Rheinische
Friedrich-Wilhelms-
Universität Bonn

Institute for Computer Science
Department VI
Autonomous Intelligent Systems

Rheinische Friedrich-Wilhelms-Universität Bonn

Master Thesis

Dynamic Trajectory Planning for Multirotors in Iterative δ-Spaces

Author:
Sebastian Schräder

First Examiner:
Prof. Dr. Sven Behnke

Second Examiner:
Dr. Marija Popović

Date: December 2, 2021

Declaration

I hereby declare that I am the sole author of this thesis and that none other than the specified sources and aids have been used. Passages and figures quoted from other works have been marked with appropriate mention of the source.

Place, Date

Signature

Abstract

Trajectory planning for multirotors is usually done in a two-step approach where a low-dimensional path is refined to a dynamic trajectory. These trajectories are only locally optimal. On the other hand, direct planning in higher dimensional graphs that represent multirotor states and their transitions generates globaly optimal solutions but takes too much time. To reduce planning times, the δ-Space restricts the higher-dimensional planning space. In contrast to other existing approaches, it does not only contain the area around a single shortest path but combines multiple lower-dimensional paths. Thus, a higher dimensional planner is able to plan around the paths that are most suitable for the dynamics of the multirotor.

We test our method both in 2D and 3D grid world. The method is compared against using a tunnel around a lower dimensional path. Although our approach is slower than the tunnel method, it generates better trajectories. Combined with a high heuristic weight, it finds initial results nearly as fast as the tunnel, but outperforms it in terms of pathcost. And if there is additional planning time avaiable, it is even possible to iteratively increase the size of the δ-Space to further improve the path.

Contents

1. Introduction

Nowadays, multirotors are used for multiple purposes, such as delivering packages to customers, as air taxis, for firefighting of scyscrapers, inspection of agricultural fields, or scanning of dangerous terrain. Most of these tasks have something in common: The multirotor should fly the fastest way from a startpoint to a given goalpoint. Instead of letting people manually fly the drone, such multirotors are usually operated autonomously, as a pilot would need to sit in the drone or the drone needs a constant connection to an operater. These algorithms somehow need to find a trajectory to the goal that can be flown, best would be the fastest or the one with the lowest cost, depending on the task. But with current hardware and methods, it is not possible to find the optimal trajectory in a short amount of time when the trajectory is not precomputed before it is needed. Thus, many different approaches with multiresolution, heuristics or even neural networks exist that try to find a fast trajectory in a short planning time. Often, the planning space for trajectory planning is restricted (e.g. a Tunnel around a 3D path) to decrease planning times. But many of these algorithm tends to find suboptimal path due to the differences between 3D-paths and trajectories. Often, the shortest path is not the fastest, as it is not suited for the dynamic model of the multirotors. An 2D-example is driving a car. Here, shortest paths often have many turns, while longer paths on the motorways let you travel much faster. To overcome this suboptimality, we want to introduce a new approach that should find better path. Therefore, we define a new planning space that includes all positions on all paths in the 3D space that are a maximum of δ longer than the shortest 3D-path. As this planning space can be interpreted as a combination of multiple tunnels, we expect planning times to be a bit longer. While some algorithms can be used for online-planning, i.e. replanning of a path while a multirotor is flying, we want to focus on offline-planning here, so an initial path is planned in a known map before the multirotor takes off.

Given a dynamic model of a multirotor, a start point, goal point and a map, we search a trajectory from the start to the goal. While planning, we try to minimize both the planning time and the flytime, i.e. the time the multirotor needs to fly to the goal positon. As optimal planning usually takes too much time, we try to find a good tradeoff between planning resolution, planning time and flytime.

Chapter 2 starts with an overview of the mathematical fundamentals of graphs and path planning. Then, Chapter 3 discusses related work about different approaches to solve the presented problem. Chapter 4 describes the δ-Space, a new approach to reduce the planning-space for higher-dimensional planning as well as different planning approaches to plan inside the δ-Space. In Chapter 5, the δ-Space is evaluated in experiments. Finally, Chapter 6 gives a conclusion and an outlook towards possible future research.

2. Fundamentals

A graph $G = (V, E)$ consist of nodes $v \in V$ and edges $e \in E \subset V \times V$. When all nodes are aligned in a grid, the graph can be called grid-based graph. Grid-based graphs have usually up to three dimensions. A grid-based graph can also be called lattice, where the lattice describes the neighborhood of gridcells. In contrast to a grid that has normally only up to three dimensions, a lattice can have arbitrary many. If the edges between nodes in a lattice are motion primatives, i.e. can be travelled with a multirotor, and the nodes represents states of the multirotor, the lattice is called state lattice. The resolution of the state lattice in each dimension is thereby the number of nodes per unit, e.g. $\frac{\#ofnodes}{m}$ are the number of nodes per meter in the spatial dimension.

Path planning is the task of finding a set of states that represent a collision-free path within a given map. Additionally, the transition from each state s_i to the next state s_{i+1} must be feasible under a given dynamic model.

Path planning is the task of finding a set of states $P = (s_0, s_1, ..., s_k)$ that represents a path with $s_i \in \mathbb{R}^n \, \forall \, i = 0$ to k that for a given map, dynamic model, the startstate s_0 and the goalstate s_k is traversable (under the given dynamic model) from each state s_i to the next state s_{i+1} for all states. Of all possible paths P the one that minimizes a cost function $c : P \to \mathbb{R}$ is considered the shortest path P_m.

In a graph-based search, we can use nodes as states, and construct possible edges as transitions between them that are dynamically feasible. That way we can used graph-based algorithms to find a path $P = (v_0, ..., v_k)$. Each edge $e = (v_s, v_g)$ from node v_s to node v_g is assigned a cost $c_e(v_s, v_g)$. In a graph, the total cost of a path P can be calculated by adding the cost of all edges on the path, i.e. $c(P) = \sum_{i=0}^{k-1} c_e(v_i, v_{i+1})$.

If we can use the edges of the path P as an direct input to a multirotor, the path is also called trajectory and the problem trajectory planning. In such a case, if the cost c_e represents the time it takes to fly between states, we will also refer to it as duration.

To evaluate the performance, only considering the flytime of the shortest path is not enough. There is no benefit if the planning takes too long. Because of that, in addition to the flytime, the planning time is used as a second evaluation criteria.

3. Related Work

In this chapter, we give an overview of different approaches to find a shortest path. There are mainly two ways to solve this problem. One is using sampling-based planners, the other is using graph-based planners. Nowadays, there are also sampling-based planners that use graphs to optimize planning. Figure 3.1 shows the history of planning algorithm.

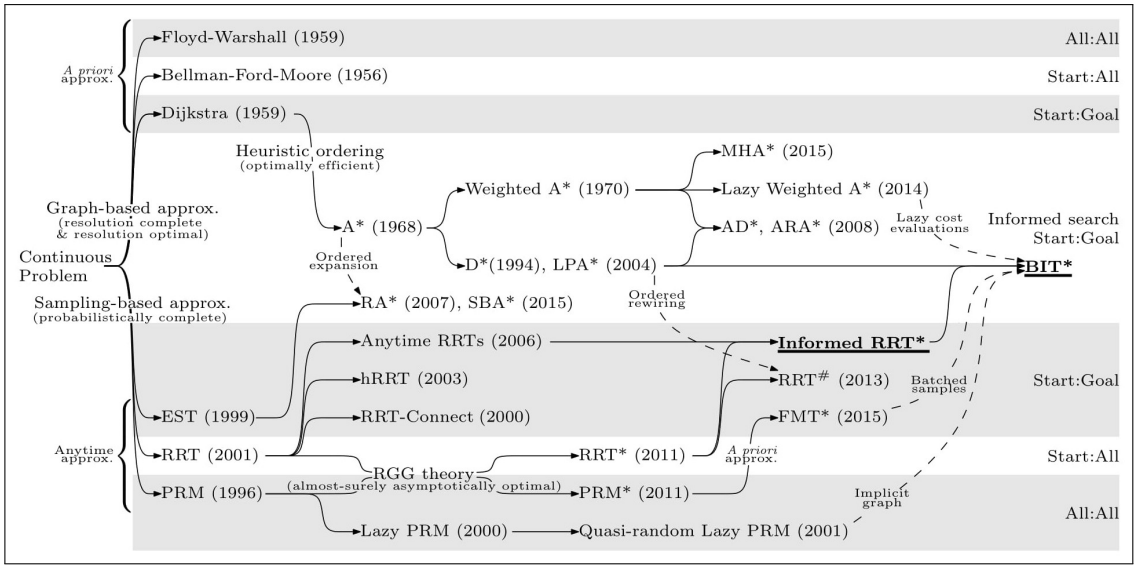


Figure 3.1: History of planning algorithms from Jonathan D Gammell 2017

3.1. Sampling-based Planners

Sampling-based planners use samples to find a path from the start to the goal. They do not need a discretized planning space, as they sample within that space.

Sampling-based planners such as **Rapidly-exploring random trees** (RRTs, LaValle et al. 1998) and their variants (e.g. Informed RRT*, Jonathan D. Gammell, Srinivasa, and Barfoot 2014) use random samples to built a tree , which is searched for a path. The differences between the variants are mainly if rewiring of connections is allowed and if the sampling space is bounded by a heuristic. **Batch**

Informed Trees (BIT*,Jonathan D. Gammell, Barfoot, and Srinivasa 2020) on the other hand orders its search by possible solution quality so that connections between the samples and the tree are formed in a way that minimizes the potential solution cost. While BIT* uses the euclidean heuristic, **Adaptively Informed Trees** (AIT*, Strub and Jonathan D. Gammell 2020) upgrades the heuristic to a heuristic that uses a graph-based seach on a graph that connects nearby samples to guide the planning. Thereby, it neglects obstacles until needed to further decrease the planning time.

3.2. Graph-based Search

Graph-based planners on the other hand need a preconstructed graph for planning. **Dijkstra's** algorithm (Dijkstra 1959) is a graph-based algorithm for graphs with non-negative edges to find a shortest path from a start node to a goal node. Therefor it expands nodes in a certain order starting with the start node, so that for every expanded node, the shortest path from the startnode to that node is known. To be able to extract the shortest path, every node thereby stores its predecessor and its cost, starting wit a cost of ∞ if no path is known yet. For expansion, nodes are ordered by their distance from the start node along the currently found shortest path, starting with the lowest. When expanding a node the algorithm looks at all connected nodes and sets or updates their shortest path, if necessary. So as long as a node is not expanded, the path from the start node to it could change several times while the cost decreases each time. The **A*** algorithm from Hart, Nilsson, and Raphael 1968 can be seen as an extension of the Dijkstra-algorithm. Using a heuristic to reorder the expansion of nodes, it guides the algorithm in the direction of the goal node, without expanding nodes that cannot provide a better solution than the shortest path. Instead of just ordering the nodes by the distance from the start node, an additional heuristic value for the cost-to-go is added to the already knwon path cost. As long as the heuristic is admissable, the algorithm returns the optimal solution, but it could require multiple expansions of some nodes. Here, admissable means that the heuristics returns a cost between nodes that is equal or lower than the shortest pathcost between these nodes. If a heuristic is also consistent, each node is expanded only once. A heuristic is consistent, if for each edge $e = (v, w)$, the heuristic for v is at most the sum of the heuristic for w and the cost of the edge e. To decrease planning times even further, it is possible to multiply the heuristic with a weight $w \geq 1$. As such a heuristic is not admissable anymore, the found path may not be optimal, but the path cost increase at most by a factor of w. If the planning time is limited, it could be good to start with a

higher weight and lower it through multiple searches to have a fast initial solution and interatively improve it while there is time left. Here, the **ARA*** algorithm described in (Likhachev, Gordon, and Thrun 2003) is able to decrease the weight w while reusing the result of previous searches by propagating local inconsistency.

Another solution is using **Multiresolution** for reducing the size of the planning space. In Behnke 2003, the resolution gets coarser with increasing distance from the startpoint, leading to faster planning times.

3.3. Trajectory Planning

Traditionally, trajectory planning for multirotors is done in two steps, a lower-dimensional path planning and an optimization step to generate a high-dimensional trajectory. The lower-dimensional planning plans in 3D, or with an additional additional yaw consideration, resulting in a 4D planning. Nieuwenhuisen and Behnke 2019 use this approach in 4D, with the additional constraint that the flight direction has to be inside the sensor field-of-view. They accelerate planning by introducing a new heuristic that incorporates the maximum flying angle into the heuristic cost function, so path outside the sensor fiel-of-view have a heuristic length of the approximation of the shortest possible flying path inside the sensor field of view instead of the euclidean length. An approach for a trajectory generation is using the lower-dimensional planner results as multiple line-segments, following by replacing the intersections between line-segments with round transition segments to smooth the path. Then, a simple trajectory can be generated along this path using a simple dynamic model of a multirotor. After that, trajectory optimizer such as CHOMP (Zucker et al. 2013), B-splines (Arney 2007) or neural networks (Simon 1993) can be used to further optimize the trajectory.

But it is also possible to directly plan a high-dimensional trajectory by searching a state lattice graph. This way it is possible to overcome the local minima optimizations of suboptimal lower-dimensional could lead to. But then, it is often necessary to decrease the planning space to get trajecories fast enough. A **Tunnel** around a lower-dimensional path is a possible approach to reduce the higher-dimensional planning space. Stachniss and Burgard 2002 are using a 5D trajectory planning for a ground robot in a channel around a 2-dimensional path. Instead of using a real tunnel, (Liu et al. 2017) are proposing a heuristic function that guides the higher-dimensional planning around a lower-dimensional path. In

both cases , the resulting high-dimensional trajectory is only locally optimal. A faster trajectory might follow a different low-dimensional path which is longer but can be followed with a higher velocity.

(Gochev et al. 2011) use an approach with adaptive dimensionality to overcome the suboptimality on reduced planning spaces. They represent different parts of the planning space at different dimensionalities, only using higher-dimension when necessary and thus keeping planning times low. Initially, they use a lower-dimensional representation for the full environment, except for an area around start and goal state. In that area, they use a higher-dimensional representation with a transition-layer to the lower-dimensional planning space at the exterior of the area. After the first planning iteration, they use a tunnel around the found path and search a trajectory inside it. If high-dimensional planning fails to track the low-dimensional path inside the tunnel, an additional high-dimensional area is added at the point of failure and the planning starts again. This is repeated until no high-dimensional areas need to be added.

As the higher-dimensional graph consists of states, straight lines between the nodes are not dynamically feasible, as instantaneous differences in different axis are not reproducable by a robot and still need to be converted into a trajecory. Instead, motion primitives that are dynamcally feasible should be used. Liu et al. 2017 use a pre-defined control set that applies a jerk input for graph edges in a node space with position, velocity and acceleration. The graph is constructed by unrolling motion primitives. Schleich and Behnke 2021 combine this idea with multiresolution.

Another solution is using a trajecory generator such as **TopiCo** (Beul and Behnke 2017). Given a 9-dimensional start state and end state, TopiCo generates a time-optimal trajectory, if such a trajectory exists. Additionally, it is given the minimum and maximum velocity, acceleration and jerk of a multirotor. Therefore, it uses jerk as the control inputs, either $j = 0$, $j = j_{min}$ or $j = j_{max}$. With these inputs, it calculates the minimal and maximum duration for each axis independantly, before synchronizing the duration of all axis if possible.

4. Approach

Algorithms like the tunnel method from Section 3.3 often reduce the higher-dimensional planning space so much that only suboptimal higher-dimensional trajectories are found. This is due to the fact that they plan along a shortest path in a lower dimension. However, a completely different position-only path might suit the dynamic model of the multirotor better and would lead to shorter flight times. By combining multiple tunnels around different lower-dimensional path that are slightly longer than the optimal one, it should be possible to find better high-dimensional trajectories. In this thesis, we propose a similar approach, but do not combine multiple tunnels. Instead, we define a planning space that contains all lower-dimensional nodes that are part of all path from start to goal that are at most δ longer than the shortest path. The resulting δ-Space also provides a search heuristic using the solution of the low-dimensional planning problem that is solved for generating the δ-Space.

4.1. The δ-Space

Definition 4.1.1. The δ-Space D of a graph $G = (V, E)$ when searching a path from startnode *start* to goalnode *goal* is defined as

$$D = \{v \in V \mid c(start, v) + c(v, goal) \leq c(start, goal) + \delta\} \tag{4.1.1a}$$

Figure 4.1 shows the δ-Space in a 2D-map with a 49-connected neighborhood and a δ of 4.

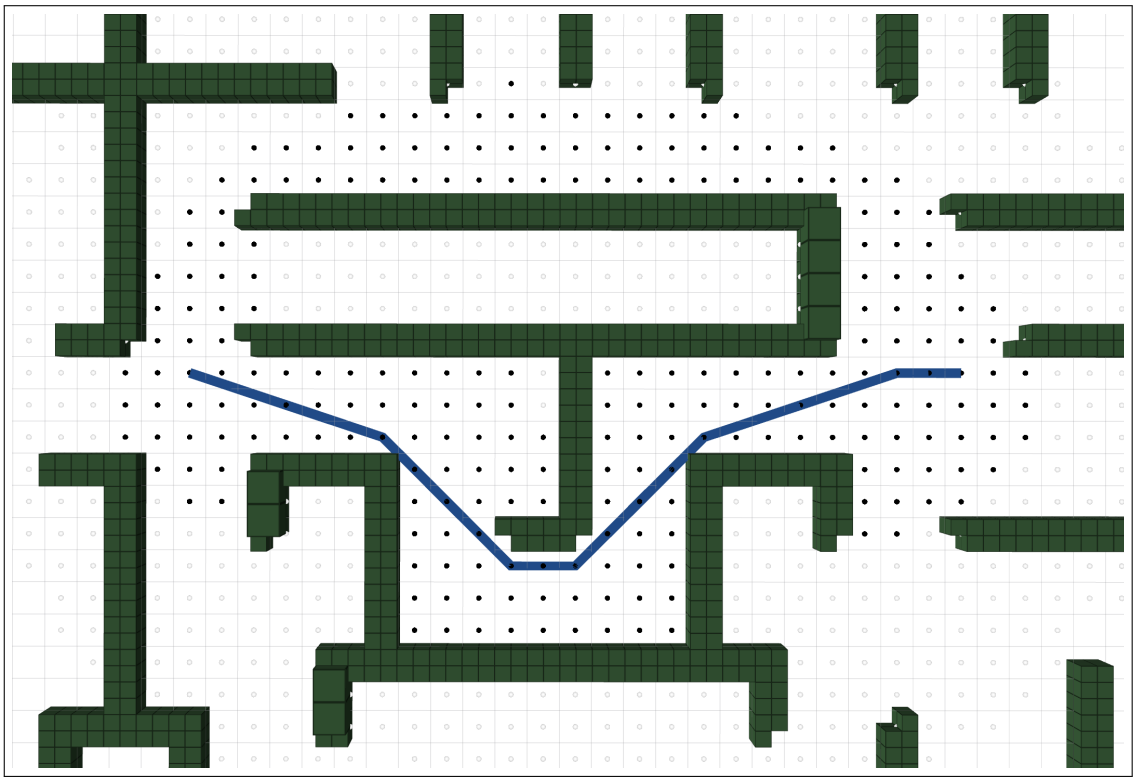


Figure 4.1: δ-Space of a 2D-maze, from start on the left to the goal on the right. All cells with a black dot inside belong to the δ-Space with a δ of 4, while the blue line is the shortest 3D path from start to goal that marks the minimal length.

Definition 4.1.2. The higher-dimensional graph $G_\delta \subseteq G_H = (V_H, E_H)$ for planning inside a lower-dimensional δ-Space of a graph $G_L = (V_L, E_H)$ when searching a path from startnode $start \in V_H$ to goalnode $goal \in V_L$ is defined as

$$
\begin{aligned}
G_\delta =& (V_\delta, E_\delta) \text{ with} && \text{(4.1.2a)}\\
V_\delta =& \{v_\delta \in V_H \mid \exists v_L \in V_L : && \text{(4.1.2b)}\\
& \quad p(v) = v_l \wedge && \text{(4.1.2c)}\\
& \quad c_L(p(start), v_L) + c_L(v_L, p(goal)) \leq && \text{(4.1.2d)}\\
& \quad c_L(p(start), p(goal)) + \delta\} \text{ and} && \text{(4.1.2e)}\\
E_\delta =& \{e_\delta := (v_i, v_j) \in E_H \mid v_i, v_j \in V_\delta\} && \text{(4.1.2f)}
\end{aligned}
$$

with

$$p : V_H \to V_L \tag{4.1.3}$$

a projection function that projects every node of the higher-dimensional space onto a node on the lower-dimensional space and

$$c_L : E_L \to \mathbb{R}^+ \tag{4.1.4}$$

a cost function for the lower-dimensional edges.

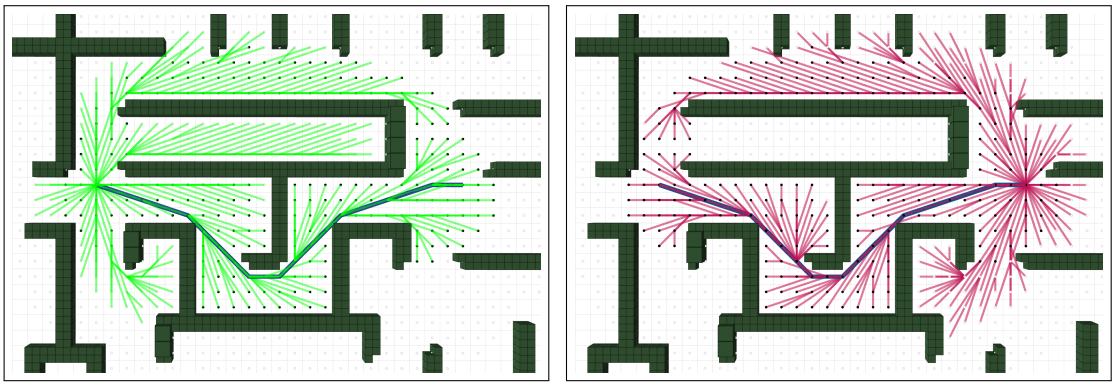
The projection function 4.1.3 used in 4.1.2c makes sure that every node of the higher-dimensional space can be projected onto one node in the lower-dimensional space, so that it could be determined if the higher-dimensional node is inside the δ-Space. In the experiments we search for the closest node in the lower-dimensional space after discarding all state components from a higher-dimensional node that the lower-dimensional space doesn't have.

Algorithm 1: Pseudo-Code planning in δ-Spaces

Input: Startnode,Goalnode,δ
Output: Trajectory

```
1 Calculating δ-Space and heuristic
2     lower-dimensional-planning forwards and backwards
3     additional lower-dimensional planning until the next node in the
        openlist is > pathcost + δ
4 Higher-dimensional planning
5     Heuristic of a node:
6         if the lower-dimensional pathcosts from start to goal through the
            node are > pathcost + δ then
7             The heuristic is ∞
8         else
9             using the lower-dimensional pathcost from goal to the node as
                heuristic
```

Algorithm 1 is a pseudo-code of a possible algorithm how the δ-Space is constructed and how it is used for high-dimensional planning. In Line 1-3, the δ-Space is generated. As planning algorithm for the low-dimensional planning in Line 2, traditional A* search can be used. As we need both the cost from the startnode as well as the cost from the goalnode, we need to use two searches, one forward and



(a) Forward search (b) Backward search

Figure 4.2: Lower-dimensional search forwards and backwards. The black dots mark all nodes that will be in the δ-Space.

one backwards. As soon as the goalnode for each of the planner is expanded, we know the pathcost in the lower dimension. As we need all nodes inside the δ-Space to be expanded from both planners, we need to restart them. In Line 3, we restart both searches and execute them until the cost of the first node in the openlist, the list of nodes that wait for expansion, exceeds the optimal pathcost by more than δ. Figure 4.2 shows such a state for both planners. Note here that all black dots marks the nodes of the δ-space to show that all necessary nodes are expanded. When using an admissable heuristic for the lower-dimensional planner, we can be sure that every node inside the δ-Space has a pathcost from both searches. As the heuristic is there to guide the searches to its goals, we can't be sure if a node has its optimal cost at the time it is expanded if the heuristic is not admissible in the direction of that node. To overcome that, a version of the A* that allows reexpansions is used. This way, we can be sure that every node has its optimal pathcost, while the heuristic still helps us to ignore some of the nodes that are outside the δ-Space.

In Line 4-9, the higher-dimensional planning is done. In Line 5-9, the heuristic is explained. For a high-dimensional node, we find the corresponding lower-dimensional node and get its pathcost to both startnode and goalnode (by adding the cost of both lower-dimensional planners). Figure 4.3 shows both searches together, showing that the pathlength can be determined by adding the cost of both planners together. Then we check, if the cost is higher than the lower-dimensional pathcost plus δ (Line 6). If it is higher, then ∞ is returned as the heuristic, as the lower-dimensional node is outside the δ-Space (Line 7). If it is inside, we can use the cost to the goal obtained by the low-dimensional backward search of the lower-dimensional planner as a heuristic, multiplied by a factor that overcomes

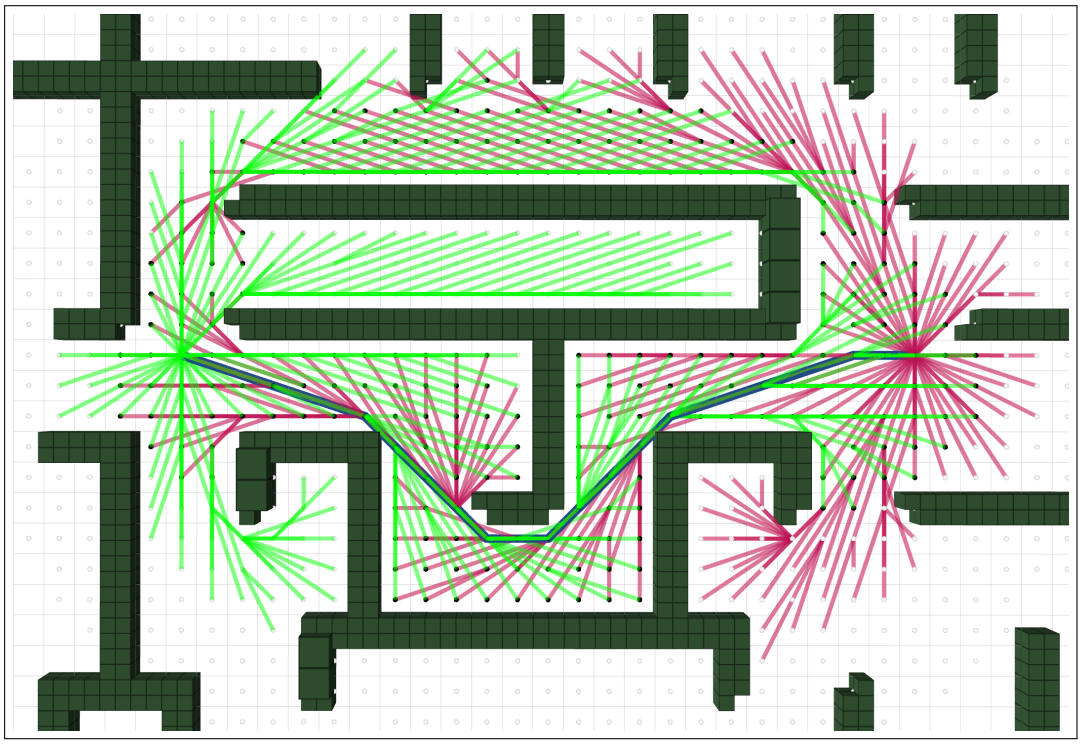


Figure 4.3: Combined lower-dimensional planning forward and backward.

the differences between the lower and higher dimension cost function. In our experiments, the cost is divided by the maximum velocity. If the node is inside the δ-Space, it is also possible to use another heuristic for the higher-dimensional planner instead, ignoring the one provided by the δ-Space.

Although the δ-Space needs a state lattice-like structure to determine if a point is inside the δ-Space or not, that only applies to the lower-dimensional planning. The higher-dimensional planning does not need a state lattice and can also be used with sampling-based planners such as RRT*, BIT* or AIT*.

4.2. Lower-dimensional Lattice

For constructring the δ-Space and the tunnel, a 3-dimensional lattice is used, where each node corresponds to a point $(p_x, p_y, p_z)^\top$ in the 3D-space. Each node is connected to all other nodes in its neighborhood via straight edges. The neighborhood of a 3-dimensional node n contains all nodes that are within a rectangular cuboid that is centered around n. For each dimension, the size of the cuboid is a multiple of twice the corresponding lattice resolution. In this thesis, we choose the same size for all dimensions. Thus, the neighborhood contains $(1 + 2 \cdot i)^3$ nodes, with

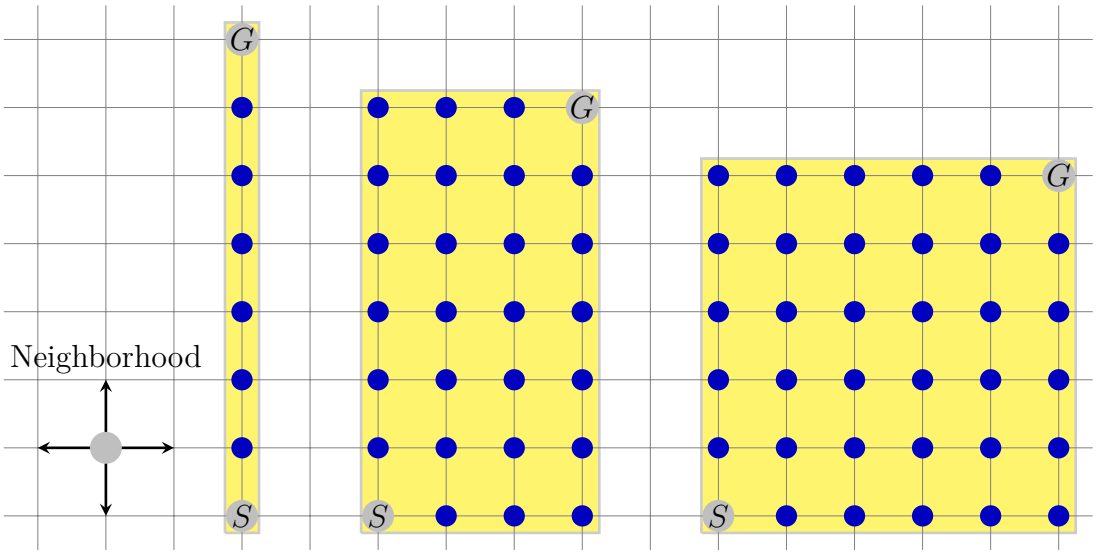


Figure 4.4: Influence of the lattice-aligning onto the δ-Space (yellow) for a distance of about 7·resolution between startnode and goalnode using $\delta = 0$ and a 4-connected neighborhood.

$i \in \mathbb{N}$. For most experiments we choose $i = 1$ and 26 edges but in Section 5.2.1 we evaluate different neighborhood sizes up to $i = 3$, ie. 342 nodes. Figure 4.4 and 4.5 show that the neighborhood size and lattice-alignment influences the δ-Space size, even for $\delta = 0$. While the euclidean distance between the nodes marked S and G is approximately the same, the size and layout of the δ-Space differ.

Even though the 4-connected neighborhood is not used in the experiments and δ is always choosen strictly greater than zero, a similar effect can be observerd for larger neighborhoods, although the difference might be smaller. When start and goal are aligned on one axis, the neighborhood choice does not affect the δ-Space, which is minimal in this case with $x + 1$ nodes. Here x is the distance from start to goal in nodes. We now consider the case where the difference between start and goal coordinates is x for the first axis and $x \cdot (i + 1)$ for the second. Here i is the multiplier for the neighborhood size as mentionend above. In this case we have the highest possible number of nodes in the δ-Space, ie $(x + 1)^2$, since the shortest path always contains x nodes on the diagonal that is closest to the second axis and x nodes on the second axis, independent of which comes first. Be aware that for greater i, the distance between start and goal increases while the number of nodes remains at $(x + 1)^2$, so the size of the δ-Space with $\delta = 0$ decreases for a fixed start and goal point the bigger the neighborhood is. The middle part of Figure 4.5 shows the maximum size for $i = 1$, which corresponds to the 8-connected neighborhood in the 2D space. As a cost for the edges, the length

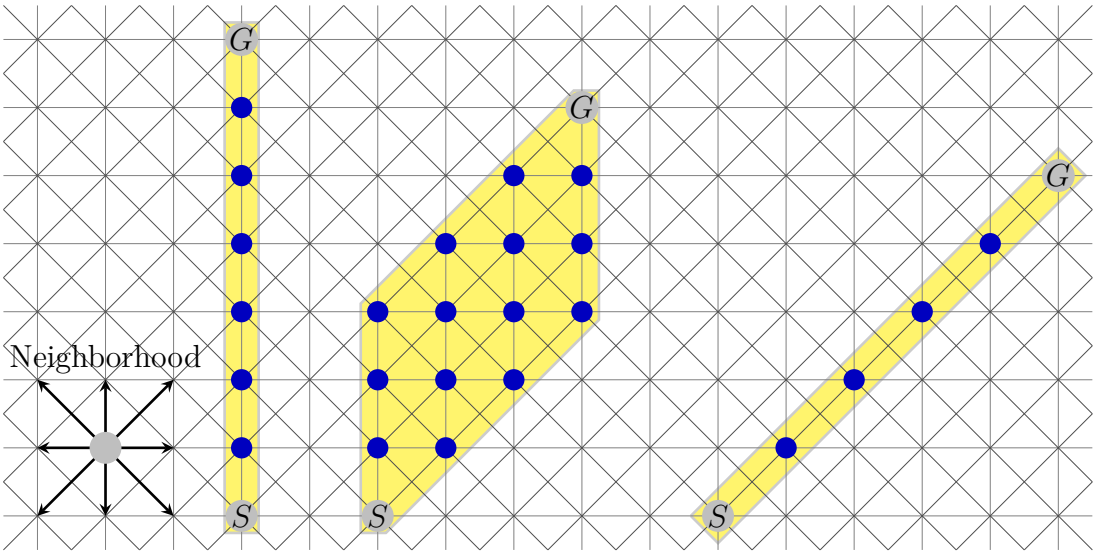


Figure 4.5: Influence of the lattice-aligning onto the δ-Space (yellow) for a distance of about 7·resolution between startnode and goalnode using a $\delta = 0$ and a 8-connected neighborhood.

is used. Since the lower-dimensional space is not only used to built the δ space and the tunnel, but also to generate a heuristic for the high-dimensional planner, you could think about adding acceleration costs to the first edges to speed up the higher-dimensional planning even more. But as that impacts the layout of the δ-Space and since the acceleration of the multirotor used in the experiments is high when jerk is neglected, we do not add additional cost.

4.3. Higher-dimensional State Lattices

There are several ways to define edges and their costs in higher-dimensional state lattices. I will present the ones used in the experiments in Chapter 5. In the following, we will present four different methods. They are evaluated against each other in Section 5.1.1, while only the best one is used to evaluate the δ-Space against the tunnel method.

9-dimensional jerk-based state lattice As jerk can be converted into input to the controller of the multirotor, the optimal state lattice should consist of 9-dimensional nodes, where each edge has a fixed jerk over the complete length of the edge. Such an edge corresponds to a motion primitive that can be described by the polynomial

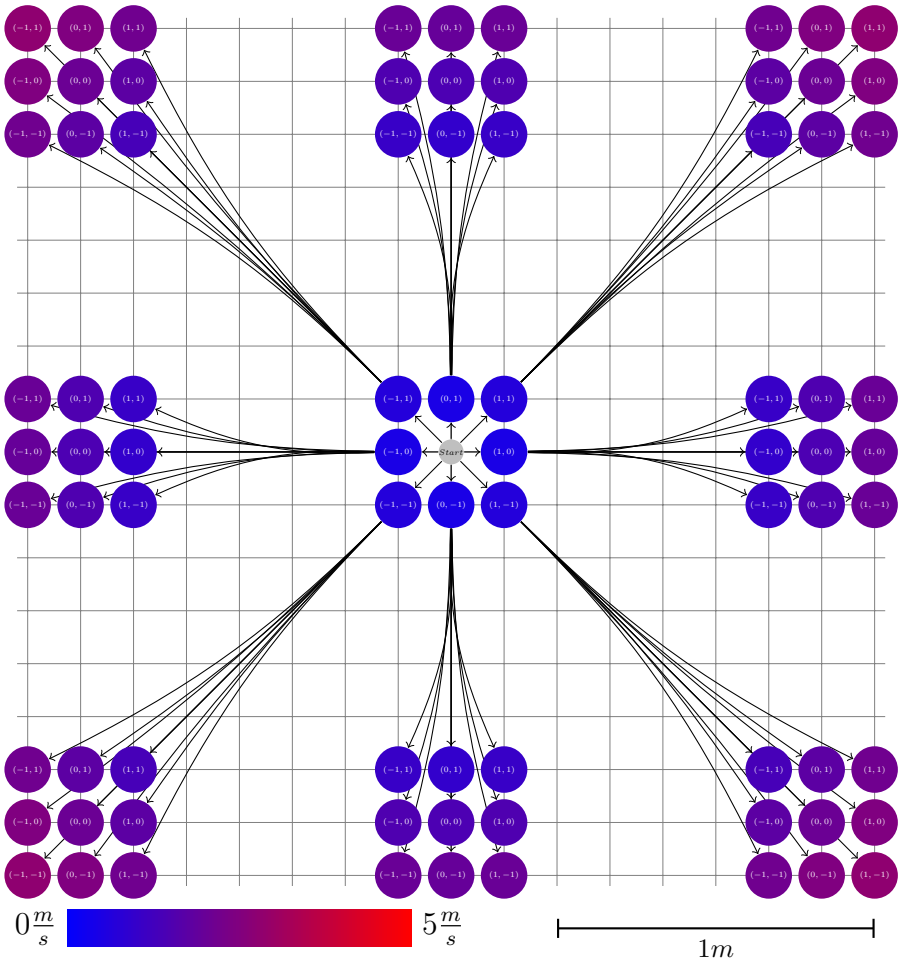


Figure 4.6: Example of 9-dimensional motion primitives with a jerk of $j = \{-1\frac{m}{s^3}, 0\frac{m}{s^3}, 1\frac{m}{s^3}\}$ as control-input for a duration of $1s$. The numbers in the nodes corresponds to the jerk input of the edges leading to them in the X and Y Axis.

$$\left(\begin{array}{c} p_x = p_{0,x} + v_x \cdot t + \frac{1}{2}a_x \cdot t^2 + \frac{1}{6}j_x \cdot t^3 \\ p_y = p_{0,y} + v_y \cdot t + \frac{1}{2}a_y \cdot t^2 + \frac{1}{6}j_y \cdot t^3 \\ p_z = p_{0,z} + v_z \cdot t + \frac{1}{2}a_z \cdot t^2 + \frac{1}{6}j_z \cdot t^3 \\ v_x = v_{0,x} + a_x \cdot t + \frac{1}{2}j_x \cdot t^2 \\ v_y = v_{0,y} + a_y \cdot t + \frac{1}{2}j_y \cdot t^2 \\ v_z = v_{0,z} + a_z \cdot t + \frac{1}{2}j_z \cdot t^2 \\ a_x = a_{0,x} + j_x \cdot t \\ a_y = a_{0,y} + j_y \cdot t \\ a_z = a_{0,z} + j_z \cdot t \end{array}\right).$$

As the corresponding primitive duration t is used as the cost, it needs to be the same duration for all axis X, Y, Z. Additionally, the jerk inputs j_x, j_y and j_z are

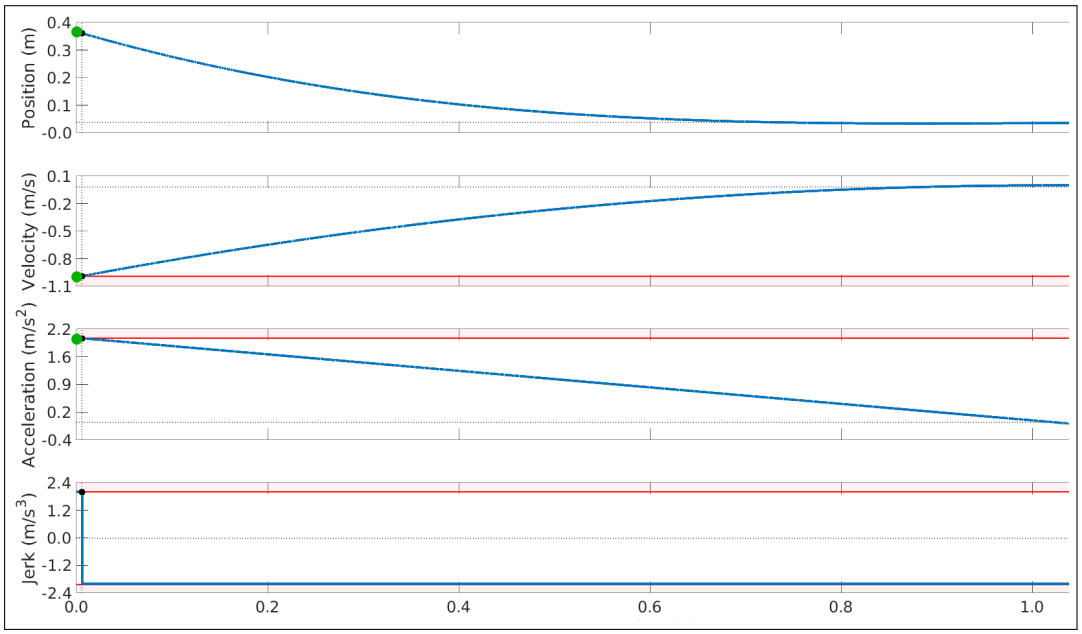


Figure 4.7: Example of a 9-dimensional edge with jerk as control-input.

independent. To be able to combine all different combinations of jerk inputs, a solution is to use a fixed duration t. This way, it is possible to calculate the lattice resolutions so that each edge ends in a node. If an edge already passes through a node at a time $t_e < t$, that node can be used as a goal point of the edge instead, with the lower duration t_e.

When calculating the lattice resolutions, we need to pay attention, as the resolutions of the different dimensions are dependent on each other. Be $\Delta_i = \frac{1}{resolution_i}$ the distance between two nodes in the i-dimension on the state lattice. When constructing the acceleration dimensions, the distance must be $\Delta_a = j \cdot t$ where t is the duration and j is the jerk input. Note here that we are only using a jerk of 0 or the maximum allowed jerk, while it would be possible to allow different jerks. For the velocity dimension, the distance is dependent of acceleration and jerk, so both $\Delta_v = dist_a \cdot t$ and $\Delta_v = \frac{1}{2} j \cdot t^2$ must land on the lattice. For the last missing dimension, the spatial dimension, the distance is dependent of velocity, acceleration and jerk, so $\Delta_p = \Delta_v \cdot t, \Delta_p = \frac{1}{2}\Delta_a \cdot t^2$ and $\Delta_p = \frac{1}{6} j \cdot t^3$.

Given a node $i = (p_{x_0}, p_{y_0}, p_{z_0}, v_{x_0}, v_{y_0}, v_{z_0}, a_{x_0}, a_{y_0}, a_{z_0})^\top$, a duration t and a possible jerk command $j = (j_x, j_y, j_z)^\top$, the goal node k of an edge $e = (i, k)$ with input $j = (j_x, j_y, j_z)^\top$ can be calculated with the polynomial above. Unfortunately, this calculation of the lattice resolution leads to a large resolution, and thus a long planning time (see Section 5.1.2 for an example with data from a real multirotor). Additionally, a goal region must be defined, as it could not be possible to reach

all nodes inside the δ-Space or tunnel with this definition for edges. The shortest distance in the spatial dimension while acceleration and velocity are zero in both start and goalstate in one axis would be an input of $(j, -j, -j, j)$, already leading to a difference of several nodes in the spatial dimension. In Figure 4.6 you can see the fist two input, starting from a startstate with a velocity and acceleration of 0. Here you can see the small grid that is needed and that there are many positions that can't be directly reached.

An option to keep the number of nodes low would be to use long durations, but that is often not possible, as even the shortest jerk could get the acceleration to a value higher than the maximum of the dynamic model, or the same can happen to the velocity. Also, the goal region would have to be even larger. Figure 4.7 shows an example for one axis for such a edge for a start velocity of $-1\frac{m}{s}$, a start acceleration of $2\frac{m}{s^2}$, a goal velocity of $0\frac{m}{s}$ and a goal acceleration of $0\frac{m}{s^2}$.

Another option would be to neglect the jerk and use acceleration directly as the input.

9-dimensional acceleration-based state lattice When using acceleration as the control input instead of jerk, it would be possible to reduce the state dimension of the planner to 6 instead of 9. This way, the polynomial that describes an edge can be reduced to

$$\begin{pmatrix} p_x = p_{0,x} + v_x \cdot t + \frac{1}{2} a_x \cdot t^2 \\ p_y = p_{0,y} + v_y \cdot t + \frac{1}{2} a_y \cdot t^2 \\ p_z = p_{0,z} + v_z \cdot t + \frac{1}{2} a_z \cdot t^2 \\ v_x = v_{0,x} + a_x \cdot t \\ v_y = v_{0,y} + a_y \cdot t \\ v_z = v_{0,z} + a_z \cdot t \end{pmatrix}.$$

But the jerk should not be neglected completely, as the dynamic model of a multirotor contains it. Instead, the acceleration-dimensions are still used to permit large changes in acceleration in nodes. Here, the state of the acceleration-dimensions corresponds to the acceleration of all outgoing edges. The reached nodes are then the same in all dimensions except the acceleration that changes by up to k This way, the goal nodes j of edges $e = (i, j)$ that start in $i = (p_{x_0}, p_{y_0}, p_{z_0}, v_{x_0}, v_{y_0}, v_{z_0}, a_{x_0}, a_{y_0}, a_{z_0})^\top$ can be calculated with the formula

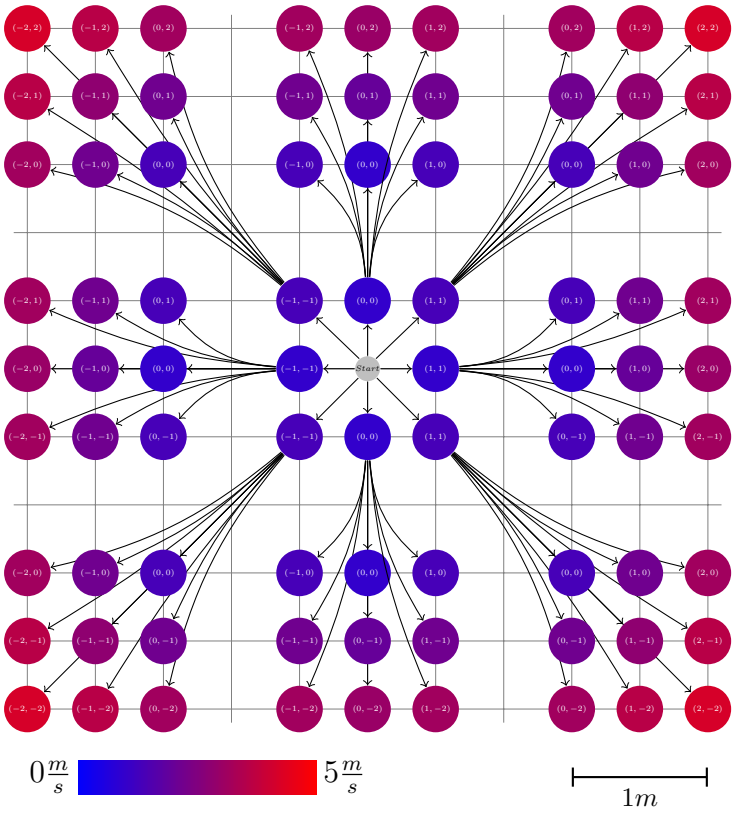


Figure 4.8: Example of 9-dimensional motion primitives with a acceleration of $a = \{-2\frac{m}{s^2}, -1\frac{m}{s^2}, ..., 2\frac{m}{s^2}\}$ as control-input for a duration of $1s$. Like in the experiments, a maximum acceleration change of 1 between nodes is allowed. The numbers in the nodes corresponds to the acceleration input of the edges leading to them in the X and Y Axis.

$$j = i + \begin{pmatrix} v_{x_0} \cdot t + \frac{1}{2} a_{x_0} \cdot t^2 \\ v_{y_0} \cdot t + \frac{1}{2} a_{y_0} \cdot t^2 \\ v_{z_0} \cdot t + \frac{1}{2} a_{z_0} \cdot t^2 \\ a_{x_0} \cdot t \\ a_{y_0} \cdot t \\ a_{z_0} \cdot t \\ a_{x_0} + k_x \\ a_{y_0} + k_y \\ a_{z_0} + k_z \end{pmatrix} = \begin{pmatrix} p_{x_0} + v_{x_0} \cdot t + \frac{1}{2} a_{x_0} \cdot t^2 \\ p_{y_0} + v_{y_0} \cdot t + \frac{1}{2} a_{y_0} \cdot t^2 \\ p_{Z_0} + v_{z_0} \cdot t + \frac{1}{2} a_{z_0} \cdot t^2 \\ v_{x_0} + a_{x_0} \cdot t \\ v_{Y_0} + a_{y_0} \cdot t \\ v_{Z_0} + a_{z_0} \cdot t \\ a_{x_0} + a_{x_0} + k_x \\ a_{y_0} + a_{y_0} + k_y \\ a_{z_0} + a_{z_0} + k_z \end{pmatrix}.$$

The acceleration of an edge could also be changed for all outgoing edges instead of all incoming edges, but with our way the state of each node corresponds to the trajectory for the outgoing edge that the multirotor should approximate. For the new acceleration of all reached nodes, a maximum change of 1 in all acceleration-

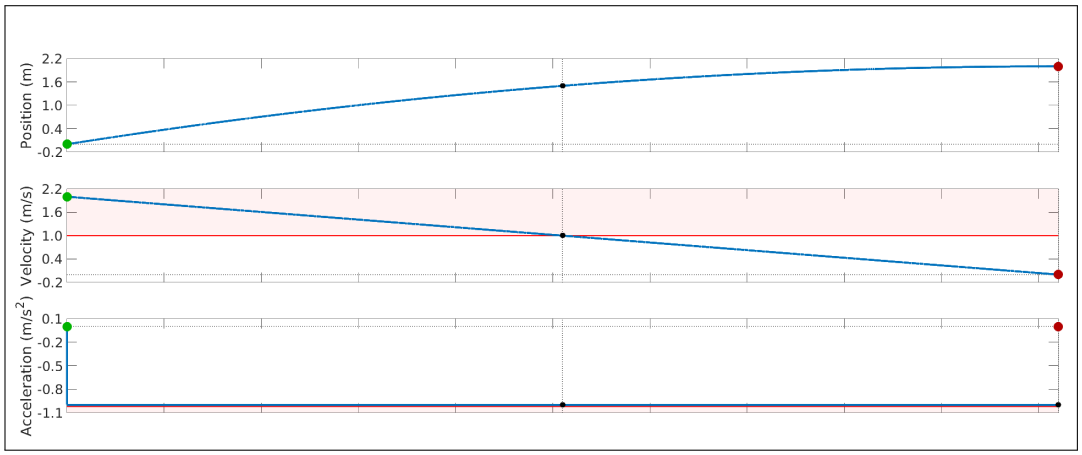


Figure 4.9: Example of a 9-dimensional edge with acceleration as control-input.

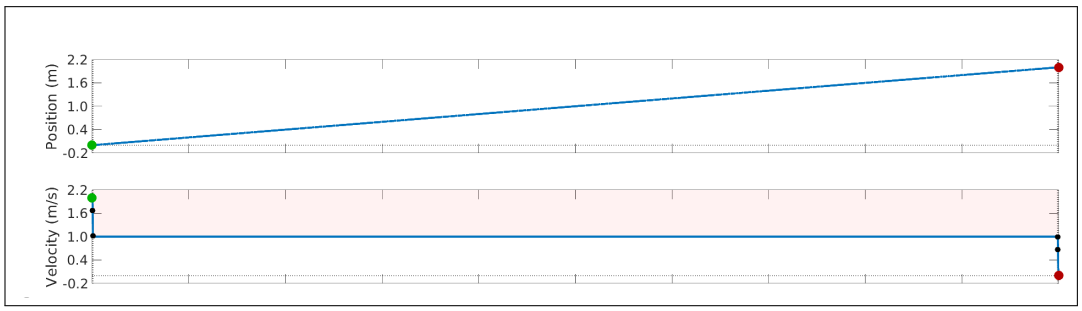
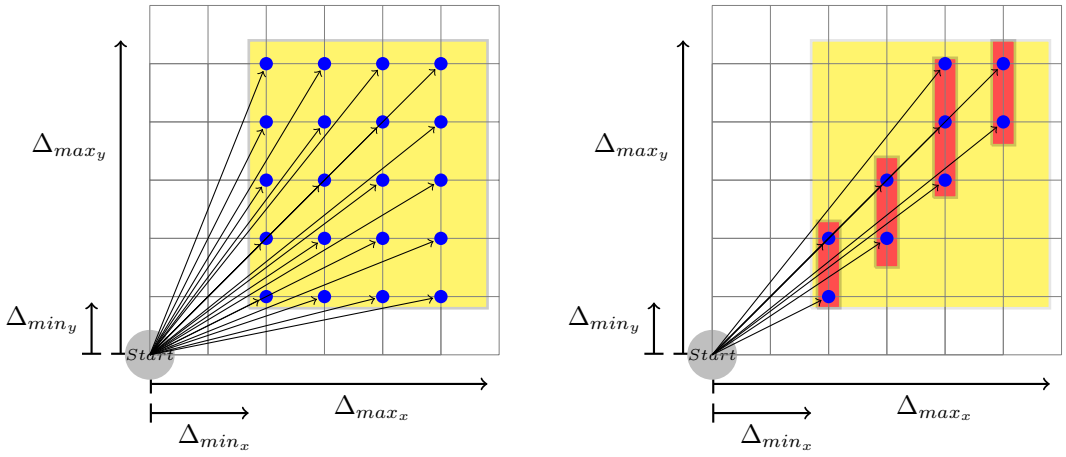


Figure 4.10: Example of a 6-dimensional edge with average velocity.

dimensions is used. That leads to only small changes in acceleration when reaching a node, so the path is closer to a real one with jerk in comparison with an path that allows going to every acceleration when reaching a node. Figure 4.8 shows the first two inputs, starting from a startstate with a velocity and acceleration of 0. Note here that in this figure the numbers in the nodes corresponds to the accelerations of incoming edges due to readability, and not outgoing. Figure 4.9 shows an example for one axis of such an edge for a start velocity of $2\frac{m}{s}$, a start acceleration of $-1\frac{m}{s^2}$ and a goal velocity of $0\frac{m}{s}$ while travelling $2m$.

6-dimensional average velocity For an easy model of edges for a 6-dimensional state lattice with position and velocity, the average velocity between start- and goalnode is used to determine the direction of the edge. As the nodes need to be on the lattice and the edge represents a straight line, the direction of the average velocity is propagated until it reaches a lattice node. The minimal length of an edge is thereby dependent on the time that is needed to accelerate between these velocities. The time it takes to fly between start and goal point with the average



(a) Example of all tests for a combination of start and goal velocity. The yellow area marks the calculated testdomain for the goalposition, while the arrows show all tests

(b) Example of all test with iterative axis. The red area is the optimized testdomain.

Figure 4.11: Example of a 2D testdomain for TopiCo for a combination of start velocity and goal velocity

velocity is used as the cost of that edge. Figure 4.10 shows an example for one axis for such a edge for a start velocity of $2\frac{m}{s}$ and a goal velocity of $0\frac{m}{s}$ while travelling $2m$. On the downside, as these edges neglect jerk completely and only use a fairly easy model to approximates acceleration, they are far from flyable.

TopiCo To have dynamically feasible edges even in 6 dimensions, it is required to reduce acceleration and jerk in each node to 0 and let each edge be a trajectory with different acceleration- and jerk-parts on the edge.

Such edges could be precomputed using a trajectory generator such as TopiCo (Beul and Behnke 2017). TopiCo is given start velocity and goal velocity as well as the minimum and maximum velocity the multicopter is allowed to fly. As TopiCo also needs start and goal point, different possible goal points from a fixed start point are tried for each edge to find the best goal point for a given start and goal velocity. To get smooth trajectories, the maximum velocity TopiCo is allowed to use on an edge is slightly higher than the maximum velocity of the start and goal point, seperately for X,Y and Z. The same is done with the minimum velocity, but in the other direction. When using only the minimum and maximum, TopiCo has problems finding edges due to its internal structure.

To minimize the number of goalpositions that need to be tested, the testdomain of possible goalpositions is restricted with a minimum and maximum duration, leading to a cuboid of possible goalpositions when combined with maximum and

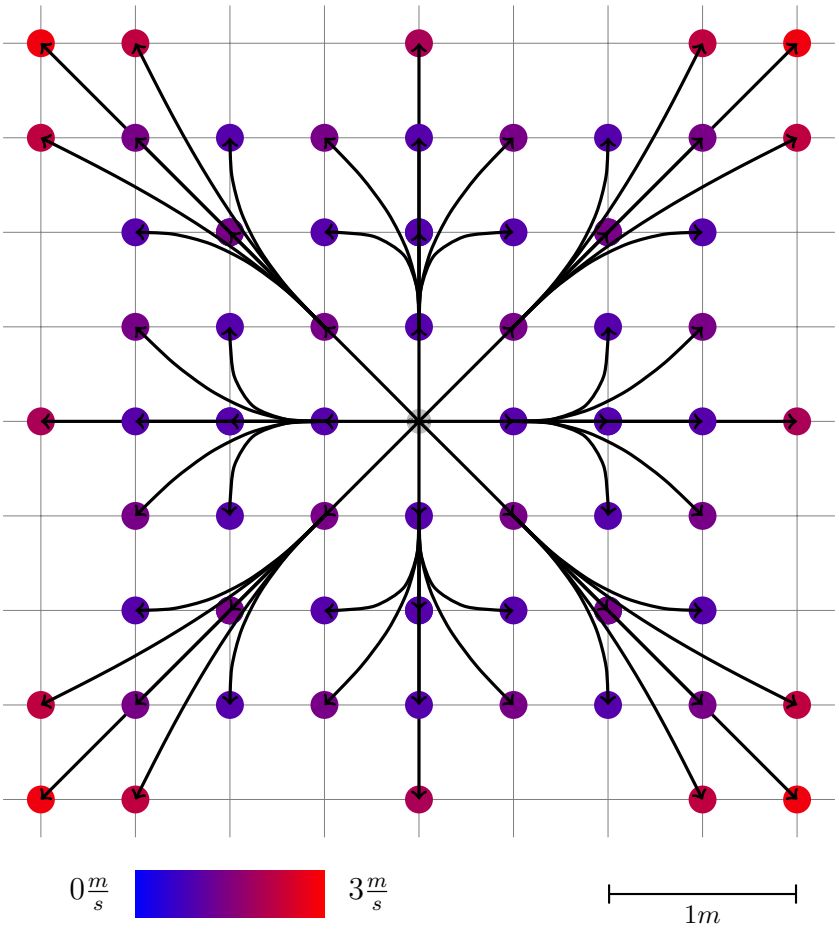


Figure 4.12: Example of 6-dimensional motion primitives precomputed with TopiCo. The maximum change in velocity between nodes is $\pm 1\frac{m}{s}$ per axis and all nodes that can be reached in $2s$ are shown.

minimum velocity. Figure 4.11a shows such a testdomain for two axis. The minimum and maximum change in the position Δ_{min} and Δ_{max} are calculated when multiplying the minimum and maximum duration with the minimum and maximum velocity. There is also the possibility to do this iteratively. When selecting a testvalue for the first axis, the minimum and maximum time getting to that value with the bounded velocity can be calculated. Combining both minimum and both maximum leads to a smaller testdomain for the second axis. Figure 4.11b shows how it could look like. To further increase the search speed, only startvelocities ≥ 0 are tested, as the result can be mirrored on each axis as long as the multirotor model is symmetric.

If TopiCo isn't able to syncronize the times for X,Y and Z, there is no solution for that edge and the tried point is not a possible goal point. In the end, there

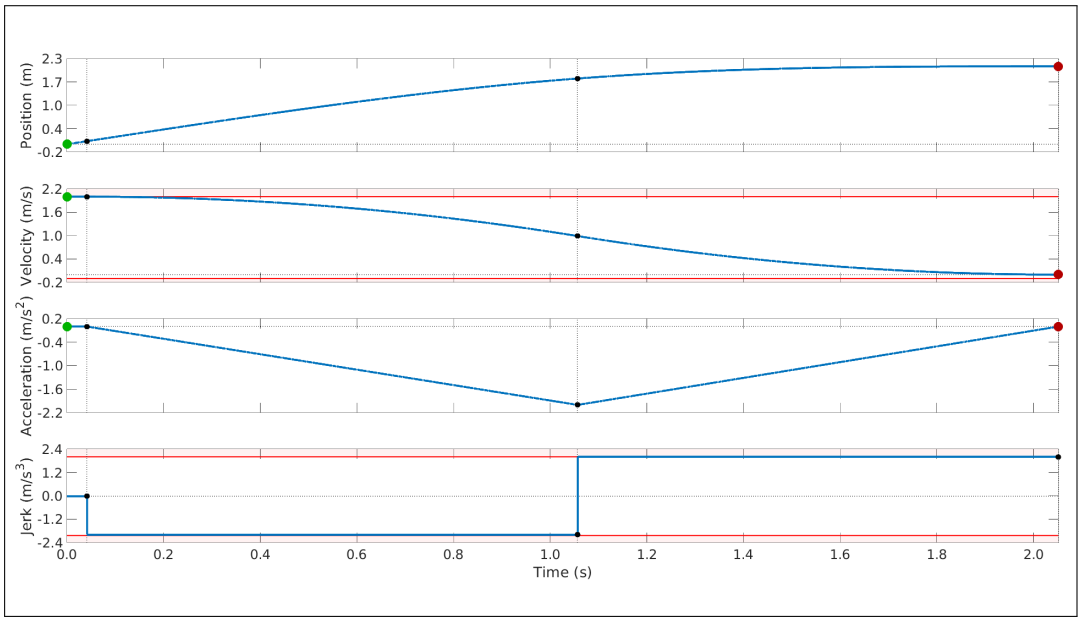
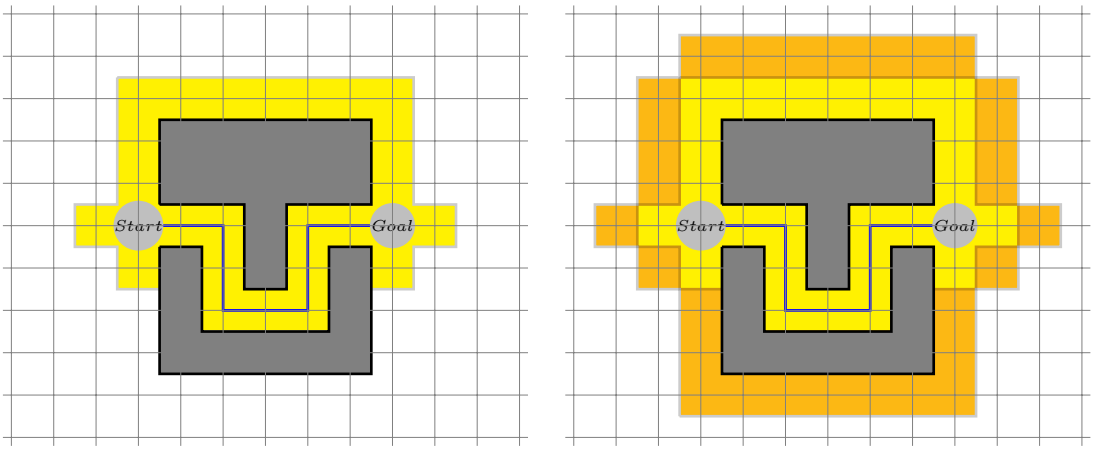


Figure 4.13: Example of a 6-dimensional edge calculated with TopiCo.

are still many possible goal points for each combination of start velocity and goal velocity, leading to too many edges if all are used for planning. One option to reduce the number of edges is only using the best edge for every combination of start and goal point. To reduce unnecessary movement, the edge that minimizes a combination of the flytime t and $t \cdot a^2$ is chosen, while the cost function for the planner is only the flytime. Figure 4.12 shows an example of motion primitives from a standing start point. Note here that in contrast to the 9D edges from above, all nodes that can be reached in a flytime of $2s$ are shown. Figure 4.13 shows an example for one axis for such a edge for a start velocity of $2\frac{m}{s}$ and a goal velocity of $0\frac{m}{s}$ while travelling $2.1m$.

4.4. A* with Iterative δ-Space

The δ-Space can also be increased iteratively. This way, a first result can be found faster while the path can still be optimized afterwards. Otherwise, when doing a single search with a high δ the maximal available planning time might be exceeded and no trajectory is found at all. Furthermore, a single search with a low value for δ might result in a trajectory with higher cost. With every iteration, the δ is increased, leading to a larger δ-Space and thus to potentionally better high-dimensional trajectories. Figure 4.14 shows an example how the δ-space can increase when increasing δ.



(a) $\delta = 2$ (b) $\delta = 4$

Figure 4.14: Increasing δ. The light orange area marks thereby the added space when increasing δ.

Algorithm 2 is an extension to Algorithm 1 (Line 1) to increase the δ after the initial planning. This can be done multiple times (Line 2) until either the δ-Space is equal to the full high-dimensional space V_H or until constraints (e.g. time or memory consumption) are violated.

Algorithm 2: Pseudo-Code planning in iterative δ-Spaces

Input: Startnode,Goalnode,δ
Output: Trajectory

```
1 Algorithm 1
2 while There is time to improve the path do
3     Incease δ
4     Restart lower-dimensional planning until the next node in the openlist
        is > pathcost + δ
5     Update Heuristic of all higher-dimensional Nodes where the heuristic is
        ∞
6     Sort openlist of higher-dimensional planner
7     Restart higher-dimensional planning until a better solution is found or
        there is no better solution
```

In Line 3, the algorithm increases δ [1] Afterwards, the low-dimensional planner is restarted (Line 4) and executed until the next node to expand has a potential path length that exceeds the optimal one by more than δ. This is the same as

[1] How different sizes of increase can impact the planning time can be seen in Section 5.2.5.

Line 3 of Algorithm 1, but now with a increased δ. Subsequently, for all high-dimensional nodes that were previously outside of the δ-Space, we check if they are part of the new δ-Space. After that all higher-dimensional nodes that were outside the δ-Space before must be rechecked, if they are inside the δ-Space now. Since we defined the heuristic of nodes outside of the δ-Space to be ∞, we can just go through the openlist and check all of them (Line 5). To regain the integrity of our openlist, we now have to sort it (Line 6), but that could also be done directly for each node when its heuristic is decreased (like A* does when the cost of a node decreases). Now we can restart the higher-dimensional A* search until we find a better path or we find that there is no better path. Since a better path could use already expanded nodes and traditional A* does not allow reexpansion of nodes, we need to modify it. Instead of allowing one expansion for each node during the A* search, we allow one expansion for the initial search and one for each iteration.

4.5. Worst-case Runtime

The worst-case runtime of A* corresponds to the worst-case of Dijkstra's Algorithm, which can be considered as a special case of A* without a heuristic. According to Korte and Vygen 2008, Dijkstra itself is $O(|E| + |V| \cdot log(|V|))$. Since $|V_\delta| \leq |V_H|$ and $|V_L| \leq |V_H|$, the first iteration (until line 10 of Algorithm 1) can be done in $O(|E_L| + |E_H| + |V_H| \cdot log(|V_H|))$. For each iteration while increasing the δ (see Section 4.4), we need to distinguish between the higher- and the lower-dimensional planning. While the lower-dimensional planning just restarts the algorithm without allowing re-expansion of nodes, it is already included in the worst case runtime of expanding everything.

The higher-dimensional planner has to re-expand nodes if an already expanded node finds a better path. As this could be nearly all nodes in the worst case, every iteration could take up to $O(|E_H| + |V_H| \cdot log(|V_H|))$. Let i be the number of iterations. This leads to a total worst-case time of $O(|E_L| + (i+1) \cdot |E_H| + (i+1) \cdot |V_H| \cdot log(|V_H|))$. This is even worse than the worst-case runtime of a high-dimensional Dikstra search with $O(|E_H| + |V_H| \cdot log(|V_H|))$. However, the average runtime is much faster than the worst-case runtime of Dijkstra's Algorithm. First, $|V_H|$ is much bigger than $|V_L|$, as can be seen in the 3-dimensional experiments in Chapter 5 where each node in the lower-dimensional planning space corresponds to at least 363 nodes in the higher-dimensional space, leading to a significantly smaller planning time in the lower-dimensional space. Secondly, the size of the higher-dimensional planning space $|V_\delta|$ is only a fraction of $|V_H|$, depending on several conditions as described in Section 4.2.

5. Evaluation

In this chapter, all experiments are described and analyzed. If not otherwise stated, the following parameters were used:

- **Map**
  All 2D experiments were done in the 2D maze shown in Figure 5.1, with a size of $58m \times 60m \times 0m$. Even though the Z-Axis was not used in the map itself, the experiments where still run in 3D, but the third axis was not allowed to change, degenerating the state space.

  For the 3D experiments the two maps shown in Figure 5.5 where used. They both have a size of $80m \times 62m \times 35m$.

- **minimum distance to obstacles**
  The minimum distance from an obstacle to a cell is dependent on the map. For the 2-D map, a distance of $0.3m$ was used, while the 3-D maps used a minimum distance of $0.9m$.

- **heuristic weight**
  For the higher dimensional A* search, the heuristic values are inflated by a factor of 1.833. To convert the pathlength from the 3D-planner to the flytime, the length was divided by the maximum velocity.

- δ **& tunnel increase**
  For all experiments that used iterative spaces, the increase is the size that δ or the tunnelradius was increased. Here, 0.5m was used.

- δ **& tunnel radius**
  The δ and the tunnelradius the planning was done with. If not otherwise stated, a δ of $1m$ was used and a tunnelradius if $2m$.

- **Resolution**
  The distance between two nodes in the state lattice for each dimension. For all 3D experiments (Section 5.2), the following resolutions are used:
  Position X,Y,Z: $\frac{1}{2}m$
  Velocity X,Y,Z: $1\frac{m}{s}$

- **Maximum velocity**
  The maximum velocity the planner is allowed to plan with:
  X and Y-axis: $\pm 5 \frac{m}{s}$
  Z-axis: $\pm 2 \frac{m}{s}$

As all planning times are highly dependent on the hardware, it could be difficult to replicate the exact times. The following hardware was used for all experiments:

- **CPU:** Ryzen 9 3900X @ 3800Mhz
- **Memory:** 64GB @ 3600MHz
- **Chipset:** X570

As this CPU also has a boost functions that boost its frequency up to 4600MHz, the times are also dependent on the temperature of the CPU. For cooling, a watercooler was used to keep the temperature low enough so that the CPU is able to constantly boost the frequency to around 4500MHz, leading to fewer errors due to these functions.

The presented planning times do not include the time used for loading the maps, preparing the planner and visualization and finally cleaning up the memory. It is assumed that for a real planning, all the outputs are disabled and the programm is started together with the multirotor so that it can load the map and other data in advance and wait for a goal state to start the planning.

5.1. 2D Experiments

The first experiments were done in the 2D-map shown in Figure 5.1. While all paths start at the same position (blue dot), the goal positions were chosen as the corners of a uniform grid with resultion of 1m. This results in a total of over 2,000 planning tasks. Instead of using a special 2D planner, we use the 3D version but set the size of the z-axis to one and forbid any vertical accelerations.

As this experiment was done to determine if the δ-Space works, a simplified dynamic model and larger lattice resolutions were used to keep the planning times low. As edges, the acceleration-based method described in Section 4.3 was used. The lattice resolution and the fixed duration between nodes were set in such a way that the resolution of the different lattice dimensions were small. Here, a duration of 1s was used, a maximum velocity of $\pm 4 \frac{m}{s}$ and a maximum acceleration of $\pm 1 \frac{m}{s^3}$. Together with a resolution of $1 \frac{m}{s}$ for the velocity and a resolution of $1 \frac{m}{s^2}$ for the acceleration, a resolution of $\frac{1}{2}m$ is necessary for the position. As only the acceleration needs a resolution of $0.5m$, while the velocity alone requires an

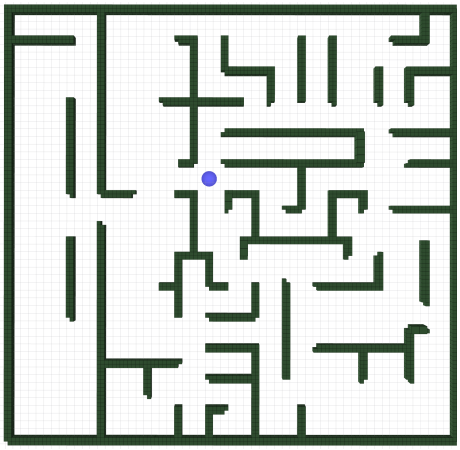


Figure 5.1: The map for all 2D experiments. The blue circle in the middle marks the startpoint for all searches

resolution of $1m$, it is possible to use resolutions of $1m$ if the state lattice is shifted by $0.5m$ for every odd velocity. In the experiment such a shifted lattice is used. For the low-dimensional planning, a neighborhood of 48 was used. Figure 5.2 shows the average planning time and average pathcost for different tunnel- and δ-sizes. Here, the δ-Space behaves like a tunnel for a small values of δ. However, increasing δ results in a larger reduction of path cost than increasing the tunnel size for similar planning times. Figure 5.3 shows example trajectories for tunnel and δ-Space. Note that with small walls and large tunnel- and δ-sizes, part of a tunnel can be inside a dead end. However the δ space only covers the entrance of the dead end. It can also be observed how the δ-Space leads to a smaller overall cost, while the planning time is only slightly longer.

5.1.1. Evaluation of Edge Types

The next experiment was done to determine the most suitable state lattice definition. Here, the four different edge types described in Section 4.3 are evaluated against each other. For the 6D state lattices, a spatial resolution of 0.5m is used, while the 9-dimensional state lattices have a much smaller resolution. A fixed jerk or acceleration over a complete edge leads to smaller resolutions if the flytime for an edge is short. On the other hand, if the flytime is long, it would lead to higher changes of velocities in one step. The 9D edges represent motion primitives where a fixed jerk or acceleration command is applied over a small time interval. Choosing a short primitive duration results in small position changes and, thus, in a lower spatial resolution. However, when choosing the duration too long, the resulting velocity resolution might be too coarse. While a smaller spatial resolu-

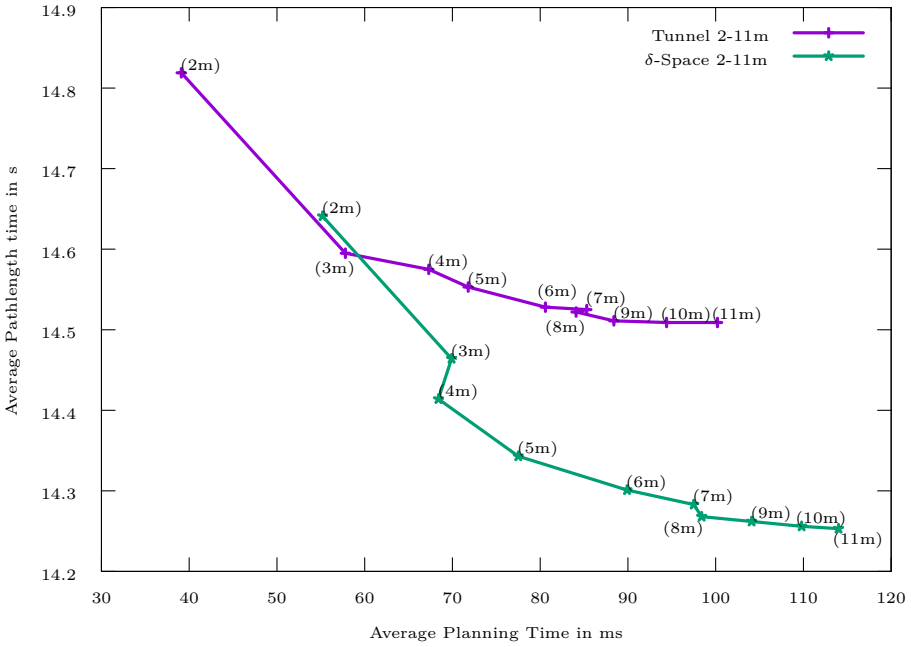
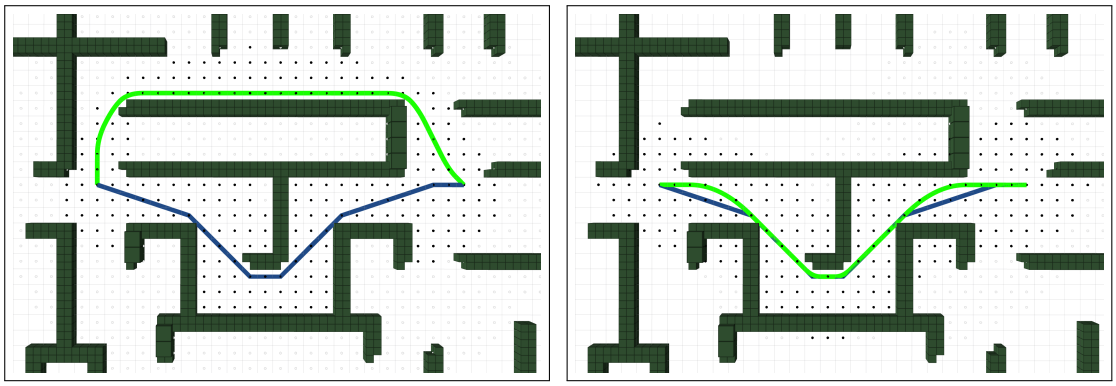


Figure 5.2: Relation of planning time and path length for the experiments on initial 2D experiment using different δ- and tunnel-sizes, without iteratively increasing the size.

tion only leads to longer planning times, a coarser velocity resolution significantly affects the path quality since low velocities have to be choosen to avoid missing turns. Thus, for both 9D state lattices, an edge duration of 0.4s is used. As the minimum change in the spatial dimension for the jerk based planner is a chain of jerk-input $j = \{+j, -j, -j, +j\}$, which corresponds to a change of several lattice nodes in the spatial dimension, it is necessary to define a goal region instead of a goal node. Otherwise, in combination with the limited planning space from the δ-Space, the planner is not able to find a valid path to every goalnode. Here, a radius of 30cm around the goalnode is used as the goalregion. For these planners, it is also necessary to find right resolutions for the dependent axis. An analysis why this could be complicated is shown in Section 5.1.2. Here, it leads to spatial resolutions of $0.16m$ for the acceleration-based planner and $\frac{4}{75}m = 0.05\overline{3}m$ for the jerk-based one. All resolutions are shown in table 5.1.

Figure 5.4 shows the lower-dimensional and higher-dimensional planning time. As the same resolution is used in the lower dimension as in the higher dimension, the 9-dimensional planner are slower in the lower dimension. They could be equalized to reduce the time, but that would lead to a slightly worse heuristics for the 9-dimensional planner. With times around $4ms$ for the lower-dimensional



(a) δ-Space
Path cost: 16s
Planning time: 96.1 ms

(b) Tunnel
Path cost: 17s
Planning time: 90.2 ms

Figure 5.3: Example of a planning task in the 2D-map. The black dots represent all nodes inside the δ-Space or tunnel, while the blue line is the shortest low-dimensional path and the green line is planned high-dimensional trajectory.

	Position X,Y	Velocity X,Y	Acceleration X,Y
9D planning with jerk	$0.05\overline{3}m$	$0.4\frac{m}{s}$	$2\frac{m}{s^2}$
9D planning with acceleration	$0.16m$	$0.8\frac{m}{s}$	$2\frac{m}{s^2}$
6D planning with average	$0.5m$	$1\frac{m}{s}$	–
6D planning with TopiCo	$0.5m$	$1\frac{m}{s}$	–

Table 5.1: Resolutions for the different state lattices

planning step of the 6-dimensional planning and up to $141ms$ for the one of the 9-dimensional planning, both are still fast enough.

For the higher-dimensional planning time, the difference is even greater. While the 6-dimensional planning is still fast with less than $20ms$, the 9-dimensional planner with acceleration takes mor than 50 times as much time, and the 9-dimensional planner with jerk even more that 600 times the time. With a planning time of $13.7s$, the planning is even slower than the average cost of $7.9s$. If we add a third axis, the planning time will even get worse, so that planning is not usable for planning in a 3D-map. As the acceleration-based planning is also already taking more than a second, it can be assumed that it is to slow too. For the 6-dimensional plannings, we have a much higher cost of around $13s$. There are two different causes: One is the resolution of $0.5m$, that gives back suboptimal accelerations as

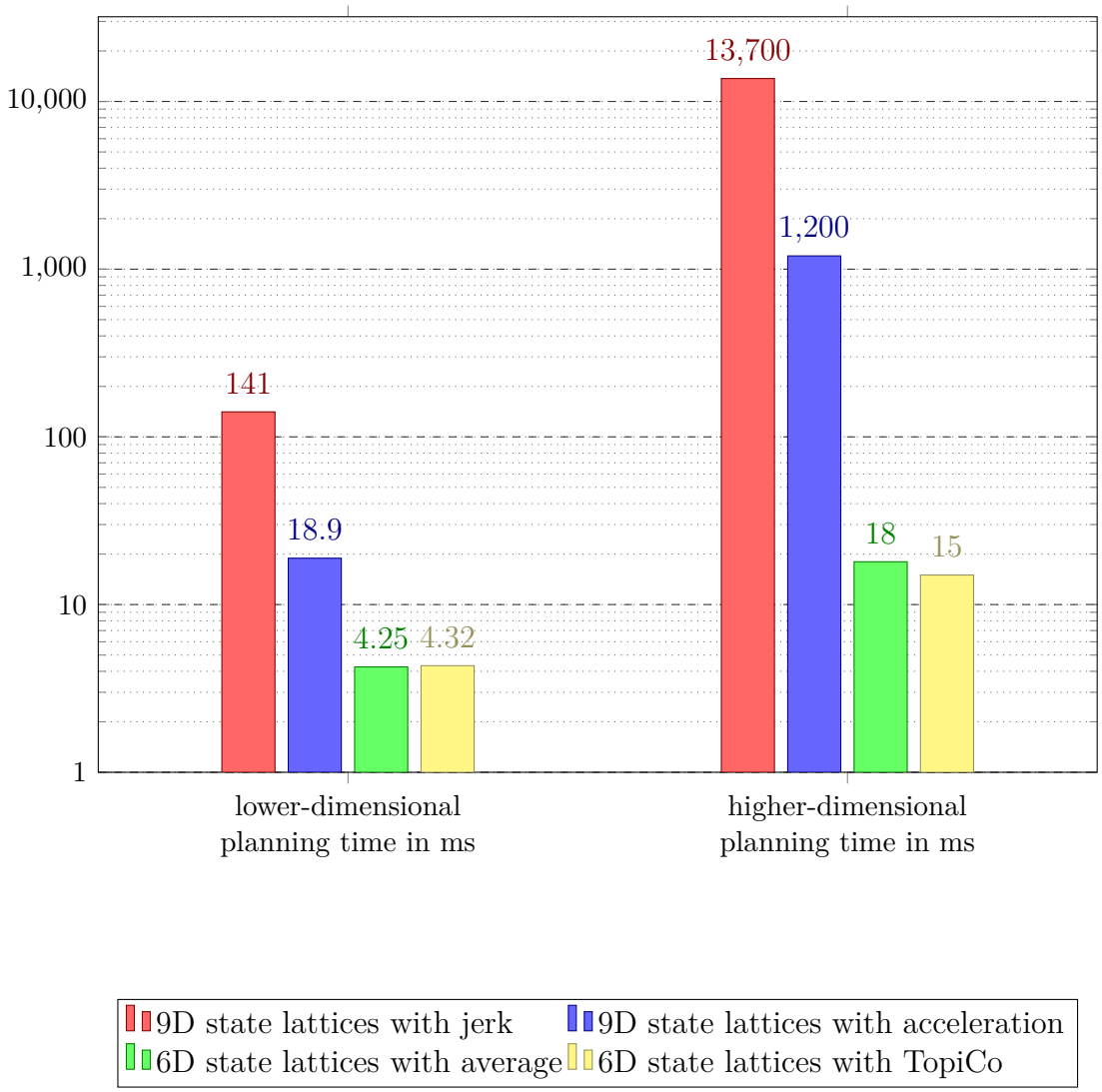


Figure 5.4: Planning time for different state lattices on the 2D-maze.

the maximum acceleration can not be used in the used resolution of the velocity. The other one is the jerk and acceleration on the TopiCo-based planner, as the acceleration is at 0 in each node, and in combination with the jerk overestimates planning cost. It is possible to reduce that cost when optimizing the trajectory, using all nodes as waypoints. Using Topico on the path again with only the positions of all pathnodes should lead to a path that is faster but close to the original one in the position-dimensions. It is also possible to try gradient-based approaches such as CHOMP. These optimizations could be part of a new project and are not a part of this thesis.

As the average-based planner doesn't lead to a better pathcost than TopiCo-

based and the path with TopiCo-edges can be directly executed by the multirotor, the 6-dimensional 6D state lattice with TopiCo is used for the following experiments.

5.1.2. On the Feasibility of 9D Planning

Above, we presented two different definitions of 9D state lattices. One that uses jerk as an input for edges, and one that uses the acceleration-dimensions as input for all outgoing edges.

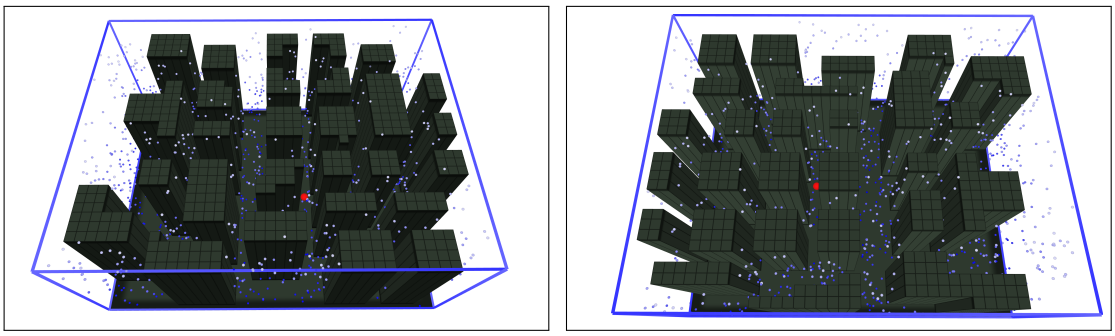
When using jerk as a input, it is not trivial to find matching lattice resolutions for acceleration, velocity and position, if the motion model doesn't match to lattice sizes. Especially as all axis need to have the same duration between nodes (independent of jerk input), as only same duration can be combined into edges. If they don't have the same duration, it is not possible to fly that way and the edge could not be used, leading to fewer edges and a worse motion model. If a constructed edge goes directly through a lattice node, that can be used as a endpoint instead. This way some edges can be shortened. As an example, when jerk and acceleration are 0, it is possible to shorten the edge by the greatest common divisor of the number of lattice cells traversed in all axis.

It is possible to calculate the lattice resolution so that the same edge duration can be used for all edges, but it could lead to a high resolution. Here a lattice should be constructed from real-world parameters of a multirotor to show the resulting resolution of the lattice. The parameters are taken from Beul, Bultmann, et al. 2020, used at the Mohamed Bin Zayed International Robotics Challenge 2020. In this model, the X and Y-axis have different parameters than the Z-axis. With a maximum jerk of $5\frac{m}{s^3}$ and a maximum acceleration of $4\frac{m}{s^2}$ for the X and Y-axis, possible durations are a fraction of $\frac{4}{5}s$, as $\frac{4}{5}s$ is the time needed to get from $a = 0$ to $a = a_{max}$ and each duration that isn't a fraction would not be able to reach the maximum acceleration. For the Z-axis with a maximum jerk of $50\frac{m}{s^3}$ and a maximum acceleration of $10\frac{m}{s^2}$, it is a fraction of $\frac{1}{5}s$. So combined, the longest possible duration would be $\frac{1}{5}s$. Using this duration, the distance between nodes in the acceleration dimension would be $\Delta a = j \cdot \Delta t = 4\frac{m}{s^3} \cdot 0.2s = 0.8\frac{m}{s^2}$ for the X and Y-axis, leading to a total of 11 lattice along the acceleration-dimension for X and Y, and a distance of $10\frac{m}{s^2}$ with 3 cells in the Z-acceleration-dimension. For the distance of the velocity lattice, we now have to look at both the change through acceleration and jerk, and find a size that matches both. For the X and Y-axis, the cellsize from the acceleration is $\Delta v = a \cdot \Delta t = \frac{4}{5}\frac{m}{s^2} \cdot \frac{1}{5}s = \frac{4}{25}\frac{m}{s}$ and from the jerk $\Delta a = \frac{1}{2}j \cdot \Delta t^2 = \frac{1}{2} \cdot 4\frac{m}{s^3} \cdot (\frac{1}{5}s)^2 = \frac{2}{25}\frac{m}{s}$. That can be combined to distance in the velocity dimension of $\frac{2}{25}\frac{m}{s}$. Although the maximum velocity is $5\frac{m}{s}$, only

$4.96\frac{m}{s}$ is used, as it is not possible to achieve the real maximum velocity with the given duration. This already results in a total of 125 cells in the X and Y velocity lattice. For the distance between nodes in the Z-velocity dimension, the distance is $2\frac{m}{s}$ from the acceleration and $1\frac{m}{s}$ from the jerk. The maximum velocity allowed in the Z-axis is only $1\frac{m}{s}$ but a positive followed by a negative negative jerk input for our duration already brings us to $2\frac{m}{s}$. To overcome this problem, there are the options to lower the maximum jerk or acceleration, shorten the duration even more or allow a maximum velocity of $2\frac{m}{s}$. Here, the latter is used, leading to a lattice with 5 cells for the Z-velocity-dimension.

Combined, for each spatial position we already get a total of $11^2 \cdot 3 \cdot 125^2 \cdot 5 \approx 28 \cdot 10^6$ cells in the velocity- and acceleration-dimensions. Even with state-of-the-art hardware for multirotors, this amount of possible nodes can only be handled for a small number of positions. And if we look at the resulting distance of of $\frac{1}{750}m$ between nodes in the spatial dimension in X and Y (as we need to be able to fly distances of $\frac{1}{150}m$, $\frac{2}{125}m$ and $\frac{1}{125}m$), a square meter already has more than half a million spatial cells, or more than 15 trillion possible cells. Only when there exist a heuristic that makes the 9-dimensional planning space really small, a multirotor is able to plan a path inside this space. Unfortunately, the δ-Space does not reduce the state space size that much (see Section 5.2.2).

5.2. 3D Experiments



(a) Map 1 (b) Map 2

Figure 5.5: Maps for planning in 3D. The red dot marks the start position, while the blue dots mark all goal positions. The brightness of the dot corresponds thereby to its height.

As mentioned in the experiments above, 6D state lattices with edges from Topico are used for the 3D experiments. In the following experiments, two different maps are used (see Figure 5.5).

They both should represent cities with high buildings, such as scyscrapers. In each experiment, a total of 1000 trajectories were computed, all from the same start state (red dot in Figure 5.5) in the map to the same random goalpositions (blue dots in Figure 5.5), that are between $1m$ and $5m$ away from a building. To have the same goalpositions in every experiment, a fixed seed for the random number generator was used.

5.2.1. Lower-dimensional Neighborhood Sizes

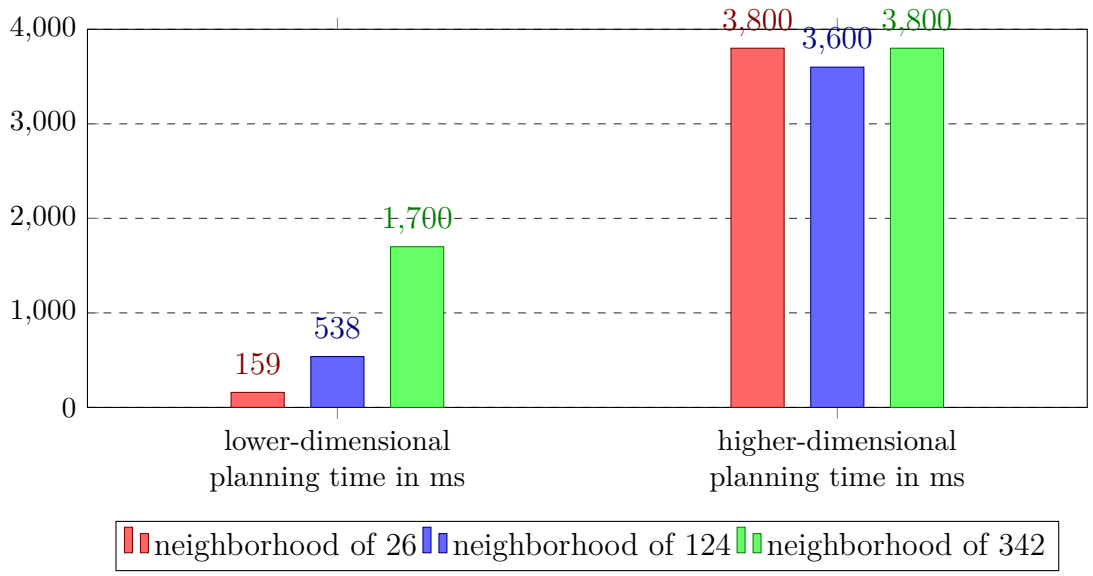


Figure 5.6: lower- and higher-dimensional planning time for different lower-dimensional neighborhood.

As we know from Section 4.2, Figure 4.4 and Figure 4.5, the size of the δ-Space is influenced by the neighborhood of the underlying graph. To analyze the influence, three cuboid neighborhoods are tested, with sizes of $3^3 - 1 = 26$, $5^3 - 1 = 124$ and $7^3 - 1 = 342$ edges. Figure 5.6 shows the corresponding planning. Since more edges have to be considered during the low-dimensional search, the planning times increase with increasing neighorhood size, from $159ms$ for the smallest tested neighborhood to $1.7s$ for the largest neighborhood.

For the higher-dimensional search, the difference between the planning times of the different neighborhoods is much smaller with about $0.2s$. However, here, the neighborhood of size 124 results in the lowest planning time, while using the larger or smaller neighborhood is slower. A main reason for the different planning times is the size of the δ-Space, which is smallest for the 124-neighborhood (see Figure 5.7) This neighborhood also has the lowest higher-dimensional pathcost (together

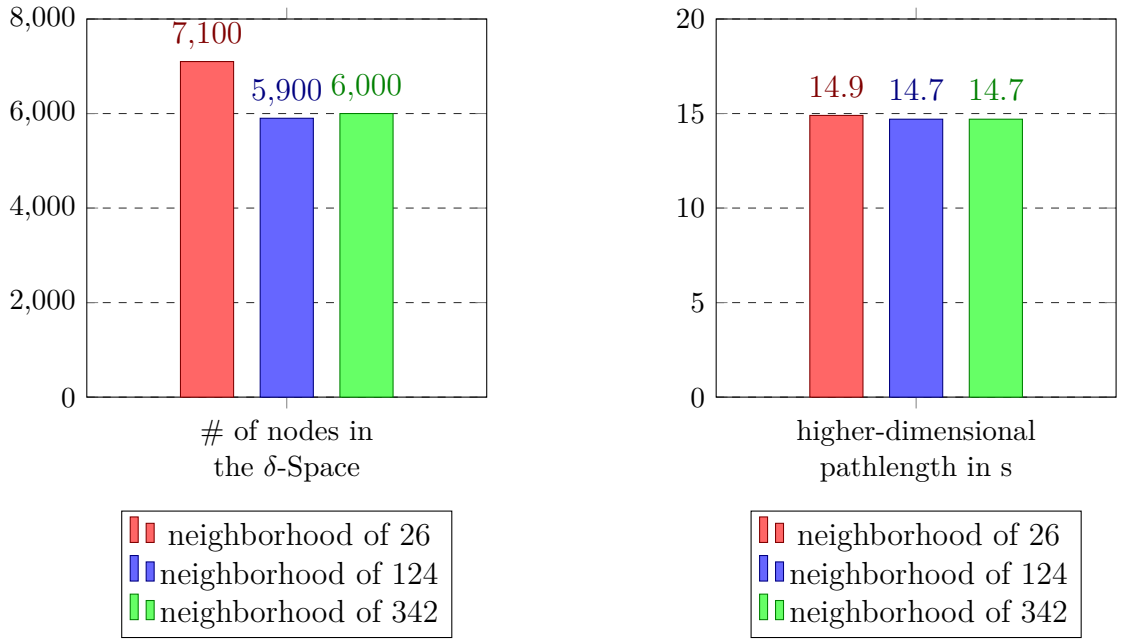


Figure 5.7: Number of nodes in the δ-Space and cost of the higher-dimensional trajectory for different lower-dimensional neighborhood on Map 1.

with the bigger neighborhood), while the pathcost of the smaller neighborhood is slightly higher. If there is a way to decrease the lower-dimensional planning time for the 124-neighborhood, this would result in the best overall performance. But it is also possible that this difference in pathcost comes from overestimating the heuristic in comparison with the smaller neighborhoods, as the higher-dimensional planner can fly diagonals that are only available in the largest neighborhood. Together with the triangle inequality we know that the lower-dimensional cost of the shortest path is the lowest for a neighborhood that includes the complete lattice.

5.2.2. Different Values of δ-Sizes

The next experiment was done to have a look at different choices for both δ and the size of the tunnel radius, and how they affect the planning. Figure 5.8 shows the number of nodes the δ-Space consists of. Here each increase of 0.5m for δ leads to roughly 3000 more nodes. This is remarkable, since in general the size of 3-dimensional spaces grows cubic. The linear instead of cubic growth results in a moderate increase of planning times. Especially when increasing the delta iteratively, linear growth in the size of the δ-Space could lead to nearly constant increases in planning time for each increase of δ (see Section 5.2.3). The lower

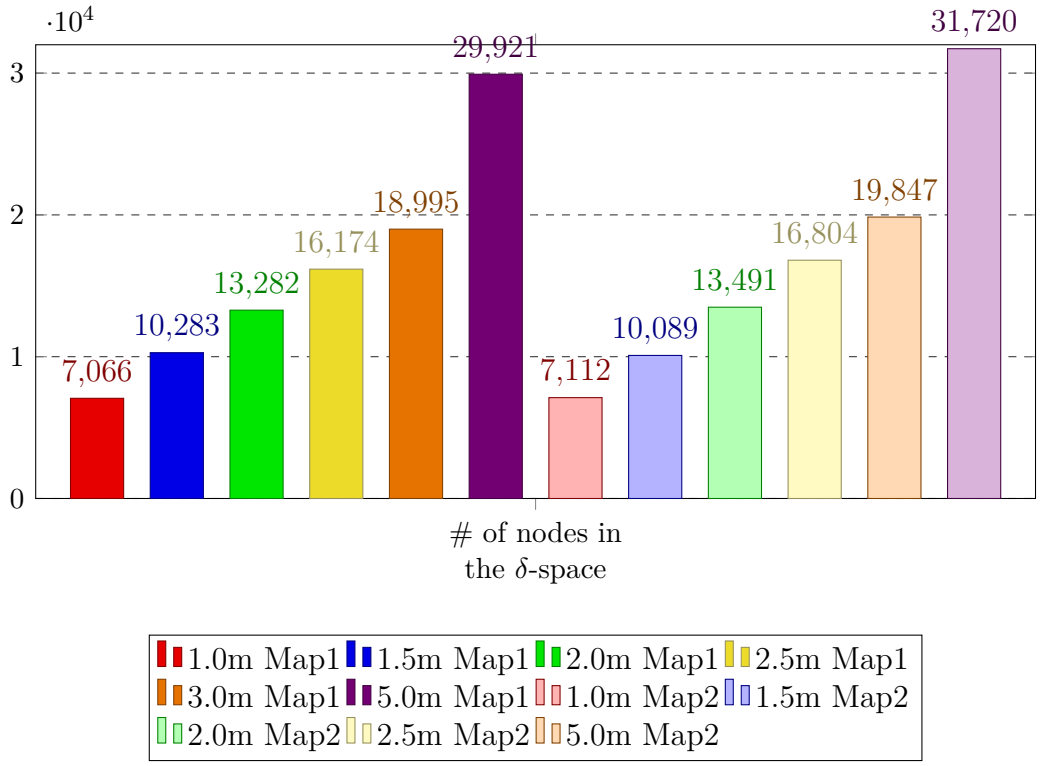


Figure 5.8: Number of nodes in the δ-Space for different δ on Map 1 and Map 2.

dimensional planning times on Map 1 (Figure 5.9) show such an increase, with the minimum increase of $7ms$ between $2.5m$ and $3m$ and a maximum increase of $19ms$ between $2m$ and $2.5m$. On average, the additional lower dimensional planning time is $28.75ms$ for each meter of increase. When combining the number of nodes with the higher-dimensional planning time, we get an average planning time per node in the δ-Space between $0.44ms$ and $0.49ms$ for Map 1 and between $0.45ms$ and $0.52ms$ on Map 2 with a deviation of less than $0.3ms$ on both maps. When assuming a planning time of $1ms$ per lower-dimensional node more than 90% of all higher-dimensional searches could be finished before time runs out. Figure 5.10 shows a boxplot of the times per node for Map 1. Note that the higher-dimensional planner has a different number of nodes, but this time can be used as an estimate for planning time after the δ-space was constructed.

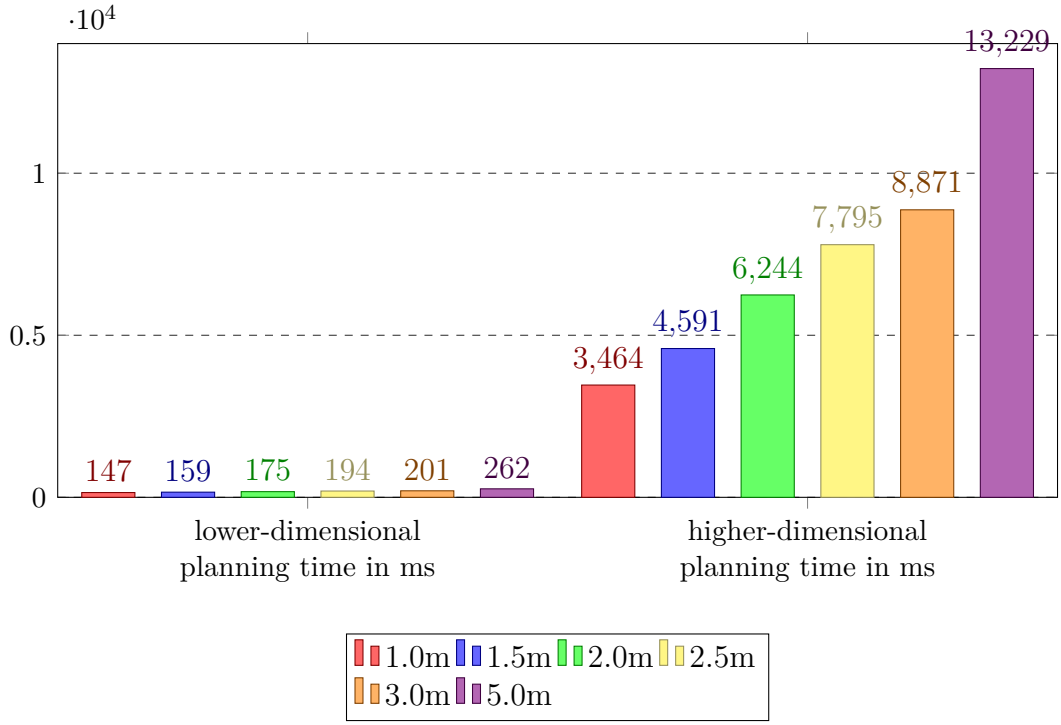


Figure 5.9: Lower- and higher-dimensional planning time for different values of δ on Map 1.

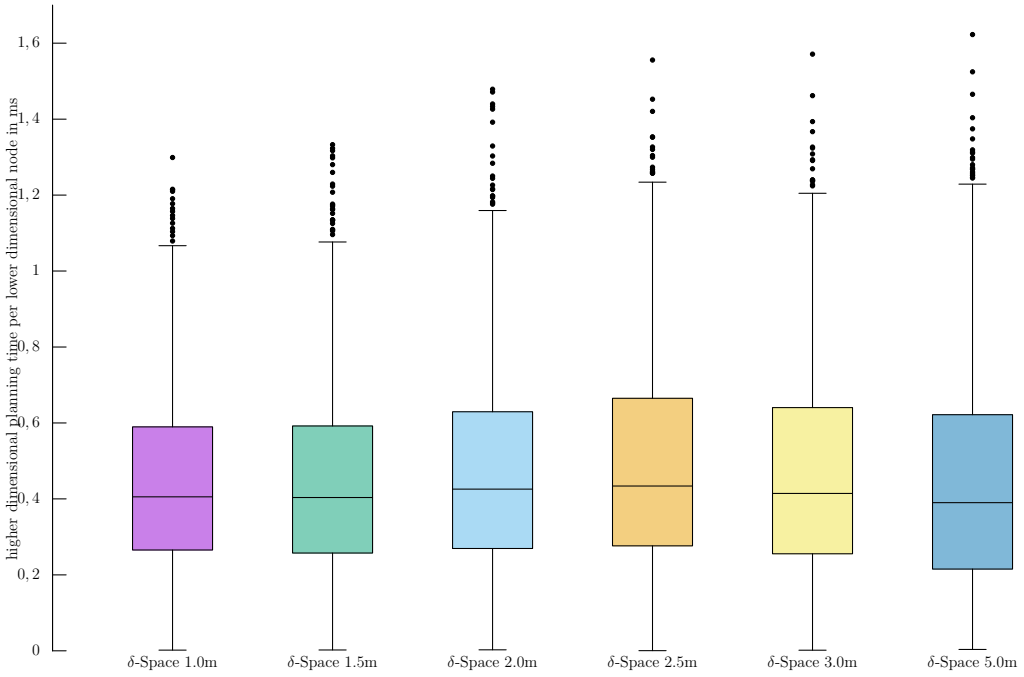


Figure 5.10: Higher-dimensional planning time per lower-dimensional node for different δ on Map 1.

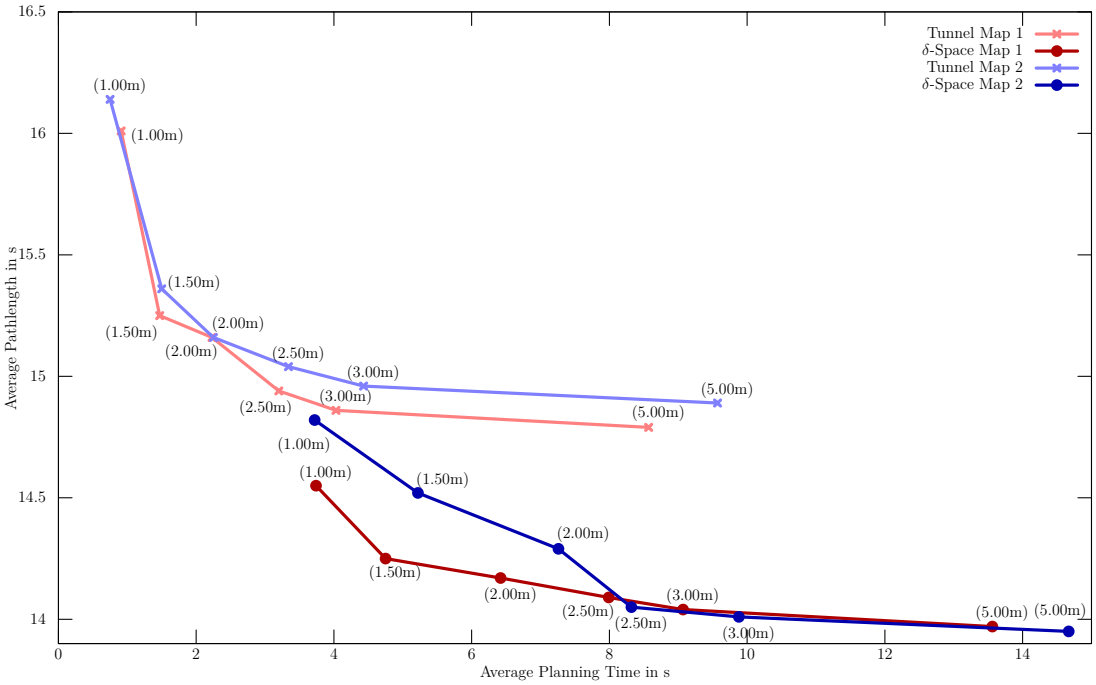


Figure 5.11: Pathcost and initial planning time for different δ and tunnel-sizes on Map 1 and Map 2.

When comparing planning times to trajectory costs for the δ-Space with the tunnel (Figure 5.11), we see that the the size of the tunnel radius should be choosen as at least $1.5m$, since $1m$ results in significantly higher trajectory costs on both maps. On the other end, even a tunnel with a radius of $5m$ still has higher trajectory cost than the smallest evaluated δ-Space with a δ of $1m$. As even the smallest δ on both maps results in planning times of nearly $4s$, using the δ-Space method is not suitable for time-critical scenarios. However, if there is enough planning time available, it is the better option in all cases. If there is only a short timeframe for planning before the multirotor should start, the size of δ shouldn't be too large, as the increase in time is greater than the decrease in the trajectory execution times. To overcome that problem, the δ should be increased iteratively.

5.2.3. Iterative δ-Space

As it is possible to increase the δ as shown in Section 4.4, the question occurs what the best initial value of δ is. Figure 5.12 shows the trajectory and planning times on Map 1 for initial sizes of the tunnel and δ-Space of $1m-2m$, with three increases

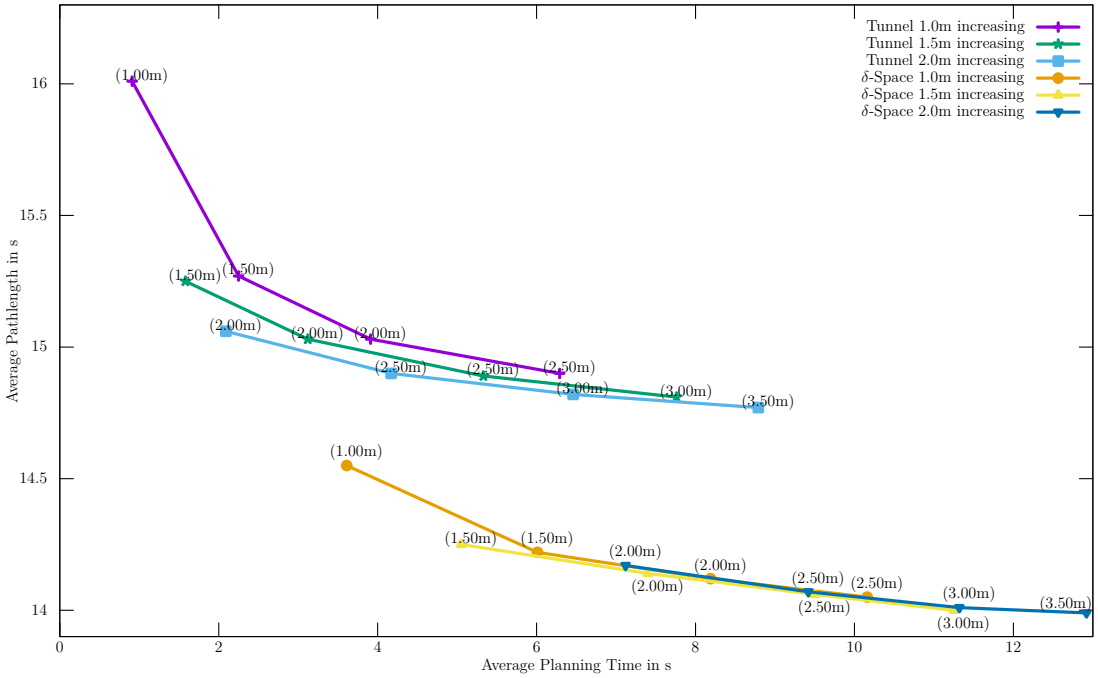


Figure 5.12: Result of the iterative model on Map 1

of $0.5m$ each. While it makes a difference in planning time and pathlength for the tunnel, the δ-Space converges to a small channel independent of the initial choice for δ. If you look closely at $\delta = 2m$ in figure 5.12, you can recognize that the initial planning for $\delta = 2m$ results in higher costs than iteratively increasing δ to $2m$. Here it seems that the additional time needed for the increase step in comparison to an initial planning in the increased δ-space is made up by the lower cost. Note that we use an inflated heuristic. Iteratively increasing δ can help to correct paths that are suboptimal due to the inadmissible heuristic. This is a possible reason why iterative searches result in lower trajectory costs. Here is an example to make clear how iterative searches can correct suboptimal choices of the inadmissible heuristic:

Assume there exist three nodes a, b and c with edges (a, c) and (b, c). The shortest path thereby goes through b and c. While node a and c are inside the smaller δ-Space, node b is only part of the larger one. In the planner that starts directly with the larger δ-Space, a is expanded first. After that, due to the weighted heuristic, c is expanded before b is, leading the A* search to ignore the edge (b, c) because c was already expanded when b was expanded. The planner that starts with the smaller δ space also expands a and c first. However, as node b is not part of the small δ-Space, it is not expanded. After increasing δ, b is inside the larger

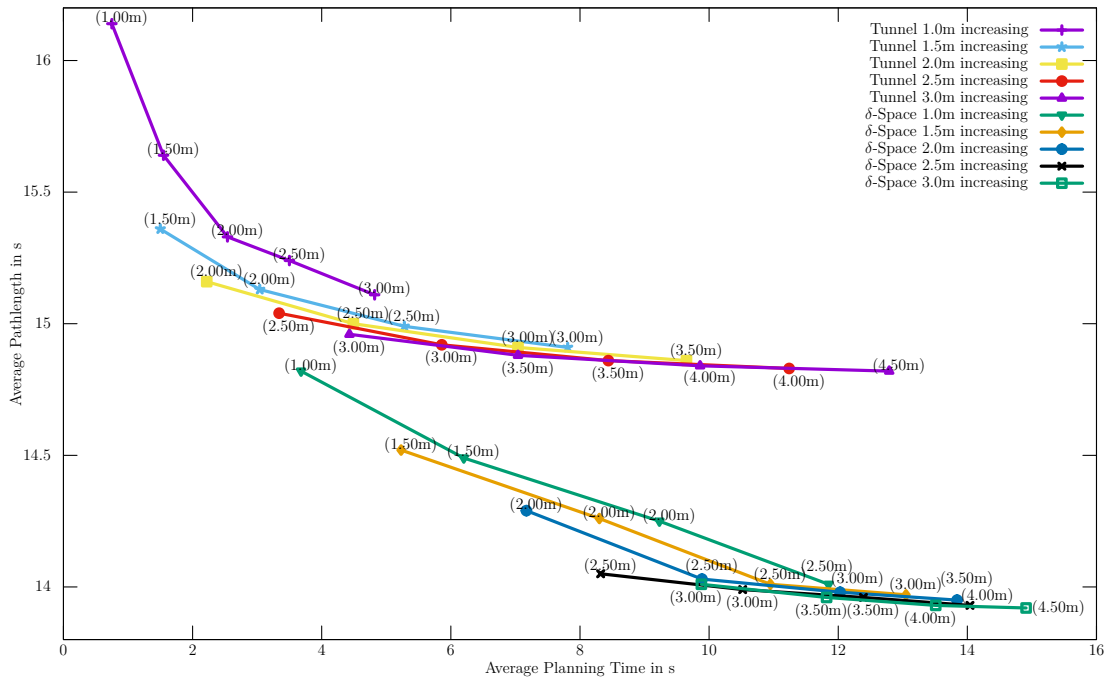


Figure 5.13: Result of the iterative model on Map 2

δ-Space and is thus expanded at some point during the replanning. Thus, the edge (b, c) is not ignored, as c was not yet expanded after the increase-step. Later, c is expanded again, together with at least all other nodes on the old shortest path. So the increase-step corrected one of the suboptimal nodes.

On Map 2 (Figure 5.13), we can also see that the increase step brings better pathcost, but here it does not cancel out the longer planning times. It still seems to converge, but that is because the decreases in pathcost for $\delta \geq 2.5m$ on are really small (between $0.03m$ and $0.05m$ when increasing δ from $2.5m$ tuo $3m$). while the increase in planning time is much higher. So for this map, a δ of $2.5m$ seems to be the point after that increasing does make much less sense. On this map you can clearly see that you can achieve a δ of $2.5m$ in a bit more than $8s$, while it takes nearly $12s$ when starting with a δ of $1m$. On the other hand, the δ of $1m$ gets the first path already after less than half the planning time.

5.2.4. Inflated Heuristics

In this experiment, we want to see the impact of different weighted heuristics on trajectory costs and planning time. Figure 5.14 shows the results for both the tunnel and the δ-Space. The colors of the graphs represents the used weight and

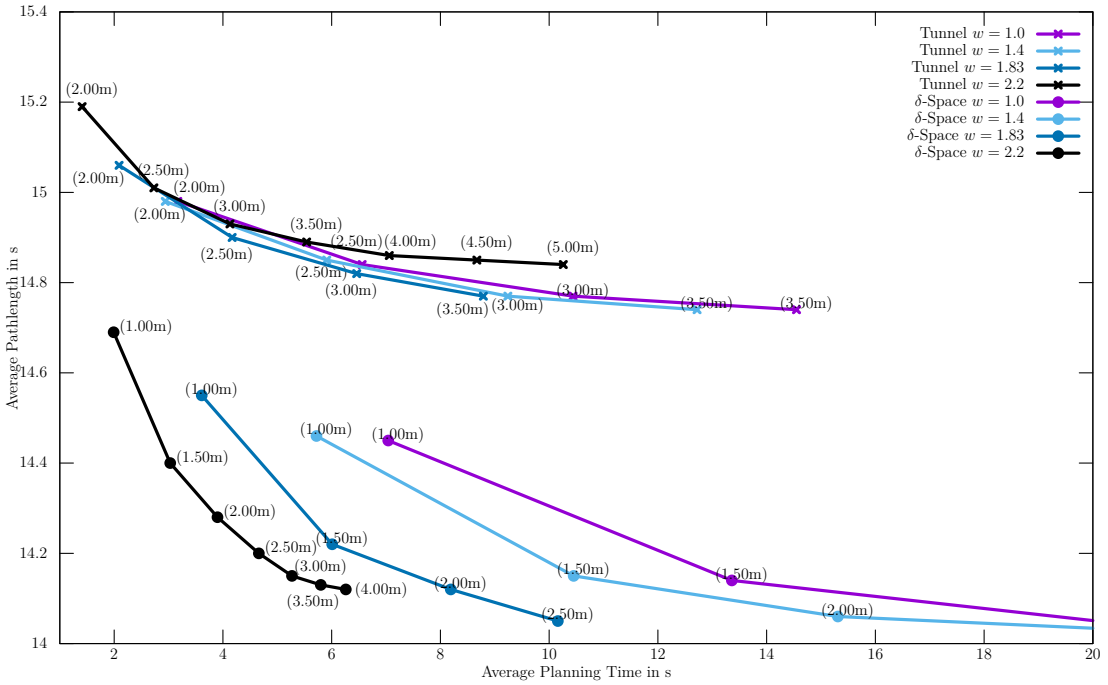


Figure 5.14: The impact of weighted heuristics

a line represents an iterative search as described in Section 5.2.3. Here, it can be clearly seen that tunnel and δ-Space behave differently. The weight does impact the tunnel slightly, increasing the tunnel radius once has nearly the same effect. However, further increases do not improve the performance significantly. On the other hand, the heuristic weight strongly influences the performance of the δ-Spaces. A larger weight significantly decreases the planning times. However, there is also a limit how much the trajectory cost can be decreased for each weight when increasing the δ-Space. This way, the δ-Space was even faster when combining the initial planning time with the execution time for $w = 2.2$.

In the end, the optimal choice of the heuristic weight depends on the maximum available planning time. High weights result in low initial planning times, but to higher trajectory costs than for lower weights as the iterative search converges. On the other hand, when using low heuristic weights, the search might be terminated after fewer iterations due to longer planning times. Thus, the resulting trajectory costs might even be higher. For the choice of δ, there is no optimal initial value for all maps, but for this map (Figure 5.14) a starting size of $\delta = 1.5$ is a good choice for all weights.

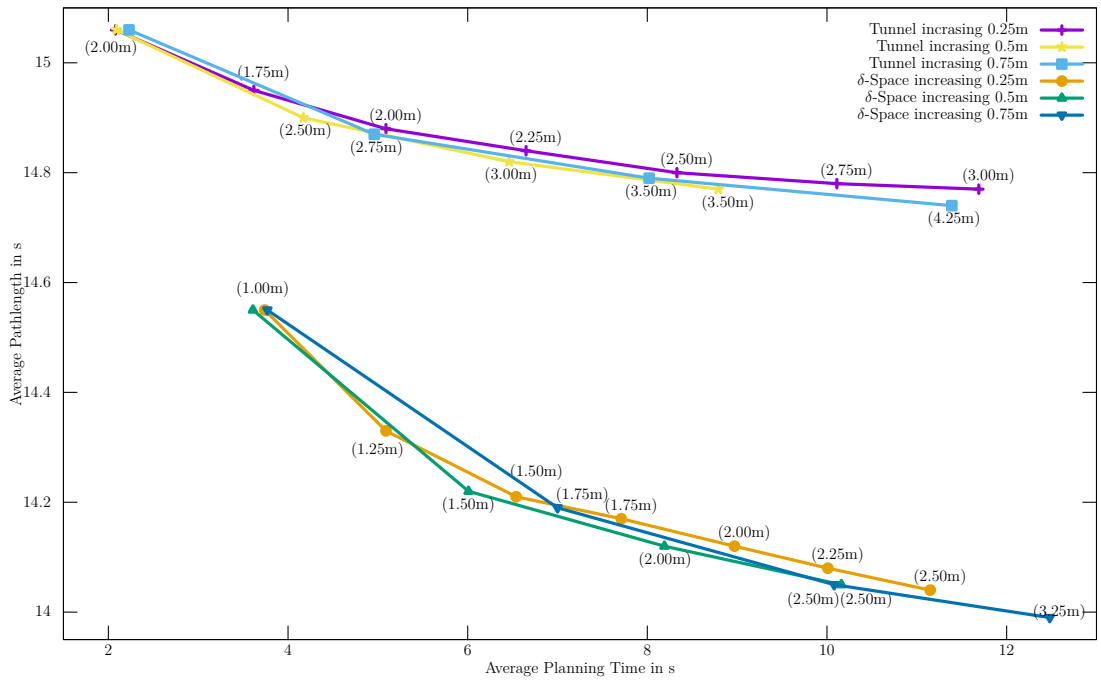


Figure 5.15: Result of the iterative model on Map 1 for different steps of δ.

5.2.5. Step Sizes for Iterative δ-Spaces

Finally, we evaluate the step size for iteratively increasing δ. Figure 5.15 compares iterative searches with different step sizes, each starting with the same initial choice of δ. The choice of the step size does not have a big influence on the performance for the δ-Space, while it has a stronger influence on the tunnel. Here, it leads to longer planning times for several small increase steps compared to a single large one. Thus, the tunnel should be increased in larger steps. For the δ-Space, planning times are only slightly increased when using small step sizes. In the end, there is no clear advantage for bigger increases of δ. On the contrary, small step sizes are better since larger ones lead to longer planning times between subsequent searches. With a suboptimal time constraint a planner with smaller increases could have a planning with a δ finished while one with a bigger increases only finished planning with a smaller δ, as the one with a larger one increased skipped the δ the other planned and the next planning with a larger δ was not yet finished.

As an example, consider a time constraint of $10s$ for the δ-Space as shown in Figure 5.15. Here, the planner with an stepsize of $0.25m$ was able to finish a δ of $2.25m$, while the planner with an stepsize of $0.5m$ only finished a δ of $2.0m$ and the planner with the biggest stepsize of $0.75m$ only finished a δ of $1.75m$.

6. Conclusion and Future Work

This thesis introduced a new planning space, the δ-Space, for higher-dimensional trajectory planning. We evaluated different parameters of the δ-Space and tested its performance against the tunnel as another planning space. Our experiments show that using the δ-Space results in faster trajectories. However, it also needs more planning time, especially without inflated heuristics. With inflated heuristic, the δ-Space was able to significantly decrease the planning time while the pathcost increased only slightly. As the tunnel was nearly unaffected by inflated heuristics, the δ-Space was able to outperfom the tunnel for heigh weights in the initial planning when adding planning time and execution time together.

To overcome the worse initial planning time and to be able to find better path, the δ-Space performs good when increasing the size of the δ-Space. Especially as each increase of the δ-Space is able to correct some paths that were suboptimal due to inflated heuristic. But as there is also a limit how much the trajectory cost can be decreased for each weight when increasing the δ, it should be considered to combine the δ-Space with an algorithm such as ARA* (Likhachev, Gordon, and Thrun 2003). Thus, it could be possible to combine decreasing weights with increasing δ-Spaces to get the best from both algorithms. As the experiments also showed that it is faster to start with a larger δ instead of starting with a smaller one, the initial value of δ needs to be guessed. Here, we showed that there is a correlation between the number of nodes in the lower-dimensional δ-Space and the higher-dimensional planning time. For a given map, heuristic weight and the number of nodes in the lower-dimensional δ-Space a probability distribution for the higher-dimensional planning time can be calculated. Our algorithm can be adapt to be able to calculate the highest δ that is lower or equal to a desired number of nodes.

While the experiments show that a neighborhood of 124 was the best for the higher dimensional planner both in planning time and pathcost, the lower dimensional planner took too much time. To overcome that an adapted Theta* or Lazy Theta* as described in (Kosenko and Schräder 2018) instead of A* could be used. Here, when only one shortcut in Theta* is allowed, the neighborhood of 124 could be simulated while each node still has only 26 neighbors. As the additional time for the Theta* should be less than the time that is saved for higher-dimensional

planning when using the neighborhood of 124, it should reduce the planning time as well as the pathcosts. Another experiment that could be done in upcoming works is using the δ-Space together with sampling-based planners such as RRT or AIT* (Strub and Jonathan D. Gammell 2020) for both the higher and lower dimension, or only for the lower one. Especially AIT* should work good, as the internal heuristic of the δ-Space can be seen as the graph-based heuristic of AIT* in the lower dimensional space. As a last point, the δ-Space could be combined with a multiresolution state lattice like the one described in Behnke 2003 for lower- and higher-dimensional planning. Especially in combination with TopiCo, it is possible to use multiresolution in the higher-dimensional planning space, as additional sets of edges for transition areas can be precomputed with TopiCo.

All in all, we have dveleoped an approach to reduce higher-dimensional planning spaces while reducing suboptimalities of searched trajectories in these spaces.

A. Experiment result

In this appendix, you find the raw data for all the experiments.

A.1. Evaluation of Edge Types

	lower-dimensional planning time	higher-dimensional planning time	# of nodes in the δ-Space	higher-dimensional pathcost
9D state lattice with jerk	$141ms$	$13.7s$	$25.7 \cdot 10^3$	$7.85s$
9D state lattice with acceleration	$18.9ms$	$1.2s$	$2.9 \cdot 10^3$	$7.98s$
6D state lattice with average	$4.25ms$	$0.018s$	$0.3 \cdot 10^3$	$13.04s$
6D state lattice with TopiCo	$4.32ms$	$0.015s$	$0.3 \cdot 10^3$	$13.00s$

Table A.1: Data of Experiment 5.4

A.2. Lower-dimensional Neighborhood Sizes

	lower-dimensional planning time	higher-dimensional planning time	# of nodes in the δ-Space	higher-dimensional pathcost
Neighborhood of 26	$159ms$	$3.8s$	$7.1 \cdot 10^3$	$14.9s$
Neighborhood of 124	$538ms$	$3.6s$	$5.9 \cdot 10^3$	$14.7s$
Neighborhood of 342	$1.7s$	$3.8s$	$6.0 \cdot 10^3$	$14.7s$

Table A.2: Results of Experiment 5.2.1

A.3. Different Values of δ-Sizes

	lower-dimensional planning time	higher-dimensional planning time	# of nodes in the δ-Space	higher-dimensional pathcost
δ-Space 1.0m Map 1	$147ms$	$3.5s$	$7.1 \cdot 10^3$	$14.55s$
δ-Space 1.5m Map 1	$159ms$	$4.6s$	$10.3 \cdot 10^3$	$14.25s$
δ-Space 2.0m Map 1	$246ms$	$6.9s$	$13.3 \cdot 10^3$	$14.17s$
δ-Space 2.5m Map 1	$194ms$	$7.8s$	$16.2 \cdot 10^3$	$14.09s$
δ-Space 3.0m Map 1	$201ms$	$8.9s$	$19.0 \cdot 10^3$	$14.04s$
δ-Space 5.0m Map 1	$262ms$	$13.3s$	$29.9 \cdot 10^3$	$13.97s$
δ-Space 1.0m Map 2	$154ms$	$3.5s$	$7.1 \cdot 10^3$	$14.82s$
δ-Space 1.5m Map 2	$166ms$	$5.1s$	$10.1 \cdot 10^3$	$14.52s$
δ-Space 2.0m Map 2	$181ms$	$7.0s$	$13.5 \cdot 10^3$	$14.29s$
δ-Space 2.5m Map 2	$195ms$	$8.1s$	$16.8 \cdot 10^3$	$14.05s$
δ-Space 3.0m Map 2	$211ms$	$9.7s$	$19.8 \cdot 10^3$	$14.01s$
δ-Space 5.0m Map 2	$267ms$	$14.4s$	$31.7 \cdot 10^3$	$13.95s$

Table A.3: Results for the δ-Space of Experiment 5.2.2

	lower-dimensional planning time	higher-dimensional planning time	higher-dimensional pathcost
Tunnel 1.0m Map 1	$116ms$	$0.79s$	$16.01s$
Tunnel 1.5m Map 1	$106ms$	$1.48s$	$15.25s$
Tunnel 2.0m Map 1	$96ms$	$1.99s$	$15.06s$
Tunnel 2.5m Map 1	$99ms$	$3.10s$	$14.94s$
Tunnel 3.0m Map 1	$96ms$	$3.93s$	$14.86s$
Tunnel 5.0m Map 1	$98ms$	$8.47s$	$14.79s$
Tunnel 1.0m Map 2	$99ms$	$0.65s$	$16.14s$
Tunnel 1.5m Map 2	$100ms$	$1.40s$	$15.36s$
Tunnel 2.0m Map 2	$100ms$	$2.12s$	$15.16s$
Tunnel 2.5m Map 2	$99ms$	$3.24s$	$15.04s$
Tunnel 3.0m Map 2	$100ms$	$4.33s$	$14.96s$
Tunnel 5.0m Map 2	$99ms$	$9.48s$	$14.89s$

Table A.4: Results for the Tunnel of Experiment 5.2.2

A.4. Iterative δ-Space

		lower-dimensional planning time	higher-dimensional planning time	# of nodes in the δ-Space	higher-dimensional pathcost
δ-Space iterative Map 1	$\delta = 1.0m$	$147ms$	$3.5s$	$7.1 \cdot 10^3$	$14.55s$
	$\delta = 1.5m$	$+17.9ms$	$+2.4s$	$10.3 \cdot 10^3$	$14.22s$
	$\delta = 2.0m$	$+18.0ms$	$+2.2s$	$13.3 \cdot 10^3$	$14.12s$
	$\delta = 2.5m$	$+18.2ms$	$+1.9s$	$16.2 \cdot 10^3$	$14.05s$
δ-Space iterative Map 1	$\delta = 1.5m$	$159ms$	$4.6s$	$10.3 \cdot 10^3$	$14.25s$
	$\delta = 2.0m$	$+18.2ms$	$+2.2s$	$13.3 \cdot 10^3$	$14.14s$
	$\delta = 2.5m$	$+18.4ms$	$+2.0s$	$16.2 \cdot 10^3$	$14.06s$
	$\delta = 3.0m$	$+18.5ms$	$+1.6s$	$19.0 \cdot 10^3$	$14.00s$
δ-Space iterative Map 1	$\delta = 2.0m$	$246ms$	$6.9s$	$13.3 \cdot 10^3$	$14.17s$
	$\delta = 2.5m$	$+21.5ms$	$+2.3s$	$16.2 \cdot 10^3$	$14.07s$
	$\delta = 3.0m$	$+21.9ms$	$+1.9s$	$19.0 \cdot 10^3$	$14.01s$
	$\delta = 3.5m$	$+22.4ms$	$+1.6s$	$21.7 \cdot 10^3$	$13.99s$
δ-Space iterative Map 2	$\delta = 1.0m$	$154ms$	$3.5s$	$7.1 \cdot 10^3$	$14.82s$
	$\delta = 1.5m$	$+18.4ms$	$+2.5s$	$10.1 \cdot 10^3$	$14.49s$
	$\delta = 2.0m$	$+18.6ms$	$+3.0s$	$13.5 \cdot 10^3$	$14.25s$
	$\delta = 2.5m$	$+18.7ms$	$+2.6s$	$16.8 \cdot 10^3$	$14.01s$
δ-Space iterative Map 2	$\delta = 1.5m$	$166ms$	$5.1s$	$10.1 \cdot 10^3$	$14.52s$
	$\delta = 2.0m$	$+18.6ms$	$+3.0s$	$13.5 \cdot 10^3$	$14.26s$
	$\delta = 2.5m$	$+18.6ms$	$+2.6s$	$16.8 \cdot 10^3$	$14.01s$
	$\delta = 3.0m$	$+18.8ms$	$+2.1s$	$19.8 \cdot 10^3$	$13.97s$
δ-Space iterative Map 2	$\delta = 2.0m$	$181ms$	$7.0s$	$13.5 \cdot 10^3$	$14.29s$
	$\delta = 2.5m$	$+18.5ms$	$+2.7s$	$16.8 \cdot 10^3$	$14.03s$
	$\delta = 3.0m$	$+18.7ms$	$+2.1s$	$19.8 \cdot 10^3$	$13.98s$
	$\delta = 3.5m$	$+18.8ms$	$+1.8s$	$22.8 \cdot 10^3$	$13.95s$
δ-Space iterative Map 2	$\delta = 2.5m$	$195ms$	$8.1s$	$16.8 \cdot 10^3$	$14.05s$
	$\delta = 3.0m$	$+18.9ms$	$+2.2s$	$19.8 \cdot 10^3$	$13.99s$
	$\delta = 3.5m$	$+19.0ms$	$+1.8s$	$22.8 \cdot 10^3$	$13.96s$
	$\delta = 4.0m$	$+19.3ms$	$+1.6s$	$25.8 \cdot 10^3$	$13.93s$
δ-Space iterative Map 2	$\delta = 3.0m$	$211ms$	$9.7s$	$19.8 \cdot 10^3$	$14.01s$
	$\delta = 3.5m$	$+19.0ms$	$+1.9s$	$22.8 \cdot 10^3$	$13.96s$
	$\delta = 4.0m$	$+19.3ms$	$+1.7s$	$25.8 \cdot 10^3$	$13.93s$
	$\delta = 4.5m$	$+19.4ms$	$+1.4s$	$28.7 \cdot 10^3$	$13.92s$

Table A.5: Results for the δ-Space of Experiment 5.2.3

		lower-dimensional planning time	higher-dimensional planning time	higher-dimensional pathcost
Tunnel iterative Map 1	radius= 1.0m	116ms	0.79s	16.01s
	radius= 1.5m	+1.33s		15.27s
	radius= 2.0m	+1.68s		15.03s
	radius= 2.5m	+2.38s		14.90s
Tunnel iterative Map 1	radius= 1.5m	106ms	1.48s	15.25s
	radius= 2.0m	+1.54s		15.03s
	radius= 2.5m	+2.21s		14.89s
	radius= 3.0m	+2.43s		14.81s
Tunnel iterative Map 1	radius= 2.0m	96ms	1.99s	15.06s
	radius= 2.5m	+2.08s		14.90s
	radius= 3.0m	+2.29s		14.82s
	radius= 3.5m	+2.34s		14.77s
Tunnel iterative Map 2	radius= 1.0m	99ms	0.65s	16.14s
	radius= 1.5m	+1.17s		15.37s
	radius= 2.0m	+1.56s		15.13s
	radius= 2.5m	+2.26s		14.99s
Tunnel iterative Map 2	radius= 1.5m	100ms	1.40s	15.36s
	radius= 2.0m	+1.53s		15.13s
	radius= 2.5m	+2.25s		14.99s
	radius= 3.0m	+2.52s		14.91s
Tunnel iterative Map 2	radius= 2.0m	100ms	2.12s	15.16s
	radius= 2.5m	+2.27s		15.00s
	radius= 3.0m	+2.55s		14.91s
	radius= 3.5m	+2.61s		14.86s
Tunnel iterative Map 2	radius= 2.5m	99ms	3.24s	15.04s
	radius= 3.0m	+2.52s		14.92s
	radius= 3.5m	+2.58s		14.86s
	radius= 4.0m	+2.80s		14.83s
Tunnel iterative Map 2	radius= 3.0m	100ms	4.33s	14.96s
	radius= 3.5m	+2.61s		14.88s
	radius= 4.0m	+2.82s		14.84s
	radius= 4.5m	+2.93s		14.82s

Table A.6: Results for the Tunnel of Experiment 5.2.3

A.5. Inflated Heuristics

		lower-dimensional planning time	higher-dimensional planning time	# of nodes in the δ-Space	higher-dimensional pathcost
δ-Space iterative $w = 1.0$	$\delta = 1.0m$	$150ms$	$6.9s$	$7.1 \cdot 10^3$	$14.45s$
	$\delta = 1.5m$	$+18.2ms$	$+6.3s$	$10.3 \cdot 10^3$	$14.14s$
	$\delta = 2.0m$	$+18.6ms$	$+6.7s$	$13.3 \cdot 10^3$	$14.05s$
	$\delta = 2.5m$	$+18.8ms$	$+6.1s$	$16.2 \cdot 10^3$	$13.98s$
δ-Space iterative $w = 1.4$	$\delta = 1.0m$	$159ms$	$5.6s$	$7.1 \cdot 10^3$	$14.46s$
	$\delta = 1.5m$	$+18.1ms$	$+4.7s$	$10.3 \cdot 10^3$	$14.15s$
	$\delta = 2.0m$	$+18.4ms$	$+4.8s$	$13.3 \cdot 10^3$	$14.06s$
	$\delta = 2.5m$	$+18.8ms$	$+4.4s$	$16.2 \cdot 10^3$	$13.98s$
δ-Space iterative $w = 1.83$	$\delta = 1.0m$	$147ms$	$3.5s$	$7.1 \cdot 10^3$	$14.55s$
	$\delta = 1.5m$	$+17.9ms$	$+2.4s$	$10.3 \cdot 10^3$	$14.22s$
	$\delta = 2.0m$	$+18.0ms$	$+2.2s$	$13.3 \cdot 10^3$	$14.12s$
	$\delta = 2.5m$	$+18.2ms$	$+1.9s$	$16.2 \cdot 10^3$	$14.05s$
δ-Space iterative $w = 2.2$	$\delta = 1.0m$	$154ms$	$1.8s$	$7.1 \cdot 10^3$	$14.69s$
	$\delta = 1.5m$	$+18.0ms$	$+1.0s$	$10.2 \cdot 10^3$	$14.40s$
	$\delta = 2.0m$	$+18.3ms$	$+0.92s$	$13.3 \cdot 10^3$	$14.28s$
	$\delta = 2.5m$	$+18.6ms$	$+0.74s$	$16.2 \cdot 10^3$	$14.20s$
	$\delta = 3.0m$	$+18.9ms$	$+0.60s$	$19.0 \cdot 10^3$	$14.15s$
	$\delta = 3.5m$	$+19.0ms$	$+0.51s$	$21.7 \cdot 10^3$	$14.13s$
	$\delta = 4.0m$	$+19.3ms$	$+0.44s$	$24.4 \cdot 10^3$	$14.12s$

Table A.7: Results for the δ-Space of Experiment 5.2.4

		lower-dimensional planning time	higher-dimensional planning time	higher-dimensional pathcost
Tunnel iterative $w = 1.0$	radius= $2.0m$	$97ms$	$3.07s$	$14.98s$
	radius= $2.5m$	$+3.39s$		$14.84s$
	radius= $3.0m$	$+3.88s$		$14.77s$
	radius= $3.5m$	$+4.11s$		$14.74s$
Tunnel iterative $w = 1.4$	radius= $2.0m$	$103ms$	$2.84s$	$14.98s$
	radius= $2.5m$	$+2.97s$		$14.85s$
	radius= $3.0m$	$+3.33s$		$14.77s$
	radius= $3.5m$	$+3.48s$		$14.74s$
Tunnel iterative $w = 1.83$	radius= $2.0m$	$100ms$	$2.12s$	$15.16s$
	radius= $2.5m$	$+2.27s$		$15.00s$
	radius= $3.0m$	$+2.55s$		$14.91s$
	radius= $3.5m$	$+2.61s$		$14.86s$
Tunnel iterative $w = 2.2$	radius= $2.0m$	$99ms$	$1.31s$	$15.19s$
	radius= $2.5m$	$+1.32s$		$15.01s$
	radius= $3.0m$	$+1.4s$		$14.93s$
	radius= $3.5m$	$+1.41s$		$14.89s$
	radius= $4.0m$	$+1.52s$		$14.86s$
	radius= $4.5m$	$+1.60s$		$14.85s$
	radius= $4.5m$	$+1.59s$		$14.84s$

Table A.8: Results for the Tunnel of Experiment 5.2.4

A.6. Step Sizes for Iterative δ-Spaces

		lower-dimensional planning time	higher-dimensional planning time	lower-dimensional pathcost	# of nodes in the δ-Space
δ-Space iterative stepsize $0.25m$	$\delta = 1.0m$	$155ms$	$3.6s$	$7.1 \cdot 10^3$	$14.55s$
	$\delta = 1.25m$	$+10.7ms$	$+1.3s$	$8.6 \cdot 10^3$	$14.33s$
	$\delta = 1.5m$	$+10.8ms$	$+1.4s$	$10.3 \cdot 10^3$	$14.21s$
	$\delta = 1.75m$	$+10.8ms$	$+1.2s$	$11.6 \cdot 10^3$	$14.17s$
	$\delta = 2.0m$	$+10.6ms$	$+1.3s$	$13.3 \cdot 10^3$	$14.12s$
	$\delta = 2.25m$	$+9.1ms$	$+1.0s$	$14.6 \cdot 10^3$	$14.08s$
	$\delta = 2.5m$	$+13.1ms$	$+1.1s$	$16.2 \cdot 10^3$	$14.04s$
δ-Space iterative stepsize $0.5m$	$\delta = 1.0m$	$147ms$	$3.5s$	$7.1 \cdot 10^3$	$14.55s$
	$\delta = 1.5m$	$+17.9ms$	$+2.4s$	$10.3 \cdot 10^3$	$14.22s$
	$\delta = 2.0m$	$+18.0ms$	$+2.2s$	$13.3 \cdot 10^3$	$14.12s$
	$\delta = 2.5m$	$+18.2ms$	$+1.9s$	$16.2 \cdot 10^3$	$14.05s$
δ-Space iterative stepsize $0.75m$	$\delta = 1.0m$	$159ms$	$3.6s$	$7.1 \cdot 10^3$	$14.55s$
	$\delta = 1.75m$	$+25.1ms$	$+3.2s$	$11.6 \cdot 10^3$	$14.19s$
	$\delta = 2.5m$	$+25.7ms$	$+3.1s$	$16.2 \cdot 10^3$	$14.05s$
	$\delta = 3.25m$	$+26.2ms$	$+2.4s$	$20.4 \cdot 10^3$	$13.99s$
Tunnel iterative stepsize $0.25m$	radius= $2.0m$	$96ms$	$1.99s$	–	$15.06s$
	radius= $2.25m$	$+1.53s$		–	$14.95s$
	radius= $2.5m$	$+1.47s$		–	$14.88s$
	radius= $2.75m$	$+1.56s$		–	$14.84s$
	radius= $3.0m$	$+1.68s$		–	$14.80s$
	radius= $3.25m$	$+1.78s$		–	$14.78s$
	radius= $3.5m$	$+1.58s$		–	$14.77s$
Tunnel iterative stepsize $0.5m$	radius= $2.0m$	$96ms$	$1.99s$	–	$15.06s$
	radius= $2.5m$	$+2.08s$		–	$14.90s$
	radius= $3.0m$	$+2.29s$		–	$14.82s$
	radius= $3.5m$	$+2.34s$		–	$14.77s$
Tunnel iterative stepsize $0.75m$	radius= $2.0m$	$103ms$	$2.12s$	–	$15.06s$
	radius= $2.75m$	$+2.73s$		–	$14.87s$
	radius= $3.5m$	$+3.06s$		–	$14.79s$
	radius= $4.25m$	$+3.37s$		–	$14.74s$

Table A.9: Results for Experiment 5.2.5

List of Figures

List of Tables

Bibliography

Arney, Timothy (2007). "Dynamic path planning and execution using b-splines." In: *2007 third international conference on information and automation for sustainability*, pp. 1–6.

Behnke, Sven (2003). "Local multiresolution path planning." In: *Robocup 2003: robot soccer world cup VII*. Ed. by Daniel Polani, Brett Browning, Andrea Bonarini, and Kazuo Yoshida. Vol. 3020. Lecture Notes in Computer Science. Springer, pp. 332–343. URL: https://doi.org/10.1007/978-3-540-25940-4%5C_29.

Beul, Marius and Sven Behnke (June 2017). "Fast full state trajectory generation for multirotors." In: *Proceedings of International Conference on Unmanned Aircraft Systems (ICUAS)*. Miami, FL, USA. URL: https://github.com/AIS-Bonn/TopiCo.

Beul, Marius, Simon Bultmann, Andre Rochow, Radu Alexandru Rosu, Daniel Schleich, Malte Splietker, and Sven Behnke (2020). "Visually guided balloon popping with an autonomous MAV at MBZIRC 2020." In: *IEEE international symposium on safety, security, and rescue robotics, SSRR 2020, abu dhabi, united arab emirates, november 4-6, 2020*. IEEE, pp. 34–41. URL: https://doi.org/10.1109/SSRR50563.2020.9292612.

Dijkstra, Edsger W. (1959). "A note on two problems in connexion with graphs." In: *Numerische mathematik* 1, pp. 269–271. URL: https://doi.org/10.1007/BF01386390.

Gammell, Jonathan D (Feb. 2017). "Informed anytime search for continuous planning problems." PhD thesis. University of Toronto.

Gammell, Jonathan D., Timothy D. Barfoot, and Siddhartha S. Srinivasa (2020). "Batch informed trees (BIT*): informed asymptotically optimal anytime search." In: *Int. j. robotics res.* 39.5. URL: https://doi.org/10.1177/0278364919890396.

Gammell, Jonathan D., Siddhartha S. Srinivasa, and Timothy D. Barfoot (2014). "Informed rrt*: optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic." In: *2014 IEEE/RSJ international conference on intelligent robots and systems, chicago, il, usa, september 14-18, 2014*. IEEE, pp. 2997–3004. URL: https://doi.org/10.1109/IROS.2014.6942976.

Gochev, Kalin, Benjamin J. Cohen, Jonathan Butzke, Alla Safonova, and Maxim Likhachev (2011). "Path planning with adaptive dimensionality." In: *Proceedings of the fourth annual symposium on combinatorial search, SOCS 2011, castell de cardona, barcelona, spain, july 15.16, 2011*. Ed. by Daniel Borrajo, Maxim

Likhachev, and Carlos Linares López. AAAI Press. URL: http://www.aaai.org/ocs/index.php/SOCS/SOCS11/paper/view/4037.

Hart, Peter E., Nils J. Nilsson, and Bertram Raphael (1968). "A formal basis for the heuristic determination of minimum cost paths." In: *IEEE trans. syst. sci. cybern.* 4.2, pp. 100–107. URL: https://doi.org/10.1109/TSSC.1968.300136.

Korte, Bernhard and Jens Vygen (2008). *Kombinatorische optimierung: theorie und algorithmen*. Springer-Verlag. ISBN: 978-3-540-76919-4. URL: http://www.springer.com/us/book/9783540769194.

Kosenko, Oleg and Sebastian Schräder (2018). "3D path planning for micro aerial vehicles." Lab Report Cognitive Robotics, University of Bonn.

LaValle, Steven M et al. (1998). "Rapidly-exploring random trees: a new tool for path planning." In:

Likhachev, Maxim, Geoffrey J. Gordon, and Sebastian Thrun (2003). "Ara*: anytime a* with provable bounds on sub-optimality." In: *Advances in neural information processing systems 16 [neural information processing systems, NIPS 2003, december 8-13, 2003, vancouver and whistler, british columbia, canada]*. Ed. by Sebastian Thrun, Lawrence K. Saul, and Bernhard Schölkopf. MIT Press, pp. 767–774. URL: https://proceedings.neurips.cc/paper/2003/hash/ee8fe9093fbbb687bef15a38facc44d2-Abstract.html.

Liu, Sikang, Kartik Mohta, Nikolay Atanasov, and Vijay Kumar (2017). "Search-based motion planning for aggressive flight in SE(3)." In: *CoRR* abs/1710.02748. arXiv: 1710.02748. URL: http://arxiv.org/abs/1710.02748.

Nieuwenhuisen, Matthias and Sven Behnke (2019). "Search-based 3d planning and trajectory optimization for safe micro aerial vehicle flight under sensor visibility constraints." In: *CoRR* abs/1903.05165. arXiv: 1903.05165. URL: http://arxiv.org/abs/1903.05165.

Schleich, Daniel and Sven Behnke (2021). "Search-based planning of dynamic MAV trajectories using local multiresolution state lattices." In: *IEEE international conference on robotics and automation, ICRA 2021, xi'an, china, may 30 - june 5, 2021*. IEEE, pp. 7865–7871. URL: https://doi.org/10.1109/ICRA48506.2021.9560969.

Simon, Dan (1993). "The application of neural networks to optimal robot trajectory planning." In: *Robotics auton. syst.* 11.1, pp. 23–34. URL: https://doi.org/10.1016/0921-8890(93)90005-W.

Stachniss, Cyrill and Wolfram Burgard (2002). "An integrated approach to goal-directed obstacle avoidance under dynamic constraints for dynamic environments." In: *IEEE/RSJ international conference on intelligent robots and systems, lausanne, switzerland, september 30 - october 4, 2002*. IEEE, pp. 508–513. URL: https://doi.org/10.1109/IRDS.2002.1041441.

Strub, Marlin P. and Jonathan D. Gammell (2020). "Adaptively informed trees (ait*): fast asymptotically optimal path planning through adaptive heuristics." In: *2020 IEEE international conference on robotics and automation, ICRA 2020,*

paris, france, may 31 - august 31, 2020. IEEE, pp. 3191–3198. URL: https://doi.org/10.1109/ICRA40945.2020.9197338.

Zucker, Matthew, Nathan D. Ratliff, Anca D. Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M. Dellin, J. Andrew Bagnell, and Siddhartha S. Srinivasa (2013). "CHOMP: covariant hamiltonian optimization for motion planning." In: *Int. j. robotics res.* 32.9-10, pp. 1164–1193. URL: https://doi.org/10.1177/0278364913488805.