,e} = 0$. From this reasoning, it is apparent that any constant value of $r_{A,e} \in \mathbb{R}^3$ and $\psi_e \in \mathbb{R}$ is admissible. It is common practice to set $\psi_e = 0$ and $r_{A,e} = [0, 0, -z_e]^{\mathrm{T}}$, where $z_e \in \mathbb{R}$ denotes the desired altitude. Hence, it is common practice to set the equilibrium point to be

$$x_e = [0,\ 0,\ -z_e,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0]^{\mathrm{T}}, \tag{3.21}$$

$$u_e = [-mg,\ 0,\ 0,\ 0]^{\mathrm{T}}. \tag{3.22}$$

Therefore, the set of differential equations given by (3.1), (3.3), (3.12), and (3.13) can be linearized and reduced to the form

$$\dot{x}(s) = A[x(s) - x_e] + B[u(s) - u_e], \qquad x(s_0) = x_0 - x_e, \qquad s \geq s_0, \tag{3.23}$$

where $A \in \mathbb{R}^{12\times 12}$, $A_{1,7} = A_{2,8} = A_{3,9} = A_{4,10} = A_{5,11} = A_{6,12} = 1$, $A_{7,5} = -g$, $A_{8,4} = g$, $A_{i,j},\ i,j = 1,\ldots,12$, denotes the element on the ith row and jth column of A, every other element of A is equal to zero, $B \in \mathbb{R}^{12\times 4}$, $B_{9,1} = m^{-1}$, $B_{10,2} = I_x^{-1}$, $B_{11,3} = I_y^{-1}$, $B_{12,4} = I_z^{-1}$, and every other element of B is equal to zero.

3.4 Equations of Motion of a Quadrotor in Discrete Time

Computers, by their nature, perform mathematical calculations in discrete time. As such, we seek to transform the linearized continuous time model given by (3.23) in a linear time-invariant discrete system represented by

$$x[(t+1)T] = G(T)[x(tT) - x_e] + H(T)[u(tT) - u_e], \tag{3.24}$$

where $T > 0$ denotes the sampling time interval and $t \in \mathbb{N}$ denotes the current time step. Note that we assume that the input is subject to a zero-order hold [17], that is

$$u[(t+\delta)T] = u(tT), \qquad \forall(\delta, t, T) \in [0,1) \times \mathbb{N} \times \mathbb{R}_+. \tag{3.25}$$

The matrices $G(T)$ and $H(T)$ can be found by [18]

$$G(T) = e^{AT} \tag{3.26}$$

$$H(T) = \int_0^{\mathrm{T}} e^{A\gamma} d\gamma B, \qquad \gamma \in [0, tT), \tag{3.27}$$

where A and B are the system dynamics matrices in (3.23). For discussion on how to compute the matrix exponentials in (3.26) and (3.27), the reader is referred to [19, pp. 659-675].

A quadrotor is an *underactuated* mechanical system since its equations of motion are characterized by six independent generalized coordinates and four control inputs [15]. To design control strategies that allow quadcopters to reach a given position with a given yaw angle, control schemes for quadrotors are conventionally divided into two parts as depicted in Figure 3.2 [15]: an inner loop and an outer loop. Recall from (3.12) and (3.13) that the four control inputs regulate the altitude and orientation of the quadrotor. However, most practical applications for quadrotors require the regulation of the position and yaw angle. Not all of these vari-

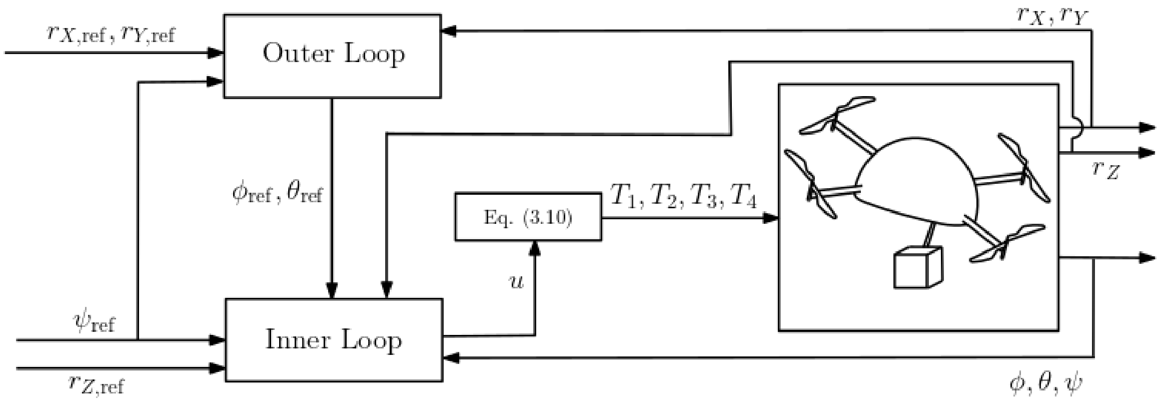


Figure 3.2: A quadrotor control scheme using inner loop - outer loop separation

ables are directly controllable and so much be reached through other variables that can be directly regulated. The control scheme described in Figure 3.2 assumes that $[r_{X,\text{ref}},\ r_{Y,\text{ref}},\ r_{Z,\text{ref}}]^{\mathrm{T}} : [s_0, \infty) \rightarrow \mathbb{R}^3$ and a desired yaw angle $\psi_{\text{ref}} : [s_0, \infty) \rightarrow [0, 2\pi)$ are known. First, we compute the roll angle $\phi_{\text{ref}} : [s_0, \infty) \rightarrow [0, 2\pi)$ and the pitch angle $\theta_{\text{ref}} : [s_0, \infty) \rightarrow (-(\pi/2), \pi/2)$ necessary to utilize the horizontal component of the thrust force to track $r_{X,\text{ref}}(s)$ and $r_{Y,\text{ref}}(s)$. This process is referred to as *outer-loop design.* Successively, the control vector $u(\cdot)$ is calculated to track the reference altitude $r_{Z,\text{ref}}(s)$ and the reference orientation $[\phi_{\text{ref}}(s), \theta_{\text{ref}}(s), \psi_{\text{ref}}(s)]^{\mathrm{T}}$. This process is known as *inner-loop design.* The FMPC algorithm generates a reference trajectory for the outer loop system.

4. Fast MPC Algorithm for Quadcopters

4.1 Introduction

The FMPC algorithm discussed in Chapter 2 is meant to be used on-board a system with fast dynamics. This makes it a strong candidate for implementation on quadrotors. Specifically, the FMPC algorithm presented in Chapter 2 has been tailored to generate reference trajectories for quadcopters by embedding the dynamical models discussed in Chapter 3. Moreover, in order to enable a tactical behavior, the FMPC algorithm presented in Chapter 2 has been modified to follow multiple waypoints and approach obstacles, which can shelter the aircraft from potential threats. The implementation of a quadcopter's linearized discrete-time dynamics in the FMPC algorithm presented in Chapter 2 is straightforward and hence, omitted for brevity. In the following, the approach used to induce a tactical behavior on the quadcopter is discussed in detail. Simulations are then presented that demonstrate the performance of FMPC.

4.2 Cost functions for FMPC and Tactical Behaviors of Quadcopters

In order to create a FMPC-based guidance algorithm for quadcopters operating in a contested environment, the underlying cost function has been modified as follows. Firstly, the underlying optimization problem captured by (2.23)–(2.24) has been modified by introducing soft constraints; for details see Section 2.4. In particular, soft constraint penalties were added to position variables in order to aid in obstacle avoidance. This change, in effect, enabled the creation of safety margins when defining obstacles.

In the FMPC formulation presented in Chapter 2, the cost function penalized the deviation of the state vector z from the origin. While this method is effective for studying the disturbance rejection properties of FMPC, it cannot be used to steer the system to a nonzero state. To this goal, the state vector z in (2.40) and (2.41) was replaced with $z_g = z - z_{\text{goal}}$ where $z_{\text{goal}} \in \mathbb{R}^{T(m+n)}$ denotes the target state. This change retains the quadratic shape of the cost function, does not require any change to the H matrix, and is valid for any distance between z and z_{goal}. The resulting quadratic optimization problem is cast as follows

$$\text{minimize:} \quad z_g^{\mathrm{T}} H z_g + g^{\mathrm{T}} z_g + \kappa\phi(z) + \theta(z), \tag{4.1}$$

$$\text{subject to:} \quad Cz = b. \tag{4.2}$$

This change allows for the implementation of waypoints that can be used to guide the vehicle around the environment. Note that, since z_{goal} is constant within a single FMPC loop, z_g is *not* a separate variable from z. Instead, z_g depends directly on z, which in turn remains the sole variable that is found by FMPC.

In order to enable a tactical behavior, whereby the quadcopter approaches obstacles in order to seek shelter while performing a mission, a cost was added to penalize the distance from the quadrotor to the nearest obstacle point denoted by $z_{\text{obs}} \in \mathbb{R}^{T(m+n)}$. This vector is held to be constant within each individual iteration of FMPC. The distance of z from z_{obs} is penalized and added to the QP described by

(4.1) and (4.2) as follows

$$\text{minimize:} \quad \bar{z}^{\mathrm{T}}\bar{H}\bar{z} + z_g^{\mathrm{T}}Hz_g + g^{\mathrm{T}}z_g + \kappa\phi(z) + \theta(z) \tag{4.3}$$

$$\text{subject to:} \quad Cz = b, \tag{4.4}$$

where $\bar{z} = z - z_{\text{obs}}$ and $\bar{H}$ denotes the cost on $\bar{z}$. Note that, much like in the case of z_g, $\bar{z}$ varies solely with z and the FMPC algorithm continues to solve for z only. This change propagates into the primal and dual residual given by (2.42) and (2.43), yielding an updated primal and dual residuals

$$\bar{r}_d(z) = 2\bar{H}\bar{z} + 2Hz_g + g + \kappa P^{\mathrm{T}}d(z) + \tilde{P}^{\mathrm{T}}\tilde{d}(z) + C^{\mathrm{T}}\nu \tag{4.5}$$

$$\bar{r}_p(z) = Cz - b. \tag{4.6}$$

The changes to the primal and dual residuals propagate into the Newton step calculation of Δz and $\Delta\nu$ given by (2.53). The resultant Newton step calculation is given by

$$\begin{bmatrix} \bar{\Phi}(z) & C^{\mathrm{T}} \\ C & 0 \end{bmatrix} \begin{bmatrix} \Delta z \\ \Delta\nu \end{bmatrix} = - \begin{bmatrix} \bar{r}_d(z) \\ r_p(z) \end{bmatrix}, \tag{4.7}$$

where

$$\bar{\Phi}(z) = 2\bar{H} + 2H + \kappa P^{\mathrm{T}}\mathrm{diag}(d(z))^2 P + \rho\tilde{P}^{\mathrm{T}}\mathrm{diag}(\tilde{d}(z))^2\tilde{P}. \tag{4.8}$$

The cost to reach the next waypoint may dominate the cost associated to coast obstacles, while performing a mission. For this reason, the vector z_g has been defined as follows

$$z_g = \begin{cases} z - z_{\text{goal}}, & ||z - z_{\text{goal}}||_2 < \gamma \\ \frac{\gamma}{||z - z_{\text{goal}}||_2}(z - z_{\text{goal}}), & \text{otherwise,} \end{cases} \tag{4.9}$$

where $\gamma > 0$ is a parameter chosen by the user.

4.3 Simulation Results

By implementing the changes to the cost function detailed in the previous section, a simulation can be developed to demonstrate the capabilities of FMPC. This simulation sets the vehicle in one corner of a box that is eight feet long on each side. Following the quadrotor equations of motion developed in Chapter 3, the vehicle is asked to travel from one corner to another as is illustrated in Figure 4.1. While

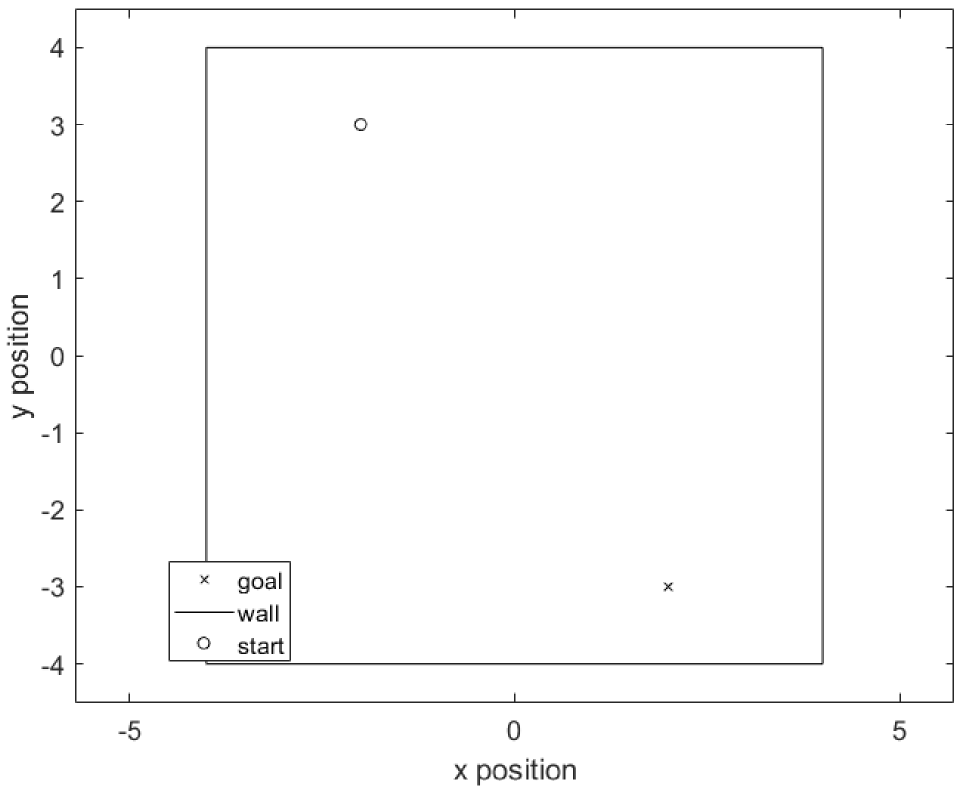


Figure 4.1: Environment for FMPC testing

this is a two dimensional representation, this is solely for ease of presentation. The simulation can easily be added to a three dimensional case.

When using the original cost function given by [5], the vehicle will proceed straight to the goal point. This result, however, is not a rigorous demonstration of the capabilities of FMPC. Instead, a simulation was developed with the objective of reaching the goal while remaining near the obstacles. This was accomplished using the alteration to the cost function given by (4.3) and (4.4). The result of this simulation is shown in Figure 4.2.

As can be seen in this simulation, the vehicle follows the wall to the goal point. However, it does not do so until it gets sufficiently close to the goal position. This

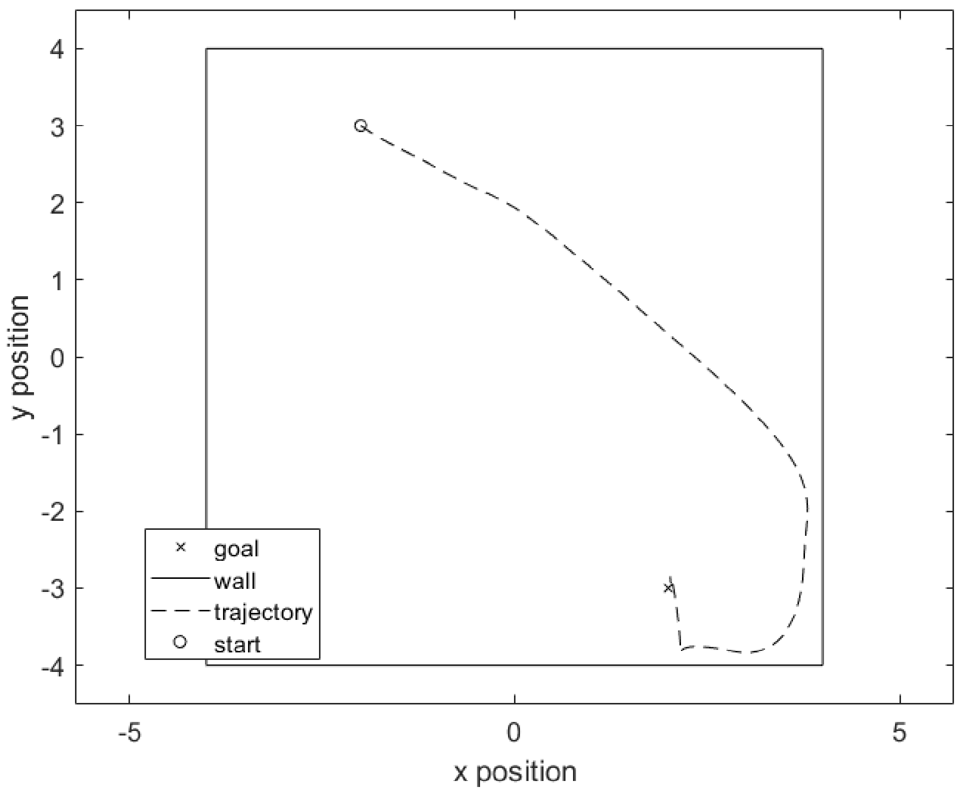


Figure 4.2: FMPC Simulation with obstacle attraction

issue was rectified by capping the penalty on distance to the goal point given by (4.9). With this change, the vehicle immediately approaches the wall and follows it around the box until it reaches the goal position. The result of this simulation is shown in Figure 4.3.

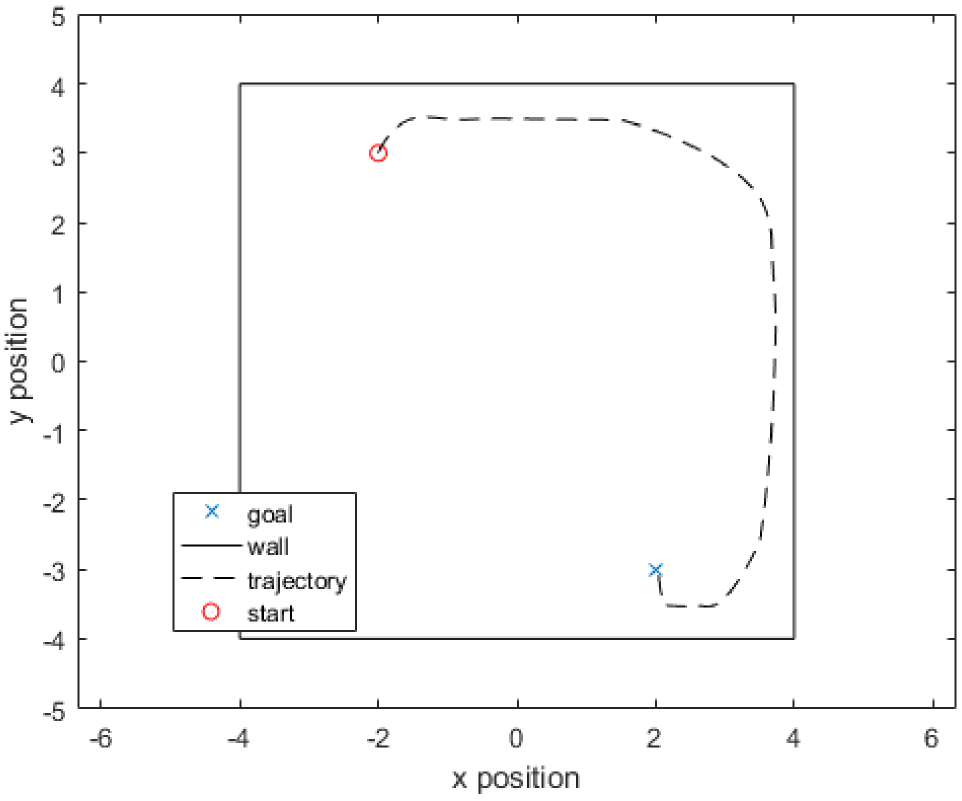


Figure 4.3: FMPC Simulation with obstacle attraction and capped cost on distance to the goal position

5. Affine Constraint Generation

5.1 Background and Motivation

The MPC algorithm employed in this thesis requires a linear plant model, affine constraints, and a quadratic cost function. However, the mission profile described in this thesis requires navigation through a non-convex operating region due to the presence of obstacles. Therefore, the algorithm presented in Chapter 2 requires modification in order to be applicable. In particular, a locally affine operating region must be identified for each computational step of FMPC.

A possible approach to the problem of creating affine constraints is to create the largest affine hull, which contains the quadcopter, whose vertices are denoted by $\mathcal{X}$, and excludes the obstacles detected by the navigation system, whose vertices are denoted by $\mathcal{Y}$. This is equivalent to finding a *convex hole* in the set $\mathcal{Y}$ that envelops the set $\mathcal{X}$. A convex hole of $\mathcal{Y}$ is defined as a convex polygon with vertices in $\mathcal{Y}$ that does not contain any point of $\mathcal{Y}$ in its interior [20]. While literature on the subject of convex holes exists, it is sparse [20] [21].

An alternative approach to this problem is plane detection in point clouds using Random Sample Consensus (RANSAC) [22]. RANSAC is a method of pattern recognition in which a small, random set of points is chosen and neighboring points that are consistent with the target pattern are found [23]. This method is employed in computer vision to analyze point clouds and is freely available through libraries such as the Point Cloud Library (PCL) [24]. Having a readily made implementation available makes plane detection an attractive option. However, there are a few considerations that led to it not being the method implemented. At the time where a constraint generation method had to be decided upon, the navigation system was still early in development. Since the form of the navigation system's data had not yet been determined, there was uncertainty about how that data would be imported in the plane detection algorithm. Even if the algorithm works correctly, plane detection tends to find several small planes. This issue translates to an increase in run time as there are significantly more constraints to satisfy and therefore requires significantly more memory. Additionally, the algorithm would detect planes along the entire map. This opens the possibility of generating constraints that are immediately violated. These constraints would have to be detected and excluded, further slowing down the run time.

To limit the amount of constraints that are generated, a linear point separation algorithm has been considered. Linear point separation finds a hyperplane that separates one set of points from another [11]. If the obstacle data is sorted into discrete obstacles, then the linear point separation algorithm can be used to find a single constraint for each obstacle. The resultant constraint is guaranteed to be satisfied by the vehicles current state. There are issues with this method that are very similar to the issues with plane detection. The navigation data can be organized into clusters using PCL [24]. However, there are cases where the clustering algorithm can cause problems. For instance, if the cluster is non-convex, there is no guarantee that a linear separation hyperplane can be found.

To solve this clustering issue, a quadratic discrimination algorithm is utilized. In this method, an ellipsoid is found that separates the vehicle data points from every obstacle data point. This avoids the pitfalls of linear discrimination by avoiding the clustering step entirely. The vertices of the ellipsoid are then used to generate hyperplanes that separate the set of the quadcopter's vertices $\mathcal{X}$ from the set of the obstacles' vertices $\mathcal{Y}$.

This chapter will develop a constraint generation algorithm to create a convex operating region for the vehicle. The linear point separation algorithm and the issues with it will be detailed. From there, the process of finding a quadratic discrimination through SDP will be detailed and a method of generating constraints from the result will be developed. Finally, pseudocode for implementing this constraint generation algorithm will be presented.

5.2 Point Separation via Quadratic Discrimination

In order for a quadratic discrimination algorithm to be implemented, it is necessary that $\text{conv}(\mathcal{X}) \cap \text{conv}(\mathcal{Y}) = 0$, where $\mathcal{X}$ denotes the set of N points representing the quadrotor and $\mathcal{Y}$ denotes the set of M points representing the obstacles and $\text{conv}(\cdot)$ denotes the convex hull of a region, that is, the smallest convex region that envelops the set [11, pp. 24]. This necessary condition is always met as violating it would indicate a collision.

Quadratic discrimination is a special case of nonlinear discrimination, which seeks to find a nonlinear function f such that

$$f(x_i) < 0, \qquad i = 1, \ldots, N, \tag{5.1}$$

$$f(y_j) > 0, \qquad j = 1, \ldots, M. \tag{5.2}$$

where x_i and y_j denote the ith and jth point in $\mathcal{X}$ and $\mathcal{Y}$, respectively. In the case of quadratic discrimination, we consider

$$f(x) = x^{\mathrm{T}} P x + q^{\mathrm{T}} x + r, \tag{5.3}$$

where $x \in \mathbb{R}^n$, $P \in \mathbb{R}^{n \times n}$, $q \in \mathbb{R}^n$, and $r \in \mathbb{R}$. To find a function $f(\cdot)$ that satisfies the quadratic discrimination, we can solve the feasibility problem

$$\begin{aligned} \text{minimize:} \quad & 0, && (5.4) \\ \text{subject to:} \quad & x^{\mathrm{T}} P x + q^{\mathrm{T}} x + r \leq 0, \qquad \forall x \in \mathcal{X}, && (5.5) \\ & y^{\mathrm{T}} P y + q^{\mathrm{T}} y + r \geq 0, \qquad \forall y \in \mathcal{Y}, && (5.6) \end{aligned}$$

which is a special form of a quadratically constrained quadratic program (QCQP). In general, solving a QCQP is NP-hard [25], but the complexity can be reduced in special circumstances allowing us to find solutions in polynomial time. One such special circumstance occurs when both the constraints and the cost are convex, which can be achieved by enforcing that $P = P^{\mathrm{T}} > 0$. This additional constraint has two advantages. First, a positive-definite matrix P implies that the quadratic function describes an ellipsoid [26]. An ellipsoidal function surrounding the quadrotor set $\mathcal{X}$ would serve as a locally convex region that FMPC can operate within. The other advantage is that a convex QCQP can be refactored into a Semidefinite Programming problem and efficiently solved using a C++ library such as SDPA [27].

5.2.1 Semidefinite Programming

A semidefinite program (SDP) is an optimization problem of the form

$$\text{minimize:} \quad c^{\mathrm{T}} x, \tag{5.7}$$

$$\text{subject to:} \quad F(x) \geq 0, \tag{5.8}$$

where $c \in \mathbb{R}^m$, $x \in \mathbb{R}^m$,

$$F(x) \triangleq F_0 + \sum_{i=1}^{m} x_i F_i, \tag{5.9}$$

the matrices $F_0, \ldots, F_m \in \mathbb{R}^{n \times n}$ are symmetric, and $x_i \in \mathbb{R},\ i = 1, \ldots, m$, denotes the ith component of x. SDP problems can be solved in polynomial time using, for instance, the primal-dual interior point method [28]. One such method is the primal-dual interior point method employed by SDPA.

Comparing (5.9) to (5.5), it is apparent that the quadratic discrimination problem is not in the same form as an SDP. To resolve this issue, the constraints need to be rearranged to a linear combination of matrices scaled by some variable. Specifically, we expand P and q such that

$$P = \begin{bmatrix} P_{11} & P_{12} & P_{13} \\ P_{12} & P_{22} & P_{23} \\ P_{13} & P_{23} & P_{33} \end{bmatrix}, \qquad q = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix}, \tag{5.10}$$

and define

$$z \triangleq \begin{bmatrix} P_{11} & P_{22} & P_{33} & P_{12} & P_{13} & P_{23} & q_1 & q_2 & q_3 & r \end{bmatrix}^{\mathrm{T}}. \tag{5.11}$$

Next, consider the constraint (5.6) and let

$$y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \in \mathcal{Y}, \tag{5.12}$$

In this case, (5.6) is equivalent to

$$y_1^2 P_{11} + y_2^2 P_{22} + y_3^2 P_{33} + 2y_1y_2P_{12} + 2y_1y_3P_{13} + 2y_2y_3P_{23} + y_1q_1 + y_2q_2 + y_3q_3 + r \geq 0. \tag{5.13}$$

The same approach can be used to capture (5.5) in the same form as (5.13) and, collecting terms, the matrices $F_0, \ldots, F_m$ are readily generated. For instance, F_1 is given by

$$F_1(z) = \begin{bmatrix} -x_{1,1}^2 & & & & & \\ & \ddots & & & & \\ & & -x_{M,1}^2 & & & \\ & & & y_{1,1}^2 & & \\ & & & & \ddots & \\ & & & & & y_{N,1}^2 \end{bmatrix}. \tag{5.14}$$

In order to enforce positive-definiteness of P, we impose that $P - I \geq 0$. Thus, the constraints for the SDP problem are captured by

$$\overline{F}(z) = \begin{bmatrix} P - I & \\ & F(z) \end{bmatrix} \geq 0, \qquad z \in \mathbb{R}^m, \tag{5.15}$$

and the quadratic optimization problem (5.4)–(5.6) is equivalent to

$$\text{minimize} \quad 0 \tag{5.16}$$

$$\text{subject to} \quad \overline{F}(z) \geq 0. \tag{5.17}$$

We can further extend this SDP to minimize the distance from the ellipsoid to the obstacle points. Accomplishing this task requires two modifications to the SDP defined in (5.16)–(5.17). First, we augment z with the term

$$\gamma = \min\{y^{\mathrm{T}}Py + q^{\mathrm{T}}y + r : \; y \in \mathcal{Y}\}. \tag{5.18}$$

so that

$$\overline{z} \triangleq [z^{\mathrm{T}}, \gamma]^{\mathrm{T}}. \tag{5.19}$$

Similarly, we introduce the matrix

$$F_{11}(z) = \begin{bmatrix} 0_M & \\ & -I_N \end{bmatrix}, \tag{5.20}$$

where 0_M denotes the $M \times M$ zero matrix and I_n denotes the $N \times N$ identity matrix and is added to the sum in (5.9). This addition can be interpreted as capturing the distance γ from the boundary of the separating ellipsoid to each obstacle point. Minimizing this value generates an ellipsoid as close to the obstacle as possible. In this case, the quadratic separation problem is given by

$$\text{minimize:} \quad \gamma, \tag{5.21}$$

$$\text{subject to:} \quad \overline{F}(\overline{z}) \geq 0, \tag{5.22}$$

where $\overline{F}(\overline{z})$ is constructed by proceeding as in (5.9).

This formulation allows us to find *an* ellipsoid separating the two sets and that is tangent to the obstacle data. While this formulation is a step in the correct direction, there is still room for development since γ only tracks the minimum distance from the ellipsoid to the obstacle. Once a minimum value is found, makes no attempt to minimize the distance to other obstacle points. This is an area of future research.

5.2.2 Constraint Generation

Quadratic discrimination was chosen due to its versatility. However, it is not without its own drawbacks. Namely, the FMPC algorithm is only capable of operating with affine constraints. Since ellipsoids are not affine, a possible approach is to use the generated ellipsoid in FMPC is to approximate the shape of the ellipsoid with affine constraints. There is no single way to make this approximation as more hyperplanes can always be generated to more closely approximate the ellipsoid. In the end, the resolution of the approximation is a parameter to be determined by the user.

The process of extracting affine constraints from a quadratic equation involves finding a representative sample of points on the ellipsoid and generating a hyperplane tangent to the ellipsoid at each sampled point. This process is repeated periodically to update the constraints based on the vehicles's new position. In order to find this sampling, the center of the ellipsoid must be found first. Towards this goal, we first define the external boundary of the ellipsoid

$$\mathcal{E} \triangleq \{x \in \mathbb{R}^3 :\ x^{\mathrm{T}} P x + q^{\mathrm{T}} x + r = 0\}, \tag{5.23}$$

where $P = P^{\mathrm{T}} > 0 \in \mathbb{R}^{3\times 3}$, $q \in \mathbb{R}^3$, and $r \in \mathbb{R}$. This representation is not the only one possible, however. For instance, the same ellipsoid boundary can be represented by

$$\mathcal{E} = \{x \in \mathbb{R}^3 :\ (x - x_c)^{\mathrm{T}} P (x - x_c) = k\}, \tag{5.24}$$

where $x_c \in \mathbb{R}^3$ denotes the location of the center of the ellipsoid and $k > 0$ since

$P > 0$. The center point x_c is defined as the location where the ellipsoid's three planes of symmetry intersect. Since the representations in (5.23) and (5.24) are equivalent, we can derive an equation to find x_c in terms of P, q, and r. We start by expanding the ellipsoid equation given in (5.24) which yields

$$x^{\mathrm{T}}Px - x^{\mathrm{T}}Px_c - x_c^{\mathrm{T}}Px + x_c^{\mathrm{T}}Px_c = k. \tag{5.25}$$

Note that $P = P^{\mathrm{T}}$ and thus

$$(x^{\mathrm{T}}Px_c)^{\mathrm{T}} = x_c^{\mathrm{T}}P^{\mathrm{T}}x = x_c^{\mathrm{T}}Px. \tag{5.26}$$

Since $x^{\mathrm{T}}Px_c$ is scalar, we can establish the relation

$$(x^{\mathrm{T}}Px_c)^{\mathrm{T}} = x^{\mathrm{T}}Px_c = x_c^{\mathrm{T}}Px. \tag{5.27}$$

Substituting this relation back into (5.25) yields

$$x^{\mathrm{T}}Px - 2x_c^{\mathrm{T}}Px + x_c^{\mathrm{T}}Px_c - k = 0. \tag{5.28}$$

Now, let $q = -2Px_c$ and $r = x_c^{\mathrm{T}}Px_c - k$. This substitution both brings us to the original formulation in (5.23) and reveals a direct formulation of the center of the ellipsoid given by

$$x_c = -\frac{1}{2}P^{-1}q. \tag{5.29}$$

From the center of the ellipsoid, a series of sample points can be found along the boundary of the ellipsoid. In particular, we want to identify a set of points $x \in \mathcal{E}$. To this goal, note that it follows from (5.23) that

$$x^{\mathrm{T}} P x + q^{\mathrm{T}} x + \frac{1}{4} q^{\mathrm{T}} P^{-1} q = -r + \frac{1}{4} q^{\mathrm{T}} P^{-1} q = k. \tag{5.30}$$

To show that the relation with k above is true, we define

$$c \triangleq -r + \frac{1}{4} q^{\mathrm{T}} P^{-1} q. \tag{5.31}$$

Substituting $q = -2Px_c$ and $r = x_c^{\mathrm{T}} P x_c - k$ into (5.31) yields

$$c = k - x_c^{\mathrm{T}} P x_c + \frac{1}{4}(-2Px_c)^{\mathrm{T}} P^{-1}(-2Px_c) = k, \tag{5.32}$$

which proves the relation in (5.30). Returning to (5.30), consolidating the left hand side into a quadratic form and dividing by k yields the standard form of the ellipsoid equation

$$\left(x + \frac{1}{2} P^{-1} q\right)^{\mathrm{T}} \frac{P}{k} \left(x + \frac{1}{2} P^{-1} q\right) = 1, \qquad x \in \mathbb{R}^3. \tag{5.33}$$

Note that $-\frac{1}{2}P^{-1}q$ captures the center of the ellipsoid as detailed in (5.24) and thus $-x_c$ can be substituted into (5.33). Next, since $P = P^{\mathrm{T}}$ and

$$\sqrt{\frac{P}{k}} = \sqrt{\frac{P^{\mathrm{T}}}{k}} = \left(\sqrt{\frac{P}{k}}\right)^{\mathrm{T}}, \qquad P \in \mathbb{R}^{3\times 3}, \tag{5.34}$$

(5.33) is equivalent to

$$(x - x_c)^{\mathrm{T}} \left(\sqrt{\frac{P}{k}}\right)^{\mathrm{T}} \left(\sqrt{\frac{P}{k}}\right) (x - x_c) = 1, \tag{5.35}$$

where $\sqrt{P}$ is such that $\sqrt{P}\sqrt{P} = P$. Lastly, recall that $x^{\mathrm{T}}x = ||x||^2$, consider the unit vector

$$\begin{bmatrix} \cos\theta \ \cos\phi \\ \cos\theta \ \sin\phi \\ \sin\theta \end{bmatrix}, \tag{5.36}$$

where $\theta \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$, and $\phi \in [0, 2\pi)$ and note that (5.35) is equivalent to

$$(x - x_c)^{\mathrm{T}} \left(\sqrt{\frac{P}{k}}\right)^{\mathrm{T}} \left(\sqrt{\frac{P}{k}}\right) (x - x_c) = \begin{bmatrix} \cos\theta \ \cos\phi \\ \cos\theta \ \sin\phi \\ \sin\theta \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} \cos\theta \ \cos\phi \\ \cos\theta \ \sin\phi \\ \sin\theta \end{bmatrix}, x \in \mathbb{R}^3, \tag{5.37}$$

which yields

$$x = \left(\sqrt{\frac{P}{k}}\right)^{-1} \begin{bmatrix} \cos\theta \ \cos\phi \\ \cos\theta \ \sin\phi \\ \sin\theta \end{bmatrix} + x_c. \tag{5.38}$$

Now that a method for finding a generic point $x \in \mathcal{E}$ has been established, we may discuss a method to systematically sample the ellipsoid and create a finite subset $\mathcal{S} \subset \mathcal{E}$. The number of points in $\mathcal{S}$ is dependent on the number of sample angles $\theta \in [-\frac{\pi}{2}, 0]$ and $\phi \in [0, 2\pi)$ that the user wishes to search along. Setting the domain of θ as $[-\frac{\pi}{2}, 0]$ instead of $[-\frac{\pi}{2}, \frac{\pi}{2}]$ allows us to calculate the result of (5.38) for half of the ellipsoid which still provides enough information to find constraint equations approximating the entire ellipsoid as will be detailed later. Once values for θ and ϕ have been defined, the angle θ is fixed and a sample is taken for each ϕ before moving on to the next value of θ. This process is repeated until $\mathcal{S}$ is fully populated.

With a representative sample of points on the ellipsoid boundary found, it is now possible to generate a set of position constraints that approximate the shape of the ellipsoid. This is done by finding the gradient of (5.3) which is given by

$$\nabla f(x) = 2Px + q, \qquad x \in \mathbb{R}^3. \tag{5.39}$$

Each sampled point can be input into this equation to determine the gradient for that instance. This gradient is, in turn, used to generate a plane equation of the form

$$\nabla f^{\mathrm{T}}(x)x = 2x^{\mathrm{T}}Px + q^{\mathrm{T}}x = x^{\mathrm{T}}Px - r, \tag{5.40}$$

where

$$-r = x^{\mathrm{T}}Px + q^{\mathrm{T}}x, \tag{5.41}$$

as follows from (5.23). At this point, the equation for the constraint has been found. However, a quick sign check must be done to make sure that the constraint, for all x in the operating region, is in the form

$$F_x x \leq\leq f_x, \tag{5.42}$$

as is expected by FMPC; F_x and f_x are to be determined in the following. The point x_c serves as an excellent test case as it must lie within the operating region and has already been calculated. Therefore, we record the spatial constraints according to

$$F_x = \begin{cases} \nabla f^{\mathrm{T}}(x), & \nabla f^{\mathrm{T}}(x) x_c \leq \nabla f^{\mathrm{T}}(x) x \\ -\nabla f^{\mathrm{T}}(x), & \nabla f^{\mathrm{T}}(x) x_c > \nabla f^{\mathrm{T}}(x) x \end{cases} \tag{5.43}$$

$$f_x = \begin{cases} \nabla f^{\mathrm{T}}(x) x, & \nabla f^{\mathrm{T}}(x) x_c \leq \nabla f^{\mathrm{T}}(x) x \\ -\nabla f^{\mathrm{T}}(x) x, & \nabla f^{\mathrm{T}}(x) x_c > \nabla f^{\mathrm{T}}(x) x \end{cases}. \tag{5.44}$$

Soft constraints can be added on the inside of each constraint to create a safety margin. Let $\delta \in \mathbb{R}$ denote the size of this safety margin. Then, for each hard constraint, there is a related soft constraint of the form

$$\tilde{F}_x x \leq\leq \tilde{f}_x, \tag{5.45}$$

where $\tilde{F}_x \in \mathbb{R}^3$ and $\tilde{f}_x \in \mathbb{R}$. The coefficients of these soft constraints are given b

$$\tilde{F}_x = F_x \tag{5.46}$$

$$\tilde{f}_x = f_x - \delta. \tag{5.47}$$

Once constraints have been generated for each sample point, half of the ellipsoid has been approximated. We can take advantage of the symmetry of ellipsoids to generate the other half of constraints. Due to this symmetry, each constraint has a specular constraint across all three planes of symmetry. Alternatively, we can say that the two parallel hyperplanes comprising the constraint and its specular partner are equidistant from the center. To find the relation between the original constraint and the mirrored constraint, we let $d = \nabla f^{\mathrm{T}}(x)x_c$ and $\epsilon = \nabla f^{\mathrm{T}}(x)x - d$. The equation of the mirrored constraint is found by translating the original plane by a factor of -2ϵ towards x_c. The resultant equations are given by

$$F_{x,m} = -F_x \tag{5.48}$$

$$f_{x,m} = -(f_x - 2\epsilon), \tag{5.49}$$

where the m in the subscript denotes a mirrored constraint.

5.3 Implementation

In this section, we gather the constraint generation method detailed in this chapter into a single routine. This routine is split into three separate functions. The first function is a direct application of the quadratic discrimination separating the vehicle from the obstacle as described in Section 5.2. The routine for converting the resultant ellipsoid into constraints as described in Section 5.2.2 is split into the second and third algorithms. The second algorithm finds a sampling of external points on the ellipsoid while the third generates constraints based on these sampled points.

A pseudocode representation of the quadratic discrimination algorithm is given by Algorithm 1. The heavy lifting within this algorithm is done by the SDPA library which handles the semidefinite programming optimization. As such, the algorithm centers entirely around organizing the problem information into a format that is suitable for SDPA. Once the obstacle point cloud data and external vehicle points are read in from the navigation system, the number of data points used in solving the SDP can be determined. Similarly, the number of variables involved in solving the problem is calculated. From this information, the structure of the problem can be initialized. In the case of quadratic discrimination, the constraints are composed of a block diagonal matrix which is, in turn, composed of two blocks. The first block represents $P - I \in \mathbb{R}^{3\times 3}$. The second block is a diagonal matrix of size $(M + N) \times (M + N)$ that contains the coefficients in (5.13) based on each vehicle and obstacle point. The second block can be populated with the coefficients in (5.13) for each point in $\mathcal{X}$ and $\mathcal{Y}$. Once the problem is fully loaded, SDPA solves the problem and outputs an ellipsoid equation.

Algorithm 1 Quadratic Discrimination Algorithm

```
 1: Input:   obstacle point cloud data from navigation system
 2: procedure QUADRATICDISCRIMINATION
 3:     $\mathbf{n} = 3 \leftarrow$ number of states
 4:     $\mathbf{ev} \leftarrow$ number of extra variables in SDP
 5:     Read obstacle data points $\mathbf{y} \in \mathbb{R}^{\mathbf{n}}$ into $\mathcal{Y}$
 6:     Read extreme vehicle points $\mathbf{x} \in \mathbb{R}^{\mathbf{n}}$ into $\mathcal{X}$
 7:     $\mathbf{M} \leftarrow$ number of points $\mathbf{y} \in \mathcal{Y}$
 8:     $\mathbf{N} \leftarrow$ number of points $\mathbf{x} \in \mathcal{X}$
 9:     Initialize problem structure for use with SDPA
10:     $\mathbf{nConstraints} = (\mathbf{n} * \mathbf{n} + \mathbf{n})/2 + \mathbf{n} + 1 + \mathbf{ev} \leftarrow$ number of constraints in SDP
11:     Populate first block in SDPA constraints with $P - I$
12:     for $(i = 0;\ i < M;\ i{+}{+})$ do
13:         Populate $\overline{F}_j(\cdot)$ for $j = 1, \ldots, \mathbf{nConstraints}$ based on $\mathbf{y}_i$
14:     for $(i = 0;\ i < N;\ i{+}{+})$ do
15:         Populate $\overline{F}_j(\cdot)$ for $j = 1, \ldots, \mathbf{nConstraints}$ based on $\mathbf{x}_i$
16:     Solve SDPA
17:     Output result of SDPA as array of floats
```

The output of Algorithm 1 is the ellipsoid equation given by P, q, and r and is then passed to Algorithm 2. This algorithm finds a sampling of points describing the ellipsoid based off of the ellipsoid equation. Overall, this algorithm is a straightforward implementation of the sampling process described in the previous subsection. However, there are some features that should be noted. First, the number of points to be sampled is determined by the user selecting how many angles θ and ϕ should be searched. Careful examination of the for loops in Algorithm 2 reveals irregularities. The counter i starts at one because the first angle $\theta = -\frac{\pi}{2}$ is calculated separately in order to avoid duplicate sample points. For similar reasons, the counter j omits the last angle of *nopts* as it is identical to the first angle of nopts. It is possible that the navigation system will create a map that contains false positives. If one of these false obstacle points intersects with the vehicle point set, the SDPA algorithm will output numbers that do not represent an ellipsoid. We detect this case when $k \leq 0$ and abort the constraint generation.

Algorithm 2 Algorithm to find representative points on ellipsoid surface

```
Input: $P, q, r$
 1: procedure sampleEllipsoid
 2:     nangles ← number of equidistant angles $\theta \in [-\frac{\pi}{2}, 0]$
 3:     nopts ← number of equidistant angles $\phi \in [0, 2\pi]$
 4:     xc $= -\frac{1}{2}P^{-1}q$ ← center of the ellipsoid
 5:     k $= \frac{1}{4}q^{\mathrm{T}}P^{-1}q - r$
 6:     if c $\leq 0$ then Return 1
 7:     rootP $= \left(\sqrt{\frac{P}{k}}\right)^{-1}$
 8:     sample ← ellipsoid point sampled at $\theta = -\frac{\pi}{2}$ according to (5.38)
 9:     for $i = 1; i <$ nangles do
10:         for $j = 0; j <$ nopts $- 1$ do
11:             sample ← point on ellipsoid at $\theta(i)$ and $\phi(j)$ using (5.38)
12:     Output sample and return 0
```

If the sampling algorithm completes successfully, then new constraints are generated for each point in Algorithm 3. Much like in Algorithm 2, this algorithm is a straightforward implementation of the constraint generation method described in the previous section with a few idiosyncrasies. The first such item involves the number of points whose related constraints should be mirrored. Every point except the last ring of samples should be mirrored. Therefore, the number of mirrored points is the number of points minus $nopts - 1$. This same logic is applied to determine the number of constraints that will be generated. Doubling the number of points would mirror all sampled constraints. However, we don't want to mirror the last sampling angle, so we subtract the number of points in that sampling angle.

This procedure is executed periodically during each flight in order to find locally convex regions around the constantly changing position of the vehicle. This constant recalculation allows the exception handling procedure in Algorithm 2 to be survivable. If a sampling must be discarded, then the constraints that were last successfully found can be used again.

Algorithm 3 Algorithm to generate constraints based on ellipsoid

Input: The ellipsoid sample **sample**, the number of points in this sample **numPoints**, the ellipsoid equation $\mathbf{P}, \mathbf{q}, \mathbf{r}$, the number of states **n**, the number of angles of θ **nopts**, the number of angles of ψ **nangles**

```
 1: procedure GENERATECONSTRAINT
 2:     Fsize ← number of constraints to be generated
 3:     $\mathbf{Fsize} = 2 * \mathbf{numPoints} - \mathbf{nopts} + 1$
 4:     mirroredPoints ← number of constraints to be mirrored
 5:     $\mathbf{mirroredPoints} = \mathbf{numPoints} - \mathbf{nopts} + 1$
 6:     for $(i = 0;\ i < \mathbf{mirroredPoints};\ i++)$ do
 7:         $\mathbf{gradient} = 2P * \mathbf{sample}(i) + \mathbf{q}$
 8:         $\mathbf{D} = \mathbf{gradient}^{\mathrm{T}}\mathbf{sample}(i)$
 9:         $\mathbf{centerD} = \mathbf{gradient}^{\mathrm{T}}\mathbf{xc}$
10:         $\epsilon = \mathbf{D} - \mathbf{centerD}$ ← distance from center
11:         if $\epsilon > 0$ then
12:             $\mathbf{D} = -\mathbf{D}$
13:             $\mathbf{gradient} = -\mathbf{gradient}$
14:         $\mathbf{F},\ \mathbf{f}$ ← hard constraints
15:         $\tilde{\mathbf{F}},\ \tilde{\mathbf{f}}$ ← soft constraints
16:         $\mathbf{F}(2i) = \mathbf{gradient}$
17:         $\mathbf{f}(2i) = \mathbf{D}$
18:         $\tilde{\mathbf{F}}(2i) = \mathbf{gradient}$
19:         $\tilde{\mathbf{f}}(2i) = \mathbf{D} - \mathbf{offset}$
20:         $\mathbf{D} = \mathbf{D} + 2\epsilon$ ← translate the constraint to other side
21:         $\mathbf{F}(2i+1) = -\mathbf{gradient}$
22:         $\mathbf{f}(2i+1) = -\mathbf{D}$
23:         $\tilde{\mathbf{F}}(2i+1) = -\mathbf{gradient}$
24:         $\tilde{\mathbf{f}}(2i+1) = -\mathbf{D} - \mathbf{offset}$
25:     for $i = 0;\ i < \mathbf{nopts} - 1;\ i++)$ do
26:         $\mathbf{gradient} = 2P * \mathbf{sample}(i) + \mathbf{q}$
27:         $\mathbf{D} = \mathbf{gradient}^{\mathrm{T}}\mathbf{sample}(i)$
28:         $\mathbf{centerD} = \mathbf{gradient}^{\mathrm{T}}\mathbf{xc}$
29:         $\epsilon = \mathbf{D} - \mathbf{centerD}$
30:         if $\epsilon > 0$ then
31:             $\mathbf{D} = -\mathbf{D}$
32:             $\mathbf{gradient} = -\mathbf{gradient}$
33:         $\mathbf{F}(\mathbf{nangles} + i) = \mathbf{gradient}$
34:         $\mathbf{f}(\mathbf{nangles} + i) = \mathbf{D}$
35:         $\tilde{\mathbf{F}}(\mathbf{nangles} + i) = \mathbf{gradient}$
36:         $\tilde{\mathbf{f}}(\mathbf{nangles} + i) = \mathbf{D} - \mathbf{offset}$
```

6. Guidance Systems for UAVs

6.1 Introduction

Thus far, this thesis has focused on developing the FMPC algorithm and developing a method to allow it to operate in a non-convex environment. However, FMPC on its own is not sufficient to act as a guidance system. This is due to the greedy nature of the gradient descent method. The FMPC algorithm will always move in a direction that minimizes the cost function. Once it reaches a minimum point, every direction results in a higher cost and the drone therefore does not move. In a convex operating region, this is not an issue as any locally optimal point is also globally optimal [11]. However, this property does not hold in non-convex regions. This opens the possibility of the quadrotor becoming stuck at a locally optimal point and failing to reach its goal position.

To overcome this issue, a global path planner can be utilized. Pathfinding is an old problem in computer science and thus several algorithms exist. A desirable algorithm will be able to guarantee that a path from any starting point to the goal point will be found if such a path exists. Typically, pathfinding algorithms are concerned with finding the shortest route. However, this does not match our use case as we wish to divert from the shortest path to seek cover along the way. Therefore, the ideal algorithm will include a heuristic that allows for solutions to be found for multiple mission profiles.

One potential option is Dijkstra's algorithm. At its core, Djikstra's algorithm is a solution to the shortest path problem developed in 1959 [29]. This algorithm begins with a weighted graph, that is, a list of nodes, the connections between them, and the costs to move along these connections. An example of this kind of weighted graph is illustrated in Figure 6.1 [1]. In this example, the cost to travel between two nodes is denoted by the number above the line connecting them while the remaining distance to the goal is marked above each node in parentheses. The shortest path is marked by the blue line. Djikstra's algorithm explores the network by travelling to the node with the lowest cost. From this new node, the cost to travel to each neighboring node is calculated. If the calculated cost for a neighbor is less than the previously calculated cost to travel to it along other paths, then the cost is updated to the lower cost and the path to the neighbor is recorded as going through the current node. The active node is then switched to the node with the lowest cost and the process is repeated until the goal node is visited.

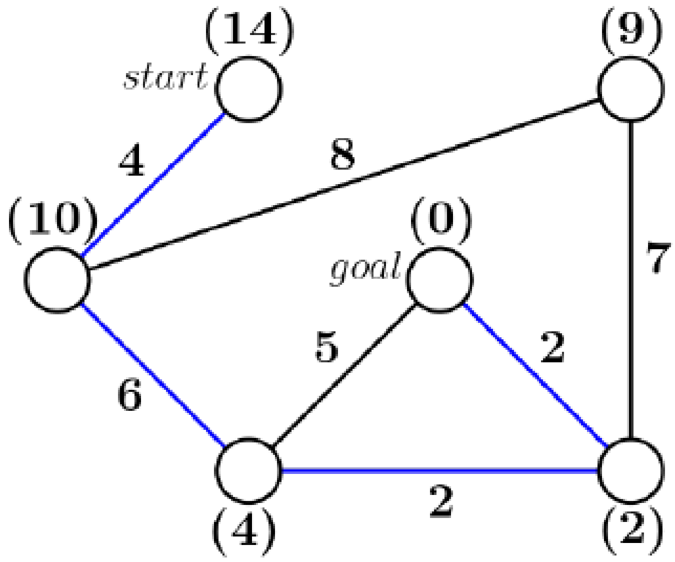


Figure 6.1: Example of a node map

Djikstra's algorithm succeeds as it is guaranteed to find a path to the goal and this path is guaranteed to be the shortest possible path, so long as any such path exists. This property is due to the search pattern applied by the algorithm. Since the lowest cost nodes are explored first, the search for the goal node is conducted along the shortest available path. However, it is not the best algorithm for our purposes. The first issue is that the shortest path is not necessarily the desired path that drives the vehicle to a goal position while staying near obstacles. Secondly, while Djikstra's algorithm finds the shortest path, it does so through brute force. Performance can be improved by implementing a heuristic function to guide the search to the goal.

6.2 Navigation Using A*

One algorithm that uses this approach is the *A* search algorithm*, simply called A* henceforth. A* is a direct descendent of Dijkstra's algorithm and operates in a nearly identical way. The sole difference comes from the addition of a heuristic value to the cost to reach a given neighbor. Including a heuristic function not only can improve performance, but can also potentially be designed to impose certain characteristics on the solution.

The heuristic function provides an estimate of the cost to reach the goal from some node. The form this function takes is dependent on the form of the graph being transversed. In our case using spatial maps, the Euclidean morm is a convenient choice as it will encourage the system to search the nodes nearest to the goal first. In an ideal scenario where the shortest path is unobstructed, the heuristic will cause the A* algorithm to search only the nodes along the shortest path. By contrast, Djikstra's algorithm would search in all directions and thus be less efficient in most cases.

We can begin building towards such an algorithm by identifying one that is suited for the dynamic and initially unknown environment into which the quadrotor will eventually be deployed. Here, we consider the Lifelong Planning A* algorithm (LPA*) [30]. This algorithm repeatedly calculates the shortest path from start to goal despite the addition or deletion of candidate nodes and changing costs. The initial step of LPA* works identically to base A*. However, instead of terminating the program once a shortest path is found, LPA* saves the cost for each node to reach the goal. This cost, referred to as the g-value, satisfies

$$g(n') = \begin{cases} 0 & \text{if } n' = n_{\text{start}} \\ \min\limits_{n} g(n) + k(n', n) & \text{otherwise,} \end{cases} \tag{6.1}$$

where n_{start} is the starting node, $g(\cdot)$ denotes a previously calculated result of (6.1), and $k(\cdot, \cdot)$ denotes the smallest cost between the two input nodes. When (6.1) is satisfied, the node is considered to be *locally consistent*. Instead of recording the previous path and calculating the subsequent path based upon it, the algorithm instead records g-values from the previous iteration. When the map is updated, the algorithm detects g-values which are no longer locally consistent and adds them to a queue. By following the heuristic function, only the g-values relevant to the path are recalculated. With this updated information, the optimal path is recalculated and the process is repeated until the quadrotor reaches the goal.

6.3 Navigation Using MPL

Building from LPA*, we discuss the Motion Primitive Library (MPL) which is a C++ implementation of [31] and [32]. This method involves the generation of *motion primitives* that convert an optimal control problem to a graph search problem solvable with LPA*. Motion primitives describe the states that the system can reach in time interval $\tau > 0$ given an initial state x_0. These motion primitives induce a discretization on the state space $\mathcal{X}$ and thus, for each node in $\mathcal{X}$, discrete successor nodes and the cost to reach them can be found. This creates a graph that can be solved using the A* variants described in the previous section.

The path through the map found by LPA* is used as a nominal trajectory to the goal Φ_0. Centered around this initial trajectory, we define a tunnel of radius r denoted as $\mathcal{T}(\Phi_0, r) \in \mathcal{X}^{\text{free}}$ where

$$\mathcal{X}^{\text{free}} = \mathcal{X} - \mathcal{X}^{\text{obs}}, \tag{6.2}$$

and $\mathcal{X}^{\text{obs}}$ denotes occupied nodes in the map. Following Φ_0, a new trajectory Φ is found according to [33]

$$\begin{aligned} & \operatorname*{argmin}_{\Phi} J_q(\Phi) + \rho_T T(\Phi) + \rho_c J_c(\Phi) && (6.3) \\ \text{subject to: } & \dot{x}(t) = Ax(t) + Bu(t) && (6.4) \\ & x(0) = x_0, \qquad x(T(\Phi)) \in \mathcal{X}^{\text{goal}} \subset \mathcal{X}^{\text{free}} && (6.5) \\ & x(t) \in \mathcal{T}(\Phi_0, r) \cap \mathcal{X}^{\text{free}}, \qquad u(t) \in \mathcal{U}, \qquad t = [0, T(\Phi)] && (6.6) \end{aligned}$$

where $\rho_T,\ \rho_c \geq 0$ denote weighting parameters, $J_c(\cdot)$ denotes the cost due to the possibility of collisions, $J_q(\cdot)$ denotes the smoothness of Φ, $\mathcal{U}$ denotes the control space, and

$$A = \begin{bmatrix} 0_3 & I_3 & 0_3 & \dots & 0_3 \\ 0_3 & 0_3 & I_3 & \dots & 0_3 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0_3 & \dots & \dots & 0_3 & I_3 \\ 0_3 & \dots & \dots & 0_3 & 0_3 \end{bmatrix}, \qquad B = \begin{bmatrix} 0_3 \\ 0_3 \\ \vdots \\ 0_3 \\ I_3 \end{bmatrix}, \tag{6.7}$$

denote the state space form of the system dynamics in (6.4). The trajectory Φ is assumed to be piecewise polynomial and consists of N segments. The smoothness cost $J_q(\cdot)$ is computed according to

$$J_q(\Phi) = \int_0^{\mathrm{T}} ||\Phi^{(q)}||^2 dt = \sum_{n=0}^{N-1} \int_0^{\Delta t_n} ||u_n(t)||^2 dt, \tag{6.8}$$

where $u_n(\cdot)$ denotes the control input for the n^{th} segment of Φ and q denotes the number of derivatives of the position included in the state $x(\cdot)$. The collision cost $J_c(\cdot)$ is computed by solving the line integral [33]

$$J_c(\Phi) = \int_{\Phi} U(s) ds, \tag{6.9}$$

where

$$U(s) = \begin{cases} 0, & d(s) \geq d_{\mathrm{thr}} \\ F(d(s)), & d_{\mathrm{thr}} > d(s) \geq 0, \end{cases} \tag{6.10}$$

$d(s)$ denotes the distance at time s from the closest obstacle, d_{thr} denotes the distance at which the collision risk is negligible, and

$$F(d(s)) = F_{\max}\left(1 - \frac{d(s)}{d_{\text{thr}}}\right)^{k}, \qquad k > 0, \qquad F_{\max} > 0. \tag{6.11}$$

To avoid calculating the line integral in (6.9), we sample I points along the trajectory Φ separated by uniform time step dt. Thus, we have the approximation

$$\int_{\Phi} U(s)ds \approx \sum_{i=0}^{I-1} U(p_i)||v_i||dt, \tag{6.12}$$

where $dt = \frac{T}{I-1}$ and p_i, v_i are the position and velocity at time $i \cdot dt$, respectively. An example of this implementation is shown in Figure 6.2 [33]. The blue line denotes the path found by the A* algorithm. A tunnel of fixed radius around this path is denoted by the translucent blue band. Within this band, a new path is found that travels along the obstacle gradients denoted by the rainbow bands bordering the obstacles.

In our use case, the distance away from the obstacle should be penalized, not rewarded as occurs in this example. This change can be implemented simply by negating the cost on distance from the obstacle. While this change increases the risk of colliding with the obstacle, it is a danger inherent to the mission and is mitigated by the constraint generation described in the previous chapter.

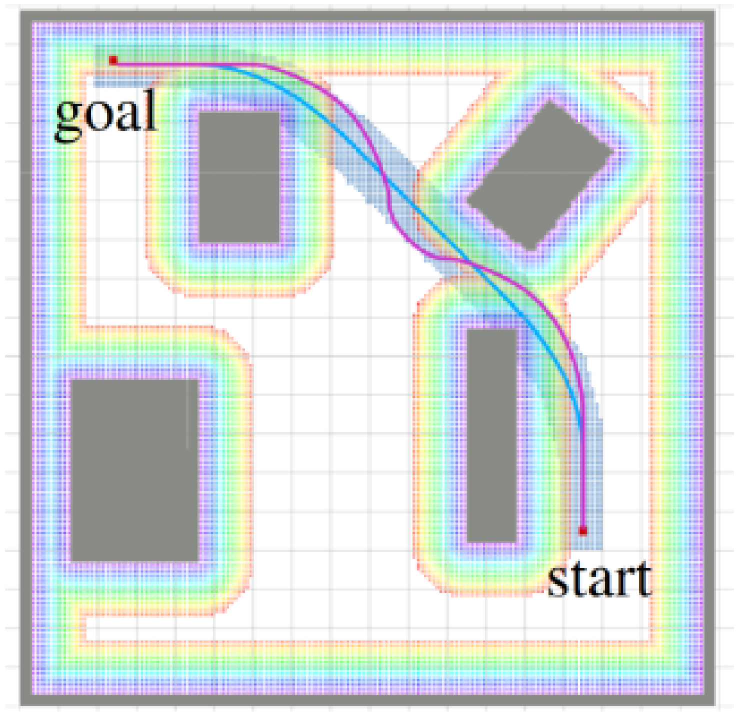
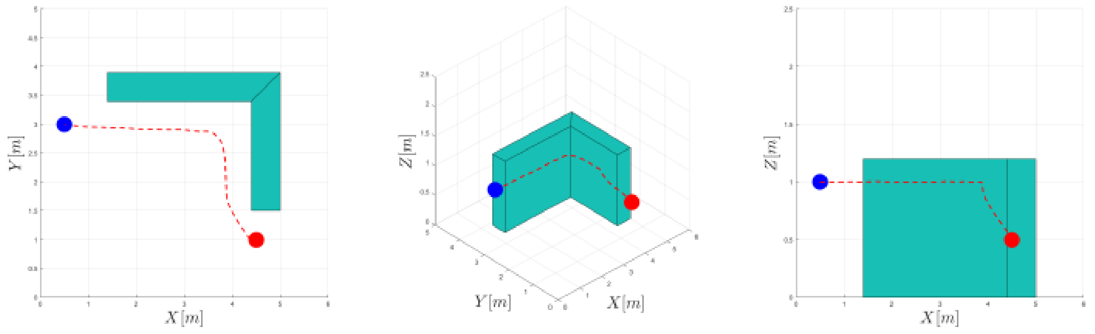


Figure 6.2: Example of MPL using gradient functions

6.4 Simulation of FMPC with MPL

Following our change to the heuristic described by (6.3)–(6.6), a simulation combining FMPC, constraint generation, and MPL can be performed. This simulation is shown in Figure 6.3. The quadrotor starts at the blue dot and travels to the red dot. Although the quadrotor could simply travel straight from the starting point to the goal node, our MPL heuristic has been designed to steer the UAV along the wall. FMPC uses its wall attraction to refine the global trajectory generated by MPL. Furthermore, FMPC is still responsible for collision avoidance.

This successful simulation demonstrates that each of the subsystems described throughout this thesis have been implemented and are interacting as expected. The simulation is also successful in that it demonstrates the tactical behavior desired in the guidance system. With this check completed, live flight tests can be performed.



(a) Top-down view (b) Isometric view (c) Profile view

Figure 6.3: Simulation of full guidance system [1]

7. Flight Test Results

Flight tests were performed inside a 40x20 ft net cage. Inside of this cage, foam boxes were stacked to form the L-shaped obstacle featured in the simulation in Figure 6.3. Within this environment, a VICON motion capture system was used to track the position of the vehicle. This position information is used to update the state vector $x(t)$ in (2.1). Additionally, the state data is recorded so that the path can be graphed.

The first flight test is a physical test of the simulation featured in Figure 6.3. The results are shown in Figure 7.1. The path followed by the vehicle in flight tests generally follows the trajectory generated by the simulation. Just like in the simulation, the vehicle elects to fly along the L-shaped wall instead of flying straight to the unobstructed goal position. The physical trajectory is oscillatory in a way

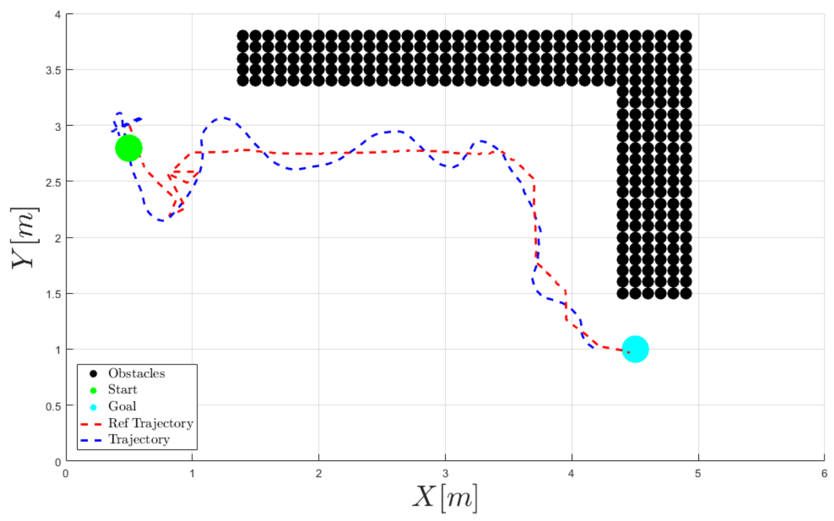


Figure 7.1: Flight test of guidance system demonstrating tactical behavior

that is absent in simulation. As can be seen in the figure, the oscillation in actual flight tests occur around the reference trajectory. This is a tuning issue in the control system and a subject of future work.

While the flight test illustrated in Figure 7.1 successfully reaches the goal in a tactical manner, it does so in a non-challenging environment. The real test occurs when the goal position is obscured and thus the operating region is non-convex. In this way, FMPC on its own is not enough to successfully complete the mission. Instead, the constraint generation developed in Chapter 5 and the global path planning developed in Chapter 6 must be utilized. A flight test where an obstructed goal point is reached by the vehicle is presented in Figure 7.2. Up to this point, the results have been effectively two-dimensional. This is a consequence of the path MPL has generated to reach the goal position. Moving the goal position further behind the obstacle will incentivize the guidance system to steer the vehicle over the wall. This scenario is shown in Figure 7.3 and demonstrates the guidance system's three dimensional capabilities.

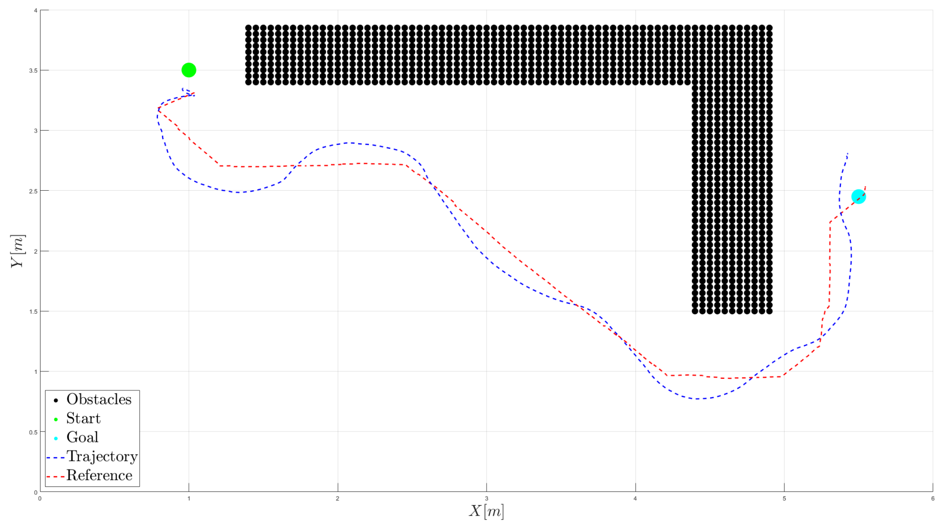


Figure 7.2: Flight test of guidance system within a non-convex environment

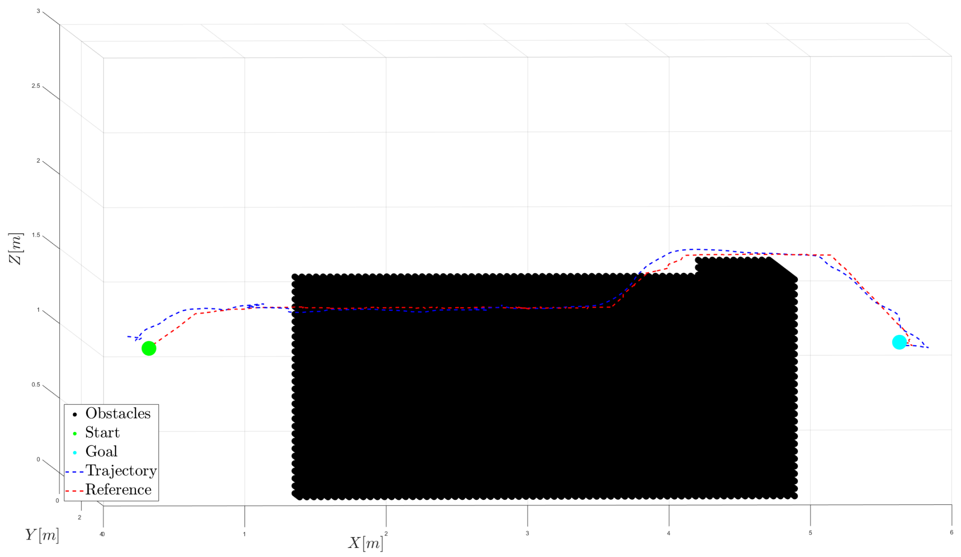


Figure 7.3: Flight test climbing above the wall

8. Conclusion

This thesis proposes a novel guidance system for use on board unmanned quadrotors. While guidance systems for quadrotors are not rare, they are typically designed to support more classical applications of quadrotors. Despite the growth in the number of applications for micro UAVs, there is an aversion towards obstacles in each of them. Instead of focusing on avoiding obstacles, this thesis seeks to use obstacles to protect the vehicle from detection or harm by hostile agents. Existing methods do not cover this use case, so a new one was developed.

The guidance system in this thesis consists of two subsystems working together in tandem. The first system, Fast Model Predictive Control (FMPC), is an extension of classical model predictive control (MPC). MPC uses an estimate of the system dynamics to plan an optimal trajectory several steps into the future at a high enough frequency to reject disturbances. The process is sped up by exploiting the structure of the problem and then referred to as FMPC. FMPC can be altered to fit many different mission profiles. However, FMPC is liable to find local minima from which it cannot escape if it is operating in a non-convex environment. This issue is partially rectified by modeling constraints using affine subsets. Finding these local regions is not a trivial problem, but an approximate solution can be found by finding an ellipsoid that separates the vehicle from surrounding obstacles from which affine hulls can be approximated. This change allows FMPC to operate in a non-convex environment, but does not solve issues caused by local minima. To address this issue, a global path planner was developed using the motion primitive library. The heuristic guiding this path planner was altered to encourage the planner to find solutions that fly nearby to obstacles. MPL finds a rough trajectory that is refined by FMPC according to quadrotor dynamics while FMPC simultaneously uses affine constraint generation to avoid collisions with the obstacles.

A simulation of a quadrotor flying past an obstacle were ran using the aggregate guidance system. The result of this simulation showed that the guidance system steered the vehicle away from a straight line path so that it remained in cover around the obstacle for as long as possible. This result was confirmed by a live flight test of the same scenario. Further flight tests were performed to demonstrate the guidance system's ability to guide the vehicle around obstacles when the goal position is obscured. This ability was demonstrated in multiple dimensions.

These successful results demonstrate the feasibility of this guidance system. However, there are several facets of it that must be further developed to fully accomplish the desired tactical flight profiles. First, there are several parameters in both the FMPC and MPL subsystems that must be set. While preliminary values for these parameters have been found, a more systematic study of how these parameters affect the system is necessary to select optimal values. Currently, a Taguchi analysis is being performed to execute this study. The creation of the guidance system is only one step along the path to a full autopilot for tactical behavior.

In the long run, this guidance system is intended to be one part of an overall autopilot that enables tactical flight. The navigation system is in external development and, once finished, must be integrated with the guidance system. This integration will allow the autopilot to explore and navigate unknown environments without any external tracking system such as VICON. Once the full autopilot system has been fit together, heuristics and cost functions can be studied further to maximize tactical behavior. An example of a possible change is to penalize the vehicle for staying in well lit areas. In this way, the vehicle avoids detection by seeking relatively dark areas near obstacles. While these updates will not happen in the immediate future, this thesis provides a foundation for their development.

Bibliography

[1] J. Karinshak, "Toward Tactical Autonomy in Aerial Robotics," Master's thesis, The University of Oklahoma, Norman, OK, 2019.

[2] D. E. Seborg, D. A. Mellichamp, T. F. Edgar, and F. J. Doyle, *Process Dynamics and Control*. Cambridge University Press, 2011.

[3] J. Richalet, A. Rault, J. Testud, and J. Papon, "Model Predictive Heuristic Control: Applications to Industrial Processes," *Automatica*, vol. 14, pp. 413–428, 1978.

[4] S. J. Qin and T. A. Badgwell, "A Survey of Industrial Model Predictive Control Technology," *Control Engineering Practice*, no. 11, pp. 733–764, 2003.

[5] Y. Wang and S. Boyd, "Fast Model Predictive Control Using Online Optimization," *IEEE Transactions on Control Systems Technology*, vol. 18, no. 2, pp. 267–278, 2010.

[6] E. Yildirim and S. Wright, "Warm-start Strategies in Interior-point Methods for Linear Programming," *SIAM J. Opt.*, vol. 12, no. 3, pp. 782–810, 2002.

[7] D. Bertsekas, *Dynamic Programming and Optimal Control*. Athena Scientific, 2003.

[8] E. Anderson *et al.*, *LAPACK Users' Guide*, 3rd ed. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999.

[9] G. Guennebaud, B. Jacob *et al.*, "Eigen v3," http://eigen.tuxfamily.org, 2010.

[10] J. Nocedal and S. Wright, *Numerical Optimization*. Springer, 1999.

[11] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[12] A. Richards, "Fast Model Predictive Control with Soft Constraint," *European Journal of Control*, no. 25, pp. 51–59, 2015.

[13] N. M. de Oliveira and L. T. Biegler, "Constraint Handling and Stability Properties of Model Predictive Control," Carnegie Mellon University, Tech. Rep., 1993.

[14] G. Kreisselmeier and R. Steinhauser, "Systematic Control Design by Optimizing a Vector Performance Index," *Computer Aided Design of Control Systems: Procedings of the IFAC Symposium*, p. 113, 1979.

[15] A. L'Afflitto, R. B. Anderson, and K. Mohammadi, "An Introduction to Nonlinear Robust Control for Unmanned Quadrotor Aircraft," *IEEE Control Systems Magazine*, pp. 102–121, 2018.

[16] A. L'Afflitto, *A Mathematical Perspective on Flight Dynamics and Control.* Springer, 2017.

[17] P. Bouffard, "On-board Model Predictive Control of a Quadrotor Helicopter: Design, Implementation, and Experiments," Tech. Rep., 2012.

[18] J. S. H. Tsai, C. C. Huang, S. M. Guo, and L. S. Shieh, "Continuous to Discrete Model Conversion for the System with a Singular System Matrix Based on Matrix Sign Function," *Applied Mathematical Modelling*, vol. 35, pp. 3893–3904, 2011.

[19] K. Ogata, *Modern Control Engineering.* Prentice Hall, 2010.

[20] J. Balogh, H. Gonzalez-Aguilar, and G. Salazar, "Large Convex Holes in Random Point Sets," *Mathematics Suject Classification*, 2010.

[21] L. Clark, K. G. Hare, and N. Sidorov, "The Baker's Map with a Convex Hole," *Nonlinearity*, vol. 31, 2018.

[22] M. Y. Yang and W. Forstner, "Plane Detection in Point Cloud Data," Tech. Rep., 2010.

[23] M. A. Fischler and R. C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[24] R. B. Rusu and S. Cousins, "3D is Here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, 2011.

[25] S. A. Cook, "The Complexity of Theorem-proving Procedures," *Annual ACM Symposium on Theory of Computing*, pp. 151–158, 1971.

[26] L. Vandenberghe and A. Richards, "Semidefinite Programming," *SIAM Review*, vol. 38, no. 1, pp. 49–95, 1996.

[27] M. Yamashita, K. Fujisawa, and M. Kojima, "Implementation and Evaluation of SDPA 6.0 (SemiDefinite Programming Algorithm 6.0)," *Optimization Methods and Software*, vol. 18, pp. 491–505, 2003.

[28] F. Alizadeh, "Interior Point Methods in Semidefinite Programming with Applications to Combinatorial Optimization," *SIAM J. Optimization*, vol. 5, no. 1, pp. 13–51, 1995.

[29] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs," *Numerische Mathematik*, no. 1, pp. 269–271, 1959.

[30] S. Koenig, M. Likhachev, and D. Furcy, "Lifelong Planning A*," *Artificial Intelligence*, vol. 155, no. 1, pp. 93–146, 2004.

[31] S. Liu, N. Atanasov, K. Mohta, and V. Kumar, "Search-based Motion Planning for Quadrotors Using Quadratic Minimum Time Control," *IEEE Conference on Intelligent Robotics and Systems*, 2017.

[32] S. Liu, K. Mohta, N. Atanasov, and V. Kumar, "Search-Based Motion Planning for Aggressive Flight in SE(3)," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2439–2446, 2018.

[33] ——, "Towards Search-based Motion Planning for Micro Aerial Vehicles," 2018.