A video presentation of this paper is available at https://www.youtube.com/watch?v=_c3kah6yfJA

Autonomous flight through cluttered outdoor environments using a memoryless planner

Junseok Lee[1†], Xiangyu Wu[1†], Seung Jae Lee[2], and Mark W. Mueller[1]

***Abstract*— This paper introduces a collision avoidance system for navigating a multicopter in cluttered outdoor environments based on the recent memory-less motion planner, rectangular pyramid partitioning using integrated depth sensors (RAPPIDS). The RAPPIDS motion planner generates collision-free flight trajectories at high speed with low computational cost using only the latest depth image. In this work we extend it to improve the performance of the planner by taking the following issues into account. (a) Changes in the dynamic characteristics of the multicopter that occur during flight, such as changes in motor input/output characteristics due to battery voltage drop. (b) The noise of the flight sensor, which can cause unwanted control input components. (c) Planner utility function which may not be suitable for the cluttered environment. Therefore, in this paper we introduce solutions to each of the above problems and propose a system for the successful operation of the RAPPIDS planner in an outdoor cluttered flight environment. At the end of the paper, we validate the proposed method's effectiveness by presenting the flight experiment results in a forest environment. A video can be found at `www.youtube.com/channel/UCK-gErmvZlBODN5gQpNcpsg`**

Fig. 1. The vehicle flies autonomously using visual-inertial odometry and the collision-avoidance planner through a forest, avoiding trees.

I. INTRODUCTION

Motion planning algorithms for multicopter unmanned aerial vehicles to fly autonomously to their destination in cluttered environments are in general grouped into two categories. One is to separately run a path planning algorithm to generate collision-free path as a purely geometrical problem without considering dynamics [1], and use a path follower to follow the collision-free path. Since it does not consider dynamics constraints, collision avoidance can not be guaranteed at high speed since dynamics constraints, such as motor thrust limit, are not considered at the time of planning. The other approach considers dynamics constraints and directly generates control commands by including obstacle avoidance as a constraint inside an optimization problem rather than separating path planning and tracking, for example, based on the rapidly-exploring random tree star (RRT*) [2], the nonlinear optimization [3], and the mixed-integer programming (MIP) [4].

We focus on the latter category which considers dynamics in planning as well as collision avoidance. The collision-free trajectories are often further constrained by minimizing a cost function depending on applications, for example, the minimum time, the minimum energy, and the shortest distance. Trajectory generation algorithms may be divided into two major types: map-based algorithms and memory-less algorithms [5].

Map-based algorithms are global planning methods of creating collision-free optimal trajectories after a single large map is constructed or while creating a map by fusing all spatial sensor information obtained during flights. For instance, in [6], a local map map of the environment is used and a nonconvex, nonlinear optimization problem is solved to get collision-free smooth trajectories. In [7]–[9] the free-space in the map is represented as multiple convex regions, and an optimization problem is then solved to find a series of trajectories through the free-space. In [10], the convex hull property of B-spline trajectories is used to solve for safe and fast trajectories, and the success rate and optimality is improved in the subsequent work of [11]. Map-based algorithms have the advantage of optimal trajectory generation because it presupposes that map information is already known when planning. However, they usually have high computational cost and due to fusing sensor data into the map.

On the other hand, memory-less algorithms use only the most recent sensor information to avoid obstacles, such as using the k-d-tree [12], [13], and a trajectory library precomputed offline to reduce a significant amount of online computation [14]. Therefore, memory-less algorithms are classified as local planning methods and are advantageous

†Junseok Lee and Xiangyu Wu contributed equally to this article. Names are in alphabetical order.

[1]Junseok Lee, Xiangyu Wu, and Mark W. Mueller are with the High Performance Robotics Lab (HiPeRLab) at the Department of Mechanical Engineering, UC Berkeley, CA 94720, USA {junseok_lee,wuxiangyu,mwm}@berkeley.edu

[2]Seung Jae Lee is with the Automation and Systems Research Institute (ASRI), Seoul National University, Seoul, Republic of Korea sjlazza@snu.ac.kr

for obstacle avoidance in a dynamic obstacle environment due to low computational cost and high update frequency. However, there is a disadvantage that it is challenging to find a globally optimal trajectory since global spatial information is not available at the time of planning.

For trajectory generation of a small-size multicopter that requires high-speed maneuver but has a limited payload capacity, memory-less algorithms have great utility due to the following reasons. First, memory-less algorithms are easy to be implemented in real-time on miniature on-board computers with limited computational resources. Second, due to the fast trajectory update speed thanks to the low computational cost, memory-less algorithms can cope with rapidly changing surroundings during high-speed flight. Lastly, the algorithm is less prone to accumulated odometry errors during flights in a cluttered environment because it utilizes only the latest sensor information for the planning.

Recently, a memory-less planner is proposed using rectangular pyramid partitioning using integrated depth sensors (RAPPIDS) motion planner, which has high computational efficiency for high-speed collision avoidance flights [15]. Using the RAPPIDS motion planner, the authors were able to achieve high collision avoidance flight performance in cluttered indoor flight environment, using only stand-alone depth images and visual-inertial odometry information processed by an on-board computer mounted on the vehicle.

In this paper, we extend the RAPPIDS planner to operate in a cluttered outdoor off-road environment. We describe the system's development process and flight results. In the experiment, the system was able to fly 30 meters in a forest environment, as shown in Fig. 1, with a maximum speed of 2.7 m/s. The remaining parts of the paper are organized as follows: in Section II, we outline the principles of the RAPPIDS planner. Section III introduces modifications to the algorithm to ensure successful outdoor flights. Finally, in Section IV, the system's performance is demonstrated by introducing the experiment results of obstacle avoidance flights in a forest environment.

II. RAPPIDS MOTION PLANNING FRAMEWORK

In this section, we repeat some details from [15], and add a velocity limiting check for stable visual-inertial odometry. Motion primitives are sampled and then go through a series of checks to find a trajectory that is minimum-cost, input-feasible, velocity-admissible, and collision-free, as shown in Algorithm 1. Since the planner has a low computational cost, we plan in a receding horizon fashion every time a new depth image arrives. This keeps a collision-free trajectory being updated with the latest depth view, and allows avoiding obstacles that are not included in the previous camera view.

A. Candidate trajectory sampling

We sample candidate trajectories first by sampling an endpoint and constructing a polynomial trajectory connecting the current position to the endpoint. Specifically, we first uniformly sample a 2D point in the pixel coordinates of a depth camera. We also draw a sampled depth from a uniform distribution, and then back-project the 2D point using the sampled depth to obtain a sampled endpoint $\mathbf{s}_T \in \mathbb{R}^3$.

Algorithm 1 Trajectory Constraint Checks

Require: A sampled candidate trajectory
procedure CONSTRAINTSCHECK
  if lower cost than known **then**
    if input feasibility **then**
      if velocity admissibility **then** ▷ Subsection II-B
        if collision-free **then** ▷ Subsection II-C
          trajectory_status ← collision_free
        else
          trajectory_status ← in_collision
        end if
      else
        trajectory_status ← velocity_inadmissible
      end if
    else
      trajectory_status ← input_infeasible
    end if
  else
    trajectory_status ← higher_cost
  end if
end procedure

Denote $\mathbf{s}(t)$, $\dot{\mathbf{s}}(t)$, and $\ddot{\mathbf{s}}(t) \in \mathbb{R}^3$ to be the position, velocity and acceleration of the vehicle in the inertial frame. The candidate motion primitives are described as below.

$$\mathbf{s}(t) = \frac{\alpha}{120}t^5 + \frac{\beta}{24}t^4 + \frac{\gamma}{6}t^3 + \frac{\ddot{\mathbf{s}}(0)}{2}t^2 + \dot{\mathbf{s}}(0)t + \mathbf{s}(0),\ t \in [0\ T], \tag{1}$$

where T is the trajectory duration, $\mathbf{s}(0)$, $\dot{\mathbf{s}}(0)$, and $\ddot{\mathbf{s}}(0)$ are the initial position, velocity, and acceleration of the vehicle at the time when the trajectory starts, and α, β and γ are coefficients such that $\mathbf{s}(T) = \mathbf{s}_T$, and $\dot{\mathbf{s}}(T) = \ddot{\mathbf{s}}(T) = 0$. This 5th order polynomial corresponds to the minimum-jerk trajectory, which minimizes the average Euclidean norm of jerk over the trajectory duration T. The trajectory is smooth and can be checked for collisions efficiently, as shown in [16].

B. Velocity constraints for stable visual-inertial odometry

We impose velocity constraints to prevent the visual-inertial odometry from losing track in high-speed flights. The sampled trajectories are filtered out if their maximum velocity exceeds a predefined threshold, $v_{\max}$. For the sake of computational tractability, the constraints are checked for each per-axis velocity using analytical solution for third-order polynomials. It should be noted that checking the magnitude requires solving higher-order polynomials numerically, because analytical solutions do not exist. The following equation can be derived by taking the derivative of (1),

$$\dot{\mathbf{s}}(t) = \frac{\alpha}{24}t^4 + \frac{\beta}{6}t^3 + \frac{\gamma}{2}t^2 + \ddot{\mathbf{s}}(0)t + \dot{\mathbf{s}}(0) \tag{2}$$

To find its extrema, we compute its derivative and find the zeros as below.

$$\ddot{\mathbf{s}}(t) = \frac{\alpha}{6}t^3 + \frac{\beta}{2}t^2 + \gamma t + \ddot{\mathbf{s}}(0) = 0 \tag{3}$$

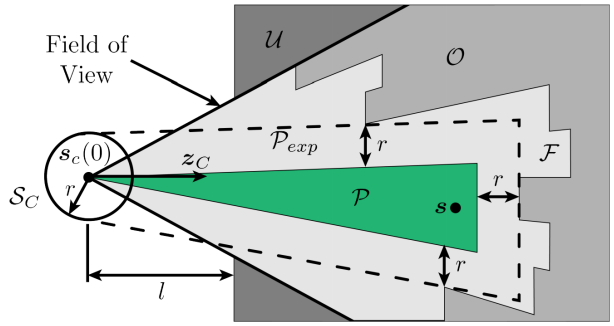

Fig. 2. The planner uses a collection of pyramids to represent the free space, which allows simple and fast collision check of a sampled trajectory. *The figure is sourced from [15].*

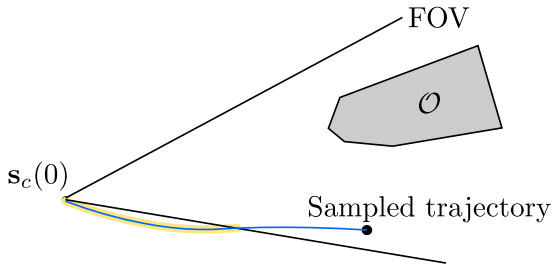

Fig. 3. Efficient trajectory sampling in the field of view (FOV) of the depth camera. An occluded space $\mathcal{O}$ at the center makes the planner construct pyramids around the FOV. A sampled trajectory (blue) with an endpoint close to the FOV is likely to have a part that resides outside the field of view (highlighted by yellow), which is classified by the planner as in-collision as space outside of the FOV is considered as occupied. To increase the sampling efficiency, we sample points only between 10% and 90% of horizontal and vertical FOV.

The third-order polynomial can be efficiently solved, and the magnitude of (2) is evaluated at the roots as well as the boundary, 0 and T. The procedure is repeated for every axis, and we discard the motion primitive candidate if the speed on any axis exceeds the per-axis velocity limit $v_{\max}$.

C. Collision check: Pyramid method

The RAPPIDS planner determines whether the candidate trajectory intrudes the obstacle by pyramid inflation. Fig. 2 shows the process in which the depth camera on the flying vehicle searches for area $\mathscr{P}$ that guarantees a non-collision path. First, we define free space $\mathscr{F}$ and occupied space $\mathscr{O}$ based on the depth camera image. We also treat all spaces outside the field of view that are l distance from the vehicle as occupied spaces to avoid collisions with unrecognized obstacles outside the camera's field of view. Next, we select the final position $\mathbf{s}(T)$ through random sampling and then search for the nearest depth pixel p from $\mathbf{s}(T)$. Then, starting at pixel p and reading the surrounding depth pixels in a spiral sequence, we get the largest possible rectangular space $\mathscr{P}_{exp}$ that does not intrude the occupied space $\mathscr{O}$. Finally, pyramid $\mathscr{P}$ distanced with vehicle radius r is created by shrinking the expanded pyramid $\mathscr{P}_{exp}$. By checking whether the $\mathbf{s}(t)$ candidate trajectory remaining inside $\mathscr{P}$, we can conclude that the trajectory is collision-free from the detected obstacles.

The algorithm can guarantee zero collision trajectory, because the trajectory generated by the algorithm inherently avoids not only the obstacles recognized by the depth sensor but also the obstacles located in an unobserved ($\mathscr{U}$) or unknown area ($\mathscr{O}$) obscured by the detected obstacles.

III. ALGORITHMS FOR OUTDOOR FLIGHT

In this section, we describe modifications to the motion planner other than the velocity check described in section II-B, for collision-free flights outdoors to autonomously reach a target waypoint. (a) We sample trajectories with final positions around the center of the view of the depth camera to improve the efficiency of sampling trajectories. (b) We proposed a new utility function that behaves similarly to the utility function of maximizing the average velocity, but also considers making the vehicle stay around the target. (c) The vehicle is always yawed towards the goal to dynamically change the view during the flight that can potentially increase the chance of finding a collision-free trajectory compare to the view with a fixed yaw. (d) The initial acceleration used for sampling trajectories is approximated by the total thrust command divided by the mass instead of using noisy IMU measurements, since the noise in acceleration hampers the planner from finding proper trajectories. (e) We compensate the thrust change because of battery voltage drop during the flight.

A. Sampling efficiency

As collision checking at the last step of the planner is computationally expensive, it is beneficial to increase the probability of finding a collision-free trajectory from candidate trajectories. We improve the sampling efficiency by excluding candidate trajectories with high chances of being classified as in collision. One of the most common cases is when a sampled candidate trajectory has an endpoint around the field of view (FOV), as shown in Fig. 3. If the endpoint is close to the edges of the FOV, it is likely that some parts of the sampled trajectory fall outside of the FOV, and it is classified by the planner as in-collision because regions outside of the FOV are considered occupied by the planner, as described in section II-C. To improve the trajectory sampling efficiency by avoiding those cases, the final trajectory position is sampled between 10% and 90% of the field-of-view.

B. Utility function

We propose a utility function as below to generate trajectories that not only consider maximizing the average velocity to the goal, but also keep the trajectories' end points around the target.

$$U(P,t) = \frac{\|\mathbf{d}_{s_tG}\| - \|\mathbf{d}_{PG}\|}{t}, \tag{4}$$

where $\|\mathbf{d}_{s_tG}\|$, $\|\mathbf{d}_{PG}\|$, and t are the distances between the current position and the goal, the distance between the endpoint of the motion primitives and the goal, and the primitive execution time, respectively. As shown in Fig. 4,

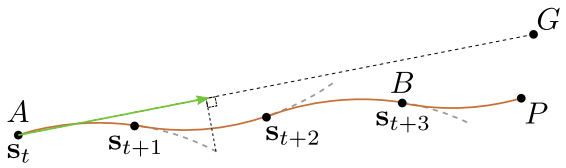

Fig. 4. A new cost function to maximize average velocity and handle the behavior around the goal point G is proposed. Every time a new depth image arrives, the planner attempts to find a new collision-free trajectory using the current estimates s_t, and if found, the tracking controller discards the previous trajectory (gray-dashed parts), and starts tracking the new trajectory (brown). When far from the goal point, for example point A, the utility function is roughly the average velocity measured in direction to the goal (depicted by the green arrow), which still allows lateral motion. However, around the goal, such as point B, the utility function encourages the planner to generate a new collision-free trajectory with the endpoint around the goal, not beyond the goal.

when the vehicle is far from the waypoint, the term $\|\mathbf{d}_{PG}\|$ does not vary much, and hence the utility function is to maximize the average velocity to the waypoint. Around the waypoint, however, the term $\|\mathbf{d}_{PG}\|$ plays a role to encourage the planner to choose a trajectory whose final position falls around the goal.

C. Desired yaw angle

The yaw angle can be arbitrarily chosen while tracking a collision-free trajectory. We yaw the vehicle always towards the goal, because it is more likely to find a trajectory to the goal when facing towards it.

$$\begin{aligned}\psi_c &= \arctan2\left([0\ 1\ 0]\,(\mathbf{s}_G-\mathbf{s}),\ [1\ 0\ 0]\,(\mathbf{s}_G-\mathbf{s})\right) \\ &\in [-\pi,\pi],\end{aligned} \tag{5}$$

where $\arctan2(y,x)$ measures the signed angle between the point (x,y) and the positive x-axis, and ψ_c, $\mathbf{s}_G$ and $\mathbf{s}$ are the commanded yaw angle, the positions of the goal and the vehicle in the inertial frame, respectively.

D. Acceleration estimation

Our planner checks collision of a trajectory that is sampled given the current position, velocity, and acceleration (section II-A). Using acceleration measurements from IMU for the current acceleration results in inaccurate sampled trajectories due to the noise in IMU acceleration measurements. We instead estimate acceleration using the last commanded thrust as below

$$\ddot{\mathbf{s}}(0) = \frac{c\mathbf{z}_B}{m} - \mathbf{g}, \tag{6}$$

where c, $\mathbf{z}_B$, m, and $\mathbf{g}$ are the last collective thrust command, body z-axis, mass, and gravitational acceleration.

E. Thrust adaptation

Once the collision-free trajectory is selected, the cascaded flight controller generates motor speed commands to maneuver the vehicle, as shown in Fig. 9. The commands are then sent to the electric speed controllers (ESCs), where ESCs control each motor's rotation speed based on the predefined protocol in their firmware‘ .

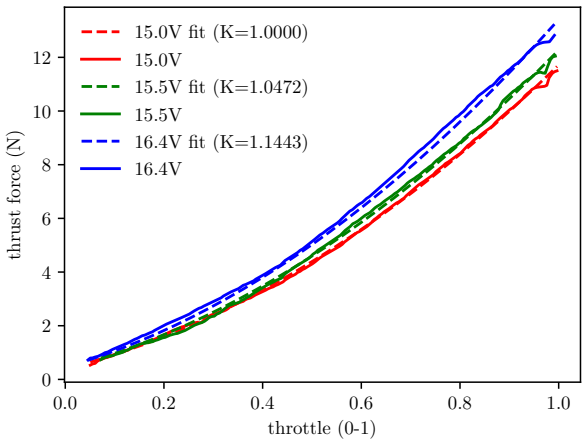

Fig. 5. Thrust force at different thrust commands (solid lines) and second-order fitting results (dotted lines). The command-thrust relationship changes depending on the voltage applied to the propulsion system.

ESCs can be classified into two groups: open-loop ESCs and closed-loop ESCs. Unlike closed-loop ESCs that can accurately control motor rotation speed through a feedback control loop, open-loop ESCs used in most multicopters control motors to rotate at different speed if the battery's voltage changes, generating different thrust, as shown in Fig. 5. Therefore, to generate the desired thrust force regardless of the voltage level, an on-line thrust model reflecting the current voltage level is required.

1) On-line thrust model: from the data and fitting results in Fig. 5, we are able to confirm that the thrust model at each voltage level could be fitted as using the following quadratic function form

$$f_i(u_i) = K(c_0(u_i+c_1)^2+c_2), \tag{7}$$

where K is the voltage-dependent term, u_i is the thrust command of the i-th motor, and $c_{\{0,1,2\}}$ are the nominal model parameters at a voltage level where K is 1. However, the model includes a fitting error and we additionally structure K as follows to overcome the error

$$K = k_V k_M, \tag{8}$$

where k_V is the modeled part and k_M is the unmodeled part. The update rules for k_V and k_M are introduced in the following parts.

2) Updating k_V gain: fig. 6 shows the hovering thrust throttle command that increases as the voltage level decreases. From the data, we can derive the affine function $h(V)$ through a first-order fitting. Then, we compensate for the thrust reduction due to the voltage drop by defining k_V as follows

$$k_V = \frac{h(V_R)}{h(V)}, \tag{9}$$

where V_R is a reference voltage for which the $c_{\{0,1,2\}}$ values in (7) are defined.

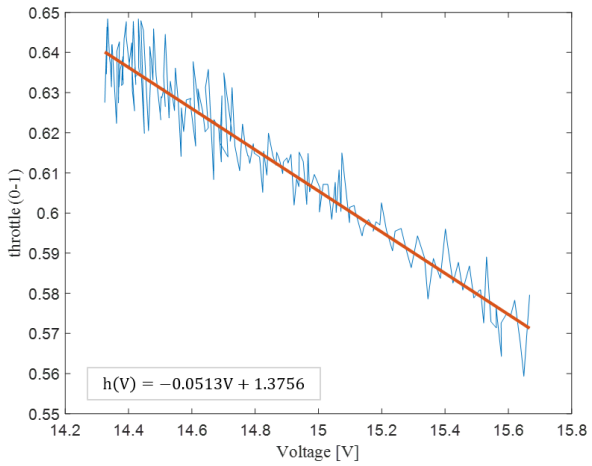

Fig. 6. Hovering command that increases as the voltage decreases. With first-order fitting, we can obtain $h(V)$ function.

3) Updating k_M gain: to compensate for unmodeled errors, we compare the target thrust and the estimated thrust. The magnitude of actual thrust during flight can be estimated from the acceleration value in the z-direction of the IMU.

First, the translational motion dynamics of a multicopter is described as follows

$$m\ddot{\mathbf{s}} = R(\mathbf{q})\mathbf{f} + m\mathbf{g}, \tag{10}$$

where $\mathbf{q} \in \mathbb{R}^3$ is the attitude, $R(\mathbf{q}) \in SO(3)$ is the rotation matrix that transforms from the body coordinate frame to the world coordinate frame, and $\mathbf{f} = [0\ 0\ \Sigma f_i]^T \in \mathbb{R}^3$ is the thrust vector. Next, the following relationship is established between the IMU acceleration measurement and the actual acceleration

$$R(\mathbf{q})\mathbf{c} + \mathbf{g} + \Delta_s = \ddot{\mathbf{s}}, \tag{11}$$

where $\mathbf{c} = [c_x\ c_y\ c_z]^T \in \mathbb{R}^3$ is the IMU's acceleration measurement and Δ_s refers to negligible microterms that occur due to the rotational motion [17].

Through (10) and (11), we can bring the following result

$$mc_z \approx \Sigma f_i, \tag{12}$$

where we can estimate the actual thrust force generated from the vehicle with z-directional IMU acceleration measurements. Then, we can estimate k_M in real-time as follows

$$k_M = 1 + \int \left(f_i(u_i) - \hat{f}_i\right)dt,\ \ \hat{f}_i = \frac{mc_z}{n}, \tag{13}$$

where n represents the number of thrusters attached to the multicopter.

By updating k_V and k_M gains on-line during the flight through (9) and (13), we can calculate control input u_i from the inverse of (7) to make each propeller's desired thrust more accurate.

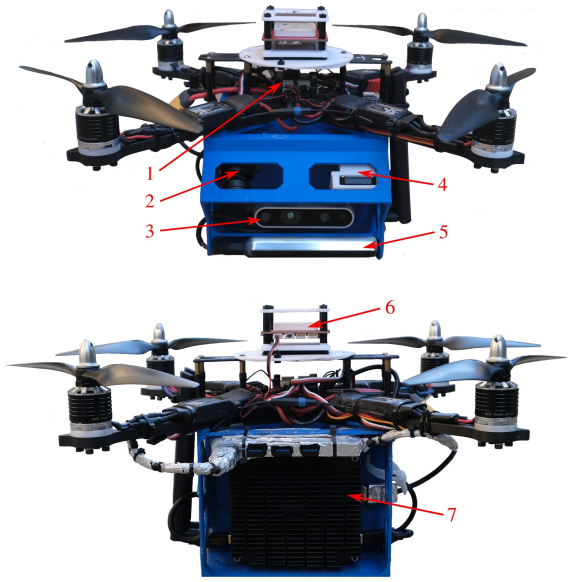

Fig. 7. The custom built quadcopter. 1 - Pixracer flight controller; 2 - RGB camera (not used in feedback); 3 - D435i depth camera; 4 - infra-red camera (not used in feedback); 5 - T265 tracking camera; 6 - GPS (not used in feedback); 7 - Jetson AGX Xavier

Fig. 8. Satellite image of the small forest at the Richmond Field Station where the experiment was conducted. Image is from `www.usgs.gov` .

IV. EXPERIMENTAL RESULTS

We have shown in outdoor experiments that the system is able to navigate in a complex forest experiment with a maximum speed of 2.7 m/s. The experiment was repeated several times and we got similar performance. In this section we give detailed explanation of one of these experiments.

A. System setup

A custom-built quadcopter, as shown in Fig. 7, was used through the experiments. It weighs 2.4 kg and the distance between two diagonal motors is 382 mm. The diameter of each propeller is 229 mm. On the vehicle, an Intel D435i depth camera is installed for collision avoidance and an Intel T265 camera is installed for state estimation. The depth camera is forward-looking to detect obstacles in forward flights, and the T265 camera has a 37-degree angle with

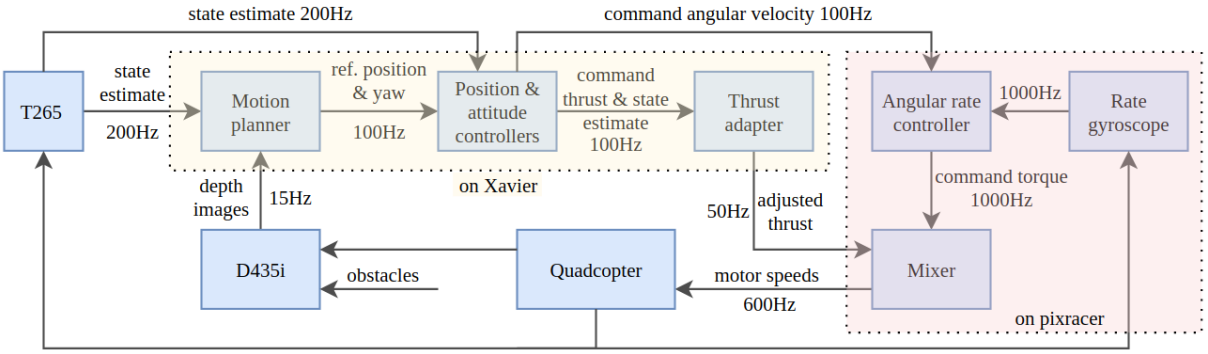

Fig. 9. A block diagram of the system, showing the relationship between components. The yellow shaded area contains components running on the AGX Xavier on-board computer, while the red shaded area contains components running on the Pixracer flight controller.

the horizontal ground, to track features on the ground for state estimation and to avoid view occlusion from other parts of the vehicle. The GPS and two other cameras (one is a RGB camera and the other one is an infra-red camera) on the vehicle are not used for the vehicle's motion planning. The relationship between the components of the system, is shown in Fig. 9. The RAPPIDS motion planner, the position and attitude controllers, and the thrust adapter runs on an on-board computer (Jetson AGX Xavier), at a frequency of 100Hz. The command thrust and angular velocity for each axis are then sent to the Pixracer flight controller via serial communication. The Pixracer runs the standard PX4 firmware and is used for the low-level control of the vehicle. The position controller is a PD controller, while the attitude and angular velocity controllers are P controllers.

B. Obstacle avoidance experiment

The experiments were conducted at a small forest at the Richmond Field Station (37.915535 N, -122.335059 E), as shown in Fig. 8. The vehicle's radius r (shown in Fig. 2) was set to 0.6m for the planner during the experiment, leaving a minimum safety margin of about 0.3 m between the vehicle and the nearest obstacles. The velocity constraint v_{max} in section II-B was set to 3 m/s because of the limit of the T265 tracking camera (a speed of above 3 m/s could make the state estimation of T265 unreliable), and trajectories exceeding this speed limit will be rejected.

The vehicle was first controlled to take off manually to about 1 m above the ground and then switched to autonomous hovering at its current position, which was used as the starting point. The target point was set to be 30 meters forward with respect to the starting point of the vehicle, to fly the vehicle to the other side of the forest. When the vehicle was close to the target point (less than 1 m in this case), the motion planner in Fig. 9 stoped generating new trajectories, and the vehicle tracked the last reference trajectory to reach the target point. After the vehicle reached the target point, it hovered there and waited for the pilot to send other commands, e.g. landing.

In the experiment the vehicle was able to reach the target point while generating and tracking collision-free trajectories. The path of the vehicle is visualized in Fig. 10 and

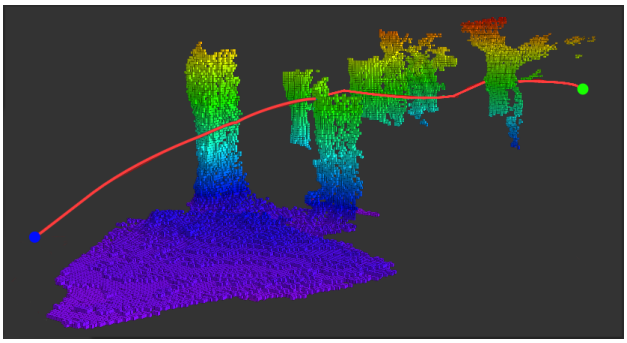

Fig. 10. Path of the vehicle (red line) during the autonomous collision avoidance flight from the start point (marked with a blue dot) to the end point (marked with a green dot). The trees detected by the depth camera were visualized using Octomap [18]. The distance between the start point and the end point is 30 meters.

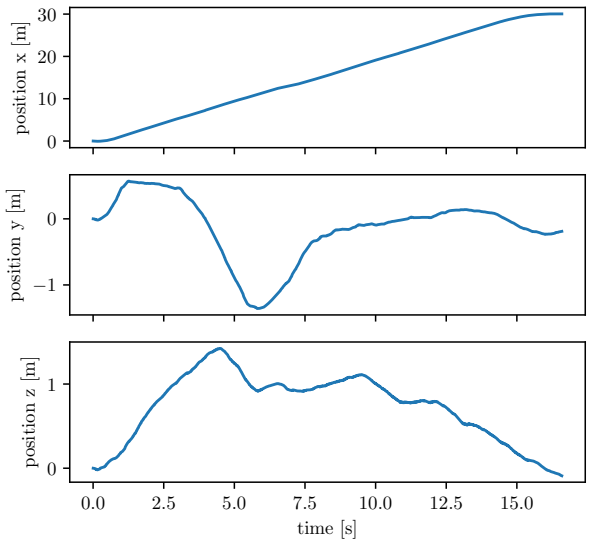

Fig. 11. The estimated position of the vehicle during the autonomous collision avoidance flight, with respect to the starting point of the autonomous flight. The target point of the vehicle was at x = 30 m, y = 0 m, z = 0 m.

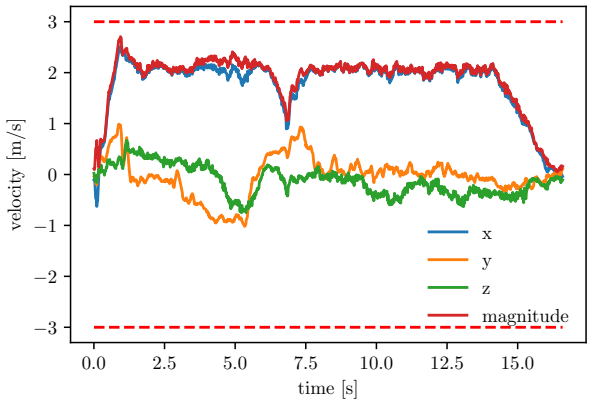

Fig. 12. The estimated velocity for each axis of the vehicle, as well as the Euclidean norm (magnitude) of the velocity during the autonomous collision avoidance flight. With the velocity check in section II-B, the velocity on none of the three axis exceeds the velocity limit of 3.0 m/s (marked as red dashed lines).

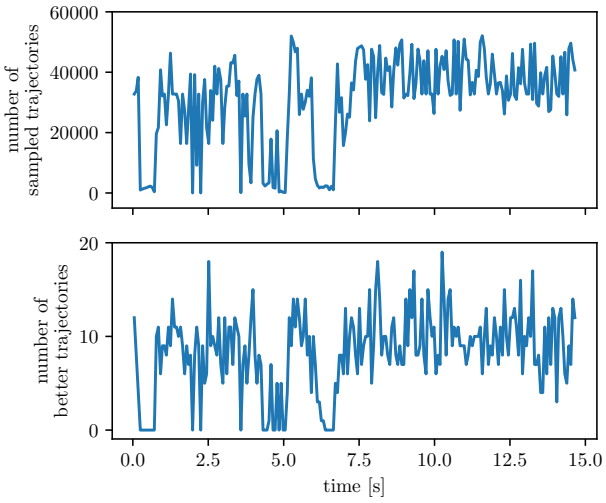

Fig. 13. A large number of trajectories were sampled during the autonomous flight of the vehicle, as shown in the first row. The sampled trajectories then went through the checks in Algorithm 1, and the number of trajectories that are better than the current trajectory is shown in the second row.

its position is shown in Fig. 11. The manual take-off and landing part are omitted and only the autonomous collision avoidance flight part is plotted for clarity. With the velocity check in section II-B on the sampled trajectories, the velocity on none of the three axis exceeds the velocity limit of 3.0 m/s, shown in Fig. 12.

The number of sampled trajectories and better-than-current trajectories (i.e. trajectories that pass all the checks in Algorithm 1 and have a lower cost than the current reference trajectory) throughout the experiment is shown in Fig. 13. Thanks to the computational efficiency of the algorithm, a large number of sampled trajectories could be processed on-board in real-time. The current reference trajectory was updated when a better-than-current trajectory was found, which happened most of the time during the flight. When no better trajectory was found (e.g. when the view of the depth camera was occluded by the obstacles), the vehicle would track the current trajectory. After 14.7 s, the vehicle was within 1 m to the target point, and the trajectory generator stopped generating new trajectories and followed the last reference trajectory to reach the target point.

V. CONCLUSIONS

In this paper, we presented various considerations of the RAPPIDS motion planner for outdoor flights. We introduced the velocity constraint of the planner to satisfy the speed limit of the visual-inertial odometry camera, increased the trajectory sampling efficiency based on the prior that sampling close to the edge of field of view of the depth camera is prone to result in in-collision trajectories, and used estimated acceleration instead of noisy IMU acceleration measurements. In addition, a new utility function was proposed to consider not only maximizing the average velocity toward the goal but also keeping the vehicle around the goal. A thrust adaptation method is introduced to compensate decrease in motor thrusts due to voltage drop during flights. Lastly, the experimental results in a challenging outdoor environment were presented, which validated the ability of this system to autonomously navigate through complex obstacles outdoors.

ACKNOWLEDGEMENT

Research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-20-2-0105. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein. The experimental testbed at the HiPeRLab is the result of contributions of many people, a full list of which can be found at `hiperlab.berkeley.edu/members/`.

REFERENCES

[1] T. Baumann, "Obstacle Avoidance for Drones Using a 3DVFH* Algorithm," Master thesis, ETH Zurich, 2015.

[2] B. Sakcak, L. Bascetta, G. Ferretti, and M. Prandini, "Sampling-based optimal kinodynamic planning with motion primitives," *Autonomous Robots*, vol. 43, no. 7, pp. 1715–1732, Oct. 2019.

[3] S. Spedicato and G. Notarstefano, "Minimum-Time Trajectory Generation for Quadrotors in Constrained Environments," *IEEE Transactions on Control Systems Technology*, vol. 26, no. 4, pp. 1335–1344, Jul. 2018.

[4] J. Park, S. Karumanchi, and K. Iagnemma, "Homotopy-Based Divide-and-Conquer Strategy for Optimal Trajectory Planning via Mixed-Integer Programming," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1101–1115, Oct. 2015.

[5] Y. Lu, Z. Xue, G.-S. Xia, and L. Zhang, "A survey on vision-based UAV navigation," *Geo-spatial Information Science*, vol. 21, no. 1, pp. 21–32, 2018.

[6] H. Oleynikova, M. Burri, Z. Taylor, J. Nieto, R. Siegwart, and E. Galceran, "Continuous-time trajectory optimization for online uav replanning," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 5332–5339.

[7] Sikang Liu, M. Watterson, S. Tang, and V. Kumar, "High speed navigation for quadrotors with limited onboard sensing," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1484–1491.

[8] Jing Chen, Tianbo Liu, and Shaojie Shen, "Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1476–1483.

[9] J. Tordesillas, B. T. Lopez, and J. P. How, "Faster: Fast and safe trajectory planner for flights in unknown environments," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 1934–1940.

[10] B. Zhou, F. Gao, L. Wang, C. Liu, and S. Shen, "Robust and efficient quadrotor trajectory generation for fast autonomous flight," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3529–3536, 2019.

[11] B. Zhou, F. Gao, J. Pan, and S. Shen, "Robust real-time uav replanning using guided gradient-based optimization and topological paths," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 1208–1214.

[12] P. Florence, J. Carter, and R. Tedrake, "Integrated perception and control at high speed: Evaluating collision avoidance maneuvers without maps," in *Algorithmic Foundations of Robotics XII*. Springer, 2020, pp. 304–319.

[13] B. T. Lopez and J. P. How, "Aggressive 3-d collision avoidance for high-speed navigation," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 5759–5765.

[14] J. Zhang, C. Hu, R. G. Chadha, and S. Singh, "Maximum likelihood path planning for fast aerial maneuvers and collision avoidance," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 2805–2812.

[15] N. Bucki, J. Lee, and M. W. Mueller, "Rectangular pyramid partitioning using integrated depth sensors (rappids): A fast planner for multicopter navigation," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4626–4633, 2020.

[16] N. Bucki and M. W. Mueller, "Rapid collision detection for multicopter trajectories," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 7234–7239.

[17] J. R. Nistler and M. F. Selekwa, "Gravity compensation in accelerometer measurements for robot navigation on inclined surfaces," *Procedia Computer Science*, vol. 6, pp. 413–418, 2011.

[18] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, Apr. 2013.