2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Madrid, Spain, October 1-5, 2018

P-CAP: Pre-computed Alternative Paths to Enable Aggressive Aerial Maneuvers in Cluttered Environments

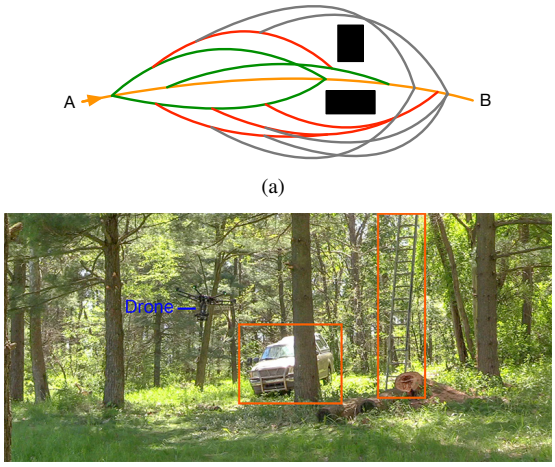
Ji Zhang, Rushat Gupta Chadha, Vivek Velivela, and Sanjiv Singh

***Abstract*— We propose a novel method to enable fast autonomous flight in cluttered environments. Typically, autonomous navigation through a complex environment requires a continuous heuristic search on a graph generated by a k-connected grid or a probabilistic scheme. As the vehicle progresses, modification of the graph with data from onboard sensors is expensive as is search on the graph, especially if the paths must be kino-dynamically feasible. We suggest that computation needed to find safe paths during fast flight can be greatly reduced if we precompute and carefully arrange a dense set of alternative paths before the flight. Any prior map information can be used to prune the alternative paths to come up with a data structure that enables very fast online computation to deal with obstacles that are not on the map but only detected by onboard sensors. To test this idea, we have conducted a large number of flight experiments in structured (large industrial facilities) and unstructured (forests-like) environments. We show that even in the most unstructured environments, this method enables flight at a speed up to 10m/s while avoiding obstacles detected from onboard sensors.**

I. Introduction

Fast autonomous flight in complex environments is challenging for many reasons. Even if high fidelity vehicle state information (typically 6 DOF) is available at a high frequency, planning paths to avoid obstacles discovered with onboard sensors requires creating and updating a map of the environment that can be searched for kinodynamically feasible paths. This is computationally expensive. Since computational resources available for flying vehicles are limited, ideally, we would like a method that can guide an aerial vehicle with low computational complexity. One method is to use a hierarchical method that separates the problem of safe flight into two subproblems. One part solves a global path-planning problem by searching a k-connected grid with a heuristic ensuring that it does not get stuck into local minima. A second part solves a local problem that runs in parallel tracks the global path while avoiding obstacles. This method has been used for ground [1] and aerial [2], [3] vehicles effectively but still requires considerable computation. Here, we propose a method that reduces computational complexity considerably such that it is possible to ensure safe flight using very lightweight computation onboard the aerial vehicle.

We propose to do this by trading computational complexity with memory. Instead of searching a graph that is continuously being updated by onboard sensors, we precompute a dense set of alternative kinodynamically feasible paths and arrange them in a manner that enables extremely fast switching between paths when obstacles are encountered. Any prior map information is used to prune the set of alternate paths before flight starts. Under this method, the task of safe guidance during the flight is to ensure that the current path segment is collision free. If not, another alternative path is examined for safe travel. Surprisingly, a practically small set of paths is able to enable fast flight even when it is necessary to negotiate obstacles that are only discovered during the flight. In our experience, the proposed method requires little onboard computation, typically $< 2\%$ of a single CPU thread on a modem embedded computer.



(a)

(b)

Fig. 1. (a) A simple illustration of pre-computed alternative paths. A main path in yellow connects between start point A and end point B. Alternative paths are organized at different levels, colored in green, red, and gray, respectively. The main path and alternative paths are generated based on a prior map, therefore the paths do not collide with structures on the map, represented by the two black rectangles. During navigation, the vehicle switches among the pre-computed paths to avoid obstacles. (b) A photo from an experiment where the proposed method enables a lightweight aerial vehicle to maneuver aggressively at 10m/s in a cluttered forest environment, avoiding a ladder and a truck placed on the path as obstacles. More details regarding the experiment are available in Section V-B, Test 1.

The method of generating and arranging these paths is conducted as follows. A set of paths are created based on a prior map, these include a main path connecting from start to goal, and local alternative paths for the purpose of obstacle avoidance. The alternative paths are organized at multiple levels (see an example in Fig. 1(a), where green, red, and gray paths are at three different levels). Paths at higher levels branch out from paths at lower levels, and eventually merge into paths at lower levels. During navigation, if a path is blocked by an obstacle, the vehicle switches to a path which branches out from the current path. This process recurs if

J. Zhang and V. Velivela are with Carnegie Mellon University, R. Gupta Chadha and S. Singh are with Near Earth Autonomy, Inc.

multiple obstacles exist blocking paths at different levels. Such a representation enables autonomous navigation even in complex environments, such as forests, that have changed enough that previously planned paths are no longer feasible. This is shown in Fig 1 (b) where obstacles as a ladder and a truck occlude the paths that were planned before flight.

A key insight is that the pre-computed paths function as a summary of the prior map – these paths are guaranteed to avoid structures on the map. Hence, during flight the obstacle avoidance does not heavily rely on onboard sensor data to generate local paths, but only uses the sensor data to check for obstacles along the path. This significantly lowers the requirement for the data density to carry out the path switching reliably. Another key insight is that all alternative paths are organized to lead to the goal. This way, the vehicle does not need to spend effort on searching for a path that points to the goal, but only focuses on collision check. In other words, for any collision-free path that is chosen, the vehicle will navigate to the goal by following the path.

The proposed method has been tested with an aerial vehicle in 42 flight experiments in 5 different environments and operates at speeds up to 10m/s. We believe that this is the fastest autonomous flight that has been demonstrated to date in such environments. Results are in a public video[1].

II. Related Work

Our work is most related to path planning and collision avoidance with an emphasis on robot navigation. The problem involves solving for a path for a vehicle to travel from start to goal, given a traversable representation of the environment. Graph search-based methods such as Dijkstra [4], A* [5], and D* [6] algorithms traverse different states on the graph to search for paths. On the other hand, sampling-based methods cover the graph with random samples. Paths are generated by connecting selected samples. Contemporary sampling-based methods such as Rapidly-exploring Random Tree (RRT) [7] and its variants [8]–[10] have high capabilities to handle maps in large scales, generating paths in a relatively short amount of time compared to graph search-based method. In this paper, we use a state-of-the-art RRT-based method, BIT* [11], for generation of the main path. BIT* method is known for its computation speed.

Some path planning methods pre-process a prior map to extract traversable information and convert the information into particular representations. The converted representations facilitate or accelerate the path search. Typical methods include Voronoi graph [12], vector field [13], and Probabilistic Roadmap (PRM) [14]–[16]. Here, PRM-based methods randomly sample on the map to create a connectivity graph. Paths are found by searching on the graph. In comparison, these methods share the same insight with the proposed method that all pre-process a prior map and summarize it into certain representations. However, a key difference is that the summarized representations in the previous methods are not unique for a single pair of start and goal. For this reason, methods such as PRM still need to traverse the graph in order to find a path given a specific goal point. In our method, all paths are generated and organized to lead to the goal. The result is that the navigation problem is simplified and becomes a collision check and path switch problem.

The novelty of the proposed method is not in computing the paths – we are open to use any suitable method for path generation. The contribution of the paper is in separation of path generation from onboard computation, by offline pre-computing the alternative paths. This way, the navigation is less sensitive to the density of onboard perception sensor data, saving computation and reducing response time. More importantly, the method avoids problems caused by varying density in the perception sensor data ranging from close to far ahead of the vehicle. The paths can also be pre-computed taking into account curvature and kinematics constraints.

III. Notations and Definitions

The proposed method uses pre-computed alternative paths to enable vehicle navigation and obstacle avoidance. We start with definitions of the alternative paths as the following.

- Define $\mathcal{Q} \subset \mathbb{R}$ as the configuration space of a vehicle, and $\mathcal{Q}_{occu} \subset \mathcal{Q}$ as the occupied subspace based on the prior map. $\mathcal{Q}_{occu}$ is untraversable. The traversable space is defined as $\mathcal{Q}_{trav} = \mathcal{Q} \backslash \mathcal{Q}_{occu}$. Let $A \in \mathcal{Q}_{trav}$ and $B \in \mathcal{Q}_{trav}$ be the navigation start and end points.
- Define a main path connecting from A to B. The main path is at level 0, denoted as $\xi_0 \in \mathcal{Q}_{trav}$. Let $S(\xi_0) = A$ and $E(\xi_0) = B$ be the start and end points of ξ_0.
- Each path can have alternative paths, called branches. The set of branches of ξ_0 are at level 1, denoted as $\mathcal{B}(\xi_0)$. The branches of a level $i \in \mathbb{Z}^+$ path, if available, are at level $i + 1$.
- A path at level i is denoted as $\xi_i^j \in \mathcal{Q}_{trav}$, where $j \in \mathbb{Z}^+$ is the branch index at level i. The start and end points are denoted as $S(\xi_i^j)$ and $E(\xi_i^j)$. The set of branches is denoted as $\mathcal{B}(\xi_i^j)$.
- Define the parent of ξ_i^j as the path that ξ_i^j is started from, denoted as $\xi_s(\xi_i^j)$. Further, the path that ξ_i^j ends at is denoted as $\xi_e(\xi_i^j)$.
- Define an ancestor of ξ_i^j as any path ξ_k^l, $k, l \in \mathbb{Z}^+$, $k < i$, that is reachable to ξ_i^j through parent connections, ξ_i^j is also called a descendant of ξ_k^l.
- A path set $\mathcal{G} = \{\xi_0, \xi_i^j\}$, $i, j \in \mathbb{Z}^+$, contains all aforementioned paths. The maximum level in the set, $n \in \mathbb{Z}^+$, is the level of the path set.

As a convention in this paper, we use 'obstacles' to refer to objects that do not exist on the prior map but appear on paths in $\mathcal{G}$. The occupied space on the prior map $\mathcal{Q}_{occu}$ refers to 'structures'. The navigation problem is to drive a vehicle from A to B and avoid obstacles using paths in $\mathcal{G}$.

IV. Method

A. Navigation Algorithm

Given a path set $\mathcal{G} = \{\xi_0, \xi_i^j\}$, $i, j \in \mathbb{Z}^+$, Algorithm 1 carries out the navigation by switching paths in $\mathcal{G}$ to avoid

[1]Experiment video: `https://youtu.be/BZ1A9SB_9EE`

Algorithm 1: Navigation

```
1  input: path set G = {ξ_0, ξ_i^j}, i, j ∈ Z^+ between A and B,
2         obstacles from processing perception sensor data,
3         obstacle avoidance distance D;
4  output: navigation command;
5  begin
6      Initialize vehicle location at A, set the current path ξ_c ← ξ_0;
7      while B is not approached do
8          Check if any obstacle is present on ξ_c;
9          if obstacle is present on ξ_c then
10             Find branches in B(ξ_c) whose start points are before
               the obstacle on ξ_c and within D to the obstacle, denote
               the set of branches as C, C ⊂ B(ξ_c);
11             Check branches in C in the decreasing order of the
               distances from their start points to the obstacle on ξ_c,
               find the first collision-free branch if available, denoted
               as ξ', if all branches in C are blocked by obstacles,
               find the one with the most number of paths branching
               out before the obstacle, denoted as ξ', ξ' has at least
               one branch before the obstacle, if multiple branches
               share the same criteria, use an empirical selection;
12             if ξ' is available then
13                 Navigate the vehicle to S(ξ') and ξ_c ← ξ';
14             end
15             else
16                 Start a breadth-first search from the start point of
                   the first branch before the obstacle on ξ_c, and find
                   a collision-free path, denoted as ξ'';
17                 if ξ'' is available then
18                     Navigate the vehicle to S(ξ'') and ξ_c ← ξ'';
19                 end
20                 else
21                     Navigate the vehicle to A following ξ_c's
                       ancestors and report navigation unsuccessful;
22                 end
23             end
24         end
25         else if E(ξ_c) is approached then
26             ξ_c ← ξ_e(ξ);
27         end
28         else
29             Navigate the vehicle on ξ_c;
30         end
31     end
32     Finish and report navigation successful;
33 end
```

obstacles. Starting on the main path, the algorithm receives obstacle information from processed perception sensor data. If an obstacle is detected, the algorithm checks available branches of the current path ξ_c, in $\mathcal{B}(\xi_c)$. Here, an obstacle avoidance distance D is used. The algorithm checks branches in $\mathcal{B}(\xi_c)$ whose start points are before the obstacle on ξ_c and within D to the obstacle. In other words, the vehicle will only leave the current path to avoid an obstacle within D to the obstacle. The branches are checked in the decreasing order of the distances from their start points to the obstacle, starting from the first branch whose start point is within D to the obstacle. The algorithm chooses the first collision-free branch found. However, if all checked branches are blocked, the algorithm chooses the one with the most number of paths branching out before the obstacle on the branch. Denote the chosen branch as ξ', ξ' is required to have at least one path branching out before the obstacle. After the vehicle approaches $S(\xi')$, the current path ξ_c is switched to ξ'. The path switching recurs if multiple obstacles exist blocking paths at different levels. Here, if multiple branches meet the same criteria, an empirical selection is made which picks the one with the largest $n-k$, where n is the level of $\mathcal{G}$ and k is the level of the chosen branch. If multiple branches still meet the same criteria, the algorithm picks the one furthest away from all surrounding obstacles.

In the case that no clear branch is available before the obstacle, the algorithm performs a breadth-first search starting from the start point of the first branch before the obstacle. As shown on lines 16-22 in Algorithm 1, this is considered a backup strategy and can possibly drive the vehicle back along the path. Preferably, the vehicle should only drive forward and use path switching to avoid obstacles. If the breadth-first search still does not find a collision-free path, the vehicle is driven to the start and navigation is unsuccessful.

Let us model the collision check time on the main path and each alternative path to be bounded by $O(1)$. Let $d_{\min}$ and $d_{\max}$ be the minimum and maximum intervals between two consecutive start points. Define h as the maximum number of branches sharing the same start point. Let R be the perception sensor range. Recall D is the obstacle avoidance distance, $R \geq D \geq d_{\max} \geq d_{\min}$. For each recursion, if not executing the breadth-first search, Algorithm 1 checks at most $h\lceil D/d_{\min}\rceil$ alternative paths as well as the main path. If executing the breadth-first search, however, Algorithm 1 traverses all alternative paths within the perception sensor range, with maximally $(2h\lceil R/d_{\min}\rceil)$ alternative paths at one level (before and after the vehicle), and $(2h\lceil R/d_{\min}\rceil)^n$ alternative paths at all levels, as well as the main path. Hence, the computational complexity can be stated below.

Theorem 1: Algorithm 1 has a computational complexity of $O(h\lceil D/d_{\min}\rceil)$ without executing the breadth-first search, and of $O((2h\lceil R/d_{\min}\rceil)^n)$ otherwise.

Next, let us analyze the probability of successful obstacle avoidance. In a path set $\mathcal{G}$, the main path ξ_0 has length L_m. A path ξ_i^j, $i, j \in \mathbb{Z}^+$, has the maximum length L_b. To simplify the analysis, we model obstacles as volumeless particles distributed on the paths. Obstacle distribution is independent and identical along all paths in $\mathcal{G}$. Let σ be the probability for an obstacle to appear on a path segment with a unit length. We assume no obstacle is within D after A or before B. Otherwise, an obstacle right after A or before B can simply cause a navigation failure. Since Algorithm 1 executes the breadth-first search as a backup strategy, in the following, we analyze the probability for the vehicle to navigate from A to B without executing the breadth-first search.

Lemma 1: The probability that the vehicle executes the breadth-first search is bounded from above by an exponential function that decreases w.r.t. n and $\lceil D/d_{\max}\rceil$.

Proof: In two cases the vehicle will execute the breadth-first search. In the first case, the vehicle navigates on a level n path ξ_n^j, $j \in \mathbb{Z}^+$, and is blocked by an obstacle. This requires an obstacle to appear on a path at each level from level 0 to n. The probability for an obstacle to be on the main path ξ_0 is σL_m, and the probability for an obstacle to be on path ξ_i^j, $i \in \mathbb{Z}^+$, is no greater than σL_b. Thus, the

probability for the first case to occur is bounded from above,

$$p_n \leq \sigma L_m (\sigma L_b)^n = \sigma^n L_m L_b^n. \quad (1)$$

In the second case, the vehicle navigates on a path ξ_k^j, $k \in \mathbb{Z}^+$, $k < n$, and cannot find a clear branch of ξ_k^j to execute before the obstacle. This must be because obstacles are also present on the branches. Let $\mathcal{C}$ be a subset of $\mathcal{B}(\xi_k^j)$, $\mathcal{C} \subset \mathcal{B}(\xi_c)$, which contains branches in $\mathcal{B}(\xi_c)$ with the start points before the obstacle and within D to the obstacle. Since the maximum interval between the start points of two consecutive branches is $d_{\max}$, there are at least $[D/d_{\max}]$ branches in $\mathcal{C}$. For each of these branches, there is an obstacle on the branch as well and there is no path branching out before the obstacle. This requires an obstacle to appear within $d_{\max}$ after the start point, with a probability $\sigma d_{\max}$. Considering all branches in $\mathcal{C}$, the second case has,

$$p_{1\dots n-1} \leq (\sigma d_{\max})^{[D/d_{\max}]}. \quad (2)$$

Considering both cases, the vehicle executing the breadth-first search to avoid an obstacle is bounded from above,

$$p_{1\dots n} \leq p_{1\dots n-1} + p_n \leq \sigma^n L_m L_b^n + (\sigma d_{\max})^{[D/d_{\max}]}. \quad (3)$$

Lemma 1 analyzes in the case that the number of obstacles is unlimited. Each obstacle has certain probability to appear on a path in $\mathcal{G}$. In Lemma 2, we tackle the problem from a different direction to understand how many obstacles can be avoided regardless of the distribution of obstacles.

Lemma 2: The vehicle can avoid a minimum number of $\min(n, [D/d_{\max}])$ obstacles without executing the breadth-first search regardless of the distribution of obstacles.

Proof: First, let us prove that the vehicle can avoid at least $\min(n, [D/d_{\max}])$ obstacles regardless of the distribution. To this end, we prove it takes at least $\min(n, [D/d_{\max}]) + 1$ obstacles for the vehicle to execute the breadth-first search. Considering two cases, if it happens on a level n path ξ_n^j, $j \in \mathbb{Z}^+$, the vehicle must be blocked on a path at each level, from level 0 to n. This requires at least $n+1$ obstacles, where $n+1 \geq \min(n, [D/d_{\max}]) + 1$. Second, if it happens on a level k path ξ_k^j, $k \in \mathbb{Z}^+, k < n$, the vehicle must check at least $[D/d_{\max}]$ branches and find at least $[D/d_{\max}]$ obstacles blocking the branches. Another obstacle must appear on ξ_k^j, leading to at least $[D/d_{\max}]+1$ obstacles in total. Likewise, there is $[D/d_{\max}] + 1 \geq \min(n, [D/d_{\max}]) + 1$.

Second, we prove there exists a distribution of no more than $\min(n, [D/d_{\max}]) + 1$ obstacles which can cause the vehicle to execute the breadth-first search. Consider $d_{\min} = d_{\max}$. If $n \leq [D/d_{\max}]$, we can place an obstacle on a path at each level, from level 0 to n. With $n+1$ obstacles in total, the vehicle is forced to enter a level $n-1$ path ξ_{n-1}^j, and then execute the breadth-first search due to not finding a clear branch before the obstacle on ξ_{n-1}^j, since another obstacle is blocking the level n path. Alternatively, if $n > [D/d_{\max}]$, we can place one obstacle on the main path ξ_0, and an obstacle on each of the $[D/d_{\max}]$ branches before the obstacle on ξ_0, right after the start point. The vehicle cannot find a clear branch and executes the breadth-first search. In both cases, we need no more than $\min(n, [D/d_{\max}]) + 1$ obstacles.

B. Main Path Generation

The main path is the best path between start and goal that can be developed before flight. We currently use the BIT* method [11] to generate this path. Given a prior map as a 3D point cloud, we first create an Octree and then run the BIT* method. When generating the path, we selectively use three types of constraints to limit the curvature, elevation, and introduce no-fly zones, based on the mission. For elevation constraints, we use Axelsson's method [17] to extract the ground from the map, then define the minimum and maximum height of the path above the ground. For no-fly zone constraints, manually defined polygons are used and the path is not allowed to enter the polygons. For all constraints, a checking step is used during expansion of the RRT. If a constraint is violated, the corresponding state is rejected. The resulting path is further smoothed in a sliding window average to meet our requirement for high-speed flights.

C. Alternative Path Generation

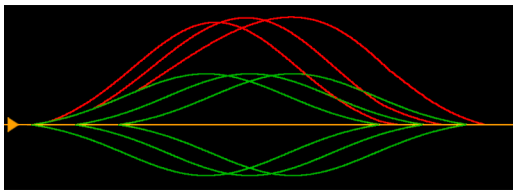
Alternative paths are generated using bell-shaped cubic spline curves. Boundary conditions are used at the start and end points to make sure continuous acceleration through path switches. Each alternative path has two free parameters, determining the length (parallel to the main path) and width (perpendicular to the main path) of the alternative path. Fig. 2(a) gives an example of alternative paths in a 2D case. This case is suitable if the vehicle elevation needs to be constant during the flight. The alternative paths at level 1 are organized in pairs. On each level 1 path, level 2 paths branch out. Here, we only show three pairs of level 1 paths and three level 2 paths from a single level 1 path, for clarity of illustration. Fig. 2(b)-(c) show an example of alternative paths in a 3D case. In this case, the alternative paths at each level are organized in batches. For each batch, the paths branch out from their parent in different directions.

The alternative paths are generated based on a prior map, hence they do not interfere with structures on the map. When generating an alternative path, we start with a default set of length and width. Collision check is run. If the path collides with the map, a two-dimensional search is started. The search reports collision-free paths found, or no collision-free path is available. The path whose parameters are the closest to the default parameters (sum of the difference to the default parameters in both dimensions) is chosen. We understand the search can be optimized, and usage of bell-shaped spline curves may neglect valid paths which do not fit into spline curves. However, considering a large number of alternative paths being generated, the neglected paths should have minor effect on reducing the chance of successful navigation.

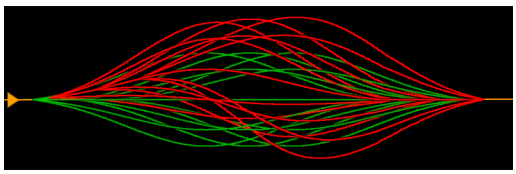
V. Experiments

A. Simulation

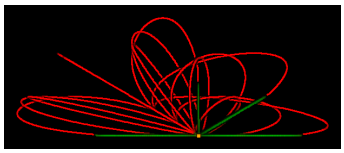
We test the proposed method in simulation using three levels of alternative paths in a 2D case. As shown in Fig. 3,



(a)



(b)



(c)

Fig. 2. Example alternative paths in (a) a 2D case in top-down view and (b)-(c) a 3D case in top-down view and front view. The yellow curve is the main path, from left to right. The green curves and red curves are alternative paths at level 1 and level 2. In the 2D case (in (a)), level 1 paths are in pairs, branching out on two sides of the main path. In the 3D case (in (b)-(c)), paths at each level are organized in batches, branching out in different directions. Note that the figures only show a few level 1 paths and level 2 paths starting from a single level 1 path, for clarity of illustration.

the alternative paths are about 40m in length along the main path. Obstacles are defined as 5m squares. Through the test, 1-5 obstacles are placed along the paths to introduce blockage. In Fig. 3(a), a single obstacle blocks the main path. An alternative path at level 1 (green curve) is chosen. In Fig. 3(b), two obstacles are employed where one blocks the main path and the other stays close to the main path, blocking alternative paths coming out on one side. A level 1 path on the other side is chosen (green curve). In Fig. 3(c), three obstacles are placed, one on the main path and the other two close to the main path blocking alternative paths coming out on both sides. The first collision-free alternative path is chosen, which is a level 1 path starting after the first obstacle (green curve). In Fig. 3(d), a different placement of tree obstacles is used, causing the vehicle to switch to a level 1 path (green curve) followed by a level 2 path (red curve) to avoid. In Fig. 3(e), one more obstacle is adopted. The vehicle goes through three levels of alternative paths (green, red, and gray curves). In Fig. 3(f), a distribution of five obstacles forces the vehicle to check the paths at the highest level and still find no choice. A breadth-first search is run thereafter which finds a level 1 path connected with a level 2 path (green and red dashed curves). Note that this trajectory would be the optimal for Fig. 3(e) and Fig. 3(f). It is not chosen because the simple path switching strategy does not check alternative paths recursively at multiple levels. However, such a case rarely happens in practice due to the fact that perception sensor data is sparse far ahead of the vehicle. When the path is determined, far obstacles are often yet undetected and the case does not start until close obstacles are passed or too close for a decision change.

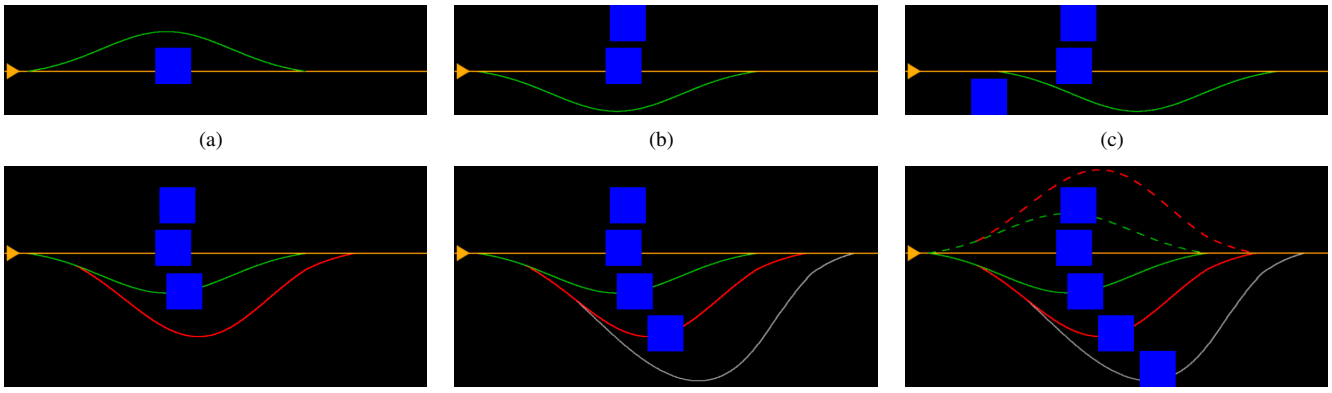
B. UAV Experiments

Our experimental platform is shown in Fig. 4. This is a DJI Matrice 600 Pro aircraft carrying a DJI Ronin MX gimbal. A sensor-computer pack is mounted to the gimbal and therefore is kept in the flight direction for obstacle detection. The sensor-computer pack consists of a Velodyne Puck laser scanner, a camera at 640×360 pixel resolution, and a MEMS-based IMU. A 3.1GHz i7 embedded computer carries out all processing. The state estimation is based on our previous work [18], integrating data from the three sensors to provide vehicle poses and registered laser scans. The map is built from a manual flight a month before the UAV experiment.

We report on three flight tests. Test 1 is in a complex forest environment. As shown in Fig. 5, the pre-computed paths consist of a main path and alternative path at two levels, based on a prior map. Alternative paths are created with a minimum curvature constraint to meet the requirement for high-speed flying. The test has three separate runs, with a clear path, one obstacle, and two obstacles on the path. For each run, the UAV flies at a speed of 10m/s. In the case of a clear path, the vehicle follows the main path to the end. In the case of one obstacle, a ladder is placed on the main path and the vehicle avoids by switching to a level 1 path. In the case of two obstacles, a ladder and a truck are left in the field. The vehicle switches to a level 1 path, then shortly after, switches to a level 2 path to avoid both obstacles.

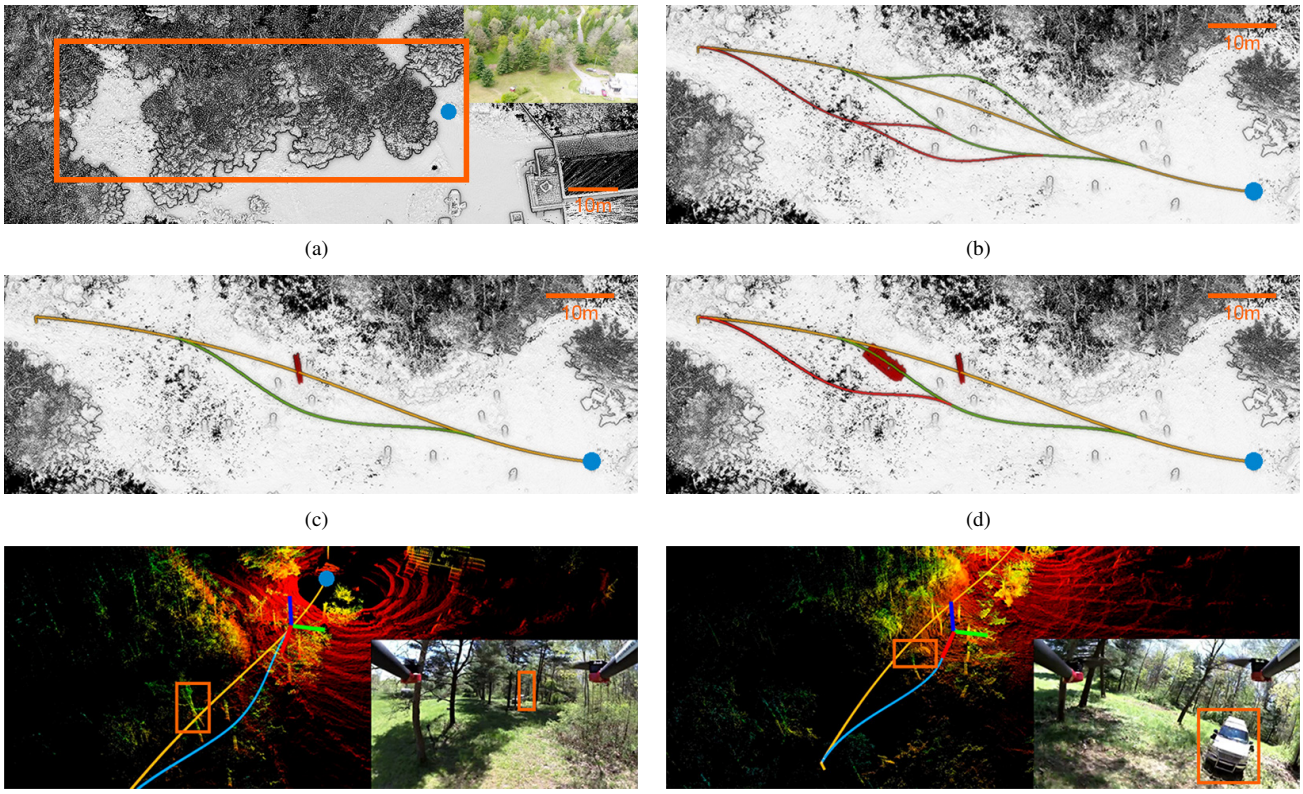
For comparison purposes, we run the BIT* method [11] with data logged during Test 1. The test is conducted on the same sensor-computer pack as in Fig. 4. Laser scans are registered w.r.t. the prior map before processing. Fig. 6(a) shows the paths generated using the registered laser scans, and Fig. 6(b) presents the paths computed using a combination of the registered laser scans and prior map. Since the BIT* method is an anytime method, we let the method run for 100ms, 500ms, and 2s, respectively. The corresponding paths are in yellow, green, and red. If the method does not generate a path within a time limit, the path is not shown. As shown in Fig. 6(a), without using the prior map, the paths change shape dramatically over time, which potentially limits the flight speed. If using the prior map, as shown in Fig. 6(b), the paths are stabler. The smoothness still does not meet our requirement for high-speed flying. The paths need to be smoothed further. However, for two out of the three start points, the method does not generate a path within 500ms (only one start point connects to a green path). The heavy processing makes it hard to use in high-speed flights.

Test 2 is on an inactive industrial site. As shown in Fig. 7, the main path is manually created with spline curves fit through waypoints. This way, it collides with three obstacles – an instant canopy, a tree, and a wire. The vehicle avoids all three obstacles by switching to level 1 paths. In Test 3, the mission is to fly over an orchard, as shown in Fig. 8. The alternative paths follow the same convention as in Test 2. The difference is that the alternative path only cover the first 100m after take-off and last 100m before landing. This



(a) (b) (c) (d) (e) (f)

Fig. 3. Simulation results. The test utilizes a 2D path set. The main path is in yellow, from left to right. Three levels of alternative paths are in green, red, and gray, respectively. Each alternative path is about 40m long along the main path. Obstacles are 5m squares placed on the paths to introduce blockage. In (a), one obstacle is used and the vehicle chooses an alternative path at level 1 to avoid. In (b), two obstacles are present, one blocking the main path and the other blocking the alternative paths on one side of the main path. The vehicle chooses an alternative path on the other side. In (c), three obstacles block the main path and its both sides. The vehicle chooses the first collision-free alternative path which starts after the first obstacle. In (d), a different placement of three obstacles causes the vehicle to chose a level 1 and a level 2 path to avoid. In (e), four obstacles are present bringing the vehicle onto three alternative paths at levels 1-3. In (f), five obstacles appear and the vehicle cannot avoid even using paths at the highest level. A breadth-first search is run which finds the dashed curves. Note that in (e)-(f), the vehicle does not choose the dashed curves initially but a more complex trajectory spanning three levels. Such decision is possible due to our simple path switching strategy not checking alternative paths recursively at multiple levels.



(a) (b) (c) (d) (e) (f)

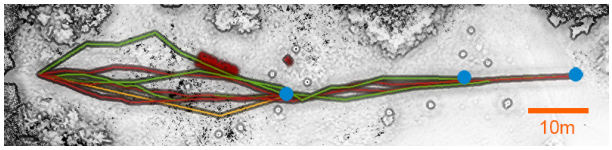
Fig. 5. Result of Test 1. (a) shows a 3D point cloud used as the prior map. The test area is in the orange rectangle with the blue dot as the start point. An aerial image from the same area is shown at the upper-right corner. (b) shows the pre-computed paths. The main path is in yellow. Two levels of alternative paths are in green and red, respectively. The test contains three separate runs, all at 10m/s. First, no obstacle is present and the vehicle follows the main path to the end. Then, in (c), a ladder is placed on the main path as an obstacle. The vehicle chooses an alternative path at level 1 to avoid. The dark-red object is the detected ladder in laser scans overlaid on the map. Finally, in (d), a ladder and a truck are left in the field as two obstacles, blocking the main path and a level 1 path. Consequently, the vehicle switches twice, to a level 1 and then a level 2 path. The dark-red objects are the ladder and truck. (e)-(f) show registered laser scans during the flights and corresponding images from an onboard video camera. The ladder and truck are labeled with the orange rectangles. Also, (e)-(f) are right at the path switches from the main path to level 1 path, and from the level 1 path to level 2 path.

is because the middle course of the flight is high above the ground and the chance to collide with an obstacle is very small. Generation of the main path uses manually defined no-fly zones to prevent the vehicle from flying above buildings

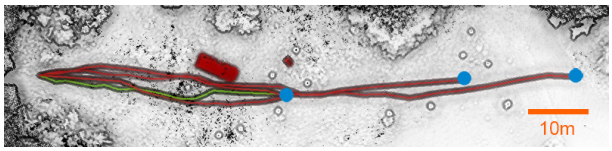
Fig. 4. UAV experimental platform. A DJI Matrice 600 Pro carries our sensor-computer pack on an DJI Ronin MX gimbal. The gimbal keeps the sensors in the flight direction for obstacle detection. The sensor-computer pack consists of a Velodyne Puck laser scanner, a camera at 640×360 pixel resolution, and a low-grade IMU. An embedded i7 computer carries out all onboard processing. The vehicle is equipped with a GPS module but GPS signals are unused through all tests included in the paper.

and a bond. Elevation of the path is also limited between 15-20m above the ground for the middle course of the flight. While approaching the end, a tractor is placed in the field. The vehicle avoids by switching to a level 1 path.

For further evaluation, we run tests in simulation using the setup in Test 2. Obstacles are modeled as 1m cubes randomly and repeatedly generated from a uniform distribution in the 3D space. As shown in Fig. 9, the rate of navigation failure or full navigation blockage increases w.r.t. the number of obstacles. Employing more alternative paths (2 levels instead of 1 level) helps reduce the rate of navigation failure to a large extent. Further, involving the breadth-first search helps produce a significantly lower rate of navigation failure. These results provide guidance to help specify the configuration of the alternative paths for the offline path generation.

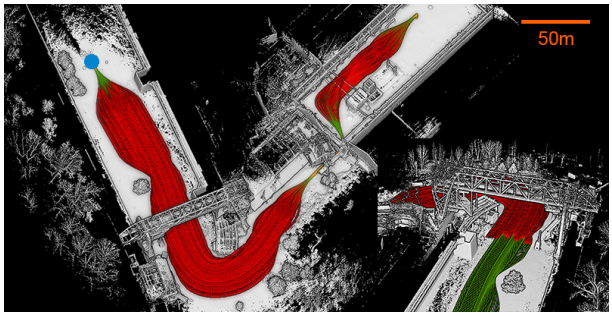


(a)

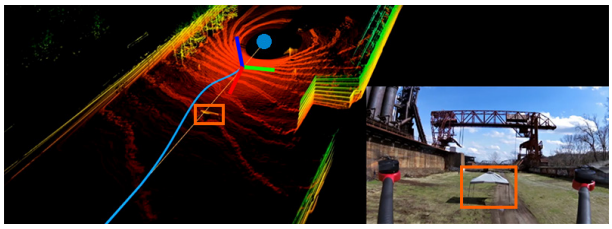


(b)

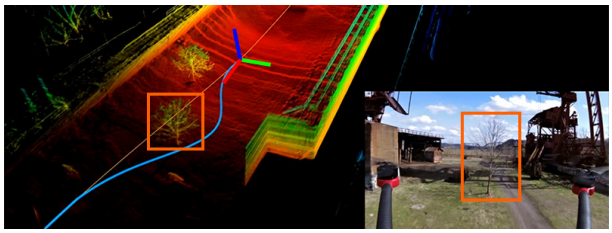
Fig. 6. Paths generated by the BIT* method [11] using data logged in Test 1. Three start points (blue dots) are selected from the path in Fig. 5(d) and from which the BIT* method is run. Since the BIT* method is an anytime method, we let the method run for 100ms, 500ms, and 2s, respectively. The corresponding paths are in yellow, green, and red. In (a), we only use the data from onboard perception sensors, and in (b), we use a combination of the data from onboard perception sensors and prior map. Laser scans are registered and overlaid on the map before processing. Here, if the method does not generate a path within a time limit, the path is not shown. As we can see, without using the prior map (in (a)), the method can generate a path from each of the three start points within 500ms, but not within 100ms (only one start point connects to a yellow path). The paths change shape dramatically over time, potentially limiting the flight speed. On the other hand, if using the prior map (in (b)), the paths are stabler. However, only one out of the three start points has a path generated within 500ms (the point connected to the green path), making the method hard to use in high-speed flights. The test reveals the difficulty for online sampling-based methods to enable high-speed maneuver in cluttered and complex environments.



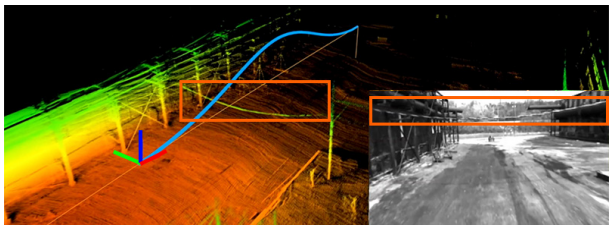
(a)



(b)



(c)

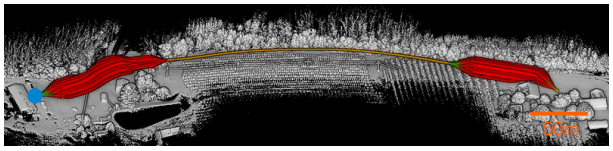


(d)

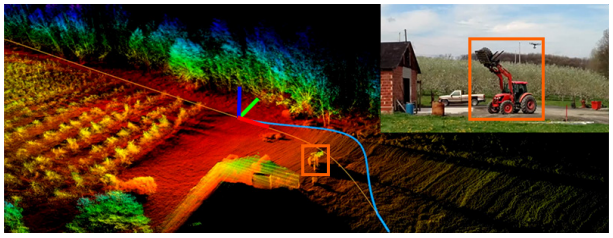
Fig. 7. Result of Test 2 conducted on an inactive industrial site. (a) shows the prior map and pre-computed paths. There are 597 level 1 paths (green curves) and 8101 level 2 paths (red curves). A different view is present at the lower-right corner with the level 2 paths cropped to reveal the vertical section. As we see, the alternative paths go around structures on the map such as the tree. (b)-(d) show the avoidance of three obstacles placed on the main path – an insistent canopy, a tree, and a wire. The vehicle avoids all three obstacles by switching to level 1 paths. The speed is 4m/s avoiding the insistent canopy and tree, and 2m/s avoiding the wire.

Finally, let us inspect some metrics in the obstacle avoidance. For collision check, the main path is stored in a 3D K-D tree in 100m segments. The closest point on the path is found for each laser point, and collision is determined. A K-D tree query takes 3.5μs. For alternative paths, each point on the path is compared to the laser points. Laser scans are received at 5Hz and downsized to a resolution of 0.4m per point (half of the UAV diameter). The downsized laser scans are then filtered by a bounding box of the path. As shown in Table I, the processing time to make a path switch decision is $< 50\mu s$. Most of the path switches are made more than 30m before the obstacle. The latest path switch is at 24m before the wire, and the earliest is 59m before the tractor.

(a)



(b)



(c)

Fig. 8. Result of Test 3 conducted in an orchard. (a) shows an aerial image with the main path. No-fly zones are used in generation of the main path to prevent the vehicle from flying above buildings and a pond. The path is limited in elevation between 15-20m above the ground for the middle course of the flight. (b) presents the prior map and pre-computed paths. The alternative paths follow the same convention as in Test 2, but only cover the first 100m after take-off and last 100m before landing. This is because the middle course of the fight is high above the ground and the chance to collide with an obstacle is very small. (c) shows the avoidance of a tractor placed on the main path close to the end. The vehicle avoids the tractor by switching to a level 1 path. The speed is 8m/s during the middle course of the flight and 4m/s within 100m to the take-off and landing.

VI. Conclusion

The paper proposes a novel method which offline generates a set of alternative paths to enable robot navigation. The set of paths are computed at multiple levels, based on a prior map. Obstacle avoidance is conducted by switching paths, from lower levels to higher levels. This method eliminates the necessity of online path creation, simplifying the navigation to a collision check and patch switch problem. The resulting system consumes little onboard computation with low latency, taking only $< 2\%$ of a single CPU thread on a modern embedded computer. Consequently, it makes possible for a lightweight UAV to maneuver aggressively in a cluttered forest environment, at a speed of 10m/s.

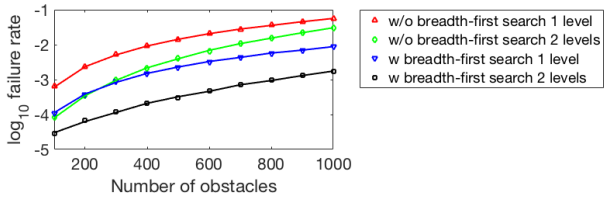


Fig. 9. Further evaluation using the setup in Test 2. Obstacles are modeled as 1m cubes randomly distributed in the 3D space. As we see, the rate of navigation failure or full navigation blockage increases w.r.t. the number of obstacles. Using 2 levels of alternative paths instead of 1 level reduces the rate of navigation failure to a large extent. Further, involving the breadth-first search helps produce a significantly lower rate of navigation failure.

TABLE I

Obstacle avoidance metrics for the three flight tests

Test	Obstacle	Proc. time	Path switch distance	Flight speed
1	Ladder (1-obstacle run)	37μs	36m	10m/s
1	Ladder (2-obstacle run)	35μs	35m	10m/s
1	Truck (2-obstacle run)	20μs	28m	10m/s
2	Instant canopy	26μs	26m	4m/s
2	Tree	33μs	53m	4m/s
2	Wire	10μs	24m	2m/s
3	Tractor	41μs	59m	4m/s

References

[1] D. Gonzlez, J. Prez, V. Milans, and F. Nashashibi, "A review of motion planning techniques for automated vehicles," *IEEE Trans. on Intelligent Transportation Sys.*, vol. 17, no. 4, pp. 1135–1145, 2016.

[2] D. Droeschel, M. Nieuwenhuisen, M. Beul, D. Holz, J. Stuckler, and S. Behnke, "Multi-layered mapping and navigation for autonomous micro aerial vehicles," *Journal of Field Robotics*, vol. 33, no. 4, pp. 451–475, 2016.

[3] S. Scherer, S. Singh, and L. Chamberlain, "Flying fast and low among obstacles: Methodology and experiments," *The International Journal of Robotics Research*, vol. 27, no. 5, pp. 549–574, 2008.

[4] R. Kala and K. Warwick, "Multi-level planning for semi-autonomous vehicles in traffic scenarios based on separation maximization," *J. of Intelligent and Robotic Systems*, vol. 72, no. 3/4, pp. 559–590, 2013.

[5] B. MacAllister, J. Butzke, A. Kushleyev, H. Pandey, and M. Likhachev, "Path planning for non-circular micro aerial vehicles in constrained environments," in *IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, May 2013.

[6] M. Rufli and R. Y. Siegwart, "On the application of the D search algorithm to time-based planning on lattice graphs," in *The European Conf. on Mobile Robots (ECMR)*, Dubrovnik, Croatia, Sept. 2009.

[7] S. M. LaValle, *Planning Algorithms*. New York, NY, USA: Cambridge University Press, 2006.

[8] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Chicago, IL, Sept. 2014.

[9] M. Otte and N. Correll, "C-FOREST: Parallel shortest path planning with superlinear speedup," *IEEE Transactions on Robotics and Automation*, vol. 29, no. 3, pp. 798–806, 2013.

[10] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[11] J. Gammell, S. Srinivasa, and T. Barfoot, "Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Seattle, WA, May 2015.

[12] P. Beeson, N. K. Jong, and B. Kuipers, "Towards autonomous topological place detection using the extended Voronio graph," in *IEEE International Conference on Robotics and Automation (ICRA)*, Barcelona, Spain, April 2005.

[13] G. A. S. Pereira, S. Choudhury, and S. Scherer, "A framework for optimal repairing of vector field-based motion plans," in *Intl. Conf. on Unmanned Aircraft Systems (ICUAS)*, Arlington, VA, June 2016.

[14] X. Yu, Y. Zhao, C. Wang, and M. Tomizuka, "Trajectory planning for robot manipulators considering kinematic constraints using probabilistic roadmap approach," *Journal of Dynamic Systems, Measurement, and Control*, vol. 139, 2017.

[15] D. Hsu, J.-C. Latombe, and H. Kurniawati, "On the probabilistic foundations of probabilistic roadmap planning," *The International Journal of Robotics Research*, vol. 25, no. 7, pp. 627–643, 2006.

[16] R. Geraerts and M. Overmars, "A comparative study of probabilistic roadmap planners," The Netherlands, July 2004.

[17] P. Axelsson, "DEM generation from laser scanner data using adaptive TIN models," *International Archives of the Photogrammetry and Remote Sensing*, vol. 33, pp. 110–117, 2000.

[18] J. Zhang and S. Singh, "Enabling aggressive motion estimation at low-drift and accurate mapping in real-time," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Singapore, May 2017.