Inertia-based Routing

New Approaches using Time-Optimal Trajectory Generation for Multirotor UAVs as an Example

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

von der KIT-Fakultät für Wirtschaftswissenschaften
des Karlsruher Instituts für Technologie (KIT)

genehmigte

DISSERTATION

von

Fabian Meyer, M.Sc.

Hauptreferent:	Prof. Dr. Stefan Nickel
Korreferent:	Prof. Dr. rer. nat. habil. Armin Fügenschuh
Tag der Prüfung	12.06.2024

Abstract

Unmanned aerial vehicles (UAVs) are, from a physical point of view, inherently inert. This means that they require an acceleration force to change their direction of movement and/or velocity magnitude. Until now, however, these physical properties associated with inertia have not been taken into account in the planning of a flight mission for a multirotor UAV in sufficient detail. In the best case, this leads to the problem that the resulting flight mission is not as efficient as the kinematic characteristics would allow. In the worst case, the calculated mission plan requires physical properties such as maneuverability or acceleration power that the considered multirotor UAV does not provide. Consequently, the determined reference trajectory of the planned mission cannot be tracked by the UAV. This results in spatial and temporal errors compared to what was planned which might lead to a mission execution with a significantly lower quality than expected.

In this work, we give deep insights into the problem of inertia-based route planning with multirotor UAVs, a special UAV type, as an example. To model the real physical capabilities of multirotor UAVs, we develop a new analytical approach for time-optimal trajectory planning for point-masses with constrained velocity and acceleration. The trajectories yielded by our new method have proven to be sufficiently precise in modeling a multirotor UAV's full physical capabilities. Further, since our approach is based on analytical solutions in closed form, it is computationally extremely cheap.

Next, we introduce the kinematic traveling salesman problem (KTSP) and the kinematic orienteering problem (KOP), which are based on the assumption that each waypoint of a flight mission can be traversed with different heading angles and velocities. For each possibility to travel between a waypoint pair, we utilize our time-optimal trajectory planning approach introduced above to describe the associated motion. We develop a mathematical model for both, the KTSP and the KOP, and hence can solve related problems with a commercial general-purpose solver to global optimality. Since both problems are combinatorial and classified as NP-hard, we additionally develop heuristic algorithms to solve both problem classes in a short time with sufficient solution quality. The presented results show that our research on inertia-based route planning problems significantly improves the achieved quality of UAV mission planning compared to state-of-the-art approaches.

Contents

1. Introduction

In the last decade, unmanned aerial vehicle (UAV) technology has been steadily gaining momentum. With technological advancements, UAVs are proving to be extremely useful in a variety of applications. These include monitoring and inspection of large infrastructures and energy facilities such as offshore wind farms, power lines, roads, oil and gas pipelines [1, 2, 3, 4, 5], monitoring of cultivated land and forests [5, 6, 7], in the mining industry [8], layout planning and digital reconstruction in construction [9], for damage assessment after disaster events [10, 11], and many more [12, 5, 13].

In general, UAVs can be classified by a set of properties. These include the weight, range, maximum altitude, maximum velocity, and engine type. However, one of the most important properties is the horizontal take-off landing (HTOL) and the vertical take-off landing (VTOL) property. HTOL systems such as fixed-wing UAVs usually fly at higher velocities than VTOL systems and hence suit better for long-range missions. However, VTOL systems such as singlerotors (e.g. unmanned helicopters), quadrotors (see Figure 1.1), or, in general, multirotor UAVs can hover efficiently and have higher maneuverability [14]. These characteristics make them perfect for airborne surveying of areas. Both systems have their advantages, which is why hybrid designs are developed, too. For further insights into the design of UAVs, we refer the reader to [14]. For the remainder of this work, we focus on the flight mission planning of multirotor UAVs.

1.1. Surveillance and Data Collection

Within this work, we specifically focus on flight mission planning for multirotor UAVs in surveillance and data collection applications, which is one of the most important use cases for UAVs [5]. The focus lies especially on the flight planning for multi-view stereo

Figure 1.1.: Example of a multirotor UAV. Source: https://www.pixabay.com/.

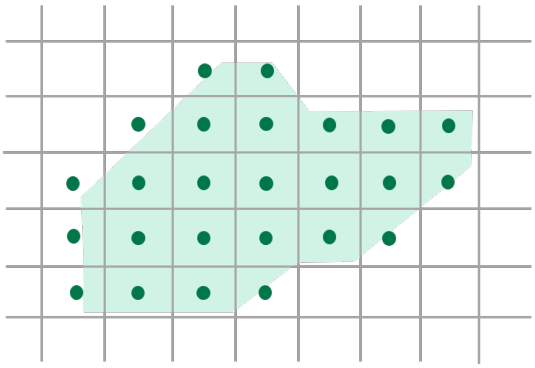


Figure 1.2.: Example for viewpoints to cover the ROI.

3D reconstruction. The goal of this application is to obtain a three-dimensional virtual point cloud representation of the environment. This is obtained by capturing multiple images from different perspectives and combining them with stereo vision methods. As an example, these point clouds can be used to detect damages to buildings after earthquakes (see [15]).

The latest approaches in 3D reconstruction rely on the identification of proper viewpoints. These are spatial locations at which images must be recorded to achieve a specified quality in the 3D reconstruction of the scene. In the literature, the approaches to determine these viewpoints are manifold. One possibility is to cover the scene with a grid-like structure of tiles of equal size (see [16, 17, 18]). If a tile in the grid is classified as a region of interest (ROI), then the center of the tile is defined as a viewpoint. An example is given in Figure 1.2. Here the green dots represent the identified viewpoints to reconstruct the green-shaded shape. The size of the tiles directly results from the parameters of the used sensor, such as the aperture angle of the camera, and the desired resolution of the sensor footprint. For more insights in this regard, we refer the reader to [17, 18]. Apart from this grid-based approach, there also exist more sophisticated approaches to determine the ensemble of viewpoints to reconstruct the scene. However, this is not the focus of this work and we refer the reader to [19, 20] as they provide a very concise review of the latest approaches for UAV viewpoint planning. For our means, these approaches yield some set of viewpoints that allows for a proper reconstruction of the scene. The leading question of the work at hand is how to visit these viewpoints as best as possible respecting the UAV's physical capabilities and restrictions. In this context of this introduction, the terminology 'best' refers to a minimum time. This question is not easily answered as we show in Section 1.1.1. Aiming at real-world applications, we explicitly allow to traverse viewpoints with a velocity higher than 0 m/s to fully exploit possible optimization potential.

1.1.1. The Impact of Inertia - A Motivating Example

When solving the task of finding a minimum-time trajectory through a given set of viewpoints, there arises a set of problems. To illustrate these problems, we investigate the set of 25 viewpoints as given in Figure 1.2 that covers a given ROI, and solve the resulting

mission planning problem with approaches from the current state-of-the-art. Note that this section mostly serves as a motivation to illuminate current research gaps. The used approaches from the state-of-the-art are introduced and discussed in depth in Section 3.1.

We assume that the flight mission is to be planned for a single multirotor UAV, since the viewpoints are rather close and high agility is required which multirotor UAVs provide [5]. The rotor thrust of the UAV allows a maximum acceleration of $1.5\,\mathrm{m/s^2}$ in the horizontal plane. Further, the velocity of the UAV is limited to 3 m/s, since otherwise, the flight would be too aggressive from which the data quality of the recorded images would suffer (see e.g. [16]).

A common approach to answering the question of how to fly through these viewpoints is to assume that the UAV constantly flies at its maximum velocity and can make on-spot turns (e.g. see [21]). The latter implies neglecting the maximum acceleration property. With these assumptions, the resulting problem reduces to the classical traveling salesman problem (TSP) in the case that the start and end points of the mission are the same. The objective of the TSP is to find a Hamiltonian cycle through a set of locations such that each location is visited exactly once except for the start location and a given cost functional - in the present case the required flight time - is minimized. The TSP is a well-studied problem in the literature and its solution can be used to generate a reference trajectory which is a function of the UAV's spatial position over time. Using this reference as a guidance, the UAV knows exactly where to be at each point in time. For example, the point in time when the UAV is required to take an image is specified as the point in time at which the reference traverses the associated viewpoint.

In Figure 1.3, we show the solution to the above-stated problem. The direction of the cycle is indicated by the black arrow. The resulting reference trajectory is given as a red curve in the lefthand plot and the entire mission is determined to last 25.41 s. Next, we simulate a UAV trying to follow this reference trajectory in space and time (see blue dashed curve in the lefthand plot). Our simulation is based on the assumption that a UAV moving in a horizontal plane can sufficiently be modeled as a point-mass with a constrained maximum velocity and acceleration. To track the reference trajectory, we apply the model predictive controller (MPC) presented in Appendix B.1 which works as follows. Based on a time discretization, our MPC determines the acceleration value applied to each time step such that the deviation to the reference trajectory at the discrete timestamps is minimized and the constraints on the maximum velocity and acceleration norms are not violated. As a performance indicator for this and all subsequent experiments of this section, we define the distance between a certain waypoint and the simulated UAV position at the exact timestamps when the reference trajectory traverses that waypoint (see red curve in the righthand plot). Hence, the values in the righthand plot represent the distances of the actual recording locations (blue squares in the lefthand plot) and the associated viewpoints (red squares in the lefthand plot). We use this metric because it reflects whether the images are taken at the pre-calculated positions which is a good indicator of the quality of the resulting data ensemble.

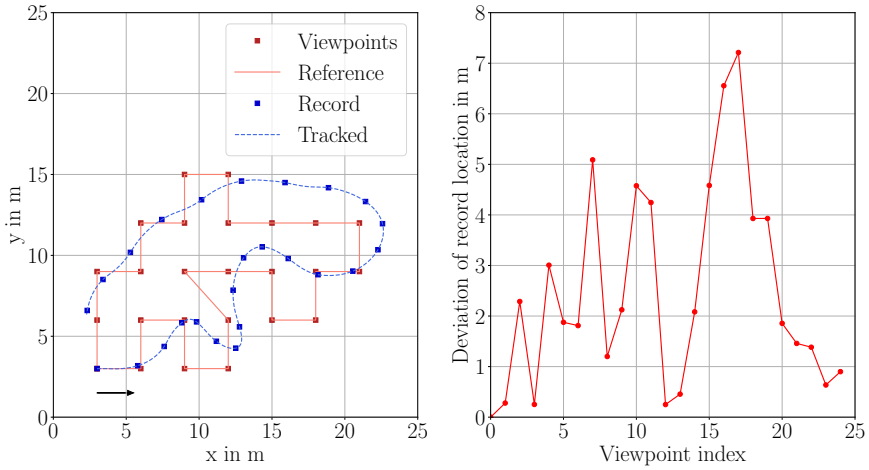


Figure 1.3.: Example mission with 25 waypoints for a multirotor UAV with a maximum velocity of v_{max} = 3 m/s and a maximum acceleration of a_{max} = 1.5 m/s^2. The reference trajectory is calculated under the assumption that the UAV can fly constantly at v_{max} and can make on-spot turns. The righthand plot shows the distance between the planned viewpoint and the actual position at the recording timestamps. The planned mission duration is 25.41 s.

As can be seen in Figure 1.3, neglecting the acceleration comes with a significant deviation between the reference and the actual UAV trajectory at the recording times of more than 7 m. Conducting a data collection flight and accepting deviations of this extent in the recording location risks producing data of low quality. As we will see later, the reason for these large deviations is that the acceleration property of a UAV cannot simply be neglected. This observation is further emphasized by our next experiment. Here, we slow down the constant velocity to 2 m/s for the generation of the reference trajectory while the maximum allowed velocity of the simulated UAV remains at 3 m/s and its maximum acceleration remains at 1.5 m/s^2. Note that the maximum velocity of the simulated UAV is higher than the constant velocity of the reference trajectory. This enables the UAV to better catch up with the reference if a course deviation has occurred. The effect of this modification on the flight mission is shown in Figure 1.4. The reduction of the constant velocity results in an increase of the total mission duration to 38.12 s. However, there are still significant deviations between the planned and the resulting recording locations of more than 4 m. This is only a small improvement considering the additional flight time invested. Consequently, a simple reduction of the constant velocity in the generation of the reference trajectory does not seem to solve the problem of the occurrence of large deviations.

Consequently, the acceleration capabilities of a UAV cannot simply be neglected in UAV mission planning in the considered application. As a first approach to consider the acceleration properties of a UAV, each waypoint can be forced to be visited at rest. Hence, to move between two viewpoints, the UAV can fully accelerate towards the end viewpoint until the maximum velocity is reached, keep the maximum velocity, and then fully decelerate to reach the end viewpoint exactly at rest. This approach is described e.g. in [11]. This type

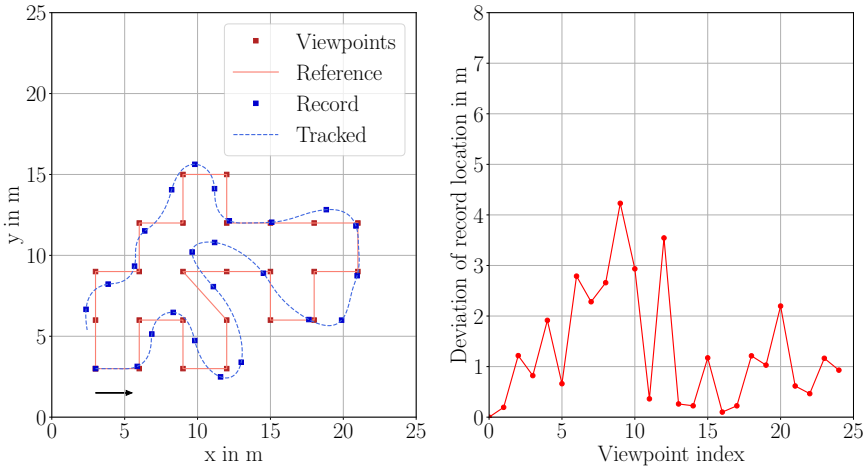


Figure 1.4.: Example mission with 25 waypoints for a multirotor UAV with a maximum velocity of $v_{max} = 3\,\mathrm{m/s}$ and a maximum acceleration of $a_{max} = 1.5\,\mathrm{m/s^2}$. The reference trajectory is calculated by the assumption the UAV can fly constantly at $v_{constant} = 2\,\mathrm{m/s}$ and is able to make on-spot turns. The righthand plot shows the distance between the planned viewpoint and the actual position at the recording timestamps. The planned mission duration is 38.12 s.

of reference trajectory generation is further denoted as *hover-2-hover* method. We provide an algorithmic solution procedure for this type of trajectory generation in Appendix A. The impact of using *hover-2-hover* trajectories for our example problem is shown in Figure 1.5. For the reference trajectory generation and the simulated UAV, the maximum velocity is $v_{max} = 3\,\mathrm{m/s}$ and the maximum acceleration is $a_{max} = 1.5\,\mathrm{m/s^2}$. It can be observed that, when using this method for reference trajectory generation, all images are taken exactly at the locations of the specified viewpoints. However, the entire mission duration is 71.25 s which is 86.9% more than reducing the constant velocity to 2 m/s. Intuitively, this cannot be the solution representing the minimum mission duration, because when multiple viewpoints lie on one straight line the UAV does not have to stop at each viewpoint to remain on track.

The *hover-2-hover* method is not the only method for reference trajectory generation considering the full kinematic property of a UAV. According to the state-of-the-art, one of the latest approaches in UAV routing problems considering a maximum velocity and acceleration is based on Dubins paths (see [5]). Dubins paths assume that the UAV moves at constant velocity and can only apply acceleration lateral to the direction of motion. This leads to trajectories containing straight segments and segments of constant turning radii. Dubins paths are especially suited to describe the motion of fixed-wing UAVs which require a minimum velocity to not suffer a stall, but they are also widely used as reference trajectories for multirotor UAVs (see e.g. [22, 23]). More details are given in Section 2.2.1. The globally optimal solution for applying this type of reference trajectory generation to our example problem is shown in Figure 1.6. For the reference trajectory generation and the simulated UAV, the maximum velocity is $v_{max} = 3\,\mathrm{m/s}$ and the maximum acceleration is $a_{max} = 1.5\,\mathrm{m/s^2}$. Although the obtained solution can be tracked by the simulated UAV with

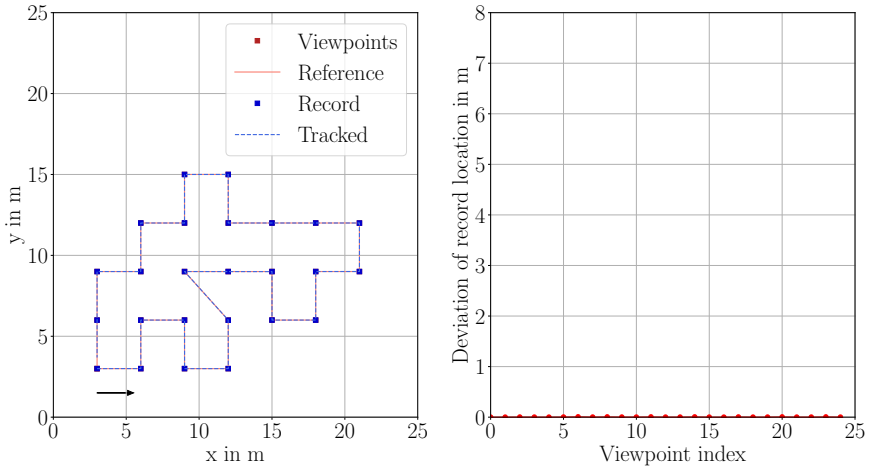


Figure 1.5.: Example mission with 25 waypoints for a multirotor UAV with a maximum velocity of v_{max} = 3 m/s and a maximum acceleration of a_{max} = 1.5 m/s^2. The reference trajectory is calculated by the assumption that the UAV starts and ends the motion between viewpoints at rest. The righthand plot shows the distance between the planned viewpoint and the actual recording position at the specified timestamps. The planned mission duration is 71.25 s.

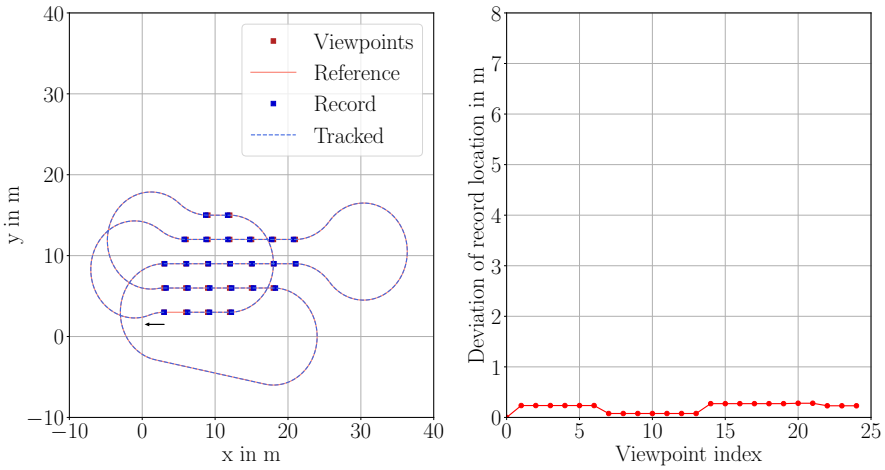


Figure 1.6.: Example mission with 25 waypoints for a multirotor UAV with a maximum velocity of v_{max} = 3 m/s and a maximum acceleration of a_{max} = 1.5 m/s^2. The reference trajectory is calculated by using Dubins paths considering the specified maximum allowed velocity and acceleration. The righthand plot shows the distance between the planned viewpoint and the actual recording position at the specified timestamps. The planned mission duration is 77.28 s.

sufficient precision, the reference trajectory itself appears suboptimal due to the occurring detours. These large loops result since the UAV must move at constantly 3 m/s and is only allowed to apply its maximum acceleration of 1.5 m/s^2 lateral to its direction of motion. This leads to a mission duration of 77.28 s, which is even higher than the *hover-2-hover* approach.

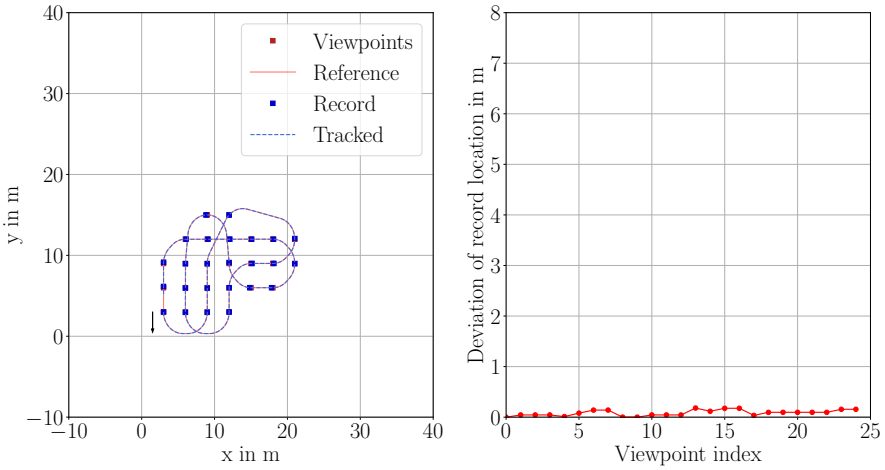


Figure 1.7.: Example mission with 25 waypoints for a multirotor UAV with a maximum velocity of v_{max} = 3 m/s and a maximum acceleration of a_{max} = 1.5 m/s^2. The reference trajectory is calculated by using Dubins path considering a reduced maximum allowed velocity of 2 m/s and an unchanged maximum acceleration of 1.5 m/s^2. The righthand plot shows the distance between the planned viewpoint and the actual recording position at the specified timestamps. The planned mission duration is 58.88 s.

To reduce the detour length, the constant velocity can be set below the maximum allowed velocity of the simulated UAV. For illustration, the constant velocity for the trajectory generation based on the Dubins path is set to 2 m/s, while the maximum allowed velocity of the simulated UAV remains at 3 m/s. The maximum allowed acceleration for both remains at 1.5 m/s^2. The result of this scenario is shown in Figure 1.7. Reducing the maximum allowed velocity the total mission duration decreases from 77.28 s to 55.88 s while the tracking performance remains high with a maximum deviation of only 0.18 m. However, this cannot be the optimal solution either, since viewpoints that lie on a straight line can be connected with higher velocity while guaranteeing trackability. With this example, we see that the optimum choice of the constant velocity for Dubins paths highly depends on the set of viewpoints itself and a general rule for selecting a proper constant velocity is not likely to be found, as we will see in Chapter 3.

1.1.2. Research Gap

In conclusion, the problem of finding a single minimum-time reference trajectory through a set of viewpoints such that the trajectory can be tracked by a UAV with limited maximum velocity and acceleration is hard. More specifically, as the above-stated preliminary experiments indicate, this problem is not yet solved by the current state-of-the-art, although it is of great economic importance. Missions of shorter duration allow more flights to be conducted in a given time. For a company working on UAV-based infrastructure monitoring as an example, this increases the number of orders/customers that can be served and hence the revenue of the company. Further, it might cause a potential cost reduction due to the need for fewer UAVs and their technical equipment.

1.2. Scope and Contribution of this Thesis

Motivated by the economical potential of solving related types of problems to optimality, we investigate whether it is even possible to find minimum-time trajectories through a set of viewpoints where each viewpoint is traversed precisely and the trajectory is fully trackable by the UAVs given kinematic properties. We propose a new mathematical programming formulation, namely the kinematic traveling salesman problem (KTSP), which allows finding a globally optimal solution for the described class of TSP-like problems. Further, we propose another new mathematical programming formulation also considering the maximum acceleration and velocity of a UAV, denoted as kinematic orienteering problem (KOP). The KOP is an extension of the orienteering problem (OP) and aims at maximizing the collected priorities for visiting prioritized viewpoints within a given maximum flight time. Due to the consideration of kinematics as well as the maximum flight time, the KOP is highly relevant for real applications, especially in scenarios where not all given waypoints can be visited within the maximum flight time of the UAV and a selection of waypoints must be made.

The potential of both of our models, the KTSP and the KOP, relies on a proper calculation of the UAV's kinematic motions between the viewpoints, i.e. to calculate the edge representation for the underlying graph-based problem formulation. This potential can only be fully unfolded when using time-optimal trajectory generation approaches for multirotor UAVs. In general, multirotor UAVs can sufficiently precisely be modeled as a point-mass with constrained maximum velocity and acceleration for which such an approach exists in the literature. However, we found that the state-of-the-art approach for time-optimal trajectory planning for such point-mass models is not generally correct. Therefore, we further propose a new optimization-based approach to solve this trajectory planning problem. Our approach is proven to be valid in general and yields a globally optimal solution.

Moreover, we show that solving problem instances of the KTSP and the KOP with a large number of viewpoints becomes computationally intractable. Therefore, we also develop powerful heuristic solvers based on the adaptive large neighborhood search solution framework (see e.g. [24]). Last, we evaluate our proposed approaches in an extensive computational study and discuss future research direction.

The work presented in this thesis has been partly published in the following three peer-reviewed scientific articles:

- [25]: F. Meyer and K. Glock, Trajectory-based Traveling Salesman Problem for Multirotor UAVs, 17th International Conference on Distributed Computing in Sensor Systems (DCOSS), Cyprus, 2021
- [26]: F. Meyer and K. Glock, Kinematic Orienteering Problem with Time-optimal Trajectories for Multirotor UAVs, IEEE Robotics and Automation Letters, vol. 7, no. 4, 2022

- [27]: F. Meyer, K. Glock and D. Sayah, TOP-UAV: Open-source time-optimal trajectory planner for point-masses under acceleration and velocity constraints, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Detroit, 2023

In these articles, the contribution of the second and third co-authors is limited to the discussion of possible solution approaches, suggestions for related literature, and support by reviewing preliminary versions of the articles with regard to mathematical formality, structure, and language. All remaining work including ideation, literature review, development of solution approaches, implementation, computational evaluation, and documentation in the form of scientific articles represents the contribution of the first author.

In addition to the published articles, we also contributed to the scientific community by providing the source code of our time-optimal trajectory generation method for point-masses as open-source in the programming languages

- **C++:** `https://github.com/fzi-forschungszentrum-informatik/top_uav_cpp` and
- **Python:** `https://github.com/fzi-forschungszentrum-informatik/top_uav_py`.

Finally, a part of the approaches presented in this work contributed to the research project LOKI which has been funded by the Federal Ministry of Education and Research of Germany, Grant No.: 03G0890A. A brief description of the research project can be found in [15].

1.3. Outline

The remainder of this work is organized into two major chapters.

Chapter 2 focuses on the trajectory generation for point-masses under acceleration and velocity constraints serving to calculate the representation of possible motions between waypoints for the inertia-based routing problems introduced in Chapter 3. This includes a literature review of existing trajectory generation approaches that can be applied to the above-described point-mass model. Based on this literature, the analytical generation of time-optimal trajectories considering the relevant kinematic constraints is identified as a potential candidate for this work. However, we show in our work [26] that the corresponding state-of-the-art approach has a general flaw that might result in the generation of trajectories that cannot be tracked. Therefore, we developed a new analytical approach that is proven to overcome the identified issue of the state-of-the-art approach. We evaluate our new approach in an extensive computational study.

In Chapter 3, we focus on inertia-based routing problems for UAVs. We describe the current state-of-the-art and derive the research gap that can be closed to a significant extent by our mathematical programming formulation for the KTSP and the KOP. Further, we develop heuristic solution approaches for both which are based on the adaptive large neighborhood search. We evaluate all proposed approaches in another detailed computational study.

Finally, we provide a summary of this work and give an outlook for further research in Chapter 4.

2. Time-Optimal Trajectory Generation for Point-Masses

As indicated in Chapter 1, the motion planning approach considered for connecting spatial waypoints has a high impact on the obtained solution quality for inertia-based routing problems. Therefore, in this chapter, we give deep insights into existing motion planning approaches. We identify a major flaw in the state-of-the-art approach to solve the time-optimal trajectory planning problem for point-masses with constrained maximum acceleration and velocity and develop a new approach that is proven to overcome the identified problem.

This chapter is organized as follows. First, we set the focus on a general introduction of the trajectory planning problem for inertia-based route planning problem and describe a set of requirements that must be fulfilled by a potential solution approach in Section 2.1. In Section 2.2, we give an overview of the related work on trajectory planning approaches from the literature, give a formal introduction to the time-optimal trajectory planning problem for point-masses under acceleration and velocity constraints (TOT-PMAV) and present the state-of-the-art (SOTA) solution approach to solve it in more detail. Next, we show in Section 2.2.3 that the SOTA approach is not valid in general and hence, state a set of open research questions associated with the invalidity. To answer the described research questions, we present our analytical approach to solving the TOT-PMAV in one dimension globally optimally in Section 2.3. Utilizing the one-dimensional approach, we present our new approach to solving the TOT-PMAV in multiple dimensions that is proven to be valid in general and we explain the cause of the flaw of the SOTA approach in Section 2.4. We close this chapter by presenting the results of our extensive computational study in two and three dimensions in Section 2.5.

2.1. Trajectory Planning for Inertia-based Routing

The basis of inertia-based routing is the generation of a trajectory that connects a given start waypoint to a given end waypoint, both specified with a predefined velocity vectors. In general, a single trajectory generation method is not suited for all physical systems, but for a small group. For example, the physical motion capabilities of a robotic arm are entirely different from the ones of a helicopter. This difference impacts the usefulness of a trajectory generation method in a given application.

The scope of this work is mainly focused on the application of UAVs for 3D reconstruction as described in Section 1.1, but is not limited to it. Therefore, we primarily consider multirotor UAVs as the applied physical system. Multirotor UAVs can hover and accelerate in any arbitrary direction. Their maneuverability makes them perfectly suited for 3D reconstruction applications. However, their motion capacities are limited. The acceleration power is restricted due to the maximum thrust of the rotors and the mechanical design of the UAV. Further, the UAV cannot move with arbitrary velocity in the real world. At some point, the air friction becomes too high to further increase the velocity. This limitation can approximately be modeled by a maximum allowed velocity constraint. Overall, this leads to the assumption that multirotor UAVs can sufficiently precisely be modeled as a point mass with a constrained maximum allowed velocity and acceleration. This assumption is widely used in the literature (see [28, 29, 30]) and we indicate in [26] via simulation that this assumption is correct. For that reason, only trajectory generation methods that are suitable for point-mass models with constrained maximum velocity and acceleration are relevant for this work.

Next, for UAV applications, the goal is to complete the flight missions as quickly as possible while ensuring that the user-defined requirements on the quality of the collected data are met. This emphasizes the need that the considered trajectory generation methods should be capable of generating time-optimal motions while considering the kinematic properties of UAVs.

Further, as indicated in Section 1, inertia-based routing requires a huge number of trajectories as edges of a graph to be calculated for a single problem instance. This number lies in the scale of millions, as we demonstrate in Chapter 3. This increases the need for a potential trajectory generation method that features a very quick computation of a single trajectory.

To conclude, we formulate the following requirements for a suitable trajectors generation method for multirotor UAVs:

- Applicability to point-mass models with constrained maximum velocity and acceleration
- Time-optimality and hence full exploitation of the underlying system's physical properties
- Quick calculation in the scale of a few microseconds

With this in mind, we give an overview of related work for trajectory generation in the next Section. From the literature, we identify only a single candidate that seems to fit the above-stated requirements. However, we will show that this approach suffers from a major flaw. Solving this problem, and hence making the approach suitable to inertia-based routing problems, is therefore the major objective in Sections 2.3 and 2.4. Lastly, we conduct the computational study for our trajectory generation (see Section 2.5) in two as well as three dimensions.

2.2. State-of-the-Art and Research Gap

In general, path and motion planning for UAVs is a well-studied subject in robotics research. At the moment, most effort is put into the development of approaches that can handle dynamic environments with obstacles. In this work, however, a static environment is considered with the objective of visiting a given set of waypoints. Intuitively, one might assume that the literature already provides an approach that matches the comparably simple requirements for this work. However, this is not the case as we show in Section 2.2.3. In the following, we first give an overview of the related work and show that the only approach covering all requirements suffers from a major flaw.

2.2.1. Related Work

Many approaches in the literature are being applied to UAV motion planning. In some cases, multiple variations and enhancements that share the same basic idea have been studied. Therefore, we introduce the approaches from the literature according to the major group they belong to.

Dubins paths:
A widely used approach to generate trajectories are so-called Dubins paths [31]. Originally, Dubins paths were designed to find the shortest curvature-constrained path between a pair of two-dimensional coordinates with given tangents. The approach demonstrated that it can be applied to all physical systems that can somehow be modeled as a so-called Dubins vehicle. It describes a vehicle that moves at constant velocity and can apply a constant maximum acceleration force lateral to the direction of motion [22]. This results in a motion that consists of straight-line segments and segments with a constant turning radius. According to the kinematics of the Dubins vehicle, Dubins paths are globally optimal. In the field of aerial surveillance, Dubins paths are primarily used to plan trajectories for fixed-wing UAVs [5] but they are also applied for multirotor UAVs [22, 23]. The advantage of using Dubins paths and thus assuming constant velocity is that acceleration and deceleration are avoided, making the trajectory more energy-efficient overall, as illustrated in Chapter 1. Another advantage is that they can be calculated with very low computational effort. In [32] it is stated that they require only 20 μs of computation time. However, their disadvantage, especially regarding multirotor-UAVs, is that they are bound to a constant velocity. In general, this comes at the risk of large detours (see Figure 1.6) since the maximum acceleration has to fight against the prevailing mass inertia while the longitudinal velocity is to be kept constant. At some point, these detours cancel out the efficiency advantage of a constant velocity and that is why an optimal trade-off has to be determined [25]. Associated with this disadvantage is the fact, that the constraint of a constant velocity does not cover the kinematics of a multirotor UAV properly [33]. The ability of multirotor UAVs to hover cannot be modeled with Dubins paths. The idea of Dubins paths is rather suited for vehicles that have to steadily move at a minimum velocity such as airplanes or fixed-wing UAVs. To better suit the requirements of fixed-wing UAVs in real-world applications, Dubins paths were extended

into the three-dimensional space e.g. [34]. Moreover, a few concepts on Dubins paths even allow the motion to vary between a minimum and a maximum longitudinal velocity e.g. [35]. However, even such approaches are bound to a constant turning radius that is only achieved by a constant velocity for the time segments that correspond to flying a turn.

Bézier curves:
Another possibility to generate motion trajectories for multirotor UAVs is by using so-called Bézier curves [36, 37]. Here, two locations are connected by a smooth curve based on Bernstein polynomials. By setting intermediate control points properly, it is possible to guide the curve safely around obstacles. However, the polynomial representation of a safe curve through Bézier curves is purely spatial and hence, represents a spatial path. To ensure physical feasibility, the Bézier curve must first be transferred into the time domain. Only by assigning each spatial point of the curve to a particular point in time is it possible to consider physical constraints such as maximum velocity and maximum acceleration (see [38]). This two-step procedure is computationally expensive (see [36]), especially for inertia-based routing problems where millions of trajectories are calculated. Further, it is argued in [38] that if the initial and final velocity are not both zero, there may not exist a feasible solution. This is a critical drawback of Bézier curves in general and especially for their applicability in this work. Moreover, another disadvantage is that the resulting trajectory is likely to be time-suboptimal since the inherent smoothness of polynomial representation does not allow for bang-bang behavior, which is required to achieve time-optimality [28]. In control engineering, the term bang-bang behavior describes that the control variable is always at its limit. Nevertheless, due to their capability of obstacle-avoidance, Bézier curves are one of the most applied trajectory generation approaches in literature for point-to-point motion planning (see [39, 38, 40, 41, 42, 43, 44]).

Minimum-snap trajectories:
A further widely used class of trajectory generation methods for multi-rotor UAVs is minimum-snap trajectory generation. Note that the snap describes the second time-derivative of the acceleration. Similar to Bézier curves the generation of minimum-snap trajectories is based on polynomials. However, unlike Bézier curves, the polynomials utilized in minimum-snap trajectory generation are defined directly in the time domain. Therefore, no transfer from the spatial to the time domain is required. The major advantage of this trajectory generation method is that the resulting trajectories can directly be transferred into control input trajectories due to the differential flatness property of the dynamic multirotor UAV model (see e.g. [45]). The computation time of minimum-snap trajectory generation lies in the range of a few hundred microseconds (see [46]). For this work, the computation time is acceptable but tends to be rather too slow. Further, it is difficult for these approaches to consider bounds on maximum velocity and maximum acceleration in a computationally efficient way. This is because the associated mathematical program can consider a maximum velocity for specific points in time, but not for the polynomial representation of the trajectory as a whole. Lastly, minimum snap trajectory generation cannot yield time-optimality since they also suffer from the inherent smoothness of polynomials which does not allow for the bang-bang behavior required to achieve time-optimality [28]. Despite this drawback, their advantage to match with the

differential flatness property of multirotor UAVs makes them widely used in literature (see [32], [46], [47], [48], [49], [50])

Model-based predictive control:
Another possibility to generate trajectories for UAVs is model-based predictive control (MPC). MPC iteratively solves the optimal control problem for time-discretized dynamical systems (see [51, Chapter 13]) and is mostly used to determine control input trajectories that enable tracking a given reference trajectory. This also holds for UAV applications and means that MPC is primarily used to calculate the required thrust at each rotor of a given UAV to follow a given reference trajectory in the best possible way [52, 53, 54]. However, MPC is also suitable to generate reference trajectories for multirotor UAVs (see [55]). Some MPC approaches even solve the reference trajectory generation problem simultaneously with the associated tracking problem in a single optimization process [56]. In general, MPC is based on a discrete-time motion model of the UAV consisting of system parameters, system state, and control variables. The most used objective is to find a sequence of control inputs that minimizes the deviation of the current state to a reference, for example, described by the desired end state, while respecting limits of control input and feasible states. Specifically, the ability to define a custom set of allowed system states and control inputs makes them one of the most successful methods in UAV motion planning [57, 58, 59, 60, 61]. However, there are also some drawbacks of MPC. First, it is hard to achieve time-optimality with MPC, and second, since MPC solves a mathematical program, usually quadratic, numerically, it is a computationally expensive method (see [62]) that requires a computation time in the scale of ms for the trajectory planning problems considered in this work.

Time-optimal trajectory generation:
So far, none of the presented approaches is capable of fulfilling the time-optimality requirement for multirotor UAV motion planning stated in Section 2.1. However, there are such approaches. These approaches mostly rely on the assumption that the UAV approximately behaves as a point-mass and utilize Pontryagin's minimum principle (see [63]), which states that time optimality is achieved by having the system always operate at its physical limits. Consequently, the overall motion of the UAV can be divided into several segments of constant maximum or minimum control input value or zero in case the system reaches state limits. This property is used to define a set of control input patterns that are used to calculate the time-optimal trajectory analytically and in a very short time. Typical point-mass models either use bounded jerk, i.e. the first time derivative of the acceleration, along each coordinate axis as control input and restrict the maximum velocity and acceleration ([64, 65]) or rely on a bounded acceleration as control input, sometimes with a further restriction on maximum velocity [28, 26, 66, 30, 29]. Although the latter point-mass model is less precise according to the full dynamics of a multirotor, the resulting trajectories can still precisely be tracked (see [26, 56, 28]). Most analytical time-optimal trajectory generation methods that rely on a point-mass model are based on a decoupling of the coordinate axes. Here, the time-optimal trajectory planning problem is first solved for each dimension individually before all coordinate axes are synchronized again. However, the process of axis synchronization appears to be a hard problem. For example, [64] reported that they were not able to find an analytical solution for the axis

Method	Time-optimality	Point-mass model	Computation time
Dubins paths	✗	✓	✓
Bézier curves / B-splines	✗	✗	✗
Minimum-snap	✗	✗	✓
MPC	✗	✓	✗
Time-optimal	✓	✓	✓

Table 2.1.: Requirements covered by the different classes of state-of-the-art approaches. 'Time-optimality' is covered if the associated trajectory generation approach yields time optimality for the point-mass model with constrained maximum velocity and acceleration. The 'point-mass model' requirement is covered if the associated approach can be applied for point-masses with a constrained maximum velocity and acceleration. The requirement 'computation time' is fulfilled if the associated approach yields solutions in the scale of up to 500 μs.

synchronization in certain cases for their point-mass model with jerk as the control input. But also for the less precise point-mass model with acceleration as the control input, this approach fails in some situations as we show in our work [26]. In Section 2.4.1.1, we give insights into why this is the case.

Conclusion:
Table 2.1 summarizes the properties of the existing approaches and offers a direct comparison regarding the declared requirements of this work. As stated above, Dubins path can be calculated in a short time. Further, they represent a time-optimal solution to the trajectory planning problem of the Dubins vehicle. However, the kinematics of a multirotor UAV modeled as point-mass do not exactly match those of the Dubins vehicle. Therefore, Dubins paths are in general time-suboptimal for pure point-mass models. However, although yielding time-suboptimal motions, the kinematics of Dubins paths are covered by point-mass models which is why Dubins paths can be applied as a feasible trajectory generation approach for multirotor UAVs.

Bézier curves have the advantage of implicit obstacle avoidance. The consideration of a maximum velocity and acceleration constraint is possible if the trajectory is to be calculated if the start and end velocity is 0 m/s. However, that requires a computationally expensive time allocation procedure for each spatial position of the curve. Moreover, if the start and end velocity are not both 0 m/s there might be no feasible solution for this time allocation process. Therefore, we specify that Beziér curves cannot be applied for point-masses with a constrained maximum acceleration and velocity in general. Further, since Bézier curves are based on polynomials they are time-suboptimal.

Minimum-snap trajectories come with the advantage, that it is possible to analytically derive the optimal control input trajectory from the calculated snap-minimizing state trajectory due to the differential flatness property. However, minimum-snap trajectories are again a polynomial approach and hence time-suboptimal. Further, it is time-consuming to consider acceleration constraints and to the best of our knowledge, no variation of the minimum-snap trajectory generation approach from the literature covers the constraint of

a maximum velocity. The required computation time depends on the computational setup but is defined as acceptable for this work since the authors in [46] report computation times of approximately 180 μs for a single optimization of the coefficients of a polynomial passing through four spatial waypoints.

Due to the possibility of intuitive customization, MPC is one of the most powerful trajectory-planning approaches in the literature that can also be designed to cover the kinematics of a point-mass with a constrained maximum velocity and acceleration. However, since it is based on the solution of a mathematical optimization problem, MPC is computationally expensive. Further, the basic concept of MPC is not suitable to achieve time-optimality in general. Hence, we consider the requirement of 'time-optimality' as not covered by MPC.

The only approach that appears to fulfill the requirements of this work is given by the time-optimal approach. As their name indicates, they yield time-optimal motions. Further, they require a short computation time and consider state and control input constraints. According to Table 2.1, the time-optimal trajectory generation approaches offer the highest potential for this work and serve as a starting point. However, our research found that the state-of-the-art approach for time-optimal trajectory generation of point-masses sometimes yields trajectories that miss the required final state by far. Hence, we developed a new method that can resolve the identified issue. To be able to illustrate this error, we formally introduce the time-optimal trajectory planning problem for point-masses with maximum acceleration and velocity constraints in the next section, followed by a description of the state-of-the-art method to solve the problem. Then in Section 2.2.3, we provide an example where the state-of-the-art to calculate the time-optimal trajectory fails. From this example, we derive the research questions for this chapter on trajectory generation.

2.2.2. Problem Definition and State-of-the-Art Approach

In this section, we formally introduce the time-optimal trajectory planning problem for point-masses under acceleration and velocity constraints (TOT-PMAV) and then describe the state-of-the-art approach to solve it.

2.2.2.1. Problem Definiton of the TOT-PMAV

In the literature, many related versions of the TOT-PMAV are defined (e.g. see [66, 56, 29, 64]). For this work, we define the TOT-PMAV as follows. Given are the start and end position and velocity vectors $\boldsymbol{p_s}, \boldsymbol{p_e}, \boldsymbol{v_s}, \boldsymbol{v_e} \in \mathbb{R}^n$ of a point-mass in the n-dimensional cartesian space. Expressions related to a single specific axis are described by $p_s, p_e, v_s, v_e \in \mathbb{R}$. Without loss of generality, the axis indices are left out for simplicity. Further, a maximum allowed velocity and acceleration in any direction $\hat{v}_{max}, \hat{a}_{max} \in \mathbb{R}^+$ is given. The TOT-PMAV is to find a bounded acceleration function $\boldsymbol{a}(t) \in \mathbb{R}^n, ||\boldsymbol{a}(t)|| < \hat{a}_{max}, t \in [0, T^*]$ such that the initial state $[\boldsymbol{p_s}, \boldsymbol{v_s}]^\top$ is transfered into the final state $[\boldsymbol{p_e}, \boldsymbol{v_e}]^\top$ and the total

trajectory duration $T^* \in \mathbb{R}^+$ is minimized. Further, the resulting overall velocity function $\boldsymbol{v}(t) \in \mathbb{R}^n$ has to respect the bound $||\boldsymbol{v}(t)|| < \hat{v}_{max}$ for all $t \in [0, T^*]$.

2.2.2.2. State-of-the-Art Solution Approach for the TOT-PMAV

In this section, we focus on the state-of-the-art (SOTA) approach to solve the time-optimal trajectory planning problem for point-masses under acceleration and velocity constraints (TOT-PMAV) as briefly introduced in Section 2.2.1 under the headline 'time-optimal trajectory generation'. The subsequently described approach is widely applied in the literature (see [66, 56, 30, 29, 64]).

The SOTA to determine a solution for the TOT-PMAV is based on the decoupling of axes. This means that in the first step, all spatial coordinate axes $i \in \{1, ..., n\}$ of the trajectory in the n-dimensional space are separated and the maximum allowed velocity and acceleration for the i-th axis are set to

$$-v_{max} \leq v_i(t) \leq v_{max} \tag{2.1}$$

$$-a_{max} \leq a_i(t) \leq a_{max} \tag{2.2}$$

with $v_{max} := \hat{v}_{max}/\sqrt{n}$ and $a_{max} := \hat{a}_{max}/\sqrt{n}$. Pontryagin's minimum principle [63] is applied for each axis yielding a bang-zero-bang control input pattern. Such a pattern defines that the acceleration is either at its limits or zero if the velocity limit is reached and no further acceleration would be allowed. For the i-th axis with bounded acceleration as control input and a maximum allowed velocity, the resulting bang-zero-bang acceleration pattern (a_1, a_2, a_3), which defines the acceleration profile

$$a_i(t) = \begin{cases} a_1, & 0 \leq t < t_{1,i} \\ a_2, & t_{1,i} \leq t < t_{1,i} + t_{2,i} \\ a_3, & t_{1,i} + t_{2,i} \leq t \leq t_{1,i} + t_{2,i} + t_{3,i}, \end{cases} \tag{2.3}$$

is given by $(+a, 0, -a)$ with either $a = -a_{max}$ or $a = +a_{max}$. Here, a_{max} represents the maximum allowed acceleration. The duration of each time segment of constant acceleration is described by $t_{1,i}, t_{2,i}$ and $t_{3,i}$. In case the velocity limit is not reached for the i-th axis, $t_{2,i} = 0$ holds. Consequently, the time-optimal trajectory consists of two segments of constant maximum acceleration with opposite signs and, if the velocity limit is reached, an additional segment of no acceleration. The time-optimal trajectory duration for the i-th axis $T_{opt,i}$ can be calculated analytically (e.g. see Section 2.3 or our work [25]). Next, the SOTA postulates that the overall duration T_{SOTA} is defined as the largest of the time-optimal durations $T_{opt,i}$ over all axes $i \in \{1, ..., n\}$, i.e.

$$T_{\text{SOTA}} = \max_{i \in \{1,...,n\}} \{T_{opt,i}\}. \tag{2.4}$$

Hence, the one-dimensional trajectories of all axes have to be stretched in time to last exactly as long as the slowest trajectory. This is further referred to as all axes are syn-

chronized with T_{SOTA}. The interesting question here is how this synchronization can be realized. To the best of our knowledge, no concrete algorithm can be found in the literature. In the following Section 2.2.3, we mathematically explain why this is the case by showing that Equation (2.4) itself is no generally valid approach to determine the correct synchronization time.

2.2.3. Research Gap and Contributions

The main research gap identified in this chapter comes from the observation, that the SOTA approach presented in the previous Section 2.2.2.2 (see [66, 56, 30, 29, 64]) is not valid in general. As we will demonstrate in the following, the SOTA approach can yield trajectory durations T_{SOTA} that sometimes cannot be applied to synchronize all coordinate axes. Starting the trajectory from a given start position and velocity vector, using T_{SOTA} for axis-synchronization might lead to large deviations between the desired and actual final state. We identified in our research (see [26]) that the reason is that the SOTA approach does not consider the inertia of the movement properly. Therefore, it sometimes results in overshooting the desired final state. To the best of our knowledge, this effect has first been reported in the literature in our work [26]. In the following, we show an example where the SOTA yields an invalid result and which has been published in our work [26].

Figure 2.1 illustrates the trajectory generation in two dimensions x and y based on the SOTA. For the x axis the initial state is $p_{x,s} = 0\,\text{m}$, $v_{x,s} = 0\,\frac{\text{m}}{\text{s}}$, where $p_{x,s}$ describes the initial position and $v_{x,s}$ the initial velocity. The resulting velocity vectors given in Figure 2.1 are normalized to one second. The desired end state for the x axis is $p_{x,e} = 5\,\text{m}$, $v_{x,e} = 2\,\frac{\text{m}}{\text{s}}$. The range of allowed per-axis velocity values is $v \in [-2, 2]$ $(\frac{\text{m}}{\text{s}})$ and per-axis acceleration values is $a \in [-0.5, 0.5]$ $(\frac{\text{m}}{\text{s}^2})$. Based on these kinematic per-axis restrictions the time-optimal duration along each axis can be calculated e.g. as we described in our work [25] and is $T_{opt,x} = 4.5\,\text{s}$ for the x axis. For the y axis the initial and end state is $p_{y,s} = 0\,\text{m}$, $v_{y,s} = 2\,\frac{\text{m}}{\text{s}}$, $p_{y,e} = 5\,\text{m}$, $v_{y,e} = 2\,\frac{\text{m}}{\text{s}}$. Here, the calculation of the time-optimal duration yields $T_{opt,y} = 2.5\,\text{s}$ which corresponds to maintaining the maximum velocity of 2 m/s for the entire motion. According to Equation (2.4), both axes must be synchronized at $T_{sync} = 4.5\,\text{s}$. Remember, the term 'synchronization' means that for both axes an acceleration pattern must be found that meets the requirements of each axis and lasts for the specified synchronization time.

We use the MPC which we provide in Appendix B.2 to validate axis synchronization due to its ability to conduct trajectory generation for point-masses with a fixed duration. This MPC is also based on a full decoupling of the coordinate axes and with the per-axis velocity constrained by $v \in [-2, 2]$ $(\frac{\text{m}}{\text{s}})$ and per-axis acceleration constrained by $a \in [-0.5, 0.5]$ $(\frac{\text{m}}{\text{s}^2})$. For the above example, it can be seen that the resulting two-dimensional trajectory with the fixed synchronization time $T_{\text{sync}} = 4.5\,\text{s}$ yielded by our MPC misses the required final state by far (see blue dots in Figure 2.1).

This behavior is due to the inertia of the system. The high initial velocity along the y axis in combination with an insufficient acceleration power and the fixed duration to reach the final state leads to overshooting the specified final waypoint. As a result, although

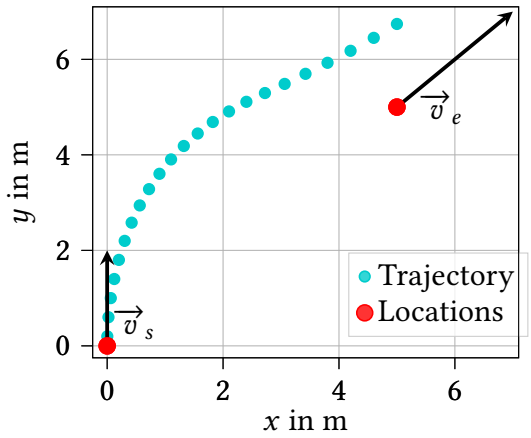


Figure 2.1.: Example for insynchronizability of a given initial and end state.

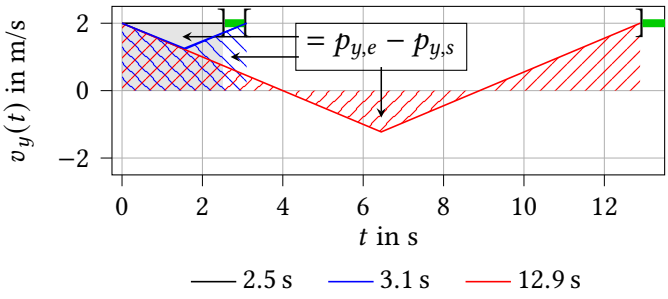


Figure 2.2.: Range of valid trajectory durations and their connection to corresponding velocity profiles shown in green. The velocity profiles that lead to the limits of the corresponding range are given by colored lines. The associated areas underneath these profiles are highlighted in the respective colors.

the y axis has a potentially faster execution, it cannot be synchronized with the slower time-optimal duration of the x axis.

Remember that our objective is to find the lowest T_{sync} that can be used to feasibly synchronize all axes. For this reason, we now present more detailed insights into the domain of feasible T_{sync} for the y axis in the above scenario. Figure 2.2 shows feasible velocity profiles for different synchronization times T_{sync} for the y axis at which the desired final state is to be entered. The property of a feasible trajectory is that the desired start and end velocity must be met exactly, and the integral under the velocity profile must equal $p_{y,e} - p_{y,s}$. The velocity profile $v_y(t)$ of the time-optimal trajectory is shown by a solid black line. Since its initial velocity already equals the maximum allowed value, it remains constant and $T_y = T_{opt,y} = 2.5\,\text{s}$ results. If the trajectory duration must be increased due to waiting for a slower axis, i.e. $T_y > 2.5\,\text{s}$, the UAV first decelerates the motion and accelerates afterward to meet the required final velocity $v_{y,e} = 2\,\frac{\text{m}}{\text{s}}$. However, this procedure works only as long as the integral under the velocity profile equals $p_{y,e} - p_{y,s}$. The longest velocity profile that does not violate this condition corresponds to $T_y \approx 3.1\,\text{s}$ (see blue line in Figure 2.2). Further increasing the required trajectory duration leads to overshooting until $T_y \geq 12.9\,\text{s}$ (see red line in Figure 2.2). From this duration on it is possible to compensate for overshooting by flying a loop. In total, the range of feasible trajectory durations with respect to the

requirements on the y axis is given in green. As can be seen, a synchronization with $T_{\text{sync}} = T_{opt,x} = 4.5\,\text{s}$ is not possible. In general, such an analysis must be done for all axes and the lowest feasible trajectory duration that is valid for all axes represents the time-optimum trajectory duration for the entire motion. However, such an analysis is entirely left out by the approaches from the SOTA, which implicitly assume that each axis can always be arbitrarily slowed down.

Research Questions:
With the research gap presented above, there are two major research questions:

1. What is the mathematical reason that sometimes not all axes can be synchronized with T_{SOTA}?
2. How to determine the time-optimal trajectory duration in a general valid way?

And a couple of complementary research questions:

1. How often does the SOTA approach fail to calculate a feasible trajectory duration?
2. What are the consequences for the trajectory generation process if the SOTA calculates the wrong trajectory duration?
3. How computationally competitive would a generally valid approach be compared to the SOTA approach?
4. According to Equations (2.1) and (2.2), the maximum allowed velocity is distributed equally among the axis. How to consider uneven distributions that enable faster motion in one specific direction?

Contributions:
By answering these research questions, we make the following contributions in this chapter:

- We mathematically show the reason why the SOTA approach is error-prone for solving the TOT-PMAV.
- We develop our new TOP-UAV trajectory planner to solve the TOT-PMAV and mathematically prove its general validity and optimality.
- We demonstrate by simulation in which situations the SOTA approach is likely to fail and evaluate the resulting consequence.
- Since the SOTA approach as well as our TOP-UAV approach both suffer from not exploiting the full kinematics due to considering the coordinate axes as decoupled, we further develop our improved TOP-UAV++ trajectory planner that better exploits the given kinematic properties. We show that TOP-UAV++ yields on average up to 14% faster trajectories than TOP-UAV.
- We provide the source code for our TOP-UAV and TOP-UAV++ trajectory planner as open source on GitHub in C++ (see [67]) and Python (see [68]).

The remainder of this section is structured as follows:

First, we start by presenting an analytical approach for solving the one-dimensional TOT-PMAV in Section 2.3 since it is the basis of our TOP-UAV and TOP-UAV++ trajectory planners. The approach presented here is an improved version of what we presented in [25] yielding a significant computational performance improvement. With this as a basis, we start to consider multiple dimensions in our approach to solve the TOT-PMAV in Section 2.4. For this, we investigate the synchronization feasibility of specific control input patterns which are explained in detail in Sections 2.4.1 - 2.4.1.2 and represent bang-zero-bang patterns. Within this investigation, we answer the first major research question in Section 2.4.1.1 by showing that the SOTA neglects domain gaps, which occur for the applied acceleration patterns. Next, we present our TOP-UAV approach to solve the TOT-PMAV, mathematically prove its correctness, and present a solution algorithm that exploits our mathematical findings in Section 2.4.2 and 2.4.3. With this, we answer the second major research question. The fourth complementary research question is answered by our development of our TOP-UAV++ trajectory planner in Section 2.4.4 and the remaining complementary research questions are addressed in our computational study presented in Section 2.5.

2.3. Time-Optimal Trajectory Generation in One Dimension

In this Section, we describe the time-optimal trajectory generation with bounded velocity and acceleration in a single dimension with acceleration as the control input. Our derivation is based on Pontryagin's minimum principle which states that time-optimality is achieved by having the system always operate at its physical limits yielding a bang-zero-bang behaviour.

The subsequently presented approach is an improved and not yet published approach of what is presented in [25] with fewer mathematical operations needed. With this approach, calculating the time-optimal duration of a one-dimensional TOT-PMAV requires on average only 24ns compared to 85ns of the approach described in [25]. Both average computation times are obtained from our C++17 implementations executed on an Intel Core i7-8565U CPU.

With bounded acceleration as control input and a maximum allowed velocity, the resulting bang-zero-bang acceleration pattern (a_1, a_2, a_3), which defines the acceleration profile

$$a(t) = \begin{cases} a_1, & 0 \leq t < t_1 \\ a_2, & t_1 \leq t < t_1 + t_2 \\ a_3, & t_1 + t_2 \leq t \leq t_1 + t_2 + t_3 = t_e, \end{cases} \tag{2.5}$$

is given by $(+a, 0, -a)$ with either $a = -a_{max}$ or $a = +a_{max}$ and a_{max} representing the maximum allowed acceleration (see Figure 2.3). The durations of each time segment of

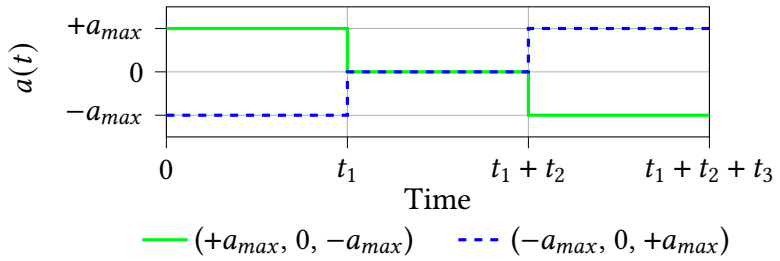


Figure 2.3.: Example acceleration patterns for bang-zero-bang behaviour in one dimension.

constant acceleration are described by $t_1, t_2, t_3 \in \mathbb{R}_0^+$. In case the velocity limit is not reached $t_2 = 0$ holds. Consequently, the time-optimal trajectory consists of two segments of constant maximum acceleration with opposite signs and, if the velocity limit is reached, one segment of no acceleration. The time-optimal trajectory duration T_{opt} for a single axis can be calculated as follows.

In the second segment of no acceleration, the constant velocity v_c is determined as

$$v_c = v_s + at_1. \tag{2.6}$$

For the end velocity holds

$$v_e = v_c - at_3. \tag{2.7}$$

For the end position holds

$$p_e = p_s + v_s t_1 + \frac{1}{2}at_1^2 + v_c t_2 + v_c t_3 - \frac{1}{2}at_3^2. \tag{2.8}$$

Note that for equations (2.6) - (2.8) the parameter $a \in \{+a_{max}, -a_{max}\}$ represents the acceleration value of the first segment in the selected acceleration pattern.

Last, depending on the start and end state of the one-dimensional trajectory, we distinguish between four cases from which a fourth equation is derived that needs to hold. These cases are listed in the following:

Case 1: $v_c = v_{max}$, in case v_{max} is reached using pattern $(+a_{max}, 0, -a_{max})$ (2.9a)

Case 2: $t_2 = 0$, in case the v_{max} is not reached using pattern $(+a_{max}, 0, -a_{max})$ (2.9b)

Case 3: $v_c = -v_{max}$, in case $-v_{max}$ is reached using pattern $(-a_{max}, 0, +a_{max})$ (2.9c)

Case 4: $t_2 = 0$, in case $-v_{max}$ is not reached using pattern $(-a_{max}, 0, +a_{max})$ (2.9d)

Equations (2.6)-(2.8) and the additional equation of the associated case from Equations (2.9) form a system of equations that can be solved analytically. We determine the solutions for each case using Maple 2021 as a symbolic solver and present them in the following:

Solution for case 1:

- Applied pattern: $(+a_{max}, 0, -a_{max})$
- Additional equation: $v_c = v_{max}$
- Solution:

$$t_1 = \frac{v_{max} - v_s}{a_{max}} \tag{2.10a}$$

$$t_2 = \frac{(p_e - p_s) \cdot a_{max} + \frac{1}{2}(v_e^2 + v_s^2) - v_{max}^2}{a_{max} \cdot v_{max}} \tag{2.10b}$$

$$t_3 = \frac{v_{max} - v_e}{a_{max}} \tag{2.10c}$$

$$v_c = +v_{max} \tag{2.10d}$$

Solution for case 2:

- Applied pattern: $(+a_{max}, 0, -a_{max})$
- Additional equation: $t_2 = 0$
- Solution:

$$t_1 = \frac{\pm \frac{1}{2}\sqrt{B^+} - v_s}{a_{max}} \tag{2.11a}$$

$$t_2 = 0 \tag{2.11b}$$

$$t_3 = \frac{\pm \frac{1}{2}\sqrt{B^+} - v_e}{a_{max}} \tag{2.11c}$$

$$v_c = \pm \frac{1}{2}\sqrt{B^+} \tag{2.11d}$$

with $B^+ := 4(p_e - p_s)a_{max} + 2(v_e^2 + v_s^2)$. As is typical for systems of equations with linear and quadratic parts, there are two solutions. The first solution is given by using the + sign, the second solution is represented by the − sign.

Solution for case 3:

- Applied pattern: $(-a_{max}, 0, +a_{max})$
- Additional equation: $v_c = -v_{max}$

- Solution:

$$t_1 = \frac{v_{max} + v_s}{a_{max}} \tag{2.12a}$$

$$t_2 = \frac{(p_s - p_e) \cdot a_{max} + \frac{1}{2}(v_e^2 + v_s^2) - v_{max}^2}{a_{max} \cdot v_{max}} \tag{2.12b}$$

$$t_3 = \frac{v_{max} + v_e}{a_{max}} \tag{2.12c}$$

$$v_c = -v_{max} \tag{2.12d}$$

Solution for case 4:

- Applied pattern: $(-a_{max}, 0, +a_{max})$
- Additional equation: $t_2 = 0$
- Solution:

$$t_1 = \frac{\pm \frac{1}{2}\sqrt{B^-} + v_s}{a_{max}} \tag{2.13a}$$

$$t_2 = 0 \tag{2.13b}$$

$$t_3 = \frac{\pm \frac{1}{2}\sqrt{B^-} + v_e}{a_{max}} \tag{2.13c}$$

$$v_c = \pm \frac{1}{2}\sqrt{B^-} \tag{2.13d}$$

with $B^- := 4(p_s - p_e)a_{max} + 2(v_e^2 + v_s^2)$. Again, there are two solutions. The first solution is given by using the + sign, the second solution is represented by the − sign.

It is not possible to say in advance which of the cases corresponds to the optimum solution for a given problem instance. Hence, all solutions must be evaluated as follows: Overall, there are six different candidates for the time-optimal solution. These six candidates are derived for each of the above cases while cases 2 and 4 yield two candidates due to their quadratic nature. The set of candidates is further reduced since each candidate has to fulfill the constraints $t_1, t_2, t_3 \geq 0$ and $-v_{max} \leq v_c \leq v_{max}$ to be a feasible one. Note that since the patterns are complementary, there is always at least one feasible candidate. The feasible candidate with the lowest sum $T_{opt} = t_1 + t_2 + t_3$ represents the time-optimal solution for a single axis with the total duration T_{opt}.

2.4. Time-Optimal Trajectory Generation in Multiple Dimensions

As introduced in Section 2.2.3 the SOTA procedure sometimes yields infeasible time-optimal trajectory durations as solutions. In the following, we show our general approach to determining time-optimal trajectories for multiple axes. First, we present acceleration patterns needed for our approach and discuss how to check whether these patterns can be synchronized with a particular trajectory duration T_{sync}. Second, we describe a general procedure to obtain the time-optimal duration.

2.4.1. Required Control Input Patterns

The content of this section is extracted from our work [26]. Here, we derive all required acceleration patterns and focus on the feasibility of synchronizing them for a single axis with a particular T_{sync}. To check feasibility of these patterns, we first define the considered acceleration patterns. On the one hand, we consider the acceleration patterns for time-optimality in a single axis given by $(+a, 0, -a)$ with $a \in \{-a_{max}, a_{max}\}$. These patterns are from now on called classical patterns. However, we further consider the patterns defined by $(+a, 0, +a)$ with $a \in \{-a_{max}, a_{max}\}$, which we denote as synchronization patterns. They are needed because classical patterns aim at finding the time-optimal behavior, however, they are not sufficient for synchronization with large values for T_{sync} in some cases. We give the following example for illustration (see Figure 2.4): It is assumed that an axis with $p_s = 0\,\text{m}, p_e = 1.75\,\text{m}$ and $v_s = 0\,\frac{\text{m}}{\text{s}}, v_e = 0.5\,\frac{\text{m}}{\text{s}}$ has to be synchronized with a large enough duration T_{sync}. Here, p_s, p_e, v_s and v_e describe the start and end position as well as the respective velocities.

The pattern yielding time-optimality is given by $(+a_{max}, 0, -a_{max})$, with the black line giving the overall time-optimal velocity profile. However, this pattern is only applicable as long as $T_{\text{sync}} \leq 4\,\text{s}$. When T_{sync} is increased, the segment of deceleration at the end of the velocity profile shortens in time (blue line) until it becomes zero for synchronization time $T_{\text{sync}} = 4$ (red line). If it is required to synchronize the axis with $T_{\text{sync}} > 4\,\text{s}$ using the classical pattern $(+a_{max}, 0, -a_{max})$, no solution can be found since it is not possible to meet the desired final velocity and area underneath the velocity profile at the same time without lowering the acceleration magnitude. In this case, the synchronization patterns $(+a, 0, +a)$, $a \in \{-a_{max}, a_{max}\}$ with two phases of acceleration pointing towards the same direction come into play (green line). With such a pattern it is possible to synchronize the axis with $T_{\text{sync}} > 4\,\text{s}$ while keeping the magnitude of the acceleration as specified. We define these patterns as synchronization patterns because they are only needed for the synchronization between different axes.

Synchronization Feasibility Check:
To guarantee synchronization feasibility of one axis with the trajectory time T_{sync}, one has to find a pattern from the set of classical and synchronization patterns where $t_1, t_2, t_3 \geq 0\,\text{s}$ and $-v_{max} \leq v(t) \leq v_{max}, \forall t \in \left[0, T_{\text{sync}}\right]$. We define these inequations as synchronization

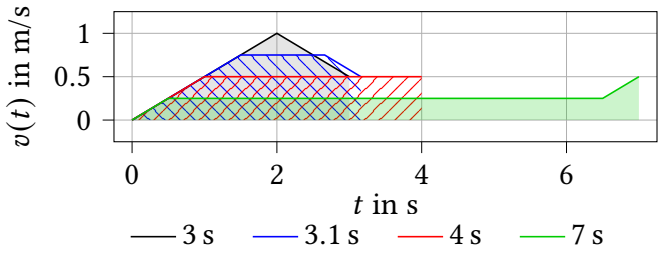


Figure 2.4.: Example of velocity profiles for different acceleration patterns and trajectory durations.

conditions. Note that if the initial and end velocities v_s and v_e are within the velocity bounds, it is sufficient to show that the constant velocity $v_c = v(t_1)$ in the segment of no acceleration is within the velocity bound to guarantee that $-v_{max} \leq v(t) \leq v_{max}$. The following subsections present how the values t_1, t_2, t_3, v_c are determined for classical and synchronization patterns.

2.4.1.1. Analysis of Classical Patterns

The classical patterns are defined by the acceleration profile

$$a(t) = \begin{cases} +a, & 0 \leq t < t_1 \\ 0, & t_1 \leq t < t_1 + t_2 \\ -a, & t_1 + t_2 \leq t \leq t_1 + t_2 + t_3. \end{cases} \tag{2.14}$$

with $a \in \{a_{max}, -a_{max}\}$. Based on this acceleration profile, the velocity profile results as

$$v(t) = \begin{cases} v_s + at, & 0 \leq t < t_1 \\ v_s + at_1, & t_1 \leq t < t_1 + t_2 \\ v_s + a(2t_1 + t_2) - at & t_1 + t_2 \leq t \leq t_1 + t_2 + t_3. \end{cases} \tag{2.15}$$

In order to meet the required velocity v_e at time $t_1 + t_2 + t_3$ the following equation has to hold:

$$\begin{aligned} v_e - v_s &= \int_0^{t_1+t_2+t_3} a(t)\mathrm{d}t \\ &= at_1 - at_3 \end{aligned} \tag{2.16}$$

To meet the required position p_e at time $t_1 + t_2 + t_3$ the equation

$$\begin{aligned} p_e - p_s &= \int_0^{t_1+t_2+t_3} v(t)\mathrm{d}t \\ &= v_s t_1 + \frac{1}{2}at_1^2 + (v_s + at_1)t_2 + (v_s + at_1)t_3 - \frac{1}{2}at_3^2 \end{aligned} \tag{2.17}$$

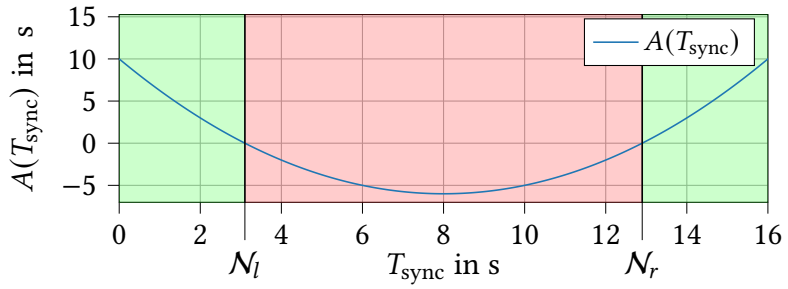


Figure 2.5.: Graph of A depending on T_{sync} for $a = -0.5\,\text{m/s}^2, p_s = 0\,\text{m}, v_s = 2\,\text{m/s}, p_e = 5\,\text{m}, v_e = 2\,\text{m/s}^2$. The area where $A \geq 0$ is colored in green. The gap between $\mathcal{N}_l$ and $\mathcal{N}_r$ is given in red.

has to hold as well. Additionally, the equation

$$t_1 + t_2 + t_3 = T_{\text{sync}} \tag{2.18}$$

has to hold to guarantee time synchronization. Further, since the velocity v_c for $t \in [t_1, t_2]$ is constant, the equation

$$v_c = at_1 + v_s \tag{2.19}$$

applies as well.

Equations (2.16), (2.17), (2.18) and (2.19) form a system of equations with variables t_1, t_2, t_3, v_c whose solution is given by

$$t_1 = \frac{aT_{\text{sync}} + v_e - v_s \pm \sqrt{A}}{2a} \tag{2.20a}$$

$$t_2 = \mp\frac{\sqrt{A}}{a} \tag{2.20b}$$

$$t_3 = \frac{aT_{\text{sync}} - v_e + v_s \pm \sqrt{A}}{2a} \tag{2.20c}$$

$$v_c = \frac{aT_{\text{sync}} + v_s + v_e + \sqrt{A}}{2} \tag{2.20d}$$

with

$$A(T_{\text{sync}}) = a^2T_{\text{sync}}^2 + 2(v_e + v_s)T_{\text{sync}} - 4a(x_e - x_s) - (v_e - v_s)^2. \tag{2.21}$$

Equations (2.20a), (2.20b), (2.20c) and (2.20d) only depend on the trajectory duration T_{sync}. Hence, it is sufficient to insert T_{sync} into these equations and verify that $t_1(T_{\text{sync}})$, $t_2(T_{\text{sync}})$, $t_3(T_{\text{sync}}) \geq 0$ and $v_{min} \leq v_c(T_{\text{sync}}) \leq v_{max}$ to show synchronization feasibility for the respective classical pattern.

Cause of the missing synchronizability for the SOTA approach:
As described above, the SOTA makes use of only two different control input patterns $p_1 = (+a_{max}, 0, -a_{max})$ and $p_2 = (-a_{max}, 0, +a_{max})$ for a single axis to determine the optimal solution to the TOT-PMAV and assumes that it is always possible to synchronize the i-th axes with any potential trajectory synchronization time $T_{\text{sync}} \in \mathbb{R}^+$ for which holds $T_{\text{sync}} \geq T_{opt,i}$. However, this assumption does not generally apply.

According to our work [26], we derive that each of the two patterns comes with a feasible domain $\mathcal{D}_{p_1}, \mathcal{D}_{p_2}$ that describes for a single axes which trajectory duration T_{sync} can be realized. For input pattern p_1, i.e. for $a = a_{max}$, this is

$$\mathcal{D}_{p_1} \coloneqq \{T_{\text{sync}} \in \mathbb{R}^+ \mid \tag{2.22a}$$

$$A(T_{\text{sync}}) \geq 0, \tag{2.22b}$$

$$\frac{aT_{\text{sync}} + v_e - v_s - \sqrt{A(T_{\text{sync}})}}{2a} \geq 0, \tag{2.22c}$$

$$+ \frac{\sqrt{A(T_{\text{sync}})}}{a} \geq 0, \tag{2.22d}$$

$$\frac{aT_{\text{sync}} - v_e + v_s - \sqrt{A(T_{\text{sync}})}}{2a} \geq 0, \tag{2.22e}$$

$$- v_{max} \leq \frac{aT_{\text{sync}} + v_e + v_s - \sqrt{A(T_{\text{sync}})}}{2} \leq v_{max}\} \tag{2.22f}$$

and for input pattern p_2, i.e. for $a = -a_{max}$, the domain is

$$\mathcal{D}_{p_2} \coloneqq \{T_{\text{sync}} \in \mathbb{R}^+ \mid \tag{2.23a}$$

$$A(T_{\text{sync}}) \geq 0, \tag{2.23b}$$

$$\frac{aT_{\text{sync}} + v_e - v_s + \sqrt{A(T_{\text{sync}})}}{2a} \geq 0, \tag{2.23c}$$

$$- \frac{\sqrt{A(T_{\text{sync}})}}{a} \geq 0, \tag{2.23d}$$

$$\frac{aT_{\text{sync}} - v_e + v_s + \sqrt{A(T_{\text{sync}})}}{2a} \geq 0, \tag{2.23e}$$

$$- v_{max} \leq \frac{aT_{\text{sync}} + v_e + v_s - \sqrt{A(T_{\text{sync}})}}{2} \leq v_{max}\}. \tag{2.23f}$$

with $A(T_{\text{sync}})$ as defined in Equation (2.21).

As can be seen, both domains $\mathcal{D}_{p_1}$ and $\mathcal{D}_{p_2}$ contain a quadratic constraint (2.22b) and (2.23b) which ensures that the square root in the other constraints can be calculated and which represents a parabola that opens upwards. Under certain conditions, the vertex of the parabola is in the fourth quadrant of the cartesian plane which is why there might be two zeros

$$\mathcal{N}_l, \mathcal{N}_r = \frac{-v_e - v_s \pm \sqrt{2(v_e^2 + v_s^2) + 4a(p_e - p_s)}}{a} \tag{2.24}$$

with $\mathcal{N}_l, \mathcal{N}_r > 0$ which might split the feasible domain into two disjunct continuous sets with a domain gap in between as is the case in Figure 2.2. One part is left of the left zero $\mathcal{N}_l$ and the other part is on the right zero $\mathcal{N}_r$. The curve of $A(T_{\text{sync}})$ and its zeros for the example illustrated in Figure 2.2 are given in Figure 2.5. For any T_{sync} within the gap in $(\mathcal{N}_l, \mathcal{N}_r)$, the square root $\sqrt{A(T_{\text{sync}})}$ in the definition of the domains cannot be determined as a real number and hence, it is impossible to synchronize the axis with the corresponding

pattern. Since the SOTA only relies on patterns p_1 and p_2 and does not account for these domain gaps, it sometimes yields an invalid solution. Note that domain gaps do not depend on v_{max} and hence, also exist for unbounded maximum velocities.

2.4.1.2. Analysis of Synchronization Patterns

Synchronization feasibility for the synchronization patterns is checked analogously. The only difference is in the applied acceleration pattern

$$a(t) = \begin{cases} +a, & 0 \leq t < t_1 \\ 0, & t_1 \leq t < t_1 + t_2 \\ +a, & t_1 + t_2 \leq t \leq t_1 + t_2 + t_3 \end{cases} \tag{2.25}$$

with $a \in \{a_{max}, -a_{max}\}$, which results in the following velocity profile

$$v(t) = \begin{cases} v_s + at, & 0 \leq t < t_1 \\ v_s + at_1, & t_1 \leq t < t_1 + t_2 \\ v_s + a(t - t_2) & t_1 + t_2 \leq t \leq t_1 + t_2 + t_3. \end{cases} \tag{2.26}$$

Analogously, this leads to a system of four equations and four variables whose solution is given by

$$t_1 = \frac{(-2v_s T_{\text{sync}} + 2(p_e - p_s))a - (v_e - v_s)^2}{2a(T_{\text{sync}}a - v_e + v_s)} \tag{2.27a}$$

$$t_2 = \frac{aT_{\text{sync}} - v_e + v_s}{a} \tag{2.27b}$$

$$t_3 = \frac{(-2v_e T_{\text{sync}} - 2(p_e - p_s))a - (v_e - v_s)^2}{2a(T_{\text{sync}}a - v_e + v_s)} \tag{2.27c}$$

$$v_c = \frac{2a(p_e - p_s) - v_e^2 + v_s^2}{2(aT_{\text{sync}} - v_e + v_s)}. \tag{2.27d}$$

Again, Equations (2.27a), (2.27b), (2.27c) and (2.27d) only depend on T_{sync} and the synchronization feasibility can easily be checked via insertion.

Defining the control input patterns $p_3 = (+a_{max}, 0, +a_{max})$ and $p_4 = (-a_{max}, 0, -a_{max})$, we can state the corresponding domains of feasible trajectory durations for both patterns which can be derived from Equations (2.27a) - (2.27d) as

$$\mathcal{D}_{p_3} \coloneqq \{T_{\text{sync}} \in \mathbb{R}^+ \mid \tag{2.28a}$$
$$-2a(v_s T_{\text{sync}} - p_e + p_s) - (v_e - v_s)^2 \lesseqgtr 0, \tag{2.28b}$$
$$aT_{\text{sync}} - v_e + v_s \leq 0, \tag{2.28c}$$
$$-2a(-v_e T_{\text{sync}} + p_e - p_s) - (v_e - v_s)^2 \lesseqgtr 0, \tag{2.28d}$$
$$2a(p_e - p_s) - v_e^2 + v_s^2 \geq -2v_{max}(aT_{\text{sync}} - v_e + v_s) \tag{2.28e}$$
$$2a(p_e - p_s) - v_e^2 + v_s^2 \leq 2v_{max}(aT_{\text{sync}} - v_e + v_s)\} \tag{2.28f}$$

and

$$\mathcal{D}_{p_4} \coloneqq \{T_{\text{sync}} \in \mathbb{R}^+ \mid \tag{2.29a}$$
$$-2a(v_s T_{\text{sync}} - p_e + p_s) - (v_e - v_s)^2 \lesseqgtr 0, \tag{2.29b}$$
$$aT_{\text{sync}} - v_e + v_s \leq 0, \tag{2.29c}$$
$$-2a(-v_e T_{\text{sync}} + p_e - p_s) - (v_e - v_s)^2 \lesseqgtr 0, \tag{2.29d}$$
$$2a(p_e - p_s) - v_e^2 + v_s^2 \leq -2v_{max}(aT_{\text{sync}} - v_e + v_s) \tag{2.29e}$$
$$2a(p_e - p_s) - v_e^2 + v_s^2 \geq 2v_{max}(aT_{\text{sync}} - v_e + v_s)\} \tag{2.29f}$$

with $a = +a_{max}$ for $\mathcal{D}_{p_3}$ and $a = -a_{max}$ for $\mathcal{D}_{p_4}$. The right direction of the Inequations (2.28b), (2.28d), (2.29b), and (2.29d) depends on the values of p_s, p_e, v_s, v_e, a and v_{max}.

2.4.2. Structural Analysis of Optimal Solutions to the TOT-PMAV

The content of this section is entirely extracted from our work [26]. According to [26] the TOP-UAV solution approach consists of four different control input patterns that can be applied on a single axis. The set containing all these control input patterns is described by

$$\begin{aligned}\mathcal{P} = \{&(+a_{max}, 0, -a_{max}), (-a_{max}, 0, +a_{max}),\\ &(+a_{max}, 0, +a_{max}), (-a_{max}, 0, -a_{max})\}\\ = \{&p_1, p_2, p_3, p_4\}.\end{aligned}$$

with p_1 and p_2 describing the classical patterns to achieve time-optimality and p_3 and p_4 as synchronization patterns that are required in some cases when the duration of an axis has to be increased significantly for axis-synchronization (see Section 2.4.1) and which serve the same purpose as lowering the applied maximum acceleration.

The domains for a single axis for patterns p_1 and p_2 are described by $\mathcal{D}_{p_1}$ and $\mathcal{D}_{p_2}$ as defined in Equations (2.22) and (2.23). The associated domains for pattern p_3 and p_4 are given in Equations (2.28) and (2.29).

To determine the overall time-optimal trajectory from an initial state to the final state in multiple dimensions with the respective kinematic constraints, we have to consider a set of possible pattern combinations Φ with

$$\Phi = \{\boldsymbol{\phi} = (\phi_1, \phi_2, ..., \phi_n) \mid \phi_i \in \mathcal{P}, i = 1, ..., n\}$$

and $(\phi_1, \phi_2, ..., \phi_n)$ describing an n-tuple with the i-th entry representing the applied pattern for the i-th axis of the trajectory.

For each $\boldsymbol{\phi} \in \Phi$ the resulting optimization problem is to find the lowest feasible synchronization time for the associated pattern combination, which is

$$T_{\text{sync}}^{\boldsymbol{\phi}} = \min T_{\text{sync}} \tag{2.30a}$$

$$s.t. \quad T_{\text{sync}} \in \mathcal{D}_{\phi_i} \quad \forall i = 1, ..., n \tag{2.30b}$$

Here, $\phi_i \in \mathcal{P}$ denotes the pattern of the i-th axis of pattern combination $\boldsymbol{\phi}$.

The overall time-optimal trajectory is obtained by solving

$$T_{opt}^{*} = \min_{\boldsymbol{\phi} \in \Phi} T_{\text{sync}}^{\boldsymbol{\phi}} \tag{2.31}$$

with $T_{\text{sync}}^{\boldsymbol{\phi}} = +\infty$ if no feasible solution can be found for the associated pattern combination $\boldsymbol{\phi}$.

Lemma 1 *If a pattern combination $\boldsymbol{\phi}$ only consists of patterns p_3 and p_4, then the resulting optimization problem described in* (2.30a) - (2.30b) *represents a linear program (LP).*

Proof 1 *Since the respective domains $\mathcal{D}_{p_3}$ and $\mathcal{D}_{p_4}$ only contain linear inequations and the objective function is linear, the problem in* (2.30a) - (2.30b) *is an LP by definition.*

Theorem 1 *If a pattern combination $\boldsymbol{\phi}$ consists of patterns p_1, p_2, p_3, p_4, then the resulting optimization problem defined in* (2.30a) - (2.30b) *represents a mixed-integer linear program (MILP).*

Proof 2 *To prove the theorem, we start by only investigating the i-the axis which applies pattern p_1 and assume that the remaining axes apply the patterns p_3 or p_4. All subsequent mathematical conversions are conducted using Maple 2021. Assume, for the i-th axis applying pattern p_1 holds $-2(v_e^2 + v_s^2) - 4a(p_e - p_s) \geq 0 \Rightarrow A(T_{sync}) \geq 0$, $\forall T_{sync} \in \mathbb{R}$. In this case, Inequations* (2.22b) *and* (2.22d) *can be neglected since they are always fulfilled. Solving Inequation* (2.22c) *for T_{sync} yields*

$$T_{sync} \lessgtr \frac{v_s^2 - 2v_e v_s + 2a(p_e - p_s) + v_e^2}{2av_s},$$

while the correct direction of the inequation sign depends on the value of the parameters p_s, p_e, v_s, v_e. *Analogously, Inequation* (2.22e) *can be represented by*

$$T_{sync} \lesseqgtr \frac{v_s^2 - 2v_e v_s + 2a(p_e - p_s) + v_e^2}{2av_e}$$

and Inequations (2.22f) *by*

$$T_{sync} \lesseqgtr \frac{v_s^2 + 2v_{max}(v_e - v_{max}) + 2a(p_e - p_s) + v_e^2}{-2av_{max}}$$
$$T_{sync} \lesseqgtr \frac{v_s^2 - 2v_{max}(v_e + v_{max}) + 2a(p_e - p_s) + v_e^2}{2av_{max}}.$$

Since $\mathcal{D}_{p_1}$ *can be expressed by only linear inequations in this case and the inequations of* $\mathcal{D}_{p_3}, \mathcal{D}_{p_4}$ *as well as the objective function are linear, the problem in* (2.30a) - (2.30b) *is an LP which is a subclass of MILP.*

If $-2(v_e^2 + v_s^2) - 4a(p_e - p_s) \geq 0$ *does not hold, Inequation* (2.22b) *cannot be neglected and the optimization problem remains nonlinear. However, there are two zeros* $\mathcal{N}_l$ *and* $\mathcal{N}_r$ *for Inequation* (2.22b) *as shown in Equation* (2.24). *Consequently, the domain of feasible* T_{sync} *gets split into two disjunctive parts* $T_{sync} \leq \mathcal{N}_l$ *or* $T_{sync} \geq \mathcal{N}_r$. *Knowing this and according to [69], the quadratic constraints* (2.22b) *can be expressed as a set of disjunctive constraints and modeled by the following big-M formulation*

$$\begin{aligned} & z_k(T_{sync}) \leq M_k(1 - y_k) \quad \forall k \in \{l, r\} \\ & \sum_{k \in \{l,r\}} y_k = 1 \\ & y_k \in \{0, 1\} \quad \forall k \in \{l, r\} \end{aligned}$$

with $M_k \in \mathbb{R}^+, k \in \{l, r\}$ *representing a sufficiently large number and* $z_k(T_{sync}), k \in \{l, r\}$ *referring to the equations*

$$\begin{aligned} z_l(T_{sync}) &= T_{sync} - \mathcal{N}_l \\ z_r(T_{sync}) &= \mathcal{N}_r - T_{sync}. \end{aligned}$$

Further, it holds for $k \in \{l, r\}$ *that*

$$y_k = \begin{cases} 1, & \text{if constraint } z_k(T_{sync}) \leq 0 \text{ is applied} \\ 0, & \text{otherwise.} \end{cases}$$

With this reformulation, we can algebraically express that either the part left of $\mathcal{N}_l$ *or right of* $\mathcal{N}_r$ *is used in the optimum solution, never both. Further, Inequation* (2.22d) *can be neglected again since it is always fulfilled for all* $T_{sync} \leq \mathcal{N}_l$ *and* $T_{sync} \geq \mathcal{N}_r$, *and Inequations* (2.22c), (2.22e), *and* (2.22e) *can be reformulated into linear constraints as stated in the first case. However, with this reformulation, we have introduced a new integer decision variable* y_k, *indicating if* $T_{sync} \leq \mathcal{N}_l$ *or* $T_{sync} \geq \mathcal{N}_r$ *holds for the affected axis. Hence, a MILP results.*

The derivation for applying pattern p_2 on the i-th axis while the remaining axis apply pattern p_3 and p_4 is analogous. In conclusion, each axis that applies pattern p_1 or p_2 adds linear constraints to the problem (2.30a) - (2.30b) *as well as integer constraints under special circumstances and therefore, results in a MILP.*

According to Theorem 1 the optimization problem (2.30a) - (2.30b) can be described as MILP for any pattern combination $\boldsymbol{\phi} \in \Phi$. MILPs are solved exactly by branch and bound approaches where each leaf of the resulting branch and bound tree represents an LP. For each LP, the optimal solution on the non-integer variable T_{sync} can be found on the boundary of the domain defined by the non-integer constraints. This is where at least one of the inequations of domains $\mathcal{D}_{p_1}, ..., \mathcal{D}_{p_4}$ is fulfilled with equality.

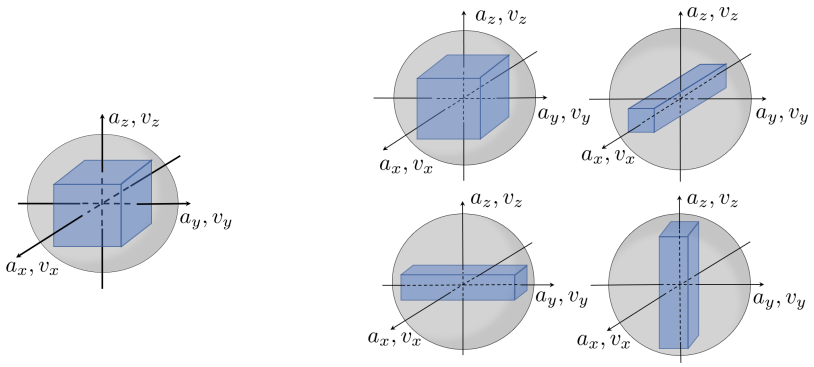
2.4.3. General Algorithmic Framework TOP-UAV [26]

As described in the previous section the solution of the TOT-PMAV has to be on the boundary of the domains $\mathcal{D}_{p_1}, ..., \mathcal{D}_{p_4}$ derived for each axis, which is where at least one of the constraints of any domain and axis is fulfilled with equality. Hence, solving each constraint of each domain and each axis for T_{sync} yields a list of potential solution candidates that must contain the overall optimum solution T^*_{opt} described in the overall optimization problem in Equation (2.31). From this list only those values $T_{\text{sync}} \geq \max_{i \in \{1,...,n\}} \{T_{opt,i}\}$ are relevant. The reason for this is that the solution of the SOTA approach serves as a physical lower bound of the trajectory duration since the synchronization of all axes cannot be faster than a time-optimal solution of the slowest axis. All the remaining candidates must then explicitly or implicitly be checked for feasibility. This means that for each axis at least one pattern must be found that is feasible for the given candidate T_{sync}. This is done by inserting the candidate into Equations (2.20a) - (2.20d) for the classical patterns and Equations (2.27a) - (2.27d) for the synchronization patterns and checking if the conditions $t_1(T_{\text{sync}}), t_2(T_{\text{sync}}), t_3(T_{\text{sync}}) \geq 0$ and $v_{min} \leq v_c(T_{\text{sync}}) \leq v_{max}$ hold. The lowest feasible candidate represents the overall optimum trajectory solution. Since the $t_1(T_{\text{sync}}), t_2(T_{\text{sync}}), t_3(T_{\text{sync}})$ are already determined during the feasibility check, they are used to derive the full control input trajectory and hence the entire trajectory representation.

We provide the source code for our TOP-UAV trajectory planner as open source on GitHub in C++ (see [67]) and Python (see [68]) using $version = "basic"$ as input argument when instantiating a trajectory planner object.

2.4.4. Improved Algorithmic Framework TOP-UAV++ [27]

In many approaches (see [56, 64, 26, 30, 29]), the kinematic velocity and acceleration constraints for n-dimensional time-optimal trajectory generation are fixed to a single value for each axis according to Equations (2.1) and (2.2). The aggregation of the n constraints for the velocity and acceleration of each axis forms an n-dimensional cuboid for feasible



(a) Distribution of the maximum velocity and acceleration norm for the SOTA approach and TOP-UAV.

(b) Distribution of the maximum velocity and acceleration norm for TOP-UAV++.

Figure 2.6.: Configurations for distributing the maximum velocity and acceleration among each coordinate axis for SOTA approach and TOP-UAV (see (a)) and for TOP-UAV++ (see (b)).

values of $\boldsymbol{v}(t)$ and $\boldsymbol{a}(t)$ which lies strictly within an n-dimensional sphere that represents the actual kinematic constraints $||\boldsymbol{v}(t)|| \leq \hat{v}_{max}$ and $||\boldsymbol{a}(t)|| \leq \hat{a}_{max}$ (see Fig. 2.6a) and which is defined by the problem definition of the TOT-PMAV in Section 2.2.2.1.

Algorithm 1: TOP-UAV^{++} trajectory planner

```
1  T*, C, a*_1(t), ..., a*_n(t) ← initialize()
2  forall c ∈ C do
3      T^c_SOTA ← max_{i∈{1,...,n}}{T^c_opt,i}        // see Section 2.3 for calculation of T^c_opt,i
4      if T^c_SOTA < T* then
5          feasible, φ = checkFeasible(T^c_SOTA, c)
6          if feasible then
7              T* ← T^c_SOTA
8              a*_1(t), ..., a*_n(t) ← updateAccelerationProfile(T^c_SOTA, φ, c)
9              continue
10         S ← calculateCandidatesSorted(T^c_SOTA, T*, c)
11         forall T^c_cand ∈ S do
12             feasible, φ = checkFeasible(T^c_cand, c)
13             if feasible then
14                 T* ← T^c_cand
15                 a*_1(t), ..., a*_n(t) ← updateAccelerationProfile(T^c_cand, φ, c)
16                 break
17 return: T*, a*_1(t), ..., a*_n(t)
```

However, using a single cuboid representation of the kinematic constraints cuts off a significant part of the feasible kinematics as indicated in [65, 66]. To improve the exploitation of the full kinematic capabilities, we propose to consider a predefined set of different

configurations of these cuboid-like constraints with each configuration complying with the actual kinematic restrictions (see Fig. 2.6b).

Based in this idea, we improve the TOP-UAV solver first described in [26] to TOP-UAV++ as described in Algorithm 1 and published in our paper [27]. In the algorithm, we initialize the optimal trajectory duration $T^* = \infty$ and the set of possible configurations C to distribute the maximum velocity and acceleration norm among the axes as illustrated in Figure 2.6b. Further, we initialize optimal acceleration profiles for each axis. This initialization procedure is given in line 1 in Algorithm 1. Next, we iterate over each configuration $c \in C$ and compute the time-optimal trajectory for each configuration (see lines 2 - 16). According to the SOTA approach, this is done by calculating the time-optimal trajectory duration of each specific axis $T^c_{opt,i}$ and determining the maximum value T^c_{SOTA} (line 3). Thereafter, we check if $T^c_{\text{SOTA}} < T^*$ since T^c_{SOTA} serves as a lower bound for all solution candidates of configuration c (line 4). If this is not the case, we have already found a better or equivalent solution for a previously investigated configuration than would be possible with configuration c and we move to the next configuration. Otherwise, we check if and at what pattern combination $\boldsymbol{\phi}$ all axes can be synchronized with T^c_{SOTA} by using the method **checkFeasible** (line 5). This method checks if for all axes at least one pattern exists for which $t_1, t_2, t_3 \geq 0$ and $-v_{max,i} \leq v_{c,i} \leq v_{max,i}$ holds $\forall i$, which is analogous to our TOP-UAV approach described in Section 2.4.3. If this is true, we update $T^*, a_1^*(t), ..., a_n^*(t)$ according to $T^c_{\text{SOTA}}, \boldsymbol{\phi}, c$ and move to the next configuration (lines 6 - 9). If not, we calculate all solution candidates that are greater than T^c_{SOTA} but less than T^*, sort them in ascending order (line 10) and iteratively check for feasibility (lines 11 - 12). We update $T^*, a_1^*(t), ..., a_n^*(t)$ with the first feasible candidate T^c_{cand} and move to the next configuration (lines 13 - 16). Finally, we return $T^*, a_1^*(t), ..., a_n^*(t)$ representing the optimum solution including the minimum trajectory duration and the corresponding acceleration profile for each axis (line 17).

Note that the difference between TOP-UAV (introduced in [26]) and TOP-UAV++ (introduced in [27]) is, that the TOP-UAV only uses a single configuration of equally distributed maximum velocity and acceleration shares, while TOP-UAV++ uses in general multiple configurations. Hence, the procedure described in lines 3 - 16 in Algorithm 1 describes the pseudocode for TOP-UAV using the standard configuration.

We provide the source code for our TOP-UAV++ trajectory planner as open source on GitHub in C++ (see [67]) and Python (see [68]) using $version = "improved"$ as input argument when instantiating a trajectory planner object.

2.5. Computational Study

In the following Sections, we focus on the complementary research questions described in Section 2.2.3 by conducting an extensive computational study. We investigate in which situations the SOTA approach is likely to yield an infeasible time-optimal trajectory duration and show the consequences of the infeasibility when using the invalid duration

for axis synchronization. Further, we compare the computational performance of TOP-UAV and TOP-UAV++ with the SOTA approach described in Section 2.2.2.2 in two and three dimensions and show the impact of our TOP-UAV++ approach to improve the kinematic exploitation.

2.5.1. Computational Study Setup

In the two-dimensional case and for the basic version of our trajectory generation, namely TOP-UAV, the maximum allowed acceleration for each axis is limited according to the SOTA to $a_{max,i} = \hat{a}_{max}/\sqrt{2}$, which guarantees that the composed acceleration over all axes does not exceed the total maximum acceleration $\hat{a}_{max}$. Analogously, we limit the maximum velocity of each axis to $v_{max,i} = \hat{v}_{max}/\sqrt{2}$. For the TOP-UAV++ version in two dimensions, we use two additional kinematic configurations. These cover the improved kinematic exploitation along a specific coordinate axis and use $v_{max,i} = \hat{a}_{max}\sqrt{3}/2$ and $a_{max,i} = \hat{a}_{max}\sqrt{3}/2$ for the specified axis while the maximum velocity and acceleration of the other axis is restricted to $v_{max,i} = \hat{v}_{max}/2$ and $a_{max,i} = \hat{a}_{max}/2$. Note that each of the configurations spans a rectangle that fits into the circle with a radius $\hat{v}_{max}$ for the maximum velocity and $\hat{a}_{max}$ for the maximum acceleration restriction.

For the TOP-UAV trajectory planner in three dimensions, the maximum allowed acceleration for each axis is limited according to the SOTA to $a_{max,i} = \hat{a}_{max}/\sqrt{3}$. Further, we limit the maximum velocity of each axis to $v_{max,i} = \hat{v}_{max}/\sqrt{3}$. For the TOP-UAV++, we use three additional kinematic configurations. Analogously, these cover improved kinematic exploitation along a specific coordinate axis and use $v_{max,i} = \hat{v}_{max}\sqrt{3}/2$ and $a_{max,i} = \hat{a}_{max}\sqrt{3}/2$ for the specified axis while the maximum velocity and acceleration of the remaining axis are restricted to $v_{max,i} = \hat{v}_{max}/\sqrt{8}$ and $a_{max,i} = \hat{a}_{max}/\sqrt{8}$. Note that in the three-dimensional case, each of the configurations spans a cubic that fits into the sphere with a radius $\hat{v}_{max}$ for the maximum velocity and $\hat{a}_{max}$ for the maximum acceleration restriction.

For both, the two- and three-dimensional versions of TOP-UAV++, any set of configurations C to share the maximum velocity and acceleration norm among the coordinate axes can be used in general. Provided the actual kinematic constraints $||\boldsymbol{v}(t)|| \leq \hat{v}_{max}$ and $||\boldsymbol{a}(t)|| \leq \hat{a}_{max}$ are fulfilled for each configuration $c \in C$.

All our experiments were implemented in C++17 and ran on an Intel Core i7-8565U CPU and 16 GB of RAM. For the remainder of this chapter, the term randomly sampled always refers to a randomly sampled from a uniform distribution.

2.5.2. Occurance of Insynchronizabilities

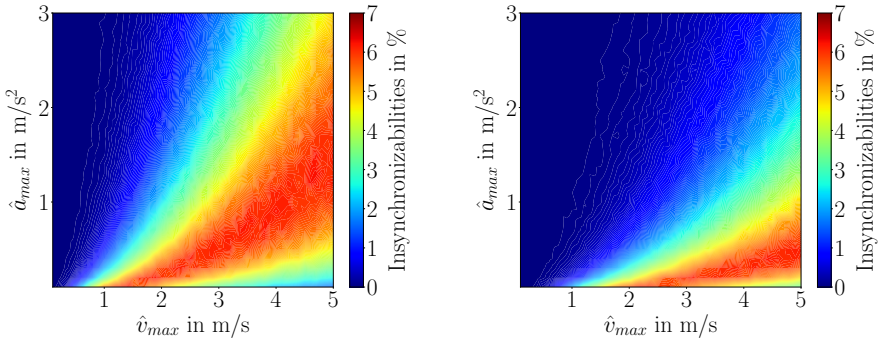
In this section, we show in which situations the SOTA approach described in Section 2.2.2.2 is likely to yield infeasible trajectory durations when solving the TOT-PMAV. For this, we utilized the TOP-UAV approach and the corresponding SOTA approach with maximum velocity and acceleration equally shared among the axes. Further, we investigate four

spatial settings of which two settings are related to two dimensions and another two settings are related to three dimensions.

In the first setting, we use randomly sampled start and end positions in a $(5\,\mathrm{m})^2$ spatial square. In the second setting, the start and end positions are randomly sampled in $(15\,\mathrm{m})^2$. For both settings, start and end velocities of each axis are randomly sampled within the interval $[-\hat{v}_{max}/\sqrt{2}, +\hat{v}_{max}/\sqrt{2}]$. In the third setting, we use randomly sampled start and end positions in a $(5\,\mathrm{m})^3$ spatial cubic. In the fourth setting, the start and end positions are randomly sampled in a $(15\,\mathrm{m})^3$ spatial cubic. For the last two settings, the start and end velocities of each axis are randomly sampled within the interval $[-\hat{v}_{max}/\sqrt{3}, +\hat{v}_{max}/\sqrt{3}]$. The waypoints and associated velocities for all spatial settings are randomly sampled from a uniform distribution.

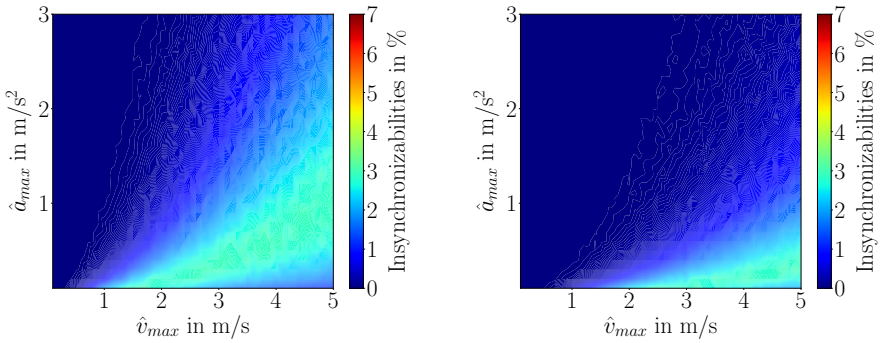
We discretize the overall maximum allowed velocity $\hat{v}_{max}$ between 0.1 m/s and 5.0 m/s with 0.1 m/s steps in-between as well as the maximum allowed acceleration $\hat{a}_{max}$ between $0.1\,\mathrm{m/s^2}$ and $3.0\,\mathrm{m/s^2}$ with $0.1\,\mathrm{m/s^2}$ steps in-between and randomly generate 100,000 trajectories for each combination of discretized maximum velocity and acceleration for each spatial setting. Figures 2.7 and 2.8 show the percentage of how many times the SOTA approach yields lower axis-synchronization times T_{SOTA} than the TOP-UAV approach $T_{\text{TOP-UAV}}$. This is the case if the SOTA yields an infeasible solution. In both three-dimensional settings, it can be seen that especially for increasing maximum velocities the probability of invalidities from the SOTA increases to up to 4%. The probability is even higher for both two-dimensional settings where it reaches up to 7%. This observation can be illustrated by Equation (2.21) since increasing start and end velocities move the vertex of $A(T_{\text{sync}})$ downwards which immediately leads to a larger domain gap $T_{\text{sync}} \in (\mathcal{N}_l, \mathcal{N}_r)$ for which $A(T_{\text{sync}}) < 0$ holds. Further, decreasing the maximum acceleration also leads to an increasing percentage of invalidities. According to Equation (2.21), increasing the maximum acceleration comes with two opposing effects. On the one hand, it elongates the opening of the parabola which leads to larger domain gaps. However, it also lifts the vertex upwards, which leads to smaller domain gaps. According to the observation of our study, we assume that the first effect dominates in the majority of cases. However, we cannot mathematically prove, that this holds for all cases. The latter might be the reason, why the percentage of insynchronizable trajectory durations increases again for very low maximum accelerations.

Having a detailed look at the results for the two-dimensional case presented in Figure 2.9 reveals even more information. Here, we focus on the 100,000 trajectories generated with $(\hat{v}_{max} = 1\,\mathrm{m/s}, \hat{a}_{max} = 1\,\mathrm{m/s^2})$, $(\hat{v}_{max} = 1\,\mathrm{m/s}, \hat{a}_{max} = 3\,\mathrm{m/s^2})$, $(\hat{v}_{max} = 5\,\mathrm{m/s}, \hat{a}_{max} = 1\,\mathrm{m/s^2})$, $(\hat{v}_{max} = 5\,\mathrm{m/s}, \hat{a}_{max} = 3\,\mathrm{m/s^2})$ that we also used to create Figure 2.7a. The figures show a heatmap plot of the optimum trajectory duration of the x and y-axis of all insynchronizable trajectories for the corresponding scenarios classified into bins of size 1 s. The color indicates the percentage of all insynchronizable trajectories that are assigned to the same bin. Viewed qualitatively, it can be seen that the SOTA approach yields infeasible trajectory durations, particularly in scenarios with one fast axis and one slower one. This observation is in line with the discovery that fast axes are likely to overshoot the required final state due to high initial velocities and due to inertia (see Section 2.2.3). Moreover,



(a) Start/end position in $(5\,\mathrm{m})^2$. **(b)** Start/end position in $(15\,\mathrm{m})^2$.

Figure 2.7.: Percentage of insynchronizable two-dimensional trajectories with the SOTA over $\hat{v}_{max}$ and $\hat{a}_{max}$.



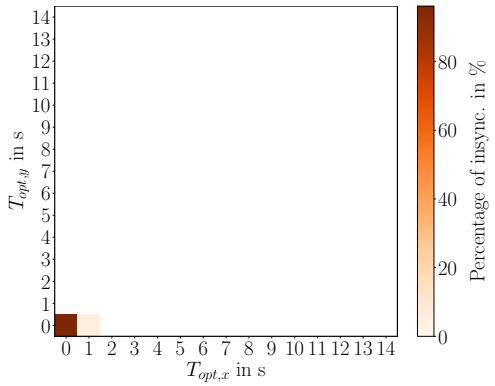
(a) Start/end position in $(5\,\mathrm{m})^3$. **(b)** Start/end position in $(15\,\mathrm{m})^3$.

Figure 2.8.: Percentage of insynchronizable three-dimensional trajectories with the SOTA over $\hat{v}_{max}$ and $\hat{a}_{max}$.

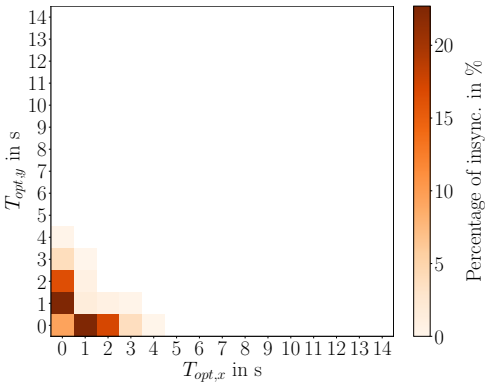
Figures 2.9c - 2.9b show that high velocities in combination with low acceleration power yield the highest probability of insynchronizability.

2.5.3. Extent of Discrepancy between SOTA and TOP-UAV

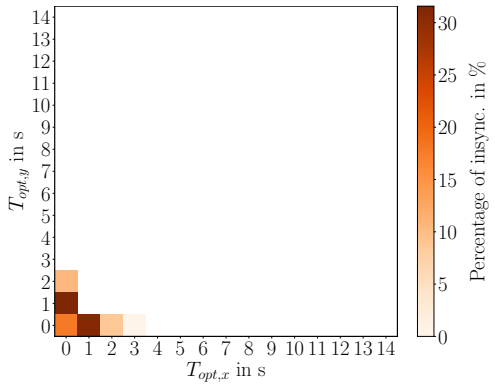
In this section, we examine the extent of the use of invalid solutions of the SOTA approach for axis-synchronization that arise for a subset of the experiments described in Section 2.5.2. For this investigation, we utilize the four different kinematic samples of maximum velocities and accelerations $\{(\hat{v}_{max} = 2\,\mathrm{m/s}, \hat{a}_{max} = 2\,\mathrm{m/s^2}), (\hat{v}_{max} = 2\,\mathrm{m/s}, \hat{a}_{max} = 1\,\mathrm{m/s^2}), (\hat{v}_{max} = 4\,\mathrm{m/s}, \hat{a}_{max} = 2\,\mathrm{m/s^2}), (\hat{v}_{max} = 4\,\mathrm{m/s}, \hat{a}_{max} = 1\,\mathrm{m/s^2})\}$ with start and end positions randomly sampled in a $(5\,\mathrm{m})^3$ spatial cubic. These samples are also part of the investigation conducted in Figure 2.8a. We show the empirical probability that the relative discrepancy between the time-optimal trajectory duration yielded by the TOP-UAV approach and the SOTA approach is higher than a certain value in Figure 2.10a. Moreover, we show the probability that the error in the final position, final velocity magnitude and velocity direction is higher than a certain absolute value (see Figures 2.10b - 2.10d).



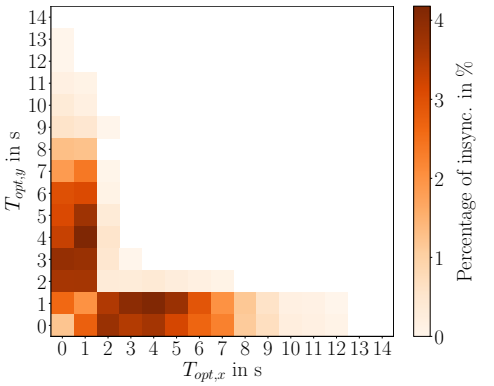
(a) Heatmap of the percentage of insynchronizable trajectories with $\hat{v}_{max} = 1\,\mathrm{m/s}$ and $\hat{a}_{max} = 3\,\mathrm{m/s^2}$. Total number of insynchronizable trajectories: 26 out of 100, 000.



(b) Heatmap of the percentage of insynchronizable trajectories with $\hat{v}_{max} = 5\,\mathrm{m/s}$ and $\hat{a}_{max} = 3\,\mathrm{m/s^2}$. Total number of insynchronizable trajectories: 5, 363 out of 100, 000.



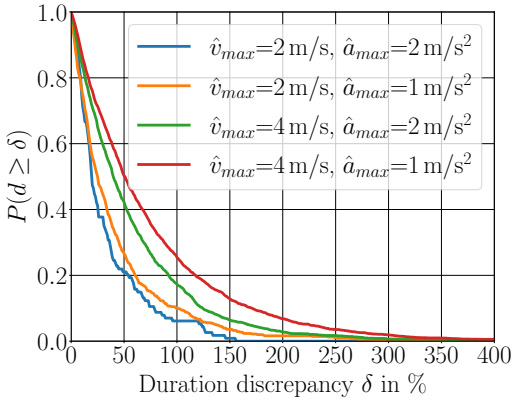
(c) Heatmap of the percentage of insynchronizable trajectories with $\hat{v}_{max} = 1\,\mathrm{m/s}$ and $\hat{a}_{max} = 1\,\mathrm{m/s^2}$. Total number of insynchronizable trajectories: 285 out of 100, 000.



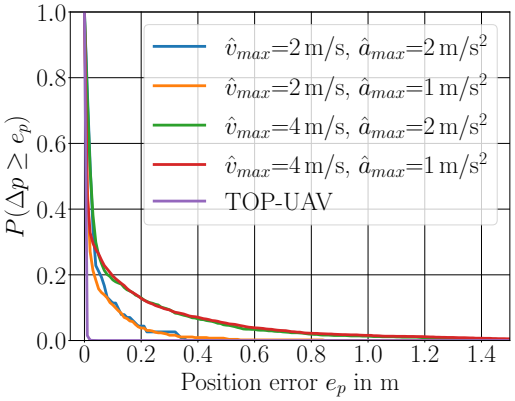
(d) Heatmap of the percentage of insynchronizable trajectories with $\hat{v}_{max} = 5\,\mathrm{m/s}$ and $\hat{a}_{max} = 1\,\mathrm{m/s^2}$. Total number of insynchronizable trajectories: 6, 725 out of 100, 000.

Figure 2.9.: Occurances of insynchronizabilities in 2D for start positions in $(5\,\mathrm{m})^2$. The time-optimal trajectory for each axis is calculated as stated in Section 2.3.

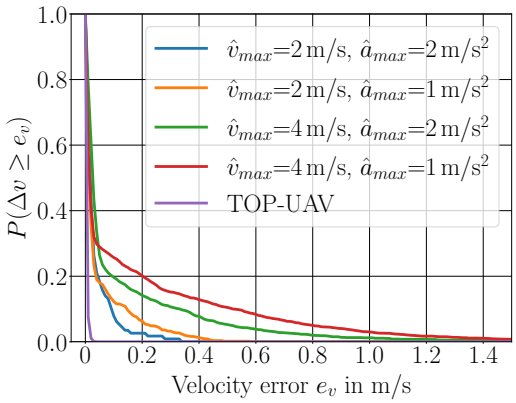
To evaluate these errors, we utilize our model predictive controller (MPC) which we provide in Appendix B.7 and which is similar to the MPC presented in [57]. Our MPC is based on the decoupling of the coordinate axes which means that each axis of the trajectory is treated separately. Instead of the jerk, we apply the acceleration as control input. As an objective, we only consider the Mayer term, i.e. the objective function term that penalizes deviation from the desired final state at the last time step, since we are only interested in minimizing the deviation to the desired final state at the given synchronization time T_{sync}. The mathematical constraint to meet the final state exactly is dropped since this would lead to infeasible solutions for the MPC in cases where the SOTA approach fails to calculate the correct time-optimal duration. To bound the applied acceleration and the velocity, we apply Inequations (2.1), (2.2) as constraints. As sample time, we apply



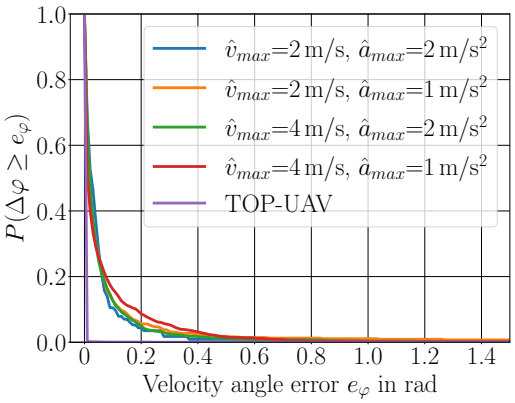
(a) Probability distribution of the discrepancy between the trajectory duration yielded by our TOP-UAV method and by the SOTA method.



(b) Probability distribution of the distance e_p between the desired $\boldsymbol{p_e}$ and position $\boldsymbol{p}(T_{\text{SOTA}})$ achieved by MPC.



(c) Probability distribution of the velocity difference e_v between the desired $\boldsymbol{v_e}$ and the velocity $\boldsymbol{v}(T_{\text{SOTA}})$ achieved by MPC.



(d) Probability distribution of the angle e_φ between the desired end velocity vector $\boldsymbol{v_e}$ and the velocity vector $\boldsymbol{v}(T_{\text{SOTA}})$ achieved by MPC.

Figure 2.10.: Error between desired end state and end state achieved by MPC with final time T_{SOTA} for trajectories where SOTA fails to find a feasible solution.

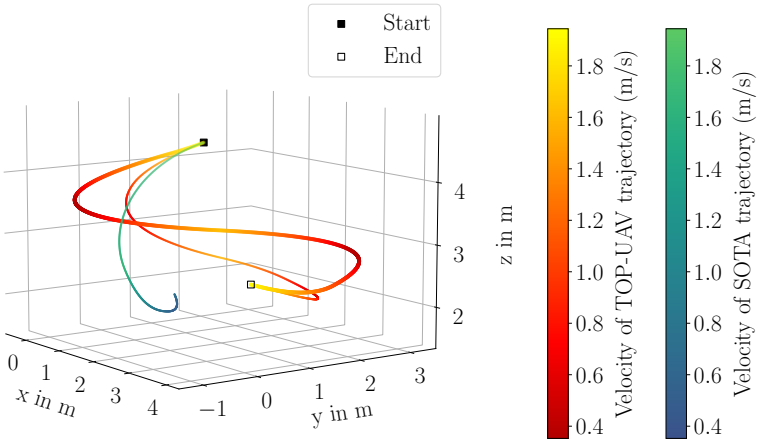


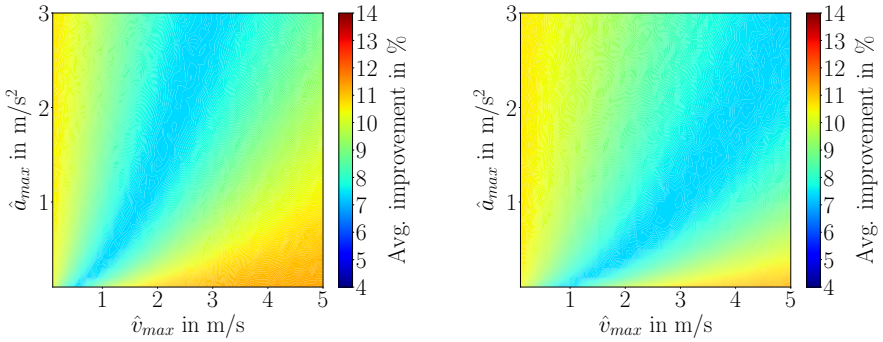
Figure 2.11.: Example setting for a time-optimal trajectory planning problem with $\hat{v}_{max} = 4\,\text{m/s}$ and $\hat{a}_{max} = 1\,\text{m/s}^2$. The SOTA solution (green profile) misses the desired final state by far. The solutions of our approaches (TOP-UAV: thick red profile, TOP-UAV++: thin red profile) work correctly.

0.05 s from which a finite horizon of $N = \lfloor T_{\text{SOTA}}/0.05\,\text{s} \rfloor$ steps is derived. The resulting three-dimensional trajectory corresponds to the composition of the trajectories along each axis.

First, we specify the discrepancy between the trajectory duration T_{SOTA} obtained from the SOTA approach and $T_{\text{TOP-UAV}}$ yielded by TOP-UAV as $\delta = (T_{\text{TOP-UAV}} - T_{\text{SOTA}})/T_{\text{SOTA}}$. To determine the position error, we evaluate $e_p = ||\boldsymbol{p_e} - \boldsymbol{p}(T_{\text{SOTA}})||$ with $\boldsymbol{p_e}$ representing the desired end position vector and $\boldsymbol{p}(T_{\text{SOTA}})$ the achieved end position at time T_{SOTA} which is determined using our MPC. For the velocity error it holds $e_v = ||\boldsymbol{v_e} - \boldsymbol{v}(T_{\text{SOTA}})||$ and for the velocity angle error $e_\varphi = \arccos(\boldsymbol{v_e}^\top \boldsymbol{v}(T_{\text{SOTA}})/(||\boldsymbol{v_e}|| \cdot ||\boldsymbol{v}(T_{\text{SOTA}})||))$ with $\boldsymbol{v_e}$ representing the desired end velocity vector and $\boldsymbol{v}(T_{\text{SOTA}})$ the yielded end velocity vector at time T_{SOTA}. Again, e_v and e_φ are determined by utilizing our MPC. In Figure 2.10a, it can be seen that there is a high probability that the discrepancy between T_{SOTA} and the optimal trajectory duration $T_{\text{TOP-UAV}}$ yielded by TOP-UAV is significant. For example, in 50% of all insynchronizable cases for $\hat{v}_{max} = 4\,\text{m/s}$ and $\hat{a}_{max} = 1\,\text{m/s}^2$ the discrepancy is higher than 51% and reaches up to 169% for 10% of these cases. Further, we see that in these cases the error between the achieved and desired final state for all kinematic samples is notable (see blue, orange, green and red graphs in Figures 2.10b - 2.10d). For comparison, the errors encountered across all kinematic samples when tracking the TOP-UAV solutions by the MPC are given in purple. It can be seen that only neglectable errors result due to the time-discrete nature of MPC. An extreme example of the discrepancy between TOP-UAV and the SOTA is given in Figure 2.11. Here the solution for the TOT-PMAV from the start position $\boldsymbol{p_s} = (0.1, 2.0, 4.3)^\top$ (m) with start velocity $\boldsymbol{v_s} = (0.1, -1.9, -0.4)^\top$ (m/s) to the end position $\boldsymbol{p_e} = (3.6, 0.4, 2.6)^\top$ (m) with end velocity $\boldsymbol{v_e} = (0.1, -1.8, 0.6)^\top$ (m/s) has to be determined for $\hat{v}_{max} = 4.0\,\text{m/s}$ and $\hat{a}_{max} = 1.0\,\text{m/s}^2$. In this scenario, the SOTA yields an optimal trajectory duration of $T_{\text{SOTA}} = 4.59\,\text{s}$ for which the above-described MPC only finds a solution with $e_p = 1.41\,\text{m}$, $e_v = 1.42\,\text{m/s}$ and $e_\varphi = 39.5°$ (see green profile), whereas the TOP-UAV approach yielding $T_{\text{TOP-UAV}} = 11.89\,\text{s}$ reaches the desired final state exactly (see thick red profile). The same holds for our TOP-UAV++ approach, which, however, yields $T_{\text{TOP-UAV++}} = 7.57\,\text{s}$ (see thin red profile). Further, Figures 2.10a - 2.10d show again that the probability of high discrepancies increases with increasing maximum velocity and decreases with increasing maximum acceleration.

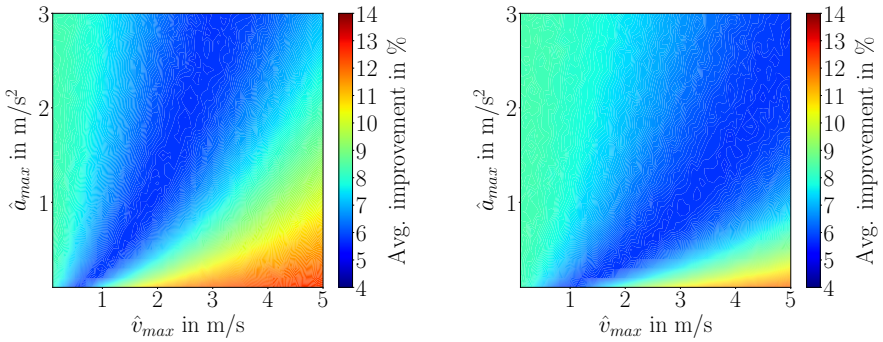
2.5.4. Improved Exploitation of Kinematic Properties

In Figures 2.12 and 2.13, we evaluate the average trajectory duration reduction $(T_{\text{TOP-UAV}} - T_{\text{TOP-UAV++}})/T_{\text{TOP-UAV}}$ of the TOP-UAV++ approach compared to TOP-UAV. Note that TOP-UAV++ always yields solutions that are at least as good as TOP-UAV in case the configuration to share the norms in TOP-UAV is also included in the set of possible configurations for TOP-UAV++. In analogy to Section 2.5.2, the evaluation is based on the generation of 100,000 trajectories for each combination of discretized maximum velocity and acceleration with start and end position randomly sampled in a $(5\,\text{m})^2$ and $(15\,\text{m})^2$ spatial square and a $(5\,\text{m})^3$ and $(15\,\text{m})^3$ spatial cubic. Depending on the maximum allowed velocities and accelerations, the average reduction ranges from 4% up to 14%. The highest



(a) Start/end position in $(5\,\text{m})^2$ **(b)** Start/end position in $(15\,\text{m})^2$

Figure 2.12.: Improvement of TOP-UAV++ compared to TOP-UAV over $\hat{v}_{max}$ and $\hat{a}_{max}$.

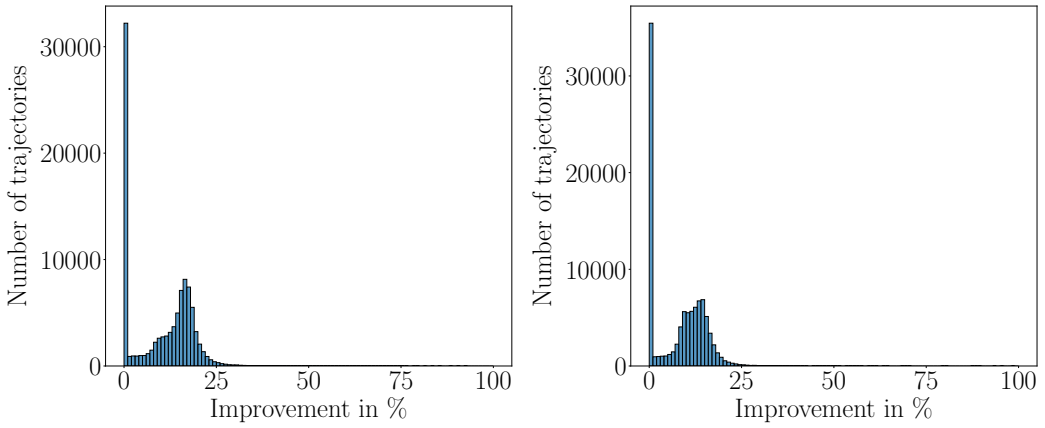


(a) Start/end position in $(5\,\text{m})^3$ **(b)** Start/end position in $(15\,\text{m})^3$

Figure 2.13.: Improvement of TOP-UAV++ compared to TOP-UAV depending on $\hat{v}_{max}$ and $\hat{a}_{max}$.

reductions occur for high velocities and low accelerations. Overall, the reduction is higher for high maximum velocities with simultaneously low accelerations but also for low velocities and high accelerations. We explain this by the fact that for distant waypoints the maximum velocity and for near waypoints the maximum acceleration is the limiting factor which can successfully be resolved by our TOP-UAV++ approach.

In Figures 2.14a and 2.14b, we present a histogram plot showing the improvement of TOP-UAV++ compared to TOP-UAV parametrized with $\hat{v}_{max} = 4\,\text{m/s}$ and $\hat{a}_{max} = 1\,\text{m/s}^2$ for the above described 100,000 trajectories sampled from $(5\,\text{m})^2$ and $(15\,\text{m})^2$. We group all generated trajectories into 100 bins of equal size representing an improvement of 1%. A single bin describes the count of how many times TOP-UAV++ yielded an improvement compared to TOP-UAV calculated as $(T_{\text{TOP-UAV}} - T_{\text{TOP-UAV++}})/T_{\text{TOP-UAV}}$ that is assigned to the bin. As can be seen in Figure 2.14a, for approximately one-third of all trajectories no improvement is achieved. For the remaining trajectories, TOP-UAV++ achieves an improvement of approximately 20% in the majority of all cases. Improvements higher than 25% are very unlikely in the two-dimensional case. In Figure 2.14b, one can see that increasing the sampling area for the start and end waypoints further reduces the improvements of TOP-UAV++ compared to TOP-UAV.



(a) Histogram of the improvement of TOP-UAV++ compared to TOP-UAV. Start/end position in $(5\,\mathrm{m})^2$.

(b) Histogram of the improvement of TOP-UAV++ compared to TOP-UAV. Start/end position in $(15\,\mathrm{m})^2$.

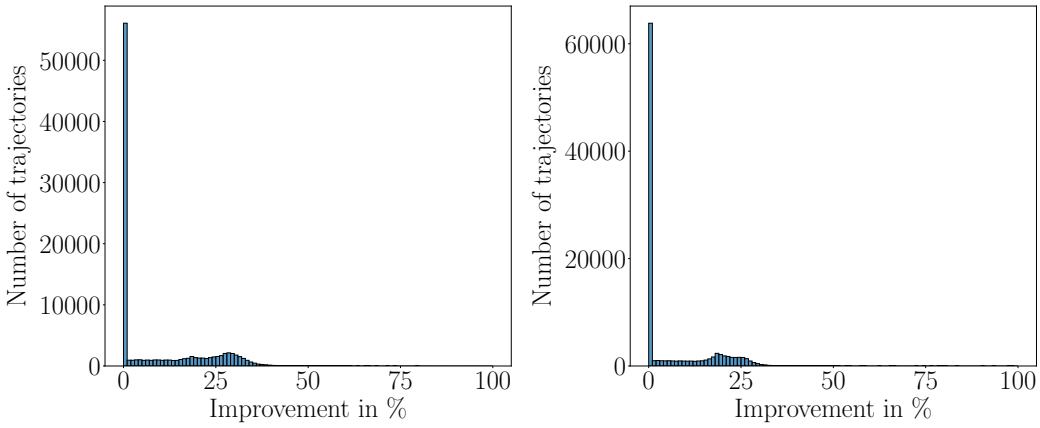
Figure 2.14.: Histogram of the improvement of TOP-UAV++ compared to TOP-UAV derived from 100,000 randomly generated trajectories with $\hat{v}_{max} = 4\,\mathrm{m/s}$ and $\hat{a}_{max} = 1\,\mathrm{m/s^2}$ in two dimensions.

Analogously, we investigate the distribution of the improvements in the three-dimensional case by randomly sampling start and end positions for another 100,000 from a $(5\,\mathrm{m})^3$ (see Figure 2.15a) and a $(15\,\mathrm{m})^3$ (see Figure 2.15b) spatial cubic. Here, it can be seen that TOP-UAV++ does not yield a better solution than TOP-UAV in approximately 55% of all cases when the waypoints are sampled from $(5\,\mathrm{m})^3$. In the remaining cases, the improvement is approximately equally distributed between 1% and 35%. An improvement of more than 35% is unlikely. In approximately 63% of the cases when the waypoints are sampled from $(15\,\mathrm{m})^3$, TOP-UAV++ does not yield faster trajectories that TOP-UAV. Here, the improvements for the remaining cases vary from 1% to approximately 30, while improvements of more than 30% are unlikely.

2.5.5. Computation Times

In the following, we present the results of our study on the computation time of our TOP-UAV and TOP-UAV++ trajectory planner. Our study distinguishes between the two- and three-dimensional approaches, each evaluated on 100,000 randomly generated trajectories sampled from $(5\,\mathrm{m})^2$ and $(5\,\mathrm{m})^3$ respectively.

Two-dimensional approach:
The computation times of TOP-UAV and TOP-UAV++ as well as the SOTA method are presented in Tab. 2.2. It can be seen that the SOTA has the fastest average computation time of 52 ns due to the missing feasibility check, which approximately doubles the required computation time and results in 111 ns in scenarios where the SOTA yields a valid solution (see column 'TOP-UAV$_{\mathrm{sync}}$'). The SOTA and our TOP-UAV approach come with a standard deviation of 9 ns and 22 ns, which we assume to be the impact of the operating system. In cases where the SOTA approach is not applicable, our TOP-UAV approach takes on



(a) Histogram of the improvement of TOP-UAV++ compared to TOP-UAV. Start/end position in $(5\,\mathrm{m})^3$.

(b) Histogram of the improvement of TOP-UAV++ compared to TOP-UAV. Start/end position in $(15\,\mathrm{m})^3$.

Figure 2.15.: Histogram of the improvement of TOP-UAV++ compared to TOP-UAV derived from 100,000 randomly generated trajectories with $\hat{v}_{max} = 4\,\mathrm{m/s}$ and $\hat{a}_{max} = 1\,\mathrm{m/s^2}$ in three dimensions.

	SOTA	TOP-UAV$_{\mathrm{sync}}$	TOP-UAV$_{\neg\mathrm{sync}}$	TOP-UAV	TOP-UAV++
Average computation time (ns)	52	111	572	139	288
Standard deviation (ns)	8	22	146	118	261

Table 2.2.: Computation time evaluation of 100,000 randomly generated trajectories in $(5\,\mathrm{m})^2$ with $\hat{v}_{max} = 4\,\mathrm{m/s}$ and $\hat{a}_{max} = 1\,\mathrm{m/s^2}$.

average 572 ns (see column 'TOP-UAV$_{\neg\mathrm{sync}}$'). We identify these cases by validating that the trajectory duration yielded by the SOTA and by TOP-UAV are different. In this case, the computation time depends on how fast a feasible solution candidate for TOP-UAV is found. Hence, the standard deviation increases to 146 ns. In this two-dimensional study, the overall average computation time of our TOP-UAV planner is 139 ns with a standard deviation of 118 ns. The overall computation time of our TOP-UAV++ trajectory planner is approximately 288 ns with a high standard deviation of 261 ns due to possible insynchronizabilities and multiple configurations. Note that although the improved version TOP-UAV++ utilizes in total three different options to share the norms among the axes its computation time is 2.07 times the computation time of the TOP-UAV planner. The reason for this is as follows: If for a specific configuration the SOTA approach yields a higher trajectory duration than the lowest feasible trajectory duration found so far, then the entire configuration can be skipped since it would not yield a better solution than already identified.

To illustrate the computational efficiency of both of our approaches: The measured computation time to calculate one million trajectories in closed form with TOP-UAV is 139 ms and with TOP-UAV++ is 288 ms in the given study.

Three-dimensional approach:
The computation time evaluation in the three-dimensional case for TOP-UAV and TOP-UAV++ as well as the SOTA method is presented in Table 2.3 and the experimental setup

	SOTA	TOP-UAV$_{\text{sync}}$	TOP-UAV$_{\neg\text{sync}}$	TOP-UAV	TOP-UAV++
Average computation time (ns)	72	160	1856	242	511
Standard deviation (ns)	11	26	389	374	749

Table 2.3.: Computation time evaluation of 100,000 randomly generated trajectories in $(5\,\text{m})^3$ with $\hat{v}_{max} = 4\,\text{m/s}$ and $\hat{a}_{max} = 1\,\text{m/s}^2$.

is analogous to the one conducted in the two-dimensional case. Again, it can be seen that the SOTA has the fastest computation time with on average 72 ns followed by the TOP-UAV approach in scenarios for which the SOTA would be applicable requiring 160 ns. Both cases come with a standard deviation of 11 ns and 26 ns, which we assume to be the impact of the operating system. In cases where the SOTA approach is not applicable, our basic approach takes on average 1.86 μs (see column 'TOP-UAV$_{\neg\text{sync}}$'). In this case, the computation time depends on how fast the fastest feasible solution candidate is found. Hence, the standard deviation increases to 389 ns. Our TOP-UAV trajectory planner requires an average computation time of 242 ns in the given study with a standard deviation of 374 ns. However, remember that the average computation time depends on the probability of how often the SOTA approach is invalid and the candidate list must be set up. The overall computation time of our TOP-UAV++ trajectory planner is approximately 511 ns with a high standard deviation of 749 ns due to possible insynchronizabilities and multiple configurations. In conclusion, the TOP-UAV++ planner requires on average 2.11 times the computation time of the TOP-UAV planner for the given study.

2.6. Conclusion

In this chapter, we presented our TOP-UAV solver for time-optimal trajectory generation for point-masses under acceleration and velocity constraints. Our trajectory planner is proven to overcome a flaw in the state-of-the-art solution approach and hence yields the globally optimum solution of the trajectory planning problem at hand for the given kinematic restriction considering the associated configurations of distributing the maximum velocity and acceleration norms among the axes. Since our approach is analytical it allows for a highly performant implementation that is capable of calculating time-optimal a trajectory in the scale of a few hundred nanoseconds as we show in Section 2.5.5. Further, we investigate the consequences of the flaw of the state-of-the-art approach and evaluate our new approach in these situations. To further improve the exploitation of the given kinematic restrictions, we allow our framework to work with multiple configurations to distribute the maximum allowed velocity and acceleration norms among the axes. This extended approach TOP-UAV++ yields on average up to 14% faster trajectories while requiring approximately two times the computation time. In the two-dimensional case, this corresponds to an additional computation time of 149 ns on average. For the three-dimensional approach, the additional computation time is 269 ns on average.

In the next chapter, we utilize the two-dimensional version of our TOP-UAV++ trajectory planner to solve inertia-based routing problems for multirotor UAVs.

3. Inertia-based Routing

As stated in Chapter 1, UAV technology is highly beneficial for a manifold of applications, especially in surveillance and data collection applications. Most of these applications can be modeled as a set of tasks that must be performed efficiently and effectively considering the UAV's motion constraints. In the majority of cases, the term 'efficiently' equals the objective of fulfilling tasks with minimum execution time [56].

The resulting problem represents a simultaneous task sequencing and motion planning problem which is often closely related to variations of the traveling salesman problem (TSP) and the orienteering problem (OP). In simple terms, the TSP describes a combinatorial optimization problem in which a cost-minimum Hamiltonian cycle is sought that connects all nodes in a given set of nodes while each node is exactly once. The OP, in turn, describes a combinatorial optimization problem in which a tour from a given start node to a given end node is sought that connects a subset of nodes in a given set of prioritized nodes such that the tour maximizes the collected priorities. Further, each node can be visited at most once and the total cost of the tour may not exceed a maximum value. For more detailed information on the TSP and the OP, we refer the reader to e.g. [70] and [71].

In the context of UAV routing, the nodes represent spatial waypoints that must be visited and edges represent the physical motion for traveling between two waypoints. In the literature, there are many variations on how these edges and their associated costs can be determined. For each alternative, an entire field of different route planning problems can be derived that is suited for specific use cases. In Section 3.1, we give an overview of the associated route planning problem and present the state-of-the-art (SOTA) approaches to solving them.

As we will see in the conclusion of Section 3.1, the SOTA approaches for UAV routing problems do not consider the full kinematic capabilities of multirotor UAVs in terms of the maximum allowed velocity and acceleration in any direction and their ability to hover. Our trajectory planner developed in Chapter 2 meets these requirements and further even guarantees time-optimality. Hence, we develop new extensions of the TSP and the OP that can be combinded with our TOP-UAV++ trajectory planner for calculating the physical motions between each pair of waypoints. These are further denoted as the kinematic traveling salesman problem (KTSP) and kinematic orienteering problem (KOP). Our new models explicitly support varying velocity magnitudes and hence allow flying with low velocity when agility is required and high velocity when larger distances have to be covered. In Section 3.4, we show that our models can successfully be solved by a commercial general-purpose solver to optimality for small and medium-sized problem instances with up to 30 waypoints within a computation time limit of five hours. Further,

we show that the optimal solutions of our models significantly outperform the optimal solutions of SOTA models.

The KTSP and the KOP are NP-hard since they are extensions of the TSP and OP which are also known to be NP-hard (see e.g. [71]). Consequently, for an increasing number of waypoints, the computational complexity increases exponentially. This explains the observation from our computation study that solving our KTSP and KOP models to optimality with commercial general-purpose solvers requires a few seconds with problem instances with up to ten waypoints, however, easily exceeds one hour of computation time for problem instances with more than ten waypoints. Therefore, we develop powerful heuristic solvers for both models which are based on the adaptive large neighborhood search (ALNS) framework.

3.1. Related Work

In general, route planning problems for UAVs are a well-studied scientific subject (see [5, 13, 72]). Thereby, most studies focus on surveillance and data collection (see e.g. [13]) with, in the majority of all studies, a minimum mission duration as an objective. However, although UAV routing is so well-studied, only very few papers focus on problems that explicitly consider the UAV's kinematics. In the majority of cases, the kinematics are significantly simplified or even neglected [72, 4]. In the following, we focus on the little existing work related to UAV routing under consideration of their kinematic properties.

3.1.1. Overview of Inertia-based Route Planning Problems

In this section, we present an overview of related approaches for route planning problems considering the kinematic restrictions of UAVs, i.e. maximum velocity and acceleration.

Inertia-based Route Planning Neglecting Acceleration:
A simple approach to consider basic kinematic restrictions in UAV routing problems is the assumption of a constant velocity for the motion but neglecting the acceleration capabilities. This approach is used to estimate the flight time between two spatial coordinates as the quotient of the Euclidean distance between the coordinates over the constant flight velocity [72, 73, 74]. Some approaches define that the constant velocity equals the maximum allowed velocity [21]. Note that although the assumption of a constant velocity is easy to be used to estimate the travel time based on the distance between two waypoints, this assumption is rarely explicitly defined in the related work. In many publications, no comment is given on how the travel times are calculated [75]. Some assume, that "the travel time between pairs of targets were set equal to the Euclidean distance between them" (see [76]). Such an assumption might be sufficient to evaluate the performance of solution approaches. However, it does not allow any conclusions to be drawn about whether the proposed solution approaches themselves can be applied to solve real-world problems properly. A simple countermeasure to consider acceleration capabilities on a very basic level are

hover-to-hover trajectories, as we provide them in Appendix A. Hover-2-hover trajectories rely on the assumption that each waypoint must be visited at rest while the travel between waypoints allows full acceleration until a possibly defined maximum velocity is reached. However, even this very simple measure is very hard to find in the literature.

Inertia-based Route Planning using Dubins Paths:
When it comes to the consideration of constrained velocity and acceleration, most approaches rely on Dubins paths. Examples include extensions of the TSP (see [77, 78, 79, 80, 81, 82, 83]) and the OP (see [22, 23, 84]). These extensions are known as the Dubins traveling salesman problem (DTSP) and the Dubins orienteering problem (DOP), whereas the DTSP has been covered much more deeply in the literature. For example, lower bounds for the solution quality have been developed (see [80, 77]). To the best of our knowledge, such investigations have not been conducted for the DOP so far.

From a top-level view, most publications on DTSP and DOP are based on a discretization of the heading angles that can be used to traverse each waypoint. This allows for powerful mathematical programming formulations that can be used to obtain globally optimal solutions by applying exact approaches such as general-purpose solvers (see [77]) or branch-and-price approaches (see [84]). On the other hand, it also enables the development of powerful heuristic solvers (see [22, 78, 85]). Since we use the DTSP and the DOP as benchmark models in this work, we provide the associated mathematical programming models in Sections 3.2.1.1 and 3.2.2.1.

In recent research, extensions of the Dubins paths are investigated in UAV routing problems. These approaches aim to better describe the physical capabilities of UAVs, to consider obstacles, and to overcome the drawback of a fixed velocity. In [78], it is investigated that UAVs cannot immediately apply full acceleration lateral to the direction of motion, as assumed for the Dubins vehicle. Hence, they consider a smooth change of acceleration. Further, they consider the presence of obstacles by considering only collision-free connecting Dubins paths between waypoints. Additionally, they state that for an existing sequence of visiting waypoints, the discrete heading angles to traverse each waypoint can be globally optimized by using dynamic programming. The presented approach, however, still suffers from the general drawback of the Dubins paths: The minimum turning radius. This radius r is constant for the entire flight mission and depends on the constant flight velocity v_{const} and the maximum acceleration $\hat{a}_{max}$ and is calculated as $r = v_{const}^2 / \hat{a}_{max}$. This limits the performance of the yielded solutions as follows: Distant waypoints can be connected in less flight time when moving at a high velocity but, in turn, connecting close waypoints requires agility which is obtained at a low velocity. In [86] this problem is addressed by designing Dubins-like trajectories that are bound to a constant velocity when making turns but are allowed to change velocity on straight segments. However, since the curvature of the Dubins paths is still bound to a fixed velocity also this approach suffers a potential performance limitation when considering multirotor UAVs.

The first to describe the Dubins orienteering problem are the authors of [22]. In their work, they investigate the application of a UAV required to visit a set of prioritized waypoints, but given a restricted maximum flight distance. Costs to traverse between each pair of waypoints are given as a distance. As a solution approach, they propose a

variable neighborhood search (VNS) and solve the problem heuristically. To evaluate the performance of their approach, they benchmark their approach against the orienteering problem (OP) with travel costs calculated as the Euclidean distance. However, they are not able to determine exact solutions for their DOP instances since their mathematical model cannot be solved by a general-purpose solver. In the same year, the same authors introduced the Dubins orienteering problem with Neighborhoods (DOPN) that deals with the problem that in UAV applications not all waypoints have to be traversed exactly but with tolerances [23]. Again, a VNS-based solution approach is presented and the results of DOPN are benchmarked against their approach for the DOP [22]. Exact solutions for the DOPN are not provided as well.

The authors of [84] introduce a branch-and-price approach to the Dubins team orienteering problem (DTOP). The DTOP is closely related to the DOP but considers multiple Dubins vehicles. Although representing a generalization of the DOP no benchmark against the solutions presented in [22] was conducted.

Lastly, we identified another unresolved problem regarding the DOP. To the best of our knowledge, there exists no mathematical programming formulation for the DOP that can directly be used by a commercial general-purpose solver. To allow benchmarking our KOP with the DOP, we develop such a model in [87] and provide it in Appendix 3.2.2.1.

Inertia-based Route Planning using Bézier Curves and B-Splines:
Apart from Dubins paths, also Bézier curves [36, 37] and its generalization B-splines [3] are utilized to cover the kinematic properties of UAVs in routing problems. However, both concepts are not represented as much as e.g. Dubins-based routing in the literature. In both approaches, two locations are connected by a smooth curve based on Bernstein polynomials. By setting intermediate control points in the right way, it is possible to guide the path safely around obstacles. However, their polynomial representation of a safe path is purely spatial. To ensure physical feasibility, the resulting curve must first be transferred to the time domain. Only by assigning each spatial point of the curve to a particular point in time is it possible to consider physical constraints such as maximum velocity and acceleration (see [38]). For routing problems where thousands of trajectories are calculated this two-step procedure is computationally expensive (see [36]). This might be the reason, why the authors of [36] proposed to use a numerical estimation of the travel duration for a given Bézier curve. Further, it is argued in [38] that if the initial and final velocity are not both zero, there may not exist a feasible solution. This can be explained by the following illustration. Given a spatial Bézier curve with changing curvature as well as a start velocity magnitude and direction. By decreasing the maximum allowed acceleration towards any direction, it is always possible to create a setting where the maximum acceleration is not high enough to keep on track with the curvature of the spatial Bézier curve. Consequently, for such a setting, there is no possibility of feasibly transferring the spatial path into the time domain respecting the given maximum acceleration property. Moreover, another disadvantage results from their polynomial representation, which does not allow for bang-bang behavior and hence does not yield time-optimal behavior (see [28]). Remember, bang-bang behavior is motivated by Pontryagin's minimum principle (see [63]), which states that time optimality is only achieved by having the system always operate at its

physical limit. These above described drawbacks might be the reason why Bézier and B-splines curves are not that often used for UAV route planning in the literature.

MPC-based Route Planning Approaches:
The MPC-based approach from [88] which solves the 'incremental motion planning with dynamical reward' can also be used to solve the OP with kinematic restrictions. In their approach, the maximum allowed flight time is used as a fixed mission duration from which the number of control input periods is derived. In these periods, the UAV is allowed to apply a constant acceleration in any direction which dynamically affects velocity and position. Related to the resulting spatial position trajectory, rewards are received when passing by a waypoint sufficiently close. The allowed maximum velocity and acceleration are considered by applying the corresponding constraints. However, as the authors state, the number of planning steps is fixed and so is the total travel time. Hence, the provided solutions are not time-optimal but satisfy the travel time budget.

In [89] and [90], two articles that build on each other, the UAV flight planning is modeled as a discrete-time mixed-integer nonlinear mathematical program. Their model considers multiple UAVs moving in the three-dimensional cartesian space and a set of environmental constraints such as obstacles, wind fields, fuel consumption, and flight range. Further, the authors consider each UAV's kinematic capabilities such as maximum velocity and acceleration in three spatial dimensions individually. To obtain a mixed-integer linear program that serves as a basis for their computational study, the nonlinear constraints are linearized. In the computational study, the authors apply a commercial general-purpose solver to solve a set of problem instances. Due to the discrete-time formulation, their solution approach primarily depends on the maximum duration of the flight mission and on the time step length for which decisions, such as applied acceleration, are defined as fixed. Larger time step lengths result in less computational complexity but also in less maneuverability which is accompanied by less kinematic efficiency. Overall, a trade-off between computational complexity and kinematic efficiency is to be found.

Mixed-Integer Optimal Control Problem (MIOCP)-based Approaches:
The authors of [91, 92] present a general approach that can be applied to numerically solve a mixed-integer optimal control problem and show that it can successfully be applied to the motorized traveling salesman problem which is closely related to our KTSP formulation but is based on a car like vehicle with a different kinematic model than multirotor UAVs. For this problem class, the authors model the problem of sequencing the cities of the TSP by introducing a discrete binary control variable. The state variables, i.e. position, velocity, and steering angle as well as the control variables, i.e. acceleration and steering angle velocity are modeled as continuous functionals. In their work, a decomposition approach is proposed that addresses the decision on the sequence of the cities represented by the discrete binary control variables via branch-and-bound (see [91]) or using a genetic algorithm (see [92]). For all intermediate waypoint sequence representations, i.e. for nodes in the branch-and-bound-tree or individuals in the population of the genetic algorithm, the underlying multi-phase optimal control problem is solved using direct collocation. For this purpose, the continuous state and control variables are modeled as piecewise cubic Hermite polynomials from which the optimal control problem reduces to a constrained nonlinear

program with the optimization variables representing the coefficients of the Hermite polynomials as well as the travel times between each pair subsequent waypoints.

The resulting constrained nonlinear program is solved via sequential quadratic programming yielding a locally optimum solution which, in general, does not neccessarily represent a globally optimum solution. Further, although the authors use the objective of minimizing the total trajectory duration, the solutions yielded by the proposed approaches will in general be time-suboptimal since they model the control variables as polynomials. Remember, polynomials do not allow for bang-bang behavior, which is required for time optimality (see [28]).

Conclusion:
According to the related literature, only a few approaches exist for UAV routing that consider the maximum allowed velocity and acceleration for multirotor UAVs properly. The first class of approaches utilizes the classical routing model formulations with only one possible edge to travel between two waypoints. This limits the range of methods, that can be applied to properly estimate the travel time between waypoints. The second class of approaches is based on Dubins paths. The associated models allow to traverse waypoints with different heading angles. Depending on heading angles for the start and end waypoint, the travel time between waypoints changes. This allows to model the inertia of a UAV. However, Dubins paths are bound to a constant velocity, which does not account for the full kinematic capabilities of multirotor UAVs. The third class is based on Bézier curves and B-splines. Both motion planning methods compute a smooth spatial path that appears to be trackable by an inert UAV. However, this path does not implicitly consider maximum velocity and acceleration and hence they do not model the kinematics of UAVs accurately. Moreover, there are MPC-based approaches. MPC is a very powerful approach for considering the physical behavior of UAVs. However, the MPC-based approaches get computationally expensive for an increasing number of time steps. Hence, for long-duration flight missions, the time step length must be increased to reduce the overall number of time steps and consequently, to keep the computational complexity manageable. However, the reduction of the time step length comes at the cost of reducing the maneuverability of the UAV in the planning process. The last class of approaches discussed here is based on iteratively solving the optimal control problem for a given sequence using direct collocations by solving a constrained nonlinear program. However, these approaches are again based on a polynomial representation of the control input trajectory, which does not allow for bang-bang behavior and again would not utilize the full kinematic capabilities of a multirotor UAV.

3.1.2. Research Gap and Contributions

There are only a few approaches for UAV routing that consider the maximum allowed velocity and acceleration, as we present in Section 3.1.1. These few approaches either apply simplifying assumptions with regard to the kinematics of multirotor UAVs, such as a constant velocity (see [22, 77, 78]), or rely on a coarse time-discretization to keep the resulting optimization problem computationally tracktable (see e.g. [4]). Both do not

properly reflect the full kinematic properties of a multirotor UAV. Some approaches use polynomial trajectory representations that are assumed, although not mathematically of experimentally demonstrated, as good enough to exploit the kinematic capabilities (see e.g. [21, 36, 91]). To the best of our knowledge, none of the existing approaches in the literature fully covers the entire maneuverability of a multirotor UAV in terms of time-optimal motions based on their kinematic capabilities. Note that this observation holds despite time optimality although time-optimality is known as the most-used objective in UAV routing [13]. With this, we identify a major research gap in the intersection between route planning and UAV motion planning and make the first steps toward this new field of research.

Research Questions:
The above research gap can be expressed by a set of research questions, which we present as follows:

- How to develop mathematical models such that the full potential of the kinematic capabilities of a multirotor UAV in terms of time-optimality is considered?
- What is the effect of such models in terms of the achieved solutions compared to state-of-the-art models such as DTSP and DOP?
- How computationally expensive is it to solve these models to optimality using commercial solvers?
- How to design heuristic solution approaches based on these models and how well is their performance regarding the achieved solution quality compared to the global optimum?

Contributions:
To answer these research questions, we focus on the two basic classes of routing problems, namely the TSP and the OP, and make the following major contributions:

- We present new models that extend the traveling salesman problem and the orienteering problem, namely the kinematic traveling salesman problem (KTSP) and the kinematic orienteering problem (KOP), that are able to integrate our time-optimal trajectory planner presented in Section 2.4 to determine travel times between waypoints by exploiting the full kinematic properties of multirotor UAVs.
- Since solving these models, i.e. the KTSP and the KOP, with a commercial solver is computationally expensive, as we show in Section 3.4, we develop heuristic solver for both problem classes which are based on the adaptive large neighborhood search (see e.g. [24]). This metaheuristic solution framework is derived from the well-known ruin and recreate mechanism (see [93]) and is widely and successfully used to solve routing problems in various domains (see [94, 95]). To unlock the greatest potential of our heuristic solvers, we tune their hyperparameters via hyperparameter optimization.

- We conduct an extensive computational study to evaluate the benefit of our new models compared to state-of-the-art models. Further, we investigate if our models suit to be solved by commercial solvers to solve realistic problem instances and we investigate the performance of our heuristic solvers regarding the achieved solution quality for predefined computation times.

What we do not focus on in this work:
Since we are the first to investigate such kind of kinematic routing problems, the following research directions are out of the scope of this work.

- We do not focus on developing and comparing multiple different mathematical models of the KTSP and the KOP in terms of the number of variables, constraints and overall computational complexity.
- We do not develop problem-specific exact solvers such as branch-and-price or branch-and-cut that aim at solving the KTSP and KOP to optimality with less computational effort than using a general-purpose solver.
- Developing and comparing multiple different heuristic solution approaches for the KTSP and KOP is also out of the scope of this work. This also holds for the development and performance evaluation of single insertion and removal operators of our adaptive large neighborhood solvers. In this work, we use and modify existing operators that can be found in the literature.

The remainder of this chapter is structured as follows. We introduce the mathematical models for inertia-based routing used in this work in Section 3.2. These contain the KTSP and the KOP model but also the associated Dubins paths-based models. Next, we present our heuristic approach to solving the KTSP and KOP in Section 3.3. Last, we present our computational study in Section 3.4.

3.2. Inertia-based Routing Models

In this section, we formally introduce inertia-based routing models to visit a set of given waypoints while respecting the kinematics of a multirotor UAV.

We first focus on models for inertia-based traveling salesman problems. These problems are especially relevant in real-world applications for UAVs in the context of continuous surveillance of infrastructures where the identical mission is executed multiple times in a series. In this regard, we present the Dubins traveling salesman problem (DTSP) as introduced in [77] which is an extension of the well-known traveling salesman problem (see [96]). Since Dubins paths are always bound to a fixed velocity, which is a potential performance limitation for multirotor UAVs (see Section 3.1.1), we present the kinematic traveling salesman problem (KTSP) as introduced in our work [25]. The KTSP enables the consideration of varying traversal velocities at each waypoint and hence unfolds the most potential in combination with our TOP-UAV++ trajectory planner presented in Section

2.4.4. In general, the KTSP can be defined for any inert system, as long as a suitable planning method is applied.

In cases where the maximum flight time must be considered, a selection of waypoints that are to be visited must often be made. This can be done by assigning each waypoint a priority value and then modeling the problem as an orienteering problem (see [97]). In the context of inertia-based orienteering problems, the Dubins orienteering problem has been defined in the literature and has been heuristically solved (see [22]). However, to the best of our knowledge, no mathematical programming formulation has been stated in the literature so far that can be solved directly using a general-purpose solver. For this reason, optimal benchmark solutions of the DOP are hard to find. In Section 3.2.2.1, we present our model which we developed in [87] and which can be solved using general-purpose solvers. It is to be mentioned that [87] is a master thesis supervised during this work whose results have not yet been published in a scientific paper. For the same reason as for the DTSP, i.e. since Dubins paths are always bound to a fixed velocity, we also present the kinematic orienteering problem as stated in our work [26]. Analogously to the KTSP, the KOP unfolds the most potential in combination with our TOP-UAV++ trajectory planner. In general, the KOP can be defined for any inert system, as long as a suitable planning method is applied.

Note that we solve the DTSP, KTSP, DOP, and KOP using the corresponding mathematical models and applying Gurobi implemented in Python as a general-purpose solver. More details on the computational setup are given in Section 3.4.1. This solution approach will henceforth be referred to as the mixed-integer programming (MIP)-based approach. To apply our MIP-based approach, the corresponding travel time matrix must be preprocessed. In this work, all travel time matrices used are symmetric.

3.2.1. Inertia-based Traveling Salesman Problem Models

In this Section, we introduce the DTSP as presented in [77] in Section 3.2.1.1, followed by our more general KTSP in Section 3.2.1.2. The DTSP and KTSP are NP-hard, since both problems are extensions of the TSP which is known to be NP-hard (see e.g. [96]).

The objective of the DTSP and the KTSP is to find a kinematically feasible trajectory through an initially unordered set of waypoints in the two-dimensional plane, whereas each waypoint is visited exactly once. In this regard, kinematically feasible means that a solution trajectory does not violate the given maximum velocity constraint and the given maximum acceleration constraint at any time.

The DTSP and KTSP are closely related and share a set of properties which are introduced in the following:

First of all, both problems are defined in an obstacle-free environment and for both a set of waypoints

$$\mathcal{L} = \{l_i \mid l_i = i, i = 1, ..., L\}$$

is given where each element $l_i \in \mathcal{L}$ must be visited. Here, L describes the number of waypoints.

Following the mathematical model of the DTSP presented in [77], we discretize the allowed heading angle to traverse each waypoint. We want to emphasize, that this discretization is an abstraction from the continuous reality to be able to mathematically model the problem. We define the set of discretized heading angles to traverse each waypoint $l_i \in \mathcal{L}$ as

$$\mathcal{H} = \{h_i \mid h_i = 2\pi(i-1)/H, i = 1, ..., H\}.$$

Here, H represents the number of equidistant discretization levels of the traversal heading angle.

With the definitions of $\mathcal{L}$ and $\mathcal{H}$, we present the DTSP and KTSP models in the following.

3.2.1.1. Dubins Traveling Salesman Problem according to [77]

In this section, we introduce the DTSP as presented in [77]. Based on the definitions of $\mathcal{L}$ and $\mathcal{H}$ in Section 3.2.1, the DTSP is formally described on a graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, where

$$\mathcal{N} = \{(l_i, h_k) \mid \forall i \in \{1, ..., L\}, \forall k \in \{1, ..., H\}\}$$

represents the set of unique nodes (l_i, h_k) such that there are $L \cdot H$ nodes in the graph. The set $\mathcal{E}$ of unique edges linking two nodes in graph $\mathcal{G}$ is represented as

$$\mathcal{E} = \{\big((l_i, h_k), (l_j, h_m)\big) \mid \forall i, j \in \{1, ..., L\}, \forall k, m \in \{1, ..., H\}\}.$$

Overall, there are $L^2 \cdot H^2$ edges in graph $\mathcal{G}$. The costs c_{ik}^{jm} associated with each edge $\big((l_i, h_k), (l_j, h_m)\big) \in \mathcal{E}$ describe the required flight time to travel the edge which is calculated as the length of the shortest Dubins path (see [31]) between both nodes divided by a given constant velocity v_{const}. In the context of this work, the optimum Dubins path depends on the minimum allowed turning radius of the multirotor UAV which is calculated as $v_{const}^2/\hat{a}_{max}$ while v_{const} can be arbitrarily chosen within $0 < v_{const} \leq \hat{v}_{max}$. Since increasing the constant mission velocity v_{const} for the DTSP leads to an increasing minimum turning radius, the selection of a proper v_{const} depends on the problem instance to be solved. This is a general problem of using Dubins paths in UAV routing ([25]).

The main decision variables for the DTSP model x_{ik}^{jm} are binary and interpreted as

$$x_{ik}^{jm} = \begin{cases} 1, & \text{if waypoint } l_i \text{ is left with heading angle } h_k \\ & \text{towards waypoint } l_j, \text{ which is} \\ & \text{entered with heading angle } h_m\,, \\ 0, & \text{otherwise} \end{cases}$$

Furthermore, there are integer decision variables $u_i \in \{1, ..., L\}, i = 1, ..., L$ that define the sequence of waypoints l_i in the resulting sequence of waypoints.

The resulting mathematical programming formulation to determine the optimal sequence of waypoints, as well as the heading angle configurations with which to visit each waypoint, is shown in the optimization problem (3.1).

$$\min \sum_{i=1}^{L} \sum_{j=1}^{L} \sum_{k=1}^{H} \sum_{m=1}^{H} x_{ik}^{jm} c_{ik}^{jm} \tag{3.1a}$$

s. t.

$$\sum_{j=1}^{L} \sum_{k=1}^{H} \sum_{m=1}^{H} x_{ik}^{jm} = 1, \quad \forall i \in \{1, ..., L\} \tag{3.1b}$$

$$\sum_{i=1}^{L} \sum_{k=1}^{H} \sum_{m=1}^{H} x_{ik}^{jm} = 1, \quad \forall j \in \{1, ..., L\} \tag{3.1c}$$

$$\sum_{i=1}^{L} \sum_{k=1}^{H} x_{ik}^{jm} - \sum_{o=1}^{L} \sum_{p=1}^{H} x_{jm}^{op} = 0 \qquad \forall j \in \{1, ..., L\}; \forall m \in \{1, ..., H\} \tag{3.1d}$$

$$u_i - u_j + (L-1) \sum_{k=1}^{H} \sum_{m=1}^{H} x_{ik}^{jm} + (L-3) \sum_{k=1}^{H} \sum_{m=1}^{H} x_{jm}^{ik} \leq L - 2 \qquad \forall i, j \in \{1, ..., L\} \tag{3.1e}$$

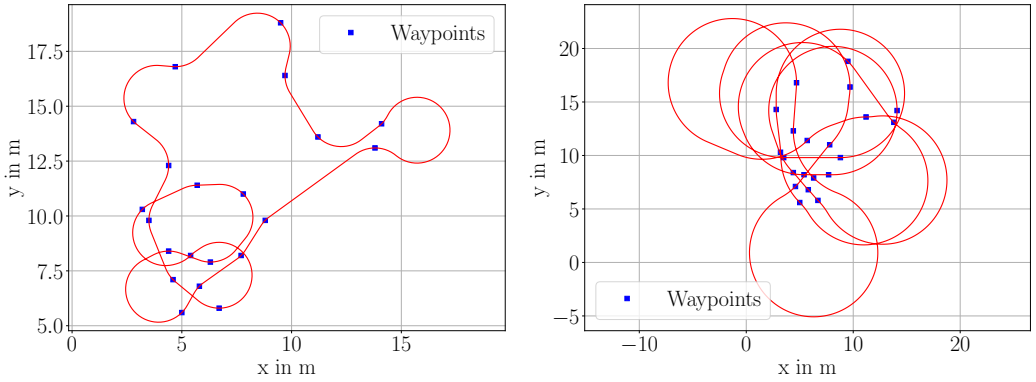
$$u_1 = 1 \tag{3.1f}$$

$$u_i \in \{2, ..., L\}, \qquad \forall i \in \{2, ..., L\} \tag{3.1g}$$

$$x_{ikg}^{jmw} \in \{0, 1\} \qquad \forall i, j \in \{1, ..., L\}; \forall k, m \in \{1, ..., H\} \tag{3.1h}$$

The objective of the presented mathematical programming formulation (3.1a) is to minimize the total required travel time. Constraint set (3.1b) enforces that each waypoint is entered exactly once for all heading angles whereas constraint set (3.1c) enforces that each waypoint is left exactly once. Flow conservation is fulfilled by constraint set (3.1d). These constraints ensure that each waypoint l_j is left in the same direction as it is entered. To prevent subtours, the presented model makes use of the subtour elimination constraints (3.1e), (3.1f), and (3.1g), which were developed in [96] and are an improved version of the Miller-Tucker-Zemlin Constraints [70]. Constraints (3.1h) enforce the decision variable x_{ik}^{jm} to be binary.

In Figure 3.1, we present two globally optimal solutions of the DTSP with different constant velocities as an example. For both, we define $H = 8$ and $\hat{a}_{max} = 1.5$ m/s^2. The optimum solution of the DTSP with $v_{const} = 1.5$ m/s requires a total mission duration of 52.92 s and is given in Figure 3.1a. The corresponding optimum solution for $v_{const} = 3.0$ m/s requires



(a) Optimum solution for the DTSP with v_{const} = 1.5 m/s. The required mission duration is 52.92 s.

(b) Optimum solution for the DTSP with v_{const} = 3.0 m/s. The required mission duration is 98.56 s.

Figure 3.1.: Example optimum solutions of the DTSP.

a total mission duration of 98.56 s and is given in Figure 3.1b. As can be seen, although the constant velocity of the righthand plot is double as high in the lefthand, the required mission duration is 86.24% higher due to occuring detours.

3.2.1.2. Kinematic Traveling Salesman Problem according to [25]

In the following, we introduce the KTSP as presented in our work [25].

As an extension of the DTSP, our KTSP model depends on a discretized set $\mathcal{V}$ of possible velocities to traverse each waypoint. This set depends on the maximum allowed velocity $\hat{v}_{max}$ and the number of spatial dimensions n and is represented by

$$\mathcal{V} = \{v_i \mid ((i-1) \cdot \hat{v}_{max})/(\sqrt{n} \cdot (V-1)), i \in \{1, ..., V\}\}$$

with $v_i \in \left[0, \hat{v}_{max}/\sqrt{2}\right]$ for $n = 2$. Note that we apply $\sqrt{n}$ to the above definition to enable the usage of our TOP-UAV++ trajectory planner as we state later. The number of discrete velocities is denoted by V. Analogously to the set of discretized heading angles, the discretization of the traversal velocities is an abstraction of the continuous reality to be able to mathematically model the problem.

Based on the set of waypoints $\mathcal{L}$, the set of discretized heading angles $\mathcal{H}$, and the set of discretized traversal velocities $\mathcal{V}$, we formally define the KTSP on a graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, where

$$\mathcal{N} = \{(l_i, h_k, v_g) \mid \\ \forall i \in \{1, ..., L\}, \forall k \in \{1, ..., H\}, \forall g \in \{1, ..., V\}\}$$

represents the set of unique nodes (l_i, h_k, v_g). Hence, graph $\mathcal{G}$ consists of a total of $L \cdot H \cdot V$ nodes. The set $\mathcal{E}$ of unique edges linking two nodes in graph $\mathcal{G}$ is represented as

$$\mathcal{E} = \{ \big((l_i, h_k, v_g), (l_j, h_m, v_w)\big) \mid \\ \forall i, j \in \{1, ..., L\}, \forall k, m \in \{1, ..., H\}, \forall g, w \in \{1, ..., V\}\}.$$

In total, there are $L^2 \cdot H^2 \cdot V^2$ edges in graph $\mathcal{G}$. Again, the costs c_{ikg}^{jmw} associated to each edge $\big((l_i, h_k, v_g), (l_j, h_m, v_w)\big) \in \mathcal{E}$ describes the required flight time to travel the edge. In this work, these costs are calculated for a given multirotor UAV defined by its maximum velocity and acceleration capabilities using our TOP-UAV++ approach presented in Section 2.4.4.

The resulting mathematical programming formulation for determining the optimal sequence of waypoints as well as the associated traversal heading angles and velocities is shown in the following.

The main decision variables for our formulation x_{ikg}^{jmw} are binary and interpreted as

$$x_{ikg}^{jmw} = \begin{cases} 1, & \text{if waypoint } l_i \text{ is left with heading angle } h_k \\ & \text{and velocity } v_g \text{ towards waypoint } l_j, \text{ which is} \\ & \text{entered with heading angle } h_m \text{ and velocity } v_w, \\ 0, & \text{otherwise} \end{cases}$$

Furthermore, there are integer decision variables $u_i \in \{1, ..., L\}, i = 1, ..., L$ that define the sequence of the associated waypoints l_i in the tour. The overall KTSP optimization model is given as follows:

$$\min \sum_{i=1}^{L} \sum_{j=1}^{L} \sum_{k=1}^{H} \sum_{m=1}^{H} \sum_{g=1}^{V} \sum_{w=1}^{V} x_{ikg}^{jmw} c_{ikg}^{jmw} \tag{3.2a}$$

s. t.

$$\sum_{j=1}^{L} \sum_{k=1}^{H} \sum_{m=1}^{H} \sum_{g=1}^{V} \sum_{w=1}^{V} x_{ikg}^{jmw} = 1, \qquad \forall i \in \{1, ..., L\} \tag{3.2b}$$

$$\sum_{i=1}^{L} \sum_{k=1}^{H} \sum_{m=1}^{H} \sum_{g=1}^{V} \sum_{w=1}^{V} x_{ikg}^{jmw} = 1, \qquad \forall j \in \{1, ..., L\} \tag{3.2c}$$

$$\sum_{i=1}^{L} \sum_{k=1}^{H} \sum_{g=1}^{V} x_{ikg}^{jmw} - \sum_{o=1}^{L} \sum_{p=1}^{H} \sum_{q=1}^{V} x_{jmw}^{opq} = 0 \qquad \forall j \in \{1, ..., L\}, \forall m \in \{1, ..., H\}, \forall w \in \{1, ..., V\} \tag{3.2d}$$

$$u_i - u_j + (L-1)\sum_{k=1}^{H}\sum_{m=1}^{H}\sum_{g=1}^{V}\sum_{w=1}^{V} x_{ikg}^{jmw} + (L-3)\sum_{k=1}^{H}\sum_{m=1}^{H}\sum_{g=1}^{V}\sum_{w=1}^{V} x_{jmw}^{ikg} \leq L-2$$

$$\forall i, j \in \{1, ..., L\} \tag{3.2e}$$

$$u_1 = 1 \tag{3.2f}$$

$$u_i \in \{2, ..., L\}, \qquad \forall i \in \{2, ..., L\} \tag{3.2g}$$

$$x_{ikg}^{jmw} \in \{0, 1\} \qquad \forall i, j \in \{1, ..., L\}$$
$$\forall k, m \in \{1, ..., H\}$$
$$\forall g, w \in \{1, ..., V\} \tag{3.2h}$$

The objective (3.2a) of the presented mathematical programming formulation (3.2) is to minimize the total required travel time. Constraint set (3.2b) enforces that each waypoint is entered exactly once for all heading angles and velocities whereas constraint set (3.2c) enforces that each waypoint is left exactly once. Flow conservation is fulfilled by constraint set (3.2d). These constraints ensure that task j is left in the same direction as it is entered as well as with the same velocity. To prevent subtours, we make use of the subtour elimination constraints (3.2e), (3.2f), and (3.2g), which were developed in [96] and are an improved version of the Miller-Tucker-Zemlin (MTZ) constraints [70]. Constraints (3.2h) enforce the decision variable x_{ikw}^{jml} to be binary.

In Figure 3.2, we present the globally optimal solution of the KTSP for the same problem instance as presented for the DTSP in Figure 3.1. For the KTSP, we defined $H = 8$ and $V = 6$. Further, we defined $\hat{v}_{max} = 3\,\text{m/s}$ and $\hat{a}_{max} = 1.5\,\text{m/s}^2$ and used our TOP-UAV++ trajectory planner to determine the edge costs. The required total mission duration is 34.03 s.

Note that our KTSP model itself is independent of the time-optimal trajectory generation approach developed in Section 2.4.4. Consequently, it is also independent of the kinematics of a multirotor UAV. In general, the KTSP can be defined for any inert system as long as a suitable trajectory planning approach exists that allows precise estimation of the edge costs.

3.2.2. Inertia-based Orienteering Problem Models

Analogously to Section 3.2.1, we present the Dubins orienteering problem (DOP) in Section 3.2.2.1 as we introduced in our work [87], followed by our more general kinematic orienteering problem (KOP) in Section 3.2.2.2 which we first introduced in [26]. Again, the DOP and KOP are NP-hard, since both problems are extensions of the orienteering problem which is known to be NP-hard (see e.g. [97]).

The DOP and KOP are motivated by the maximum flight time of a multirotor UAV in reality. Their objective is to find a kinematically feasible trajectory from an initial to a final waypoint and maximize the collected priorities by visiting waypoints in a given list of prioritized waypoints within the maximum flight time budget. Analogously to

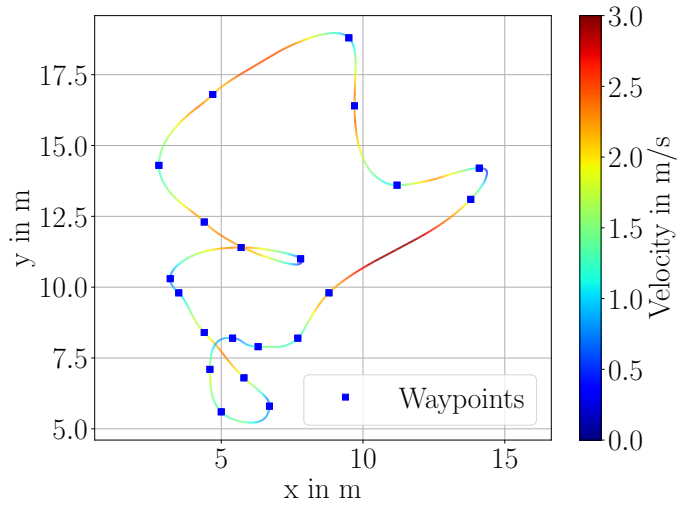


Figure 3.2.: Example solution for the KTSP. The required mission duration is 34.03 s.

Section 3.2.1, kinematically feasible means that a solution trajectory does not violate a given maximum velocity constraint and a given maximum acceleration constraint at any time.

The DOP and KOP are both defined in an obstacle-free environment and for both a set of waypoints

$$\mathcal{L} = \{l_i \mid l_i = i, i = 1, ..., L\}$$

is given where each element $l_i \in \mathcal{L}$ is assigned a priority $r_i \in \mathbb{R}_0^+$. Here, L describes the number of waypoints.

Following the mathematical models of the DTSP and KTSP presented in Section 3.2.1, we discretize the heading angles allowed to traverse each waypoint. This discretization is an abstraction from reality to be able to mathematically model the problem. We define the set of discretized heading angles to traverse each waypoint $l_i \in \mathcal{L}$ as

$$\mathcal{H} = \{h_i \mid h_i = 2\pi(i-1)/H, i = 1, .., H\}.$$

Here, H represents the number of equidistant discretization levels of the traversal heading angle.

With the definitions of $\mathcal{L}$ and $\mathcal{H}$, we present the DOP and KOP models in the following.

3.2.2.1. Dubins Orienteering Problem according to [87]

We introduce the Dubins orienteering problem (DOP) as stated in our work [87]. The DOP was first introduced in [22] but only solved heuristically, since their mathematical model does not suit to be solved by a general-purpose solver. In [84], the DOP is extended to the Dubins team orienteering problem (DTOP). They propose a route-based model and solve the problem using a branch-and-price approach. However, also their model is

not suited to be solved by a general-purpose solver. To be able to apply general-purpose solvers to the DOP and benchmark different solution approaches against DOP solutions, we propose a suitable model for the DOP in [87]. Our model is described in more detail in the following.

We define the Dubins orienteering problem for a set of prioritized waypoints $\mathcal{L}$ as stated in Section 3.2.2 that can be visited at discretized heading angles from the set $\mathcal{H}$. Start and end waypoints are represented by l_1 and l_L.

We formally define the DOP on a graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, where

$$\mathcal{N} = \{(l_i, h_k) \mid \forall i \in \{1, ..., L\}, \forall k \in \{1, ..., H\}\}$$

represents the set of unique nodes (l_i, h_k). Hence, the total number of nodes is $L \cdot H$. The set $\mathcal{E}$ of unique edges linking two nodes in graph $\mathcal{G}$ is represented as

$$\mathcal{E} = \{\big((l_i, h_k), (l_j, h_m)\big) \mid \forall i, j \in \{1, ..., L\}, \forall k, m \in \{1, ..., H\}\}.$$

There are $L^2 \cdot H^2$ edges in graph $\mathcal{G}$. Analogously to the definition of the DTSP in Section 3.2.1.1, the costs c_{ik}^{jm} associated to each edge $\big((l_i, h_k), (l_j, h_m)\big) \in \mathcal{E}$ describe the required flight time to travel the edge which is calculated as the length of the shortest Dubins path (see [31]) between both nodes divided by a given constant velocity v_{const}. Again, the optimum Dubins path depends on the minimum allowed turning radius of the multirotor UAV which is calculated as $v_{const}^2 / \hat{a}_{max}$ with $0 < v_{const} \leq \hat{v}_{max}$.

The main decision variables for our DOP formulation x_{ik}^{jm} are binary and interpreted as

$$x_{ik}^{jm} = \begin{cases} 1, & \text{if location } l_i \text{ is left with heading angle } h_k \\ & \text{towards location } l_j, \text{ which is} \\ & \text{entered with heading angle } h_m, \\ 0, & \text{otherwise} \end{cases}$$

Furthermore, the integer decision variables $u_i \in \{1, ..., L\}, i = 1, ..., L$ define the sequence of visited locations l_i in the tour. The overall DOP model is given as follows.

$$\max \sum_{i=2}^{L-1} \sum_{j=2}^{L} \sum_{k=1}^{H} \sum_{m=1}^{H} x_{ik}^{jm} r_i \tag{3.3a}$$

s. t.

$$\sum_{j=2}^{L} \sum_{k=1}^{H} \sum_{m=1}^{H} x_{1k}^{jm} = 1 \tag{3.3b}$$

$$\sum_{i=1}^{L-1}\sum_{k=1}^{H}\sum_{m=1}^{H} x_{ik}^{Lm} = 1 \tag{3.3c}$$

$$\sum_{i=1}^{L-1}\sum_{k=1}^{H}\sum_{m=1}^{H} x_{ik}^{jm} \leq 1 \qquad \forall j \in \{2, ..., L\} \tag{3.3d}$$

$$\sum_{i=1}^{L-1}\sum_{k=1}^{H} x_{ik}^{jm} = \sum_{o=2}^{L}\sum_{p=1}^{H} x_{jm}^{op} \qquad \forall j \in \{2, ..., L-1\};\ \forall m \in \{1, ..., H\} \tag{3.3e}$$

$$\sum_{i=1}^{L-1}\sum_{j=2}^{L}\sum_{k=1}^{H}\sum_{m=1}^{H} x_{ik}^{jm} c_{ik}^{jm} \leq C_{max} \tag{3.3f}$$

$$u_i - u_j + 1 \leq (L-1)\left(1 - \sum_{k=1}^{H}\sum_{m=1}^{H} x_{ik}^{jm}\right) \qquad \forall i, j \in \{1, ..., L\} \tag{3.3g}$$

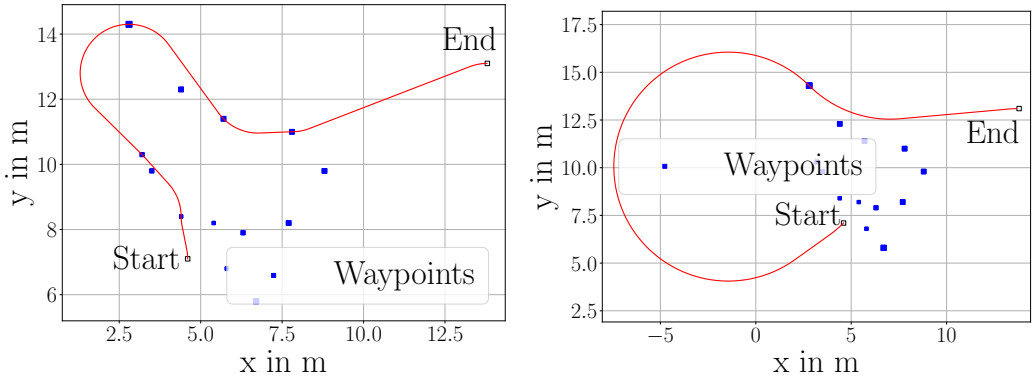
$$u_1 = 1 \tag{3.3h}$$

$$u_i \in \{2, ..., L\} \qquad \forall i \in \{2, ..., L\} \tag{3.3i}$$

$$x_{ik}^{jm} \in \{0, 1\} \qquad \forall i, j \in \{1, ..., L\};\ \forall k, m \in \{1, ..., H\} \tag{3.3j}$$

The objective of the presented mathematical programming formulation (3.3a) is to maximize the collected priorities. Constraint (3.3b) enforces that the start location is left whereas constraint (3.3c) enforces that the end location is entered. Constraint set (3.3d) ensures that each location is entered at most once. Flow conservation is fulfilled by constraint set (3.3e). These constraints further ensure that location l_j is left in the same direction as it is entered. Constraints (3.3f) ensure that the maximum allowed flight time is not exceeded. To prevent subtours, we make use of the subtour elimination constraints (3.3g), (3.3h) and (3.3i) which are formulated according to the Miller-Tucker-Zemlin (MTZ) formulation [70]. Constraints (3.3j) enforce the decision variable x_{ik}^{jm} to be binary.

In Figure 3.3, we present two globally optimal solutions of the DOP with different constant velocities as an example. For both, we define $H = 8$ and $\hat{a}_{max} = 1.5\,\mathrm{m/s^2}$. The maximum flight time budget is set to 15 s. The optimum solution of the DOP with $v_{const} = 1.5\,\mathrm{m/s}$ given in Figure 3.3a collects an aggregated priority value of 95. The corresponding optimum solution for $v_{const} = 3.0\,\mathrm{m/s}$ collects an aggregated priority value of 30 and is given in Figure 3.3b. Again, the detours due to the higher velocity lead to a significant reduction of the collected priorities. In both figures, each waypoint l_i except for the start and end waypoint is assigned a priority $r_i \in \{10, 15, 20, 25, 30\}$ which is indicated by the size of the associated blue markers. The larger the marker, the higher the associated priority. The maximum aggregated priority value that can be collected in the given problem instance is 230.



(a) Optimum solution for the DOP with v_{const} = 1.5 m/s. The collected priorities sum up to 95.

(b) Optimum solution for the DOP with v_{const} = 3.0 m/s. The collected priorities sum up to 30.

Figure 3.3.: Example optimum solutions of the DOP.

3.2.2.2. Kinematic Orienteering Problem according to [26]

In the following, we introduce the KOP as presented in our work [26]. As an extension of the DOP, our KOP model depends on a discretized set $\mathcal{V}$ of possible velocities to traverse each waypoint. This set depends on the maximum allowed velocity $\hat{v}_{max}$ and the number of spatial dimensions n and is represented by

$$\mathcal{V} = \{v_i \mid ((i-1) \cdot \hat{v}_{max})/(\sqrt{n} \cdot (V-1)), i \in \{1, ..., V\}\}$$

Again, we apply $\sqrt{n}$ to the above definition to enable the usage of our TOP-UAV++ trajectory planner. The number of discrete velocities is denoted by V. Analogously to the set of discretized heading angles, the discretization of the traversal velocities is an abstraction of the continuous reality to be able to mathematically model the problem.

Analogously to Section 3.2.1.2 on the KTSP, we formally define the KOP on a graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, where

$$\mathcal{N} = \{(l_i, h_k, v_g) \mid \forall i \in \{1, ..., L\}, \forall k \in \{1, ..., H\}, \forall g \in \{1, ..., V\}\}$$

represents the set of unique nodes (l_i, h_k, v_g). Overall, there exist $L \cdot H \cdot V$ nodes in graph $\mathcal{G}$. The set $\mathcal{E}$ of unique edges linking two nodes in graph $\mathcal{G}$ is represented as

$$\mathcal{E} = \{ \big((l_i, h_k, v_g), (l_j, h_m, v_w)\big) \mid \forall i, j \in \{1, ..., L\}, \forall k, m \in \{1, ..., H\}, \forall g, w \in \{1, ..., V\}\}.$$

Overall, there exist $L^2 \cdot H^2 \cdot V^2$ edges in graph $\mathcal{G}$. The edge costs c_{ikg}^{jml} associated to each edge $\big((l_i, h_k, v_g), (l_j, h_m, v_w)\big)$ describes the required flight time to travel the edge and is

determined by our TOP-UAV++ trajectory generation method presented in Section 2.4.4. The maximum flight time is denoted as C_{max}.

The mathematical programming formulation for determining the priority-maximizing sequence of waypoints as well as the associated heading angles and velocities, i.e. for solving the KOP, is shown in the following.

The main decision variables for our formulation x_{ikg}^{jml} are binary and interpreted as

$$x_{ikg}^{jml} = \begin{cases} 1, & \text{if location } l_i \text{ is left with heading angle } h_k \\ & \text{and velocity } v_g \text{ towards location } l_j, \text{ which is} \\ & \text{entered with heading angle } h_m \text{ and velocity } v_l, \\ 0, & \text{otherwise} \end{cases}$$

Furthermore, integer decision variables $u_i \in \{1, ..., L\}, i = 1, ..., L$ define the sequence of visited locations l_i in the tour.

The overall KOP model is given as follows.

$$\max \sum_{i=2}^{L-1} \sum_{j=2}^{L} \sum_{k=1}^{H} \sum_{m=1}^{H} \sum_{g=1}^{V} \sum_{w=1}^{V} x_{ikg}^{jmw} r_i \tag{3.4a}$$

s.t.

$$\sum_{j=2}^{L} \sum_{k=1}^{H} \sum_{m=1}^{H} \sum_{w=1}^{V} x_{1k1}^{jmw} = 1 \tag{3.4b}$$

$$\sum_{i=1}^{L-1} \sum_{k=1}^{H} \sum_{m=1}^{H} \sum_{g=1}^{V} x_{ikg}^{Lm1} = 1 \tag{3.4c}$$

$$\sum_{i=1}^{L-1} \sum_{k=1}^{H} \sum_{m=1}^{H} \sum_{g=1}^{V} \sum_{w=1}^{V} x_{ikg}^{jmw} \leq 1 \qquad \forall j \in \{2, ..., L\} \tag{3.4d}$$

$$\sum_{i=1}^{L-1} \sum_{k=1}^{H} \sum_{g=1}^{V} x_{ikg}^{jmw} - \sum_{o=2}^{L} \sum_{p=1}^{H} \sum_{q=1}^{V} x_{jmw}^{opq} = 0 \qquad \forall j \in \{2, ..., L-1\}, \forall m \in \{1, ..., H\}, \forall w \in \{1, ..., V\} \tag{3.4e}$$

$$\sum_{i=1}^{L-1} \sum_{j=2}^{L} \sum_{k=1}^{H} \sum_{m=1}^{H} \sum_{g=1}^{V} \sum_{w=1}^{V} x_{ikg}^{jmw} c_{ikg}^{jmw} \leq C_{max} \tag{3.4f}$$

$$u_i - u_j + 1 \leq (L-1) \left(1 - \sum_{k=1}^{H} \sum_{m=1}^{H} \sum_{g=1}^{V} \sum_{w=1}^{V} x_{ikg}^{jmw} \right) \qquad \forall i, j \in \{1, ..., L\} \tag{3.4g}$$

$$u_1 = 1 \tag{3.4h}$$

$$u_i \in \{2, ..., L\} \qquad \forall i \in \{2, ..., L\} \tag{3.4i}$$

$$x_{ikg}^{jmw} \in \{0, 1\} \qquad \forall i, j \in \{1, ..., L\}, \forall k, m \in \{1, ..., H\}, \forall g, w \in \{1, ..., V\} \tag{3.4j}$$

The objective of the presented mathematical programming formulation (3.4a) is to maximize the collected priorities. Constraint (3.4b) enforces that the start location is left in any direction with the start velocity at rest whereas constraint (3.4c) enforces that the end location is entered from any direction with the final velocity at rest. Constraint set (3.4d) ensures that each location is entered at most once. Flow conservation is fulfilled by constraint set (3.4e). These constraints also ensure that location l_j is left in the same direction as it is entered as well as with the same velocity. Constraints (3.4f) ensure that the maximum allowed flight time is not exceeded. To prevent subtours, we make use of the subtour elimination constraints (3.4g), (3.4h) and (3.4i) which are formulated according to the Miller-Tucker-Zemlin (MTZ) formulation [70]. Constraints (3.4j) enforce the decision variable x_{ikg}^{jmw} to be binary.

In Figure 3.4, we present the optimum solution of the KOP for the same problem instance as presented for the DOP in Figure 3.1. The number of traversal velocities is set to $V = 6$, and the number of traversal heading angles is set to $H = 8$. The maximum flight time budget is again set to 15 s. As the trajectory generation approach to determine the edge costs between waypoints, we use TOP-UAV++. The maximum allowed velocity is $\hat{v}_{max} = 3$ m/s, and the maximum allowed acceleration is $\hat{a}_{max} = 1.5$ m/s^2. In the given figure, each waypoint l_i except for the start and end waypoint is assigned a priority $r_i \in \{10, 15, 20, 25, 30\}$ which is indicated by the size of the associated blue markers. The larger the marker, the higher the associated priority. The aggregated collected priorities by the KOP solution sum up to 135. The maximum aggregated priority value that can be collected in the given problem instance is 230.

Analogously to the KTSP defined in Section 3.2.1.2, the KOP model itself is independent of the time-optimal trajectory generation approach developed in Section 2.4.4. In general, the KOP can be defined for any inert system as long as a suitable trajectory planning approach exists that allows precisely estimating the edge costs.

3.3. Heuristic Solution Frameworks

In this Section, we present the heuristic solution frameworks to solve the KTSP and KOP presented in Sections 3.2.1.2 and 3.2.2.2. Both frameworks are based on the adaptive large neighborhood search (ALNS) which was first introduced in [24] and utilizes an iterative remove-and-insert procedure (see [93]). ALNS is a widely used metaheuristic solution approach in routing problems and demonstrated to yield high-quality results (see e.g.

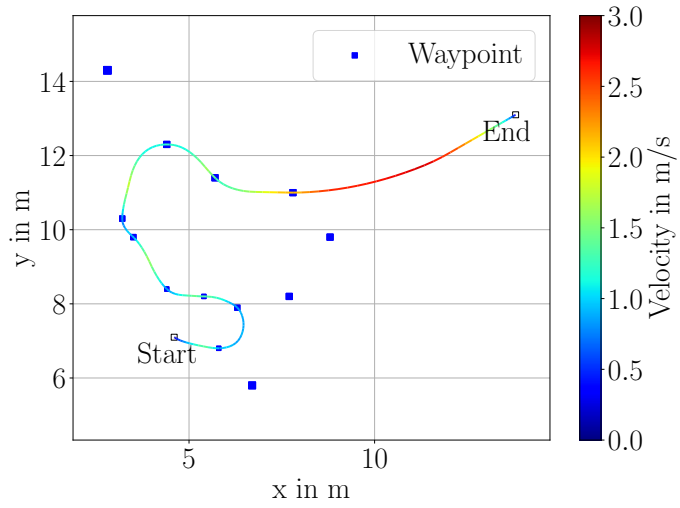


Figure 3.4.: Example solution for the KOP. The total priorities sum up to 135.

the surveying articles [95, 94]). As we state in Section 3.1.2, developing and comparing different heuristics solution approaches is out of the scope of this work. Instead, we focus on the development of heuristic solution approaches for the KTSP, but also the KOP, to show that heuristic solution approaches are sufficient to exploit the advantage of the mathematical KTSP and KOP models despite their higher complexity. For this purpose, we use well-established ideas from the literature and modify them to fit the needs for solving our kinematic version of the TSP and the OP.

The remainder of this section is as follows: First, we give the reader a brief introduction to how the general ALNS metaheuristic works in Section 3.3.1. With these foundations in mind, we start to introduce the operators used to remove and insert waypoints in a current solution representation in Sections 3.3.2 and 3.3.3. These operators originate from existing literature and are adapted for the needs of this work. Next, we make use of a technique that is widely used in the literature on heuristic solution approaches for the DTSP and DOP (see e.g. [78, 79, 22]), which is the optimization of the traversal heading angles for each waypoint of a given sequence via dynamic programming (DP). Since the KTSP and the KOP additionally consider the traversal velocity for each waypoint in a sequence, we adapt this idea such that heading angles and velocities are considered. We present our DP-based approach in Section 3.3.4. Next, at the end of each iteration of an ALNS approach, it must be decided if the generated solution is used as the start solution for the next iteration, or not. This decision can be automated by using acceptance criteria. In Section 3.3.5, we briefly present three different acceptance criteria from the literature and give a detailed introduction to one of them, namely the simulated annealing (SA), since it significantly outperformed its competitors in preliminary experiments.

With the removal and insertion operators, as well as the DP traversal property optimization and the acceptance criteria introduced, we present our ALNS-based solvers for the KTSP (see Section 3.3.7) and the KOP (see Section 3.3.8) including the initial solution construction in Section 3.3.6

Overall, in this Section, we develop heuristic solution approaches for new classes of routing problems by composing existing ideas from the literature.

Note that our ALNS-based solvers depend on numerous hyperparameters, which affect their performance. Hence, at the beginning of our computational study, we present the hyperparameter optimization for our solvers (see Section 3.4.4).

3.3.1. The General ALNS Solution Framework

In this Section, we briefly introduce the foundations of the ALNS metaheuristic (see [24, 98, 94]). We selected ALNS as a heuristic solution framework since it is widely used especially for routing problems (see [95]). In general, an ALNS is an extension of the classical large neighborhood search (LNS) presented by [99]. The main idea of an LNS is to search through the neighborhoods of a solution which are implicitly defined by applying simple but fast heuristics that are applied to destroy and repair a solution [94]. This process is also known as the ruin-and-recreate mechanism which is first discussed in [93]. In an LNS, the removal and insertion heuristics, which implement the destroy and repair process, are selected based on a static policy, which indicates the difference between LNS and ALNS. In contrast to the LNS, an ALNS selects the removal and insertion heuristics based on a dynamically adaptive routine, which ensures that heuristics that performed well in the past are more likely to be selected in future iterations. This dynamic routine is realized using weights, which are assigned to each removal and insertion heuristic. These weights are dynamically updated during the search process and leveraged to select well-performing removal and insertion heuristics in each iteration. Note that the additional complexity of the adaptive weights adjustment is especially profitable when the best-performing heuristics change during the search process (see [100]).

In the following, we present the general procedure of the ALNS illustrated by Algorithm 2. The weights for the removal heuristics are further denoted as $\gamma_{\mathcal{R}}$, the ones for the insertion heuristics as $\gamma_{\mathcal{I}}$. In this general introduction as well as in the remainder of this work, the selection of removal and insertion heuristics is done via roulette-wheel selection using a uniform distribution. The general algorithm of the ALNS metaheuristic works as follows. First, the sets of insertion heuristics $\mathcal{I}$ and removal heuristics $\mathcal{R}$ are defined as well as a feasible initial solution s (see lines 1-3). Then the best solution is defined as the initial solution and the weights for the selection of each heuristic are initialized with one (lines 4-5). Next, the optimization loop is entered in line 6 of the pseudocode. It starts by the selection of a removal and insertion heuristic for the current iteration (see lines 7-8). The probability $P(\lambda^i_{\mathcal{R}})$ for selecting the i-th removal heuristic $\lambda^i_{\mathcal{R}} \in \mathcal{R}$ is based on its associated weight $\gamma^i_{\mathcal{R}}$ and calculated as

$$P(\lambda^i_{\mathcal{R}}) = \frac{\gamma^i_{\mathcal{R}}}{\sum_{\gamma^j_{\mathcal{R}} \in \gamma_{\mathcal{R}}} \gamma^j_{\mathcal{R}}}. \tag{3.5}$$

Algorithm 2: General procedure of the ALNS

```
1  Input: Set of insertion heuristics 𝓘
2  Input: Set of removal heuristics ℛ
3  Input: A feasible initial solution s
4  s_best = s; γ_𝓘 = (1, ..., 1); γ_ℛ = (1, ..., 1)
5  stop = false
6  while stop == false do
7      select removal heuristic λ^i_ℛ ∈ ℛ
8      select insertion heuristic λ^i_𝓘 ∈ 𝓘
9      s_new = s.removal(λ^i_ℛ).insertion(λ^i_𝓘)
10     if accepted(s_new, s) then
11         s = s_new
12     if J(s_new) better than J(s_best) then
13         s_best = s_new
14     Update weights γ_𝓘, γ_ℛ
15     if stopping criterion met then
16         stop = true
17 return: s_best
```

The probability $P(\lambda^i_{\mathcal{I}})$ for selecting the i-th insertion heuristic $\lambda^i_{\mathcal{I}}$ is analogous. It is determined as

$$P(\lambda^i_{\mathcal{I}}) = \frac{\gamma^i_{\mathcal{I}}}{\sum_{\gamma^j_{\mathcal{I}} \in \gamma_{\mathcal{I}}} \gamma^j_{\mathcal{I}}}. \tag{3.6}$$

Then, the removal procedure using the selected removal heuristic is applied followed by the insertion procedure to repair the solution based on the selected insertion heuristic as given in line 9. Next, the acceptance criterion of the generated solutions is evaluated in lines 10 - 11. If the new solution passes the acceptance criterion then it is defined as the start solution for the next iteration. Next, it is evaluated if a new best solution is found in lines 12-13. if this holds, the best solution found is updated. In line 14, the weights $\gamma_{\mathcal{I}}, \gamma_{\mathcal{R}}$ are updated. This is mostly done by assessing if the heuristics used for the current iteration contributed to finding a new best solution, an accepted solution, or a solution that has not been found before to enhance exploration. Finally, a particular stopping criterion is evaluated and in case it decides to stop, the termination of the ALNS is initiated and the best solution found is returned (see lines 15-17).

In the following section, we present the removal and insertion operators that we utilize for our ALNS solvers.

Removal heuristic	Description	KTSP	KOP
MD	Most distance	✓	✗
MAFT	Most additional flight time	✓	✗
WA	Worst angle	✓	✓
RW	Random waypoint	✓	✓
RS	Random sequence	✓	✓
LP	Least priority	✗	✓
MR	Most ratio	✗	✓

Table 3.1.: Overview of utilized removal heuristics.

3.3.2. Removal Heuristics

In this section, we present the removal heuristics that we use in our ALNS solvers. Among the numerous removal heuristics that can be found in the literature, the heuristics we present here provided good performance in preliminary computational experiments and hence, are utilized for the ALNS frameworks to solve the KTSP and KOP in this work. Note that a full study on the best composition of removal heuristics is out of the scope of this work. An overview of the used removal heuristics is given in Table 3.1 together with an indication if the corresponding heuristic is used to solve the KTSP, the KOP, or both.

In general, the removal heuristics that we apply are based on the assumption that during the search process of the ALNS, there exist two sets of waypoints. The first one is an ordered set that contains all waypoints in the exact order as they are visited. Therefore, this ordered set represents the current solution that is to be executed by the UAV. The second set is unordered and contains the waypoints not visited in the current solution. The task of the removal heuristic is to select a waypoint that has to be removed from the current solution to the unordered set of unvisited waypoints.

One important aspect is the termination of the removal process. In this work, we define the number of waypoints that have to be removed from the current solution representation a-priori. More details on how this number is determined are given in Sections 3.3.7 and 3.3.8.

In the following, we present all removal heuristics applied in this work.

MD removal:
→*Usage: KTSP*
The Most Distance (MD) removal heuristic recurrently removes the waypoint from the current solution, where the associated sum of Euclidean distances to its adjacent waypoints is highest. The MD removal operator can be found e.g. in [101] and corresponds to the class of "worst removal" operators which are described in [98]. Worst removal operators have been applied in numerous ALNS approaches in different routing problem domains (see [101, 98, 11, 102, 24]).

MAFT removal:
→Usage: KTSP
The Most Additional Flight Time (MAFT) removal determines the sum of flight times required to connect a particular waypoint with its adjacent waypoints with fixed traversal properties, and subtracts the flight time required to directly connect its predecessor with its successor. The resulting value represents the additional flight time required to incorporate the associated waypoint at its current position into the tour. This procedure is conducted for all waypoints in the tour. The waypoint with the most additional flight time is then removed from the current solution representation. This removal operator also corresponds to the class of "worst removal" operators (see [101, 98, 11, 102, 24]). Note that the MAFT removal represents the counterpart of the LAFT insertion heuristic which we introduce in Section 3.3.3.

WA removal:
→Usage: KTSP, KOP
The Worst Angle (WA) removal is based on a metric, that estimates if a waypoint is placed well (i.e. can be connected efficiently) between its predecessor and successor. This metric evaluates the angle $\angle(i-1, i, i+1) \in [0, \pi]$ that spans between the arc between the $(i-1)$-th and the i-th waypoint and the arc between the i-th and the $(i+1)$-th waypoint and assumes that acute angles correspond to inefficient motions. Overall, this metric is evaluated for all waypoints in the current solution representation excluding the start and end waypoints. The waypoint l_i that is associated with the acutest angle is removed. A similar type of insertion heuristic is introduced in [11]. WA removal is another implementation of the "worst removal" operators class that is widely used in the literature (see e.g. [101, 98, 11, 102, 24]).

RW removal:
→Usage: KTSP, KOP
The Random Waypoint (RW) removal randomly selects a waypoint from the current solution and removes it. Random removal operators are widely used in many different domains (see[101, 98, 11, 102, 24, 103]) to enhance the exploration of the solution space.

RS removal:
→Usage: KTSP, KOP
Given the number of $n \in \mathbb{N}$ waypoints that are allowed to be removed in the current removal process, the Random Sequence (RS) heuristic randomly selects a number $\hat{n}$ from the integer interval $[1, ..., n]$ from a uniform distribution. Further, a seed position in the current solution is selected randomly based on a uniform distribution. Starting from this seed position the $\hat{n}$ consecutive waypoints are removed in a batch. For the KTSP case, the seed position can be arbitrarily selected among all positions of waypoints in the current solution representation. In the KOP case, the seed position is randomly selected within the integer interval $[2, ..., N - \hat{n}]$. Here, N denotes the number of waypoints visited in the current solution representation. The waypoint at position 1 is excluded and cannot be removed since it represents the given start waypoint. This analogously holds for the waypoint at positions $> N - \hat{n}$ since that would imply removing the given end waypoint.

Random sequence removal operators are used e.g. in [104, 11].

LP removal:
→*Usage: KOP*
The Least Priority (LP) removal identifies the waypoint in the current tour assigned with the lowest priority and removes it. If multiple waypoints with equal priority exist, then the selection is made randomly among these minimum-priority waypoints based on a uniform distribution. LP removal belongs to the class of "worst removal" operators that are commonly used in the literature (see e.g. [101, 98, 11, 102, 24]).

MR removal:
→*Usage: KOP*
The Most Ratio (MR) removal calculates the additional flight time used for visiting a particular based on its predecessor and successor. This means the sum of flight times of reaching the given waypoint from its predecessor and reaching its successor from the given waypoint is determined. Next, the flight time required for moving immediately from the predecessor to the successor with their traversal properties fixed as in the current solution is subtracted from this sum. This value represents the additional flight time currently invested to visit the investigated waypoint. Finally, the ratio of the additional flight time divided by the priority assigned to the given waypoint is determined. This process is conducted for all waypoints in the current solution except for the start and end waypoints. The waypoint with the highest ratio is removed since it represents the worst trade-off between invested flight time and collected priorities. MR removal represents the counterpart of the least ratio (LR) insertion which we introduce in Section 3.3.3 and belongs to the class of "worst removal" operators (see e.g. [101, 98, 11, 102, 24]).

3.3.3. Insertion Heuristics

Analogously to Section 3.3.2, we introduce the insertion heuristics that performed best in preliminary computational experiments in this section. An overview of all used insertion heuristics in this work is given in Table 3.2. For each insertion heuristic given, we further indicate if the heuristic is used to solve the KTSP, the KOP, or both.

In contrast to the removal heuristics introduced in Section 3.3.2, the task of the insertion heuristics is to move waypoints from the unordered set of unvisited waypoints to a potentially good position in the ordered set of visited waypoints. The terminology "good" corresponds to the desire to yield a new best solution at the end of the insertion process. In the KTSP case, the insertion process stops when the set of unvisited waypoints is empty. In the KOP case, the insertion process stops when no further waypoint can be inserted into the tour without violating the maximum flight time budget, or when the set of unvisited waypoints is empty.

In the following, all insertion heuristics developed for this work are introduced.

RW insertion:
→*Usage: KTSP, KOP*
The Random Waypoint (RW) insertion randomly selects a waypoint of the set of unvisited waypoints and greedily inserts it into the current solution at the position where it can

Insertion heuristic	Description	KTSP	KOP
RW	Random waypoint	✓	✓
LAFT	Least additional flight time	✓	✓
MP	Most priority	✗	✓
LR	Least ratio	✗	✓

Table 3.2.: Overview of utilized insertion heuristics for the ALNS for the KTSP and the KOP.

be visited with the overall minimum additional flight time. This means that the selected waypoint is evaluated between each consecutive waypoint pair in the current solution, which is represented by the current ordered set of waypoints. For each possible insertion position, the optimum traversal velocity $v \in \mathcal{V}$ and angle $h \in \mathcal{H}$ is determined by minimizing the additional flight time. In the end, the waypoint is inserted with a velocity and angle configuration between the best-fitting consecutive waypoints such that the overall required additional flight time is minimized. For the KOP case: If the sum of the additional required flight time for the insertion and the total duration of the current solution exceeds the maximum flight time budget, no feasible insertion is found and the insertion process for the ALNS-iteration terminates. RW insertion is also used analogously in different routing problems (see e.g. [104])

LAFT insertion:
→*Usage: KTSP, KOP*
The Least Additional Flight Time (LAFT) insertion is similar to the RW-insertion, but conducted for all unvisited waypoints. Hence, it is more powerful but also more time-consuming. The one waypoint among all unvisited waypoints that can be inserted into the current trip with minimum additional flight time is inserted at the corresponding position. Again, if the resulting total trip duration exceeds the maximum flight time budget of the KOP, no insertion is conducted and the insertion process of the current ALNS iteration terminates. LAFT insertion belongs to the general class of "greedy insertion" heuristics (see e.g. [98, 104, 11, 102])

MP insertion:
→*Usage: KOP*
The Most Priority (MP) insertion heuristic determines a subset of waypoints from the set of unvisited waypoints that are assigned the highest priority. For each waypoint in this subset containing the unvisited highest priority waypoints, the best position in the current solution representation is determined which enables the insertion of the associated waypoint into the tour with minimum additional flight time while simultaneously ensuring that the resulting total mission duration does not exceed the maximum flight time budget. In the end, the highest-priority waypoint that can be feasibly inserted with minimum additional flight time into the solution is inserted at the best-fitting position. If no feasible insertion is found, then the insertion process of the current ALNS iteration terminates. MP insertion is also part of the class of "greedy insertion" heuristics (see e.g. [98, 104, 11, 102])

LR insertion:
→*Usage: KOP*
The Least Ratio (LR) insertion iterates over all unvisited waypoints. For each waypoint, it determines the minimum additional flight time to feasibly insert the waypoint into the current solution. If the waypoint cannot be inserted without violating the maximum flight time budget, then the corresponding candidate is excluded from the insertion process. Next, the ratio of the minimum additional flight time divided by the associated priority is determined. This ratio serves as a metric quantifying the trade-off of investing additional flight time and the priorities collected by this investment. A low ratio represents that a high-priority waypoint can be inserted into the current solution by investing low additional flight time. If no feasible insertion can be found, then no insertion is conducted and the insertion process of the current ALNS iteration terminates. LR insertion is also part of the class of "greedy insertion" heuristics (see e.g. [98, 104, 102]). The explicit consideration of the ratio between the additionally collected priorities and invested flight time can be seen e.g. in [11].

3.3.4. Waypoint Traversal Optimization via Dynamic Programming

Due to inertia, the quality of the insertion of a new waypoint into the current solution, i.e. an ordered set of waypoints, does not simply depend on the traversal angle and velocity of the inserted waypoint itself but also on the traversal properties of its predecessors and successors. Hence, the optimum traversal velocity and heading angles of each waypoint in the tour can change after each insertion. Therefore, we aim to ensure optimality in the traversal properties of each waypoint in the solution by using dynamic programming (DP). Due to its computational efficiency, this approach is commonly used in the literature on related routing problems using the Dubins vehicle (see [22, 78, 79]).

Assume a new waypoint $l_i \in \mathcal{L}$ is inserted into an existing sequence of waypoints. The first and last waypoints are assumed to be traversed with a predefined and immutable traversal property. All other waypoints can be traversed at an arbitrary heading angle $h \in \mathcal{H}$ and an arbitrary velocity $v \in \mathcal{V}$. The problem at hand is how to travel from the start to the end waypoint by traversing the given sequence of waypoints in exactly the given order with minimum time. The underlying problem represents a shortest path problem with the associated search graph given in Figure 3.5. In the graph the start node is defined by a fixed heading angle and velocity h_s, v_s. Beginning with the start node, it has to be decided how to traverse each waypoint in the sequence of waypoints among all possible combinations of $h_i \in \mathcal{H}, v_i \in \mathcal{V}$ until the final waypoint is visited with a fixed heading angle and velocity h_e, v_e.

To understand the behavior of the DP approach used to solve this problem, one essential expression is introduced in Equation (3.7) which is also known as the Bellman equation and which was first presented in [105]. Generally speaking, it describes that the optimal costs to reach a certain node in the graph are determined based on the optimal costs

to reach its predecessor and the cost of reaching the given node from that particular predecessor. In this context, a node is represented by a tuple (i, h, v). These nodes are described by the identifier i specifying the i-th waypoint in a given sequence of waypoints, the heading angle $h \in \mathcal{H}$, and the velocity $v \in \mathcal{V}$. The costs $c^{ihv}_{(i-1)kg}$ to move between nodes $(i-1, k, g)$ and (i, h, v) represent the required flight times which we determined by using TOP-UAV++ trajectory planner. The optimal costs to reach the node (i, h, v) are calculated by

$$costs^*(i, v, h) = \min_{\substack{k=1,\dots,H \\ g=1,\dots,V}} \left\{ costs^*(i-1, k, g) + c^{ihv}_{(i-1)kg} \right\}. \tag{3.7}$$

Analogously, the optimal traversal property required to reach node (i, h, v) from its predecessor waypoint can be obtained by solving

$$u^*(i, v, h) = \arg\min_{\substack{k=1,\dots,H \\ g=1,\dots,V}} \left\{ costs^*(i-1, k, g) + c^{ihv}_{(i-1)kg} \right\}. \tag{3.8}$$

With Equations (3.7) and (3.8) introduced, we now present the dynamic programming procedure used to optimize the traversal properties of a given sequence of waypoints. This procedure is shown in the Algorithm 3.

In the first step, starting with the given traversal properties of the start waypoint, we determine the optimal cost to reach the nodes specified with $i = 1$ and all of its possible traversal configurations by calculating the time optimal trajectory duration between each node pair. For each of these nodes, the optimal traversal properties of its predecessor are assigned the immutable traversal property of the start node.

In the second step, we exploit that iteratively the optimal costs to reach all of the possible predecessors of the nodes specified with $i \in \{2, \dots, L-1\}$ are determined in the previous iteration. Hence, to determine the optimal cost to reach node (i, h, v) Equation (3.7) has to be applied. The optimal traversal properties required at waypoint $i-1$ to reach node (i, h, v) are calculated using Equation (3.8).

The third step is similar to the second step. However, since the last waypoint L is assigned an immutable traversal property, Equations (3.7) and (3.8) have to be applied only once once to obtain the optimal costs to reach that very last node and to obtain the optimal traversal properties of its predecessor waypoint $L-1$. Note that the third step can be interpreted as a special case of the second step with only a single node given.

At this point, the optimal traversal properties of each node in the sequence of waypoints can be obtained by iterating over all $u^*(i, v, h)$ backward. This means that beginning from the last node and in a recurrent manner, the traversal properties of the predecessor waypoint are investigated which specify the preceding node. This process is repeated until the start node is reached and the optimal traversal property of the entire sequence

Algorithm 3: Dynamic programming

1 //1st step: initialize first free configuration
2 **for** $h = 1, ..., H$: **do**
3 **for** $v = 1, ..., V$: **do**
4 $\text{costs}^*(1, h, v) = c^{1hv}_{s,k_0,g_0}$
5 $\text{u}^*(1, h, v) = (k_0, g_0)$
6 //2nd step: run through the entire sequence
7 **for** $i - 1 = 1, ..., L - 1$: **do**
8 **for** $h = 1, ..., H$: **do**
9 // successor heading
10 **for** $v = 1, ..., V$: **do**
11 // successor velocity
12 $costs^*(i, h, v) = \min_{\substack{k=1,...,H \\ g=1,...,V}} \left\{ costs^*(i-1, k, g) + c^{ihv}_{(i-1)kg} \right\}$
13 $u^*(i, h, v) = \arg\min_{\substack{k=1,...,H \\ g=1,...,V}} \left\{ \text{costs}^*(i-1, k, g) + c^{ihv}_{(i-1)kg} \right\}$
14 //3rd step: last node
15 $costs^*(L, h_L, v_L) = \min_{\substack{k=1,...,H \\ g=1,...,V}} \left\{ costs^*(L-1, k, g) + c^{Lh_Lv_L}_{(L-1)kg} \right\}$
16 $u^*(L, h_L, v_L) = \arg\min_{\substack{k=1,...,H \\ g=1,...,V}} \left\{ \text{costs}^*(L-1, k, g) + c^{Lh_Lv_L}_{(L-1)kg} \right\}$
17 // 4th step: reconstruct solution from back
18 updateHeadingAnglesAndVelocities( )

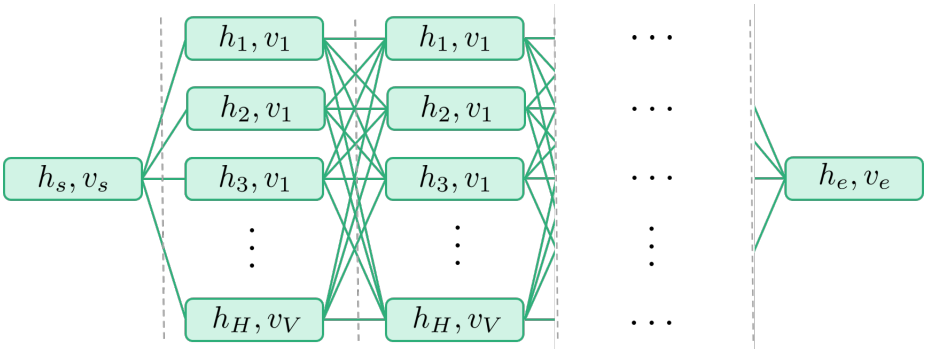


Figure 3.5.: Visualization of the graph for the DP approach.

of waypoints is determined. In line 18 of Algorithm 3 this process is abbreviated as updateHeadingAnglesAndVelocities( ).

Note that we use the DP approach presented in Algorithm 3 in two different ways. First, we use it to optimize the heading angles and velocities of all visited waypoints. This use case is further referred to as global DP optimization of the traversal properties. However, since we assume that the impact of inserting a new waypoint into an existing sequence only

affects its immediate neighbors in the sequence, we also use the DP algorithm to locally optimize the traversal properties for the $\eta \in \mathbb{N}$ immediate successors and predecessors. Analogously, this use case is further referred to as local DP optimization of the traversal properties. In Sections 3.4.4.1 and 3.4.4.2, we derive proper values of η for the KTSP and the KOP via hyperparameter optimization.

3.3.5. Acceptance Criteria

One crucial element in applying metaheuristics to solve combinatorial optimization problems such as the KTSP and the KOP is the acceptance of solutions for future iterations. This means the following: in each iteration of the ALNS solution process, a new solution is generated based on an existing one. This new solution serves as a starting point for the next iteration, i.e. for the removal and insertion process, if it is accepted. Otherwise, the previous solution serves as a starting point again.

The set of potential acceptance criteria is manifold and each criterion differently affects e.g. the exploration of the solution space, or the ability to escape local optima.

In this section, we briefly introduce the acceptance criteria that have been investigated in this work. These are simulated annealing, hill climbing and tabu search. Since preliminary results showed that the simulated annealing approach significantly outperformed the hill climbing and tabu search approach, we describe the simulated annealing approach more deeply.

Simulated annealing:
Simulated annealing is the most sophisticated acceptance criterion used in this work. It is inspired by statistical mechanics, that investigates "[...] the behavior of a system with many degrees of freedom in the thermal equilibrium of a finite temperature [...]" (see [106]). Based on the difference of the objective function values J of the current solution s_k and a new candidate solution s_{k+1} and a fictional temperature parameter τ_k the acceptance probability P_A of the new solution s_{k+1} for the next iteration is determined for minimization and maximization problems as follows:

$$min: \quad P_A = e^{-\frac{J(s_{k+1})-J(s_k)}{\tau_k}} \tag{3.9}$$

$$max: \quad P_A = e^{\frac{J(s_{k+1})-J(s_k)}{\tau_k}} \tag{3.10}$$

If P_A is greater than 1, then $P_A = 1$ is set. Starting from a start temperature τ_0, the temperature of the $(k+1)$-th iteration τ^{k+1} is determined as $\tau_{k+1} = \vartheta \cdot \tau_k$, whereas for the cooling parameter $\vartheta \in \mathbb{R}$ holds $0 \leq \vartheta \leq 1$.

A common approach to cope with problem instances of different sizes, i.e. with different numbers of waypoints, is to define the start temperature by utilizing the objective value of the initial solution and a start temperature control parameter (see [98]). With this approach, the start temperature is defined in a way that a new solution is accepted with a

probability 50 % if it is ω times worse than the initial solution. From Equation (3.9), we derive

$$e^{-\frac{\omega \cdot J(s_0)}{\tau_0}} = 0.5 \tag{3.11}$$

$$-\omega \cdot J(s_0) = ln(0.5) \cdot \tau_0 \tag{3.12}$$

$$\tau_0 = -\frac{\omega \cdot J(s_0)}{ln(0.5)}, \tag{3.13}$$

where ω represents the start temperature control parameter, $J(s_0)$ represents the objective value of the initial solution, and τ_0 represents the start temperature of the simulated annealing process. We determine the start temperature control parameter ω and cooling parameter ϑ as part of a hyperparameter optimization as described in Section 3.4.4.

Preliminary results showed that the simulated annealing approach significantly outperformed the hill climbing and the tabu search approach. Further, many ALNS approaches found in the literature rely on SA as an acceptance criterion. Examples can be found in [11, 98, 24, 102, 101, 103]. Therefore, in the future course of this work, only the simulated annealing approach considering the start temperature control mechanism is used.

Hill climbing:
One of the simplest acceptance criteria considered in this work is denoted as hill climbing (see e.g. [107]). It defines, that the solution generated in the current iteration is accepted as the start solution of the next iteration if and only if the new solution is the best solution found so far. This procedure implicitly comes at the risk of getting stuck in local optima in non-convex optimization problems. Compared to the SA acceptance criterion, hill climbing performed significantly worse in preliminary experiments. Therefore, the hill climbing acceptance criterion is not pursued any further in this work.

Tabu search:
The tabu search acceptance criterion (see [107, 108]) aims at diversification of the search process rather than directly considering the objective function values of new solutions. With this philosophy, tabu search is the counterpart of the hill climbing approach. Each time a new solution candidate is found it is checked whether the new solution has already been visited in the search process before. This is usually done by storing the hash values of each solution in the so-called tabu list. If the hash value of a new solution is not found in the tabu list, then the new solution is used as a starting point for the next iteration and its hash is inserted into the tabu list. Otherwise, the new solution is rejected and the search process starts again with the previous solution as the start solution. To not block the search for good solutions in potentially profitable areas of the solution space, the tabu list is assigned a maximum length and implements the first-in-first-out (FIFO) principle. This means that solutions that were found early in the solution process and hence blocked for further investigation are allowed to be reentered in the future. Again, compared to the SA acceptance criterion, also tabu search performed significantly worse in preliminary experiments. Hence, the tabu search acceptance criterion is not further pursued in the remainder of this work as well.

3.3.6. Multistart Initial Solution Construction

In this section, we present our algorithm to construct an initial solution. The algorithm is based on the idea of a multistart approach, which is a widely used concept in mathematical optimization to enhance diversification in solution space exploration and to escape local optima (see e.g. [109]). For both, the KTSP and the KOP, the initial solution is constructed by applying Algorithm 4. The only structural difference between applying Algorithm 4 for the KTSP and the KOP is the termination criterion of the insertion process which is explained in detail throughout this section. As an input, the algorithm receives a set of insertion heuristics $\mathcal{I}$ (see line 1). For the KTSP this set is represented by $\mathcal{I} = \{RW_{(10)}, LAFT\}$ and for the KOP it is $\mathcal{I} = \{RW_{(10)}, LAFT, MP, LR\}$. Here, to exploit the exploration capability and computational efficiency of the RW insertion, we append the RW insertion heuristic ten times to $\mathcal{I}$ for both, the KTSP and the KOP. Next, a list is generated in which the constructed solutions are being stored (line 2). Note that a solution is represented as an ordered set of waypoints, each with traversal heading angle and velocity specified. Then, for each insertion heuristic $\lambda_{\mathcal{I}}^{i} \in \mathcal{I}$ a construction process is conducted (see line 3). This process starts by creating an empty solution (see line 4) that serves as a starting point for inserting waypoints. For the KTSP, the empty solution is represented as a list where the first waypoint defined in the problem instance is inserted as the first and last element. Throughout the entire solution construction process for the KTSP, both waypoints are identical in terms of their heading angle and velocity as they represent the same physical waypoint. Note that in general, an arbitrary waypoint can be used as the first/last waypoint since the solution of the KTSP must traverse each waypoint. For the KOP, the start and end waypoints of the associated problem instance are used and inserted into the empty solution with a traversal velocity of 0 m/s. This corresponds to our definition for the KOP where the start and the end waypoint are to be visited at rest (see Section 3.2.2.2). Further, for the start and end waypoint, the heading angle is set to zero since at a velocity of 0 m/s and on an infinitesimal scope, the heading angle does not affect the time-optimal trajectory duration.

In line 5, the construction process is started and proceeds until no further valid insertion can be found. Each iteration of the while loop considers all unvisited waypoints, identifies the insertion decision according to the selected insertion heuristic and inserts the corresponding waypoint with traversal properties as determined by the insertion heuristic into the current solution representation (see lines 6-8). In general, all unvisited waypoints can be inserted between all waypoints of the current solution but not as the first and last one. For the KTSP, the insertion process terminates when the set of unvisited waypoints is empty. For the KOP, the insertion process terminates when either the set of unvisited waypoints is empty, or when the associated insertion heuristic cannot find another insertion that does not violate the maximum flight time budget C_{max}. This behavior is encoded in lines 7, 9, and 10.

For the KTSP as well as the KOP, the waypoints are inserted heuristically with locally optimal traversal properties as defined by the insertion heuristics (see Section 3.3.3). Here, local optimality refers to that the traversal properties of the predecessor and successor are

Algorithm 4: Multistart initial solution construction for KTSP and KOP

```
1  Input: Set of insertion heuristics 𝓘
2  initialSolutionCandidates ← { }
3  for λ^i_𝓘 ∈ 𝓘 do
4      s← Solution.empty( )
5      while true do
6          insertionDecision ← getBestInsertionDecision(s, λ^i_𝓘)
7          if insertionDecision.isValid( ) then
8              s.apply(insertionDecision)
9          else
10             break;
11     s.optimizeTraversalPropertiesByDP( )
12     initialSolutionCandidates.append(s)
13 return: getBestSolution(initialSolutionCandidates);
```

fixed to the associated values before the insertion and the inserted waypoint's traversal properties are optimized with regard to the fixed predecessor and successor using full enumeration. With this in mind, the global dynamic programming optimization approach presented in Section 3.3.4 is applied once to each initial solution candidate right after the insertion process terminates. This step is specified in line 11 of Algorithm 4. For the KOP, this dynamic programming procedure is straightforward, since the traversal properties of the start and end waypoint are defined a-priori and classified as immutable. Therefore, the DP solver presented in Section 3.3.4 yields global optimality of the traversal properties of each waypoint in the given waypoint sequence.

However, this does not hold for the KTSP since all waypoints can be traversed with arbitrary traversal properties including the starting waypoint which, however, must have the same traversal properties as the last waypoint in the current solution representation. Leaving the traversal properties of the first and last waypoint fixed to the value assigned to them at the construction of the empty solution would likely yield a time-suboptimal solution. Therefore, the dynamic programming approach must be performed for any possible combination of traversal heading angles and traversal velocities of the start/end waypoint synchronization. This approach would require $H \cdot V$ runs of the global DP optimization approach which empirically proved to be computationally very expensive. Therefore, for the KTSP and only for the KTSP, we apply the following two-step approach. First, the dynamic programming approach is conducted with the currently specified traversal properties of the first waypoint and last waypoint in the current solution representation. Then, right after this global DP optimization on the current solution, we conduct a second local DP optimization where the traversal properties of the first/last waypoint in the solution and its η predecessors and successors are optimized. This procedure demonstrates to yield very good results and is computationally cheap.

The above-described steps are conducted for all insertion heuristics $\lambda^i_{\mathcal{I}} \in \mathcal{I}$. In the end, the overall best solution found among all used insertion heuristics is classified as the initial

solution and used for further optimization in the subsequent adaptive large neighborhood search (see Sections 3.3.7 and 3.3.8).

3.3.7. Adaptive Large Neighborhood Search for the KTSP

In the following, we present our adaptive large neighborhood (ALNS) approach to solve the KTSP and the KOP which consists of two stages and is applied immediately after the initial solution is constructed (see Section 3.3.6). The first stage of our ALNS aims at exploring the solution space widely to find profitable solutions. This is done by allowing large parts of the solution to be destroyed. In the second stage, the best solution found by the first stage serves as a starting point for a rather local optimization. The second stage is based on the idea that even better solutions could exist close to the best solution found during the first stage.

In general, our ALNS receives a set of insertion heuristics $\mathcal{I}$ and removal heuristics $\mathcal{R}$ as input. Further, a feasible initial solution which is produced by the construction heuristic (see Algorithm 4) and a computation time limit $\mathrm{CT}_{max}(\mathrm{s})$ are given. Since we apply a simulated annealing acceptance criterion, the start temperature is given as an input as well. The value of this parameter is determined based on the objective value of the initial solution as described in more detail in Section 3.3.5.

Next, we focus more deeply on the structure of our two-step approach. The share of computation time for the first step s_g, which corresponds to the global search phase, is derived from the overall time limit CT_{max} and determined via hyperparameter tuning (see Section 3.4.4). Analogously, the share of computation time for the second step is $1 - s_g$. In this two-stage process, the best solution found in the first stage serves as a start solution for the second stage. When the second improvement phase terminates, the overall best solution found is returned. Note that both improvement stages follow the same structure. They only differ in their computation time limit CT_{max} and the allowed range of destruction, i.e. the minimum and maximum number of waypoints to be removed from the current solution during the removal process of a single iteration. For each iteration, we randomly select a real number that represents the concrete share of destruction from this integer range. The destruction shares for the first and second search phases are different and serve as hyperparameters that are described in more detail in Sections 3.4.4.1 and 3.4.4.1.

In the following, we give more insights into the improvement process of each stage. The pseudocode of the improvement phase of the ALNS for the KTSP is given in Algorithm 5.

Each improvement phase is given an argument that distinguishes whether the solver searches for better solutions globally or locally, i.e. if the first or the second stage is executed. Further, the set of allowed insertion and removal heuristics that can be applied is given. Moreover, a feasible start solution and the computation time limit CT_{max} for the entire optimization process of the ALNS are given (see lines 1-6). As the first command, the current system timestamp is stored to evaluate the elapsed computation time of the

Algorithm 5: Improvement phase

```
1  Input:
2     - Search phase p ∈ {"global","local"}
3     - Set of insertion heuristics 𝓘
4     - Set of removal heuristics 𝓡
5     - A feasible initial solution s
6     - Computation time limit CT_max
7  startTime = getCurrentTime( )
8  s_best = s, γ_I = (1, ..., 1), γ_R = (1, ..., 1)
9  stop = false
10 weightUpdateCounter = 0
11 while stop == false do
12     if weightUpdateCounter ≥ weightUpdateCounterMax then
13         Update weights γ_I, γ_R
14         weightUpdateCounter = 0;
15     select removal heuristic λ^i_R ∈ 𝓡
16     select insertion heuristic λ^i_I ∈ 𝓘
17     destructionShare = rand(minShareDestr(p),maxShareDestr(p))
18     s_new = s.removalPhase(λ^i_R, destructionShare).insertionPhase(λ^i_I)
19     if acceptedBySimulatedAnnealing(s_new, s, λ^i_R, λ^i_I) then
20         s_new.optimizeTraversalPropertiesByDP( )
21     if J(s_new) < J(s_best) then
22         s_best = s_new
23     if getCurrentTime() - startTime > CT_max · getComputationTimeShare(p) then
24         stop = true
25     weightUpdateCounter++
26 return: s_best
```

current search phase (line 7). Next, the weights for selecting the insertion heuristics $\gamma_{\mathcal{I}}$ and removal heuristics $\gamma_{\mathcal{R}}$ are initialized with 1 and the best solution found in the process is set to the start solution (line 8). To consider the average performance of the heuristics selected, we further introduce the $weightUpdateCounter$ and initialize it with zero. This parameter specifies the number of iterations that are used for averaging the performance of the individual heuristics (line 10).

The actual process for solution improvement is embedded into the while loop starting in line 11. The while loop continuously iterates until a boolean flag enforces termination when the current iteration exceeds the computation time limit.

The first routine checks if the weightUpdateCounter exceeds the maximum value (line 12). In this case, the accumulated scores that each heuristic collected on average during the last $weightUpdateCounterMax$ iterations are used as new weights for the removal and insertion heuristics and the weightUdateCounter is reset to zero again while the accumulated scores for each heuristic are reset to one (lines 13-14). Following [24, 98, 94,

11], scores can be collected in three possible ways. First, if the pair of removal and insertion heuristics selected for the current iteration finds a new best solution, then it collects a score of σ_1. If the heuristic pair finds a solution that is worse than the best solution, but better than the start solution of the current iteration and additionally has not been found in previous iterations, then the associated removal and insertion heuristic collect a score of σ_2. If the heuristic pair does neither find a new best solution nor a solution that is better than the incumbent solution of the iteration, but the new solution has not been found in previous iterations, then the two heuristics collect a score of σ_3. In all other cases, no scores are collected.

In each iteration, the removal and insertion heuristics are selected randomly based on the current weights of each heuristic (see lines 15-16). The probability $P(\lambda^i_{\mathcal{R}})$ of selecting removal heuristic $\lambda^i_{\mathcal{R}}$ is presented in Equation (3.5) while the probability $P(\lambda^i_{\mathcal{I}})$ of selecting insertion heuristic $\lambda^i_{\mathcal{I}}$ is given in Equation (3.6).

Next, the share of destruction to remove waypoints is selected (line 17). This process is guided by two hyperparameters, namely *minShareDestr* and *maxShareDestr*, which vary depending on the search phase. Both values define an interval from which the share of destruction in the current iteration is selected randomly based on a uniform distribution. Then, depending on the selected destruction share and removal heuristic, a part of the solution is destroyed and fully repaired with the previously selected insertion heuristic (line 18). To improve the quality of each insertion of waypoints immediately after each insertion of the insertion phase, we optimize the traversal properties of all waypoints at the positions $i-\eta, i-\eta+1, ..., i+\eta-1, i+\eta$ of the current solution representation via the local DP optimization approach presented in Section 3.3.4. Here i denotes the position of the newly inserted waypoint. The parameter η is further denoted as the dynamic programming horizon and it represents a hyperparameter that affects the overall performance of the ALNS. More details on the hyperparameter η are given in Section 3.4.4. In this process, the waypoints at the locations $i-\eta-1$ and $i+\eta+1$ are considered as start and end nodes of the dynamic programming procedure and their traversal properties are fixed. Note that this local DP optimization is encoded in the insertion phase in line 18.

When no further insertion is possible, the creation of a new solution is finished and the quality of that solution needs to be evaluated. Hence, in the first step, it is accessed via the simulated annealing acceptance criterion presented in Section 3.3.5 if the solution is accepted and serves as a starting point for the next iteration (see line 19). This step also includes updating the temperature for the next iteration. If the solution is accepted, the solution is classified as good enough to be investigated further and the global DP optimization procedure is applied for the entire sequence of waypoints (see line 20). This includes the post-optimization of the traversal attributes of the start and end waypoint in the current solution representation as described in Section 3.3.6 for the KTSP. Note that we only apply the global DP optimization to accepted solutions since this procedure is computationally too expensive to be applied in each iteration. The objective value of the newly generated solution is evaluated and compared with the current best solution found in the next step (line 21). If with the new solution, a new best solution is found, the best solution is updated (line 22). In any case, the new solution serves as the start

solution for the next iteration. As the last step, the termination criterion is evaluated and the weightUpdateCounter is incremented (see lines 23-25). In this work, termination is enforced when the elapsed time of the current improvement phase exceeds the maximum computation time of the associated search phase p. At termination, the algorithm returns the best solution found.

3.3.8. Adaptive Large Neighborhood Search for the KOP

The ALNS for the KOP works almost identically to the ALNS procedure for the KTSP, which is presented in the previous section. The major difference is that in the KOP case, the objective is to maximize the collected priorities while a maximum travel time budget must be considered. Hence the insertion procedure either ends when all unvisited waypoints are inserted into the solution, or when the maximum flight time budget cannot be held for any further insertion. Further, the KOP utilizes a different set of insertion and removal heuristics as described in Sections 3.3.3 and 3.3.2. Lastly, the traversal properties of the start and end waypoint are not post-optimized after the global DP procedure since they are fixed to specific values a-priori as given in the associated problem instance.

3.4. Computational Study

In this section, we present the results of the computational study on our approaches to solving inertia-based routing problems. We first present the general computational study settings which are applied as default if not specified differently. Next, we shortly introduce the problem instances, that we use within this work and present the performance indicators we use to assess our proposed solution approaches.

With these general settings, we present the results of the hyperparameter optimization of our ALNS approaches to solve the KTSP and the KOP. This is followed by an investigation of our proposed approaches by benchmarking them against current solution approaches from the literature. Here, we focus on the computational complexity, solution quality and applicability to real-world problems.

3.4.1. General Computational Study Setup

All our experiments were run on an Intel(R) Core(TM) i7-8565U CPU with 16 GB of RAM. MIP-based solutions are obtained by utilizing Gurobi 10.0.1 as a commercial general-purpose solver implemented in Python 3.9 and assigned a computation time limit of 18000 s if not specified differently. Our ALNS solvers for the KTSP and KOP are implemented in C++17 as a single-threaded executable. Both of our ALNS solvers are parameterized using the best hyperparameters identified in our hyperparameter optimization which we present in Section 3.4.4. As default, the ALNS approaches are assigned a maximum computation time of either 30 s, 60 s , or 120 s.

To determine the edge costs for our KTSP and KOP models, i.e. to determine the travel times between each waypoint pair with specified traversal heading angle and velocity, we use the two-dimensional version TOP-UAV++ as default. If not specified differently, our TOP-UAV++ trajectory planner generates trajectories with a maximum allowed velocity of 3 m/s and a maximum allowed acceleration of 1.5 m/s^2. These kinematic properties are realistic values for surveillance and data collection applications using multirotor UAVs for flights at lower altitudes (see e.g. [22, 37, 36]), as required for example in damage detection of buildings. Further, if not specified differently, we set the number of traversal heading angles $H = 8$ and the number of traversal velocities $V = 6$. As we derive in Section 3.4.5.1, this combination offers a good trade-off between solution quality and computational complexity.

3.4.2. Problem Instances

Here, we present the problem instances we use to evaluate our solvers for the KTSP and the KOP and to compare them with solutions yielded by existing approaches from the literature. As described in the following, we introduced problem instances for solution quality benchmarking, hyperparameter optimization and computation time evaluation for both, the KTSP and the KOP.

3.4.2.1. Problem Instances for Benchmarking - TSP Variants

As benchmark instances for considered variants of the TSP, we make use of the problem instances introduced in [110], further denoted as Tsiligirides dataset 1, 2, and 3. These instances can be found online under https://www.mech.kuleuven.be/en/cib/op#section-0. Originally, these datasets are designed for the orienteering problem. To be able to evaluate the performance of our KTSP model and our associated solvers, we modified these instances by relaxing the maximum travel budget and neglecting the priorities assigned to each waypoint. By this modification, each dataset is reduced to a single problem instance. On these three problem instances, we apply spatial scaling factors from the set $\{0.25, 0.5, 1.0, 2.0, 4.0\}$ to each coordinate of each waypoint to these instances. The scaling is applied to evaluate how well our KTSP model performs compared to SOTA models, such as the DTSP, when the distances between waypoints get larger / lower while the kinematic properties remain constant. We provide these problem instances as open source on GitHub under [111].

In Table 3.3, we give an overview of the properties of all problem instances used for the solution quality benchmarking purpose of the considered TSP variants. As can be seen for the scaled Tsiligirides 1 problem instances, the area that contains all waypoints ranges from 3.275 m · 5.35 m to 54.4 m · 85.6 m. The associated area for the Tsiligirides 2 problem instances is slightly smaller for the corresponding scaling factor while the area of the associated Tsiligirides 3 problem instances is slightly larger.

Basic instance	L	Scaling factor	Area
Tsiligirides 1	32	0.25	3.275 m · 5.350 m
		0.5	6.550 m · 10.700 m
		1.0	13.100 m · 21.400 m
		2.0	26.200 m · 42.800 m
		4.0	52.400 m · 85.600 m
Tsiligirides 2	21	0.25	2.825 m · 3.300 m
		0.5	5.650 m · 6.600 m
		1.0	11.300 m · 13.200 m
		2.0	22.600 m · 26.400 m
		4.0	45.200 m · 52.800 m
Tsiligirides 3	33	0.25	3.775 m · 7.125 m
		0.5	7.550 m · 14.250 m
		1.0	15.100 m · 28.500 m
		2.0	30.200 m · 57.000 m
		4.0	60.400 m · 114.000 m

Table 3.3.: Benchmark problem instances for TSP-variants as we provide them on GitHub under [111]. Here, the name of the basic problem instances is given as well as the number of waypoints L, the scaling factor and the area size in which the waypoints are located.

3.4.2.2. Problem Instances for Hyperparameter Optimization - KTSP

To conduct the hyperparameter optimization for our ALNS to solve the KTSP, we make use of six different problem instances. These problem instances differ in the number of waypoints they contain. Overall, the number of waypoints of these problem instances is given by the set $\{10, 15, 20, 25, 30, 35\}$. For each problem instance, the x, y coordinates of the waypoints are randomly sampled from a uniform distribution in the interval $[0, 20]\,(\mathrm{m})$. The interval length of 20 m is selected since it approximately represents the spatial scale of the problem instances described in Section 3.4.2.1 with a spatial scaling factor of 1.0. We provide all problem instances for our hyperparameter optimization as open source on GitHub under [112].

3.4.2.3. Problem Instances for Computation Time Evaluation - TSP Variants

To evaluate the required computation time of our MIP-based approach for the KTSP and compare it with the corresponding approach for the DTSP, we solve 80 different problem instances using each approach. The number of waypoints L in each problem instance ranges from ten to 40 with steps of two additional waypoints in between. For each number of waypoints L, we generate five different problem instances where for each problem instance the x, y coordinates of L waypoints are randomly sampled from a uniform distribution in the interval $[0, 20]\,(\mathrm{m})$. Again, the interval length of 20 m is

selected to enable a comparison with the problem instances which we describe in Section 3.4.2.1. We provide these problem instances as open source on GitHub under [113].

3.4.2.4. Problem Instances for Benchmarking - OP Variants

Analogously to the benchmark instances for the TSP variants, for the OP variants, we make use of the problem instances introduced in [110], which can be found online under https://www.mech.kuleuven.be/en/cib/op#section-0. The area in which each waypoint of these problem instances is located corresponds to the area given in Table 3.3. For Tsiligirides dataset 1 the priorities assigned to each waypoint excluding the start and the end waypoint range from five to 15. For datasets 2 and 3 the priorities range from ten to 50. For all datasets, the start and end waypoints are assigned a priority of zero.

However, the problem instances of the original Tsiligirides datasets are hard to solve to optimality by our MIP-based approach within the given computation time limit. Therefore, we additionally modify the waypoints set of Tsiligirides dataset 2 by only using the first 15 waypoints. The first waypoint represents the start and the 15-th waypoint is the final waypoint. To evaluate the impact of changing distances between the prioritized waypoints on our solution approaches and the ones from the SOTA, we apply spatial scaling factors from the set $\{0.25, 0.5, 1.0, 2.0, 4.0\}$ to each coordinate of each waypoint of this reduced waypoint set. We provide the scaled versions of the waypoint set derived from Tsiligirides dataset 2 as open source on GitHub under [114]. The maximum flight time constraint that complements these waypoint sets to an OP problem instance is described in the corresponding parts of this work. The area in which each waypoint is located is given in Table 3.4 with the corresponding spatial scaling factors. The priorities for the problem instances of the reduced Tsiligirides dataset 2 range from ten to 40 except for the start and end waypoints which are assigned a priority of zero.

Basic instance	L	Scaling factor	Area
Tsiligirides 2 - reduced	15	0.25	2.125 m · 3.300 m
		0.5	5.500 m · 4.250 m
		1.0	11.000 m · 8.500 m
		2.0	22.000 m · 17.000 m
		4.0	44.000 m · 34.000 m

Table 3.4.: Reduced Tsiligirides dataset 2 problem instances for OP-variants as we provide them on GitHub under [114].

3.4.2.5. Problem Instances for Hyperparameter Optimization - KOP

To optimize the hyperparameters for our ALNS to solve the KOP, we design and use six ordered sets of prioritized waypoints which we provide on GitHub under [115]. The

maximum travel time budget that complements these ordered sets of prioritized waypoints to actual problem instances is given at the corresponding parts in the computational study. These described waypoint sets differ in the number of waypoints they contain and which is an element of the set $\{10, 15, 20, 25, 30, 35\}$. The x, y coordinates of each waypoint of each ordered set of waypoints are sampled randomly in the interval $[0, 20]$(m) using a uniform distribution. Each waypoint is assigned a random integer from the interval $[1, 35]$ as priority except for the start and end waypoints which are assigned a priority of zero. Analogously to the KTSP case, the interval length of 20 m is selected it approximately represents the spatial scale of the problem instances described in Section 3.4.2.1 with a spatial scaling factor of 1.0.

3.4.2.6. Problem Instances for Computation Time Evaluation - OP Variants

Our evaluation of the required computation time of our MIP-based approach for the KOP in comparison with our MIP-based SOTA approaches is based on a total of 25 ordered sets of prioritized waypoints. Again, for these sets, the maximum travel time budget is specified at the corresponding parts of the computational study.

The number of waypoints in each waypoint set ranges from five to 25 with steps of five additional waypoints in between. In this setup, five waypoint sets always contain the same number of waypoints. The x, y coordinates of each waypoint L in each ordered set of prioritized waypoints are randomly sampled in the interval $[0, 20]$(m) using a uniform distribution. Further, each waypoint is assigned a random integer from the interval $[1, 9]$ as priority except for the start and end waypoints which are assigned a priority of zero. We provide these waypoint sets as open source on GitHub under [116].

3.4.3. Performance Indicators

In the following, we describe the indicators to assess the performance of all approaches used in this work as well as the approaches from the SOTA. The performance indicators that apply to both, the TSP variants including the KTSP and the OP variants including the KOP, are summarized in Table 3.5. The problem-specific performance indicators are presented in Sections 3.4.3.1 and 3.4.3.2.

The first performance indicator to be introduced is the MAE (mean absolute error). It measures the averaged tracking error between the reference trajectory and the trajectory obtained when tracking the reference by the MPC described in Appendix B.1. This MPC is, on the one hand, assigned a time horizon of $N = 10$ steps with a time step length of $\Delta T = 0.1$ s. Further, the maximum velocity and acceleration are considered as coupled for the x and y axis and are constrained by the associated maximum norms of $\hat{v}_{max}$ and $\hat{a}_{max}$. As usual for MPC-based trajectory tracking, the trajectory tracking process of our MPC utilizes the moving horizon principle by only applying the acceleration of the very first time step of the optimized solution as control input. Then the kinematic equations are evaluated

based on this acceleration and the process of determining the optimal acceleration for the next time step starts again. For our MAE performance indicator holds

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}\sqrt{(p_{x,i}^{ref} - p_{x,i}^{mpc})^2 + (p_{y,i}^{ref} - p_{y,i}^{mpc})^2}, \tag{3.14}$$

where $p_{x,i}^{ref}, p_{y,i}^{ref} \in \mathbb{R}$ describes the reference position trajectory yielded by one of the used planning approaches sampled at time $i \cdot \Delta T$. Here, ΔT represents the discretized time step length as explained in Appendix B.1. Further, $p_{x,i}^{mpc}$ and $p_{y,i}^{mpc} \in \mathbb{R}$ describe the corresponding simulated position at the i-th time step of the MPC trying to track the reference. Last, n describes the number of time steps for the entire flight mission.

To assess the yielded solution quality of our MIP-based approaches to solve the DTSP, KTSP, DOP, and KOP, we introduce the performance indicator Gap^{MIP} as given in Table 3.5. This performance indicator describes the relative gap between the primal bound (PB), which represents the best feasible solution found so far, and the dual bound (DB), which represents the currently best bound on how good the global optimal solution can get. The braces in the corresponding row in Table 3.5 describe that this equation holds for both, the KTSP and the KOP.

As stated in Section 3.4.1, all MIP-based approaches are given a maximum computation time limit of 18000 s as default. However, this time limit is not always entirely used, since an optimal solution can be found in less computation time. To evaluate the computation time consumption of the MIP-based approaches, we determine for lowest required computation time for solving a set of closely related problem instances as CT_{LB} and the highest required computation time as CT_{UB}. The average computation time is denoted as CT. Note in case only a single problem instance is solved using a MIP-based approach, the required computation time is also denoted as CT.

KPI	Unit	Description
MAE	m	Mean absolute error of the tracking
Gap^{MIP}	%	Relative MIP gap $\lvert\{\}_{DB} - \{\}_{PB}\rvert / \{\}_{PB}$
CT_{LB}	s	Lowest computation time
CT	s	Single or averaged computation time
CT_{UB}	s	Highest computation time

Table 3.5.: General performance indicators.

3.4.3.1. Performance Indicators for TSP-related Problems

To assess the performance of the approaches used to solve TSP variants within this work, we introduce a set of specifically tailored performance indicators which are summarized

in Table 3.6. The main indicator is $\mathcal{M}_T$, which describes the minimum mission duration found.

Further, to assess the performance of our ALNS approach in terms of yielded solution quality specifically for the KTSP, we define a set of performance indicators that follow up with the idea of a relative gap. We define the performance indicator Gap^{ALNS} as the relative gap between the solution yielded by our ALNS approach and the best solution yielded by the corresponding MIP-based approach within a computation time limit of five hours. Accordingly, $\text{Gap}_{\text{UB}}^{\text{ALNS}}$ describes the highest relative gap between the solutions yielded by our ALNS approach and the corresponding MIP-based approach over multiple problem instances. The performance indicator $\text{Gap}_{\text{LB}}^{\text{ALNS}}$ describes the lowest relative gap. In cases where the ALNS yields a solution with a lower mission duration than yielded by the MIP-based approach due to the computation time limit of the MIP-based approach, the relative gap takes a negative value.

Moreover, we use the performance indicator $\text{Gap}_{\text{SOTA}}^{\text{MIP}}$ to evaluate the performance of a particular SOTA approach compared to the best solution found by our MIP-based approach within a computation time limit of five hours.

KPI	Unit	Description
$\mathcal{M}_T$	s	Required mission duration
$\text{Gap}_{\text{LB}}^{\text{ALNS}}$	%	Lowest relative gap $(\text{KTSP}_{\text{ALNS}} - \text{KTSP}_{\text{PB}})/\text{KTSP}_{\text{ALNS}}$
Gap^{ALNS}	%	Relative gap $(\text{KTSP}_{\text{ALNS}} - \text{KTSP}_{\text{PB}})/\text{KTSP}_{\text{ALNS}}$
$\text{Gap}_{\text{UB}}^{\text{ALNS}}$	%	Highest relative gap $(\text{KTSP}_{\text{ALNS}} - \text{KTSP}_{\text{PB}})/\text{KTSP}_{\text{ALNS}}$
$\text{Gap}_{\text{SOTA}}^{\text{MIP}}$	%	Relative gap $(\text{SOTA} - \text{KTSP}_{\text{PB}})/\text{KTSP}_{\text{PB}}$

Table 3.6.: Performance indicators for TSP-related problems.

3.4.3.2. Performance Indicators for OP-related Problems

The performance indicators specifically tailored for variants of the OP are summarized in Table 3.7 The main performance indicator considered in this work is $\mathcal{M}_P$. It measures the sum of all priorities collected by visiting the associated waypoints within the given travel time budget.

Moreover, in analogy to Section 3.4.3.1, we define Gap^{ALNS} as the relative gap between the total collected priorities yielded by our ALNS approach and the corresponding MIP-based approach with a computation time limit of five hours. In case multiple problem instances are considered, $\text{Gap}_{\text{LB}}^{\text{ALNS}}$ represents the lowest and $\text{Gap}_{\text{UB}}^{\text{ALNS}}$ the highest relative gap. If the solution yielded by our ALNS collects more priorities within the given travel time budget than the solution yielded by our MIP-based approach, the relative gap takes a negative value. Note that the nomenclature of Gap^{ALNS}, $\text{Gap}_{\text{LB}}^{\text{ALNS}}$, and $\text{Gap}_{\text{UB}}^{\text{ALNS}}$ for the OP variants

is identical to the one for the TSP variant. We point out to the reader that these identifiers must be interpreted problem-specifically for the rest of the work.

The performance index $\text{Gap}^{\text{MIP}}_{\text{SOTA}}$ is not specified for the KOP since there are no associated experiments conducted.

KPI	Unit	Description
$\mathcal{M}_P$	-	Accumulation of collected priorities
$\text{Gap}^{\text{ALNS}}_{\text{LB}}$	%	Lowest relative gap $(\text{KOP}_{\text{PB}} - \text{KOP}_{ALNS})/\text{KOP}_{ALNS}$
Gap^{ALNS}	%	Relative gap $(\text{KOP}_{\text{PB}} - \text{KOP}_{ALNS})/\text{KOP}_{ALNS}$
$\text{Gap}^{\text{ALNS}}_{\text{UB}}$	%	Highest relative gap $(\text{KOP}_{\text{PB}} - \text{KOP}_{ALNS})/\text{KOP}_{ALNS}$

Table 3.7.: Performance indicators for OP-related problems.

3.4.4. Hyperparameter Optimization

In this section, we present the optimization of the hyperparameters associated with our ALNS solvers to solve the KTSP and the KOP. For the sake of comparability, we discretize all hyperparameters with different discretization depths, as we specify in the following. In general, the discretization of hyperparameters creates a hyperdimensional grid of possible hyperparameter combinations. As we describe later on, there are intractably many combinations and it would be computationally too expensive to consider all of them for the hyperparameter optimization. However, in contrast to enlarging the discretization steps of the hyperparameter grid to reduce the set of hyperparameter combinations, we follow a random search approach on the associated grid. Random search is a simple but effective approach for hyperparameter optimization and '[...] often has much better performance than full grid search [...]' (see [117]). This is explained in Figure 3.6, which refers to a figure we extracted from [118]. Under the assumption that some parameters are more important than others to achieve high solution quality, random search allows sampling of these important parameters more densely.

In the following, we give detailed insights into the hyperparameter optimization of the ALNS solution frameworks for the KTSP and the KOP.

3.4.4.1. Hyperparameter Optimization for the KTSP

Our ALNS to solve the KTSP depends on a set of tunable hyperparameters affecting the performance. These hyperparameters can be divided into two groups. The first group contains three parameters σ_1, σ_2, and σ_3 that are directly related to the general ALNS solution framework and that specify its adaptiveness as described in the following enumeration.

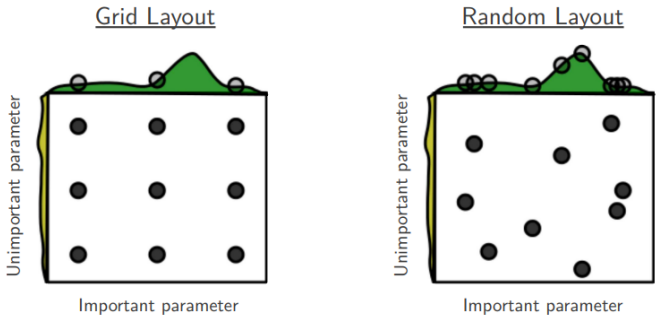


Figure 3.6.: Comparison of grid search and random search for hyperparameter optimization as given in [118]. Random search allows the sampling of the important parameters more densely.

- σ_1: Score update of a removal/insertion pair of heuristics when it contributed to finding a new best solution.
- σ_2: Score update of a removal/insertion pair of heuristics when it contributed to finding a solution that has not been accepted before and is better than the incumbent solution.
- σ_3: Score update of a removal/insertion pair of heuristics when it contributed to finding a solution that has not been accepted before but is worse than the incumbent solution.

For these hyperparameters, there exist values that are widely used in the literature. We set the values of these parameters to $\sigma_1 = 33, \sigma_2 = 9, \sigma_3 = 13$, as they are defined in [24] and demonstrated to yield solutions of high quality in a variaty of applications (see [98, 11, 119]).

Apart from these general ALNS hyperparameters, there is also a set of hyperparameters that are introduced specifically for this work and for which no proper values can be derived from the literature. These parameters are described in more detail in the following.

- **Share of global search s_g:** As described in Section 3.3.7, we utilize a two-step search, for which in the first step the destruction of large parts of the solution is allowed to support global solution space exploration. In the second step, only a smaller part of the solution can be modified to locally optimize the best solution found in the first step. The parameter that specifies the share of global exploration in the search process is defined by $s_g \in \{0.0, 0.1, ..., 1.0\}$. The share of the local exploration procedure is $1 - s_g$. Note that the value of s_g relates to the maximum computation time limit of the ALNS solver. Hence, $s_g = 0.0$ represents, that during the entire computation time of the ALNS, improvements of the current solution are searched only locally. Vice versa, for $s_g = 1.0$ the improvements are searched entirely globally.
- **Minimum destruction share for global improvement phase δ_g^{min}:**
  In the first step, a search for good solutions is conducted on a rather global scale. This is achieved by destroying large parts of the solution. The hyperparameter

δ_g^{min} defines the minimum share of destruction. This means that at least $\lceil \delta_g^{min} \cdot L \rceil$ waypoints must be removed from the solution. Here, L again describes the number of waypoints in the problem instance. For the hyperparameter optimization conducted in this work, we define $\delta_g^{min} \in \{0.1, 0.2, \ldots, 0.5\}$.

- **Maximum destruction share for global improvement phase δ_g^{max}:**
  The parameter δ_g^{max} represents the counterpart of δ_g^{min}. It defines the maximum share of destruction. For the KTSP it holds that the maximum number of removed waypoints from a solution equals $\lceil \delta_g^{max} \cdot L \rceil$. Further, $\delta_g^{max} \in \{0.5, 0.6, \ldots, 0.9\}$ holds.

- **Minimum destruction share for local improvement phase δ_l^{min}:**
  In the second optimization phase, the neighborhood around the best solution found in the first phase is explored locally. Hence, only small parts of the solution are allowed to be destroyed. The minimum value of destruction is defined by δ_l^{min}. Analogously to the global search phase, the minimum number of waypoints to remove is set to $\lceil \delta_l^{min} \cdot L \rceil$. Allowed values for δ_l^{min} are defined by $\{0.05, 0.10, 0.15, 0.20\}$.

- **Maximum destruction share for local improvement phase δ_l^{max}:**
  Again δ_l^{max} represents the counterpart to δ_l^{min}. It defines the maximum share of destruction for the local improvement phase. Hence, the maximum number of waypoints to remove during the second phase is set to $\lceil \delta_l^{max} \cdot L \rceil$. Allowed values for δ_l^{max} are defined by $\{0.20, 0.25, 0.30, 0.35\}$.

- **Start temperature control parameter ω:**
  The start temperature control parameter ω defines the start temperature of the simulated annealing process (see Section 3.3.5). For our solver, it specifies that in the very first iteration a new solution which is ω times worse than the initial solution is accepted with a probability of $P_A = 50\%$. For the hyperparameter optimization of this work, we define $\omega \in \{0.00, 0.05, \ldots, 0.35, 0.4\}$.

- **Temperature decrease parameter ϑ:**
  The temperature decrease parameter defines the factor applied to each iteration to decrease the temperature for the next iteration in the simulated annealing procedure. With a lower temperature comes a lower probability of accepting bad solutions (see Section 3.3.5). As a hyperparameter, we enable ϑ to be selected among $\{0.99000, 0.99005, \ldots, 0.99995, 1\}$, whereas $\vartheta = 1$ would mean that the temperature remains constant during the entire search process.

- **Horizon of dynamic programming traversal property optimization η:**
  When inserting a waypoint into an existing solution, not only the inserted waypoint's traversal property must be optimized but also optimizing the traversal properties of the inserted waypoint's neighbors must be considered. We define $\eta = 1$ to optimize the traversal properties of the inserted waypoint as well as its immediate predecessor and successor while the enclosing waypoints' traversal properties are fixed. A horizon of $\eta = 2$ represents that the two predecessors and two successors are free for optimization of their traversal properties. The optimization is realized via the dynamic programming approach as shown in Section 3.3.4. An example for $\eta = 1$

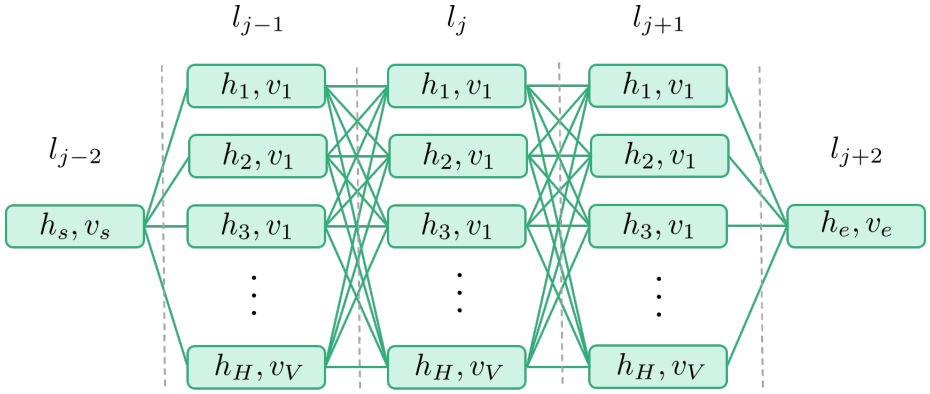


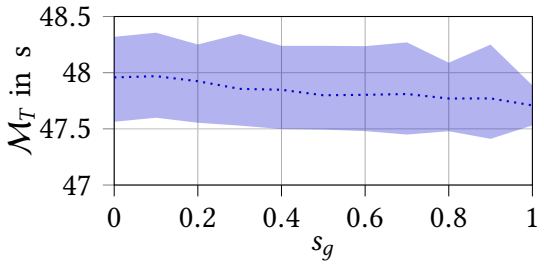
Figure 3.7.: Dynamic programming traversal property optimization graph for $\eta = 1$ when waypoint l_j is inserted.

is given in Figure 3.7. If $\eta = 0$ holds, only the traversal property of the inserted waypoint is optimized.

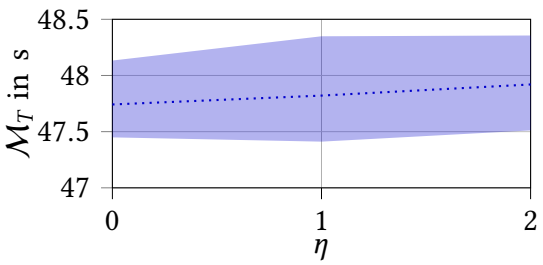
For hyperparameter optimization of this work, we allow $\eta \in \{0, 1, 2\}$. Note that higher values for η might represent better insertion qualities, however, since the dynamic programming is executed for each insertion, the computational effort gets higher. Consequently, the number of iterations performed within a given computation time limit for the entire ALNS solver gets lower which reduces the exploration of the solution space. Hence, the purpose of considering η in the hyperparameter optimization is to find a suitable trade-off between insertion quality and exploration.

The full grid of possible hyperparameter configurations with the above-specified discretization consists of $\approx 23.88 \cdot 10^6$ combinations, which is too many for a full grid search. Therefore, as described in Section 3.4.4, we make use of random search and randomly sample 500 different hyperparameter combinations from the above-described hyperparameter domains. For each hyperparameter configuration, we solve each of the six problem instances given in Section 3.4.2.2 three times with different random seeds. For each hyperparameter configuration, we aggregate the worst objective function value obtained for each problem instance over the three runs and define the hyperparameter configuration with the best worst-case performance for all problem instances as the best hyperparameter configuration found. We apply the best worst-case behavior to obtain a constantly well-performing solver for a wide range of problem instances.

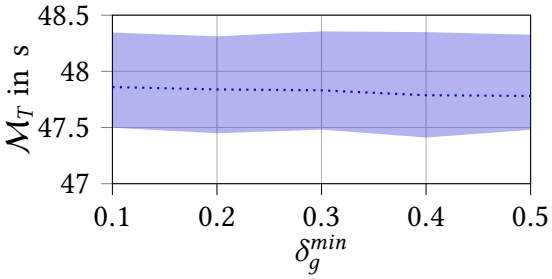
The computation time limit for each run of our ALNS-solver specified by the associated hyperparameters is set to $\mathrm{CT}_{max} = 30$ s. We choose this computation time limit for two reasons: First, in real-world field applications, UAV mission planning solutions must be obtained in a short time. Second, we needed to find a trade-off between the invested computation time for the hyperparameter optimization and the density of the hyperparameter samples in the hyperparameter space. The overall computation time invested for hyperparameter optimization of our ALNS for the KTSP was 270,000 s $\approx$ 3.1 days.



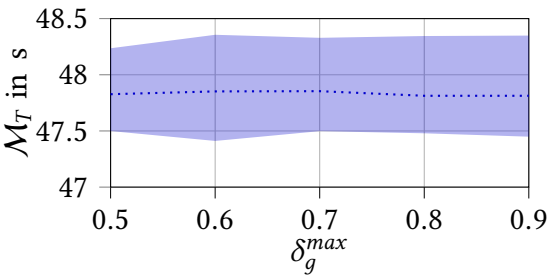
(a) Effect of the share of global search.



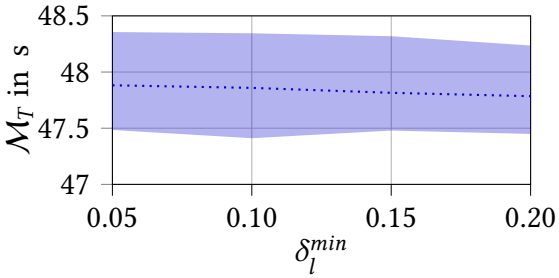
(b) Effect of the horizon of DP.



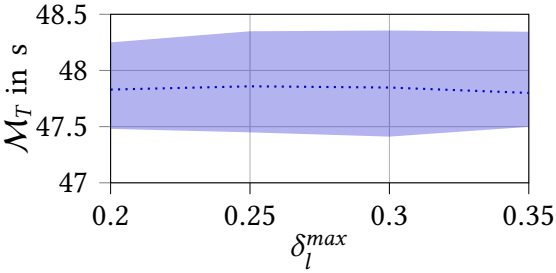
(c) Effect of the minimum destruction share for searching globally.



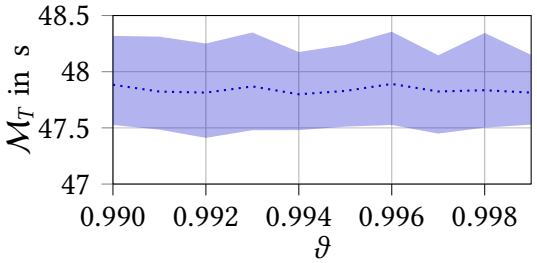
(d) Effect of the maximum destruction share for searching globally.



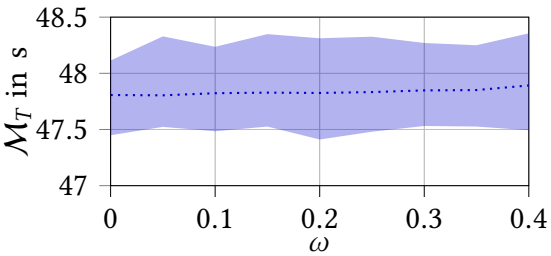
(e) Effect of the minimum destruction share for searching locally.



(f) Effect of the maximum destruction share for searching locally.



(g) Effect of the parameter describing the temperature decrease.



(h) Effect of the start temperature control parameter.

Figure 3.8.: Effect of the hyperparameters on the performance of the ALNS for the KTSP.

Parameter	s_g	η	δ_g^{min}	δ_g^{max}	δ_l^{min}	δ_l^{max}	ϑ	ω
Optimized values	0.9	1	0.4	0.6	0.1	0.3	0.99251	0.2

Table 3.8.: Best hyperparameter configuration found for the KTSP.

In Figures 3.8, we show the effect of each hyperparameter on the performance of the ALNS for the KTSP. The bounds of the light blue shape represent the minimum and maximum mission duration $\mathcal{M}_T$ yielded for the hyperparameter configurations with the corresponding value averaged over all problem instances and runs. The dotted blue represents the average $\mathcal{M}_T$ for the associated hyperparameter value averaged over all problem instances and runs. The most significant hyperparameters are given as follows. As can be seen in Figure 3.8a, the higher the share of the global search phase s_g the lower the $\mathcal{M}_T$. Moreover, the lower the dynamic programming horizon, the lower the average $\mathcal{M}_T$ (see 3.8b). However, the least values for $\mathcal{M}_T$ is achived for $\eta = 1$. A less significant but also notable effect is obtained for the minimum destruction shares of the global and local searching process (see Figures 3.8c and 3.8e). The higher the share the lower the average required mission duration $\mathcal{M}_T$. The remaining parameters seem to have no notable effect on the ALNS performance. However, their composition is likely to be important, e.g. since the temperature decrease parameter ϑ and the start temperature control parameter ω are closely coupled.

Among all configurations, the hyperparameter configuration yielding the best worst-case performance is given in Table 3.8. It can be seen that in alignment with Figure 3.8a a high share of global optimization is favored since it allows an efficient search for good solutions on a global scale. Then, only a small part of the total computation time is required to locally optimize the best solution found so far. The next interesting result relates to the horizon of the dynamic programming optimization applied after each insertion. Although a smaller dynamic programming horizon yields on average better results, the horizon belonging to the best worst-case hyperparameter configuration is $\eta = 1$. Consequently, the optimization of the traversal properties of the direct neighbors of an inserted waypoint is favored since it seems to yield the best trade-off between investment of computational resources and quality of insertion.

3.4.4.2. Hyperparameter Optimization for the KOP

Since the KOP and the KTSP are closely related problems and since the ALNS algorithm of the KOP is almost identical to the one for the KTSP, their set of hyperparameters is equal including the domain of feasible values for each hyperparameter. For the same reason as in the KTSP case, the values of the weight update parameters are set to $\sigma_1 = 33, \sigma_2 = 9, \sigma_3 = 13$ as these values showed good performance in various studies in the literature (see [98, 11, 119]).

Analogously to the optimization of the hyperparameters for the KTSP, we randomly sampled a total of 500 different hyperparameter configurations for the KOP that were evaluated on the 24 hyperparameter optimization instances described in Section 3.4.2.5.

Again, each problem instance is solved by each solver configuration three times with different random seeds. The computation time limit of each solver is set to 30 s. Again, this computation time limit is chosen for two reasons: First, in real-world field applications, UAV mission planning solutions must be obtained in a short time. Second, a trade-off between the invested computation time for the hyperparameter optimization and the density of the hyperparameter samples in the hyperparameter space is to be defined. The overall computation time invested for hyperparameter optimization of our ALNS for the KOP was 1,080,000 s $\approx$ 12.5 days.

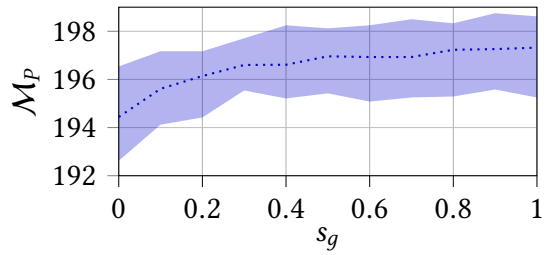
In contrast to the KTSP case, the hyperparameters related to the share of destruction, i.e. δ_g^{min}, δ_g^{max}, δ_l^{min}, and δ_l^{max}, are not related to the number L of all waypoints given in the problem instance. Instead, they are related to the number of waypoints visited in the start solution of the improvement current iteration. With this approach, the removal phase is also suited to cover problem instances in which only a fraction of the number of waypoints can be visited in the given maximum flight time budget.

In Figures 3.9, we show the effect of each hyperparameter on the performance of the ALNS for the KOP. Analogously, the bounds of the light blue shape represent the minimum and maximum collected priorities $\mathcal{M}_P$ yielded for the hyperparameter configurations with the corresponding value averaged over all problem instances and runs. The dotted blue represents the average $\mathcal{M}_P$ for the associated hyperparameter value averaged over all problem instances and runs. The most effective hyperparameter is s_g. The higher its value the higher the minimum, maximum, and average collected priorities, as can be seen in Figure 3.9a. Moreover, the hyperparameters η, δ_g^{min}, and δ_g^{max} are effective in terms of the average obtained solution quality as well. Interestingly and in contrast to the KTSP case, the higher the value of η the better the average obtained collected priorities.

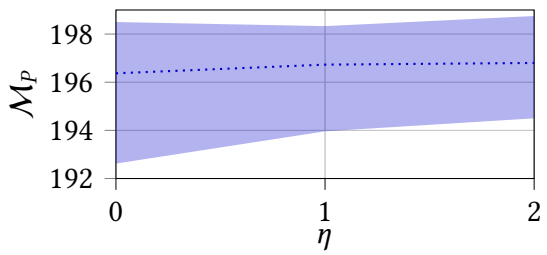
This can be explained since higher values for η correspond to more time-efficient motions between a given sequence of waypoints. The resulting flight time savings can be exploited by e.g. inserting a few more waypoints with potentially higher priorities into the tour without violating the maximum flight time constraint.

The remaining parameters seem to have no directly notable effect on the ALNS performance. However, analogously the hyperparameter of the ALNS for the KTSP, the composition of these hyperparameters is likely to be important, e.g. since the temperature decrease parameter ϑ and the start temperature control parameter ω are closely coupled.

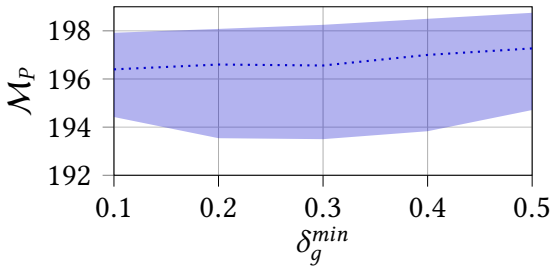
To assess the performance of each configuration, we again consider the best worst-case performance, i.e. for each hyperparameter configuration, the worst maximum priorities obtained among each run of each problem instance are accumulated and finally divided by the number of problem instances. The hyperparameter configuration associated with the highest resulting collected priorities is defined as best. The values of the optimized hyperparameters are given in Table 3.9. Interestingly, the values are similar to the ones obtained for the KTSP.



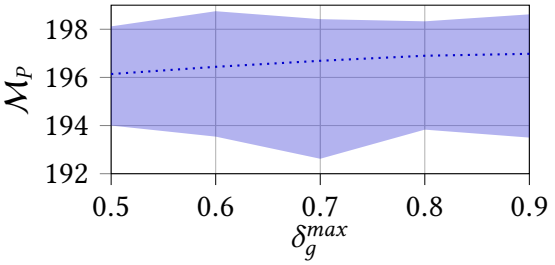
(a) Effect of the share of global search.



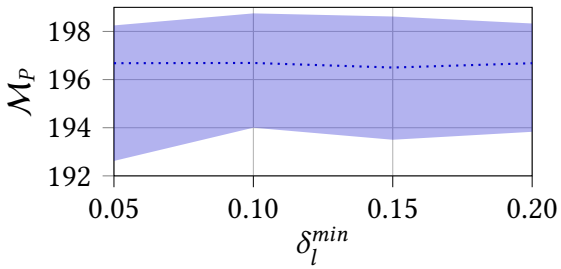
(b) Effect of the horizon of DP.



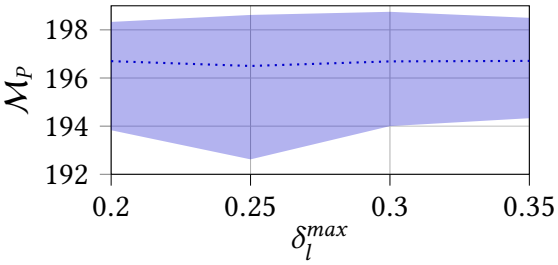
(c) Effect of the minimum destruction for searching globally.



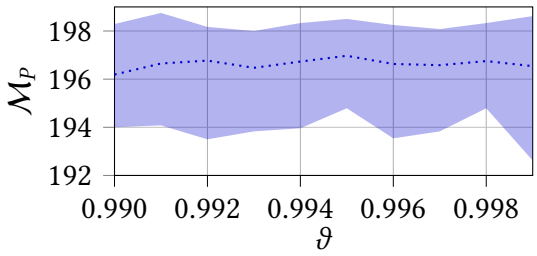
(d) Effect of the maximum destruction for searching globally.



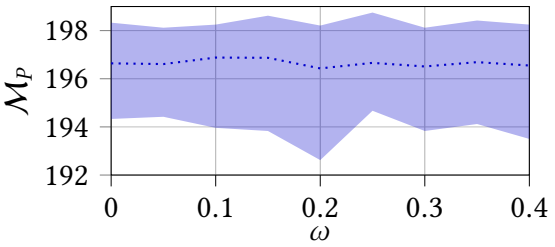
(e) Effect of the minimum destruction for searching locally.



(f) Effect of the maximum destruction for searching locally.



(g) Effect of the parameter describing the temperature decrease.



(h) Effect of the start temperature control parameter.

Figure 3.9.: Effect of the hyperparameters on the performance of the ALNS for the KOP.

Parameter	s_g	η	δ_g^{min}	δ_g^{max}	δ_l^{min}	δ_l^{max}	ϑ	ω
Optimized values	0.9	2	0.5	0.6	0.1	0.3	0.99109	0.25

Table 3.9.: Best hyperparameter configuration found for the KOP.

3.4.5. Computational Results for the KTSP

In this section, we present the computational results for the KTSP and its comparison to TSP variants from the literature. This contains an evaluation of the best trade-off between the number of traversal velocities V and the computational complexity. Further, we evaluate the benefit of using the KTSP model compared to models from the state-of-the-art including an evaluation of the required computation time for increasing problem sizes. Last, we evaluate the performance of our heuristic ALNS solver.

3.4.5.1. Effect of the Number of Traversal Velocities

First, we evaluate the effect of the number of traversal velocities V on the resulting mission duration for the KTSP. The purpose of this evaluation is to identify V such that it represents the best trade-off between the obtained solution quality and the associated computational complexity. Therefore, we solve a total of five different problem instances labeled as 10a, 10b, ... 10e. These problem instances are a subset of the problem instances described in Section 3.4.2.3 and are those that contain exactly ten waypoints.

To solve the corresponding KTSP instances, we set the maximum allowed velocity to 3 m/s and the maximum allowed acceleration to 1.5 m/s^2. The number of traversal heading angles is fixed to $H = 8$ which we derived from the existing literature on the DTSP (see [77]), showing that a discretization $H > 8$ does not contribute notably to the achieved mission duration $\mathcal{M}_T$ but significantly increases the computational complexity. This property is also discovered for the DOP (see [22]).

Hence, for each KTSP instance, we determine the global optimal solution for an increasing number of traversal velocities. For $V = 1$ the traversal velocity is set to $\hat{v}_{max}/\sqrt{2}$. For $V > 1$, the set of possible traversal velocities is defined by $\{i/(V-1) \cdot \hat{v}_{max}/\sqrt{2} \mid i = 0, 1, ..., V-1\}$ following the definition of the KTSP in Section 3.2.1.2. The optimal mission durations for the associated problem instances and for the considered number of traversal velocities V are given in Figure 3.10.

For each instance, it can be seen that the highest mission duration results for $V = 1$. The highest decrease of the minimum mission duration can be observed up to $V = 3$. Increasing the number of traversal velocities to $V = 6$ continues to slightly decrease the resulting mission duration. Further increasing the number of traversal velocities to $V > 6$ has no significant impact on the obtained mission duration, but is likely to negatively impact the computational complexity. Therefore, we set the number of traversal velocities to $V = 6$ for the remainder of this work.

3.4.5.2. Benefit of the KTSP Model

To evaluate the benefit of using the KTSP model, we benchmark it against three different TSP variants. The first two variants are based on the classical traveling salesman problem using the mathematical programming formulation discussed in [96]. Based on this problem

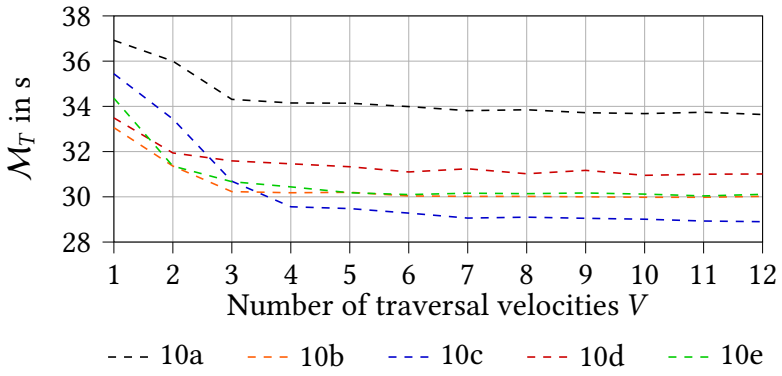


Figure 3.10.: Impact of the number of traversal velocities. The number of traversal angles is fixed to $H = 8$.

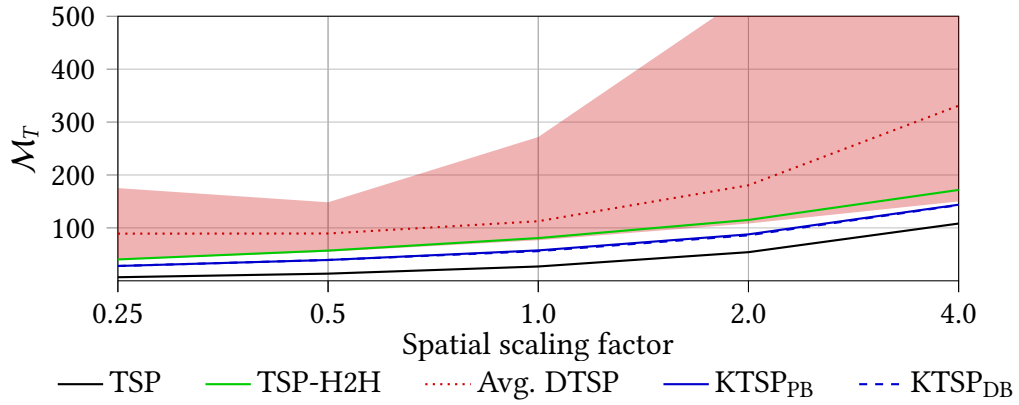
formulation, we, on the one hand, calculate the edge costs as the distance between two nodes divided by the maximum velocity and denote the resulting program as TSP-classic. This approach is widely used in the literature as described in Section 3.1. However, it utilizes edge costs that only represent a lower bound on the physically feasible motion since it assumes a constant flight at maximum velocity neglecting the acceleration properties that are required to change the direction of motion.

As a second benchmark model, we utilize also a classical TSP model, but with the edge costs calculated as hover-to-hover motion, which is described in the following. This model is further denoted as TSP-H2H. The hover-to-hover motion describes that the motion between any pair of waypoints has to start and end at rest. In between, the system is allowed to fully accelerate towards the final waypoint until the maximum velocity is reached and no further acceleration is allowed. The constant velocity at this point is maintained until in the last phase, the system fully decelerates to visit the final waypoint exactly at rest. The algorithm on how to determine hover-to-hover trajectories and especially their associated durations is given in Appendix A. The advantage of the hover-2-hover trajectories is, that since they consider the kinematic constraints of a maximum allowed velocity $\hat{v}_{max}$ and acceleration $\hat{a}_{max}$, they are entirely feasible for the given kinematics.

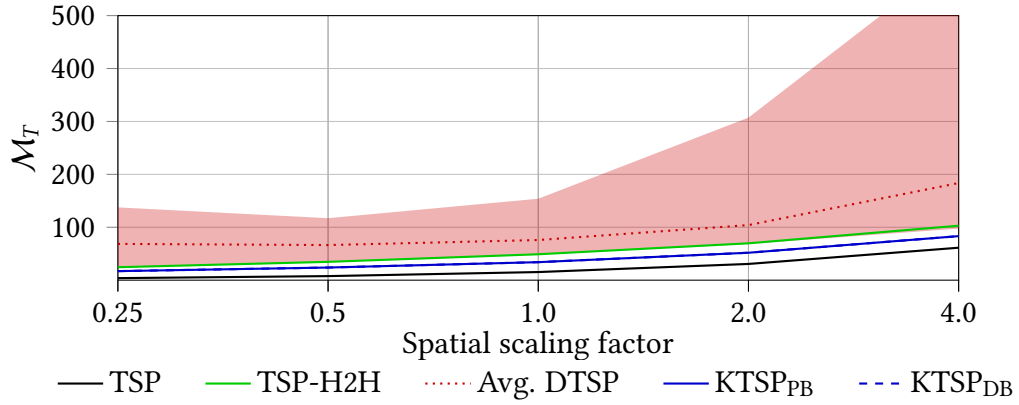
The third benchmark model utilizes the Dubins traveling salesman problem (DTSP) model as we introduce it in Section 3.2.1.1.

For the evaluation of each model, we set the maximum allowed velocity to $v_{max} = 3\,\mathrm{m/s}$ and the maximum allowed acceleration to $a_{max} = 1.5\,\mathrm{m/s^2}$. As benchmark instances, we use the problem instances described in Section 3.4.2.1. These are three basic problem instances with between 21 and 33 waypoints in an area of up to $15.1 \cdot 28.5\,m^2$. For each of these problem instances, we further applied different scaling factors. The results of this benchmark study are illustrated in Figures 3.11, while the original results in tabular format are given in Table C.1 in Appendix C.

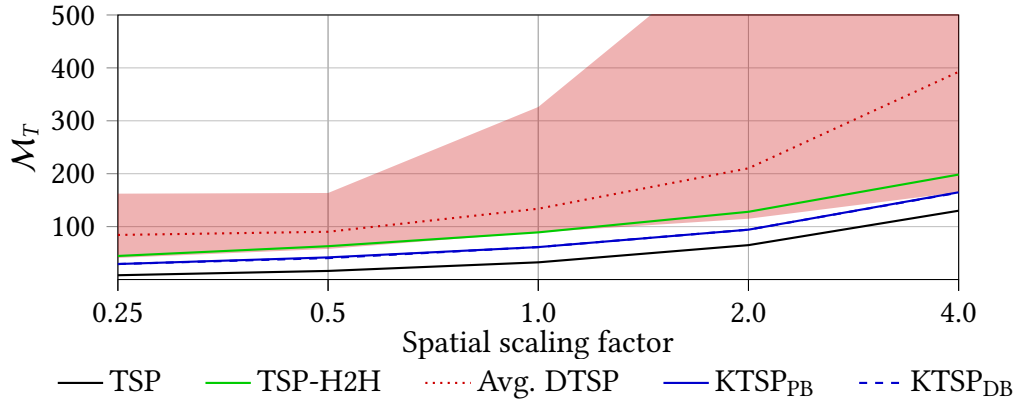
As can be seen, the TSP-classic approach yields the lowest mission duration for each problem instance and each scaling factor. However, the solution can not be tracked by a



(a) Required mission duration for scaled versions of Tsiligirides dataset 1 and a computation time limit of 18000 s.



(b) Required mission duration for scaled versions of Tsiligirides dataset 2 and a computation time limit of 18000 s.



(c) Required mission duration for scaled versions of Tsiligirides dataset 3 and a computation time limit of 18000 s.

Figure 3.11.: Potential of the KTSP model.

simulated UAV as can be seen in Table 3.10. The MAE tracking error ranges from 1.100 m up to 22.175 m.

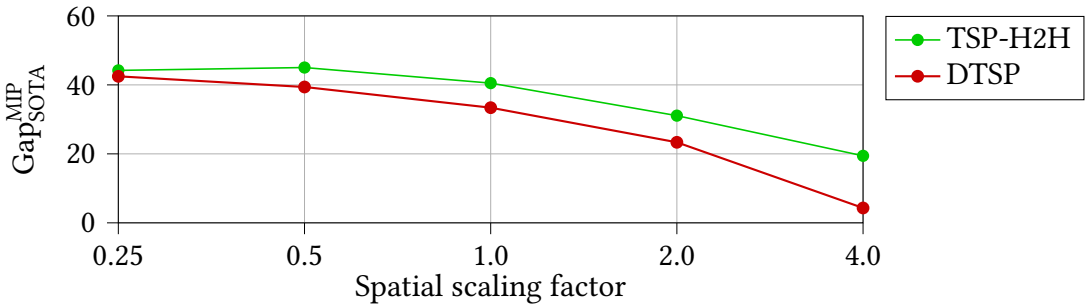
Problem instance	Scale	MAE (m)			
		TSP-classic	TSP-H2H	DTSP	KTSP
TSI1	0.25	1.480	0.001	0.002	0.001
TSI1	0.50	1.264	0.001	0.002	0.001
TSI1	1.00	1.830	0.002	0.004	0.001
TSI1	2.00	2.706	0.005	0.006	0.000
TSI1	4.00	4.237	0.007	0.009	0.000
TSI2	0.25	1.100	0.001	0.001	0.001
TSI2	0.50	1.976	0.001	0.002	0.001
TSI2	1.00	2.858	0.002	0.003	0.001
TSI2	2.00	2.166	0.005	0.023	0.000
TSI2	4.00	3.122	0.008	0.007	0.000
TSI3	0.25	3.408	0.001	0.002	0.001
TSI3	0.50	4.291	0.001	0.003	0.000
TSI3	1.00	4.086	0.003	0.006	0.001
TSI3	2.00	22.175	0.003	0.010	0.001
TSI3	4.00	7.106	0.006	0.009	0.000

Table 3.10.: MAE tracking error for MIP-based solutions of the TSP-classic, TSP-H2H, DTSP, and KTSP in m.

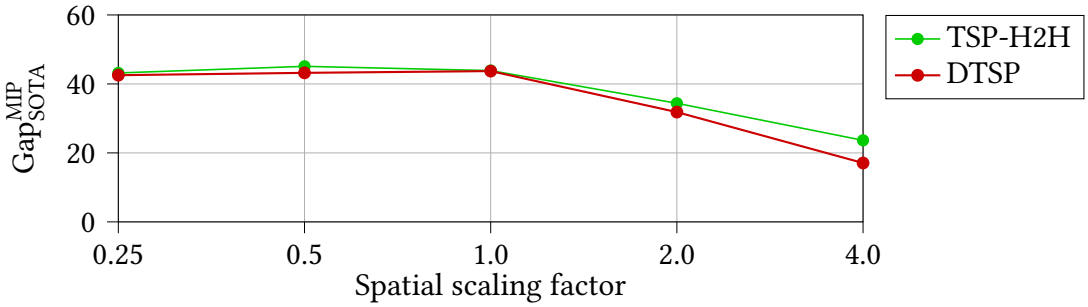
The worst results regarding the mission duration $\mathcal{M}_T$ for each problem instance and scaling factor are achieved by the TSP-H2H approach (see Figures 3.11). However, the solutions are trackable as can be seen by the very low MAE in Table 3.10.

For the results of the DTSP for each problem instance and scale, we solve the corresponding DTSP ten times with ten different constant velocities $v_{const} \in \{i \cdot \hat{v}_{max}, i = 0.1, 0.2, ..., 1.0\}$ to find the best trade-off between velocity and maneuverability. The range of results for the DTSP are given as red shapes in Figures 3.11 while the average required mission duration over all investigated constant velocities is given as a red dotted line. Among the solutions of each constant velocity, we present the best one in the Appendix in Table C.1 which represents the lower bounds of the red shape depicted in Figures 3.11. This lower bound can be interpreted as the best solution among all possible DTSP formulations. Note that the constant velocity yielding the minimum mission duration is not known a-priori and the resulting mission duration for a suboptimal constant velocity might be significantly worse as illustrated in Figure 3.11. However, it can be seen, that the best possible solution of the DTSP constantly yields better results than the TSP-H2H while ensuring the trackability of the solution by a UAV with the given kinematic properties.

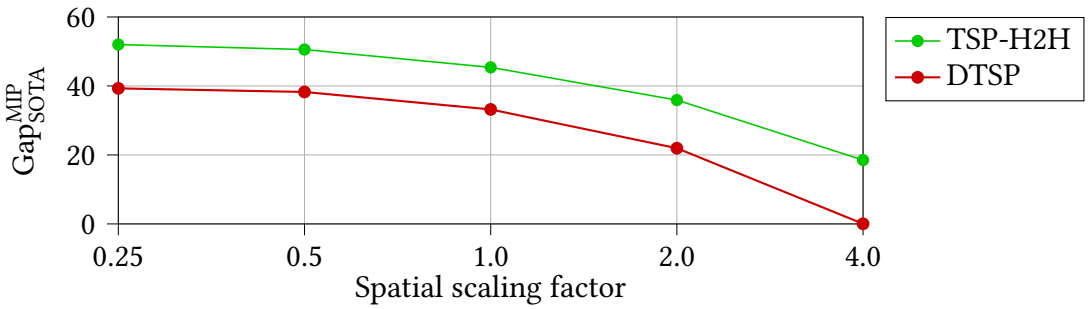
The results of the DTSP are, however, all outperformed by the results of the KTSP (see blue line in Figures 3.11). The results for the KTSP are obtained by using Gurobi with a computation time limit of 5h. Note that some of the KTSP problem instances could not be solved to optimality within the given computation time limit, as given in Table C.1 in Appendix C. On average, the KTSP yields 27.60% better solutions than the best possible solution of the DTSP and 38.19% better solution than TSP-H2H. Further, the TSP-classic approach yielded on average −50.49% faster mission durations than the KTSP.



(a) Deterioration of TSP-H2H and the best DTSP compared to the $KTSP_{PB}$ on the scaled Tsiligirides dataset 1 instances.



(b) Deterioration of TSP-H2H and the best DTSP compared to the $KTSP_{PB}$ on the scaled Tsiligirides dataset 2 instances.

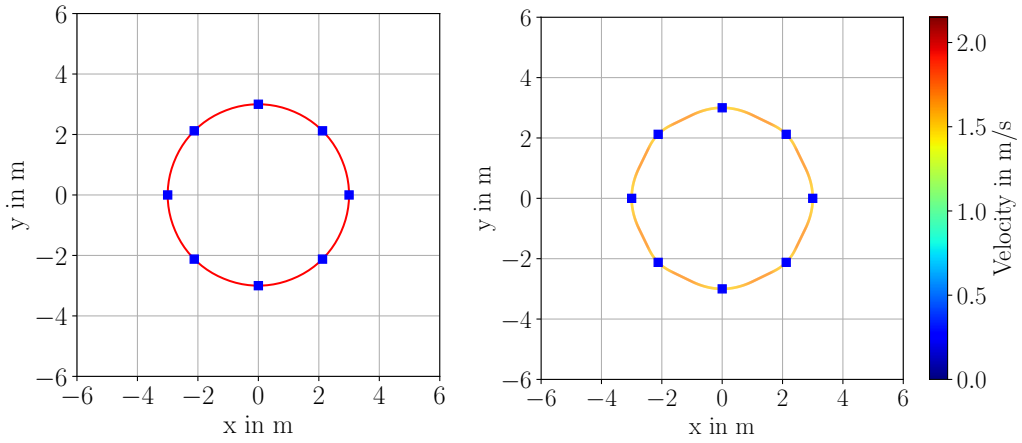


(c) Deterioration of TSP-H2H and the best DTSP compared to the $KTSP_{PB}$ on the scaled Tsiligirides dataset 3 instances.

Figure 3.12.: Deterioration of the optimal solution of TSP-H2H and the best optimal solution of the DTSP among multiple constant velocity options compared to the best solutions for the KTSP for scaled versions of the Tsiligirides datasets [110].

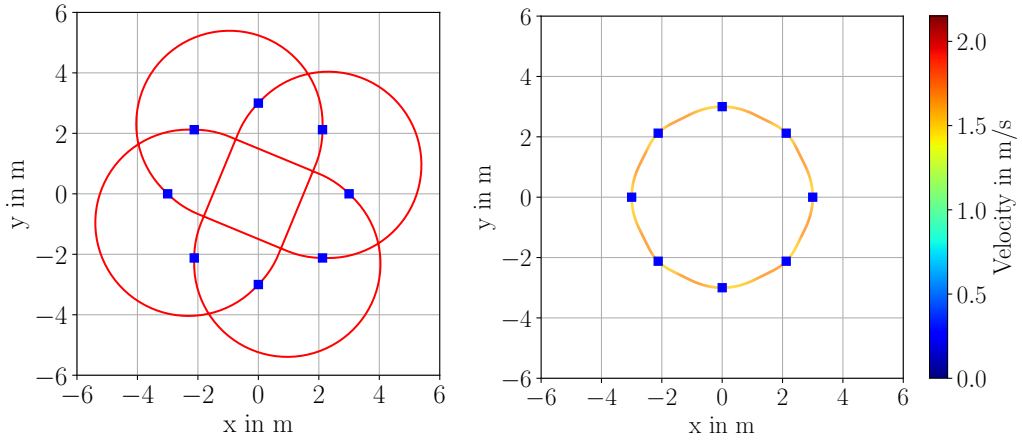
This discrepancy between the TSP-classic and the KTSP approach emphasizes the need to properly consider the acceleration properties of a UAV. The above-described values are calculated using the term $Gap^{MIP}_{SOTA} = SOTA - KTSP_{PB}/KTSP_{PB}$.

The overall improvements of the KTSP compared to the SOTA approaches are given in Figure 3.12. As can be seen, the improvement of the KTSP compared to the TSP-H2H and the DTSP decreases for increasing scaling factors. Consequently, the KTSP seems to offer the highest benefit for flight missions with close waypoints when high agility is required. However, note that in real-world applications, missions with distant waypoints would certainly allow the UAV to move with higher velocities which will increase the benefit of



(a) DTSP with $v_{max} = 2.10\,\text{m/s}$. Total mission duration is 8.97 s.

(b) KTSP with $v_{max} = 2.10\,\text{m/s}$. Total mission duration is 11.91 s.



(c) DTSP with $v_{max} = 2.15\,\text{m/s}$. Total mission duration is 33.65 s.

(d) KTSP with $v_{max} = 2.15\,\text{m/s}$. Total mission duration is 11.65 s.

Figure 3.13.: Robustness of the KTSP model compared with the DTSP.

the KTSP compared to DTSP and TSP-H2H again. This can be illustrated by the following example based on the scaled problem instances derived from Tsiligirides dataset 1. Using a scaling factor of two and changing the maximum allowed velocity from $\hat{v}_{max} = 3\,\text{m/s}$ to $\hat{v}_{max} = 4.5\,\text{m/s}$, then $\text{Gap}^{\text{MIP}}_{\text{SOTA}}$ for the TSP-H2H increases from 31.08% to 41.01% and for the DTSP increases from 23.35% to 33.69%. For the identical setting but a scaling factor of four, $\text{Gap}^{\text{MIP}}_{\text{SOTA}}$ for the TSP-H2H increases 19.43% to 32.52% and for DTS increases from 4.31% to 22.57%.

However, there is unused potential of the KTSP in its current form that can best be illustrated by a comparison with the DTSP on a special problem instance that is given in Figure 3.13. In this experiment, we set the maximum allowed acceleration for the DTSP and KTSP to $1.5\,\text{m/s}^2$ and maximum allowed velocities to once 2.10 m/s and once 2.15 m/s. For both, we set $H = 8$ and additionally for the KTSP, we set $V = 6$. The given problem

instance to be solved represents eight waypoints equally distributed on a two-dimensional circle with a radius of 3 m.

As can be seen in Figures 3.13a and 3.13c the DTSP is very sensitive to changing its constant velocity. Even the small change of the maximum velocity of 0.05 m/s significantly increases the total mission duration $\mathcal{M}_T$ by a factor of ≈ 3.75 due to resulting detours. The KTSP (see Figures 3.13b and 3.13d) is much more robust against varying initial conditions due to its ability to vary the longitudinal velocity. However, the drawback of the KTSP in its current form is also revealed in this example. Comparing the mission duration of DTSP in Figure 3.13a with the mission durations for the KTSP shown in Figures 3.13b, it can be seen that the DTSP is faster than the KTSP. This is because the KTSP only allows to traverse the waypoints with a maximum velocity magnitude of at maximum $\frac{1}{\sqrt{2}} \cdot v_{max}$, since otherwise the decoupling approach for the TOP-UAV time-optimal trajectory generation approach could be infeasible. In general, using TOP-UAV++ would allow higher traversal velocities that would compensate for the drawback. However, in this work, we rely on the KTSP model as stated in Section 3.2.1.2. Note that the adaptation of the mathematical problem to the capabilities of a specific trajectory planner is rather a technical problem than a scientific one. Hence, in this work, we uniformly define the set of possible traversal velocities as given for the KTSP but also the KOP.

3.4.5.3. Limits of Solving the KTSP to Optimality

To evaluate how the KTSP model computationally scales with increasing complexity in comparison to the DTSP, we solve the problem instances introduced in Section 3.4.2.3. These problem instances range from a number of waypoints $L = 10$ to $L = 40$, with steps of two additional waypoints between. For each number of waypoints L, there are five different problem instances given. The kinematic properties are again set to $v_{max} = 3\,\mathrm{m/s}$ and $a_{max} = 1.5\,\mathrm{m/s^2}$. For the solutions of all problem instances with an equal number of waypoints, we list the lowest solution time ($\mathrm{CT_{LB}}$), the average solution time required (CT) and the highest solution time ($\mathrm{CT_{UB}}$) required to solve the problem instances in original form in Table C.2 in Appendix C and visualize these results in Figure 3.14. The computation time limit for Gurobi was set to 3600 s. To enable comparability, Table C.2 also contains the required computation times for using the MIP-based approach provided by Gurobi (see [120]) for our TSP-classic formulation. However, since all TSP-classic problem instances are solved in less than a second, we leave the corresponding results out for the remainder of this section.

In Figure 3.14 the range of required computation times for the associated number of waypoints for the KTSP is illustrated as a blue shape. The lower bound represents the lowest, and the upper bound the highest required computation time. The average required computation time is given as a blue dotted line. Analogously, the results for the DTSP are given in red. As can be seen, the DTSP tends to require increasing computation time for an increasing number of waypoints. However, in all but one problem instances, the required computation time for the DTSP is well below the given computation time limit of one hour. The computation times for the KTSP are significantly higher. Even for smaller instances

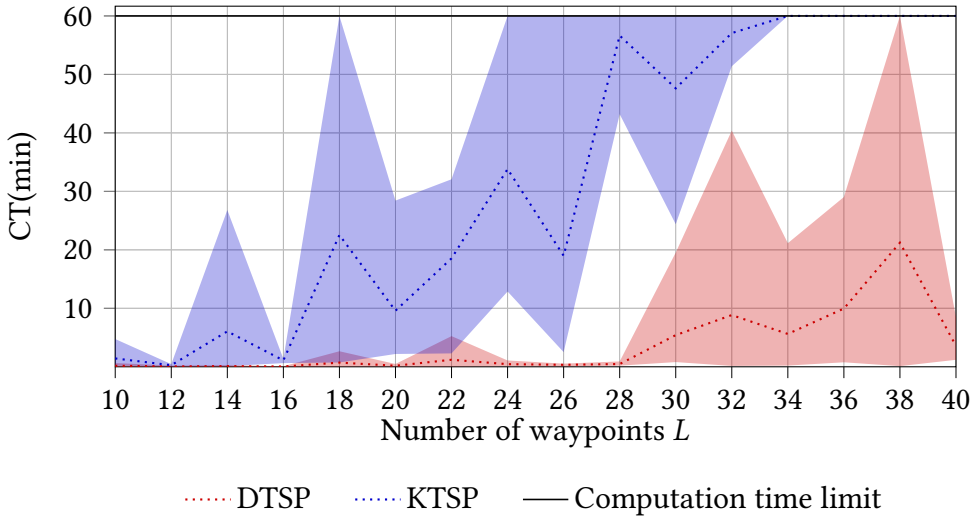


Figure 3.14.: To evaluate the runtimes, we randomly generate 5 different problem instances with an equal number of waypoints. These instances are solved for different TSP variants and the computation time is measured. The time limit for computation is set to 3600 s. The number of allowed heading angles for DTSP and KTSP are $H = 8$. The number of allowed traversal velocities for the KTSP is set to $V = 6$.

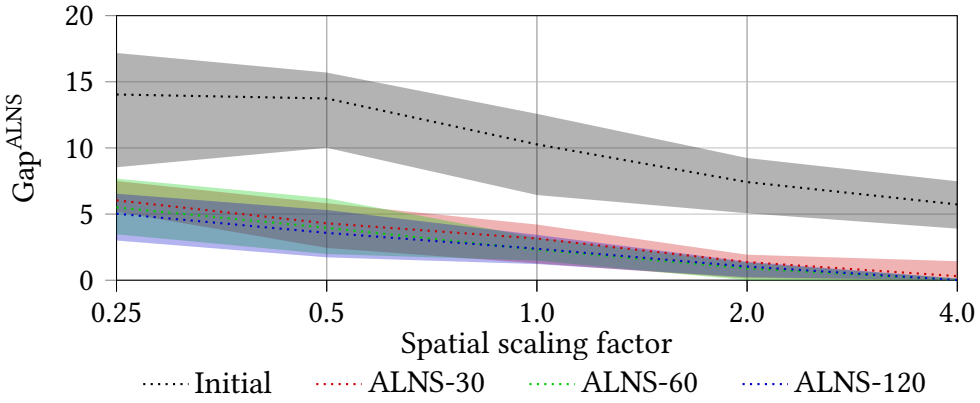
with $L = 18$ waypoints, the computation time limit is reached. For all problem instances with $L \geq 34$ waypoints, our MIP-based approach for the KTSP reaches the computation time limit. With the identified benefit of the KTSP compared to SOTA-models which we derived from the findings in Section 3.4.5.2, the required computation times of the KTSP for larger problem sizes emphasize the need for a heuristic solution approach.

3.4.5.4. Performance of the ALNS to Solve the KTSP

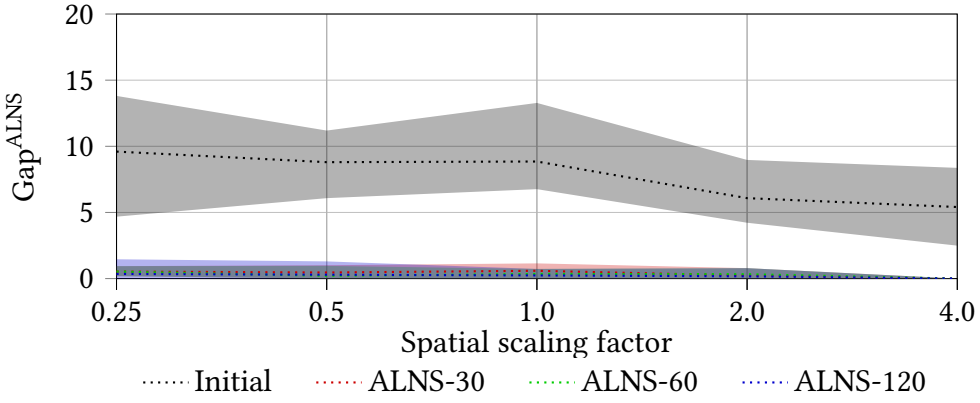
To evaluate the performance of the ALNS, we make use of the benchmark instances, which we introduce in Section 3.4.2.1. The problem instances considered are modified versions of the problem instances provided in [110] to which we apply different spatial scaling factors. This modification allows us to evaluate the connection between the distances of the waypoints and the given kinematic properties.

The evaluation is done for the ALNS with a computation time limit of 30 s, 60 s, as well as 120 s. These short computation time limits are selected since in UAV field applications, missions must usually be calculated in similarly short times.

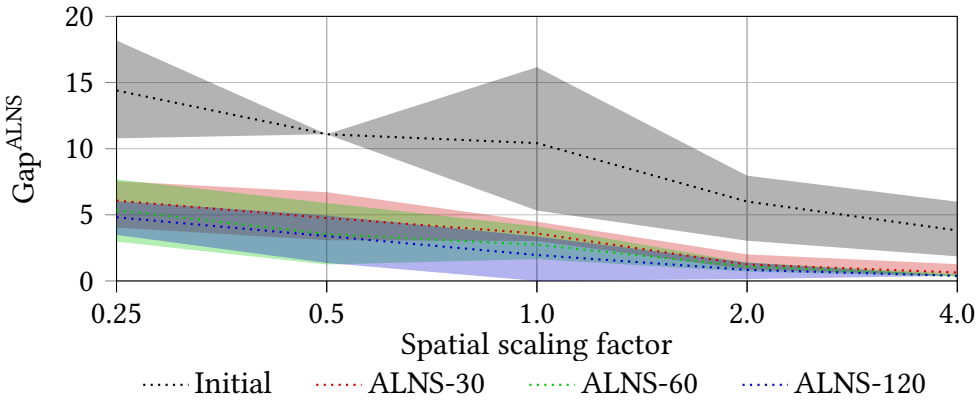
In Figures 3.15, we visualize the results of our computational study as given in original form in Table C.3 in Appendix C. The figures show the Gap^{ALNS} between the solutions obtained by our ALNS solver and the corresponding solution of our MIP-based approach with a computation time limit of five hours. Since our ALNS is based on randomness, we solve each problem instance considering each scaling factor ten times with different random seeds. From these ten runs, a best, worst, and average obtained Gap^{ALNS} is derived as depicted by the colored band shapes. The gray band shapes represent the range of obtained



(a) Deterioration of the initial solution and the ALNS solution compared to the corresponding $KTSP_{PB}$ for scaled versions of Tsiligirides dataset 1 with a computation time limit of 18000 s.



(b) Deterioration of the initial solution and the ALNS solution compared to the corresponding $KTSP_{PB}$ for scaled versions of Tsiligirides dataset 2 with a computation time limit of 18000 s.



(c) Deterioration of the initial solution and the ALNS solution compared to the corresponding $KTSP_{PB}$ for scaled versions of Tsiligirides dataset 3 with a computation time limit of 18000 s.

Figure 3.15.: Performance of the ALNS to solve the KTSP.

solution qualities of the initially constructed solution. The red, green, and blue band shapes represent the range of obtained solution qualities for our ALNS with a computation time limit of 30 s, 60 s, and 120 s. The average obtained Gap^{ALNS} is given in the corresponding colors as dotted lines.

Overall, the problem instances with close waypoints seem to be harder to solve for our ALNS since the obtained optimality gap reaches up to 7.68% in the worst case. This holds especially for larger problem instances such as Tsiligirides dataset 1 with 32 waypoints and Tsiligirides dataset 3 with 33 waypoints. For increasing scaling factors, the Gap^{ALNS} significantly decreases for each problem instance yielding a worst-case gap of 1.44% among all problem instances. We explain this as follows: For high scaling factors, the traversal properties lose importance because the motion between two waypoints is mostly defined by the distance and the maximum velocity. In the case of a high scaling factor, the motion remains for a large part of the mission duration at velocities close to $\hat{v}_{max}$. Consequently, the figures indicate, that the heuristics used by our ALNS show a good performance in finding waypoint sequences of high quality.

Further, it is worth mentioning that our ALNS performs highly effectively for small problem instances such as the Tsiligirides dataset 2 with 21 waypoints. Here, the worst Gap^{ALNS} among all considered problem instances with a computation time limit of 30 s is only 1.15%.

Overall, increasing the computation time limit for the ALNS tends to slightly increase the quality of the yielded solution. This observation is in line with the expected behavior of our solver since it can search for better solutions for a longer time.

In this context, it has to be mentioned that our current ALNS implementation does not support multithreading to fully exploit the capability of the CPU. Each CPU core could be assigned a single thread to solve the KTSP with a different random seed. Hence, the worst-case performance using all 4 cores of our Intel(R) Core(TM) i7-8565U CPU would increase the probability that the yielded solution by our solver is similar to the best-case performance over ten runs as shown in Table C.3. This clearly shows the potential of the ALNS to solve even larger KTSPs while given only short time limits.

To conclude, the ALNS approach is highly competitive to the exact approach, especially for scenarios with limited computation time. However, the results also indicated that yielding close to optimum solutions of the KTSP is difficult, especially for solving larger problem instances with close waypoints.

3.4.6. Computational Results for the KOP

We present the computational results for the KOP in this section. First, we investigate the benefit of the KOP model compared to models from the SOTA. This is followed by an investigation and comparison of the computational complexity of the KOP and the associated SOTA approaches for problem instances with an increasing number of waypoints. Next, we evaluate the performance of our ALNS approach and compare it

with the results yielded using our MIP-based approach. Lastly, we benchmark our ALNS approach against the latest results of an MPC-based approach from the literature.

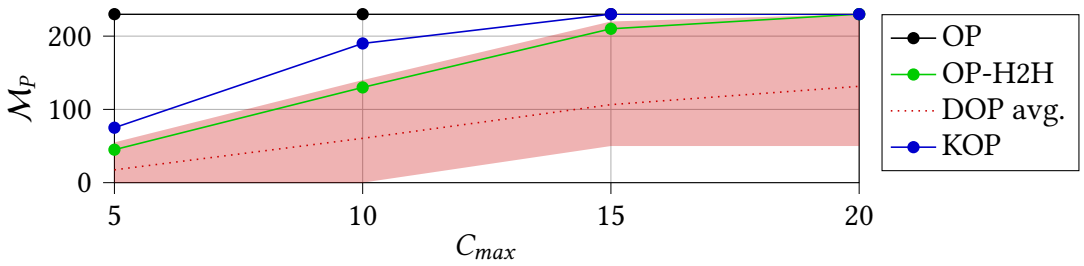
3.4.6.1. Benefit of the KOP Model

As a benchmark model for the KOP, we consider the classical OP (see e.g. [97]) in two variations. On the one hand, we use the distance between waypoints divided by the maximum velocity as edge costs. This variant is further denoted as OP-classic. On the other hand, we utilize the duration of the hover-to-hover trajectories described in Appendix A as edge costs. As the last benchmark model, we utilize the Dubins orienteering problem (DOP) which is an extension of the OP using Dubins path (see Section 3.2.2.1).

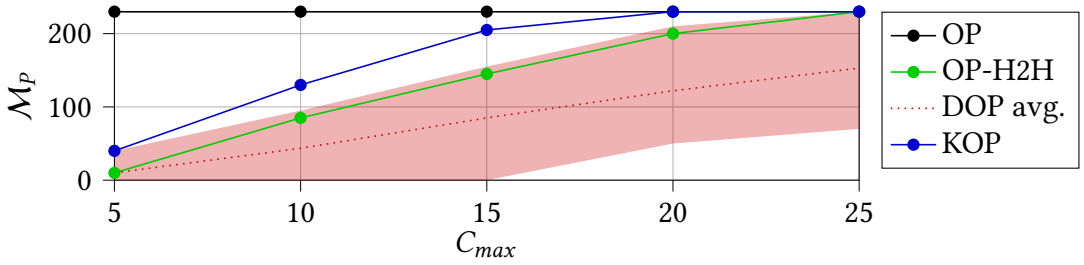
For the evaluation of each model, we set the maximum allowed velocity to $v_{max} = 3\,\mathrm{m/s}$ and the maximum allowed acceleration to $a_{max} = 1.5\,\mathrm{m/s^2}$. As a benchmark instance, we use a modified version of the Tsiligirides datasets 2 as we describe in Section 3.4.2.4. All the problem instances from the modified dataset offer a total of 230 priorities for collection. In analogy to the computational study conducted on the KTSP in Section 3.4.5.2, we evaluate the connection between the distance between the waypoints and the given kinematic properties of our system by scaling the coordinates of each waypoint. In this evaluation, the scaling factors are again 0.25, 0.5, 1.0, 2.0, and 4.0. The maximum allowed travel time for each problem instance as well as the computational results for each benchmark model are given in Figures 3.16. These figures visualize the results given in the original form in Table 3.11 and show the performance of each investigated method on the given problem instance for different maximum travel time budgets and different spatial scaling factors. To generate the results, the corresponding problems are all solved to optimality using Gurobi as general-purpose solver and a computation time limit of 18000 s. The accumulated collected priorities of the OP-classic approach are given in black, the OP-H2H in green, and the KOP in blue. The collected priorities for the DOP depend on the constant velocity. Analogously to the DTSP in Section 3.4.5.2, we solve the corresponding DOP ten times each with a different constant velocity $v_{const} \in \{i \cdot \hat{v}_{max}, i = 0.1, 0.2, ..., 1.0\}$ to find the best trade-off between the velocity magnitude and the maneuverability. The range of the obtained accumulated collected priorities is indicated by the red shape and its bounds represent the maximum and minimum collected priorities over all investigated constant velocities. The average collected priority values are given as a red dotted line.

For all problem instances, it can be seen that the OP-classic model always collects the most priorities. This is because it does not account for the kinematics of the systems which leads to the problem that the resulting solution cannot be tracked with sufficient precision, as can be seen in Table 3.11. On the other hand, the OP-H2H model yields feasible solutions which indicated by the low MAE given in Table 3.11. However, due to the restriction of resting at each waypoint, the yielded solutions collect significantly fewer priorities than the OP-classical.

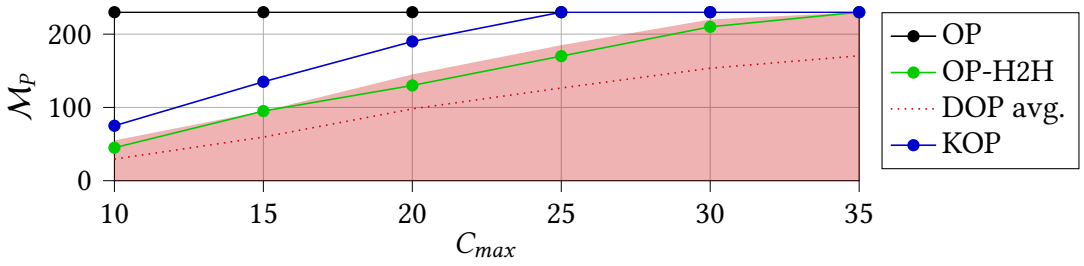
The performance of the DOP model highly depends on the selected constant velocity. In the majority of all cases, the DOP collects fewer priorities than all other models. However,



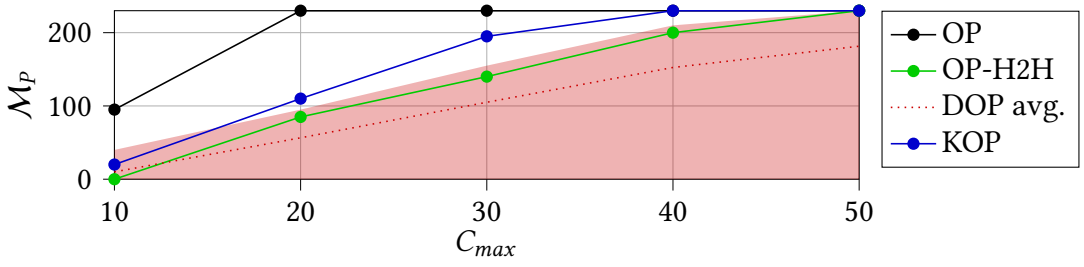
(a) Spatial scaling factor 0.25



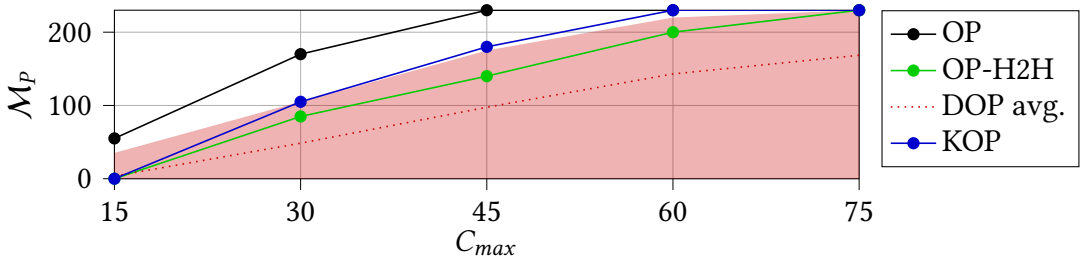
(b) Spatial scaling factor 0.50



(c) Spatial scaling factor 1.00



(d) Spatial scaling factor 2.00



(e) Spatial scaling factor 4.00

Figure 3.16.: Optimal for OP variants for the first 15 waypoints of Tsiligirides dataset 2 scaled with different factors. All investigated problem instances for each model could be solved to optimality within a computation time limit of 18000 s. The quantitative results are given in Table 3.11.

Scale	C_{max} (s)	OP-classic		OP-H2H		DOP		KOP	
		$\mathcal{M}_P$	MAE	$\mathcal{M}_P$	MAE	$\mathcal{M}_P$	MAE	$\mathcal{M}_P$	MAE
0.25	5	230	0.480	45	0.001	55	0.078	75	0.001
0.25	10	230	0.390	130	0.001	140	0.022	190	0.001
0.25	15	230	0.390	210	0.001	220	0.015	230	0.001
0.25	20	230	0.390	230	0.001	230	0.022	230	0.001
0.50	5	230	1.040	10	0.002	40	0.394	40	0.000
0.50	10	230	1.025	85	0.001	95	0.202	130	0.001
0.50	15	230	1.023	145	0.001	155	0.047	205	0.001
0.50	20	230	0.914	200	0.001	210	0.044	230	0.001
0.50	25	230	0.914	230	0.001	230	0.012	230	0.001
1.00	10	230	2.890	45	0.001	55	0.364	75	0.000
1.00	15	230	2.830	95	0.001	95	0.356	135	0.001
1.00	20	230	2.534	130	0.001	145	0.428	190	0.001
1.00	25	230	2.160	170	0.001	185	0.452	230	0.000
1.00	30	230	2.428	210	0.001	220	0.063	230	0.000
1.00	35	230	2.705	230	0.002	230	0.067	230	0.000
2.00	10	95	1.278	0	0.000	40	1.610	20	0.000
2.00	20	230	5.693	85	0.002	95	1.079	110	0.000
2.00	30	230	4.019	140	0.010	155	0.260	195	0.000
2.00	40	230	5.390	200	0.005	210	0.369	230	0.000
2.00	50	230	3.649	230	0.030	230	0.092	230	0.000
4.00	30	170	12.732	85	0.003	105	1.714	105	0.000
4.00	45	230	14.266	140	0.048	175	0.871	180	0.000
4.00	60	230	3.998	200	0.068	220	0.382	230	0.000
4.00	75	230	4.393	230	0.091	230	0.350	230	0.000

Table 3.11.: Optimal results for OP variants for the first 15 waypoints of Tsiligirides dataset 2 scaled with different factors. All investigated problem instances for each model could be solved to optimality within a computation time limit of 18000 s.

when the constant velocity for the DOP is selected properly, it constantly yields solutions that are at least as good as the solutions of the OP-H2H. Another interesting observation is that especially for short maximum flight time budgets the DOP sometimes outperforms the KOP. This can be explained by the fact that our KOP solution must start and end rest. The DOP starts and ends at its associated constant velocity. Especially for short maximum flight time budgets, this explains why the DOP collects more priorities than the KOP. This also explains the notable tracking error MAE for the DOP solutions, which can be seen in Table 3.11. However, remember that only the best possible solution of the DOP is given in the table. Moving at lower constant velocities would increase the capability of the UAV to track the reference with reduced MAE tracking errors. This is the reason why the MAE is smaller for small scaling factors of the problem instances although in these cases the mission durations are shorter and the timeshare of the transient to follow the reference trajectory compared to the entire mission duration is smaller.

As Figures 3.16 show, the collected priorities for the KOP are significantly higher than the ones collected for OP-H2H for all considered problem instances. Further, the results are also significantly better solutions than the best possible solution of the DOP in the majority of cases while always guaranteeing the kinematic executability of the calculated flight missions. Only for very short mission durations and spatial scaling factors ≥ 2, the DOP yields more collected priorities than the KOP. However, this can be explained by

the kinematically infeasible constant velocity at the start and end waypoint. Overall, the results show that the KOP model is highly beneficial when solving orienteering problems for physically inert systems.

3.4.6.2. Limits of Solving the KOP to Optimality

To assess the scaling properties of the KOP model compared to the OP-classic and the DOP, we make use of the set of 25 ordered sets of waypoints with start and end waypoint specified and which we introduce in Section 3.4.2.6. Each of these waypoint sets is complemented to a proper problem instance by assigning five different maximum travel time budgets $C_{max} \in \{10, 15, 20, 25\}$ as given in Table 3.12. This accumulates to a total of 100 investigated problem instances. For this study, we apply the default settings as described in Section 3.4.1 except for the computation time limit for our MIP-based approaches, which we set to 3600 s. Table 3.12 shows the lowest, average, and highest computation time required to solve a single problem instance by our MIP-based approaches for the OP-classic, DOP, and KOP.

L	C_{max}	OP-classic			DOP			KOP		
		CT_{LB} (s)	CT (s)	CT_{UB} (s)	CT_{LB} (s)	CT (s)	CT_{UB} (s)	CT_{LB} (s)	CT (s)	CT_{UB} (s)
5	10	0.00	0.00	0.01	0.00	0.00	0.00	0.08	1.42	6.66
	15	0.00	0.00	0.00	0.00	0.01	0.01	0.23	0.81	1.72
	20	0.00	0.00	0.00	0.01	0.01	0.02	0.37	0.57	0.92
	25	0.00	0.00	0.01	0.02	0.02	0.03	0.32	0.36	0.42
10	10	0.01	0.02	0.03	0.01	0.01	0.02	1.93	19.23	69.27
	15	0.01	0.01	0.02	0.04	0.07	0.13	5.70	739.31	3600.00
	20	0.01	0.02	0.04	0.11	0.24	0.42	13.91	1107.77	3600.00
	25	0.00	0.01	0.01	0.23	0.44	0.96	10.64	133.23	558.80
15	10	0.04	0.19	0.41	0.03	0.04	0.05	13.52	64.94	21.63
	15	0.03	0.17	0.45	0.48	0.62	0.75	24.39	1480.36	3600.00
	20	0.04	0.16	0.30	0.24	0.34	0.56	65.85	120.00	208.68
	25	0.02	0.03	0.05	0.40	0.63	0.83	41.50	1470.65	3600.00
20	10	0.06	0.32	0.73	0.05	0.06	0.09	13.99	142.51	344.28
	15	0.09	0.20	0.49	1.21	1.29	1.44	68.75	1172.36	3600.00
	20	0.10	1.13	4.72	0.47	1.12	3.20	241.59	945.30	3163.05
	25	0.04	0.29	0.73	1.10	1.70	3.11	135.23	396.33	718.77
25	10	0.18	0.60	1.34	0.09	0.16	0.21	47.48	70.61	108.01
	15	0.52	9.63	41.91	1.92	2.31	3.31	249.86	1027.59	3600.00
	20	0.45	3.82	10.05	1.18	2.60	5.60	393.65	2616.26	3600.00
	25	0.23	1.71	4.03	1.78	4.11	6.87	324.37	1943.62	3600.00

Table 3.12.: To evaluate the runtimes, we randomly generate 5 different problem instances with an equal number of waypoints. These instances are solved for different OP variants and the computation time is measured. The time limit for computation is set to 3600 s. The number of allowed heading angles for the DOP and the KOP are $H = 8$. The number of allowed traversal velocities for the KOP is set to $V = 6$.

As can be seen, the OP-classic approach mostly requires only a fraction of a second until the global optimum solution is found. The overall highest computation time is obtained for a single problem instance with 25 waypoints and is 41.91 s. The second highest computation time is 10.05 s, which is also obtained for a problem instance with 25 waypoints. The DOP

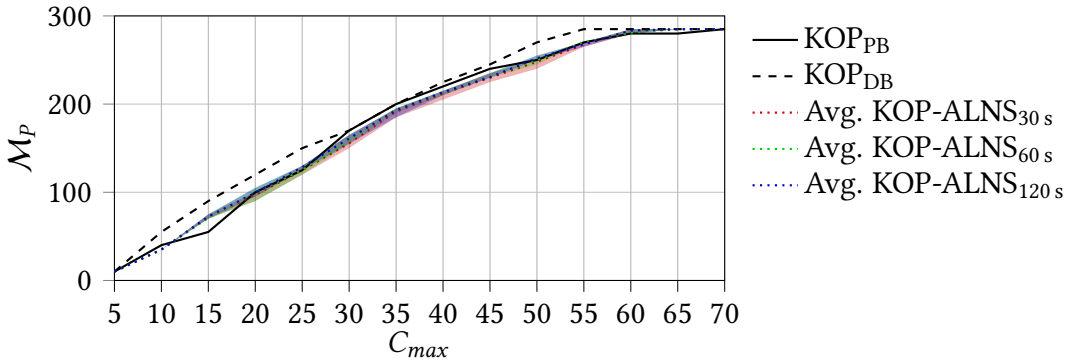
requires computation times in the scale of a few seconds as well. However, the average computation times are slightly higher than the ones required by the OP-classic approach for the problem instances related to 25 waypoints and a maximum travel time budget of $C_{max} \in \{15, 20\}$. Remind, that all waypoints are placed in a square with an edge length of 20 meters. Given the constant maximum velocity of 3 m/s and a maximum acceleration of 1.5 m/s yielding a turning diameter of 12 m shows, that the waypoints are too dense to be suited for the DOP. Hence, we assume that the investigated problem instances are easy to solve since many solution candidates are quickly identified to be infeasible. The highest runtime is required by the KOP approach. Especially for problem instances with a larger number of waypoints, i.e. $L \geq 10$. In these cases, the computation time limit of $\mathrm{CT}_{max} = 3600\,\mathrm{s}$ is mostly reached for at least one problem instance of each group. Only for problem instances with five waypoints, or with a maximum travel time budget of $C_{max} = 10\,\mathrm{s}$, the required computation time is constantly rather low with a maximum of 344.28 s. These experiments show two things: First, the KOP is significantly more complex than the DOP and requires powerful heuristic solution approaches to unfold the benefits identified in Section 3.4.6.1. Second, the KOP can be applied to a wide range of problem instances and applications for which the DOP is not well-suited.

3.4.6.3. Performance of the ALNS to Solve the KOP

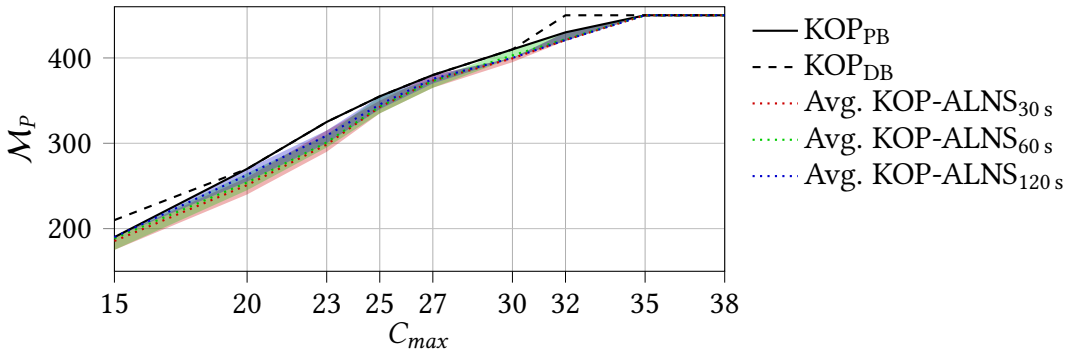
For the KOP, we evaluate the performance of the ALNS against the best solutions yielded by our MIP-based approach with a time limit of 18000 s on the original problem instances Tsiligirides datasets 1, 2, and 3 (see [110]) with the specified travel budget interpreted as maximum flight time. These problem instances are provided as open source at https://www.mech.kuleuven.be/en/cib/op#section-0. For our evaluation, we make use of the default computational setup as introduced in Section 3.4.1 which among other things defines a maximum velocity of $\hat{v}_{max} = 3\,\mathrm{m/s}$ and a maximum acceleration $\hat{a}_{max} = 1.5\,\mathrm{m/s^2}$.

We solved each problem instance ten times with different random seeds for ALNS solvers with a computation time limit of 30 s, 60 s, and 120 s. For each solver, the best and worst solution found among the ten runs is illustrated by the colored band shape in red (KOP-ALNS$_{30\,\mathrm{s}}$), green (KOP-ALNS$_{60\,\mathrm{s}}$), and blue (KOP-ALNS$_{120\,\mathrm{s}}$) in Figures 3.17. The average collected priorities are presented as dotted lines in the respective colors. Further, we present the objective values of the best solution found by our MIP-based approach as a black solid line and the corresponding dual bound as a black dotted line. The original results are also provided in Tables C.4, C.5, and C.6 in Appendix C.

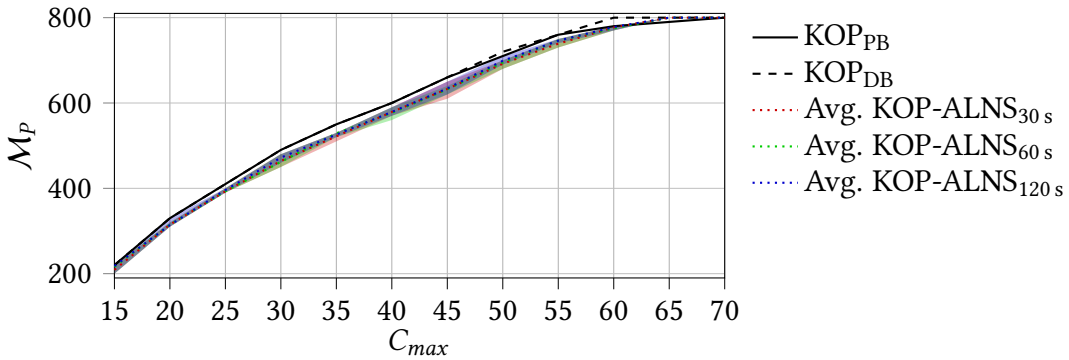
For dataset 1 with 32 waypoints according to Table C.4, it can be seen that our MIP-based approach reached the computation time limit of five hours in 10 out of 14 problem instances. Among these cases, the average $\mathrm{Gap}^{\mathrm{MIP}}$ is 22.60% while the highest $\mathrm{Gap}^{\mathrm{MIP}}$ is 63.64%. For all three computation time limits, our ALNS solution approach yields solutions of high quality as can be seen in the following while requiring only a fraction of the computation time requried by the our MIP-based approach. The average gap between the best solutions found by our ALNS compared to the solutions yielded by our MIP-based approach is −1.9%. Note that negative values indicate that the ALNS did find better solutions than the exact



(a) Performance of the ALNS for the KOP on Tsiligirides dataset 1



(b) Performance of the ALNS for the KOP on Tsiligirides dataset 2



(c) Performance of the ALNS for the KOP on Tsiligirides dataset 3

Figure 3.17.: Performance of the ALNS for the KOP on Tsiligirides datasets.

approach. For dataset 1, our KOP-ALNS$_{30\,s}$ did find better solutions than our MIP-based approach in three out of 14 problem instances while our KOP-ALNS$_{120\,s}$ approach bet our MIP-based approach in 6 out of 14 problem instances.

Considering all investigated problem instances, it can be seen that the assignment of a higher computation time limit to our ALNS increases the probability of yielding a

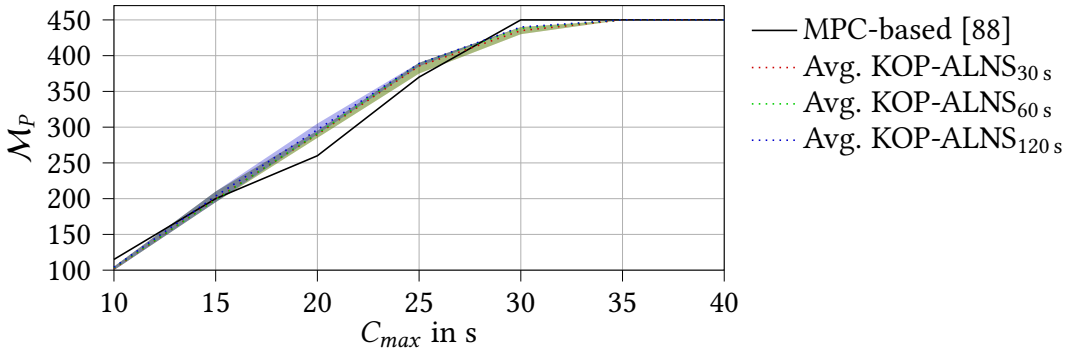
solution of high quality. This can be explained as follows by focussing on the average performance of our solvers averaged over all problem instances of a dataset: As an example, in dataset 1, the averaged Gap^{ALNS} value for the KOP-ALNS$_{30\,\text{s}}$ is 0.36 %, while the averaged Gap^{ALNS} for the KOP-ALNS$_{60\,\text{s}}$ is -0.19 % and for KOP-ALNS$_{120\,\text{s}}$ is -0.65 %. For dataset 2, the corresponding averaged Gap^{ALNS} values are 3.04 %, 2.54 %, and 1.77 % and for dataset 3, the averaged Gap^{ALNS} values are 3.11 %, 2.70 %, and 2.35 %. These results demonstrated the competitiveness of our ALNS solvers compared to the MIP-based approach. Remember that our ALNS is a single-threaded executable and these results are based on the average performance over ten runs for each problem instance, not the best performance. Hence, running our ALNS for a single problem instance in parallel on four different threads with a different random seed, i.e. each on a dedicated core of our Intel(R) Core(TM) i7-8565U CPU such that all available cores are utilized, the yielded solutions would tend to be closer to the best performance of our solver. This further emphasizes the potential benefit of our approach. However, note that for some runs, our ALNS yielded insufficient solution qualities with gaps of up to 11.8 %. This shows that further work must be conducted to improve the consistency of our ALNS solver. in yielding high-quality solutions.

3.4.6.4. Benchmark Against the Approach from [88]

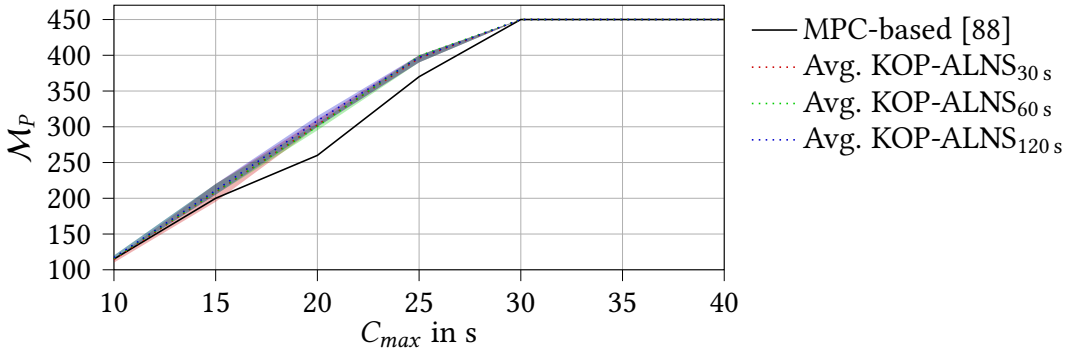
Last, we compare our ALNS approach for the KOP against the MPC-based approach published in [88]. In their work, the maximum velocity norm is constrained by $\sqrt{v_x^2 + v_y^2} \leq \hat{v}_{max} = 3\ \text{m/s}^2$. Further, the authors restricted the maximum acceleration for the i-the axis to $|a_{max,i}| = \hat{a}_{max}/\sqrt{2}$ with $\hat{a}_{max} = 1.5\,\text{m/s}^2$.

In our study, we comply with these constraints by using on the one hand our TOP-UAV and on the other hand our TOP-UAV++ trajectory planner specified with $\hat{v}_{max} = 3$ m/s and $\hat{a}_{max} = 1.5\ \text{m/s}^2$. We solve the associated problem instance derived from Tsiligirides dataset 2 (see [110]) using our ALNS with a computation time limit of 30 s, 60 s, and 120 s. We want to emphasize, that our KOP model is constrained to start and end at rest while the solutions presented in [88] may start and end with an arbitrary velocity magnitude between 0 m/s and $\hat{v}_{max} = 3\ \text{m/s}^2$. Note, our TOP-UAV trajectory planner allows lower maximum velocity in certain directions due to the decoupling of the axes and equal maximum acceleration properties compared to the approach in [88]. Further, our TOP-UAV++ planner also allows a slightly lower but similar maximum velocity compared to the approach in [88]. However, it allows a higher maximum acceleration for certain directions due to the multiple configurations for sharing the acceleration norm among the axes.

In Figures 3.18a and 3.18b, we present the solutions yielded in [88] as a black solid line while the solutions yielded by our ALNS approaches are given colored. For each of our ALNS solvers, we provide the range between the worst and best solution found over ten runs with different seeds as a colored band shape. The average performance of the associated solver is given as a dotted line in the corresponding color. The results of our KOP-ALNS$_{30\,\text{s}}$ are given in red, of our KOP-ALNS$_{60\,\text{s}}$ in green, and of our KOP-ALNS$_{120\,\text{s}}$



(a) Results for the our ALNS based on the TOP-UAV trajectory planner.



(b) Results for the our ALNS based on the TOP-UAV++ trajectory planner.

Figure 3.18.: Performance evaluation of the ALNS against best solutions yielded by the MPC approach from [88] on Tsiligirides dataset 2. The ALNS using our TOP-UAV and TOP-UAV++ trajectory planner is evaluated given 30 s, 60 s and 120 s as computation time limit and ran ten times with different random seeds on each problem instance.

in blue. As can be seen in Figure 3.18a, although our solver using TOP-UAV is more constrained in velocity than the approach in [88] and must start and end at rest, it yields on average at least as good solution as the benchmarked MPC-approach in four out of seven problem instances. Only for the problem instances with $C_{max} \in \{10\,s, 30\,s\}$, our ALNS approach was outperformed by the MPC approach. We explain this observation by the more constrained kinematics of our solver and the start/end at rest restriction.

For our ALNS solvers using TOP-UAV++ (see Figure 3.18b), the average performance is at least as good as the MPC approach in all instances. In three out of 7 instances, the average performance of our ALNS solvers significantly outperforms the MPC approach. Note that in three out of seven problem instances, i.e for $C_{max} \in \{30\,s, 35\,s, 40\,s\}$, our ALNS and the MPC-based approach constantly collect all priorities available for the problem instances.

This indicates the potential of using our ALNS approach compared to the MPC-based approach. We further want to emphasize a further advantage of our ALNS approach. Our KOP approach is based on a set of edges that must be precomputed, for which we use our efficient TOP-UAV or TOP-UAV++ trajectory planner. The computation time of these

edge costs does not depend on the distance between the waypoints, the maximum velocity, or the maximum acceleration. This is different for the MPC approach since for each time step, there is a decision variable deciding which acceleration to apply on each coordinate axis, there would be an increasing number of decision variables for an increasing number of time steps, e.g. due to longer mission durations. The keep track of the computational complexity, the time step length must be increased for long-duration missions from which consideration of the maneuverability of the UAV suffers.

The original results accompanying the Figures can be found in Appendix C in Tables C.7 and C.8.

3.5. Conclusion

In this chapter, we presented existing work on inertia-based routing. Further, we provided two new problem formulations, namely the KTSP and the KOP and introduced a heuristic solution framework based on the ALNS to solve them. The results show that the KTSP and KOP models are highly beneficial when it comes to routing problems with inertia. However, we show that obtaining the proven global optimum solution by applying a commercial solver such as Gurobi becomes computationally very expensive, especially for larger problem instances. This motivates the need for powerful heuristic solvers. Our ALNS is capable of yielding high-quality solutions for the KTSP and the KOP in a short time.

4. Conclusions and Outlook

In this thesis, we presented deep insights into inertia-based routing problems. In Chapter 1, we motivated why inertia-based routing is especially important for multirotor UAV applications and indicated the limits of modern approaches. In Chapter 2, we introduced our time-optimal trajectory planner that is used in Chapter 3 to estimate the flight time between spatial waypoints and which overcome major flaw in the current state-of-the-art to calculating time-optimal trajectories for point-masses. In Chapter 3, we specifically focus on inertia-based routing problems. We introduce new mathematical models, namely the KTSP and the KOP, and develop powerful heuristic solution approaches based on the adaptive large neighborhood search framework. In this chapter, we summarize the findings of this work and give an outlook for promising future research directions.

4.1. Summary and Results

In Chapter 1, we presented a general introduction to UAV routing problems and especially focused on surveillance and data collection applications. Further, we give an example problem of such a data collection mission and show the solution of the mission planning with approaches from the current state-of-the-art. Assuming a simulated multirotor UAV trying to follow the trajectory representing the yielded solution of the routing problem, we see that the SOTA approaches either significantly underestimate the required flight time between two waypoints by neglecting essential physical capabilities of the simulated UAV, or overestimate them by not properly consider these capabilities of the UAV. This might lead to two different problems: On the one hand, the computed reference trajectory is not compatible with the capabilities of the UAV and spatial discrepancies in the mission execution result. On the other hand, although the computed reference trajectory is compatible with the physical capabilities of the UAV, it does not fully exploit these. Hence, the reference trajectory turns out to be time-inefficient.

We identify the root of this problem to be the type of applied trajectory planning approaches since none of the state-of-the-art approaches for UAV routing correctly cover the full kinematic properties of a multirotor UAV, i.e. their ability to arbitrarily change velocity magnitude and direction by applying acceleration forces in specific directions.

Therefore, we investigate possible candidates for trajectory planning of multirotor UAVs in Chapter 2. Crucial aspects of an ideal candidate for trajectory planning are, that the approach considers the full kinematics of a multirotor UAV, yields time-optimal motions since minimum time is one of the highest objectives in UAV routing applications, and

calculates trajectories within the scale of a few hundred microseconds since in UAV routing problems thousands of trajectories must be calculated. In the literature, we identified one promising candidate that covers all these specifications. However, we also identified that this approach is not generally correct and sometimes yields trajectories that miss the required final waypoint by far. Hence, we developed a new optimization-based approach which we could prove to overcome the identified issue of the state-of-the-art approach. During the development of our TOP-UAV approach, we also identified the cause of why the SOTA approach fails under some circumstances. To even better exploit the kinematic capabilities of multirotor UAVs, we improved our trajectory planner to TOP-UAV++ and provided the source code of both trajectory planners as open source in C++ and Python for the two- and three-dimensional case. Moreover, we presented an extensive computational study and show, that our approaches solve the time-optimal trajectory planning problem optimally within the scale of a few hundred nanoseconds which is even faster than required.

With our TOP-UAV++ trajectory planner at hand, we focussed on the inertia-based routing problem for multirotor UAVs in Chapter 3. We first give an overview of existing literature and then define the kinematic traveling salesman problem and the kinematic orienteering problem. As we show in the computational results, solving the KTSP and the KOP to optimality with a commercial solver such as Gurobi becomes computationally intractable for larger problem instances. Hence, we developed a heuristic solver for both, which is based on the adaptive large neighborhood search. The results show, that the KTSP and the KOP offer a significant advantage compared to related problem formulations from the state-of-the-art and that our ALNS is capable of yielding high-quality solutions in a short time.

4.2. Future Work and Outlook

The findings of this work indicate that a proper consideration of the kinematic properties for route planning of a physical system is possible and offers lots of potential improvements compared to state-of-the-art approaches. However, this work represents just the entry into a highly relevant and interesting new research area that combines the classical vehicle routing domain from operations research with the motion planning domain from robotics. Having a look at the work on UAV routing conducted in this thesis, lots of promising future research can be done. As indicated in Sections 3.4.5.3 and 3.4.6.2, the computational complexity for solving the KTSP and the KOP to its global optimum with a commercial solver is very time-consuming. Hence, one future research direction could be to design problem-specific mathematical solvers based on e.g. branch-and-cut or branch-and-price, that are capable of solving larger problem instances in less time. The same holds for heuristic solvers for the ALNS. Other metaheuristic frameworks could be applied to solve the KTSP and the KOP such as Variable Neighborhood Search (VNS), Local Search (LS), or Genetic Algorithms (GA), to get a better understanding of which approach is suited best to solve even larger KTSP or KOP instances consistently with high quality in short time.

Apart from the future research direction directly related to the KTSP and KOP defined in this work, there are extensions considerable. For example, UAVs move in three dimensions. Hence the KOP and KTSP could be extended to 3-dimensional problem instances. As we showed in Section 2.5 our trajectory planners TOP-UAV and TOP-UAV++ are already capable of calculating three-dimensional time optimal motions. When having a look at the model for the KTSP and the KOP in Sections 3.2.1.2 and 3.2.2.2, an extension to 3D would require two additional indices for the azimuthal heading angle for the start and end waypoints in addition to the polar heading angles already defined. Such a model would be highly beneficial in uneven terrain when a constant altitude above ground level is required and hence the waypoints are at different altitudes above mean sea level. Further, in many real-world applications for surveillance and data collection, a fixed set of obstacles must be considered. This would require extending our TOP-UAV and TOP-UAV++ trajectory planners to consider these obstacles. In general, the calculation of obstacle-avoiding trajectories is much more complex than calculating trajectories in an obstacle-free environment. However, there are some capable and efficient approaches. For example, one lately published framework that could be used is based on rapidly exploring random trees (RRT), and also incorporates time-optimal trajectories with constrained maximum acceleration [121]. The next promising research direction lies in the extension of the KTSP and the KOP model to consider teams of homogenous multirotor UAVs with the same kinematic properties, teams of heterogenous multirotor UAVs with different kinematic properties, but also teams of heterogenous UAVs of multirotor and fixed-wing UAVs with different kinematic properties. Apart from these extensions, it is to be mentioned that the KTSP and the KOP are rather basic concepts that show a potential benefit when considering the kinematics of a system properly. There are numerous application-specific models to be investigated, e.g. time-window constraints, multiple depots, and many more.

Another promising research direction is to investigate the impact of kinematic routing models in non-UAV applications. For example in additive manufacturing such as laser powder-bed fusion. One of potentially multiple micro-mirrors is required to conduct a rotational route such that a laser fuses the metal powder at specific locations. Although mirrors can be adjusted very fast, these mirrors are still inert (see [122, 123]) and the angle positions of the mirrors that correspond to a specific position in the metal powder plane are very close. Moreover, in such use cases, the velocity is constrained since the power of the laser is limited and the metal must absorb enough energy to fuse. Hence, kinematic routing models have the potential to come at a reduction of the printing time. Especiall in such applications it would be necessary to modify our TOP-UAV trajectory planner to be able to deal with the rotational motion of the mirrors rather than the translational motion of the projected waypoint in the powder-bed plane.

Apart from laser powder-bed fusion, also extrusion-based additive manufacturing applications could benefit from kinematic routing formulations in terms of required printing time. This potential benefit is indicated e.g. in [124, 125, 126, 127], as such systems are constrained by the maximum acceleration of the extruder as well as a constraint in the maximum velocity due to a maximum feed rate of the extruder with 3D printing material.

For our time-optimal trajectory planning framework there are also numerous promising extensions considerable. In general, our TOP-UAV and TOP-UAV++ trajectory planners for two- and three-dimensional translational motion are very mature for symmetric maximum velocities and acceleration distribution on a single axis, i.e. $-v_{max} \leq v(t) \leq v_{max}$ and $-a_{max} \leq a(t) \leq a_{max}$. One further immediate extension of our trajectory planner could be to investigate the effect of higher numbers of configurations to share the velocity and acceleration norms among the axes and potentially find a trade-off between required computation time and improvement of the motion. Further, the consideration of unsymmetric maximum velocities and accelerations for a single axis, i.e. $v_{min} \leq v(t) \leq v_{max}$ and $a_{min} \leq a(t) \leq a_{max}$, and especially their behavior when coupled with the remaining axis in multi-dimensional time-optimal trajectory planning. In this case, an extended evaluation of the feasibility of the resulting solutions and optimality proofs must be conducted as done in the symmetric case in Section 2.4.2. This helps to better consider e.g. the gravitational force in time-optimal trajectory planning and subsequently in inertia-based UAV routing applications. Another possible extension of our trajectory planner is to simultaneously consider rotational and translational motion. Such trajectories could help in surveillance and data collection applications for multirotor UAVs when not only the waypoints are given, but also the orientation of the sensor to capture a scene is specified.

A. Hover-to-Hover Trajectories

Hover-to-hover trajectories represent the time-optimal motion between any two spatial waypoints, whereas the motion has to start at rest at the start waypoint and end at rest at the end waypoint. Further, the overall motion is restricted by a maximum allowed velocity and maximum allowed acceleration. Although hover-to-hover trajectories are comparably simple to calculate, they are a kinematically feasible approach and yet used to compute UAV missions for surveillance and data collection (see [11]). In Algorithm 6, we present the algorithm to calculate the duration and the associated acceleration pattern as control input used in this work. In analogy to Equation 2.5, the acceleration pattern (a_1, a_2, a_3) is described by the segments times t_1, t_2, and t_3, whereas t_i specifies the duration the acceleration a_i is applied. Since hover-to-hover trajectories start at rest, they always implement a bang-zero-bang control input pattern of the form $(+\hat{a}_{max}, 0, -\hat{a}_{max})$ with $t_1 = t_3$. In the following, we describe the algorithm.

Algorithm 6: Hover-2-Hover

1 **Input:**
2 - Distance between waypoints: *dist*
3 - Maximum velocity: $\hat{v}_{max}$
4 - Maximum acceleration: $\hat{a}_{max}$
5 $threshold\_time \leftarrow 2 \cdot \hat{v}_{max}/\hat{a}_{max}$
6 $threshold\_dist \leftarrow \hat{a}_{max} \cdot (threshold\_time/2)^2$
7 **if** $dist < threshold\_dist$ **then**
8 $t_1, t_3 \leftarrow \sqrt{dist/\hat{a}_{max}}$
9 $t_2 \leftarrow 0$
10 **else**
11 $t_1, t_3 \leftarrow threshold\_time/2$
12 $t_2 \leftarrow (dist - threshold\_dist)/\hat{v}_{max}$
13 **return:** t_1, t_2, t_3

First, we calculate the time *threshold_time* that would be needed to accelerate from rest to $\hat{v}_{max}$ and decelerate to rest again in line 5. Next, we calculate the distance *threshold_dist* in line 6 that would be traversed in this case. That is, when starting at rest and accelerating with $\hat{a}_{max}$ until $\hat{v}_{max}$ is reached, and, immediately at this point, decelerating with $\hat{a}_{max}$ until the rest state is reached again.

This information is used to identify if the hover-to-hover trajectory reaches $\hat{v}_{max}$ for the given input arguments, or not. If the input argument *dist* is less than *threshold_dist*, then

$\hat{v}_{max}$ is not reached and $t_2 = 0$ follows. Further, $t_1, t_3 = \sqrt{dist/\hat{a}_{max}}$ are obtained. This case can be seen in lines 7 - 8.

On the other hand, if *dist* is greater than or equal to *threshold_dist*, then $\hat{v}_{max}$ is reached. In this case, t_1 and t_3 are assigned $threshold\_dist/2$ as it represents the duration for which $\hat{a}_{max}$ must be applied to reach $\hat{v}_{max}$ from starting at rest. This implies that during the acceleration phases, the distance *threshold_dist* is covered and the trajectory remains for the distance (*dist* - *threshold_dist*) at velocity $\hat{v}_{max}$ from which t_2 is derived. The associated assignments for t_1, t_2, and t_3 are given in lines 11 and 12.

In the end, t_1, t_2, and t_3 are returned. Note that the duration of the resulting hover-to-hover trajectory can be calculated as $T_{H2H} = t_1 + t_2 + t_3$.

B. Trajectory Tracking in 2D

Within this work, we utilize two different MPC-based approaches, namely the 'full-kinematic system model' and the 'restricted-kinematic system model'. We use the 'full-kinematic system model' to simulate a real UAV with a maximum allowed velocity $\hat{v}_{max}$ and a maximum allowed acceleration of $\hat{a}_{max}$ in any direction. On the other hand, we use to 'restricted-kinematic system model' for the validation of the SOTA approach presented in Section 2.2.2.2 as it allows us to consider the same kinematic properties, i.e. the decoupled maximum allowed velocity v_{max} and acceleration a_{max} per axis. In the following, we describe both approaches.

B.1. Full-Kinematic System Model

The kinematic system model covering the multirotor UAV's full kinematics with a maximum allowed velocity $\hat{v}_{max}$ and acceleration $\hat{a}_{max}$ in any direction is given as follows:

$$\underbrace{\begin{bmatrix} p_x(k+1) \\ p_y(k+1) \\ v_x(k+1) \\ v_y(k+1) \end{bmatrix}}_{\boldsymbol{x}(k+1)} = \underbrace{\begin{bmatrix} 1 & 0 & \Delta T & 0 \\ 0 & 1 & 0 & \Delta T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\boldsymbol{A}} \cdot \underbrace{\begin{bmatrix} p_x(k) \\ p_y(k) \\ v_x(k) \\ v_y(k) \end{bmatrix}}_{\boldsymbol{x}(k)} + \underbrace{\begin{bmatrix} \frac{\Delta T^2}{2} & 0 \\ 0 & \frac{\Delta T^2}{2} \\ \Delta T & 0 \\ 0 & \Delta T \end{bmatrix}}_{\boldsymbol{B}} \cdot \underbrace{\begin{bmatrix} a_x(k) \\ a_y(k) \end{bmatrix}}_{\boldsymbol{u}(k)} \tag{B.1}$$

Equation (B.1) describes a linear, time-discrete and time-invariant kinematic system model of a UAV and defines how the application of acceleration power due to rotor thrust affects the velocity and finally the position of the UAV. In each time step k, it is assumed that the control input $\boldsymbol{u}(k)$ is constant for the entire time step length ΔT.

With this model, we can state the MPC for the *full-kinematic system model* as optimal control problem as follows:

$$\min_{\boldsymbol{u}(k)} \quad (\boldsymbol{x}(N) - \boldsymbol{x}_r(N))^\top \boldsymbol{Q}_N (\boldsymbol{x}(N) - \boldsymbol{x}_r(N)) \tag{B.2a}$$

$$\sum_{k=0}^{N-1} (\boldsymbol{x}(k) - \boldsymbol{x}_r(k))^\top \boldsymbol{Q} (\boldsymbol{x}(k) - \boldsymbol{x}_r(k)) \tag{B.2b}$$

$$\sum_{k=0}^{N-1} \boldsymbol{u}(k)^\top \boldsymbol{R} \boldsymbol{u}(k) \tag{B.2c}$$

$$s.t.$$

$$\boldsymbol{x}(0) = \boldsymbol{x}_s \tag{B.2d}$$

$$\boldsymbol{x}(k+1) = \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{B}\boldsymbol{u}(k),\ \forall k = 0, ..., N-1 \tag{B.2e}$$

$$\boldsymbol{x}^\top(k) \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \boldsymbol{x}(k) \leq v_{max}^2,\ \forall k = 0, ..., N \tag{B.2f}$$

$$\boldsymbol{u}^\top(k) \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \boldsymbol{u}(k) \leq a_{max}^2\ \forall k = 0, ..., N-1 \tag{B.2g}$$

$$\boldsymbol{x}(k) \in \mathbb{R}^4,\ \forall k = 0, ..., N \tag{B.2h}$$

$$\boldsymbol{u}(k) \in \mathbb{R}^2,\ \forall k = 0, ..., N-1 \tag{B.2i}$$

While the weight matrices $\boldsymbol{Q}_N, \boldsymbol{Q}$, and $\boldsymbol{R}$ are defined as

$$\boldsymbol{Q}_N = \begin{bmatrix} 1000 & 0 & 0 & 0 \\ 0 & 1000 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix} \tag{B.3}$$

$$\boldsymbol{Q} = \begin{bmatrix} 100 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{B.4}$$

$$\boldsymbol{R} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{B.5}$$

for this work.

As can be seen in the Equations (B.2), we first specify the objective. Given the discrete number of time steps and a reference trajectory $\boldsymbol{x}_r(k), k = 0, ..., N$, MPC applies a term penalizing the deviation from the desired final state as given in Equation (B.2a) which is also known as Mayer-term. Further, the deviation along the reference trajectory is also penalized for each time step $k = 0, ..., N-1$ (see Equation (B.2b)) as well as the applied control input (see Equation (B.2c)). Both are referred to as Lagrange-term. In this work, we do not consider a penalization of the applied control input by the $\boldsymbol{R}$ as specified in Equation (B.5). Penalizing the control input would allow a trade-off between the precision of the trajectory tracking and the application of control input values that do not correspond to the maximum allowed acceleration. This is not in line with the concept of time-optimality.

Moreover, the optimal control problem in (B.2) is constrained by a set of inequations. First, the initial state $\boldsymbol{x}(0)$ is defined that specifies the start position and velocity along the x and y coordinate axis. Second, the kinematic system model must hold for each time step $k = 0, ..., N-1$. Then, the norm of the velocity and acceleration at each timestep $k = 0, ..., N$ are constrained by $\hat{v}_{max}$ and $\hat{a}_{max}$ respectively as defined in Inequations (B.2f)

and (B.2g). Last, the state $\boldsymbol{x}$ is defined to be an element of $\mathbb{R}^4$ and the control input $\boldsymbol{u}$ is an element of $\mathbb{R}^2$.

We solve the nonlinear mathematical optimization problem defined (B.2) using Gurobi 10.0.1 implemented in Python 3.9. The number of time steps N is derived from the yielded synchronization time T_{sync} and the time step length ΔT as $N = \lfloor T_{\text{sync}}/\Delta T \rfloor$.

B.2. Restricted-Kinematic System Model

The SOTA approach defined in Section 2.2.2.2 as well as our basic TOP-UAV trajectory planner is based on the decoupling of the axis, where each axis is assigned a specific maximum velocity v_{max} and acceleration a_{max}. To check the feasibility of both approaches, we design an MPC-based trajectory planner, that considers the same kinematics as both approaches including the decoupled axes. Hence it is perfectly suited for validation.

Due to the decoupling approach, the system model considered for this MPC covers the kinematics of only a single axis. The corresponding time-discrete, linear and time-invariant system model is given as follows:

$$\underbrace{\begin{bmatrix} p(k+1) \\ v(k+1) \end{bmatrix}}_{\boldsymbol{x}(k+1)} = \underbrace{\begin{bmatrix} 1 & \Delta T \\ 0 & 1 \end{bmatrix}}_{\boldsymbol{A}} \cdot \underbrace{\begin{bmatrix} p(k) \\ v(k) \end{bmatrix}}_{\boldsymbol{x}(k)} + \underbrace{\begin{bmatrix} \frac{\Delta T^2}{2} \\ \Delta T \end{bmatrix}}_{\boldsymbol{B}} \cdot \underbrace{\begin{bmatrix} a(k) \end{bmatrix}}_{\boldsymbol{u}(k)} \tag{B.6}$$

As can be seen, the control input $\boldsymbol{u}(k)$ defines a scalar value that represents the applied acceleration that is applied in time step k for the duration ΔT on the axis. This acceleration affects the velocity and the position in the next time step $k+1$. Further, the velocity in time step k affects the position of time step $k+1$.

The resulting optimal control problem can be seen below:

$$\min_{\boldsymbol{u}(k)} \quad (\boldsymbol{x}(N) - \boldsymbol{x}_r(N))^\top \boldsymbol{Q}_N (\boldsymbol{x}(N) - \boldsymbol{x}_r(N)) \tag{B.7a}$$

$$\sum_{k=0}^{N-1} (\boldsymbol{x}(k) - \boldsymbol{x}_r(k))^\top \boldsymbol{Q} (\boldsymbol{x}(k) - \boldsymbol{x}_r(k)) \tag{B.7b}$$

$$\sum_{k=0}^{N-1} \boldsymbol{u}(k)^\top \boldsymbol{R} \boldsymbol{u}(k) \tag{B.7c}$$

$$s.t.$$

$$\boldsymbol{x}(0) = \boldsymbol{x}_s \tag{B.7d}$$

$$\boldsymbol{x}(k+1) = \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{B}\boldsymbol{u}(k),\ \forall k = 0, ..., N-1 \tag{B.7e}$$

$$-v_{max} \leq \begin{bmatrix} 0 & 1 \end{bmatrix} \boldsymbol{x}(k) \leq v_{max},\ \forall k = 0, ..., N \tag{B.7f}$$

$$-a_{max} \leq \boldsymbol{u}(k) \leq a_{max},\ \forall k = 0, ..., N-1 \tag{B.7g}$$

$$\boldsymbol{x}(k) \in \mathbb{R}^2,\ \forall k = 0, ..., N \tag{B.7h}$$
$$\boldsymbol{u}(k) \in \mathbb{R},\ \forall k = 0, ..., N-1 \tag{B.7i}$$

Analogously to Section B.1, the general form of the optimal control problem contains a Mayer term that penalizes the deviation between the required and the achieved final state after N time steps. It also contains the Lagrange term that penalizes deviation from the reference trajectories and the applied control input. Since we focus on time-optimal trajectories in this work and the SOTA approach defined in Section 2.2.2.2 does not yield the entire reference but only the optimum trajectory duration, we are only interested in the Mayer-term and neglect the Lagrange term. This is done by setting the weight matrices $\boldsymbol{Q}_N, \boldsymbol{Q}$, and $\boldsymbol{R}$ as follows:

$$\boldsymbol{Q}_N = \begin{bmatrix} 1000 & 0 \\ 0 & 1000 \end{bmatrix}, \tag{B.8}$$
$$\boldsymbol{Q} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \tag{B.9}$$
$$\boldsymbol{R} = \begin{bmatrix} 0 \end{bmatrix}. \tag{B.10}$$

Further, the resulting optimal control problem is constrained by the initial state in Inequation (B.7d), by the considered restricted-kinematic system model in each time step as (see Equations (B.7d)), the maximum velocity and acceleration bounds for each time step (see Inequations (B.7f) and (B.7g)) and the domains of $\boldsymbol{x}(k)$ and $\boldsymbol{u}(k)$ (see Equations (B.7h) and (B.7i)).

We solve the mathematical program defined (B.7) for each considered coordinate axis using Gurobi 10.0.1 implemented in Python 3.9. The number of time steps N is derived from the yielded synchronization time T_{sync} and the time step length ΔT as $N = \lfloor T_{\text{sync}} / \Delta T \rfloor$.

C. Original Computational Results

Inst.	Scale	TSP-classic		TSP-H2H		DTSP		KTSP		
		$\mathcal{M}_T$	MAE	$\mathcal{M}_T$	MAE	$\mathcal{M}_T$	MAE	$\mathcal{M}_T$	Gap^{MIP}	MAE
TSI1	0.25	6.76	1.480	40.37	0.001	39.88	0.002	27.99	0.00	0.001
TSI1	0.50	13.53	1.264	57.10	0.001	54.88	0.002	39.37	0.00	0.001
TSI1	1.00	27.06	1.830	80.75	0.002	76.66	0.004	57.47	2.44	0.001
TSI1	2.00	54.12	2.706	115.05	0.005	108.26	0.006	87.77	2.04	0.000
TSI1	4.00	108.23	4.237	171.75	0.007	150.01	0.009	143.81	0.51	0.000
TSI2	0.25	3.83	1.100	24.47	0.001	24.36	0.001	17.09	0.00	0.001
TSI2	0.50	7.67	1.976	34.62	0.001	34.17	0.002	23.86	0.00	0.001
TSI2	1.00	15.33	2.858	48.96	0.002	48.72	0.003	34.03	0.00	0.001
TSI2	2.00	30.66	2.166	69.67	0.005	68.34	0.023	51.84	0.00	0.000
TSI2	4.00	61.32	3.122	102.91	0.008	97.44	0.007	83.23	0.00	0.000
TSI3	0.25	8.13	3.408	44.55	0.001	40.83	0.002	29.31	0.00	0.001
TSI3	0.50	16.26	4.291	63.01	0.001	57.86	0.003	41.85	3.18	0.000
TSI3	1.00	32.51	4.086	89.14	0.003	89.14	0.006	61.31	0.00	0.001
TSI3	2.00	65.02	22.175	128.05	0.003	114.90	0.010	94.22	0.00	0.001
TSI3	4.00	130.05	7.106	195.38	0.006	164.92	0.009	164.86	0.37	0.000

Table C.1.: Mission duration and MAE tracking error for optimal solutions of TSP-classic, TSP-H2H, DTSP, and KTSP on the problem instances presented in Section 3.4.2.1. For the KTSP, we additionally give the Gap^{MIP} since not all problem instances were solved to proven optimality due to the time limit of 18000 s.

L	TSP-classic			DTSP			KTSP		
	CT_{LB}	CT	CT_{UB}	CT_{LB}	CT	CT_{UB}	CT_{LB}	CT	CT_{UB}
10	0.00	0.00	0.01	0.21	10.39	38.26	9.82	85.53	284.77
12	0.00	0.01	0.02	0.30	3.41	12.31	4.95	16.22	27.28
14	0.00	0.01	0.02	0.32	6.83	18.11	6.76	363.29	1609.52
16	0.01	0.01	0.01	1.40	3.10	5.84	37.46	65.79	81.42
18	0.01	0.02	0.04	2.65	42.88	159.44	44.03	1352.06	3600.00
20	0.01	0.03	0.06	4.26	10.78	28.66	131.33	575.39	1705.34
22	0.01	0.03	0.05	2.07	72.04	314.67	137.23	1112.57	1924.09
24	0.04	0.05	0.07	7.10	26.97	65.02	772.39	2028.60	3600.00
26	0.02	0.06	0.11	5.43	20.29	32.37	150.20	1139.73	3600.00
28	0.05	0.07	0.09	11.28	28.09	55.08	2592.10	3398.42	3600.00
30	0.07	0.12	0.21	48.19	324.91	1171.91	1462.61	2857.08	3600.00
32	0.06	0.13	0.19	8.68	530.61	2423.46	3083.45	3424.04	3600.00
34	0.02	0.10	0.17	12.28	336.84	1268.42	3600.00	3600.00	3600.00
36	0.07	0.12	0.25	45.90	597.42	1742.12	3600.00	3600.00	3600.00
38	0.04	0.13	0.23	6.60	1273.25	3600.00	3600.00	3600.00	3600.00
40	0.03	0.07	0.13	71.53	214.01	508.49	3600.00	3600.00	3600.00

Table C.2.: To evaluate the lowest (CT_{LB}), average (CT) and highest (CT_{UB}) computation times for different TSP variants, we use the problem instances presented in Section 3.4.2.3. The computation time limit is set to 3600 s. The number of allowed heading angles for DTSP and KTSP are $H = 8$. The number of allowed traversal velocities for the KTSP is set to $V = 6$.

Inst.	Scale	KTSP-exact			KTSP-ALNS$_{30\,s}$(%)			KTSP-ALNS$_{60\,s}$(%)			KTSP-ALNS$_{120\,s}$(%)		
		$\mathcal{M}_T$	Gap$^{\text{MIP}}$	CPU (s)	Gap$^{\text{ALNS}}_{\text{LB}}$	Gap$^{\text{ALNS}}$	Gap$^{\text{ALNS}}_{\text{UB}}$	Gap$^{\text{ALNS}}_{\text{LB}}$	Gap$^{\text{ALNS}}$	Gap$^{\text{ALNS}}_{\text{UB}}$	Gap$^{\text{ALNS}}_{\text{LB}}$	Gap$^{\text{ALNS}}$	Gap$^{\text{ALNS}}_{\text{UB}}$
TSI1	0.25	27.99	0.00	7604	5.18	6.04	7.50	3.47	5.50	7.68	3.00	5.03	6.54
TSI1	0.50	29.37	0.00	15380	2.44	4.30	5.82	1.98	3.96	6.20	1.73	3.58	5.31
TSI1	1.00	57.47	2.44	18000	1.27	3.15	4.21	1.48	2.31	3.22	1.22	2.38	3.43
TSI1	2.00	87.77	2.04	18000	0.13	1.37	1.93	0.00	0.90	1.32	0.22	1.03	1.40
TSI1	4.00	143.81	0.51	18000	0.00	0.31	1.44	0.00	0.01	0.13	0.00	0.01	0.13
TSI2	0.25	17.09	0.00	212	0.18	0.53	0.94	0.18	0.53	0.94	0.00	0.35	1.46
TSI2	0.50	23.86	0.00	529	0.00	0.46	1.01	0.00	0.21	0.96	0.00	0.29	1.30
TSI2	1.00	34.03	0.00	1724	0.00	0.59	1.15	0.00	0.35	0.73	0.00	0.24	0.73
TSI2	2.00	51.84	0.00	3078	0.00	0.23	0.79	0.00	0.33	0.79	0.00	0.17	0.79
TSI2	4.00	83.23	0.00	4987	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
TSI3	0.25	29.31	0.00	7204	4.03	6.07	7.54	2.97	5.36	7.68	3.48	4.81	6.04
TSI3	0.50	41.85	3.18	18000	3.08	4.76	6.71	1.24	3.54	5.90	1.36	3.39	4.95
TSI3	1.00	61.31	0.00	8910	3.02	3.59	4.48	1.65	2.74	4.13	0.00	1.96	3.36
TSI3	2.00	97.22	0.00	17403	0.87	1.24	2.01	0.71	1.09	1.39	0.14	0.84	1.39
TSI3	4.00	164.86	0.37	18000	0.39	0.64	1.27	0.32	0.39	0.55	0.39	0.39	0.39

Table C.3.: Performance evaluation of the ALNS against solutions yielded by solving the KTSP exactly with Gurobi with a computation time limit 18000 s. The ALNS is evaluated given 30 s, 60 s and 120 s as computation time limit and run ten times with different random seeds on each problem instance. For each computation time limit the table shows the best, average and worst yielded gap to the best found solution obtained from Gurobi.

C_{max}	KOP			KOP-ALNS$_{30\,s}$			KOP-ALNS$_{60\,s}$			KOP-ALNS$_{120\,s}$		
	$\mathcal{M}_P$	Gap^{MIP}	CT	$\overline{\mathcal{M}_P}$	$\mathcal{M}_P$	$\underline{\mathcal{M}_P}$	$\overline{\mathcal{M}_P}$	$\mathcal{M}_P$	$\underline{\mathcal{M}_P}$	$\overline{\mathcal{M}_P}$	$\mathcal{M}_P$	$\underline{\mathcal{M}_P}$
5	10	0.00	129.21	10	10.0	10	10	10.0	10	10	10.0	10
10	40	37.50	18000.00	35	35.0	35	35	35.0	35	35	35.0	35
15	55	63.64	18000.00	75	71.5	70	75	71.0	70	75	72.5	70
20	100	20.00	18000.00	100	97.0	90	105	99.5	90	105	99.5	95
25	125	20.00	18000.00	130	125.0	120	130	125.5	120	130	128.5	125
30	170	0.00	13095.55	165	155.5	150	165	159.0	155	165	161.0	155
35	200	0.00	12283.97	195	191.5	185	195	194.0	190	195	193.0	185
40	220	2.25	18000.00	215	212.0	205	215	213.5	210	215	213.5	210
45	240	2.08	18000.00	235	230.0	225	235	231.5	230	235	230.5	230
50	250	8.00	18000.00	250	248.0	240	255	250.0	245	255	252.0	250
55	270	5.56	18000.00	270	267.5	265	270	270.0	270	270	268.5	265
60	280	1.79	18000.00	285	282.5	280	285	281.0	280	285	283.5	280
65	280	1.79	18000.00	285	285.0	285	285	285.0	285	285	285.0	285
70	285	0.00	3531.04	285	285.0	285	285	285.0	285	285	285.0	285

Table C.4.: Tsiligirides dataset 1. Performance evaluation of the ALNS against solutions yielded by solving the KOP exactly with Gurobi with a computation time limit of 18000 s on Tsiligirides dataset 1. The ALNS is evaluated given 30 s, 60 s and 120 s as a computation time limit and ran ten times with different random seeds on each problem instance. For each time limit the table shows the best, average and worst collected priorities.

C_{max}	KOP			KOP-ALNS$_{30\,s}$			KOP-ALNS$_{60\,s}$			KOP-ALNS$_{120\,s}$		
	$\mathcal{M}_P$	Gap^{MIP}	CT	$\overline{\mathcal{M}_P}$	$\mathcal{M}_P$	$\underline{\mathcal{M}_P}$	$\overline{\mathcal{M}_P}$	$\mathcal{M}_P$	$\underline{\mathcal{M}_P}$	$\overline{\mathcal{M}_P}$	$\mathcal{M}_P$	$\underline{\mathcal{M}_P}$
15	190	10.53	18000.00	190	185.5	175	190	188.5	175	190	190.0	190
20	270	0.00	9797.15	260	251.0	240	265	254.0	245	270	263.0	255
23	325	0.00	741.65	315	298.5	290	310	301.0	295	315	309.0	300
25	355	0.00	508.11	350	342.5	335	355	343.0	335	355	345.5	340
27	380	0.00	253.19	380	373.5	365	375	373.0	365	380	375.5	370
30	410	0.00	6822.06	400	399.5	395	410	403.0	400	400	400.0	400
32	430	4.65	18000.00	430	421.0	420	430	422.0	420	430	421.0	420
35	450	0.00	4994.18	450	450.0	450	450	450.0	450	450	450.0	450
38	450	0.00	893.50	450	450.0	450	450	450.0	450	450	450.0	450

Table C.5.: Tsiligirides dataset 2. Performance evaluation of the ALNS against solutions yielded by solving the KOP exactly with Gurobi with a computation time limit of 18000 s on Tsiligirides dataset 2. The ALNS is evaluated given 30 s, 60 s and 120 s as a computation time limit and ran ten times with different random seeds on each problem instance. For each time limit the table shows the best, average and worst collected priorities.

C_{max}	KOP			KOP-ALNS$_{30\,s}$			KOP-ALNS$_{60\,s}$			KOP-ALNS$_{120\,s}$		
	$\mathcal{M}_P$	Gap^{MIP}	CT	$\overline{\mathcal{M}_P}$	$\mathcal{M}_P$	$\underline{\mathcal{M}_P}$	$\overline{\mathcal{M}_P}$	$\mathcal{M}_P$	$\underline{\mathcal{M}_P}$	$\overline{\mathcal{M}_P}$	$\mathcal{M}_P$	$\underline{\mathcal{M}_P}$
15	220	0.00	7497.43	220	209.0	200	220	216.0	200	220	218.0	200
20	330	0.00	1954.54	320	315.0	310	320	315.0	310	330	314.0	310
25	410	0.00	1726.59	400	393.0	390	400	393.0	390	400	396.0	390
30	490	0.00	806.89	480	464.0	450	480	462.0	450	480	472.0	460
35	550	0.00	2613.76	530	522.0	510	530	529.0	520	530	526.0	520
40	600	0.00	7826.57	590	579.0	570	590	577.0	560	590	579.0	570
45	660	0.00	3022.42	650	633.0	610	640	632.0	620	650	634.0	620
50	710	1.41	18000.00	700	692.0	680	700	695.0	680	710	699.0	690
55	760	0.00	5240.66	750	739.0	730	750	746.0	730	750	746.0	740
60	780	2.56	18000.00	780	777.0	770	780	775.0	770	780	778.0	770
65	790	1.27	18000.00	800	800.0	800	800	800.0	800	800	800.0	800
70	800	0.00	3193.12	800	800.0	800	800	800.0	800	800	800.0	800

Table C.6.: Tsiligirides dataset 3. Performance evaluation of the ALNS against solutions yielded by solving the KOP exactly with Gurobi with a computation time limit of 18000 s on Tsiligirides dataset 3. The ALNS is evaluated given 30 s, 60 s and 120 s as a computation time limit and ran ten times with different random seeds on each problem instance. For each time limit the table shows the best, average and worst collected priorities.

C_{max}	MPC [88]		KOP-ALNS$_{30\,s}$			KOP-ALNS$_{60\,s}$			KOP-ALNS$_{120\,s}$		
	$\overline{\mathcal{M}_P}$	CT	$\overline{\mathcal{M}_P}$	$\mathcal{M}_P$	$\underline{\mathcal{M}_P}$	$\overline{\mathcal{M}_P}$	$\mathcal{M}_P$	$\underline{\mathcal{M}_P}$	$\overline{\mathcal{M}_P}$	$\mathcal{M}_P$	$\underline{\mathcal{M}_P}$
10	115.0	1.7	105.0	102.0	100.0	105.0	103.0	100.0	105.0	104.0	100.0
15	200.0	2.8	210.0	199.5	195.0	210.0	198.5	195.0	210.0	203.0	195.0
20	260.0	8.7	295.0	290.0	285.0	295.0	291.0	285.0	305.0	296.5	290.0
25	370.0	11.4	390.0	385.5	375.0	390.0	388.5	375.0	390.0	389.0	385.0
30	450.0	15.9	440.0	435.0	430.0	440.0	438.0	430.0	440.0	440.0	440.0
35	450.0	22.8	450.0	450.0	450.0	450.0	450.0	450.0	450.0	450.0	450.0
40	450.0	45.7	450.0	450.0	450.0	450.0	450.0	450.0	450.0	450.0	450.0

Table C.7.: Performance evaluation of the ALNS against best solutions yielded by the MPC approach from [88] on Tsiligirides dataset 2. The ALNS using our TOP-UAV trajectory planner is evaluated given 30 s, 60 s and 120 s as computation time limit and ran ten times with different random seeds on each problem instance.

C_{max}	MPC [88]		KOP-ALNS$_{30\,s}$			KOP-ALNS$_{60\,s}$			KOP-ALNS$_{120\,s}$		
	$\overline{\mathcal{M}_P}$	CT	$\overline{\mathcal{M}_P}$	$\mathcal{M}_P$	$\underline{\mathcal{M}_P}$	$\overline{\mathcal{M}_P}$	$\mathcal{M}_P$	$\underline{\mathcal{M}_P}$	$\overline{\mathcal{M}_P}$	$\mathcal{M}_P$	$\underline{\mathcal{M}_P}$
10	115.0	1.7	115.0	114.5	110.0	120.0	116.0	115.0	120.0	117.0	115.0
15	200.0	2.8	220.0	208.5	195.0	220.0	209.0	205.0	220.0	211.0	205.0
20	260.0	8.7	310.0	302.5	300.0	305.0	300.5	295.0	315.0	308.5	300.0
25	370.0	11.4	400.0	396.0	390.0	400.0	399.0	390.0	400.0	397.5	390.0
30	450.0	15.9	450.0	450.0	450.0	450.0	450.0	450.0	450.0	450.0	450.0
35	450.0	22.8	450.0	450.0	450.0	450.0	450.0	450.0	450.0	450.0	450.0
40	450.0	45.7	450.0	450.0	450.0	450.0	450.0	450.0	450.0	450.0	450.0

Table C.8.: Performance evaluation of the ALNS against best solutions yielded by the MPC approach from [88] on Tsiligirides dataset 2. The ALNS using our TOP-UAV++ trajectory planner is evaluated given 30 s, 60 s and 120 s as computation time limit and ran ten times with different random seeds on each problem instance.

List of Figures

List of Tables

Glossary

Abbreviation	Meaning
ALNS	Adaptive large neighborhood search
DOP	Dubins orienteering problem
DOPN	Dubins orienteering problem with neighborhoods
DP	Dynamic programming
DTSP	Dubins traveling salesman problem
GA	Genetic algorithm
H2H	Hover-to-hover
HTOL	Horizontal take-off and landing
KTSP	Kinematic traveling salesman problem
KOP	Kinematic orienteering problem
LS	Local Search
LNS	Large neighborhood search
LP	Linear program
MAE	Mean absolute error
MPC	Model-based predictive control
MIOCP	Mixed-integer optimal control problem
MIP	Mixed-integer program
OP	Orienteering problem
ROI	Region of interest
RRT	Rapidly exploring random tree
SA	Simulated annealing
SOTA	State-of-the-art
TOP-UAV	Time-optimal trajectory planner for point-masses under acceleration and velocity constraints
TOP-UAV++	Improved version of TOP-UAV
TOT-PMAV	Time-optimal trajectory planning problem for point-masses under acceleration and velocity constraints
TSP	Traveling salesman problem
UAV	Unmanned aerial vehicle
VNS	Variable neighborhood search
VTOL	Vertical take-off and landing

References

[1] J. A. Besada et al. „Drone Mission Definition and Implementation for Automated Infrastructure Inspection Using Airborne Sensors“. In: *Sensors* 18.4 (2018). ISSN: 1424-8220. DOI: https://doi.org/10.3390/s18041170.

[2] D. Mader et al. „Potential of UAV-based laser scanner and multispectral camera data in building inspection“. In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLI-B1 (2016), pp. 1135–1142. DOI: https://doi.org/10.5194/isprs-archives-XLI-B1-1135-2016.

[3] J. Keller et al. „Coordinated Path Planning for Fixed-Wing UAS Conducting Persistent Surveillance Missions“. In: *IEEE Transactions on Automation Science and Engineering* 14.1 (2017), pp. 17–24. DOI: https://doi.org/10.1109/TASE.2016.2623642.

[4] J. Schmidt and A. Fügenschuh. „Planning inspection flights with an inhomogenous fleet of micro aerial vehicles“. In: *Cottbus Mathematical Preprints* 21 (2021). DOI: https://doi.org/10.26127/BTUOpen-5656.

[5] A. Otto et al. „Optimization approaches for civil applications of unmanned aerial vehicles (UAVs) or aerial drones: A survey“. In: *Networks* 72.4 (2018), pp. 411–458. DOI: https://doi.org/10.1002/net.21818.

[6] J. del Cerro et al. „Unmanned Aerial Vehicles in Agriculture: A Survey“. In: *Agronomy* 11.2 (2021). ISSN: 2073-4395. DOI: https://doi.org/10.3390/agronomy11020203.

[7] D. C. Tsouros, S. Bibi, and P. G. Sarigiannidis. „A Review on UAV-Based Applications for Precision Aggriculture“. In: *Information* 10.11 (2019). DOI: https://doi.org/10.3390/info10110349.

[8] J. Shahmoradi et al. „A Comprehensive Review of Applications of Drone Technology in the Mining Industry“. In: *Drones* 4.3 (2020). ISSN: 2504-446X. DOI: https://doi.org/10.3390/drones4030034.

[9] X. Zheng, F. Wang, and Z. Li. „A multi-UAV cooperative route planning methodology for 3D fine-resolution building model reconstruction“. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 146 (2018), pp. 483–494. ISSN: 0924-2716. DOI: https://doi.org/10.1016/j.isprsjprs.2018.11.004.

[10] A. Restas. „Drone Applications for Supporting Disaster Management“. In: *World Journal of Engineering and Technology* 4 (2015), pp. 316–321. DOI: https://doi.org/10.4236/wjet.2015.33C047.

[11] K. Glock and A. Meyer. „Mission Planning for Emergency Rapid Mapping with Drones“. In: *Transportation Science* 54.2 (2020), pp. 534–560. DOI: https://doi.org/10.1287/trsc.2019.0963.

[12] J. Delmerico et al. „The current state and future outlook of rescue robotics“. In: *Journal of Field Robotics* 36.7 (2019), pp. 1171–1191. DOI: https://doi.org/10.1002/rob.21887.

[13] D. R. Viloria et al. „Unmanned aerial vehicles/drones in vehicle routing problems: a literature review“. In: *International Transactions in Operational Research* 28.4 (2021), pp. 1626–1657. DOI: https://doi.org/10.1111/itor.12783.

[14] M. Hassanalian and A. Abdelkefi. „Classifications, applications, and design challenges of drones: A review“. In: *Progress in Aerospace Sciences* 91 (2017), pp. 99–131. ISSN: 0376-0421. DOI: https://doi.org/10.1016/j.paerosci.2017.04.003.

[15] J. Kohns et al. „Innovative methods for earthquake damage detection and classification using airborne observation of critical infrastructures (project LOKI)“. In: *23rd EGU General Assembly* (2021). DOI: https://doi.org/10.5194/egusphere-egu21-2712.

[16] S. D. Apostolidis et al. „Cooperative multi-UAV coverage mission planning platform for remote sensing applications“. In: *Autonomous Robots* 46 (2022), pp. 373–400. DOI: https://doi.org/10.1007/s10514-021-10028-3.

[17] S. Xiao, X. Tan, and J. Wang. „A Simulated Annealing Algorithm and Grid Map-Based UAV Coverage Path Planning Method for 3D Reconstruction“. In: *Electronics* 10.7 (2021). DOI: https://doi.org/10.3390/electronics10070853.

[18] P. Tripicchio et al. „Smooth Coverage Path Planning for UAVs with Model Predictive Control Trajectory Tracking“. In: *Electronics* 12.10 (2023). DOI: https://doi.org/10.3390/electronics12102310.

[19] Z. Shang and Z. Shen. „Topology-based UAV path planning for multi-view stereo 3D reconstruction of complex structures“. In: *Complex and Intelligent Systems* 9.1 (2023), pp. 909–926. DOI: https://doi.org/10.1007/s40747-022-00831-5.

[20] M. Maboudi et al. „A Review on Viewpoints and Path Planning for UAV-Based 3-D Reconstruction“. In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 16 (2023), pp. 5026–5048. DOI: https://doi.org/10.1109/JSTARS.2023.3276427.

[21] E. Fountoulakis, G. S. Paschos, and N. Pappas. „UAV Trajectory Optimization for Time Constrained Applications“. In: *IEEE Networking Letters* 2.3 (2020), pp. 136–139. DOI: https://doi.org/10.1109/LNET.2020.3007310.

[22] R. Pěnička et al. „Dubins Orienteering Problem“. In: *IEEE Robotics and Automation Letters* 2.2 (2017), pp. 1210–1217. DOI: https://doi.org/10.1109/LRA.2017.2666261.

[23] R. Pěnička et al. „Dubins orienteering problem with neighborhoods“. In: *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2017, pp. 1555–1562. DOI: https://doi.org/10.1109/ICUAS.2017.7991350.

[24] S. Ropke and D. Pisinger. „An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows“. In: *Transportation Science* 40.4 (2006), pp. 455–472. DOI: https://doi.org/10.1287/trsc.1050.0135.

[25] F. Meyer and K. Glock. „Trajectory-based Traveling Salesman Problem for Multirotor UAVs“. In: *2021 17th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. 2021, pp. 335–342. DOI: https://doi.org/10.1109/DCOSS52077.2021.00061.

[26] F. Meyer and K. Glock. „Kinematic Orienteering Problem With Time-Optimal Trajectories for Multirotor UAVs“. In: *IEEE Robotics and Automation Letters* 7.4 (2022), pp. 11402–11409. DOI: https://doi.org/10.1109/LRA.2022.3194688.

[27] F. Meyer, K. Glock, and D. Sayah. „TOP-UAV: Open-Source Time-Optimal Trajectory Planner for Point-Masses Under Acceleration and Velocity Constraints“. In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2023, pp. 2838–2845. DOI: https://doi.org/10.1109/IROS55552.2023.10342270.

[28] P. Foehn, A. Romero, and D. Scaramuzza. „Time-optimal planning for quadrotor waypoint flight“. In: *Science Robotics* 6.56 (2021), eabh1221. DOI: https://doi.org/10.1126/scirobotics.abh1221.

[29] P. Foehn et al. „AlphaPilot: autonomous drone racing“. In: *Autonomous Robots* 46 (2022), pp. 307–320. DOI: https://doi.org/10.1007/s10514-021-10011-y.

[30] A. Romero, R. Penicka, and D. Scaramuzza. „Time-Optimal Online Replanning for Agile Quadrotor Flight“. In: *IEEE Robotics and Automation Letters* 7.3 (2022), pp. 7730–7737. DOI: https://doi.org/10.1109/LRA.2022.3185772.

[31] L. E. Dubins. „On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents“. In: *American Journal of Mathematics* 79.3 (1957), pp. 497–516. DOI: https://doi.org/10.2307/2372560.

[32] A. Bry et al. „Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments“. In: *International Journal of Robotics Research* 34 (7 2015), pp. 969–1002. DOI: https://doi.org/doi:10.1177/0278364914558129.

[33] M. Henchey and S. Rosen. „Emerging approaches to support dynamic mission planning: survey and recommendations for future research“. In: *The Journal of Defense Modeling and Simulation* 18.4 (2021), pp. 453–468. DOI: https://doi.org/10.1177/1548512919898750.

[34] P. Váňa et al. „Minimal 3D Dubins Path with Bounded Curvature and Pitch Angle“. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 8497–8503. DOI: https://doi.org/10.1109/ICRA40945.2020.9197084.

[35] K. Kučerová, P. Váňa, and J. Faigl. „On Finding Time-Efficient Trajectories for Fixed-Wing Aircraft Using Dubins Paths with Multiple Radii“. In: *Proceedings of the 35th Annual ACM Symposium on Applied Computing*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 829–831. ISBN: 9781450368667. DOI: https://doi.org/10.1145/3341105.3374112.

[36] J. Faigl and P. Váňa. „Surveillance Planning with Bézier Curves“. In: *IEEE Robotics and Automation Letters* 3.2 (2018), pp. 750–757. DOI: https://doi.org/10.1109/LRA.2018.2789844.

[37] J. Faigl, P. Váňa, and R. Pěnička. „Multi-Vehicle Close Enough Orienteering Problem with Bézier Curves for Multi-Rotor Aerial Vehicles“. In: (2019), pp. 3039–3044. DOI: https://doi.org/10.1109/ICRA.2019.8794339.

[38] F. Gao et al. „Optimal Time Allocation for Quadrotor Trajectory Generation“. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018, pp. 4715–4722. DOI: https://doi.org/10.1109/IROS.2018.8593579.

[39] F. Gao et al. „Optimal Trajectory Generation for Quadrotor Teach-and-Repeat“. In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 1493–1500. DOI: https://doi.org/10.1109/LRA.2019.2895110.

[40] F. Gao et al. „Online Safe Trajectory Generation for Quadrotors Using Fast Marching Method and Bernstein Basis Polynomial“. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 344–351. DOI: https://doi.org/10.1109/ICRA.2018.8462878.

[41] F. Stoican et al. „Constrained trajectory generation for UAV systems using a B-spline parametrization“. In: *2017 25th Mediterranean Conference on Control and Automation (MED)*. 2017, pp. 613–618. DOI: https://doi.org/10.1109/MED.2017.7984185.

[42] P. Shen, X. Zhang, and Y. Fang. „Complete and Time-Optimal Path-Constrained Trajectory Planning With Torque and Velocity Constraints: Theory and Applications“. In: *IEEE/ASME Transactions on Mechatronics* 23.2 (2018), pp. 735–746. DOI: https://doi.org/10.1109/TMECH.2018.2810828.

[43] P. Shen et al. „Real-Time Acceleration-Continuous Path-Constrained Trajectory Planning With Built-In Tradeoff Between Cruise and Time-Optimal Motions“. In: *IEEE Transactions on Automation Science and Engineering* 17.4 (2020), pp. 1911–1924. DOI: https://doi.org/10.1109/TASE.2020.2980423.

[44] Q.-C. Pham. „A General, Fast, and Robust Implementation of the Time-Optimal Path Parameterization Algorithm“. In: *IEEE Transactions on Robotics* 30.6 (2014), pp. 1533–1540. DOI: https://doi.org/10.1109/TRO.2014.2351113.

[45] D. Mellinger and V. Kumar. „Minimum snap trajectory generation and control for quadrotors“. In: *2011 IEEE International Conference on Robotics and Automation*. 2011, pp. 2520–2525. DOI: https://doi.org/10.1109/ICRA.2011.5980409.

[46] C. Richter, A. Bry, and N. Roy. „Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments“. In: *Robotics Research: The 16th International Symposium ISRR*. Ed. by Masayuki Inaba and Peter Corke. Springer International Publishing, 2016, pp. 649–666. ISBN: 978-3-319-28872-7. DOI: https://doi.org/10.1007/978-3-319-28872-7_37.

[47] D. Burke, A. Chapman, and I. Shames. „Generating Minimum-Snap Quadrotor Trajectories Really Fast“. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, pp. 1487–1492. DOI: https://doi.org/10.1109/IROS45743.2020.9341794.

[48] F. Gao and S. Shen. „Online quadrotor trajectory generation and autonomous navigation on point clouds“. In: *2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. 2016, pp. 139–146. DOI: https://doi.org/10.1109/SSRR.2016.7784290.

[49] F. Gao, Y. Lin, and S. Shen. „Gradient-based online safe trajectory generation for quadrotor flight in complex environments“. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 3681–3688. DOI: https://doi.org/10.1109/IROS.2017.8206214.

[50] S. Liu et al. „Planning Dynamically Feasible Trajectories for Quadrotors Using Safe Flight Corridors in 3-D Complex Environments“. In: *IEEE Robotics and Automation Letters* 2.3 (2017), pp. 1688–1695. DOI: https://doi.org/10.1109/LRA.2017.2663526.

[51] Markos Papageorgiou, Marion Leibold, and Martin Buss. „Optimale Steuerung zeitdiskreter dynamischer Systeme“. In: *Optimierung: Statische, dynamische, stochastische Verfahren für die Anwendung*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 343–355. ISBN: 978-3-662-46936-1. DOI: https://doi.org/10.1007/978-3-662-46936-1_13.

[52] M. Kamel et al. „Model Predictive Control for Trajectory Tracking of Unmanned Aerial Vehicles Using Robot Operating System“. In: *Robot Operating System (ROS): The Complete Reference (Volume 2)*. Ed. by A. Koubaa. Cham: Springer International Publishing, 2017, pp. 3–39. ISBN: 978-3-319-54927-9. DOI: https://doi.org/10.1007/978-3-319-54927-9_1.

[53] G. Ganga and M. M. Dharmana. „MPC controller for trajectory tracking control of quadcopter“. In: *2017 International Conference on Circuit ,Power and Computing Technologies (ICCPCT)*. 2017, pp. 1–6. DOI: https://doi.org/10.1109/ICCPCT.2017.8074380.

[54] M. Brunner et al. „Trajectory Tracking Nonlinear Model Predictive Control for an Overactuated MAV“. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 5342–5348. DOI: https://doi.org/10.1109/ICRA40945.2020.9197005.

[55] T. Engelhardt et al. „Flatness-based control for a quadrotor camera helicopter using model predictive control trajectory generation“. In: *2016 24th Mediterranean Conference on Control and Automation (MED)*. 2016, pp. 852–859. DOI: https://doi.org/10.1109/MED.2016.7536036.

[56] A. Romero et al. „Model Predictive Contouring Control for Time-Optimal Quadrotor Flight“. In: *IEEE Transactions on Robotics* 38.6 (2022), pp. 3340–3356. DOI: https://doi.org/10.1109/TRO.2022.3173711.

[57] M. W. Mueller and R. D'Andrea. „A model predictive controller for quadrocopter state interception". In: *2013 European Control Conference (ECC)*. 2013, pp. 1383–1389. DOI: https://doi.org/10.23919/ECC.2013.6669415.

[58] T. Konrad, T. Salesch, and D. Abel. „Flatness-based model predictive trajectory optimization for inspection tasks of multirotors". In: *2019 American Control Conference (ACC)*. 2019, pp. 2264–2270. DOI: https://doi.org/10.23919/ACC.2019.8815191.

[59] M. Kamel et al. „Fast nonlinear model predictive control for multicopter attitude tracking on SO(3)". In: *2015 IEEE Conference on Control Applications (CCA)*. 2015, pp. 1160–1166. DOI: https://doi.org/10.1109/CCA.2015.7320769.

[60] H. Nguyen et al. „Model Predictive Control for Micro Aerial Vehicles: A Survey". In: *2021 European Control Conference (ECC)*. 2021, pp. 1556–1563. DOI: https://doi.org/10.23919/ECC54610.2021.9654841.

[61] M. Castillo-Lopez et al. „Model Predictive Control for Aerial Collision Avoidance in Dynamic Environments". In: *2018 26th Mediterranean Conference on Control and Automation (MED)*. 2018, pp. 1–6. DOI: https://doi.org/10.1109/MED.2018.8442967.

[62] G. Kulathunga et al. „Optimization-Based Trajectory Tracking Approach for Multi-Rotor Aerial Vehicles in Unknown Environments". In: *IEEE Robotics and Automation Letters* 7.2 (2022), pp. 4598–4605. DOI: https://doi.org/10.1109/LRA.2022.3151157.

[63] L. S. Pontryagin. *Mathematical Theory of Optimal Processes*. Taylor & Francis, 1987. ISBN: 9780203749319. DOI: https://doi.org/10.1201/9780203749319.

[64] M. Beul and S. Behnke. „Fast full state trajectory generation for multirotors". In: *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2017, pp. 408–416. DOI: https://doi.org/10.1109/ICUAS.2017.7991304.

[65] M. Hehn and R. D'Andrea. „Real-Time Trajectory Generation for Quadrocopters". In: *IEEE Transactions on Robotics* 31.4 (2015), pp. 877–892. DOI: https://doi.org/10.1109/TRO.2015.2432611.

[66] R. Pěnička and D. Scaramuzza. „Minimum-Time Quadrotor Waypoint Flight in Cluttered Environments". In: *IEEE Robotics and Automation Letters* 7.2 (2022), pp. 5719–5726. DOI: 10.1109/LRA.2022.3154013.

[67] F. Meyer. *TOP-UAV: Open-Source Time-optimal Trajectory Planner for Point-Masses under Acceleration and Velocity Constraints (C++)*. Online; accessed 11. February 2024. 2023. URL: https://github.com/fzi-forschungszentrum-informatik/top_uav_cpp.

[68] F. Meyer. *TOP-UAV: Open-Source Time-optimal Trajectory Planner for Point-Masses under Acceleration and Velocity Constraints (Python)*. Online; accessed 11. February 2024. 2023. URL: https://github.com/fzi-forschungszentrum-informatik/top_uav_py.

[69] I. E. Grossmann and F. Trespalacios. „Systematic modeling of discrete-continuous optimization models through generalized disjunctive programming“. In: *AIChE Journal* 59.9 (2013), pp. 3276–3295. DOI: https://doi.org/10.1002/aic.14088.

[70] C. E. Miller, A. W. Tucker, and R. A. Zemlin. „Integer Programming Formulation of Traveling Salesman Problems“. In: *J. ACM* 7.4 (1960), pp. 326–329. ISSN: 0004-5411. DOI: https://doi.org/10.1145/321043.321046.

[71] B. Golden, L. Levy, and R. Vohra. „The orienteering problem“. In: *Naval Research Logistics (NRL)* 34.3 (1987), pp. 307–318. DOI: https://doi.org/10.1002/1520-6750(198706)34:3<307::AID-NAV3220340302>3.0.CO;2-D.

[72] A. Thibbotuwawa et al. „Unmanned Aerial Vehicle Routing Problems: A Literature Review“. In: *Applied Sciences* 10.13 (2020). ISSN: 2076-3417. DOI: https://doi.org/10.3390/app10134504.

[73] G. S. C. Avellar et al. „Multi-UAV Routing for Area Coverage and Remote Sensing with Minimum Time“. In: *Sensors* 15.11 (2015), pp. 27783–27803. ISSN: 1424-8220. DOI: https://doi.org/10.3390/s151127783.

[74] S. Poikonen and B. Golden. „Multi-visit drone routing problem“. In: *Computers & Operations Research* 113 (2020), p. 104802. ISSN: 0305-0548. DOI: https://doi.org/10.1016/j.cor.2019.104802.

[75] S. Wang et al. „Multi-UAV Route Planning for Data Collection from Heterogeneous IoT Devices“. In: *2022 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*. 2022, pp. 1556–1560. DOI: https://doi.org/10.1109/IEEM55944.2022.9989729.

[76] S. K. K. Hari et al. „Optimal UAV Route Planning for Persistent Monitoring Missions“. In: *IEEE Transactions on Robotics* 37.2 (2021), pp. 550–566. DOI: https://doi.org/10.1109/TRO.2020.3032171.

[77] I. Cohen, C. Epstein, and T. Shima. „On the Discretized Dubins Traveling Salesman Problem“. In: *IISE Transactions* 49.2 (2017), pp. 238–254. DOI: https://doi.org/10.1080/0740817X.2016.1217101.

[78] L. Babel. „Curvature-constrained traveling salesman tours for aerial surveillance in scenarios with obstacles“. In: *European Journal of Operational Research* 262.1 (2017), pp. 335–346. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2017.03.067.

[79] L. Babel. „New heuristic algorithms for the Dubins traveling salesman problem“. In: *Journal of Heuristics* 26 (2020), pp. 503–530. DOI: https://doi.org/10.1007/s10732-020-09440-2.

[80] K. Savla, E. Frazzoli, and F. Bullo. „Traveling Salesperson Problems for the Dubins Vehicle“. In: *IEEE Transactions on Automatic Control* 53.6 (2008), pp. 1378–1391. DOI: https://doi.org/10.1109/TAC.2008.925814.

[81] J. Ny, E. Feron, and E. Frazzoli. „On the Dubins Traveling Salesman Problem“. In: *IEEE Transactions on Automatic Control* 57.1 (2012), pp. 265–270. DOI: https://doi.org/10.1109/TAC.2011.2166311.

[82] X. Yu and J. Y. Hung. „A genetic algorithm for the Dubins Traveling Salesman Problem“. In: *2012 IEEE International Symposium on Industrial Electronics*. 2012, pp. 1256–1261. DOI: https://doi.org/10.1109/ISIE.2012.6237270.

[83] P. Isaiah and T. Shima. „Motion planning algorithms for the Dubins Travelling Salesperson Problem“. In: *Automatica* 53 (2015), pp. 247–255. ISSN: 0005-1098. DOI: https://doi.org/10.1016/j.automatica.2014.12.041.

[84] K. Sundar, S. Sanjeevi, and C. Montez. „A branch-and-price algorithm for a team orienteering problem with fixed-wing-drones“. In: *EURO Journal on Transportation and Logistics* 11 (2022). DOI: https://doi.org/10.1016/j.ejtl.2021.100070.

[85] N. Tsiogkas and D. M. Lane. „DOCP: Dubins Correlated Orienteering Problem Optimizing Sensing Missions of a Nonholonomic Vehicle under Budget Constraints“. In: *IEEE Robotics and Automation Letters* 3.4 (2018), p. 2926. DOI: https://doi.org/19.1109/LRA.2018.2847719.

[86] K. Kučerová, P. Váňa, and J. Faigl. „Variable-Speed Traveling Salesman Problem for Vehicles with Curvature Constrained Trajectories“. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021, pp. 4714–4719. DOI: https://doi.org/10.1109/IROS51168.2021.9636762.

[87] Tim Knissel. „Dubins Team Orienteering Problem in a post-earthquake scenario“. MA thesis. Karlsruher Institute of Technology, 2022.

[88] F. Nekovář, J. Faigl, and M. Saska. „Multi-Vehicle Dynamic Water Surface Monitoring“. In: *IEEE Robotics and Automation Letters* 8.10 (2023), pp. 6323–6330. DOI: https://doi.org/10.1109/LRA.2023.3304533.

[89] A. Fügenschuh, D. Müllenstedt, and J. Schmidt. „Flight Planning for Unmanned Aerial Vehicles“. In: *Military Operations Research* 26.3 (2021), pp. 49–71. DOI: https://www.jstor.org/stable/27070890.

[90] J. Schmidt and A. Fügenschuh. „A two-time-level model for mission and flight planning of an inhomogeneous fleet of unmanned aerial vehicles“. In: *Computational Optimization and Applications* 85 (2023), pp. 292–335. DOI: https://doi.org/10.1007/s10589-023-00450-x.

[91] O. V. Stryk and M. Glocker. „Numerical mixed-integer optimal control and motorized traveling salesmen problems“. In: *APII – JESA (Journal européen des systèmes automatisés – European Journal of Control)* 35.4 (2001), pp. 519–533.

[92] M. Glocker and O. V. Stryk. „Hybrid Optimal Control of Motorized Traveling Salesmen and beyond“. In: vol. 35. July 2002, pp. 987–992. DOI: 10.3182/20020721-6-ES-1901.00561.

[93] G. Schrimpf et al. „Record Breaking Optimization Results Using the Ruin and Recreate Principle“. In: *Journal of Computational Physics* 159.2 (2000), pp. 139–171. ISSN: 0021-9991. DOI: https://doi.org/10.1006/jcph.1999.6413.

[94] D. Pisinger and S. Ropke. „Large Neighborhood Search“. In: *Handbook of Metaheuristics*. Ed. by M. Gendreau and J.-Y. Potvin. Cham: Springer International Publishing, 2019, pp. 99–127. ISBN: 978-3-319-91086-4. DOI: https://doi.org/10.1007/978-3-319-91086-4_4.

[95] S. T. W. Mara et al. „A survey of adaptive large neighborhood search algorithms and applications“. In: *Computers & Operations Research* 146 (2022), p. 105903. ISSN: 0305-0548. DOI: https://doi.org/10.1016/j.cor.2022.105903.

[96] M. Desrochers and G. Laporte. „Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints“. In: *Operations Research Letters* 10.1 (1991), pp. 27–36. ISSN: 0167-6377. DOI: https://doi.org/10.1016/0167-6377(91)90083-2.

[97] P. Vansteenwegen, W. Souffiau, and D. V. Oudheusden. „The orienteering problem: A survey“. In: *European Journal of Operational Research* 209.1 (2011), pp. 1–10. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2010.03.045.

[98] D. Pisinger and S. Ropke. „A general heuristic for vehicle routing problems“. In: *Computers and Operations Research* 34.8 (2007), pp. 2403–2435. DOI: https://doi.org/10.1016/j.cor.2005.09.012.

[99] P. Shaw. „Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems“. In: (1998). Ed. by M. Maher and J.-F. Puget, pp. 417–431. DOI: https://doi.org/10.1007/3-540-49481-2_30.

[100] R. Turkeš, K. Sörensen, and L. M. Hvattum. „Meta-analysis of metaheuristics: Quantifying the effect of adaptiveness in adaptive large neighborhood search“. In: *European Journal of Operational Research* 292.2 (2021), pp. 423–442. DOI: https://doi.org/10.1016/j.ejor.2020.10.045.

[101] P. A. Tu, N. T. Dat, and P. Q. Dung. „Traveling Salesman Problem with Multiple Drones“. In: *Proceedings of the 9th International Symposium on Information and Communication Technology*. SoICT '18. Danang City, Viet Nam: Association for Computing Machinery, 2018, pp. 46–53. ISBN: 9781450365390. DOI: 10.1145/3287921.3287932.

[102] A. A. Kovacs, S. N. Parragh, and K. F. Doerner. „Adaptive large neighborhood search for service technician routing and scheduling problems“. In: *Journal of Scheduling* 15 (2012), pp. 579–600. DOI: https://doi.org/10.1007/s10951-011-0246-9.

[103] D. Sacramento, D. Pisinger, and S. Ropke. „An adaptive large neighborhood search metaheuristic for the vehicle routing problem with drones“. In: *Transportation Research Part C: Emerging Technologies* 102 (2019), pp. 289–315. DOI: https://doi.org/10.1016/j.trc.2019.02.018.

[104] A. Santini. „An adaptive large neighbourhood search algorithm for the orienteering problem“. In: *Expert Systems with Applications* 123 (2019), pp. 154–167. DOI: https://doi.org/10.1016/j.eswa.2018.12.050.

[105] R. Bellman. „The theory of dynamic programming“. In: *Bulletin of the American Mathematical Society* 60 (1954), pp. 503–515. DOI: https://doi.org/10.1090/S0002-9904-1954-09848-8.

[106] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. „Optimization by Simulated Annealing“. In: *Science* 220.4598 (1983), pp. 671–680. DOI: https://doi.org/10.1126/science.220.4598.671.

[107] F. Glover. „Tabu search-part I“. In: *ORSA Journal on Computing* 1.3 (1989), pp. 190–206. DOI: https://doi.org/10.1287/ijoc.1.3.190.

[108] M. Gendreau and J.-Y. Potvin. „Tabu Search“. In: *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Ed. by E. K. Burke and G. Kendall. Boston, MA: Springer US, 2005, pp. 165–186. ISBN: 978-0-387-28356-2. DOI: https://doi.org/10.1007/0-387-28356-0_6.

[109] R. Martí et al. „Intelligent Multi-Start Methods“. In: *Handbook of Metaheuristics*. Ed. by M. Gendreau and J. Y. Potvin. Cham: Springer International Publishing, 2019, pp. 221–243. ISBN: 978-3-319-91086-4. DOI: https://doi.org/10.1007/978-3-319-91086-4_7.

[110] T. Tsiligirides. „Heuristic methods applied to orienteering“. In: *Journal of the Operational Research Society* 35.9 (1984), pp. 797–809. DOI: https://doi.org/10.1057/jors.1984.162.

[111] F. Meyer. *Problem instances for the KTSP*. Online; accessed 22. January 2024. 2023. URL: https://github.com/fameyer94/KinematicRoutingInstances/tree/main/KTSP/BENCHMARK.

[112] F. Meyer. *Problem instances for the KTSP*. Online; accessed 22. January 2024. 2023. URL: https://github.com/fameyer94/KinematicRoutingInstances/tree/main/KTSP/HPT.

[113] F. Meyer. *Problem instances for the KTSP*. Online; accessed 22. January 2024. 2023. URL: https://github.com/fameyer94/KinematicRoutingInstances/tree/main/KTSP/RUNTIME.

[114] F. Meyer. *Problem instances for the KTSP*. Online; accessed 22. January 2024. 2023. URL: https://github.com/fameyer94/KinematicRoutingInstances/tree/main/KOP/BENCHMARK.

[115] F. Meyer. *Problem instances for the KTSP*. Online; accessed 22. January 2024. 2023. URL: https://github.com/fameyer94/KinematicRoutingInstances/tree/main/KOP/HPT.

[116] F. Meyer. *Problem instances for the KTSP*. Online; accessed 22. January 2024. 2023. URL: https://github.com/fameyer94/KinematicRoutingInstances/tree/main/KOP/RUNTIME.

[117] B. Bischl et al. „Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges“. In: *WIREs Data Mining and Knowledge Discovery* 13.2 (2023), e1484. DOI: https://doi.org/10.1002/widm.1484.

[118] J. Bengstra and Y. Bengio. „Random Search for Hyper-Parameter Optimization“. In: *Journal of Machine Learning Research* 13.10 (2012), pp. 281–305. URL: http://jmlr.org/papers/v13/bergstra12a.html.

[119] R. Liu, Y. Tao, and X. Xie. „An adaptive large neighborhood search heuristic for the vehicle routing problem with time windows and synchronized visits“. In: *Computers & Operations Research* 101 (2019), pp. 250–262. ISSN: 0305-0548. DOI: https://doi.org/10.1016/j.cor.2018.08.002.

[120] Gurobi. *tsp.py*. Online; accessed 19. October 2023. 2023. URL: https://www.gurobi.com/documentation/current/examples/tsp_py.html.

[121] A. J. LaValle, B. Sakcak, and S. M. LaValle. „Bang-Bang Boosting of RRTs“. In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2023, pp. 2869–2876. DOI: https://doi.org/10.1109/IROS55552.2023.10341760.

[122] H. Yeung et al. „Laser path planning and power control strategies for powder bed fusion systems“. In: *2016 International Solid Freeform Fabrication Symposium*. 2016. URL: https://hdl.handle.net/2152/89543.

[123] H. Yeung et al. „Implementation of Advanced Laser Control Strategies for Powder Bed Fusion Systems“. In: *Procedia Manufacturing* 26 (2018). 46th SME North American Manufacturing Research Conference, NAMRC 46, Texas, USA, pp. 871–879. ISSN: 2351-9789. DOI: https://doi.org/10.1016/j.promfg.2018.07.112.

[124] N. Ganganath et al. „Trajectory planning for 3D printing: A revisit to traveling salesman problem“. In: *2016 2nd International Conference on Control, Automation and Robotics (ICCAR)*. 2016, pp. 287–290. DOI: https://doi.org/10.1109/ICCAR.2016.7486742.

[125] F. Xie et al. „Path smoothing and feed rate planning for robotic curved layer additive manufacturing“. In: *Robotics and Computer-Integrated Manufacturing* 65 (2020), p. 101967. ISSN: 0736-5845. DOI: https://doi.org/10.1016/j.rcim.2020.101967.

[126] R. Guamán-Rivera et al. „Recent Developments and Challenges of 3D-Printed Construction: A Review of Research Fronts“. In: *Buildings* 12.2 (2022). ISSN: 2075-5309. DOI: https://doi.org/10.3390/buildings12020229.

[127] H. Giberti, L. Sbaglia, and M. Urgo. „A path planning algorithm for industrial processes under velocity constraints with an application to additive manufacturing“. In: *Journal of Manufacturing Systems* 43 (2017), pp. 160–167. ISSN: 0278-6125. DOI: https://doi.org/10.1016/j.jmsy.2017.03.003.