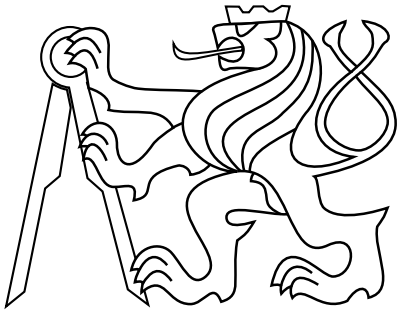
Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Cybernetics
Multi-robot Systems



Replanning of Collision-Free Trajectories for Unmanned Aerial Vehicle

Master's Thesis

Kryštof Teissing

Prague, May 2023

Study program: Cybernetics and Robotics

Supervisor: Ing. Robert Pěnička, Ph.D.

Acknowledgments

I would like to thank all the people who have helped me to complete this thesis. First and foremost, I would like to express my gratitude to my supervisor Ing. Robert Pěnička, Ph.D. for his guidance and valuable ideas, which made this work possible. I also owe thanks to my mother Mgr. Alžběta Soperová for proofreading this thesis, and the rest of my family and friends for their support along the way.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, May 26, 2023

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Teissing Kryštof** Personal ID number: **474544**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

II. Master’s thesis details

Master’s thesis title in English:

Replanning of Collision-Free Trajectories for Unmanned Aerial Vehicle

Master’s thesis title in Czech:

Přeplánování trajektorií bezkolizního letu bezpilotních vzdušných robotů

Guidelines:

1. Study the state-of-the-art path planning and trajectory generation methods for a collision-free time-optimal flight of unmanned aerial vehicles.
2. Propose a new method of finding the optimal velocity in given waypoints which is used for point-mass trajectory generation over multiple waypoints.
3. Design and implement an algorithm for computing a collision-free multi-waypoint trajectory for an unmanned aerial vehicle.
4. Compare the implemented methods with the state-of-the-art algorithms and evaluate the benefits of the proposed approach.

Bibliography / sources:

[1] Richter, Charles, Adam Bry, and Nicholas Roy. 2016. “Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments”. Robotics Research: 649-666.
[2] Foehn, Philipp, Angel Romero, and Davide Scaramuzza. “Time-optimal planning for quadrotor waypoint flight”. Science Robotics 6 (56), 2021.
[3] Penicka, Robert, and Davide Scaramuzza. 2022. “Minimum-Time Quadrotor Waypoint Flight in Cluttered Environments”. IEEE Robotics and Automation Letters 7 (2): 5719-5726.
[4] Romero, Angel, Robert Penicka, and Davide Scaramuzza. 2022. “Time-Optimal Online Replanning for Agile Quadrotor Flight”. IEEE Robotics and Automation Letters 7 (3): 7730-7737.

Name and workplace of master’s thesis supervisor:

Ing. Robert Pěnička, Ph.D. Multi-robot Systems FEE

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **27.01.2023** Deadline for master's thesis submission: **26.05.2023**

Assignment valid until: **22.09.2024**

Ing. Robert Pěnička, Ph.D.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Abstract

Trajectory generation is one of the fundamental problems solved under the motion planning task for autonomous Unmanned Aerial Vehicles (UAVs). This thesis deals with the replanning of minimum-time collision-free trajectories for a multirotor UAV, which is a task solved in time-critical applications such as search and rescue. We propose a gradient-method-based algorithm for computing and optimizing approximate trajectories using a limited acceleration point-mass model to facilitate real-time computation. Further, we use an iterative approach for trajectory acceleration distribution to account for gravity and better reflect the design constraints of a multirotor. An algorithm for collision-free trajectory generation is also introduced, where the latest developments in path planning are utilized to guide trajectory planning using paths with different topologies. We showcase that the proposed approach is capable of finding a minimum-time collision-free trajectory in sub-millisecond times.

Keywords Unmanned Aerial Vehicles, Motion Planning, Trajectory Planning

Abstrakt

Generování trajektorie je jedním z klíčových problémů řešených v rámci úlohy plánování pohybu autonomních bezpilotních prostředků (UAV). Tato práce se zabývá přeplánováním bezkolizních trajektorií s minimální dobou trvání pro multirotorové UAV, které najde využití v oblastech, kde hraje zásadní roli čas, jako je vyhledávání a záchrana osob. Představujeme algoritmus založený na gradientní metodě pro výpočet a optimalizaci aproximačních trajektorií za využití zjednodušeného modelu hmotného bodu s omezeným zrychlením, který umožňuje výpočet v reálném čase. Zároveň používáme iterativní metodu k rozložení zrychlení v rámci trajektorie, abychom lépe zohlednili gravitaci a konstrukční omezení multirotorového UAV. Dále je představen algoritmus pro výpočet bezkolizních trajektorií, ve kterém uplatňujeme nejnovější poznatky v oblasti plánování drah, které umožňují využít dráhy s různou topologií jako základ pro výpočet trajektorií. Ukazujeme, že navrhovaná metoda je schopna najít bezkolizní trajektorii s minimální dobou trvání v časech pod milisekundu.

Klíčová slova Bezpilotní prostředky, plánování pohybu, plánování trajektorií

Abbreviations

DOF Degrees of Freedom

UAV Unmanned Aerial Vehicle

VO Velocity Optimization

TD Thrust Decomposition

CFTG Collision-free Trajectory Generaiton

PRM Probabilistic Roadmap

RRT Rapidly-exploring Random Trees

CTopPRM Clustering Topological PRM

UDV Uniform Visibility Deformation

PMM Point-Mass Model

CR Cone Refocusing

GM Gradient Method

PTS Point-mass Trajectory Search

Contents

1 Introduction **1**

2 Preliminaries **3**
2.1 Path and trajectory . . . 3
2.2 Path planning . . . 4
2.3 Trajectory generation . . . 4
2.4 Gradient method for unconstrained minimization of differentiable functions . . 5

3 Related Work **7**
3.1 Path planning . . . 7
3.2 Trajectory generation . . . 9

4 Methodology **13**
4.1 Point-mass model trajectory . . . 14
4.2 Point-mass model trajectory - Axis synchronization . . . 16
4.3 Velocity optimization using Gradient method . . . 17
4.3.1 Velocity optimization of a one-dimensional trajectory . . . 18
4.3.2 Velocity optimization of a multi-dimensional trajectory . . . 20
4.4 Limited thrust decomposition . . . 27
4.5 Limited thrust decomposition in velocity optimization . . . 29
4.6 Path planning using Clustering Topological PRM . . . 31
4.7 Collision-free trajectory computation . . . 32

5 Results **37**
5.1 Velocity optimization algorithm . . . 37
5.1.1 Velocity optimization method parameter selection . . . 39
5.1.2 Thrust decomposition in velocity optimization . . . 41
5.2 Visualization of the velocity optimization convergence . . . 44
5.3 Comparison with state-of-the-art method . . . 45
5.4 Collision-free trajectory generation . . . 49

6 Conclusion **53**

References **55**

A Supplementary Definitions and Results **59**
A.1 Point-Mass Model Trajectory Acceleration Scaling Solutions . . . 59
A.2 Definition of Waypoints for the Testing Paths . . . 60
A.3 Velocity Optimization Parameter Grid-Search Results . . . 61

B Content of the Attached CD **63**

Chapter 1

Introduction

Unmanned Arial Vehicles (UAVs) have become increasingly popular for performing various tasks such as filmmaking [1], monitoring [2], or search and rescue [3] in the last years given their agility and maneuverability. This was further boosted by the latest development of autonomous capabilities, which reduce the demands on the pilot in command and enable the deployment of autonomous UAVs for complex tasks. Collision-free trajectory replanning is one of the crucial aspects of the autonomous flight problem, as it is necessary for successful navigation of the UAV in real-life environments with obstacles. Deployment of an autonomous UAV for time-critical operations such as search and rescue further requires minimum-time trajectories generated online so that the UAV can respond to abrupt changes in the environment, i.e., for collision avoidance or disturbance compensation.

In this thesis, we will deal with the problem of collision-free minimum-time trajectory replanning for agile multirotor UAVs. It is a complex task where a collision-free sequence of the multirotor states must be generated together with the respective time allocation, while simultaneously respecting the non-linear dynamics of the multirotor UAV. Due to the underactuated design of a multirotor, additional constraints are placed on the translational and rotational accelerations of a multirotor UAV. Moreover, for multi-waypoint trajectories, where the final motion must connect a predefined set of states (waypoints), the waypoints are usually defined only by the robot's pose, while the higher derivatives necessary for trajectory computation are missing. The underdetermined problem of trajectory planning must therefore be solved while respecting various criteria. In our case, these include the collision-free aspect of the final trajectory and minimizing the trajectory duration. Solving the trajectory planning in real-time poses an even greater challenge to the task due to the limited computational power of the UAV's onboard computers.

Several different approaches have been used to solve the collision-free trajectory generation task. For real-time applications, polynomials [4] or piecewise polynomials [5] have been used to efficiently compute collision-free trajectories; however, they do not lead to minimum-time trajectories due to the inherent smoothness of polynomials. Time-optimal trajectories were either found for a full dynamical model of a multirotor using computationally demanding methods [6] or computed with real-time sampling-based methods [7] that solved the task for an approximated point-mass model. The latter relied on recent advancements in control theory [8], which enable tracking of infeasible guiding trajectories in a near-time-optimal way. However, even the sampling-based methods were able to compute only trajectories with a limited number of waypoints in real time, which prevented their application in methods where several trajectories must be considered to account for obstacles.

This thesis brings the following contributions. We first define a closed-form solution for limited acceleration minimum-time multidimensional point-mass model trajectory planning, including a method for axis synchronization. Next, we extend this concept to a multi-waypoint trajectory and introduce a novel algorithm based on the Gradient method for velocity opti-

mization of the ambiguous velocities in the via-waypoints. Further, we present an iterative method for distributing accelerations to individual trajectory axes in order to account for gravity and the underactuated design of the multirotor UAV, where the acceleration norm is limited. Several approaches to its incorporation into the velocity optimization algorithm are also presented. We combine the above-mentioned methods to define a collision-free trajectory generation method that utilizes the benefits of topologically distinct paths returned by a state-of-the-art path planner [9], where we compute trajectories based on the fond paths to find the minimum-time collision-free trajectory. Subsequently, we showcase on a selection of scenarios that our approach to multi-waypoint trajectory generation results in small computational times outperforming the state-of-the-art sampling-based method. Additionally, we test the proposed collision-free trajectory generation in a forest-like environment and show that our method is capable of finding a minimum-time trajectory in sub-millisecond computational time, which makes it applicable in solving online replanning tasks.

Chapter 2

Preliminaries

We will first formally define the basic concepts used for motion planning in the field of robotics. In Section 2.1, we will define the fundamental terms used in motion planning, namely path and trajectory. Further, in Sections 2.2 and 2.3, we define the problems of collision-free path planning and path-guided trajectory planning, as we will be dealing with solving these tasks in this thesis with the goal of efficiently computing collision-free trajectories. We also aim to find minimum-time trajectories; therefore, the gradient descent method for iterative optimization of differentiable functions will be defined in Section 2.4. It will be used as the backbone for our trajectory optimization algorithm.

2.1 Path and trajectory

A spatial construct called path is used in the field of robotics to describe a motion of a robot geometrically in space [10]. The motion is defined by its start and end points, which are usually expressed using the robot's configurations $\mathbf{q} \in \mathcal{C}$, where $\mathcal{C}$ is the configuration space of the robot [11]. The number of parameters necessary to define a robot's configuration is called Degrees of Freedom (DOF). For a robot with n DOF, the configuration is usually defined as n-dimensional vector $\mathbf{q} \in \mathbb{R}^n$.

For a given start and end configurations $\mathbf{q}_{\text{start}}$ and $\mathbf{q}_{\text{end}}$, a path P can be described by the following continuous function

$$\begin{aligned} P\colon [0,1] &\longrightarrow \mathcal{C}, \\ s &\longmapsto \mathbf{q}(s), \\ s.t.\ \mathbf{q}(0) &= \mathbf{q}_{start}, \\ \mathbf{q}(1) &= \mathbf{q}_{end}, \end{aligned} \tag{2.1}$$

where a robot configuration $\mathbf{q}(s)$ is defined for every step $s \in [0,1]$ along the path.

A path with added time allocation is called a trajectory $\mathbf{\Pi}$ [10] and similarly to (2.1) can be described by a continuous function

$$\begin{aligned} \mathbf{\Pi}\colon [0, T_{tr}] &\longrightarrow \mathcal{C}, \\ t &\longmapsto \mathbf{q}(s(t)), \end{aligned} \tag{2.2}$$

where a pose is defined for every time instance $t \in [0, T_{tr}]$, T_{tr} is the trajectory duration.

Generally, an UAV in a three-dimensional space has $n = 6$ DOF, of which three define its position and three describe its rotation. For our application, we will use a simplified geometrical model of an UAV, namely a sphere with a given radius $r \in \mathbb{R}$. The DOFs of the simplified model are reduced to $n = 3$, where in a three-dimensional space, the configuration of an UAV is defined only by the position of its centroid. This will significantly simplify the path and trajectory planning problem-solving.

2.2 Path planning

Path planning is one of the problems solved under the motion planning task [11] for a robot. Given a space, otherwise referred to as the world $\mathcal{W} = \mathbb{R}^3$, in which the path planning is performed, the objective of the path planning problem is to find a collision-free path P for a robot $\mathcal{A} \subset \mathcal{W}$ spanning from the start configuration $\mathbf{q}_{\text{start}}$ to the goal configuration $\mathbf{q}_{\text{end}}$.

To define what a collision-free path is, we first need to define obstacles $\mathcal{O} \subseteq \mathcal{W}$ present inside $\mathcal{W}$. The geometry of a robot in a configuration $\mathbf{q}$ is denoted as $\mathcal{A}(\mathbf{q})$. The configuration space can then be divided into a collision space $\mathcal{C}_{obs} = \{\mathbf{q} \in \mathcal{C} | \mathcal{A}(\mathbf{q}) \cap \mathcal{O} \neq \emptyset\}$ and a free space $\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obs}$ [11]. A collision-free path is a path $P(s)$, where for all s the corresponding configurations $\mathbf{q}_s = P(s)$ lie inside $\mathcal{C}_{free}$, formally, $\forall s \in [0, 1], P(s) \subset \mathcal{C}_{free}$.

Path planning can be performed in various environments $\mathcal{W}$; in this thesis, we will focus on path planning in a known environment represented by a 3D grid map. Additional objectives can be placed on the path planning process so that the resulting path or paths are selected according to certain criteria. As we are interested in minimum-time trajectory generation, we are generally looking for the shortest possible paths in terms of L2 norm. However, the duration of the final trajectory can not be determined from the length of a path. Therefore, a method that returns a set of different paths from which we can then select the one that results in the fastest trajectory is suitable for our task.

As we have stated in the previous section, we are using a simplified model of an UAV in the form of a sphere with a radius r, whose configurations correspond to UAV's position in the world. The process of determining whether a configuration is collision-free, commonly known as collision checking, is then reduced to evaluating the distance $\delta(\cdot, \cdot)$ of the robot's configuration (position of the sphere's center) from all obstacles $\mathcal{O}$, i.e., using Euclidean Signed Distant Field [12]. The free space can then be defined as $\mathcal{C}_{free} = \{\mathbf{q} \in \mathcal{C} | \delta(\mathbf{q}, \mathcal{O}) > r\}$. This significantly reduces the computation time of the path-planning process.

2.3 Trajectory generation

The trajectory generation is a task in the classical motion planning pipeline that follows after the path planning. The general principle is to add timing to a path $P(s)$ generated by a path-planning algorithm to obtain a trajectory $\mathbf{\Pi}(t)$, also given the initial conditions in the form of start and end states, $\mathbf{x}_{start} \in \mathbb{R}^m$ and $\mathbf{x}_{end} \in \mathbb{R}^m$, respectively. A state usually contains not only the configuration $\mathbf{q}$ of the robot, but also its derivatives in time. The dimensionality of the start and end states depend on the method that determines the timing, commonly known as path time parametrization. According to the definition (2.2), path parametrization can be defined as a continuous function

$$\begin{aligned} s\colon [0, t_{end}] &\longrightarrow [0, 1], \\ t &\longmapsto s(t). \end{aligned} \tag{2.3}$$

However, the path time parameterization must respect the kinetic and dynamic limitations of the robot, also referred to as kinodynamic constraints [13]. Moreover, additional criteria can be placed on the resulting trajectory. In our field of interest, namely motion planning for agile UAVs, the emphasis is placed on the trajectories minimizing the flight time of an UAV, i.e. the time-optimal trajectories.

Due to the complexity of the trajectory generation for a full dynamic model of a robot, additional approximations can be used to obtain a simplified model. The trajectories are then computed for the simplified model with approximate translational and rotational dynamics of the robot, which allow faster planning. Although the resulting trajectories are consequently infeasible for the full dynamic model, modern control methods such as the Model Predictive Contouring Control [8] can track these trajectories in a near-time-optimal fashion. This significantly reduces the complexity of the trajectory generation problem and can be utilized in real-time computation methods.

Until now, we have defined the trajectory generation for a single trajectory segment, i.e., a segment between two states $\mathbf{x}_{start}$ and $\mathbf{x}_{end}$. However, in many applications, the path found by the path-planning algorithm is returned in the form of an ordered set containing n_w waypoints. Subsequently, a multi-segment trajectory has to be computed, where each segment spans between the corresponding two waypoints. To do that, we need fully defined states $\mathbf{x}_{start_i}$ and $\mathbf{x}_{end_i}$ for each of the $i = \{1, ..., n_w\}$ trajectory segments. However, a full state consisting of the robot's configuration $\mathbf{q}$ and several of its derivatives, is usually defined only for the start and end points of the whole multi-segment trajectory. For the remaining via-points, only the configurations $\mathbf{q}_i$ are defined in most cases. The ambiguity in the states corresponding to the via-points must be also addressed in the time-optimal trajectory generation task, which will be the main focus of this work.

2.4 Gradient method for unconstrained minimization of differentiable functions

The Gradient method (also referred to as the Gradient descent method [14]) is a numerical iterative method for finding local minima of differentiable functions on $\mathbb{R}^n$. Formally, given a differentiable function $f : \mathbb{R}^n \to \mathbb{R}$, the problem can be expressed as finding the minimum

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}), \tag{2.4}$$

where $\mathbf{x} \in \mathbb{R}^n$. The Gradient method solves the problem (2.4) iteratively updating an initial value $\mathbf{x}_0$ according to the following equation

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k), \tag{2.5}$$

where $\nabla f(\mathbf{x})$ is the gradient of the function f at the point $\mathbf{x}_k$ and $\alpha_k > 0$ is the step length at iteration k. Depending on the approach, the step length can either vary for every iteration or remain constant throughout the whole minimization process. The gradient of a function always points in the direction of the greatest growth, so when updating the current value $\mathbf{x}_k$ by adding the negative of the gradient, the vector is updated in the direction of steepest descent towards the minimum.

Different stopping criteria can be used to terminate the iterative minimization. Given that the iterative method computes successive approximations of the minimum, a threshold ε is usually used and, after each iteration, it is checked whether the update is below the threshold, i.e., after $|f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k)| < \varepsilon$ the optimization is terminated. Also, a maximum number of iterations $N \in \mathbb{Z}$ is used to terminate the optimization in a reasonable time should the convergence be too slow.

Chapter 3

Related Work

In this chapter, we will present the state-of-the-art methods used for the motion planning of UAVs. We will first describe different approaches to collision-free path planning in Section 3.1. Next, several methods for trajectory generation will be introduced in Section 3.2. We present the solutions to the two aforementioned motion planning problems separately as in most cases, the path-planning and trajectory-planning algorithms can be combined depending on the application.

3.1 Path planning

The path planning problem is a task in motion planning that precedes the trajectory generation task. For our main goal of trajectory replanning, we first require a collision-free path(s) in the environment where the UAV is moving, to guide the trajectory computation. As we have stated in previous sections, due to the complexity and high dimensionality of the path planning problem in a full configuration space of an UAV, the path planning is computed in an approximated space with reduced degrees-of-freedom (DOFs). In this work, the configuration space will be reduced to $\mathbb{R}^3$, where a configuration corresponds to the position of the center point of a UAV in a given coordinate system of the world/map.

In this thesis, we deal with path planning in a known environment to simplify the path planning task. The known environment can be represented by a 3D grid map, where the continuous space is discretized into 3D cells to reduce the computational burden of searching for a path in the whole configuration space. The grid map resolution determines the size of the discrete cells. For a known environment, if a grid cell lies inside $\mathcal{C}_{free}$, the corresponding configuration is collision-free and it can be used for path planning search. Otherwise, the cell is treated as an obstacle. A collision-free trajectory can then be found in the grid map using common graph searching algorithms such as Dijkstra's or A*, where each unoccupied grid cell represents a graph node. However, for larger maps or maps with high resolution, the exhaustive search of the whole grid map can be replaced with a sampling-based approach.

Commonly used sampling-based path planners, such as Rapidly-exploring Random Trees (RRT) [15], search for a path using a reduced graph structure with edges drawn from $\mathcal{C}_{free}$. In the case of RRT, a tree graph rooted at the start configuration is built towards the end configuration, where in every step the tree is expanded towards a random configuration sampled from $\mathcal{C}$. The new configuration is checked as to whether it lies in $\mathcal{C}_{free}$, if that is the case, it is connected to the closest tree node using a local planner. A local planner is an algorithm that computes a simple admissible path between two configurations. However, the RRT path planer is, suitable only for single query path planning, i.e., only for one pair of start and end configurations, because a new tree graph has to be built for every query.

For our application of online replanning, a method that allows multi-query planning is much more appropriate. One of the commonly used algorithms is the Probabilistic Roadmap (PRM) [16], which uses the concept of roadmap, i.e. a topological graph with configurations as its nodes. A constructed roadmap can be easily reused in a multi-query path planning task. Currently, there are several variants and modifications to the PRM, but the general approach consists of two phases. First, a pre-determined number of samples from $\mathcal{C}_{free}$ is sampled and connected into a roadmap. In the sampling process, a sample is drawn from $\mathcal{C}$, checked as to whether it lies in $\mathcal{C}_{free}$, and if the configuration is collision-free, it is connected with the graph nodes in its vicinity using a local planner. Only collision-free connections between the nodes are allowed. Then, for a given query, the start and goal nodes are connected to the created roadmap and the resulting graph is searched for a collision-free path using common graph-searching algorithms. Usually, the path with the shortest length is selected from all the possible ones.

In this work, we are focusing on generating trajectories for time-critical applications, where a minimum-time trajectory is desired. Since the shortest path does not generally translate to a trajectory with the minimum duration for dynamic models of an UAV, obtaining multiple alternative paths for subsequent trajectory generation is beneficial. Moreover, for time-efficient trajectory computation, we are interested in distinct paths with different homotopy classes. A path falls in a different topological class if it cannot be continuously deformed into another one without crossing any obstacles, for example by translation of the path points.

The authors of [5] have utilized this concept in their perception-aware trajectory replanning framework. Their approach to finding homotopically distinct paths uses a modified visibility-PRM [17] algorithm, where the visibility between two configurations is defined as the existence of a simple collision-free path (usually a line) between the two configurations planned by a local planner. In the sampling process of the PRM, a sample is either assigned a role, or it is discarded. A sample can have the role of a guard that guards all visible regions of the $\mathcal{C}_{free}$, where two guards can not be visible to each other. In addition, a sample can be also a connector if there are exactly two guards visible from it. A connector can be discarded if there is a new connector that connects two guards using a shorter path. The new samples are discarded if no role is assigned to them. The building of a roadmap then consists of sampling, assigning a role to the newly sampled configuration, and/or discarding unused configurations. A depth-first search is then used to find all distinct paths in the final roadmap. The described approach provides great results and fast computational times; however, its performance is dependent on the placement of the initial guards which can be challenging in environments with limited visibility.

The concept of homotopically distinct path planning for minimum-time trajectory generation has been utilized also in the work [18], where the authors use the following method for obtaining collision-free paths from different homotopy classes. An initial roadmap is computed using Informed-PRM [19], where the sampling is focused inside using an ellipsoid, which is iteratively relaxed until at least one collision-free path is found using Dijkstra's algorithm. Then, to capture as many homotopically distinct paths as possible, a configuration with the smallest clearance from an obstacle is found for all current paths and all other configurations of the roadmap in its vicinity are removed from the roadmap. The resulting roadmap is again searched for a collision-free path until no new paths can be found. To account for a case, where vertices in a narrow passage have been removed, the algorithm is recursively called from the removed region both towards the start and end configuration. The resulting paths are then connected to join the start and end configurations. The output of this approach is a set of all paths found during this process. As it was reported by [9], the lack of information about

which regions should be optimally removed results in an inability to find all paths in some scenarios.

A different approach to finding distinct paths using PRM was introduced in [9], where the sampled roadmap is clustered into a reduced roadmap so that homotopically distinct paths can be effectively obtained from it. Given that the authors claim to outperform both [5] and [18] in several aspects, we will further describe this method in Section 4.7 as it will be used for path planning in our trajectory generation pipeline.

3.2 Trajectory generation

Several approaches have been used in the field of research to compute a path-guided trajectory for an agile UAV. Specifically, for our application of trajectory replanning in time-critical scenarios, we are interested in real-time capable methods which result in minimum time trajectories. We focus on trajectory planning for multirotor UAVs, where the state-of-the-art methods mainly use continuous-time polynomial-based, sampling-based, and/or optimization-based approaches.

In real-time applications, continuous-time polynomials are often used because of their computational efficiency. Representing a trajectory of a multirotor UAV with a continuous polynomial is possible thanks to the differential flatness property of the multirotor, where all states and inputs can be expressed using flat outputs (in x, y, z and yaw axis) and their finite number of derivatives [20]. Polynomials are ideal for such flat systems as their derivatives are easily obtainable. In [4], the authors have used this approach to efficiently compute polynomial trajectories in real-time by solving a quadratic program, which can be solved in closed form. A subsequent gradient method optimization is used, where the polynomial trajectory is altered to minimize both snap and total trajectory duration. However, due to the inherent smoothness of polynomials, this representation can not render time-optimal policies with rapid state or input changes.

Polynomials are also used in [5] as part of a method of perception-aware online trajectory replanning. A concept of polynomial interpolation is used, where the final trajectory is defined by a piece-wise polynomial function called B-spline [21]. In addition, a sampling approach is used for the yaw angle planning, where a graph model of sampled yaw angles in all waypoints is created and searched for a trajectory that maximizes the environment coverage of the onboard sensors. Similar to the previous work, a numeric gradient-based optimization process follows, where the B-spline trajectory is deformed into more perception-aware trajectories. This, however, results in trajectories that are not time optimal due to the smoothness of the polynomials and different optimization criteria.

A method for time-optimal trajectory planning for a full dynamical model of a multirotor has been introduced in [6]. The authors address the coupling of the linear and rotational acceleration of a multirotor which is caused by the design of a multirotor UAV, where both linear and rotational acceleration is controlled using the thrusts of the UAV's actuators. This is an important attribute of a multirotor UAV, not always addressed by state-of-the-art trajectory planning methods. The method uses a complex optimization process, where time allocation and a full state are assigned for each of the trajectory waypoints using a discretized dynamic model of the multirotor. This, however, results in a computationally very demanding method where a single trajectory computation time ranges from several minutes to hours based on the number of waypoints. It is thus not suitable for real-time replanning.

Due to the non-linearity and high dimensionality of a full multirotor model [22], many methods use a point-mass model approximation, where the multirotor is assumed to have the reduced dynamics of a point mass. As we have stated in the previous sections, innovative control methods are able to follow trajectories generated using the simplified point-mass model. The approximate trajectory can also be used as a guiding trajectory for other methods respecting the full dynamics of a multirotor [23]. The algorithms for time-optimal trajectory planning generate the point-mass model trajectory using Pontryagin's maximum principle [24], which results in bang-bang policy [22] in the case of limited acceleration trajectories and in bang-singular-bang policy [25] in the case of a limited jerk trajectory.

The limited jerk point-mass model trajectory is a more accurate approximation as a real-life UAV cannot change its acceleration in an infinitesimally small time. It is used in [25] for an online time-optimal trajectory computation. However, many additional approximations are used in the work wherefore this method is not generally applicable. First, a movement in one axis at a time is assumed so that no axis synchronization has to be handled. Secondly, the authors assume a constant vertical thrust and full state knowledge in all waypoints, which results in 2D trajectories.

The method [26] is applicable in a more general case where the axis synchronization, i.e., synchronization of the single-axis motion primitives for all dimensions, is solved by stretching the zero acceleration segment in a bang-singular-bang policy trajectory. Nevertheless, the presented solution to the axis synchronization problem does not always have a feasible solution for a given synchronization time. Moreover, this approach still assumes known full states in all waypoints and does not address the problem of coupled accelerations of the multirotor. We will call this problem the limited thrust decomposition, where the acceleration limits have to be distributed in such a way, that the norm of the collective thrust does not exceed the maximal value the multirotor's actuators can produce. Omitting the limited thrust decomposition problem results in non-time-optimal trajectories because the limits on per-axis accelerations have to be conservative to ensure that the resulting thrust is within the limits at all times.

The authors of [27] propose an enhancement of the bang-singular-bang point-mass model trajectory generation, where additional synchronization patterns are introduced to solve the issue with axis synchronization present in the previous work. A limited acceleration scenario is assumed. The trajectory planning is solved under the kinematic orienteering problem, which also solves the ambiguity in the states of the via waypoints. Solving a kinematic orienteering problem is, however, not necessary in most time-critical applications and leads to increased computation times of several seconds which is unsuitable for real-time trajectory planning. Moreover, the solution to the problem is not guaranteed to converge.

In the work [22], the bang-bang policy point-mass model has been used to achieve a real-time capability of trajectory planning, i.e., a double-integrator motion primitive. The point-mass model trajectory is a part of an obstacle avoidance framework, where it is used as a local planner of a kinodynamic variant of the Fast Marching Tree [28] motion planning algorithm. Moreover, as it is used only to approximate a trajectory between two configurations, no axis synchronization approach is presented in the work. The point-mass model trajectories are used to guide the spline-based method [4] which then generates the final trajectories with optimized states at via waypoints. Consequently, the final trajectories suffer from the aforementioned drawbacks of polynomial trajectories and are generally not time-optimal.

The authors of [23] also use the bang-bang policy point-mass model trajectories with bounded accelerations to plan approximate time-optimal paths in a motion planning framework for drone racing. Specifically, the approximate trajectories are used as guiding trajec-

tories for online replanning, where the final trajectory is then recomputed using fourth-order polynomials for better tracking by the UAV's controller, similar to [22]. This time, the authors present a solution for the axis-synchronization problem, where the single-axis bang-bang policy trajectories are synchronized using acceleration scaling. The ambiguity in states of via waypoints is solved using a sampling approach, where at every waypoint a predefined number of states with different velocities are sampled. To reduce the computational time, the minimum-time problem is approximated as a shortest path problem, where the arc length between the samples is set to be equivalent to the shortest trajectory duration. The sampled states are then connected into a graph structure with the arc length as the cost of the edges; in the next step, the final states are found using a graph searching algorithm. To reduce the size of the search graph, a receding horizon approach is used, where the trajectory is generated only for the first few waypoints. A further approximation of this method is that no thrust decomposition task is solved when computing the point-mass model trajectories as they are used to guide the final polynomial trajectories.

The ability of modern control methods to track infeasible but differentiable paths was utilized in works [18] and [7], where a Model Predictive Contouring Control [8] method is used to track point-mass model bang-bang policy trajectories. Similar to [23], the ambiguity in velocities for multi-waypoint trajectories is solved using a sampling approach, where at every via waypoint, different velocity vectors are randomly sampled. The velocity samples differ in the vector direction and norm. They are then connected in a velocity search graph where the trajectory duration of the point-mass model connecting two sampled states (waypoint position and sampled velocity) is used as the cost of an edge. Moreover, to make the sampling more time efficient, the velocity samples are sampled from inside a cone spreading from the corresponding waypoint towards the next waypoint. Throughout the whole sampling process, the sampling cone is refocused based on the best sample currently available, i.e. the state which results in the shortest trajectory duration.

Further, in [18] the authors also use a gradient descent method on a sphere to optimize the thrust acceleration of a bang-bang policy trajectory segment to solve the limited thrust decomposition problem and obtain thus the minimum-time trajectories. The main drawback of the presented method is, that the computational complexity grows quadratically with the number of velocity samples per gate and thus also the computational time. For this reason, when the method was used for a real-time replanning in [7], a receding horizon approach was used, where the trajectory was generated only for several subsequent waypoints. In addition, the number of samples per gate has been limited to speed up the velocity search process.

Chapter 4

Methodology

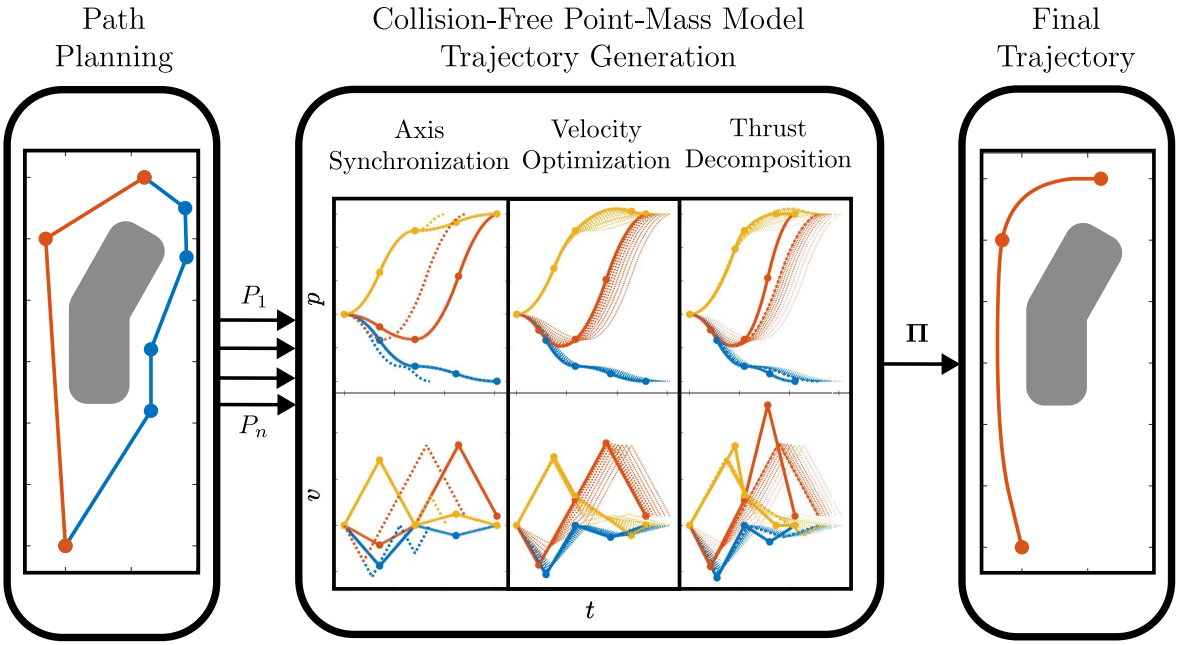


Figure 4.1: Block diagram of the proposed algorithm for collision-free trajectory planning. All possible distinct paths P_i, $i = 1, \ldots, n$, found by a path planning algorithm are processed by the Collision-free trajectory generation algorithm and a minimum-time point-mass model trajectory $\mathbf{\Pi}$ is returned.

In this chapter, we will describe the whole process of generating collision-free trajectories for UAV in more detail. Firstly, we will introduce the point-mass model motion primitive for one-dimensional trajectory in Section 4.1, extend it for a multi-dimensional trajectory in Section 4.2 and solve the axis-synchronization problem which is a part of the multi-dimensional trajectory generation. Secondly, in Section 4.3, we will introduce a novel method for velocity optimization of a multi-waypoint point-mass model trajectory. A method for limited thrust decomposition will be introduced in Section 4.4 and several approaches for its incorporation into the velocity optimization process will be defined in Section 4.5. Having these building blocks ready, we can utilize the path planning method described in Section 4.6 to define an algorithm for collision-free trajectory generation in Section 4.7. The whole process of collision-free trajectory planning is visualized in Figure 4.1.

4.1 Point-mass model trajectory

Firstly, we shall define a simple trajectory segment, i.e., a trajectory between two consecutive waypoints without any via points. A multi-waypoint trajectory consists of multiple connected simple trajectory segments, where the end state of one segment is also the start state of the consecutive segment, if any. Even though there are several works dealing with limited jerk point-mass trajectory computation for UAVs which use a bang-singular-bang policy and bounds on jerk, velocity, and acceleration, we will reduce the kinematics of the point-mass model to a bang-bang policy with bounds on acceleration. This will significantly reduce the size and number of analytic solutions to problems encountered when computing and optimizing a multi-waypoint trajectory. Also, proving the functionality of our concept for velocity optimization described in Section 4.3 will enable further generalization for more sophisticated point-mass models.

Given the reduced kinematics of a point-mass model, we shall define the start and end state of a given trajectory segment as positions $\mathbf{p}_0$, $\mathbf{p}_2 \in \mathbb{R}^n$ and velocities $\mathbf{v}_0$, $\mathbf{v}_2 \in \mathbb{R}^n$, respectively, where n is the number of dimensions of the space in which the trajectory is computed. In our case of a UAV flight, we are considering $n = 3$. The control input of the simplified point-mass model is the acceleration $\mathbf{a} \in \mathbb{R}^n$, which is bounded by the per-axis acceleration limits $\mathbf{a}_{min}$, $\mathbf{a}_{max} \in \mathbb{R}^n \setminus \{0\}$, where $\mathbf{a}_{min} \leq \mathbf{a} \leq \mathbf{a}_{max}$. We are also considering $\mathbf{a} \neq \mathbf{0}$ for a non-zero trajectory with $\mathbf{p}_0 \neq \mathbf{p}_2$.

As we will describe in the following sections, the per-axis acceleration limits for a multirotor UAV are bound together by the limited thrust force of the UAV. However, we will consider only per-axis acceleration limits first to make the trajectory computation as general as possible and deal with the bounded acceleration limits later.

First, we will study the kinematics of a single trajectory axis (dimension) and then generalize it for a multi-dimensional trajectory in the following section. Given the Pontryagin's maximum principle [24] applied on a point-mass model, a time-optimal trajectory segment can be described by the following equations:

$$\begin{aligned} p_1 &= p_0 + v_0 t_1 + \frac{1}{2} a_1 t_1^2, \\ v_1 &= v_0 + a_1 t_1, \\ p_2 &= p_1 + v_1 t_2 + \frac{1}{2} a_2 t_2^2, \\ v_2 &= v_1 + a_2 t_2, \end{aligned} \tag{4.1}$$

where $p_0, v_0 \in \mathbb{R}$ are the start position and velocity, $p_2, v_2 \in \mathbb{R}$ are the end position and velocity, $t_1, t_2 \in \mathbb{R}$ are the time durations of the corresponding sub-segments of the bang-bang policy trajectory and $a_1, a_2 \in \mathbb{R}$ are the optimal control inputs such that

$$a_i \in \{a_{\min}, a_{\max}\},\ i = 1, 2. \tag{4.2}$$

We also omit the case where $a_1 = a_2$ as this is equivalent to either $t_1 = 0$ or $t_2 = 0$ which leaves us with only two combinations of possible values for accelerations a_1 and a_2.

Given the start state, end state, and limits on the acceleration, the remaining unknowns are the sub-segment durations t_1, t_2, velocity v_1 and position p_1 at time $t = t_1$. It can be shown

that analytical solutions to the equations (4.1) exist, which are as follows:

$$\begin{aligned} t_1 &= -\frac{v_0 \pm \sigma}{a_1}, \\ t_2 &= \frac{v_2 \pm \sigma}{a_2}, \\ p_1 &= \frac{-{v_0}^2 + {v_2}^2 + 2\,a_1\,p_0 - 2\,a_2\,p_2}{2\,(a_1 - a_2)}, \\ v_1 &= \pm\sigma, \\ \sigma &= \sqrt{-\frac{a_2\,{v_0}^2 - a_1\,{v_2}^2 - 2\,a_1\,a_2\,p_0 + 2\,a_1\,a_2\,p_2}{a_1 - a_2}}. \end{aligned} \tag{4.3}$$

There are four solutions in total; for each combination of a_1 and a_2 values there are two solutions as shown in equations (4.3). From those solutions, the final solution is chosen such that all unknowns are real numbers, and the real-world condition of non-negative time is satisfied, i.e., $t_1 > 0$ and $t_2 > 0$. In addition, the total trajectory duration of the selected solution

$$T_{\mathrm{tr}} = t_1 + t_2 \tag{4.4}$$

is the smallest possible.

From the analysis of the domain of the solutions (4.1), it can be derived that a real solution for given initial conditions exists if and only if

$$\begin{cases} (a_1 - a_2)(a_1\,{v_2}^2 - a_2\,{v_2}^2 + 2\,a_1\,a_2\,p_0 - 2\,a_1\,a_2\,p_2) \geq 0 & \text{for } v_1 = v_2, \\ (a_1 - a_2)(a_2\,{v_0}^2 - a_1\,{v_2}^2 - 2\,a_1\,a_2\,p_0 + 2\,a_1\,a_2\,p_2) \leq 0 & \text{for } v_1 \neq v_2, \end{cases} \tag{4.5}$$

where both a_1 and a_2 are non-zero. This follows from (4.2) and the definition of acceleration bounds. If conditions (4.5) are not satisfied, there is no feasible solution for a point-mass trajectory; these conditions can be checked prior to the computation of all the solutions to save computation time.

For further use in Section 4.3, we also define partial derivatives of total trajectory time (4.4) with respect to the start velocity v_0 and end velocity v_2. These will be needed to optimize segment endpoint velocities for multi-waypoint trajectories. Using general differentiation formulas, the following equations for the corresponding partial derivatives can be obtained:

$$\begin{aligned} \frac{\partial T_{\mathrm{tr}}}{\partial v_0} &= \pm\left(\frac{\frac{a_2\,v_0}{\sigma} \mp 1}{a_1} - \frac{v_0}{\sigma}\right), \\ \frac{\partial T_{\mathrm{tr}}}{\partial v_2} &= \pm\left(\frac{\frac{a_1\,v_2}{\sigma} \pm 1}{a_2} - \frac{v_2}{\sigma}\right), \\ \sigma &= (a_1 - a_2)\,\sqrt{-\frac{a_2\,{v_0}^2 - a_1\,{v_2}^2 - 2\,a_1\,a_2\,p_0 + 2\,a_1\,a_2\,p_2}{a_1 - a_2}}. \end{aligned} \tag{4.6}$$

Similarly to (4.5), after evaluating the domain of the partial derivatives, equations (4.6) have a real solution under the following conditions:

$$\begin{cases} (a_1 - a_2)(a_1\,{v_2}^2 - a_2\,{v_2}^2 + 2\,a_1\,a_2\,p_0 - 2\,a_1\,a_2\,p_2) > 0 & \text{for } v_1 = v_2, \\ (a_1 - a_2)(a_2\,{v_0}^2 - a_1\,{v_2}^2 - 2\,a_1\,a_2\,p_0 + 2\,a_1\,a_2\,p_2) < 0 & \text{for } v_1 \neq v_2. \end{cases} \tag{4.7}$$

4.2 Point-mass model trajectory - Axis synchronization

When applying the trajectory computation method described in Section 4.1 to a multi-dimensional trajectory, one will face the issue of axis synchronization. Using the equations (4.3) we are able to compute a minimum-time trajectory for each of the trajectory dimensions, also called axes. However, the duration of all the trajectory axes will generally not be equal; a method for recomputing the corresponding axis trajectories to an equal duration is therefore needed.

For this purpose, approaches from literature such as additional synchronization patterns described in [27] or stretching of a constant velocity profile used in [26] are not applicable. These methods were designed for a bang-singular-bang policy point-mass model trajectory, not a bang-bang policy used in our work. For a bang-bang policy approach, we will build on the work [7], where the authors use acceleration scaling for extending trajectory duration to a given time.

We will showcase this approach on a one-segment multi-dimensional trajectory which has the whole start and end states defined, velocities included. Using the equations (4.3), we obtain segment durations $T_{\mathrm{ax}_1}, \ldots, T_{\mathrm{ax}_n}$ for each of the n axes. From these durations, we will select the longest one as the trajectory duration T_{sync} of the synchronized multi-dimensional trajectory. Now, for all remaining trajectory axes, we recompute the trajectory to the selected duration T_{sync} using acceleration scaling.

To that end, we introduce an acceleration scaling factor $\gamma \in (0, 1]$ together with an additional constraint on the total trajectory duration into the point-mass trajectory equations (4.1); this results in the following equations for the synchronization trajectory:

$$\begin{aligned} p_1 &= p_0 + v_0 t_1 + \frac{1}{2}\gamma\, a_1 t_1^2, \\ v_1 &= v_0 + \gamma\, a_1 t_1, \\ p_2 &= p_1 + v_1 t_2 + \frac{1}{2}\gamma\, a_2 t_2^2, \\ v_2 &= v_1 + \gamma\, a_2 t_2, \\ T_{\mathrm{sync}} &= t_1 + t_2, \end{aligned} \tag{4.8}$$

where $a_1, a_2 \in \{a_{\min}, a_{\max}\}$ and $a_1 \neq a_2$, similarly to (4.1). We have thus two possible combinations of values for a_1 and a_2 in total.

We end up with five equations in five unknowns t_1, t_2, p_1, v_1, and γ. As in the previous case, the solutions to these equations can be expressed in closed form, but we have to distinguish between two cases; one where $v_0 \neq v_2$ and one where $v_0 = v_2$; otherwise, there would be a division by zero in the solution equations. Solutions for both cases are listed in Appendix A.1 due to their large size.

There are generally four solutions to the problem (4.8), two solutions for each combination of a_1 and a_2, as listed in Appendix A.1. From them, we choose the solution for which all unknowns are real numbers, the real-world condition of non-negative time, i.e., $t_1 > 0$ and $t_2 > 0$, is satisfied, and the scale γ lies in the interval $(0, 1]$. The reason is that we require the accelerations to be non-zero and within the acceleration limits, and since we have used the maximal accelerations possible for the initial minim-time single-axis trajectory, the scaling factor must lie within the interval $(0, 1]$. Even though we are trying to scale down the acceleration to increase the trajectory time of an axis with smaller trajectory durations, a case may

occur where an even higher acceleration would be necessary to integrate the position to the given value in the given time T_{sync}. This case is not discussed in [7]; we will therefore propose an additional step to axis synchronization that solves this issue.

If no feasible solution for γ has been found for the given trajectory duration T_{sync}, we must find the smallest possible trajectory duration $T_{\text{sync}_{\text{new}}}$ the axis is able to synchronize. Again, this can be done by computing a minimum-time trajectory for the given segment using the approach described in Section 4.1, but this time enforcing values on a_1 and a_2, where

$$\begin{aligned} a_{1_{\text{new}}} &= a_{2_{\text{old}}}, \\ a_{2_{\text{new}}} &= a_{1_{\text{old}}}. \end{aligned} \tag{4.9}$$

The reason is that for the old values $a_{1_{\text{old}}}$ and $a_{2_{\text{old}}}$ of the initial minimum-time trajectory, there is no feasible scale γ to scale them to the given trajectory duration T_{sync} greater than the minimum-time trajectory duration we are trying to synchronize. Therefore, the next smallest trajectory duration greater then T_{sync} is the duration $T_{\text{sync}_{\text{new}}}$ of a minimum-time trajectory (4.1) with swapped acceleration values $a_{1_{\text{new}}}$ and $a_{2_{\text{new}}}$ as defined in (4.9).

Given the new trajectory duration $T_{\text{sync}_{\text{new}}} > T_{\text{sync}_{\text{old}}}$ for synchronization, all other trajectory axes must be recomputed using (4.8), where $T_{\text{sync}} := T_{\text{sync}_{\text{new}}}$. Should some of those axes not be able to synchronize to the newly selected duration, the above-described process is repeated. This step repeats until all axes are able to synchronize to a common trajectory duration. If one of the single-axis trajectories is a zero trajectory, i.e., a trajectory with $p_0 = p_2$ and $v_0 = v_2$, we select $\gamma = 0$ and set the trajectory sub-segment durations as $t_1 = 0$ and $t_2 = T_{tr}$. We refer to the above-described method for a one-segment multi-dimensional point-mass model trajectory with synchronized axis computation as **computeTrajectory3D** in Algorithm 1, Algorithm 2.

4.3 Velocity optimization using Gradient method

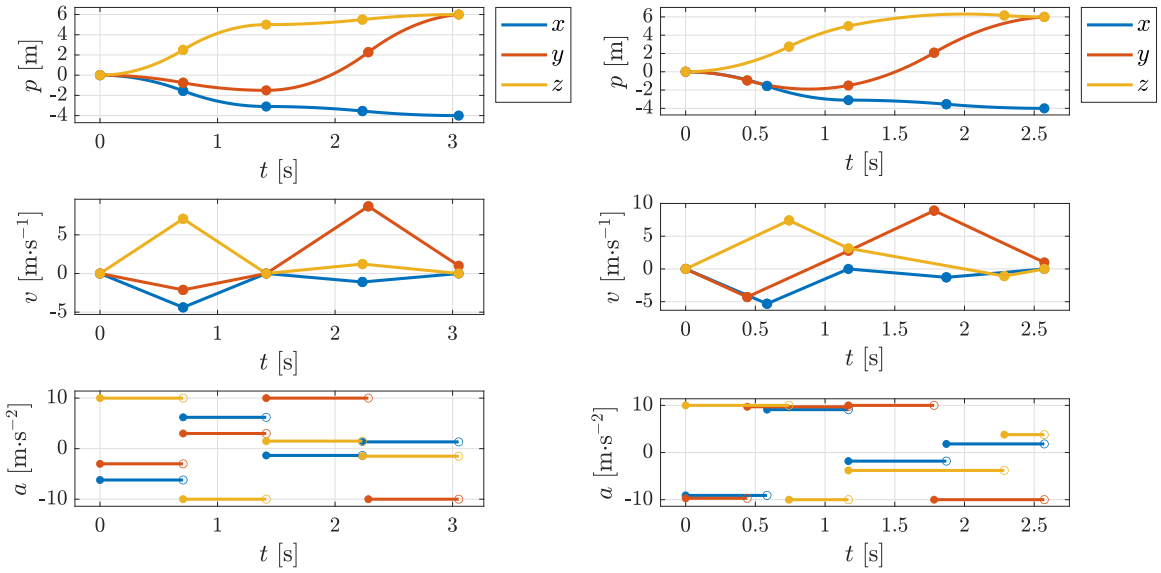
Throughout the previous section, we assumed that all starting and ending states of all trajectory segments of a multi-dimensional trajectory are known. i.e., the positions and velocities are known for all waypoints. This, however, is not generally the case. Usually, we have complete knowledge only about the desired start and end states, and the computed collision-free path we want to follow is defined only by the positions of the path points.

A possible solution to this problem is to define zero velocity in all the via-waypoints, which would result in minor deviations of the trajectory from the original path, but also in slow movement. The UAV would have to speed up and decelerate between every waypoint. Such a trajectory computed using the methods described in the previous sections can be seen in Figure 4.2a.

In this section, we propose a new method for finding velocities in the via-waypoints so that the trajectory is smoother and the trajectory duration is minimized. It is an iterative method based on an optimization algorithm for finding a local minimum of a differentiable non-linear function called Gradient method [14]. The Gradient method has been used for optimizing polynomial trajectories in previous works such as [4].

The problem of finding optimal velocities $\mathbf{v}_i \in \mathbb{R}^n$ for all via-waypoints of a point-mass model trajectory that minimize the trajectory duration $T_{\text{tr}} : \mathbb{R}^n \to \mathbb{R}$ can be formally written as

$$\{\mathbf{v}_1^*, \ldots, \mathbf{v}_n^*\} = \arg \min_{\mathbf{v}_1,\ldots,\mathbf{v}_n \in \mathbb{R}^n} T_{\text{tr}}(\mathbf{v}_1, \ldots, \mathbf{v}_n), \tag{4.10}$$



(a) Original trajectory.

(b) Optimized trajectory.

Figure 4.2: Visualization of a point-mass model bang-bang policy three-dimensional trajectory with synchronized axes before velocity optimization (a) and after velocity optimization (b). Displayed are positions p, velocities v, and accelerations a for all three axes.

where n is the number of via-waypoints of the given path from which the trajectory is computed. The problem (4.10) is also subjected to all the constraints on a feasible point-mass model trajectory defined in Sections 4.1 and 4.2.

4.3.1 Velocity optimization of a one-dimensional trajectory

We will first showcase the application of the gradient method on the velocity optimization using a one-dimensional trajectory since velocity optimization of a multi-dimensional trajectory is more complicated and it will be explained later. In addition, we explain the method on a two-segment trajectory because the optimization is generally done by consecutively applying the same steps on two consecutive segments; a two-segment trajectory optimization process can thus be easily expanded for a multi-segment trajectory.

The velocity profile of such a trajectory can be seen in Figure 4.3. Here, the notation from Section 4.1 is adjusted to fit the two-segment trajectory, but the equations (4.1) are applicable for both segments, i.e., for second segment $v_0 := v_{12}$ and so forth. The two-segment trajectory duration is equal to

$$T_{\text{tr}} = T_1 + T_2 = t_{11} + t_{12} + t_{21} + t_{22}. \tag{4.11}$$

In all following sections, when we refer to an update of a velocity in a two-segment scope or two-segment trajectory, we will always mean the velocity v_{12} which corresponds to the waypoint that joins the two trajectory segments and which is generally not defined prior to computing the trajectory. We also call such a velocity the unknown velocity.

For our example trajectory, the problem (4.10) is reduced to

$$v_{12}^* = \arg\min_{v_{12}\in\mathbb{R}} T_{\text{tr}}(v_{12}) = \arg\min_{v_{12}\in\mathbb{R}} T_1(v_{12}) + T_2(v_{12}), \tag{4.12}$$

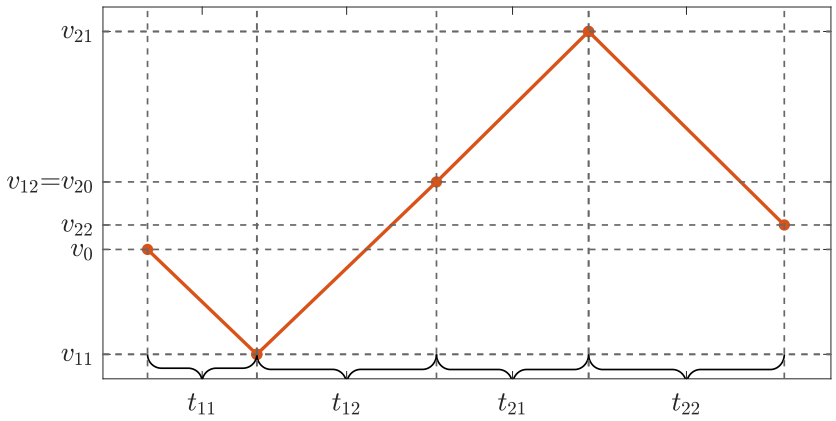


Figure 4.3: Velocity profile of an example two-segment one-dimensional trajectory.

where the objective function $T_{\text{tr}}(v_{12})$ is defined as the sum of its two segment trajectory durations $T_1(v_{12})$ and $T_2(v_{12})$ as defined in equation (4.4).

Following the notation of a trajectory segment (4.1), the velocity v_{12} is equal to v_2 for the first segment and v_0 for the second segment. Then it is straightforward to show that using equations (4.6), the gradient $\nabla T_{\text{tr}}(v_{12})$ is equal to

$$\nabla T_{\text{tr}}(v_{12}) = \frac{\partial T_{\text{tr}}}{\partial v_{12}} = \frac{\partial T_1}{\partial v_{12}} + \frac{\partial T_2}{\partial v_{12}}. \tag{4.13}$$

Having a differentiable objective function $T_{\text{tr}}(v_{12})$ and its first derivative, we can apply the gradient method on the problem (4.12). The update step (2.5) of the gradient method can then be written as

$$v_{12_{k+1}} = v_{12_k} - \alpha \nabla T_{\text{tr}}(v_{12_k}). \tag{4.14}$$

Velocity optimization of a two-segment trajectory is then computed by iteratively updating an initial estimate of the unknown velocity v_{12_0} using the update step (4.14) until convergence of the objective function. After every velocity update, both the trajectory segments are recomputed using (4.3) so that the terminating conditions of the gradient method can be checked as described in Section 2.4. We will discuss the choice of the initial estimate v_{12_0} in the following text.

To extend this concept to a multi-segment trajectory, we apply a single update step (4.14) to all two consecutive segments in every iteration of the optimization process. This means that a single iteration of multi-segment trajectory consists in calculating and evaluating a single update step and for all the unknown velocities one by one in a two-segment scope; by evaluating, we mean recomputation of the two corresponding segments according to the new velocity value. Again, the unknown velocity is the first segment's end velocity and the second segment's start velocity in the two-segment window.

We also change the order of the two-segment scope velocity updates in every iteration of the optimization process. It has been observed that the velocity optimization converges faster if the unknown velocities are updated in the opposite order in every iteration. First, we calculate and evaluate the unknown velocities starting from the beginning of the trajectory, moving one by one toward the end. In the next iteration, we update the two-segment sub-trajectories starting from the end and moving towards the beginning of the trajectory.

In other words, we have a two-segment sliding window, and in every iteration of the optimization process, we go through the whole trajectory from one end to the other. In every step, we calculate a velocity update of the unknown velocity in the current scope, recompute the two neighboring segments, and slide the window one step to the side according to the current direction of updates. For a three-segment trajectory, this approach will result in two velocity updates for a single iteration.

After every iteration of the optimization process, updates of each of the unknown velocities are checked, and if all updates lie below a threshold ε, the optimization will terminate with the best possible velocities found, i.e., when the condition

$$|T_{\mathrm{tr}}(v_{12_{k+1}}) - T_{\mathrm{tr}}(v_{12_k})| < \varepsilon \tag{4.15}$$

is satisfied.

Also, to apply the gradient method to the velocity optimization, an initial value for all the unknown velocities must be selected. Setting all initial velocities to zero has been selected as the best option as this guarantees the trajectory's feasibility before the optimization process, provided that all other initial parameters allow it. Additionally, setting all initial values to the same value ensures that for a multi-segment trajectory, all velocity values are iteratively updated according to how much their change will reduce the total trajectory duration. This is given by the corresponding gradients.

4.3.2 Velocity optimization of a multi-dimensional trajectory

Extending the concept described in Section 4.3.1 to multi-dimensional trajectories is not straightforward. There are two main reasons, the first one being that the single-axis trajectories of a multi-dimensional trajectory contain not only minimum time segments (4.1) but also synchronization segments (4.8). The duration of the synchronization segments is given and selected according to the slowest axis, as described in Section 4.2. The second reason is that any adjustment to a velocity in one axis and the subsequent change in the axis trajectory duration directly affects all other axes. For example, a case can occur, where an axis trajectory segment to which all other axes are synchronized has a shorter duration due to the velocity update. Subsequently, all other axes now have a different time to which they must be synchronized.

We will introduce the following notation to distinguish all cases that can arise during the velocity optimization. A point-mass bang-bang policy minimum time trajectory segment (4.1) will be referred to as a MIN segment. For a synchronization point-mass bang-bang policy trajectory segment (4.8), we will use the designation SYNC segment. We will also distinguish different roles of a single-axis segment in the multi-axis trajectory, where a role means whether it is a MIN segment to which other axes are synchronized or a SYNC segment that needs to be synchronized.

In a two-segment scope, the following combinations of the previously defined segments can arise for a synchronized single-axis trajectory of a multi-axis trajectory: MIN-MIN, SYNC-SYNC, MIN-SYNC and SYNC-MIN.

In the MIN-MIN case, the trajectory optimization of the velocity for the given axis is done the same way we described in Section 4.3.1 using the partial derivative of the corresponding MIN segment with respect to its start or end velocity.

The SYNC-SYNC case does not require any update to the velocities as both segments of the given axis are adjusted according to some other axis, so updating the velocity would not influence the total trajectory time at all.

The MIN-SYNC and SYNC-MIN cases are treated similarly, but additional calculations are needed to ensure correct velocity updates. The difference between the approach used in the case MIN-MIN is that the SYNC segment has a constant duration, which results using the equations (4.6) and (4.11) in the following expression for the gradient of the two-segment trajectory

$$\begin{aligned}\nabla T_{\text{MIN-SYNC}}(v_{12}) &= \frac{\partial T_{\text{MIN-SYNC}}}{\partial v_{12}} = \frac{\partial T_{\text{MIN}}}{\partial v_{12}} + \underbrace{\frac{\partial T_{\text{SYNC}}}{\partial v_{12}}}_{0} = \frac{\partial T_{\text{MIN}}}{\partial v_{12}},\\ \nabla T_{\text{SYNC-MIN}}(v_{12}) &= \frac{\partial T_{\text{SYNC-MIN}}}{\partial v_{12}} = \underbrace{\frac{\partial T_{\text{SYNC}}}{\partial v_{12}}}_{0} + \frac{\partial T_{\text{MIN}}}{\partial v_{12}} = \frac{\partial T_{\text{MIN}}}{\partial v_{12}},\end{aligned} \tag{4.16}$$

where v_{12} is the velocity of the boundary waypoint between the two segments as depicted in Figure 4.3.

This alone is insufficient for the velocity update as the SYNC segment places additional bounds on the final updated velocity. A solution to the SYNC segment equations (4.8) must still exist that ensures the final acceleration scale γ is within the defined interval $(0, 1]$. We are thus searching for the solution of the equations (4.8) for $\gamma = 1$ and velocities v_0 or v_2 become new additional unknowns for SYNC-MIN and MIN-SYNC cases, respectively. By solving this system of equations, we get the solutions for our boundary velocities

$$\begin{aligned}v_{12_{\text{B1}}}^{\text{S-M}} &= v_0 + T_{\text{sync}}\, a_{11} \pm \sqrt{(a_{11} - a_{12})\,\left(a_{11}\, T_{\text{sync}}^2 + 2\, v_0\, T_{\text{sync}} + 2\, p_0 - 2\, p_{12}\right)},\\ v_{12_{\text{B1}}}^{\text{M-S}} &= v_{22} - T_{\text{sync}}\, a_{22} \pm \sqrt{(a_{21} - a_{22})\,\left(-a_{22}\, T_{\text{sync}}^2 + 2\, v_{22}\, T_{\text{sync}} + 2\, p_{12} - 2\, p_{22}\right)},\end{aligned} \tag{4.17}$$

for the SYNC-MIN and MIN-SYNC cases, respectively. Again, as described in Section 4.2, only real number values of the boundary velocities corresponding to non-negative real values of the trajectory sub-segments durations t_1 and t_2 are considered and evaluated for both combinations of the sub-segment accelerations a_1 and a_2 values. Consequently, four possible values for the velocity bounds exist for each of the MIN-SYNC and SYNC-MIN cases. From those, only the valid and closest to the initial value of the boundary velocity v_{12_k} in terms of Euclidean distance are taken as the lower and upper bound, respectively, to which the velocity update is limited.

Velocity bounds for the y-axis trajectory displayed in Figure 4.2a for both combinations of a_{11} and a_{12} are shown in Figure 4.4. We will refer to the described method for the computation of velocity bounds given by the SYNC segment as **getSyncVelocityBounds** in Algorithm 1.

Bounds on the velocity update given by the SYNC segment are not the only limitations that must be accounted for in the velocity optimization of a multi-dimensional trajectory. In Section 4.1, we have described the process of computing and selecting feasible solution(s) of a point-mass model trajectory. One of the requirements on the computed trajectory sub-segment durations t_1 and t_2 was that they are non-negative. This condition is given by the physical sense of the trajectory generation problem as we consider time to be strictly increasing. However, this requirement is not encoded into the equations (4.1); we only select the correct solution

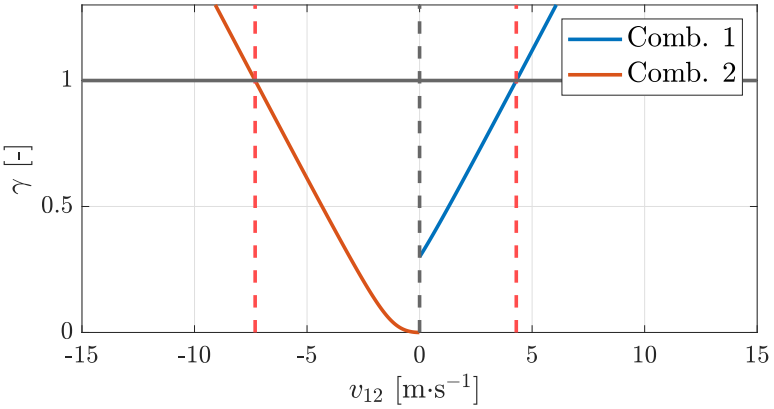


Figure 4.4: Visualization of bounds (red vertical lines) on the optimized velocity v_{12} given the limited interval of the acceleration scale $\gamma \in (0, 1]$ and two possible combinations of sub-segment acceleration values a_{11} and a_{22}. The dependence of γ on the possible velocity values v_{12} is shown for the y-axis SYNC-MIN trajectory displayed in Fig. 4.2a.

out of all possible solutions for the given initial conditions, if any. As we update the boundary velocity of the two-segment trajectory, we also change the initial conditions of both the MIN and SYNC segments. We have already described how we deal with the bounds given by the SYNC segment equation solutions, but we also need to guarantee that the velocity update will not cause infeasibility of the updated MIN segment because of negative sub-segment durations.

For this purpose, we have chosen a modified function for computation of the trajectory time of the MIN segment, namely the sum of absolute values of the sub-segment durations

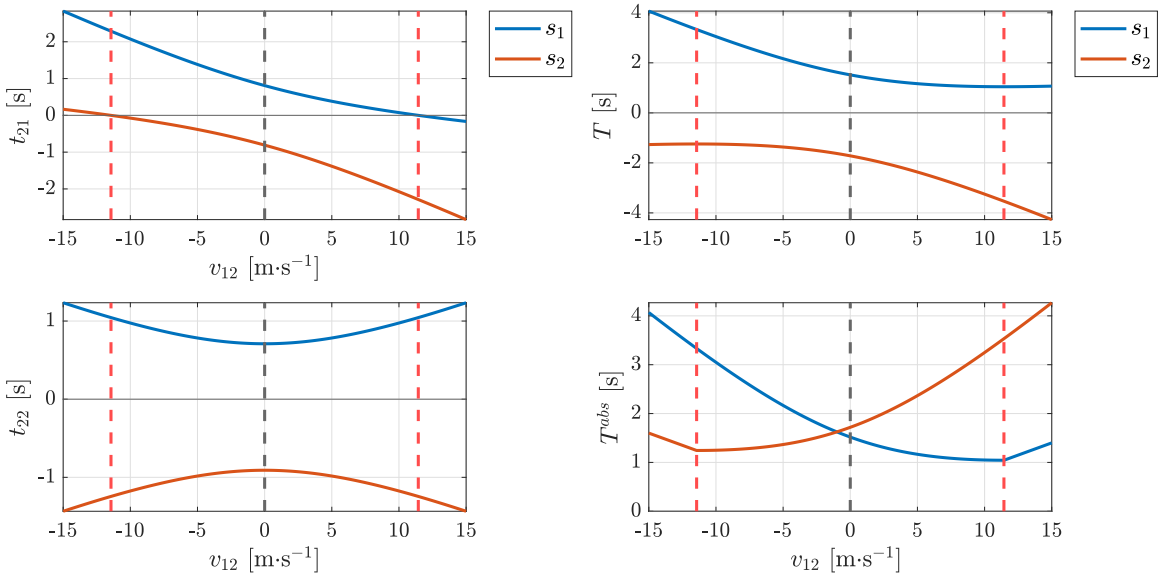
$$T_{tr}^{abs} = |t_1| + |t_2|. \tag{4.18}$$

The motivation behind the equation (4.18) is that a real absolute value function $f(x) = |x|$ is continuous and differentiable everywhere except for value $x = 0$, $x \in \mathbb{R}$. And this is precisely the value where x changes its sign. Subsequently, by finding the non-differentiable points of the function (4.18), we can also obtain the boundary values for velocity v_{12} for the velocity update of a SYNC-MIN and MIN-SYNC optimization step, respectively.

This can be seen in Figure 4.5, where the discontinuities in the MIN trajectory function (4.18), marked by red vertical lines in Figure 4.5b correspond to the change in sign of the sub-segment duration t_{21} in Figure 4.5a. The trajectory duration T of the MIN segment alone is insufficient for finding the bounds. This can be observed when comparing T and the modified trajectory duration function T^{abs} in Figure 4.5b. It should also be noted that due to the reasons explained in Section 4.1, only the bound corresponding to the discontinuity in T^{abs} for feasible solution s_1 is taken into account.

Now, we will describe the exact method for finding the velocity bounds of a MIN-SYNC trajectory. The process is analogous to the SYNC-MIN case and for both cases, we will refer to it as **getMinVelocityBounds** in Algorithm 1.

By substituting the resulting solutions of a one-segment point-mass model trajectory (4.3) into equation (4.11) while using equation (4.18) for the MIN segment duration, we obtain the following expression for the SYNC-MIN two-segment trajectory duration with respect to



(a) Solutions of trajectory sub-segment durations. (b) Comparison of trajectory duration functions.

Figure 4.5: Visualization of bounds (red vertical lines) on the optimized velocity v_{12} given by the feasibility condition of non-negative time. Displayed are the in (a) the four solutions s_1, s_2 of the sub-segment durations t_{21}, t_{22} of a MIN segment of a SYNC-MIN trajectory given different values of the velocity v_{12}. In Figure (b), trajectory durations T and T^{abs} of the MIN segment are displayed to showcase the connection between the discontinuities (red vertical lines) in T^{abs} and change in sign of the sub-segment duration t_{21}. The displayed values correspond to the y-axis trajectory displayed in Fig. 4.2a.

the optimized velocity v_{12}:

$$\begin{aligned} T_{\text{MIN-SYNC}}(v_{12}) &= T^{abs}_{\text{MIN}}(v_{12}) + T_{\text{SYNC}} = |t_{11}(v_{12})| + |t_{12}(v_{12})| + T_{\text{SYNC}} \\ &= \frac{|v_0 \pm \sigma(v_{12})|}{|a_{11}|} + \frac{|v_{12} \pm \sigma(v_{12})|}{|a_{12}|} + T_{\text{SYNC}}, \\ \sigma(v_{12}) &= \sqrt{-\frac{a_{12}\, v_0{}^2 - a_{11}\, v_{12}{}^2 - 2\, a_{11}\, a_{12}\, p_0 + 2\, a_{11}\, a_{12}\, p_{12}}{a_{11} - a_{12}}}. \end{aligned} \tag{4.19}$$

By calculating the partial derivative of the function (4.19) with respect to the velocity v_{12} and analyzing its domain, or simply by evaluating the absolute value terms in equation (4.19) and setting them equal to zero, we can obtain all the values of v_{12} where a sign of a sub-segment duration t_{11} or t_{12} changes. To do that, we solve the equations

$$\begin{aligned} 0 &= |v_0 \pm \sigma(v_{12})|\,, \\ 0 &= |v_{12} \pm \sigma(v_{12})|\,, \\ 0 &= \sigma(v_{12}), \end{aligned} \tag{4.20}$$

where the first two equations and $\sigma(v_{12})$ follow from the equation (4.19), and the last equation follows from the requirements on the existence of a point-mass trajectory (4.7).

The resulting bounds on the updated velocity $v_{12_{k+1}}$ given by the physical requirement on positive time are for the MIN-SYNC case are

$$\begin{aligned} v_{12_{\mathrm{B2}}}^{\text{M-S}} &= \pm\sqrt{{v_0}^2 - 2\,a_{12}\,p_0 + 2\,a_{12}\,p_{12}}, \\ v_{12_{\mathrm{B2}}}^{\text{M-S}} &= \pm\sqrt{{v_0}^2 - 2\,a_{11}\,p_0 + 2\,a_{11}\,p_{12}}, \\ v_{12_{\mathrm{B2}}}^{\text{M-S}} &= \pm\frac{\sqrt{a_{12}}\,\sqrt{{v_0}^2 - 2\,a_{11}\,p_0 + 2\,a_{11}\,p_{12}}}{\sqrt{a_{11}}}, \end{aligned} \tag{4.21}$$

each solution corresponding to equations (4.20). Due to the domain of the square root function, not all of the six velocity bounds $v_{12_{\mathrm{B2}}}$ must exist; therefore we choose only the ones that are real numbers. Similarly to velocity bounds given by the SYNC segment (4.17), we then sort the bounds and, according to their Euclidean distance to the current value v_{12_k}, we select the lower and upper bound of the velocity update $v_{12_{k+1}}$. As mentioned above, the bounds $v_{12_{\mathrm{B2}}}^{\text{S-M}}$ for the SYNC-MIN case are computed analogously. In conclusion, for the MIN-SYNC and SYNC-MIN segments, the velocity update is computed only from the gradient given by the MIN segment, and there are two sets of bounds, within which the resulting velocity update must lie.

Until now, we have described our approach to velocity optimization of a single-axis trajectory in a two-segment scope. Expanding this approach to a multi-dimensional case means that for every two-segment window of a multi-waypoint trajectory, we must adjust the velocities for every axis separately. In other words, a single iteration of the optimization process consists in updating the multi-dimensional trajectory in a two-segment scope, starting from one of the trajectory endpoints, and moving the two-segment window one by one towards the other endpoint. In every update of a two-segment sub-trajectory, we go through each axis individually and according to the roles of the two single-axis trajectory segments, we compute the velocity update according to the method described in this section. The updates are also adjusted so that they comply with the corresponding bounds. After all updates of the single-axis trajectories in the two-segment window have been computed, we recompute the whole two-segment trajectory using the approach described in Section 4.2. Next, we slide the two-segment window one step to the side according to the current direction of updates to get a new two-segment sub-trajectory with one of the segments identical to the previous window position. As described in Section 4.3.1, we alter the endpoint of the trajectory at which the velocity update process starts in every iteration for faster convergence.

However, the per-axis optimization of velocities does bring certain complications with it. With this approach, we *a priori* assume that after the per-axis velocity update, the roles of the segments in the two-segment scope remain the same for each axis. But this is not the case because, due to the change in the unknown velocities, the role of the segments change, e.g., the MIN-SYNC case for one axis can change to MIN-MIN case after the velocity update. This could lead to oscillations in the velocity updates as the updates are calculated using different approaches in each of the cases. A situation could occur where in every iteration, the case would switch back and forth. This would cause issues with the algorithm termination as there would still be velocity updates. It could also happen that the case switch would increase the two-segment trajectory duration, which is something we try to avoid.

To prevent this case, we check, after each update of the two-segment window, whether the velocity update did not increase the two-segment trajectory duration. If that is the case, we first look at the new roles of the two segments in each axis, where we check whether the trajectory duration of the new MIN segments has not increased. The reason is that for a one-segment trajectory, there is always some combination of MIN and SYNC roles for all axis,

and as the SYNC segments are synchronized to the MIN segment(s), the duration increase could be caused by one of the MIN segments. If so, we reduce the gradient method update step size α_i for the corresponding axis i by a reduction factor $\eta \in (0, 1)$ as

$$\alpha_i = \eta\, \alpha_i \tag{4.22}$$

and recompute the two-segment trajectory. This is done repeatedly until either the two-segment duration does not increase or the update step α_i is reduced to a value below a given threshold $\zeta \in \mathbb{R}$. In the second case, the update is terminated for the given axis and two-segment trajectory, i.e., $\alpha_i = 0$, and the two-segment trajectory is recomputed again. After each window shift, the vector of update steps for all axis $\boldsymbol{\alpha}_k$ are reinitialized to the initial value α_{init} as

$$\boldsymbol{\alpha} = [\alpha_{\text{init}}, \ldots, \alpha_{\text{init}}]^{\text{T}}. \tag{4.23}$$

The cause of the two-segment trajectory duration increase can also lie in the SYNC trajectory as there is not a continuous interval of trajectory times the SYNC segment could synchronize, as described in Section 4.2. For this reason, we keep track of which velocities were updated, and if the trajectory duration increase was not caused by the MIN segment, the step size α_{i_k} is reduced for all updated axis the same way analogously to the previous case. This whole process of searching for axes in which the velocity update caused the trajectory duration increase and subsequent step scaling is called **findAllAxesThatIncreasedTrajectoryDuration** in Algorithm 1.

A detailed description of the velocity optimization process is shown in Algorithm 1, and an example trajectory computed using this approach is depicted in Figure 4.2b.

In conclusion, all hyper-parameters used for the optimization process for our algorithm based on the gradient method, we have defined the following parameters:

1. the initial update step $\alpha_{\text{init}} \in \mathbb{R}$ equal for all trajectory axis;
2. the update step reduction factor $\eta \in (0, 1)$ which is used to reduce the currently used update step α_i corresponding to the i-th trajectory axis;
3. the threshold $\zeta \in \mathbb{R}$ for the update step, which determines when the update for the given axis and segment should be terminated;
4. the threshold $\varepsilon \in \mathbb{R}$ for terminating the iterative optimization if the difference between the current trajectory duration and the trajectory duration in the previous iteration is too small; and
5. the maximum number of iterations $N \in \mathbb{Z}$ after which the velocity optimization is terminated.

The impact of different values for these parameters on the convergence and computation time of the velocity optimization will be discussed in the following Chapter 5.

Algorithm 1: Point-Mass Model Trajectory Generation with Velocity Optimization

Input: $\mathbf{P}[i]$, $i = 0, \ldots, n$ - path points, $\mathbf{v}_0$ - start velocity, $\mathbf{v}_n$ - end velocity, $\mathbf{a}_{min}$ - minimal per-axis accelerations, $\mathbf{a}_{max}$ - maximal per-axis accelerations, α - initial step length, η - step length reduction factor, ζ - step length threshold, ε - time change threshold, N - maximum number of iterations

Output: $\mathbf{\Pi}$ - Optimized trajectory

$\mathbf{V}_0 \leftarrow [\mathbf{v}_0, \mathbf{0}, \ldots, \mathbf{0}, \mathbf{v}_n]$ ▷ Initial velocities
$\mathbf{\Pi}_0 \leftarrow$ Compute trajectories for all segments using **computeTrajectory3D** and $\mathbf{V}_0$
for $k \in \{1, \ldots, N\}$ **do**
  $\mathbf{V}_k \leftarrow \mathbf{V}_{k-1}$
  Switch $order \leftarrow \{1, \ldots, n-2\}$ and $order \leftarrow \{n-2, \ldots, 1\}$ every iteration k
  for $seg \in order$ **do**
    $\boldsymbol{\alpha} \leftarrow [\alpha, \alpha, \alpha]^{\mathrm{T}}$ ▷ Reinitialize for all axes $i \in \{0, 1, 2\}$
    for $i \in \{0, 1, 2\}$ **do**
      $p_0 \leftarrow \mathbf{P}[seg-1][i],\ p_{12} \leftarrow \mathbf{P}[seg][i],\ p_{22} \leftarrow \mathbf{P}[seg+1][i]$
      $v_0 \leftarrow \mathbf{V}_{k-1}[seg-1][i],\ v_{12} \leftarrow \mathbf{V}_{k-1}[seg][i],\ v_{22} \leftarrow \mathbf{V}_{k-1}[seg+1][i]$
      $\Pi_1^i \leftarrow \mathbf{\Pi}_k[seg-1][i],\ \Pi_2^i \leftarrow \mathbf{\Pi}_k[seg][i]$
      if Π_1^i *is SYNC and* Π_2^i *is SYNC* **then**
        continue
      else if Π_1^i *is MIN and* Π_2^i *is MIN* **then**
        $v_{12_k} \leftarrow v_{12} - \boldsymbol{\alpha}[i](\nabla T_1(v_{12}) + \nabla T_2(v_{12}))$
      else if Π_1^i *is MIN and* Π_2^i *is SYNC* **then**
        $\mathbf{v}_{12_{B1}} \leftarrow$ **getSyncVelocityBounds**$(T_2, p_{12}, p_{22}, v_{22}, \mathbf{a}_{min}[i], \mathbf{a}_{max}[i])$
        $\mathbf{v}_{12_{B2}} \leftarrow$ **getMinVelocityBounds**$(p_0, v_0, p_{12}, v_{12}, \mathbf{a}_{min}[i], \mathbf{a}_{max}[i])$
        $v_{12_k} \leftarrow v_{12} - \boldsymbol{\alpha}[i]\nabla T_1(v_{12})$
        Clip v_{12_k} given bounds $\mathbf{v}_{12_{B1}}$ and $\mathbf{v}_{12_{B2}}$
      else if Π_1^i *is SYNC and* Π_2^i *is MIN* **then**
        $\mathbf{v}_{12_{B1}} \leftarrow$ **getSyncVelocityBounds**$(T_1, p_0, v_0, p_{12}, \mathbf{a}_{min}[i], \mathbf{a}_{max}[i])$
        $\mathbf{v}_{12_{B2}} \leftarrow$ **getMinVelocityBounds**$(p_{12}, v_{12}, p_{22}, v_{22}, \mathbf{a}_{min}[i], \mathbf{a}_{max}[i])$
        $v_{12_k} \leftarrow v_{12} - \boldsymbol{\alpha}[i]\nabla T_2(v_{12})$
        Clip v_{12_k} given bounds $\mathbf{v}_{12_{B1}}$ and $\mathbf{v}_{12_{B2}}$
      $\mathbf{V}_k[seg][i] \leftarrow v_{12_k}$
    Update Π_1 and Π_2 using **computeTrajectory3D** given $\mathbf{V}_k[seg]$
    if $(T_{1_k} + T_{2_k}) > (T_{1_{k-1}} + T_{2_{k-1}})$ **then**
      $j \leftarrow$ **findAllAxesThatIncreasedTrajectoryDuration**()
      $\boldsymbol{\alpha}[j] \leftarrow \eta\boldsymbol{\alpha}[j] \quad \forall j$
      if $\exists j$ *s.t.* $\boldsymbol{\alpha}[j] < \zeta$ **then**
        $\boldsymbol{\alpha}[j] \leftarrow 0$
      Update Π_1 and Π_2 using **computeTrajectory3D** given $\mathbf{V}_{k-1}[seg]$
      $seg \leftarrow seg - 1$
    else
      $\mathbf{\Pi}_k[seg-1][i] \leftarrow \Pi_1$
      $\mathbf{\Pi}_k[seg][i] \leftarrow \Pi_2$
  if $\left|T_{traj_k} - T_{traj_{k-1}}\right| < \varepsilon$ **then**
    break

4.4 Limited thrust decomposition

In the previous section, we introduced a trajectory computing method that requires per-axis acceleration limits. However, as described in Section 3.2, for a fixed-frame multirotor UAV flight in a three-dimensional space, the three-axis accelerations are coupled with the limited thrust vector of the multirotor.

Given a maximal thrust force $f_{\max}$ the motors of a multirotor can produce, the limited thrust acceleration limit $a_{\max}^T$ is computed for a multirotor of mass m as

$$a_{\max}^T = \frac{f_{\max}}{m}. \tag{4.24}$$

Then, including also the gravity, the body acceleration values a_i^B, $i \in \{x, y, z\}$ of a trajectory must for all times $t \in [0, T_{\text{tr}}]$ satisfy the following constraint

$$a_{\max}^T \geq \sqrt{(a_x^B(t))^2 + (a_y^B(t))^2 + (a_z^B(t) + g)^2}, \tag{4.25}$$

where T_{tr} is the trajectory duration, and g is the gravitational acceleration at the surface of the Earth. This constraint follows from the definition of the multirotor thrust acceleration

$$\mathbf{a}^T = \mathbf{a}^B - \mathbf{g}, \tag{4.26}$$

where $\mathbf{a}^B$ is the multirotor body acceleration, i.e., acceleration of its point-mass model, and $\mathbf{g} = [0, 0, -g]$ is the gravity vector.

For a multi-dimensional point-mass model trajectory, distributing the limited thrust acceleration into the different axis accelerations in every trajectory segment is not an easy task due to the non-linear constraint (4.25). Rather than deriving a complex optimization approach to find these accelerations and to minimize the computation time, we will build on the work [18] and introduce an algorithm that iteratively approximates an optimal distribution of body accelerations

$$\mathbf{a}^B(t) = [a_x^B(t), a_y^B(t), a_z^B(t)]^{\mathrm{T}},\ t \in [0, T_{tr}] \tag{4.27}$$

so that the trajectory duration of a point-mass model bang-bang policy trajectory segment T_{tr} is minimized.

For a single three-dimensional trajectory segment with per-axis trajectory equations (4.1), the problem can be formulated using (4.25) and (4.26) as

$$\begin{aligned}
\{\mathbf{a}_1^{B*}, \mathbf{a}_2^{B*}\} &= \arg \min_{\mathbf{a}_1^B, \mathbf{a}_2^B \in \mathbb{R}^3} T_{\text{tr}}(\mathbf{a}_1^B, \mathbf{a}_2^B) \\
\text{s.t. } \mathbf{a}_1^B &= [a_{1_x}^B, a_{1_y}^B, a_{1_z}^B]^{\mathrm{T}}, \\
\mathbf{a}_2^B &= [a_{2_x}^B, a_{2_y}^B, a_{2_z}^B]^{\mathrm{T}}, \\
a_i^B(t) &= a_{1_i}^B \text{ for } t \in [0, t_{1_i}], \text{ for } i \in \{x, y, z\} \\
a_i^B(t) &= a_{2_i}^B \text{ for } t \in [t_{1_i}, T_{tr}], \text{ for } i \in \{x, y, z\} \\
\|\mathbf{a}^B(t) - \mathbf{g}\| &\leq a_{\max}^T, \text{ for } t \in [0, T_{tr}]
\end{aligned} \tag{4.28}$$

where $a_{1_i}^B$ is the body acceleration for the i-axis in the first trajectory sub-segment, $a_{2_i}^B$ the body acceleration in the second sub-segment, and t_{1_i} is the switch time of the bang-bang policy trajectory corresponding to the i-axis.

To approximate the solution to the problem (4.28), we first decompose the maximal thrust acceleration $a^T_{\max}$ into per-axis limits $\mathbf{a}_{\min}$ and $\mathbf{a}_{\max}$ with equal limits across all axes. This is done by solving the following equation

$$a^T_{\max} = \sqrt{\left(a^B_{\text{init}}\right)^2 + \left(a^B_{\text{init}}\right)^2 + \left(a^B_{\text{init}} + g\right)^2} \tag{4.29}$$

for the initial per-axis thrust acceleration limit estimate a^B_{init}. The problem (4.29) has a closed form solution

$$a^B_{\text{init}} = -\frac{g}{3} \pm \frac{\sqrt{3\ \left(a^T_{\max}\right)^2 - 2\,g^2}}{3}. \tag{4.30}$$

We always select the real-number non-negative solution from the two possible values. The initial body per-axis acceleration limits are then computed as

$$\begin{aligned} \mathbf{a}^B_{\max_0} &= [a^B_{\text{init}}, a^B_{\text{init}}, a^B_{\text{init}}]^{\mathrm{T}}, \\ \mathbf{a}^B_{\min_0} &= -\mathbf{a}_{\max_0} + 2\mathbf{g}, \end{aligned} \tag{4.31}$$

where the body acceleration limits are symmetrical for x and y axes and asymmetrical for the z axis due to the gravity compensation. The second equation in (4.31) can be derived from the following analysis of the z component of the thrust acceleration (4.29):

$$\begin{aligned} a^T_z &= a^B_{\text{init}} + g, \\ -a^B_z &= -a^T_z - g = -a^B_{\text{init}} - 2g, \end{aligned} \tag{4.32}$$

where we use (4.26) for transforming the thrust acceleration into body acceleration.

Using these initial values, we compute the multi-dimensional trajectory segment with synchronized axes using the method described in Section 4.2. As the synchronization trajectories are computed using acceleration down-scaling, the norm of acceleration will generally be different from $a^T_{\max}$ throughout the trajectory segment. We want to use the maximal acceleration possible according to the Pontryagin's maximum principle [24] for the minimum time trajectory. This means in our case that there is a thrust acceleration vector $\mathbf{a}^T$ used in the trajectory such that

$$\|\mathbf{a}^T\| = a^T_{\max}. \tag{4.33}$$

Using the bang-bang policy and axis synchronization, we generally have different per-axis switch times $t_{1_i},\ i \in \{x, y, z\}$ which lead to four different body acceleration vectors $\mathbf{a}^B_l,\ l \in 1, \ldots, 4$. Each of the accelerations $\mathbf{a}^B_l$ is used in the corresponding time intervals between the times t_0, t_{1_x}, t_{1_y}, t_{1_z}, and T_{tr}. However, as we are using symmetrical body acceleration limits in x and y axis, there are only two possible values for the thrust acceleration norm of the given trajectory, namely

$$\|\mathbf{a}^T_l\| \in \left\{\|\mathbf{a}^B_1 - \mathbf{g}\|, \|\mathbf{a}^B_2 - \mathbf{g}\|\right\}, \tag{4.34}$$

where $\mathbf{a}^B_1 := \mathbf{a}^B_{l=1}$ is the start acceleration and $\mathbf{a}^B_2 := \mathbf{a}^B_{l=4}$ is the end acceleration.

To find the body per-axis acceleration limits $\mathbf{a}^B_{\min}$ and $\mathbf{a}^B_{\max}$ which result in satisfying the condition (4.33) for at least one thrust acceleration $\mathbf{a}^T_l$, we propose the following iterative approach.

First, in every iteration j, we select the largest thrust acceleration used in terms of norm

$$\mathbf{a}^T_j = \arg\max(\|\mathbf{a}^B_{1_j} - \mathbf{g}\|, \|\mathbf{a}^B_{2_j} - \mathbf{g}\|). \tag{4.35}$$

In the next step, new per-axis acceleration limits $\mathbf{a}^B_{\min_{j+1}}$ and $\mathbf{a}^B_{\max_{j+1}}$ are approximated by finding a factor $\beta \in \mathbb{R}$ with which the current maximal acceleration is scaled to achieve (4.33) as follows

$$\begin{aligned} \mathbf{a}^B_{\max_{j+1}} &= \beta \mathbf{a}^T_j \\ \mathbf{a}^B_{\min_{j+1}} &= -\beta \mathbf{a}^T_j + 2\mathbf{g}. \end{aligned} \tag{4.36}$$

The factor β is computed by solving the equation

$$a^T_{\max} = \sqrt{\left(\beta a^T_{x_j}\right)^2 + \left(\beta a^T_{y_j}\right)^2 + \left(\beta a^T_{z_j} + g\right)^2}, \tag{4.37}$$

where $\mathbf{a}^T_j = [a^T_{x_j}, a^T_{y_j}, a^T_{z_j}]$, which has a closed form solution

$$\beta = -\frac{a^T_{z_j} g \pm \sqrt{\left(a^T_{\max}\right)^2 \|\mathbf{a}^T_j\|^2 - \left(a^T_{x_j}\right)^2 g^2 - \left(a^T_{y_j}\right)^2 g^2}}{\|\mathbf{a}^T_j\|^2}. \tag{4.38}$$

Again, we are choosing a real-number and positive value out of the two possible solutions in order to correctly assign maximal and minimal acceleration values.

Multiplying all elements of the vector $\mathbf{a}^T_j$ by the same factor does not guarantee that after recomputing the trajectory with the new per-axis acceleration bounds $\mathbf{a}^B_{\min_{j+1}}$ and $\mathbf{a}^B_{\max_{j+1}}$ the norm of the maximal sub-segment acceleration $\mathbf{a}^T_{j+1}$ will equal to $a^T_{\max}$; yet as we will show in the following chapter, after few iterations of the above-described algorithm we can approximate the acceleration distribution so that

$$\left| \|\mathbf{a}^T_j\| - a^T_{\max} \right| < \varepsilon \tag{4.39}$$

holds for some predefined threshold $\varepsilon \in \mathbb{R}$. This is also the stopping criterion of the thrust decomposition method for point-mass model trajectory generation. The whole algorithm is described in Algorithm 2.

This approach for limited thrust acceleration decomposition for one trajectory segment can be extended to a multi-waypoint trajectory by running the algorithm for every trajectory segment separately. As it is an iterative method, we use again a maximum number of iterations $N \in \mathbb{Z}$ to terminate the algorithm in case of slow convergence.

4.5 Limited thrust decomposition in velocity optimization

Implementing the algorithm for thrust decomposition introduced in Section 4.4 into the velocity optimization approach explained in Section 4.3.2 is challenging. In the velocity optimization process, we have assumed in every step that the per-axis acceleration bounds $\mathbf{a}_{\min}$ and $\mathbf{a}_{\max}$ are constant. This assumption is necessary to compute bounds on the velocity update to ensure the feasibility of the updated two-segment trajectory in the next segment. However, when altering the initial conditions for the thrust acceleration decomposition by updating the unknown velocity value, the distribution of the limited thrust acceleration norm will result in different per-axis accelerations, which can lead to a slower two-segment trajectory.

Moreover, while using the Algorithm 2 for thrust decomposition equalizes the axis trajectories implicitly by distributing the per-axis accelerations according to the minimum trajectory time of each axis, doing so by reducing the maximum allowed accelerations leaves a

Algorithm 2: Limited Thrust Decomposition

Input: $\mathbf{p}_0$ - start position, $\mathbf{v}_0$ - start velocity, $\mathbf{p}_{12}$ - end position, $\mathbf{v}_{12}$ - end velocity, $a_{T_{max}}$ - maximal thrust acceleration, ε - acceleration norm precision, N - maximum number of iterations

Output: Π - resulting trajectory

$a_{max} \leftarrow$ **getEqualPerAxisMaxAcceleration**$(a_{T_{max}})$ ▷ (4.31)

$\mathbf{a}_{\max_0} \leftarrow [a_{\text{init}}, a_{\text{init}}, a_{\text{init}}]^{\text{T}}$

$\mathbf{a}_{\min_0} \leftarrow -\mathbf{a}_{\max_0} + 2\mathbf{g}$

$\Pi \leftarrow$ **computeTrajectory3D**$(\mathbf{p}_0, \mathbf{v}_0, \mathbf{p}_{12}, \mathbf{v}_{12}, \mathbf{a}_{min_0}, \mathbf{a}_{max_0})$

for $j \in 0, \ldots, N-1$ **do**

  $\mathbf{a}_{1_j}, \mathbf{a}_{2_j} \leftarrow$ **getAccelerationVectors**(Π)

  $\mathbf{a}_{L_j} \leftarrow \arg\max(\|\mathbf{a}_{1_j} - \mathbf{g}\|, \|\mathbf{a}_{2_j} - \mathbf{g}\|)$

  if $\left|\|\mathbf{a}_{L_j}\| - a_{max}^T\right| < \varepsilon$ **then**

    break

  $\beta \leftarrow$ **computeScalingFactor**$(\mathbf{a}_{L_j})$ ▷ (4.38)

  $\mathbf{a}_{\max_{j+1}} \leftarrow \beta \mathbf{a}_{L_j}$

  $\mathbf{a}_{\min_{j+1}} \leftarrow -\beta \mathbf{a}_{L_j} + 2\mathbf{g}$

  Update Π given $\mathbf{a}_{\min_{j+1}}$, $\mathbf{a}_{\max_{j+1}}$

---

limited room for velocity optimization. This could result in a case where a velocity update would reduce the trajectory time but the acceleration limits placed on the corresponding axis would be too tight and the new updated trajectory would be infeasible.

However, the Algorithm 1 is relatively robust to cases where the updates lead to infeasible or trajectory duration increasing updates. So, including the limited thrust decomposition into trajectory re-calculation steps in Algorithm 1 could lead to reasonable results.

To test this theory, we propose three different approaches to including the thrust decomposition in the velocity optimization:

1. In the first approach, the velocity optimization Algorithm 1 is run for constant acceleration limits, which are calculated using the same equation (4.30) that is used in the initial step of the thrust decomposition algorithm. After convergence of the velocity optimization, the resulting trajectory is recomputed using the Algorithm 2 to minimize the trajectory duration further.
2. The second approach is similar to the first one, but this time, after convergence of the Algorithm 1, the velocity optimization algorithm is run again. Only this time, the thrust decomposition algorithm is used in every instance where a trajectory segment is (re-)computed.
3. In the last approach, the velocity optimization is run using the thrust decomposition only.

It can be expected that the different approaches will vary in computation time and quality of the solution. In our case, it is the final trajectory duration we are trying to minimize.

4.6 Path planning using Clustering Topological PRM

For path planning, we use the Clustering Topological PRM (CTopPRM) [9] method. It provides an efficient way of computing collision-free topologically distinct paths in a known environment given the start and end pose of an UAV. This is highly beneficial for our task of finding a minimum-time trajectory, as we are provided with a set of possible paths from which we can compute the corresponding trajectories and select the one with the shortest duration. Moreover, the authors of [9] reported that the CTopPRM method is superior in finding all homotopically distinct paths to the previous methods, namely [18] and [5]. Therefore, the CTopPRM algorithm has been chosen as a suitable tool for the path planning task and we will briefly summarize the main concept in the following paragraphs.

The CTopPRM algorithm is designed using several methods that are run in a hierarchical order. First, a graph-based representation of the continuous configuration space called roadmap is created using a sampling-based motion planner called Informed Probabilistic Roadmap [19]. It is a modified Probabilistic Roadmap (PRM) [16] motion planner, where in the sampling phase, the configurations are sampled only from an ellipsoid with its principal axis spanning from the start to goal configuration. The size of the ellipsoid is reduced based on the length of the shortest path found throughout the sampling process. The sample configurations are added to the roadmap only if they lie in $\mathcal{C}_{free}$ and there exists a collision-free path connecting it to the roadmap. The connecting paths are computed using a local planner.

In the next step, the resulting roadmap is iteratively divided into clusters to obtain a low-order graph consisting of the cluster centers as its vertices. The reduced graph can then be time-efficiently searched for all distinct paths in the following step. This cluster placement is done methodically to find distinct paths with different homotopy classes, which means that they pass around obstacles from different sides. The homotopy classification is approximated using the Uniform Visibility Deformation (UDV) [5] concept, where two paths $P_1(s)$ and $P_2(s)$ belong to the same UDV case if for all $s \in [0, 1]$ the line segment between configurations $P_1(s)$ and $P_2(s)$ is collision-free.

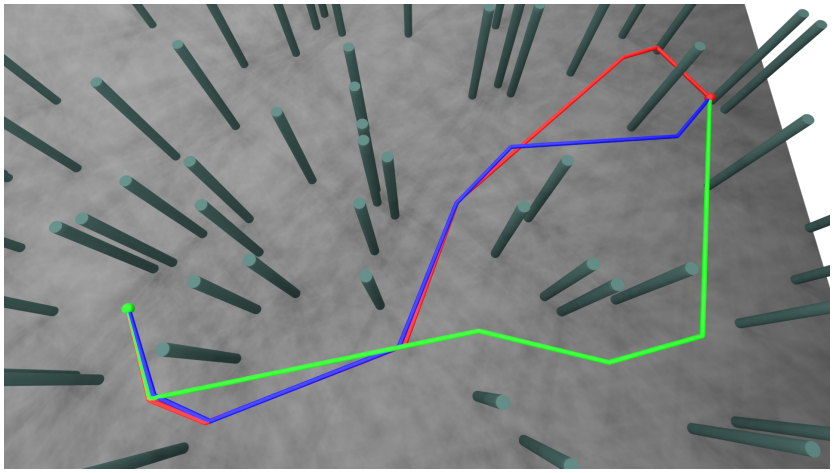


Figure 4.6: Example distinct paths generated with the CTopPRM [9] algorithm in a forest-like environment between the start configuration (green point) and end configuration (red point).

After a pre-determined number of clusters have been reached, the resulting sparse roadmap is searched for all possible paths with different UDV classes. To this end, a modified

depth-first graph-search algorithm is used. Additionally, the resulting paths are shortened and all paths with lengths above a threshold are filtered out. The threshold is set as the multiple of the shortest distinct path's length. An example output of this path-planning algorithm is shown in Figure 4.6, where a forest-like environment was used to simulate an agile-flight scenario.

4.7 Collision-free trajectory computation

In the previous Section 4.6, we have described a method for planning collision-free paths in a known environment. The goal of guided collision-free trajectory computation is to effectively find a collision-free trajectory based on those paths. Additionally, given our application of time-critical tasks, the resulting trajectory should have the smallest trajectory duration possible. An approach for solving this task has been introduced in [18]. However, as we will discuss in this section and showcase in Section 5.4, the proposed method is suitable only for certain environments.

The two main building blocks of collision-free trajectory computation are an algorithm for trajectory computation and a method for collision checking. We have introduced several approaches to computing a minimum-time point-mass model trajectory in Section 4.5, which combined the proposed methods for point-mass model trajectory computation, velocity optimization and limited thrust decomposition. For collision checking, we first need to be able to sample the trajectory.

A trajectory can be sampled in different ways; we will deal with sampling in time and sampling based on the maximum distance in position between the consecutive trajectory samples. Sampling in time is usually used for discrete control, where a controller running at a frequency f_c is given a sampled trajectory for execution. To sample a trajectory in time, we start with the initial trajectory state containing initial position $\mathbf{p}(0) = \mathbf{p}_0$, velocity $\mathbf{v}[0] = \mathbf{v}_0$ and $\mathbf{a}[0]$ at time $t_0 = 0$. Then, until the whole trajectory is sampled, the time is increased by the sampling period $T_s = 1/f_c$ after every sample $t_{k+1} = t_k + T_s$ and the trajectory is integrated according to the equations (4.1) to obtain the corresponding samples of time t_k, position $\mathbf{p}[t_k]$, velocity $\mathbf{v}[t_k]$ and acceleration $\mathbf{a}[t_k]$.

However, for collision checking, we need the trajectory sampled so that the Euclidean distance between two consecutive trajectory points is below a predefined threshold $\delta \in \mathbb{R}$. There are different methods for collision checking. In our case, we use Euclidean Signed Distant Field, where our planning environment is discretized into 3D grid map and each cell value contains the distance to the closest obstacle. The grid has a defined resolution, i.e., the size of the grid cells. For collision checking, it makes sense for the distance between two consecutive trajectory positions δ to be greater or equal to the grid resolution. Otherwise, the collision checking would be ineffective as a collision would be checked for each cell more than once.

An equidistant sampling is then executed by starting at the initial state and solving the following equation

$$\delta_{\text{ax}} = v[t_{k-1}]t_k + \frac{1}{2}a[t_{k-1}]t_k^2, \tag{4.40}$$

where δ_{ax} is the per-axis maximum distance in position between two samples, $v[t_{k-1}]$ is the singe-axis velocity at the previous time-stamp, $a[t_{k-1}]$ is the single-axis acceleration at the given timestamp, and t_k is the unknown time of the next sample. For simplification, we are using a per-axis threshold. It is sufficient to ensure that the collision is checked for a sample lying inside a neighboring grid map cell for effective computation.

The equation (4.40) has a closed form solution

$$t_k = \frac{vt_{k-1} \pm \sqrt{v[t_{k_1}]^2 - 2a[t_{k-1}]\delta_{\text{ax}}}}{a[t_{k-1}]}, \tag{4.41}$$

where we choose the non-negative value for the time stamp t_k. Due to the bang-bang policy trajectory, it must be checked that $a[t_{k-1}] = a[t_k]$. Otherwise, if a change of acceleration occurs at switch time t_s, the distance $\delta_s = |p[t_s] - p[t_{k-1}]|$ must be calculated using acceleration $a[t_{k-1}]$ and equation (4.40) solved again, this time for $\delta_{\text{ax}} := \delta_{\text{ax}} - \delta_s$ and $a[t_{k-1}] := a[t_k]$ to obtain the next time-stamp.

By solving (4.40) for every trajectory axis, we obtain three time-stamps t_{k_x}, t_{k_y} and t_{k_z}, from which the minimum is selected as the next sample time at which the trajectory is sampled. We refer to this equidistant sampling method as **sampleTrajectory** in Algorithm 3.

A collision of an UAV is then checked by obtaining the 3D grid map value at the cell corresponding to the current UAV's body center position. If the distance to the closest obstacle is smaller than the distance from the center of the UAV to the closest point on its surface in the corresponding direction, there is a collision. Usually, the geometrical model of the UAV's body is simplified to a sphere with radius $r \in \mathbb{R}$, and the collision checking is thus simplified to checking if the distance to the closest obstacle is smaller or equal to the UAV's radius r. In our application, we use this simplification to reduce the computational time.

Having these building blocks ready, the authors of [18] suggested the following approach. To omit the computational burden of generating trajectories for all the paths in the initial path set $\mathbb{P} = \{P_1, \ldots, P_n\}$ found by the path-planning algorithm, the proposed method starts by computing a naive trajectory between the start and goal states disregarding all obstacles along the way and adds it to a set of possible trajectories. The approach then uses a dynamic programming search with a priority queue to select the trajectory with the shortest duration. A selected naive trajectory is sampled and checked for collisions. At the point of the first collision, the closest point lying on the nearest path $P_s \in \mathbb{P}$ is computed and added to the corresponding naive sequence of path points, from which a new trajectory is computed. By repeatedly adding via-points lying on the already computed collision path to the initial naive two-waypoint trajectory, the algorithm computes the final shortest-time collision-free trajectory.

This approach, however, has its drawbacks. As we will show in Section 5.3, the computational time of our method for point-mass trajectory generation increases linearly with the number of via-waypoints. Adding a large number of waypoints is thus detrimental to the effectiveness of the trajectory generation process. Using the proposed approach can in some cases lead to a significant increase in computational time as shown in Figure 4.7 or even to an endless cycle. Moreover, it can render non-minimum-time trajectories as adding many waypoints results in a slow flight of the UAV due to many steering maneuvers and consequently shorter trajectory durations.

Figure 4.7 displays a scenario, where two paths around an obstacle have been found from the start (green) to the goal (red) position. A 2D environment is used for simplification. Applying the above-described approach results in a) multiple via-waypoint insertions and trajectory recomputations in the case of the blue path, and b) endless adding of the start position in the case of the red path, as the start position is the closest point to the initial collision marked with the black cross. It should be noted, that this approach was designed for multi-goal trajectory generation in cluttered environments, where computing all the computed paths for all goals becomes much more computationally demanding than in our case of a single-

goal trajectory generation. However, as we will show in Section 5.4, scenario (a) can also occur in cluttered environments for larger clearances from the obstacles.

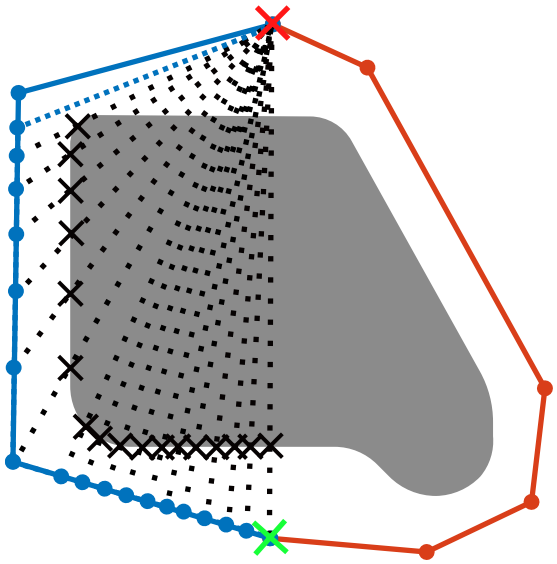


Figure 4.7: Example scenario showcasing drawbacks of the collision-free trajectory generation approach proposed in [18]. The planning of naive trajectories (black lines) from the start state (greed cross) to the end state (red cross) results in many added waypoints (blue dots) due to the collision points (black crosses) of the naive trajectories with an obstacle (grey). The final trajectory computed using the blue path is marked with the blue dashed line.

Due to the aforementioned drawbacks of the method [18], we will adapt the method to suit a more general application. Combining the trajectory generation algorithm, trajectory sampling, and collision checking in a known map, we can compute a minimum-time collision-free trajectory using the Algorithm 3. First, we calculate the corresponding trajectories $\mathbb{T} = \{\Pi_1, \ldots, \Pi_n\}$ with Velocity Optimization (VO) for all n paths in the initial path set $\mathbb{P}$ using the approach described in Section 4.3, while simultaneously using one of the approaches for limited Thrust Decomposition (TD) described in Section 4.4. From the calculated trajectories, trajectory Π_s with the minimum duration is selected (method **findAndRemoveBestTrajectory** in Algorithm 3), sampled according to the grid resolution and checked for collision.

If the computed trajectory collides with an obstacle, we find the closest path segment of the corresponding path P_s to the collision point using basic algebra. This method is called **findColsestPathSegmentToTheCollsion** in Algorithm 3. The path segment is defined by the corresponding via-waypoints $\mathbf{p}_a$ and $\mathbf{p}_b$. Then, the closest point $\mathbf{p}_c$ on the path between $\mathbf{p}_a$ and $\mathbf{p_b}$ is computed (method **findColsestPointOnPathToTheCollsion** in Algorithm 3) and added between the original path points into the initial path P_s to prevent the collision. An updated trajectory Π'_s is then computed from the updated path P'_s, and both replace the original trajectory and path in the sets $\mathbb{T}$ and $\mathbb{P}$, respectively. From the updated set of trajectories, the path with minimal duration is again selected. This process repeats until a collision-free path is found.

Algorithm 3: Collision-Free Point-Mass Trajectory Generation

Input: $\mathbb{P} = \{P_1, \ldots, P_n\}$ - initial path set, δ - collision checking precision
Output: $\mathbf{\Pi}_s$ - resulting trajectory

```
1
2  𝕋 ← Compute trajectories with VO and TD for all paths P ∈ ℙ      ▷ Alg. 1, Alg. 2
3  while 𝕋 ≠ ∅ do
4      Π_s ← findAndRemoveBestTrajectory(𝕋)
5      Π_δ ← sampleTrajectory(Π_s, δ)
6      if Π_δ is collision-free then
7          break
8      else
9          P_s ← path corresponding to Π_s
10         (p_a, p_b) ← findColsestPathSegmentToTheCollsion(P_s)
11         p_c ← findColsestPointOnPathToTheCollsion(p_a, p_b)
12         P'_s ← insert p_c between p_a, p_b in P_s
13         Π'_s ← Compute trajectory with VO and TD from P_s      ▷ Alg. 1, Alg. 2
14         𝕋 ← 𝕋 ∪ {Π'_s}
```

Chapter 5

Results

5.1 Velocity optimization algorithm

We have selected four paths to test the performance of the proposed method for a minimum-time multi-waypoint point-mass model trajectory computation. All of them are three-dimensional paths that have been selected with the following aim:

1. Path P_1 depicted in Figure 5.1a and defined in Table A.1 simulates online trajectory planning using the receding horizon approach, where only next n waypoints are considered. This method can be applied in situations, where the UAV has diverted from a preplanned mission due to an obstacle-avoiding maneuver or external disturbance. In our case, we consider $n = 3$ and the path is a segment of Path 2 to simulate a disturbance correction similar to the scenario presented in [7].
2. Path P_2 shown in Figure 5.1b and defined in Table A.2 is used to compare our method for trajectory generation and optimization with the method introduced in article [7], from which this path has been taken over. It describes an agile flight through a drone racing circuit where the UAV flies through some of the waypoints/gates multiple times.
3. Paths P_3 and P_4 shown in Figures 5.1c and 5.1d, respectively, are collision-free paths that have been computed using the method [9] described in Section 4.6. The paths were planned in a forest-like environment shown in Figure 4.6 to simulate a scenario where agile flight is necessary to avoid obstacles. We have chosen similar paths with different numbers of waypoints to demonstrate the effect of an increased number of trajectory segments that require velocity optimization on the performance of our method for trajectory computation. The paths are defined in Table A.3 and Table A.4, respectively.

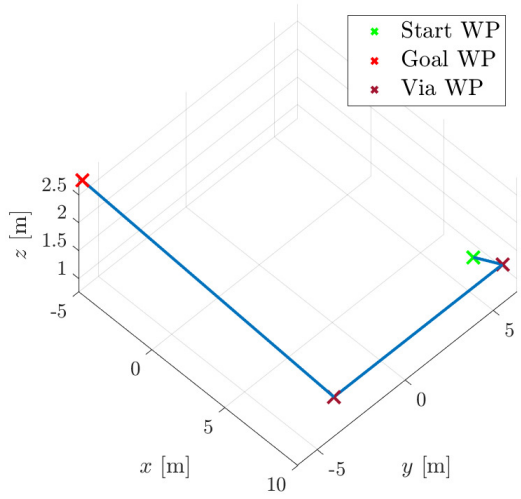
As we have stated at the beginning of this thesis, we are searching for a minimum-time trajectory that can be computed in real-time. This was the main motivation behind the selection of the simplified point-mass model of the multirotor UAV. The testing criteria are thus the resulting trajectory duration T_{tr}^* after velocity optimization and the computational time τ_{cmp} of the trajectory generation process.

All our methods introduced in this work have been implemented in C++ and were run on an Intel Core i9-12900HK processor with a clock speed of 5 GHz to obtain all the results for the experiments described in this chapter.

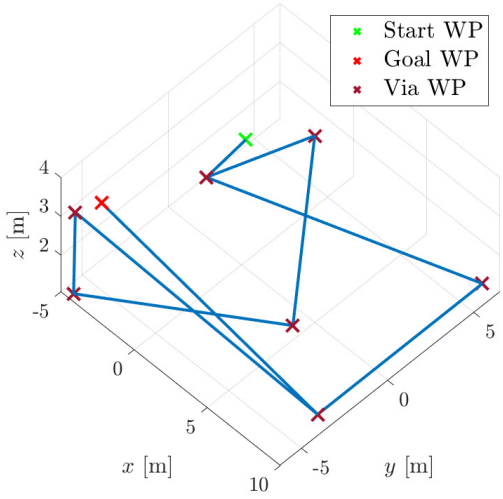
Table 5.1: Multirotor UAV parameters used for the experiments.

Parameter	Value
m [kg]	1.0
f_{max} [N]	40

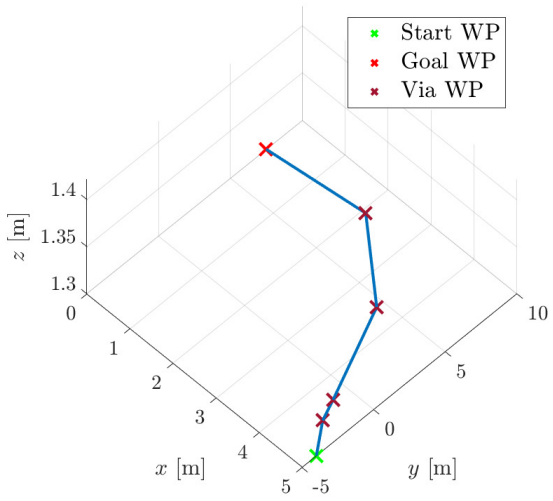
Due to time constraints, the proposed method has been tested only in a simulated



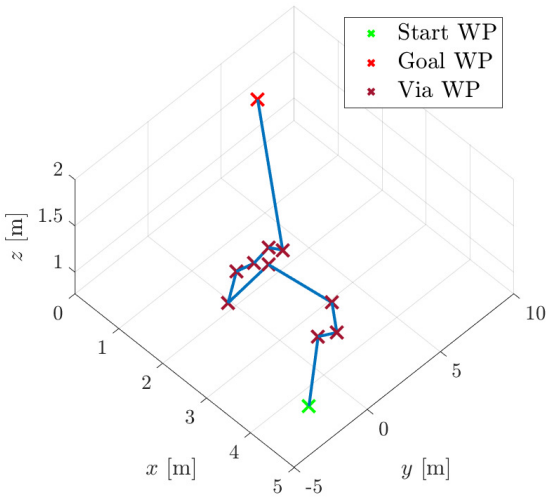
(a) Path P_1.



(b) Path P_2.



(c) Path P_3.



(d) Path P_4.

Figure 5.1: Visualization of the testing paths P_1 (a), P_2 (b), P_3 (c), and P_4 (d). Each one of them is a three-dimensional path with starting waypoint, goal waypoint, and via waypoints.

environment using a virtual model of an UAV. The selected multirotor UAV parameters used for all the testing are shown in Table 5.1, where m is the UAV weight and $f_{\max}$ is the maximal thrust force the UAV can produce using its propulsion system. The resulting 4:1 thrust-to-weight ratio is suitable for an agile flight where minimum time trajectories are usually required; it is thus suitable also for our application.

5.1.1 Velocity optimization method parameter selection

For efficient use of the velocity optimization process introduced in Algorithm 1, reasonable hyperparameters have to be selected first. To recapitulate, the parameters of the velocity optimization Algorithm 1 are the initial update step α_{init}, the update step reduction factor η, the update step threshold ζ, the trajectory time update threshold ε_{VO}, and the maximum number of iterations N_{VO}. A more detailed description of those parameters can be found at the end of Section 4.3.

Parameters N_{VO} and ε_{VO} are standard parameters of iterative optimization methods, and their value is usually selected as a tradeoff between the precision of the result and computational time. For our application, it has been observed that

$$N_{VO} = 30 \tag{5.1}$$

is a reasonable value as most of the tested trajectories converged within this limit. The threshold

$$\varepsilon_{VO} = 10^{-3} \tag{5.2}$$

has been selected for the trajectory time difference threshold after an update. As we are computing a trajectory for a simplified model, there is no need for more precise results as the final trajectory executed by a real UAV will be, in either case, different due to the physical limitations of a UAV. The value of the update step threshold has been selected as

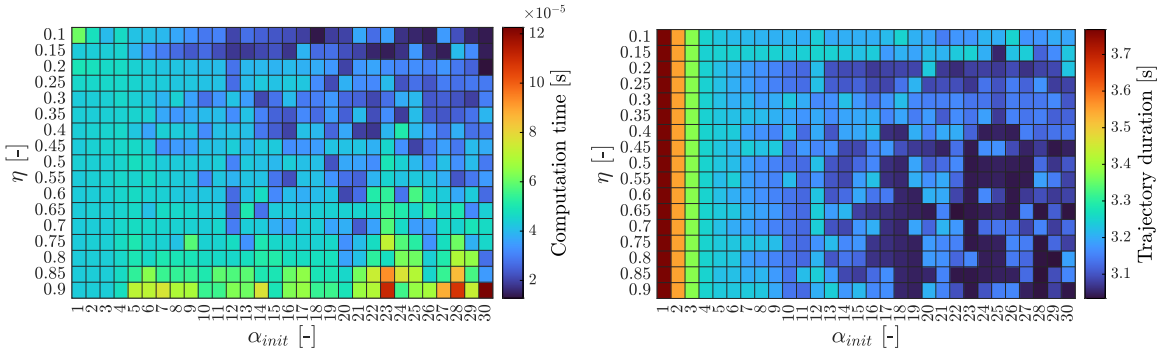
$$\zeta = 10^{-3} \tag{5.3}$$

for the same reason. For a smaller update step value, the effect of the velocity change on the resulting trajectory would be marginal when taking the selected ε_{VO} into account.

To test the effect of the two remaining parameters α_{init} and η on the velocity optimization process, we have made the following experiments. For all four testing paths, we have computed optimized trajectories according to the Algorithm 1 using different combinations of values of the parameters α_{init} and η. We have also recorded the duration and computational time of each trajectory. The resulting values are visualized using heat maps which can be found in Appendix A.3 for all four testing paths $P_1, \ldots, P_4$.

Due to space constraints, we will demonstrate the results of the hyperparameter grid search only on heat maps displayed in Figure 5.2, which show the resulting computational times and durations of trajectory $\mathbf{\Pi}_1$ computed from path P_1.

We analyze the impact of various initial step size α_{init} values on the computed trajectory first. Figure 5.2b shows the resulting trajectory durations and the first thing that can be deduced about the step size of the gradient-method-based algorithm is that a small step size leads to increased trajectory duration. This is probably the consequence of updating the velocities separately for each axis and the subsequent trajectory recomputation and axis synchronization. As the axis influence each other in the axis-synchronization process, the



(a) Trajectory $\mathbf{\Pi}_1$ computational times.

(b) Trajectory $\mathbf{\Pi}_1$ durations.

Figure 5.2: Heat maps of computational times and trajectory durations for different values of Algorithm 1 hyperparameters step length α_{init} and step length reduction factor η. The trajectory $\mathbf{\Pi}_1$ has been computed from the testing path P_1.

small velocity updates can get stuck in a local minimum which is far from the minimum time achievable by our method using different parameter values. Also, updating the unknown velocities in a two-segment window means that the velocity update is constrained by the current values of velocities at the two-segment trajectory endpoints which restricts the velocity updates. Small changes to those values can result in early termination of the optimization process as the improvement in trajectory duration lies below the given threshold. Increasing α_{init} leads to smaller trajectory durations as the larger updates will escape from the local minima. However, a too-large step size can also lead to worse results; this can be seen by comparing the trajectory duration heat-maps in Figure A.1. For all four tested cases, the best results can be achieved by selecting the interval $\alpha_{\text{init}} \in [20, 30]$ for the initial step size.

The reduction factor η also influences the resulting trajectory duration. As we are using the reduction factor to scale down the step size in problematic sections where the objective function has increased its value due to the velocity update, selecting higher values will lead to more evaluations of different velocity updates before the step size is scaled down below the threshold ζ. This results in a more precise velocity optimization in some cases. However, an increased number of evaluations leads to increased computational time, as illustrated in Figure 5.2a. The displayed values are the mean values of computational times from 100 runs. In our case, choosing η from the interval $[0.4, 0.7]$ seems reasonable when comparing the results for all the testing paths displayed in Figure A.1 as those values represent a good tradeoff between computational time and resulting trajectory duration.

For all further testing, the following values will be used for the initial step size and the step size reduction factor:

$$\begin{aligned} \alpha_{\text{init}} &= 25, \\ \eta &= 0.5. \end{aligned} \tag{5.4}$$

Using the results of the parameter search, we can also demonstrate some characteristics of the velocity optimization process. The heat maps presented in Figure A.1 show that the several approximations, separate treating of the velocity updates, and the influence of the single-axis trajectories on each other lead to different results for similar values of the selected hyperparameters, i.e., a small change in one parameter can lead to a significant difference in the resulting trajectory duration. Further, a small selection of hyperparameters for the given problem usually results in a smaller trajectory duration compared to the rest of the tested

cases, as shown in Figure A.1d. However, we were unable to derive any general rules for the parameter selection to obtain the shortest trajectory duration possible for a given path.

The proposed values for the hyperparameters have been selected based on the four tested trajectories and should represent a tradeoff between the resulting trajectory duration and computational time. Other parameter values could be more suitable in different applications, i.e., for an UAV with different parameters or for paths containing more than 20 waypoints.

5.1.2 Thrust decomposition in velocity optimization

When defining Algorithm 2 for limited thrust acceleration decomposition in a point-mass model three-dimensional trajectory segment, we claimed that the proposed method for iterative per-axis acceleration bounds adjustment converges reasonably fast despite the approximations used.

Table 5.2 shows the resulting convergence of computing point-mass model trajectories $\mathbf{\Pi}_1, \ldots, \mathbf{\Pi}_4$ from testing paths $P_1, \ldots, P_4$ using the proposed thrust decomposition approach. The precision

$$\varepsilon_{TD} = 10^{-3} \tag{5.5}$$

was used because the point-mass model trajectory is only an approximation of a trajectory that a physical UAV can execute; a higher precision is, therefore, unnecessary.

Table 5.2: Results for thrust decomposition convergence shown on trajectories $\mathbf{\Pi}_1, \ldots, \mathbf{\Pi}_4$ corresponding to paths $P_1, \ldots, P_4$ with n trajectory segments. Displayed are the mean number of iterations μ_{iter} needed for thrust decomposition of a trajectory segment for all segments of the corresponding trajectory, the maximum and minimum number of iterations.

	n [-]	μ_{iter} [-]	Min Iter. [-]	Max Iter. [-]
$\mathbf{\Pi}_1$	3	5.000	2	9
$\mathbf{\Pi}_2$	18	3.278	2	5
$\mathbf{\Pi}_3$	5	2.400	2	3
$\mathbf{\Pi}_4$	10	3.800	2	5

The data in Table 5.2 clearly show that for all trajectory segments, the thrust decomposition approach converged within ten iterations, and except for the trajectory $\mathbf{\Pi}_1$, the mean number of iterations necessary for thrust acceleration decomposition lies under four iterations. The mean computational time for a single iteration was measured at 0.2321 μs; the proposed method is, therefore, suitable for incorporation into a real-time trajectory generation algorithm.

In Section 4.5, we have also proposed three approaches to including the limited thrust decomposition into the velocity optimization process. To recapitulate:

1. The first approach consists of running the Velocity Optimization (VO) process described in Algorithm 1 with per-axis acceleration limits and then recomputing the final trajectory using the Thrust Decomposition (TD) Algorithm 2. We will refer to this approach as Approach 1 or "$VO \rightarrow TD$."
2. The second proposed approach also uses VO with per-axis acceleration limits to obtain an intermediate result but then runs the VO process again, this time using the TD algorithm in every instance of the VO process, where a trajectory is recomputed. We will refer to this approach as Approach 2 or "$VO \rightarrow VO + TD$."

3. The last approach, referred to as Approach 3 or "$VO+TD$," skips the VO process with per-axis acceleration limits and runs the VO using the TD method only.

We have computed trajectories $\mathbf{\Pi}_i$ from the corresponding testing paths P_i, $i = 1, \ldots, 4$, for all three listed approaches using the UAV parameters specified in Table 5.1 and Algorithm 1 parameters selected in Section 5.1.1. The start and end velocities were set to zero for all test cases except for $\mathbf{\Pi_1}$ with

$$\begin{aligned} v_{0_{\mathbf{\Pi_1}}} &= [12.4, 4.53, -2.59]^{\mathrm{T}}\ \mathrm{m \cdot s^{-1}}, \\ v_{3_{\mathbf{\Pi_1}}} &= [-11.0, 0, 0]^{\mathrm{T}}\ \mathrm{m \cdot s^{-1}}, \end{aligned} \tag{5.6}$$

to simulate a mid-flight replanning scenario. The resulting trajectory durations are listed in Table 5.3. The first two columns also show displayed trajectory durations of initial Point-Mass Model (PMM) trajectories with zero velocities at the via-waypoint computed using a) symmetric acceleration bounds (4.30) designed as T_{PMM} and b) thrust decomposition listed under T_{TD}. For the sake of clarity, the resulting trajectory durations have also been visualized using a bar graph in Figure 5.3.

Table 5.3: Trajectory durations T of tested trajectories $\mathbf{\Pi}_1, \ldots, \mathbf{\Pi}_4$ computed using different thrust decomposition (TD) approaches in velocity optimization (VO).

	T_{PMM} [s]	T_{TD} [s]	$T_{VO\rightarrow TD}$ [s]	$T_{VO\rightarrow VO+TD}$ [s]	T_{VO+TD} [s]
$\mathbf{\Pi}_1$	4.0493	2.7189	2.4418	2.3866	**2.3164**
$\mathbf{\Pi}_2$	23.4416	17.8943	16.2220	**15.6055**	15.8866
$\mathbf{\Pi}_3$	3.2833	2.4549	1.6999	**1.4974**	1.6839
$\mathbf{\Pi}_4$	4.6045	3.5298	2.5828	**2.4160**	3.3766

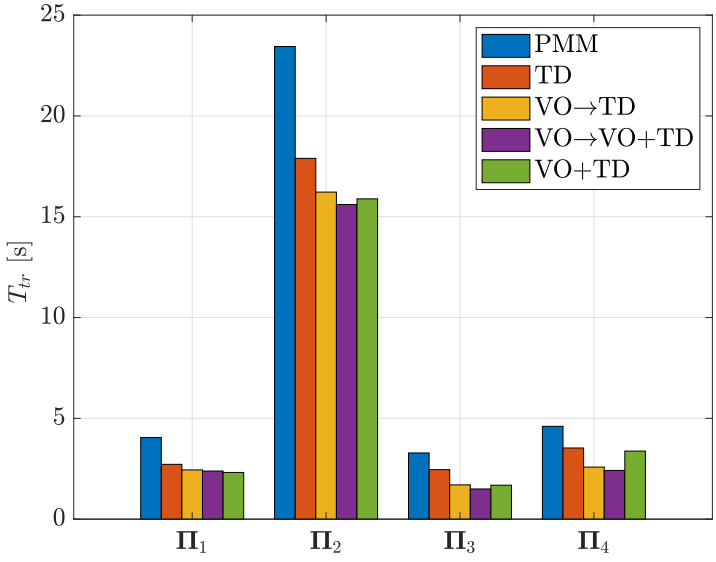


Figure 5.3: Trajectory durations T_{tr} for trajectories $\mathbf{\Pi}_i$, $i = 1, \ldots, 4$ computed a) without velocity optimization (VO) and thrust decomposition (TD) labeled as "PMM," b) without VO using TD labeled as "TD," c) with VO and subsequent TD ("$VO \rightarrow TD$"), d) with VO and subsequent VO with TD ("$VO \rightarrow VO + TD$"), and d) using only VO with TD marked as "$VO + TD$."

When comparing the initial trajectory durations T_{PMM} with T_{TD}, it is clear that the

TD alone reduces the resulting trajectory duration significantly. It is, therefore, vital for minimum-time trajectory planning for a multirotor UAV to solve the limited thrust decomposition problem. Further, the VO using symmetrical per-axis (thrust) acceleration bounds and subsequent recalculation of the optimized trajectory using the TD in Approach 1 reduces trajectory duration even more. Using the proposed VO method for a multirotor UAV trajectory generation is thus beneficial even if the algorithm assumes per-axis (body) acceleration bounds to be constant and decoupled. Moreover, given the mean computational times computed from 100 test runs and displayed in Table 5.4, the trajectory optimization can be computed in sub-millisecond time for all tested cases. The computational times are also more than four times shorter compared to the two remaining approaches in virtually all cases. The only exception is the trajectory Π_4, where Approach 3 leads to a shorter computational time τ_{VO+TD}. However, this is due to an early termination of the optimization process, as we will explain below.

Table 5.4: Computational times τ of tested trajectories $\mathbf{\Pi}_1, \ldots, \mathbf{\Pi}_4$ computed using different Thrust Decomposition (TD) approaches in Velocity Optimization (VO).

	$\tau_{VO \rightarrow TD}$ [ms]	$\tau_{VO \rightarrow VO+TD}$ [ms]	τ_{VO+TD} [ms]
$\mathbf{\Pi}_1$	**0.0418**	0.1603	1.7350
$\mathbf{\Pi}_2$	**0.2347**	4.7060	18.1900
$\mathbf{\Pi}_3$	**0.0869**	0.4017	0.7896
$\mathbf{\Pi}_4$	0.7917	5.6290	**0.2286**

Using the TD method inside the VO process in Approach 3 results in a shorter trajectory duration than when using Approach 1 in most cases. This, however, is not guaranteed which can be seen on a significantly longer trajectory duration T_{VO+TD} for trajectory $\mathbf{\Pi}_4$. In this case, the VO optimization terminated early, which can be also deduced from the short computational time τ_{VO+VD} in Table 5.4 and a small reduction of trajectory duration when comparing the corresponding trajectory duration T_{VO+TD} in Table 5.3 with the duration T_{TD} where no VO was used. The most probable cause for the early termination is that our VO method was designed for constant acceleration limits and all the velocity update step bounds are computed based on this assumption. The optimization process can thus get stuck in a local minimum due to an unfit velocity update. Consequently, while Algorithm 1 is relatively robust to infeasible velocity updates, the usage of the thrust decomposition method in the VO process leads either to early termination or slow convergence. This can be seen when comparing the computational times $\tau_{VO \rightarrow TD}$ of Approach 1 and τ_{VO+TD} of Approach 3 in Table 5.4.

Running a VO process with per-axis limits before the VO with TD in Approach 2 reduces the computational times compared to Approach 3 in all relevant cases. Moreover, except for trajectory $\mathbf{\Pi}_1$, Approach 2 resulted in the shortest trajectory durations $T_{VO \rightarrow VO+TD}$ of all three Approaches, as can be seen in the corresponding column of Table 5.4. The exception in the case of trajectory $\mathbf{\Pi}_1$ underlines the unpredictability of Approach 3 where the best results in terms of trajectory duration can be obtained in rare cases, but there are also cases where the optimization process terminates early and provides the worst results in comparison with the other approaches. The best results in terms of trajectory duration can be thus obtained by running the VO algorithm twice, the second time using Algorithm 2 for the trajectory computation. This is, however, at the cost of the computational time, which is usually more than four times longer than in the case of Approach 1 due to the slow convergence and additional computational burden of the second optimization run. Nevertheless, the additional

computational time can be reduced by constraining the second VO process using the maximum number of iterations, i.e., after the first VO process, only a few iterations of the second optimization process are performed to obtain faster trajectories.

To conclude, using Approach 2 results in a trajectory with minimal duration at the cost of significantly increased computational time in most cases; Approach 1 is best in terms of computational time and is suitable for time-critical applications. Using Approach 3 can, in some cases, lead to faster trajectories; this, however, can not be guaranteed and, in some cases, the resulting trajectory duration is similar to a trajectory that was computed without velocity optimization.

5.2 Visualization of the velocity optimization convergence

To illustrate the geometrical meaning of the proposed velocity optimization Algorithm 1, we will take a closer look at the trajectory $\mathbf{\Pi_2}$ computed using the guiding path P_2. We used Approach 1 for incorporating limited thrust decomposition into the velocity optimization process presented in Section 4.5, namely recomputing the resulting trajectory of the velocity optimization process using the thrust decomposition Algorithm 2.

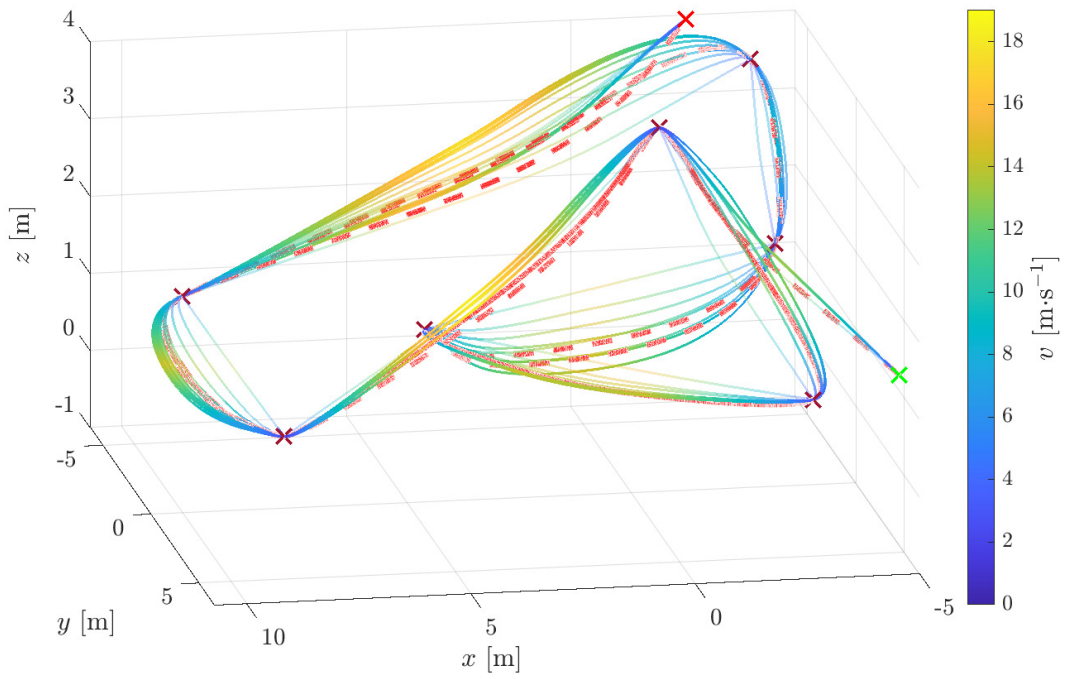


Figure 5.4: Visualization of the velocity optimization process described in Algorithm 1. Displayed are computed trajectories with increased opacity for every iteration of the optimization algorithm. The velocity v of the point-mass model is visualized using a color bar. The final trajectory recomputed using thrust decomposition Algorithm 2 is shown using a dashed red line.

Figure 5.4 shows the computed trajectory for every iteration of the optimization process, where with increasing iteration number, the opacity of the displayed trajectory also increased. In addition, the speed v of the point-mass model is displayed using a color map. With increased

iteration count, the sharp bends at the waypoints become gradually wider as the velocity at the waypoints becomes nonzero. This decreases the trajectory duration, as shown in Figure 5.5, where the convergence of the time-minimizing algorithm is visualized.

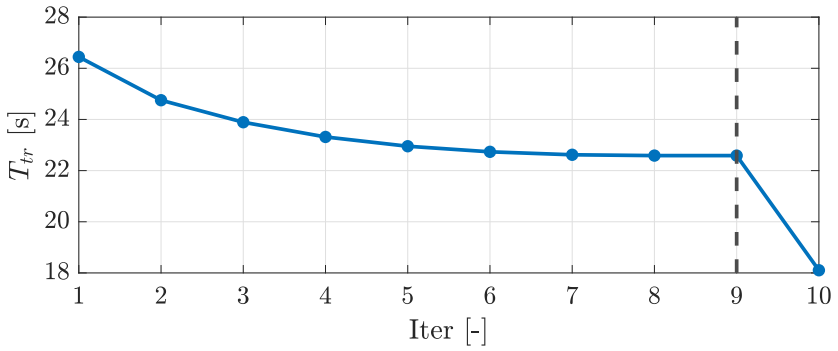


Figure 5.5: Convergence visualization of the velocity-optimizing Algorithm 1, where the trajectory duration T_{tr} is displayed for every iteration of the optimization process. The dashed vertical line at iteration number nine marks the termination of the velocity optimization process, and the trajectory duration at the tenth iteration corresponds to the recomputed final trajectory using the thrust decomposition Algorithm 2.

The optimization Algorithm 1 terminates after the 9th iteration, which is marked by a vertical line in Figure 5.5. After that, the resulting trajectory is recomputed using the thrust decomposition Algorithm 2, which decreases trajectory duration between the 9th and 10th. The recomputed trajectory is also shown in Figure 5.4 using a dashed red line.

5.3 Comparison with state-of-the-art method

In the following section, we will compare our point-mass model trajectory computation method with the state-of-the-art method proposed in [7]. The authors of [7] used similar concepts for point-mass model trajectory generation including the bang-bang approach. We thus consider it to be the most suitable candidate for the performance evaluation of the newly proposed method.

To objectively evaluate and compare the two methods for trajectory generation, it is crucial to discuss the parameter selection used for the experiments. The selected parameters strongly influence the performance of the methods, and their wrong selection could lead to inaccurate conclusions.

Regarding our proposed approach, we will use the parameters listed in Section 5.1.1 for the initial Velocity Optimization (VO) process using Algorithm 1 with subsequent recomputation of the resulting trajectory using a Thrust Decomposition (TD) Algorithm 2, i.e., Approach 1 described in Section 5.1.2. For TD, the following parameters were selected

$$\begin{aligned} \varepsilon_{TD} &= 10^{-3}, \\ N_{TD} &= 10. \end{aligned} \tag{5.7}$$

We will also present the results for proposed Approach 2, where the intermediate results for the optimized velocities are used for a second VO process with TD applied in every instance

where a trajectory is (re-)computed. However, we will use different values for some parameters in the second VO run, namely

$$\begin{aligned} N_{VO_2} &= 10, \\ \zeta_2 &= 10^{-2}, \\ \eta_2 &= 0.4. \end{aligned} \tag{5.8}$$

The reduction of the maximum number of iterations N_{VO_2}, the update step threshold ζ_2, and the step reduction factor η_2 will shorten the second VO run computational time. This is the approach we have suggested in Section 5.1.2, which is a tradeoff between increased computational time and shorter trajectory durations obtained by the second VO run.

For the state-of-the-art sampling-based approach to trajectory generation introduced in [7], we selected

$$s_{yp} = 20 \tag{5.9}$$

samples in yaw and pitch for every waypoint. A smaller number of samples led to increased computational time as the cone-refocusing approach took longer to converge. Additionally, to reduce the computation time, we have computed the heading toward the next waypoint in every waypoint. Using these headings, we can subsequently compute the initial velocity samples only in a cone spreading in the direction towards the next waypoint. The norm of the velocity samples $\mathbf{v}_s$ was reduced to the interval $\|\mathbf{v}_s\| \in [2, 20]$ m·s^{-1} and the number of velocity samples in a single direction was limited to

$$s_v = 8 \tag{5.10}$$

samples with varying velocity norm. This further reduces the number of samples in each waypoint, resulting in a smaller velocity search graph for finding the fastest trajectory.

The resulting trajectory durations and computational times computed with the Cone Refocusing (CR) method and our method based on the Gradient Method (GM) using two approaches for TD in VO are shown in Table 5.5 and Table 5.4, respectively. The presented computational times are the mean value of 100 measurements.

Table 5.5: Trajectory durations T of tested trajectories $\mathbf{\Pi}_1, \ldots, \mathbf{\Pi}_4$ computed using sampling method with Cone Refocusing (CR) proposed in [7] and our proposed method using two different approaches for Thrust Decomposition (TD) in Velocity Optimization (VO). Displayed are also the initial point-mass model trajectory durations T_{PMM} without VO for reference.

	T_{PMM} [s]	T_{CR} [s]	$T_{VO\rightarrow TD}$ [s]	$T_{VO\rightarrow VO+TD}$ [s]
$\mathbf{\Pi}_1$	4.0493	**2.2900**	2.4418	2.3866
$\mathbf{\Pi}_2$	23.4416	**14.9891**	16.2220	15.6485
$\mathbf{\Pi}_3$	3.2833	1.7360	1.6999	**1.4985**
$\mathbf{\Pi}_4$	4.6045	2.4302	2.5828	**2.4266**

For both trajectories $\mathbf{\Pi}_1$ and $\mathbf{\Pi}_2$, the CR approach found faster trajectories than those computed by our method. One of the main reasons is that throughout the whole trajectory $\mathbf{\Pi}_1$, the z-axis segments are in the role of SYNC trajectories. As our method handles the velocity updates per axis, the z-component of velocities in the trajectory $\mathbf{\Pi}_1$ via waypoints remain unchanged at the initial zero value throughout the whole velocity optimization process. Because the path P_1 that guides the trajectory $\mathbf{\Pi}_1$ corresponds to a path segment of path P_2, the z-components of the velocities in the corresponding trajectory $\mathbf{\Pi}_2$ waypoints also remains zero, which, which results in a slower trajectory from the global point of view.

Table 5.6: Computational times τ of tested trajectories $\mathbf{\Pi}_1, \ldots, \mathbf{\Pi}_4$ computed using sampling method with Cone Refocusing (CR) proposed in [7] and our proposed method using two different approaches for Thrust Decomposition (TD) in Velocity Optimization (VO).

	τ_{CR} [ms]	$\tau_{VO \to TD}$ [ms]	$\tau_{VO \to VO+TD}$ [ms]
$\mathbf{\Pi}_1$	9.6996	**0.0465**	0.1281
$\mathbf{\Pi}_2$	178.6230	**0.2291**	2.3182
$\mathbf{\Pi}_3$	39.7393	**0.0899**	0.3385
$\mathbf{\Pi}_4$	88.8859	**0.7571**	2.9721

The consequence of the per-axis optimization approach can be seen in Figure 5.6 where the resulting trajectories computed from path P_2 are shown for both tested methods. Approach 2 for thrust decomposition in velocity optimization was used to compute the displayed trajectory corresponding to our method based on the Gradient method. When comparing the trajectory courses between the two waypoints on the far left side in Figure 5.6 it can be seen that the CR approach resulted in a trajectory with a significant change in the z-component of positions between the corresponding waypoints. Our GM-based approach, on the other hand, keeps the change in the z-component of the positions between the corresponding waypoints tight which results in a trajectory with a longer duration.

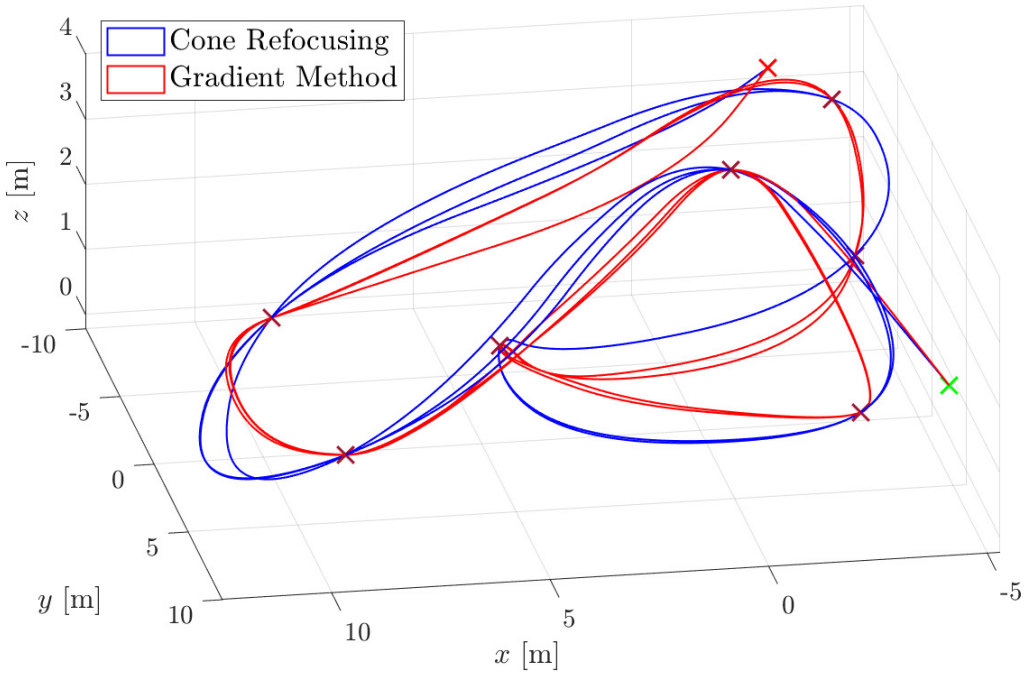


Figure 5.6: Visualization of trajectories computed from path P_1 using the cone-refocusing-based method [7] and our approach based on the gradient method. The "$VO \to VO + TD$" Approach 2 was used for thrust decomposition incorporation into our velocity optimization method.

For the remaining two trajectories, our Approach 2 resulted in shorter trajectory durations $T_{VO \to VO+TD}$ than the CR-based method For Approach 1, it is the case only with trajectory $\mathbf{\Pi}_3$. Nonetheless, in both cases, the resulting trajectory durations are significantly

shorter than the durations T_{PMM} of the original point-mass model trajectories with zero velocities at the via waypoints.

The main contribution of our approach compared to the CR-based algorithm is the reduced computational time of the trajectory generation. The computational times are more than 100 times shorter when using our Approach 1 in all of the tested cases. Even though the resulting trajectory durations are longer in most cases, the significantly shorter computational times mean that our method can be used in real-time applications even for trajectories with a larger number of waypoints. Moreover, using Approach 2 with the reduced maximal number of iterations leads to computational times that are more than 20 times shorter than those of the CR-based algorithm. The resulting trajectory durations under this approach are comparable with the CR-based method and are even shorter in several instances.

Further, the disadvantage of the sampling-based approach is noticeable when comparing the computational times of trajectories with small and large waypoint counts in Table 5.6. As the number of graph edges in the velocity search graph grows quadratically with an increasing number of samples per waypoint, and thus also the complexity of the graph search, short computational times are obtained only for a limited number of waypoints. Reducing the number of samples for longer trajectories would lead to slower convergence and thus increased computational time, so larger numbers of samples have to be used. This can be seen when comparing the τ_{CR} values for trajectories $\mathbf{\Pi}_1$ and $\mathbf{\Pi}_3$ in Table 5.6, where the difference in waypoints count is only two waypoints. An even more noticeable increase in computational time can be seen in the case of trajectory $\mathbf{\Pi}_2$ with 19 waypoints.

The tested trajectories with a larger number of waypoints resulted in one-order longer computation times when our method was used. A slower convergence of the optimization is also a contributing factor; however, the difference in the computational time is not as drastic as in the case of the CR approach. The worst-case number of operations needed to update a single trajectory segment is equal for all trajectory segments, which means that the worst-case computational time increases linearly with the number of waypoints.

However, for scenarios similar to online trajectory replanning, i.e., trajectories with a small number of waypoints, both presented approaches using our method for VO are applicable. This follows from the results for trajectories $\mathbf{\Pi}_1$ and $\mathbf{\Pi}_3$. In the case of Approach 1, the computational time for trajectories with $n \leq 6$ waypoints lies below 0.1 ms. Even though these values were measured on a desktop computer and the onboard computers on a UAV are generally less powerful, this approach is suitable for time-critical applications, where sub-optimal trajectory durations are an acceptable tradeoff for short computation times. In the case of Approach 2, the computational times increase to several milliseconds, which is still acceptable for many applications where the trajectory (re-)planning is usually run with a frequency greater or equal to 100 Hz. Both approaches can also be combined depending on the concrete application, where for example new longer flights can be computed using Approach 2 to obtain the minim trajectory duration achievable using our method, and Approach 1 can be used for time-critical trajectory replanning in collision avoidance of disturbance compensation.

5.4 Collision-free trajectory generation

To test the performance of our Collision-free Trajectory Generaiton (CFTG) described in Algorithm 3, we have computed three distinct paths using the CTopPRM [9] path planning algorithm in a forest-like map displayed in Figure 4.6. Using these paths, we have computed a minimum-time collision-free trajectory using our CFTG algorithm described in Section 4.7 and a Point-mass Trajectory Search (PTS) introduced in [18]. In both cases, the point-mass model trajectory alone was computed using our Approach 1 for velocity optimization with thrust decomposition described in Section 4.5, so we are comparing only the approaches to trajectory generation and selection based on a set of possible paths. This approach has been selected to simulate the online replanning scenario, where the trajectory computation must be completed in short time cycles.

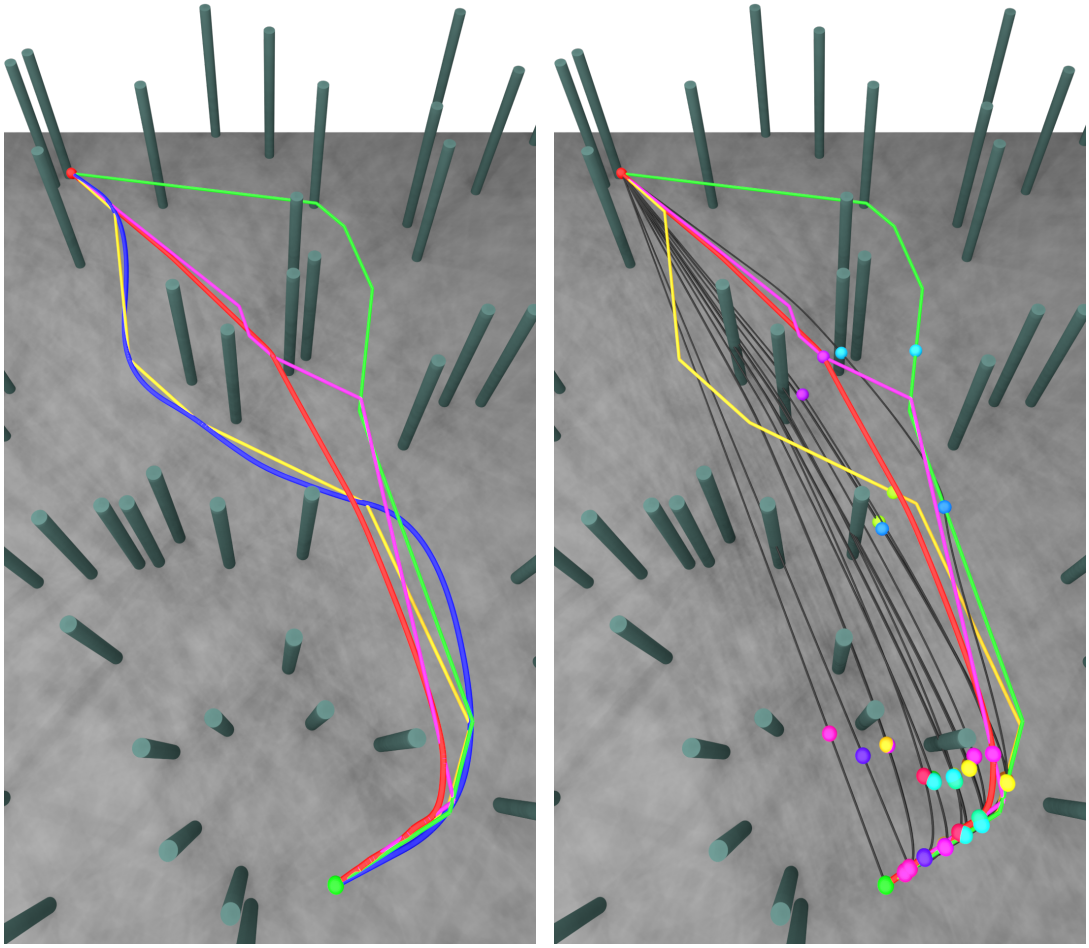
Table 5.7: Computational times and trajectory durations for resulting trajectories computed using our Collision-free Trajectory Generation (CFTG) method defined in Algorithm 3 and Point-mass Trajectory Search (PTS) [18] method. The values correspond to the case displayed in Figure 5.7 where the distinct paths were computed using CTopPRM [9] path planning algorithm.

	Computational Time [ms]	Trajectory Duration [s]
CFTG	**0.8685**	**2.1048**
PTS [18]	1.6372	2.27801

Table 5.7 shows the resulting trajectory durations and computational times averaged over 100 runs. The final trajectories are displayed in Figure 5.7a using blue and red lines for our CFTG method and the PTS method, respectively. Comparing the computational times in Table 5.7, our proposed method is nearly two times faster than the PTS method in the tested scenario. This is due to the issue with the PTS method we have described in Section 4.7. As the closest point to the first collision point of a naive trajectory is added to the new path used in the next iteration, a situation can occur, where many waypoints need to be added before the final collision-free trajectory is found. This situation is shown in Figure 5.7b, where the naive paths (black lines) are displayed together with collision points and waypoints that are consequently added to the next naive trajectory, both displayed with points in matching colors.

The many added waypoints also caused the final trajectory computed with the PTS method to have a longer duration compared to our method. As the trajectory connects waypoints that are close to each other, the velocity of the motion must be kept small to allow short maneuvers joining those waypoints. If it was not for this issue at the beginning of the final trajectory, the resulting trajectory duration would be arguably shorter than the one found with our method as the rest of the trajectory marked with a red line in Figure 5.7a has fewer bends and a higher velocity can thus be used. It can be expected that the PTS can lead to faster trajectories and shorter computational times in some cases. This, however, is not guaranteed and the method can even fail to compute a feasible trajectory in some cases, one of which was discussed in Section 4.7. Therefore we believe that our CFTG method is more suitable for general applications.

The effect of path curvature on the final trajectory duration can also be seen in Figure 5.7 where the resulting trajectory generated using our approach is displayed. The final trajectory (blue) is guided using the yellow path although a shorter trajectory duration would



(a) Comparison of resulting trajectories. (b) Visualization of naive trajectories.

Figure 5.7: Visualization of computed paths (yellow, pink, and green lines) and minimum time trajectories computed using Algorithm 3 (blue line) and Point-mass Trajectory Search (PTS) method introduced in [18] (red line). The start and end configurations are marked with green and red points, respectively. Figure (a) displays the results for both compared methods. In (b), the whole trajectory search process of the PTS is shown, where the naive trajectories (black lines) are displayed together with first collisions and consequently added points in matching colors (marked with colored points).

be achievable using a shortened pink path, which is followed by the PTS method (red). This showcases the dependence of the resulting trajectory computed using our approach on the distinct paths returned by the path planning algorithm. A more extensive path shortening could result in shorter durations; however, this task goes beyond the scope of this work.

The resulting computational times of our approach to collision-free trajectory generation also show that the method is suitable for online trajectory replanning. Even though the trajectory generation was run on a desktop machine CPU and an onboard computer of a UAV is generally less powerful, with computation time under one millisecond for three distinct paths with a waypoint count of 7 to 8 waypoints, there is a lot of room for slower computation as for most UAV applications, a trajectory (re-)planning loop is usually executed with frequencies under 100 Hz. The worst-case computation time will increase for a larger number of distinct paths returned by the path planner. Nonetheless, a limited number of these paths can, be selected for the final trajectory computation, for example based on the path length.

Chapter 6

Conclusion

In this thesis, we have introduced a novel approach to collision-free trajectory planning for UAVs, where approximate minimum-time trajectories are computed using a point-mass model motion primitive. We use topologically distinct paths to guide the trajectory planning. The point-mass model approximation was selected to reduce the computational burden of trajectory planning. The selection was also motivated by the fact that differentiable trajectories infeasible by the UAV can be tracked by modern control methods in a near-optimal way [8]. We have defined a method for axis synchronization of a limited acceleration multidimensional point-mass model trajectory, where the single-axis trajectories are synchronized to the shortest feasible duration possible using acceleration scaling. It has been shown that the multidimensional point-mass model trajectory can be computed in a closed form. We have introduced a gradient-method-based algorithm for velocity optimization of a multi-waypoint trajectory, where the velocities in the via-waypoints are iteratively adjusted to obtain the shortest trajectory duration possible. The velocity updates are handled per-axis and subjected to bounds we have derived to guarantee feasibility in the next iteration; constant per-axis acceleration bounds are assumed. To account for gravity and underactuated multirotor design, where per-axis accelerations are bounded by the multirotor's limited collective trust, we propose an iterative method for limited thrust decomposition. Further, several different approaches for incorporation of the limited thrust decomposition into the velocity optimization process were introduced. Based on all these building blocks, an algorithm for collision-free trajectory generation was presented, where topologically distinct paths generated by the CTopPRM [9] path planner were used to guide the trajectory computation. Given a set of possible paths, the collision-free trajectory generation algorithm finds the minimum-time trajectory guided by one of the paths.

Given a set of testing paths containing four to nineteen waypoints, we have tested the proposed method for multi-waypoint point-mass model trajectory generation using different hyperparameter values. Both computational time and the duration of the resulting trajectory were evaluated to test the proposed approach for online minimum-time trajectory replanning. It has been observed that due to the per-axis velocity optimization, the optimization process is sensitive to the hyperparameter settings. We were also unable to derive a general rule for parameter selection to obtain the shortest trajectory duration possible. In most cases, the resulting trajectory represents a tradeoff between computational time and trajectory duration.

The convergence of the limited thrust decomposition method was also tested, where it was shown that the proposed algorithm converges within a few iterations. An average computational time of a single iteration was measured to be 0.2321 μs, which makes the method suitable for real-time trajectory generation. Three different approaches to incorporating the thrust decomposition method into the velocity optimization algorithm were tested. The first one, composed of a single velocity optimization run while assuming per-axis acceleration limits and subsequent re-computation using the thrust decomposition algorithm, resulted in short

computational times but increased trajectory durations compared to the other methods, which makes it suitable for time-critical applications. The second approach added a second velocity optimization run to the first approach, where this time the trust decomposition algorithm was used for all trajectory computations. This resulted in the shortest trajectory durations for most tested cases but at the cost of increased computational times. However, the increase can be reduced by terminating the second velocity run after a few iterations to quickly improve the trajectories computed using the first approach. The last approach consisting only of the velocity optimization with incorporated thrust decomposition led to unpredictable behavior.

The performance of the proposed algorithm for multi-waypoint trajectory planning was further compared with a state-of-the-art sampling-based method [7]. Our approach outperformed the sampling-based method in terms of computational times, where the resulting times were 20 to 100 times shorter, depending on the used approach for thrust decomposition. Moreover, the difference in computational times between trajectories containing a small and a large number of waypoints, respectively, was not as substantial as in the case of [7]. However, due to the per-axis velocity optimization approach of our method, the resulting trajectory durations were longer compared to the state-of-the-art method in several cases. The computational times of our point-mass model trajectory planning method ranged from 50 μs to 3 ms (depending on the approach and number of waypoints) for trajectories containing less than twenty waypoints, which makes the proposed method suitable for real-time trajectory replanning.

Finally, we have tested the proposed algorithm for collision-free trajectory replanning in a forest-like environment and compared it with a state-of-the-art method [18]. In a scenario where three distinct paths with waypoint counts ranging from seven to eight waypoints were found, our approach was able to generate a collision-free trajectory in less than one millisecond, which is half the time required for the state-of-the-art method. The final trajectory duration was also shorter than in the case of method [18]. However, as the trajectories generated by our method are guided by the planned paths, it has been observed that the performance of the proposed method depends on the post-processing (path shortening) of initial paths by the path planner.

The presented approach to collision-free trajectory replanning offers several possibilities for further development. Enhancements in velocity initialization or velocity update method could eliminate the described issue where the per-axis optimization approach resulted in omitting certain components of velocities that corresponded to single-axis synchronization trajectories. This leads to a longer trajectory duration from the global point of view. The concept of velocity optimization could be also extended to the limited jerk point-mass model in future works, as that would approximate the full dynamic model of a multirotor UAV more accurately.

References

[1] A. Sabirova and R. Fedorenko, "Drone cinematography system design and new guideline model for scene objects interaction," *2020 International Conference Nonlinearity, Information and Robotics (NIR)*, pp. 1–6, 2020-12-3. DOI: 10.1109/NIR50484.2020.9290236. [Online]. Available: https://ieeexplore.ieee.org/document/9290236/.

[2] H. Ren, Y. Zhao, W. Xiao, and Z. Hu, "A review of uav monitoring in mining areas, Current status and future perspectives," vol. 6, no. 3, pp. 320–333, 2019, ISSN: 2095-8293. DOI: 10.1007/s40789-019-00264-5. [Online]. Available: http://link.springer.com/10.1007/s40789-019-00264-5.

[3] M. Atif, R. Ahmad, W. Ahmad, L. Zhao, and J. J. P. C. Rodrigues, "Uav-assisted wireless localization for search and rescue," *IEEE Systems Journal*, vol. 15, no. 3, pp. 3261–3272, 2021, ISSN: 1932-8184. DOI: 10.1109/JSYST.2020.3041573. [Online]. Available: https://ieeexplore.ieee.org/document/9345802/.

[4] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," *Robotics Research*, pp. 649–666, 2016. DOI: 10.1007/978-3-319-28872-7_37. [Online]. Available: http://link.springer.com/10.1007/978-3-319-28872-7_37.

[5] B. Zhou, J. Pan, F. Gao, and S. Shen, "Raptor, Robust and perception-aware trajectory replanning for quadrotor fast flight," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 1992–2009, 2021, ISSN: 1552-3098. DOI: 10.1109/TRO.2021.3071527. [Online]. Available: https://ieeexplore.ieee.org/document/9422918/.

[6] P. Foehn, A. Romero, and D. Scaramuzza, "Time-optimal planning for quadrotor waypoint flight," *Science Robotics*, vol. 6, no. 56, 2021-07-21, ISSN: 2470-9476. DOI: 10.1126/scirobotics.abh1221. [Online]. Available: https://www.science.org/doi/10.1126/scirobotics.abh1221.

[7] A. Romero, R. Penicka, and D. Scaramuzza, "Time-optimal online replanning for agile quadrotor flight," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 7730–7737, 2022, ISSN: 2377-3766. DOI: 10.1109/LRA.2022.3185772. [Online]. Available: https://ieeexplore.ieee.org/document/9805699/.

[8] A. Romero, S. Sun, P. Foehn, and D. Scaramuzza, "Model predictive contouring control for time-optimal quadrotor flight," *IEEE Transactions on Robotics*, vol. 38, no. 6, pp. 3340–3356, 2022, ISSN: 1552-3098. DOI: 10.1109/TRO.2022.3173711. [Online]. Available: https://ieeexplore.ieee.org/document/9802523/.

[9] M. Novosad, R. Penicka, and V. Vonasek, "Ctopprm: Clustering topological prm for planning multiple distinct paths in 3d environments," 2023. DOI: arXiv:2305.13969. [Online]. Available: http://arxiv.org/abs/2305.13969.

[10] P. Corke, *Robotics, vision and control, fundamental algorithms in MATLAB*, 2nd ed. Berlin: Springer, 2013, ISBN: 978-3-642-20143-1.

[11] S. M. LaValle, *Planning algorithms*. New York: Cambridge University Press, 2006, ISBN: 05-218-6205-1.

[12] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, "Voxblox, Incremental 3d euclidean signed distance fields for on-board mav planning," *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1366–1373, 2017. DOI: 10.1109/IROS.2017.8202315. [Online]. Available: http://ieeexplore.ieee.org/document/8202315/.

[13] B. Donald, P. Xavier, J. Canny, and J. Reif, "Kinodynamic motion planning," *Journal of the ACM*, vol. 40, no. 5, pp. 1048–1066, 1993, ISSN: 0004-5411. DOI: 10.1145/174147.174150. [Online]. Available: https://dl.acm.org/doi/10.1145/174147.174150.

[14] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge: Cambridge University Press, 2004, ISBN: 9780521833783. DOI: 10.1017/CBO9780511804441. [Online]. Available: https://www.cambridge.org/core/books/convex-optimization/17D2FAA54F641A2F62C7CCD01DFA97C4.

[15] S. LaValle and J. Kuffner, "Randomized kinodynamic planning," *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, pp. 473–479, 1999. DOI: 10.1109/ROBOT.1999.770022. [Online]. Available: http://ieeexplore.ieee.org/document/770022/.

[16] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, ISSN: 1042296X. DOI: 10.1109/70.508439. [Online]. Available: http://ieeexplore.ieee.org/document/508439/.

[17] T. Siméon, J.-P. Laumond, and C. Nissoux, "Visibility-based probabilistic roadmaps for motion planning," *Advanced Robotics*, vol. 14, no. 6, pp. 477–493, 2012-04-02, ISSN: 0169-1864. DOI: 10.1163/156855300741960. [Online]. Available: https://www.tandfonline.com/doi/full/10.1163/156855300741960.

[18] R. Penicka and D. Scaramuzza, "Minimum-time quadrotor waypoint flight in cluttered environments," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 5719–5726, 2022, ISSN: 2377-3766. DOI: 10.1109/LRA.2022.3154013. [Online]. Available: https://ieeexplore.ieee.org/document/9721033/.

[19] M. Aria, "Optimal path planning using informed probabilistic road map algorithm," *Journal of Engineering Research*, 2021-12-14, ISSN: 23071877. DOI: arXiv:2007.03465v. [Online]. Available: https://kuwaitjournals.org/jer/index.php/JER/article/view/16105.

[20] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," *2011 IEEE International Conference on Robotics and Automation*, pp. 2520–2525, 2011. DOI: 10.1109/ICRA.2011.5980409. [Online]. Available: http://ieeexplore.ieee.org/document/5980409/.

[21] B. Zhou, F. Gao, L. Wang, C. Liu, and S. Shen, "Robust and efficient quadrotor trajectory generation for fast autonomous flight," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3529–3536, 2019, ISSN: 2377-3766. DOI: 10.1109/LRA.2019.2927938. [Online]. Available: https://ieeexplore.ieee.org/document/8758904/.

[22] R. Allen and M. Pavone, "A real-time framework for kinodynamic planning with application to quadrotor obstacle avoidance," *AIAA Guidance, Navigation, and Control Conference*, pp. –, 2016-01-04. DOI: 10.2514/6.2016-1374. [Online]. Available: https://arc.aiaa.org/doi/10.2514/6.2016-1374.

[23] P. Foehn, D. Brescianini, E. Kaufmann, *et al.*, "Alphapilot, Autonomous drone racing," *Autonomous Robots*, vol. 46, no. 1, pp. 307–320, 2022, ISSN: 0929-5593. DOI: 10.1007/s10514-021-10011-y. [Online]. Available: https://link.springer.com/10.1007/s10514-021-10011-y.

[24] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 3rd ed. Athena Scientific, 2007, ISBN: 1-886529-43-4.

[25] M. Beul and S. Behnke, "Analytical time-optimal trajectory generation and control for multirotors," *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 87–96, 2016. DOI: 10.1109/ICUAS.2016.7502532. [Online]. Available: http://ieeexplore.ieee.org/document/7502532/.

[26] M. Beul and S. Behnke, "Fast full state trajectory generation for multirotors," *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 408–416, 2017. DOI: 10.1109/ICUAS.2017.7991304. [Online]. Available: http://ieeexplore.ieee.org/document/7991304/.

[27] F. Meyer and K. Glock, "Kinematic orienteering problem with time-optimal trajectories for multirotor uavs," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 11 402–11 409, 2022, ISSN: 2377-3766. DOI: 10.1109/LRA.2022.3194688. [Online]. Available: https://ieeexplore.ieee.org/document/9844257/.

[28] L. Janson, E. Schmerling, A. Clark, and M. Pavone, "Fast marching tree, A fast marching sampling-based method for optimal motion planning in many dimensions," *The International Journal of Robotics Research*, vol. 34, no. 7, pp. 883–921, 2015, ISSN: 0278-3649. DOI: 10.1177/0278364915577958. [Online]. Available: http://journals.sagepub.com/doi/10.1177/0278364915577958.

Appendix A

Supplementary Definitions and Results

A.1 Point-Mass Model Trajectory Acceleration Scaling Solutions

Solutions to the equations (4.8) for the case $v_0 \neq v_2$:

$$
\begin{aligned}
t_1 &= \left\{ \begin{array}{l} \frac{a_2\,p_0 - a_1\,p_0 + a_1\,p_2 - a_2\,p_2 + \sigma_1 - T_{\mathrm{sync}}\,a_1\,v_2 + T_{\mathrm{sync}}\,a_2\,v_2}{(a_1 - a_2)\,(v_0 - v_2)} \\ -\frac{a_1\,p_0 - a_2\,p_0 - a_1\,p_2 + a_2\,p_2 + \sigma_1 + T_{\mathrm{sync}}\,a_1\,v_2 - T_{\mathrm{sync}}\,a_2\,v_2}{(a_1 - a_2)\,(v_0 - v_2)} \end{array} \right\}, \\
t_2 &= \left\{ \begin{array}{l} -\frac{a_2\,p_0 - a_1\,p_0 + a_1\,p_2 - a_2\,p_2 + \sigma_1 - T_{\mathrm{sync}}\,a_1\,v_0 + T_{\mathrm{sync}}\,a_2\,v_0}{(a_1 - a_2)\,(v_0 - v_2)} \\ \frac{a_1\,p_0 - a_2\,p_0 - a_1\,p_2 + a_2\,p_2 + \sigma_1 + T_{\mathrm{sync}}\,a_1\,v_0 - T_{\mathrm{sync}}\,a_2\,v_0}{(a_1 - a_2)\,(v_0 - v_2)} \end{array} \right\}, \\
\gamma &= \left\{ \begin{array}{l} \frac{a_1\,p_0 - a_2\,p_0 - a_1\,p_2 + a_2\,p_2 + \sigma_1 - T_{\mathrm{sync}}\,a_2\,v_0 + T_{\mathrm{sync}}\,a_1\,v_2}{{T_{\mathrm{sync}}}^2\,a_1\,a_2} \\ -\frac{a_2\,p_0 - a_1\,p_0 + a_1\,p_2 - a_2\,p_2 + \sigma_1 + T_{\mathrm{sync}}\,a_2\,v_0 - T_{\mathrm{sync}}\,a_1\,v_2}{{T_{\mathrm{sync}}}^2\,a_1\,a_2} \end{array} \right\}, \\
\sigma_1 &= a_1\,a_2 \sqrt{\frac{(a_1 - a_2)\left(a_1 {p_0}^2 - a_2 {p_0}^2 + a_1 {p_2}^2 - a_2 {p_2}^2 - {T_{\mathrm{sync}}}^2 a_2 {v_0}^2 + {T_{\mathrm{sync}}}^2 a_1 {v_2}^2 - 2a_1 p_0 p_2 + 2a_2 p_0 p_2 - 2T_{\mathrm{sync}} a_2 p_0 v_0 + 2T_{\mathrm{sync}} a_1 p_0 v_2 + 2T_{\mathrm{sync}} a_2 p_2 v_0 - 2T_{\mathrm{sync}} a_1 p_2 v_2\right)}{{a_1}^2 {a_2}^2}}, \\
p_1 &= p_0 + v_0 t_1 + \frac{1}{2}\gamma a_1 t_1^2, \\
v_1 &= v_0 + \gamma a_1 t_1.
\end{aligned}
\tag{A.1}
$$

Solutions to the equations (4.8) for the case $v_0 = v_2$:

$$
\begin{aligned}
t_1 &= -\frac{T_{\mathrm{sync}}\,a_2}{a_1 - a_2}, \\
t_2 &= \frac{T_{\mathrm{sync}}\,a_1}{a_1 - a_2}, \\
\gamma &= \frac{(2\,a_1 - 2\,a_2)\;(p_0 - p_2 + T_{\mathrm{sync}}\,v_0)}{{T_{\mathrm{sync}}}^2\,a_1\,a_2}, \\
p_1 &= p_0 + v_0 t_1 + \frac{1}{2}\gamma a_1 t_1^2, \\
v_1 &= v_0 + \gamma a_1 t_1.
\end{aligned}
\tag{A.2}
$$

A.2 Definition of Waypoints for the Testing Paths

The position of all waypoints of the paths P_i, $i = 1, ..., 4$, used for the testing of our method are listed in the tables below.

Table A.1: Path P_1 waypoint positions.

Waypoint Id [-]	x [m]	y [m]	z [m]
1	7.0000	6.3400	0.7570
2	9.0900	6.2600	1.0800
3	9.2700	-3.4600	1.1700
4	-4.7500	-6.1200	2.8100

Table A.2: Path P_2 waypoint positions.

Waypoint Id [-]	x [m]	y [m]	z [m]
1	-5.0000	4.5000	1.2000
2	-0.9000	-1.2700	3.4800
3	9.0900	6.2600	1.0800
4	9.2700	-3.4600	1.1700
5	-4.0000	-6.2500	3.4000
6	-4.4800	-5.9400	1.0500
7	4.4500	-0.8000	1.0900
8	-2.6500	6.5100	1.3000
9	-0.9000	-1.2700	3.4800
10	9.0900	6.2600	1.0800
11	9.2700	-3.4600	1.1700
12	-4.0000	-6.2500	3.4000
13	-4.4800	-5.9400	1.0500
14	4.4500	-0.8000	1.0900
15	-2.6500	6.5100	1.3000
16	-0.9000	-1.2700	3.4800
17	9.0900	6.2600	1.0800
18	9.2700	-3.4600	1.1700
19	-2.5000	-6.0000	4.0000

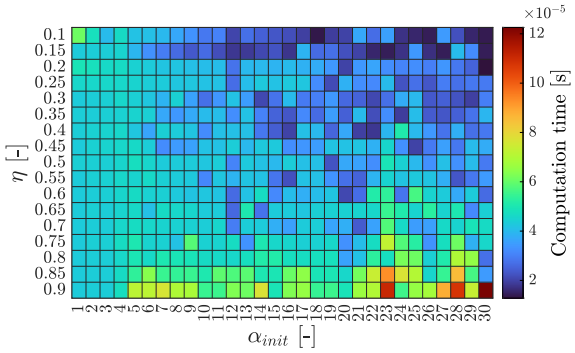
Table A.3: Path P_3 waypoint positions.

Waypoint Id [-]	x [m]	y [m]	z [m]
1	5.0000	-4.0000	1.3000
2	4.6557	-2.5236	1.3069
3	4.7364	-2.0418	1.3258
4	4.9264	0.4307	1.4008
5	3.7276	3.2632	1.4214
6	0.0000	7.5000	1.3000

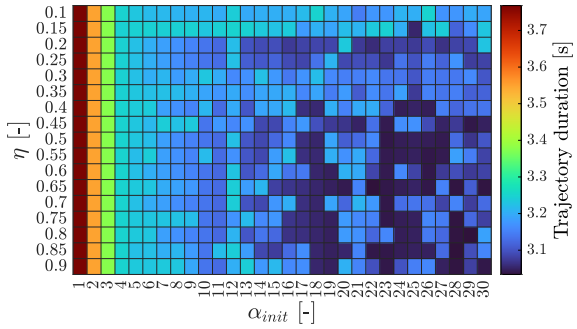
Table A.4: Path P_4 waypoint positions.

Waypoint Id [-]	x [m]	y [m]	z [m]
1	5.0000	-4.0000	1.3000
2	4.7282	-2.5568	1.7704
3	4.9787	-2.0328	1.8426
4	4.4680	-0.8303	1.8237
5	2.8401	-0.2817	1.5554
6	1.8756	-0.1497	0.7626
7	2.0409	-0.0806	1.1545
8	2.3520	0.1991	1.3286
9	2.5729	0.5183	1.5392
10	2.7684	0.8974	1.5380
11	0.0000	7.5000	1.3000

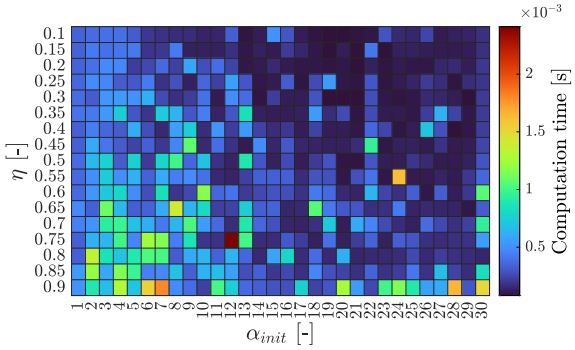
A.3 Velocity Optimization Parameter Grid-Search Results



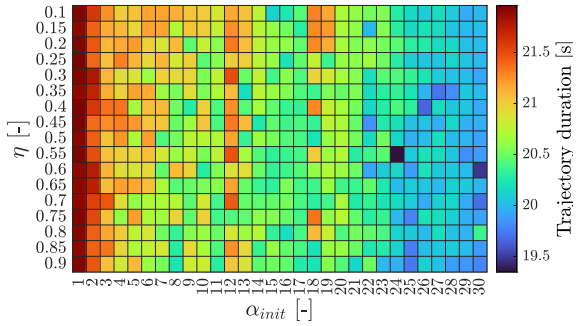
(a) Trajectory $\mathbf{\Pi}_1$ computational times.



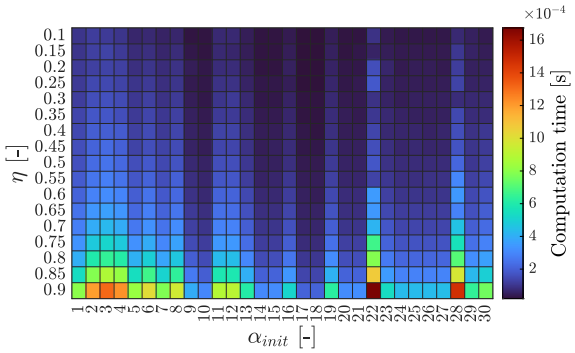
(b) Trajectory $\mathbf{\Pi}_1$ durations.



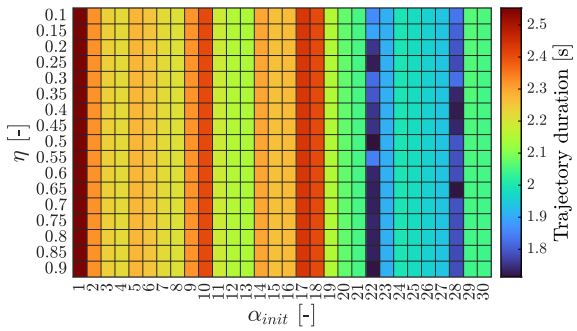
(c) Trajectory $\mathbf{\Pi}_2$ computational times.



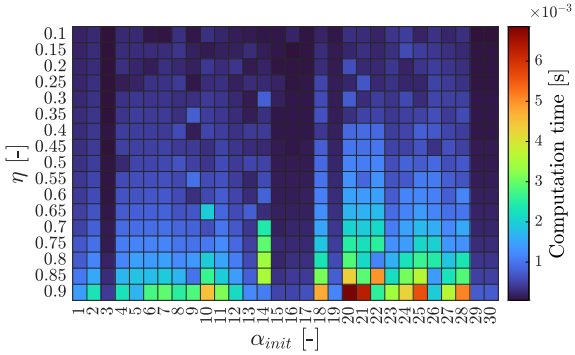
(d) Trajectory $\mathbf{\Pi}_2$ durations.



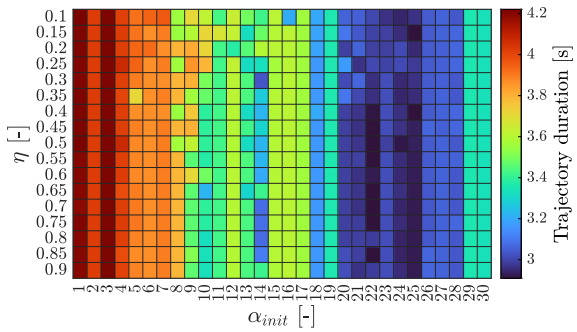
(e) Trajectory $\mathbf{\Pi}_3$ computational times.



(f) Trajectory $\mathbf{\Pi}_3$ durations.



(g) Trajectory $\mathbf{\Pi}_2$ computational times.



(h) Trajectory $\mathbf{\Pi}_2$ trajectory durations.

Figure A.1: Heat maps of computational times and trajectory durations for different values of Algorithm 1 hyperparameters, namely step length α_{init} and step length reduction factor η. The trajectories $\mathbf{\Pi}_1, \ldots, \mathbf{\Pi}_4$ are computed from the corresponding testing paths $P_1, \ldots, P_4$.

Appendix B

Content of the Attached CD

The content of the attached CD is listed in Table B.1. An electronic version of this thesis can be found in the file `Masters_Thesis_Krystof_Teissing.pdf`. The file `main.cpp` includes a function demonstrating the implemented algorithms contained in the rest of the source codes; the results can be visualized by running the `plot_results.py` script.

Table B.1: Directories and files on the attached CD.

```
/source_code/
    /conf
    /include/
        cftg.hpp
        common.hpp
        heap.hpp
        pmm_mg_trajectory3d.hpp
        pmm_trajectory.hpp
        pmm_trajectory3d.hpp
    /MK
    /scripts/
        plot_results.py
    /src/
        main.cpp
        cftg.cpp
        common.cpp
        heap.cpp
        pmm_mg_trajectory3d.cpp
        pmm_trajectory.cpp
        pmm_trajectory3d.cpp
    /Makefile
/thesis/
    Masters_Thesis_Krystof_Teissing.pdf
```