TÉCNICO LISBOA

Model-Predictive Trajectory Planning for Autonomous Aerial Surveillance Applications

Hugo Miguel Tavares Matias

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors: Prof. Daniel de Matos Silvestre
Prof. Rita Maria Mendes de Almeida Correia da Cunha

Examination Committee

Chairperson: Prof. António Manuel Raminhos Cordeiro Grilo
Supervisor: Prof. Rita Maria Mendes de Almeida Correia da Cunha
Member of the Committee: Prof. Bruno João Nogueira Guerreiro

November 2023

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

The development of this thesis would not have been possible without the support of several people.

My first words of deepest appreciation and profound gratitude go to my supervisor, Professor Daniel Silvestre, for his help, support and guidance throughout the entire journey that culminated in this dissertation. I am genuinely thankful for all the insightful discussions that expanded my horizons and kindled my curiosity towards new and challenging problems. Moreover, thanks for giving me the opportunity to delve into a field I am particularly passionate about. My gratitude extends to Professor Rita Cunha, my co-supervisor, for her valuable assistance and support in this academic endeavor.

I am also thankful to all my friends and colleagues at IST for their friendship and support in the past five years. I had the privilege of meeting many extraordinary people who will undoubtedly remain an integral part of my life in the future. Specially, I want to express my heartfelt gratitude to my closest friends, Gonçalo Gomes, José Reis, João Luzio, and Pedro Taborda, for making these years so memorable. This journey would definitely not have been possible without you, and I will forever cherish the wonderful memories we have created together.

Lastly, but certainly not least, I am profoundly grateful to my family for their unconditional support, endless love and tolerance. I will be forever in your debt for being such a foundation in my life.

This work was partially supported by the Portuguese Fundação para a Ciência e a Tecnologia (FCT) through project FirePuma (https://doi.org/10.54499/PCIF/MPG/0156/2019), through Institute for Systems and Robotics (ISR), under Laboratory for Robotics and Engineering Systems (LARSyS) project UIDB/50009/2020, through COPELABS, University Lusófona project UIDB/04111/2020 and FCT project CAPTURE (https://doi.org/10.54499/PTDC/EEI-AUT/1732/2020).

Resumo

A presente dissertação aborda o desenvolvimento de algoritmos de planeamento de trajetórias, baseados em controlo preditivo, para aplicações de vigilância autónoma usando um veículo aéreo não tripulado. Numa primeira fase, aborda-se o planeamento de trajetórias ótimas para prevenção de incêndios florestais com base num mapa que caracteriza a incerteza da presença de fogo numa determinada região. A nossa abordagem envolve um algoritmo projetado para promover a exploração do mapa, evitando que o veículo retorne a áreas que já foram cobertas. Isto é alcançado penalizando interseções entre as regiões de observação do veículo ao longo da sua trajetória. O algoritmo é inicialmente testado num ambiente MATLAB e posteriormente validado no simulador Gazebo, bem como por meio de testes reais realizados num ambiente exterior. Os resultados demonstram que o algoritmo proposto é capaz de gerar trajetórias de alta qualidade para vigilância.

Numa segunda fase, aborda-se um problema mais complexo, em que o objetivo é perseguir um alvo móvel estocástico caracterizado por uma distribuição de probabilidade variável no tempo. O algoritmo proposto baseia-se em filtragem Bayesiana recursiva para sistemas probabilísticos descritos por distribuições de mistura gaussiana. Semelhante ao papel das penalizações no primeiro problema, quando o alvo está fora da região de visibilidade do veículo, a função ideal de medição é aproximada usando uma grid de distribuições Gaussianas. A eficácia do algoritmo proposto é demonstrada através de exemplos simples considerando um modelo de evolução linear para o alvo.

Palavras-chave: Vigilância Autónoma; Veículos Aéreos Não Tripulados; Planeamento de Trajetórias; Controlo Preditivo; Distribuições de Mistura Gaussiana.

Abstract

This dissertation addresses the development of trajectory planning algorithms based on Model Predictive Control (MPC) for autonomous surveillance applications using an Unmanned Aerial Vehicle (UAV). In a first stage, we address the generation of optimal trajectories for wildfire prevention based on a map that characterizes the uncertainty regarding fire presence in a given region. Our approach involves an algorithm designed to promote the exploration of the map by preventing the UAV from revisiting previously covered areas. This is achieved by penalizing intersections between the visibility regions of the UAV along its trajectory. The algorithm is initially tested in MATLAB and subsequently validated in the Gazebo simulator, as well as through actual experiments conducted in an outdoor environment. The results demonstrate that the proposed algorithm can generate high-quality trajectories for surveillance.

In a second stage, we consider a more complex scenario, where the objective is to search and track a stochastic moving target characterized by a time-varying probability distribution. The proposed algorithm relies on recursive Bayesian filtering for probabilistic systems described by Gaussian mixture distributions. Similar to the role of the penalizations in the first problem, when the target is outside the visibility region of the UAV, the ideal measurement model is approximated using a grid-based Gaussian mixture distribution. The efficacy of the proposed algorithm is assessed through simple examples considering a linear model for the target.

Keywords: Autonomous Surveillance; Unmanned Aerial Vehicles; Trajectory Planning; Model Predictive Control; Gaussian Mixture Distributions.

Contents

List of Figures

Acronyms

API Application Programming Interface. 31, 63

EKF Extended Kalman Filter. 21, 37, 42

FOV Field of View. xv, 12, 37, 39, 40, 48, 53, 55, 57

GPS Global Positioning System. 34, 35

HITL Hardware In The Loop. 31

IP Interior Point. 5

LTI Linear Time-Invariant. 7

MPC Model Predictive Control. 3–6, 8, 10, 13, 14, 18, 20–22, 28, 30–32, 34, 37, 39, 46, 50, 51, 53, 55, 57, 58

NLP Nonlinear Program. 5, 14

ODE Ordinary Differential Equation. 6, 7, 13

PDF Probability Density Function. 11, 39, 41, 43, 44, 46, 47, 55

PF Particle Filter. 42

PID Proportional-Integral-Derivative. 21

PRM Probabilistic Roadmap Method. 2

RHC Receding Horizon Control. 5

ROS Robot Operating System. 30, 31, 63, 64

RRT Rapidly-Exploring Random Tree. 2

SITL Software In The Loop. 31

SQP Sequential Quadratic Programming. 5

UAV Unmanned Aerial Vehicle. 1, 3, 8–13, 18–20, 22, 24, 25, 31, 37, 39, 40, 47–50, 52, 55, 57, 58

UKF Unscented Kalman Filter. 42

Chapter 1

Introduction

1.1 Motivation

Unmanned Aerial Vehicles (UAVs) have emerged as a popular technology for autonomous surveillance, offering a range of applications across various sectors, from security and law enforcement to environmental monitoring and disaster response. These versatile aerial platforms, often equipped with high-resolution cameras, sensors, and cutting-edge technology, have the capacity to perform surveillance operations autonomously, reducing the need for constant human intervention, as depicted in Figure 1.1. By leveraging UAVs for surveillance, organizations and government agencies can monitor vast areas with unprecedented ease and speed, enhancing their ability to respond to threats and emergencies in real time. As technology continues to advance, the use of UAVs for autonomous surveillance is on the verge of revolutionizing the way we safeguard our communities, assets, and natural resources, offering a more cost-effective, precise, and adaptive approach to security and monitoring.

Figure 1.1: A drone performing a surveillance mission.

In the context of autonomous surveillance using UAVs, developing efficient trajectory planning algorithms is a pivotal research area focused on optimizing flight trajectories to enhance the effectiveness and efficiency of surveillance missions. Trajectory planning algorithms play a crucial role in enabling UAVs to autonomously navigate and monitor diverse environments, offering the potential for more accurate and adaptable surveillance. Considering this underlying motivation, this thesis aims to develop innovative algorithms for potential application in autonomous surveillance scenarios involving UAVs.

1.2 Review of Trajectory Planning

Trajectory planning, also known as trajectory generation, is a fundamental element in the execution of autonomous vehicle missions. Designing the trajectory amounts to a complex decision-making and control problem, where the goal is to define the motion of autonomous vehicles over time in order to accomplish some objective. Due to the growing integration and deployment of autonomous vehicles, the demand for effective trajectory planning algorithms is significantly increasing. This section aims to provide a brief overview of the many approaches that have been proposed for trajectory planning.

1.2.1 Traditional Algorithms

The traditional planning algorithms are primarily geometric. The process consists of initially creating a spatial path from a set of path primitives, known as path planning, and subsequently parameterizing the generated path in time. The classical planning algorithms can be divided into four main groups: graph-search, sampling-based, interpolating curve, and reaction-based algorithms [1, 2].

In the early works, the predominant approach involved the utilization of graph-based planners to efficiently generate paths on a graph constructed by the discretization of the environment. Well-known methods, such as the Dijkstra's algorithm, the A* algorithm, and the state lattice, have all been employed in the field of trajectory planning [3, 4, 5]. Nevertheless, a significant drawback of graph-based planners is that the quality of the planned trajectories highly depends on the graph's resolution. In addition, another limitation is that vehicle dynamics cannot be easily taken into account during planning.

Sampling-based planners and their variants have also been widely applied to trajectory planning for autonomous vehicles. The approach consists of randomly sampling the configuration space, looking for connectivity inside it [6]. The two most commonly used methods are the Probabilistic Roadmap Method (PRM) [7] and the Rapidly-Exploring Random Tree (RRT) [8]. These methods offer the advantage of generating feasible trajectories for both holonomic and nonholonomic systems. The RRT can even guarantee the asymptotic optimality of the generated path. Nevertheless, these approaches are hindered by the high computational complexity of the sampling procedure, limiting their applicability.

Curve interpolation planners, including the Clothoid curves [9], polynomial curves [10], and spline curves [11], have also been widely used for trajectory planning. These algorithms take a previous set of waypoints and generate a smoother path, prioritizing factors such as trajectory continuity, vehicle constraints, and the environment. The advantage of curve interpolation planners is their low computational cost, since the curve can be defined by only a few control points or parameters. Nonetheless, the optimality of the generated trajectory cannot be guaranteed. Furthermore, as the planning process does not consider the vehicle dynamics, a smoothing process is usually needed to refine the generated paths.

Intensive research has also been conducted on potential field methods [12]. A potential field is a differentiable real-valued function whose value can be seen as energy, and its gradient can be seen as a force. Hence, vehicles in potential fields move along the gradient of the field. Due to their intuitive problem formulation, potential field methods have been applied to various vehicle applications. However, as shown in [13], integrating system dynamics and constraints remains a challenge within this approach.

1.2.2 Optimal Control Approaches

In many applications, generating trajectories that optimize some performance metric is a desirable goal. Hence, trajectory planning problems are usually formulated as optimal control problems. However, optimal control problems are generally nonlinear and, therefore, do not have analytic solutions. As a result, it is necessary to employ numerical optimization methods to solve optimal control problems. In this context, optimization-based approaches have been applied efficiently and have emerged as a promising approach for trajectory generation, often surpassing the traditional methods.

Different optimization-based approaches can be derived from a generic optimal control formulation (see Figure 1.2). In the early years, Indirect Methods were the main approach for solving optimal control problems. These methods consist of using the Pontryagin's principle [14] to obtain the necessary first-order conditions for optimality, which result in a boundary-value problem. For particular problem formulations and simple system models, solutions to the optimization problem may be found efficiently. However, the boundary-value problem is usually extremely difficult to solve.

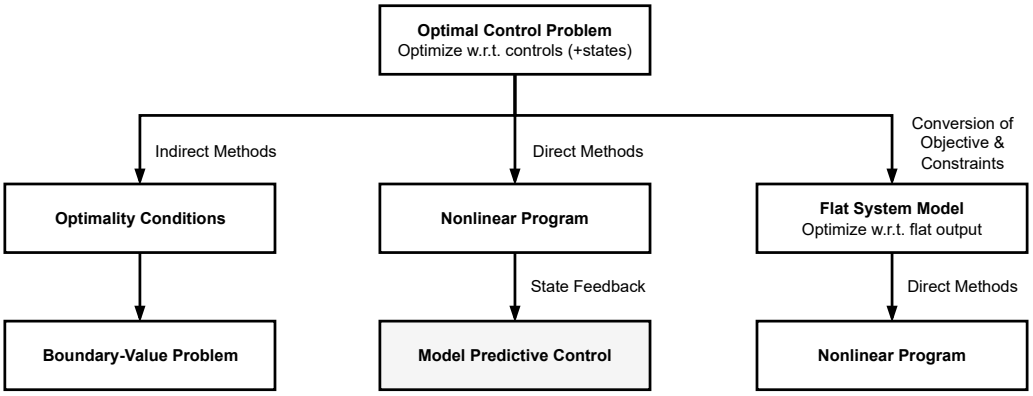


Figure 1.2: Overview of optimization-based approaches.

The approach that has risen to prominence is that of Direct Methods. Direct Methods discretize optimal control problems to obtain nonlinear programs for which many efficient solving techniques exist. Instead of planning a complete trajectory offline, as in traditional optimization approaches, Direct Methods are also suitable for online trajectory generation. In particular, Model Predictive Control (MPC) is a prominent method that has been commonly used for online trajectory generation, in contrast to its typical application for reference tracking problems. Since computational resources have increased and algorithms have become more efficient, the interest in MPC-based approaches has been growing. Recent examples include trajectory generation applications to robotic manipulators [15], autonomous driving [16], and UAVs [17]. A more detailed background on MPC is provided in the following section.

Ultimately, a third approach relies on a significant property exhibited by certain systems, known as differential flatness. This property allows the state and the input of the system to be expressed as a function of a flat output and a finite number of its derivatives. Such property is of particular interest because the optimal control problem can be completely formulated in terms of the flat output. As a result, the optimal control problem may be discretized by parameterizing the flat output, avoiding the use of numerical integration methods. Such methods are also suitable for online trajectory generation [18].

1.3 Background on Model Predictive Control

This section provides a more detailed background on MPC, as it is the primary technique used to address the problems considered in this dissertation. We begin by presenting a brief overview of MPC, followed by its general mathematical formulation, and then we discuss some details regarding the implementation of MPC-based algorithms.

1.3.1 General Overview

Online optimization has gained significant traction as the primary method for addressing control and estimation problems in science and engineering. One of the main reasons for this is the capability to directly account for state and input constraints within online optimization methods. MPC, in particular, has become one of the most prominent online optimization control techniques, extensively employed in numerous industrial applications [19]. Due to its increasing popularity, substantial efforts have been devoted to developing a stability theory for MPC (see e.g. [20, 21]). A comprehensive overview of the latest theoretical developments and future perspectives is available in [22].

MPC is an advanced control method that consists of a repeated real-time optimization based on a mathematical model of a system. Based on the system model, MPC predicts the future system behavior and determines the optimal sequence of control inputs according to some objective (see Figure 1.3). Therefore, MPC comes with an intuitive formulation at the cost of an increased computational effort. Because of its high computational cost, MPC has traditionally been more popular for problems where the plant dynamics are slow enough so that the optimization can be solved efficiently between consecutive sampling times. However, thanks to progress in optimization algorithms and as the computational power of embedded control platforms increases, MPC can be employed in broader application areas [23, 24].

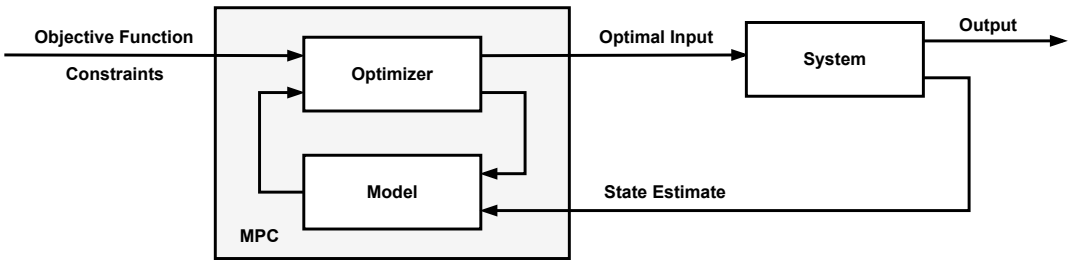


Figure 1.3: Simplified block diagram of an MPC-based control loop.

The anticipating behavior and the fact that MPC can consider hard constraints makes this method extremely valuable for controlling real systems. Moreover, MPC relies on mathematical models, which are available in almost every subject. Such characteristic allows us to use the knowledge about models and avoid the formulation of explicit control laws. In MPC, the control law is determined implicitly through the model-based optimization. The implicit formulation, the flexibility, and the use of models are the main advantages of MPC. Such advantages aligned with the rise of computational power are the reasons why MPC has become one of the most popular control methods in the engineering community.

1.3.2 Mathematical Formulation

MPC consists in solving an open-loop discrete-time optimal control problem at each sampling time. Each of these optimizations results in a sequence of future optimal control actions and a sequence of corresponding future states. Only the first control action in the sequence is applied to the plant, and then the optimization is solved again at the next sampling time. Due to the repeated prediction and optimization process, MPC is also known as Receding Horizon Control (RHC). Basically, the idea is that short-term predictive optimizations achieve optimality over a long time. This intuition is based on the fact that distant predictions may contain redundant information that is not relevant for shorter predictions.

More specifically, at every discrete-time instant k, for a given initial state $\mathbf{x}_k$ of the system, the control policy is defined by solving a discrete-time optimal control problem of the form

$$\begin{aligned} \underset{\hat{\mathbf{X}}_k, \hat{\mathbf{U}}_k}{\text{maximize}} \quad & J_k(\hat{\mathbf{X}}_k, \hat{\mathbf{U}}_k) \\ \text{subject to} \quad & \hat{\mathbf{x}}_{k,0} = \mathbf{x}_k, \\ & \hat{\mathbf{x}}_{k,j+1} = \mathbf{f}(\hat{\mathbf{x}}_{k,j}, \hat{\mathbf{u}}_{k,j}),\ j = 0, \dots, N-1, \\ & \hat{\mathbf{x}}_{k,j} \in \mathcal{X},\ j = 0, \dots, N, \\ & \hat{\mathbf{u}}_{k,j} \in \mathcal{U},\ j = 0, \dots, N-1, \end{aligned} \tag{1.1}$$

where N is the horizon length and J_k is a user-defined objective function to be maximized at each discrete-time instant k. The matrices $\hat{\mathbf{X}}_k$ and $\hat{\mathbf{U}}_k$ represent the predicted state and control sequences over the prediction horizon at time instant k, i.e.,

$$\begin{aligned} \hat{\mathbf{X}}_k &\triangleq \begin{bmatrix} \hat{\mathbf{x}}_{k,0} & \hat{\mathbf{x}}_{k,1} & \dots & \hat{\mathbf{x}}_{k,N-1} & \hat{\mathbf{x}}_{k,N} \end{bmatrix}, \\ \hat{\mathbf{U}}_k &\triangleq \begin{bmatrix} \hat{\mathbf{u}}_{k,0} & \hat{\mathbf{u}}_{k,1} & \dots & \hat{\mathbf{u}}_{k,N-1} \end{bmatrix}. \end{aligned} \tag{1.2}$$

The sets $\mathcal{X}$ and $\mathcal{U}$ constitute the admissible states and inputs for the system, and the function $\mathbf{f}$ describes an arbitrary discrete-time system. The input applied to the system at sampling time k, $\mathbf{u}_k$, is given by

$$\mathbf{u}_k = \hat{\mathbf{u}}^*_{k,0}, \tag{1.3}$$

where $\hat{\mathbf{u}}^*_{k,0}$ is the first sample of the predicted optimal control sequence at time instant k.

The formulation in (1.1) describes an optimization problem that, in a general sense, may be a nonlinear optimization problem. That is the case when some of the constraints or the objective function are nonlinear, and, in that context, the optimization problem is usually called a Nonlinear Program (NLP). Fortunately, considerable progress has been achieved in the field of nonlinear optimization, and NLPs may be solved efficiently using available numerical solvers. The two most used approaches for solving nonlinear optimization problems are Sequential Quadratic Programming (SQP) and Interior Point (IP) methods [25]. MPC algorithms may then be implemented using optimization modeling software such as YALMIP [26] or CasADi [27], combined with a numerical solver such as the widely used IPOPT [28].

1.3.3 Warm-Start Strategies

The optimization algorithms that can be used to solve the optimization problem in (1.1) require an initial guess $\hat{\mathbf{X}}_k^0$, $\hat{\mathbf{U}}_k^0$ as input. Selecting an appropriate initial guess is essential to obtain a fast and reliable convergence from the optimization algorithms. Assuming that a good solution has been obtained at the time instant $k-1$, it is possible to construct a suitable initial guess for time instant k. In this context, a warm-start strategy that is usually employed is the shifting method [29]. Shifting builds an initial guess $\hat{\mathbf{X}}_k^0$, $\hat{\mathbf{U}}_k^0$ for time instant k by shifting the solution $\hat{\mathbf{X}}_{k-1}^*$, $\hat{\mathbf{U}}_{k-1}^*$ obtained at time $k-1$. Shifting assumes that the system follows closely the predicted evolution, $\hat{\mathbf{x}}_{k-1,1} \simeq \mathbf{x}_k$, and is performed as

$$\begin{aligned}
\hat{\mathbf{x}}_{k,j}^0 &= \hat{\mathbf{x}}_{k-1,j+1}^*,\ j = 0, \dots, N-1, \\
\hat{\mathbf{u}}_{k,j}^0 &= \hat{\mathbf{u}}_{k-1,j+1}^*,\ j = 0, \dots, N-2, \\
\hat{\mathbf{x}}_{k,N}^0 &= \mathbf{f}(\hat{\mathbf{x}}_{k,N-1}^0, \hat{\mathbf{u}}_{k,N-1}^0).
\end{aligned} \tag{1.4}$$

The guess for the last control input $\hat{\mathbf{u}}_{k,N-1}^0$ can be selected through different approaches (see e.g. [21]). In practice, a straightforward approach is commonly adopted, which consists of duplicating the control input at the prediction instant $N-2$,

$$\hat{\mathbf{u}}_{k,N-1}^0 = \hat{\mathbf{u}}_{k,N-2}^0. \tag{1.5}$$

If there are no perturbations to the system, shifting provides a guess that is extremely close to the solution of the problem. However, if there are perturbations to the system, a greater correction of the guess is necessary. Nevertheless, if the perturbations are not very significant, shifting will still provide a guess that is close to the solution of the optimization problem.

1.3.4 Discretization Methods

Algorithms based on MPC require a discrete-time model of the system dynamics. Nevertheless, in the problems considered in this dissertation and many other applications, the system dynamics are available in continuous time. For that reason, in this subsection we present a few main methods that are commonly used to discretize continuous-time systems. For the sake of simplicity, we only consider time-invariant systems described by an explicit Ordinary Differential Equation (ODE) of the form

$$\dot{\mathbf{x}}(t) = \mathbf{F}(\mathbf{x}(t), \mathbf{u}(t)), \tag{1.6}$$

but the following developments can be easily extended to the time-varying case and to implicit ODEs. We also consider a piecewise constant parametrization of the control inputs, such that $\mathbf{u}(t) = \mathbf{u}_k$, for $t \in [t_k, t_{k+1}[$. The restriction to piecewise constant control parameterizations is not mandatory since there are other types of parametrizations that may also be used. However, piecewise constant controls are the most commonly used in practice for the ease of implementation using zero-order holders.

In the case of Linear Time-Invariant (LTI) systems, which are described by an ODE of the form

$$\dot{\mathbf{x}}(t) = \mathbf{A}_c\mathbf{x}(t) + \mathbf{B}_c\mathbf{u}(t), \tag{1.7}$$

an analytical solution for the ODE is available, and is given by the following convolution integral

$$\mathbf{x}(t) = e^{\mathbf{A}_c t}\mathbf{x}(0) + \int_0^t e^{\mathbf{A}_c(t-\tau)}\mathbf{B}_c\mathbf{u}(\tau)\,d\tau. \tag{1.8}$$

In this case, there is an exact discrete-time version of the system dynamics given by

$$\mathbf{x}_{k+1} = \mathbf{A}_d\mathbf{x}_k + \mathbf{B}_d\mathbf{u}_k, \tag{1.9}$$

where the matrices $\mathbf{A}_d$ and $\mathbf{B}_d$ depend on the sampling period, T_s, and are obtained as

$$\mathbf{A}_d = e^{\mathbf{A}_c T_s} \quad \text{and} \quad \mathbf{B}_d = \int_0^{T_s} e^{\mathbf{A}_c\tau}\mathbf{B}_c\,d\tau. \tag{1.10}$$

When the ODE in (1.6) is nonlinear, an analytical solution is not usually available, and numerical integration methods need to be considered. The simplest numerical integration scheme is the Euler method, which approximates the exact discrete-time dynamics by

$$\mathbf{x}_{k+1} = \mathbf{x}_k + T_s\mathbf{F}(\mathbf{x}_k, \mathbf{u}_k), \tag{1.11}$$

and provides a total accumulated error in the order of $\mathcal{O}(T_s)$. Nevertheless, despite its simplicity, the Euler method usually does not yield the most efficient approach because it is only a first-order method. In fact, many more integration techniques have been developed and are available (see e.g. [30]). One of the most popular examples is the Runge-Kutta method of order four, which, unlike the Euler method, results in a total accumulated error in the order of $\mathcal{O}(T_s^4)$. The fourth-order Runge-Kutta method approximates the exact discrete-time dynamics by

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \frac{T_s}{6}(k_1 + 2k_2 + 2k_3 + k_4), \tag{1.12}$$

where the variables $k_1, \ldots, k_4$ are given by

$$\begin{aligned} k_1 &= \mathbf{F}(\mathbf{x}_k, \mathbf{u}_k), \\ k_2 &= \mathbf{F}(\mathbf{x}_k + \frac{T_s}{2}k_1, \mathbf{u}_k), \\ k_3 &= \mathbf{F}(\mathbf{x}_k + \frac{T_s}{2}k_2, \mathbf{u}_k), \\ k_4 &= \mathbf{F}(\mathbf{x}_k + T_s k_3, \mathbf{u}_k). \end{aligned} \tag{1.13}$$

1.4 Outline

The remaining chapters of this dissertation are structured as follows:

- **Chapter 2**: addresses the UAV trajectory generation problem for wildfire prevention based on a map that characterizes the uncertainty of fire presence in a given region. We propose an algorithm based on MPC designed to promote the exploration of the map by preventing the UAV from revisiting previously covered areas. The algorithm is initially tested in a MATLAB environment and subsequently validated in the Gazebo simulator, as well as through actual experiments conducted in an outdoor environment;
- **Chapter 3**: considers a second problem, where the objective is to pursue a stochastic moving target characterized by a time-varying probability distribution. We propose an MPC algorithm that relies on recursive Bayesian filtering for stochastic state-space systems described by Gaussian mixture distributions. The efficacy of the proposed algorithm is demonstrated through simple examples considering a linear model for the target;
- **Chapter 4**: summarizes the research work undertaken in this dissertation and provides suggestions for future research avenues.

Chapter 2

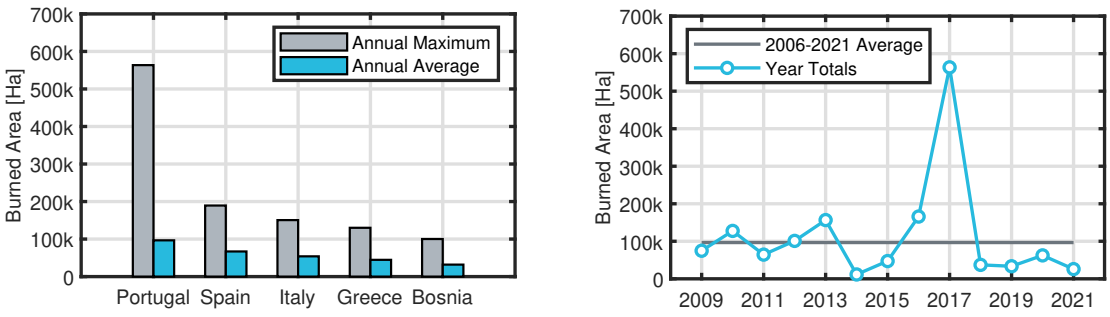
Wildfire Prevention through Uncertainty Minimization

In this chapter, we develop a trajectory planning algorithm for autonomous wildfire surveillance using a UAV. The main goal is to generate optimal trajectories for surveillance based on a map describing the uncertainty of fire presence in a given territory. The effectiveness of the proposed algorithm is demonstrated through simulations and actual experiments. While the primary focus is on wildfire prevention, the proposed algorithm has the potential to be applied in other domains.

2.1 Introduction

2.1.1 Motivation

Over the past few decades, wildfires have emerged as a significant global threat. A combination of factors, including insufficient planning, more extreme weather conditions, and a lack of forest maintenance, has led to consecutive disasters. As a result, these events have imposed substantial financial burdens, claimed numerous human lives, and caused extensive damage to forests. In Europe, particularly, several countries witness an average yearly destruction of more than 50,000 hectares (ha) of land due to wildfires, with some years surpassing 100,000 ha [31]. Among these, Portugal has been the most severely impacted, with nearly 600,000 ha of land being burned in 2017 [32], as shown in Figure 2.1.



(a) Land burned by wildfires in Europe

(b) Land burned by wildfires fires in Portugal

Figure 2.1: A look at wildfires in Europe, with an emphasis on Portugal (obtained from *Statista*).

According to a study conducted by the European Commission, Portugal experienced approximately 21,000 wildfires in 2017, which resulted in 117 deaths and caused nearly 1,500 million euros in damage costs [33, "Super Case Study 4"]. In particular, the wildfire of Pedrógão Grande resulted in a burned area of roughly 45,000 acres and caused nearly 500 million euros in damage costs to the Portuguese government [34]. This outbreak also had a significant impact on the population, either directly through the destruction of agricultural resources and private property or indirectly through the impact on public infrastructures such as roads, energy networks, and telecommunications.

In light of such consecutive disasters, there has been a growing recognition of the urgent need to find technological solutions to prevent wildfires. As a result, governments and authorities have been interested in searching for solutions within the scientific community. The Portuguese government, in particular, has been promoting scientific research and innovation to improve the national forest defense system against wildfires, opening calls for research and development projects on the subject [35].

2.1.2 Objectives

The aim of this chapter is to develop a trajectory planning algorithm for autonomous wildfire surveillance using a UAV. The main goal is to generate optimal trajectories for surveillance based on a map describing the uncertainty of fire presence in a given area. In order to reduce the total uncertainty in the map, the generated trajectories must guide the autonomous vehicle through the most uncertain areas while the vehicle infers about the existence of fire using onboard sensors. The uncertainty map is assumed to be provided by a previously developed estimation filter that takes into account the uncertainty of measurements from different sources of information, such as satellite images and crowdsourced data.

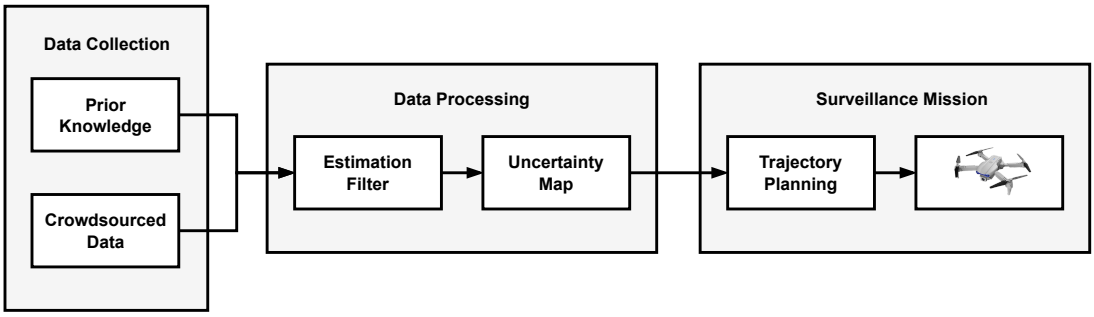


Figure 2.2: Framework.

2.1.3 Organization

This chapter consists of five main sections. Section 2.2 presents a formal mathematical description of the problem discussed in this chapter from an optimal control standpoint. In Section 2.3, we develop an algorithm based on MPC to tackle the problem. Section 2.4 describes the full control architecture used to implement the proposed algorithm in a quadrotor UAV. In Section 2.5, the proposed algorithm is tested in a MATLAB environment. Finally, in Section 2.6, we perform additional tests in a Gazebo environment and a final validation in a real setting.

2.2 Problem Description

This section presents the mathematical definition of the problem discussed in this chapter. We start by introducing relevant assumptions concerning the uncertainty map and the sensing model of the UAV. The section culminates with the formulation of the problem from an optimal control standpoint.

2.2.1 Uncertainty Map

The first topic to be defined is the uncertainty map. The uncertainty map is mathematically described by a nonnegative function $h : \mathbb{R}^2 \to \mathbb{R}_0^+$ that represents the *a priori* level of uncertainty about the existence of fire at each position $\mathbf{p} \in \mathbb{R}^2$. Since the original structure of the uncertainty map typically may not follow common and well-known models, we assume that the uncertainty function can be arbitrarily well approximated by a Gaussian mixture, which is a weighted sum of Gaussian components. Consequently, for a model with M components, the uncertainty function is given by

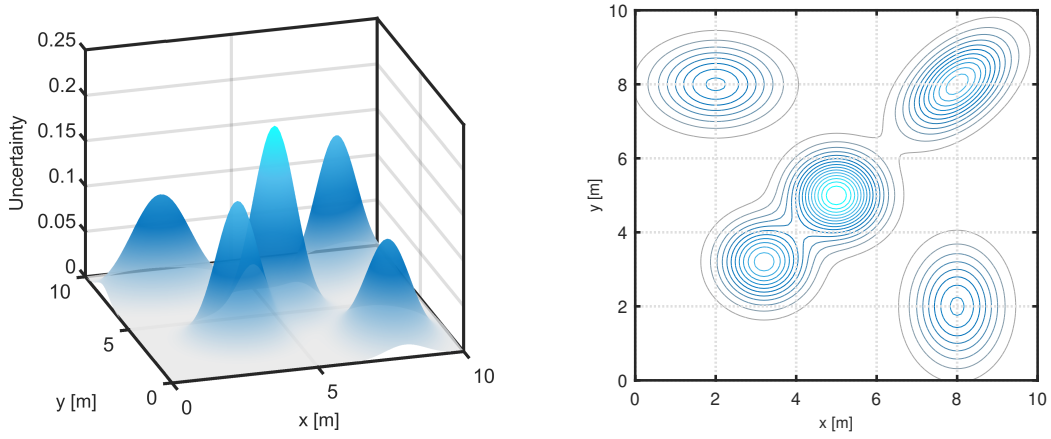
$$h(\mathbf{p}) = \sum_{i=1}^{M} w_i \mathcal{N}(\mathbf{p}; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i), \tag{2.1}$$

where each component is a two-dimensional Gaussian distribution defined by

$$\mathcal{N}(\mathbf{p}; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \triangleq \frac{1}{\sqrt{4\pi^2 |\boldsymbol{\Sigma}_i|}} \exp\left\{-\frac{1}{2}(\mathbf{p} - \boldsymbol{\mu}_i)^\top \boldsymbol{\Sigma}_i^{-1} (\mathbf{p} - \boldsymbol{\mu}_i)\right\}. \tag{2.2}$$

The parameters $w_i > 0$, $\boldsymbol{\mu}_i \in \mathbb{R}^2$, and $\boldsymbol{\Sigma}_i \in \mathbb{R}^{2\times 2}$ are, respectively, the weight, the mean vector, and the covariance matrix of the i^{th} Gaussian component.

Additionally, we clarify that the uncertainty map is not originally a Probability Density Function (PDF), so the volume of uncertainty in the map is not necessarily one. However, it is convenient to assume that this function is normalized, meaning that the *a priori* volume of uncertainty is one, and, therefore, the weights verify $\sum_{i=1}^{M} w_i = 1$. A plausible instance of an uncertainty map is shown in Figure 2.3.



(a) Graph (b) Level curves

Figure 2.3: Example of an uncertainty function composed of five Gaussian distributions.

2.2.2 Sensing Model

In this work, we assume that the UAV flies at a constant altitude and is equipped with a gimbal camera, which always aims straight down even when the UAV performs pitch or roll maneuvers. Consequently, at each time instant t, we assume that the drone analyzes a given area, $\mathcal{B}_r(\mathbf{p}_c)$, defined as a circle centered in the UAV's horizontal position, $\mathbf{p}_c$, and with a radius of observation r, i.e.,

$$\mathcal{B}_r(\mathbf{p}_c) \triangleq \left\{\mathbf{p} \in \mathbb{R}^2 : \|\mathbf{p} - \mathbf{p}_c\| < r\right\}, \tag{2.3}$$

as illustrated in Figure 2.4. Additionally, the vehicle is assumed to have a perfect quality of exploration, meaning that all points within the observation radius are analyzed perfectly. This assumption implies that, immediately after the UAV analyses a given area, the uncertainty becomes zero for all points inside the area covered by the UAV, and, therefore, there is no reward in revisiting the same region.

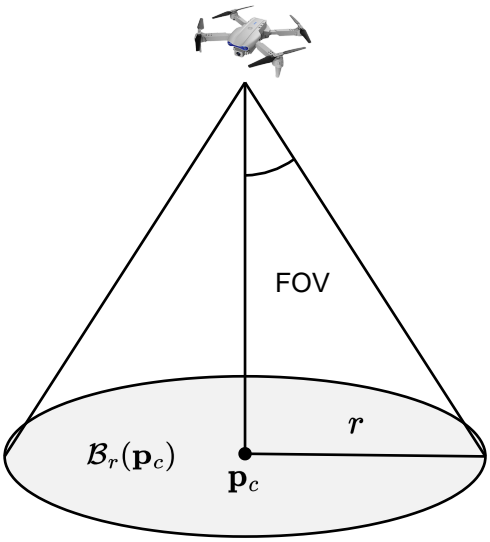


Figure 2.4: Illustration of sensor FOV and visibility region.

2.2.3 Optimal Control Problem

The trajectory generation problem addressed in this chapter can be defined as finding optimal trajectories that guide the UAV. The trajectories should maximize an objective functional regarding the mission goals while satisfying constraints accounting for their dynamic feasibility. Consequently, this problem can be formulated as the following optimal control problem

$$\begin{aligned}
\underset{\mathbf{x}(.),\mathbf{u}(.)}{\text{maximize}} \quad & J[\mathbf{x}(.), \mathbf{u}(.)] \\
\text{subject to} \quad & \mathbf{x}(0) = \mathbf{x}_0, \\
& \dot{\mathbf{x}}(t) = \mathbf{F}(\mathbf{x}(t), \mathbf{u}(t)),\ t \in [0, T], \\
& \mathbf{x}(t) \in \mathcal{X},\ t \in [0, T], \\
& \mathbf{u}(t) \in \mathcal{U},\ t \in [0, T],
\end{aligned} \tag{2.4}$$

where T denotes the total flight time, the functions $\mathbf{x}(.) : [0, T] \to \mathbb{R}^{n_x}$ and $\mathbf{u}(.) : [0, T] \to \mathbb{R}^{n_u}$ denote the state and input of the vehicle's model described by an ODE, and $\mathbf{x}_0$ is the initial value of the state. Moreover, the sets $\mathcal{X}$ and $\mathcal{U}$ constitute the admissible states and inputs for the vehicle, which are derived from limits imposed by vehicle dynamics and the surrounding environment.

Let $\boldsymbol{\varphi} : [0, T] \to \mathbb{R}^2$ denote the vehicle's trajectory on the horizontal plane, which is related to the state of the vehicle by

$$\boldsymbol{\varphi}(t) = \mathbf{C}\mathbf{x}(t), \ t \in [0, T], \tag{2.5}$$

where $\mathbf{C} \in \mathbb{R}^{2\times n_x}$ is an auxiliary matrix that extracts the horizontal position of the vehicle from the state. The mission objective is to maximize the uncertainty reduction, i.e., the difference between the uncertainty volume in the map before and after the surveillance mission. Therefore, considering the previously mentioned assumptions, the objective functional J is given by

$$J[\boldsymbol{\varphi}] = \int_{\mathcal{C}_r[\boldsymbol{\varphi}]} h(\mathbf{p})\, d\mathbf{p}, \tag{2.6}$$

where the set $\mathcal{C}_r[\boldsymbol{\varphi}]$ is defined as the union of all circles of observation along the trajectory of the vehicle,

$$\mathcal{C}_r[\boldsymbol{\varphi}] \triangleq \bigcup_{t=0}^{T} \mathcal{B}_r(\boldsymbol{\varphi}(t)), \tag{2.7}$$

as illustrated in Figure 2.5. The usefulness of the set $\mathcal{C}_r[\boldsymbol{\varphi}]$ arises from the fact that each position is only taken into account once to increase the uncertainty integration since we are assuming that the UAV has a perfect quality of exploration.

(a) 3D illustration (b) Top view

Figure 2.5: Illustration of the set $\mathcal{C}_r[\boldsymbol{\varphi}]$.

Solving the problem in (2.4) is very complex since the objective functional J, as defined in (2.6), does not have a closed-form expression. Therefore, a relaxed formulation needs to be considered. In addition, to make the problem computationally tractable, it also needs to be discretized. However, even after relaxing and discretizing the problem, solving the problem globally for a relatively large time horizon T is computationally challenging. Consequently, we consider a local approach based on MPC to approximate the solutions of (2.4) while adding the possibility for feedback to the control law.

2.3 Model-Predictive Approach

In order to address the problem defined in the previous section, we follow an MPC-based approach. As described in Section 1.3, MPC involves the solution of an open-loop discrete-time optimal control problem at each sampling time k. Each of these optimizations results in a sequence of future optimal control actions and a sequence of corresponding future states. The first control action in the sequence is applied to the plant, and then the optimization is solved again at the next sampling time.

More specifically, at every discrete-time instant k, for a given initial state $\mathbf{x}_k$ of the system, the control policy is defined by solving a discrete-time optimal control problem of the form

$$\begin{aligned} \underset{\hat{\mathbf{X}}_k, \hat{\mathbf{U}}_k}{\text{maximize}} \quad & J_k(\hat{\mathbf{X}}_k, \hat{\mathbf{U}}_k) \\ \text{subject to} \quad & \hat{\mathbf{x}}_{k,0} = \mathbf{x}_k, \\ & \hat{\mathbf{x}}_{k,j+1} = \mathbf{f}(\hat{\mathbf{x}}_{k,j}, \hat{\mathbf{u}}_{k,j}),\ j = 0, \ldots, N-1, \\ & \hat{\mathbf{x}}_{k,j} \in \mathcal{X},\ j = 0, \ldots, N, \\ & \hat{\mathbf{u}}_{k,j} \in \mathcal{U},\ j = 0, \ldots, N-1, \end{aligned} \tag{2.8}$$

where N is the horizon length, the sets $\mathcal{X}$ and $\mathcal{U}$ constitute the admissible states and inputs for the vehicle, and the function $\mathbf{f}$ represents a discrete-time version of the vehicle dynamics. The matrices $\hat{\mathbf{X}}_k$ and $\hat{\mathbf{U}}_k$ are the optimization variables and represent the predicted state and control sequences over the time horizon at time instant k, i.e.,

$$\begin{aligned} \hat{\mathbf{X}}_k &\triangleq \begin{bmatrix} \hat{\mathbf{x}}_{k,0} & \hat{\mathbf{x}}_{k,1} & \ldots & \hat{\mathbf{x}}_{k,N-1} & \hat{\mathbf{x}}_{k,N} \end{bmatrix}, \\ \hat{\mathbf{U}}_k &\triangleq \begin{bmatrix} \hat{\mathbf{u}}_{k,0} & \hat{\mathbf{u}}_{k,1} & \ldots & \hat{\mathbf{u}}_{k,N-1} \end{bmatrix}. \end{aligned} \tag{2.9}$$

The input applied to the system at the discrete-time instant k, $\mathbf{u}_k$, is given by

$$\mathbf{u}_k = \hat{\mathbf{u}}^*_{k,0}, \tag{2.10}$$

where $\hat{\mathbf{u}}^*_{k,0}$ is the first sample of the predicted optimal control sequence at time instant k. The optimization problem in (2.8) may be solved efficiently using available NLP solvers.

2.3.1 Objective Function

In order to approximate the problem described in the previous section, we define the MPC objective function as a combination of two objectives as

$$J_k(\hat{\mathbf{\Phi}}_k) = \tilde{J}_k(\hat{\mathbf{\Phi}}_k) - \lambda P_k(\hat{\mathbf{\Phi}}_k), \tag{2.11}$$

where $\hat{\mathbf{\Phi}}_k \triangleq [\hat{\boldsymbol{\varphi}}_{k,0}\ \hat{\boldsymbol{\varphi}}_{k,1}\ \ldots\ \hat{\boldsymbol{\varphi}}_{k,N}] = \mathbf{C}\hat{\mathbf{X}}_k$ is the predicted discrete-time trajectory of the vehicle at the discrete-time instant k and $\lambda > 0$ is a scaling coefficient.

The first term in (2.11), $\tilde{J}_k$, represents the objective of prioritizing the most uncertain areas and is defined as

$$\tilde{J}_k(\hat{\mathbf{\Phi}}_k) = \sum_{j=0}^{N} \int_{\mathcal{B}_r(\hat{\boldsymbol{\varphi}}_{k,j})} h(\mathbf{p})\, d\mathbf{p}. \tag{2.12}$$

However, this term does not consider the previously covered areas neither the intersections between the areas of observation within the prediction horizon. Consequently, if the objective function was defined by this term alone, the trajectories would converge to an uncertainty maximizer and remain at the maximizer. Therefore, to encode the previously covered areas and the intersections between the areas of observation within the prediction horizon, we add a penalization term $P_k(\hat{\mathbf{\Phi}}_k)$ to the objective function.

The penalization term is constructed by penalizing intersections between the circles of observation along the trajectory of the vehicle. Two types of intersections need to be considered: intersections between the predicted circles and circles corresponding to positions that were already covered, and intersections between the predicted circles of observation. Thus, the penalization term has the form

$$P_k(\hat{\mathbf{\Phi}}_k) = P_k^B(\hat{\mathbf{\Phi}}_k) + P_k^H(\hat{\mathbf{\Phi}}_k), \tag{2.13}$$

where $P_k^B(\hat{\mathbf{\Phi}}_k)$ penalizes intersections between the predicted circles and previously covered circles, and $P_k^H(\hat{\mathbf{\Phi}}_k)$ penalizes intersections within the prediction horizon. Hence, assuming that $p : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}_0^+$ is a function that penalizes the intersection between two circles and that $\boldsymbol{\varphi}_i$ is the actual position of the vehicle at the discrete-time instant i, $P_k^B(\hat{\mathbf{\Phi}}_k)$ is defined by

$$P_k^B(\hat{\mathbf{\Phi}}_k) = \sum_{j=1}^{N} \sum_{i=0}^{k} p(\hat{\boldsymbol{\varphi}}_{k,j}, \boldsymbol{\varphi}_i), \tag{2.14}$$

and $P_k^H(\hat{\mathbf{\Phi}}_k)$ is defined as

$$P_k^H(\hat{\mathbf{\Phi}}_k) = \sum_{j=2}^{N} \sum_{i=1}^{j-1} p(\hat{\boldsymbol{\varphi}}_{k,j}, \hat{\boldsymbol{\varphi}}_{k,i}). \tag{2.15}$$

To conclude the definition of the objective function, it remains to define how the penalty function p is constructed, which is addressed in the following subsection.

Prior to moving forward, it is worth noting that the integrals presented in (2.12) still do not have a closed-form expression. Nevertheless, since the integrals are now computed over circular domains, they may be approximated through numerical methods such as quadrature rules [36] or simply by discretizing the area of observation using a grid. However, we will typically consider examples where the radius of observation is small when compared to the structure of the uncertainty map, and, therefore, (2.12) may be well approximated by

$$\tilde{J}_k(\hat{\mathbf{\Phi}}_k) \simeq \pi r^2 \sum_{j=0}^{N} h(\hat{\boldsymbol{\varphi}}_{k,j}). \tag{2.16}$$

2.3.2 Penalty Function

A plausible definition for the penalty function p would be the intersection area between two circles, as detailed in Figure 2.6.

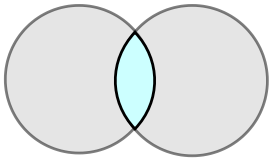


Figure 2.6: Overlap between two circles.

The area of intersection between two circles centered at the positions $\mathbf{c}_1$ and $\mathbf{c}_2$, both with the same radius r, can be computed analytically by

$$a(\mathbf{c}_1, \mathbf{c}_2) = \begin{cases} 2r^2 \arccos\left(\frac{1}{2r}\|\mathbf{c}_1 - \mathbf{c}_2\|\right) - \|\mathbf{c}_1 - \mathbf{c}_2\| \sqrt{r^2 - \|\mathbf{c}_1 - \mathbf{c}_2\|^2}, \text{ if } \|\mathbf{c}_1 - \mathbf{c}_2\| \leq 2r \\ 0, \text{ if } \|\mathbf{c}_1 - \mathbf{c}_2\| > 2r \end{cases} . \quad (2.17)$$

However, an expression of such complexity would be a computational bottleneck. In addition, the function in (2.17) is piecewise defined, posing additional implementation difficulties. For instance, the logic condition would have to be addressed through the Big-M notation from YALMIP [26], which serves to convert the logic condition into a set of constraints using auxiliary binary variables and logic constraints.

Nevertheless, it is not necessary to precisely calculate the overlap area between two circles to penalize the intersection between them. Such penalization might be achieved by constructing a function that simply penalizes the condition of existing intersection. Consequently, we design the penalty function by applying an exponential penalty to the violation of the condition $\|\mathbf{c}_1 - \mathbf{c}_2\| > 2r$ as

$$p(\mathbf{c}_1, \mathbf{c}_2) = \exp\left\{\gamma\left((2r)^2 - \|\mathbf{c}_1 - \mathbf{c}_2\|^2\right)\right\} - 1, \quad (2.18)$$

where $\gamma > 0$ is a parameter that can be tuned. Additionally, the subtraction of 1 is included so that the function has a value of zero when $\|\mathbf{c}_1 - \mathbf{c}_2\| = 2r$, but it has no effect on the optimization since it is a constant term. Figure 2.7 illustrates the evolution of the penalization as a function of the distance between the centers of the two circles.

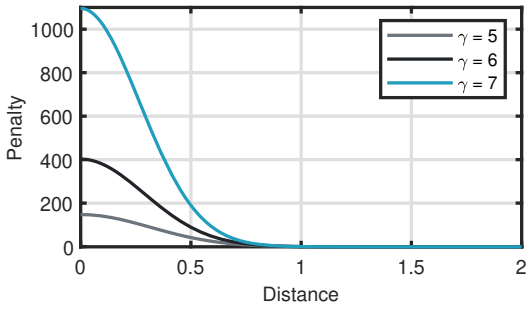


Figure 2.7: Illustration of the penalty function for some values of γ while considering $r = 0.5$.

2.3.3 Computational Complexity

From a computational standpoint, it is essential to assess the complexity of the proposed algorithm. Besides the inherent complexity of the problem, determined by the structure of the uncertainty map and the imposed restrictions, it is crucial to examine the number of terms comprising the objective function, which directly impacts the number of evaluations that the solver must carry out. In particular, it is worth noting that the number of terms comprising $\tilde{J}_k$ and P_k^H is determined by the prediction horizon length. More specifically, the number of terms in $\tilde{J}_k$ increases linearly with the horizon length, while P_k^H comprises $N(N-1)/2$ terms and, therefore, grows quadratically with the horizon.

Besides the quadratic growth of P_k^H as the horizon length increases, a significant computational burden arises from P_k^B. At each time instant k, the number of terms in P_k^B increases by N, meaning that P_k^B grows linearly with the flight time assigned for the surveillance mission. One apparent solution could involve defining a maximum backward horizon length N_B, thereby limiting P_k^B to a maximum number of terms. Consequently, in such a case, P_k^B would be given by

$$P_k^B(\hat{\mathbf{\Phi}}_k) = \sum_{j=1}^{N} \sum_{i=k-N_B+1}^{k} p(\hat{\boldsymbol{\varphi}}_{k,j}, \boldsymbol{\varphi}_i). \tag{2.19}$$

Nevertheless, if the backward time horizon is not sufficiently long, the vehicle would possibly revisit previously covered areas. Therefore, a better future approach revolves around developing a subroutine that can progressively reduce the number of components in the penalization term while retaining the information about all the previously explored regions.

Additionally, it is essential to clarify that despite the notion that the objective function grows at each time step, the optimization solvers are constructed by allocating the necessary resources for the entire mission duration. This decision follows from the substantial additional overhead that there would be in building a solver at each time instant k. Hence, the number of terms in the objective function is actually constant throughout the whole mission, with the terms regarding future time steps in P_k^B being attributed a null weight. As a result, despite potential fluctuations introduced by the problem, the computational times are expected to remain approximately constant throughout the surveillance mission.

2.3.4 Evaluation Metric

It is necessary to establish an overall metric to evaluate the performance of the algorithm and perform comparisons. In this context, a reliable method of assessing the quality of the generated trajectories is computing the time evolution of the volume of uncertainty covered by the vehicle. By disregarding the coverage between sampling times, this metric can be approximated as

$$H_k(\mathbf{\Phi}_k) = \int_{\bigcup_{i=0}^{k} \mathcal{B}_r(\boldsymbol{\varphi}_i)} h(\mathbf{p})\, d\mathbf{p}, \tag{2.20}$$

where $\mathbf{\Phi}_k \triangleq [\boldsymbol{\varphi}_0 \ \boldsymbol{\varphi}_1 \ \dots \ \boldsymbol{\varphi}_k]$ is the discrete-time trajectory of the vehicle until time instant k. The numerical approximation of (2.20) is accomplished by discretizing the map into a grid.

2.4 Quadrotor Motion Control

We are particularly interested in multirotor aerial vehicles because of their agility, hovering performance, low cost, and production ease. Moreover, in this project, a quadrotor is available for performing experimental tests of the proposed MPC algorithm. Therefore, this section presents the control architecture used to implement the proposed MPC algorithm on a quadrotor aerial vehicle, as well as the dynamic model of the quadrotor.

2.4.1 Control Architecture

We consider a dual-layer structure of motion control, as illustrated in Figure 2.8. The proposed MPC algorithm serves as a high-level controller (trajectory planner) that generates high-level references for the UAV. The inner-loop controller (trajectory tracker) directly applies control inputs to the vehicle to accurately track the references provided by the MPC algorithm. For the purpose of efficiency, the MPC algorithm considers a simplified model of the vehicle, while the lower-level controller takes into account the full dynamic model of the quadrotor.

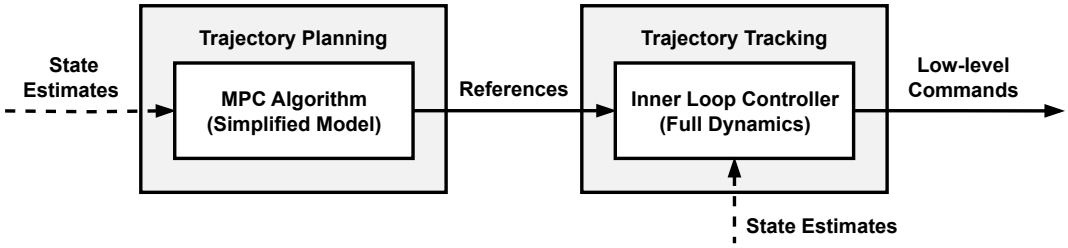


Figure 2.8: Full motion control scheme of the UAV.

2.4.2 Full Dynamics

For completeness, we start by presenting the full nonlinear dynamics of a quadrotor. The nonlinear dynamics of the quadrotor are described in the body $\{B\}$ and inertial $\{I\}$ frames depicted in Figure 2.9, while assuming that the origin of $\{B\}$ is coincident with the center of mass of the quadrotor.

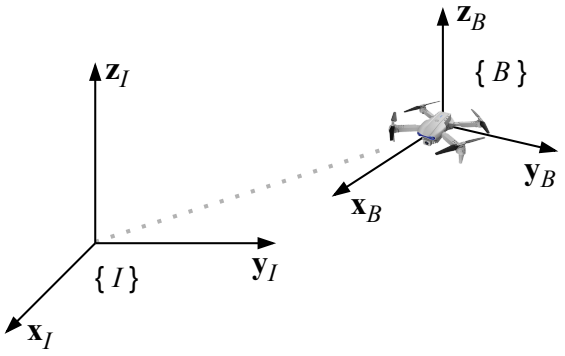


Figure 2.9: Quadrotor reference frames.

Let $\mathbf{p} \in \mathbb{R}^3$ denote the position of the quadrotor's center of mass in the inertial frame. Let $\boldsymbol{\eta} = [\phi\ \theta\ \psi]^\top$ describe the orientation of the of the body frame with respect to the inertial frame, where ϕ, θ and ψ are the roll, pitch and yaw angles, respectively. Let $\mathbf{v}$ denote the linear velocity of $\{B\}$ with respect to $\{I\}$ expressed in $\{I\}$. Let $\boldsymbol{\omega} = [p\ q\ r]^\top$ denote the angular velocity of $\{B\}$ with respect to $\{I\}$, this time expressed in $\{B\}$. Finally, let m be the mass of the rigid object, $\mathbf{I} \in \mathbb{R}^{3\times3}$ the inertia matrix expressed in $\{B\}$, and g the gravitational acceleration. The quadrotor equations of motion, based on the Newton-Euler formalism [37], are given by

$$\begin{aligned} \dot{\mathbf{p}} &= \mathbf{v}, \\ m\dot{\mathbf{v}} &= -mg\mathbf{e}_3 + {}^I\mathbf{R}_B(\boldsymbol{\eta})\, F_T \mathbf{e}_3, \\ \dot{\boldsymbol{\eta}} &= \mathbf{T}(\boldsymbol{\eta})\, \boldsymbol{\omega}, \\ \mathbf{I}\dot{\boldsymbol{\omega}} &= -\boldsymbol{\omega} \times \mathbf{I}\boldsymbol{\omega} + \boldsymbol{\tau}, \end{aligned} \tag{2.21}$$

where F_T is the net thrust and $\boldsymbol{\tau} = [\tau_\phi\ \tau_\theta\ \tau_\psi]^\top$ is the vector of moments applied to the UAV described in $\{B\}$. Additionally, ${}^I\mathbf{R}_B(\boldsymbol{\eta}) \in SO(3)$ is the rotation matrix from $\{B\}$ to $\{I\}$ and $\mathbf{T}(\boldsymbol{\eta}) \in \mathbb{R}^{3\times3}$ is a matrix that converts the angular velocity to angle rates.

Assuming that the Euler angles follow the sequence of rotation Z-Y-X that is described in [38], ${}^I\mathbf{R}_B(\boldsymbol{\eta})$ is given by

$${}^I\mathbf{R}_B(\boldsymbol{\eta}) = \begin{bmatrix} \cos\theta\cos\psi & \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi \\ \cos\theta\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta \end{bmatrix}, \tag{2.22}$$

and $\mathbf{T}(\boldsymbol{\eta})$ is

$$\mathbf{T}(\boldsymbol{\eta}) = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{bmatrix}. \tag{2.23}$$

The steady-state thrust and the yaw moment generated by rotor i are modeled as

$$\begin{aligned} F_{T_i} &= K_i\Omega_i^2, \\ \tau_{\psi_i} &= C_i F_{T_i}, \end{aligned} \tag{2.24}$$

where Ω_i is the rotation speed of rotor i, and the constants K_i and C_i may be determined experimentally. The roll and pitch moments, τ_ϕ and τ_θ, result from the generated rotor thrusts and their arrangement relative to the quadrotor's center of mass. Therefore, the net thrust and moments, for a quadrotor with an X-configuration, are computed through

$$\begin{bmatrix} F_T \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ L & -L & -L & L \\ -L & -L & L & L \\ C_1 & -C_2 & C_3 & -C_4 \end{bmatrix} \begin{bmatrix} F_{T_1} \\ F_{T_2} \\ F_{T_3} \\ F_{T_4} \end{bmatrix}, \tag{2.25}$$

where L denotes the perpendicular distance of the rotors to the x or y axis of the body frame.

To conclude, we point out that there are additional aerodynamic effects, which would increase the complexity of the model. However, a model with such a level of precision is typically not required.

2.4.3 Simplified Model

At the trajectory planning level, considering that the UAV flies at a constant altitude and ignoring the fast rotational dynamics of the vehicle, the UAV might be modeled as a two-dimensional point-mass system that follows double-integrator dynamics. Thus, the state vector $\mathbf{x}$ is composed of the positions and velocities on the horizontal plane, $\mathbf{x} = [\boldsymbol{\varphi}^\top \ \dot{\boldsymbol{\varphi}}^\top]^\top$, and the control input $\mathbf{u}$ consists of the acceleration on the horizontal plane. Consequently, at the planning level, the quadrotor dynamics take the linear form

$$\begin{bmatrix} \dot{\boldsymbol{\varphi}} \\ \ddot{\boldsymbol{\varphi}} \end{bmatrix} = \begin{bmatrix} \mathbf{0}_{2\times2} & \mathbf{I}_{2\times2} \\ \mathbf{0}_{2\times2} & \mathbf{0}_{2\times2} \end{bmatrix} \begin{bmatrix} \boldsymbol{\varphi} \\ \dot{\boldsymbol{\varphi}} \end{bmatrix} + \begin{bmatrix} \mathbf{0}_{2\times2} \\ \mathbf{I}_{2\times2} \end{bmatrix} \mathbf{u}, \tag{2.26}$$

where $\mathbf{0}_{2\times2}$ denotes a matrix of zeros and $\mathbf{I}_{2\times2}$ denotes the identity matrix, both with dimension 2×2.

The MPC algorithm considers this simplified model of the UAV, which reduces the computational cost of the optimization process, while the inner-loop controller processes the remaining dynamics of the UAV. This mismatch is not critical for obtaining good performance in real conditions as long as the generated trajectories are not extremely aggressive so that the inner-loop dynamics become visible.

2.4.4 Actuation Limits

As any other vehicle, a quadrotor is subject to limitations imposed by its actuators. In this case, the maximum thrust magnitude of the quarotor is limited. At any time, the quadrotor should have a vertical force to balance its weight, i.e., a force of magnitude mg. Then, it must be able to maneuver around this equilibrium. A descent can be achieved by decreasing the vertical force that balances the weight. However, to ascend it must be able to produce a higher thrust on the vertical direction.

In the vertical direction, the quadrotor should have available a thrust force of magnitude $m(g + a_z^{\max})$, where $a_z^{\max} \in \mathbb{R}^+$ represents the maximum acceleration along the vertical axis. Let $F_T^{\max} \in \mathbb{R}^+$ be the maximum thrust magnitude that the quadrotor can produce, and $a_{xy}^{\max} \in \mathbb{R}^+$ be the maximum acceleration on the horizontal plane. The maximum acceleration on the horizontal plane can be computed by

$$a_{xy}^{\max} = \sqrt{(F_T^{\max}/m)^2 - (g + a_z^{\max})^2}. \tag{2.27}$$

The saturation along the vertical direction must first be chosen, to ensure that the multirotor is capable of maintaining its altitude. Then, the horizontal saturation is a result of the choice made about $a_z^{\max}$.

2.4.5 Implementation Details

In practical terms, the proposed motion control scheme is implemented using a PX4 Autopilot [39]. The PX4 Autopilot provides both the inner-loop controller and an Extended Kalman Filter (EKF) to process sensor measurements. In this implementation, the MPC algorithm generates references to be tracked by the inner-loop controller provided by the PX4 Autopilot, while both controllers receive the corresponding state estimates provided by the EKF algorithm. Figure 2.10 illustrates the described implementation.

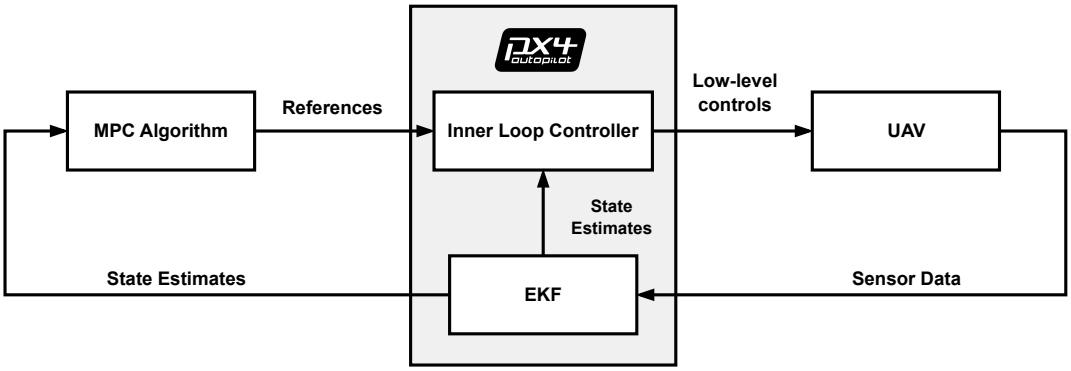


Figure 2.10: Implementation of the proposed motion control scheme.

As detailed in Figure 2.11, the controller supplied by the PX4 Autopilot follows a standard cascaded architecture with several stages. Each stage is composed of a proportional or Proportional-Integral-Derivative (PID) controller that generates references for the upcoming stage based on references provided by the previous stage. From a general perspective, the PX4 controller consists of two main control loops: position and attitude. The position control loop commands accelerations, which are then converted into attitude and net thrust references. The attitude control loop receives attitude and net thrust references and commands low-level thrust references for the vehicle motors.

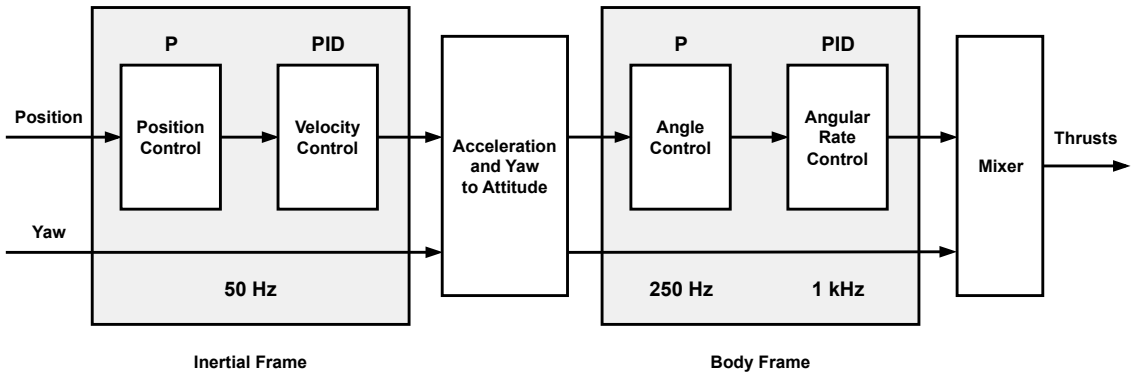


Figure 2.11: PX4 controller architecture (adapted from PX4 documentation).

With such an architecture, the PX4 controller is able to receive different kinds of references from the MPC algorithm, from high-level references to low-level ones. In our case, we resort to high-level references such as position, velocity, or acceleration references, while keeping the altitude constant. The type of references used may then be adjusted given the experimental results obtained.

2.5 Simulation Results

In this section, the efficacy of the proposed MPC algorithm is assessed through different simulation examples obtained within a MATLAB environment. The objective is to analyze the behavior and performance of the algorithm, as well as the quality of the generated trajectories. We begin by presenting some simulations that illustrate the trajectories that the algorithm is able to produce. Subsequently, we study the influence of some parameters on the algorithm.

2.5.1 Simulation Setup

The goal of this section is to perform an initial analysis of the behavior of the proposed MPC algorithm. Therefore, the simulations presented in this section are performed assuming that the UAV follows ideal double-integrator dynamics. The full nonlinear dynamics of the UAV and the PX4 inner-loop controller are then included in the simulations and experiments of Section 2.6. At each discrete-time instant k, the MPC algorithm is based on the following optimization problem

$$\begin{aligned}
\underset{\hat{\mathbf{X}}_k, \hat{\mathbf{U}}_k}{\text{maximize}} \quad & J_k(\hat{\mathbf{X}}_k, \hat{\mathbf{U}}_k) \\
\text{subject to} \quad & \hat{\mathbf{x}}_{k,0} = \mathbf{x}_k, \\
& \hat{\mathbf{x}}_{k,j+1} = \mathbf{A}\hat{\mathbf{x}}_{k,j} + \mathbf{B}\hat{\mathbf{u}}_{k,j},\ j = 0, \dots, N-1, \\
& \|\mathbf{C}'\hat{\mathbf{x}}_{k,j}\| \leq v_{xy}^{\max},\ j = 0, \dots, N, \\
& \|\hat{\mathbf{u}}_{k,j}\| \leq a_{xy}^{\max},\ j = 0, \dots, N-1,
\end{aligned} \tag{2.28}$$

where the objective function J_k is obtained as described in Section 2.3, and the matrices $\mathbf{A}$ and $\mathbf{B}$, corresponding to the discrete-time double-integrator dynamics (zero-order hold), are given by

$$\begin{aligned}
\mathbf{A} &= \begin{bmatrix} \mathbf{I}_{2\times 2} & T_s\,\mathbf{I}_{2\times 2} \\ \mathbf{0}_{2\times 2} & \mathbf{I}_{2\times 2} \end{bmatrix}, \\
\mathbf{B} &= \begin{bmatrix} T_s^2/2\,\mathbf{I}_{2\times 2} \\ T_s\,\mathbf{I}_{2\times 2} \end{bmatrix}.
\end{aligned} \tag{2.29}$$

The auxiliary matrix $\mathbf{C}'$, given by

$$\mathbf{C}' = \begin{bmatrix} \mathbf{0}_{2\times 2} & \mathbf{I}_{2\times 2} \end{bmatrix}, \tag{2.30}$$

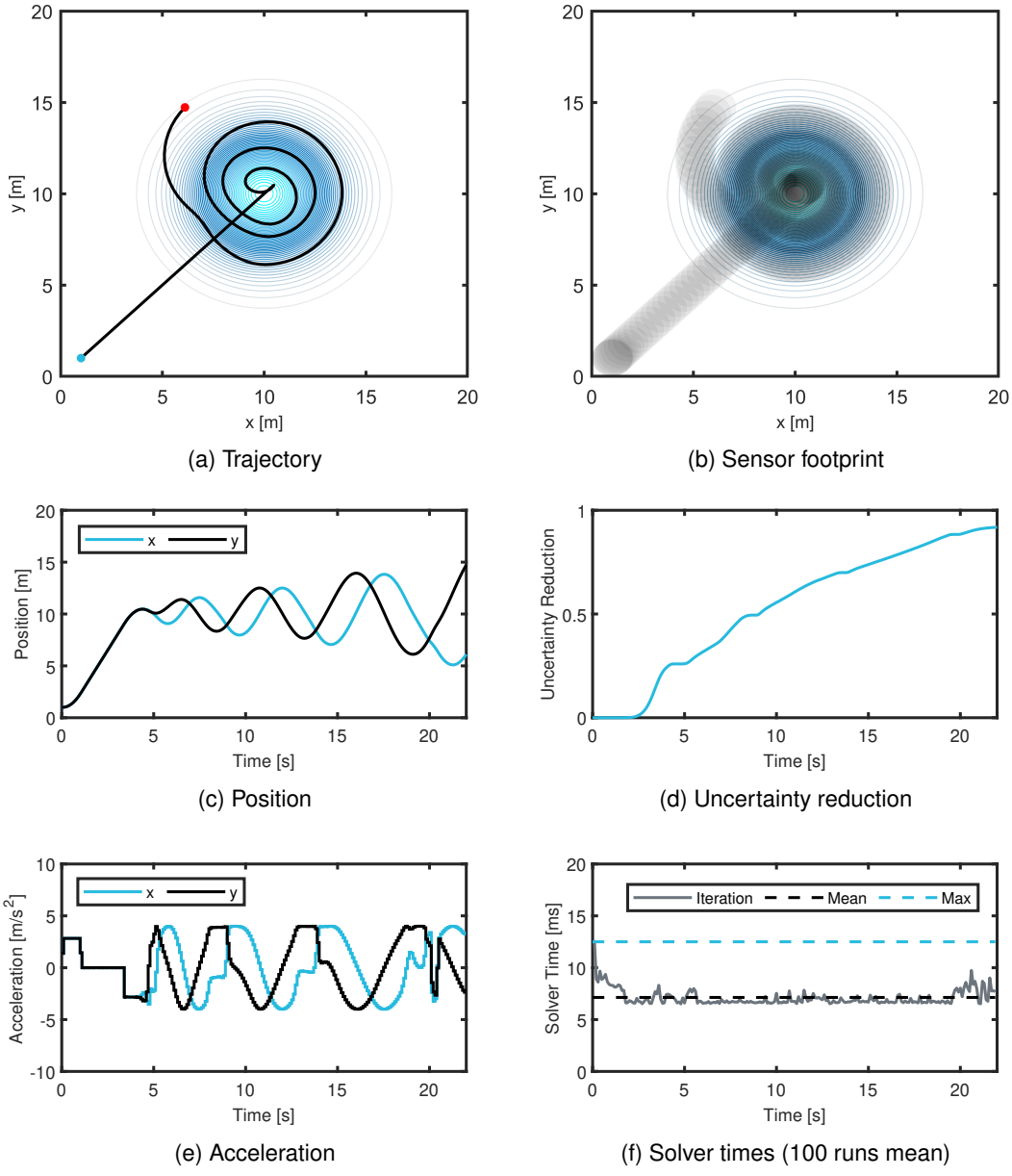
extracts the velocity from the state. The parameters $v_{xy}^{\max}$ and $a_{xy}^{\max}$ are, respectively, the maximum velocity and acceleration that the vehicle may achieve on the horizontal plane.

The simulation results presented in this section were obtained in MATLAB [40] using the CasADi [27] optimization modeling toolbox, along with the IPOPT [28] numerical solver. At each sampling time, the solution obtained at the previous step is used to set the initial guess for the current step by performing the shifting warm-start method described in Section 1.3. All computations were executed on a single desktop computer equipped with an Intel Core i7-6700K @ 4.00 GHz processor and 32.00 GB of RAM.

In the following examples, the drone starts at the position $\mathbf{p} = [1\ 1]^\top$ with no initial velocity, and the radius of observation is assumed to be $r = 1$ m. The sampling period is $T_s = 0.1$ s, the horizon length is $N = 15$, and the vehicle has a maximum velocity of 4 m/s and a maximum acceleration of 4 m/s^2.

2.5.2 Example 1

We begin by presenting a simple simulation where the uncertainty function is composed of only one Gaussian component with a circular shape. The simulation results are shown in Figure 2.12.



(a) Trajectory (b) Sensor footprint

(c) Position (d) Uncertainty reduction

(e) Acceleration (f) Solver times (100 runs mean)

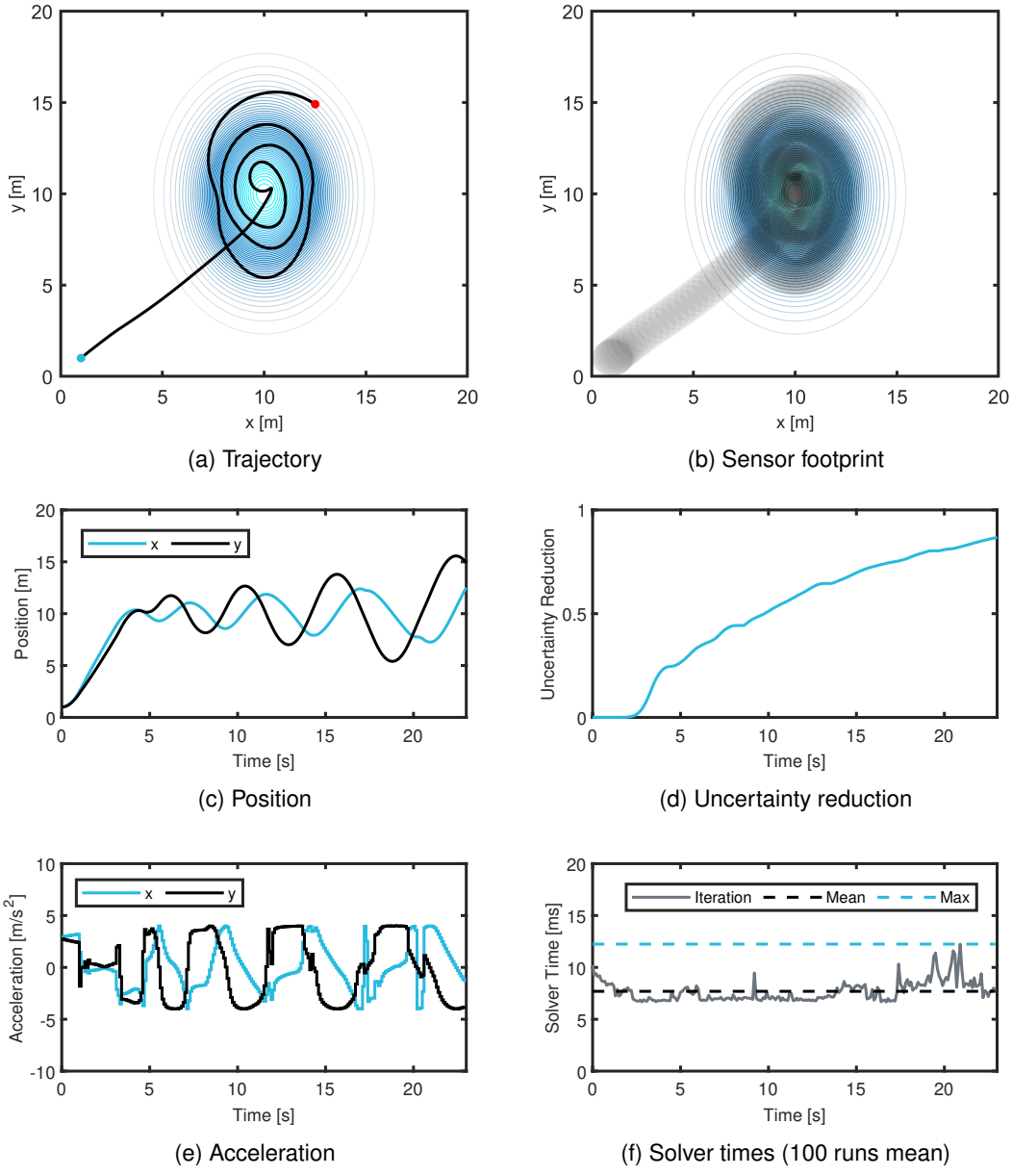
Figure 2.12: Simple simulation with one Gaussian component with a circular shape.

As shown in Figure 2.12 (a), initially the vehicle moves towards the maximum of the Gaussian component. Subsequently, as a result of the penalizations applied by the algorithm, the vehicle goes to wider areas by executing a spiral curve. The evolution of the x and y components of the position and accel-

eration of the vehicle is depicted in Figures 2.12 (c) and 2.12 (e). Moreover, Figure 2.12 (b) illustrates the sensor footprint of the UAV, and Figure 2.12 (d) shows the accumulation of the uncertainty volume covered by the vehicle. In addition, we draw attention to Figure 2.12 (f), which presents the mean solver times acquired through 100 simulations, with each iteration taking approximately 7 ms on average.

2.5.3 Example 2

We introduce another simple simulation, in which the uncertainty function consists of a Gaussian component with an elliptical shape. As depicted in Figure 2.13 (a), the trajectory adjusts itself to the elliptical shape of the Gaussian component. Moreover, as shown in Figures 2.13 (d) and 2.13 (f), the resulting uncertainty reduction profile and the mean solver times are similar to those from the previous example.



(a) Trajectory (b) Sensor footprint

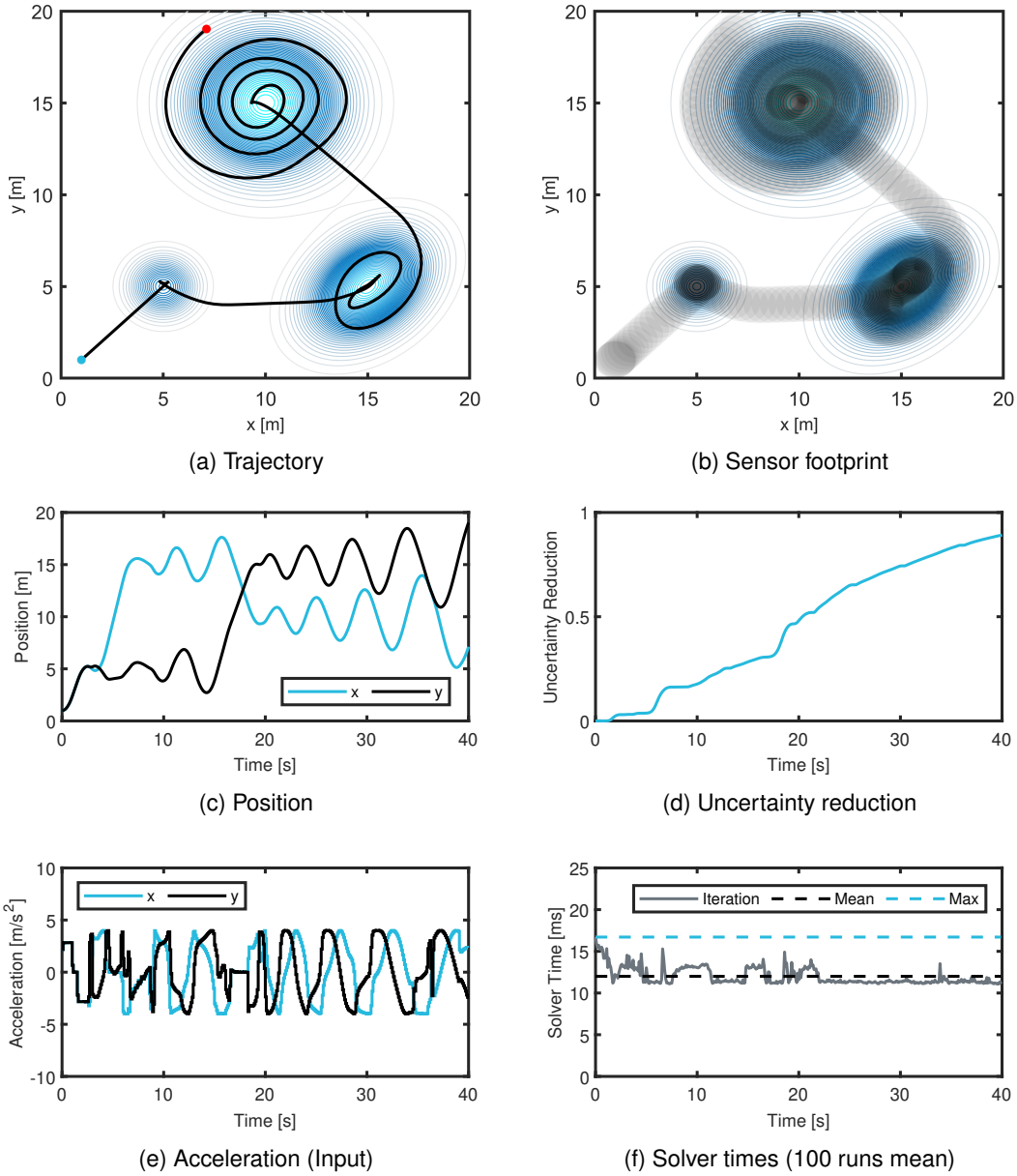
(c) Position (d) Uncertainty reduction

(e) Acceleration (f) Solver times (100 runs mean)

Figure 2.13: Simple simulation with one Gaussian component with an elliptical shape.

2.5.4 Example 3

Now we introduce a more complex example where the uncertainty map comprises three Gaussian components, with the simulation results displayed in Figure 2.14. As depicted in Figure 2.14 (a), the drone analyzes each component individually. In particular, it is worth noting that the components with means at the positions $\mathbf{p} = [15\ 5]^\top$ and $\mathbf{p} = [10\ 15]^\top$ are similar to those from the previous examples, and the flight paths observed when the drone analyzes such components are also similar to the previous ones. However, the third component located at $\mathbf{p} = [5\ 5]^\top$ has a smaller variance when compared to the observation radius of the UAV. Consequently, when the drone analyzes this component, it simply remains at the maximum of the component. Additionally, it should be noticed that the solver times are slightly higher in this example, with each iteration averaging approximately 12 ms.



(a) Trajectory (b) Sensor footprint

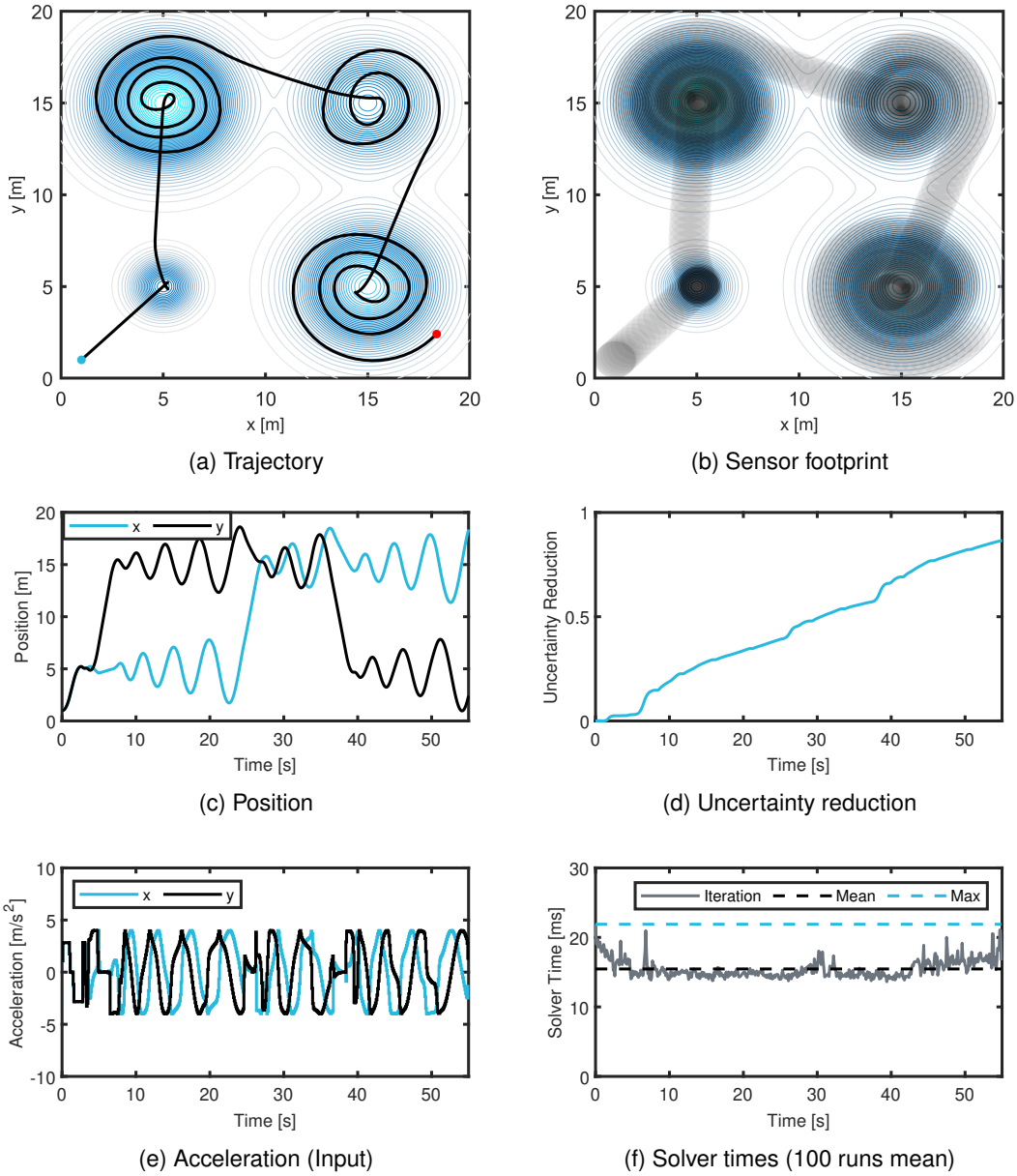
(c) Position (d) Uncertainty reduction

(e) Acceleration (Input) (f) Solver times (100 runs mean)

Figure 2.14: Example where the uncertainty map comprises three Gaussian components.

2.5.5 Example 4

We present another example in which the uncertainty map is now composed of four radially-symmetric Gaussian components, with the simulation results displayed in Figure 2.15. As it can be observed in Figure 2.15 (a), the component with mean at $\mathbf{p} = [5\ 5]^\top$ is similar to the one from the previous example, and the vehicle exhibits a similar behavior when analyzing this specific component. The remaining components with means at $\mathbf{p} = [5\ 15]^\top$, $\mathbf{p} = [15\ 15]^\top$, and $\mathbf{p} = [15\ 5]^\top$ all have similar covariance matrices but different associated weights. By observing Figures 2.15 (a) and 2.15 (b), one can notice that, as the weights of the components increase, the spiral curves become more tightly concentrated, and there is a greater overlap of the vehicle's observation circles. In this example, each solver iteration averages approximately 16 ms, as shown in Figure 2.15 (f).



(a) Trajectory (b) Sensor footprint

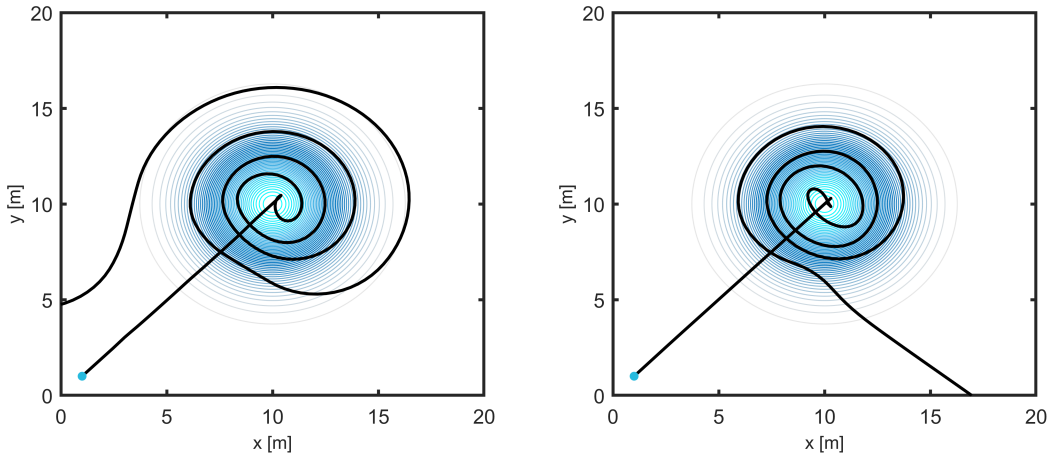
(c) Position (d) Uncertainty reduction

(e) Acceleration (Input) (f) Solver times (100 runs mean)

Figure 2.15: Example where the uncertainty map comprises four radially-symmetric components.

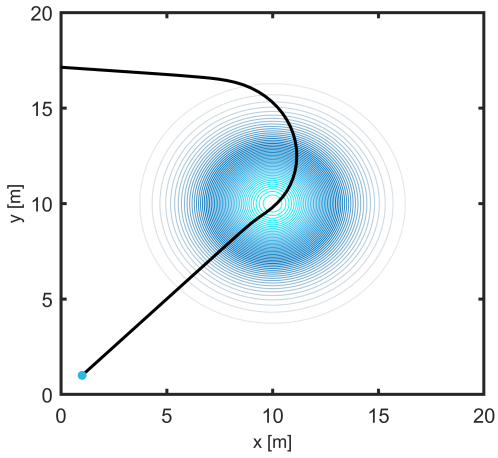
2.5.6 Effect of the Weights

Given that the objective function of the proposed algorithm relies on the exponent γ to penalize intersections and the scaling coefficient λ, it is essential to assess how these two parameters influence the algorithm. In this context, we consider the conditions of the initial example, where the uncertainty map consists of a single radially-symmetric Gaussian component, and we manipulate the weights λ and γ. Figure 2.16 displays the resulting trajectories for four distinct combinations of values of λ and γ.

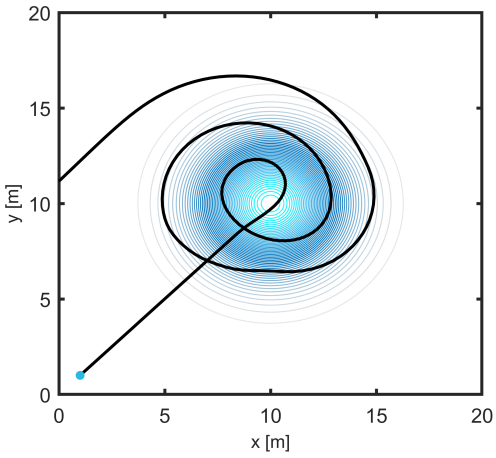


(a) $\lambda = 1/7000$, $\gamma = 0.4$

(b) $\lambda = 1/10000$, $\gamma = 0.4$



(c) $\lambda = 1/7000$, $\gamma = 1$



(d) $\lambda = 1/7000$, $\gamma = 0.8$

Figure 2.16: Trajectories obtained for different values of λ and γ.

As λ decreases in value, less emphasis is placed on the penalization term. Consequently, the trajectories are expected to become more tightly concentrated, resulting in a greater overlap of the vehicle's observation circles. This effect is evident in the examples depicted in Figures 2.16 (a) and 2.16 (b), and it becomes more pronounced when examining Figure 2.17, which illustrates the evolution of the uncertainty volume covered by the vehicle for the various scenarios presented in Figure 2.16. As shown in Figure 2.17, in the case of Figure 2.16 (b), the vehicle remains closer to the peak of the Gaussian component for an extended duration compared to Figure 2.16 (a), resulting in a slower initial convergence. However, note that at $t = 30$ s, both trajectories exhibit a similar coverage.

A similar impact can be anticipated when examining the variation of γ. As the value of γ increases, the penalizations become more pronounced, leading to the expectation of a reduced overlap in the resulting trajectories. This effect is clearly observable in the examples presented in Figures 2.16 (a) and 2.16 (d). In particular, as shown in Figure 2.17, note that the trajectory in Figure 2.16 (d) initially exhibits a quicker convergence when compared to the trajectory in Figure 2.16 (a). However, at the final simulation instant $t = 30$ s, the uncertainty volume covered by the trajectory in Figure 2.16 (a) is greater than that achieved by the trajectory in Figure 2.16 (d). Additionally, Figure 2.16 (c) illustrates a more extreme case where γ is sufficiently high to prevent the vehicle from executing a spiral curve.

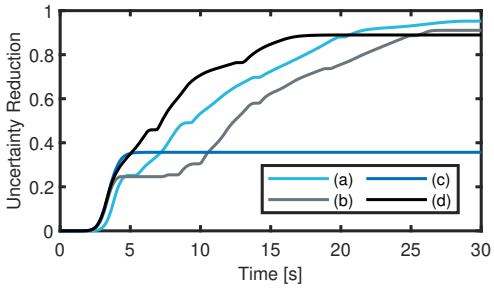


Figure 2.17: Uncertainty volume accumulation for the different values of λ and γ.

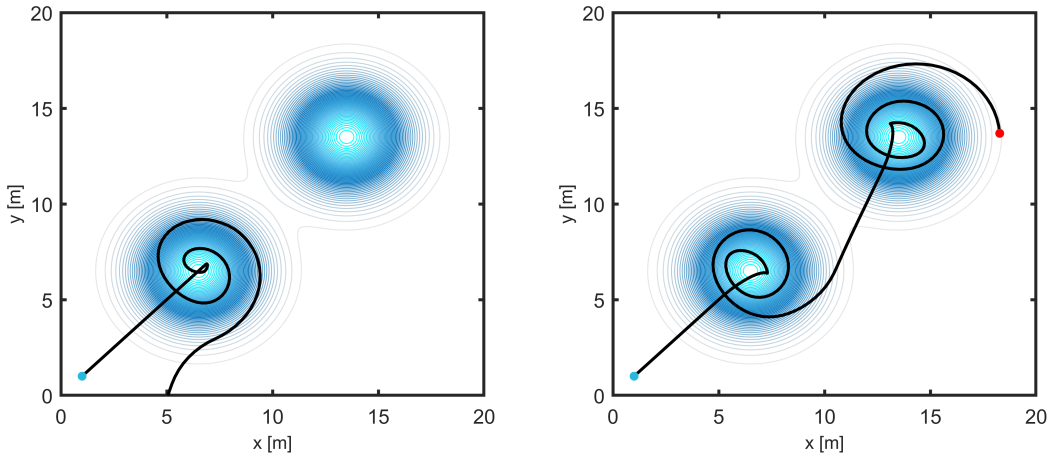
In light of the previous discussion, it is evident that there exists some parameter tuning associated with the proposed algorithm. Nevertheless, it should be acknowledged that the algorithm has the potential to be extended and generalized through the incorporation of variable weights. For instance, one could consider assigning higher penalizations in regions where the uncertainty function has higher values, and lower penalizations in regions where the uncertainty is lower. Moreover, one could employ decaying weights in the term $\tilde{J}_k$ of the objective function to prioritize earlier prediction instants, potentially resulting in a faster convergence. Such variations of the algorithm could be easily incorporated, and a more exhaustive analysis could be performed. However, the decision to implement these variations is left as a user choice and may be a subject of consideration in future research.

2.5.7 Effect of the Horizon

It is also important to evaluate how the performance of the proposed MPC algorithm is impacted by varying the length of the prediction horizon. In this context, we begin our analysis by considering an uncertainty map comprising two Gaussian components, with Figure 2.18 depicting the generated trajectories for two different prediction horizon lengths. As depicted in Figure 2.18 (a), for a horizon length of $N = 5$, the vehicle's predictive ability falls short and it is not able to predict the second Gaussian component. In contrast, when a longer horizon length is employed, as illustrated in Figure 2.18 (b), the vehicle is able to predict the second Gaussian component, resulting in a trajectory with a higher coverage.

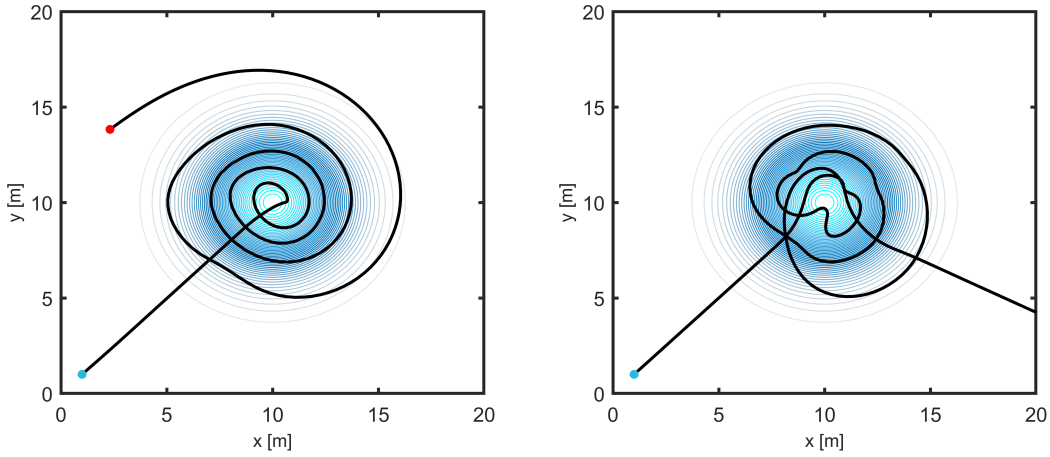
However, an extended horizon does not necessarily result in higher-quality trajectories, as illustrated in Figure 2.19. Specifically, as highlighted in Figure 2.19 (d), it can be noticed that the trajectories depicted in Figures 2.19 (a) and 2.19 (b) exhibit similar coverage profiles, and, in fact, the trajectory from Figure 2.19 (b) achieves a lower final coverage than the trajectory in Figure 2.19 (a). Furthermore, the

trajectory from Figure 2.19 (a) exhibits a smoother profile than that in Figure 2.19 (b). A longer prediction horizon also increases the computational load, as shown in Figure 2.19 (c). In the context of this simple simulation with a duration of 30 seconds and featuring a single Gaussian component, for a horizon of $N = 40$, each solver iteration already takes an average of approximately 35 ms, while for a horizon of $N = 15$ the average iteration time is about 8 ms.

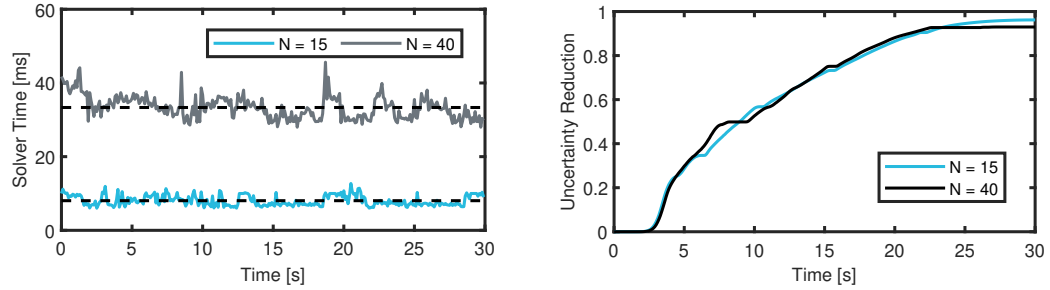


(a) N = 5 (b) N = 15

Figure 2.18: Trajectories obtained for two distinct prediction horizon lengths.



(a) N = 15 (b) N = 40



(c) Solver times (d) Uncertainty reduction

Figure 2.19: Results obtained for different prediction horizon lengths.

2.6 Experimental Validation

In this section, the efficacy of the proposed MPC algorithm is assessed through simulations in a higher-fidelity simulation software and by conducting actual experiments. We begin by providing a concise overview of the software architecture employed for simulating and carrying out tests in the physical drone. Subsequently, we showcase the results achieved from these evaluations.

2.6.1 Software Architecture

The software used to perform simulations and conduct actual experiments in the drone follows from the previous work done by Oliveira et al. [41] and Jacinto [42]. The employed software architecture is illustrated in Figure 2.20. The operating system consists of the Ubuntu 18.04 LTS version along with the melodic variant of the Robot Operating System (ROS). The instructions followed to install and configure the environment are presented in Appendix A. In the remainder of this section, we briefly describe each block shown in Figure 2.20.

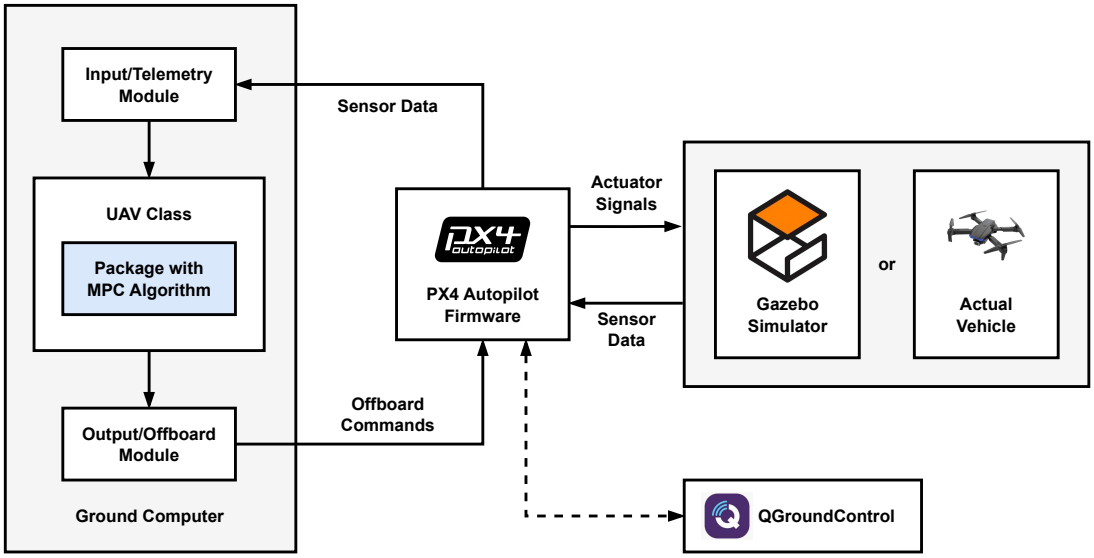


Figure 2.20: Simplified scheme of the employed software architecture.

2.6.2 Gazebo Simulator

The simulations conducted in this section are performed using the Gazebo simulator [43]. Gazebo stands out as a high-fidelity robotics simulator featuring a physics engine that accurately models the dynamic and kinematic characteristics of vehicles. Moreover, it offers the flexibility to incorporate various sensors and actuators via plugins. In essence, Gazebo simulates the motion of vehicles and the resulting sensor data, based on the inputs provided to the vehicles. One key advantage of Gazebo lies in its seamless integration with the ROS middleware. Such integration facilitates a modular approach to structure the entire system by defining each entity as a separate package and enabling communication via the publication and subscription of messages to topics and services.

2.6.3 PX4 Autopilot

PX4 [39] is an autopilot firmware that can operate within a vehicle, offering two distinct modes: Hardware In The Loop (HITL) and Software In The Loop (SITL). Its primary function is to act as an intermediary between the offboard modules and vehicle actuators. It provides essential functions such as sensor data acquisition, actuators control, estimators, safety features, and communication with external systems. As shown in Figure 2.20, the PX4 Autopilot provides raw sensor and estimator data and accepts commands to control the vehicle. The PX4 Autopilot software also includes a collection of available quadrotor models, in particular the Iris quadrotor [44] that is used in the simulations.

2.6.4 Ground Computer Stack

The Ground Computer stack, as illustrated in Figure 2.20, enables users to implement control algorithms and execute missions using the vehicles. The UAV class represents a UAV and is accessible thanks to the input and output modules developed by Oliveira et al. [41] and adapted by Jacinto [42]. The input module, responsible for subscribing to data published by the PX4 Autopilot, makes this data available to the user in a standardized manner. The output module consists of the methods that allow the user to send offboard commands and control references to the PX4 Autopilot. The proposed MPC algorithm is implemented within this framework using the C++ CasADi Application Programming Interface (API). The instructions followed to install CasADi are also provided in Appendix A. Additionally, QGroundControl is an application that serves as a graphical user interface offering comprehensive flight control and vehicle configuration capabilities.

2.6.5 Launching Simulations

In order to conduct simulations, a set of launch files is employed to initiate all the fundamental services. The launch files facilitate the initiation of services like Gazebo within the ROS environment, thereby granting further access to variables pertaining to the simulation and the state of the world. The hierarchical organization of the launch files is illustrated in Figure 2.21. Firstly, the control algorithm to execute is developed within a ROS package, and then the drone_sim.launch file is updated to call this package. Subsequently, the simulation may be initiated using the command ”roslaunch drone_sim_bringup simulator_bringup.launch.”

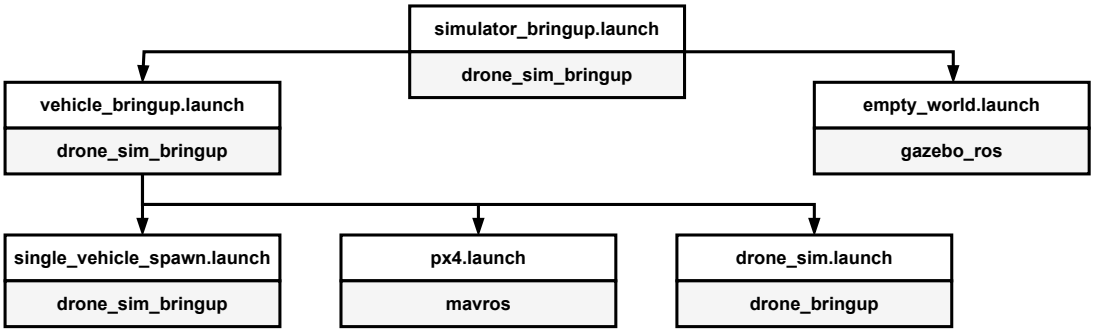


Figure 2.21: Hierarchical organization of the launch files.

2.6.6 Experimental Setup

In the upcoming subsections, we present the results obtained from different Gazebo simulations and the corresponding experimental tests performed in an outdoor environment (field trials). The Gazebo simulations were executed using the Iris quadrotor [44], available through the PX4 Autopilot SITL plugin. Meanwhile, the field trials were conducted using the M690B drone from a joint effort between FirePuma and Capture projects [45]. Additionally, to ensure validation prior to the experimental tests, we also conducted simulations using the Typhoon H480 hexacopter model, which shares more analogous characteristics with the M690B drone. However, the results were equivalent to those obtained using the Iris quadrotor. Figure 2.22 depicts the Iris and the M690B quadrotors used for the experimental validation.

(a) Iris quadrotor (Gazebo) (b) M690B (field trials)

Figure 2.22: Quadrotors used in the Gazebo simulations and field trials.

Concerning the Gazebo simulations, our initial approaches involved providing acceleration and, subsequently, velocity references to the PX4 controller. Despite our efforts, these approaches posed challenges in achieving smooth and stable trajectories consistent with those obtained in MATLAB. Nonetheless, when commanding a complete trajectory generated offline under identical vehicle constraints, it yielded the anticipated outcomes, enabling the vehicle to follow the trajectories with minimal error.

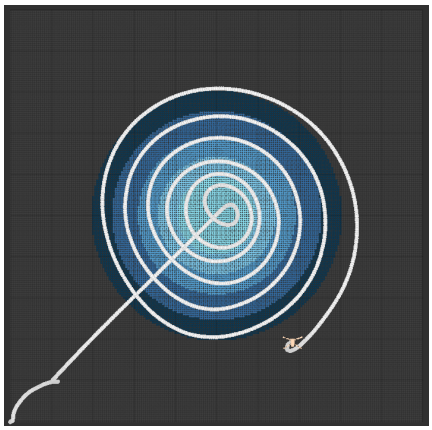
Despite the lack of significant advantages in running the algorithm in real-time in this particular scenario, there is a natural desire to enable the real-time execution of the algorithm to accommodate dynamic alterations in the future, like time-varying maps or obstacle avoidance. To enable the real-time execution of the algorithm and overcome the bad results obtained using lower-level references, we opted for a more conservative approach. The approach consists in sending a given slice of the predicted optimal sequence of position waypoints to the PX4 controller. With such an approach, the penalizations of the objective function are still updated at each sampling time k, but the optimization problem is only solved after the application of each sequence of waypoints. This method ultimately produced results similar to those obtained by commanding a complete trajectory generated *a priori*.

In the examples presented in the following subsections, the drone starts at the position $\mathbf{p} = [1\ 1]^\top$ with no initial velocity and the radius of observation is assumed to be $r = 0.5$ m. In the Gazebo simulations, the MPC operates with a sampling period of $T_s = 0.1$ s, a prediction horizon length of $N = 20$, and the first 5 predicted optimal waypoints are commanded to the PX4 controller. Consequently, the optimization problem is only solved from 0.5 s to 0.5 s. The MPC is warm-started using the shifting method but by shifting 5 steps. Moreover, the MPC considers a maximum velocity of 2 m/s and a maximum

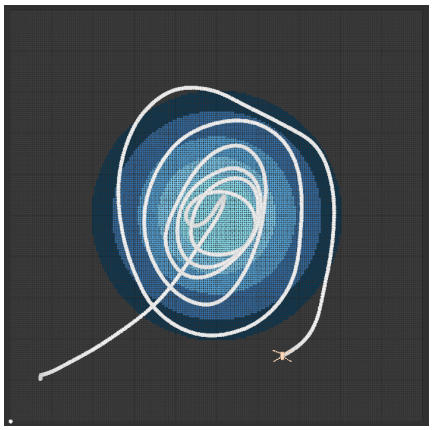
acceleration of 2 m/s^2 for the vehicle. Regarding the field trials, due to difficulties faced when attempting to execute the algorithm onboard, the experimental tests were carried out by instructing waypoints generated *a priori* under the same conditions.

2.6.7 Experiment 1

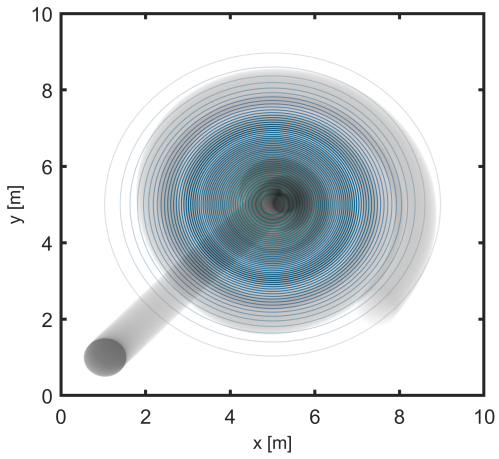
We begin by considering an example where the uncertainty map is composed of one radially-symmetric Gaussian component, with the corresponding results being displayed in Figure 2.23.



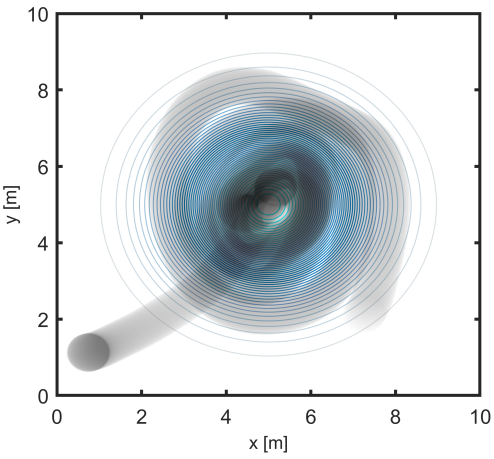
(a) Gazebo - Trajectory (RViz)



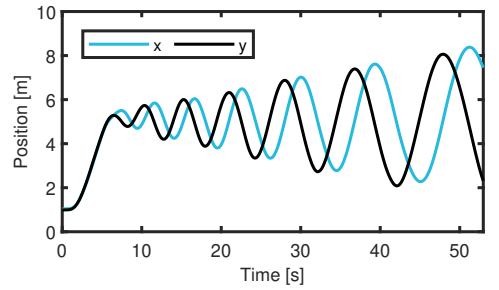
(b) Field trial - Trajectory (RViz)



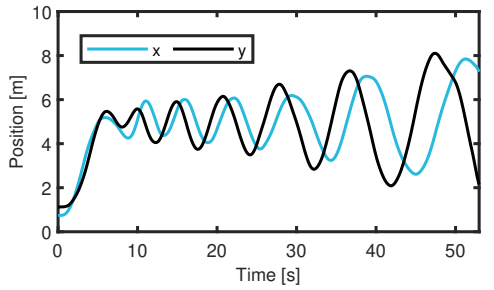
(c) Gazebo - Sensor footprint



(d) Field trial - Sensor footprint

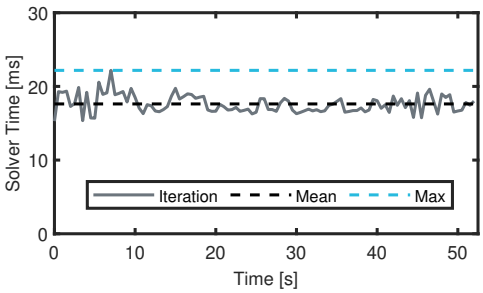


(e) Gazebo - Position

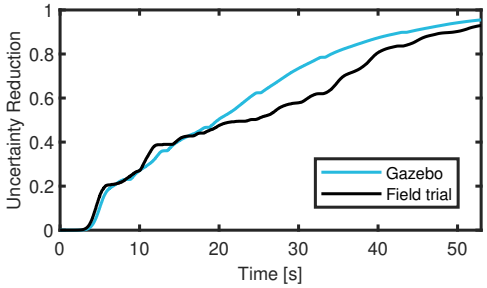


(f) Field trial - Position

Figure 2.23: Gazebo and field trial results for a simple uncertainty map.



(g) Gazebo - Solver times



(h) Uncertainty reduction

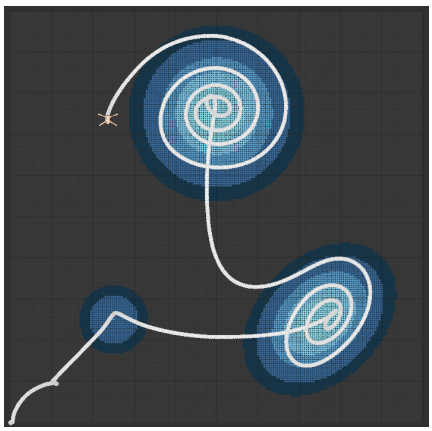
Figure 2.23: Gazebo and field trial results for a simple uncertainty map.

As illustrated in Figure 2.23 (a), the drone exhibits the expected behavior in the Gazebo simulation, executing a smooth spiral curve, as also reflected in the position profiles displayed in Figure 2.23 (e). Such behavior is obviously possible due to the appropriately tuned parameters of the MPC objective function and the selection of the limits for the acceleration and velocity of the vehicle. These choices allow the generation of sufficiently smooth trajectories that the PX4 controller can track efficiently. In addition, as depicted in Figure 2.23 (g), the computational times are also sufficiently fast to allow for a good performance, with each solver iteration taking approximately 18 ms on average.

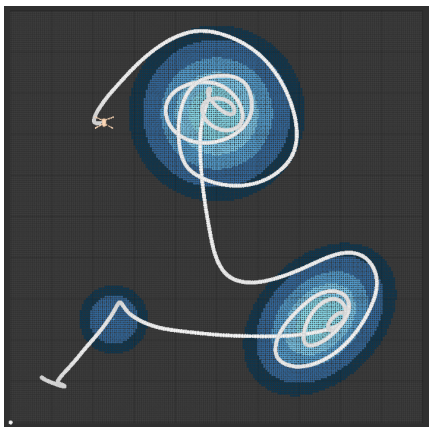
Concerning the field trial, as shown in Figure 2.23 (b), it is possible to observe that the resulting trajectory is not as consistent as the one from Gazebo. This discrepancy primarily arises from the influence of wind disturbances encountered in the outdoor experimental environment. In addition, there are also some inaccuracies associated with the Global Positioning System (GPS) of the drone. Nevertheless, for the designated observation radius, both trajectories show a similar coverage by the final time instant, as demonstrated in Figure 2.23 (h).

2.6.8 Experiment 2

In the second example, we consider an uncertainty map composed of three Gaussian components with different shapes. The Gazebo and field trial results are displayed in 2.24.

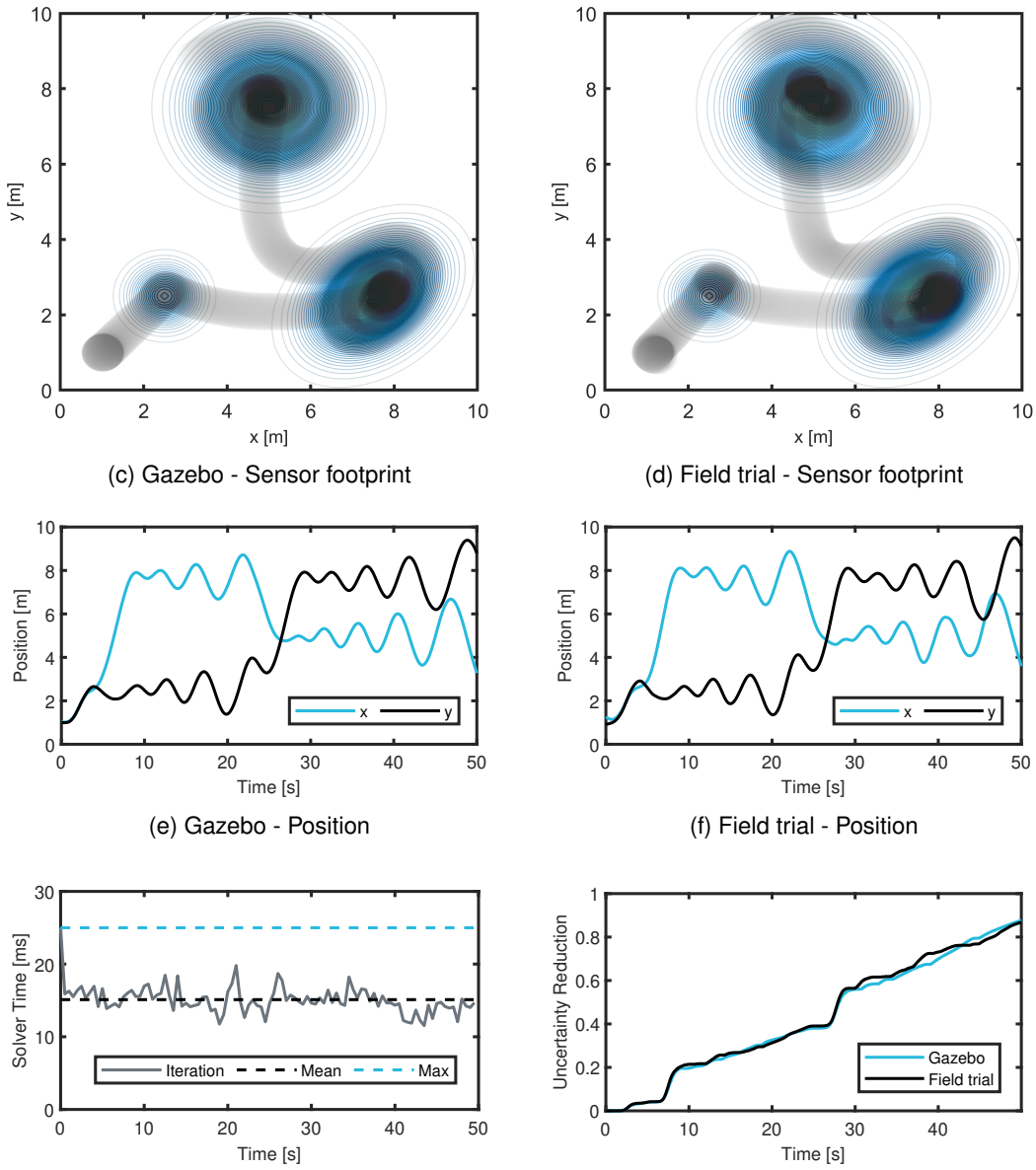


(a) Gazebo - Trajectory (RViz)



(b) Field trial - Trajectory (RViz)

Figure 2.24: Results for an uncertainty map composed of three Gaussian components.



(c) Gazebo - Sensor footprint

(d) Field trial - Sensor footprint

(e) Gazebo - Position

(f) Field trial - Position

(g) Gazebo - Solver times

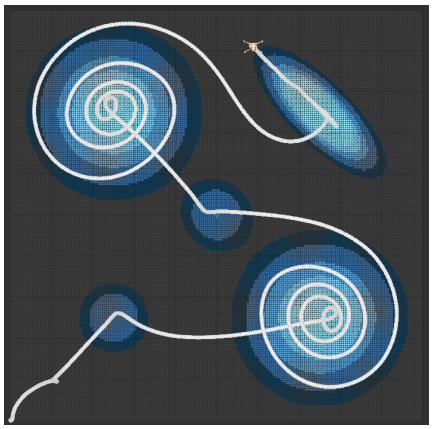
(h) Uncertainty reduction

Figure 2.24: Results for an uncertainty map composed of three Gaussian components.

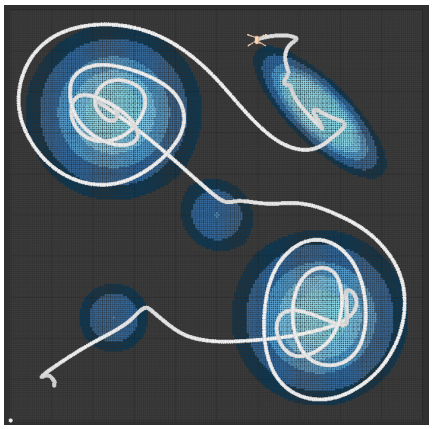
As shown in Figure 2.24 (a), in the Gazebo simulation, the drone executes the expected behavior, and the trajectory adapts to the characteristics of each Gaussian component. Concerning the field trials, besides the wind and the GPS errors, we also draw attention to the structural differences between the drones used in Gazebo and in the real trials, as well as differences in the tuning of the inner-loop controllers of the PX4 Autopilot. Despite that, in this example, both trajectories exhibit similar coverage profiles, as shown in Figure 2.23 (h), as well as in Figures 2.23 (c) and 2.23 (d).

2.6.9 Experiment 3

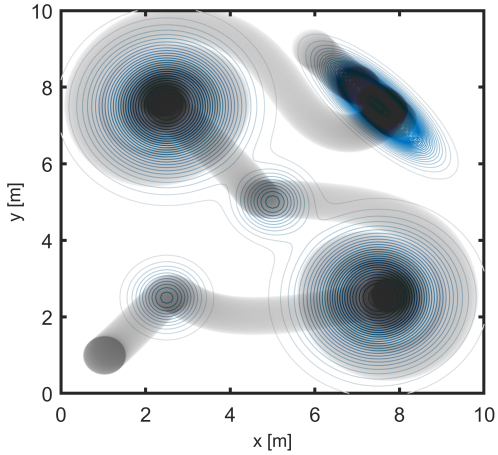
Ultimately, we present an example where the uncertainty map is composed of five Gaussian components. Figure 2.25 displays the results obtained in Gazebo and in the corresponding experimental test.



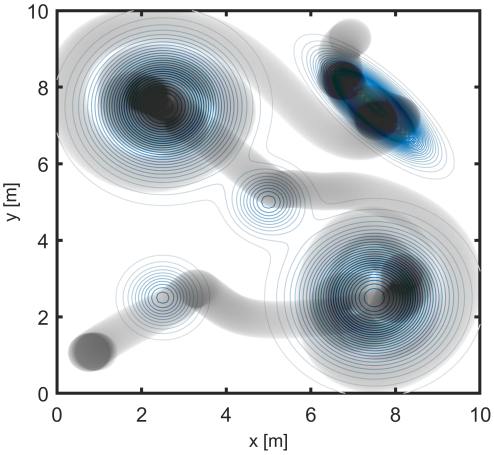
(a) Gazebo - Trajectory (RViz)



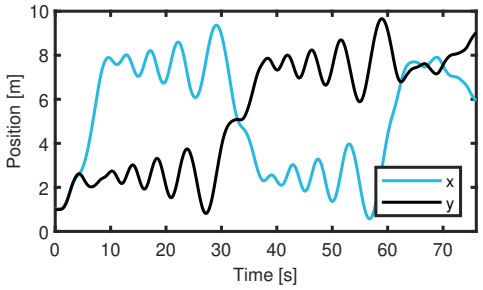
(b) Field trial - Trajectory (RViz)



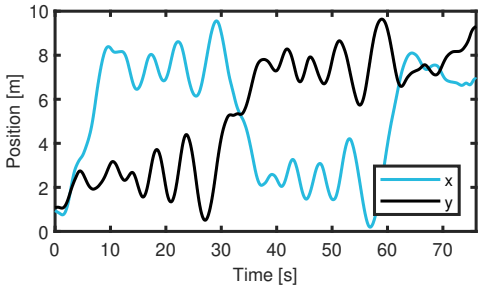
(c) Gazebo - Sensor footprint



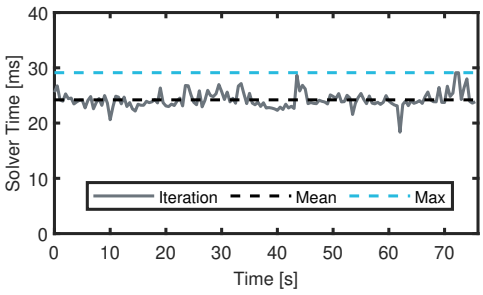
(d) Field trial - Sensor footprint



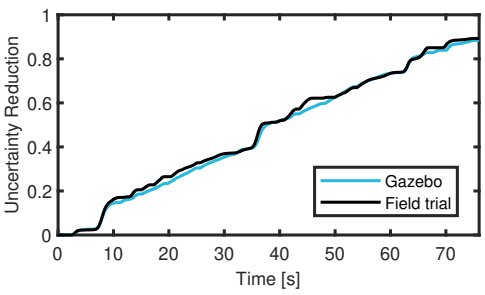
(e) Gazebo - Position



(f) Field trial - Position



(g) Gazebo - Solver times



(h) Uncertainty reduction

Figure 2.25: Results for an uncertainty map composed of five Gaussian components.

As illustrated in Figure 2.25 (a), the map comprises four circular Gaussian components. Two of these components have relatively small variances in comparison to the UAV's observation radius, while the other two exhibit higher variances. Additionally, there is a fifth component with an elliptical shape, and its variance along one of its axes is small when compared to the UAV's observation radius. In particular, as this scenario had not been previously introduced, we draw attention to the drone's behavior when analyzing the former component. In such a case, the drone follows a straight path along the major axis of the elliptical component, as depicted in Figure 2.25 (a). Additionally, as shown in Figure 2.25 (g), each solver iteration takes approximately 25 ms in average. This duration is slightly longer than the previous examples due to the map having more components and the mission having a slightly extended duration. Ultimately, it can be observed that the outcomes of the field trial display a comparable pattern to the results obtained from the simulation.

2.7 Summary

This chapter addressed the generation of optimal trajectories for autonomous wildfire prevention using a UAV. The main goal was designing a trajectory planning algorithm based on a map characterizing the uncertainty of fire presence in a given region. We began by establishing the mathematical definition of the uncertainty map, described as a weighted sum of Gaussian components. Then, we outlined the assumptions regarding the sensing capabilities of the UAV. In this context, the vehicle was assumed to fly at a constant altitude, equipped with a gimbal sensor with a limited FOV, and capable of perfectly analyzing a circular region around the UAV's horizontal position. Considering these assumptions, the problem was formulated from an optimal control standpoint, with the objective of maximizing the uncertainty volume covered by the vehicle during the surveillance mission.

Due to the complexity of the original problem formulation, we proposed an alternative approach based on discrete-time MPC. To promote the map exploration and to prevent the UAV from revisiting previously covered regions, the proposed algorithm penalizes intersections between the circular observation areas along the trajectory of the UAV. Given the complexity of precisely calculating the overlap area between two circles, we designed an exponential surrogate function to penalize such intersections.

Subsequently, we described the control architecture used to implement the algorithm in a quadrotor UAV. We considered a dual-layer structure of motion control, with the MPC algorithm serving as a high-level trajectory planner and an inner-loop controller responsible for tracking references generated by the MPC. For the purpose of efficiency, at the planning level, the UAV was modeled as a point-mass system following double-integrator dynamics. The motion control scheme was implemented using a PX4 Autopilot, providing both the inner-loop controller and an EKF to process sensor measurements.

The algorithm was initially tested in MATLAB, assuming ideal double-integrator dynamics for the UAV. Different examples were provided to showcase the trajectories that the algorithm is able to produce, and we conducted an analysis to assess the impact of some parameters on the algorithm's performance. Subsequently, the algorithm was assessed through higher-fidelity simulations in a Gazebo environment and by conducting actual experiments in an outdoor setting.

Chapter 3

Searching and Tracking a Stochastic Moving Target

In this chapter, we consider the problem of searching and tracking a stochastically moving, nonadversarial ground target using an autonomous aerial vehicle with a limited FOV sensor. The target is assumed to be modeled by a discrete-time stochastic system with an affine disturbance. The objective is to design a motion planning algorithm that enables the UAV to search and track the target. Once more, we propose an algorithm based on MPC by considering an approximation based on Gaussian mixture distributions. The efficacy of the proposed algorithm is demonstrated through simple illustrative examples.

3.1 Introduction

3.1.1 Motivation

While the primary motivation behind this dissertation was originally tied to wildfire prevention, the problem considered in the preceding chapter shares a strong connection with search-and-rescue and pursuit-evasion scenarios. In fact, the previous problem could be viewed as a search mission to locate a stationary ground target, where the goal would be maximizing the probability of successfully finding the target during the surveillance mission. Taking this motivation into consideration, we now delve into a more complex scenario where a ground target is moving randomly, and the goal is to search and keep track of the target using an autonomous aerial vehicle featuring a limited FOV sensor.

3.1.2 Organization

This chapter is organized as follows. Section 3.2 describes mathematically the problem addressed in this chapter. In Section 3.3, we perform a review on recursive Bayesian filtering, focusing on probabilistic systems where the model PDFs are described by Gaussian mixture distributions. In Section 3.4, we develop a motion planning algorithm based on MPC using recursive Bayesian filtering and an approximation based on Gaussian mixtures. Section 3.5 demonstrates the efficacy of the proposed strategy through illustrative examples, and conclusions are summarized in the final section.

3.2 Problem Description

In this chapter, we study the problem of finding and tracking a randomly moving ground target using a UAV equipped with sensor with a limited sensing range. The target is assumed to be nonadversarial, i.e., the movement of the target is assumed to be independent of the vehicle's movement. Moreover, we assume that the target (evader) is modeled by a stochastic discrete-time system of the form

$$\mathbf{x}_{k+1}^e = \mathbf{f}_e(\mathbf{x}_k^e) + \mathbf{w}_k, \tag{3.1}$$

with state $\mathbf{x}_k^e \in \mathbb{R}^{n_{x_e}}$ and disturbance $\mathbf{w}_k \in \mathbb{R}^{n_{x_e}}$. The disturbance vector $\mathbf{w}_k$ is assumed to be a random vector with a known density function $p(\mathbf{w}_k)$ at each discrete-time instant k. The disturbance process $\{\mathbf{w}_k : k \geq 0\}$ is assumed to be a random process with an independent distribution at each time step. Additionally, we also assume that the initial state of the target $\mathbf{x}_0^e$ is a random vector described by a known density function $p(\mathbf{x}_0^e)$. Furthermore, we point out that, to keep the notation short and clear, we use the abuse of notation $p(\mathbf{z})$ to denote the probability distribution associated with a random vector $\mathbf{z}$, and we make no distinction between random vectors and realizations of random vectors.

The assumptions regarding the UAV dynamics and sensing model considered in this chapter are similar to the assumptions considered in the previous chapter. The UAV (pursuer) is modeled by a generic deterministic discrete-time system

$$\mathbf{x}_{k+1}^p = \mathbf{f}_p(\mathbf{x}_k^p, \mathbf{u}_k^p) \tag{3.2}$$

with state $\mathbf{x}_k^p \in \mathbb{R}^{n_{x_p}}$ and control $\mathbf{u}_k^p \in \mathbb{R}^{n_{u_p}}$, which may be obtained from the continuous-time model using the previously described discretization methods. Once more, we assume that the vehicle flies at a constant altitude and is equipped with a gimbal camera, the line-of-sight of which always aims straight down. Consequently, at each discrete-time instant k, we assume that the drone analyzes a given area, $\mathcal{B}_r(\boldsymbol{\varphi}_k^p)$, defined as a circle centered in the UAV's horizontal position, $\boldsymbol{\varphi}_k^p$, and with radius r, i.e.,

$$\mathcal{B}_r(\boldsymbol{\varphi}_k^p) \triangleq \left\{\mathbf{p} \in \mathbb{R}^2 : \|\mathbf{p} - \boldsymbol{\varphi}_k^p\| < r\right\}. \tag{3.3}$$

The onboard camera is assumed to have high accuracy and is always capable of detecting the location of the target when the target is in the sensor's FOV. In that case, the camera generates a measurement $\mathbf{y}_k$ of the target's position corrupted with additive noise,

$$\mathbf{y}_k = \boldsymbol{\varphi}_k^e + \mathbf{v}_k = \mathbf{C}_e \mathbf{x}_k^e + \mathbf{v}_k, \tag{3.4}$$

where $\mathbf{v}_k$ is assumed to be Gaussian noise with a known covariance. On the other hand, if the target is not present in the vehicle's FOV, then the sensor does not generate any measurement. Taking this assumptions into account, the goal of this chapter is to develop a motion planning algorithm for the UAV to search and track the target while taking into account the information obtained from the sensor.

3.3 Recursive Bayesian Filtering

In this section, we review fundamental concepts related to recursive Bayesian filtering, focusing on the theoretical results available for probabilistic state-space systems described by Gaussian mixture distributions. The major purpose of the discussion performed in this section is to provide valuable insight into the development of an algorithm to address the problem described in Section 3.2.

3.3.1 Filtering Problem

Recursive Bayesian filtering is a general probabilistic approach to iteratively estimate the PDF of a system's state over time using incoming measurements and a mathematical model of the system. Given the widespread application of state estimation across various fields of science and engineering, this problem has been the subject of substantial research focus over the past decades.

A general probabilistic state-space model is expressed via an initial state distribution $p(\mathbf{x}_0)$, a process model defined in terms of the state transition probability distribution $p(\mathbf{x}_{k+1}|\mathbf{x}_k)$, and a sensor model specified by the measurement likelihood $p(\mathbf{y}_k|\mathbf{x}_k)$. Consequently, a general probabilistic state-space model is usually expressed in the following form

$$\begin{aligned} \mathbf{x}_0 &\sim p(\mathbf{x}_0), \\ \mathbf{x}_k &\sim p(\mathbf{x}_{k+1}|\mathbf{x}_k), \\ \mathbf{y}_k &\sim p(\mathbf{y}_k|\mathbf{x}_k), \end{aligned} \tag{3.5}$$

with state $\mathbf{x}_k \in \mathbb{R}^{n_x}$ and output $\mathbf{y}_k \in \mathbb{R}^{n_y}$. For a given probabilistic state-space model and a sequence of sensor measurements up to time instant k,

$$\mathbf{Y}_k \triangleq \begin{bmatrix} \mathbf{y}_0 & \mathbf{y}_1 & \dots & \mathbf{y}_{k-1} & \mathbf{y}_k \end{bmatrix}, \tag{3.6}$$

the Bayesian filtering problem can be solved recursively over time through the widely-known update and prediction equations. The update (filtering) step is given by

$$p(\mathbf{x}_k|\mathbf{Y}_k) = \frac{p(\mathbf{y}_k|\mathbf{x}_k)\, p(\mathbf{x}_k|\mathbf{Y}_{k-1})}{p(\mathbf{y}_k|\mathbf{Y}_{k-1})}, \tag{3.7}$$

and the prediction step is performed as

$$p(\mathbf{x}_{k+1}|\mathbf{Y}_k) = \int_{\mathbb{R}^{n_x}} p(\mathbf{x}_{k+1}|\mathbf{x}_k)\, p(\mathbf{x}_k|\mathbf{Y}_k)\, d\mathbf{x}_k. \tag{3.8}$$

Moreover, the denominator in (3.7) is given by

$$p(\mathbf{y}_k|\mathbf{Y}_{k-1}) = \int_{\mathbb{R}^{n_x}} p(\mathbf{y}_k|\mathbf{x}_k)\, p(\mathbf{x}_k|\mathbf{Y}_{k-1})\, d\mathbf{x}_k, \tag{3.9}$$

which is a normalization factor that is constant relative to $\mathbf{x}_k$, and, therefore, may be ignored in practice.

Solving the equations in (3.7) and (3.8) has attracted considerable attention for many decades, and this general approach has been successfully used in various fields of science and engineering. It is well known that if the model in (3.5) is linear with Gaussian additive noise, then (3.7) and (3.8) simplify to the conventional Kalman filtering equations [46]. In more general cases, the update and prediction equations have no closed-form solutions, and this has led to significant research efforts. The research activity has resulted in many approximation methods to the state-estimation problem, including the employment of the EKF [47], the Unscented Kalman Filter (UKF) [48], and the Particle Filter (PF) [49]. Each of these methods delves into distinct structural aspects of the state-space model, and each has its known advantages and corresponding limitations.

3.3.2 Gaussian Mixture Models

In this section, we limit our attention to a class of state-space systems that are not quite as general as (3.5) but may approximate generic stochastic systems arbitrarily well. We consider state-space systems where the prior, the process model, and the measurement likelihood are described by Gaussian mixture distributions. More specifically, the initial state distribution is given by

$$p(\mathbf{x}_0) = \sum_{l=1}^{M_0} \alpha_l \, \mathcal{N}(\mathbf{x}_0; \bar{\mathbf{x}}_0^l, \mathbf{P}_0^l), \tag{3.10}$$

the process model is of the form

$$p(\mathbf{x}_{k+1}|\mathbf{x}_k) = \sum_{m=1}^{M_x} \beta_k^m \, \mathcal{N}(\mathbf{x}_{k+1}; \mathbf{A}_k^m \mathbf{x}_k + \bar{\mathbf{w}}_k^m, \mathbf{Q}_k^m), \tag{3.11}$$

and the measurement model is given by

$$p(\mathbf{y}_k|\mathbf{x}_k) = \sum_{n=1}^{M_y} \gamma_k^n \, \mathcal{N}(\mathbf{y}_k; \mathbf{C}_k^n \mathbf{x}_k + \bar{\mathbf{v}}_k^n, \mathbf{R}_k^n). \tag{3.12}$$

In the above expressions, the form $\mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ is used to denote a standard multivariate Gaussian distribution, which is defined by

$$\mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) \triangleq \frac{1}{\sqrt{(2\pi)^d |\boldsymbol{\Sigma}|}} \exp\left\{ -\frac{1}{2} (\mathbf{z} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{z} - \boldsymbol{\mu}) \right\}, \tag{3.13}$$

where d corresponds to the dimension of the input vector $\mathbf{z}$. The parameters $\boldsymbol{\mu} \in \mathbb{R}^d$ and $\boldsymbol{\Sigma} \in \mathbb{R}^{d \times d}$ are, respectively, the mean vector and the covariance matrix of the Gaussian distribution. Moreover, the covariance matrix $\boldsymbol{\Sigma}$ is symmetric and positive definite. The mixture weights α_l, β_k^m, $\gamma_k^n > 0$ verify $\sum_{l=1}^{M_0} \alpha_l = 1$, $\sum_{m=1}^{M_x} \beta_m = 1$, $\sum_{n=1}^{M_y} \gamma_n = 1$. Additionally, we point out that the mean offset terms $\bar{\mathbf{w}}_k^m$ and $\bar{\mathbf{v}}_k^n$ allow for the inclusion of input signals or other generated signals that do not depend on the state. The primary advantage of restricting attention to this class of stochastic state-space models is that the prediction and update equations can be expressed in closed form.

In consequence, let us start with with the measurement update step (3.7). Assuming that we have available a predicted Gaussian mixture given by

$$p(\mathbf{x}_k|\mathbf{Y}_{k-1}) = \sum_{l=1}^{M_{k|k-1}} w_{k|k-1}^l \mathcal{N}(\mathbf{x}_k; \bar{\mathbf{x}}_{k|k-1}^l, \mathbf{P}_{k|k-1}^l), \tag{3.14}$$

the measurement update (3.7) can be expressed as

$$p(\mathbf{x}_k|\mathbf{Y}_k) = \sum_{l=1}^{M_{k|k-1}} \sum_{n=1}^{M_y} w_{k|k-1}^l \gamma_k^n \frac{\mathcal{N}(\mathbf{y}_k; \mathbf{C}_k^n \mathbf{x}_k + \bar{\mathbf{v}}_k^n, \mathbf{R}_k^n)}{p(\mathbf{y}_k|\mathbf{Y}_{k-1})} \mathcal{N}(\mathbf{x}_k; \bar{\mathbf{x}}_{k|k-1}^l, \mathbf{P}_{k|k-1}^l). \tag{3.15}$$

Due to the linear Gaussian structure, it can be shown the updated PDF of the state is also a Gaussian mixture distribution (see e.g. [50]). In this case, the updated distribution of the state is given by

$$p(\mathbf{x}_k|\mathbf{Y}_k) = \sum_{s=1}^{M_{k|k}} w_{k|k}^s \mathcal{N}(\mathbf{x}_k; \bar{\mathbf{x}}_{k|k}^s, \mathbf{P}_{k|k}^s), \tag{3.16}$$

where for each $l = 1, \ldots, M_{k|k-1}$ and $n = 1, \ldots, M_y$ it holds that

$$\begin{aligned}
M_{k|k} &= M_{k|k-1} M_y, \\
s &\triangleq M_y(l-1) + n, \\
\bar{\mathbf{x}}_{k|k}^s &= \bar{\mathbf{x}}_{k|k-1}^l + \mathbf{K}_k^s \mathbf{e}_k^s, \\
\mathbf{e}_k^s &= \mathbf{y}_k - \mathbf{C}_k^n \bar{\mathbf{x}}_{k|k-1}^l - \bar{\mathbf{v}}_k^n, \\
\mathbf{S}_k^s &= \mathbf{C}_k^n \mathbf{P}_{k|k-1}^l (\mathbf{C}_k^n)^\top + \mathbf{R}_k^n, \\
\mathbf{K}_k^s &= \mathbf{P}_{k|k-1}^l (\mathbf{C}_k^n)^\top (\mathbf{S}_k^s)^{-1}, \\
\mathbf{P}_{k|k}^s &= \mathbf{P}_{k|k-1}^l - \mathbf{K}_k^s \mathbf{S}_k^s (\mathbf{K}_k^s)^\top,
\end{aligned} \tag{3.17}$$

and the updated mixture weights are computed as

$$\begin{aligned}
\bar{w}_{k|k}^s &= \frac{w_{k|k-1}^l \gamma_k^n \exp\left\{-\frac{1}{2}(\mathbf{e}_k^s)^\top (\mathbf{S}_k^s)^{-1} \mathbf{e}_k^s\right\}}{\sqrt{(2\pi)^{n_x} |\mathbf{S}_k^s|}}, \\
w_{k|k}^s &= \frac{\bar{w}_{k|k}^s}{\sum_{s=1}^{M_{k|k}} \bar{w}_{k|k}^s}.
\end{aligned} \tag{3.18}$$

The above equations basically correspond to the standard Kalman filtering equations applied to all possible pairs of Gaussian components associated with the prior state PDF and the measurement likelihood density. A drawback of this class of models, which is already evident, is the growth of the number of Gaussian components. This issue is discussed later in this section.

Given that the updated distribution has been computed, we can proceed to the prediction step (3.8). In this context, the predicted state distribution can be expressed as

$$p(\mathbf{x}_{k+1}|\mathbf{Y}_k) = \sum_{s=1}^{M_{k|k}} \sum_{m=1}^{M_x} w_{k|k}^s \beta_k^m \int_{\mathbb{R}^{n_x}} \mathcal{N}(\mathbf{x}_{k+1}; \mathbf{A}_k^m \mathbf{x}_k + \bar{\mathbf{w}}_k^m, \mathbf{Q}_k^m)\, \mathcal{N}(\mathbf{x}_k; \bar{\mathbf{x}}_{k|k}^s, \mathbf{P}_{k|k}^s)\, d\mathbf{x}_k. \tag{3.19}$$

Once more, due to the linear nature of the model and Gaussian densities involved, the predicted state PDF is also a Gaussian mixture distribution. In this case, the predicted state distribution is given by

$$p(\mathbf{x}_{k+1}|\mathbf{Y}_k) = \sum_{l=1}^{M_{k+1|k}} w_{k+1|k}^l \mathcal{N}(\mathbf{x}_{k+1}; \bar{\mathbf{x}}_{k+1|k}^l, \mathbf{P}_{k+1|k}^l), \tag{3.20}$$

where for each $s = 1, \ldots, M_{k|k}$ and $m = 1, \ldots, M_x$ it holds that

$$\begin{aligned} M_{k+1|k} &= M_{k|k} M_x, \\ l &\triangleq M_x(s-1) + m, \\ \bar{\mathbf{x}}_{k+1|k}^l &= \mathbf{A}_k^m \bar{\mathbf{x}}_{k|k}^s + \bar{\mathbf{w}}_k^m, \\ \mathbf{P}_{k+1|k}^l &= \mathbf{A}_k^m \mathbf{P}_{k|k}^s (\mathbf{A}_k^m)^\top + \mathbf{Q}_k^m, \\ w_{k+1|k}^l &= w_{k|k}^s \beta_k^m. \end{aligned} \tag{3.21}$$

We have therefore reached the initial starting assumption in (3.14). Hence, the previous computations can be repeated recursively. Now it remains to address how it is possible to deal with exponential growth in the number of Gaussian components.

3.3.3 Gaussian Mixture Reduction

A significant issue with probabilistic state-space systems described by Gaussian mixture distributions, which can be observed by applying the prediction and update equations for just a few steps, is the exponential growth in the number of Gaussian components. Just after the first update and prediction steps, the predicted state distribution is already a Gaussian mixture with $M_0 M_x M_y$ components. This becomes intractable whenever any two of the set of numbers $\{M_0, M_x, M_y\}$ are greater than one. Consequently, in order to avoid the growth in computational load and make these probabilistic models applicable in practice, a method to reduce the number of Gaussian components is necessary.

The application of Gaussian mixture reduction techniques extends to various domains within science and engineering, and it is naturally desirable to reduce the number of components with a minimal loss of fidelity. As a result, several approaches for the reduction problem have been proposed in the literature. For completeness, this subsection presents a brief overview of the most common methods, but for a more extensive survey the reader is referred to [51].

The simplest approach to the mixture reduction problem is obviously pruning. Pruning consists of eliminating Gaussian components that have low probabilities. More specifically, given a Gaussian mixture consisting of M components, pruning consists in discarding the $M - L$ components with the lowest associated weights, or the components whose weights fall below a given threshold, and subsequently renormalizing the weights of the remaining components. Nevertheless, despite its simplicity, it may not be the most efficient method. Indeed, more sophisticated techniques have been devised.

Alternatively, instead of pruning, Gaussian mixture reduction can be performed by merging components according to a similarity measure. The process comes from taking an expected value across the components that are to be merged, which preserves the moments of the overall mixture. For example, to merge components 1 to L, the equations for the merged result are given by

$$\begin{aligned} w_{\text{merged}} &= \sum_{i=1}^{L} w_i, \\ \boldsymbol{\mu}_{\text{merged}} &= \frac{1}{w_{\text{merged}}} \sum_{i=1}^{L} w_i \boldsymbol{\mu}_i, \\ \boldsymbol{\Sigma}_{\text{merged}} &= \sum_{i=1}^{L} \frac{w_i}{w_{\text{merged}}} \left(\boldsymbol{\Sigma}_i + (\boldsymbol{\mu}_i - \boldsymbol{\mu}_{\text{merged}})(\boldsymbol{\mu}_i - \boldsymbol{\mu}_{\text{merged}})^\top \right). \end{aligned} \tag{3.22}$$

Merging algorithms differ in how they choose the components to be merged. In this context, a prominent technique is the Runnals' algorithm [52], which iteratively joins two components by minimizing an upper bound on the increase in the Kullback-Leibler divergence between the original and the reduced mixtures. However, one drawback of the algorithm is that it requires a desired maximum number of components as input. Nevertheless, the recent work in [53] extends the algorithm to perform Gaussian mixture reduction based on a user-defined threshold. More precisely, the algorithm is extended to produce a mixture by successive merging until a threshold is exceeded, even if the number of components is already below the desired maximum number of components. The algorithm proposed in [53] is detailed below.

Algorithm 1 Kullback-Leibler Gaussian Mixture Reduction

1: **Inputs**: Integers M_{min} and M_{max} that determine a minimum and maximum number of components in the reduced mixture, a threshold $\lambda > 0$, and the initial mixture with M components.

2: The cost of merging components i and j is the upper bound on the increase in the KL divergence,

$$c_{ij} = \frac{1}{2}((w_i + w_j) \log |\boldsymbol{\Sigma}_{ij}| - w_i \log |\boldsymbol{\Sigma}_i| - w_j \log |\boldsymbol{\Sigma}_j|), \tag{3.23}$$

where $\boldsymbol{\Sigma}_{ij}$ corresponds to (3.22) for merging components i and j. Compute the costs c_{ij} noting that $c_{ii} = 0$ and $c_{ij} = c_{ji}$, so it is only necessary to compute $\frac{1}{2}M(M-1)$ combinations.

3: **While** $M > M_{\text{max}}$ or ($M > M_{\text{min}}$ and $\min\limits_{i,j} c_{ij} < \lambda$) **do**

4: Merge the components of the current mixture having the lowest c_{ij} using (3.22) and set the current mixture to the result. Set $M \leftarrow M - 1$.

5: Update the costs c_{ij}.

6: **End**

3.4 Proposed Solution

In this section, we leverage the previous concepts concerning recursive Bayesian filtering for stochastic state-space systems described by Gaussian mixture distributions, and we apply them to the design of an algorithm to address the problem considered in Section 3.2. Firstly, we relate the previous discussion with the problem at hand and discuss how Gaussian mixtures can be used to approximate the update and prediction steps. The section culminates with the formulation of an algorithm based on MPC.

3.4.1 Target Model

To address the problem described in Section 3.2, we consider that the initial PDF of the target's state can be arbitrarily well approximated by a Gaussian mixture distribution,

$$p(\mathbf{x}_0^e) = \sum_{l=1}^{M_0} \alpha_l \, \mathcal{N}(\mathbf{x}_0^e; \bar{\mathbf{x}}_0^{e,l}, \mathbf{P}_0^{e,l}). \tag{3.24}$$

The density function of the disturbance vector $\mathbf{w}_k$ is also assumed to be a Gaussian mixture distribution at each discrete-time instant k,

$$p(\mathbf{w}_k) = \sum_{m=1}^{M_x} \beta_k^m \, \mathcal{N}(\mathbf{w}_k; \bar{\mathbf{w}}_k^m, \mathbf{Q}_k^m). \tag{3.25}$$

Consequently, if the target model is linear,

$$\mathbf{x}_{k+1}^e = \mathbf{A}_e \mathbf{x}_k^e + \mathbf{w}_k, \tag{3.26}$$

the target's state transition probability distribution is a Gaussian mixture given by

$$p(\mathbf{x}_{k+1}^e | \mathbf{x}_k^e) = \sum_{m=1}^{M_x} \beta_k^m \, \mathcal{N}(\mathbf{x}_{k+1}^e; \mathbf{A}_e \mathbf{x}_k^e + \bar{\mathbf{w}}_k^m, \mathbf{Q}_k^m). \tag{3.27}$$

Therefore, in this case, the prediction step can be computed by directly applying the equations described in (3.21). However, if the target model is nonlinear, the equations in (3.21) cannot be directly applied. In such a case, the nonlinearity might be dealt with by taking the Laplace approximation about the mean of each component [50],

$$\mathbf{A}_k^{e,s} = \nabla \mathbf{f}(\bar{\mathbf{x}}_{k|k}^{e,s}), \tag{3.28}$$

and the predicted component means are now given by

$$\bar{\mathbf{x}}_{k+1|k}^{e,l} = \mathbf{f}(\bar{\mathbf{x}}_{k|k}^{e,s}) + \bar{\mathbf{w}}_k^m. \tag{3.29}$$

Nevertheless, for simplicity, we will consider that the model of the target is linear.

3.4.2 Measurement Model

Concerning the measurement model of the UAV, there are two possible scenarios to consider. The target may or may not be present in the visibility region of the UAV, which is determined by whether the condition $\|\boldsymbol{\varphi}_k^e - \boldsymbol{\varphi}_k^p\| < r$ is verified or not, respectively. Since the observations are performed in the position space, the measurement likelihood can be seen as a function solely dependent on the target's position, $p(\mathbf{y}_k|\boldsymbol{\varphi}_k^e)$, and can then be converted to a function of the state by $\boldsymbol{\varphi}_k^e = \mathbf{C}_e\mathbf{x}_k^e$. In this case, the updated marginal distribution of the target's position, $p(\boldsymbol{\varphi}_k^e|\mathbf{Y}_k)$, depends only on the predicted marginal distribution of the target's position, $p(\boldsymbol{\varphi}_k^e|\mathbf{Y}_{k-1})$, and is given by

$$p(\boldsymbol{\varphi}_k^e|\mathbf{Y}_k) = \frac{p(\mathbf{y}_k|\boldsymbol{\varphi}_k^e)\, p(\boldsymbol{\varphi}_k^e|\mathbf{Y}_{k-1})}{p(\mathbf{y}_k|\mathbf{Y}_{k-1})}, \tag{3.30}$$

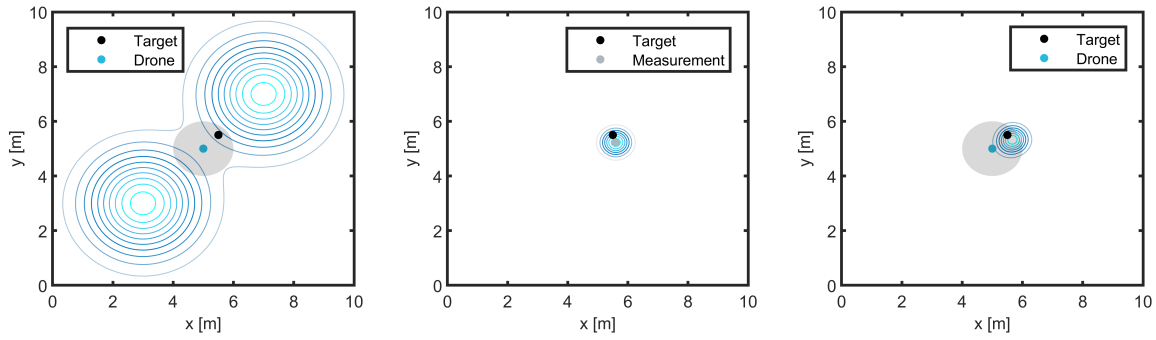
which follows from performing an integration of (3.7) over the remaining components of the state. Additionally, we remark that, for a given Gaussian mixture PDF of the state with means $\boldsymbol{\mu}_i$ and covariances $\boldsymbol{\Sigma}_i$, the marginal distribution of the position is a Gaussian mixture obtained by extracting the corresponding submatrices through the transformations $\mathbf{C}_e\boldsymbol{\mu}_i$ and $\mathbf{C}_e\boldsymbol{\Sigma}_i\mathbf{C}_e^\top$.

The simplest scenario occurs when the UAV detects the target. In such a case, the drone's sensor generates a measurement $\mathbf{y}_k \in \mathbb{R}^2$ of the target's position corrupted with additive Gaussian noise $\mathbf{v}_k$ with a known covariance $\mathbf{R}_\mathbf{v} \in \mathbb{R}^{2\times 2}$. This corresponds to a measurement likelihood density given by

$$p(\mathbf{y}_k|\boldsymbol{\varphi}_k^e) = \mathcal{N}(\mathbf{y}_k; \boldsymbol{\varphi}_k^e, \mathbf{R}_\mathbf{v}), \tag{3.31}$$

and, therefore, the equations in (3.17) can be directly applied to update the target's state distribution. The only difference between this scenario and a standard Kalman filter is that the prior/predicted state distribution may be a Gaussian mixture instead of a single Gaussian component.

Figure 3.1 illustrates the update of the target's position distribution for a scenario in which the target is detected. In this case, the marginal distribution of the target's position is initially composed of two larger Gaussian components. However, after the update, it converges into a single Gaussian component, closely resembling the measurement likelihood. In fact, there are still two components, but one of them has a negligible weight and could be eliminated after a reduction step.



(a) Prior/Predicted (b) Measurement (c) Updated

Figure 3.1: Update of the marginal distribution of the position for a target detection.

The nontrivial scenario occurs when the target is not in the vehicle's FOV. In such a case, the sensor does not generate an actual measurement. However, since the target is outside the vehicle's FOV, meaning that $\|\boldsymbol{\varphi}_k^e - \boldsymbol{\varphi}_k^p\| > r$, the measurement model would ideally be given by some function of $\boldsymbol{\varphi}_k^e$ that is zero inside the observation radius of the UAV and a positive constant otherwise. More specifically, the sensor model would ideally be given by

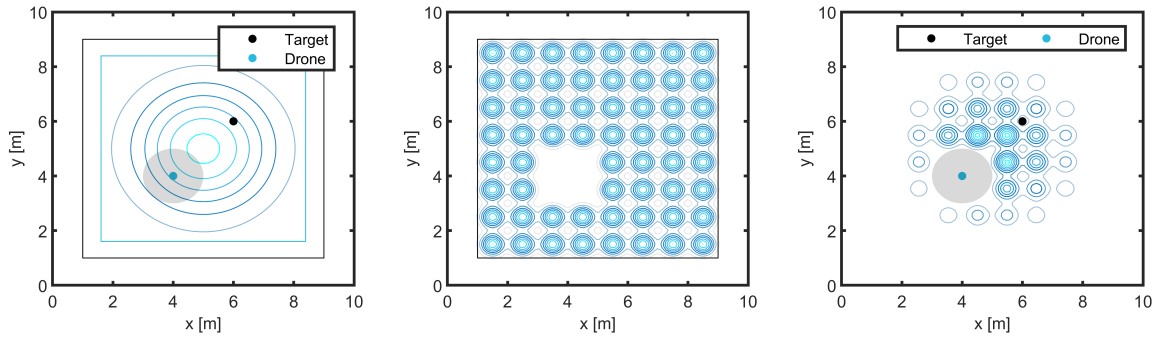
$$p(\mathbf{y}_k|\boldsymbol{\varphi}_k^e) = \begin{cases} c, \text{ if } \|\boldsymbol{\varphi}_k^e - \boldsymbol{\varphi}_k^p\| > r \\ 0, \text{ otherwise} \end{cases}, \tag{3.32}$$

where c is some positive constant. However, it is important to highlight that, in the interest of maintaining consistent notation, we also employ the notation $p(\mathbf{y}_k|\boldsymbol{\varphi}_k^e)$ for the sensor model. Nonetheless, in this instance, it can be considered a misuse of notation, given that there is no actual measurement involved, and the function described in (3.32) does not have the characteristics of a density function.

In order to perform the update step in closed form using the equations described in (3.17), the sensor model in (3.32) has to be approximated using a Gaussian mixture distribution. Consequently, to carry out the approximation, we consider a measurement model given by

$$p(\mathbf{y}_k|\boldsymbol{\varphi}_k^e) = \sum_{n=1}^{M_y} \gamma_k \, \mathcal{N}(\mathbf{y}_k; \boldsymbol{\varphi}_k^e + \bar{\mathbf{v}}_k^n, \mathbf{R}), \tag{3.33}$$

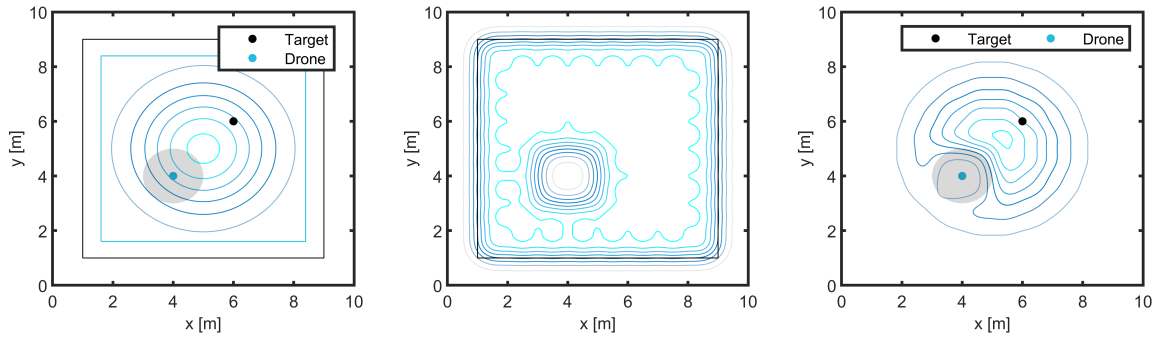
where the Gaussian components, as a function of $\boldsymbol{\varphi}_k^e$, are arranged in a grid over a rectangular confidence region that bounds a high probability of the target's position marginal distribution. The grid cells have a smaller side length than the observation diameter of the UAV, and a radially-symmetric Gaussian component is placed at the center of each cell, except for the cell positions that are inside the visibility region of the UAV, where no Gaussian component is placed. When there are no cell positions within the vehicle's visibility region, the update step is simply bypassed. This can be accomplished by appropriately choosing a pseudo-measurement $\mathbf{y}_k$ and the vectors $\bar{\mathbf{v}}_k^n$. For instance, a simple way is to set $\mathbf{y}_k = \mathbf{0}_2$ and then the vectors $\bar{\mathbf{v}}_k^n$ correspond to the symmetrical positions of the desired cell positions. Figure 3.2 illustrates the update of target's position distribution for a case where the target is outside the visibility region of the UAV.



(a) Prior/Predicted (b) Measurement (c) Updated

Figure 3.2: Update of the marginal distribution of the position when the target is not detected.

In the example shown in Figure 3.2 (a), the blue rectangle is a confidence region with a probability of at least 95% and is obtained as detailed in Section 3.4.3. The measurement model is defined based on the black rectangle, obtained by expanding the blue one to encompass an integer number of cell positions. In the example of Figure 3.2 (b), each grid cell has a side length equal to the observation radius of the UAV, and the covariance matrix of each Gaussian component was chosen to emphasize how the approximation is carried out. However, we can construct a better approximation by choosing a higher covariance for each Gaussian component of the measurement model, as depicted in Figure 3.3.



(a) Prior/Predicted (b) Measurement (c) Updated

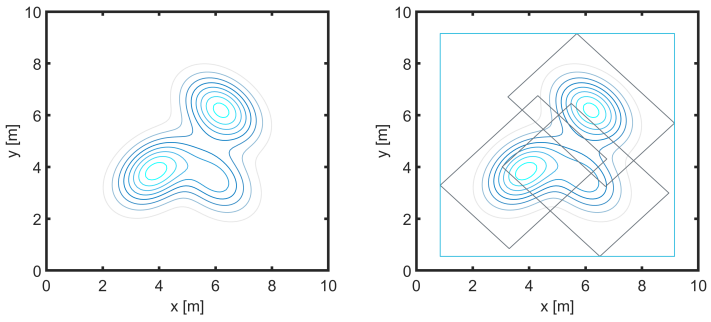
Figure 3.3: Update of the marginal distribution of the position when the target is not detected.

3.4.3 Confidence Rectangle

Consider a standard multivariate Gaussian distribution $\mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$, where the input vector has dimension d. The ellipsoid defined by

$$\mathcal{E}_\epsilon(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \triangleq \left\{ \mathbf{z} \in \mathbb{R}^d : (\mathbf{z} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{z} - \boldsymbol{\mu}) \leq \chi_d^2(\epsilon) \right\}, \tag{3.34}$$

is a confidence ellipsoid of probability $1 - \epsilon$, where $\chi_d^2(\epsilon)$ is the upper (100ϵ)% percentile of a chi-square distribution with d degrees of freedom. Consequently, to obtain the rectangular confidence region, we determine the ellipsoid of probability $1 - \epsilon$ for each component, and the corresponding enclosing rectangle that aligns with the axis of the ellipsoid. The extreme x and y coordinates of the overall confidence rectangle are then obtained through the vertices of each enclosing rectangle, as illustrated in Figure 3.4.



(a) Before (b) After

Figure 3.4: Demonstration of the process for obtaining the rectangular confidence region.

3.4.4 Model-Predictive Approach

Once again, in order to tackle the problem considered in this chapter, we adopt an MPC-based approach. At every discrete-time instant k, for a given initial state $\mathbf{x}_k^p$ of the drone (pursuer), the control policy is defined by solving the following optimal control problem

$$\begin{aligned}
\underset{\hat{\mathbf{X}}_k^p, \hat{\mathbf{U}}_k^p}{\text{maximize}} \quad & J_k(\hat{\mathbf{X}}_k^p, \hat{\mathbf{U}}_k^p) \\
\text{subject to} \quad & \hat{\mathbf{x}}_{k,0}^p = \mathbf{x}_k^p, \\
& \hat{\mathbf{x}}_{k,j+1}^p = \mathbf{f}_p(\hat{\mathbf{x}}_{k,j}^p, \hat{\mathbf{u}}_{k,j}^p),\ j = 0, \dots, N-1, \\
& \hat{\mathbf{x}}_{k,j}^p \in \mathcal{X}_p,\ j = 0, \dots, N, \\
& \hat{\mathbf{u}}_{k,j}^p \in \mathcal{U}_p,\ j = 0, \dots, N-1,
\end{aligned} \tag{3.35}$$

where the matrices $\hat{\mathbf{X}}_k^p$ and $\hat{\mathbf{U}}_k^p$ represent the predicted state and control sequences of the UAV,

$$\begin{aligned}
\hat{\mathbf{X}}_k^p &\triangleq \begin{bmatrix} \hat{\mathbf{x}}_{k,0}^p & \hat{\mathbf{x}}_{k,1}^p & \dots & \hat{\mathbf{x}}_{k,N-1}^p & \hat{\mathbf{x}}_{k,N}^p \end{bmatrix}, \\
\hat{\mathbf{U}}_k^p &\triangleq \begin{bmatrix} \hat{\mathbf{u}}_{k,0}^p & \hat{\mathbf{u}}_{k,1}^p & \dots & \hat{\mathbf{u}}_{k,N-1}^p \end{bmatrix}.
\end{aligned} \tag{3.36}$$

The input applied to the UAV is the first sample of the optimal control sequence at time instant k.

At each sampling time k, the target's state distribution is updated and subsequently reduced using the previously described methods. Then, the MPC objective function is established by considering the propagation of the target's position distribution over the prediction horizon, conditioned on the measurements up to time instant k. More specifically, the objective function is defined by summing the probabilities, conditioned on the measurements up to time k, of the target being inside the vehicle's observation radius at each prediction instant j, i.e.,

$$J_k(\hat{\mathbf{\Phi}}_k^p) = \sum_{j=0}^{N} \int_{\mathcal{B}_r(\hat{\boldsymbol{\varphi}}_{k,j}^p)} p(\boldsymbol{\varphi}_{k+j}^e | \mathbf{Y}_k)\, d\boldsymbol{\varphi}_{k+j}^e, \tag{3.37}$$

where $\hat{\mathbf{\Phi}}_k^p \triangleq [\hat{\boldsymbol{\varphi}}_{k,0}^p\ \hat{\boldsymbol{\varphi}}_{k,1}^p\ \dots\ \hat{\boldsymbol{\varphi}}_{k,N}^p] = \mathbf{C}_p \hat{\mathbf{X}}_k^p$ is the predicted discrete-time trajectory of the vehicle at time instant k. However, similarly to what was discussed in the previous chapter, the integrals outlined in (3.37) lack a closed-form expression but can be approximated through numerical means. Nevertheless, for the sake of simplicity, the objective function is established by considering the target's position distribution evaluated at the predicted pursuer positions,

$$J_k(\hat{\mathbf{\Phi}}_k^p) = \sum_{j=0}^{N} p(\boldsymbol{\varphi}_{k+j}^e | \mathbf{Y}_k)|_{\boldsymbol{\varphi}_{k+j}^e = \hat{\boldsymbol{\varphi}}_{k,j}^p}. \tag{3.38}$$

One issue with this formulation is that if the disturbance vector $\mathbf{w}_k$ is modeled by a Gaussian mixture distribution composed by multiple components, the number of components of the target's distribution explodes over the prediction horizon. Consequently, when facing this scenario, it would be necessary to either introduce an additional approximation or redefine the objective function in another way.

3.5 Simulation Results

In this section, the efficacy of the proposed MPC algorithm is assessed through some simple illustrative examples obtained within a MATLAB environment. The objective is to analyze the behavior and performance of the proposed MPC-based approach.

3.5.1 Simulation Setup

For simplicity, we assume that both the pursuer and the target are modeled by double-integrator systems. More precisely, the target is modeled by the following discrete-time stochastic system

$$\mathbf{x}_{k+1}^{e} = \mathbf{A}_e \mathbf{x}_k^e + \mathbf{w}_k, \tag{3.39}$$

where $\mathbf{w}_k \sim \mathcal{N}(\mathbf{w}_k; \mathbf{B}_e \mathbf{u}_k^e, \mathbf{Q})$, and the drone is modeled by the following deterministic system

$$\mathbf{x}_{k+1}^{p} = \mathbf{A}_p \mathbf{x}_k^p + \mathbf{B}_p \mathbf{u}_k^p, \tag{3.40}$$

where the model matrices are given by

$$\mathbf{A}_p = \mathbf{A}_e = \begin{bmatrix} \mathbf{I}_{2\times 2} & T_s\,\mathbf{I}_{2\times 2} \\ \mathbf{0}_{2\times 2} & \mathbf{I}_{2\times 2} \end{bmatrix},\ \mathbf{B}_p = \mathbf{B}_e = \begin{bmatrix} T_s^2/2\,\mathbf{I}_{2\times 2} \\ T_s\,\mathbf{I}_{2\times 2} \end{bmatrix}. \tag{3.41}$$

Additionally, the auxiliary matrices that extract the position and velocity from the state are

$$\mathbf{C}_p = \mathbf{C}_e = \begin{bmatrix} \mathbf{I}_{2\times 2} & \mathbf{0}_{2\times 2} \end{bmatrix},\ \mathbf{C}_p' = \mathbf{C}_e' = \begin{bmatrix} \mathbf{0}_{2\times 2} & \mathbf{I}_{2\times 2} \end{bmatrix}. \tag{3.42}$$

At each discrete-time instant k, the MPC algorithm is based on the following optimization problem

$$\begin{aligned} \underset{\hat{\mathbf{X}}_k^p, \hat{\mathbf{U}}_k^p}{\text{maximize}} \quad & J_k(\hat{\mathbf{X}}_k^p, \hat{\mathbf{U}}_k^p) \\ \text{subject to} \quad & \hat{\mathbf{x}}_{k,0}^p = \mathbf{x}_k^p, \\ & \hat{\mathbf{x}}_{k,j+1}^p = \mathbf{A}_p \hat{\mathbf{x}}_{k,j}^p + \mathbf{B}_p \hat{\mathbf{u}}_{k,j}^p,\ j = 0, \ldots, N-1, \\ & \left\| \mathbf{C}_p' \hat{\mathbf{x}}_{k,j}^p \right\| \leq v_{xy}^{\max},\ j = 0, \ldots, N, \\ & \left\| \hat{\mathbf{u}}_{k,j}^p \right\| \leq a_{xy}^{\max},\ j = 0, \ldots, N-1, \end{aligned} \tag{3.43}$$

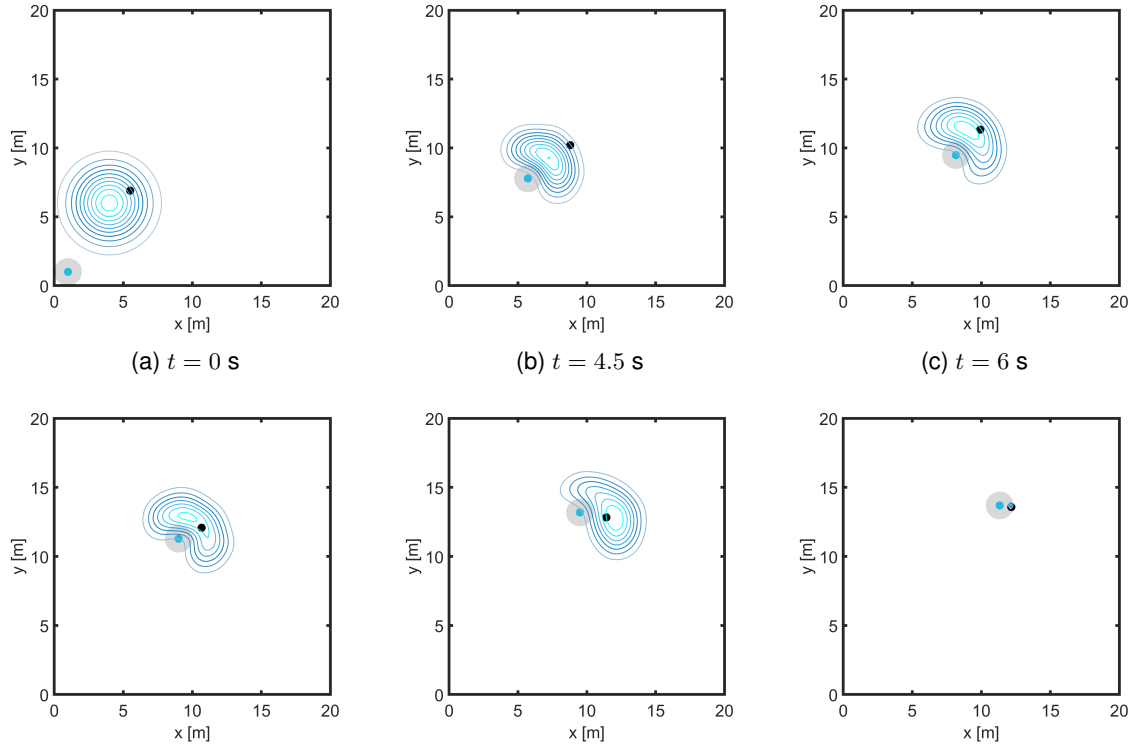
where the objective function J_k is obtained as described in 3.4. The parameters $v_{xy}^{\max}$ and $a_{xy}^{\max}$ are, respectively, the maximum velocity and acceleration that the vehicle (pursuer) may achieve.

The simulation results presented in this section were obtained in MATLAB [40] using the CasADi [27] optimization modeling toolbox, along with the IPOPT [28] numerical solver. At each sampling time, the solution obtained at the previous step is used to set the initial guess for the current step by performing the shifting warm-start method described in Section 1.3. All the computations were executed on a single desktop computer equipped with an Intel Core i7-6700K @ 4.00 GHz processor and 32.00 GB of RAM.

In the following examples, the drone starts at the position $\mathbf{p} = [1\ 1]^\top$ with no initial velocity, and the radius of observation is assumed to be $r = 1$ m. The sampling period is $T_s = 0.1$ s, the horizon length is $N = 15$, and the vehicle has a maximum velocity of 4 m/s and a maximum acceleration of 4 m/s^2. When the target is outside the UAV's visibility region, the rectangular confidence region is obtained as described in Section 3.4 using the 95% confidence ellipsoids of each component. The grid cells of the measurement model have a side length equal to the radius of the UAV ($r = 1$ m). Moreover, when the target is outside the UAV's visibility region, the covariance matrix of each component of the measurement model is selected so that the circle inscribed within each grid square encompasses 0.4% of the associated Gaussian component, similar to the case presented in Figure 3.3. When the target is inside the visibility region of the UAV, the covariance matrix of the additive Gaussian noise is given by $\mathbf{R_v} = 0.01\mathbf{I}_{2\times 2}$. The reduction of the target's distribution is performed using the Kullback-Leibler reduction method described in Section 3.3, with a hard limit of 200 Gaussian components and a threshold on the KL divergence of $\lambda = 0.001$.

3.5.2 Example 1

In this first example, the initial state distribution of the target is defined by a single Gaussian component with mean $\bar{\mathbf{x}}_0^e = [4\ 6\ 0.5\ 0.5]^\top$ and covariance matrix $\mathbf{P}_0^e = \text{diag}([3\ 3\ 1\ 1])$. The disturbance vector is chosen to have little influence and is defined by $\mathbf{w}_k \sim \mathcal{N}(\mathbf{w}_k; \mathbf{0}_4, 0.01\mathbf{I}_{4\times 4})$. The actual initial state of the target is $\mathbf{x}_0^e = [5.5\ 6.9\ 0.75\ 0.75]^\top$. Figure 3.5 presents snapshots of the resulting evolution of the drone, the target, and the updated distribution of the target's position.



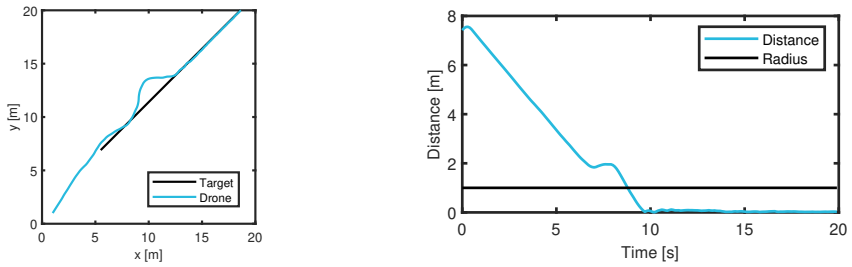
(a) $t = 0$ s (b) $t = 4.5$ s (c) $t = 6$ s

(d) $t = 7$ s (e) $t = 8$ s (f) $t = 9$ s

Figure 3.5: Snapshots of the drone, the target, and the updated distribution of the target's position.

As illustrated in Figure 3.5, the drone initially searches for the target, and the updates are carried out using the proposed approximation of the measurement model. In particular, it should be noticed that the target is initially positioned away from the maximizer of its prior position distribution, and the initial target's velocity is higher than its initial expected velocity. However, as time progresses and updates are applied, the distribution maximizer gradually converges towards the actual position of the target. Specifically, it can be noticed that, right before the initial detection, the distribution maximizer is very close to the actual target's location, as illustrated in Figure 3.5 (e).

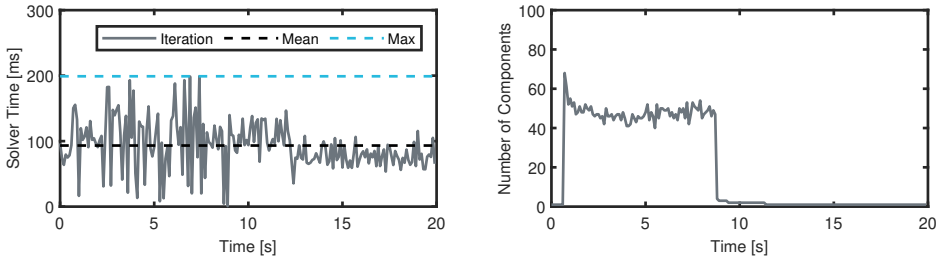
Due to the sufficiently fast dynamics of the pursuer, and since the disturbance is not significant, the pursuer then manages to keep the target in its FOV after the first detection, as shown in Figure 3.6. It is also worth emphasizing that the initial state distribution of the target plays a crucial role since it reflects the initial belief about the state of the target. Consequently, if the prior distribution is not reasonable or sufficiently accurate, it can significantly impact the pursuer's ability to locate the target.



(a) Trajectories (b) Distance

Figure 3.6: Drone and target trajectories and distance between them as a function of time.

Additionally, Figure 3.7 displays the solver times and the number of Gaussian components comprising the target's distribution over time. The solver times are significantly higher when compared to the second chapter. This increased duration can be attributed to the greater complexity of the objective function. Moreover, it should be noticed that the updates only occur at each sampling time, and the objective function does not consider any updates. Consequently, at each time step, a more substantial adjustment of the MPC solution is required when compared to the second chapter, where the penalizations within the horizon made the predicted evolution closer to the real one. Ultimately, it can be observed that the target's distribution is composed of approximately 50 components when it is outside the vehicle's FOV, and the distribution converges to a single component once the target is located.

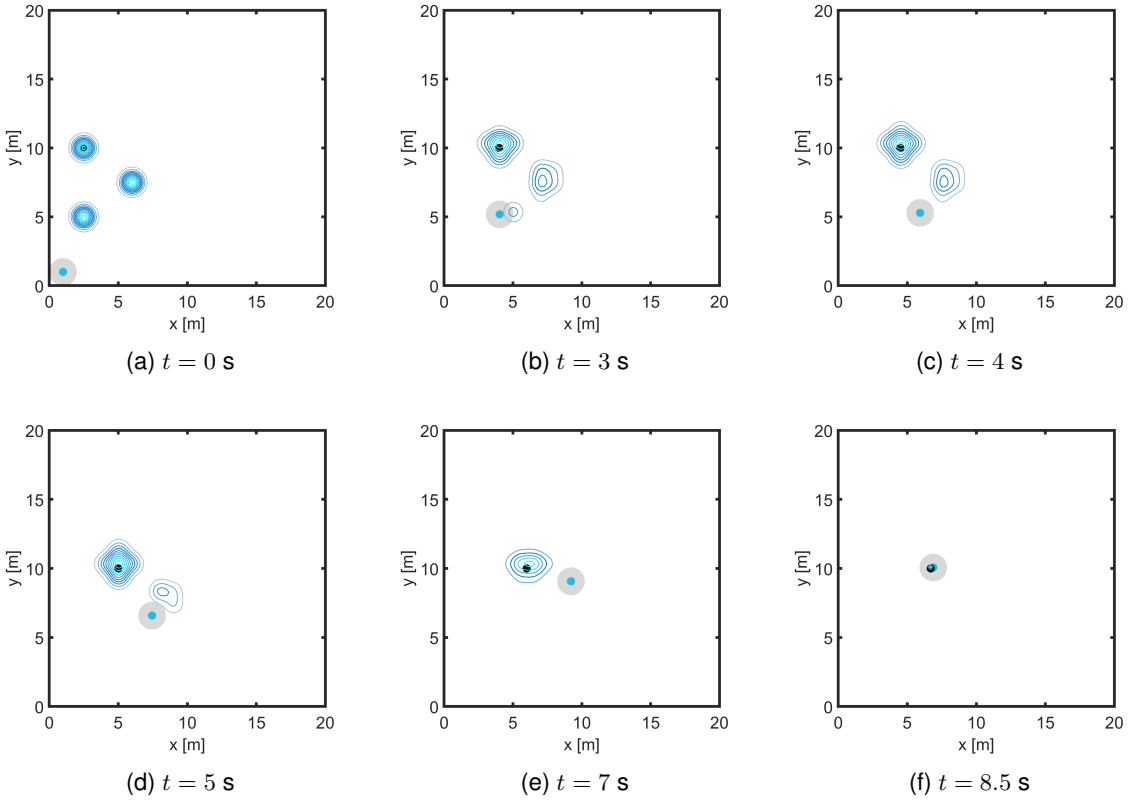


(a) Solver times (b) Number of components

Figure 3.7: Solver times and number of Gaussian components in the distribution of the target.

3.5.3 Example 2

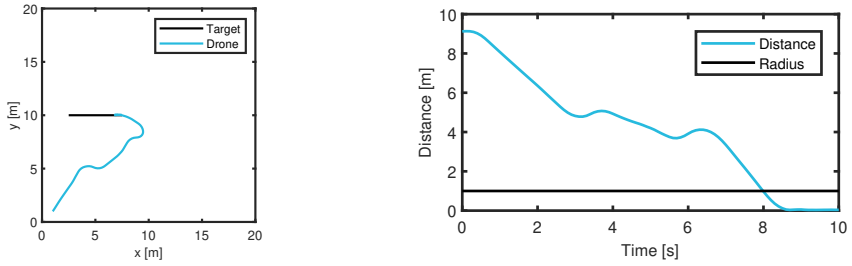
In addition, we present a second example where the initial state distribution of the target is defined by three Gaussian components. The initial means are given by $\bar{\mathbf{x}}_0^{e,1} = [2.5\ 5\ 0.5\ 0]^\top$, $\bar{\mathbf{x}}_0^{e,2} = [2.5\ 10\ 0.5\ 0]^\top$, and $\bar{\mathbf{x}}_0^{e,3} = [6\ 7.5\ 0.5\ 0]^\top$. Moreover, the initial components have all the same weight (1/3), and the same covariance matrix given by $\mathbf{P}_0^{e,1} = \mathbf{P}_0^{e,2} = \mathbf{P}_0^{e,3} = \text{diag}([0.25\ 0.25\ 0.1\ 0.1])$. The disturbance vector is again defined by $\mathbf{w}_k \sim \mathcal{N}(\mathbf{w}_k; \mathbf{0}_4, 0.01\mathbf{I}_{4\times4})$. The initial state of the target coincides with the mean of the initial second component, i.e., $\mathbf{x}_0^e = \bar{\mathbf{x}}_0^{e,2} = [2.5\ 10\ 0.5\ 0]^\top$. Figure 3.8 presents snapshots of the resulting evolution of the drone, the target, and the updated distribution of the target's position. Furthermore, Figure 3.9 displays the resulting trajectories and the evolution of the distance between the drone and the target.



(a) $t = 0$ s (b) $t = 3$ s (c) $t = 4$ s

(d) $t = 5$ s (e) $t = 7$ s (f) $t = 8.5$ s

Figure 3.8: Snapshots of the drone, the target, and the updated distribution of the target's position.

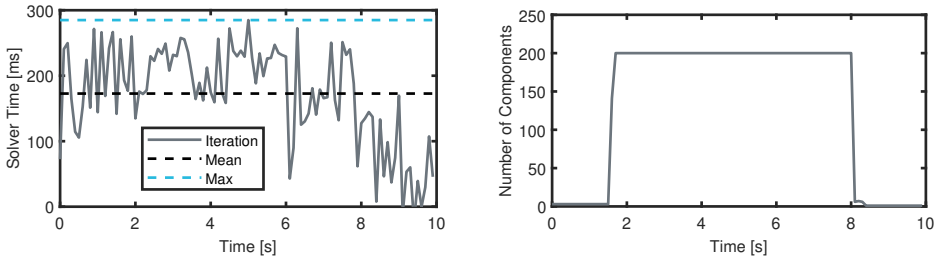


(a) Trajectories (b) Distance

Figure 3.9: Drone and target trajectories and distance between them as a function of time.

In this example, there are three initial components with distinct position means, all moving with the same velocity along the horizontal axis. Moreover, the position covariance of each initial component is small when compared to the visibility region of the UAV. As a result, as illustrated in Figure 3.8, the vehicle moves towards one component at a time, until it finds the target, which is coincident with the mean of one of the components. As shown in Figure 3.9, the vehicle then manages to keep the target within its FOV after the first detection, similar to the first example.

Additionally, in Figure 3.10 we present the solver times and the number of components in the target's distribution as a function of time. As depicted in Figure 3.10 (a), the solver times exhibit a similar profile to the first example, with each iteration averaging approximately 200 ms. In contrast to the first example, Figure 3.10 illustrates that the target's distribution is actually composed of 200 components when the vehicle is searching for the target, which was the maximum limit set for the Kullback-Leibler reduction.



(a) Solver times

(b) Number of components

Figure 3.10: Solver times and number of Gaussian components in the distribution of the target.

3.6 Summary

This chapter considered a more complex scenario: pursuing a stochastic moving target using a UAV. The target was modeled as a generic stochastic discrete-time system with an affine disturbance with known PDF. Additionally, we assumed prior knowledge of an initial state distribution for the target.

We began by reviewing concepts concerning recursive Bayesian filtering, focusing on stochastic systems where the prior, the process, and the sensor model are described by Gaussian mixture distributions. The main advantage of such systems is that the update and prediction steps can be expressed in closed form. However, one drawback is that the number of components comprising the state distribution may increase exponentially. Therefore, a method to reduce the number of components is necessary.

The main difficulty was the approximation of the sensor model when the target is outside the vehicle's visibility region. In such a case, the ideal measurement model would be a function that is zero inside the vehicle's FOV and a positive constant otherwise. To perform the Bayesian update in closed form, we approximated the ideal sensor model using a Gaussian mixture distribution where the components are arranged in a grid over a rectangular confidence region of the target's position distribution.

Once more, we opted for an MPC approach. At each time step, the target's distribution is updated and reduced, and the objective function considers the propagation of the target's position distribution over the horizon. The algorithm was assessed through simple examples with a linear target model.

Chapter 4

Conclusion

4.1 Summary

This dissertation delved into two trajectory planning problems within the context of autonomous surveillance using UAVs. In the initial stage, we addressed the problem of generating optimal trajectories for wildfire prevention based on a map describing the uncertainty of fire presence in a given region, with the map being defined as a linear combination of Gaussian distributions. Our approach involved an MPC algorithm designed to promote the exploration of the map by preventing the UAV from revisiting previously covered areas. This was achieved by penalizing intersections between the circular observation regions along the trajectory of the UAV. Given the computational complexity of precisely computing the intersection area between two circles, we introduced an exponential surrogate function for penalizing such intersections. Regarding the practical implementation of the algorithm, we considered a dual-layer structure of motion control, with the MPC algorithm serving as a high-level trajectory planner and an inner-loop controller responsible for tracking references generated by the MPC. The algorithm was initially tested in a MATLAB environment and subsequently validated in the Gazebo simulator, as well as through actual experiments conducted in an outdoor environment. The results demonstrated that the algorithm can generate high-quality trajectories for surveillance.

Given the link between the first problem and the problem of locating a stationary lost target, we explored a more complex scenario. In the second problem the objective was pursuing a nonadversarial stochastic moving target characterized by a time-varying probability distribution. The target was represented as a generic discrete-time stochastic system with an affine disturbance. Our approach involved an MPC algorithm based on recursive Bayesian filtering for state-space systems described by Gaussian mixture distributions. Such systems have the potential to approximate generic stochastic systems arbitrarily well and provide closed-form expressions for the update and prediction steps. For the case in which the target is outside the visibility region of the UAV, the ideal measurement model was approximated using a grid of Gaussian components. The efficacy of the proposed algorithm was demonstrated through simple examples considering a linear model for the target. In the presented examples, the drone was able to find the target and managed to keep it inside its FOV after the first detection.

4.2 Future Directions

Several issues related to the problems that we have addressed remain open. Specifically, potential avenues for future research can be pursued in the following directions:

- In the context of the first problem, a significant challenge arises from the fact that the number of components in the penalization term increases infinitely with the flight time allocated for the mission. Consequently, for extended missions, the proposed MPC algorithm may impose a substantial computational burden. One evident solution, as discussed in the second chapter, involves setting a maximum length for the backward time horizon, thereby limiting the penalization term to a maximum number of components. Nevertheless, if the backward time horizon is not long enough, the vehicle may revisit areas it has already covered. Therefore, the motivation in this direction revolves around developing a subroutine that can progressively reduce the number of components in the penalization term while retaining information about all the previously explored regions. A potential approach could involve merging the previously covered areas through their convex hulls;
- Another avenue within the framework of the initial problem, which was briefly introduced in the second chapter, involves the extension of the algorithm by incorporating variable weights. It would be interesting to explore the potential of variable weights to finetune the performance of the algorithm across different scenarios and conditions;
- Finally, considering that the experimental trials outlined in this dissertation were conducted with the MPC operating in an offline mode, a prospective direction involves trying to conduct field trials with the MPC algorithm running in real-time onboard the drone;
- Regarding the second problem, as the provided examples were relatively simple, it would be valuable to investigate how the algorithm behaves in more complex scenarios. For instance, one could consider cases where the target's control inputs are not consistently set to zero, scenarios involving nonlinear target models, and situations with significant disturbances where the target can exit the visibility region of the UAV after the initial detection;
- A challenge associated with the suggested approach for addressing the second problem is its limitation in handling disturbances composed of multiple Gaussian components. This limitation arises from the exponential increase in the number of components within the objective function. Therefore, there is motivation to explore methods capable of addressing scenarios involving multiple Gaussian components in disturbances;
- Ultimately, within the context of the second problem, it would be valuable to devise an improved approach for approximating the measurement model when the target is beyond the visibility range of the UAV. Given that the approximation was performed in a heuristic way, exploring more robust methods in this regard could be a potential direction;
- To conclude, it would also be interesting to generalize both problems for multiple UAVs and delve into the field of distributed MPC.

Bibliography

[1] D. González, J. Pérez, V. Milanés, and F. Nashashibi, "A review of motion planning techniques for automated vehicles", *IEEE Transactions on intelligent transportation systems*, vol. 17, no. 4, pp. 1135-1145, 2015.

[2] C. Zhou, B. Huang, P. Fränti, "A review of motion planning algorithms for intelligent robots", *Journal of Intelligent Manufacturing*, vol. 33, no. 2, pp. 387-424, 2022.

[3] R. Kala, K. Warwick, "Multi-level planning for semi-autonomous vehicles in traffic scenarios based on separation maximization", *Journal of Intelligent & Robotic Systems*, vol. 72, pp. 559-590, 2013.

[4] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, et al., "Autonomous driving in urban environments: Boss and the urban challenge", *Journal of field Robotics*, vol. 25, no. 8, pp. 425-466, 2008.

[5] D. Ferguson, T. Howard, and M. Likhachev, "Motion planning in urban environments", *Journal of Field Robotics*, vol. 25, no. 11-12, pp. 939–960, 2008.

[6] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review", *Ieee access*, vol. 2, pp. 56-77, 2014.

[7] L. Kavraki, P. Svestka, J. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces", *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566-580, 1996.

[8] S. LaValle and J. Kuffner, "Randomized kinodynamic planning", *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[9] M. Brezak and I. Petrovíc, "Real-time approximation of clothoids with bounded error for path planning applications", *IEEE Transactions on Robotics*, vol. 30, no. 2, pp. 507–515, 2014.

[10] P. Petrov and F. Nashashibi, "Modeling and nonlinear adaptive control for autonomous vehicle overtaking", *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, no. 4, pp. 1643–1656, 2014.

[11] A. Piazzi, C. L. Bianco, M. Bertozzi, A. Fascioli, and A. Broggi, "Quintic g 2-splines for the iterative steering of vision-based autonomous vehicles", *IEEE transactions on Intelligent Transportation Systems*, vol. 3, no. 1, pp. 27–36, 2002.

[12] B. Krogh, ”A generalized potential field approach to obstacle avoidance control”, *Proc. SME Conf. on Robotics Research: The Next Five Years and Beyond*, pp. 11-22, 1984.

[13] Y. Koren et. al., ”Potential field methods and their inherent limitations for mobile robot navigation”, *ICRA*, vol. 2, no. 1, pp. 1398-1404, 1991.

[14] L. Pontryagin, V. Boltyanskii, R. Gamkrelidze, and E. Mishchenko, *The mathematical theory of optimal processes*. Interscience, 1962.

[15] M. Ardakani, B. Olofsson, A. Robertsson, and R. Johansson, “Real-time trajectory generation using model predictive control”, *2015 IEEE International Conference on Automation Science and Engineering*, pp. 942–948, 2015.

[16] C. Liu, S. Lee, S. Varnhagen, and H. E. Tseng, ”Path planning for autonomous vehicles using model predictive control”, *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 174-179, 2017.

[17] M. Hehn and R. D’Andrea, “Real-time trajectory generation for quadrocopters”, *IEEE Transactions on Robotics*, vol. 31, no. 4, pp. 877–892, 2015.

[18] B. Sabetghadam, R. Cunha, and A. Pascoal, “A distributed algorithm for real-time multi-drone collision-free trajectory replanning”, *Sensors*, vol. 22, no. 5, p. 1855, 2022.

[19] S. Qin and T. Badgwell, ”A survey of industrial model predictive control technology”, *Control engineering practice*, vol. 11, no. 7, pp. 733-764, 2003.

[20] L. Grüne and J. Pannek, *Nonlinear model predictive control*. Springer, 2011.

[21] J. Rawlings and D. Mayne, *Model predictive control: theory and design*. Nob Hill Publishing, 2009.

[22] D. Mayne, ” Model predictive control: Recent developments and future promise”, *Automatica*, vol. 50, no. 12, pp. 2967-2986, 2014.

[23] T. Albin, D. Ritter, D. Abel, N. Liberda, R. Quirynen, and M. Diehl, ”Nonlinear MPC for a two-stage turbocharged gasoline engine airpath”, *54th IEEE Conference on Decision and Control*, pp. 849-856, 2015.

[24] S. Gros, R. Quirynen, and M. Diehl, ”Aircraft control based on fast non-linear MPC & multiple-shooting”, *51st IEEE Conference on Decision and Control*, pp. 1142-1147, 2012.

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

[26] J. Löfberg, ”YALMIP : A Toolbox for Modeling and Optimization in MATLAB”, *CACSD Conference*, 2004.

[27] J. Andersson, J. Gillis, G. Horn, J. Rawlings, and M. Diehl, “CasADi – A software framework for nonlinear optimization and optimal control”, *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[28] A. Wätcher and L. Biegler, ”On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming”, *Mathematical programming*, vol. 106, no. 12, pp. 25-57, 2006.

[29] S. Gros, M. Zanon, R. Quirynen, A. Bemporad, and M. Diehl, “From linear to nonlinear mpc: bridging the gap via the real-time iteration”, *International Journal of Control*, vol. 93, no. 1, pp. 62–80, 2020.

[30] E. Hairer, S. Norsett, and G. Wanner, *Solving ordinary differential equations I*. Springer, 1993.

[31] N. Sönnichsen. ”Land burned by wildfires in Europe by country 2022”. Statista. Available online: https://www.statista.com/statistics/1260777/area-burned-by-wildfire-in-european-countries/ (accessed on June 10, 2023).

[32] N. Sönnichsen. ”Land burned by forest fires in Portugal 2009-2022”. Statista. Available online: https://www.statista.com/statistics/1265367/area-burned-by-wildfire-in-portugal/ (accessed on June 10, 2023).

[33] A. Valles, M. Ferrer, K. Poljanšek, and I. Clark (eds.), *Science for disaster risk management 2020: acting today, protecting tomorrow*. Publications Office, 2020.

[34] Departamento de Gestão de Áreas Públicas e de Protecção Florestal, *10º Relatório Provisório de Incêndios Florestais*. Instituto de Conservação da Natureza e das Florestas, 2017.

[35] Presidência do Conselho de Ministros, ”Resolução do Conselho de Ministros n.º 159/2017”, *Diário da República*, vol. 1, no. 209, pp. 5820-5822, 2017.

[36] W. Press, *Numerical recipes: The art of scientific computing*. Cambridge University Press, 2007.

[37] R. Mahony, V. Kumar, and P. Corke, ”Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor”, *IEEE Robotics & Automation Magazine*, vol. 19, no. 3, pp. 20–32, 2012.

[38] B. Siciliano, L. Sciavicco, L.Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*. Springer, 2010.

[39] ”PX4 Autopilot User Guide”. PX4 Autopilot. Available online: https://docs.px4.io/main/en/ (accessed on June 10, 2023).

[40] The MathWorks Inc., *MATLAB, 9.11.0 (R2021b)*. The MathWorks Inc., 2021.

[41] T. Oliveira, P. Trindade, D. Cabecinhas, P. Batista, and R. Cunha, ”Rapid development and prototyping environment for testing of unmanned aerial vehicles”, *2021 IEEE International Conference on Autonomous Robot Systems and Competitions*, pp. 191–196, 2021.

[42] M. Jacinto, *Cooperative motion control of aerial and marine vehicles for environmental applications*. Master’s Thesis, Instituto Superior Técnico, 2021.

[43] ”Gazebo simulator”. Gazebo. Available online: https://gazebosim.org/ (accessed on June 10, 2023).

[44] ”Iris quadcopter uav”. Arducopter. Available online: https://www.arducopter.co.uk (accessed on June 10, 2023).

[45] M690B Wiki. Available online: https://hardtekpt.github.io/M690B-Wiki/ (accessed on June 10, 2023).

[46] R. Kalman, ”A new approach to linear filtering and prediction problems”, 1960.

[47] G. Smith, S. Schmidt, and L. McGee, *Application of statistical filter theory to the optimal estimation of position and velocity on board a circumlunar vehicle*. National Aeronautics and Space Administration, 1962.

[48] S. Julier, J. Uhlmann, and H. Durrant-Whyte, ”A new approach for filtering nonlinear systems”, *Proceedings of 1995 American Control Conference*, vol. 3, pp. 1628-1632, 1995.

[49] N. Gordon, D. Salmond, and A. Smith, ”Novel approach to nonlinear/non-Gaussian Bayesian state estimation”, *IEE proceedings F (radar and signal processing)*, vol. 140, no. 2, pp. 107-113, 1993.

[50] D. Alspach and H. Sorenson, ”Nonlinear Bayesian estimation using Gaussian sum approximations”, *IEEE transactions on automatic control*, vol. 17, pp. 439-448, 1972.

[51] D. Crouse, P. Willett, K. Pattipati, and L. Svensson, ”A look at Gaussian mixture reduction algorithms”, *14th International Conference on Information Fusion*, pp. 1-8, 2011.

[52] A. Runnals, ”Kullback-leibler approach to gaussian mixture reduction”, *IEEE Transactions on Aerospace and Electronic Systems*, vol. 43, no. 3, pp. 989-999, 2007.

[53] A. Wills, J. Hendriks, C. Renton, and B. Ninness, ”A Numerically Robust Bayesian Filtering Algorithm for Gaussian Mixture Models”, *IFAC-PapersOnLine*, vol. 56, no. 1, pp. 67-72, 2023.

Appendix A

ROS/Gazebo Environment Documentation

This appendix describes the instructions followed to install the ROS/Gazebo simulation enviroment used in this thesis. The instructions are based on the previous work by Oliveira et al. [41] and Jacinto [42]. We also present the instructions followed to build the CasADi C++ API from source and use it within a ROS node. This guide assumes a fresh installation of the Ubuntu 18.04 LTS operating system.

A.1 Environment Setup

Install Gazebo, ROS and MAVROS:

```
$ wget https://raw.githubusercontent.com/PX4/Devguide/master/build_scripts/
    ubuntu_sim_ros_melodic.sh
$ source ubuntu_sim_ros_melodic.sh
$ sudo apt-get install ros-melodic-mavros
```

Install the following Python libraries:

```
$ sudo apt-get install libgstreamer-plugins-base1.0-dev python3-empy python3-jinja2
    python3-toml python3-pip
```

Install the PX4 Autopilot Firmware:

```
$ git clone --recursive https://github.com/PX4/PX4-Autopilot.git
$ cd PX4-Autopilot
$ git fetch --all --tags
$ git checkout v1.12.3 -b latest
$ git submodule update --init --recursive
$ make px4_sitl gazebo
```

The `make px4_sitl gazebo` command will probably fail due to some missing packages. Install the missing packages and run the command again until a Gazebo window opens. Then shut down Gazebo. Also, the file *.ignition/fuel/config.yaml* must be changed so that the server url is "https://fuel.ignitionrobotics.org". After that, add the following commands to your *.bashrc* file, where *username* should be replaced by the name of your user directory:

```
$ alias killros="killall -9 rosmaster gzserver gzclient"
$ alias s="source ~/.bashrc"
```

```
$ export CATKIN_WORKSPACE=~/catkin_ws
$ CATKIN_ROOT=/home/username
$ export GAZEBO_RESOURCE_PATH=/usr/share/gazebo-9
$ export GAZEBO_PLUGIN_PATH=$GAZEBO_PLUGIN_PATH:/home/username/PX4-Autopilot/build_gazebo
$ export GAZEBO_MODEL_PATH=$GAZEBO_MODEL_PATH:/home/username/PX4-Autopilot/Tools/
    sitl_gazebo/models
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/username/PX4-Autopilot/build_gazebo
$ export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:/home/username/PX4-Autopilot:/home/username/
    PX4-Autopilot/Tools/sitl_gazebo
$ source /home/username/PX4-Autopilot/Tools/setup_gazebo.bash /home/username/PX4-
    Autopilot /home/username/PX4-Autopilot/build/px4_sitl_default
```

Run the `make px4_sitl gazebo` command again from the PX4-Autopilot directory and then close Gazebo.

Now that all the necessary packages have been installed, the simulation environment must be configured. Copy the C++ packages from the repository at the url https://github.com/hardtekpt/iris_simulator and place them in the */catkin ws/src* folder.

The installation process should be done by now. To build the packages run the following command:

```
$ catkin build
```

Finally, to launch the simulator run:

```
$ roslaunch drone_sim_bringup simulator_bringup.launch
```

A.2 CasADi Installation

Get the following dependencies:

```
$ sudo apt-get install gcc g++ gfortran git cmake liblapack-dev pkg-config --install-
    recommends
$ sudo apt-get install swig ipython python-dev python-numpy python-scipy python-
    matplotlib --install-recommends
```

Install IPOPT:

```
$ sudo apt-get install coinor-libipopt-dev
```

Clone the CasADi repository:

```
$ git clone https://github.com/casadi/casadi.git -b master casadi
```

Create a build directory:

```
$ cd casadi
$ mkdir build
$ cd build
```

Generate a Makefile with:

```
$ cmake -DWITH_IPOPT=ON ..
```

Finally, build CasADi from source:

```
$ make
$ sudo make install
```

CasADi may then be used in a ROS node by using the `find_package()` function in the corresponding *CMakeLists.txt* file.