IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), Abu Dhabi, UAE, 2020.

Trajectory Generation with Fast Lidar-based 3D Collision Avoidance for Agile MAVs

Marius Beul, and Sven Behnke

***Abstract*— Micro aerial vehicles (MAVs), are frequently used for exploration, examination, and surveillance during search and rescue missions. Manually piloting these robots under stressful conditions provokes pilot errors and can result in crashes with disastrous consequences. Also, during fully autonomous flight, planned high-level trajectories can be erroneous and steer the robot into obstacles.**

In this work, we propose an approach to efficiently compute smooth, time-optimal trajectories MAVs that avoid obstacles. Our method first computes a 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. Afterward, we efficiently check the trajectory for collisions in the 3D-point cloud, recorded with the onboard lidar. We exploit the piecewise polynomial formulation of our trajectories to analytically compute axis-aligned bounding boxes (AABB) to speed up the collision checking process. If collisions occur, we generate a set of alternative trajectories in real-time. Alternative trajectories bring the MAV in a safe state, while still pursuing the original goal. Subsequently, we choose and execute the best collision-free alternative trajectory based on a distance metric.

The evaluation in simulation and during a real firefighting exercise shows the capability of our method.

I. Introduction

Micro aerial vehicles (MAVs) are becoming a key element in reducing the required risks, time, and costs for search and rescue missions, aerial reconnaissance, and disaster examination. In most cases, a human pilot operates the MAV remotely to fulfill a mission, or the MAV is following a predefined path of GPS waypoints at an altitude assumed to be obstacle-free. However, permanent line-of-sight from the pilot to the MAV may not be maintainable at all times due to large obstacles. Also, thin obstacles like antennas or power lines may only be perceivable inaccurately by the pilot. Other (probably moving) participants in the airspace like drones, rescue helicopters, birds, or debris proposes a risk for the MAV. Furthermore, during a real rescue scenario, the operator's cognitive load is significant [1], which provokes human errors. The loss of an MAV due to a crash is expensive, but even worse, it can have disastrous consequences for the executed mission.

To tackle these challenges, in this article, we present an efficient method to compute smooth, time-optimal trajectories for MAVs that automatically avoid obstacles. On the one hand, our method can be used as an obstacle avoidance and control layer in a hierarchy of planners for fully autonomous flight. On the other hand, since it accepts intuitive setpoints, it can be employed to execute the pilot's commands during manual flight, providing an additional safety layer. This article extends our previous work on trajectory generation for MAVs [2] with an obstacle avoidance feature,

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

Institute for Computer Science VI, Autonomous Intelligent Systems, University of Bonn, Endenicher Allee 19a, 53115 Bonn, Germany, mbeul@ais.uni-bonn.de

Fig. 1. Operation of our trajectory generation method onboard an MAV during a real firefighting exercise.

Our main contributions are

- computation of dynamically feasible collision-free alternative trajectories,
- fast analytical bounding box computation for cropping of large point clouds,
- analytical modeling and integration of sensor coverage during planning, and
- evaluation in simulation and a real-world firefighting exercise.

Fig. 1 shows our MAV supporting firefighters by surveying the area with its multimodal sensor setup.

II. Related Work

Low-level obstacle avoidance is an active field of research.

For example, Zhang et al. [3] present a method to instantaneously avoid static obstacles by following precomputed paths in cluttered environments. The paths are hierarchically organized, such that branches from alternative paths can be efficiently stored and executed. The method is fast ($\leq 50\,\mathrm{ms}$), but generated trajectories are not optimal.

Similarly, also Barry et al. [4] show impressive results with fast flights of up to $14\,\mathrm{m/s}$. Like the above-noted approach, the authors precompute trajectories from which they select a collision-free instance during runtime. In contrast,

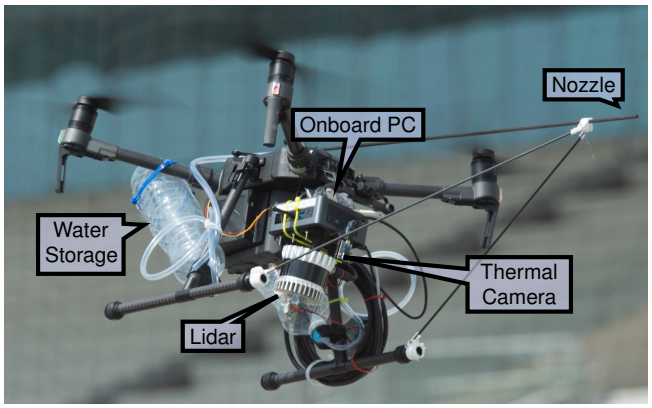

Fig. 2. Design of our MAV equipped with an Ouster OS1-64 Gen 1, a FLIR Lepton thermal camera, a fire extinguisher, and a lightweight but powerful onboard computer.

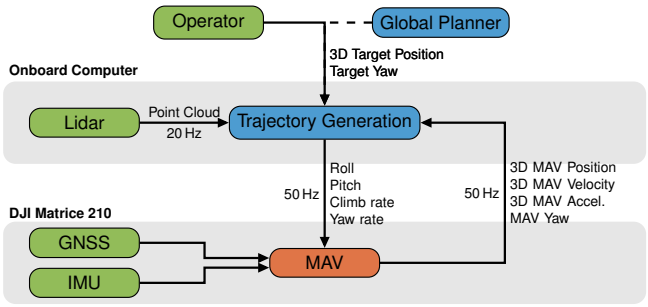

Fig. 3. Structure of our method. Green boxes represent external inputs like sensors, blue boxes represent software modules, and the red box indicates the MAV flight control. All software components use ROS as middleware. Position, velocity, acceleration, and yaw are allocentric. Commands for our trajectory generator can either directly come from an operator or from a high-level planner.

our approach quickly generates trajectories during runtime, incorporating the current continuous 3D state of the robot.

Also, the idea to instantaneously react to sensor measurements instead of planning a trajectory is not new, and many different approaches from vector field histograms [5] to potential field-based obstacle avoidance have been presented.

For example, Falanga et al. [6] use an event camera to quickly detect obstacles in the vicinity of the MAV. While in contrast to normal RGB-cameras, event cameras are advantageous in terms of latency and dynamic range, they still suffer from a narrow field of view and low range compared to lidars. To guarantee a low latency, instead of sophisticated trajectory generation techniques, the MAV performs potential field-based obstacle avoidance.

Similarly, also Nieuwenhuisen et al. [7] perform potential field-based obstacle avoidance with an MAV. To account for the MAV's non-neglectable dynamics, the authors extend the method with a motion model and predict the MAV state in the near future. The authors report that trajectories with active motion model are smoother than with pure reactive potential field obstacle avoidance.

The idea to generate a set of alternative dynamically feasible trajectories and subsequently selecting a suitable one has been shown in, e.g., [8]. Instead of obstacle avoidance, the authors use the motion primitives to catch a flying ball with an MAV. Like our approach, the authors use closed-form solutions to find suitable trajectories, resulting in computation times in the order of microseconds. The alternative trajectories are not time-optimal and not guaranteed to be collision-free.

Lindqvist et al. [9] present an MPC that features collision avoidance and can also deal with moving obstacles. The authors use a nonlinear solver to find collision-free trajectories within the $50\,\mathrm{ms}$ replanning time. In contrast to our technique, the pipeline only predicts $2\,\mathrm{s}$ of the trajectory, making it unsuitable for fast flight or fast obstacles. Instead of onboard sensors, the technique relies on a known obstacle path, measured with a motion capture system.

Trajectories produced by all methods mentioned above are dynamically feasible, but *not* time-optimal. To avoid nimble obstacles (or during fast flight), the control inputs of the robot have to be maximized to achieve maximum dexterity. Thus, similar to our approach, the method from Lopez and How [10] generates time-optimal state and input constrained trajectories by sampling terminal states. With a computation time of 2-5 µs, the method is real-time capable. The authors use an RGB-D camera to measure a point cloud in front of the MAV. Without omnidirectional perception (e.g., obtained with a lidar), safe trajectories are forced to lie in front of the MAV. An additional drawback of the method is that it works in (constant) velocity space instead of position space; thus, e.g., it restricts the MAV from hovering. The method assumes obstacles to be static.

Watterson and Kumar [11] report a hybrid approach for obstacle avoidance. On the one hand, the proposed method uses a receding horizon control policy (RHCP) to steer an MAV through a cluttered environment. On the other hand, the approach always guarantees that there exists a safe stopping policy that brings the MAV to hover. Similarly, our approach searches for multiple classes of trajectories.

To our knowledge, no method exists that can compute smooth, time-optimal obstacle-avoiding 3D trajectories for MAVs within typical control-loop frequencies. The method proposed in this work replaces our reactive low-level obstacle avoidance mechanism [12] that does not scale to aggressive high-speed trajectories.

III. System Setup

In the following sections, we first describe the hardware of our MAV in Sec. III-A. We continue by presenting our approach to point cloud filtering with axis-aligned bounding boxes in Sec. III-B. Collision checking is described in Sec. III-D and Sec. III-E. Lastly, we present our method for generating alternative trajectories in Sec. III-F.

A. Hardware Design

To support firefighters during their mission, we developed the MAV shown in Fig. 2. It is based on the DJI Matrice 210 platform and features a lidar to perform simultaneous localization and mapping (SLAM), and a thermal camera to detect and map, e.g., fires or victims. Furthermore, it features small but fast Intel Bean Canyon NUC8i7BEH onboard PC with

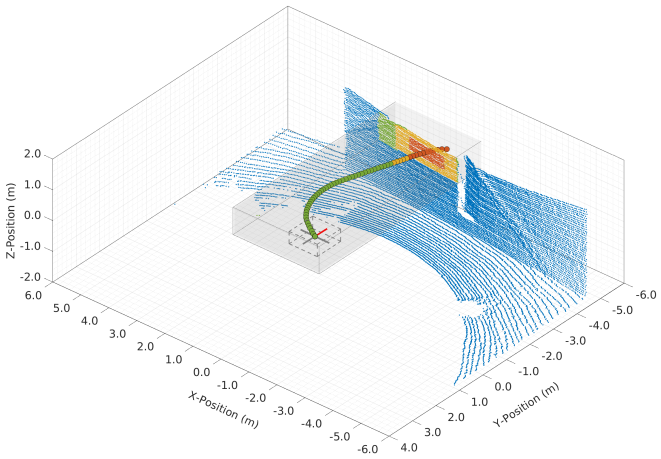

Fig. 4. Analytical computation of an axis-aligned bounding box (AABB) for a polynomial trajectory (gray, expanded by the MAV dimensions). The AABB is used to select a subset (green, yellow, red) of the point cloud (blue). Afterward, we sample the trajectory with a constant position Δp and check for collisions with the MAV (red) and collisions with the MAV expanded by a warning distance (yellow). The points in the point cloud subset are safe (green), within warning distance of at least one trajectory sample (yellow), and in collision distance of at least one trajectory sample (red).

an Intel® Core™ i7-8559U processor and 32 GB of RAM. To extinguish small fires, the MAV is equipped with a fire extinguisher. We use the robot operating system (ROS) as middleware on the MAV.

Fig. 3 shows an excerpt of our software pipeline, running onboard the MAV. In the shown example, an operator directly defines 4D setpoints (X, Y, Z, Yaw) for the trajectory generator. Instead, the commands can also be generated by a higher layer planning pipeline during a fully autonomous flight (see [12]).

The depicted trajectory generator[1] generates smooth third-order time-optimal trajectories that respect asymmetrical state and input constraints. It is described in detail in [2], [13], and [14]. We now extend the capabilities of our trajectory generator by real-time obstacle avoidance employing point clouds recorded by the onboard lidar.

Since our lidar produces up to 65 536 measurements per scan (1 310 720 per second), we first demonstrate how we analytically compute bounding boxes to crop the large point clouds in Sec. III-B. We then describe our measurement model and how we check if trajectories pass unknown space in Sec. III-E. In Sec. III-C, we show how we exploit the polynomial formulation of our trajectories to analytically sample the trajectories with a constant position offset (per dimension). Subsequently, we present how we generate alternative collision-free trajectories that bring the MAV in a safe state while still pursuing the original goal in Sec. III-F.

B. Analytical Bounding Box

To speed up computation, we first filter the point cloud. Instead of removing random points or other filtering methods, we crop the point cloud such that points that are outside of an axis-aligned bounding box (AABB) around the trajectory

[1] https://github.com/AIS-Bonn/opt_control

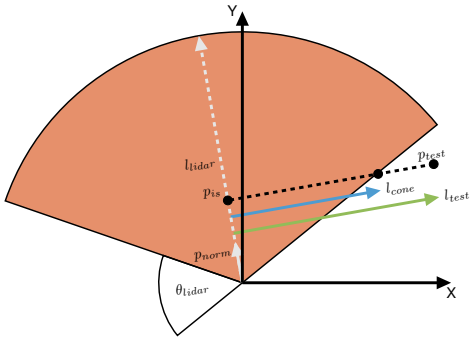

Fig. 5. Simplified schematic of our lidar model projected onto a 2D-plane. By checking if $l_{test} < l_{cone}$, we can determine if the point p_{test} is inside unobservable space. For simplicity, we only show one of two cones.

are discarded. To do so, we analytically compute the axis-aligned bounding box employing the polynomial formulation of the trajectory.

As described in [2], our trajectories consist of a concatenation of n-dimensional polynomials with constant n-dimensional jerk. In each segment and each dimension, the position p follows Eq. 1 with the jerk j, the acceleration a, and the velocity v at the beginning of the segment. We determine the global maxima and minima of each polynomial by deriving the position with respect to time. With t_{ex} being the segment's extremal times from Eq. 2 and Eq. 3, we subsequently check if the extremal time lies within the segment's interval. If so, the segment produces a global extremum of the trajectory at point p_{ex} with Eq. 4. We execute this procedure for each dimension for each segment to compute the axis-aligned bounding box. As a last step, we expand the bounding box by the MAV size.

$$p(t) = v\,t + \tfrac{1}{2}\,a\,t^2 + \tfrac{1}{6}\,j\,t^3 \tag{1}$$

$$p'(t_{ex}) := 0 \Rightarrow$$

$$0 = v + a\,t_{ex} + \tfrac{1}{2}\,j\,t_{ex}^2 \tag{2}$$

$$t_{ex} = -\frac{(a \pm \sqrt{(a^2 - 2\,j\,v)}}{j} \tag{3}$$

$$p_{ex} = p + v\,t_{ex} + \tfrac{1}{2}\,a\,t_{ex}^2 + \tfrac{1}{6}\,j\,t_{ex}^3 \tag{4}$$

Fig. 4 displays an example AABB. In this typical example, only 1372 of the 65 536 total scan points are in the proximity of the trajectory and considered for the following computations. This means a reduction of 97.9 %.

C. Constant Distance Sampling

We now sample the trajectory with a constant time difference, or with a constant position difference. We depict sampling with a constant time difference in Eq. 5, with p_{n+1} being the position after Δt with the initial position p_n, velocity v_n, acceleration a_n, and jerk j_n.

$$p_{n+1} = p_n + v_n\,\Delta t + \tfrac{1}{2}\,a_n\,\Delta t^2 + \tfrac{1}{6}\,j_n\,\Delta t^3 \tag{5}$$

However, sampling the trajectory with a constant time difference gives inhomogeneous resolution over the trajectory. For collision checking, we thus prefer the (more complicated)

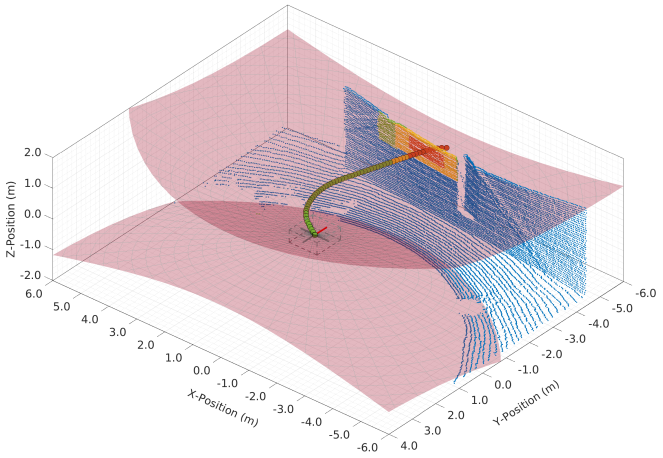

Fig. 6. Analytical lidar coverage computation. We model the lidar sensor coverage by two cones (red) with a $33.2°$ opening angle. The range of the lidar is $120\,\mathrm{m}$. We detect and reject trajectories that pass through unobservable space.

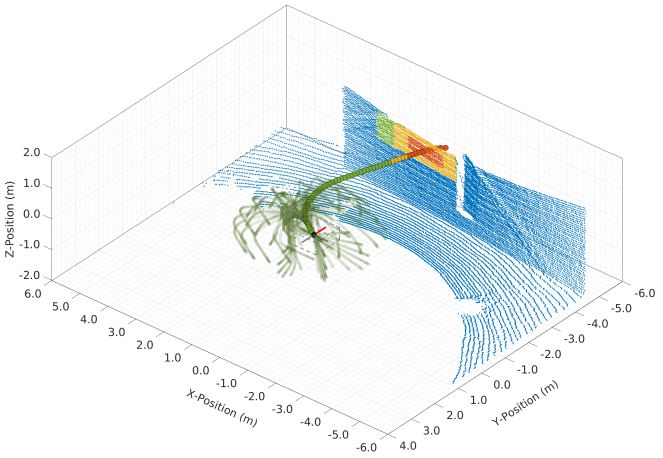

Fig. 7. Alternative safe trajectories that end in concentric spheroids around the MAV. All 67 trajectories are safe in terms of collisions with the point cloud *and* do not pass unobservable space.

constant distance sampling. To do so, we analytically solve Eq. 6 for t in a preprocessing step.

$$\Delta p = v_n\, t + \tfrac{1}{2}\, a_n\, t^2 + \tfrac{1}{6}\, j_n\, t^3 \tag{6}$$

This gives three analytical equations for t_{1-3}. For brevity, we refrain from picturing the solutions here. Subsequently, during runtime, we evaluate the three solutions with the desired position offset Δp, and select the smallest positive non-imaginary t of all dimensions. We compute the position at this timestep with Eq. 5 as the next sample and continue this procedure until the end of the trajectory is reached. Exemplarily, Fig. 4 shows a trajectory sampled with $\Delta p = 10\,\mathrm{cm}$ in each dimension. Since trajectories can have multiple jerk segments (e.g., the trajectory in Fig. 4 has 18 segments), we also account for effects at the segment borders that we can not cover here for brevity. The distance offset in each dimension does not have to be equal. Thus by assigning inhomogeneous Δp, individual dimensions can be sampled denser or sparser.

D. Collision Checking

With the trimmed point cloud from Sec. III-B, and the trajectory positions equidistantly sampled from Sec. III-C, we check each dimension n of each trajectory position $p_{test,n}$ for two distances $l_{test} = l_{warn} \vee l_{coll}$ indicating if a point $p_{pc,n}$ is within warning distance or if a point is so close that it causes a collision. Checking for two individual distances allows for hysteresis at the border of obstacles and for a more sophisticated validation (see Sec. III-F).

$$\begin{aligned} b_{close,n} = & p_{pc,n} > p_{test,n} - l_{test,n} \quad \wedge \\ & p_{pc,n} < p_{test,n} + l_{test,n} \end{aligned} \tag{7}$$

Since the world is not static, we model moving observations with a constant velocity model. Although our lidar does not give 3D velocities for the measurements, future sensor modalities like radar possibly will do so, and thus we extend the approach to collision checking to points moving with the 3D velocity vector $v_{pc,n}$.

For each dimension n, we calculate the times $t_{1,n}$ and $t_{2,n}$, when the point cloud point with position $p_{pc,n}$ and velocity $v_{pc,n}$ enters resp., leaves the axis-aligned hyperrectangle defined by center point $p_{test,n}$ and elongation $l_{test,n}$ with Eq. 8 and Eq. 9. Subsequently, we sort all times according to their size and check if there exists a period where all dimensions of the line are inside the hyperrectangle, as shown in Eq. 10 and Eq. 11. If this is the case, the point cloud point moves through the AABB. Thus, if the line defined by p_{pc} and v_{pc} crosses the hyperrectangle defined by p_{test} and l_{test}, it is causing a collision with $b_{close} = \text{true}$.

$$t_{1,n} = \frac{p_{test,n} - l_{test,n} - p_{pc,n}}{v_{pc,n}} \tag{8}$$

$$t_{2,n} = \frac{p_{test,n} + l_{test,n} - p_{pc,n}}{v_{pc,n}} \tag{9}$$

$$t_{sort} = \mathrm{sort}(t_n) \tag{10}$$

$$b_{close} = \max(t_{sort,1}) < \min(t_{sort,2}) \tag{11}$$

In this paper, we mark each trajectory sample within a warning distance of at least one point (and the corresponding points) yellow. Points that are causing a collision, and the corresponding trajectory points, are marked red.

E. Analytical Coverage Test

Besides checking for collisions with the (potentially moving) point cloud, we also check if the trajectory passes unobservable terrain. For this, we model the 3D lidar with an omnidirectional field of view that has cone-shaped blind spots on top and bottom depicted in Fig. 5. We also model the maximum range of the lidar. To test if a 3D point lies within the blind cones, we first obtain the normal vector of the MAV p_{norm} from the Inertial Measurement Unit (IMU)

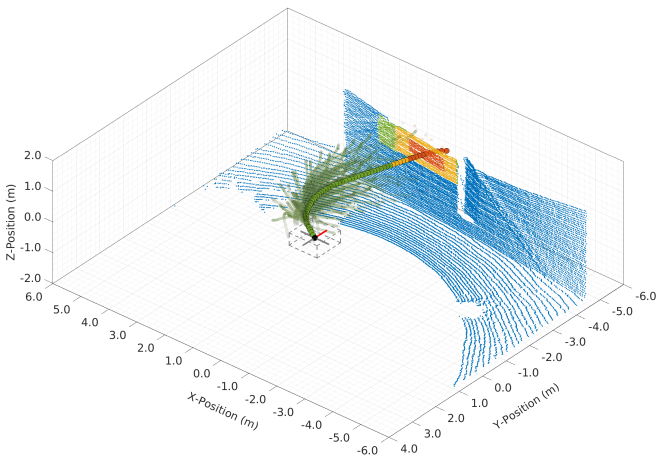

Fig. 8. Alternative safe trajectories that end in a concentric tube around the original trajectory. All 204 trajectories are safe in terms of collisions with the point cloud *and* do not pass unobservable space.

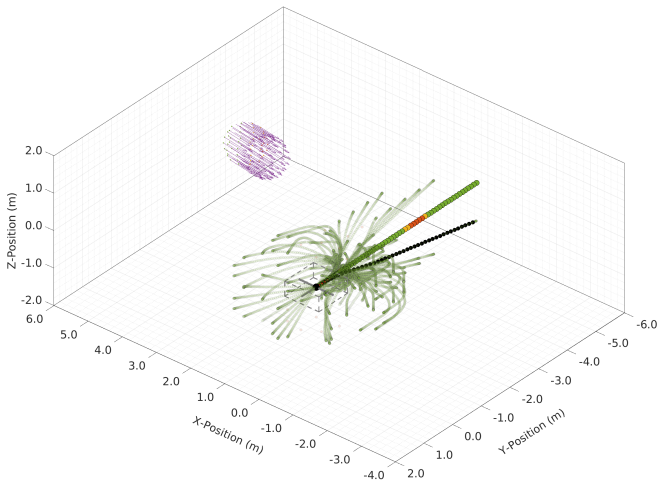

Fig. 9. Avoidance of a simulated obstacle moving with $1.25\,\mathrm{m/s}$. Every point in the point cloud can have an associated velocity. Moving points are predicted forward, so that collisions with the original trajectory as well as potential rescue trajectories can be predicted. Here, the MAV avoids the obstacle thrown into its original trajectory. Our method considers 44 potential rescue trajectories from which the black one is executed.

which points upward during hover.

$$p_{is} = (p_{norm} \cdot p_{test}) \cdot p_{norm} \tag{12}$$

$$l_{cone} = |p_{is}| \cdot \tan(\frac{\pi}{2} - \frac{\theta_{lidar}}{2}) \tag{13}$$

$$l_{test} = |p_{test} - p_{is}| \tag{14}$$

$$l_{test} < l_{cone} \wedge p_{is,z} > 0 \Rightarrow \text{In upper cone} \tag{15}$$

$$l_{test} < l_{cone} \wedge p_{is,z} < 0 \Rightarrow \text{In lower cone} \tag{16}$$

$$l_{test} > l_{lidar} \Rightarrow \text{Out of range} \tag{17}$$

We then compute the point on the normal vector p_{is} that is closest to the tested trajectory point p_{test}, as shown in Eq. 12. With the lidar opening angle θ_{lidar}, we obtain the horizontal extend of the cone at the specific height of p_{is} with Eq. 13. Subsequently, we decide if point p_{test} is inside the unobservable area with Eq. 14–Eq. 17.

Since both cones touch at the origin, the model is very susceptible for small z-values. Therefore, we assume that all points that lie inside of the MAV are always collision-free. The trajectory from Fig. 6 conforms to the highlighted constraints with opening angle $\theta_{lidar} = 33.2°$ and maximum range $l_{lidar} = 120\,\mathrm{m}$.

F. Alternative Trajectories

With a confirmed collision somewhere on the trajectory or a warning that will *not* clear itself on the current trajectory, we generate alternative safe trajectories. If the trajectory is within warning distance at the beginning, but enters a safe state on its own, we classify it as safe. Since our trajectory generation method is very fast ($\ll 1\,\mathrm{ms}$), we can generate multiple alternative trajectories with the lidar frequency of $20\,\mathrm{Hz}$. One can imagine many different metrics that would yield good target points for safe trajectories. We pursue two strategies. Our safe trajectories target a) waypoints on concentric spheroids around the current MAV position, and b) waypoints on concentric tubes around the original trajectory.

Fig. 7 shows a set of alternative trajectories targeting waypoints that lie on concentric spheroids. These time-optimal trajectories bring the MAV to stop as fast as possible without bringing it significantly nearer to the original target.

Alternatively, trajectories targeting a concentric tube around the original trajectory can be safe, while still pursuing the intend of the original trajectory. Fig. 8 shows an example of these trajectories.

After generating all alternative trajectories, we execute the steps from Sec. III-B–Sec. III-E to check if each trajectory is safe. As with the original trajectory, an alternative trajectory is safe if it does not enter a warning (or even collision) zone around any point and does not enter unperceivable space. Leaving a warning zone, however, is permitted, since we want to be able to recover from states where the MAV is already inside a warning zone. From the set of safe alternative trajectories, we select and execute the one with the closest Euclidean distance to the original waypoint.

The number of alternative trajectory candidates determines the total computation time needed for the approach. Thus, we either fix the number of trajectory candidates considered in each replanning step, or adaptively generate alternative trajectories until the replanning time (of in our case $50\,\mathrm{ms}$) is reached.

IV. Evaluation

We evaluate our approach in simulation as well as with a real MAV. To evaluate our approach, we use point clouds recorded during a fully autonomous flight during the Mohamed Bin Zayed International Robotics Challenge (MBZIRC) 2020. Here, our MAV flew into a wall and crashed due to a bug in the coordinate system alignment. During the challenge, we employed the same trajectory generator we present here, but unfortunately, without any obstacle avoidance. We report results from the challenge in [15] and [16]. We previously showed the applicability of our method to MAV flight in, e.g., [12]. The laser scans from the scenarios in Fig. 4, 6, 7, and 8 are extracted from recordings

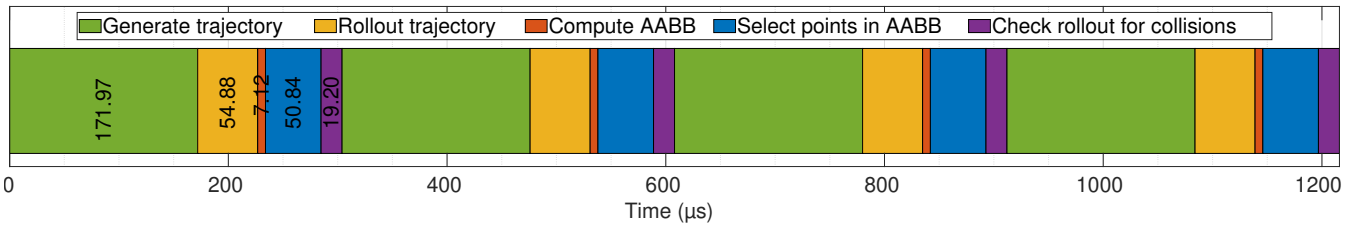

Fig. 10. Approximate computation time of our approach for a scenario with three alternative trajectories. Most time is spent on computing the individual trajectories (green). Due to the computation of the AABB (yellow), collision checking for each rollout (violet) is relatively fast. The 5-step process can be repeated arbitrarily, with approximately $304\,\mu s$ per iteration.

of the challenge. A video showcasing the evaluation can be found on our website[2].

A. Computation Time

We evaluate the computation time with the computer onboard the MAV. Fig. 10 shows the total computation time of our approach for a typical scenario.

It can be seen that most of the time is spent on computing the time-optimal trajectories and that we are able to generate, rollout, and check a total of approximately 164 trajectories within the $50\,\mathrm{ms}$ time frame.

To evaluate the effectiveness of subsetting the point cloud with the AABB, we conducted an ablation study. We removed this specific feature and directly checked for collisions of each rolled out state with the entire point cloud. By comparing the computation times for the typical scenario from Fig. 10, we found that the process takes averagely $1147\,\mu s$ instead of the $77.16\,\mu s$ ($7.12\,\mu s + 50.84\,\mu s + 19.20\,\mu s$) from Fig. 10. This corresponds to a speedup of $1487\,\%$ for the collision checking method and $452\,\%$ speedup in relation to the entire algorithm.

To further speed up our pipeline, we plan to adopt the approach from [17] for collision checking with moving obstacles. Furthermore, the structure of our method offers excellent possibilities for parallelization. Thus, we are currently working on parallelizing our technique with alternative waypoints distributed to individual CPU cores.

B. Moving Obstacles

As described in Sec. III-D, every point in the point cloud can have an associated 3D velocity. To evaluate our approach, we simulate lidar measurements on an object that moves into the trajectory. We show the scenario in Fig. 9. It can be seen that the obstacle crosses the trajectory. Instantaneously, the MAV generates multiple valid rescue trajectories and avoids the obstacle. It decides that it is safe and dynamically feasible to fly in front of the obstacle in the target direction (using the tube trajectories). If the obstacle moved faster, trajectories behind the obstacle would be feasible. If the object were more extensive or additional obstacles would be present, a hard braking maneuver that uses the full dynamic capabilities of the MAV would be performed.

C. Real-World Experiments

We evaluate our method with a dataset that was recorded during the MBZIRC 2020 challenge, where our MAV actually flew into a wall and crashed. We use the recorded lidar and GPS measurements as input to our method and assess if the crash could be hindered by our new obstacle avoidance feature. Fig. 11 shows a still from the world model during the challenge. It can be seen that the MAV successfully avoids a collision by targeting an alternative instead of the commanded waypoint.

The experiments with the dataset show that our method never plans trajectories that lie close to measured obstacles. Instead, it always generates feasible alternative waypoints and corresponding trajectories. This experiment is, however, not sufficient to show that our method works reliably, since it does not close the loop and thus does not execute the planned trajectories.

To evaluate our method in a real-world scenario in a closed-loop, we brought our MAV to a real firefighting exercise. Here, we a) intentionally commanded the MAV to fly into a tree, and b) approached the MAV during hovering to provoke an evasive maneuver. Fig. 12 shows one of the experiments and the corresponding world model. The MAV detects the approaching person and avoids it with a trajectory in the opposing direction. During the entire exercise, no crash with static nor dynamic obstacles happened. The closed-loop avoidance maneuvers, however, were not always smooth since the MAV often switched between alternative waypoints, causing a jittery behavior. We address this issue by favoring alternative waypoints that lie close to previously selected alternative waypoints for the alternative waypoint costs $C_{awp} = \alpha \cdot |AC| + (1 - \alpha) \cdot |AB|$ with $|AC|$ beeing the distance of the alternative waypoint to the commanded waypoint and $|AB|$ beeing the distance from the alternative waypoint to the previously selected alternative waypoint. By computing more alternative trajectories in parallel, we want to increase the waypoint density in the future and thus further reduce jumps between individual alternatives. We also found that the horizontal and vertical dynamics of our MAV differ significantly. Therefore, we had to adjust the parameters of the spheroids during the experiments to be oblate (height $\ll$ diameter) since the MAV strongly preferred vertical movement.

V. Conclusion

We have provided detailed insight into our robust trajectory generation framework. The viability of our approach has been demonstrated in simulation as well as in a real-world scenario during a real firefighting exercise.

In particular, the ability to analytically crop large point clouds, makes our method scale to larger and denser point

[2] www.ais.uni-bonn.de/videos/ssrr_2020_beul

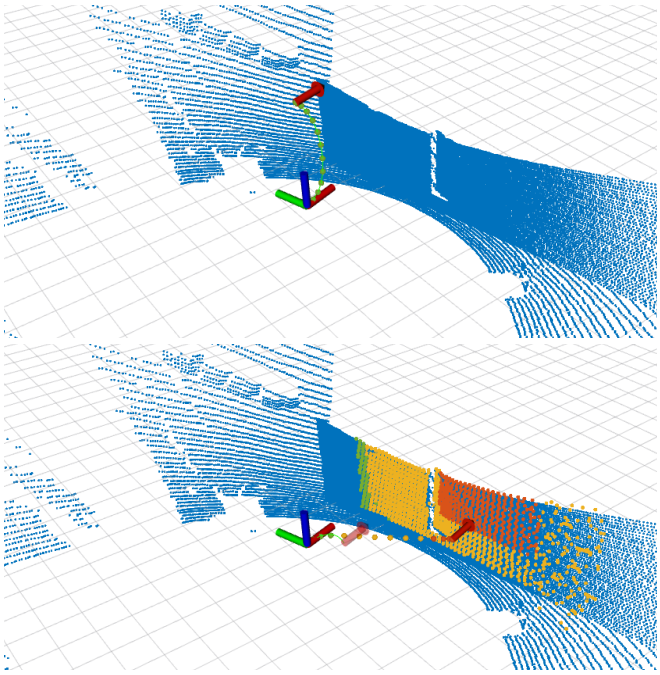

Fig. 11. Evaluation with the MBZIRC 2020 dataset. Top: The waypoint (red arrow) can be reached by the MAV (axis) with the green trajectory without colliding with the measured point cloud (blue). Bottom: If the waypoint lies close to the perceived wall, an alternative waypoint (transparent red) is computed that can be reached without interfering.

clouds in the future. Due to the fast runtime, our method can run in real-time as MPC onboard the MAV, and it can react to obstacles, perceived with the lidar rate of 20 Hz. We showed that the ability to reject unskilled control inputs could help prevent crashes and thus increase the reliability of flying robots.

We believe that our contribution will make the operation of MAVs during rescue missions safer and more reliable.

References

[1] G. J. M. Kruijff, I. Kruijff-Korbayová, S. Keshavdas, B. Larochelle, M. Janíček, F. Colas, M. Liu, F. Pomerleau, R. Siegwart, M. A. Neerincx, R. Looije, N. J. J. M. Smets, T. Mioch, J. van Diggelen, F. Pirri, M. Gianni, F. Ferri, M. Menna, R. Worst, T. Linder, V. Tretyakov, H. Surmann, T. Svoboda, M. Reinstein, K. Zimmermann, T. Petříček, and V. Hlaváč, "Designing, developing, and deploying systems to support human-robot teams in disaster response," *Advanced Robotics*, vol. 28, no. 23, pp. 1547–1570, 2014.

[2] M. Beul and S. Behnke, "Fast full state trajectory generation for multirotors," in *Proceedings of the International Conference on Unmanned Aircraft Systems (ICUAS)*, 2017.

[3] J. Zhang, R. G. Chadha, V. Velivela, and S. Singh, "P-CAP: Precomputed alternative paths to enable aggressive aerial maneuvers in cluttered environments," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[4] A. J. Barry, P. R. Florence, and R. Tedrake, "High-speed autonomous obstacle avoidance with pushbroom stereo," *Journal of Field Robotics (JFR)*, vol. 35, no. 1, pp. 52–68, 2018.

[5] I. Ulrich and J. Borenstein, "VFH+: Reliable obstacle avoidance for fast mobile robots," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1998.

[6] D. Falanga, K. Kleber, and D. Scaramuzza, "Dynamic obstacle avoidance for quadrotors with event cameras," *Science Robotics*, vol. 5, no. 40, 2020.

[7] M. Nieuwenhuisen, M. Schadler, and S. Behnke, "Predictive potential field-based collision avoidance for mmulticopters," *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences (ISPRS Archives)*, vol. W2, no. XL-1, pp. 293–298, 2013.

Fig. 12. Avoidance of an approaching person. Our MAV (axis) senses the threat (yellow points) and plans an avoidance trajectory (green).

[8] M. W. Mueller, M. Hehn, and R. D'Andrea, "A computationally efficient motion primitive for quadrocopter trajectory generation," *IEEE Transactions on Robotics*, vol. 31, no. 6, pp. 1294–1310, 2015.

[9] B. Lindqvist, S. S. Mansouri, A. A. Agha-mohammadi, and G. Nikolakopoulos, "Nonlinear MPC for collision avoidance and control of UAVs with dynamic obstacles," *IEEE Robotics and Automation Letters (RA-L)*, vol. 5, no. 4, 2020.

[10] B. T. Lopez and J. P. How, "Aggressive 3-D collision avoidance for high-speed navigation," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[11] M. Watterson and V. Kumar, "Safe receding horizon control for aggressive mav flight with limited range sensing," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015.

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

[13] M. Beul and S. Behnke, "Analytical time-optimal trajectory generation and control for multirotors," in *Proceedings of the International Conference on Unmanned Aircraft Systems (ICUAS)*, 2016.

[14] ——, "Fast time-optimal avoidance of moving obstacles for high-speed MAV flight," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019.

[15] M. Schwarz, M. Beul, J. Quenzel, S. Bultmann, P. Lowin, D. Pavlichenko, A. Rochow, R. A. Rosu, B. Scheider, D. Schleich, M. Schreiber, M. Splietker, F. Süberkrüb, and S. Behnke, "Team NimbRo at MBZIRC 2020: Heterogeneous MAVs for solving challenging tasks," *Under Review at Journal of Field Robotics (JFR)*.

[16] M. Beul, S. Bultmann, A. Rochow, R. A. Rosu, D. Schleich, M. Splietker, and S. Behnke, "Visually guided balloon popping with an autonomous MAV at MBZIRC 2020," in *Proceedings of the IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, 2020.

[17] J. Bialkowski, M. Otte, S. Karaman, and E. Frazzoli, "Efficient collision checking in sampling-based motion planning via safety certificates," *The International Journal of Robotics Research (IJRR)*, vol. 35, no. 7, pp. 767–796, 2016.