Optimal Motion Planning in GPS-Denied Environments using Nonlinear Model Predictive Horizon

by

Younes Moh'd Awad Al-Younes

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Mechanical Engineering
University of Alberta

Abstract

Navigating robotic systems autonomously through unknown, dynamic, and GPS-denied environments is a challenging task. One requirement of this is a path planner which provides safe trajectories under real-world conditions such as nonlinear vehicle dynamics, real-time computation requirements, complex 3D environments, and moving obstacles. The limitations of existing methods for path planning related to computational efficiency, consideration of complex system dynamics, and achieving consistency and optimality of the solution, have pushed our research towards proposing an approach that tackles these challenges all at once.

The goals of the proposed research are threefold. The first goal is proposing a novel framework for trajectory generation based on optimization and implementing it on a multi-rotor drone. The nonlinear dynamics of a drone make it an excellent test candidate for this work. The proposed trajectory planning system, named Nonlinear Model Predictive Horizon (NMPH), extends the well-known Nonlinear Model Predictive Control (NMPC) methodology by combining it with a nonlinear control design, for instance FeedBack Linearization (FBL) or BackStepping Control (BSC). The purpose of embedding a nonlinear control law within the optimization is to reduce the non-convexity of the problem and thus provide faster solutions for the trajectory planning problem. NMPH provides feasible solutions, generates smooth and collision-free trajectories, supports moving obstacles, is able to run in real-time, and reduces battery power consumption by producing smooth flight trajectories for the drone to follow.

The second goal is developing a global motion planning system to allow the drone to explore a complex and GPS-denied environment with the presence of both static and dynamic obstacles. This system is a three-stage modular design that incorporates the NMPH trajectory planning algorithm and a graph-based planner. The first stage operates by building an incremental map of the environment, also containing a volumetric representation of the static and dynamic obstacles. The second stage is a graph-based planner that generates waypoints within unexplored areas of the map. The final stage uses the NMPH algorithm to produce continuous and optimal trajectories from the vehicle's current pose to the waypoints generated by the graph-based planner. For a smooth integration between the three layers, computationally efficient algorithms for obstacle mapping and avoidance plus robust path guidance algorithms are developed. The presented approaches are then implemented in software to generate optimal paths for a drone navigating an unexplored GPS-denied environment, with several simulations and experimental results provided to demonstrate the features and evaluate the performance of the overall design.

The third goal is proposing an adaptive learning scheme for the NMPH algorithm based on Deep Reinforcement Learning (DRL). The resulting design is called 'adaptive NMPH', which generates optimal trajectories for an autonomous drone based on the system's states and its environment. This is done by online tuning the NMPH's optimization parameters using two different Actor-Critic DRL-based algorithms, Deep Deterministic Policy Gradient (DDPG) and Soft Actor-Critic (SAC). Both adaptive NMPH variants are trained and evaluated on an aerial drone inside a high-fidelity simulation environment. The results demonstrate the learning curves, sample complexity, and stability of the DRL-based adaptation scheme, and show the superior performance of adaptive NMPH relative to non-adaptive designs.

Preface

This thesis is an original work by Younes Al-Younes. The work contained herein is my own except where explicitly stated otherwise in the text, and has not been submitted for any other degree or professional qualification.

Chapters 2 and 3 of this thesis have been published as: Y. Al Younes and M. Barczyk, "Nonlinear model predictive horizon for optimal trajectory generation," Robotics, vol. 10, no. 3, p. 90, 2021. I was responsible for conceptualization, methodology, software, validation, formal analysis, investigation, data curation, writing—original draft preparation, and visualization. My supervisor M. Barczyk was involved with conceptualization, and responsible for supervision, project administration, resources, funding acquisition, and writing—review and editing the manuscript.

Chapter 4 of this thesis has been published as: Y. Al Younes and M. Barczyk, "A Backstepping Approach to Nonlinear Model Predictive Horizon for Optimal Trajectory Planning," Robotics, vol. 11, no. 5, p. 87, 2022. I was responsible for conceptualization, methodology, software, validation, formal analysis, investigation, experimentation, data curation, writing—original draft preparation, and visualization. My supervisor M. Barczyk was involved with conceptualization, and responsible for supervision, project administration, resources, funding acquisition, and writing—review and editing the manuscript.

Chapter 5 of this thesis has been published as: Y. Al Younes and M. Barczyk, "Optimal motion planning in GPS-denied environments using nonlinear model predictive horizon," Sensors, vol. 21, no. 16, p. 5547, 2021. I was responsible for conceptualization, methodology, software, validation, formal analysis, investigation, experimentation, data curation, writing—original draft preparation, and visualization. My supervisor M. Barczyk was involved with conceptualization, and responsible for supervision, project administration, resources, funding acquisition, and writing—review and editing the manuscript.

Chapter 6 of this thesis has been published as the article: Y. Al Younes and M. Barczyk, "Adaptive Nonlinear Model Predictive Horizon using Deep Reinforcement Learning for Optimal Trajectory Planning," Drones, vol. 6, no. 11, p. 323, 2022. I was responsible for conceptualization, methodology, software, validation, formal analysis, investigation, implementation, experimentation, data curation, writing—original draft preparation, and visualization. My supervisor M. Barczyk was involved with conceptualization, and responsible for supervision, project administration, resources, funding acquisition, and writing—review and editing the manuscript.

Dedicated to my beloved parents,

Ruwaida** & **Mohammad

to my lovely wife,

Nedaa

and to my precious kids,

Lamar** & **Eyas

for their boundless love, support, and encouragement.

Acknowledgements

This amazing and exciting journey would not have been possible without the help of many people. First and foremost, I would like to express my heartfelt gratitude to my supervisor, Dr. Martin Barczyk, for his profound knowledge and unwavering guidance throughout my Ph.D. studies. Without your unyielding assistance and encouragement, this research would not have been possible. You have truly been an inspiration to me.

To my supervisory committee members Dr. Hossein Rouhani and Dr. Michael Lipsett, I would like to thank you for your invaluable suggestions and support throughout this journey. Please accept my deepest thanks.

To my colleagues at the Mechatronic Systems Lab, thank you very much for fostering a lively and welcoming workplace environment for me. I cannot adequately describe how grateful I am to you for the priceless moments we had together. Thank you, you're amazing!

I would like to thank my family for all of the love, support, inspiration, and prayers they have given me along this journey.

To the joy of my life, Lamar and Eyas, you are my inspiration to reach greatness. You are my soul and the driving force to achieve the unachievable. Thank you for being in my life. I breathe your love every single moment.

Last, but not least, I cannot begin to express my gratitude to my beloved wife, Nedaa, for her unending support before and during my Ph.D. studies. You have been the source of love and strength that empowered me to succeed. Your unconditional love and support have meant the world to me. I hope that I have made you proud.

Table of Contents

List of Tables

List of Figures

List of Algorithms

List of Acronyms

A3C	Asynchronous Advantage Actor–Critic.
AI	Artificial Intelligence.
ANN	Artificial Neural Network.
ARA*	Anytime Repairing A*.
BFS	Breadth-First Search.
BSC	Backstepping Control.
CG	Center of Gravity.
CHOMP	Covariant Hamiltonian Optimization for Motion Planning.
DDPG	Deep Deterministic Policy Gradient.
DFS	Depth-First Search.
DLOM	Dynamic Local Obstacle Mapping.
DOF	Degrees-Of-Freedom.
DRL	Deep Reinforcement Learning.
EKF	Extended Kalman Filter.
ENU	East, North, and Up.
ESDF	Euclidean Signed Distance Field.
FBL	Feedback Linearization.
FLOAM	Fast Lidar Odometry And Mapping.

FUEL Fast UAV Exploration.

GPS Global Positioning System.

GPU Graphics Processing Unit.

HITL Hardware-In-The-Loop.

IMU Inertial Measurement Unit.

LPA* Lifelong Planning A*.

LQR Linear Quadratic Regulator.

MIMO Multi-Input Multi-Output.

MPC Model Predictive Control.

MSBE) Mean Square Bellman Error.

NED North, East, and Down.

NLP Nonlinear Programming.

NMHE Nonlinear Moving Horizon Estimation.

NMPC Nonlinear Model Predictive Control.

NMPH Nonlinear Model Predictive Horizon.

OCP Optimal Control Problem.

OP Optimization Problem.

OTP Optimal Trajectory Problem.

PRM Probabilistic Road-Map.

RL Reinforcement Learning.

ROS Robot Operating System.

RRG Rapidly-exploring Random Graph.

RRT	Rapidly-exploring Random Tree.
SAC	Soft Actor-Critic.
SE()	Special Euclidean group.
SO()	Special Orthogonal group.
SQP	equential Quadratic Programming.
TD3	Twin Delayed Deep Deterministic.
TSDF	Truncated Signed Distance Field.

Chapter 1

Introduction

1.1 Motion Planning

Throughout the last century, injuries and fatalities in subterranean environments have remained a major concern around the world. For example, mine workers are vulnerable to hazards such as cave-ins, underground flooding, and gas explosions. Unmanned vehicles can play a key role in performing both tedious and dangerous tasks, for instance air quality sampling, tunnel inspections, and search-and-rescue missions. A flying drone is a particularly attractive platform for underground operations due to its abilities to move quickly, traverse any terrain, navigate through tight spaces, and capture data from any angle. Recent advances in robotics have motivated research into designing novel path planning approaches, allowing the vehicle to plan safe paths and navigate through previously unknown environments.

Planning collision-free trajectories for autonomous unmanned vehicles operating within unknown, dynamic, 3D, geometrically complex, and GPS-denied environments is a challenging and exciting research problem for both academia and industry. For agile drone systems, generating efficient trajectories in real-time requires using path or trajectory planning methods which respect the vehicle's dynamics and input constraints as part of the prediction process. Researchers are now studying different planning approaches which can take these considerations into account, for instance receding horizon-based methods [1–3].

Path planning, also called motion planning, is a computational problem to generate and follow a collision-free trajectory from one point to another [4]. It has many applications, such as robotic surgery [5], driverless cars [6], automation [7], and mining [8]. An extensive amount of research has been conducted in the field of path planning for autonomous vehicles [6, 9]. However, most of the presented approaches provide non- or sub-optimal solutions and do not account for the dynamics of the vehicle, instead treating it as a kinematic model with velocity inputs [4], for instance a unicycle or kinematic car [10]. Moreover, navigating through dynamic and unknown environments is a challenging task as it requires safe navigation around both static and dynamic obstacles, which adds computational load for the onboard computer of the autonomous vehicle. On the other hand, trajectory planning provides a parameterized path from a starting configuration to a terminal setpoint while avoiding obstacles. It is considered a superset of path planning by generating reference kinematics over the entire path instead of geometric paths only [11]. Nonlinear Model Predictive Control (NMPC [12]) is an attractive methodology to address the above-named challenges, since it is capable of predicting parameterized optimal paths (trajectories), accounts for the dynamics of the plant, and supports hard state constraints which can be used to model either static or dynamic obstacles.

Planning algorithms have received much attention from robotics researchers, where most of the published algorithms fall under one of the following categories: search-based, sampling-based, artificial potential field, artificial intelligence, and optimization-based methods. The search-based methods, a.k.a. grid-based, discretize the environment map into a graph of grids then use a search algorithm to find a collision-free path through these grids [9]. The two fundamental graph search algorithms are Breadth-First Search (BFS) and Depth-First Search (DFS) [13]. BFS is based on a first-in-first-out queue and can produce an optimal solution if the graph is uniformly weighted. Meanwhile, a last-in-first-out stack is used in DFS until the goal is reached, but no optimality is guaranteed.

One of the most widely used optimal searching algorithms for quickly finding the shortest path is the Dijkstra algorithm [14]. It directs the search towards unvisited nodes, then calculates and updates the shortest distances to the neighbor nodes from the root node. It keeps doing this until all the nodes are visited. Meanwhile, A* [15] is a commonly used algorithm for path planning. A* is an extension to Dijkstra algorithm, which combines the cost search with heuristics that guide the search towards the goal point to achieve quicker searching performance. Many extensions of A* have been proposed, for instance Lifelong Planning A* (LPA*) [16] was developed to support changes in the environment without recalculating the entire graph, D* Lite [17] extended LPA* works to re-plan the path while the robot is moving, Anytime Repairing A* (ARA*) [18] improves the optimality of the path by reusing suboptimal solutions from previous executions, and Hybrid-state A* [19] generates the graph based on the robot's velocity, acceleration, and force constraints, and thus searches for a dynamically feasible path.

The sampling-based methods are considered one of the main motion planning methods for robots with a high number of Degrees-Of-Freedom (DOF) [20]. In these methods, feasible robot poses are randomly sampled to form admissible paths. Probabilistic Road-Map (PRM) [21] and Rapidly-Exploring Random Tree (RRT) [22] are the fundamental sampling-based methods for motion planning. In PRM, a graph is built from random configurations and connected using a local planner (for instance Dijkstra's searching algorithm for the shortest path between two configurations). PRM is complete but does not necessarily provide an optimal path solution. The RRT method is designed to randomly build a space-filling tree of vertices and edges inside a complex environment to find a feasible path to the goal node. However, the RRT-generated paths are not optimal [23]. Asymptotic optimality of paths can be achieved by employing various extensions of RRT, such as RRT* and Rapidly-Exploring Random Graph (RRG) methods [23]. RRG constructs a graph by connecting new samples with all nodes within a specified distance, then finding the shortest path using a local planner such as the Dijkstra algorithm. Meanwhile, RRT* searches the local nodes and finds the shortest path from the start to the end nodes.

The artificial potential field method developed in [24] plans a path to a goal that avoids collisions by assigning an attractive force to the desired goal and repulsive forces to obstacles [25]. A variety of artificial intelligence-based algorithms for path planning have been proposed in the literature, for instance Artificial Neural Network (ANN) [26], Genetic Algorithm [27], Ant Colony Optimization [28], Particle Swarm Optimization [29], and Simulated Annealing [30] algorithms. Reference [31] presents a comprehensive review of artificial intelligence-based methods for path planning up to 2018.

In general, there are three main limitations of the previously mentioned motion planning approaches. First, they do not account for the constraints imposed by the robot dynamics, even if some support kinematic and/or dynamic constraints (e.g. velocity and/or acceleration limits, respectively) [32]. A second limitation is consistency, since for several executions the algorithms may not produce identical trajectories between a start and goal configuration in the very same environment. Third, the computational load of these methods generally prevents them from being able to actively regenerate paths while moving between the start and goal configuration, which keeps them from being used in dynamic environments. In addition, some approaches generate non-smooth paths that lead to jerky motions and create inefficiency in the vehicle's power draw [4]. However, some optimization-based methods can overcome these limitations, and the present work is directed at using optimization for planning safe, consistent, and time-efficient paths which also respect the dynamics of the vehicle. This last feature allows generating smooth trajectories for the robot vehicle, avoiding the jerky motions and rapidly changing trajectories often generated by other planning methods [4].

Recently, optimization-based motion planning methods have gained researchers' attention due to their ability to resolve some or all of the above-mentioned limitations. The optimization-based approaches solve a constrained non-convex optimization problem, where some of the methods use cost-gradient information of a trajectory's waypoints for refinement purposes. The best-known optimization-based methods for motion planning are Covariant Hamiltonian Optimization for Motion Planning (CHOMP) [33] and Stochastic Trajectory Optimization for Motion Planning [34]. The former uses gradient-based optimization while the latter uses stochastic optimization, and both produce collision-free trajectories that satisfy given constraints, but are computationally expensive. In addition, CHOMP is prone to getting trapped in local minima, where it returns infeasible or sub-optimal solutions.

Other optimization-based methods are more closely related to optimal control, which focus on system dynamics more than collision prevention. Examples include dynamic programming [35], LQR-based [36], and Model Predictive Control (MPC) [37].

One of the challenges of using an optimization-based path planning approach is accounting for obstacle constraints at each time instant the optimization problem is solved, especially for real-time implementations [38]. For a small number of obstacles, it has been demonstrated that finding local optimal trajectories is possible with MPC in outdoor environments [39]. Conversely, increasing the number of obstacles and considering 3D and dynamic environments makes the optimization problem much more computationally expensive to find feasible paths in real-time. Also, using the Nonlinear version of MPC (a.k.a. NMPC) for nonlinear systems makes solving the optimization problem more challenging in terms of feasibility and stability of the solution.

Despite these challenges, NMPC is considered an interesting candidate for path planning that can generate real-time trajectories for highly nonlinear systems which respect the system dynamics as well as internal and external constraints.

1.2 Nonlinear Model Predictive Approach

Nonlinear Model Predictive Control (NMPC) is the nonlinear variant of Model Predictive Control (MPC), which was originally developed for process control applications and can be traced back to the late 1970s [40, 41]. MPC, sometimes called moving horizon or receding horizon optimal control, involves using a system dynamics model to predict a sequence of control inputs over a time interval known as the prediction horizon. For nonlinear systems, NMPC can be used as an optimization-based feedback control technique [12], and it has been viewed as one of the few control strategies which can account for state and input constraints plus respect the nonlinearities and coupling of the system dynamics.

NMPC is employed in applications including setpoint stabilization, trajectory tracking, and path following. In setpoint stabilization, the system is controlled to converge to a specified setpoint within a terminal region. NMPC for setpoint stabilization has been used in many applications, such as fluid level and temperature control [42, 43]. In trajectory tracking, the system must track a time-varying reference, which is a more challenging problem. Some examples of this taken from the fields of aerial vehicles and medicine are presented in [44, 45]. Meanwhile, the path following problem considers time-invariant reference trajectories, where the goal is to achieve the best possible tracking of the geometric path regardless of the time taken. A common example of a path following problem is controlling the end-effector of a robot manipulator to follow a prescribed path within its workspace, which is treated using MPC in [46, 47]. [48] provides a comprehensive overview of the setpoint stabilization, trajectory tracking, and path following problems and their relevant features and challenges.

Within the model predictive control approach, the optimization problem solves for both the input and the state of the system over a finite time horizon. These local plans can then be combined online to generate a prediction of the state trajectory, which is used for motion planning [49]. The work in [50] considers the problem of point-to-point trajectory generation using linear MPC, while NMPC is used for trajectory optimization and tracking control in [51], where it solves the nonlinear problem using an iterative sequential linear quadratic algorithm to obtain the optimal feedforward and feedback control inputs to a hexacopter vehicle.

Our work presents the development of an algorithm which allows the generation of optimal trajectories based on the NMPC approach. This proposed formulation is called *Nonlinear Model Predictive Horizon* (NMPH). The novelty of NMPH is to incorporate a nonlinear control law to reduce or eliminate the non-convexity of the optimization problem, as opposed to working directly with the nonlinear plant dynamics as in standard NMPC.

To understand the differences between the proposed method and approaches based on model predictive control, Table 1.1 shows a comparison between NMPC, NMPH and NMHE (Nonlinear Moving Horizon Estimation). NMHE is an optimization-based technique which inputs measurements, which may contain noise and other uncertainties, and outputs estimates of the nonlinear system state or parameters. Further information about NMHE is given in [52].

Table 1.1: Comparison between Nonlinear Model Predictive Control-based approaches.

	NMPC	NMHE	NMPH (ours)
Objective	Predicts future control inputs and states of the system	Estimates the system states from previous measurements over the estimation horizon	Plans an optimal reference trajectory for the system under an existing feedback control design

Optimization Problem (OP)	Dynamic OP is solved iteratively for the optimal control inputs over the prediction horizon	OP is solved for state estimates and model parameters	Dynamic OP is solved iteratively for the optimal trajectory over the given prediction horizon
Cost Function	In general, a quadratic function which penalizes deviations of the predicted system states and control inputs. Composed of a *stage cost* and a *terminal cost*	In general, a quadratic function which penalizes deviations of the estimated outputs from the measured outputs. Composed of an *arrival cost* and a *stage cost*	Quadratic function which penalizes the deviation of the predicted system states and reference trajectory. Composed of a *stage cost* and a *terminal cost*.
Optimization Variables	System inputs (states might be considered in some implementations)	System states and parameters	System states and prediction of the reference trajectory.
OP Convexity	Non-convex	Non-convex	Reduced non-convexity or convex
OP Constraints	* Initial state * Nonlinear system model * Limits on states and control inputs	* Nonlinear system model * Limits on states and parameter values	* Initial state * Nonlinear system model * Limits on trajectories, states, controls; obstacles
Optimization Perfor-mance	Depends on the accuracy of the system model and initial state estimate	Sensitive to the accuracy of the system model. Process noise may affect the solution, leading to inaccurate or unstable results	Relies on the accuracy of the system model, stability of closed-loop system, and accuracy of the initial state estimate

1.3 Adaptation and Deep Reinforcement Learning

For autonomous robotic systems, some path planning and trajectory tracking control formulations require accurate system dynamics models to design the control and navigation algorithms [53]. However, obtaining accurate models is challenging in practice, especially if the system dynamics are time- or task-varying. Changes in system dynamics require updating the system model and/or the associated control and navigation algorithms. For instance, adaptive control designs adjust the controller's parameters in response to changes in the system dynamics and the environment [54]. Adaptive control methods can be traced back to the 1950s and early 1960s [55]. Richard Bellman showed how Dynamic Programming is related to the different aspects of adaptation [56], and various adaptive flight control systems from this era are reported in [57]. One of the simplest instances of adaptive control is dynamically adjusting the gains of a PID control law; some techniques proposed by researchers for online PID tuning include [58–61].

The world is witnessing rapid progress in the use of Artificial Intelligence (AI) techniques for self-adaptive systems [62]. In particular, some AI-based techniques have generated great interest for adaptive control designs for mobile robots [63–65]. One of the most productive paradigms in AI is Reinforcement Learning (RL), which is a learning method for an agent interacting with its environment [66]. In the literature, RL has been used by researchers as an adaptive control strategy, for instance a Q-learning based cruise control method was developed by [67] to control a vehicle's speed on curved lanes, where Q-learning [68] is an RL algorithm that learns the value of an action for a given state of the system. For online tuning purposes, [69] used the Q-learning method to auto-tune fuzzy PI and PD controllers for both single- and multi-input/output systems, while [70] used an Actor-Critic RL technique to tune the weights of an LQR controller to adjust to different payloads being carried by a robot arm manipulator.

Recent Developments in RL have made it possible to use neural networks as approximators of the RL value and policy functions [66]. In general, RL methods that use neural networks in their structure are called Deep Reinforcement Learning (DRL). A class of DRL methods which support continuous-time system models belong to the Actor-Critic family [71], such as the Deep Deterministic Policy Gradient (DDPG) [72], Twin Delayed Deep Deterministic (TD3) [73], Soft Actor-Critic (SAC) [74], and Asynchronous Advantage Actor–Critic (A3C) [75]) algorithms. Actor-Critic methods simultaneously learn policy and value functions that are maintained independently using separate memory structures [66]. The Actor is a policy function that selects the best action for the current observations, and the Critic is a value function that criticizes the actions made by the actor. The algorithms listed above have recently begun being used to implement adaptive control. For example, the DDPG algorithm was used by [53] for self-tuning gains of PID controllers onboard mobile robots, while [76] utilized the A3C algorithm to tune the gains of a PID controller used for position control of a two-phase hybrid stepping motor. DRL-based algorithms can also be used to autonomously tune the parameters of algorithms other than control, for instance path planning. This will be the focus of one of our work directions, specifically using two different DRL algorithms as adaptive solutions to enhance the performance of our proposed motion planning approach by tuning the parameters of the optimization problem online.

1.4 Thesis Objectives

This research aims to develop an optimization-based trajectory planning algorithm for an autonomous drone vehicle to navigate through unexplored and GPS-denied environments. In particular, this thesis encompasses the development, implementation, and validation of the following methodologies:

1. Proposing a novel framework for an optimization-based trajectory generation

approach and implementing it on a multi-rotor drone. This framework was formulated with two different nonlinear control laws to efficiently solve the optimization problem and provide feasible, optimal, smooth, and collision-free trajectories that support static and moving obstacles in real-time.

2. Incorporating the proposed trajectory planning method into a methodological design of a global motion planner to explore unknown, unstructured, and GPS-denied environments; and developing computationally efficient algorithms for obstacle mapping and avoidance, plus robust path guidance algorithms to enhance the vehicle's robustness in performing autonomous navigation and exploration within these environments.

3. Proposing an adaptive learning scheme based on Deep Reinforcement Learning to dynamically adjust the parameters of the optimization problem based on the vehicle's state and the environment conditions, leading to better flight trajectory generation performance. The system was implemented using two different deep reinforcement learning algorithms, and the resulting designs were evaluated in terms of their learning performance and ability to provide the best possible optimal trajectories for an autonomous drone vehicle in real-time.

1.5 Thesis Outline

This thesis is based on four published manuscripts [77–80]. The thesis chapters are formatted to ensure structured and non-repetitive contents. Chapter 7 provides an overview of the hardware and software used in the real-time testing.
The remainder of this thesis is organized as follows:

- Chapter 2 introduces the Nonlinear Model Predictive Horizon framework for optimal trajectory generation, and the different constraints employed within its optimization problem.

- Chapter 3 presents the formulation of NMPH design with the state feedback linearization control law (NMPH-FBL) for optimal trajectory generation.

- Chapter 4 provides a new candidate to the NMPH design by integrating it with the nonlinear backstepping control technique (NMPH-BSC) for optimal and efficient trajectory planning.

- Chapter 5 presents a methodological global motion planner design and choices of algorithms to provide robust path planning and obstacle avoidance.

- Chapter 6 introduces the development of the adaptive NMPH framework with deep reinforcement learning methods for online tuning of the NMPH optimization problem parameters to enhance trajectory planning performance.

- Chapter 7 provides an overview of the hardware of our drone vehicle and the software architecture used in the real-time flight tests.

- Chapter 8 summarizes and concludes the work presented in this thesis, and outlines different future research directions of the work.

The overall system architecture of the work included in this thesis is depicted in Figure 1.1.

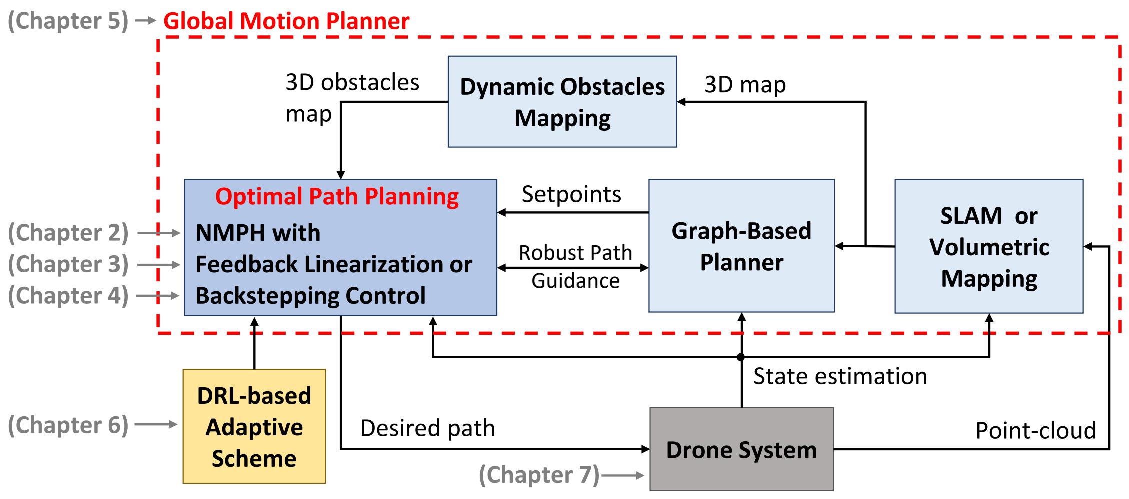


Figure 1.1: Overall System Architecture.

Chapter 2

Nonlinear Model Predictive Horizon for Optimal Trajectory Generation

2.1 Introduction

Our proposed Nonlinear Model Predictive Horizon (NMPH) is an optimization-based trajectory generation method based on Nonlinear Model Predictive Control (NMPC). Unlike NMPC, which yields an optimal feedback control law for a nonlinear system, the objective of NMPH is to generate optimal reference trajectories that a closed-loop system can follow.

The goal of NMPH is to generate a smooth trajectory which is continuously updated by solving an Optimal Control Problem (OCP) in real time while respecting the state and input constraints of the closed-loop system. The resulting optimization problem will be referred to as an *Optimal Trajectory Problem* (OTP).

An overview of the NMPH architecture is depicted in Figure 2.1. The *Nonlinear System Model* and the *Nonlinear Control Law*, representing the model of the plant and the nonlinear control law design, are both involved in the solution of the optimization problem. The NMPH inputs the current system state and a desired setpoint stabilization, and outputs an optimal reference trajectory by solving the OTP at each time instant t_n.

Our proposed NMPH method:

- predicts the trajectory of a nonlinear closed-loop system,

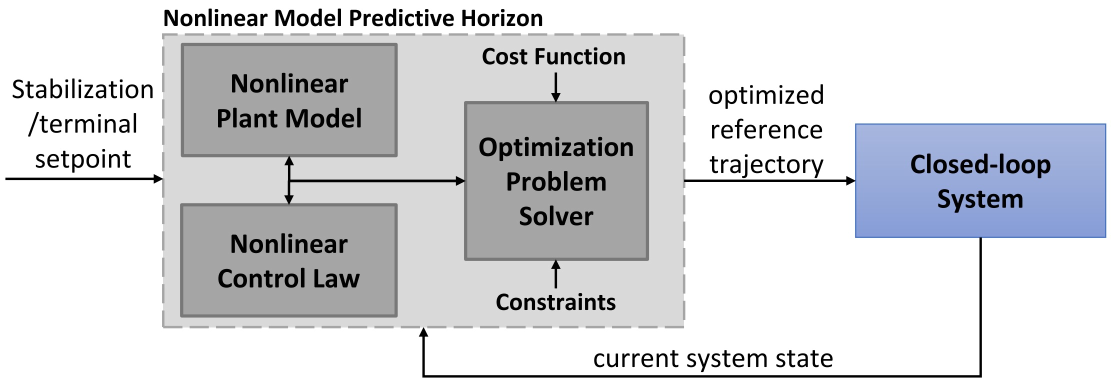

Nonlinear Model Predictive Horizon
Stabilization /terminal setpoint
Nonlinear Plant Model
Nonlinear Control Law
Cost Function
Optimization Problem Solver
Constraints
optimized reference trajectory
Closed-loop System
current system state

Figure 2.1: Nonlinear Model Predictive Horizon architecture. A model of the nonlinear system dynamics is used to perform the optimization process within NMPH (gray box). The resulting optimized reference trajectory is passed to the actual closed-loop system for tracking purposes.

- works in real-time using a specified time horizon,
- uses a nonlinear control law, such as feedback linearization, to reduce the non-convexity of the optimization problem,
- supports state and input constraints of the closed-loop system, and is able to account for environmental constraints such as dynamic obstacles,
- assumes that a stable terminal point is specified, and the state vector of the closed-loop system is available (measured or estimated), and
- provides a combination of stabilization and tracking functionality:
  - *Stabilization*: provides a solution which guides the closed-loop system to a specified setpoint or terminal condition.
  - *Tracking*: generates a smooth reference trajectory for the closed-loop system to track or follow.

The NMPH provides a dynamic parameterization of the reference trajectory. This provides a continually evolving optimal reference trajectory from the current state of the closed-loop system to the terminal setpoint, which respects the system dynamics and environmental constraints such as dynamic obstacles.

The main research contributions presented in this chapter are:

- Formulating a new variant of NMPC (named NMPH), which employs a model of a closed-loop system under nonlinear control law to efficiently solve for the optimal reference trajectory of the target closed-loop system.

- Supporting static and dynamic obstacles within the NMPH, enabling collision-free reference trajectory generation in unknown and dynamic environments.

In the following subsections, a discrete-time formulation of the NMPH algorithm is presented in Section 2.2. The different constraints employed within the NMPH optimization problem are discussed in Section 2.3. The continuous-time representation of NMPH is presented in Section 2.4, then the method of solving the NMPH optimization problem is described in 2.5. Finally, a summary of this chapter is presented in Section 2.6.

2.2 Nonlinear Model Predictive Horizon Algorithm

Consider a discrete-time model of the nonlinear closed-loop system at time instant t_n,

$$x\left(n+1\right) = f\big(x\left(n\right), u\left(n\right)\big) \tag{2.1a}$$

$$\xi\left(n\right) = h\big(x\left(n\right)\big) \tag{2.1b}$$

$$u\left(n\right) = g\left(x\left(n\right), \xi_{ref}(n)\right) \tag{2.1c}$$

where $x\left(n\right) \in X \subseteq \mathbb{R}^{n_x}$ are the system states, $\xi\left(n\right) \in \Xi \subseteq \mathbb{R}^{n_\xi}$ are the system outputs, $u\left(n\right) \in U \subseteq \mathbb{R}^{n_u}$ are the system inputs, and $\xi_{ref}(n) \in \Xi$ is the reference trajectory at time instant t_n. We assume that the system outputs are a subset of the system state vector, $\Xi \subseteq X$, in our case the drone's position and yaw angle. The map

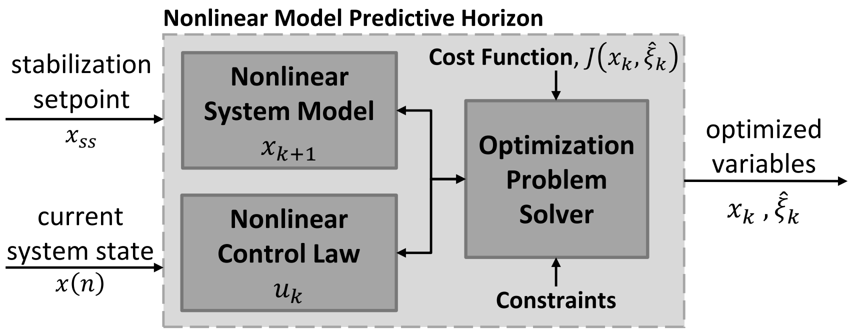

Nonlinear Model Predictive Horizon
stabilization setpoint x_{ss}
current system state $x(n)$
Nonlinear System Model x_{k+1}
Nonlinear Control Law u_k
Cost Function, $J(x_k, \hat{\xi}_k)$
Optimization Problem Solver
Constraints
optimized variables x_k , $\hat{\xi}_k$

Figure 2.2: NMPH structure which combines a model of the nonlinear plant dynamics plus a nonlinear control law within its optimization problem.

$f : X \times U \to X$ represents the discrete-time plant dynamics and $x\,(n+1)$ are the states at the next sampling instant. $g : X \times \Xi \to U$ is the control law that is used to steer the system output to follow the reference trajectory.

A closer look at the NMPH structure is shown in Figure 2.2. The OTP solver uses the plant dynamics (2.1a), output (2.1b) and control law (2.1c) plus any applicable constraints (e.g. obstacles in the environment) to predict the sequence of future states and outputs of the closed-loop system model over the finite time horizon. In order to differentiate between the variables of the actual nonlinear system and its model within the NMPH, the latter uses subscripts as seen in Figure 2.2 and discussed below in Algorithm 2.1.

As shown in Figure 2.2, the sequence x_k provides a prediction of the trajectory of the system (2.1) between its current state $x(n)$ and the setpoint stabilization x_{ss}, which is re-generated each time the OTP is solved. Meanwhile, the sequences ξ_k and $\hat{\xi}_k$ calculated by the OTP are defined as follows:

- ξ_k is the *predicted output trajectory* sequence which represents a subset of the vector entries of the state sequence x_k (in our case the quadrotor's position and yaw angle). It is important to distinguish between $\xi(n)$, the current output of the actual closed-loop system, and ξ_k, the predicted output sequence produced by the OTP solution.

- $\hat{\xi}_k$ is the *estimated reference trajectory* sequence which is calculated by solving an optimization problem inside the NMPH. Using $\hat{\xi}_k$ as the reference trajectory for the actual closed-loop system yields smoother flight paths plus the ability to deal with constraints such as obstacles in the environment. In this way $\xi_{ref}(n)$ in (2.1c) acquires its value from the first predicted point of the $\hat{\xi}_k$ sequence.

As will be illustrated in Figure 2.3, the trajectory sequences ξ_k and $\hat{\xi}_k$ will converge to each other and towards the terminal point. Thus, either of them can be taken as the reference trajectory for the actual closed-loop system.

The predicted future of the closed-loop system's behaviour is optimized at each sampling instant n and over a finite time horizon $k = 0,\ 1,\ \ldots,\ \ N-1$ of length $N \geq 2$. The system is assumed to follow the first j elements of the predicted optimal trajectory sequence until the next sampling instant, at which time the trajectory is re-calculated, and so on. The predicted state x_k and the control input u_k sequences at each time instant n are calculated as

$$x_k\left(0\right) = x\left(n\right), \quad x_{k+1} = f\left(x_k, u_k\right), \quad u_k = g(x_k, \hat{\xi}_k), \quad k = n,\ \ldots,\ n+N-1 \tag{2.2}$$

Now, consider $\mu\left(x_k, u_k\right) : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \rightarrow \mathbb{R}^{n_\xi}$ which is the mapping from the predicted state and control law sequences to the estimated trajectory sequence. This is written as

$$\hat{\xi}_k := \mu\left(x_k, u_k\right), \quad k = n,\ \ldots,\ n+N-1 \tag{2.3}$$

Our objective is to use an optimization methodology to determine the estimated reference trajectory $\hat{\xi}_k$ and the state x_k (whose subset is ξ_k) sequences that both converge to the stabilization setpoint $x_{ss} = x_N^{ref}$. To do this, a cost function $J(x_k, \hat{\xi}_k)$ is chosen to penalize the deviation of the system states from the reference, and the predicted output from the estimated reference trajectory, as shown below in Algorithm 2.1 and the cost function in (2.5).

Algorithm 2.1 NMPH algorithm with stabilizing terminal condition x_{ss}

```
1: Let $n = 0$; measure the initial state $x_0 \leftarrow x(n)|_{n=0}$
2: while $\|x_{ss} - x_0\| \geq \delta$ do
    Solve the following *Optimal Trajectory Problem*,
```

$$\min_{x_k, \hat{\xi}_k} \left(J(x_k, \hat{\xi}_k) := \sum_{k=n}^{n+N-1} L(x_k, \hat{\xi}_k) \; + \; E\left(x_{n+N}\right) \right) \tag{2.4}$$

$$\begin{aligned} \text{subject to} \quad & x_{k=0} = x\left(n\right) & & \text{(2.4a)} \\ & x_{k+1} = f\left(x_k, u_k\right), & k = n, \; \ldots, \; n+N-1, & \quad \text{(2.4b)} \\ & u_k = g(x_k, \hat{\xi}_k), & k = n, \; \ldots, \; n+N-1, & \quad \text{(2.4c)} \\ & x_k \in \mathcal{X}, & k = n, \; \ldots, \; n+N, & \quad \text{(2.4d)} \\ & \hat{\xi}_k \in \mathcal{Z}, \; u_k \in \mathcal{U}, & k = n, \; \ldots, \; n+N-1, & \quad \text{(2.4e)} \\ & \mathcal{O}_i\left(x_k\right) \leq 0 \;, & i = 1, \; 2, \; \ldots, \; p & \quad \text{(2.4f)} \end{aligned}$$

```
    if $x_k \to x_{ss}$ then (estimated trajectory converges to terminal condition)
        $n \leftarrow n + 1$;
    else
        break;
```

In Algorithm 2.1, N is the prediction horizon, $x\left(n\right)$ is the current measured state at a time instant t_n, which represents the initial condition of the OTP, and $L\left(\cdot,\cdot\right)$ and $E\left(\cdot\right)$ are the stage cost function and the terminal cost function, respectively. The constraint sets $\mathcal{X}$, $\mathcal{U}$, and $\mathcal{Z}$ will be defined in Section 2.3, and the inequality constraints $\mathcal{O}_i\left(x_k\right) \leq 0$ allow modeling a set of p static and dynamic obstacles. The optimization process of Algorithm 2.1 is summarized as follows:

1. Measure or estimate the actual closed-loop system's current state $x(n)$.

2. Obtain a prediction of the reference trajectory sequence $\hat{\xi}_k$ for an admissible control input by minimizing the cost function over the prediction horizon subject to the dynamics of the closed-loop system plus state and input constraints.

3. Send the predicted reference trajectory sequence to the closed-loop system for

tracking.

4. Repeat until the system reaches the desired terminal point or encounters an infeasible optimization solution.

Within NMPH, convergence can be achieved by proper choices of the stage cost $L(x, \hat{\xi})$ and the terminal cost $E(x)$ for a setpoint stabilization problem [48]. The requirements for these cost functions are summarized below in Assumption 2.1:

Assumption 2.1 (Cost Function) The stage cost $L(x, \hat{\xi}) : \mathbb{R}^{n_x} \times \mathbb{R}^{n_\xi} \to \mathbb{R}_0^+$ and terminal cost $E(x) : \mathbb{R}^{n_x} \to \mathbb{R}_0^+$ functions introduced in (2.4) have the following properties:

- The stage cost is continuous and bounded from below, meaning that $L(x, \hat{\xi}) \geq \alpha$ for all $(x, \hat{\xi}) \in X \times \Xi \backslash \{0, 0\}$ and $L(0, 0) = 0$.
- The terminal cost is a positive semi-definite function, which is continuously differentiable and satisfies

$$\frac{\partial E}{\partial x} f\big(x, g(x, \hat{\xi})\big) + L(x, \hat{\xi}) \leq 0$$

- The terminal constraint set $\mathcal{E}_N$ is a subset of the state constraint set $\mathcal{X}$ and it is compact.
- For every $x_N \in \mathcal{E}_N$ there exists an estimate of the reference trajectory $\hat{\xi}_k$ and predicted output trajectory ξ_k sequences where both converge to the terminal setpoint, and stay within the terminal region $\mathcal{E}_N$.

Our cost function is chosen to penalize the deviation of states from their reference values, and the deviation of the predicted output trajectory from the estimated reference trajectory, as follows:

$$J(x_k, \hat{\xi}_k) := \sum_{k=n}^{n+N-1} \Big(\|x_k - x_k^{ref}\|_{W_x}^2 + \|\xi_k - \hat{\xi}_k\|_{W_\xi}^2 \Big) + \|x_N - x_N^{ref}\|_{W_N}^2 \tag{2.5}$$

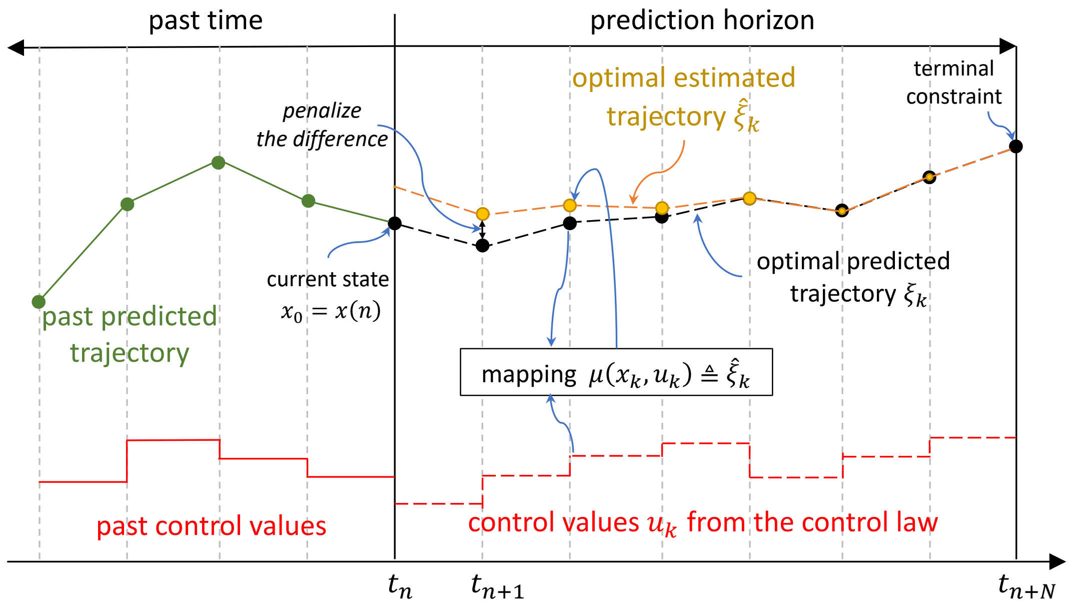

past time
prediction horizon
penalize the difference
optimal estimated trajectory $\hat{\xi}_k$
terminal constraint
current state $x_0 = x(n)$
past predicted trajectory
optimal predicted trajectory ξ_k
mapping $\mu(x_k, u_k) \triangleq \hat{\xi}_k$
past control values
control values u_k from the control law
t_n
t_{n+1}
t_{n+N}

Figure 2.3: NMPH process at time t_n, which predicts the optimal trajectory until time t_{n+N}. The difference between the predicted output ξ_k and the estimated reference $\hat{\xi}_k$ trajectories is penalized to ensure their convergence towards each other.

where $x_k^{ref} \in X$ is the reference states sequence used in the optimization problem. The terminal cost $\|x_N - x_N^{ref}\|^2_{W_N}$ with its weighting matrix W_N steers the system towards the stabilization setpoint $x_{ss} = x_N^{ref}$, while the stage cost function $L(x_k, \hat{\xi}_k)$ uses the weighting matrices W_x and W_ξ to penalize deviations of the states and outputs, respectively. The entries of the weighting matrices are selected to adjust the relative importance of these three factors for the optimization problem.

A visual interpretation of the NMPH process can be seen in Figure 2.3, where the path planning task is to guide the closed-loop system described by (2.1) to follow a predicted trajectory from $x(n)$ (at time instant t_n) to x_{ss} (at a future time t_{n+N}) while minimizing the cost function (2.5) and respecting the system's state and input constraints.

Some of the features of the predicted trajectory sequences are:

- The first j elements of the reference trajectory sequence $\hat{\xi}_k$ are passed to the closed-loop system, which is different from the NMPC control problem where only the first element of the predicted control sequence $u^*(n)$ is used. This provides some flexibility in choosing the rate at which the OTP is solved, which addresses the computation time issue of solving a Nonlinear Program (NLP).

- Thanks to recent advancements in computing, specifically graphics processing units (GPUs), the computations required for optimization problems can be performed very quickly, meaning solving the NLP problem for OTP or even OCP can be done in real-time. Irrespective of this, OTP has an advantage over OCP since the computational power requirement can be controlled by adjusting the rate of solving the optimization problem while allowing the vehicle to track the first j elements of the estimated reference trajectory.

- While the tailing $N - j$ elements of the reference trajectory sequence are discarded, the entire trajectory is still required to be calculated over the prediction horizon. The reason for this is that optimizing over the full horizon ensures a smooth trajectory from the initial state to the terminal setpoint.

- The optimization problem is solved iteratively using a reliable and accurate optimization approach based on the multiple shooting method and sequential quadratic programming.

2.3 NMPH Constraints

Support for constraints within the NMPH algorithm provides full control over the optimization problem. The constraints can apply to the state, input and output trajectories, and also model dynamic obstacles.

State constraints belong to the subset $\mathcal{X} \subseteq X$ while outputs, which are assumed to be a subset of the state vector entries, are the subset $\mathcal{Z} \subseteq \mathcal{X}$. $\mathcal{U}\left(x, \hat{\xi}\right) \subseteq U$ is defined by physical input constraints in the system, for instance due to actuator limits.

The objective of introducing the constraint sets are to ensure that the optimized trajectories are bounded and lie within their allowable ranges. The following assumptions regarding the constraint sets are made [12, 48]:

Assumption 2.2 (Closed and Bounded Sets) The constraint sets of the state $\mathcal{X}$ and the reference trajectory $\mathcal{Z}$ are closed, and the control constraint set $\mathcal{U}$ is compact.

Assumption 2.3 (Differentiability and Lipschitz) The system dynamics $f(x,u)$: $\mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$ is continuously differentiable for all $(x,u) \in \mathcal{X} \times \mathcal{U}$. Also, $f(x,u)$ and the reference trajectory mapping $\mu(x,u) : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_\xi}$ are considered to be locally Lipschitz.

Assumption 2.4 (Uniqueness) For any element of the estimated reference trajectory $\hat{\xi}$ resulting from a control input u and any possible initial states $x_0 \in \mathcal{X}$, the system dynamics produce a unique and continuous solution.

Assumption 2.5 (Viability) For each state $x \in \mathcal{X}$ and estimated reference trajectory $\hat{\xi} \in \mathcal{Z}$ there exists a control $u = g(x, \hat{\xi}) \in \mathcal{U}$ such that $f(x,u) \in \mathcal{X}$.

Taking Assumptions 2.2–2.4, we can make the following definition:

Definition 2.1 (Admissibility) *In the discrete-time OTP, consider the system dynamics* $x_{k+1} = f(x_k, u_k)$ *and the control law* $u_k = g(x_k, \hat{\xi}_k)$*, which maps the state to the estimated trajectory as* $\mu(x_k, u_k) := \hat{\xi}_k \in \mathcal{Z}$*, with the constraint sets for state* $\mathcal{X} \subseteq X$*, control* $\mathcal{U}(x, \hat{\xi}) \subseteq U$*, and reference trajectory* $\mathcal{Z} \subseteq X$*.*

- *The system states* $x_k \in \mathcal{X}$ *and the estimated reference trajectory* $\hat{\xi}_k \in \mathcal{Z}$ *are called admissible states and trajectories, respectively, and the control* $u_k = g(x_k, \hat{\xi}_k) \in \mathcal{U}$ *are called admissible control values for* x_k *and* $\hat{\xi}_k$*. Hence, the admissible set can be defined as*

$$\mathcal{Y} := \left\{ (x_k, \hat{\xi}_k, u_k) \in X \times \Xi \times U \mid x_k \in \mathcal{X}, \hat{\xi}_k \in \mathcal{Z}, u_k = g(x_k, \hat{\xi}_k) \in \mathcal{U} \right\} \quad (2.6)$$

- *The control sequence u_k and its associated estimated reference trajectory $\hat{\xi}_k$ and state sequence x_k from the time t_0 of the initial value $x_0 \in \mathcal{X}$ up to time t_N of the setpoint stabilization value x_N, are admissible if $(x_k, \hat{\xi}_k, u_k) \in \mathcal{Y}$ for $k = 0,\ \ldots,\ N-1$ and $x_N \in \mathcal{X}$.*
- *The control law is called admissible if $g(x_k, \hat{\xi}_k) \in \mathcal{U}$ for all $x_k \in \mathcal{X}$ and $\hat{\xi}_k \in \mathcal{Z}$.*
- *The estimated reference trajectory is called admissible if $\mu(x_k, u_k) \in \mathcal{Z}$ for all $x_k \in \mathcal{X}$ and $u_k \in \mathcal{U}$.*

A feasible problem is defined as an optimization problem in which there exists at least one set of solutions that satisfies all the constraints [12]. Based on Assumptions 2.2-2.5 and the admissibility Definition 2.1, the feasibility of the OTP is determined by Theorem 2.1.

Theorem 2.1 (Feasibility) *If the OTP is feasible for an initial condition x_0 and the cost function for a setpoint stabilization problem with associated constraint sets satisfy Assumptions 2.2-2.5, then the OTP is recursively feasible, meaning the state converges to the stabilizing terminal point x_N, and both the estimated reference trajectory $\hat{\xi}_k$ and predicted output trajectory ξ_k sequences converge toward the terminal stabilization setpoint under sampled-data NMPH.*

Proof: The solution of the OTP is feasible for an initial value $x_0 \in \mathcal{X}$ to a stabilizing terminal point $x_N \in \mathcal{X}$ if the sets over which we optimize are non-empty. The viability considered in Assumption 2.5 for $\mathcal{Z}$ and $\mathcal{X}$ implies that the OTP is feasible for each initial state x_0 and consequently ensures that the control $g(x_k, \hat{\xi}_k)$ is properly defined for each $x \in \mathcal{X}$ and $\hat{\xi} \in \mathcal{Z}$. Since the OTP is performed with respect to admissible predicted state trajectory and control law sequences (as stated in Definition 2.1), the future behavior of the system is consequently feasible. $\square$

It is important to note that the solution of the OTP is viable in the case of a stabilizing terminal constraint, meaning the NMPH problem is confined to feasible subsets since the terminal constraint is viable. The closed-loop system embedded within NMPH satisfies the desired constraints, which will lead to a feasible solution in the OTP. The stability of a feasible solution is governed by Theorem 2.2.

Theorem 2.2 (Stability) *Assume that the OTP within Algorithm 2.1 satisfies Assumption 2.1 and has a feasible solution as determined by Theorem 2.1. Then the optimized solution leads to a stable prediction of the system state x_k and estimated reference trajectory $\hat{\xi}_k$.*

Proof: Assume that at any time instant t_i, $i \in [n, ..., n+N-2]$, x_i^* and $\hat{\xi}_i^*$ are the optimal solutions of the OTP in Algorithm 2.1, with their associated control value u_i^*. A Lyapunov-like function is defined as

$$V(x_i, \hat{\xi}_i) = \min_{x_i^*, \hat{\xi}_i^*} J(x_i, \hat{\xi}_i) = J(x_i^*, \hat{\xi}_i^*) \tag{2.7}$$

The cost function in (2.4) guarantees a positive semi-definite Lyapunov-like candidate [81], meaning that $0 \leq V(x_i, \hat{\xi}_i) < \infty$, which can be written at time t_i as

$$V(x_i, \hat{\xi}_i) = E\left(x_{n+N}^*\right) + \sum_{i=n}^{n+N-1} L(x_i^*, \hat{\xi}_i^*) \tag{2.8}$$

Considering the solution at a subsequent time $t_{i+\delta}$, the feasible solution of the cost function is

$$J(x_{i+\delta}, \hat{\xi}_{i+\delta}) = E\left(x_{n+N+\delta}\right) + L(x_{n+N-1+\delta}, \hat{\xi}_{n+N-1+\delta}) + \sum_{i=n+\delta}^{n+N-1} L\left(x_i^*, \hat{\xi}_i^*\right) \tag{2.9}$$

Since $V(x_{i+\delta}, \hat{\xi}_{i+\delta}) \leq J(x_{i+\delta}, \hat{\xi}_{i+\delta})$, we have

$$\begin{aligned}
V(x_{i+\delta}, \hat{\xi}_{i+\delta}) - V(x_i, \hat{\xi}_i) &\leq J(x_{i+\delta}, \hat{\xi}_{i+\delta}) - V(x_i, \hat{\xi}_i) \\
&\leq E\left(x_{n+N+\delta}\right) + L(x_{n+N-1+\delta}, \hat{\xi}_{n+N-1+\delta}) \\
&\quad + \Sigma_{i=n+\delta}^{n+N-1} L(x_i^*, \hat{\xi}_i^*) - \left(E(x_{n+N}^*) + \Sigma_{i=n}^{n+N-1} L(x_i^*, \hat{\xi}_i^*)\right) \\
&\leq L(x_{n+N-1+\delta}, \hat{\xi}_{n+N-1+\delta}) + E\left(x_{n+N+\delta}\right) \\
&\quad - L(x_n^*, \hat{\xi}_n^*) - E(x_{n+N}^*)
\end{aligned}$$

where $E\left(x_{n+N+\delta}\right) - E\left(x^*_{n+N}\right) + L(x_{n+N-1+\delta}, \hat{\xi}_{n+N-1+\delta}) \leq 0$ based on the inequality considered in Assumption 2.1. Therefore,

$$V(x_{i+\delta}, \hat{\xi}_{i+\delta}) - V(x_i, \hat{\xi}_i) \leq -L\big(x^*_n, \hat{\xi}^*_n\big)$$

this implies that the rate of change of the Lyapunov-like function is decreasing with time. Hence, the solution of the OTP problem in Algorithm 2.1 converges asymptotically to the terminal setpoint. □

To perform safe navigation, it is necessary to include the obstacle constraints within the optimization problem. The inequality constraint presented in (2.4f) accounts for the space that the predicted trajectory should avoid. For instance, the obstacle constraints $\mathcal{O}_i\left(x_k\right) \leq 0$ are defined as

$$\|x^{pos}_k - o_i\| - \epsilon_i \leq 0 \ , \qquad i = 1,\ 2,\ \ldots,\ p$$

where $x^{pos}_k \in \mathbb{R}^3$ is the predicted vehicle position over the prediction horizon, $o_i \in \mathbb{R}^3$ are the position of the obstacles centers, and ϵ_i represents the safety distance between the i^{th} obstacle and the vehicle, which accounts for the drone and obstacle sizes including a safety tolerance.

2.4 Continuous-time Representation of NMPH

The discrete-time representation (2.1) shown in Section 2.2 is presented to understand the problem formulation analysis and NMPH development, with the optimization being performed in the discrete-time domain as in sampled-data MPC [82]. Conversely, a continuous-time representation is important for NMPH implementation since our chosen optimization algorithm (ACADO [83]) has the ability to discretize the system equations.

Consider the continuous-time nonlinear closed-loop system

$$\begin{aligned} \dot{x}(t) &= f\big(x(t), u(t)\big) \\ \xi(t) &= h\big(x(t)\big) \\ u(t) &= g\big(x(t), \xi_{ref}(t)\big) \end{aligned} \tag{2.10}$$

where $x(t) \in X \subseteq \mathbb{R}^{n_x}$ are the system states and $u(t) \in U \subseteq \mathbb{R}^{n_u}$ are the system inputs. $\xi(t) \in \Xi \subseteq \mathbb{R}^{n_\xi}$ are the system outputs, assumed to be a subset of the state vector $x(t)$, such that $\Xi \subseteq X$ (e.g. for a drone vehicle the outputs are the 3D position and heading). The map $f : X \times U \to X$ represents the system dynamics and $g : X \times \Xi \to U$ is the control law that is used to make the system output follow the reference trajectory.

Remark 2.3 *The character $\tilde{(\cdot)}$ is added to the variables used within the continuous-time optimization problem to differentiate them easily from the actual closed-loop system variables.*

The optimization problem presented in Algorithm 2.1 can be rewritten in the continuous-time domain

Algorithm 2.2 Continuous-time NMPH algorithm

```
1: Let $t_n$, $n = 0, 1, 2, \cdots$ represent successive sampling times; set $n = 0$
2: while $\|x_{ss} - x(t_n)\| \geq \delta$ do
```

$$\min_{\tilde{x},\hat{\xi}_{ref}} \left(\int_{t_n}^{t_n+T} L\big(\tilde{x}(\tau), \hat{\xi}_{ref}(\tau)\big)\, d\tau \;+\; E\left(\tilde{x}(t_n+T)\right) \right) = \tag{2.11}$$

$$\min_{\tilde{x},\hat{\xi}_{ref}} \left(\int_{t_n}^{t_n+T} \Big(\|\tilde{x}(\tau) - x_{ss}\|^2_{W_x} + \|\tilde{\xi}(\tau) - \hat{\xi}_{ref}(\tau)\|^2_{W_\xi} \Big)\, d\tau + \|\tilde{x}(t_n+T) - x_{ss}\|^2_{W_T} \right)$$

$$\begin{aligned} \text{subject to}\quad & \tilde{x}(t_n) = x(t_n), && \text{(2.11a)}\\ & \dot{\tilde{x}}(\tau) = f\left(\tilde{x}(\tau), \tilde{u}(\tau)\right), && \text{(2.11b)}\\ & \tilde{u}(\tau) = g\big(\tilde{x}(\tau), \hat{\xi}_{ref}(\tau)\big), && \text{(2.11c)}\\ & \tilde{x}(\tau) \in \mathcal{X},\quad \tilde{u}(\tau) \in \mathcal{U},\quad \hat{\xi}_{ref}(\tau) \in \mathcal{Z}, && \text{(2.11d)}\\ & \mathcal{O}_i(\tilde{x}) \leq 0\ ,\quad i = 1,\ 2,\ \ldots,\ p. && \text{(2.11e)} \end{aligned}$$

```
    if $\tilde{x} \to x_{ss}$ then
        $n \leftarrow n + 1$;
    else
        break;
```

The continuous-time optimization problem is solved over the finite time horizon $\tau \in [t_n, t_n + T]$. As discussed in Section 2.2, the closed-loop system will be asked to track a portion of the resulting reference trajectory running from t_n to $t_n + t_j$, where $t_j < T$ and where t_j can be adjusted online to affect the trajectory generation and tracking performance and control the computational power required by the optimization.

2.5 NMPH Optimization Problem

The optimization problem of NMPH uses a model of the nonlinear system dynamics and nonlinear control law to solve for an optimal reference trajectory for the resulting closed-loop system. The OTP, which is written in the continuous-time domain and presented in (2.11), is solved using an efficient direct multiple shooting technique [84]. The solver discretizes the system dynamics, control law, and state and input constraints over the prediction horizon into $k = n, ..., \ n + N$ at each time instant t_n as discussed in (2.4). An overview of the process from the optimization problem formulation to the trajectory generation is given in Figure 2.4.

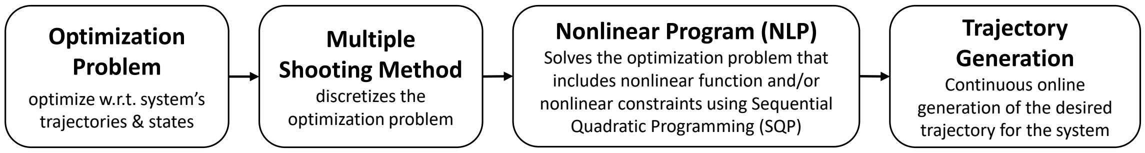

Optimization Problem
optimize w.r.t. system's trajectories & states
Multiple Shooting Method
discretizes the optimization problem
Nonlinear Program (NLP)
Solves the optimization problem that includes nonlinear function and/or nonlinear constraints using Sequential Quadratic Programming (SQP)
Trajectory Generation
Continuous online generation of the desired trajectory for the system

Figure 2.4: Optimization problem for NMPH

The optimization problem operates on the dynamics of a closed-loop system which may not be convex. Therefore, the optimization problem is solved iteratively using a Sequential Quadratic Programming (SQP [85]) approach of splitting the problem into a sequence of subproblems, each of which solves for a quadratic objective function subject to linearized constraints about their operating point using the qpOASES solver [86].

To ensure local convergence of the SQP, the quadratic function of the subproblem has to be bounded within a feasible region of the optimization problem sets. Starting from an initial condition x_0, the optimization variables should be sufficiently close to the terminal condition x_{ss}; then the sequence x_k generated by the NMPH converges to the terminal condition at a quadratic rate.

2.6 Summary

In this chapter, we proposed a novel reference trajectory generation framework for a nonlinear closed-loop system based on the NMPC approach. The proposed formulation, called NMPH, applies a nonlinear control law to the nonlinear plant model, aiming a closed-loop dynamics model with decreased non-convexity used by the online optimization problem to generate feasible and optimal reference trajectories for the actual closed-loop system. The proposed NMPH algorithm supports both static and dynamic obstacles, enabling trajectory generation in continuously changing environments.

Chapter 3

Nonlinear Model Predictive Horizon with Feedback Linearization Control

3.1 Introduction

This chapter presents the development of an algorithm which allows the generation of optimal trajectories based on the NMPH approach by using a closed-loop system model consisting of the nonlinear plant connected to a state feedback linearization (FBL) control law. This proposed formulation is denoted by NMPH-FBL. The purpose of employing FBL within the NMPH is to reduce or eliminate the non-convexity of the optimization problem relative to working directly with the nonlinear plant dynamics as in standard NMPC. The nonlinear system studied in this work, a drone vehicle, is a Multi-Input Multi-Output (MIMO) system. We thus start by reviewing the method of feedback linearization for a class of MIMO systems, as presented next.

In this chapter, the research contributions are:

- Formulating the NMPH design with a feedback linearization control law, aiming on decreasing the non-convexity used by the online optimization problem to efficiently generate feasible and optimal reference trajectories for the actual closed-loop system.

- Designing a feedback linearization control law for a model augmented with integral states to achieve more robust performance in the presence of modeling uncertainties.

- Implementing support for static and dynamic obstacles within the NMPH, enabling trajectory generation in continuously changing environments.
- Validating the ability of the system to generate optimal trajectories for the quadrotor vehicle using realistic various flight environment simulation scenarios.

The remainder of this chapter is structured as follows. Section 3.2 presents the formulation of NMPH design with the the state feedback linearization control law. The application of the algorithm to a drone vehicle is presented in Section 3.3. Section 3.4 presents different simulation scenarios to evaluate and validate the proposed approach. Concluding remarks are given in Section 3.5.

3.2 NMPH Closed-loop Form with Feedback Linearization Control Law

Consider a MIMO nonlinear control-affine system of the form

$$\dot{x} = f(x) + \sum_{i=1}^{n_u} \mathfrak{g}_i(x)\ u_i \triangleq f(x) + G(x)\,u \tag{3.1}$$

in which $x \in \mathbb{R}^{n_x}$, and f, $\mathfrak{g}_1, \ldots, \mathfrak{g}_{n_u}$ are smooth vector fields in $\mathbb{R}^{n_x}$. $G(x)$ is an $n_x \times n_u$ matrix and its rank at $x = 0$ is rank $G(0) = n_u$. For notation simplicity in the following sections, take $n_x \equiv n$ and $n_u \equiv m$.

Prior to feedback linearization analysis, the following theorems and definitions are presented in the context of differential geometry.

Definition 3.1 (Diffeomorphism) *A diffeomorphism is a differentiable map φ between two manifolds $\mathcal{M}$ and $\mathcal{N}$, such that $\varphi : \mathcal{M} \to \mathcal{N}$ is one-to-one and onto (bijective), and its differentiable inverse map $\varphi^{-1} : \mathcal{N} \to \mathcal{M}$ is bijective as well. φ is called a C^ω diffeomorphism if it is ω times continuously differentiable. If $\omega = \infty$, then φ is called a C^∞ smooth map [87].*

A change of coordinates can be defined globally or locally. A map $\xi : \mathbb{R}^n \to \mathbb{R}^n$ is called a global diffeomorphism between two coordinates if, and only if, the determinant $\det \frac{\partial \xi}{\partial x} \neq 0$ for all $x \in \mathbb{R}^n$, and $\lim_{\|x\| \to \infty} \|\xi(x)\| = \infty$ [88]. For a local change of coordinates, let $\mathcal{U}$ be an open subset of $\mathbb{R}^n$ with $\xi : \mathcal{U} \to \mathbb{R}^n$. If $\det \frac{\partial \xi}{\partial x} \neq 0$ at some $x \in \mathcal{U}$, then there exists $\mathcal{V} \subset \mathcal{U}$, which is an open set that includes x such that the map $\xi : \mathcal{V} \to \mathcal{V}(\xi)$ is a diffeomorphism [87].

A specific class of nonlinear systems can be transformed into a linear state feedback controllable form by satisfying the conditions to be presented in Theorem 3.1. To facilitate our understanding of this process, we first define a non-singular state feedback transformation and controllability indices in Definitions 3.2 and 3.3, respectively [87]:

Definition 3.2 (Non-singular state feedback transformation) *Consider $\mathcal{V}_0 \subset \mathcal{U}_0$ which is a neighborhood of the origin. On the one hand, the* non-singular state feedback *is defined as:*

$$u = \beta(x) + D(x)^{-1} v \tag{3.2}$$

where the function $\beta(x)$ is smooth from $\mathcal{V}_0$ into $\mathbb{R}^n$ and $\beta(0) = 0$. $D(x)^{-1}$ is the inverse of a non-singular $m \times m$ matrix in $\mathcal{V}_0$.
On the other hand, a local diffeomorphism *in $\mathcal{V}_0$ (defined by: $\xi = T(x)$, $T(0) = 0$), exists if, and only if, in $\mathcal{U}_0$ the distributions $\mathcal{G}_l = span\left\{ad_f^j \mathfrak{g}_i : \ 1 \leq i \leq m, \quad 0 \leq j \leq l\right\}$ are involutive and of constant rank for $0 \leq l \leq n-2$, and the rank of the distribution of $n-1$ is rank $\mathcal{G}_{n-1} = n$.*
A transformation that contains a non-singular state feedback *and a* local diffeomorphism *is called a non-singular state feedback transformation.*

Definition 3.3 (Controllability Indices) *Controllability indices r_i, $i = 1, \ldots, m$ associated with control-affine systems of the form (3.1) are calculated as*

$$r_i = \text{card}\{m_j \geq i : j \geq 0\} \tag{3.3}$$

with

$$m_0 = \text{rank} \ \ \mathcal{G}_0,$$
$$m_k = \text{rank} \ \ \mathcal{G}_k - \text{rank} \ \ \mathcal{G}_{k-1}, \qquad k = 1, \ldots, n-1.$$

Employing Definitions 3.2 and 3.3, the sufficient conditions for feedback linearization and the form of its feedback transformation are given in Theorem 3.1 [87].

Theorem 3.1 (MIMO Feedback Linearization) *A system of the form (3.1) can be locally transformed by means of a non-singular state feedback transformation as given in Definition 3.2 into a linear Brunovsky controller form in* $\mathcal{U}_0$ *if:*

i) The controllability indices satisfy the condition $r_1 \geq \ldots \geq r_m$*;*

ii) The distributions $\mathcal{G}_{r_i-2}$ *are involutive and of constant rank for* $i = 1, \ldots, m$*;*

iii) The rank of the distribution $\mathcal{G}_{r_1-1}$ *is* $\operatorname{rank}\ \mathcal{G}_{r_1-1} = n$.

In this case there are smooth functions $\{\varphi_i(x) : \langle d\varphi_i, \mathcal{G}_{r_i-2}\rangle = 0,\ j \geq i,\ i = 1, \ldots, m\}$ *that form a non-singular matrix* $D(x)$ *in* $\mathcal{V}_0$,

$$D(x) = \begin{bmatrix} \langle d\varphi_1, ad_f^{r_1-1}\mathfrak{g}_1\rangle & \ldots & \langle d\varphi_1, ad_f^{r_1-1}\mathfrak{g}_m\rangle \\ \vdots & \ddots & \vdots \\ \langle d\varphi_m, ad_f^{r_m-1}\mathfrak{g}_1\rangle & \ldots & \langle d\varphi_m, ad_f^{r_m-1}\mathfrak{g}_m\rangle \end{bmatrix}, \tag{3.4}$$

and consequently the non-singular state feedback transformation is

$$v = \begin{bmatrix} L_f^{r_1}\varphi_1 \\ \vdots \\ L_f^{r_m}\varphi_m \end{bmatrix} + \begin{bmatrix} L_{\mathfrak{g}_1}L_f^{r_1-1}\varphi_1 & \ldots & L_{\mathfrak{g}_m}L_f^{r_1-1}\varphi_1 \\ \vdots & \ddots & \vdots \\ L_{\mathfrak{g}_1}L_f^{r_m-1}\varphi_m & \ldots & L_{\mathfrak{g}_m}L_f^{r_m-1}\varphi_m \end{bmatrix} u\ , \qquad \xi = \begin{bmatrix} \varphi_1 \\ \vdots \\ L_f^{r_1-1}\varphi_1 \\ \vdots \\ \varphi_m \\ \vdots \\ L_f^{r_m-1}\varphi_m \end{bmatrix} \tag{3.5}$$

In Theorem 3.1, $L_f\varphi = \langle d\varphi, f\rangle$ is the Lie derivative which can be realized as a directional derivative of the smooth function φ along the smooth vector field f, and $ad_f^r\mathfrak{g} = [f, ad_f^{r-1}\mathfrak{g}]$ is the iterated Lie bracket between the vector fields f and g.

Using the feedback linearization control law yields a locally stable closed-loop system model within the NMPH structure. One of the main motivations for choosing the feedback linearization approach in our work was to reduce nonlinearities. Since the NMPH works with a closed-loop system model, the optimization problem will have a reduced non-convexity as compared to working directly with the nonlinear plant model as in NMPC. This will lead to better optimization performance in terms of computation time to find feasible solutions.

Modeling errors, system uncertainties, and external disturbances can affect the performance of the state feedback linearization control law. For instance, a constant wind gust applied to the drone while following a trajectory will lead to an offset in the corresponding position outputs. The baseline feedback linearization controller is unable to compensate for this type of offset, which will consequently degrade the accuracy of the predicted states and reference trajectories produced by the NMPH. To overcome this issue, an extension of the state feedback linearization can be achieved by augmenting the system model with integral states of the position vector and yaw angle of the drone. The validity of using this extension is demonstrated by using Theorem 3.1 in the feedback linearization design Section 3.3.2.

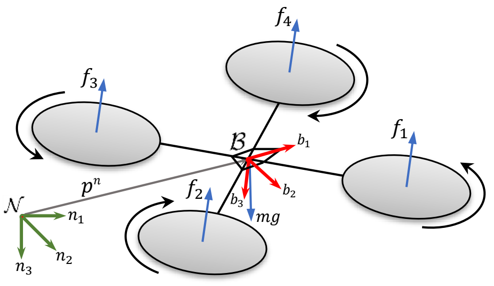
3.3 Application of NMPH-FBL to a Drone Vehicle

In this section, the NMPH is developed for a drone vehicle based on the feedback linearization control law.

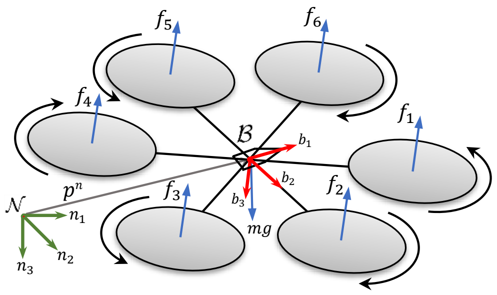
3.3.1 System Model

In this section, both a quadcopter and a hexacopter system are modeled as rigid bodies with lumped force and torque inputs at each rotor. A standard nonlinear rigid-body dynamics vehicle model is adopted in this work from [89]. For simplicity, drag forces, rotor gyroscopic effects, and propeller dynamics are not included in the model. The rigid-body dynamics are formulated using the Newton-Euler equations [90].

The rigid-body kinematics and dynamics are developed using two reference frames, which are the fixed navigation frame $\mathcal{N}$ and the moving body frame $\mathcal{B}$ (fixed to the drone's Center of Gravity, CG). The basis of both frames are selected based on the North, East, and Down (NED) directions in a local tangent plane as the orthonormal vector sets $\{n_1, n_2, n_3\}$ and $\{b_1, b_2, b_3\}$ for the navigation and body frames, respectively. The two basis for each drone are depicted in Figure 3.1.



(a)



(b)

Figure 3.1: Reference frames used for (a) quadcopter and (b) hexacopter vehicles.

In general, any configuration of a rigid-body in space belongs to the Special Euclidean group SE(3), the product space of the rigid-body orientation and position $(R_{nb}, p^n) \in \mathrm{SO}(3) \times \mathbb{R}^3 = \mathrm{SE}(3)$, where the Special Orthogonal group SO(3) is defined as $\mathrm{SO}(3) = \{R \in \mathbb{R}^{3\times 3} \mid RR^T = R^T R = I,\ \det(R) = +1\}$, and the rotation matrix of $\mathcal{B}$ with respect to $\mathcal{N}$ is denoted as $R_{nb} \in \mathrm{SO}(3)$. To represent the orientation, the ZYX Euler angle parameterization is employed. Based on the roll-pitch-yaw Euler angles $\eta = [\phi, \theta, \psi]^T$, the rotation matrix can be written as

$$R = R_{nb} = R_\psi R_\theta R_\phi = \begin{bmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{bmatrix} \tag{3.6}$$

where $s_{(\cdot)} = \sin(\cdot)$ and $c_{(\cdot)} = \cos(\cdot)$. Note $t_{(\cdot)} = \tan(\cdot)$ will be used in (3.9).

The most prominent issue of using Euler angles is the singularity when the parameterization loses injectivity at $\theta = \pi/2 + k\pi$, $k \in \mathbb{Z}$. A recent work by [91] tackles this issue by using Non-Euclidean rotation groups in the NMPC, but in this work the problem is addressed simply by adding state constraints within the NMPH optimization problem in (2.11d).

The rigid-body kinematics and dynamics are modelled as shown below,

$$\begin{aligned} \dot{p}^n &= v^n \\ m\dot{v}^n &= -\bar{u}Rn_3 + gn_3 \\ \dot{R} &= R\,\mathcal{S}(\omega^b) \\ J\dot{\omega}^b &= -\mathcal{S}(\omega^b)J\omega^b + \tau^b \end{aligned} \tag{3.7}$$

where $p^n, v^n \in \mathbb{R}^3$ are the vehicle's position and velocity with respect to the inertial frame $\mathcal{N}$, respectively. m is the mass of the drone, $g = 9.81m/s^2$ is the gravitational acceleration, and the vehicle's mass moment of inertia matrix is assumed to be diagonal with $J = \text{diag}([J_1, J_2, J_3])$. The system input vector is $[\bar{u}, \tau^b]^T$, where $\bar{u} = \sum_{i=1}^{4} f_i > 0$ is the total generated thrust in the direction of $-b_3$, and $\tau^b = [\tau^{b1}, \tau^{b2}, \tau^{b3}]^T$ are the torques created by the rotors about the body frame axes. ω^b and $\dot{\omega}^b$ are the angular velocity and acceleration vectors, respectively. The operator $\mathcal{S}(\cdot) : \mathbb{R}^3 \to \mathfrak{so}(3)$ maps a vector to a skew-symmetric matrix such that $\mathcal{S}(a)b = a \times b$ for $a, b \in \mathbb{R}^3$.

It is important to mention that each of the vehicle configurations (quadcopter and hexacopter) transforms the rotors' thrusts and torques to the system input vector $[\bar{u}, \tau^b]^T$ differently. These transformations are assumed to be performed in the onboard flight controller, and consequently both configurations are represented by the same dynamics (3.7). Hence, the proposed algorithm development is the same for both configurations.

3.3.2 Development of Feedback Linearization Control Law on a Drone Vehicle

The system represented in (3.7) has to be described in a nonlinear control-affine form as shown in (3.1). The state and input vectors are,

$$\begin{aligned} x &= \left[(p^n)^T, (v^n)^T, (\eta)^T, (\omega^b)^T\right]^T \in \mathbb{R}^{12} \\ u &= \left[\bar{u}, (\tau^b)^T\right]^T \in \mathbb{R}^4 \end{aligned} \tag{3.8}$$

The control affine form is as described below,

$$\dot{x} = f(x) + \sum_{i=1}^{4} \mathfrak{g}_i(x) u_i \triangleq f(x) + G(x) u \tag{3.9}$$

where,

$$f(x) = \begin{bmatrix} x_4 \\ x_5 \\ x_6 \\ 0 \\ 0 \\ g \\ x_{10} + s_{x_7} t_{x_8} x_{11} + c_{x_7} t_{x_8} x_{12} \\ c_{x_7} x_{11} - s_{x_7} x_{12} \\ \frac{s_{x_7}}{c_{x_8}} x_{11} + \frac{c_{x_7}}{c_{x_8}} x_{12} \\ \left(\frac{J_2 - J_3}{J_1}\right) x_{11} x_{12} \\ \left(\frac{J_3 - J_1}{J_2}\right) x_{10} x_{12} \\ \left(\frac{J_1 - J_2}{J_3}\right) x_{10} x_{11} \end{bmatrix}, G(x) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -\frac{1}{m}\left(c_{x_7} s_{x_8} c_{x_9} + s_{x_7} s_{x_9}\right) & 0 & 0 & 0 \\ -\frac{1}{m}\left(c_{x_7} s_{x_8} s_{x_9} - s_{x_7} c_{x_9}\right) & 0 & 0 & 0 \\ -\frac{1}{m} c_{x_7} c_{x_8} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \frac{1}{J_1} & 0 & 0 \\ 0 & 0 & \frac{1}{J_2} & 0 \\ 0 & 0 & 0 & \frac{1}{J_3} \end{bmatrix}$$

As stated in Theorem 3.1, the controllability indices $\{r_1, r_2, r_3, r_4\}$ of the system need to be found first to verify whether the system is state feedback linearizable or not. To find them, first $m_j, 0 \leq j \leq 4$ is computed based on the distributions $\mathcal{G}_l = \text{span}\left\{ad_f^j \mathfrak{g}_i : \ 1 \leq i \leq 4, \ \ 0 \leq j \leq l\right\}$ for $l = 0, ..., 10$. Therefore, the calculated controllability indices are $\{5, 2, 2, 2\}$. These need to be checked against the conditions of Theorem 3.1. $\mathcal{G}_3$ is found to be not involutive and $\dim \mathcal{G}_4 = 11 \neq n$. Hence, the system is not state feedback linearizable as conditions (ii) and (iii) are not satisfied. It can also be noted that $\sum_{i=1}^{4} r_i = 11 \neq n$.

From above, the system is not locally state feedback linearizable, meaning it cannot be transformed into a linear controllable system written in Brunovsky controller form. The system states and inputs are thus reformulated by augmenting the state vector with two additional states, which are the thrust $x_{13} = \bar{u}$ and its rate $x_{14} = \dot{\bar{u}}$, and replacing the thrust by $\ddot{\bar{u}}$ in the input vector. The same approach was successfully validated in [92–94]. Furthermore, the system is extended by including the integral states ζ defined by $\dot{\zeta}^p = p^n$, $\dot{\zeta}^\psi = \psi$ as shown below:

$$\begin{aligned} x &= \left[(p^n_{3\times1})^T, (v^n_{3\times1})^T, (\eta_{3\times1})^T, (\omega^b_{3\times1})^T, \bar{u}, \dot{\bar{u}}, (\zeta^{p^n}_{3\times1})^T, \zeta^\psi\right]^T \in \mathbb{R}^{18} \\ u &= \left[\ddot{\bar{u}}, (\tau^b_{3\times1})^T\right]^T \in \mathbb{R}^4 \end{aligned} \tag{3.10}$$

Based on the extended system's vectors, the presented state space control-affine form in (3.10) can be written as follows,

$$\dot{x} = \bar{f}(x) + \bar{G}(x)\,u \tag{3.11}$$

where,

$$\bar{f}(x) = \begin{bmatrix} x_4 \\ x_5 \\ x_6 \\ -\frac{1}{m}\left(c_{x_7}s_{x_8}c_{x_9} + s_{x_7}s_{x_9}\right)x_{13} \\ -\frac{1}{m}\left(c_{x_7}s_{x_8}s_{x_9} - s_{x_7}c_{x_9}\right)x_{13} \\ g - \frac{1}{m}c_{x_7}c_{x_8}x_{13} \\ x_{10} + s_{x_7}t_{x_8}x_{11} + c_{x_7}t_{x_8}x_{12} \\ c_{x_7}x_{11} - s_{x_7}x_{12} \\ \frac{s_{x_7}}{c_{x_8}}x_{11} + \frac{c_{x_7}}{c_{x_8}}x_{12} \\ \left(\frac{J_2-J_3}{J_1}\right)x_{11}x_{12} \\ \left(\frac{J_3-J_1}{J_2}\right)x_{10}x_{12} \\ \left(\frac{J_1-J_2}{J_3}\right)x_{10}x_{11} \\ x_{14} \\ 0 \\ x_1 \\ x_2 \\ x_3 \\ x_9 \end{bmatrix}, \qquad \bar{G}(x) = \begin{bmatrix} & \mathbf{0}_{9\times 4} & & \\ 0 & \frac{1}{J_1} & 0 & 0 \\ 0 & 0 & \frac{1}{J_2} & 0 \\ 0 & 0 & 0 & \frac{1}{J_3} \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ & \mathbf{0}_{4\times 4} & & \end{bmatrix}$$

The controllability indices of the extended system are $\{5, 5, 5, 3\}$, where $\sum_{i=1}^{4} r_i = 18 = n$. With $r_1 = 5$, the distribution $\mathcal{G}_3$ is found to be involutive and $\dim \mathcal{G}_4 = 18 = n$, meaning all conditions of Theorem 3.1 are satisfied. Therefore, the system is state feedback linearizable, meaning locally transformable into a linear controllable system in Brunovsky controller form about the origin as shown below in (3.12).

We are guaranteed the existence of four smooth functions representing the linearizing position and yaw outputs $\{\varphi_1(x) = x_1,\ \varphi_2(x) = x_2,\ \varphi_3(x) = x_3,\ \varphi_4(x) = x_9\}$, such that matrix $D(x)$ in (3.4) is non-singular about the origin. The resulting linear system is written as

$$\begin{aligned}\dot{z} &= A_c\, z + B_c\, v\ , \qquad z \in \mathbb{R}^{18},\ v \in \mathbb{R}^4 \\ \xi &= C_c\, z \qquad\qquad ,\qquad \xi \in \mathbb{R}^4\end{aligned} \tag{3.12}$$

where,

$$\begin{aligned} z &= \left[\varphi_1, \ldots, L_f^{r_1-1}\varphi_1,\ \ldots\ , \varphi_m, \ldots, L_f^{r_m-1}\varphi_m\right]^T \\ &= \Big[x_{15}, x_1, x_4, \dot{x}_4, \ddot{x}_4,\ x_{16}, x_2, x_5, \dot{x}_5, \ddot{x}_5,\ x_{17}, x_3, x_6, \dot{x}_6, \ddot{x}_6,\ x_{18}, x_9, \dot{x}_9\Big]^T \end{aligned}$$

$$\dot{z} = \Big[z_2, z_3, z_4, z_5, v_1, z_7, z_8, z_9, z_{10}, v_2, z_{12}, z_{13}, z_{14}, z_{15}, v_3, z_{17}, z_{18}, v_4\Big]^T$$

To determine the domain of the transformation, the determinant of the matrix $D(x)$ is calculated to be

$$\det D(x) = -\frac{\bar{u}^2 \cos(\phi)}{m^3 J_1 J_2 J_3 \cos(\theta)}$$

Therefore, the domain for a non-singular solution is $\{\bar{u} \neq 0,\ -\frac{\pi}{2} < \phi < \frac{\pi}{2},\ -\frac{\pi}{2} < \theta < \frac{\pi}{2}\}$. As discussed earlier in Section 3.3.1, these constraints will be included within the NMPH optimization problem in (2.11d).

The actual control law is obtained using (3.5), giving

$$\begin{aligned}
\ddot{\bar{u}} &= m\ c_{x_9} s_{x_7} v_2 - m\ s_{x_7} s_{x_9} v_1 - m\ c_{x_7}\left(s_{x_8} c_{x_9} v_1 + s_{x_8} s_{x_9} v_2 + c_{x_8} v_3\right) \\
&\quad + x_{13}\left(x_{10}^2 +\ x_{11}^2\right) \\
\tau^{b1} &= \frac{mJ_1}{x_{13}} s_{x_7}\left(s_{x_8} c_{x_9} v_1 + s_{x_8} s_{x_9} v_2 + c_{x_8} v_3\right) + \frac{mJ_1}{x_{13}} c_{x_7}\left(c_{x_9} v_2 - s_{x_9} v_1\right) - (J_2 - J_3)\, x_{12} x_{11} \\
&\quad + \frac{J_1}{x_{13}}\left(x_{11} x_{12} x_{13} -\ 2 x_{10} x_{14}\right) \\
\tau^{b2} &= -\frac{mJ_2}{x_{13}}\ c_{x_8}\left(c_{x_9} v_1 + s_{x_9} v_2\right) + \frac{mJ_2}{x_{13}}\ s_{x_8} v_3 - \frac{J_2}{x_{13}}\left(x_{10} x_{12} x_{13} +\ 2 x_{11} x_{14}\right) \\
&\quad + (J_1 - J_3)\, x_{12} x_{10} \\
\tau^{b3} &= -\frac{1}{c_{x_7} c_{x_8} x_{13}}\Big[2J_3\left(2 x_{12} x_{11} c_{x_7}^2 + s_{x_7} x_{11}^2 - x_{12}^2 c_{x_7} - x_{12} x_{11}\right) x_{13} s_{x_8} \\
&\quad + \Big((J_1 - J_2 + J_3)\, x_{10} x_{11} x_{13} c_{x_7} + J_3 s_{x_7}\left(m s_{x_8} v_3 - 2 x_{10} x_{12} x_{13} - 2 x_{11} x_{14}\right)\Big) c_{x_8} \\
&\quad - J_3\big(m\ s_{x_7}\left(c_{x_9} v_1 + s_{x_9} v_2\right) +\ x_{13} v_4\big) c_{x_8}^2\Big]
\end{aligned} \tag{3.13}$$

where the feedback inputs are selected as follows,

$$v := \left\{ v_4 = \sum_{i=16}^{18} k_i e_{z_i} \ , \quad v_j = \sum_{i=5j-4}^{5j} k_i e_{z_i} \ : \ j = 1, 2, 3 \right\} \tag{3.14}$$

and the error e_{z_i} is defined as the difference between the desired and the actual feedback state $e_{z_i} = z_i^{ref} - z_i \ , \ i = 1, \ldots, n$. The errors can be interpreted as the differences between the desired outputs $[p_d^n, \psi_d]^T$ with their rates, and the corresponding actual system outputs $[p^n, \psi]^T$ with their rates.

3.3.3 Trajectory Generation using NMPH-FBL

As presented in (3.7) and (3.13), the drone's behavior is described by its nonlinear system dynamics and the feedback linearization control. The optimization within NMPH exploits their integration to enhance the performance of generating the reference trajectory. The continuous-time NMPH presented in Algorithm 2.2 is solved using a multiple shooting optimization technique. The solver used in our work, ACADO [83], discretizes the system dynamics, control law, and inequality constraints over the prediction horizon at each time instant t_n. Figure 2.4 shows the optimization process from the problem formulation to the trajectory generation.

A perfect model of the system dynamics would allow the closed-loop form to be represented by a linear canonical form as shown in (3.12), for which the feasibility and stability of the optimized solution are guaranteed and the computational power needed to solve the optimization problem is greatly reduced over the non-convex case. However, even an imperfect model still reduces the non-convexity of the optimization problem as compared to working directly with the (nonlinear) plant dynamics as in standard NMPC.

3.4 Simulation Results

In this Section, several simulations are presented to validate the proposed NMPH approach to generating optimal trajectories for a quadcopter vehicle.

The Robot Operating System (ROS) [95] is the base environment used to implement our algorithm. It is a platform that integrates different software packages or frameworks by handling communication between them and the host hardware. The ACADO Toolkit [83] is used for dynamic optimization in this work. It allows users to write their code within a self-contained C++ environment and generates the nonlinear solver that can solve the optimization problems in real-time. The compiled codes run within ROS to communicate with either a simulation model or the actual hardware [84]. For testing and validation of the proposed approaches on a quadcopter vehicle, we use the AirSim simulator [96], which is an open-source software that includes a physics engine and provides photo realistic images.

The NMPH optimization problem (2.11) was written in C++ code using ACADO and compiled into a highly efficient C code that is able to solve the optimization problem online. The AirSim simulator, ACADO optimization solver, and ROS environment run on a system with an Intel Core i7-10750H CPU @ 2.60-5.00 GHz equipped with the NVIDIA GeForce RTX 2080 Super (Max-Q) GPU. The prediction horizon of the optimization problem was set to $N = 40$ with a sampling period of 0.2 s, which was found sufficient for the purposes of trajectory generation. The cost function weights W_x, W_ξ, and W_T were adjusted heuristically to ensure a balanced trajectory generation performance towards the terminal setpoint.

The initial state of the quadcopter is acquired from the AirSim simulator and sent to the NMPH solver. The solution of the optimization problem is sent back to AirSim as a reference trajectory for the vehicle. The 3D visualization tool for ROS (RViz) is used to monitor and visualize the simulation process. Figure 3.2 shows the network architecture of the nodes and topics employed for running the simulation.

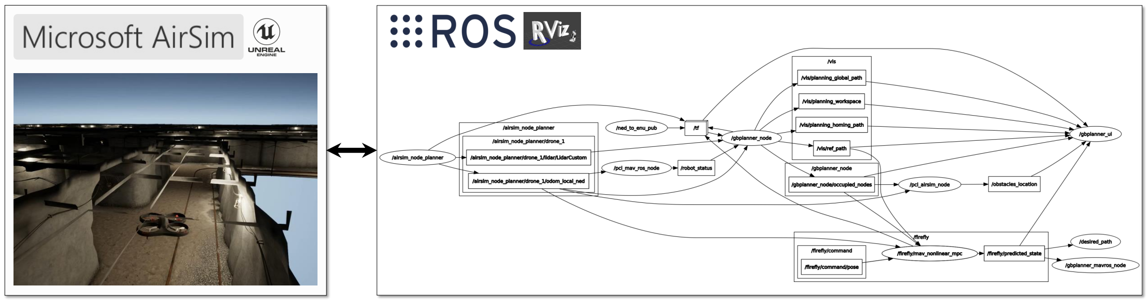


Figure 3.2: Simulation Architecture.

In the following Sections, various simulation scenarios are provided to verify the features and evaluate the performance of the proposed NMPH approach. Note the NED frame representation used within NMPH is converted to ENU (East, North, and Up) representation used by AirSim, and so the simulation results will be presented using the ENU convention as well.

3.4.1 Predicted Output and Estimated Reference Trajectories

The purpose of this simulation is to show the convergence of the estimated reference $\hat{\xi}_k$ and predicted output ξ_k trajectories toward the setpoint stabilization x_{ss} while minimizing the difference between them. The quadcopter is assumed to start at position $p^n = [0, 0, 0.2]^T$ m and NMPH is used to generate a trajectory to the terminal setpoint $p_d = [6, -3, 5]^T$ m as shown in Figure 3.3. The plots depict a sequence of the optimal predicted trajectory ξ_k, $k = 0, ..., N$, and the estimated reference trajectory $\hat{\xi}_k$, $k = 0, ..., N-1$ produced by the NMPH. It is important to mention that the convergence of the estimated reference trajectory to the terminal point ensures that the closed-loop system is steered to the desired endpoint.

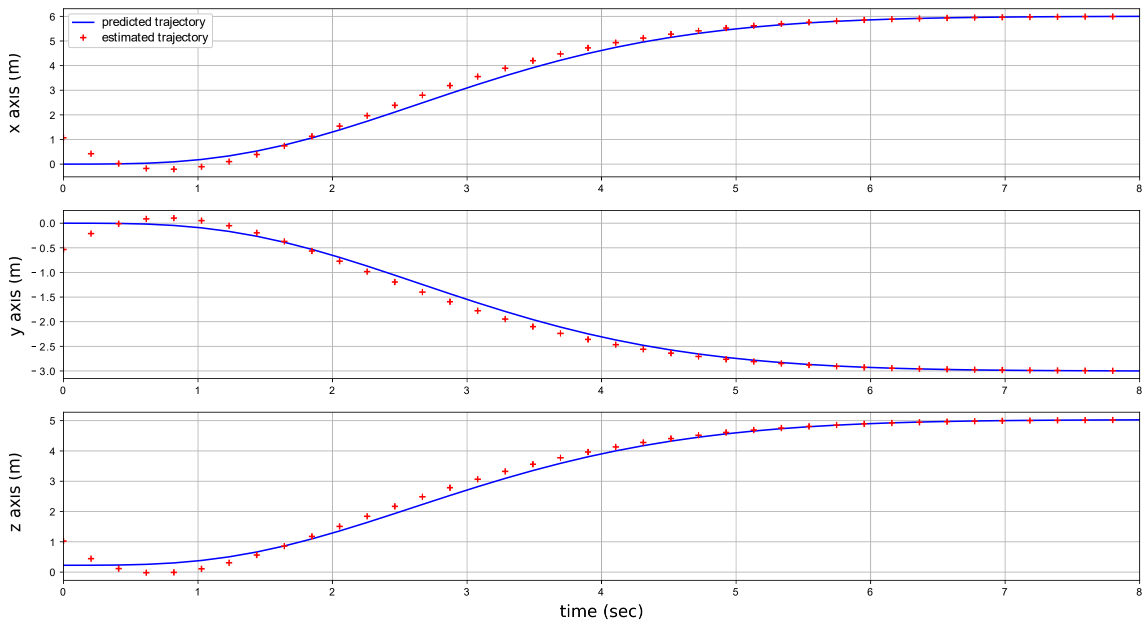


Figure 3.3: Predicted and estimated trajectories obtained from NMPH algorithm for an 8 s prediction horizon. For conciseness, the sequences of predicted output trajectory ξ_k and the estimated reference trajectory $\hat{\xi}_k$ represent only the quadcopter position p^n.

3.4.2 Trajectory Generation and Initial Conditions

In this simulation, the generated trajectory is investigated for different initial conditions. The initial conditions being tested are related to the quadcopter's kinematics, where the vehicle is commanded to move in a straight path between $[4, 2, 0]^T$ and $[8, 8, 8]^T$ while changing its speed $|v|$ linearly from 0 to 1.5 m/s as shown in Figure 3.4. Note the objective is not to track generated trajectories, but just to observe the behaviour of OTP solutions towards the stabilization setpoint $p_d = [5, 10, 5]^T$.

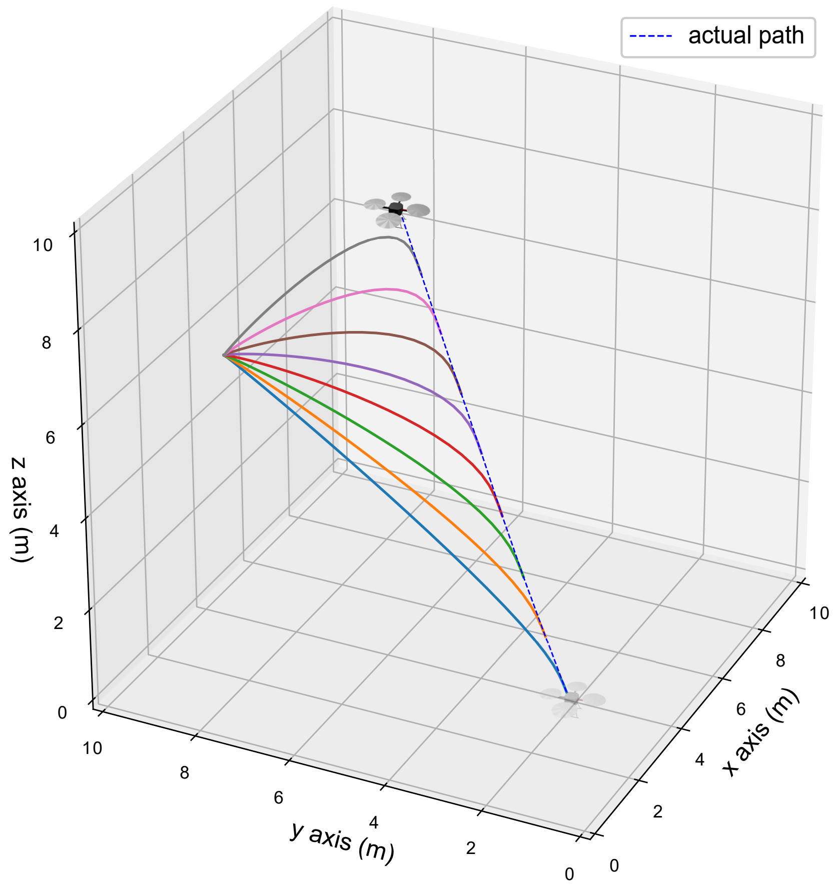


Figure 3.4: Trajectory generation for different initial conditions. The quadcopter moves along the dashed line. The trajectories all converge toward the stabilization setpoint shown to the left at $[5, 10, 5]^T$.

The solution of the optimization problem for 8 different trajectories are plotted in Figure 3.4 to show the effect of the initial conditions on them. The resulting solution of each one shows a trajectory which convergences smoothly to the stabilization set-point while taking into consideration the initial position and velocity of the system. Commanding the quadcopter to track generated trajectories which account for its initial conditions will reduce jerky flight motions and therefore reduce flight power consumption, which is especially important for exploration missions.

Using the setup described in Section 3.4, our NMPH achieves a 250 Hz generation rate, meaning a reference trajectory is generated every 4 ms. If running on lower-powered hardware, the computational power can be minimized by reducing the rate of trajectory generation, which still provides a smooth reference trajectory for the vehicle.

3.4.3 Trajectory Tracking

In this simulation, the quadcopter's trajectory tracking and static obstacle avoidance performance are examined. First, the vehicle is commanded to track a continuously updated trajectory generated on-the-fly by the NMPH algorithm while avoiding static obstacles, as shown in Figure 3.5. Each static obstacle is considered to be a sphere of 1 m in diameter. A radial allowance of 1 m is considered about the obstacle to avoid crashing to it. Hence, the constraint of each obstacle represents a sphere with a diameter of 3 m, which makes the safety distance $\epsilon = 1.5$. The smooth tracking performance while avoiding the obstacles can be seen in Figure 3.5, which shows the importance of using the NMPH in regenerating the trajectory while tracking it.

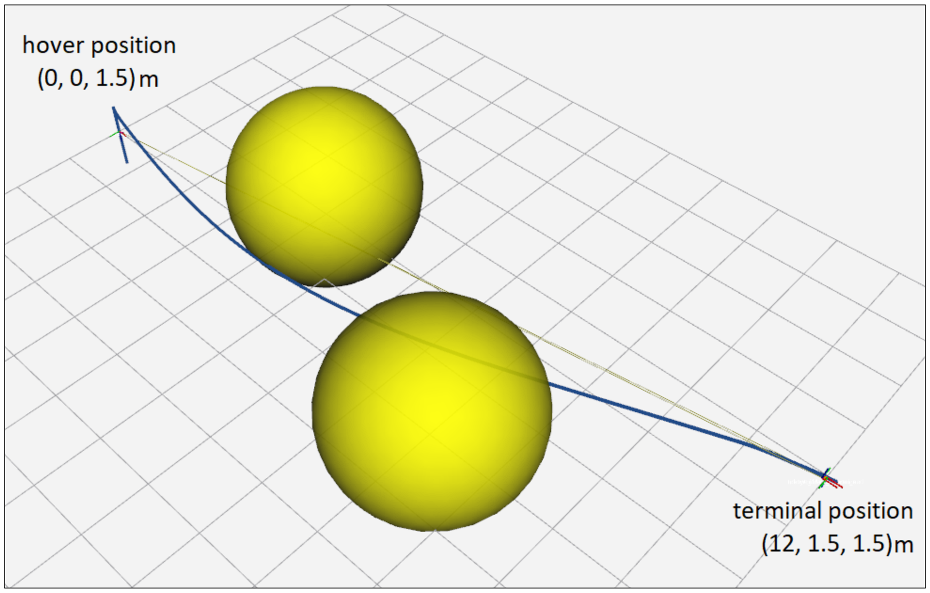


Figure 3.5: Drone trajectory tracking of a continuously updated trajectory by NMPH while avoiding two static obstacles. The drone is commanded first to hover at a height of 1.5 m, then to track the NMPH trajectory between the start and the terminal position.

In the second study, the regeneration process of the predicted trajectory is limited to one regeneration in order to examine its effect on the tracking performance while avoiding the obstacles. The simulation result is depicted and explained in Figure 3.6.

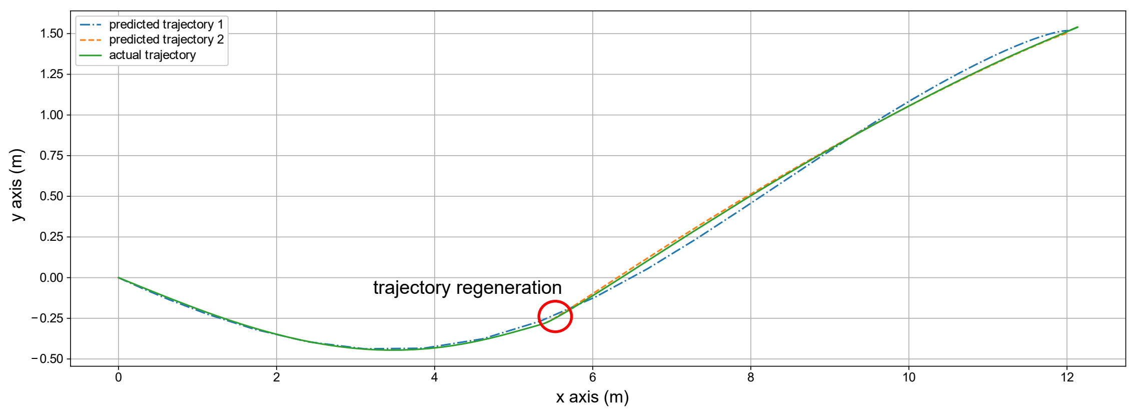


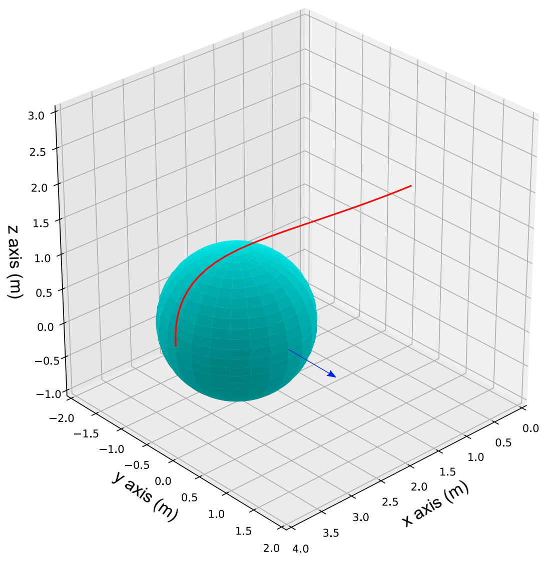
Figure 3.6: Drone trajectory tracking of the predicted reference trajectory by NMPH. At the start position $p^n = [0, 0, 1.5]^T$ m, NMPH generated the *predicted trajectory 1*, and when the drone reached $[5.5, -0.25, 1.5]^T$ m, NMPH re-optimized the trajectory, which is represented by *predicted trajectory 2*.

The continuous regeneration of the reference trajectory provides optimal flight paths in real time based on the system's state. This ability also enables handling dynamic obstacles, as shown next in Section 3.4.4.

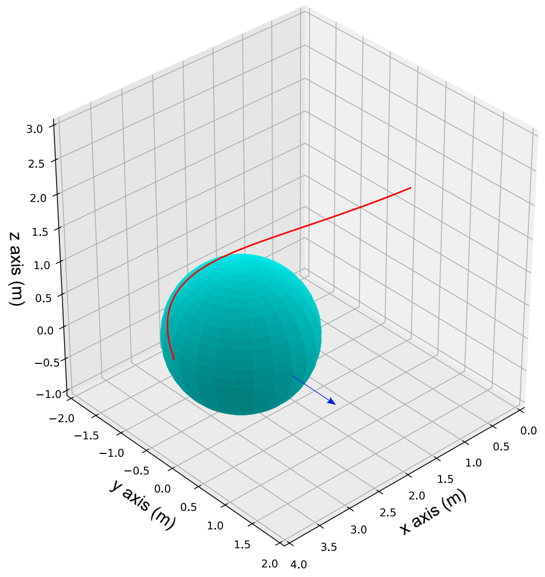
3.4.4 Dynamic Obstacle Avoidance

Figures 3.7(a) to 3.7(c) depict the online regeneration of the predicted optimal trajectory when the obstacle moves in the direction of y-axis. The optimized trajectory starts at the hover position $p = [0, 0, 1.5]^T$ m and converges to the terminal setpoint $p_d = [4, 0, 0.5]^T$ m while the predicted reference trajectory is being continuously regenerated. An obstacle placed at the initial position $(3, 0, 0.5)$ m with total diameter of 2 m moves at a velocity of 0.5 m/s in the y-axis direction.

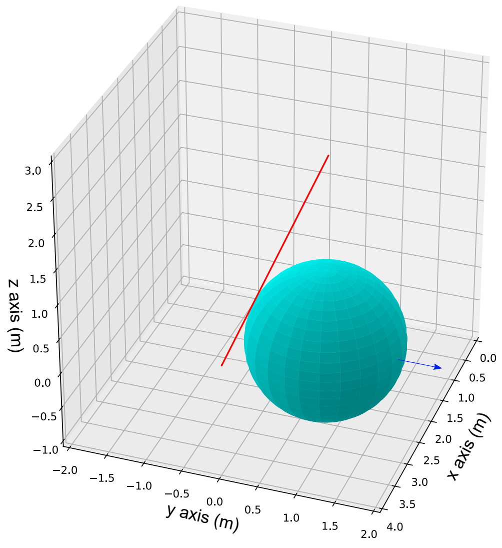
Selected predictions over 2 s are shown in Figure 3.7(d) which illustrates the smooth regeneration of the trajectories while avoiding the dynamic obstacle. It is important to note that about 500 trajectories are generated in 2 s.



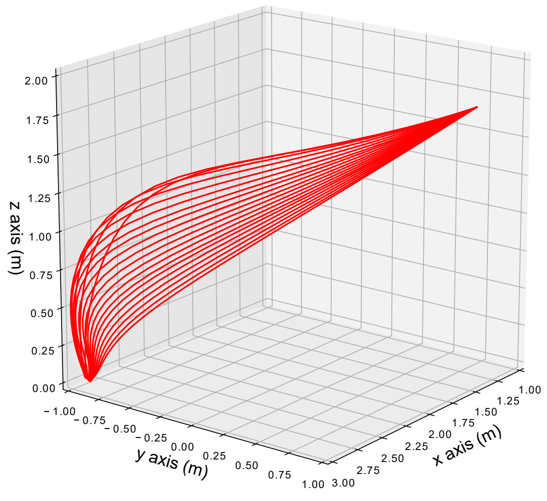
(a)



(b)



(c)



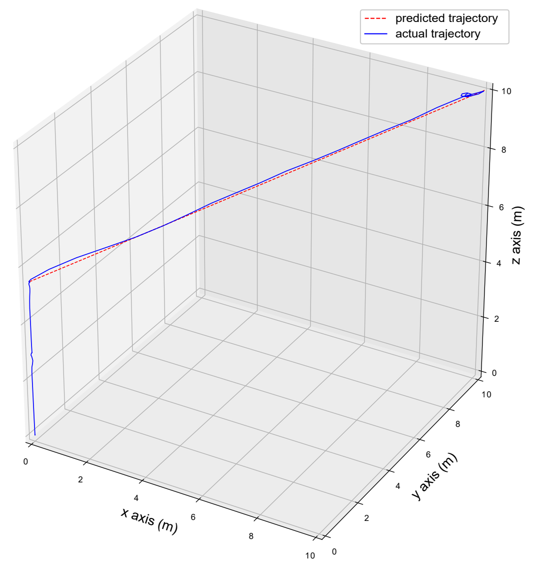
(d)

Figure 3.7: Dynamic obstacle avoidance for a 2 m spherical obstacle that moves at a velocity of 0.5 m/s in the y-axis direction starting from the initial position $(3, 0, 0.5)$ m. (a) to (c) show the continuous regeneration of the NMPH predicted optimal trajectory which avoids the moving obstacle, and (d) depicts the smooth regeneration process for a selected number of the trajectory updates.

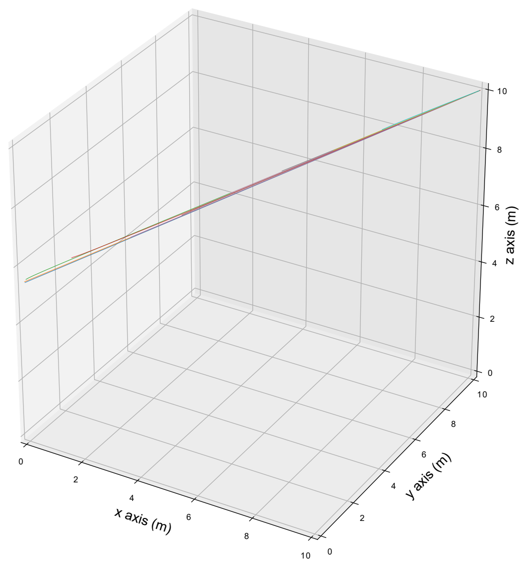
3.4.5 Hardware-In-The-Loop Simulation

Realistic simulations of PX4 autopilot [97] can be achieved using hardware-in-the-loop (HITL) simulation with AirSim. In this test, the Pixhawk 2.1 flight controller is used, which runs PX4 firmware, to evaluate the performance of the developed codes on the real hardware.

Two trajectory tracking scenarios are offered in HITL tests. The first scenario allows the quadcopter to track the generated NMPH trajectory from an initial position to a stabilization setpoint. The NMPH trajectory keeps updating while the drone is moving towards the setpoint. Figure 3.8 shows the NMPH predicted trajectory and the vehicle's tracking performances in HITL mode. The second scenario aims to test the drone's predicted trajectory tracking at the presence of two obstacles, each represents a sphere with a radius of 1.5 m, and are located at $(4, 1, 3.5)$ and $(8, -1, 3.5)$. Figure 3.9 depicts the vehicle's tracking response while avoiding the obstacles. Both figures show how the drone managed to track the predicted trajectories at the presence of the real hardware uncertainties and obstacles.

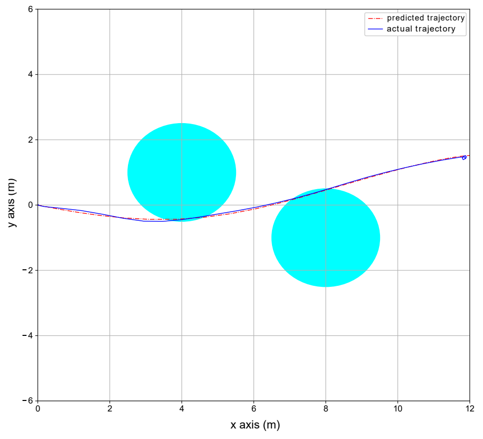


(a)

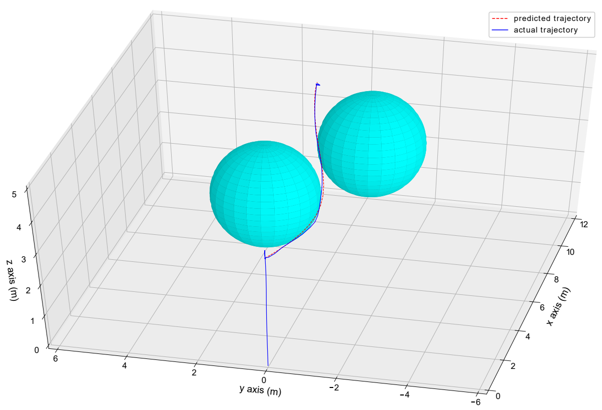


(b)

Figure 3.8: HITL simulation for NMPH trajectory generation and tracking. (a) NMPH generates the predicted trajectory when the quadcopter is at a hover state $(0, 0, 5.5)$ m, then the vehicle is asked to track the trajectory. (b) shows some of the NMPH predicted trajectory updates while the drone is moving towards the setpoint stabilization $(10, 10, 10)$ m.



(a)



(b)

Figure 3.9: HITL simulation for NMPH trajectory generation and tracking at the presence of obstacles. The obstacles, predicted trajectory, and the actual trajectory are depicted in (a) a top view 2D plot, and (b) a 3D plot. The drone generates the predicted trajectory from the hovering position at $(0, 0, 3.5)$ m to the setpoint stabilization at $(12, 1.5, 3.5)$ m.

3.5 Conclusions

In this chapter, a proposed formulation, called NMPH-FBL, applies a feedback linearization control law to the nonlinear plant model, resulting in a closed-loop dynamics model with decreased non-convexity used by the online optimization problem to generate feasible and optimal reference trajectories for the actual closed-loop system. The feedback linearization design includes integral states to compensate for modeling uncertainties and external disturbances in the system. The proposed NMPH algorithm supports both static and dynamic obstacles, enabling trajectory generation in continuously changing environments.

The NMPH-FBL was implemented on a simulated quadcopter drone and validated to generate 3D optimal reference trajectories in real time. Different simulation scenarios (including HITL simulations) were carried out to evaluate the performance of the proposed method. Convergence of the predicted and estimated trajectories, trajectory generation under different initial conditions, trajectory tracking performance, and the ability to navigate around static and dynamic obstacles were validated through simulation results.

Chapter 4

A Backstepping Approach to Nonlinear Model Predictive Horizon for Optimal Trajectory Planning

4.1 Introduction

This chapter presents an alternative solution of generating optimal trajectories using the NMPH approach based on Backstepping Control (BSC) technique [98]. This is done by combining the nonlinear plant model with a recursive design of the backstepping control method within the optimization problem. The new formulation is named NMPH-BSC.

In Chapter 3, we proposed an NMPH formulation which used Feedback Linearization (FBL) within its dynamics to compensate for nonlinearities [77]. In that work, the state augmentation process required to make the system state feedback linearizable created numerical difficulties due to the need to obtain the second-order time derivative of the total thrust of the drone. Also, the NMPH-FBL formulation relied on state feedback linearization [87], which creates challenges dealing with more sophisticated system models than the one presented in (3.7).

The presented difficulties with using the NMPH-FBL design steered us to find an alternative feedback design methodology to be used within the NMPH optimization problem. The new NMPH-BSC formulation compensates for the system nonlinearities and guarantees global asymptotic stability of the closed-loop system within the optimization problem used to predict the reference trajectories for a system, which is a drone vehicle in this work. The recursive structure of BSC provides stable response of a dynamic system and makes it more robust to parameter uncertainties.

The research contributions of this chapter are:

- Implementing the BSC method within NMPH to compensate for nonlinearities in order to reduce the non-convexity of the trajectory generation optimization problem.
- Demonstrating the versatility of the NMPH-BSC approach by using both a simplified and a higher-fidelity dynamics model of the drone.
- Using the NMPH optimization problem to predict both the reference trajectory as well as its rates of change for the onboard flight trajectory controller.
- Validating and evaluating the performance of the proposed approach in both simulation and hardware drone flight experiments.

The remainder of this chapter is arranged as follows: the design of the NMPH-BSC for two variants of drone dynamics models is presented in Section 4.2. Section 4.3 evaluates the proposed designs in simulation and hardware flight tests. Section 4.4 summarizes the chapter.

4.2 Backstepping Control Law Integration within NMPH

In this section, both high-fidelity and simplified drone dynamics models are presented and a backstepping design for each model is derived and integrated into the NMPH.

4.2.1 Drone Dynamics

To demonstrate the versatility of the NMPH-BSC approach, a more detailed model of the vehicle dynamics is employed (compared to the model presented in Section 3.3.1) which includes aerodynamic drag forces and rotor gyroscopic effects .

The dynamics of a multi-rotor drone vehicle can be modeled using the Newton-Euler equations [90] governing six degree of freedom rigid-body motion, augmented with force and torque generation models of the individual rotors. The model can either assume static hover conditions for simplicity, or include linear and angular velocity drag forces and rotor gyroscopic effects to yield a more complicated but higher-fidelity model.

To model rigid-body dynamics, two reference frames are used: a stationary ground-fixed navigation frame $\mathcal{N}$, and a moving body-fixed frame $\mathcal{B}$. The origin of the latter is placed at the drone's center of gravity, as shown in Figure 4.1. In this model, the frame bases employ the East, North, and Up (ENU) convention, with orthonormal basis vectors $\{n_1, n_2, n_3\}$ and $\{b_1, b_2, b_3\}$ for the navigation and body frames, respectively.

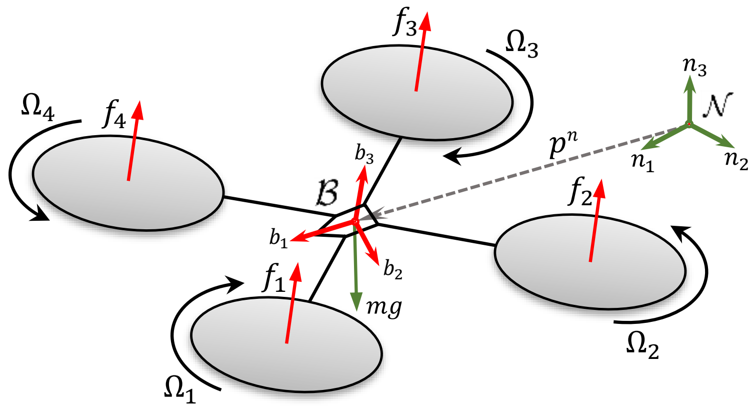


Figure 4.1: Reference frames used for our quadrotor vehicle

Rigid-body pose in space can be described as a member of the Special Euclidean group SE(3) = SO(3) $\times \mathbb{R}^3$, the product space of the orientation and position (R_{nb}, p^n) where $R_{nb} \in$ SO(3) is the rotation matrix of the body frame with respect to the navigation frame, $p^n = [x \quad y \quad z]^T \in \mathbb{R}^3$ is the position vector of the vehicle's body frame with respect to the navigation frame. The roll-pitch-yaw Euler angles $\eta = [\phi \quad \theta \quad \psi]^T$ are employed to parametrize the rotation matrix.

Remark 4.1 *The Euler angle parameterization exhibits singularities at $\theta = \pi/2 + k\pi$, $k \in \mathbb{Z}$. One solution is to maintain $-\pi/2 < \theta < \pi/2$ by adding constraints within the NMPH optimization problem under (2.11d).*

Conversion between translational and rotational velocity vectors can be done using the transformations [99]

$$\dot{p}^n = v^n = R_{nb} v^b \tag{4.1a}$$

$$\omega^b = W \dot{\eta} \tag{4.1b}$$

where $v^n, v^b \in \mathbb{R}^3$ are the translational velocity vectors in frame $\mathcal{N}$ and $\mathcal{B}$ coordinates, respectively, $\dot{\eta} = [\dot{\phi} \quad \dot{\theta} \quad \dot{\psi}]^T$ is the vector of Euler angle rates, and $\omega^b \in \mathbb{R}^3$ is the angular velocity vector in frame $\mathcal{B}$ coordinates. The rotational velocity transformation matrix W is given by

$$W = \begin{bmatrix} 1 & 0 & -s_\theta \\ 0 & c_\phi & s_\phi c_\theta \\ 0 & -s_\phi & c_\phi c_\theta \end{bmatrix} \tag{4.2}$$

The time derivative of the rotation matrix is $\dot{R}_{nb} = R_{nb} \mathcal{S}(\omega^b)$, where $\mathcal{S}(\cdot) : \mathbb{R}^3 \to \mathrm{so}(3)$ maps a vector to a skew-symmetric matrix such that $\mathcal{S}(a)b = a \times b$ for $a, b \in \mathbb{R}^3$. Taking the time derivatives of (4.1a) and (4.1b),

$$\dot{v}^n = R_{nb} \dot{v}^b + R_{nb} \mathcal{S}(\omega^b) v^b = R_{nb} \left(\dot{v}^b + \omega^b \times v^b \right) \tag{4.3a}$$

$$\dot{\omega}^b = \dot{W} \dot{\eta} + W \ddot{\eta} \tag{4.3b}$$

where $\dot{W} = \dot{\phi}(\partial W / \partial \phi) + \dot{\theta}(\partial W / \partial \theta)$.

The Newton-Euler equations for a multi-rotor drone read [99]

$$m \dot{v}^b + \omega^b \times m v^b = \bar{u} - K_t v^b - m R_{nb}^T \bar{g} \tag{4.4a}$$

$$J \dot{\omega}^b + \omega^b \times J \omega^b = \bar{\tau} - K_r \omega^b - \sum_{i=1}^{4} \left(\mathcal{S}(\omega^b) J_r \, q_i \right) \tag{4.4b}$$

where m is the drone's mass, $\bar{u} = [0 \quad 0 \quad u]^T$ is the thrust vector with $u = \sum_{i=1}^{4} f_i$ the total thrust generated in the direction of b_3, $\bar{\tau} = [\tau^{b_1} \quad \tau^{b_2} \quad \tau^{b_3}]^T$ is the vector of torques about the b_1, b_2 and b_3 frame axes, $\bar{g} = [0 \quad 0 \quad g]^T$ is the gravitational acceleration vector where $g = 9.81$ m/s^2, $J = \mathrm{diag}(J_x, J_y, J_z)$ is the drone's mass moment of inertia matrix which is assumed to be diagonal, the scalar J_r is the rotor's inertia, $q_i = [0 \quad 0 \quad (-1)^{i+1}\omega_i]^T$ where ω_i is the angular speed of the i^{th} propeller, and $K_t = \mathrm{diag}(k_{t1}, k_{t2}, k_{t3})$, $K_r = \mathrm{diag}(k_{r1}, k_{r2}, k_{r3})$ represent the translational and rotational drag coefficient matrices of the drone, respectively.

To express the drone's dynamics with respect to the navigation frame, equations (4.4) are combined with (4.1) and (4.3) to yield

$$mR_{nb}^T \dot{v}^n = \bar{u} - K_t R_{nb}^T v^n - mR_{nb}^T \bar{g} \tag{4.5a}$$

$$JW\ddot{\eta} + J\dot{W}\dot{\eta} + \mathcal{S}(W\dot{\eta})JW\dot{\eta} = \bar{\tau} - K_r W\dot{\eta} - \mathcal{S}(W\dot{\eta}) \sum_{i=1}^{4} J_r\, q_i \tag{4.5b}$$

This leads to

$$\dot{v}^n = -\frac{1}{m} R_{nb} K_t R_{nb}^T v^n - \bar{g} + \frac{1}{m} R_{nb} \bar{u} \tag{4.6a}$$

$$\ddot{\eta} = -(JW)^{-1}\big(J\dot{W}\dot{\eta} + K_r W\dot{\eta} + \mathcal{S}(W\dot{\eta})\big(JW\dot{\eta} + \sum_{i=1}^{4} J_r\, q_i\big) - \bar{\tau}\big) \tag{4.6b}$$

Remark 4.2 *Each multi-rotor configuration (quadrotor, hexarotor, octorotor, and so on) has a different expression for the net body-frame thrust and torque vectors $\bar{u}$ and $\bar{\tau}$. These expressions are algebraic and can be readily calculated. The dynamics presented in (4.6) thus model any multi-rotor drone as long as the correct $\bar{u}$ and $\bar{\tau}$ expressions are used.*

The development of the proposed NMPH with a backstepping control design will be based on the dynamics model presented in (4.6). We will also present a design based on a simplified version of (4.6) to illustrate the ease of adapting the proposed approach to different model representations. This is in contrast to the formulation of NMPH with feedback linearization presented in Chapter 3 where this adaptation requires a fundamental re-derivation of the expressions involved.

In the simplified version of (4.6), body and propeller gyroscopic effects are dropped from the model, and the translational and rotational drags are neglected as well. The simplified model can then be written as

$$\begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \\ \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} \frac{J_y - J_z}{J_x} \dot{\theta}\dot{\psi} + \frac{1}{J_x}\tau^{b_1} \\ \frac{J_z - J_x}{J_y} \dot{\phi}\dot{\psi} + \frac{1}{J_y}\tau^{b_2} \\ \frac{J_x - J_y}{J_z} \dot{\phi}\dot{\theta} + \frac{1}{J_z}\tau^{b_3} \\ (c_\phi s_\theta c_\psi + s_\phi s_\psi)\frac{u}{m} \\ (c_\phi s_\theta s_\psi - s_\phi c_\psi)\frac{u}{m} \\ -g + (c_\phi c_\theta)\frac{u}{m} \end{bmatrix} \tag{4.7}$$

4.2.2 Backstepping Control Design

Because a drone's dynamics are nonlinear, solving the NMPC optimization problem is challenging because of its non-convexity. Introducing backstepping control into the optimization problem (within the framework of NMPH) will either remove or reduce the nonlinearity of the overall system, and consequently the non-convexity of the optimization problem. This will make it possible to find an optimal solution more quickly.

In this section, a backstepping control law is derived for the drone dynamics, which will be used within the NMPH framework to enhance the performance of the reference trajectory prediction. To demonstrate the versatility of this methodology, both the simplified (4.7) and high-fidelity (4.6) models of the drone dynamics will be considered.

Our Backstepping control design consists of a coupling of inner and outer control loops [100]. The inner loop controls the rotational dynamics of the drone and tracks desired values provided by the outer loop, which controls the translational dynamics. In the literature, many studies of backstepping control applied to multi-rotor drones considered applying the design steps to each system output separately [99, 101–103], but in our work the method is first applied to the rotational dynamics subsystem by itself, then to the translational dynamics subsystem. This approach will facilitate the integration of BSC within the NMPH framework as discussed later in Section 4.2.3.

First, recall the terms

$$\eta = [\phi \quad \theta \quad \psi]^T, \quad \dot{\eta} = [\dot{\phi} \quad \dot{\theta} \quad \dot{\psi}]^T, \quad \eta_{_d} = [\phi_d \quad \theta_d \quad \psi_d]^T \tag{4.8}$$

where $\eta_{_d} \in \mathbb{R}^3$ are the desired Euler angles, to be provided by the outer loop design. Now, define the tracking error vector $\delta_1 \in \mathbb{R}^3$ as

$$\delta_1 = \eta_{_d} - \eta, \tag{4.9}$$

and choose a positive semi-definite Lyapunov function candidate $V_1 \geq 0 \in \mathbb{R}$, such that

$$V_1 = \frac{1}{2}\delta_1^T \delta_1 \tag{4.10}$$

The time derivative of (4.10) is

$$\dot{V}_1 = \delta_1^T \dot{\delta}_1 = \delta_1^T (\dot{\eta}_{_d} - \dot{\eta}) \tag{4.11}$$

Next, define the virtual tracking error rate $\delta_2 \in \mathbb{R}^3$ and the first virtual control $v_1 = [v_\phi \quad v_\theta \quad v_\psi]^T \in \mathbb{R}^3$ as

$$\delta_2 = v_1 - \dot{\eta} \tag{4.12}$$

$$v_1 = \dot{\eta}_{_d} + \Lambda_1 \delta_1 \tag{4.13}$$

where $\Lambda_1 = \text{diag}(\lambda_1, \lambda_2, \lambda_3) \in \mathbb{R}^{3\times 3}$ is a diagonal gain matrix that contains positive entries such that Λ_1 is positive definite or $\Lambda_1 > 0$. Using (4.12) and (4.13), the derivative of the Lyapunov function candidate (4.11) can be written as

$$\dot{V}_1 = \delta_1^T (v_1 - \Lambda_1 \delta_1 - \dot{\eta}) = \delta_1^T (\delta_2 - \Lambda_1 \delta_1) = \delta_1^T \delta_2 - \delta_1^T \Lambda_1 \delta_1 \tag{4.14}$$

which by inspection may or may not be negative semi-definite. Therefore, a recursive step must be performed. Note that the time derivatives of (4.9) and (4.12) are

$$\begin{aligned}\dot{\delta}_1 &= \dot{\eta}_d - \dot{\eta} = v_1 - \Lambda_1\delta_1 + \delta_2 - v_1 = -\Lambda_1\delta_1 + \delta_2 && (4.15)\\ \dot{\delta}_2 &= \dot{v}_1 - \ddot{\eta} = \ddot{\eta}_d + \Lambda_1\dot{\delta}_1 - \ddot{\eta} && (4.16)\end{aligned}$$

Now define the new positive semi-definite Lyapunov function candidate $V_2 \geq 0 \in \mathbb{R}$ as

$$V_2 = \frac{1}{2}\delta_1^T\delta_1 + \frac{1}{2}\delta_2^T\delta_2 \tag{4.17}$$

such that

$$\begin{aligned}\dot{V}_2 &= \delta_1^T\dot{\delta}_1 + \delta_2^T\dot{\delta}_2\\ &= -\delta_1^T\Lambda_1\delta_1 + \delta_1^T\delta_2 + \delta_2^T\big(\ddot{\eta}_d + \Lambda_1(-\Lambda_1\delta_1 + \delta_2) - \ddot{\eta}\big) && (4.18)\\ &= -\delta_1^T\Lambda_1\delta_1 + \delta_2^T\big(\delta_1 + \ddot{\eta}_d - \Lambda_1^2\delta_1 + \Lambda_1\delta_2 - \ddot{\eta}\big)\end{aligned}$$

where $\delta_1^T\delta_2 = \delta_2^T\delta_1$. To stabilize the tracking errors δ_1 and δ_2, the backstepping control formulation will introduce a second virtual control $v_2 \in \mathbb{R}^3$. We will define v_2 based on the system dynamics, and then recursively design it within the backstepping control structure.

As mentioned in Section 4.2.1, we will apply the backstepping technique to both the full and simplified system dynamics presented in (4.6) and (4.7), respectively. For the full model, the attitude dynamics in (4.6b) can be written as

$$\ddot{\eta} = \bar{f}_1(x) + \bar{g}_1(x, \bar{\tau}) \tag{4.19}$$

where

$$\begin{aligned}\bar{f}_1(x) &= -(JW)^{-1}\big(J\dot{W}\dot{\eta} + K_rW\dot{\eta} + \mathcal{S}(W\dot{\eta})\big(JW\dot{\eta} + \sum_{i=1}^{4} J_rq_i\big)\big) && (4.20)\\ \bar{g}_1(x, \bar{\tau}) &= (JW)^{-1}\bar{\tau} = \begin{bmatrix} \frac{1}{J_x}\tau^{b_1} + \frac{1}{J_y}s_\phi t_\theta\tau^{b_2} + \frac{1}{J_z}c_\phi t_\theta\tau^{b_3}\\ \frac{1}{J_y}c_\phi\tau^{b_2} - \frac{1}{J_z}s_\phi\tau^{b_3}\\ \frac{1}{J_y}\frac{s_\phi}{c_\theta}\tau^{b_2} + \frac{1}{J_z}\frac{c_\phi}{c_\theta}\tau^{b_3}\end{bmatrix} := \begin{bmatrix}\tau_\phi\\ \tau_\theta\\ \tau_\psi\end{bmatrix} && (4.21)\end{aligned}$$

The second virtual control is defined as $v_2 = \bar{g}_1(x, \bar{\tau}) = [\tau_\phi \quad \tau_\theta \quad \tau_\psi]^T$, where τ_ϕ, τ_θ and τ_ψ are the virtual inputs of the rotational subsystem.

We now design v_2 to make the time derivative of the Lyapunov candidate function (4.18) negative semi-definite. Let

$$\begin{aligned} v_2 &= \dot{v}_1 + \delta_1 - \bar{f}_1(x) + \Lambda_2\delta_2 \\ &= \ddot{\eta}_d + \Lambda_1\dot{\delta}_1 + \delta_1 - \bar{f}_1(x) + \Lambda_2\delta_2 \\ &= \ddot{\eta}_d + \Lambda_1(-\Lambda_1\delta_1 + \delta_2) + \delta_1 - \bar{f}_1(x) + \Lambda_2\delta_2 \end{aligned} \tag{4.22}$$

where $\Lambda_2 = \text{diag}(\lambda_4, \lambda_5, \lambda_6) \in \mathbb{R}^{3\times3}$ is the second diagonal gain matrix with positive entries such that $\Lambda_2 > 0$. By substituting (4.19) and (4.22) into (4.18) we obtain

$$\begin{aligned} \dot{V}_2 &= -\delta_1^T\Lambda_1\delta_1 + \delta_2^T\big(\delta_1 + \ddot{\eta}_d - \Lambda_1^2\delta_1 + \Lambda_1\delta_2 - \bar{f}_1(x) - v_2\big) \\ &= -\delta_1^T\Lambda_1\delta_1 - \delta_2^T\Lambda_2\delta_2 \leq 0 \end{aligned} \tag{4.23}$$

By (4.17) and (4.23), we can thus conclude the asymptotic stability of the error terms δ_1 and δ_2, and thus the rotational subsystem. Consequently, the physical control law for the rotational subsystem can be found by returning to (4.22),

$$v_2 = [\tau_\phi \quad \tau_\theta \quad \tau_\psi]^T = \ddot{\eta}_d + (I - \Lambda_1^2)\delta_1 + (\Lambda_1 + \Lambda_2)\delta_2 - \bar{f}_1(x), \tag{4.24}$$

then solving (4.21) to obtain the physical control inputs as

$$\begin{bmatrix} \tau^{b_1} \\ \tau^{b_2} \\ \tau^{b_3} \end{bmatrix} = \begin{bmatrix} J_x\big(\tau_\phi - s_\theta\tau_\psi\big) \\ J_y\big(c_\phi\tau_\theta + s_\phi c_\theta\tau_\psi\big) \\ J_z\big(- s_\phi\tau_\theta + c_\phi c_\theta\tau_\psi\big) \end{bmatrix} \tag{4.25}$$

We now perform the backstepping design for the translational dynamics of the drone. The actual and desired position vectors with respect to the navigation frame are written as

$$\chi = p^n = [x \quad y \quad z]^T, \quad \dot{\chi} = v^n = [\dot{x} \quad \dot{y} \quad \dot{z}]^T, \quad \chi_d = p_d^n = [x_d \quad y_d \quad z_d]^T \tag{4.26}$$

For the first step of the backstepping design, the position tracking error vector and its time derivative are defined as

$$\delta_3 = \chi_d - \chi \ , \quad \dot{\delta}_3 = \dot{\chi}_d - \dot{\chi} \in \mathbb{R}^3, \tag{4.27}$$

Consider the Lyapunov candidate function $V_3 \in \mathbb{R}$ and its time derivative

$$V_3 = \frac{1}{2}\delta_3^T \delta_3 \tag{4.28}$$

$$\dot{V}_3 = \delta_3^T \dot{\delta}_3 = \delta_3^T(\dot{\chi}_d - \dot{\chi}) \tag{4.29}$$

Also define the virtual tracking error rate and the first virtual control for the translational subsystem as

$$\delta_4 = v_3 - \dot{\chi} \tag{4.30}$$

$$v_3 = \dot{\chi}_d + \Lambda_3 \delta_3 \tag{4.31}$$

where $\Lambda_3 = \text{diag}(\lambda_7, \lambda_8, \lambda_9) \in \mathbb{R}^{3\times3}$ is a diagonal gain matrix with positive entries, and $v_3 = [v_x \quad v_y \quad v_z]^T \in \mathbb{R}^3$ is the virtual control vector. Substituting (4.30) and (4.31) into the Lyapunov candidate function rate (4.29) yields

$$\begin{aligned}\dot{V}_3 &= \delta_3^T(v_3 - \Lambda_3\delta_3 + \delta_4 - v_3)\\ &= -\delta_3^T \Lambda_3 \delta_3 + \delta_3^T \delta_4\end{aligned} \tag{4.32}$$

which cannot be guaranteed to be negative semi-definite. Therefore, a new Lyapunov candidate function $V_4 \in \mathbb{R}$ is defined as

$$V_4 = \frac{1}{2}\delta_3^T \delta_3 + \frac{1}{2}\delta_4^T \delta_4 \tag{4.33}$$

$$\begin{aligned}\dot{V}_4 &= \delta_3^T \dot{\delta}_3 + \delta_4^T \dot{\delta}_4 \qquad (4.34)\\ &= -\delta_3^T \Lambda_3 \delta_3 + \delta_3^T \delta_4 + \delta_4^T\left(\ddot{\chi}_d + \Lambda_3(\dot{\chi}_d - \dot{\chi}) - \ddot{\chi}\right)\\ &= -\delta_3^T \Lambda_3 \delta_3 + \delta_4^T \delta_3 + \delta_4^T\left(\ddot{\chi}_d + \Lambda_3(v_3 - \Lambda_3\delta_3 + \delta_4 - v_3) - \ddot{\chi}\right)\\ &= -\delta_3^T \Lambda_3 \delta_3 + \delta_4^T\left(\delta_3 + \ddot{\chi}_d - \Lambda_3^2\delta_3 + \Lambda_3\delta_4 - \ddot{\chi}\right)\end{aligned}$$

The translational dynamics of the full model (4.6a) can be written as

$$\ddot{\chi} = \bar{f}_2(x) + \bar{g}_2(x, u) \tag{4.35}$$

where

$$\bar{f}_2(x) = -\frac{1}{m} R_{nb} K_t R_{nb}^T v^n - \bar{g} \tag{4.36}$$

$$\bar{g}_2(x,u) = \frac{1}{m}\begin{bmatrix}(c_\phi s_\theta c_\psi + s_\phi s_\psi)u \\ (c_\phi s_\theta s_\psi - s_\phi c_\psi)u \\ (c_\phi c_\theta)u\end{bmatrix} := \begin{bmatrix}u_x \\ u_y \\ u_z\end{bmatrix} \tag{4.37}$$

and where $u \in \mathbb{R}$ is the total thrust of the propellers, a physical input.

The next step of the backstepping design introduces the second virtual control for the translational system $v_4 = \bar{g}_2(x,u) = [u_x \quad u_y \quad u_z]^T$. Assign this control as

$$\begin{aligned} v_4 &= \dot{v}_3 + \delta_3 - \bar{f}_2(x) + \Lambda_4\delta_4 \\ &= \ddot{\chi}_d + \Lambda_3\big(\dot{\chi}_d - \dot{\chi}\big) + \delta_3 - \bar{f}_2(x) + \Lambda_4\delta_4 \\ &= \ddot{\chi}_d + \Lambda_3\big(v_3 - \Lambda_3\delta_3 + \delta_4 - v_3\big) + \delta_3 - \bar{f}_2(x) + \Lambda_4\delta_4 \\ &= \ddot{\chi}_d + \delta_3 - \Lambda_3^2\delta_3 + \Lambda_3\delta_4 + \Lambda_4\delta_4 - \bar{f}_2(x) \end{aligned} \tag{4.38}$$

where $\Lambda_4 = \text{diag}(\lambda_{10}, \lambda_{11}, \lambda_{12}) \in \mathbb{R}^{3\times3}$ contains positive entries. Substituting (4.35) and (4.38) into (4.34) yields

$$\begin{aligned} \dot{V}_4 &= -\delta_3^T\Lambda_3\delta_3 + \delta_4^T\big(v_4 + \bar{f}_2(x) - \Lambda_4\delta_4 - \bar{f}_2(x) - v_4\big) \\ &= -\delta_3^T\Lambda_3\delta_3 - \delta_4^T\Lambda_4\delta_4 \end{aligned} \tag{4.39}$$

such that $\dot{V}_4 \leq 0$, meaning the error terms δ_3, δ_4 are asymptotically stable, and thus the translational subsystem dynamics.

For our cascaded control design, the desired roll and pitch angles for the inner loop system are extracted from (4.37) after computing (4.38). Assume ψ is a measured state of the system. Then, the desired roll and pitch angles ϕ_d, θ_d and thrust u are obtained by solving (4.37), which gives

$$\begin{aligned} \theta_d &= \tan^{-1}\left(\frac{c_\psi u_x + s_\psi u_y}{u_z}\right) \\ \phi_d &= \tan^{-1}\left(\frac{s_\psi u_x - c_\psi u_y}{u_z}\cos\theta_d\right) \\ u &= \frac{m u_z}{\cos\phi_d \cos\theta_d} \end{aligned} \tag{4.40}$$

Remark 4.3 *In addition to the Euler angles limitations mentioned in* Remark 4.1, *the outer loop control law provides solutions if and only if the total thrust is a non-zero positive value, $u > 0$. This condition must be included within the constraints of the optimization problem in (2.11d) to avoid solution infeasibility.*

We can also perform the backstepping control design using the simplified system model (4.7). The second time derivative of the Euler angles vector η is written as

$$\ddot{\eta} = \bar{f}_3(x) + \bar{g}_3(\bar{\tau}) \tag{4.41}$$

where

$$\bar{f}_3(x) = \begin{bmatrix} \frac{J_y - J_z}{J_x}\dot{\theta}\dot{\psi} \\ \frac{J_z - J_x}{J_y}\dot{\phi}\dot{\psi} \\ \frac{J_x - J_y}{J_z}\dot{\phi}\dot{\theta} \end{bmatrix}, \quad \bar{g}_3(\bar{\tau}) = \begin{bmatrix} \frac{1}{J_x}\tau^{b_1} \\ \frac{1}{J_y}\tau^{b_2} \\ \frac{1}{J_z}\tau^{b_3} \end{bmatrix} \tag{4.42}$$

The terms δ_1, δ_2 and virtual input v_1 are defined exactly as in the full model backstepping design. Analogously to (4.24), the second virtual input v_2 for the (simplified) rotational dynamics is now assigned as

$$v_2 = \ddot{\eta}_d + (I - \Lambda_1^2)\delta_1 + (\Lambda_1 + \Lambda_2)\delta_2 - \bar{f}_3(x) \tag{4.43}$$

and since $v_2 = [\tau_\phi \quad \tau_\theta \quad \tau_\psi]^T = \bar{g}_3(\bar{\tau})$, the physical inputs are obtained from (4.41) as

$$\bar{\tau} = \begin{bmatrix} \tau_{b_1} \\ \tau_{b_2} \\ \tau_{b_3} \end{bmatrix} = \begin{bmatrix} J_x\tau_\phi \\ J_y\tau_\theta \\ J_z\tau_\psi \end{bmatrix} = \mathrm{diag}(J_x, J_y, J_z)\ v_2 \tag{4.44}$$

For the translational dynamics, define the position vector χ, whose second time derivative is written as $\ddot{\chi} = \bar{f}_4(x) + \bar{g}_4(x,u)$ where

$$\bar{f}_4(x) = \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} \tag{4.45}$$

$$\bar{g}_4(x,u) = \begin{bmatrix} (c_\phi s_\theta c_\psi + s_\phi s_\psi)\frac{u}{m} \\ (c_\phi s_\theta s_\psi - s_\phi c_\psi)\frac{u}{m} \\ (c_\phi c_\theta)\frac{u}{m} \end{bmatrix} \tag{4.46}$$

The definitions of δ_3, δ_4 and virtual input v_3 remain identical to the full model case. Assign the virtual control v_4 like (4.38)

$$v_4 = \ddot{\chi}_d + (I - \Lambda_3^2)\delta_3 + (\Lambda_3 + \Lambda_4)\delta_4 - \bar{f}_4(x) \tag{4.47}$$

where $\bar{f}_4(x)$ is given in (4.45). Since $v_4 = [u_x \quad u_y \quad u_z]^T = \bar{g}_4(x, u)$, assuming the yaw angle ψ is known, we solve this expression for the desired roll and pitch angles ϕ_d, θ_d and the total thrust u using equation (4.40).

4.2.3 NMPH-BSC Design of a Drone Vehicle

A copy of the full system model in (4.6) is used within the NMPH optimization problem as $\dot{\tilde{x}} = [\tilde{p}^n \quad \tilde{v}^n \quad \tilde{\eta} \quad \dot{\tilde{\eta}}]^T$. Let $\tilde{\chi} = \tilde{p}^n$ and $\dot{\tilde{\chi}} = \tilde{v}^n$ below. The backstepping control design representing $\tilde{u}(\tau) = g\big(\tilde{x}(\tau), \hat{\xi}_{ref}(\tau)\big)$ within the NMPH (2.11c) is implemented in two stages. The first stage is the outer loop, which takes the desired position vector of the drone, and computes the thrust plus the desired roll and pitch angles. The second stage is the inner loop, which takes the computed roll and pitch angles plus a desired yaw angle, and computes the torque inputs. The details of the two-stage process (implemented as input constraints within the NMPH) are as follows:

1. For the first NMPH input constraint, define the virtual input for the translational dynamics (4.38) as

$$\tilde{v}_4 = \ddot{\tilde{\chi}}_d + (I - \Lambda_3^2)\tilde{\delta}_3 + (\Lambda_3 + \Lambda_4)\tilde{\delta}_4 - \bar{f}_2(\tilde{x}) \tag{4.48}$$

   where $\tilde{\delta}_3 = \tilde{\chi}_d - \tilde{\chi}$ from (4.27) and $\tilde{\delta}_4 = \dot{\tilde{\chi}}_d + \Lambda_3\tilde{\delta}_3 - \tilde{\chi}$ from (4.30)

2. Associate the total thrust u and the desired roll and pitch angles ϕ_d, θ_d from (4.40) with $[u_x \quad u_y \quad u_z]^T = \tilde{v}_4$ and $\psi = \tilde{\eta}(3)$

3. For the second NMPH input constraint, define the virtual input for the rotational dynamics (4.24)

$$\tilde{v}_2 = \ddot{\tilde{\eta}}_d + (I - \Lambda_1^2)\tilde{\delta}_1 + (\Lambda_1 + \Lambda_2)\tilde{\delta}_2 - \bar{f}_1(\tilde{x}) \tag{4.49}$$

where $\tilde{\delta}_1 = \tilde{\eta}_d - \tilde{\eta}$ from (4.9) and $\tilde{\delta}_2 = \dot{\tilde{\eta}}_d + \Lambda_1\tilde{\delta}_1 - \tilde{\eta}$ from (4.12).

4. Associate the input torques τ^{b_1}, τ^{b_2} and τ^{b_3} from (4.25) with $[\tau_\phi \quad \tau_\theta \quad \tau_\psi] = \tilde{v}_2$ and $\phi = \tilde{\eta}(1)$, $\theta = \tilde{\eta}(2)$.

5. Let $[u \quad \tau^{b_1} \quad \tau^{b_2} \quad \tau^{b_3}]$ be $\tilde{u}(\tau)$ (2.11c) in the NMPH optimization problem, a function of the NMPH states $\tilde{x}$ and the estimated reference trajectories $\hat{\xi}_{ref}$, where

$$\hat{\xi}_{ref} = [\tilde{\chi}_d \quad \dot{\tilde{\chi}}_d \quad \ddot{\tilde{\chi}}_d \quad \tilde{\eta}_d(3) \quad \dot{\tilde{\eta}}_d(3) \quad \ddot{\tilde{\eta}}_d(3)]^T \tag{4.50}$$

6. Solve the optimization problem (2.11), which leads to the prediction of the system states $\tilde{x}$ and the estimated reference trajectories $\hat{\xi}_{ref}$. The latter is used as the reference trajectory for the actual closed-loop system.

4.3 Evaluation of NMPH-BSC

For testing and validation, the NMPH-BSC approach was implemented and tested in simulated and hardware flight tests on quadcopter and hexacopter drone vehicles, respectively.

The algorithms are implemented within the Robot Operating System (ROS) [95], a Linux-based software environment that handles communications between the vehicle's onboard computer and its hardware subsystems. The ACADO Toolkit [83] is used to solve the optimization problem. For implementation, the overall NMPH problem (2.11) was coded in C++, then converted into highly efficient C code by ACADO to be able to run the calculations in real-time.

The set of continuous-time equations in (2.11) is a Nonlinear Programming (NLP) optimization problem, which can be discretized using the direct multiple-shooting method. NLP solves optimization problems which include nonlinear functions and/or nonlinear constraints using Sequential Quadratic Programming (SQP) [85], and in our case the qpOASES solver is used to solve SQP numerically [86].

4.3.1 Simulation Environment

The proposed approach is first implemented in simulation on a quadcopter drone using the AirSim simulator [96]. The photo-realistic rendered environments and a physics engine of AirSim enable performing lifelike simulations of drone vehicles. Moreover, we used the PX4 autopilot [97] running onboard the drone for software-in-the-loop operation to make the simulated drone's characteristics more closely resemble the hardware unit.

In our work, an incremental volumetric mapping technique named Voxblox [104] is used. Voxblox represents the environment volumetrically using a signed distance field and classification into unknown, free, or occupied spaces. Voxblox is part of the global motion planning framework which will be presented in Chapter 5. The drone then uses this generated map to continuously build dynamic obstacle constraints that are used by the NMPH optimization problem to generate collision-free trajectories [79].

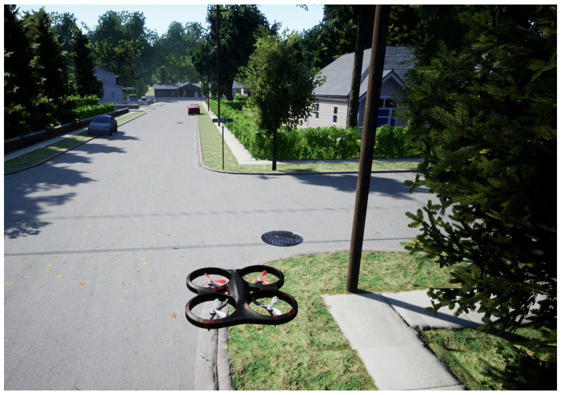
A desktop computer equipped with an Intel Core i7-10750H CPU and an NVIDIA GeForce RTX 2080 Super GPU is used to run the optimization calculations and simulation environment in conjunction with ROS. The prediction horizon of the optimization problem was set to 8.0 s and discretized into 40 samples, which was found sufficient for motion planning purposes. The cost function weights were empirically tuned to provide good trajectory planning performance.

The drone's pose and environmental sensor readings are obtained from the AirSim simulator and communicated to our NMPH-BSC trajectory planning system. The resulting output is used as a reference trajectory for the drone vehicle's flight controller.

4.3.1.1 Trajectory Planning

In this simulation, an optimal reference trajectory is planned and tracked within the AirSim simulator as shown in Figure 4.2a. The quadcopter vehicle starts at $p^n =$

$[-9, -3.5, 2]^T$ m, $\psi = 0°$ and the NMPH-BSC algorithm is used to generate an optimal trajectory to the desired setpoint $p_d^n = [-5, -8, 5]^T$ m, $\psi_d = 90°$ while avoiding an obstacle as shown in Figure 4.2b. The optimization problem within NMPH-BSC provides an estimate of the reference output trajectory $\hat{\xi}_{ref}$ and a prediction of the system state trajectory $\tilde{x}$, which includes the predicted output trajectory $\tilde{\xi}$ as a subset. Figure 4.3 shows that the estimated reference trajectories of the vehicle's position and velocities $\hat{\xi}_{ref} = [\hat{\xi}_{x,ref}, \hat{\xi}_{y,ref}, \hat{\xi}_{z,ref}, \hat{\xi}_{\psi,ref}, \hat{\xi}_{\dot{x},ref}, \hat{\xi}_{\dot{y},ref}, \hat{\xi}_{\dot{z},ref}, \hat{\xi}_{\dot{\psi},ref}]$ perfectly match their corresponding predicted reference trajectories $\tilde{\xi} = [\tilde{x}_{x,ref}, \tilde{x}_{y,ref}, \tilde{x}_{z,ref}, \tilde{x}_{\psi,ref}, \tilde{x}_{\dot{x},ref}, \tilde{x}_{\dot{y},ref}, \tilde{x}_{\dot{z},ref}, \tilde{x}_{\dot{\psi},ref}]$. This confirms that using the stage cost function in (2.5) minimizes the deviation between the estimated and predicted reference trajectories, and thus ensures their convergence towards each other. This validates the statement made in Section 2.2 that either the estimated or the predicted trajectory can be used as the reference trajectory for the closed-loop system.


(a)

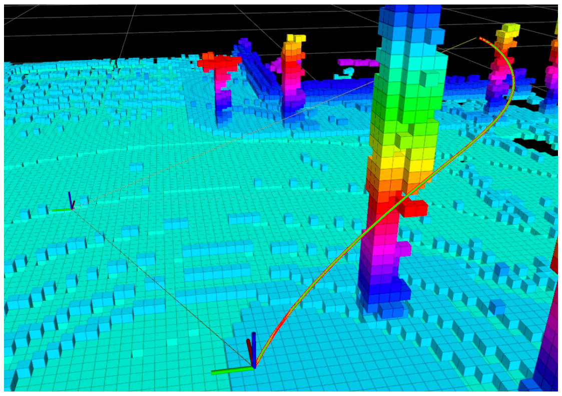
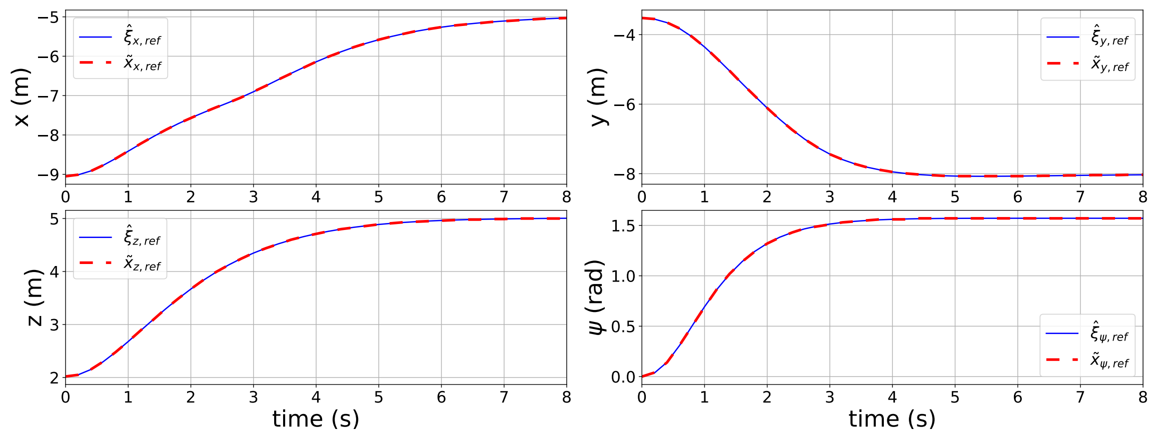

(b)

Figure 4.2: Trajectory Planning using the NMPH-BSC approach. (a) AirSim simulation environment. (b) Trajectory generation while avoiding static obstacle.


(a)

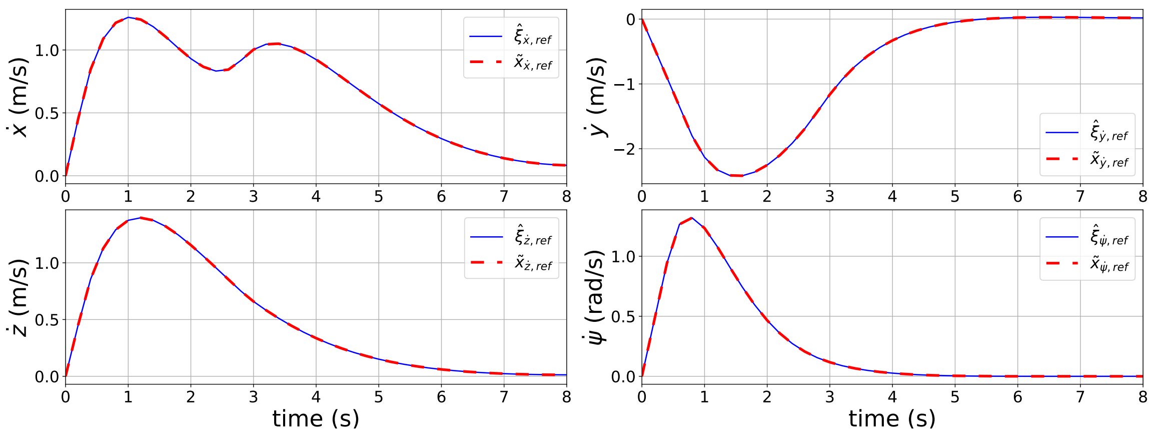

(b)

Figure 4.3: NMPH reference trajectory generation. The estimated and predicted reference position trajectories are depicted in (a), and the estimated reference velocity trajectories are shown in (b).

In the second simulation, tracking performance is assessed by having the vehicle move from the initial position $p^n = [0, 0, 2]^T$ m to the desired terminal point $p_d = [6, -4, 2]^T$ m using the optimized reference trajectory provided by the NMPH-BSC. The tracking performance is plotted in Figure 4.4 showing the vehicle satisfactorily tracks the time-varying reference trajectory generated by the NMPH-BSC algorithm. The small variation between the desired and actual outputs is because the former are obtained from numerical integration of the drone dynamics (4.6), while the latter are obtained from the physics engine of the simulation environment, which likely uses more complicated aerodynamic force and torque models than our design. This modeling mismatch can also be expected for real-world hardware testing, which will be covered in Section 4.3.2.

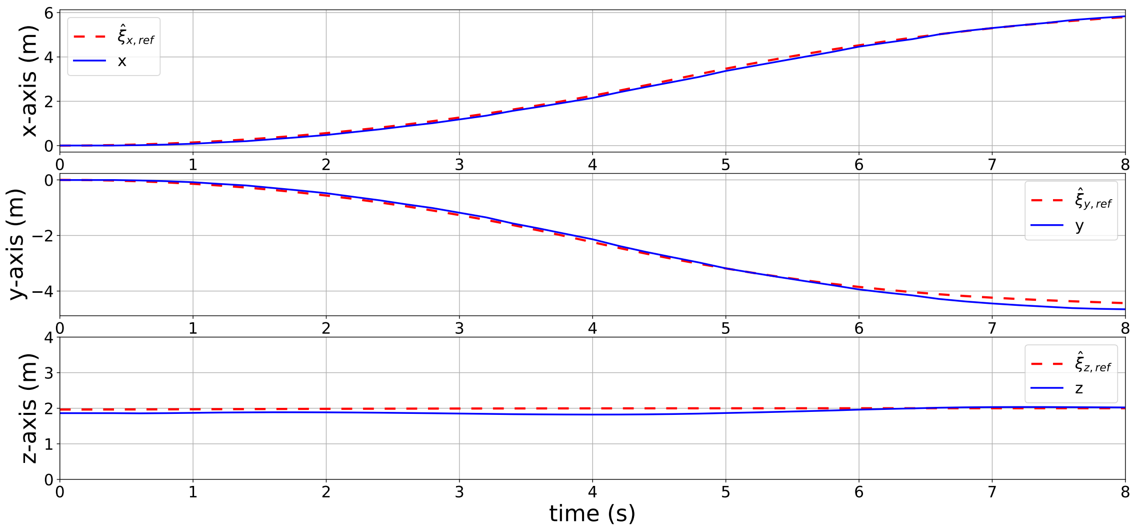


Figure 4.4: Vehicle's trajectory tracking performance between the start position $(0, 0, 2)$ m and the terminal setpoint $(5.8, -4.5, 2)$ m.

4.3.1.2 Exploration of Unknown Environment

In this simulation test, the drone explores an unknown environment by using a modular global motion planner, as described in Chapter 5. This graph-based motion planner generates terminal setpoints within unexplored areas of an incrementally built-up volumetric map of the environment [8, 104], and the NMPH-BSC algorithm is used to calculate optimal trajectories from the vehicle's current pose to these terminal setpoints. To achieve a smooth integration between the graph-based planner and the NMPH-BSC trajectory planning approach, computationally efficient algorithms for obstacle mapping and avoidance plus robust path guidance algorithms are used. Further details of this methodology are discussed thoroughly in Chapter 5.

Figure 4.5 depicts a part of the exploration mission performed by the drone. The vehicle explored an unknown environment using the global motion planner in conjunction with the NMPH-BSC for local trajectory planning, which leads to smoother flight trajectories than the stop-and-go patterns obtained from the motion planner alone. The data was collected over an exploration time of 1002 s. Table 4.1 offers a comparison between the Graph-based planner only [8] and the Graph-based planner plus NMPH-BSC. Based on this comparison, it can be seen that NMPH-BSC algorithm has the effect of reducing the distance traveled, which reduces the mission time and consequently the energy consumption of the vehicle.

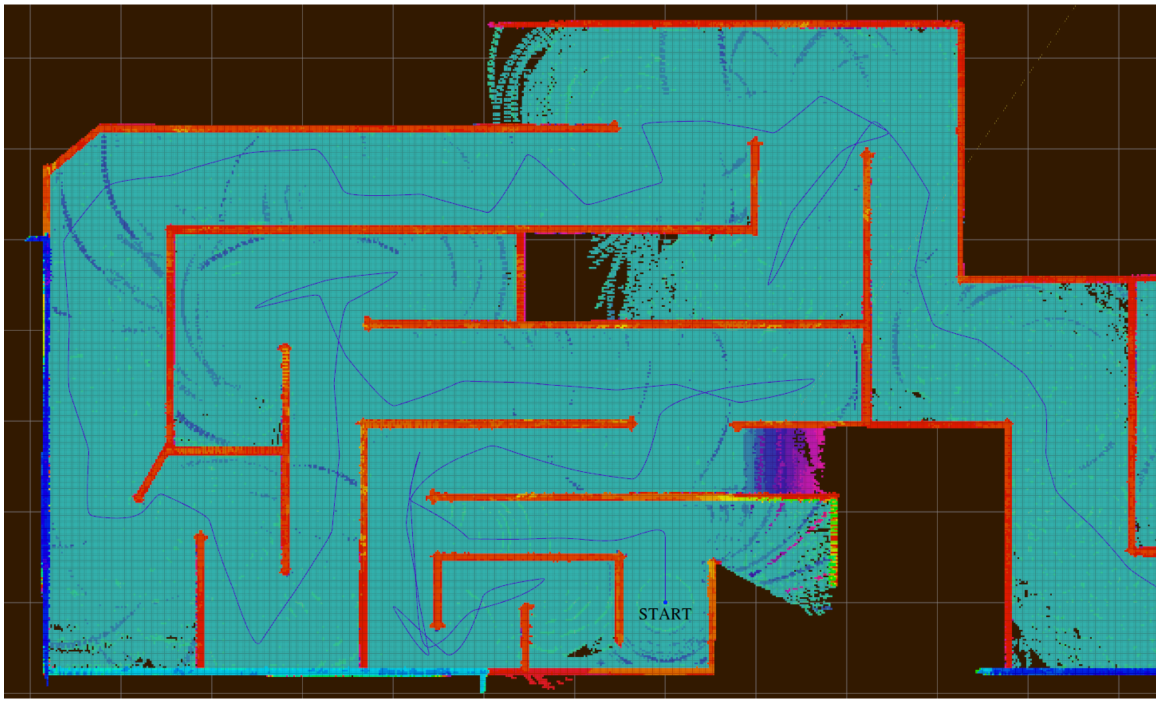


Figure 4.5: Exploration of unknown environment using global motion planner using NMPH-BSC for local trajectory planning.

Table 4.1: Comparison between Graph-based and Graph-based-plus-NMPH-BSC to motion planning

	Total length of the generated paths	Average path length (between terminal points)	Exploration time	Continuous path generation
Graph-based	1252 m	8.03 m	1322 s	No
Graph-based & NMPH-BSC	**949 m (24.2% improvement)**	**6.08 m (24.3% improvement)**	**1002 s (24.2%)**	**Yes**

4.3.2 Hardware Flight Experiment

For hardware experiments, a custom DJI FlameWheel F550 hexacopter was built and instrumented to test our proposed NMPH-BSC approach for local planning. The drone is equipped with a Pixhawk 2.1 flight controller running the PX4 autopilot system [97], plus an NVIDIA Jetson Xavier NX system-on-module running ROS Melodic Morenia [95] under Ubuntu 18.04. A Velodyne Puck LITE LiDAR sensor and an Intel RealSense T265 stereo camera are mounted on the drone to provide 360° point cloud and estimated pose data, respectively. The drone system used for testing is depicted in Figure 4.6.

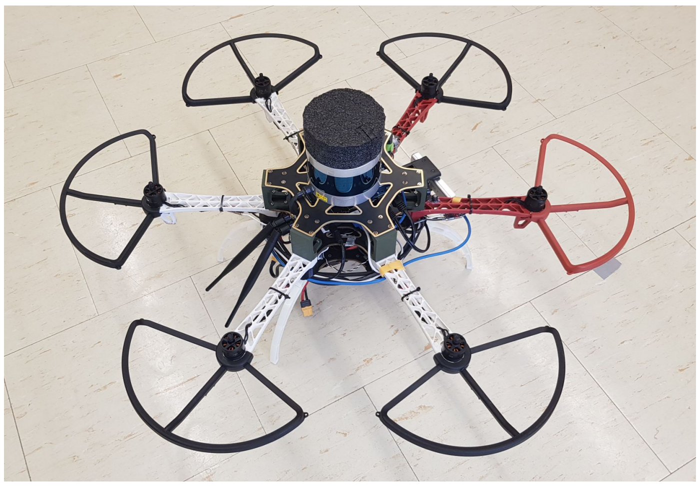


Figure 4.6: DJI FlameWheel F550 hexacopter vehicle equipped with onboard sensors and computing systems.

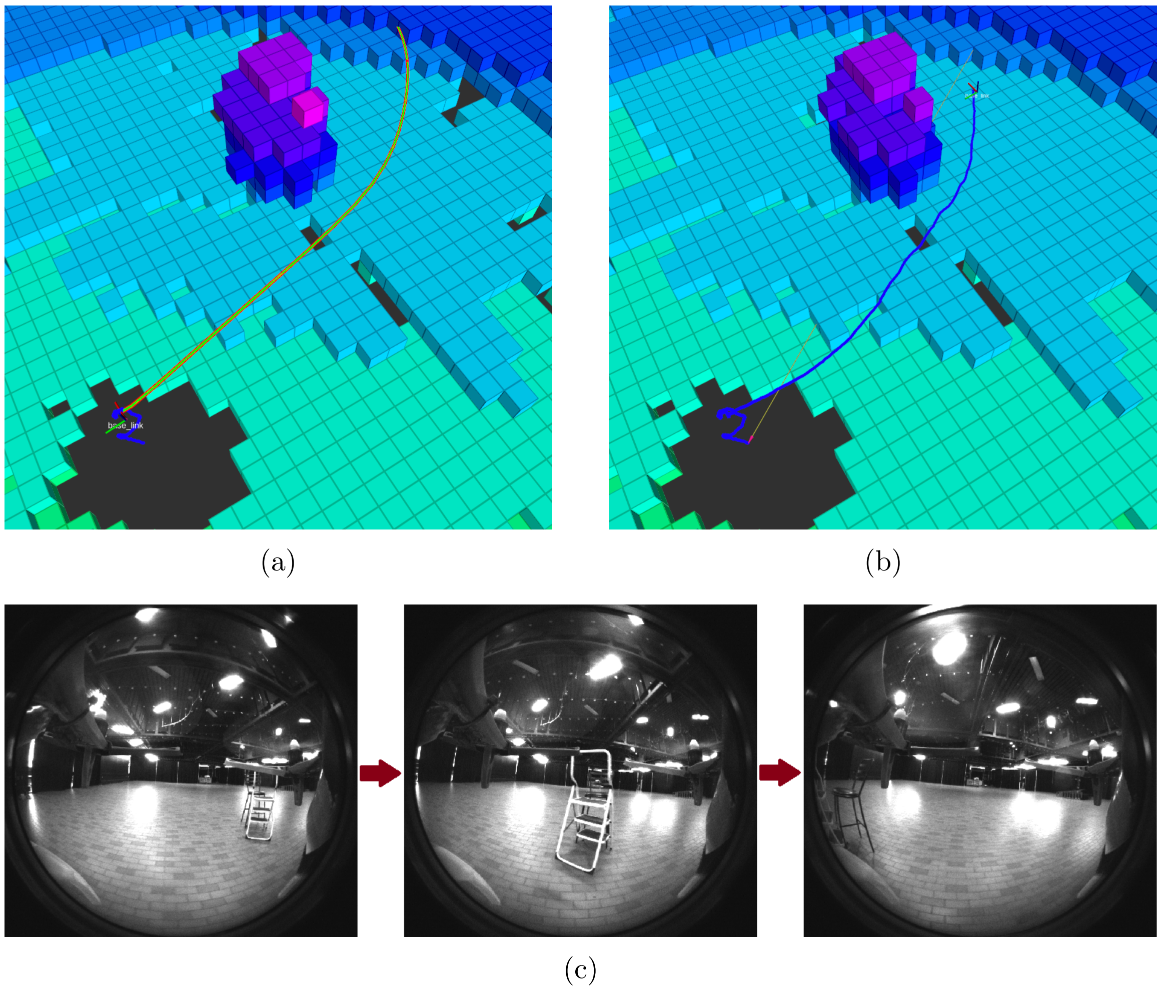


Figure 4.7: Trajectory planning with obstacle avoidance flight test using the NMPH-BSC algorithm. (a) Trajectory generation. (b) Trajectory tracking. (c) The mapped obstacle as seen from the left fisheye lens of the onboard camera.

In this preliminary flight test, trajectory generation and tracking were evaluated by running the NMPH-BSC algorithm onboard the vehicle. Figure 4.7a shows the planned trajectory between two setpoints avoiding a sensed obstacle generated by the NMPH-BSC algorithm, while Figure 4.7b depicts the flight trajectory achieved by the drone using its flight controller to track the planned trajectory. The NMPH-BSC solver provides continuous updates of the estimated and predicted reference trajectories as the vehicle moves towards its endpoint. In this way, the system can handle uncertainties and disturbances such as dynamic environments and moving obstacles. The regeneration rate was set to 5 Hz, although this can be set as high as

100 Hz with the presented hardware.

Trajectory planning using NMPH-BSC in the presence of dynamic obstacles was evaluated experimentally as shown in Figure 4.8. In this Figure, it can be seen that the drone was able to regenerate trajectories to avoid a moving obstacle while flying through a constrained indoor environment. It is worth pointing out that the continuous trajectory regeneration process of NMPH-BSC provided smooth transitions between the generated trajectories, leading to smooth flight around the moving obstacle.

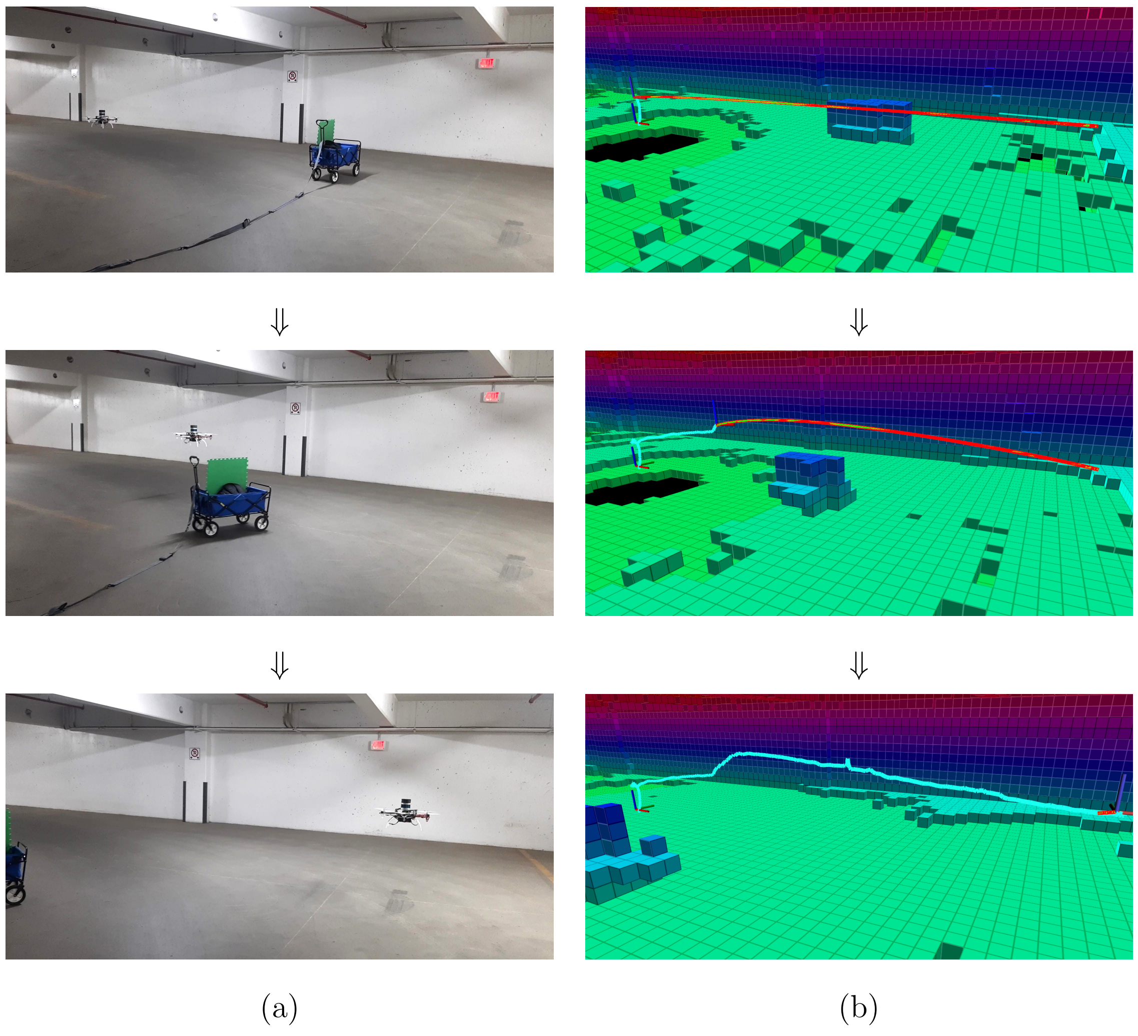


Figure 4.8: Hardware flight test for trajectory planning involving a dynamic obstacle using the NMPH-BSC algorithm. (a) Hardware drone avoiding the moving obstacle. (b) RViz visualization of the trajectory regeneration and flight path.

4.4 Conclusions

In this chapter, we presented an optimization-based trajectory planning approach for drones operating in unknown environments. The proposed method embeds the Backstepping Control technique within our proposed Nonlinear Model Predictive Horizon framework [77] for generating reference trajectories for nonlinear dynamical systems. This integration reduces the non-convexity of the optimization problem and thus enables real-time computation of optimal trajectories which respect the nonlinear dynamics of the vehicle while avoiding static and dynamic obstacles.

The resulting NMPH-BSC design was tested in simulation and hardware flight experiments on quadcopter and hexacopter drone vehicles, respectively. The results showed an improvement in performance over the conventional path planning algorithms. The new design was shown to offer additional implementation advantages over NMPH-FBL including the ability to readily extend to more complicated plant models and avoiding numerical differentiation.

Chapter 5

Optimal Motion Planning for Exploration of GPS-denied Environments using Nonlinear Model Predictive Horizon

5.1 Introduction

NMPH algorithm provides feasible solutions, generates smooth and collision-free paths, supports moving obstacles, is able to run in real-time, and reduces battery draw by minimizing abrupt drone motions. The goal of this chapter is developing an NMPH-based global motion planner for the drone to explore a subterranean environment. This operates by building a map of the environment and guiding the vehicle to unexplored areas within this map using a graph-based planner. The global motion planner is a design that integrates the local NMPH path planning design with a graph-based planner named GBPlanner [8, 105, 106]. We propose a choice of computationally efficient algorithms for obstacle mapping and avoidance, plus robust path guidance. A block diagram of the proposed global motion planner design is shown in Figure 5.1.

The contributions of the work presented in this chapter are as follows:

- The NMPH approach is integrated inside a global motion planner and produces optimal local trajectories for the drone vehicle in real-time.
- A methodological three-stage global motion planner design is proposed. The first stage operates by building an incremental volumetric map of the environ-

ment. The second stage utilizes a graph-based planner to generate terminal setpoints within the map for the third stage, which uses the NMPH design to generate continuous optimal paths from the vehicle's current pose to the terminal setpoint in real time.

- Efficient algorithms for *obstacle mapping* and avoidance are proposed which produce models of static and dynamic obstacles used by the NMPH to generate safe and collision-free paths in a dynamically changing environment.
- A robust path guidance algorithm is implemented to avoid the risk of NMPH getting trapped into a local minima.
- The overall design is implemented using quadcopter and hexacopter drone dynamics, enabling navigation through unknown, dynamic and GPS-denied environments.
- Several simulation and experimental results are presented in this chapter to validate the proposed approach.

The remainder of this chapter is organised as follows; the three-layer global motion planner design and choices of algorithms to provide robust path planning and obstacle avoidance are discussed in Section 5.2. In Section 5.3, various simulation and experimental results are presented to evaluate and validate the proposed approach. Finally, concluding remarks are given in Section 5.4.

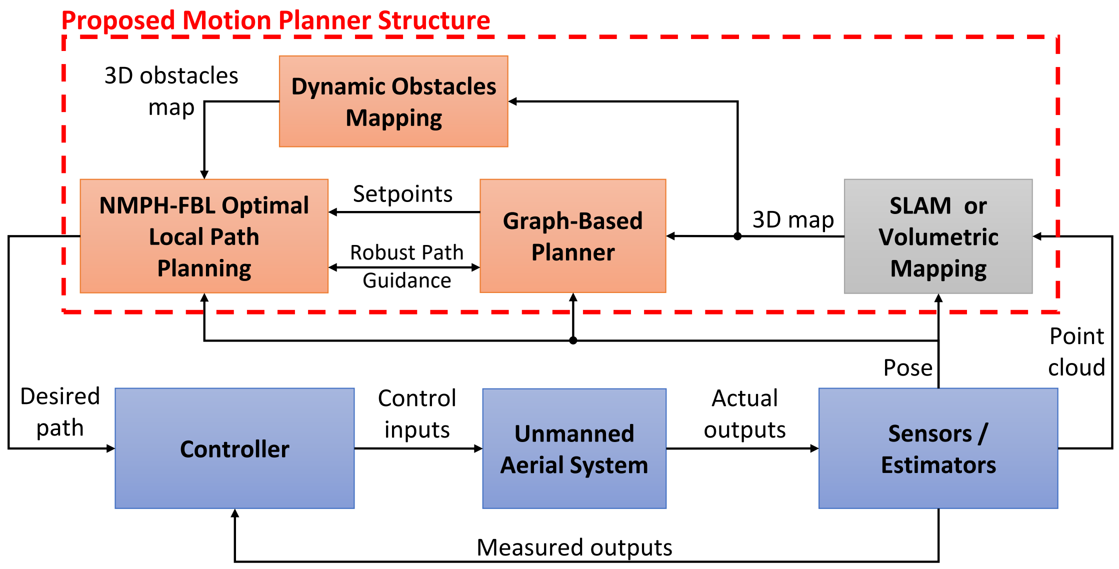


Figure 5.1: Block diagram of the proposed global motion planner.

5.2 Motion Planning in GPS-denied Environments

Our proposed motion planning design aims to produce optimal vehicle paths while navigating in unexplored, dynamic and GPS-denied environments. We combine a graph-based exploration technique with a Nonlinear Model Predictive Horizon-based approach based on optimization which respects the vehicle's dynamics and supports dynamic obstacles. This integration yields feasible, optimal, and robust paths while exploring challenging environments.

5.2.1 Motion Planner Architecture

Figure 5.2 describes the overall architecture of our motion planner. The design is composed of three successive stages. The first stage acquires sensor data to build a physical representation of the environment which contains both static and dynamic objects in it. Volumetric mapping is used for this stage since it is computationally efficient, easy to visualize, can be incrementally constructed and reconstructed online, and provides the voxel grid structure needed for the next stage. Details about the volumetric mapping and its layers will be discussed in Section 5.2.2.

Section 5.2.3 discusses the second stage of the motion planner, which is built around a graph-based planning approach. It consists of the sampling-based RRG algorithm, which builds a connected roadmap graph, and the Dijkstra searching algorithm to extract the best path from within the graph. The main purpose of the graph-based approach is to guide the vehicle towards unexplored areas within the environment and provide terminal vertices, a.k.a. stabilization or terminal setpoints, to the local path planner.

The third stage of the motion planner uses the NMPH local path planning method. Fusing this method with the earlier stages improves the robustness of generating optimal paths and avoiding static and dynamic obstacles. The considerations involved in finding a feasible path are shown in Figure 5.2. Further details are provided in Section 5.2.4.

The reference trajectory computed by the path planner is fed to the control system of the vehicle for tracking purposes. As the drone vehicle moves, the NMPH continues to update its reference trajectory in response to feedback of the vehicle's state and new obstacles. Once the vehicle reaches a setpoint, the motion planning process is repeated, which continues until the environment is fully explored or the mission is interrupted by the operator.

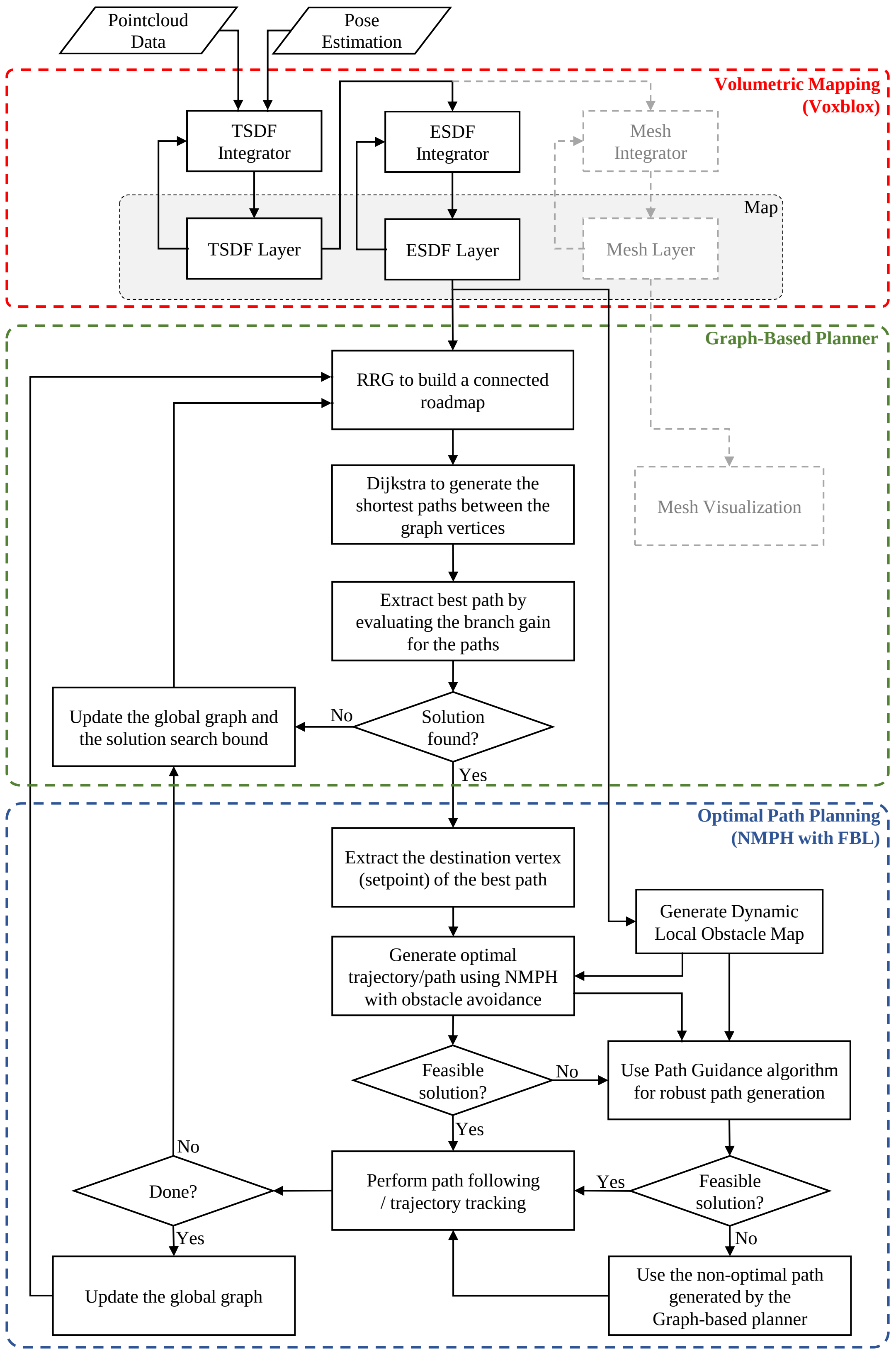

Pointcloud Data
Pose Estimation
Volumetric Mapping (Voxblox)
TSDF Integrator
ESDF Integrator
Mesh Integrator
Map
TSDF Layer
ESDF Layer
Mesh Layer
Graph-Based Planner
RRG to build a connected roadmap
Dijkstra to generate the shortest paths between the graph vertices
Mesh Visualization
Extract best path by evaluating the branch gain for the paths
Update the global graph and the solution search bound
No
Solution found?
Yes
Optimal Path Planning (NMPH with FBL)
Extract the destination vertex (setpoint) of the best path
Generate Dynamic Local Obstacle Map
Generate optimal trajectory/path using NMPH with obstacle avoidance
Feasible solution?
No
Use Path Guidance algorithm for robust path generation
Yes
No
Done?
Perform path following / trajectory tracking
Yes
Feasible solution?
Yes
No
Update the global graph
Use the non-optimal path generated by the Graph-based planner

Figure 5.2: Motion Planner Architecture. The Mesh blocks are used for visualization of the environment.

5.2.2 Volumetric Mapping

Volumetric mapping is the foundation of motion planning and navigation strategies in 3D environments. The volumetric mapping algorithm named Voxblox [104] is used in our work. In this technique, the map of the environment is represented volumetrically using the signed distance field to distinguish between known, unknown, free, and occupied spaces [107]. The grid consists of voxels with a corresponding type. Groups of occupied voxels represent surfaces of an object, walls, and so on. The main advantage of volumetric mapping is its real-time capability for incrementally representing unstructured and unexplored environments, which makes it a suitable solution for online planning and exploration. The Truncated Signed Distance Field (TSDF [108]) is one of the most efficient methods of representing volumetric maps. TSDF is an implicit surface representation that consists of a 3D voxel array. Each voxel is indexed by the distance of the ray between the sensor and the surface, and is truncated near the surface to decrease the errors that are caused by sensor noise. TSDFs are computationally efficient and can be constructed online. Also, they are capable of filtering out sensor noise and create meshes with voxel resolution for visualization purposes.

In contrast to TSDF, the Euclidean Signed Distance Field (ESDF) uses the Euclidean distance to the nearest occupied cell in labeling the voxel grid [104]. ESDFs are directly built out of existing TSDFs to make use of the distance information in determining the obstacle surface location for planning purposes. In other words, TSDF is for mapping and ESDF for planning, and the main difference between them is the way that distances are computed [109].

As presented in Figure 5.2, the volumetric mapping process consists of three layers. The sensor data is processed to build the TSDF layer, then the voxels are integrated into the ESDF and mesh layers as presented in [104]. The ESDF voxels and mesh blocks are updated incrementally allowing real-time map generation for planning and online visualization of the environment. To reduce the complexity of calculating the layers data, a voxel hashing approach [110] is used to store the information of each layer in a hash table, which results in $O(1)$ complexity for adding or retrieving the data.

5.2.3 Graph-based Path Planning

In this section, we summarize the graph-based planner presented in [8], which is used to help the vehicle navigate through unknown GPS-denied environments.

Assume that $\mathbb{M}_G$ is a global 3D voxel-based map, which consists of voxels $m \in \mathbb{M}_G$. The map is incrementally built by a depth sensor $\mathbb{S}$ plus the vehicle's pose estimation using the volumetric mapping approach previously discussed in Section 5.2.2. The map is categorized into three spaces, free space $\mathbb{M}_G^f$, occupied space $\mathbb{M}_G^o$, and unknown space $\mathbb{M}_G^u$. The map has a global volume V_G and is incrementally constructed within a local map sub-space $\mathbb{M}_L$ of volume V_L centered around the current vehicle's output (here 3D position and heading) $\xi_0 = \begin{bmatrix} x_0 & y_0 & z_0 & \psi_0 \end{bmatrix}^T$.

The graph-based planner [8] performs a local search towards unknown areas of $\mathbb{M}_G$. It is based on the sampling-based RRG algorithm [111] which builds a connected roadmap graph $\mathbb{G}_G$ composed of collision-free vertices ν and edges e stored in vertex set $\mathbb{V}$ and edge set $\mathbb{E}$, respectively. The edges are straight paths connecting the vertices using the nearest neighbor search [112]. The global graph $\mathbb{G}_G$ is continuously constructed from the undirected local graph $\mathbb{G}_L$ within the local space $\mathbb{M}_L$. The local search within the bounded volume V_L considers the physical size of the vehicle V_R and bounds it within a sub-space $\mathbb{M}_R$. Collision detection is performed to ensure collision-free paths σ_L, where $\mathbb{M}_R \in \mathbb{M}_G^f$ for all randomly generated vertices and edges.

The set of all shortest paths Σ_L, starting from the initial or current vertex ξ_0 to all destination vertices ξ_ν, is found using the Dijkstra algorithm [13]. After that, the best path is evaluated by calculating the Volumetric Gain $\mathcal{V}$ for each vertex. The Volumetric Gain of a vertex is a measure of the unmapped volume based on the depth reading around that vertex. The weight functions related to distance and direction combined with $\mathcal{V}$ are used to compute the Exploration Gain $\mathcal{E}(\sigma_i)$ for all $\sigma_i \in \Sigma_L$, $i = 1, \ldots,$ n. The vertices of these paths are ν_j^i , $j = 1, \ldots, m_i$, and ν_0^i is the initial vertex along path σ_i. The Exploration Gain for a path is calculated as [8]

$$\mathcal{E}(\sigma_i) = e^{-\lambda_{\mathcal{S}} \ \mathcal{S}(\sigma_i, \sigma_{sp})} \sum_{j=1}^{m_i} \mathcal{V}\left(\nu_j^i\right) \ e^{-\lambda_{\mathcal{D}} \ \mathcal{D}(\nu_0^i, \nu_j^i)} \tag{5.1}$$

where $\mathcal{S}(\sigma_i, \sigma_{sp})$ is a distance factor between a path σ_i and its corresponding straight path σ_{sp} which has the same length along the estimated exploration direction. This factor prevents the vehicle from sudden changes in its exploration direction which might happen in branched environments within $\mathbb{M}_L$. $\mathcal{D}(\sigma_i, \sigma_{sp})$ is the distance between ν_j^i and ν_0^i of the path σ_i, which penalizes longer paths for a higher exploration rate. The tunable gains $\lambda_{\mathcal{S}}$ and $\lambda_{\mathcal{D}}$ are positive gains.

Subsequently, the best path σ_{best} that maximizes the Exploration Gain is selected and sent to the NMPH local motion planner to find the optimal path that the vehicle will follow. The whole procedure is repeated once the vehicle reaches the destination vertex. The detailed algorithm for building the map and planning the best path is presented in Algorithm 5.1.

If all vertices within $\mathbb{G}_L$ are explored, the search will be expanded to the unexplored vertices of $\mathbb{G}_G$. This will guide the vehicle to another location on the global map and continue the exploration mission. For the return-to-home feature, the Dijkstra algorithm is also used to find the shortest path between the vehicle's current output ξ_0 and the homing vertex on $\mathbb{G}_G$. This feature can be invoked once the exploration mission is completed, the battery level is low, or by intervention from the operator.

5.2.4 NMPH for Local Path Planning

The graph-based planner in Section 5.2.3 generates non-optimal or sub-optimal paths because the vertices are created randomly within V_L. In addition, the straight edges connecting vertices cause jerky motions for a drone following the path. Finally, the

Algorithm 5.1 Graph-based Planner

```
1: $\xi_0 \leftarrow$ **CurrentMeasurement**();
2: $\mathbb{M}_G \leftarrow$ **VolumetricMapping**($\mathbb{S}$);
3: $\mathbb{V} \leftarrow \{\xi_0\}$;   $\mathbb{E} \leftarrow \emptyset$;   $\mathbb{G}_L = (\mathbb{V}, \mathbb{E})$;
4: $\mathbb{M}_L \leftarrow$ **LocalBoundSpace**($\xi_0, \mathbb{M}_G$);
5: **for** $i = 1, \dots, n$ **do**                          ▷ RRG to build the local graph $\mathbb{G}_L$
6:     $\xi_{rand} \leftarrow$ **SampleFree**$_i$($\mathbb{M}_L$);
7:     $\xi_{nearest} \leftarrow$ **NearestVertex**($\mathbb{G}_L, \xi_{rand}$);
8:     **if CollisionFree**($\xi_{rand}, \xi_{nearest}$) **then**
9:         $\mathbb{V} \leftarrow \mathbb{V} \cup \{\xi_{rand}\}$;
10:        $\mathbb{E} \leftarrow \mathbb{E} \cup \{\xi_{rand}, \xi_{nearest}\}$;
11:        $\mathbb{N}_{near} \leftarrow$ **NearVertices**($\mathbb{G}_L, \xi_{rand}$);
12:        **for each** $\xi_{nearest} \in \mathbb{N}_{near}$ **do**
13:            **if CollisionFree**($\xi_{rand}, \xi_{near}$) **then**
14:                $\mathbb{E} \leftarrow \mathbb{E} \cup \{\xi_{rand}, \xi_{near}\}$;
15:            **end if**
16:        **end for**
17:    **end if**
18: **end for**
19: $\Sigma_L \leftarrow$ **DijkstraAlgorithm**($\mathbb{G}_L, \xi_\nu$);          ▷ Find the shortest paths
20: $\sigma_{best} \leftarrow \emptyset$;   $\mathcal{E}_{best} \leftarrow 0$;
21: **for each** $\sigma \in \Sigma_L$ **do**                                ▷ Find the best path
22:    $\mathcal{E}_\sigma \leftarrow$ **ExplorationGain**($\sigma$, **VolumetricGain**($\mathbb{V}$));
23:    **if** $\mathcal{E}_\sigma > \mathcal{E}_{best}$ **then**
24:        $\sigma_{best} \leftarrow \sigma$;   $\mathcal{E}_{best} \leftarrow \mathcal{E}_\sigma$;
25:    **end if**
26: **end for**
27: $\mathbb{G}_G \leftarrow \mathbb{G}_G \cup \mathbb{G}_L$;                          ▷ Update the global graph
28: **if** $ReturnHome = true$ **then**
29:    $\sigma_{best} \leftarrow$ **DijkstraAlgorithm**($\mathbb{G}_G, \xi_{home}$);   ▷ Find the shortest path to home
30: **end if**
```

path generated by the graph-based planner does not respect the vehicle's dynamics. The NMPH-equipped path planning approach presented in Algorithm 5.2 overcomes these issues by generating a path which respects the system's dynamics and provides a smooth and optimal path which also avoids obstacles. From Figure 5.2, the NMPH path planning stage includes

- *Dynamic Local Obstacle Mapping* (c.f. Section 5.2.4.1), a technique which utilizes the continously updated volumetric map of the environment to generate a dynamically changing map of obstacles which are used as constraints for the optimization within the NMPH algorithm.

- *Obstacle Avoidance* (c.f. Section 5.2.4.2), an algorithm which allows the optimization problem solver to select constraints which correspond to obstacles in the path of the vehicle.

- *Path Guidance* (c.f. Section 5.2.4.3), an algorithm which enhances the robustness of path generation to situations by making use of all the vertices of the graph-based planner-generated path, not just the terminal vertex. This allows the generation of multiple consecutive and feasible paths, leading to an overall path to the terminal vertex.

5.2.4.1 Dynamic Local Obstacle Mapping

Transforming physical obstacles within the volumetric map to optimization constraints is a challenging task. These obstacles need to be represented by a cluster of constraints while respecting the limitations of the optimization process, specifically a limit on the number of inequality constraints that the optimization problem can handle.

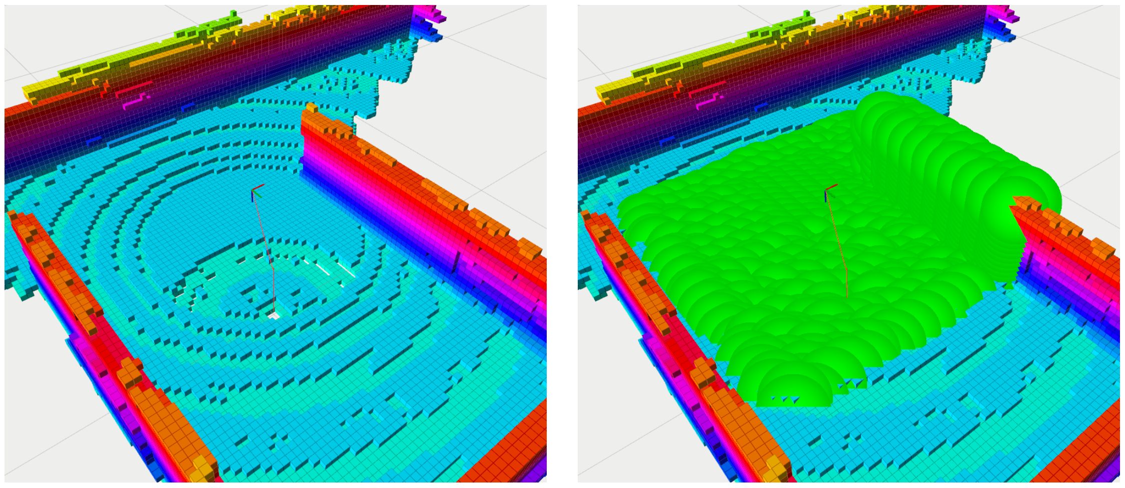
In this Section, we will present a strategy that maps obstacles in the environment into a dynamically moving space around the vehicle. This facilitates representing the obstacles as inequality constraints for optimization. This mapping technique, called

Algorithm 5.2 Local Optimal Path Planning using NMPH

```
1: $\sigma_{best} \leftarrow \textbf{GraphBasedPlanner}(\xi_0, \mathbb{M}_G)$;
2: $\nu_{term} \leftarrow \textbf{ExtractVertex}_{terminal}(\sigma_{best})$;
3: $\mathbb{M}_{obs} \leftarrow \textbf{LocalObstacleBound}(\xi_0, \mathbb{M}_G)$;        ▷ Consider certain voxels around $\xi_0$
4: for $i = k, \ldots, n$ do                                            ▷ Remove extra voxels
5:     for $j = i - k, \ldots, i$ do
6:         if $\|m_i - m_j\| < \delta$ then
7:             $\mathbb{M}_{obs} \leftarrow \mathbb{M}_{obs} \backslash m_i$;          ▷ Remove $\nu_i$ from the obstacles map
8:         end if
9:     end for
10: end for
11: $\mathbb{C}_L \leftarrow \textbf{ObstacleConstraint}(\nu_{term}, \xi_0, \mathbb{M}_{obs})$;     ▷ Find the obstacles constraints
12: $\sigma_{opt} \leftarrow \textbf{NMPH\_Planning}(\nu_{term}, \xi_0, \mathbb{C}_L)$;
13: if $\sigma_{opt}$ not $feasible$ then
14:     for $i = 1, \ldots, n$ do                                      ▷ Path Guidance Algorithm
15:         $\nu_i \leftarrow \textbf{ExtractVertex}_i(\sigma_{best})$;
16:         $\sigma_{opt} \leftarrow \textbf{NMPH\_Planning}(\nu_i, \xi_0, \mathbb{C}_L)$;
17:     end for
18:     if $\sigma_{opt}$ not $feasible$ then
19:         $\sigma_{opt} = \sigma_{best}$;
20:     end if
21: end if
22: $\textbf{PathFollowing}(\sigma_{opt})$;                                ▷ Follow the optimal path
```

Dynamic Local Obstacle Mapping (DLOM), generates a continuously changing map $\mathbb{M}_{obs}$.

Based on the occupied voxel in $\mathbb{M}_G^o$, the DLOM strategy generates virtual spheres centered on occupied voxels within a certain space surrounding the vehicle (e.g., a box of dimensions D_{obs}). These virtual spheres have a radius which ensures a safe clearance between the vehicle sides and the occupied voxel. Figure 5.3 shows the volumetric map without and with DLOM. One advantage of using a sphere is for modeling the obstacle as a state constraint. This inequality constraint requires r^i_{obs}, the distance between the vehicle and the center of the i^{th} sphere, to be larger than a specific threshold r_{thld} representing the radius of the virtual sphere as $(r^i_{obs} \geq r_{thld})$. The solution of the optimization problem within NMPH will thus generate a path that doesn't pass through the virtual spheres and hence avoids the obstacles in the environment.



(a) Without DLOM (b) With DLOM

Figure 5.3: Dynamic Local Obstacle Mapping (DLOM). The virtual drone is flying through AirSim's *SimpleMaze* environment; further information will be provided in Section 5.4.

Modeling all occupied voxels in D_{obs} as obstacles would result in an excessively large computational burden to continuously generate $\mathbb{M}_{obs}$ and solve the optimization problem. Instead, any voxels inside the i^{th} sphere are excluded from $\mathbb{M}_{obs}$. Lines (3-

10) of Algorithm 5.2 employ a simple running window strategy to remove extra voxels, and those remaining are represented as virtual spheres which provide constraints to the optimization problem. Figure 5.4 shows how the extra spheres are removed to reduce the computational load involved in producing the obstacles map. The exact time needed to build the dynamic obstacles map depends on the number of occupied voxels within D_{obs}. Experimentally, we found that the time required to build such a map on a desktop-class machine with a modern GPU (detailed specifications are given in Section 5.4) takes approximately 3 ms.

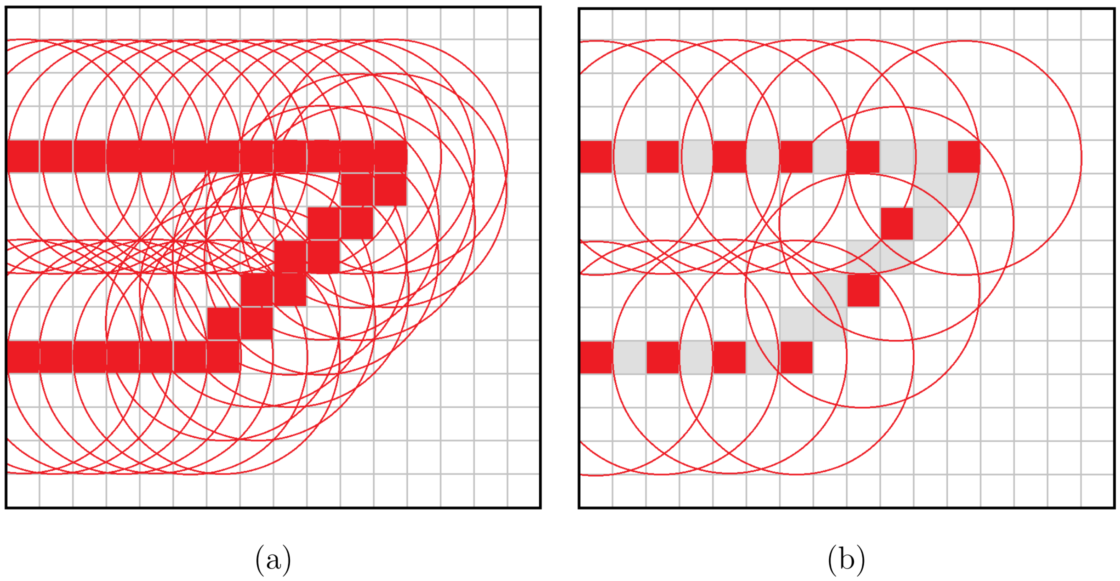


Figure 5.4: Dynamic Local Obstacle Mapping. (a) All voxels are used to map the obstacle's surface. (b) A subset of voxels (highlighted in red) is selected to represent the obstacle's surface, and their neighbouring voxels are excluded.

5.2.4.2 Obstacle Avoidance Algorithm

As soon as the obstacle map is created, the NMPH creates an optimal local path respecting the constraints acquired from $\mathbb{M}_{obs}$. The optimization solver is limited in the number of inequality constraints it can handle, making it impossible to include all the mapped obstacles in $\mathbb{M}_{obs}$ within the optimization problem. Hence, a dynamic method for selecting a specific number of constraints is described next and included in Algorithm 5.3.

Our chosen solver provides a solution to the optimization problem every ~ 4 ms (running on the computer described in Section 5.4), which makes it possible to solve the problem several times before sending the optimum reference path to the vehicle's flight control system. The Obstacle Avoidance algorithm takes advantage of this by first solving the optimization problem without considering obstacle constraints, then running a collision check on the generated path to find whether it crosses any virtual spheres in $\mathbb{M}_{obs}$. It is important to mention that the collision check is performed over the entire optimization horizon $[t_n, t_n + T]$ in Algorithm 2.2, which is discretized into N points for numerical computation.

If a collision is detected at some points within the optimization horizon, a *Dynamic Constraint Array* registers the center of a sphere $\mathbf{s} \in \mathbb{R}^3$ that contains these collision points, and passes them to the solver as inequalities used to compute a new solution which avoids them. The *Dynamic Constraint Array* has dimensions of $N \times 3$ and can register up to N different inequality constraints for the next run of the optimization problem. For example, assume that a collision is detected on horizon points 3, 4 and 5, and each of the collision points are located within the 40th virtual sphere. In this case, the coordinates of the center of this sphere are registered in the *Dynamic Constraint Array* at indices 3, 4 and 5, while the rest of the array entries are kept Null. In the next iteration of the solver, a new constraint representing the cloned entries of the Dynamic Constraint Array will yield a path which avoids the region where the collisions previously occurred.

To enhance the reliability of the Obstacle Avoidance algorithm while the vehicle is in motion, a specific number of *Temporary Constraint Arrays* (labeled by k in Algorithm 5.3) store the information from the *Dynamic Constraint Array* and are used in the optimization solution as well. The Temporary Constraint Arrays are static, which means that each registers only one virtual sphere over all its N indices.

Algorithm 5.3 Obstacle Constraints

```
1: function ObstacleConstraint($\nu_{term}, \xi_0, \mathbb{M}_{obs}$)
2:     $\sigma_{opt} \leftarrow$ NMPH_Planning($\nu_{term}, \xi_0$);
3:     $k = 1$;
4:     $\mathbb{C}_D \leftarrow \emptyset$; $\mathbb{C}_T^k \leftarrow \emptyset$;        ▷ Dynamic and Temporary Constraint Arrays
5:     for $j = 1, \ldots, N$ do        ▷ $N$ is the number of the horizon points
6:         for $i = 1, \ldots, n_{obs}$ do        ▷ $n_{obs}$ is the number of obstacles
7:             if $r^i_{obs,\nu_j} < r_{thld}$ then
8:                 $\mathbb{C}_D \leftarrow \mathbf{s}_i^j$;        ▷ Store $i^{\text{th}}$ obstacle position which is indexed by $j$
9:                 $\mathbb{C}_T^k \leftarrow \mathbf{s}_i$;    ▷ Store $i^{\text{th}}$ obstacle position in the $k^{\text{th}}$ temp constraint
10:                $k = k + 1$;
11:                if $k$ is $n_T$ then
12:                    $k = 1$;
13:                end if
14:            end if
15:        end for
16:    end for
17:    $\mathbb{C}_L = (\mathbb{C}_D, \mathbb{C}_T^k)$;
18: return $\mathbb{C}_L$
```

5.2.4.3 Robust Path Guidance Algorithm

The initial state of the vehicle, the nature of the environment (e.g. branched narrow passages), and the terminal condition location may all affect the feasibility of the optimization problem solution. Figure 5.5 depicts two different path planning scenarios. In Figure 5.5(a), the obstacle is small and NMPH can easily find a feasible solution. In Figure 5.5(b), the obstacles almost block the way to the destination point. In this situation, the NMPH solver risks producing infeasible solutions by getting trapped in local minima.

As mentioned in Section 5.2.3, the graph-based path planning yields multiple vertices, which are used by the NMPH approach to generate multiple feasible paths, ranging from the nearest to the most distal (terminal) vertex. The small obstacle depicted in Figure 5.5(a) does not cause any issues for the NMPH in generating a feasible path directly to the terminal vertex. However, Figure 5.5(b) illustrates how the NMPH algorithm uses multiple consecutive paths (gray lines) generated to the intermediate vertices of the path generated by the graph-based planner (green line) to eventually find the resulting optimal path (blue line). Lines 12-21 in Algorithm 5.2 demonstrate the Path Guidance algorithm that adds robustness to the NMPH approach in finding a feasible solution. Note in case the Path Guidance algorithm is unable to help NMPH find a feasible path to the terminal vertex, the system can still use the path generated by the graph-based planner.

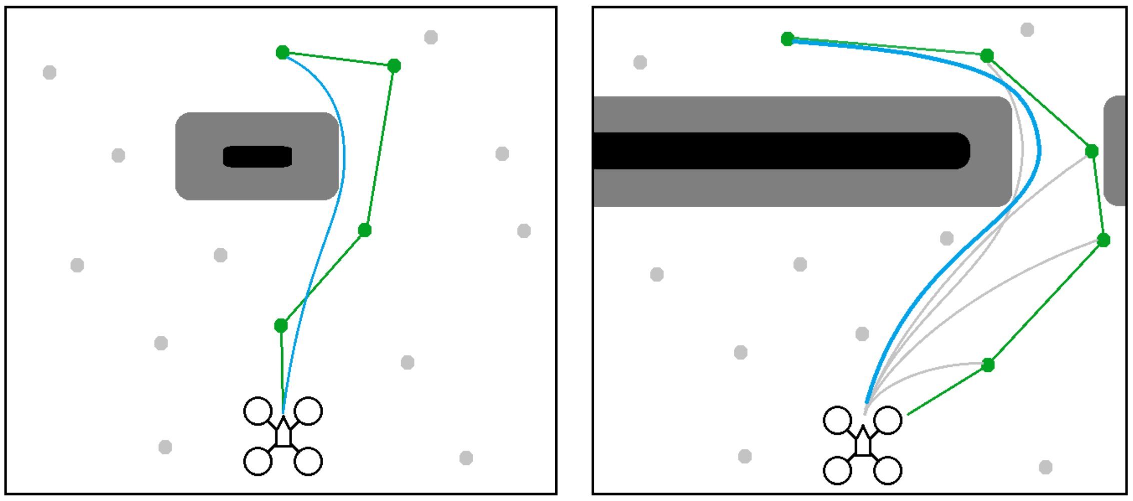

(a) (b)

Figure 5.5: Graph-based vs NMPH path planning. (a) The terminal vertex of the green path (from graph-based planner) is sufficient to generate the optimum blue path by NMPH. (b) All the vertices of the green path are used successively to guide the solutions of NMPH to the final optimal trajectory (blue path).

5.3 Experimental Results

In this section, simulation and a preliminary real-time hardware flight test are presented to evaluate and validate the proposed design on quadcopter and hexacopter vehicles while operating in GPS-denied environments.

The algorithms are implemented within the Robot Operating System (ROS) [95], a Linux-based system that handles communication between the individual subsystems and the vehicle. The ACADO Toolkit [83] is used for optimization. The optimization problem is programmed in a self-contained C++ environment within this toolkit, then a real-time nonlinear solver is generated to run the optimizations online. The resulting code can be compiled and run within ROS, which also handles the communication between the solver and the vehicle, either simulated or real [84]. The NMPH optimization problem (2.11) was written in C++ code using ACADO, then automatically converted into a highly efficient C code that is able to solve the optimization problem in real-time.

5.3.1 Simulation Results

In order to test the proposed approach before implementing it on a real drone, the quadcopter drone vehicle is simulated within AirSim [96]. All frameworks and the AirSim simulator are run in ROS on an Intel Core i7-10750H CPU @ 2.60-5.00 GHz equipped with the NVIDIA GeForce RTX 2080 Super Max-Q GPU. The prediction horizon for NMPH was set to $T = 8$ s using a sampling time of 0.2 s, which was found satisfactory for trajectory generation. The cost function weights W_x, W_ξ, and W_T were adjusted heuristically to ensure a balanced trajectory generation performance towards the terminal setpoint.

The drone's measured pose and pointcloud information are obtained from the AirSim simulator and sent to the motion planner. The global graph-based and local NMPH planners generate reference trajectories for the vehicle, which are forwarded to AirSim for trajectory tracking purposes. RViz, the 3D visualization tool for ROS, is used to monitor and visualize the simulation process.

Different simulation results are now presented to evaluate the performance of the proposed approach on a quadcopter drone navigating autonomously through a previously unexplored, GPS-denied underground environment available within the AirSim simulator. The motion planner design illustrated in Figure 5.2 is implemented for this purpose. Within AirSim, the virtual quadcopter is equipped with a customized 32-channel 360° scanning Lidar sensor with a 45° vertical field of view, 10 Hz rotation rate, and 50 meters range. The pointcloud data plus the vehicle pose are acquired and used to build a volumetric map of the environment and locate the vehicle within it.

As discussed in Section 5.2.3, the graph-based planning algorithm guides the vehicle towards unexplored areas within the map and provides vertices as terminal setpoints x_{ss} for the NMPH local path planner. The design's robustness is increased by implementing the Obstacle Avoidance and Path Guidance algorithms proposed in Sections 5.2.4.2 and 5.2.4.3, respectively. Finally, the generated reference path from the motion planner is sent to the AirSim quadcopter for tracking. The NMPH continues updating the path during the vehicle's movement toward a setpoint. This allows to avoid dynamic obstacles and improves the tracking performance. Once the vehicle reaches a setpoint, the planning process is repeated until the environment is fully explored or the mission is interrupted by the operator.

Figure 5.6 shows the paths generated by the graph-based and the NMPH path planners. The NMPH is seen to provide a smooth and optimal path as compared to the sharp-corner path generated by the graph-based planner. Moreover, the NMPH keeps updating the path dynamically from the start to the terminal point at a rate of up to 100 Hz, while the graph-based planner generates only one path between the two points. To reduce computational load, the NMPH algorithm rate is set to 5 Hz, which was found to be suitable in generating continuous and smooth paths in the environment. This rate of path regeneration also provides good path following performance in the presence of static and dynamic obstacles.

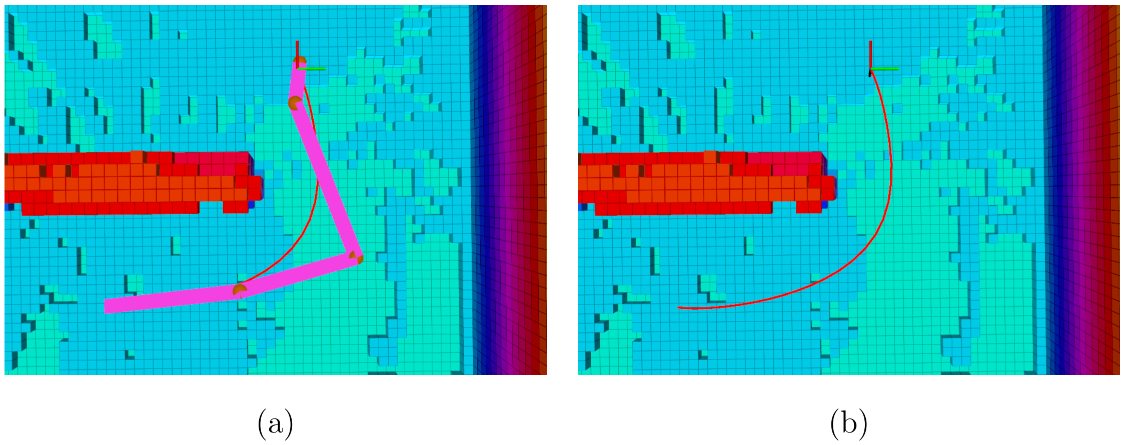


Figure 5.6: Motion Planner. (a) Path planning using graph-based approach (pink) and NMPH algorithm (red). (b) Optimal path using NMPH algorithm.

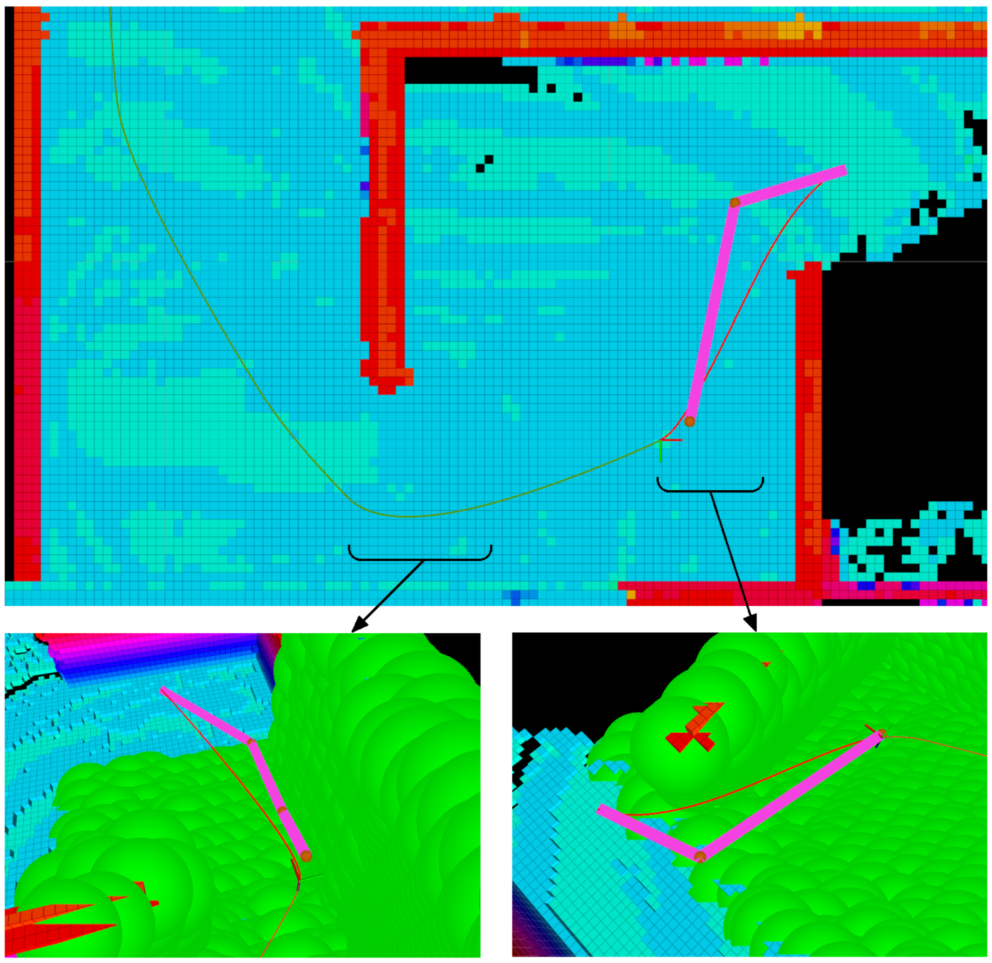


Figure 5.7: Illustration of trajectory generation and tracking. The green path is the trajectory of the vehicle, and the red path is the future reference path.

A portion of the overall tracked trajectory between multiple vertices using the NMPH algorithm can be seen in Figure 5.7. Respecting the system dynamics provides smooth flight paths and thus reduces power consumption caused by abrupt changes in the trajectory.

The DLOM generates a continuously changing obstacle map modeled by virtual spheres as depicted in Figure 5.8. As discussed in Sections 5.2.4.1 and 5.2.4.2, mapped obstacles are represented by (continuously updated) inequality constraints within the optimization problem. The Obstacle Avoidance Algorithm helps in creating and updating a path that avoids the obstacles as shown in Figure 5.8.

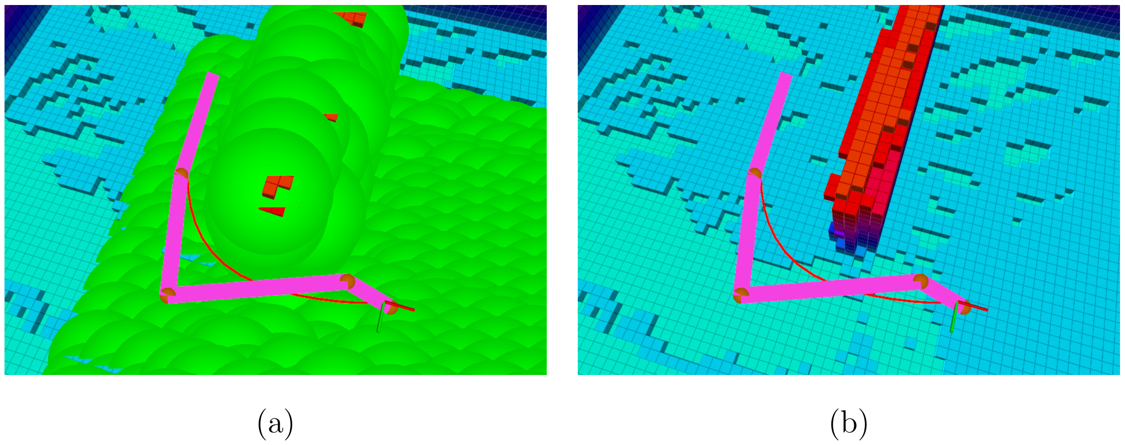
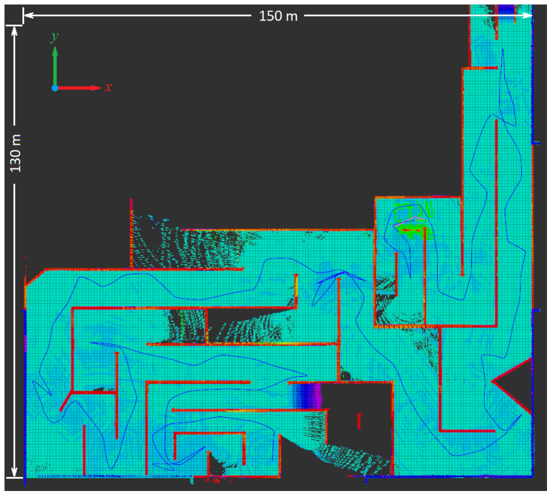


Figure 5.8: Dynamic Local Obstacle Mapping and Avoidance. In (a) the DLOM is made visible while in (b) it is hidden.

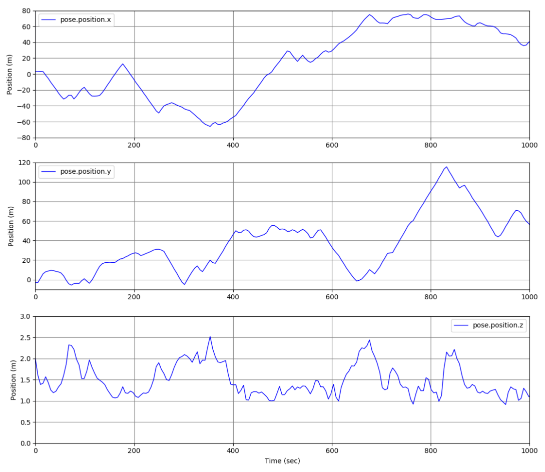
In the next simulation test, the quadcopter autonomously navigates an unexplored GPS-denied environment. Figure 5.9 shows the exploration mission performed by the quadcopter. The drone travels a total distance of 774.5 m while following smooth trajectories generated by our proposed algorithm. Meanwhile, the graph-based planner generated paths with a total length of 993.1 m for the same exploration mission. Table 5.1 and Figure 5.10 offer a mission performance comparison between using the graph-based planner solo versus using the graph-based planner integrated with our NMPH approach in terms of exploration time, average computation time of the generated paths, path lengths between terminal vertices, and average and total length of the generated paths. This comparison shows the impact of using the NMPH-FBL algorithm for reducing power consumption, total mission time, and unwanted abrupt motions while following the generated reference paths.

Table 5.1: Comparison between Graph-based and Graph-based-plus-NMPH-FBL approaches to path planning.

	Total length of the generated paths	Average path length (between setpoints)	Average path computation time	Exploration time	Continuous path generation
Graph-based	993.1 m	8.79 m	733 ms	1327 s	No
Graph-based & NMPH-FBL	**774.5 m**	**6.86 m**	**4.34 ms**	**1035 s**	**Yes**

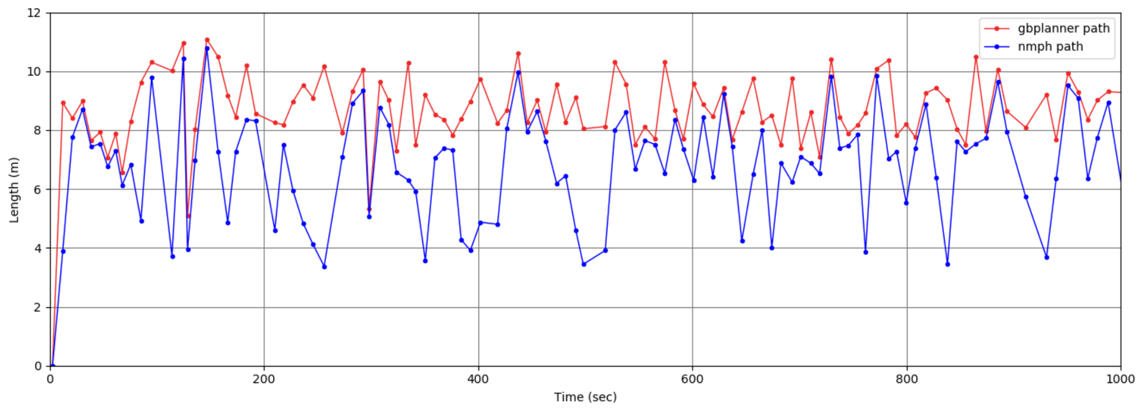


(a)

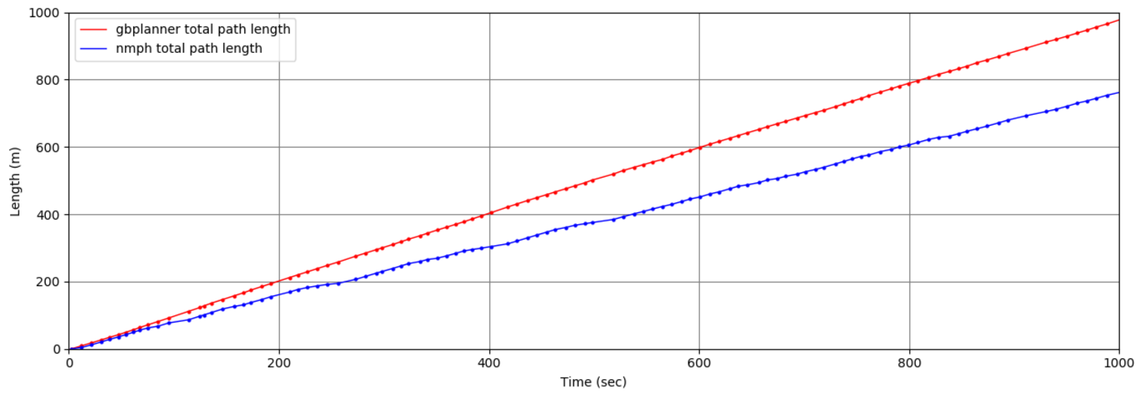


(b)

Figure 5.9: Autonomous navigation and exploration in GPS-denied environment. The vehicle travelled a total distance of 774.5 m in about 1035 s.



(a)



(b)

Figure 5.10: Comparison of path lengths between graph-based planner and our proposed NMPH-FBL path planner. (a) Path length between stabilization points. (b) Total length of generated paths.

For a second exploration mission, another comparison between the global motion planning with NMPH-based approaches (NMPH-FBL and NMPH-BSC) and the graph-based planner is offered in Table 5.2. It can be concluded that the NMPH-based approaches provide better exploration performances compared to the conventional graph-based planner. Moreover, the NMPH-BSC approach outperforms its predecessor (NMPH-FBL), which supports our claims from Chapter 4 that NMPH-BSC can provides more efficient local trajectories than NMPH-FBL.

Table 5.2: Comparison between Graph-based and Graph-based-plus-NMPH approaches to motion planning.

	Total length of the generated paths	Average path length (between setpoints)	Exploration time	Continuous path generation
Graph-based	1252m	8.03m	1322s (calculated)	No
Graph-based & NMPH-FBL	977m	6.25m	1032s (calculated)	**Yes**
Graph-based & NMPH-BSC	**949**m	**6.08**m	**1002**s	**Yes**

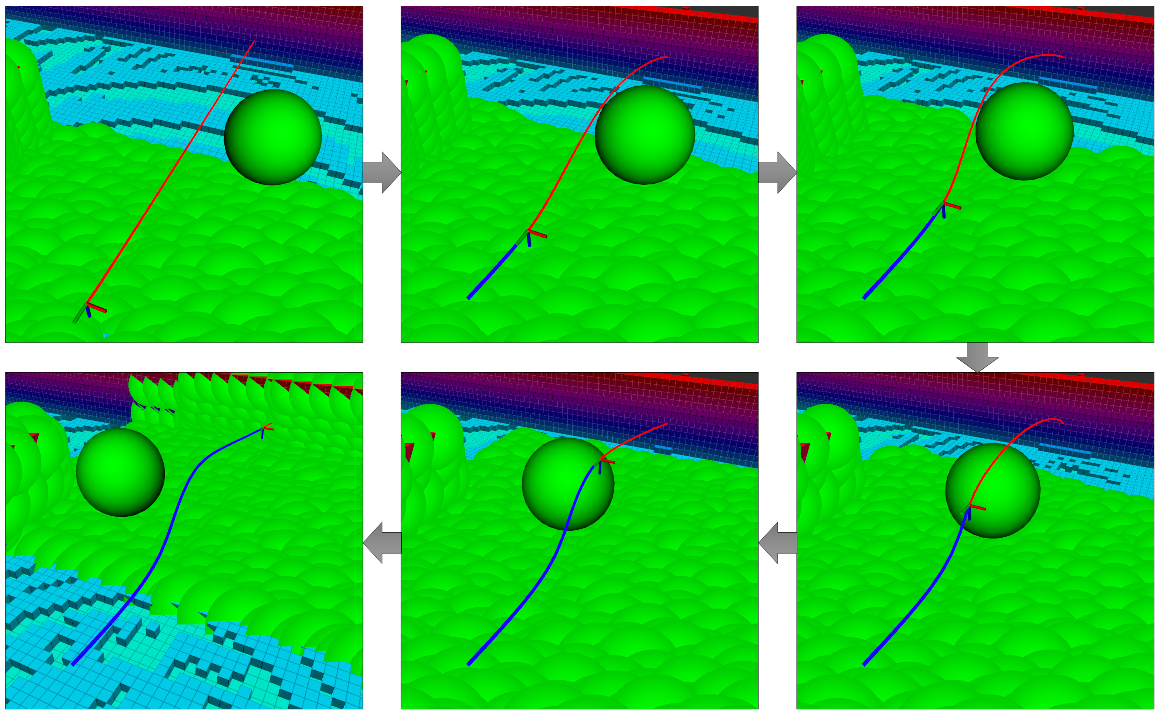


Figure 5.11: Obstacle avoidance for a moving object. The object (sphere) is moving to the left. The NMPH regenerates the red path continuously to avoid the object. The blue curve represents the flight trajectory of the drone.

In the final simulation test, obstacle avoidance for a moving object is tested while the drone navigates through the environment. This is shown in Figure 5.11, where the continuous regeneration of the path by the NMPH algorithm enables the drone to safely navigate to the stabilization point.

5.3.2 Real-time flight test results

For real-time hardware testing, a DJI FlameWheel F550 hexacopter built and instrumented to explore unknown environments using our proposed NMPH approach in conjunction with a global motion planner. The vehicle is equipped with a Pixhawk flight control board running the ArduPilot autopilot software [113], and an onboard NVIDIA Jetson NX computer running ROS. The communication between ROS and ArduPilot is established through MAVLink. A Velodyne Puck LITE LiDAR is mounted on the hexacopter to provide pointcloud data required for 3D mapping and state estimation. The detailed hardware setup and software architecture are described in detail in Chapter 7.

The flight test evaluates the exploration performance of NMPH running onboard a real drone. Also, local trajectory tracking and the functionality of the global motion planner are tested in this experiment in an underground parking area, as shown in Figure 5.12.

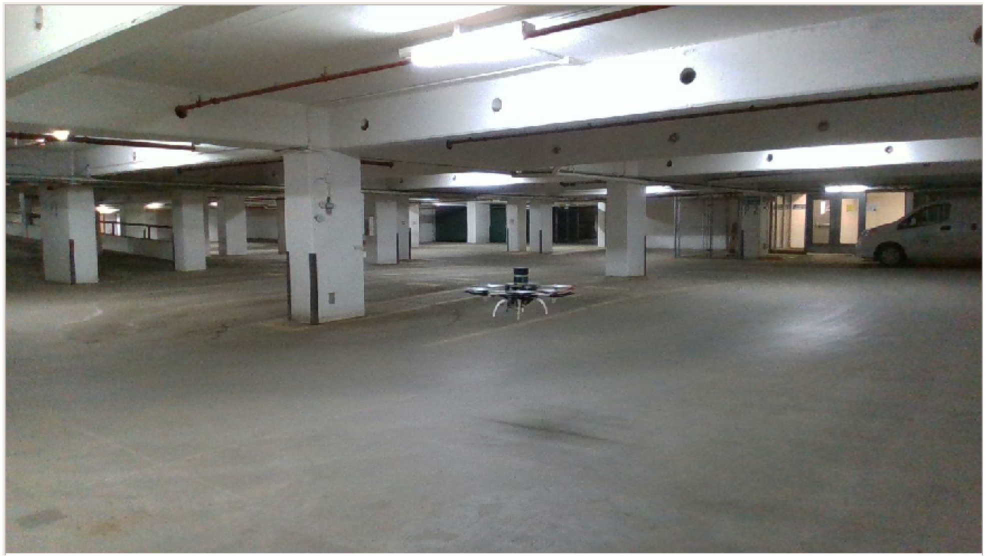


Figure 5.12: The hardware hexacopter vehicle navigating an underground parking area using the presented global motion planner approach.

The navigation capabilities of the system were tested as the motion planner generated several terminal setpoints, and the NMPH algorithm provided continuously regenerating local reference trajectories between the successive setpoints to ensure a smooth flight.

Figure 5.13 shows the start of the exploration mission, and Figure 5.14 depicts the end of the operation as the drone returned to the home position. In Figure 5.13, it can be seen that the NMPH planning algorithm generated a continuous trajectory to the first setpoint calculated by the graph-based planner. Also, it can be noted that the estimated predicted trajectory $\hat{\xi}_{ref}$ perfectly matches the predicted out trajectory $\tilde{\xi}$, and fulfilling the objective of the stage cost function presented in (2.11) in minimizing the deviation between the estimated and the predicted reference trajectories, and ensuring their convergence to each other. As highlighted in Section 2.2, the estimated reference trajectories or the predicted reference trajectory can be used as the desired trajectories for the actual closed-loop system.

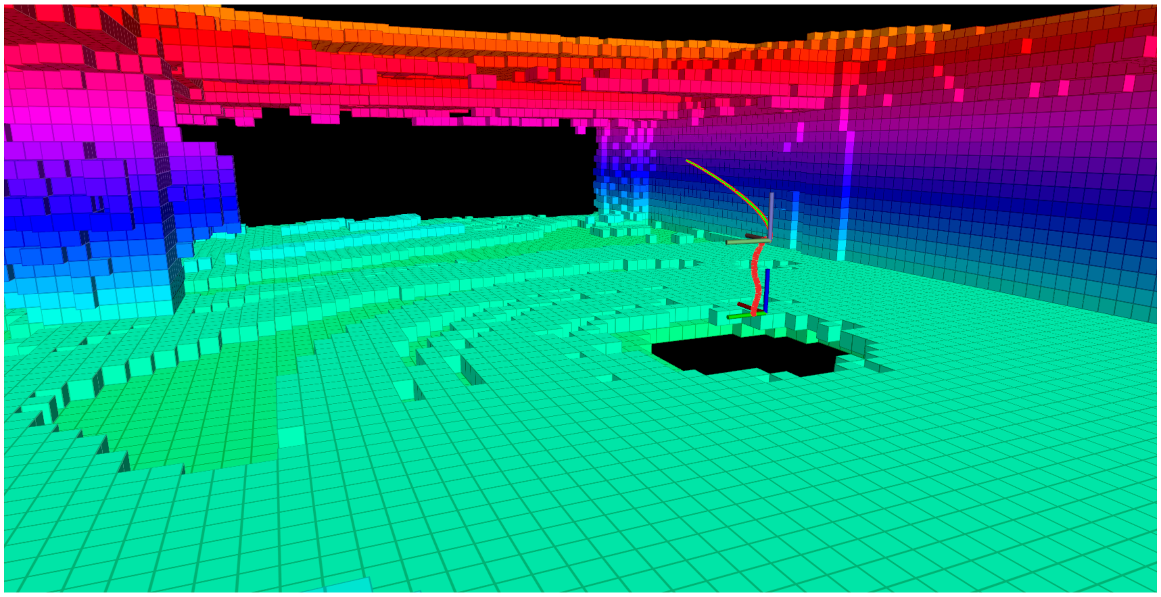


Figure 5.13: 3-D volumetric map of the unknown space at the beginning of the flight test. The estimated predicted trajectory $\hat{\xi}_{ref}$ to a setpoint can be seen in this figure (green curve). It overlaps another trajectory (red curve) which represents the predicted out trajectory $\tilde{\xi}$.

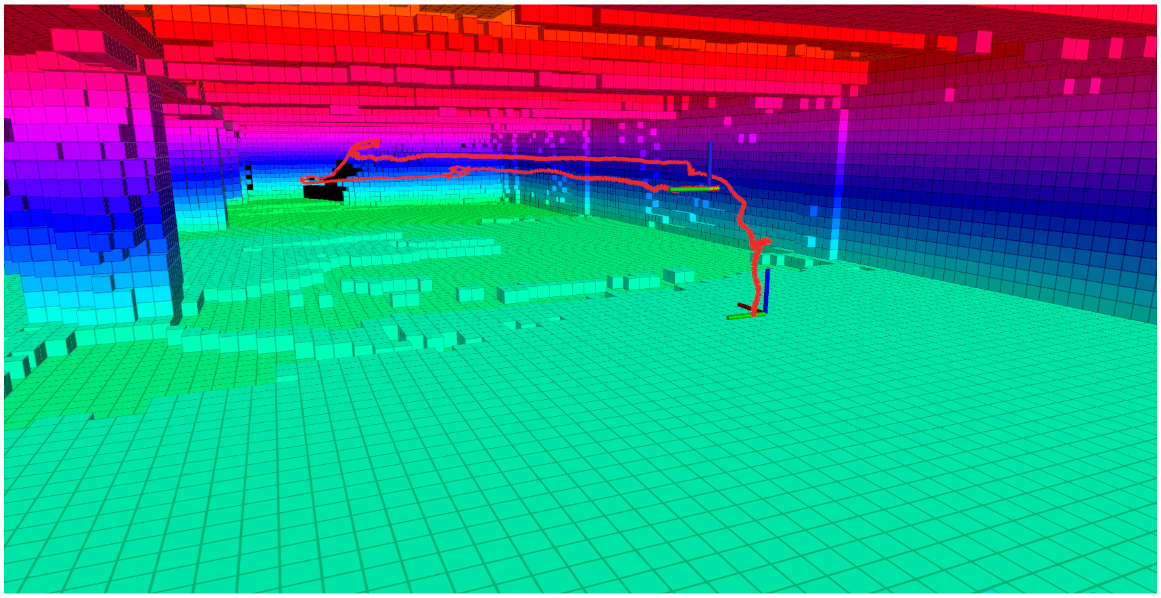


Figure 5.14: 3-D volumetric map of the explored underground space at the end of the flight test. The volumetric map is constructed from free and occupied voxels and built nicely to represent the space for a safe exploration mission.

Furthermore, Figure 5.15 shows the top and 3D perspective views of the tracked paths by the hexacopter drone after completing the exploration mission. The space covered by the drone is approximately $18 \times 9 \times 3$ m and total flight time is 3 min. The drone was asked to return home after generating 8 setpoints by the global motion planner. All in all, the navigation capabilities and our system reliability (equipped with the proposed approaches) were tested and validated in an unknown, unstructured, and GPS-denied environment and proved that system can achieve autonomous navigation and exploration in confined places.

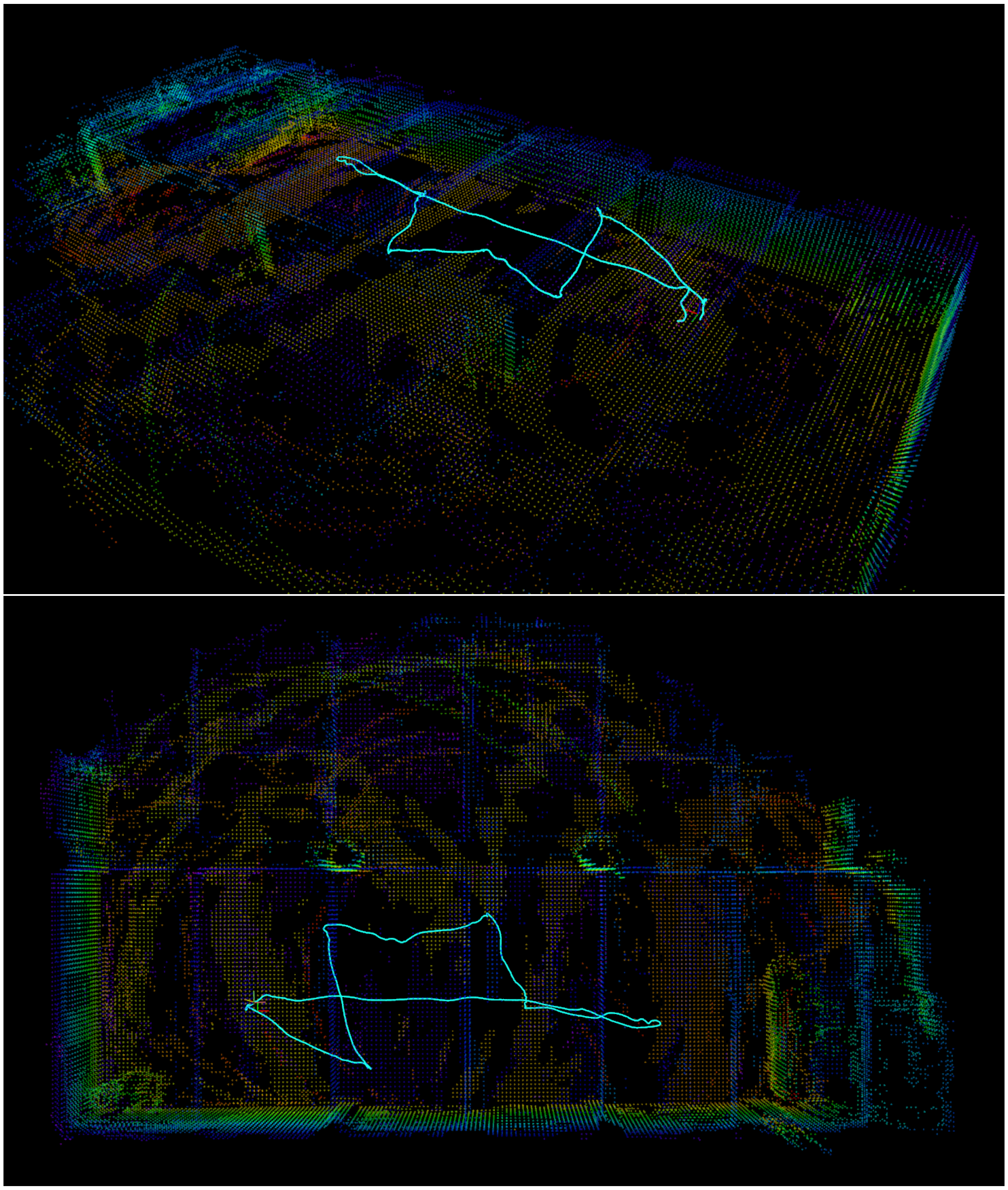


Figure 5.15: Tracked trajectories by the drone in the underground parking space using graph-based planner to generate terminal setpoints and NMPH-BSC to generate local trajectories between setpoints.

A final hardware flight test was performed outdoors in order to assess the trajectory planning performance in the presence of wind, in this case approximately 15 km/h. As before the graph-based motion planner provided multiple terminal setpoints, while our NMPH-BSC planned smooth trajectories between them in real-time. The resulting flight trajectory can be seen in Figure 5.16. Despite the presence of a wind disturbance, the drone was able to smoothly navigate between generated setpoints as in the earlier tests.

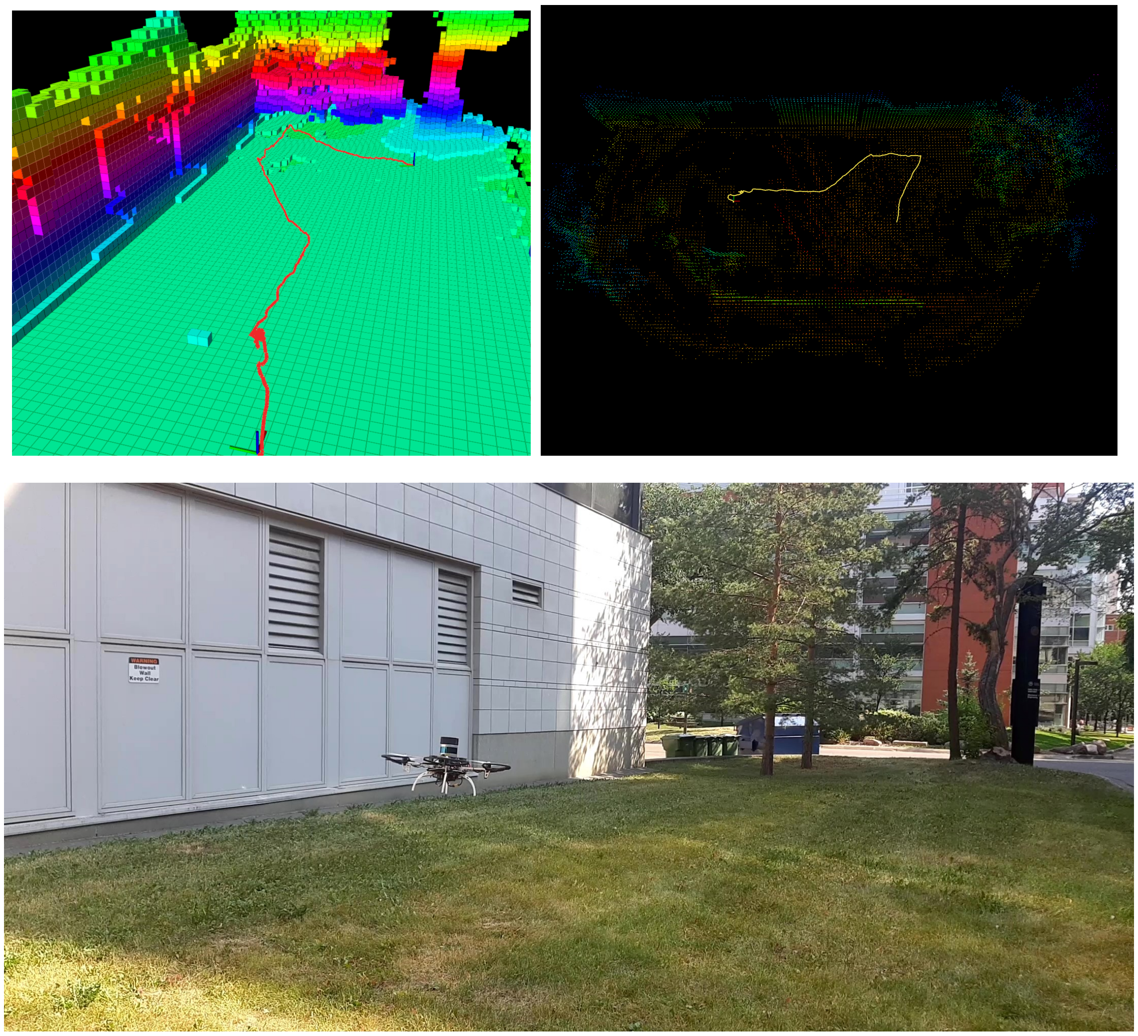


Figure 5.16: Flight test employing the NMPH-BSC algorithm in an outdoor environment with a 15 km/h wind speed.

5.4 Conclusions

This chapter presented a methodological motion planning approach for drone exploration in GPS-denied environments, which integrates our proposed NMPH trajectory planning approach with a graph-based planner. The NMPH formulation employs the nonlinear system dynamics model with feedback linearization or backstepping control inside an online optimization-based process to generate feasible, optimal and smooth reference trajectories for the vehicle. The performance of the overall motion planner is increased by introducing methods for robust path generation and dynamic obstacle mapping and avoidance.

The developed motion planner was evaluated through a series of simulation flights as well as a real-time hardware flight test to validate the performance of the proposed design on quadcopter and hexacopter drones navigating the environment. The results show the ability our algorithm to improve motion planning performance over conventional techniques and generate smooth and safe flight trajectories in a computationally efficient way.

Chapter 6

Adaptive Nonlinear Model Predictive Horizon using Deep Reinforcement Learning

6.1 Introduction

The optimization problem of NMPH (2.2) contains different parameters that penalize the deviations within its formulation. In this work a new formulation is proposed to online tune the NMPH parameters and achieve the best possible planning performance while the drone vehicle is running (instead of selecting them heuristically with fixed values). Hence, this chapter presents an adaptive trajectory planning approach for nonlinear dynamical systems using NMPH with Deep Reinforcement Learning (DRL) adaptation scheme. The proposed adaptive design of this work is called 'adaptive NMPH', which aims to generate the most efficient optimal trajectories that an autonomous vehicle can follow based on the system and environment states. This is done by online tuning the NMPH optimization problem parameters using two different Actor-Critic DRL algorithms, which are Deep Deterministic Policy Gradient (DDPG) and Soft Actor-Critic (SAC), one at a time.

The research contributions of this chapter are presented below:

- Introducing an adaptive NMPH framework which uses a DRL-based method to tune the parameters of the underlying optimization problem of generating the best possible reference trajectories for the vehicle.

- Designing the RL components (the agent, the environment, and the reward scheme) of the proposed system.
- Implementing two different Actor-Critic DRL algorithms — the deterministic DDPG approach and the probabilistic SAC algorithm — within the adaptive NMPH framework, comparing them in terms of learning speed and stability.
- Evaluating the performance of the overall system with each of the above DRL algorithms in a life-like simulation environment.

The remainder of this chapter is organized as follows: Section 6.2 describes the various methodologies used in the adaptive schemes. Section 6.3 presents the adaptive NMPH framework for trajectory planning. Section 6.4 evaluates the proposed designs in simulation, and Section 6.5 concludes the work presented in this chapter.

6.2 Deep Reinforcement Learning Overview

This Section covers the preliminaries of reinforcement learning, then describes the DDPG and SAC algorithms used within the adaptive NMPH frameworks.

6.2.1 Reinforcement Learning Preliminaries

A Reinforcement Learning (RL) system is composed of an agent that interacts with an environment in a sampling-based manner. Assuming the environment is fully observed, at each time sample the agent observes the environment state $s \in \mathcal{S}$, applies the action $a \in \mathcal{A}$ decided by a policy, and receives a scalar reward $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, where $\mathcal{S}$ and $\mathcal{A}$ are the environment state space and the action space, respectively. In our work, we consider continuous action spaces with a real-valued vector $a \in \mathbb{R}^n$. The main components of an RL framework are depicted in Figure 6.1.

It is important to differentiate between the system state x (presented in Chapter 2, Section 2.2) and the state s used in RL. The former means the state vector of a dynamic system while the latter represents the state of the environment within the

RL context. Hence, to avoid confusion, the word 'state' used in the following sections means 'environment state' as used in the RL literature, unless stated otherwise.

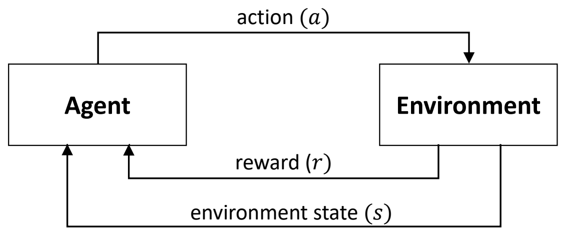


Figure 6.1: Block diagram of an RL framework

The agent's policy can be deterministic (denoted by $\mu(s)$), or stochastic (denoted by $\pi(\cdot|s)$). In deep RL, we parameterize the policy and represent it using a universal function approximator realized by a neural network. The parameters (representing the weights and biases of the policy's neural network) are denoted by θ and the corresponding policies for the deterministic and stochastic cases are denoted by $\mu_\theta(s)$ and $\pi_\theta(\cdot|s)$, respectively.

We consider a stochastic environment with transition probability function $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$, where $p(s', r|s, a)$ is the probability of transition from the current state s and action a to the next state s' with reward $r \in \mathcal{R}$. Also, we define the 'return' as the expected weighted sum of future rewards $R = \sum_{t=0}^{\infty} \gamma^t r(s, a)$, where $r(s, a)$ is the reward function and $0 \leq \gamma \leq 1$ is the discounting factor. The main objective in RL is to find a policy that maximizes the expected sum of rewards $J = \underset{\tau\sim\pi}{\mathbb{E}}\big[R\big]$, where $\tau = (s_0, a_0, s_1, a_1, ...)$ is the trajectory sequence of states and actions in the RL system.

The state-action value function (a.k.a. Q-function) specifies the expected return of an agent after performing an action a at a state s by following a policy π or μ. The Q-function can be described by a Bellman equation [66], which for a stochastic policy can be defined as

$$Q^\pi(s, a) = \underset{s'\sim p}{\mathbb{E}}\Big[r(s, a) + \gamma \underset{a'\sim\pi}{\mathbb{E}}\Big[Q^\pi(s', a')\Big]\Big] \tag{6.1}$$

where $s' \sim p$ represents the stochastic transition to the next state s', and $a' \sim \pi$ is the action taken at the state s' based on a stochastic policy π.

For a deterministic policy, the Q-function can be written as

$$Q^{\mu}(s,a) = \mathop{\mathbb{E}}_{s' \sim p}\Big[r(s,a) + \gamma Q^{\mu}\big(s', \mu(s)\big)\Big]. \tag{6.2}$$

The process of computing the value functions for a policy using a Bellman equation is called *policy evaluation*, while using the computed value functions to find better policies is known as *policy improvement* [66].

Many recent advances in deep reinforcement learning consider a replay buffer (a.k.a. experience buffer or experience replay) during the learning process. The replay buffer is a memory that collects the previous experience tuples $(s, a, r, s') \in \mathcal{B}$, in which the agent uses them to increase the computation efficiency and speed up learning [114].

We will now review the DDPG and SAC deep reinforcement learning algorithms used within our proposed adaptive NMPH frameworks.

6.2.2 Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) [72] is a model-free deep reinforcement learning technique that is designed for applications with continuous and deterministic action spaces. It uses stored experiences in a replay buffer to concurrently learn a Q-function and a policy. DDPG is classified as an Actor-Critic technique, where the Actor is a policy network that receives the state of the environment and provides continuous action to the system, while the Critic is a Q-function network that inputs a state and action pair and outputs a Q-value.

Similarly to Q-learning, DDPG seeks to find the optimal action-value function $Q^*(s,a)$ followed by the optimal action $a^*(s)$, where $a^*(s) = \arg\max_a Q^*(s,a)$. As a deep reinforcement learning approach, DDPG uses universal function approximators represented by neural networks to learn $Q^*(s,a)$ and $a^*(s)$. Consider a neural network approximator $Q_\phi(s,a)$ (a.k.a. Q-network) with parameters ϕ, where the objective is to make the approximator as close as possible to the optimal action-value function written in the form of a Bellman equation. The associated Mean Square Bellman Error (MSBE [115]) function is defined as

$$J_Q(\phi, \mathcal{B}) = \mathop{\mathbb{E}}_{(s,a,r,s')\sim\mathcal{B}}\left[\Big(Q_\phi(s,a) - \big(r + \gamma \max_{a'} Q_\phi(s',a')\big)\Big)^2\right] \tag{6.3}$$

where a random batch of data (s, a, r, s') from the replay buffer $\mathcal{B}$ is used for each update. The goal is to minimize the loss in (6.3) by performing a gradient descent of the MSBE $J_Q(\phi, \mathcal{B})$.

As shown in (6.3), the neural network parameters represented by ϕ are used for both the action-value function approximator $Q_\phi(s,a)$ *and* the network that estimates $Q_\phi(s',a')$, which uses the *next* states and actions. Unfortunately, this makes it impossible for the gradient descent to converge. To tackle this issue, a time delay is added to the network parameters ϕ for $Q_\phi(s',a')$. The adjusted network is called the target Q-network $Q_{\phi_{\text{targ}}}(s',a')$ with parameters ϕ_{targ}. A copy of the Q-network $Q_\phi(s',a')$ is used for the target Q-network $Q_{\phi_{\text{targ}}}(s',a')$, where the latter uses the weighted average of the model parameters $\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1-\rho)\phi$ to stabilize Q-function learning [116]. It should be noted that the parameters of the target Q-network are not trained. However, they are periodically synchronized with the original Q-network's parameters.

The MSBE function given in (6.3) contains a maximization term for the Q-value. One way to perform this maximization is to apply the optimal action $a^*(s)$. This can be achieved by creating another approximator for the policy $\mu_\theta(s)$ with parameters θ and maximizing the associated Q-function w.r.t. the replay buffer $\mathcal{B}$. This new policy also requires a time delay to stabilize its learning. Therefore, a target policy $\mu_{\theta_{\text{targ}}}(s)$ is introduced to maximize $Q_{\phi_{\text{targ}}}$. The Bellman equation, MSBE, and policy learning function are respectively given by

$$
\begin{aligned}
y(r,s') &= r+\gamma\,\overbrace{Q_{\phi_{\text{targ}}}(s',\underbrace{\mu_{\theta_{\text{targ}}}(s')}_{\text{target policy network}})}^{\text{target Q network}} && (6.4)\\
J_Q(\phi,\mathcal{B}) &= \mathop{\mathbb{E}}_{(s,a,r,s')\sim\mathcal{B}}\Big[\big(\underbrace{Q_\phi(s,a)}_{\text{Q network}}-y(r,s')\big)^2\Big] && (6.5)\\
J_\mu(\theta,\mathcal{B}) &= \mathop{\mathbb{E}}_{s\sim\mathcal{B}}\Big[Q_\phi\big(s,\mu_\theta(s)\big)\Big] && (6.6)
\end{aligned}
$$

Practically, for a random sample $B=\{(s,a,r,s')\}$ from the replay buffer $\mathcal{B}$ with cardinality $|B|$, equations (6.5) and (6.6) can be expressed as

$$
\begin{aligned}
J_Q(\phi,B) &= \frac{1}{|B|}\sum_{(s,a,r,s')\in B}\left(Q_\phi(s,a)-y(r,s')\right)^2 && (6.7)\\
J_\mu(\theta,B) &= \frac{1}{|B|}\sum_{s\in B}Q_\phi\big(s,\mu_\theta(s)\big) && (6.8)
\end{aligned}
$$

During training, Ornstein–Uhlenbeck noise is added to the action vector to enhance the exploration of the DDPG policy [115]. The pseudo-code summarizing the DDPG process is given in Algorithm 6.1.

Algorithm 6.1 Deep Deterministic Policy Gradient

```
1: Initialize: $\theta$, $\phi$, $\mathcal{B} \leftarrow \emptyset$
2: Set $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
3: **repeat**
4:     Observe the state $s$
5:     Find and apply noise to the action $a = \mu_\theta(s) + \eta_{\text{OU-noise}}$
6:     Apply $a$ by the agent
7:     Observe the next state $s'$ and calculate the reward $r$
8:     Store $(s,a,r,s')$ in the replay buffer $\mathcal{B}$
9:     **for** a given number of episodes **do**
10:        Obtain a random sample $B = \{(s,a,r,s')\}$ from $\mathcal{B}$
11:        Compute Bellman function $y(r,s')$
12:        Update the Q-function by applying gradient descent to MSBE: $\nabla_\phi J_Q(\phi,B)$
13:        Update the policy by applying gradient ascent to (6.8): $\nabla_\theta J_\mu(\theta,B)$
14:        Update target networks' parameters: $\begin{cases}\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1-\rho)\phi \\ \theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1-\rho)\theta\end{cases}$
15: **until** convergence
```

The hyperparameters used for the DDPG algorithm are the number of training episodes, target update factor (ρ), actor and critic networks learning rates, replay buffer size, random batch size, and discount factor value. The sensitivity to the hyperparameter values and the interaction between the Q-value and policy approximator $\mu_\theta(s)$ make analyzing the stability and convergence of DDPG difficult tasks [117], especially when using high-dimensional nonlinear universal function approximators [118]. Moreover, DDPG is expensive in terms of its sample complexity, which is measured by the number of training samples needed to complete the learning process. The Twin Delayed Deep Deterministic algorithm (a.k.a. TD3) [73] improves the overall efficiency of the DDPG process in terms of learning time and computational resources, and it is less sensitive to changes in the hyperparameters [119], however a careful choice of the hyperparameter values is still required for a stable learning process.

An alternative approach which overcomes the issues of the DDPG algorithm is Soft Actor-Critic (SAC) [74, 118], a probabilistic DRL algorithm, considered next.

6.2.3 Soft Actor-Critic

Soft Actor-Critic (SAC) is a model-free deep reinforcement learning technique that obtains a stochastic policy by maximizing its expected return and entropy [74]. Maximizing the expected entropy in the policy leads to broader exploration in complicated domains, which enhances the sampling efficiency, increases robustness, and guards against convergence to a local maximum [115]. SAC is a probabilistic framework that builds on Soft Q-learning within an Actor-Critic formulation for either continuous or discrete action spaces. In the present work, we will focus exclusively on continuous actions.

SAC involves simultaneously learning two Q-functions Q_{ϕ_1}, Q_{ϕ_2} using two different Q-networks, as well as a stochastic policy π_θ using a policy network. Both Q-functions use a modified MSBE (we call it here soft-MSBE) to be presented in (6.10), where the minimum Q-value of both functions is used to update the policy (known as clipped double Q-learning [73]). SAC employs a 'target network' associated with each Q-network to enhance the stability of the learning process, where both target Q-networks are copies of the corresponding Q-network, but employ weighted averaging on the network parameters during training. Because of the policy's stochastic nature, SAC uses the current policy to obtain the next state-action values without needing to have a target policy [115]. Also, the stochastic nature of the exploration process means it's not necessary to artificially introduce noise, as was done in the deterministic DDPG.

The objective of SAC is to maximize the sum of the expected return and entropy. The Bellman equation within its Q-value function thus includes the expected entropy of the policy as follows:

$$Q_\pi(s,a) \approx r + \gamma \left(Q_\pi(s'_\mathcal{B}, a'_\pi) - \alpha \log \pi(a'_\pi | s'_\mathcal{B})\right) \tag{6.9}$$

where α is the coefficient which regulates the trade-off between the expected entropy and return, $s'_\mathcal{B}$ indicates that the replay buffer is used to obtain the expectation of the future states, and $a'_\pi \sim \pi(\cdot|s')$ indicates that the current policy is used to obtain future actions. For simplicity of notation, we will denote $s'_\mathcal{B}$ by s' and a'_π by a' in the sequel.

For *policy evaluation* (Q-function learning), two Bellman residuals are used within SAC [74], referred to as soft-MSBEs. In addition to the policy network π_θ, each soft-MSBE includes a Q-network and two target Q-networks in its calculation as

$$J_Q(\phi_i, \mathcal{B}) = \underset{(s,a,r,s',a')\sim\mathcal{B}}{\mathbb{E}} \left[\Big(Q_{\phi_i}(s,a) - y(r,s',a')\Big)^2\right], \qquad i = 1,2 \tag{6.10}$$

and their Bellman equation forms are

$$y(r,s',a') = r + \gamma \left(\min_{j=1,2} Q_{\phi_{\text{targ},j}}(s',a') - \alpha \log \pi_\theta(a'|s')\right), \quad a' \sim \pi_\theta(\cdot|s') \tag{6.11}$$

Similarly to DDPG, the Q-functions are updated using gradient descent, while gradient ascent is utilized to update the policy network.

For *policy improvement* (policy learning), the policy should maximize the state-value function $V_\pi(s)$ defined as

$$V_\pi(s) = \mathop{\mathbb{E}}_{a\sim\pi}\Big[Q_\pi(s,a) - \alpha \log \pi(a|s)\Big] \tag{6.12}$$

which represents the expected return when starting from a state s and following a policy π.

For the optimal value of the action, we can employ reparameterization [74, 115] to obtain a continuous action from a deterministic function that represents the policy. The function is expressed by the state and additive Gaussian noise as

$$a_\theta(s,\xi) = \tanh\left(\mu_\theta(s) + \sigma_\theta(s)\,\xi\right), \qquad \xi \sim \mathcal{N}\big(0, \mathrm{diag}(1,\ldots,1)\big). \tag{6.13}$$

The policy optimization can be performed by maximizing the Q-function, which implicitly maximizes the entropy of the trajectory. Using the computed value of the action from (6.13), the function to be maximized is

$$J_\pi(\theta,\mathcal{B}) = \mathop{\mathbb{E}}_{s\sim\mathcal{B},\,\xi\sim\mathcal{N}}\left[\min_{j=1,2} Q_{\phi_j}(s, a_\theta(s,\xi)) - \alpha \log \pi_\theta(a_\theta(s,\xi)|s)\right] \tag{6.14}$$

and the optimum policy can be obtained by finding $\arg\max_\theta\ J_\pi(\theta,\mathcal{B})$ using gradient ascent. For a random sample $B = \{(s,a,r,s',a')\}$ from the buffer $\mathcal{B}$, equations (6.10) and (6.14) can be expressed as

$$J_Q(\phi_i, B) = \frac{1}{|B|}\sum_{(s,a,r,s')\in B}\left(Q_{\phi_i}(s,a) - y(r,s',a')\right)^2, \qquad i = 1,2 \tag{6.15}$$

$$J_\mu(\theta, B) = \frac{1}{|B|}\sum_{s\in B}\Big(\min_{j=1,2} Q_{\phi_j}\big(s, a_\theta(s,\xi)\big) - \alpha \log \pi_\theta\big(a_\theta(s,\xi)|s\big)\Big) \tag{6.16}$$

Using a fixed value for the coefficient α in (6.16) makes the learning process sensitive. The authors of the original SAC method [74] provided an updated version of this algorithm in [118] that optimizes this parameter to predict the optimal actions over the span of the RL space. This is done by introducing an optimization problem which constrains the average entropy of the policy while allowing the entropy to vary based on the states of the system. The parameter α can be tuned by minimizing an objective function $J(\alpha)$ that includes the policy's log probability and the expected entropy constraint. More details about the optimization problem for α can be found in [118].

The pseudo-code for the SAC algorithm is provided in Algorithm 6.2.

Algorithm 6.2 Soft Actor-Critic

```
1: Initialize: $\theta$, $\phi_i$, $\alpha$, $\mathcal{B} \leftarrow \emptyset$,    $i = 1, 2$
2: Set $\phi_{\text{targ},i} \leftarrow \phi_i$
3: repeat
4:     Observe the state $s$
5:     Find the action $a \sim \pi_\theta(\cdot|s)$, and apply it through the agent
6:     Observe the next state $s'$ and the reward $r$
7:     Find the next action $a' \sim \pi_\theta(\cdot|s')$
8:     Store $(s, a, r, s', a')$ in the replay buffer $\mathcal{B}$
9:     for a given number of episodes do
10:        Obtain a random sample $B = \{(s, a, r, s', a')\}$ from $\mathcal{B}$
11:        Compute $y(r, s', a')$ in (6.11) and find the soft-MSBEs (6.10)
12:        Apply gradient descent on the soft-MSBEs: $\nabla_{\phi_i} J_Q(\phi_i, B)$
13:        Reparametrize the action: $a_\theta(s, \xi) = \tanh\left(\mu_\theta(s) + \sigma_\theta(s)\,\xi\right)$
14:        Apply gradient ascent on the policy: $\nabla_\theta J_\mu(\theta, B)$
15:        Apply gradient descent to tune $\alpha$: $\nabla_\alpha J(\alpha)$
16:        Update target networks: $\phi_{\text{targ},i} \leftarrow \rho\phi_{\text{targ},i} + (1 - \rho)\phi_i$
17: until convergence
```

6.3 Adaptive Trajectory Planning Framework

In this Section, we present the DRL-based adaptive framework used to adjust the gains of the NMPH trajectory planning algorithm. First, we will describe the agent and environment involved in the DRL problem, then present two adaptive NMPH architectures based on the DDPG and the SAC algorithm, respectively.

6.3.1 Agent and Environment Representations

Figure 6.2 shows the main components of the adaptive NMPH system. The environment is an autonomous drone which flies within an incrementally built-up 3D volumetric map of the surroundings. The drone uses the NMPH algorithm for planning local trajectories between the current pose and a terminal setpoint provided by the exploration algorithm presented in [79]. As covered in Chapter 2 (Section 2.2), the NMPH optimization process (blue box in Figure 6.2) contains models of the nonlinear system dynamics and nonlinear control law, as well as constraints representing actuation limits and environmental obstacles. The onboard flight control system tracks the optimum reference trajectories generated by the NMPH.

From an RL perspective, at each episode the drone is commanded to fly through k terminal setpoints. Hence, each episode consists of k iterations. Following each iteration, three observations are sent to the Agent: initial velocity v_o, angle φ between the initial velocity vector v_o and the vector $\vec{r} = p_{ss} - p_o$ running from the initial point p_o to the terminal point p_{ss}, and the distance $|\vec{r}|$.

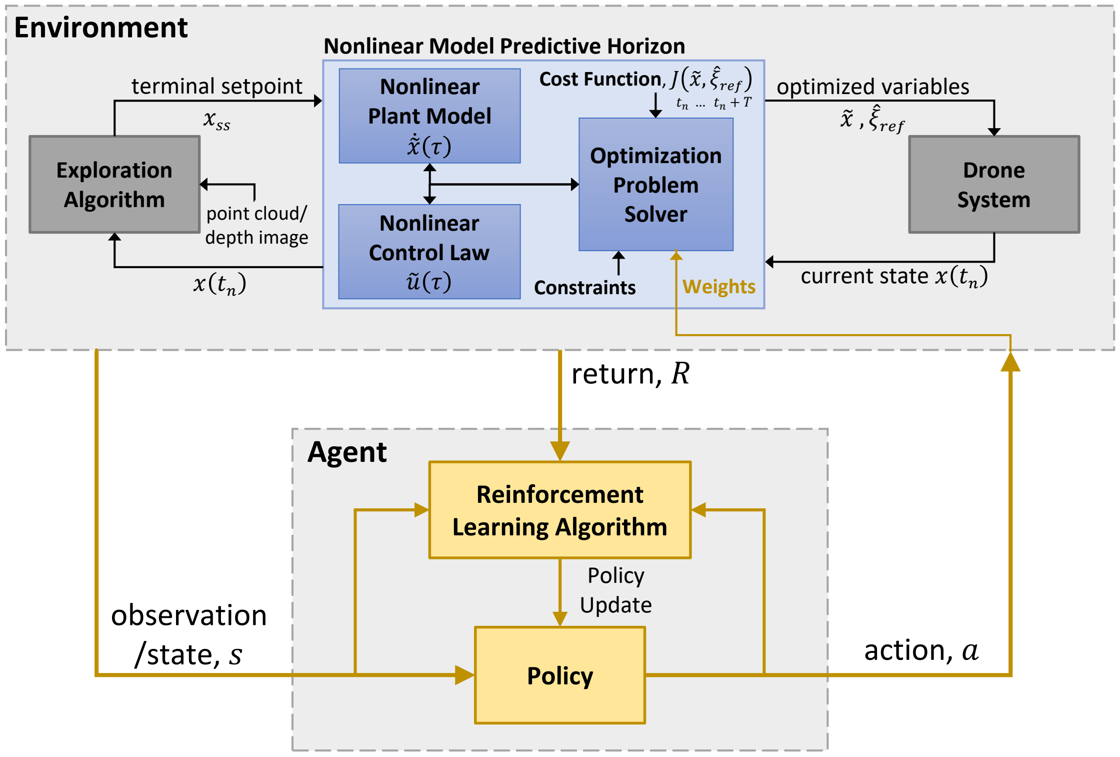


Figure 6.2: Adaptive NMPH Architecture

A sketch of the observations $\{v_o, \varphi, |\vec{r}|\}$ for one iteration is given in Figure 6.3.

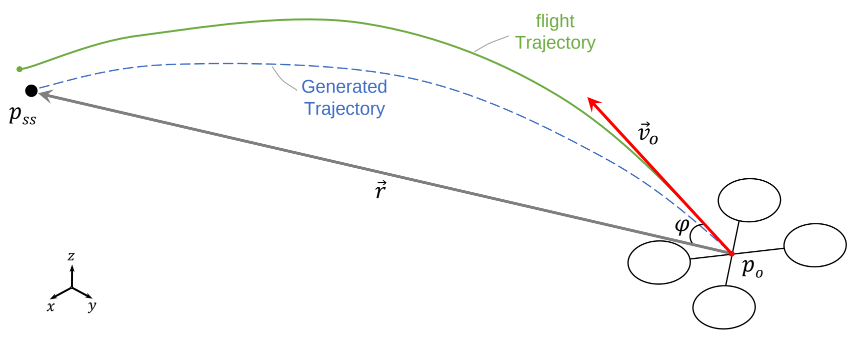


Figure 6.3: Observations from the Environment for one Iteration

Our objective is to tune the NMPH parameters online by using reinforcement learning to maximize the total reward. This reward is a function of the tracking performance by the drone of the reference path generated by the NMPH algorithm, which consists of three indicators:

- *Trajectory tracking reward*, which reflects how well the flight trajectory matches the generated reference. The trajectory tracking reward is calculated as

$$r_{traj} = \begin{cases} -\frac{r_{t,max}}{r_{t,th}} e_{t,\text{RMS}} + r_{t,max}, & \text{for } e_{t,\text{RMS}} \leq r_{t,th} \\ 0, & \text{otherwise} \end{cases}$$

  where $e_{t,\text{RMS}}$ is Root-Mean-Square (RMS) error between the generated and flight trajectories, and $r_{t,max}$ and $r_{t,th}$ are the maximum and threshold values of the trajectory tracking reward, respectively.

- *Terminal setpoint reward*, which reflects how close the ending point of the flight trajectory is to the terminal setpoint of the reference trajectory. The terminal setpoint reward is calculated as

$$r_{ss} = \begin{cases} -\frac{r_{s,max}}{r_{s,th}} e_{ss} + r_{s,max}, & \text{for } e_{ss} \leq r_{s,th} \\ 0, & \text{otherwise} \end{cases}$$

  where $e_{ss} = \|p_{ss} - \hat{\xi}_{ref}^{pos}(t_n + T)\|$ is the error between the terminal point and the final point of the reference trajectory generated by the NMPH, and $r_{s,max}$, $r_{s,th}$ are the maximum and threshold values of this reward, respectively.

- *Completion reward* which reflects how far the drone travels along its prescribed flight trajectory in the associated time interval. This is given by

$$r_c = \begin{cases} -\frac{r_{c,max}}{r_{c,th}} e_c + r_{c,max}, & \text{for } e_c \leq r_{c,th} \\ -5, & \text{otherwise} \end{cases}$$

  where $e_c = \|p_{ss} - p|_{t_n+T}\|$ is the error between the drone's position at $t_n + T$ and the flight trajectory's endpoint, while $r_{c,max}$, $r_{c,th}$ are respectively the maximum and threshold values of the completion reward. We place more importance on this factor by reducing the total reward ($r_c < 0$) whenever the error e_c exceeds the assigned threshold value $r_{c,th}$. Consequently, the overall algorithm will give priority to ensuring the drone reaches the desired setpoint in the allotted timeframe.

6.3.2 DRL-based Adaptive NMPH Architecture

The objective of adaptive NMPH is to integrate deep learning, here chosen as an Actor-Critic method (DDPG or SAC) within the NMPH optimization problem to adaptively tune the NMPH parameters and thus provide the best possible reference flight trajectories for the drone.

The structures of the NMPH-DDPG and NMPH-SAC algorithms are illustrated in Figures 6.4 and 6.5, respectively. Both DRL structures contain two parts, the Actor and the Critic. The Actor contains the policy network, which selects the action which maximizes the total reward (a function of the state of the vehicle and environment) and subsequently improves the policy based on feedback from the Critic. A target policy network is used in DDPG to obtain a stable learning process, while SAC doesn't need a target network because of its probabilistic nature. The Critic is responsible for policy evaluation; within DDPG it consists of a Q-network and a target Q-network, while in SAC it is composed of two Q-networks, two target Q-networks, and an optimization problem for α tuning. Both DDPG and SAC employ a replay buffer to store previous experiences which are used to refine the Actor and Critic networks. The policy evaluation and improvement processes within DDPG and SAC are explained in Sections 6.2.2 and 6.2.3, and depicted in Figures 6.4 and 6.5, respectively.

The action produced by the Actor is a vector of positive values representing the entries of the weighting matrices used in the NMPH optimization problem. Using these, NMPH calculates its stage and terminal cost functions used to perform its optimization and generates the estimated reference trajectory $\hat{\xi}_{ref}$. This result is used by the drone's flight control system, and the vehicle's resulting trajectory is used to calculate the observations $\{v_o,\ \varphi,\ |\vec{r}|\}$ and the total reward $r_t = r_{traj} + r_{ss} + r_c$ sent to the replay buffer to be used in the learning process.

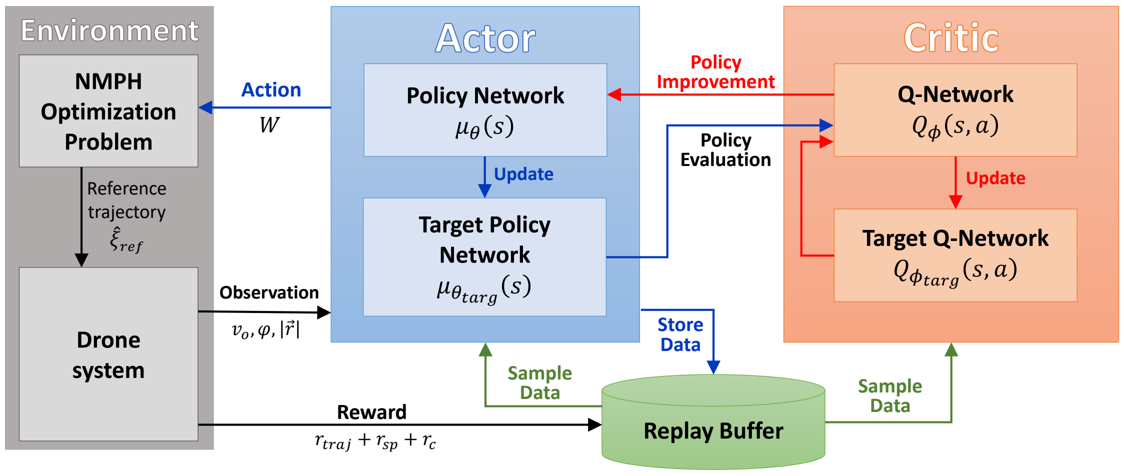
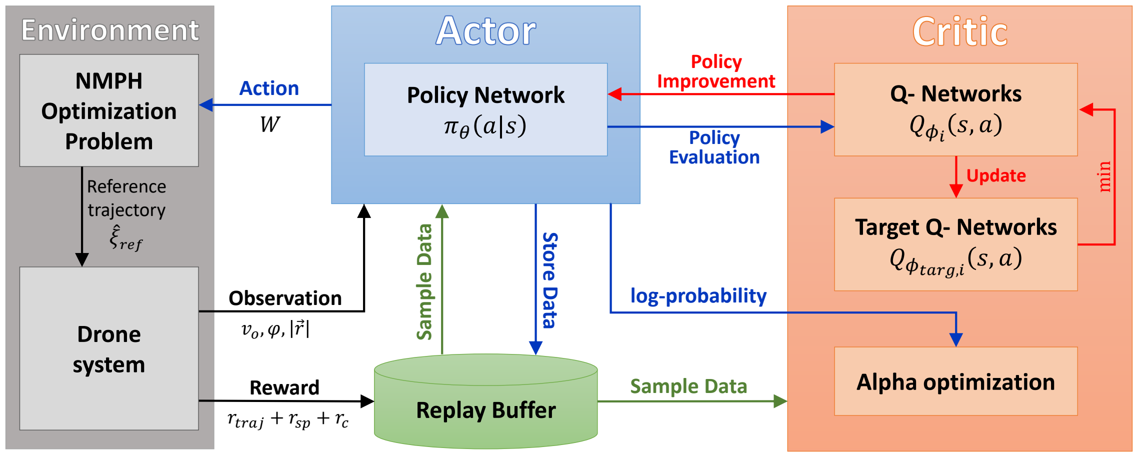


Figure 6.4: Adaptive NMPH-DDPG structure



Environment
NMPH Optimization Problem
Reference trajectory $\hat{\xi}_{ref}$
Drone system
Action
W
Observation
$v_o, \varphi, |\vec{r}|$
Reward
$r_{traj} + r_{sp} + r_c$
Actor
Policy Network $\pi_\theta(a|s)$
Sample Data
Store Data
Replay Buffer
Policy Improvement
Policy Evaluation
log-probability
Sample Data
Critic
Q- Networks $Q_{\phi_i}(s,a)$
Update
min
Target Q- Networks $Q_{\phi_{targ,i}}(s,a)$
Alpha optimization

Figure 6.5: Adaptive NMPH-SAC structure

6.4 Implementation and Evaluation

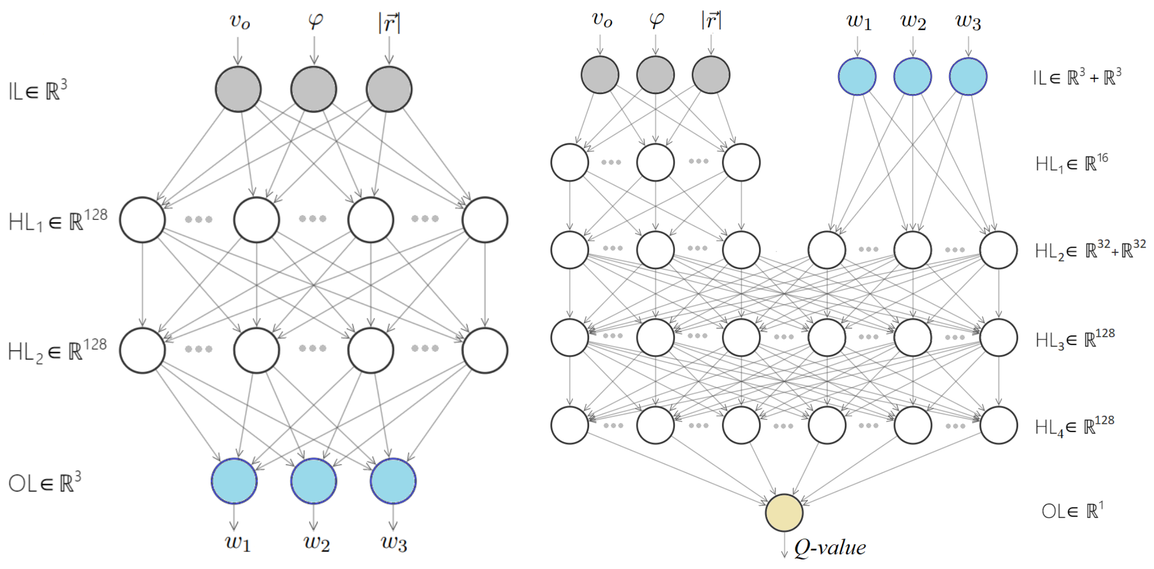
This Section evaluates the effectiveness of tuning the NMPH parameters in real-time via two DRL algorithms (DDPG and SAC). It also assesses the sample complexity and stability of both methods.

The overall architecture is implemented within the Robot Operating System (ROS) [95] which handles the interactions between the various subsystems including physics simulation, optimization calculations and DRL algorithm. The AirSim open-source simulator [96] is used to simulate the physics of the drone and provides photo-realistic environment data. For optimization, the ACADO Toolkit [83] is used to solve the NMPH's optimization problem in real-time. The TensorFlow [120] and Keras [121] libraries are used to train the deep neural networks within the DDPG and SAC algorithms. In addition, the TensorLayer library [122] was used to tailor the SAC algorithm to our application. TensorLayer is a TensorFlow-based package which offers various RL and DRL modules for learning system implementations.

As stated in Section 6.3.2, three observations of the system are fed back to the individual neural networks: v_o, φ, and $|\vec{r}|$. DDPG is very sensitive to hyperparameters when the action space has a high dimension, in which case achieving stable learning becomes challenging. Therefore, we employ only three actions corresponding to the weights of the NMPH optimization dealing with position states. The learning process for the three weight factors $\{w_1 = w_x,\, w_2 = w_y,\, w_3 = w_z\}$ is performed using DDPG and SAC in parallel for comparison purposes.

Each episode is composed of a sequence of iterations, where each iteration represents a trajectory between two endpoints (terminal points). At the start of each iteration, the velocity vector of the drone v_o, the angle φ between the velocity and endpoint-to-endpoint vectors, and the distance $|\vec{r}|$ between endpoints are calculated, followed by the errors $\{e_{t,\mathrm{RMS}},\, e_{ss},\, e_c\}$ and the total reward at the end of the iteration. All this data is stored in the replay buffer. In order to cover a wider portion of the state and action spaces of the system, the initial velocity is randomly selected at the beginning of each episode.

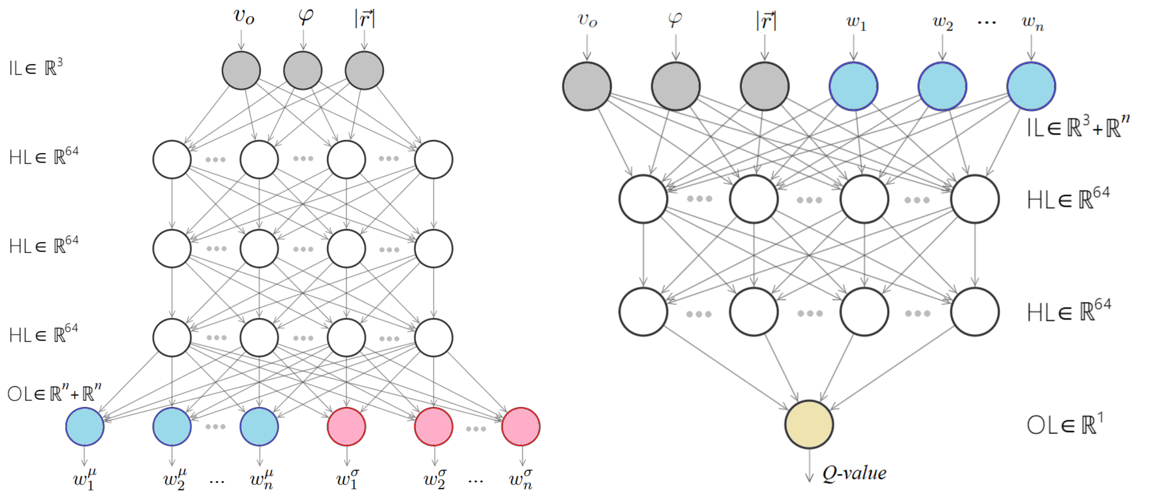
The structures of the Actor-Critic DRL (policy and Q-networks) for DDPG and SAC algorithms are presented in Figures 6.6 and 6.7, respectively. Each network is composed of an input layer, multiple hidden layers, and an output layer. Figure 6.6 and 6.7 depict our neural network designs in terms of the layer structure of each network and the number of nodes in each layer. The policy networks in the Actor are responsible for generating actions that maximize the total reward based on observations of the environment, while the Q-networks in the Critic compute a Q-value which is used for policy improvement. For DDPG, four networks are used: a policy network, a Q-network (depicted in Figure 6.6), a target policy network and a target Q-network. The target networks are replicas of the policy and Q-networks with a delay added to their parameters. Meanwhile, SAC consists of five networks: a policy network, two Q-networks, and two target Q-networks. The SAC's policy and Q-network structures are shown in Figure 6.7.



v_o
φ
$|\vec{r}|$
$\mathrm{IL} \in \mathbb{R}^3$
$\mathrm{HL}_1 \in \mathbb{R}^{128}$
$\mathrm{HL}_2 \in \mathbb{R}^{128}$
$\mathrm{OL} \in \mathbb{R}^3$
w_1
w_2
w_3
$\mathrm{IL} \in \mathbb{R}^3 + \mathbb{R}^3$
$\mathrm{HL}_1 \in \mathbb{R}^{16}$
$\mathrm{HL}_2 \in \mathbb{R}^{32} + \mathbb{R}^{32}$
$\mathrm{HL}_3 \in \mathbb{R}^{128}$
$\mathrm{HL}_4 \in \mathbb{R}^{128}$
$\mathrm{OL} \in \mathbb{R}^1$
Q-value

(a) Actor Network (Policy Network) (b) Critic Network (Q Network)

Figure 6.6: Neural networks used by DDPG. IL: Input Layer, HL: Hidden Layer, OL: Output Layer.



v_o
φ
$|\vec{r}|$
$\mathrm{IL} \in \mathbb{R}^3$
$\mathrm{HL} \in \mathbb{R}^{64}$
$\mathrm{HL} \in \mathbb{R}^{64}$
$\mathrm{HL} \in \mathbb{R}^{64}$
$\mathrm{OL} \in \mathbb{R}^n + \mathbb{R}^n$
w_1^μ w_2^μ ... w_n^μ w_1^σ w_2^σ ... w_n^σ
w_1 w_2 ... w_n
$\mathrm{IL} \in \mathbb{R}^3 + \mathbb{R}^n$
$\mathrm{HL} \in \mathbb{R}^{64}$
$\mathrm{HL} \in \mathbb{R}^{64}$
$\mathrm{OL} \in \mathbb{R}^1$
Q-value

(a) Actor Network (Policy Network) (b) Critic Network (Q Network)

Figure 6.7: Neural networks used by SAC. IL: Input Layer, HL: Hidden Layer, OL: Output Layer.

Figure 6.8 shows the average episodic reward during the training processes of the DDPG and SAC architectures. In this comparison, each framework is learning to optimize the values of only three actions, which represent the entries of the weight matrix corresponding to the position states within the NMPH optimization problem.

To enhance DDPG performance in terms of sample complexity and its sensitivity to hyperparameters, we propose and apply a 'pre-exploration' technique which traverses the RL problem spaces before the training process is started. Pre-exploration is performed by applying a set of predefined actions, and considering a random system state for each action. The collected experiences of the pre-exploration process are then stored in the replay buffer, which is used during the training process. It was found that using this technique helps DDPG to improve convergence and stability over the case without pre-exploration, as can be seen from Figure 6.8. Conversely, a number of episodes must be spent for pre-exploration, which delays the learning process in the real-time adaptation. Note the results shown in Figure 6.8 also show that SAC generally outperforms DDPG (either with or without pre-exploration) in terms of learning speed. In addition, during the training process SAC showed noticeably better learning stability relative to DDPG w.r.t. the process of selecting the hyperparameter values for each algorithm.

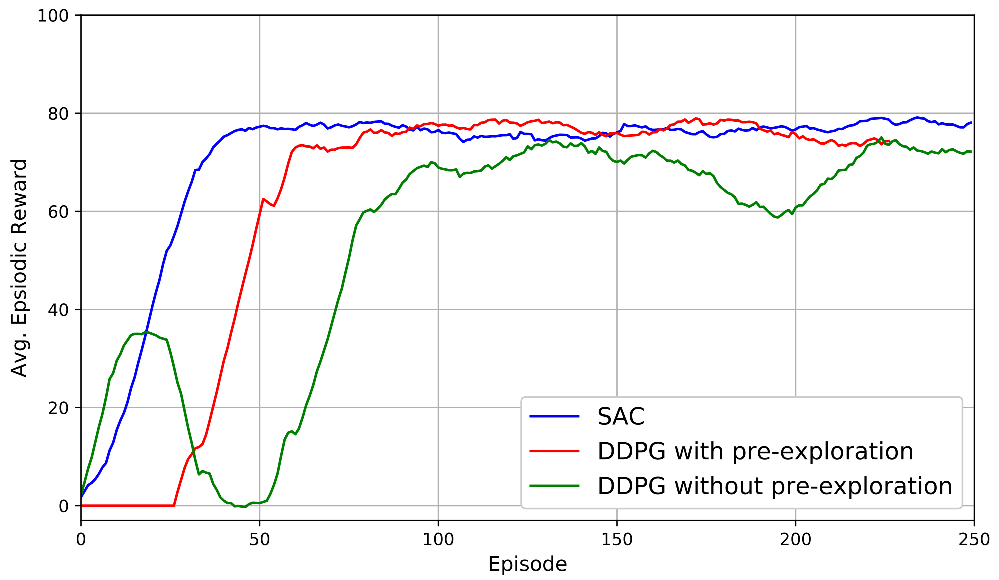


Figure 6.8: Training curves of SAC, DDPG with pre-exploration, and DDPG without pre-exploration for adaptively tuning three NMPH parameters.

To test the performance of the SAC approach in a higher-dimensional setting, the number of actions was increased to 12 to estimate the weight matrix gains corresponding to the position, velocity and acceleration states $\{w_x,\ w_y,\ w_z,\ w_\psi,\ w_{\dot{x}},\ w_{\dot{y}},\ w_{\dot{x}},\ w_{\dot{\psi}},\ w_{\ddot{x}},\ w_{\ddot{y}},\ w_{\ddot{x}},\ w_{\ddot{\psi}}\}$ within the NMPH optimization problem. Figure 6.9 shows the resulting training curve of SAC; DDPG failed to complete the learning process in this case. The effect of increasing the number of NMPH parameters being tuned can be seen by comparing the SAC training curves in Figures 6.8 and 6.9 in terms of the average episodic reward. In the 12-parameter trial, SAC has better training performance than in the 3-parameter case, which is because the former covers a larger action space and consequently provides better solutions of the NMPH optimization problem.

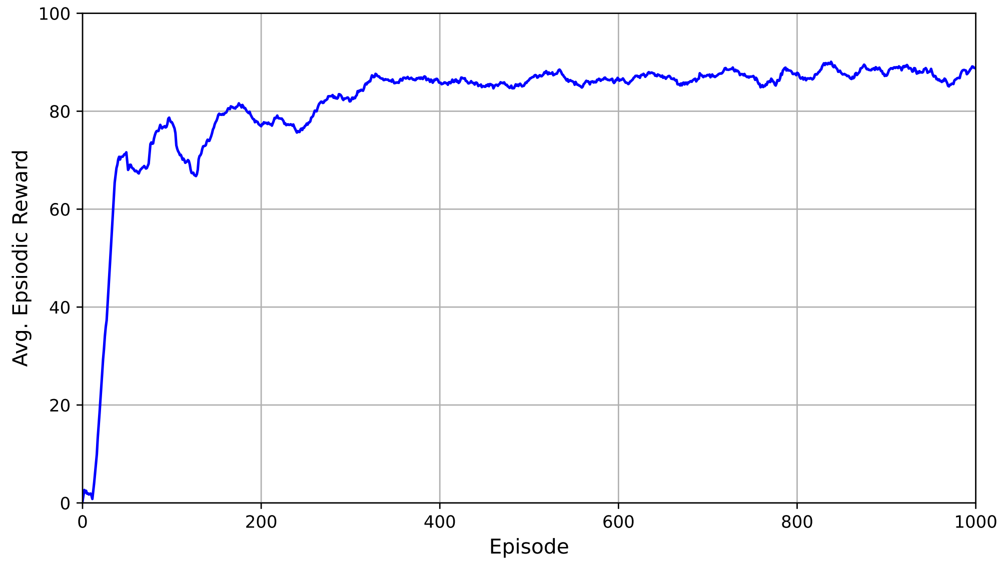


Figure 6.9: Training curve of SAC adaptively tuning 12 parameters of the NMPH optimization.

To test the trajectory planning performance of NMPH with and without the proposed adaptation scheme, four different flight tests were performed within the AirSim simulation environment. For the second case, the weighting matrices within NMPH used fixed parameters, which were used as the initial values in the DRL-based adaptation method. Table 6.1 provides a comparison between the conventional NMPH design with fixed parameter values and the adaptive NMPH-SAC design. The comparison is based on the average of the error metrics discussed in Section 6.3.1, namely

$e_{t,\text{RMS}}$, e_{ss}, and e_c. Each flight trajectory consists of ten trials, and each trial includes five iterations. The initial velocity and drone orientation were selected randomly at the beginning of each trial. The first trial uses a zigzag pattern, which consists of five paths, each with length 5.6m. For the second trial (square pattern), the side length was 5m. For the third trial (ascending square pattern), the elevation gain was set to 1m. The fourth trial involved a set of position setpoints provided by a Graph-based exploration algorithm (see [79] for the complete details). As shown in Table 6.1, the flight performance obtained with the adaptive NMPH is much better than the one from the non-adaptive (conventional) NMPH. The reason for this is that real-time adaptation of NMPH parameters works better than using a single set of fixed values when performing a variety of different flying trajectories.

Table 6.1: Comparison between the conventional NMPH design (fixed values of the NMPH parameters) and the adaptive NMPH-SAC approach, for different flight trials.

	Average error	Zigzag pattern	Square pattern	Ascending square pattern	Random setpoints (Exploration)
Fixed NMPH parameters	$e_{t,\text{RMS}}$	0.11353	0.09758	0.10741	0.09646
	e_{ss}	0.08659	0.07547	0.07663	0.07339
	e_c	0.12033	0.06426	0.07413	0.07739
Adaptive NMPH-SAC	$e_{t,\text{RMS}}$	0.08877	0.08495	0.09212	0.06749
	e_{ss}	0.01029	0.00919	0.01046	0.01150
	e_c	0.04400	0.04419	0.04952	0.05874

To show how the values of the NMPH parameters are adjusted online using SAC, Figures 6.10 and 6.11 present the results of a flight through 20 randomly generated setpoints. Figure 6.10 depicts the values of the observations v_o, φ and $|\vec{r}|$ at the beginning of each iteration and Figure 6.11 shows the changing values of the NMPH weighting matrix entries.

6.5 Conclusions

This chapter presented a DRL-based adaptive scheme to tune the optimization parameters of our previously-proposed NMPH trajectory planning approach. The overall design aims to provide the best-performing flight trajectory generation for an aerial drone across a wide range of flight patterns and environments by tuning these parameters in real-time flights instead of selecting them a-priori. The adaptation scheme is implemented through two different Actor-Critic DRL algorithms — the deterministic DDPG and the probabilistic SAC.

The two variants of DRL-based NMPH were trained and tested on an aerial drone in a simulation environment. The results showed a marked improvement in flight performance when using the adaptive NMPH-DDPG and NMPH-SAC over the conventional NMPH. Comparisons between DDPG and SAC showed that the latter outperforms the former in terms of learning speed, ability to handle a larger set of tuning parameters, and overall flight performance.

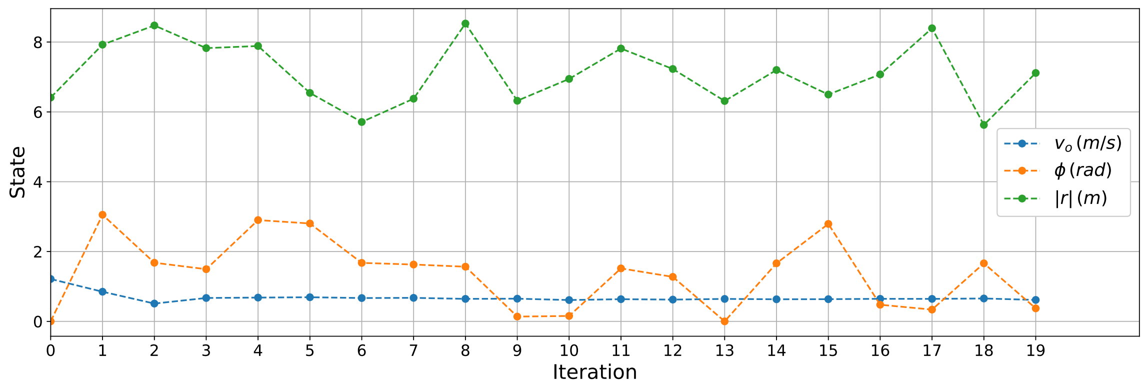


Figure 6.10: Observations at start of iterations.

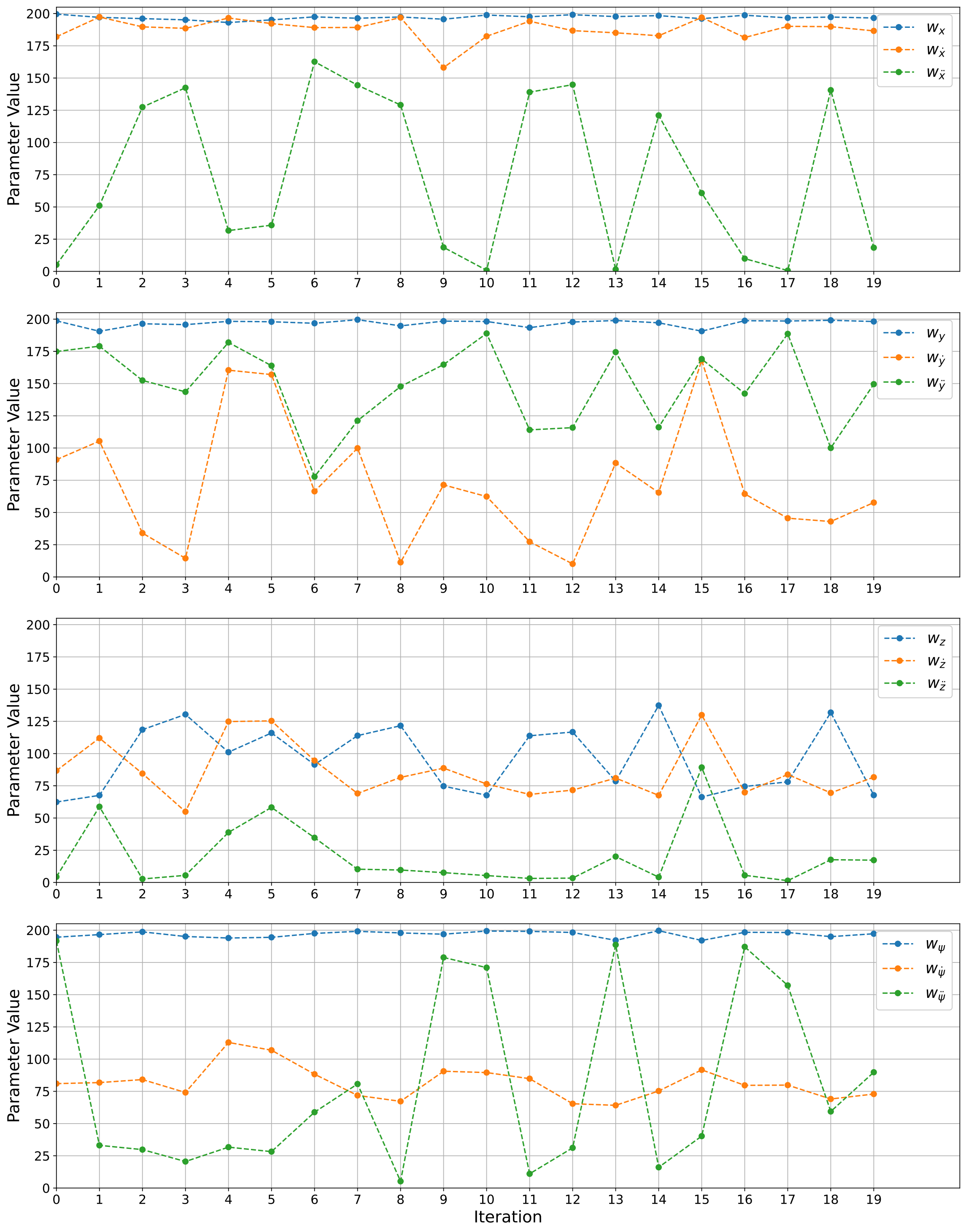


Figure 6.11: Values of NMPH weighting matrix entries being adjusted online by SAC.

Chapter 7

Hardware Platform and Software Architecture

7.1 Hardware Platform

Our modular drone system design consists of state-of-the-art computing units for real-time environment perception and flight control, onboard vision and 3-D laser scanning sensors, and a modular airframe for carrying the overall payload; and includes a software stack running the implementations of the NMPH designs described in this thesis.

The drone is built on a DJI FlameWheel F550 hexacopter platform. Figure 7.1 provides a high-level illustration and overview of our system.

The hexacopter vehicle is equipped with a Pixhawk 5X flight controller running either PX4 or ArduPilot autopilot system, plus an NVIDIA Jetson Xavier NX single-board computer running ROS Melodic Morenia under Ubuntu 18.04. In addition to a computer vision system, a Velodyne Puck LITE LiDAR sensor is mounted on the drone to provide 360° point cloud data for mapping and state estimation. The drone system used for testing is depicted in Figure 7.2.

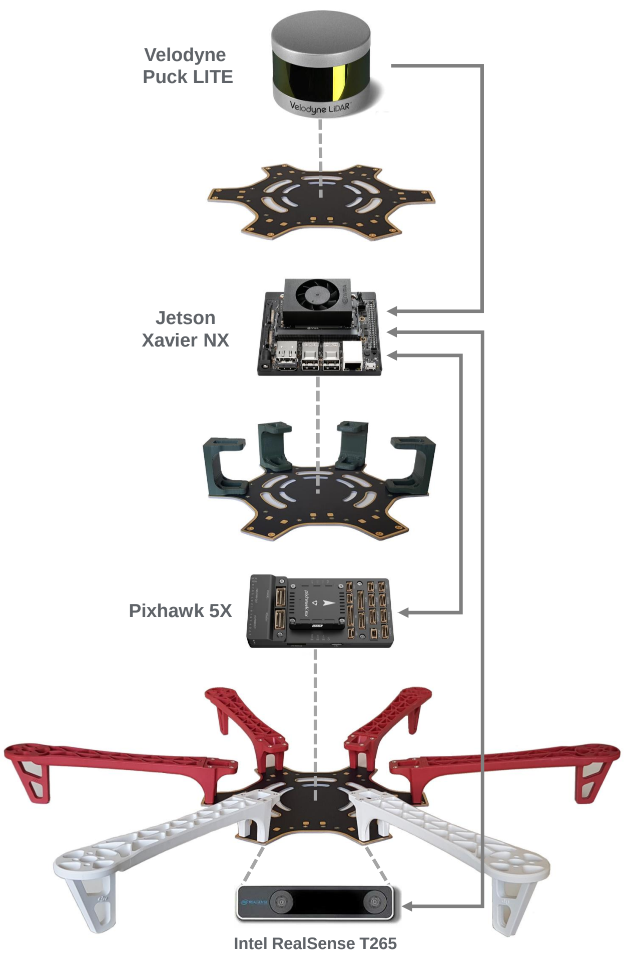


Figure 7.1: Exploded view of the hardware setup used in our autonomous drone.

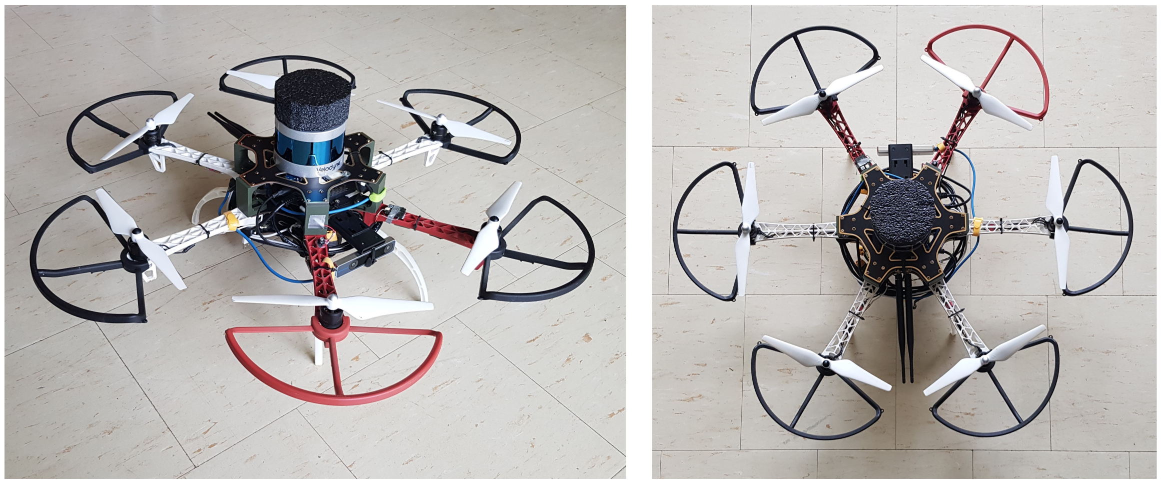


Figure 7.2: The drone system used for testing.

The design choices for the different system components used for our vehicle are explained below.

7.1.1 High-level onboard computing module

The high-level onboard computer has the processing power required to run the software stack, including estimation, optimization-based solver, global motion planning, and other applications. The module used in our work (NVIDIA Jetson Xavier NX) supports fast prototyping, flight experimentation, and a wide variety of Linux-based software.

The NVIDIA Jetson Xavier NX is a powerful device that offers excellent computational performances along with low power consumption. The board is equipped with 6-core Carmel ARM v8.2 64-bit CPU, 384 NVIDIA CUDA Cores GPU, and 8 GB of LPDDR4x RAM, with all running under Linux-based system (here Ubuntu 18.04). The Jetson Xavier NX development kit has a mass of 172 grams and consumes 10 to 20 W based on the running condition. The kit provides 4 x USB 3.1 ports, Ethernet, and other interfaces for display, storage, and wireless modules.

This Xavier NX onboard computer is perfect for implementing and processing multiple tasks including deep learning models. The Xavier NX board runs the NVIDIA JetPack software, which includes the Jetson Linux Driver Package and CUDA libraries that are suitable for computer vision, optimization, and navigation applications.

7.1.2 Low-level flight controller

The low-level flight controller offers real-time and reliable control. Pixhawk 5X is a powerful and low-latency flight controller based on the Pixhawk FMUv5X open-source project. The FMUv5X open standard includes high-performance FMU Processor (32 Bit Arm Cortex-M7, 216MHz, 2MB memory, 512KB RAM); three low-noise and Temperature-controlled IMUs; two barometers; and a vibration isolation system.

Pixhawk supports two well-known open-source autopilot firmware projects, PX4 and ArduPilot. Both projects provide flexible set of tools for our design solutions and applications. Both PX4 and ArduPilot were tested in our application, and found to provide good flight control performance in terms of flight stability and safety. After conducting many flight trials, ArduPilot was chosen over the PX4, as the former provided more stable state estimation which did not exhibit drift when deployed in indoor environments.

7.1.3 Sensors

To navigate through unknown, unstructured and GPS-denied environments, our drone uses different sensing devices to measure the drone's position with respect to its surroundings. The sensors used in our drone include a stereo camera, 3-D laser scanner, plus the various inertial sensors equipped onboard the Pixhawk 5X flight controller.

A Velodyne Puck LITE was used to acquire pointcloud measurements used for mapping and state estimation. The Puck LITE is a 16-channel LiDAR device that provides 10Hz laser scan measurements with a range of $100\,m$ and a field of view of $(360°, 30°)$. The device weighs approximately 0.55kg and acquires 300,000 points/second in single return mode.

An Intel RealSense T265 tracking camera was used as an alternative source of state estimation. The T265 camera has a mass of only 55 g, and comes with a visual SLAM implementation in its firmware which provides 6-DOF estimation of the camera's position and velocity relative to the environment at 200 Hz. The camera consists of two global shutter fisheye cameras with 173° diagonal field of view and 848×800 pixel resolution, a 3-DOF gyroscope, and a 3-DOF accelerometer.

7.1.4 Airframe

The DJI FlameWheel F550 hexacopter is a multi-rotor flying platform that provides high thrust-to-weight ratio and modular design features, making it a suitable choice for our application. The DJI F550 strucutural elements are made from PA66+GF30 (glass fibre-reinforced polyamide plastic), and the drone is equipped with six brushless DC motors with Electronic Speed Controllers (ESCs), 8-inch propellers, and an a central PCB board for wiring between the battery and ESCs.

7.2 Software Architecture

Figure 7.3 shows the software architecture used in our system. The algorithms are implemented within the Robot Operating System (ROS) [95], a Linux-based system that handles communication between the individual software modules and the vehicle's hardware, either simulated or real [84]. The ACADO Toolkit [83] is used for optimization calculations. The optimization problem is programmed in a self-contained C++ environment within this toolkit, then a real-time nonlinear solver is generated to run the optimizations online. The resulting code can be compiled and run within ROS. The NMPH optimization problem (2.11) was written in C++ code using ACADO, then automatically converted into efficient C code which is able to solve the optimization problem in real-time.

The drone's pose and pointcloud information obtained from the onboard sensors is sent to the global motion planner, which uses Voxblox [104] to build a volumetric map of the environment to distinguish between unknown, free, and occupied spaces. The global motion planner integrates a graph-based planning approach called GBPlanner [8, 105, 106] with the NMPH algorithm to generate optimal reference trajectories for the vehicle, which are forwarded to the drone through MAVROS for trajectory tracking purposes, where MAVROS is a ROS package that enables communication through the MAVLink communication protocol [123]. The ROS tool RViz is a ROS tool that is used for robot monitoring and 3D visualization, while QGroundControl [124] is a software that provides mission flight planning and control for MAVLink-based drones.

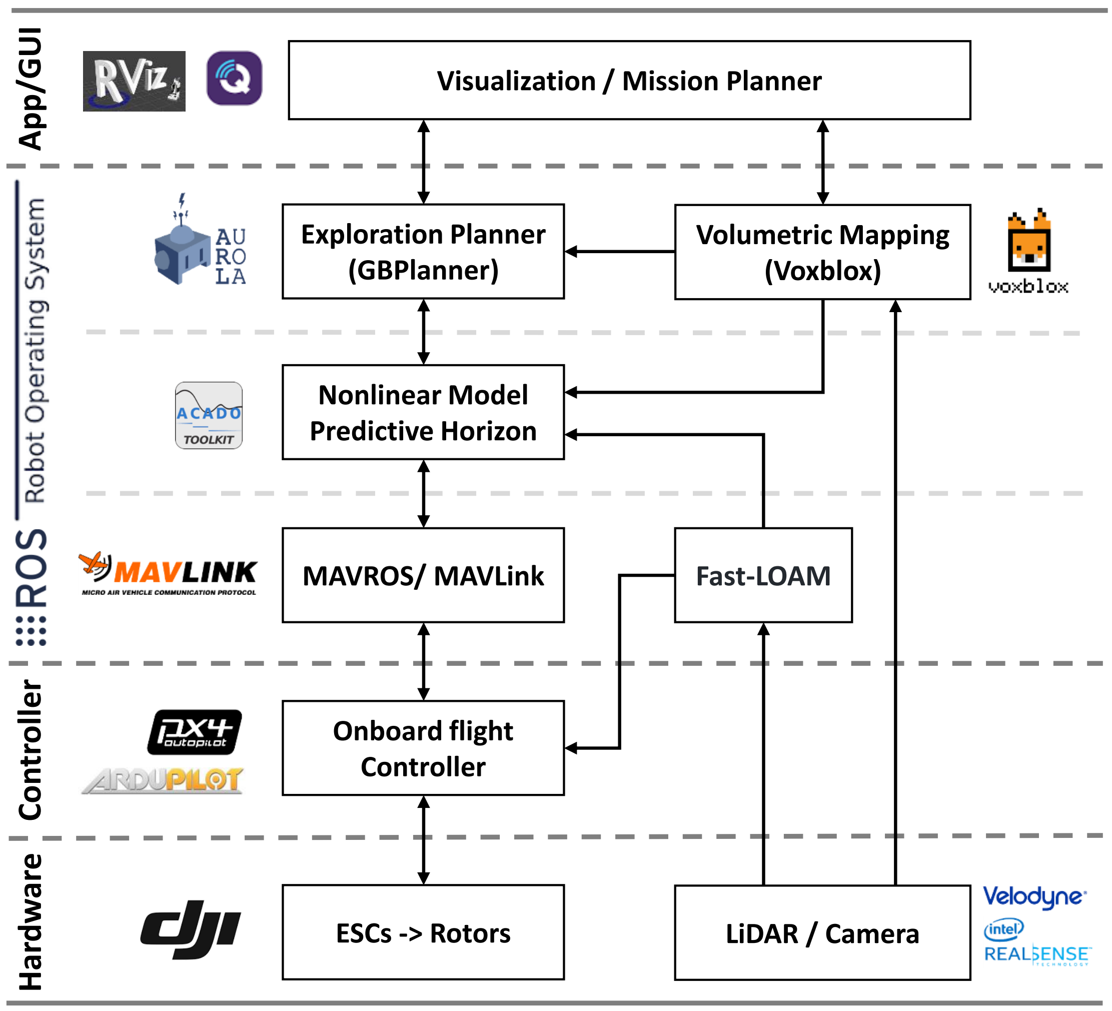

App/GUI
Visualization / Mission Planner
ROS Robot Operating System
Exploration Planner (GBPlanner)
Volumetric Mapping (Voxblox)
voxblox
AU RO LA
Nonlinear Model Predictive Horizon
ACADO TOOLKIT
MAVLINK
MAVROS/ MAVLink
Fast-LOAM
Controller
Onboard flight Controller
PX4 autopilot
ARDUPILOT
Hardware
ESCs -> Rotors
LiDAR / Camera
Velodyne
intel REALSENSE

Figure 7.3: Software Architecture used in our system and the development of the presented approaches.

Our setup provides different solutions to estimate the pose of the drone relative to its environment. The first one is a stereo camera-based solution which relies on a proprietary Visual-SLAM algorithm running onboard the Intel RealSense T265 stereo camera, while the second one is a LiDAR-based solution that uses a front-end SLAM algorithm called Fast Lidar Odometry And Mapping (Fast-LOAM a.k.a. FLOAM) [125]. The autopilot then fuses the pose estimates (either from T265 or FLOAM) with IMU data using an Extended Kalman Filter (EKF) to provide accurate estimates of the drone's states, which are subsequently employed for mapping, motion planning, and flight control.

7.3 Summary

In this chapter, we presented an overview of the hardware setup of the hexacopter drone used in the real-time flight experiments. The software architecture used to implement our proposed approaches was also described.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

In this dissertation, a novel optimization-based reference trajectory generation method for a nonlinear system (here an aerial drone) was developed, named Nonlinear Model Predictive Horizon or NMPH. Different algorithms and methodologies complimentary to NMPH were also proposed to provide a comprehensive solution for a vehicle to autonomously navigate and explore an unknown GPS-denied environment. The following sections summarize and conclude the work presented in this thesis.

8.1.1 Nonlinear Model Predictive Horizon

The main part of this research focused on proposing a novel optimization-based reference trajectory generation framework for a nonlinear closed-loop system based on the NMPC approach. The proposed formulation (NMPH) considered a nonlinear control law and a nonlinear plant model within its optimization problem constrains, aiming for a closed-loop dynamics model with decreased non-convexity used by the online optimization problem to generate feasible and optimal reference trajectories for the actual closed-loop system. The proposed NMPH algorithm was presented in discrete and continuous time domains, and supports both static and dynamic obstacles, enabling trajectory generation in continuously changing environments.

8.1.2 NMPH with Feedback Linearization

In the second part of the research, a feedback linearization control law was developed within the NMPH framework. The proposed formulation is abbreviated as NMPH-FBL. The feedback linearization control law converts the closed-system to a linear form, which results in reduced non-convexity of the optimization problem to generate feasible and optimal reference trajectories for the actual closed-loop system. In addition, the feedback linearization design uses integral states to compensate for modeling uncertainties and external disturbances present in the actual system. The proposed NMPH-FBL algorithm supports both static and dynamic obstacles as constrains to generate continuous trajectories in dynamic environments.

The NMPH-FBL approach was implemented on a simulated quadrotor drone and validated to generate 3D optimal reference trajectories in real time. Different simulation scenarios and hardware-in-the-loop (HITL) simulations were carried out to evaluate the performance of the proposed method. Convergence of the predicted and estimated trajectories, trajectory generation under different initial conditions, trajectory tracking performance, and the ability to navigate around static and dynamic obstacles were validated through simulation results.

8.1.3 NMPH with Backstepping Control

In the third part of the research, an enhancement to the NMPH was developed by using a Backstepping control law within the optimization problem of NMPH instead of feedback linearization. The NMPH-BSC design was shown to offer additional implementation advantages over NMPH-FBL including the ability to readily extend to more complicated plant models and avoiding numerical differentiation. We showed that the NMPH-BSC reduced the nonlinearities of the overall closed-loop system, and thus improved the real-time computation of optimal trajectories by reducing the non-convexity of the optimization problem.

The resulting NMPH-BSC design was tested in simulation and hardware flight experiments on quadrotor and hexarotor drone vehicles, respectively. The results showed an improvement in performance over conventional path planning algorithms, and demonstrated the algorithm's ability to generate optimal and smooth trajectories for the vehicle while avoiding static and dynamic obstacles.

8.1.4 Global Motion Planning using NMPH approaches

The fourth part of this research presented a methodological motion planning approach for drone exploration in GPS-denied environments. This motion planing method integrated the NMPH approach with a graph-based planner in a three-stage modular design: building an incremental volumetric map of the environment, generating stabilization setpoints within this map using a graph-based planner, and using the NMPH approach to produce local optimal trajectories to the generated stabilization setpoints. In order to smoothly integrate the three stages, computationally efficient algorithms for obstacle mapping and avoidance plus robust path guidance were developed and implemented.

The developed motion planner was evaluated through a series of simulation flights as well as a real-time hardware flight test to validate the performance of the proposed design on quad- and hexrotor drones navigating within unexplored GPS-denied environments. The results showed the ability of the proposed design to improve motion planning performance over conventional techniques and generate smooth and safe flight trajectories in a computationally efficient way.

8.1.5 Adaptive NMPH Design using Deep Reinforcement Learning

In the final part of this research, an adaptive scheme optimization-based trajectory planning approach was presented, which aims to continuously plan the best-performing flight trajectories for an aerial drone across a wide range of flight envelopes and environments by online tuning the NMPH parameters in real-time flights rather than selecting them a-priori. The adaptation scheme was implemented through two different Actor-Critic DRL algorithms, the deterministic DDPG and the probabilistic SAC.

The presented designs were trained and tested in a simulation environment. The results showed a marked improvement in flight performance when using the adaptive NMPH-DDPG and NMPH-SAC over the conventional NMPH. Comparisons between DDPG and SAC showed that the latter outperforms the former in terms of learning speed, ability to handle a larger set of tuning parameters, and overall flight performance.

The pros, cons and limitations of this study are summarized as follows:

- Pros:
  - The proposed design is able to dynamically adjust the parameters of the optimization problem online during flight, which is preferable to tuning them before flight and evaluating the resulting performance afterwards.
  - The DRL model can adapt the gains of the optimization problem in response to changes in the vehicle, such as new payload configurations or replacing hardware components.
- Cons:
  - DRL algorithms employ a large number of hyperparameters. While SAC is less sensitive to hyperparameters than DDPG, finding the best combination of these parameters to achieve fast training is a challenging task.
- Limitations:
  - The present study was performed entirely within a simulation environment, and does not include hardware testing results.

8.2 Future Perspectives

8.2.1 Experimental Testing

In this work, tremendous efforts were made in implementing and testing most of the presented techniques in a variety of real-world settings. Further experiments should be performed to challenge the presented techniques in harsh environments. Some of the possible steps towards this are:

- Testing and validating the proposed methods onboard the hardware drone in large-scale, GPS-denied environments such as subterranean mines.
- Implementing the adaptation scheme (adaptive NMPH) onboard the hardware drone and testing its performance in a variety of real-world environments, as well as using the DRL algorithms for disturbance and parameter estimation.

8.2.2 Towards Faster and Smarter Drone Exploration

A novel local trajectory planning algorithm was introduced and integrated within a modular global motion planner as presented in Chapter 5. In particular, the global motion planner uses a graph-based algorithm for exploration, which suffers from limitations such as exploration speed, high computational power, and the lack of smartness in making decisions based on the environment features. For faster, smarter, and more efficient robot exploration and navigation, future work should consider one of the following directions:

- Integration of the just-released Fast UAV Exploration (FUEL) [126] package into the NMPH framework. Since graph-based exploration is currently the most processor-intensive task in our motion planning pipeline, this new package is expected to greatly increase efficiency and free up resources for other tasks.
- An investigation of using deep learning for the drone's mapping and exploration to achieve smart sensing, cognition, and motion planning in unknown,

unstructured GPS-denied environments [127].

8.2.3 Deep Reinforcement Learning for Intelligent Robots

Nowadays, many research works incorporate machine learning (DRL algorithms in particular) within the various components of a robotic system to create more intelligent machines that are capable of performing challenging tasks in an optimum way. Some possibilities for this project include proposing DRL algorithms with continuous action that are more robust and less sensitive to hyperparameters, and developing DRL algorithms for computer vision applications such as 3D point-cloud registration [128] and depth estimation [129].

8.2.4 Next Generation Drones

One of the main future directions is to design and build a versatile, standardized, and intelligent platform of a drone vehicle with a custom-tailored software stack, which can be used for the next generation of applications related to the fourth industrial revolution and the Internet of Things.

Bibliography

[1] K. Bergman, O. Ljungqvist, T. Glad, and D. Axehill, “An optimization-based receding horizon trajectory planning algorithm,” *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 15 550–15 557, 2020.

[2] A. Manoharan, R. Sharma, and P. Sujit, “Multi-AAV cooperative path planning using nonlinear model predictive control with localization constraints,” *arXiv preprint arXiv:2201.09285*, 2022.

[3] D. M. Wu *et al.*, “Fast velocity trajectory planning and control algorithm of intelligent 4WD electric vehicle for energy saving using time-based MPC,” *IET Intelligent Transport Systems*, vol. 13, no. 1, pp. 153–159, 2019.

[4] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[5] A. Sozzi, M. Bonfè, S. Farsoni, G. De Rossi, and R. Muradore, “Dynamic motion planning for autonomous assistive surgical robots,” *Electronics*, vol. 8, no. 9, p. 957, 2019.

[6] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, “A survey of motion planning and control techniques for self-driving urban vehicles,” *IEEE Transactions on intelligent vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

[7] S. M. Ahmed, Y. Z. Tan, G. H. Lee, C. M. Chew, and C. K. Pang, “Object detection and motion planning for automated welding of tubular joints,” in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2016, pp. 2610–2615.

[8] T. Dang, F. Mascarich, S. Khattak, C. Papachristos, and K. Alexis, “Graph-based path planning for autonomous robotic exploration in subterranean environments,” in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2019, pp. 3105–3112.

[9] L. Quan, L. Han, B. Zhou, S. Shen, and F. Gao, “Survey of UAV motion planning,” *IET Cyber-systems and Robotics*, vol. 2, no. 1, pp. 14–21, 2020.

[10] P. Corke, *Robotics, vision and control: fundamental algorithms in MATLAB second, completely revised*. Springer, 2017, vol. 118.

[11] A. Gasparetto, P. Boscariol, A. Lanzutti, and R. Vidoni, “Path planning and trajectory planning algorithms: A general overview,” *Motion and operation planning of robotic systems*, pp. 3–27, 2015.

[12] L. Grüne and J. Pannek, *Nonlinear Model Predictive Control: Theory and Algorithms*. Springer, 2017.

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*, Third. MIT press, 2009.

[14] E. W. Dijkstra *et al.*, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[15] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[16] S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain," *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 354–363, 2005.

[17] K. Al-Mutib, M. AlSulaiman, M. Emaduddin, H. Ramdane, and E. Mattar, "D* lite based real-time multi-agent path planning in dynamic environments," in *2011 Third International Conference on Computational Intelligence, Modelling & Simulation*, IEEE, 2011, pp. 170–174.

[18] M. Likhachev, G. J. Gordon, and S. Thrun, "ARA*: Anytime A* with provable bounds on sub-optimality," *Advances in neural information processing systems*, vol. 16, pp. 767–774, 2003.

[19] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Path planning for autonomous vehicles in unknown semi-structured environments," *The international journal of robotics research*, vol. 29, no. 5, pp. 485–501, 2010.

[20] Y. Yang, J. Pan, and W. Wan, "Survey of optimal motion planning," *IET Cyber-systems and Robotics*, vol. 1, no. 1, pp. 13–19, 2019.

[21] T. Siméon, J.-P. Laumond, and C. Nissoux, "Visibility-based probabilistic roadmaps for motion planning," *Advanced Robotics*, vol. 14, no. 6, pp. 477–493, 2000.

[22] S. M. LaValle, J. J. Kuffner, B. Donald, *et al.*, "Rapidly-exploring random trees: Progress and prospects," *Algorithmic and computational robotics: new directions*, vol. 5, pp. 293–308, 2001.

[23] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[24] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Autonomous robot vehicles*, Springer, 1986, pp. 396–404.

[25] I. Iswanto, A. Ma'arif, O. Wahyunggoro, and A. Imam, "Artificial potential field algorithm implementation for quadrotor path planning," *Int. J. Adv. Comput. Sci. Appl*, vol. 10, no. 8, pp. 575–585, 2019.

[26] P. Martin and A. Del Pobil, "Application of artificial neural networks to the robot path planning problem," *WIT Transactions on Information and Communication Technologies*, vol. 6, 1970.

[27] M. Zhao, N. Ansari, and E. S. Hou, "Mobile manipulator path planning by a genetic algorithm," *Journal of Robotic Systems*, vol. 11, no. 3, pp. 143–153, 1994.

[28] H. J. Wang and W. Xiong, "Research on global path planning based on ant colony optimization for AUV," *Journal of Marine Science and Application*, vol. 8, no. 1, pp. 58–64, 2009.

[29] Z. Qiaorong and G. Guochang, "Path planning based on improved binary particle swarm optimization algorithm," in *2008 IEEE Conference on Robotics, Automation and Mechatronics*, IEEE, 2008, pp. 462–466.

[30] H. Martinez-Alfaro and D. R. Flugrad, "Collision-free path planning for mobile robots and/or AGVs using simulated annealing," in *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, IEEE, vol. 1, 1994, pp. 270–275.

[31] H. Y. Zhang, W. M. Lin, and A. X. Chen, "Path planning for the mobile robot: A review," *Symmetry*, vol. 10, no. 10, p. 450, 2018.

[32] J. Canny, A. Rege, and J. Reif, "An exact algorithm for kinodynamic planning in the plane," *Discrete & Computational Geometry*, vol. 6, no. 3, pp. 461–484, 1991.

[33] M. Zucker *et al.*, "Chomp: Covariant hamiltonian optimization for motion planning," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1164–1193, 2013.

[34] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *2011 IEEE international conference on robotics and automation*, IEEE, 2011, pp. 4569–4574.

[35] C. G. Atkeson *et al.*, "Using local trajectory optimizers to speed up global optimization in dynamic programming," *Advances in neural information processing systems*, pp. 663–663, 1994.

[36] A. Perez, R. Platt, G. Konidaris, L. Kaelbling, and T. Lozano-Perez, "LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics," in *2012 IEEE International Conference on Robotics and Automation*, IEEE, 2012, pp. 2537–2542.

[37] M. Nolte, M. Rose, T. Stolte, and M. Maurer, "Model predictive control based trajectory generation for autonomous vehicles—An architectural approach," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2017, pp. 798–805.

[38] O. Andersson, O. Ljungqvist, M. Tiger, D. Axehill, and F. Heintz, "Receding-horizon lattice-based motion planning with dynamic obstacle avoidance," in *2018 IEEE Conference on Decision and Control (CDC)*, IEEE, 2018, pp. 4467–4474.

[39] O. Andersson, M. Wzorek, P. Rudol, and P. Doherty, “Model-predictive control with stochastic collision avoidance using bayesian policy optimization,” in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2016, pp. 4597–4604.

[40] J. Richalet, A. Rault, J. Testud, and J Papon, “Model predictive heuristic control: Applications to industrial processes,” *Automatica*, vol. 14, no. 5, pp. 413–428, 1978.

[41] C. E. Garcia, D. M. Prett, and M. Morari, “Model predictive control: Theory and practice—a survey,” *Automatica*, vol. 25, no. 3, pp. 335–348, 1989.

[42] M. El Ghoumari, H.-J. Tantau, and J Serrano, “Non-linear constrained MPC: Real-time implementation of greenhouse air temperature control,” *Computers and electronics in agriculture*, vol. 49, no. 3, pp. 345–356, 2005.

[43] L. O. Santos, P. A. Afonso, J. A. Castro, N. M. Oliveira, and L. T. Biegler, “On-line implementation of nonlinear MPC: An experimental case study,” *Control Engineering Practice*, vol. 9, no. 8, pp. 847–857, 2001.

[44] A. P. Aguiar and J. P. Hespanha, “Trajectory-tracking and path-following of underactuated autonomous vehicles with parametric modeling uncertainty,” *IEEE transactions on automatic control*, vol. 52, no. 8, pp. 1362–1379, 2007.

[45] R. Hovorka *et al.*, “Nonlinear model predictive control of glucose concentration in subjects with type 1 diabetes,” *Physiological measurement*, vol. 25, no. 4, p. 905, 2004.

[46] T. Faulwasser, T. Weber, P. Zometa, and R. Findeisen, “Implementation of nonlinear model predictive path-following control for an industrial robot,” *IEEE Transactions on Control Systems Technology*, vol. 25, no. 4, pp. 1505–1511, 2016.

[47] J. Matschek, J. Bethge, P. Zometa, and R. Findeisen, “Force feedback and path following using predictive control: Concept and application to a lightweight robot,” *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 9827–9832, 2017.

[48] J. Matschek, T. Bäthge, T. Faulwasser, and R. Findeisen, “Nonlinear predictive control for trajectory tracking and path following: An introduction and perspective,” in *Handbook of Model Predictive Control*, Springer, 2019, pp. 169–198.

[49] T. A. Teatro, J. M. Eklund, and R. Milman, “Nonlinear model predictive control for omnidirectional robot motion planning and tracking with avoidance of moving obstacles,” *Canadian Journal of Electrical and Computer Engineering*, vol. 37, no. 3, pp. 151–156, 2014.

[50] M. M. G. Ardakani, B. Olofsson, A. Robertsson, and R. Johansson, “Model predictive control for real-time point-to-point trajectory generation,” *IEEE Transactions on Automation Science and Engineering*, vol. 16, no. 2, pp. 972–983, 2018.

[51] M. Neunert *et al.*, "Fast nonlinear model predictive control for unified trajectory optimization and tracking," in *2016 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2016, pp. 1398–1404.

[52] M. Mehndiratta, E. Kayacan, S. Patel, E. Kayacan, and G. Chowdhary, "Learning-based fast nonlinear model predictive control for custom-made 3D printed ground and aerial robots," in *Handbook of Model Predictive Control*, Springer, 2019, pp. 581–605.

[53] I. Carlucho, M. De Paula, and G. G. Acosta, "An adaptive deep reinforcement learning approach for MIMO PID control of mobile robots," *ISA transactions*, vol. 102, pp. 280–294, 2020.

[54] K. J. Åström, "Theory and applications of adaptive control—a survey," *automatica*, vol. 19, no. 5, pp. 471–486, 1983.

[55] K. Åström, *History of adaptive control.* 2015.

[56] R Bellman, "Adaptive control processes; a guided tour, princeton univ," *Press, NJ*, 1961.

[57] P. Gregory, "Proceedings of the self adaptive flight control systems symposium, january 13-14, 1959," Aeronautical Systems Div Wright-Patterson AFB OH Flight Control Lab, Tech. Rep., 1959.

[58] S. K. Panda, J. Lim, P. Dash, and K. Lock, "Gain-scheduled PI speed controller for PMSM drive," in *Proceedings of the IECON'97 23rd International Conference on Industrial Electronics, Control, and Instrumentation (Cat. No. 97CH36066)*, IEEE, vol. 2, 1997, pp. 925–930.

[59] H. P. Huang, M. L. Roan, and J. C. Jeng, "On-line adaptive tuning for PID controllers," *IEE Proceedings-Control Theory and Applications*, vol. 149, no. 1, pp. 60–67, 2002.

[60] F. Gao and H. Tong, "Differential evolution: An efficient method in optimal PID tuning and on–line tuning," in *Proceedings of the First International Conference on Complex Systems and Applications. Wuxi, China*, 2006.

[61] N. J. Killingsworth and M. Krstic, "PID tuning using extremum seeking: Online, model-free performance optimization," *IEEE control systems magazine*, vol. 26, no. 1, pp. 70–79, 2006.

[62] O. Gheibi, D. Weyns, and F. Quin, "Applying machine learning in self-adaptive systems: A systematic literature review," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 15, no. 3, pp. 1–37, 2021.

[63] R. Jafari and R. Dhaouadi, "Adaptive PID control of a nonlinear servomechanism using recurrent neural networks," *Adv. Reinforcement Learning*, pp. 275–296, 2011.

[64] I. Dumitrache and M. Dragoicea, "Mobile robots adaptive control using neural networks," *arXiv preprint arXiv:1512.03345*, 2015.

[65] F. G. Rossomando and C. M. Soria, "Identification and control of nonlinear dynamics of a mobile robot in discrete time using an adaptive technique based on neural PID," *Neural Computing and Applications*, vol. 26, no. 5, pp. 1179–1191, 2015.

[66] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT press, 2018.

[67] B. Hu *et al.*, "Reinforcement learning approach to design practical adaptive control for a small-scale intelligent vehicle," *Symmetry*, vol. 11, no. 9, p. 1139, 2019.

[68] C. J. C. H. Watkins, "Learning from delayed rewards," 1989.

[69] H. Boubertakh, M. Tadjine, P.-Y. Glorennec, and S. Labiod, "Tuning fuzzy PD and PI controllers using reinforcement learning," *ISA transactions*, vol. 49, no. 4, pp. 543–551, 2010.

[70] B. Subudhi and S. K. Pradhan, "Direct adaptive control of a flexible robot using reinforcement learning," in *2010 International Conference on Industrial Electronics, Control and Robotics*, IEEE, 2010, pp. 129–136.

[71] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE transactions on systems, man, and cybernetics*, no. 5, pp. 834–846, 1983.

[72] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[73] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International conference on machine learning*, PMLR, 2018, pp. 1587–1596.

[74] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*, PMLR, 2018, pp. 1861–1870.

[75] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, PMLR, 2016, pp. 1928–1937.

[76] Q. Sun, C. Du, Y. Duan, H. Ren, and H. Li, "Design and application of adaptive PID controller based on asynchronous advantage actor–critic learning method," *Wireless Networks*, vol. 27, no. 5, pp. 3537–3547, 2021.

[77] Y. Al Younes and M. Barczyk, "Nonlinear model predictive horizon for optimal trajectory generation," *Robotics*, vol. 10, no. 3, p. 90, 2021.

[78] Y. Al Younes and M. Barczyk, "A backstepping approach to nonlinear model predictive horizon for optimal trajectory planning," *Robotics*, vol. 11, no. 5, p. 87, 2022.

[79] Y. A. Younes and M. Barczyk, "Optimal motion planning in GPS-denied environments using nonlinear model predictive horizon," *Sensors*, vol. 21, no. 16, p. 5547, 2021.

[80] Y. Al Younes and M. Barczyk, "Adaptive nonlinear model predictive horizon using deep reinforcement learning for optimal trajectory planning," *Drones*, vol. 6, no. 11, p. 323, 2022.

[81] S. Yu, X. Li, H. Chen, and F. Allgöwer, "Nonlinear model predictive control for path following problems," *International Journal of Robust and Nonlinear Control*, vol. 25, no. 8, pp. 1168–1182, 2015.

[82] R. Findeisen, "Nonlinear model predictive control: A sampled data feedback perspective," Ph.D. dissertation, University of Stuttgart, 2005.

[83] B. Houska, H. Ferreau, and M. Diehl, "ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization," *Optimal Control Applications and Methods*, vol. 32, no. 3, pp. 298–312, 2011.

[84] M. Kamel, T. Stastny, K. Alexis, and R. Siegwart, "Model predictive control for trajectory tracking of unmanned aerial vehicles using robot operating system," in *Robot operating system (ROS)*, Springer, 2017, pp. 3–39.

[85] P. T. Boggs and J. W. Tolle, "Sequential quadratic programming," *Acta numerica*, vol. 4, no. 1, pp. 1–51, 1995.

[86] H. Ferreau, C. Kirches, A. Potschka, H. Bock, and M. Diehl, "qpOASES: A parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, 2014.

[87] R. Marino and P. Tomei, *Nonlinear Control Design: Geometric, Adaptive, and Robust.* Prentice Hall, 1995, ISBN: 9780133426359.

[88] F. Wu and C. Desoer, "Global inverse function theorem," *IEEE Transactions on Circuit Theory*, vol. 19, no. 2, pp. 199–201, 1972.

[89] H. Xie, "Dynamic visual servoing of rotary wing unmanned aerial vehicles," Ph.D. dissertation, University of Alberta, 2016.

[90] R. M. Murray, Z. Li, S. S. Sastry, and S. S. Sastry, *A mathematical introduction to robotic manipulation.* CRC press, 1994.

[91] C. Rösmann, A. Makarow, and T. Bertram, "Online motion planning based on nonlinear model predictive control with non-euclidean rotation groups," *arXiv preprint arXiv:2006.03534*, 2020.

[92] F. Sabatino, "Quadrotor control: Modeling, nonlinear control design, and simulation," M.S. thesis, KTH Royal Institute of Technology, 2015.

[93] A. Spitzer and N. Michael, "Feedback linearization for quadrotors with a learned acceleration error model," *arXiv preprint arXiv:2105.13527*, 2021.

[94] A. Mokheari, A. Benallegue, and B. Daachi, "Robust feedback linearization and GH-inf controller for a quadrotor unmanned aerial vehicle," in *Proceedings of the 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005, pp. 1198–1203.

[95] M. Quigley *et al.*, "ROS: An open-source robot operating system," in *ICRA workshop on open source software*, Kobe, Japan, 2009.

[96] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics: Results of the 11th International Conference*, ser. Springer Proceedings in Advanced Robotics, M. Hutter and R. Siegwart, Eds., vol. 5, Cham, Switzerland: Springer, 2018, pp. 621–635.

[97] L. Meier, D. Honegger, and M. Pollefeys, "PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms," in *2015 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2015, pp. 6235–6240.

[98] M. Krstic, P. V. Kokotovic, and I. Kanellakopoulos, *Nonlinear and adaptive control design*. John Wiley & Sons, Inc., 1995.

[99] T. Madani and A. Benallegue, "Control of a quadrotor mini-helicopter via full state backstepping technique," in *Proceedings of the 45th IEEE Conference on Decision and Control*, IEEE, 2006, pp. 1515–1520.

[100] Y. A. Younes, A. Drak, H. Noura, A. Rabhi, and A. E. Hajjaji, "Quadrotor position control using cascaded adaptive integral backstepping controllers," in *Applied Mechanics and Materials*, Trans Tech Publ, vol. 565, 2014, pp. 98–106.

[101] S. Bouabdallah and R. Siegwart, "Full control of a quadrotor," in *2007 IEEE/RSJ international conference on intelligent robots and systems*, IEEE, 2007, pp. 153–158.

[102] M. A. M. Basri, A. R. Husain, and K. A. Danapalasingam, "Enhanced backstepping controller design with application to autonomous quadrotor unmanned aerial vehicle," *Journal of Intelligent & Robotic Systems*, vol. 79, no. 2, pp. 295–321, 2015.

[103] F. Chen, R. Jiang, K. Zhang, B. Jiang, and G. Tao, "Robust backstepping sliding-mode control and observer-based fault estimation for a quadrotor UAV," *IEEE Transactions on Industrial Electronics*, vol. 63, no. 8, pp. 5044–5056, 2016.

[104] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, "Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2017, pp. 1366–1373.

[105] T. Dang, M. Tranzatto, S. Khattak, F. Mascarich, K. Alexis, and M. Hutter, "Graph-based subterranean exploration path planning using aerial and legged robots," *Journal of Field Robotics*, vol. 37, no. 8, pp. 1363–1388, 2020.

[106] M. Kulkarni *et al.*, "Autonomous teamed exploration of subterranean environments using legged and aerial robots," in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, 2022, pp. 3306–3313.

[107] F. Steinbrücker, J. Sturm, and D. Cremers, "Volumetric 3D mapping in real-time on a CPU," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 2021–2028.

[108] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 1996, pp. 303–312.

[109] H. Oleynikova, A. Millane, Z. Taylor, E. Galceran, J. Nieto, and R. Siegwart, "Signed distance fields: A natural representation for both mapping and planning," in *RSS 2016 Workshop: Geometry and Beyond-Representations, Physics, and Scene Understanding for Robotics*, 2016.

[110] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger, "Real-time 3D reconstruction at scale using voxel hashing," *ACM Transactions on Graphics*, vol. 32, no. 6, p. 169, 2013.

[111] S. Karaman and E. Frazzoli, "Sampling-based motion planning with deterministic μ-calculus specifications," in *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, 2009, pp. 2222–2229.

[112] A. W. Moore, "Efficient memory-based learning for robot control," University of Cambridge, Computer Laboratory, Tech. Rep., 1990.

[113] ArduPilot Development Team, *ArduPilot Autopilot Software Suite*. [Online]. Available: https://ardupilot.org/.

[114] R. Liu and J. Zou, "The effects of memory replay in reinforcement learning," in *2018 56th annual allerton conference on communication, control, and computing (Allerton)*, IEEE, 2018, pp. 478–485.

[115] J. Achiam, "Spinning Up in Deep Reinforcement Learning," 2018. [Online]. Available: https://github.com/openai/spinningu.

[116] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[117] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International conference on machine learning*, PMLR, 2016, pp. 1329–1338.

[118] T. Haarnoja *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.

[119] M. K. M. Meral, "Comparing model-free deep reinforcement learning algorithms on stock market," 2021.

[120] M. Abadi *et al.*, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: https://www.tensorflow.org/.

[121] F. Chollet *et al.*, *Keras*, 2015. [Online]. Available: https://keras.io.

[122] C. Lai, J. Han, and H. Dong, "Tensorlayer 3.0: A deep learning library compatible with multiple backends," in *2021 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*, IEEE, 2021, pp. 1–3.

[123] L. Meier, *MAVLink: Micro Air Vehicle Communication Protocol*, 2009. [Online]. Available: https://mavlink.io/en/.

[124] *QGroundControl GCS*. [Online]. Available: http://qgroundcontrol.com/.

[125] H. Wang, C. Wang, C. Chen, and L. Xie, “F-LOAM : Fast LiDAR odometry and mapping,” in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[126] B. Zhou, Y. Zhang, X. Chen, and S. Shen, “FUEL: Fast UAV exploration using incremental frontier structure and hierarchical planning,” *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 779–786, 2021.

[127] T. Lee, S. Mckeever, and J. Courtney, “Flying free: A research overview of deep learning in drone navigation autonomy,” *Drones*, vol. 5, no. 2, p. 52, 2021.

[128] D. Bauer, T. Patten, and M. Vincze, “Reagent: Point cloud registration using imitation and reinforcement learning,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 14 586–14 594.

[129] C. Zhao, Q. Sun, C. Zhang, Y. Tang, and F. Qian, “Monocular depth estimation based on deep learning: An overview,” *Science China Technological Sciences*, vol. 63, no. 9, pp. 1612–1627, 2020.