P-CAL: Pre-computed Alternative Lanes for Aggressive Aerial Collision Avoidance

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

Abstract We here address the issue of air vehicles flying autonomously at a high speed in complex 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. The process is expensive especially if the paths must be kino-dynamically feasible. Aimed at tackling the problem from a different angle, we consider the case that the environment is mostly known from a prior map. The proposed method suggests the computation needed to find safe paths during fast flight can be greatly reduced if we pre-compute and carefully arrange a set of alternative paths before the flight. During the navigation, the vehicle selects a pre-computed path to navigate without the need to generate a new path. The result is that majority of the processing is migrated to offline path generation. Effectively, the onboard computation is significantly reduced, taking $< 3\%$ of a CPU thread on a modern embedded computer. In experiments, it enables a lightweight aerial vehicle to maneuver aggressively through a cluttered forest environment at 10m/s.

1 Introduction

Fast autonomous flight in complex environments is challenging. Planning paths to avoid obstacles discovered with onboard perception 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 way is to use a hierarchical method that separates the problem of safe flight into two subproblems. One problem 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 problem solves a local problem that runs in parallel tracks the global path while avoiding ob-

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

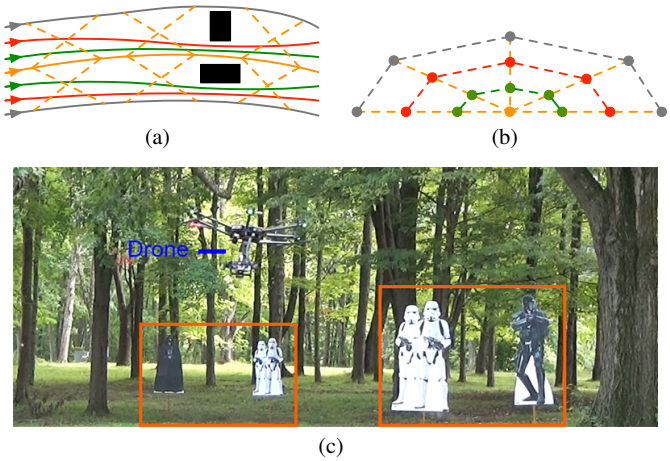


Fig. 1 A simple illustration of pre-computed alternative lanes in (a) a 2D case and (b) a 3D case. (a) is in top-down view. The yellow solid curve represents the main lane. The green, red, and gray solid curves are alternative lanes at three different levels. The yellow dashed curves are crossways connecting lanes at different levels. The paths are generated based on a prior map, therefore do not collide with structures on the map, represented by the two black rectangles. (b) is in front view. The yellow dot indicates the main lane. The green, red, and gray dots are alternative lanes. Same with (a), the yellow dashed curves are crossways. The green, red, and gray dashed curves are beltways connecting lanes at the same level. (c) 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 obstacles labeled with the orange rectangles which do not exist on the map but are only detected by onboard perception sensors. More details regarding the experiment are available in Section 5.2, Test 1.

stacles. This method has been used successfully in aerial navigation but still requires considerable computation. Here, we propose a method that reduces computational complexity considerably such that it can 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 perception sensors, we pre-compute and carefully arrange a set of alternative paths before the flight. Any prior map information is used to prune the set of alternative paths. During the navigation, the vehicle does not generate a path but only chooses a pre-computed path to follow. These pre-computed paths include a default path connecting from start to goal, namely the "main lane", a number of alternative paths to the default path, namely "alternative lanes", and short path segments for transitions among the main lane and alternative lanes, namely "crossways" and "beltways" (see an example in Fig. 1(a)-(b)). In such a method, the vehicle does not need to rely on the entire perception sensor data to update the graph but uses only part of the data for collision check on the pre-computed paths. This significantly reduces the onboard computation. The resulting online system consumes $< 3\%$ of a single CPU thread executed on a modern embedded computer.

One key insight is that the pre-computed paths function as a summary of the map – these paths avoid structures on the map. The pre-computed paths gather abundant information from the map in comparison to the existing planners. This heavily releases the burden of online processing. Data from onboard perception sensors is only for checking obstacle existence along the paths. This also lowers the requirement for the data density to carry out the path selection reliably. Another key insight is that all alternative lanes lead to the goal. This way, the task is simplified where the vehicle does not need to search for a path that ends at the goal – with any collision-free path selected from the pre-computed paths, the vehicle will reach the goal by simply following the path.

To the best of our knowledge, the demonstrated ability of aerial collision avoidance has not been achieved by existing methods as in our experiment video[1].

2 Related Work

Our work is most related to literature in path planning and collision avoidance with an emphasis on robot navigation. Given a traversable representation of the environment, graph search-based methods such as Dijkstra [1], A* [2], and D* [3] algorithms tessellate the space into nodes connected by links, and traverse different nodes to search for paths. Sampling-based methods cover the space with random samples. Paths are generated by connecting selected samples. Contemporary sampling-based methods such as Rapidly-exploring Random Tree and its variants [4, 5] can handle maps in a large scale, generating paths in a relatively short amount of time.

Certain path planning methods pre-process a map to extract traversable information based on a particular representation. Such a representation facilitates the path search. For example, Probabilistic Roadmap (PRM) [6] based methods randomly sample on the map to create a connectivity graph. Paths are then found by searching on the graph. Other examples include Voronoi graph [7] and vector field [8]. In comparison, these methods share the same insight with the proposed method that all pre-process a map and summarize it into a certain representation. However, a key difference is that the summarized representation in our method is for a single pair of start and goal, while methods such as PRM still need to traverse the graph to find a path ending at the goal. The result is that our navigation problem is simplified since finding any collision-free path can bring the vehicle to the goal.

For autonomous on-road driving, various methods [9] pre-compute paths to indicate traffic lanes. The paths and their intersections form a graph of the road network. In these methods, the pre-computed paths are arranged based on existing roads. Since roads are supposed to be collision-free by default, the paths are not to avoid structures left on the roads but only represent the connectivity of the roads.

The contribution of the paper is in separation of path generation from onboard computation, by offline pre-computing a set of alternative lanes. This way, the onboard computation is reduced to the minimum. The method is adaptable to vehicles and applications with limited onboard processing power. Further, our previous

[1] Experiment video: `https://youtu.be/pR7Jeq9wCVU`

work [10] used short alternative paths to enable navigation and collision avoidance. A major improvement in this paper is to use long alternative lanes with crossways and beltways for lane switching. The proposed method is proven to have a significantly lower probability of full navigation blockage, and uses a much fewer number of pre-computed paths in comparison to our previous method.

3 Problem Definition

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

- Define $\mathscr{Q} \subset \mathbb{R}$ as the configuration space of a vehicle, and $\mathscr{Q}_{\text{occu}} \subset \mathscr{Q}$ as the occupied subspace based on the prior map. $\mathscr{Q}_{\text{occu}}$ is untraversable. The traversable space is defined as $\mathscr{Q}_{\text{trav}} = \mathscr{Q} \backslash \mathscr{Q}_{\text{occu}}$. Let $A \in \mathscr{Q}_{\text{trav}}$ and $B \in \mathscr{Q}_{\text{trav}}$ be the navigation start and end points.
- Define a path named main lane connecting from A to B. The main lane is at level 0, denoted as $l_0 \in \mathscr{Q}_{\text{trav}}$.
- Define alternative lanes to the main lane as paths connecting from A to B. The alternative lanes adjacent to the main lane are at level 1. Away from the main lane, the alternative lanes adjacent to level $i \in \mathbb{Z}^+$ lanes are at level $i+1$. An alternative lane at level i is denoted as $l_i^j \in \mathscr{Q}_{\text{trav}}$, where $j \in \mathbb{Z}^+$ is the index at level i.
- Define crossways as paths connecting the main lane l_0 and alternative lanes l_i^j sharing the same lane index at different levels (e.g. lanes connected to a yellow dashed curve in Fig. 1(b)). A crossway can start and end at any level, denoted as $c_{(i,j)}^k \in \mathscr{Q}_{\text{trav}}$, where i is the level that $c_{(i,j)}^k$ starts from, j the lane index that $c_{(i,j)}^k$ connects to, and $k \in \mathbb{Z}^+$ is the crossway index given i and j.
- Define beltways as paths connecting alternative lanes l_i^j at the same level. A beltway is denoted as $b_{(i,j)}^k \in \mathscr{Q}_{\text{trav}}$, where i and j are the level and lane index that $b_{(i,j)}^k$ starts from, and k is the beltway index given i and j.
- Define vertices as intersections between lanes and crossways or beltways. A vertex is denoted as $v_{(i,j)}^k \in \mathscr{Q}_{\text{trav}}$, where i and j are the level and lane index that $v_{(i,j)}^k$ locates on, and k is the vertex index given i and j. Note that all vertices are on lanes. That says, crossways and beltways can only intersect at connections to lanes.
- A path set $\mathscr{G} = \{l_0, l_i^j, c_{(i,j)}^k, b_{(i,j)}^k, v_{(i,j)}^k\}$, $i, j, k \in \mathbb{Z}^+$, consists of all aforementioned paths and vertices.

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

4 Method

4.1 Online Algorithm

The navigation and path selection algorithm is implemented based on the Dijkstra's algorithm [11]. Given a path set $\mathscr{G}$, let $v, v' \in \mathscr{G}$ be two adjacent vertices – v and v' are connected by a path in $\mathscr{G}$ with no other vertex in between. Let us use $\mathscr{N}(v)$ to denote the set of adjacent vertices of v, $v' \in \mathscr{N}(v)$ and vice versa. Our algorithm uses a combination of two costs in determining a path. For an edge connecting two adjacent vertices v and v', a distance cost, $c_d(v,v')$, is based on the distance from the edge to the closest surrounding object,

$$c_d(v,v') = \int_v^{v'} \max\{D - d(\delta), 0\} d\delta, \tag{1}$$

where

$$d(\delta) = \min\{d_{\text{stru}}(\delta), d_{\text{obst}}(\delta)\}.$$

Here, $d_{\text{stru}}(\delta)$ defines the distance from each point on the edge to the closest structure on the map, pre-computed during the path generation, $d_{\text{obst}}(\delta)$ denotes the distance to the closest online discovered obstacle, δ is the length along the edge starting from v, and D is a pre-defined safety distance threshold. In (1), $d(\delta)$ is assigned $d_{\text{stru}}(\delta)$ or $d_{\text{obst}}(\delta)$ depending on which one is closer, and contributes to $c_d(v,v')$ only if $d(\delta) < D$. In other words, if a path is further than D away from all surrounding objects, there is no penalty for traveling through the path. The distance cost $c_d(v,v')$ uses an integration over the length of the edge, therefore if $d(\delta) < D$ along the path, the further the vehicle travels, the higher the cost is.

The second cost comes from that the vehicle switches from one lane to another. We prefer the vehicle to stay on the same lane instead of switching between lanes frequently. To this end, a switching cost, $c_s(v,v')$, is designed. An edge on a lane has a zero switching cost, an edge on a crossway or beltway has a non-zero switching cost. This means that passing through a crossway or beltway for lane switching is penalized. The cost of an edge, $c(v,v')$, is the weighted sum,

$$c(v,v') = c_d(v,v') + w_s c_s(v,v'), \tag{2}$$

where w_s is the weight. We evaluate the vehicle being away from surrounding objects to be much more important than staying on the same lane, hence we have $1 >> w_s > 0$.

With the costs defined, our navigation and path selection algorithm solves an optimization problem that minimizes the accumulated cost along the path.

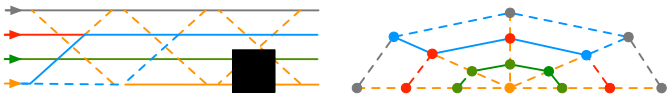
Problem 1 *Given a path set $\mathscr{G}$, compute a path l^* connecting adjacent vertices in $\mathscr{G}$ by minimizing the cost,*

$$l^* = \arg\min \sum_{v,v' \in l,\ v' \in \mathscr{N}(v)} c(v,v'). \tag{3}$$

Algorithm 1 solves the problem. The algorithm takes as inputs a path set $\mathscr{G}$, a set $\mathscr{V} \subset \mathscr{G}$ containing vertices within the perception range based on the current vehicle pose, a set $\mathscr{V}_{\text{end}} \subset \mathscr{V}$ consists of vertices on the boundary of the perception range, a set $\mathscr{E}_{\text{occl}}$ composed of occluded edges based on the current perception sensor data, and the current path l_c. The algorithm outputs the vehicle navigation command. Upon the navigation starts, the vehicle is at the start point A and l_c is set to the main lane l_0. At the beginning of each function call, the algorithm initializes a priority queue, $\mathscr{Q}$, with the first vertex on l_c ahead of the vehicle, v_0, on line 7. The algorithm propagates through all accessible vertices in $\mathscr{V}$ by processing vertices in $\mathscr{Q}$ with the lowest cost and pushing unvisited vertices into $\mathscr{Q}$, on lines 8-20. Once a vertex is pushed into $\mathscr{Q}$, it is labeled as visited, and once a vertex is processed, it is removed from $\mathscr{Q}$. Through the propagation, the algorithm updates the costs of the vertices on line 12, and the pointers to the previous vertices on line 13. After the propagation finishes, the algorithm checks each vertex in $\mathscr{V}_{\text{end}}$, on lines 21-26. If the end point $B \in \mathscr{V}_{\text{end}}$ and B is unvisited, or all vertices in $\mathscr{V}_{\text{end}}$ are unvisited, meaning a full blockage is found, the algorithm reports navigation unsuccessful, on line 22. Otherwise, the algorithm chooses B if $B \in \mathscr{V}_{\text{end}}$, or a vertex in $\mathscr{V}_{\text{end}}$ with the lowest cost if $B \notin \mathscr{V}_{\text{end}}$, back-tracks to v_0 from the vertex to determine a path, and updates l_c to the path. The above process recurs until B is reached, and the algorithm reports navigation successful, on line 30.

We have certain preferences for the path selection. As shown in Fig. 2(a), when an obstacle is discovered, we prefer the vehicle to avoid early by switching to another lane, keeping the safety margin high. This is realized by a priority check on line 11. The priority check compares two vertices $\{v, v'\}$ and prefers v' to be on the same lane with v as the first choice, v' at the same level with v as the second choice, v' at a lower level than v as the third choice, and v at a higher level than v as the last choice. The result is that the algorithm selects paths switched early and kept on the same lane further before the obstacle. Also, the algorithm selects paths that avoid obstacles by taking lanes at a lower level than a higher level, being closer to the main lane l_0. The priority check is employed again on line 25 in choosing a vertex in $\mathscr{V}_{\text{end}}$ as the end of the path. The algorithm compares vertices with the same lowest cost based on the priority check to determine the vertex.

Further, when switching between lanes, we prefer the vehicle to take beltways at a lower level then a higher level, as illustrated in Fig. 2(b). This is by adjusting



(a) (b)

Fig. 2 Two examples of preferred paths. In both examples, the blue solid curves are the preferred paths as opposed to the blue dashed curves. In (a), when an obstacle is discovered, we prefer the path to switch early keeping the safety margin high. In (b), when switching between the two blue vertices, we prefer the vehicle to take a beltway at a lower level than a higher level, being closer to the main lane l_0 as that is the default lane to follow.

the switch costs to be minimally different. As discussed, edges on lanes have zero switch costs. Edges on crossways have the minimum non-zero costs. Edges on beltways have higher costs than crossways, and the higher the level, the higher the cost. Note that the differences among the switch costs are significantly smaller than the switch costs themselves such that accumulating the differences along a path has a minor effect on selecting the lanes but only the crossways and beltways.

Let us analyze the computational complexity of Algorithm 1. Denote R as the perception range. Recall $\mathscr{V}$ is the set of vertices within R. Denote $\mathscr{E}$ as the set of edges connecting adjacent vertices in $\mathscr{V}$. The Dijkstra's algorithm runs in $O((|\mathscr{V}|+|\mathscr{E}|)\log|\mathscr{V}|)$ time when implemented with a priority queue. In our setup,

Algorithm 1: Navigation and Path Selection

input: a path set $\mathscr{G}$ connecting between A and B, sets of vertices $\mathscr{V}, \mathscr{V}_{\text{end}} \subset \mathscr{G}$, set of occluded edges $\mathscr{E}_{\text{occl}}$, current path l_c;
output: navigation command;
begin
  while B is not approached **do**
    if l_c consists of any edge $e \in \mathscr{E}_{\text{occl}}$ or $e \notin l_0$ **then**
      For each $v \in \mathscr{V}$, label v as unvisited, $c(v) \leftarrow \infty$;
      Find the first vertex on l_c ahead of the vehicle as v_0, label v_0 as visited, $c(v_0) \leftarrow 0$, $\mathscr{Q} \leftarrow \{v_0\}$;
      while $\mathscr{Q} = \emptyset$ **do**
        Find $v \in \mathscr{Q}$ with the lowest cost, remove v from $\mathscr{Q}$;
        for each $v' \in \mathscr{V} \cap \mathscr{N}(v)$ and $(v,v') \notin \mathscr{E}_{\text{occl}}$ **do**
          if $c(v') > c(v)+c(v,v')$ **or** ($c(v') = c(v)+c(v,v')$ **and** $\{v,v'\}$ passes a priority check) **then**
            $c(v') \leftarrow c(v)+c(v,v')$;
            $p(v') \leftarrow v$;
            if v' is unvisited **then**
              Label v' as visited;
              Push v' into $\mathscr{Q}$;
            end
          end
        end
      end
      if ($B \in \mathscr{V}_{\text{end}}$ **and** B is unvisited) **or** ($\forall v \in \mathscr{V}_{\text{end}}$, v is unvisited) **then**
        Finish and report navigation unsuccessful;
      end
      else
        If $B \in \mathscr{V}_{\text{end}}$, $v'' \leftarrow B$, otherwise, find all $v \in \mathscr{V}_{\text{end}}$ sharing the lowest cost, choose v'' among those so that $\{v_0, v''\}$ passes a priority check, back-track to v_0 by recurring $v'' \leftarrow p(v'')$, then update l_c;
      end
    end
    Navigate the vehicle on l_c;
  end
  Finish and report navigation successful;
end

each vertex on an alternative lane $l_i^j \in \mathscr{G}$, $i, j \in \mathbb{Z}^+$, connects to at most a lane, a crossway, and a beltway at the same time. All edges in $\mathscr{E}$ are connected to vertices on alternative lanes. Therefore, we have $O(|\mathscr{E}|) = O(|\mathscr{V}|)$. The algorithm checks if an edge is occluded (edge $\in \mathscr{E}_{\text{occl}}$) on lines 5 and 10. This step is conducted by keeping a full list of the edges in $\mathscr{E}$. A flag is associated with each edge to indicate occlusion. Hence, checking each edge takes $O(1)$ time. Before running Algorithm 1, the system performs collision check for all edges in $\mathscr{E}$ using the current perception sensor data. Accelerated by a voxel grid implementation (more details in Section 5.2), each edge takes $O(1)$ time. The overall time for collision check is in $O(|\mathscr{E}|) = O(|\mathscr{V}|)$. Therefore, the computational complexity can be stated.

Theorem 1 *Algorithm 1 runs in $O((\mathscr{V}|\log|\mathscr{V}|)$ time, where $\mathscr{V} \subset \mathscr{G}$ is the set of vertices within the perception range R. Collision check consumes $O(|\mathscr{V}|)$ time.*

4.2 Path Generation

In our previous work [10], the state-of-the-art BIT* path planner [4] is used to generate the main path connecting from start point *A* to end point *B*. The resulting path is further smoothed to meet our requirement for high-speed flights. In this paper, we retain the same method for generation of the main lane, which is considered a solved problem. The contribution of this paper is using pre-computed alternative lanes to enable navigation and collision avoidance.

Consider the main lane is given, the alternative lanes are generated by solving an optimization problem. The optimization utilizes a number of key-points on the alternative lanes. As shown in Fig. 3(a), the green dots represent the key-points on an alternative lane at level 1 (green line). The alternative lane is initialized with a constant lateral interval to the main lane. Then, a collision check is executed. If occlusion is found on the alternative lane, the corresponding key-points are shifted in one out of two directions, toward or away from the main lane as indicated by the silver arrows. The direction w.r.t. each structure is set randomly through multiple iterations considering all structures occluding the alternative lane. The set of directions with the minimum accumulated distance of key-point movements are chosen. Here, moving toward the main lane is decided. The optimization starts thereafter which adjusts the key-points to the light green dots. A spline curve is then fit through the key-points to generate a path.

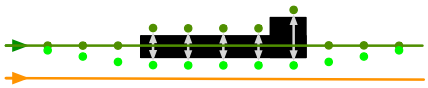


Fig. 3 Generation of alternative lanes. The yellow line represents the main lane, the green line presents an alternative lane at level 1, and the green dots are key-points. The alternative lane is initialized with a constant lateral interval to the main lane. Then, a collision check is run. If the alternative lane intersects with structures on the map as represented by the black region, the corresponding key-points can be shifted in one out of two directions, toward or away from the main lane as indicated by the silver arrows. The directions w.r.t. the structures are chosen in a random and iterative process. An optimization follows which adjusts the key-points to the light green dots. A spline curve is then fit through the key-points to generate a path.

The optimization takes into account the curvature on the alternative lanes, distances to structures on the prior map, and lateral intervals between adjacent lanes. Define a distance cost, $c_d(l_i^j)$, for an alternative line $l_i^j \in \mathscr{G}$, $i, j \in \mathbb{Z}^+$ as,

$$c_d(l_i^j) = \int_A^B \max\{D - d_{\text{stru}}(\delta), 0\} d\delta, \tag{4}$$

where $d_{\text{stru}}(\delta)$ is the distance from each point on l_i^j to the closest structure, δ is the length along the lane starting from A, and D the safety distance threshold. Similar to (1), $c_d(l_i^j)$ is only effective when the lane is closer than D to a structure. Denote r as the radius of the vehicle. We require $d_{\text{stru}}(\delta) \geq r$ through the lane. Define a curvature cost, $c_c(l_i^j)$, for l_i^j,

$$c_c(l_i^j) = \int_A^B \kappa(\delta) d\delta, \tag{5}$$

where $\kappa(\delta)$ is the curvature associated with each point on l_i^j. Next, to maintain a relatively constant lateral interval between adjacent lanes, we define another cost. Let $\mathscr{P}$ be the set of pairwise lanes in $\mathscr{G}$ that are adjacent, $\mathscr{P} = \{(l, l') | l, l' \in \mathscr{G}$ and l, l' are adjacent$\}$. A lateral interval cost, $c_l(l, l')$, for an adjacent pair of lanes $(l, l') \in \mathscr{P}$ is defined as,

$$c_l(l, l') = \int_A^B |H - h(\delta)| d\delta, \tag{6}$$

where $h(\delta)$ is the lateral interval between l and l' for each point on l, and H is the desired lateral interval. The optimization problem is to minimize a joint cost combining all three costs defined above. Here, note that each alternative lane starts at A and ends at B. Denote $S(l_i^j)$ and $E(l_i^j)$ as the start point and end point of l_i^j, $S(l_i^j) = A$ and $E(l_i^j) = B$. The optimization computes the alternative lanes in a path set $\mathscr{G}$ as stated in Problem 2. The problem is solved by the Levenberg-Marquardt method [12] through iterations.

Problem 2 *Given a path set $\mathscr{G}$, adjust key-points on each alternative lane in $\mathscr{G}$ to minimize the following cost,*

$$\min \sum_{\{l, l'\} \in \mathscr{P}} c_l(l, l') + \sum_{l_i^j \in \mathscr{G}} (w_d c_d(l_i^j) + w_c c_c(l_i^j)), \tag{7}$$

subject to $S(l_i^j) = A$, $E(l_i^j) = B$ where $l_i^j \in \mathscr{G}$, $i, j \in \mathbb{Z}^+$.

With the alternative lanes computed, the method places crossways and beltways. A collision check is run for each placement. If a crossway or a beltway intersects with structures on the map, the corresponding edges are removed. In addition, the method uses segmented paths for smooth transitions between lanes and crossways or beltways. These paths are generated online based on template spline curves.

5 Experiments

5.1 Simulation

We test the proposed method in simulation using alternative lanes in a 2D case. As shown in Fig. 4, the alternative lanes are separated with 5m intervals. Obstacles are defined as 5m squares. Through the test, 1-10 obstacles are placed along the paths to introduce blockage, from simple to complicated scenarios. It is worth to mention that In Fig. 4(e) and Fig. 4(f), we compare the proposed method with our previous method based on short alternative paths [10]. We use a sequence of obstacles to block the main lane. In Fig. 4(e), the proposed method simply switches to an alternative lane (green line) and keeps on that lane to avoid the obstacles. In Fig. 4(f), however, the previous method recursively takes alternative paths at higher levels and eventually finds no collision-free path. Note that the problem in Fig. 4(f) is partially caused by that all alternative paths end on the main path. If organized differently where alternative paths at level 2 and above end on alternative paths at one level lower, the vehicle will switch among alternative paths at different levels to handle such a case. Nevertheless, the vehicle will behave unnaturally curing left and right.

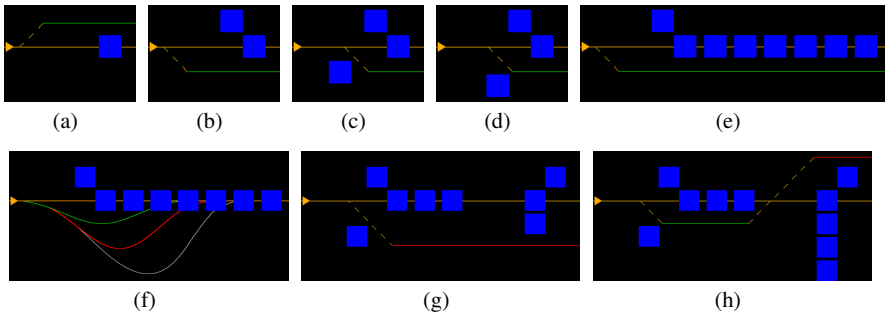


Fig. 4 Simulation results. The test is based on a 2D path set. The main lane is in yellow, from left to right. Alternative lanes at level 1 and level 2 are in green and red, respectively. Crossways are yellow dashed lines. The lanes are separated with 5m intervals. Obstacles are 5m squares placed on the paths to introduce blockage. In (a), one obstacle is used and the vehicle chooses an alternative lane at level 1 to avoid. In (b), two obstacles are used, one blocking the main lane and the other blocking the alternative lane on one side of the main lane. The vehicle chooses an alternative lane on the other side. In (c), three obstacles block the main lane and its both sides. The vehicle switches to the alternative lane below the main lane after the obstacle. In (d), the three obstacles are retained and the one on the bottom is moved away from the main lane. This obstacle does not block the alternative lane as in (c) but stays close to it. The vehicle still switches to the alternative lane after the obstacle because of the effect of the distance cost. Further, in (e) and (f), we compare the proposed method with our previous method which uses short alternative paths to realize collision avoidance [10]. A sequence of obstacles are placed on the main lane. With the proposed method, in (e), the vehicle simply switches to an alternative lane and keeps on that lane to avoid these obstacle. With the previous method, in (f), however, the vehicle is recursively forced to take alternative paths at higher levels and eventually finds no collision-free path. In (g) and (h), we employ more complicated displacements of obstacles. The vehicle chooses an alternative lane at level 2 in (g), and two alternative lanes at level 1 and level 2 on different sides of the main lane in (h).

5.2 UAV Experiments

Our experimental platform is shown in Fig. 5. 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 onboard processing. The state estimation is based on our previous work [10], which integrates 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 flight test.

We report on two flight tests. Test 1 is in a cluttered forest environment. As shown in Fig. 6, the pre-computed paths consist of a main lane, alternative lanes at two levels, and associated crossways, based on a prior 3D laser map. The map is built by hand-holding the sensor-computer pack and walking in the environment. The test has two separate runs, with first a clear path and then artificial 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 lane to the end. Then, with artificial obstacles, the vehicle switches among pre-computed lanes to avoid the obstacles.

Test 2 is on an inactive industrial site. As shown in Fig. 7, the test involves a main lane and two levels of alternative lanes. Crossways and beltways are used for lane switching. There are totally 11 lanes, 1234 crossways, and 551 beltways. Three obstacles are placed on the path – an insistent canopy, a tree, and a wire. The vehicle avoids each obstacle by switching from one lane to another. Fig. 7(c)-(e) compare the switched paths to our previous method that uses short alternative paths for collision avoidance [10]. The yellow curve is the main lane and the blue curve is the executed path. The coordinate frame represents the vehicle. As we see, with the proposed method, the path switch takes place earlier as soon as the obstacle is discovered, while the obstacle is further ahead. This leaves a higher safety margin. In comparison, the previous method does not switch path until the obstacle is close. This is due to the usage of short alternative paths – even the vehicle sees a distanced obstacle, only an alternative path starting close to the obstacle is taken. Further, the previous method uses a main path, and 597 and 8101 alternative paths at levels 1 and 2, with 8699 paths in total. The number of paths increases exponentially w.r.t.

Fig. 5 UAV experimental platform. A DJI Matrice 600 Pro aircraft carries our sensor-computer pack on a 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 MEMS-based IMU. An i7 embedded computer carries out all onboard processing. Note that GPS data is unused in all tests.

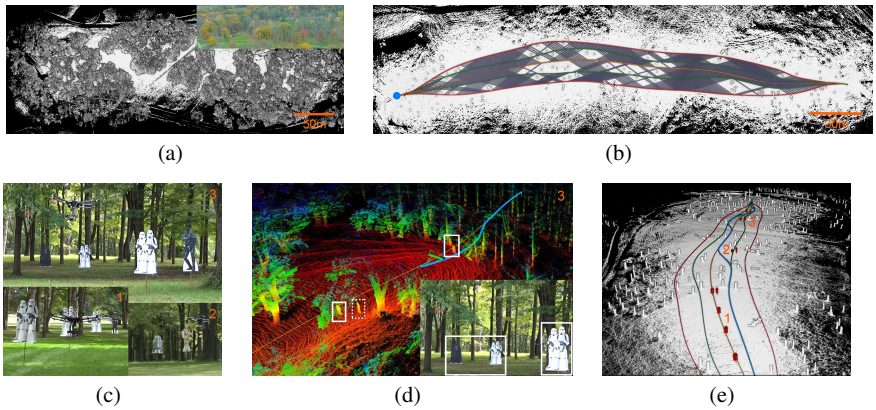


Fig. 6 Result of Test 1. (a) shows a 3D laser map used as the prior map. An aerial image from the same area is shown at the top-right corner. (b) shows the pre-computed paths overlaid on the prior map where the canopy of the trees is cropped. The main lane is in yellow. Two levels of alternative lanes are in green and red, respectively. Crossways are in slate blue. The test contains two separate runs, both at 10m/s. First, no obstacle is present and the vehicle follows the main lane to the end. Then, multiple artificial obstacles are placed on the path as shown in (c). The vehicle avoids the obstacles by switching among pre-computed lanes. (d) shows online registered laser scan data while the vehicle is avoiding the obstacles labeled with the white rectangles. The obstacle in the dotted white rectangle is out of the camera field of view and not present in the image at the bottom-right corner. The yellow curve is the main lane and the blue curve is the executed path. The coordinate frame represents the vehicle. (e) shows the complete executed path for the second run in blue and the obstacles in dark red. The numbers 1-3 correspond to the numbers in (c) and (d) illustrating the obstacle locations w.r.t. the prior map.

the level. The proposed method uses 1796 paths and the number of paths is constant at each level.

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. 8, the rate of navigation failure or full navigation blockage increases w.r.t. the number of obstacles. Employing more alternative lanes (2 levels instead of 1 level) helps reduce the rate of navigation failure to a large extent. When compared to the previous method, the proposed method constantly produces a significantly lower rate of navigation failure. These results validate the advantage of using long alternative lanes for collision avoidance. Further, we compare to the BIT* path planner [4], which uses the randomly generated obstacles combined with the online registered laser scan data to find paths along the main lane. The failure criterion is set as not being able to find a collision-free path within 1s. We can see that both of our methods result in lower rate of failure.

Finally, let us inspect some metrics for the collision avoidance. Collision check is implemented with a voxel grid at 0.4m resolution overlaid on the prior map. Correspondences are pre-established from the voxels to the corresponding occluded paths. During the navigation, we index the voxel for each laser point to find the correspondences and hence determine the path occlusions. As shown in Table 1, the process

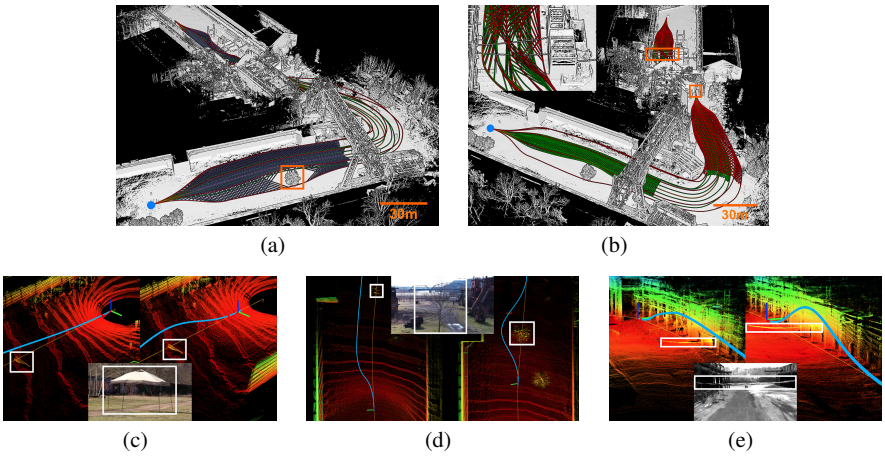


Fig. 7 Result of Test 2 conducted on an inactive industrial site. (a) shows the prior 3D laser map with the main lane (yellow), two levels of alternative lanes (green and red), and crossways (slate blue). (b) shows the same prior map with beltways (green and red). The paths avoid a tree as labeled with the orange rectangle in (a), pass through a narrow area and avoid two wires as labeled with the orange rectangles in (b). A close view of the paths around the wires is at the top-left corner in (b). The crossways and beltways in some sections are removed for a better view. (b)-(d) show the avoidance of three obstacles – an insistent canopy, a tree, and a wire left on the path in addition to those labeled with the orange rectangles. On the left side of each figure, we draw the executed path (blue) and the main lane (yellow) by the proposed method. The coordinate frame represents the vehicle. On the right side, we compare to the previous method using short alternative paths. We can see that with the proposed method, the path switches earlier while the obstacle is further ahead, leaving a higher safety margin. With the previous method, the path switch does not take place until the obstacle is close.

of collision check takes 33.7μs at most for each laser scan (5Hz). Execution of Algorithm 1 for path selection takes another 362.1μs at most, resulting in 395.8μs of overall online processing. This is comparable to our previous method. Further, the BIT* path planner [4], known for the computational speed, takes 200ms to gener-

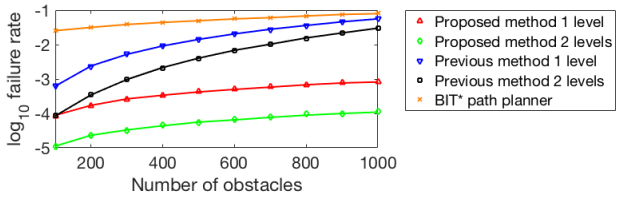


Fig. 8 Further evaluation using the setup in Test 2. Obstacles are modeled as 1m cubes randomly distributed in the 3D space. As we see, the rate of navigation failure or full navigation blockage increases w.r.t. the number of obstacles. Using 2 levels of alternative lanes instead of 1 level reduces the rate of navigation failure to a large extent. Compared to the previous method, the proposed method constantly produces a significantly lower rate of failure. Also, we compare to the BIT* planner [4] using the randomly generated obstacles combined with the online laser scan data.

Table 1 Online and offline computation time for Tests 1 and 2

	Collision check		Path selection		Offline
Test	Mean	Worst	Mean	Worst	path generation
1	13.5μs	29.1μs	236.1μs	253.0μs	2.1 minutes
2	16.2μs	33.7μs	348.4μs	362.1μs	3.4 minutes

ate an acceptably smooth path with the same datasets. The worst case is over 1s to produce the very first path.

6 Conclusion

The paper proposes a novel method which utilizes offline generated alternative lanes to enable robot navigation. The alternative lanes are computed based on a prior map and organized at different levels. Collision avoidance is conducted by switching among the pre-computed lanes, through crossways and beltways. The method eliminates the necessity of online path generation, migrating majority of the computation to offline processing. The resulting system consumes $< 3\%$ of a single CPU thread on a modern embedded computer. It makes possible for a lightweight UAV to maneuver aggressively in a cluttered forest environment, at a constant speed of 10m/s.

References

1. 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.
2. 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.
3. 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.
4. 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.
5. 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.
6. D. Hsu, J.-C. Latombe, and H. Kurniawati, "On the probabilistic foundations of probabilistic roadmap planning," *The Intl. J. of Robotics Research*, vol. 25, no. 7, pp. 627–643, 2006.
7. 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.
8. 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*, June 2016.
9. B. Paden, M. Cap, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 33–55, 2016.
10. J. Zhang, R. G. Chadha, V. Velivela, and S. Singh, "P-CAP: Pre-computed alternative paths to enable aggressive aerial maneuvers in cluttered environments," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Madrid, Spain, Oct. 2018.
11. S. LaValle, *Planning Algorithms*. New York, NY, USA: Cambridge University Press, 2006.
12. D. Bertsekas, *Nonlinear Programming*. Cambridge, MA, 1999.