International Conference on Intelligent Robots and Systems (IROS), Macau, China, November 2019.

Fast Time-optimal Avoidance of Moving Obstacles for High-Speed MAV Flight

Marius Beul and Sven Behnke

***Abstract*— In this work, we propose a method to efficiently compute smooth, time-optimal trajectories for micro aerial vehicles (MAVs) evading a moving obstacle. Our approach first computes an n-dimensional trajectory from the start- to an arbitrary target state including position, velocity and acceleration. It respects input- and state-constraints and is thus dynamically feasible. The trajectory is then efficiently checked for collisions, exploiting the piecewise polynomial formulation. If collisions occur, viastates are inserted into the trajectory to circumvent the obstacle and still maintain time-optimality. These viastates are described by position, velocity, and acceleration. The evaluation shows that the computational demands of the proposed method are minimal such that obstacle avoidance can begin within few milliseconds. Optimality of generated trajectories, combined with the ability for frequent online re-planning from non-hover initial conditions, make the approach well suited for evasion of suddenly perceived obstacles during fast flight.**

I. Introduction

In recent years, multiple novel applications for flying robots emerged, enabled by two main factors: i) manufacturers developed affordable and capable micro aerial vehicles (MAVs) for hobby, recreation, and professional usage that do not require extensive flight training; ii) recent advances in robotic research led to efficient methods for environment perception and safe navigation, making various applications that can only be performed autonomously possible. This includes operations at high velocities and with multiple MAVs. One driver for developing such systems was also the DARPA-formulated goal of flying fast and autonomously in their Fast Lightweight Autonomy (FLA) program [1].

In the field, applications often require fast flight in nearly obstacle-free environments. For example, in a typical outdoor inspection task, only the to-be-inspected object (windmill, power line, ...) obstructs the otherwise free space. Also, when executing an exploration mission with multiple MAVs, often the only obstacles present are the other MAVs in mostly free space.

A possible use case for our method emerged from the Mohamed Bin Zayed International Robotics Challenges (MBZIRC) 2017 [2] and 2020 [3]. At MBZIRC 2017, multiple MAVs shared the same workspace while picking and delivering small discs. For MBZIRC 2020, an MAV has to avoid static balloons while interacting with a flying object. Here, fast mission accomplishment is key for a high score. Since the environment is mostly obstacle free, one can omit global planning methods like A* [4]. Instead, direct flight to the target, only avoiding small static no-fly zones around the balloons and dynamic no-fly zones around the other MAVs is a feasible approach.

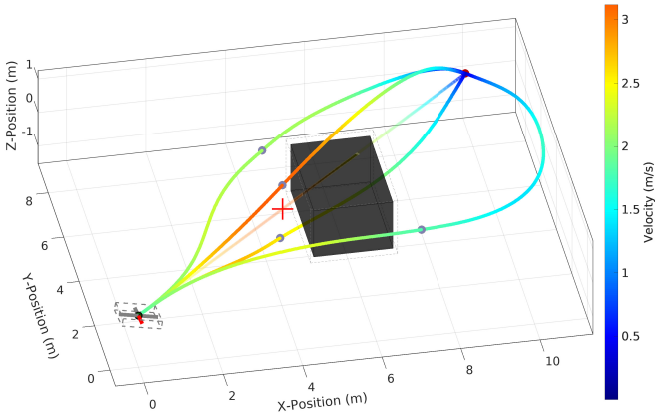


Fig. 1: The MAV on the bottom left flies towards the top right target (red dot) with a velocity of $1.8\,\mathrm{m/s}$ and an acceleration of $0.5\,\mathrm{m/s^2}$ when suddenly perceiving an obstacle. Our method instantaneously detects the collision (red cross) of the original semitransparent trajectory. Within $6\,\mathrm{ms}$, it computes a set of four time-optimal avoidance trajectories from the current state to the target state that steer through the gray viastates. These trajectories evade the obstacle in varying dimensions (over the top, underneath, left and right). The fastest collision-free trajectory (over the top of the obstacle) is chosen and executed by the MAV. The detour only takes $7.33\,\mathrm{s}$ vs. $7.31\,\mathrm{s}$ of the original trajectory.

Fig. 1 illustrates the proposed method. We first compute an time-optimal trajectory from the start state to an arbitrary target state that is efficiently checked for collisions. If collisions occur, via states described by position, velocity, and acceleration are inserted to circumvent the obstacle and maintain time-optimality. In our obstacle avoidance algorithm, we employ and extend methods based on our own previous work [5]. Our main contributions are:

- generation of trajectories targeting only partially defined target states (Sec. III),
- fast computation of optimal trajectories that avoid a static or moving obstacle (Sec. IV),
- evaluation of our method in simulation including profiling of computational requirements (Sec. V).

In this work, we employ our method to 3-dimensional problems for ease of explanation. The method itself however does not make any assumption about the dimensionality of the planning problem.

The code used in this work is open source[1].

This work has been supported by a grant of the Mohamed Bin Zayed International Robotics Challenge (MBZIRC).

The authors are with the Autonomous Intelligent Systems Group, University of Bonn, Germany mbeul@ais.uni-bonn.de

[1] http://www.ais.uni-bonn.de/videos/IROS_2019_Beul

II. Related Work

In literature, several works address collision avoidance of MAVs in dynamic environments. The approaches can be mainly categorized into by considered dimension (2D vs. 3D), complexity of the environment, runtime, and the ability to deal with non-static obstacles.
While planning in complex environments can be solved with sample-based planners, grid-based planners, etc., these approaches often either do not consider system-dynamics or are not fast enough for real-time planning. Hence, most obstacle avoidance techniques for MAV assume a simple environment to be able to run with the control frequency of at least $10\,\mathrm{Hz}$ instead. Since they run as a lower layer in a hierarchy of planners, the missing consideration of complex environments can be compensated by higher layers.

Zhang et al. [6] use a set of precomputed alternative 3D paths to quickly react to suddenly perceived obstacles. The set is hierarchically organized in levels such that branches from alternative paths can be efficiently stored. The method works in 3D and is real-time capable ($<50\,\mu\mathrm{s}$) due to the precomputation. However, the precomputation needs to be performed whenever the target waypoint changes and generated trajectories are not optimal. Depending on the discretization, generated trajectories can be far from optimal whilst always collision free. Also, the evaluation only considers static obstacles.

Zhu and Alonso-Mora [7] propose a probabilistic collision avoidance method for navigation among moving obstacles. They explicitly consider the collision probability and solve a chance constrained nonlinear model predictive control problem (CCNMPC) to find valid trajectories. The method is real-time capable (average of $14.3\,\mathrm{ms}$ with peaks over $70\,\mathrm{ms}$) and can deal with constantly moving obstacles. The method explicitly respects uncertainty in the obstacle state. No information about the quality of the generated trajectories (e.g., optimality) is given.

Gao and Shen [8], and Gao et al. [9] use an optimization-based framework to generate collision free MAV trajectories. The method models both static and moving obstacles. It guarantees the trajectories to be optimal with respect to control effort, but computation is slow ($1.0\,\mathrm{s}$).

Liu et al. [10], use safe flight corridors to plan 3D trajectories that are dynamically feasible. They use quadratic programming techniques to find trajectories in a subset of the free space defined by convex polyhedra. Replanning takes 50-$300\,\mathrm{ms}$ and is thus limitedly real-time capable.

In [11], Liu et al. present a search-based motion planning method that used LQR-techniques to find dynamically feasible minimum-time trajectories in a discretized state-space. The method can generate smooth second- and third-order trajectories in 2D and 3D. The authors claim that the method can be used for fast online re-planning of trajectories, but the runtime for the 3D third-order system—equivalent to our method—is high (avg. $2.98\,\mathrm{s}$, max $9.5\,\mathrm{s}$). Also the optimality of the trajectories is governed by the discretization of the state space. Our method does not suffer from discretization-induced suboptimality. In [12], Liu et al. continue the work on the search-based planning method. They now model the MAV with an ellipsoid bounding box (instead of a sphere). The characteristics of the method including its capabilities and restrictions are similar to their previous work.

Lopez and How [13] use a 3D triple-integrator MAV-model similar to ours. Like Liu, they sample the state space to find collision-free trajectories, but dont't give information on the optimality of the solution. The authors report computation times in the low milliseconds, but due to the sampling-based nature of their method, it is unclear how the method scales with varying complexity of the environment.

Szmuk et al. [14] use convex optimization to find 3-dimensional paths in real-time. They show avoidance of up to three moving obstacles. The optimization takes up to $81.4\,\mathrm{ms}$ and is thus fairly real-time capable. The authors use a double-integrator model for the problem formulation. They justify this decision by the assumption that the attitude dynamics are significantly greater than the translational dynamics. This may be true for small agile MAVs like the ones used by the authors in the evaluation, but may introduce large errors when executed on a slower MAV.

Similar to our method, Mellinger et al. [15] present a method to calculate trajectories with free degrees of freedom (DoF). Trajectories also pass through fixed positions with optimal velocities. However, the solution is obtained by solving a constrained gradient descent problem including the numerical computation of the derivatives. In comparison to our method, generated trajectories are minimum snap and the trajectories are tracked by a separate controller instead of directly using the trajectory generator as feedforward with fast replanning.

As shown above, multiple approaches exist that generate smooth collision-free trajectories in environments with varying complexity and runtime requirements. Unfortunately, no standard benchmark exists to compare the methods, specifically for use on MAVs that require replanning times in the order of milliseconds, strict constraints on state and input variables, smoothness and satisfaction of optimality conditions like control effort or total trajectory time.

In the work of Kröger [16], third-order polynomial trajectories are generated that are similar to our work. The method lacks any kind of obstacle avoidance, though. Nevertheless, applications that utilize this method and need obstacle avoidance can be easily transferred to our proposed approach.

To our knowledge, no method exists that can generate smooth, optimal obstacle-avoiding 3D trajectories for MAVs that can be computed within typical control-loop frequencies. The method proposed in this work replaces our reactive low-level obstacle avoidance mechanism [17] that is not applicable to generating aggressive high-speed trajectories.

III. Partially Defined Target States

Our method extends our previously published work [5] in which we generate time-optimal second- and third-order trajectories from arbitrary fully defined start states to arbitrary target states with piecewise constant jerk output.

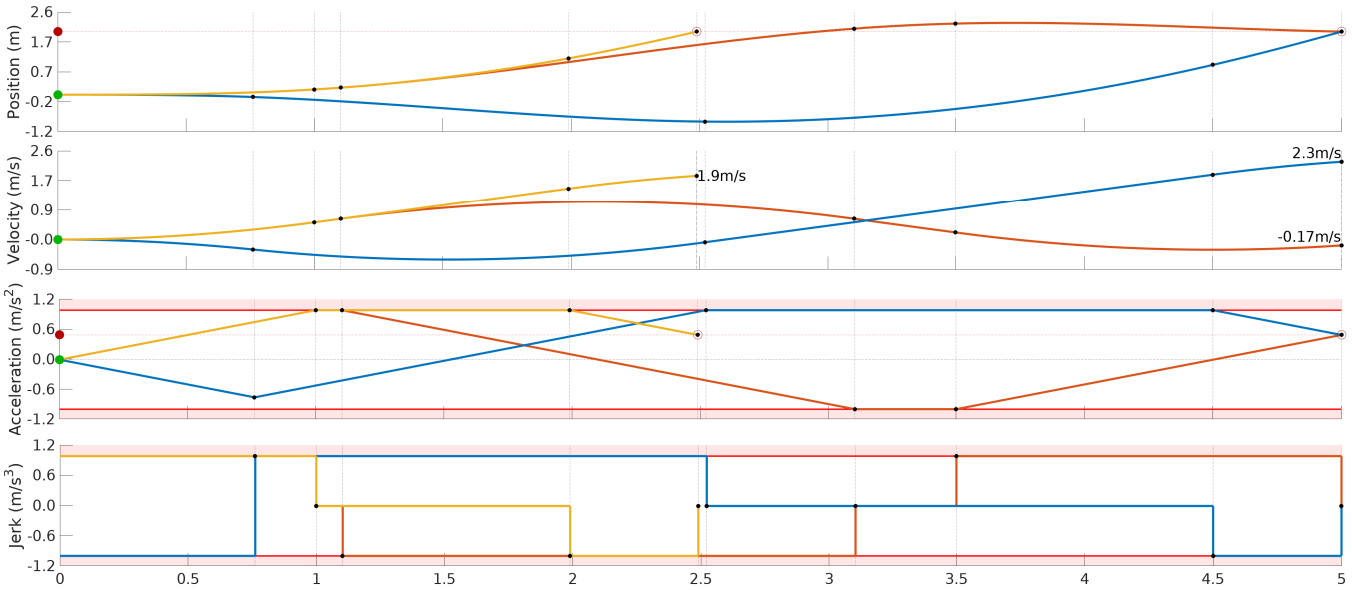


Fig. 2: Trajectories from state $\mathbf{x} = (0.0, 0.0, 0.0)^\intercal$ to $\mathbf{x} = (2.0, \mathit{NaN}, 0.5)^\intercal$. Yellow: The undefined DoF (velocity) is chosen such that the trajectory is time-optimal. This results in a target velocity of $1.9\,\mathrm{m/s}$. Blue, Orange: Trajectories that exactly last $5.0\,\mathrm{s}$. The undefined DoF is maximized (blue) or minimized (orange). One can see how the trajectories build up momentum by first accelerating in the opposing direction to maximize the time of acceleration (1.5-$5.0\,\mathrm{s}$ for blue and 2.1-$5.0\,\mathrm{s}$ for red trajectory.)

A. Previous Method for Trajectory Generation

Trajectories generated with our previous method respect per-axis constraints on minimum and maximum velocity, acceleration and jerk. Any number of individual axes can be coupled by synchronizing the total time of each trajectory. Since the method is very fast ($\ll 1\,\mathrm{ms}$ per axis per trajectory), it can be used in closed loop even for fast systems. With the ability to predict the target state, trajectories end in an optimal interception state when the target state is non-stationary. The method has been successfully used as model predictive controller on different micro aerial vehicles, in different research projects and robotic competitions. Like most works, our previous method requires fully defined target state inputs. Furthermore, it does not feature obstacle avoidance.

B. Solving Partially Defined Target States

In certain situations, the targeted state cannot be defined completely, e.g., when the MAV is supposed to pass through a narrow gap in a wall. Here, the 3-dimensional position and the pitch angle of the MAV is fixed such that it fits through the gap. Also the lateral velocity is fixed to zero such that the MAV passes the gap on the shortest path. However, the velocity orthogonal to the wall can be chosen freely. Also, our obstacle avoidance method described in Sec. IV builds upon the ability to find trajectories for partially defined target states.

Since our pipeline assumes a 3-dimensional state for each axis of the start- and target states, the permutations stated in Tab. I for the definition of target-states can occur. The first instance is the nominal condition with all derivatives defined. In the following, we describe how we treat the remaining six permutations.

TABLE I: Combinations of possible target state variables.

Position	✓	✓	✓	✓	✘	✘	✘
Velocity	✓	✓	✘	✘	✓	✓	✘
Acceleration	✓	✘	✓	✘	✓	✘	✓

✓: the value is defined, ✘: the value can be chosen freely

In order to generate trajectories without fully defined target states, we first define the trajectory for each axis as a system of 21 differential equations:

$$p_n = p_{n-1} + \int_{t_{n-1}}^{t_n} v_n \,\mathrm{d}t, \tag{1}$$

$$v_n = v_{n-1} + \int_{t_{n-1}}^{t_n} a_n \,\mathrm{d}t, \qquad n = \{1; \ldots; 7\} \tag{2}$$

$$a_n = a_{n-1} + \int_{t_{n-1}}^{t_n} j_n \,\mathrm{d}t, \tag{3}$$

with p_0, v_0, a_0 being the current state and p_7, v_7, a_7, the (possibly not fully defined) target state. Generated trajectories consist of up to $n = 7$ phases of constant jerk input, resulting in bang-singular-bang trajectories.
Depending on:

- whether the trajectory shall be time-optimal or with defined total duration,
- which DoF is defined (undefined DoF are indicated as *NaN*),
- which state limits are enforced (unlimited states are represented by setting the corresponding limit to *Inf*),
- whether the trajectory starts with state limits already violated,

we define sets of 21 second-order conditions to make the system of differential equations solvable. For example, when only acceleration and position is defined, like in Fig. 2, acceleration and velocity limits are not *Inf*, and the trajectory start is feasible, 20 different second-order condition combinations are possible. The optimal yellow trajectory in Fig. 2,

for example, features only three constant jerk times. Thus, $t_4, ..., t_7 := 0$. Furthermore, $a_2 := a_{max}$, $j_1 := j_{max}$, $j_2 := 0$, and $j_3 := j_{min}$. In contrast, the red trajectory features five jerk inputs. It is defined by the second-order conditions $t_4 \wedge t_5 := 0$, $a_2 := a_{max}$, $a_6 := a_{min}$, $j_1 \wedge j_7 := j_{max}$, $j_2 \wedge j_6 := 0$, $j_3 := j_{min}$, and $t_1 + ... + t_7 := T$. After defining the set of possible second-order conditions, we solve the equations. For t_1 and t_3 of the yellow example trajectory, this yields

$$t_1 = \frac{-a_{init} + a_{max}}{j_{max}}, \tag{4}$$

$$t_3 = \frac{-a_{max} + a_{target}}{j_{min}}. \tag{5}$$

The solution for t_2 is more complex and consists of ≈ 200 computational operations.

All equations that solve $t_1, ..., t_7$ for a particular trajectory shape are calculated only once and stored in a database to be evaluated quickly during operation. We make no assumption on the value of any variable during solution so that all solutions are persistent. Since they neither change before nor during runtime, no precomputation is needed.

Trajectories do not only respect constraints during the trajectory time interval, but the target state also is defined such that future state constraints are not infringed. For example, when acceleration is an undefined DoF and the target velocity is close to the maximum allowed velocity, the chosen acceleration is not allowed to be positive. Although velocity as well as acceleration are inside the allowed bounds, a future trajectory would overshoot the maximum velocity during the deceleration phase. The maximum allowed target state velocity v_{target} with a given acceleration is computed according to (6) and the corresponding acceleration a_{target} at a given velocity according to (7):

$$v_{target} = \frac{a_{max}^2}{2 \cdot j_{max}} + v_{min}, \tag{6}$$

$$a_{target} = \sqrt{2 \cdot j_{max} \cdot (v_{max} - v_{min})}. \tag{7}$$

As previously stated, trajectories can be time-optimal or have a fixed duration. If the duration is fixed, our method can be configured to maximize or minimize any undefined DoF. In Fig. 2, one can see that in comparison to the yellow trajectory, the additional time of $2.5\,\mathrm{s}$ is used to gain/loose velocity. This results in a target velocity of $2.3\,\mathrm{m/s}$, respective $-0.17\,\mathrm{m/s}$, in comparison to $1.9\,\mathrm{m/s}$. This behavior does not only work for the depicted example with undefined velocity, but also for all other combinations from Tab. I.

IV. Obstacle Avoidance

A collision with an obstacle in n-dimensional space happens exactly when at one time all positions of all axes of the trajectory are simultaneously inside a section of a corresponding n-dimensional obstacle. We assume the axes of the obstacles to be independent and axis-aligned. For two dimensions, this leads to rectangular obstacles. For three dimensions, this generates cuboid obstacles, and for higher dimensions hyperrectangles. Not only the position of the obstacle is defined in n-dimensions, but also its velocity, acceleration, etc.

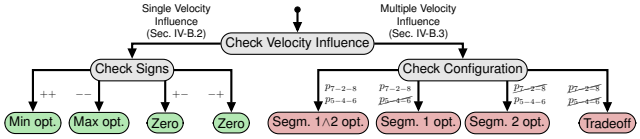


Fig. 3: Structure of the obstacle avoidance approach.

A. Collision Detection

We assume the current state and the target state to be in free space. In order to efficiently detect collisions of the trajectory between both, we make use of the piecewise polynomial formulation of our trajectories. For every axis, the trajectory consists of a concatenation of up to seven polynomials with rank ≤ 3. Assuming that the obstacle motion can also be described with a concatenation of polynomials with rank ≤ 3, we can determine the time of the collision by calculation of zero crossings of the combined function. We then decide for every timestamp if the trajectory enters or leaves an obstacle. Given we found all entering and leaving times for all axes, we now merge the timestamps to search for times when all axes are simultaneously inside an obstacle. These timestamps indicate the entering and leaving of the n-dimensional trajectory into/out of the obstacle. Including the directional information stated above, we can detect the first entering which is the point of collision. Evaluating the polynomials at this timestamp gives the state-vector at the collision including velocity and acceleration. When we find a collision, we insert viastates into the trajectory such that the trajectory bypasses the obstacle. Finding the full viastate such that the trajectory is still time-optimal is not trivial and will be described in the following sections.

B. Bound and Free Axes

As stated above, a collision occurs only when all axes are simultaneously inside an obstacle. We insert a viastate into the trajectory to temporally disconnect this simultaneity. For this, we choose two axes that we want to temporally disconnect. We call these two axes "bound axes". We call all other axes (if there are more than two axes in total) "free axes". First, we describe how to derive the state vector for the two bound axes in Sec. IV-B.1. After this, we describe the derivation of the state vector of the free axes in Sec. IV-B.4. The generated trajectory consist of a concatenation of up to 14 polynomials with rank ≤ 3 per axis.

1) State Vector for Bound Axes: We first define the viastate position of the bound axes to be at one of the four corners of the obstacle and project the MAV radius + an additional margin outwards. This determination is intuitive, since in order to find the fastest trajectory, one has to cut the corner as narrow as possible.

We now define the acceleration in these viastates to be zero. Setting the acceleration to the maximum/minimum in the viastates could gain a small amount of velocity, but if the state estimation of either the MAV or the obstacle is only slightly erroneous, the MAV is in a configuration that is close

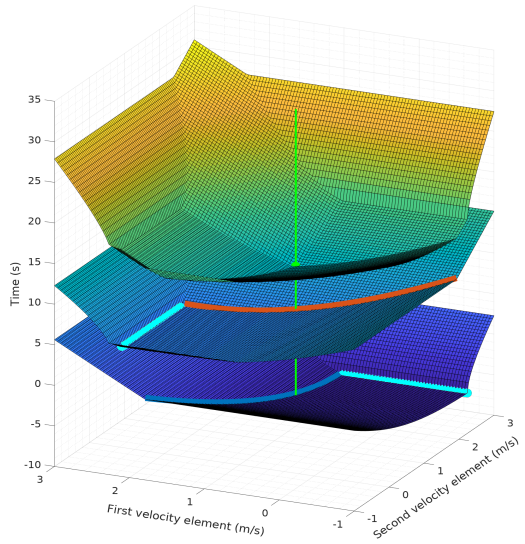


Fig. 4: Typical plot of the time needed for the first segment (middle), the second segment (bottom, shifted by 10s for better visibility) and the combined total trajectory (top). The time is shown in dependence of the velocity vector of the bound axes in the viastate. Local optima are marked with cyan dots. The global optimum is marked with a green dot and a vertical line. It can be seen that the global optimum lies at the intersection of one groove (marked blue and orange) of the individual segments. In Sec. IV-B.3, we detail our elaborated method to efficiently calculate these grooves and their intersection.

to the obstacle and possibly further accelerating towards the obstacle. By fixing the acceleration to zero, the MAV can react to these inaccuracies. So, a nonzero acceleration could be slightly faster (with massively increased collision risk), but under the assumption of zero acceleration, the generated evading trajectories are optimal.

Following this, we calculate the optimal viastate velocity of the two bound axes. Fig. 4 shows a sampled map of the time needed to complete the first segment from start to the viastate (middle), the second segment from the viastate to the target state (bottom, shifted by 10s for better visibility) and the combined total time for a typical trajectory (top). Although trajectory sampling with our method is fast ($\approx 519\,\mu s$ per trajectory), sampling the 20,000 trajectories (100×100 sampling grid for each segment) took $10.38\,s$ which makes it impossible to run at the control-frequency of the MAV. Besides the disadvantage of computational demands, this sampling-based method is comparatively inaccurate due to discretization. Although not viable for real-time calculation of the optimal velocity, the sampled map gives insight into the dependence of the timing from the two velocities and can be used to verify our real-time approach. We exploit the characteristics of the shown surface plots to develop a more sophisticated method and speed up the process by multiple orders of magnitude. Among others, we make use of the methods for partially defined target states we described in Sec. III. The process works as follows:

First, we use our method described in Sec. III to find time-optimal trajectories from the current state to the position of the viastate of the bound axes with undefined velocity. As

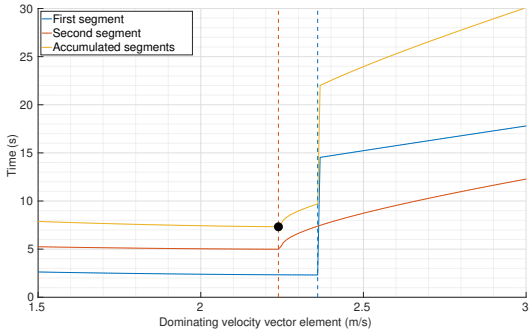


Fig. 5: Time needed for Segment 1 (blue), Segment 2 (orange), and total trajectory time (yellow) in dependence of a single velocity vector element. The other vector elements do not affect trajectory time. The optimum velocity for the individual segments is marked with a dashed line. The global optimum is marked with a black dot. The optimum is located at the optimal individual segment velocity that is closer to zero. This plot corresponds to the trajectory that goes over the obstacle in Fig. 1.

stated above, the velocity is chosen such that the trajectory is as fast as possible. We do the same backwards from the target state to the viastate position. This gives four velocities that would lead to optimal trajectories for both axes for the first and the second segment, respectively. We now compare which axis dominates the first and second segment. Fig. 3 shows the structure of the now following procedure. If a single axis dominates both segments, we continue as stated in Sec. IV-B.2 (left in Fig. 3). Otherwise, we continue as stated in Sec. IV-B.3 (right in Fig. 3).

2) Single Velocity Influence: If only a single velocity component impacts the total trajectory time, again multiple cases can occur. If the signs of the velocities are different, we choose the velocity to be zero. While a positive velocity would speed up one segment of the trajectory, it would slow down the other segment even more. In this case, none of the individual segments is optimal, but the combination of both shows optimality.

If the sign of the velocities is equally positive, we use the minimum optimal velocity. This means that the segment with the smaller velocity is optimal resulting in the combination of both segments being optimal. See Fig. 5 for clarification. Analogously, we choose the maximum optimal velocity if both signs are equally negative.

Since optimality is only depending on the velocity component we derived above, the other bound component can be chosen such that the total time of the trajectory does not change. To calculate the region of velocities that does not impact total trajectory time, we employ the approach we described in Sec. III. We first determine the time the trajectory needs for the first and second segment with the computed optimal velocity individually. We then find the maximum and minimum velocity that is possible in this time for the second bound axis. We superimpose the intervals and get an interval from that we can freely choose the velocity without compromising optimality. While we could, e.g., choose the midpoint of the interval, we choose the end of the interval that maximizes the curvature from the

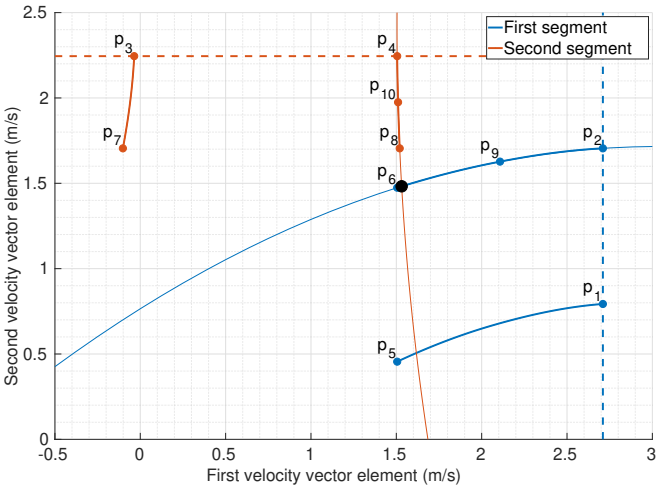


Fig. 6: Optimal tradeoff between two trajectory segments. For the first trajectory segment (blue), a viastate velocity of $2.71\,\mathrm{m/s}$ (dashed line) is optimal. If this velocity would be chosen, the other velocity component could be freely chosen between p_1 and p_2 without changing the total trajectory time. The $p_{1\text{-}2}$ line corresponds to a local minimum in Fig. 4. For the second segment, a velocity of $2.25\,\mathrm{m/s}$ would be optimal (orange dashed line). The other component is allowed to change between p_3 and p_4 (corresponding to the other local minimum Fig. 4). We first fit second order polynomials based on three dots per segment (thin lines). The lines $p_{2\text{-}9\text{-}6}$ and $p_{4\text{-}10\text{-}8}$ correspond to the blue and orange grooves in Fig. 4. We then calculate the intersection (black dot). This point represents the globally optimal velocity vector that is needed to generate time-optimal trajectories. This plot corresponds to the trajectory that goes left around the obstacle in Fig. 1.

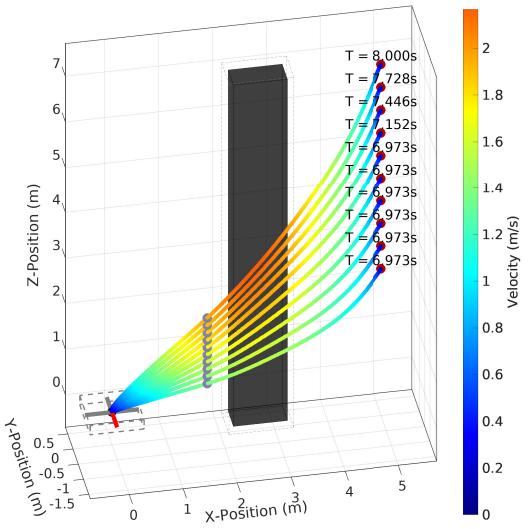


Fig. 7: Trajectory stack with bound x- and y-axis and free z-axis. The transition between trajectories with dominating bound (bottom 6) and free (top 4) axes can be seen. The total trajectory duration does not change for the bottom 6 trajectories since the free z-axis is faster than both bound axes. The z-viastate adapts in position, velocity and acceleration. The free z-axis dominates the top 4 trajectories so that the viastate in the bound axes in turn adapts to the free axis.

trajectory away from the obstacle.

3) Multiple Velocity Influence: If both velocity components of the bound axes influence the total trajectory time, we employ the method illustrated in Fig. 6. First we calculate the optimum velocities for both bound axes for both segments. We now determine which axis dominates the total time. The optimal velocity of the dominant axes are shown in Fig. 6. We now determine the maximum and minimum velocities that are achievable in the time it takes to gain the optimal velocity (p_1, p_2, p_3, p_4) for both segments. Afterwards, we decide which point lies closer to the optimal counterpart of the other segment (p_2 respective p_4). This gives the direction of the limit.

After that, we determine p_5 and p_6, and p_7 and p_8 respectively. We do this by calculating the required time for Segment 1 to reach p_4 speed ($1.51\,\mathrm{m/s}$) and again the possible maximum and minimum velocity component. The velocity vector is the x-component from p_4 and the y-component of p_5 and p_6. The same is conducted for p_7 and p_8 with the y-component of p_2. Subsequently four cases can occur:

1) p_2 lies between p_7 and p_8 in x-direction and p_4 lies between p_5 and p_6 in y-direction. Both segments are optimal and the optimum velocity of both segments can be used.

2) p_2 lies between p_7 and p_8 in x-direction, but p_4 does not lie between p_5 and p_6 in y-direction. The first segment dominates the trajectory. The optimum lies at p_2 for both axes.

3) p_4 lies between p_5 and p_6 in y-direction, but p_2 does not lie between p_7 and p_8 in x-direction. The second segment dominates the trajectory. The optimum lies at p_4 for both axes.

4) Neither does p_2 lie between p_7 and p_8 in x-direction, nor does p_4 lie between p_5 and p_6 in y-direction. Here, a tradeoff between the calculated velocities determines the optimum. This case is depicted in Fig. 6. The tradeoff is calculated by first determining p_9 and p_{10}. The x-coordinate of p_9 lies in the middle of p_2 and p_6. The y-direction is again calculated by first determining the time needed to achieve p_9 speed. Afterwards, the maximum/minimum (depending on the direction of the limit) achievable velocity in this time is calculated. This gives the y-coordinate of p_9. Analogously, p_{10} is derived. We now approximate the optimal tradeoff for both bound axes separately by fitting a second order polynomial ((8) and (9)) through points p_2, p_9, and p_6, respectively p_4, p_{10}, and p_8 (thin lines). Subsequently, we now find the intersection of both polynomials by inserting (9) into (8):

$$y = a_1 \cdot x^2 + b_1 \cdot x + c_1, \tag{8}$$

$$x = a_2 \cdot y^2 + b_2 \cdot y + c_2, \tag{9}$$

$$\begin{aligned} 0 = {} & t^4 \cdot a_1^2 \cdot a_2 \\ & + t^3 \cdot (2 \cdot a_1 \cdot b_1 \cdot a_2) \\ & + t^2 \cdot (2 \cdot a_1 \cdot c_1 \cdot a_2 + b_1^2 \cdot a_2 + a_1 \cdot b_2) \\ & + t \cdot (2 \cdot b_1 \cdot c_1 \cdot a_2 + b_1 \cdot b_2 - 1) \\ & + c_1^2 \cdot a_2 + c_1 \cdot b_2 + c_2. \end{aligned} \tag{10}$$

This results in the fourth-order polynomial in (10). We do not simply equalize both equations since in the frame of the first axis the second axis is a transposed polynomial. By finding

the root of (10) that lies in the interval shown, we find the optimal tradeoff velocity vector marked with a black dot in Fig. 6.

4) State Vector for Free Axes: After both 3-dimensional viastates of the two bound axes are calculated, we calculate the corresponding state for the remaining free axes.

For this, we first compute optimal trajectories for all free axes. We then compare the maximum free trajectory time with the bound trajectory time and choose the viastate of the free axes, depending on which one is larger.

If the bound axes dominate the total trajectory (bottom six trajectories in Fig. 7), we synchronize all free axes to the total trajectory time of the bound axes by the methods presented in [5]. The derivatives in the viastate of the free axes are simply the states of the free axes at the time the bound axes pass the viastate. If the free axes dominate the total trajectory, however, (top four trajectories in Fig. 7) the free axis determines the viastate and we discard the viastate calculated in Sec. IV-B.1. Instead, the viastate for the bound axes is the state when it passes the obstacle edge. Since the viastate for the bound axes is uncritical to guarantee time-optimality, it can vary as long as the free axis still dominates the total trajectory time. We choose the bound axis viastate such that the viastate is passed at a time that scales the trajectory segments.

Fig. 7 illustrates multiple trajectories with bound x/y axes and a varying free z-axis. It shows that the total trajectory time is constant as long as the bound axes dominate the trajectory. Only the top four trajectories are dominated by the free axis.

V. Evaluation

We evaluate our method in simulation with arbitrarily placed obstacles. The dynamics of the MAV and the obstacle are simulated with MATLAB utilizing a realistic parameter set. A video showing avoidance of static and non-static obstacles can be found on our website[1]. Here, we also publish recorded datasets, tools, and parts of our pipeline.

We first evaluate our approach to partially defined target states. Based on this, we evaluate our obstacle avoidance pipeline. Since we conducted extensive flight tests with the predecessor of our software published in e.g., [2], [17], and [18], the practical applicability of the method to real MAVs ranging from $3.0\,\mathrm{kg}$ to $11.2\,\mathrm{kg}$ has already been shown.

A. Partially Defined Target States

First, we evaluate the reliability of our method by fuzz testing our approach. Therefore, random data including *NaN* for the target state and *Inf* for the state limits are generated. Our method successfully found solutions for over 10,000,000 consecutive trajectories.

B. Obstacle Avoidance

In order to test the soundness of our approach, obstacle avoidance was tested in over 100 different scenarios with changing obstacle configurations.

We measure the time, our obstacle avoidance method requires on a single core of a Intel Core i7-4710MQ processor

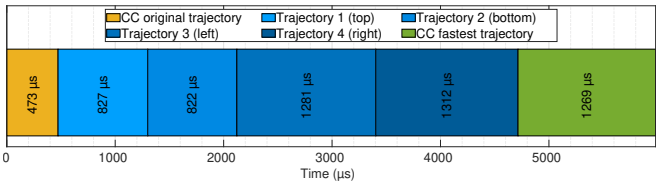


Fig. 8: Results of the timing experiment for the three-dimensional trajectories shown in Fig. 1. The total computation time of $5.98\,\mathrm{ms}$ is mostly needed for generating the individual evading trajectories. Collision checking (CC) only takes a quarter of the time.

for the scenario shown in Fig. 1. In Fig. 8, we show the results of our profiling for the experiment. It can be seen that our whole pipeline computes in less than $6\,\mathrm{ms}$, including collision checking. Since the four individual trajectories are independent of each other, this process can be easily parallelized.

We further extend our evaluation to a constantly moving obstacle depicted in Fig. 9. We observe that the movement has no measurable impact on the computation time. We also examine the dependence of the runtime to the dimensionality of the planning problem. Since our method always temporally disconnects two axes per instance, the number of possible evading trajectories p regarding n axes scales with

$$p = \frac{n!}{(n-2)!}. \tag{11}$$

Experiments with four axes (x, y, z, yaw) also confirm this. Here, obstacles are 4-dimensional and problems like "Do not look in direction X while being in the volume Y." can be represented. Depending on the configuration of the start- and target conditions, one can further omit several bound axis combinations. Thus, our experiments only show four trajectories instead of six. In Fig. 1, the collision of the original trajectory happens in the y/z plane. Thus, the bound axis-combinations are x/y (left/right) and x/z (top/bottom). A y/z combination makes no sense since the main axis of movement is in x-direction.

Before executing the fastest trajectory of the set, it is checked for collisions with all obstacles (green bar in Fig. 8). If it collides with an obstacle, the next bests trajectory is chosen. This process is repeated until the fastest trajectory without collision is found. So, if for example another obstacle was standing on top of the obstacle in Fig. 1, the next best trajectory below the obstacle would be selected.

Please note that our method does *not* explicitly consider uncertainty in state estimation of MAV and obstacle and assumes a constant motion of the obstacle. However, due to the fast computation time, this can be compensated by a high replanning rate. The high replanning rate also promotes the performance when obstacles are not known in advance but are perceived abruptly and with uncertain velocity estimates. Regarding uncertainty, the position margin of the obstacle can be adjusted depending on the used sensor. Also due to the obstacle representation by two independent borders, obstacles can also grow or shrink over time representing growing uncertainty over time.

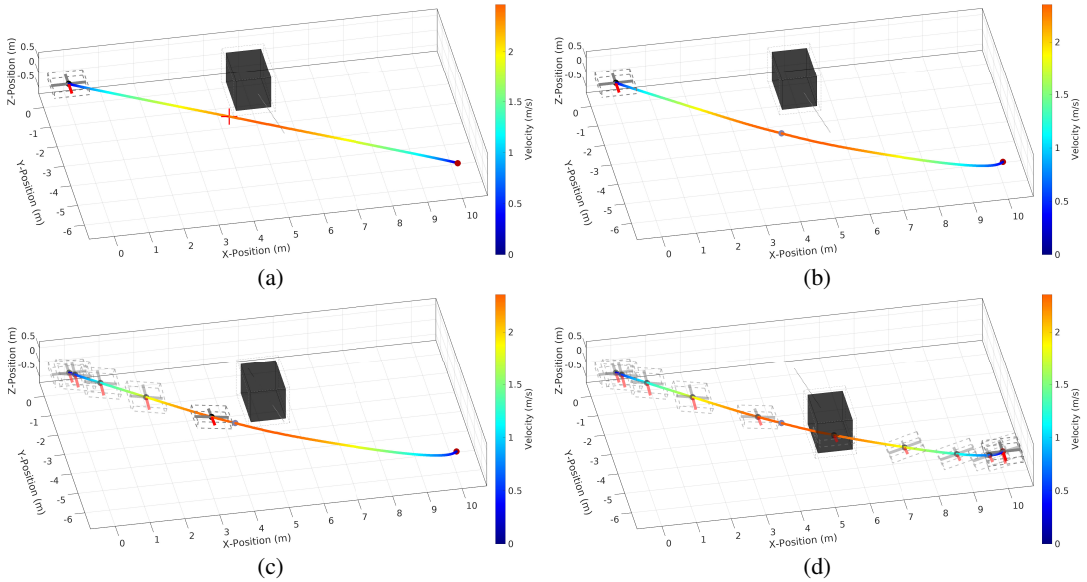


Fig. 9: Avoiding a moving obstacle. (a) Our method detects a future collision at the position marked with the red cross of the original trajectory and the moving obstacle trajectory (marked with the gray dotted line). With a static obstacle no collision would occur, but since our pipeline explicitly considers the movement, the collision is reliably detected. (b) It now generates an optimal viastate (gray) including an optimal velocity vector that guides the MAV around the obstacle incorporating the movement of the object. (c) The trajectory is executed. (d) The MAV successfully avoided the obstacle. The trajectory only took $9.52\,\mathrm{s}$ vs. the original $9.45\,\mathrm{s}$.

VI. Conclusion

In this paper, we propose a novel method to plan smooth, time-optimal obstacle avoiding trajectories which is realized through viastate insertion. Additional viastates are inserted into the trajectory and guide the trajectory around the obstacle. Finding the optimal position derivatives such that the trajectory is still time-optimal is accomplished by a method that requires only minimal computational effort.

Our evaluation shows that due to the fast runtime of our method, the approach can be used in real-time to avoid suddenly perceived obstacles in the flight path of the MAV. Our approach works with an arbitrary number of dimensions and with constantly moving obstacles.

We demonstrated the method in simulation and fuzz-tested components of the system several million times to show the robustness of the approach.

References

[1] DARPA, "Fast Lightweight Autonomy (FLA)," https://www.darpa.mil/program/fast-lightweight-autonomy, 2015, accessed: 2019-02-10, Solicitation number DARPA-BAA-15-16.

[2] M. Nieuwenhuisen, M. Beul, R. A. Rosu, J. Quenzel, D. Pavlichenko, S. Houben, and S. Behnke, "Collaborative object picking and delivery with a team of micro aerial vehicles at MBZIRC," in *Proc. of European Conf. on Mobile Robots (ECMR)*, 2017.

[3] MBZIRC, "MBZIRC challenge description," https://www.mbzirc.com/challenge/2020, 2019, accessed: 2019-01-15.

[4] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *Intelligence/sigart Bulletin - SIGART*, vol. 37, pp. 28–29, 12 1972.

[5] M. Beul and S. Behnke, "Fast full state trajectory generation for multirotors," in *Proc. of Int. Conf. on Unmanned Aircraft Systems (ICUAS)*, 2017.

[6] J. Zhang, R. G. Chadha, V. Velivela, and S. Singh, "P-CAP: Pre-computed alternative paths to enable aggressive aerial maneuvers in cluttered environments," in *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2018.

[7] H. Zhu and J. Alonso-Mora, "Chance-constrained collision avoidance for MAVs in dynamic environments," in *IEEE Robotics and Automation Letters (RA-L)*, 2019.

[8] F. Gao and S. Shen, "Quadrotor trajectory generation in dynamic environments using semi-definite relaxation on nonconvex QCQP," in *Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2017.

[9] F. Gao, W. Wu, J. Pan, B. Zhou, and S. Shen, "Optimal time allocation for quadrotor trajectory generation," in *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2018.

[10] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar, "Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-D complex environments," in *Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2017.

[11] S. Liu, N. Atanasov, K. Mohta, and V. Kumar, "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.

[12] S. Liu, K. Mohta, N. Atanasov, and V. Kumar, "Search-based motion planning for aggressive flight in SE(3)," in *Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2018.

[13] B. T. Lopez and J. P. How, "Aggressive 3-D collision avoidance for high-speed navigation," in *Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2017.

[14] M. Szmuk, C. A. Pascucci, and B. Açikmeşe, "Real-time quadrotor path planning for mobile obstacle avoidance using convex optimization," in *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2018.

[15] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2011.

[16] T. Kröger, "Opening the door to new sensor-based robot applications - the reflexxes motion libraries," in *Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2011.

[17] M. Beul, D. Droeschel, M. Nieuwenhuisen, J. Quenzel, S. Houben, and S. Behnke, "Fast autonomous flight in warehouses for inventory applications," in *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2018.

[18] M. Beul, S. Houben, M. Nieuwenhuisen, and S. Behnke, "Fast autonomous landing on a moving target at MBZIRC," in *Proc. of European Conf. on Mobile Robots (ECMR)*, 2017.