17th International Conference on Intelligent Autonomous Systems (IAS), Zagreb, Croatia, June 2022.

Two-step Planning of Dynamic UAV Trajectories using Iterative δ-Spaces

Sebastian Schräder, Daniel Schleich, and Sven Behnke

Institute for Computer Science VI, Autonomous Intelligent Systems,
University of Bonn, Germany,
schleich@ais.uni-bonn.de

Abstract. UAV trajectory planning is often done in a two-step approach, where a low-dimensional path is refined to a dynamic trajectory. The resulting trajectories are only locally optimal, however. On the other hand, direct planning in higher-dimensional state spaces generates globally optimal solutions but is time-consuming and thus infeasible for time-constrained applications. To address this issue, we propose δ-Spaces, a pruned high-dimensional state space representation for trajectory refinement. It does not only contain the area around a single lower-dimensional path but consists of the union of multiple near-optimal paths. Thus, it is less prone to local minima. Furthermore, we propose an anytime algorithm using δ-Spaces of increasing sizes.
We compare our method against state-of-the-art search-based trajectory planning methods and evaluate it in 2D and 3D environments to generate second-order and third-order UAV trajectories.

1 Introduction

In recent years, unmanned aerial vehicles (UAVs) have gained increasing interest for practical usage in many different fields like industrial inspection, agriculture, search & rescue, and delivery. Due to their ability for agile flight and high velocities, UAVs have huge potential for time-critical applications, e.g., in disaster-response scenarios. When flying close to obstacles, the systems' dynamics have to be considered during trajectory planning.

Liu et al. [1] directly incorporate UAV dynamics into a global planner. They use high-dimensional state lattices, from which they extract the optimal trajectory using A* and a heuristic based on the solution of a Linear Quadratic Minimum Time problem. Such approaches become infeasible for large environments, however, since searching high-dimensional state spaces is computationally expensive. Thus, a reduction of the state space size is necessary, either explicitly via pruning or implicitly using more informed search heuristics [2].

Many existing methods reduce the computational load by splitting the trajectory planning problem into two parts. A lower-dimensional geometric path can efficiently be found using search-based [3] or sampling-based [4] planners. After choosing an initial time allocation, this path can be refined to a high-dimensional

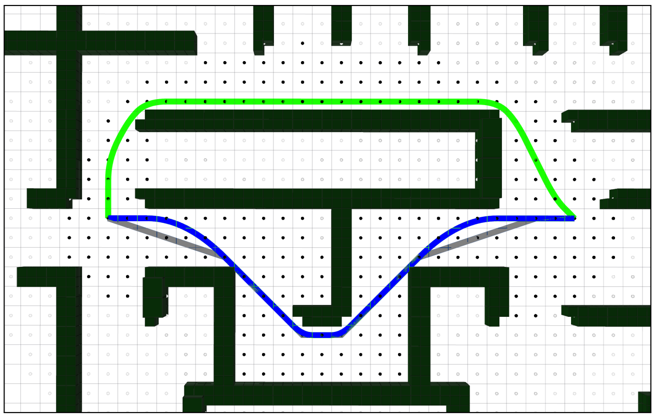


Fig. 1: Refining a 2D path (grey) to a dynamic trajectory. The blue trajectory is obtained by planning in a tunnel around the 2D path. The green trajectory is obtained by planning in the δ-Space (black dots). Although the green trajectory is more distant to the shortest 2D path, it can be tracked at higher velocities and has thus a shorter flight time.

dynamic trajectory [5]. To optimize the smoothness of high-dimensional trajectories, multiple methods haven been proposed, including gradient-based optimizers [6,7], B-splines [8], and neural networks [9]. These approaches only locally optimize the trajectory and are prone to local minima, however. The shortest geometric path towards the goal is not necessarily a good initialization for trajectory optimization, especially when it contains many turns or if we consider non-static initial states. A longer path might suit the UAV dynamics better, allowing higher velocities and thus resulting in shorter flight times.

In this work, we propose a method where the high-dimensional trajectory generation is not guided by a single shortest path but multiple near-optimal paths are considered. For this, we introduce the δ-Space, a set of paths whose length is within a certain sub-optimality bound δ. The motivation behind this definition is visualized in Fig. 1. By allowing larger deviations from the shortest low-dimensional path, we find trajectories that can be tracked at higher velocities and reduce the risk of local minima. Additionally, our method considers the structure of the environment and can efficiently prune dead ends, which is not possible if a tunnel with large radius around a single low-dimensional path is used.

In summary, the main contributions of this paper are:

- the introduction of δ-Spaces to prune high-dimensional state spaces while reducing the risk of local minima,
- an anytime algorithm allowing to iteratively increase the size of the δ-Space,
- and the application to second-order and third-order dynamic trajectory planning for UAVs, including an efficient search heuristic that reuses information from the δ-Space construction.

2 Related Work

Several approaches to reduce the size of high-dimensional state spaces have been proposed. A common method is to restrict the high-dimensional search to the vicinity of the solution of a lower-dimensional problem representation. For example, this can be done by only considering high-dimensional states, whose position components lie within a tunnel around a previously computed geometric path [10]. Such approaches are prone to local minima, however. The high-dimensional search might not even find a solution due to kinodynamic constraints if the tunnel size is too small.

Instead of pruning the state space, the solution of a lower-dimensional problem formulation can be used as a heuristic to guide the high-dimensional search. MacAllister et al. [11] solve a simplified spatial path-planning problem using breadth-first search to guide the planning, but they only consider 4D state lattices. Liu et al. [2] proposed to iteratively compute first-, second- and third-order UAV trajectories, where the lower-dimensional solution is used to guide the next-higher dimensional trajectory. This ensures that a solution can be found since large deviations from the low-dimensional trajectory are discouraged but possible. The resulting trajectories are no longer globally optimal, however. One approach to address the local minima problem of hierarchical methods was introduced by Ding et al. [12], who proposed a global B-spline-based kinodynamic search algorithm.

Another approach is to reduce the state space size globally using multiresolution. Du et al. [13] perform multiple A* searches in parallel, each at a different resolution of the state space. This approach has been extended to an anytime algorithm by Saxena et al. [14]. The idea of multiresolution has also be considered in combination with high-dimensional state lattices. González-Sieira et al. [15] adapt the resolution of a state lattice locally to the environment. This is done by grouping motion primitives that represent similar actions and considering only the longest collision-free primitive from each group. Schleich and Behnke [16] apply the idea of local multiresolution [17] to state lattices for faster replanning of dynamic UAV trajectories.

Besides the resolution, the state dimensionality can also be adapted to accelerate the planning. Gochev et al. [18] represent the state space locally at different dimensionalities. This allows them to only use high-dimensional state representations when necessary, i.e., in narrow areas of the environment, and to plan in a lower-dimensional space otherwise. Gochev et al. [19] extend this method with an incremental version of weighted A* to achieve a better performance. Vemula et al. [20] apply adaptive dimensionality to the problem setting of path planning with dynamic obstacles.

In this work, we propose a state space pruning technique similar to the tunnel method [10] but use multiple lower-dimensional paths to determine which high-dimensional states are pruned. This makes our method less prone to local minima. Additionally, we can iteratively increase the size of our state space efficiently in an anytime fashion to further improve the solution quality.

3 Method

Applying search-based planning methods to high-dimensional state spaces is computationally expensive. Thus, many methods reduce the complexity of such search problems by projecting the high-dimensional robot state onto a lower-dimensional space, where a solution can be found with less effort. This low-dimensional solution can then be used to accelerate high-dimensional planning, either by guiding the search or by pruning the state space. In the context of UAV trajectory planning, the first step usually consists of finding an optimal spatial path without dynamic consideration. Here, optimality is usually defined as minimizing the path length. However, if the shortest path contains many turns, it can only be tracked by the UAV at low velocities. A longer path might suit the UAV dynamics much better and can lead to shorter flight times. Thus, the high-dimensional trajectory generation should not be guided by a single shortest path. Instead, we propose to use the δ-Space, i.e., the union of all paths whose length is within a sub-optimality bound δ. The high-dimensional trajectory search can then be restricted to states whose position components are part of this space. In Sec. 3.1, we give a formal definition of the δ-Space and how it is used to solve high-dimensional search-problems.

The choice of δ significantly influences the performance of the proposed method. A small value results in fast planning times but yields only locally optimal solutions. When choosing high values, the search is less prone to local minima, but planning times are increased. In Sec. 3.2, we propose an efficient method how to iteratively increase δ while reusing the previous search results. Thus, an initial result can be found quickly using a small value of δ and the obtained solution can be subsequently improved in an anytime fashion.

Finally, we point out that δ-Spaces cannot only be used to prune high-dimensional state spaces. When generating the δ-Space, we already explore the structure of the environment. The obtained information can be integrated into a search heuristic to guide the higher-dimensional planning. Since this depends on the specific problem setting, we present an example for such a heuristic in Sec. 3.3, which describes the application of δ-Spaces to dynamic UAV trajectory planning.

3.1 δ-Space Definition

We consider a planning problem with a set of high-dimensional robot states $\mathcal{S}_h$, valid state transitions $\mathcal{E}_h \subset \mathcal{S}_h \times \mathcal{S}_h$, a cost function $c_h : \mathcal{E}_h \to \mathbb{R}_{\geq 0}$, an initial state s, and a goal state g. A trajectory $\mathcal{T}_{h,s\mapsto g}$ from s to g is a sequence $(s_0, \dots, s_n)$ with $(s_i, s_{i+1}) \in \mathcal{E}_h$, $s_0 = s$, and $s_n = g$. The cost of a trajectory is defined as the sum of the individual state transition costs, i.e.,

$$\mathcal{C}_h(\mathcal{T}_{h,s\mapsto g}) := \sum_{i=0}^{n-1} c_h(s_i, s_{i+1}). \tag{1}$$

We are interested in finding a trajectory $\mathcal{T}^{opt}_{h,s\mapsto g}$ minimizing these costs. In the following, we use the shorthand notation $\mathcal{C}_h(s,g) := \mathcal{C}_h(\mathcal{T}^{opt}_{h,s\mapsto g})$ for the costs of an optimal trajectory from s to g.

To support the planning, we additionally define an abstract, lower-dimensional problem representation, consisting of states $\mathcal{S}_l$, transitions $\mathcal{E}_l \subset \mathcal{S}_l \times \mathcal{S}_l$ and costs $\mathcal{C}_l : \mathcal{E}_l \to \mathbb{R}$. A high-dimensional robot state can be transformed into an abstract one using the projection $\pi : \mathcal{S}_h \to \mathcal{S}_l$.

For a given instance of the planning task, we define the δ-Space $\mathcal{S}_l^\delta$ as

$$\mathcal{S}_l^\delta := \{s_l \in \mathcal{S}_l \mid \mathcal{C}_l(\pi(s), s_l) + \mathcal{C}_l(s_l, \pi(g)) \leq \mathcal{C}_l(\pi(s), \pi(g)) + \delta\}\,. \tag{2}$$

Note that this definition depends on the specific start and goal states of the current planning instance.

For solving the higher-dimensional planning problem, we restrict the state space to states for which the corresponding abstract state is part of the δ-Space. Thus, we consider the pruned state space

$$\mathcal{S}_h^\delta := \left\{s_h \in \mathcal{S}_h \mid \pi(s_h) \in \mathcal{S}_l^\delta\right\}. \tag{3}$$

Additionally, we define the closure $\overline{\mathcal{S}}_h^\delta$ of this set as the set of all states that can be reached from $\mathcal{S}_h^\delta$, i.e.,

$$\overline{\mathcal{S}}_h^\delta := \left\{s_1 \in \mathcal{S}_h \mid \exists s_0 \in \mathcal{S}_h^\delta : (s_0, s_1) \in \mathcal{E}_h\right\}. \tag{4}$$

Correspondingly, we define the set of possible state transitions

$$\mathcal{E}_h^\delta := \left\{e_h := (s_0, s_1) \in \mathcal{E}_h \mid s_0, s_1 \in \mathcal{S}_h^\delta\right\} \tag{5}$$

and

$$\overline{\mathcal{E}}_h^\delta := \left\{e_h := (s_0, s_1) \in \mathcal{E}_h \mid s_0, s_1 \in \overline{\mathcal{S}}_h^\delta\right\}. \tag{6}$$

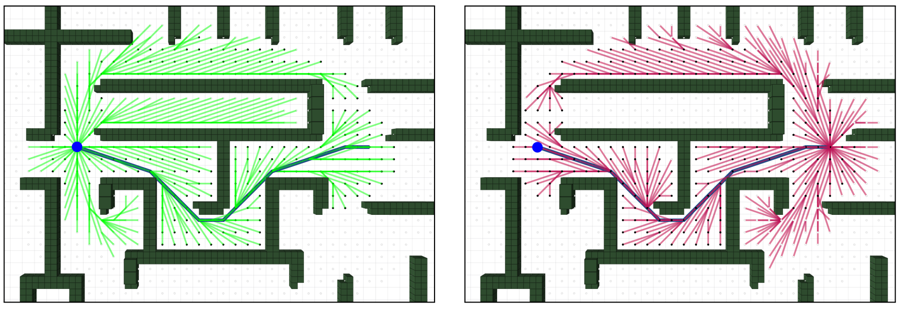
To ensure that only states from $\mathcal{S}_h^\delta$ are considered during planning, we adjust the cost function to be

$$c_h^\delta(e_h) = \begin{cases} c_h(e_h), & \text{if } e_h \in \mathcal{E}_h^\delta \\ \infty, & \text{otherwise.} \end{cases} \tag{7}$$

States from the boundary of the state space, i.e., from $\overline{\mathcal{S}}_h^\delta \setminus \mathcal{S}_h^\delta$, are only needed for the anytime planning algorithm in Sec. 3.2 to determine possible states that will be added to the search space when increasing δ.

Although the definition of the δ-Space can be applied to continuous state spaces as well, we restrict ourselves in this work to discretized states. Thus, we can cast planning as graph search. After the δ-Space is constructed, the high-dimensional trajectory can be generated by using graph search algorithms like A* on the state lattice graph $\mathcal{G}_h^\delta(\overline{\mathcal{S}}_h^\delta, \overline{\mathcal{E}}_h^\delta)$ consisting of nodes $\overline{\mathcal{S}}_h^\delta$ and edges $\overline{\mathcal{E}}_h^\delta$.

To construct the δ-Space, we solve the lower-dimensional planning problem using A* search. According to (2), for each state $s_l \in \mathcal{S}_l$, we need to know both



(a) Forward search (b) Backward search

Fig. 2: Lower-dimensional A* searches to calculate the δ-Space. The start state is marked with the blue circle. The nodes forming the δ-Space (black dots) are expanded by both forward and backward search. The blue line is thereby the shortest path.

the cost from the start state to s_l as well as the cost from s_l towards the goal. Thus, we use two searches, one forward search from the start and one backward search starting at the goal, as shown in Fig. 2. These searches are independent of each other and can be parallelized for a better performance. If a state s_l has been expanded by both searches, we can easily decide whether it belongs to the δ-Space by checking the criterion (2). To find all states of the δ-Space, we slightly adjust the A* algorithm: The f-value of a state s' is defined as the sum of the cost from start to s' and an heuristic value $h(s')$ estimating the remaining cost-to-go. If the heuristic function h used for the A* search is admissible, $f(s')$ represents a lower threshold on the costs of an optimal trajectory containing s'. Thus, after finding the optimal trajectory $\mathcal{T}^{opt}_{l,\pi(s)\mapsto\pi(g)}$, we extract its costs and continue to expand states until the next state to be expanded has an f-value exceeding $\mathcal{C}_l(\mathcal{T}^{opt}_{l,s\mapsto g}) + \delta$. For an admissible heuristic, we thus can ensure that all states of the δ-Space are expanded by the forward and backward searches.

3.2 Iterative δ-Spaces

In general, a small δ results in low planning times. However, larger values of δ are necessary to avoid local minima. In this section, we describe an anytime planning algorithm for the case where a planning time limit is set, e.g., when replanning at a fixed frequency. The anytime planner starts with a small δ to quickly generate a first solution and afterwards iteratively increases δ to further improve the solution until the planning time limit is reached.

As described above, the states within the δ-Space are found by executing forward and backward A* searches until f-values are reached that exceed $\mathcal{C}_l(\mathcal{T}^{opt}_{l,s\mapsto g}) + \delta$. For higher values of δ, the same searches are executed, but up to higher f-values. Since the f-values increase monotonically during A* searches, we can pause and later resume the searches for planning in δ-Spaces of increasing sizes.

Algorithm 1: Anytime Planning In Iterative δ-Spaces

```
   Input: Start state $s$, goal state $g$, $\delta_0$, step size $\Delta\delta$
   Output: Trajectory $\mathcal{T}^{opt}_{\delta,s\mapsto g}$
1  $\mathcal{S}_l^{\delta_0} \leftarrow$ GenerateDeltaSpace($\pi(s), \pi(g), \delta_0$)
   /* get optimal Trajectory and final search state            */
2  $\mathcal{T}^{opt}_{\delta,s\mapsto g}$, $(\mathcal{O}_0, \mathcal{O}'_0, \mathcal{F}_0, \mathcal{G}_0) \leftarrow$ HighDimSearch($s, g, \mathcal{G}_h^{\delta_0}$)
3  $i \leftarrow 0$
4  while There is time to improve the trajectory do
5      $\delta_{i+1} \leftarrow \delta_i + \Delta\delta$
6      $\mathcal{S}_l^{\delta_{i+1}} \leftarrow$ UpdateDeltaSpace($\mathcal{S}_l^{\delta_i}, \delta_{i+1}$)
7      Update $\mathcal{F}_i, \mathcal{G}_i$ for all nodes in $\mathcal{O}'_i \cap \mathcal{S}_h^{\delta_{i+1}}$
8      $\mathcal{O}_i \leftarrow \mathcal{O}_i \cup (\mathcal{O}'_i \cap \mathcal{S}_h^{\delta_{i+1}})$
9      $\mathcal{T}^{opt}_{\delta,s\mapsto g}$, $(\mathcal{O}_{i+1}, \mathcal{O}'_{i+1}, \mathcal{F}_{i+1}, \mathcal{G}_{i+1}) \leftarrow$
         HighDimSearch($s, g, \mathcal{G}_h^{\delta_{i+1}}, (\mathcal{O}_i, \mathcal{F}_i, \mathcal{G}_i)$)
10     $i \leftarrow i + 1$
11 return $\mathcal{T}^{opt}_{\delta,s\mapsto g}$
```

The complete anytime planning algorithm is given in Algorithm 1. First, we calculate the δ-Space for the initial size δ_0 (Line 1) and plan a high-dimensional trajectory by searching the pruned state-lattice graph $\mathcal{G}_h^{\delta_0}$ using A* (Line 2). Here, we save the final state of the A* search, including the open list $\mathcal{O}_0$, as well as the set of f-values $\mathcal{F}_0$ and g-values $\mathcal{G}_0$. Additionally, we keep track of all states where the search hits the boundary of the δ-Space, i.e., all states within $\overline{\mathcal{S}}_h^{\delta_0} \setminus \mathcal{S}_h^{\delta_0}$ that are neighbours of expanded states. Since these states can only be reached via edges with infinite costs, they are not considered as valid states in the current search. However, they might become valid once δ is increased. Thus, instead of adding them to the open list, we add them to a separate list $\mathcal{O}'_0$ for later use.

If the planning time limit is not met yet, we can iteratively plan in larger δ-Spaces to further improve the trajectory (Line 4). First, we increase the size of the δ-Space by resuming the low-dimensional searches with increased δ (Line 5 and 6) as described above. Afterwards, we continue the high-dimensional search on the extended state space. Here, we first update the cost function for all states from $\mathcal{O}'_i$ that have been added to the larger δ-Space $\mathcal{S}_h^{\delta_{i+1}}$ (Line 7) and add them to the open list of the previous search (Line 8). Then, we execute the high-dimensional search on the extended search space (Line 9), starting with the already initialized open list, f-values and g-values from the previous search. Note, that we do not include the closed list. If the optimal trajectory contains states that were not part of the previous state space $\mathcal{S}_h^{\delta_i}$, the f-values of previously expanded states might be suboptimal. Thus, we need to allow one re-expansion per state in each iteration.

3.3 Application to UAV Trajectory Planning

We apply δ-Spaces to plan second-order and third-order UAV trajectories. Thus, we model the UAV state as 6-tuples $s = (\mathbf{p}, \mathbf{v}) \in \mathbb{R}^6$ or 9-tuples $s = (\mathbf{p}, \mathbf{v}, \mathbf{a}) \in \mathbb{R}^9$, consisting of a discretized 3D position $\mathbf{p}$, velocity $\mathbf{v}$, and acceleration $\mathbf{a}$. To compare against the State of the Art, we combine δ-Spaces with the search-based trajectory generation methods from [2] and [16]. In both works, the set of state transitions $\mathcal{E}_h$ consists of motion primitives $e_{\mathbf{u},\tau}$ which are generated by applying constant acceleration or jerk commands $\mathbf{u}$ over a short time interval τ. The corresponding costs are defined as the weighted sum of control effort and primitive duration:

$$c_h(e_{\mathbf{u},\tau}) = ||\mathbf{u}||_2^2 \tau + \rho\tau. \tag{8}$$

For the low-dimensional problem representation, we project a high-dimensional state $s = (\mathbf{p}, \mathbf{v}, \mathbf{a})$ onto its position components $\mathbf{p}$ and use a simple 3D grid with a 26-connected neighbourhood as low-dimensional planning space.

When using search-based methods like A*, the choice of a good heuristic function is crucial for the planning performance. In this work, we propose a heuristic that efficiently uses information obtained by solving the lower-dimensional planning problem.

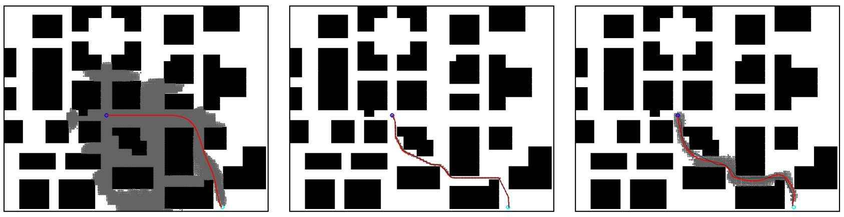
From the construction of the δ-Space, we directly obtain the distance from each state towards the goal along the shortest lower-dimensional path. We assume that the distance towards the goal is travelled on a straight line and the UAV reaches a zero velocity at the goal. Thus, we can assume that the UAV accelerates to the maximal reachable velocity, keeps this velocity for a certain time and then decelerates to stop at the goal. In the following, we describe how this can be incorporated into a search heuristic for second-order trajectories. The extension to third-order trajectories is left for future work.

For a pair of discretized 1D velocities v_1, v_2, let the minimum time needed to accelerate the UAV from v_1 to v_2 be t_{v_1,v_2}. The corresponding control effort is denoted by c_{v_1,v_2} and the distance the UAV moves during the acceleration by d_{v_1,v_2}. These values can efficiently be precomputed. During search, we calculate the maximal velocity that can be reached from an initial velocity v when travelling a distance of d as $v_{max} := \max\{v' | d_{v,v'} + d_{v',0} \leq d\}$. Then, we can estimate the flight time

$$T = \frac{d - d_{v,v_{max}} - d_{v_{max},0}}{v_{max}} + t_{v,v_{max}} + t_{v_{max},0} \tag{9}$$

and the corresponding control cost

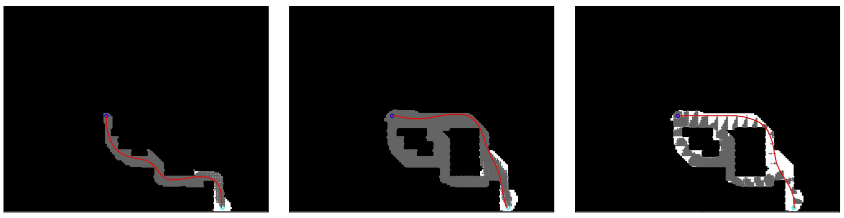
$$c = c_{v,v_{max}} + c_{v_{max},0}. \tag{10}$$



(a) Full State Space [2]

(b) Iterative (Pos. $\rightarrow$ Jerk, $\epsilon = 1.0$) [2]

(c) Iterative (Pos. $\rightarrow$ Jerk, $\epsilon = 0.6$) [2]



(d) Tunnel

(e) δ-Space

(f) δ-Space Heuristic

Fig. 3: Examples for generating third-order 2D trajectories with the method of Liu et al. [2]. The start is marked by the blue circle. Obstacles and pruned states are black. Expanded states are grey. (a) Searching the full state space finds the optimal trajectory but is computationally expensive. (b) The iterative heuristic with a high greediness parameter ϵ has low planning times but results in a slow trajectory which tightly follows the guiding path. (c) A lower greediness generates a faster trajectory at the cost of higher planning times, but the obtained solution is still close to the 2D path and thus only locally optimal. (d) The tunnel also results in a suboptimal trajectory. (e) δ-Spaces find the optimal trajectory while still having low planning times. (f) The additional δ-Space heuristic further improves the planning time.

For the initial velocity v, we choose the maximum of the absolute velocities along the independent axes. Note that this heuristic is not admissible since it overestimates the cost for diagonal movements. As our experiments show, this slightly increases the trajectory costs compared to admissible heuristics but it results in lower costs than the inadmissible heuristic used by Liu et al. [2].

4 Evaluation

In a first experiment, we use δ-Spaces on top of the method of Liu et al. [2] to plan third-order trajectories in a cluttered 2D environment of size 80×62 m. We use the source code[1] from Liu et al. [2] with the following parameters:

ρ	τ	$v_{\max}$	$a_{\max}$	$u_{\max}$	du	Timeout
10	1.0 s	3 m/s	1 m/s^2	1 m/s^3	0.5 m/s^3	1 s

.

[1]`https://github.com/sikang/motion_primitive_library`

Table 1: Planning statistics for using the δ-Space with the method from Liu et al. [2] to generate third-order trajectories in a 2D environment. Planning time, number of expansions and trajectory costs are averaged over the tasks where all methods found a solution. Success denotes the fraction of tasks for which a solution was found within the allowed planning time.

	Success	Time	Expansions	Costs
Full State Space [2]	74.0%	214 ms	12 291	121.92
Iterative (Pos. $\to$ Jerk, $\epsilon = 0.6$) [2]	89.7%	334 ms	6 125	180.33
Iterative (Pos. $\to$ Jerk, $\epsilon = 0.7$) [2]	96.5%	233 ms	4 231	204.49
Iterative (Pos. $\to$ Jerk, $\epsilon = 0.8$) [2]	96.3%	97 ms	1 608	223.44
Iterative (Pos. $\to$ Jerk, $\epsilon = 0.9$) [2]	95.7%	76 ms	1 128	247.48
Iterative (Pos. $\to$ Jerk, $\epsilon = 1.0$) [2]	95.2%	**28 ms**	**260**	267.87
Tunnel ($r = 1.0$ m)	94.6%	50 ms	3 153	127.25
Tunnel ($r = 1.5$ m)	95.4%	58 ms	3 586	124.12
Tunnel ($r = 2.5$ m)	95.4%	67 ms	4 070	123.27
Tunnel ($r = 3.5$ m)	94.6%	72 ms	4 328	123.12
δ-Space ($\delta = 1.0$ m)	95.9%	115 ms	6 039	122.53
δ-Space ($\delta = 1.5$ m)	91.7%	139 ms	7 175	122.46
δ-Space ($\delta = 2.5$ m)	84.3%	182 ms	9 014	122.37
δ-Space Heuristic ($\delta = 1.0$ m)	**97.6%**	53 ms	1 980	117.79
δ-Space Heuristic ($\delta = 1.5$ m)	96.1%	62 ms	2 231	117.71
δ-Space Heuristic ($\delta = 2.5$ m)	**97.6%**	75 ms	2 484	**117.53**

All other parameters were not changed and are the default ones from the implementation [2]. To reduce planning times, Liu et al. propose to generate a lower-order trajectory first, which can then be used as a heuristic to guide the higher-dimensional search. In this experiment, we choose a 2D path as guiding trajectory for a fair comparison against the δ-Space. The resulting trajectories for one example task are shown in Fig. 3 and the average planning performance over a set of 1000 tasks is summarized in Tab. 1. The iterative heuristic of Liu et al. uses a greediness parameter ϵ to control the deviation of the high-dimensional trajectory from the guiding 2D path. With a high parameter, the trajectory tightly follows the shortest path (Fig. 3b). This results in planning times that are significantly faster than for all other methods. However, these trajectories can only be traversed at low velocities and thus, their average costs are more than twice the global optimum. The costs can be reduced by lowering the greediness parameter, which results in higher planning times. Note that with a low greediniess of $\epsilon = 0.6$, the method of Liu et al. has significantly higher planning times than the δ-Space method, while still generating trajectories with higher costs. Fig. 3c shows an example trajectory, which is only locally optimal since it still tightly follows the shortest 2D path. Similar trajectories but higher velocities are obtained when restricting the search to a tunnel around the 2D path

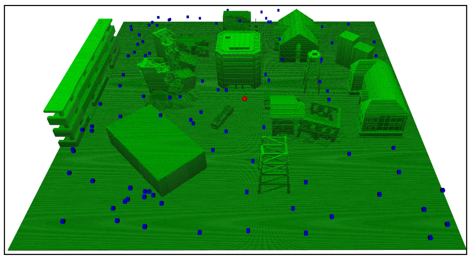


Fig. 4: Simulated 3D outdoor environment. The UAV starts at the map center (red circle) and plans trajectories to all goal positions (blue squares).

(Fig. 3d). Our method (Fig. 3e) allows larger deviations from the shortest 2D path and generates a trajectory close to the optimal one. This result is obtained by using the same (inadmissible) heuristic that Liu et al. use when planning in the full state space (Fig. 3a). From the construction of the δ-Space we already know the 2D distances from all states to the goal. We can use these as a heuristic if we assume that the UAV flies at maximum velocity. This further reduces the planning times of our method and results in the lowest trajectory costs (Fig. 3f).

In a second experiment, we evaluate our method in a simulated outdoor environment (Fig. 4). Here, we use the method from [16] with the same parameters as in the original paper:

ρ	τ	$v_{\max}$	$u_{\max}$	du	Timeout
16	0.5 s	4 m/s	2 m/s^2	2 m/s^2	1 000 000 expansions

.

Note that we abort searches, if no solution within the first 1 000 000 expansions is found. Although we compare against multiresolutional state lattices, the δ-Space is applied on top of a uniform state lattice, since the δ-Space already reduces the size of the state space. The results are reported in Tab. 2. When using standard A* search, the δ-Space outperforms uniform planning in the full state space as well as the multiresolutional planning method from [16] with respect to planning time and success rates, while achieving trajectory costs that are close to the optimal ones. Using the δ-Space heuristic reduces the planning time even further. However, since the heuristic is not admissible, this comes at the cost of significantly increased path costs.

Note that, compared to uniform planning, multiresolution increases the success rates but has a slightly higher average planning time. This has already been reported in [16]. For some planning tasks, the suboptimal trajectory costs introduced by multiresolution require more expansions during A* search. To address this issue, a level-based expansion scheme was introduced in [16]. This reduces the planning times at the cost of more suboptimal trajectories. Here, multiresolution is significantly faster than δ-Space planning. This is due to the overhead of the low-dimensional search needed to generate the δ-Space, which took on average 0.266 s. The high-dimensional search itself is actually faster than multiresolutional planning. Overall, in open-spaced environments, multiresolution

Table 2: Planning statistics for using the δ-Space with the method from [16] to generate second-order trajectories in a 3D environment. We report results for different expansion schemes during search: standard A* search and a level-based expansion scheme (Level-A*) proposed in [16], which prioritize expansions of states on higher resolution levels. Planning time, number of expansions and trajectory costs are averaged over the tasks where all methods found a solution. Success denotes the fraction of tasks for which a solution was found within the allowed planning time.

	Success	Time	Expansions	Costs
Uniform (A*)	91.8%	0.923 s	79 883	**251.95**
Multiresolution (A*) [16]	95.9%	1.096 s	81 700	259.55
Tunnel (A*)	**100.0%**	0.603 s	34 994	259.18
δ-Space (A*)	98.97%	0.834 s	55 138	253.05
δ-Space with Heuristic (A*)	**100.0%**	**0.324 s**	**4695**	264.64
Uniform (Level-A*)	97.9%	0.108 s	9099	262.82
Multiresolution (Level-A*) [16]	**100.0%**	**0.076 s**	4491	277.34
Tunnel (Level-A*)	**100.0%**	0.287 s	6450	268.91
δ-Space (Level-A*)	**100.0%**	0.324 s	6056	**262.14**
δ-Space with Heuristic (Level-A*)	**100.0%**	0.276 s	**479**	288.45

can be faster than δ-Space planning but results in higher trajectory costs. However, δ-Spaces can also be used in more complex environments (like the one in Fig. 3), where multiresolution is not applicable.

Finally, we evaluate the anytime version of our planning algorithm. It iteratively increases the size of the δ-Space but can also be applied to plan in tunnels of increasing radius around the low-dimensional path. In this experiment, we investigate which state space representation is more suitable for the anytime planning algorithm. Furthermore, we analyse the influence of different heuristic weights that might speed up planning at the cost of sub-optimal solutions. Since the anytime algorithm requires significant adjustments of the source code of the high-dimensional planning algorithms, we implemented our own planning method for this experiment. Our planner generates third-order trajectories on a 3D map. For this, it uses motion primitives that were precomputed by the time-optimal trajectory generation method from [21]. Figure 5 shows the results for both the tunnel and the δ-Space with different heuristic weights. The colors of the graphs represents the used weight and a line represents an iterative search as described in Sec. 3.2. Here, it can be clearly seen that tunnel and δ-Space behave differently. The heuristic weight w does impact the tunnel only slightly. Increasing the tunnel radius does not improve the trajectory costs much and the costs start to stagnate after few iterations. On the other hand, the heuristic weight strongly influences the performance of the δ-Spaces. A larger weight significantly decreases the planning times. Although the trajectory costs are in-

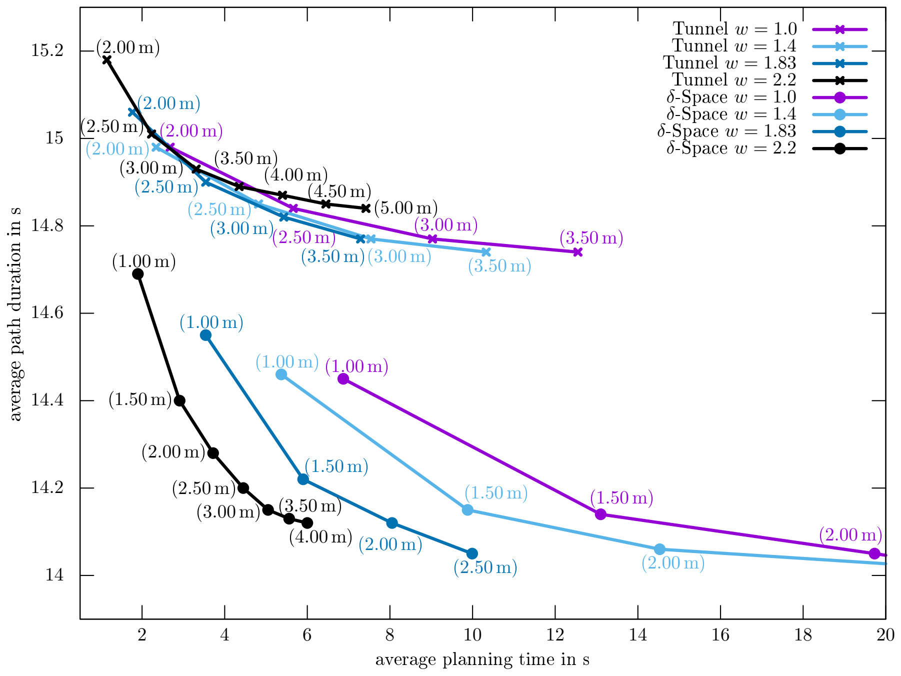


Fig. 5: The impact of weighted heuristics and anytime planning. A line represents an iterative search as described in Sec. 3.2.

creased, this can be made up for by iteratively increasing the size of the δ-Space. Thus, using the δ-Space with a large heuristic weight generates trajectories with similar planning times as the tunnel method but significantly lower costs.

Tab. 3 summarizes the planning performance for each iteration of the anytime algorithms using a heuristic weight $w = 1.83$. As expected, the accumulated planning time for iterative planning is higher than the time for directly planning in the corresponding state space. However, the relative increase is smaller for the δ-Space, compared to the Tunnel. This shows that δ-Spaces are more suitable for anytime planning. Interestingly, the trajectory costs are slightly lower for anytime planning. Since each iteration allows additional re-expansions, iteratively increasing δ can help to correct trajectories that are suboptimal due to the inadmissible heuristic.

5 Conclusion

In this paper, we propose a new method for pruning of high-dimensional state spaces. Instead of restricting the search to the vicinity of an optimal lower-dimensional path, we introduce the δ-Space, which represents the union of multiple paths within a suboptimality bound δ. Our experiments show that this

Table 3: Planning statistics for the anytime algorithm to generate third-order trajectories in a 3D environment.

		iterative planning time	accumulated iterative planning time	direct planning time	trajectory cost (iterative)	trajectory cost (direct)
δ-Space	$\delta = 1.0\,\mathrm{m}$	$+3.5s$	$3.5s$	$3.5s$	$14.55s$	$14.55s$
	$\delta = 1.5\,\mathrm{m}$	$+2.4s$	$5.9s$	$4.7s$	$14.22s$	$14.25s$
	$\delta = 2.0\,\mathrm{m}$	$+2.2s$	$8.1s$	$6.3s$	$14.12s$	$14.17s$
	$\delta = 2.5\,\mathrm{m}$	$+1.9s$	$10.0s$	$7.7s$	$14.05s$	$14.09s$
Tunnel	$r = 1.0\,\mathrm{m}$	$+0.6s$	$0.6s$	$0.6s$	$16.01s$	$16.01s$
	$r = 1.5\,\mathrm{m}$	$+0.9s$	$1.5s$	$1.2s$	$15.27s$	$15.25s$
	$r = 2.0\,\mathrm{m}$	$+1.3s$	$2.8s$	$1.8s$	$15.03s$	$15.06s$
	$r = 2.5\,\mathrm{m}$	$+1.8s$	$4.6s$	$2.8s$	$14.90s$	$14.94s$

significantly reduces planning times. In contrast to other state space reduction methods, δ-Spaces are less prone to local minima and can find trajectories close to the globally optimal solution. Furthermore, we introduced an anytime planning algorithm that efficiently plans in δ-Spaces of increasing sizes. Thus, a first initial solution can be found quickly, while subsequent iterations further improve the solution quality.

Acknowledgements. This work has been funded by the German Federal Ministry of Education and Research (BMBF) in the project "Kompetenzzentrum: Aufbau des Deutschen Rettungsrobotik-Zentrums (A-DRZ)".

References

1. Liu, S., Atanasov, N., Mohta, K., Kumar, V.: Search-based motion planning for quadrotors using linear quadratic minimum time control. In: Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS). (2017)
2. Liu, S., Mohta, K., Atanasov, N., Kumar, V.: Search-based motion planning for aggressive flight in SE(3). In: Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA). (2018)
3. E. Hart, P., J. Nilsson, N., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. Intelligence/sigart Bulletin - SIGART **37** (12 1972) 28–29
4. LaValle, S.M.: Rapidly-exploring random trees: A new tool for path planning. Computer Science Dept., Iowa State University (1998)
5. Nieuwenhuisen, M., Behnke, S.: Local multiresolution trajectory optimization for micro aerial vehicles employing continuous curvature transitions. In: Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS). (2016)
6. Kalakrishnan, M., Chitta, S., Theodorou, E., Pastor, P., Schaal, S.: STOMP: Stochastic trajectory optimization for motion planning. In: Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA). (2011)

7. Zucker, M., Ratliff, N., Dragan, A.D., Pivtoraiko, M., Klingensmith, M., Dellin, C.M., Bagnell, J.A., Srinivasa, S.S.: CHOMP: Covariant hamiltonian optimization for motion planning. The International Journal of Robotics Research **32**(9-10) (2013) 1164–1193
8. Arney, T.: Dynamic path planning and execution using B-splines. In: Third International Conference on Information and Automation for Sustainability. (2007)
9. Simon, D.: The application of neural networks to optimal robot trajectory planning. Robotics and Autonomous Systems **11**(1) (1993) 23–34
10. Stachniss, C., Burgard, W.: An integrated approach to goal-directed obstacle avoidance under dynamic constraints for dynamic environments. In: Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS). (2002)
11. MacAllister, B., Butzke, J., Kushleyev, A., Pandey, H., Likhachev, M.: Path planning for non-circular micro aerial vehicles in constrained environments. In: Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA). (2013)
12. Ding, W., Gao, W., Wang, K., Shen, S.: An efficient B-spline-based kinodynamic replanning framework for quadrotors. IEEE Trans. on Robotics **35**(6) (2019) 1287–1306
13. Du, W., Islam, F., Likhachev, M.: Multi-resolution A*. In: Thirteenth Annual Symposium on Combinatorial Search. (2020)
14. Saxena, D.M., Kusnur, T., Likhachev, M.: AMRA*: Anytime multi-resolution multi-heuristic A*. arXiv preprint arXiv:2110.05328 (2021)
15. González-Sieira, A., Mucientes, M., Bugarín, A.: Graduated fidelity lattices for motion planning under uncertainty. In: Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA). (2019)
16. Schleich, D., Behnke, S.: Search-based planning of dynamic MAV trajectories using local multiresolution state lattices. In: Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA). (2021)
17. Behnke, S.: Local multiresolution path planning. In: RoboCup 2003: Robot Soccer World Cup VII, Springer (2003) 332–343
18. Gochev, K., Cohen, B., Butzke, J., Safonova, A., Likhachev, M.: Path planning with adaptive dimensionality. In: Fourth Annual Symposium on Combinatorial Search. (2011)
19. Gochev, K., Safonova, A., Likhachev, M.: Incremental planning with adaptive dimensionality. In: Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS). (2013)
20. Vemula, A., Muelling, K., Oh, J.: Path planning in dynamic environments with adaptive dimensionality. In: Ninth Annual Symposium on Combinatorial Search. (2016)
21. Beul, M., Behnke, S.: Fast full state trajectory generation for multirotors. In: Proc. of Int. Conf. on Unmanned Aircraft Systems (ICUAS), IEEE (2017)