Received: 30 October 2019 | Revised: 5 February 2020 | Accepted: 4 April 2020

DOI: 10.1002/rob.21952

REGULAR ARTICLE

WILEY

Falco: Fast likelihood-based collision avoidance with extension to human-guided navigation

Ji Zhang[1] | Chen Hu[1] | Rushat Gupta Chadha[2] | Sanjiv Singh[1,2]

[1]Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania

[2]Near Earth Autonomy, Pittsburgh, Pennsylvania

Correspondence
Ji Zhang, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213.
Email: zhangji@cmu.edu

Abstract

We propose a planning method to enable fast autonomous flight in cluttered environments. Typically, autonomous navigation through a complex environment requires a continuous search on a graph generated by a *k*-connected grid or a probabilistic scheme. As the vehicle travels, updating the graph with data from onboard sensors is expensive as is the search on the graph especially if the paths must be kinodynamically feasible. We propose to avoid the online search to reduce the computational complexity. Our method models the environment differently in two separate regions. Obstacles are considered to be deterministically known within the sensor range and probabilistically known beyond the sensor range. Instead of searching for the path with the lowest cost (typically the shortest path), the method maximizes the likelihood to reach the goal in determining the immediate next step for navigation. With such a problem formulation, the online method realized by a trajectory library can determine a path within 0.2–0.3 ms using a single central processing unit thread on a modem embedded computer. The method supports two configurations working with and without a prior map. Both configurations can be used to plan toward a goal point. Further, the later can allow human guidance for the navigation through a directional input. In experiments, it enables a lightweight unmanned aerial vehicle to fly at 10 m/s in a cluttered forest environment (see Figure 1 as an example).

KEYWORDS

aerial robotics, obstacle avoidance, planning

1 | INTRODUCTION

The paper aims to solve a path planning problem to enable fast autonomous flight in complex environments. The problem remains challenging because planning paths to avoid obstacles discovered by onboard sensors requires creating and updating a representation of the environment that can be searched for kinodynamically feasible paths. The process is computationally expensive. Since computational resources available for lightweight aerial vehicles are limited, we need a method that can guide an aerial vehicle with low computational complexity. A typical way is to use a hierarchical approach that separates the planning problem into two subproblems. The first problem solves a global planning problem possibly assisted by a heuristic to ensure the path does not fall into local minima. A second problem solves a local planning problem that runs in parallel to track the global path as well as avoid obstacles. This method has been used successfully in autonomous navigation (Droeschel et al., 2016; Gonzlez, Prez, Milans, & Nashashibi, 2016; Scherer, Singh, & Chamberlain, 2008) but still requires considerable computation. In this paper, we propose a method that reduces the computational complexity considerably such that it can ensure safe flight using very lightweight computation onboard the aerial vehicle (Figure 1).

The key idea to make the low computational complexity possible is avoiding the online search. Instead of searching a graph that is

FIGURE 1 A photo from a flight experiment where our method enables a lightweight aerial vehicle to maneuver at 10 m/s in a cluttered forest environment. More details regarding the experiment are in Section 5.2 [Color figure can be viewed at wileyonlinelibrary.com]

continuously being updated by onboard sensors, we formulate the planning problem from a likelihood point of view. The method does not seek the path with the lowest cost (typically the shortest path) but maximizes the probability to reach the goal in determining the immediate next step for execution of the navigation. This is through modeling of the configuration space differently in two separate regions. Obstacles are considered to be deterministically known within the sensor range as they are perceived by onboard sensors, and probabilistically known beyond the sensor range as they are from a prior map. A trajectory library is used to bridge the probabilities across the sensor range, where the trajectories are separated into groups. During the navigation, the method evaluates each of the groups to determine the path.

Solving the planning problem with a probabilistic representation of the environment also associates new behaviors to the vehicle. Most of the existing methods find a single path with the lowest cost. The path can be the shortest in length but may guide the vehicle through narrow pathways leaving very few choices for the vehicle to avoid more obstacles, if more obstacles are discovered due to dynamic obstacles or environmental changes such that the prior map is outdated. The proposed method, seeking the highest probability of successful navigation, prefers opener spaces for traversal even though the resulting path can be longer, leaving more choices for obstacle avoidance during the navigation. Further, we have identified certain cases where existing methods based on deterministic representations of the environment encounter difficulty in finding feasible paths. The proposed method handles the cases (see Section 5.1 for details).

The method supports two configurations working with and without a prior map. In both configurations, the method can be used to guide the vehicle toward a goal point. In addition, the configuration without a prior can also accept human input in guiding the navigation. For example, an operator can use a joystick controller to give a directional input. The method determines a path for safe navigation taking into account the human input.

During the navigation, the method can find a path within 0.2–0.3 ms using a single central processing unit (CPU) thread on a modem embedded computer. Our experiment results are in a public video.[1]

[1]Experiment video: https://youtu.be/VYtQt2NcY0Q

2 | RELATED WORK

The proposed method is most related to the literature in path planning and collision avoidance. The problem involves solving for a path for a vehicle to travel from start to goal given a representation of the environment. Graph search-based methods, such as Dijkstra (Kala & Warwick, 2013), A* (MacAllister, Butzke, Kushleyev, Pandey, & Likhachev, 2013), and D* (Rufli & Siegwart, 2009) 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; LaValle, 2006) and its variants (Akgun & Stilman, 2011; Gammell, Srinivasa, & Barfoot, 2014, 2015; Karaman & Frazzoli, 2011; Kuffner & LaValle, 2019; Otte & Correll, 2013), have shown promising results to handle maps in large scales, generating paths in a relatively short amount of time. However, these methods require updating the graph and searching the graph continuously during the navigation. The computational complexity can be excessive if the environment is cluttered and complex. Finding a path is not guaranteed within a fixed amount of time.

Certain planning methods preprocess a map to extract traversable information as a means to facilitates the online search. For example, Probabilistic Roadmap (PRM; Hsu, Latombe, & Kurniawati, 2006; Kavraki, Kolountzakis, & Latombe, 1998) based methods randomly sample on the map to create a connectivity graph. Paths are then found by searching the graph. Other examples include Voronoi graph (Beeson, Jong, & Kuipers, 2005) and vector field (Pereira, Choudhury, & Scherer, 2016). In essence, these methods share the insight of moving part of the processing offline before the navigation starts to accelerate the online processing. However, the online processing still needs to traverse the graph to search for the path, and hence can be computationally expensive.

The proposed method employs a probabilistic representation of the environment. The concept of modeling uncertainty has been introduced to the path planning literature (Fraichard & Mermond, 1998). For example, the method of Van Den Berg, Abbeel, and Goldberg (2011) uses a linear-quadratic controller with Gaussian models to take into account the uncertainties of the robot motion and state. Melchior and Simmons (2007) extend RRT with particles on each node to handle the uncertainty of terrain friction. These methods involve probabilities to model the motion or state of the vehicle. Chung, Smith, Skeele, and Hollinger (2019) model the edge costs on a graph with uncertainties for graph search. Heiden, Hausman, Sukhatme, and Agha-mohammadi (2017) use probabilities to model the traversability of map voxels. The proposed method, however, models obstacles within the sensor range deterministically, and beyond the sensor range probabilistically as they are known from a prior map.

Our previous work dedicated to enabling fast autonomous flight in cluttered environments (Zhang, Chadha, Velivela, & Singh, 2018, 2019a). These methods use a prior map to preplan alternative paths offline. The online navigation chooses one of the preplanned paths to execute. This paper is an extended version of

our conference paper (Zhang, Hu, Chadha, & Singh, 2019b). The contribution is proposing a method to enable the capability of fast flight in cluttered environments without the necessity of a prior map. On the basis of a probabilistic representation of the environment, the method maximizes the likelihood of successful navigation to the goal. Further, this journal version extends the method to incorporate human-guided autonomous navigation, which is validated with both aerial and ground vehicles. To the best of our knowledge, the resulting capability of fast aerial maneuver without a prior map has not yet been demonstrated.

3 | PROBLEM DEFINITION

Define $\mathcal{Q} \subset \mathbb{R}$ as the configuration space of a vehicle. Let $A \in \mathcal{Q}$ be the vehicle current position and $B \in \mathcal{Q}$ be the goal point. The vehicle is equipped with perception sensors. Define $\mathcal{S} \subset \mathcal{Q}$ as the space covered in the range of the perception sensors, namely, sensor range. Obstacles are modeled to be deterministically known in $\mathcal{S}$ and probabilistically known in $\mathcal{Q}\backslash\mathcal{S}$. Consider the vehicle has multiple directions to choose for the immediate first step as it starts to move from A. For convenience, let us name the state of the vehicle at this step the start state, denoted as x_s. Obviously, different choices of x_s can lead to different routes. As a convention of this paper, let us define $P_B(\cdot)$ to be the probability for the vehicle to successfully reach B from a given state. The probability associated with start state x_s is $P_B(x_s)$. Our planning problem can be defined as the following:

Problem 1. Given $A, B \in \mathcal{Q}, \mathcal{S} \subset \mathcal{Q}$, and obstacles in $\mathcal{Q}$, determine start state x_s^* to maximize the probability $P_B(x_s)$,

$$x_s^* = \arg\max_{x_s} P_B(x_s). \tag{1}$$

The above problem is solved at each step as the vehicle travels along the path, that is, the vehicle maximizes the probability to reach B at every instant time during the navigation.

4 | METHOD

4.1 | Probabilistic model

The proposed method maximizes the likelihood for the vehicle to successfully travel from A to B. As stated in the problem definition, obstacles within sensor range $\mathcal{S}$ are considered to be deterministically known as the information is acquired from the perception sensors. Obstacles beyond $\mathcal{S}$ are considered to be probabilistically known if a prior map is available. Otherwise, however, the case is equivalent to no obstacle being present a priori. Figure 2 illustrates $\mathcal{S}$ as the gray area. Define $\mathcal{F} \subset \mathcal{S}$ as the sensor frontier indicated by the red solid curve. Given start state x_s, a path connects A and B as the black curve. For all possible paths, they must intersect with $\mathcal{F}$. Here,

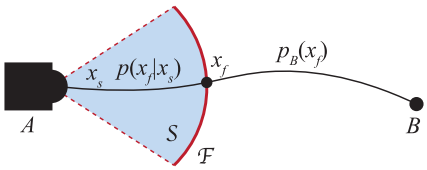

FIGURE 2 Illustration of sensor range $\mathcal{S} \subset \mathcal{Q}$ as the gray area and sensor frontier $\mathcal{F} \subset \mathcal{S}$ as the red solid curve. The black curve is a path to navigate from A to B, which starts at x_s and intersects with $\mathcal{F}$ at x_f [Color figure can be viewed at wileyonlinelibrary.com]

one can argue that the vehicle can move laterally and does not intersect with $\mathcal{F}$. In this case, due to the nature of the problem, $\mathcal{S}$ has to be expanded based on the vehicle motion model so that the vehicle does not traverse an area uncovered by $\mathcal{S}$. Define x_f as the state of the vehicle while passing $\mathcal{F}$. The conditional distribution of x_f given x_x, $p(x_f|x_s)$, can be derived from the obstacle information provided by the perception sensors. Further, the probability density for the vehicle to reach B from x_f, $p_B(x_f)$, can be obtained from the obstacles on the prior map. We have

$$p_B(x_s) = \int p_B(x_f) p(x_f|x_s)\, dx_f. \tag{2}$$

Here, notation $P_B(x_s)$ in (1) is rewritten as $p_B(x_s)$ to denote the probability density. Consider $n \in Z^+$ samples ξ_i, $i = 1, 2, ..., n$, drawn from $p(x_f|x_s)$. According to the Monte Carlo theory of sampling (Robert, 2004), we can establish

$$\int p_B(x_f) p(x_f|x_s)\, dx_f \approx_{n\uparrow\infty} \frac{1}{n}\sum_{i=1}^{n} p_B(\xi_i). \tag{3}$$

Combine (2) and (3) and consider n as a constant:

$$p_B(x_s) \approx \frac{1}{n}\sum_{i=1}^{n} p_B(\xi_i). \tag{4}$$

Equation (4) indicates that the probability density to navigate to B from x_s, $p_B(x_s)$, can be approximated by $n \gg 1$ samples drawn from the conditional distribution $p(x_f|x_s)$. Note that modeling the distribution with samples discretes the problem and leads the problem to be solved using a finite number of paths and voxels.

4.2 | Local probabilities

Given start state x_s, the vehicle can follow different paths to reach sensor frontier $\mathcal{F}$. Here, let us name a path group as the set of paths sharing the same x_s. Consider a discrete model of x_s. Figure 3 gives an example of path groups. On the top row, 7 path groups are present in top-down view, where x_s is at the start of the paths curving left or right. The path group in the middle corresponds to straight forward motion. On the bottom row, 5 path groups are shown in side view, where the paths curve upward or downward. Consider both

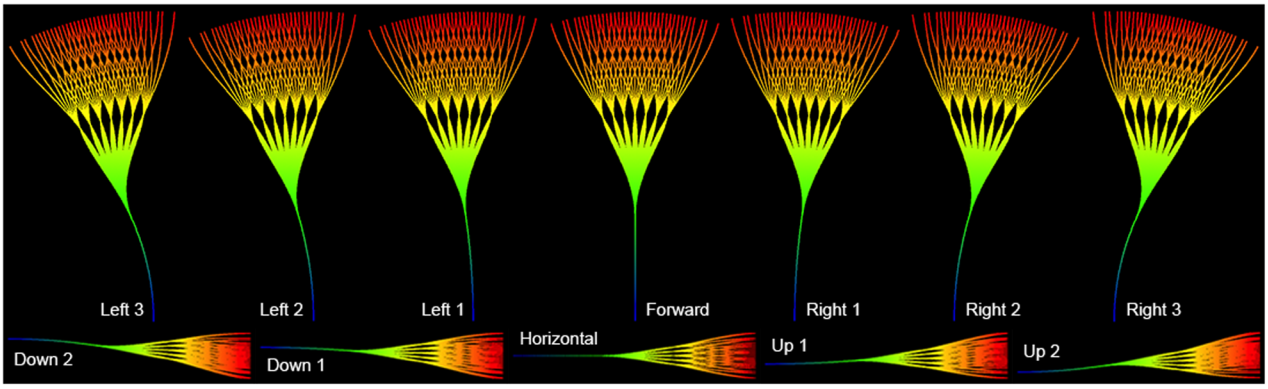

FIGURE 3 Example path groups. On the top row, we show 7 path groups curving from left to right in top-down view. On the bottom row, we show 5 path groups curving from downward to upward in side view. Consider both horizontal and vertical directions, there are $7 \times 5 = 35$ path groups [Color figure can be viewed at wileyonlinelibrary.com]

horizontal and vertical directions, there are totally $7 \times 5 = 35$ path groups in this example. All paths end on $\mathcal{F}$.

Each path is generated as a cubic spline curve. The paths in a group split in multiple directions horizontally and vertically. In Figure 3, the path first splits in 35 directions (7 horizontal and 5 vertical) and each splits in another 35 directions. This results in $35^2 = 1{,}225$ paths in a group. Consider the 35 path groups, there are $35 \times 1{,}225 = 42{,}875$ paths in total. Figure 4 shows all path groups together where color codes the group index. Note that these example paths are generated based on the vehicle motion constraints. The method, however, is not limited to a specific motion model and can support various path group configurations.

The paths in a group can be considered as viable routes from x_s to $\mathcal{F}$. The states of the paths at the ends can be viewed as samples ξ_i, $i = 1, 2, ..., n$, of x_f, where the distribution is drawn from $p(x_f|x_s)$. During the navigation, obstacles are detected by perception sensors occluding certain paths. Figure 5 gives an example of an obstacle and the corresponding collision-free paths in a group. Define a Boolean function $c(\xi_i)$ to indicate the path clearance:

$$c(\xi_i) = \begin{cases} 1, & \xi_i \text{ is unoccluded}, \\ 0, & \text{otherwise}. \end{cases} \tag{5}$$

We can compute $P_B(x_s)$ based on (4):

$$P_B(x_s) \approx \frac{\sum_{i=1}^{n} c(\xi_i) p_B(\xi_i)}{\sum_{i=1}^{n} c(\xi_i)}. \tag{6}$$

Equation (6) is applied to all path groups and x_s^* of the group with the highest $P_B(x_s)$ is chosen for the vehicle to execute.

4.3 | Global probabilities

Our environment is represented with voxels. Different from the traditional voxel representation, our voxels contain both position and orientation information. As shown in Figure 6a, a voxel is separated into multiple directions based on a constant angular interval, denoted as δ. The angular intervals set the directions from which the vehicle inters the voxel. Define $x_j^k, j, k \in Z$ as the state of the voxel, where j is the voxel index and k is the direction index. The position associated with x_j^k is modeled to be uniformly distributed within the voxel and the orientation is modeled to be uniformly distributed within $[-\delta/2 \delta/2]$ around the direction of x_j^k. The probability density to reach B from x_j^k is denoted as $p_B(x_j^k)$.

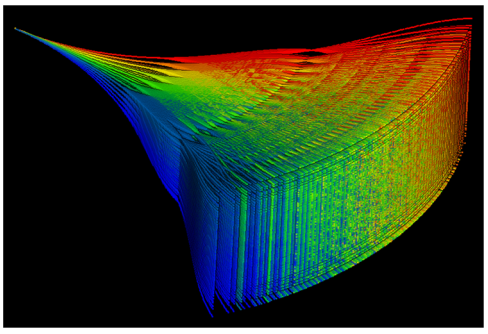

FIGURE 4 All 35 path groups. The paths are color coded based on the group index. There are 1,225 paths in each group and 42,875 paths in total [Color figure can be viewed at wileyonlinelibrary.com]

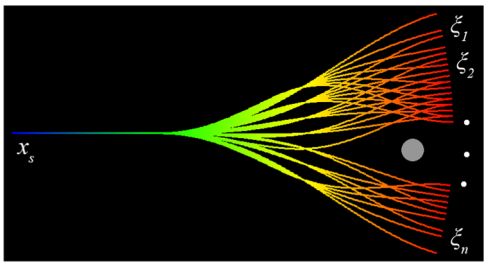

FIGURE 5 Collision-free paths in a group with an obstacle as the gray dot. The paths start at x_s. The path ends are considered Monte Carlo samples ξ_i, $i = 1, 2, ..., n$, whose distribution is drawn from $p(x_f|x_s)$ [Color figure can be viewed at wileyonlinelibrary.com]

The probabilities are transmittable between adjacent voxels. As illustrated in Figure 6b, consider the case that the probabilities are transmitted to x_j^k from the adjacent voxels, denoted as j_l on the bottom-left side and j_r on the upper-right side. Let θ_k be the direction associated with x_j^k. As the position is modeled to be uniformly distributed within a voxel, the probabilities to be transmitted to x_j^k are from the gray regions in voxels j_l and j_r, with areas $1 - \tan\theta_k/2$ and $\tan\theta_k/2$ of a voxel, respectively. Here, the gray regions are determined by drawing a line in j_l and j_r in parallel to the direction of x_j^k joining the bottom-left and upper-right vertices of voxel j. From each of the gray regions, the probabilities are transmitted from three adjacent directions. The probability density transmission is defined as

$$p_B(x_j^k) = \left(\left(1 - \frac{\tan\theta_k}{2}\right)a + \frac{\tan\theta_k}{2}b\right)r_j, \tag{7}$$

where

$$\begin{aligned} a &= w_y p_B\left(x_{j_l}^{k-1}\right) + w_f p_B\left(x_{j_l}^{k}\right) + w_y p_B\left(x_{j_l}^{k+1}\right), \\ b &= w_y p_B\left(x_{j_r}^{k-1}\right) + w_f p_B\left(x_{j_r}^{k}\right) + w_y p_B\left(x_{j_r}^{k+1}\right). \end{aligned}$$

In (7), r_j represents the traversability of voxel j due to obstacles, where $r_j = 1$ means complete clearance and $r_j = 0$ means complete occlusion. w_f and w_y determine the probability distribution corresponding to forward motion and tuning in yaw, respectively. We require that

$$w_f + 2w_y = 1. \tag{8}$$

The three-dimensional (3D) case is a direct extension to the 2D case where each voxel has multiple layers of the representation in Figure 6a. Each layer is associated with a pitch angle. Let α_l be the pitch of layer $l, l \in Z$. We modify the voxel state for the 3D case to be $x_j^{l,k}$. The probability density $p_B(x_j^{l,k})$ is transmitted from three adjacent voxels—two voxels are the same as in the 2D case and the third voxel is right above or below voxel j. If $\alpha_l > 0$, the voxel which can transit to voxel j is on the bottom, and if $\alpha_l < 0$, the voxel is on the top. Consider $\alpha_l > 0$ as an example. Let us denote the voxel below voxel j as voxel j_b. The region

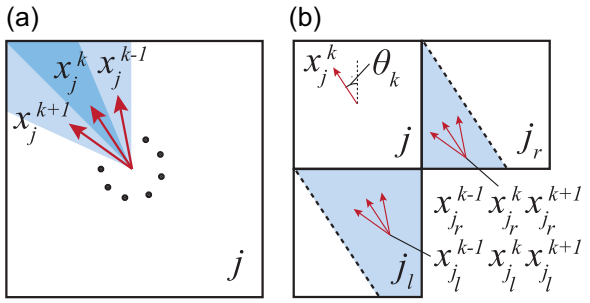

FIGURE 6 (a) Voxel representation. Each voxel contains multiple directions at a constant angular interval. The state of a voxel is denoted as $x_j^k, j, k \in Z$, where j is the voxel index and k is the direction index in the voxel. (b) Probability transmission. The probabilities are transmitted to x_j^k from the adjacent voxels j_l and j_r in three directions from each voxel [Color figure can be viewed at wileyonlinelibrary.com]

in voxel j_b which can transit to voxel j by following the direction of $x_j^{l,k}$ has an area $|\tan\alpha_l|(\sin\theta_k + \cos\theta_k)/2$ of a voxel. Correspondingly, the regions in voxels j_l and j_r which can transit to voxel j have areas $(1 - |\tan\alpha_l|(\sin\theta_k + \cos\theta_k)/2)(1 - \tan\theta_k/2)$ and $(1 - |\tan\alpha_l|(\sin\theta_k + \cos\theta_k)/2)\tan\theta_k/2$ of a voxel, respectively. The probability density is calculated as

$$\begin{aligned} p_B(x_j^{l,k}) = &\left(\left(1 - \frac{|\tan\alpha_l|}{2}(\sin\theta_k + \cos\theta_k)\right)\left(\left(1 - \frac{\tan\theta_k}{2}\right)c\right.\right. \\ &\left.\left. + \frac{\tan\theta_k}{2}d\right) + \frac{|\tan\alpha_l|}{2}(\sin\theta_k + \cos\theta_k)e\right)w_j^{l,k}. \end{aligned} \tag{9}$$

where

$$\begin{aligned} c = {} & w_{py} p_B\left(x_{j_b}^{l-1,k-1}\right) + w_p p_B\left(x_{j_b}^{l-1,k}\right) + w_{py} p_B\left(x_{j_b}^{l-1,k+1}\right) \\ & + w_y p_B\left(x_{j_b}^{l,k-1}\right) + w_f p_B\left(x_{j_b}^{l,k}\right) + w_y p_B\left(x_{j_b}^{l,k+1}\right) \\ & + w_{py} p_B\left(x_{j_b}^{l+1,k-1}\right) + w_p p_B\left(x_{j_b}^{l+1,k}\right) + w_{py} p_B\left(x_{j_b}^{l+1,k+1}\right), \\ d = {} & w_{py} p_B\left(x_{j_l}^{l-1,k-1}\right) + w_p p_B\left(x_{j_l}^{l-1,k}\right) + w_{py} p_B\left(x_{j_l}^{l-1,k+1}\right) \\ & + w_y p_B\left(x_{j_l}^{l,k-1}\right) + w_f p_B\left(x_{j_l}^{l,k}\right) + w_y p_B\left(x_{j_l}^{l,k+1}\right) \\ & + w_{py} p_B\left(x_{j_l}^{l+1,k-1}\right) + w_p p_B\left(x_{j_l}^{l+1,k}\right) + w_{py} p_B\left(x_{j_l}^{l+1,k+1}\right), \\ e = {} & w_{py} p_B\left(x_{j_r}^{l-1,k-1}\right) + w_p p_B\left(x_{j_r}^{l-1,k}\right) + w_{py} p_B\left(x_{j_r}^{l-1,k+1}\right) \\ & + w_y p_B\left(x_{j_r}^{l,k-1}\right) + w_f p_B\left(x_{j_r}^{l,k}\right) + w_y p_B\left(x_{j_r}^{l,k+1}\right) \\ & + w_{py} p_B\left(x_{j_r}^{l+1,k-1}\right) + w_p p_B\left(x_{j_r}^{l+1,k}\right) + w_{py} p_B\left(x_{j_r}^{l+1,k+1}\right). \end{aligned}$$

Here, $w_j^{l,k}$ represents the traversability of $x_j^{l,k}$ due to obstacles. w_p is the weight corresponding to turning in pitch, for the probability transmission from $p_B(x_{j_*}^{l-1,k})$ and $p_B(x_{j_*}^{l+1,k})$ to $p_B(x^{l,k})$, where * stands for b, l, or r. w_{py} is the weight corresponding to turning in both pitch and yaw, for the probability transmission from $p_B(x_{j_*}^{l-1,k-1})$, $p_B(x_{j_*}^{l-1,k+1})$, $p_B(x_{j_*}^{l+1,k-1})$, and $p_B(x_{j_*}^{l+1,k+1})$ to $p_B(x^{l,k})$. Again, we require no probability loss during the transmission, with

$$w_f + 2w_p + 2w_y + 4w_{py} = 1. \tag{10}$$

If $\alpha_l < 0$, voxel j_b in (9) is replaced by the voxel above voxel j, namely, voxel j_a. Two special cases exist. First, if $\alpha_l = 0$, the vehicle moves horizontally. The probabilities are transmitted from voxels j_l and j_r but not voxel j_a or j_b to voxel j, in 3D case. Second, if $\theta_k = 0$, the vehicle moves in parallel to voxel j_r. The probabilities are transmitted from voxel j_l but not voxel j_r to voxel j, in both 2D and 3D cases. During initialization, the probability densities are evenly distributed among all directions in the voxel containing B. Propagation of the probability is through an iteration process. Figure 7 gives an example of the propagated probability densities in a 2D environment. Brighter voxel indicates higher probability density.

4.4 | Method implementation

The path groups described in Section 4.2 are generated offline. For collision check, we use a voxel grid overlaid with sensor range $\mathcal{S}$.

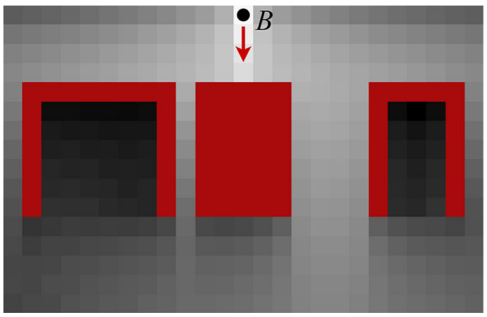

FIGURE 7 Propagated probability densities in a 2D environment. The arrow represents the direction in the voxels that the probability densities are associated. Red areas are obstacles with the traversability defined in (7) set at $r_j = 0.01$. Brighter voxels have higher probability densities to navigate from itself to B and darker voxels have lower probability densities. 2D, two dimensional [Color figure can be viewed at wileyonlinelibrary.com]

The correspondences between the voxels and paths are pre-established and stored in an adjacency list. In the adjacency list, each row is associated with a voxel and consists of indexes of the paths that are occluded by an obstacle placed at the center of the voxel. Here, the vehicle radius is taken into account for calculating the occlusions. Upon system starts, the paths and adjacency list are loaded into the vehicle computer memory. The online collision check processes all perception sensor data points and labels the corresponding paths to be occluded according to the adjacency list. Then, the algorithm traverses all paths in each group to compute $P_B(x_s)$ based on (6) and chooses the path group with the highest $P_B(x_s)$. The algorithm returns x_s^* as

$$x_s^* = \arg\max P_B(x_s). \tag{11}$$

Recall n is the number of paths in a path group. Let $h \in Z^+$ be the number of path groups and $m \in Z^+$ be the number of perception sensor data points. The online processing algorithm is described in Algorithm 1.

Theorem 1. *The online processing algorithm has a computational complexity of* $O(mnh)$.

Theorem 1 analyzes the computational complexity in the worst case where every perception sensor data point blocks all paths in each group. In practice, a data point can block a few paths so that the computation is much lighter. The probability propagation in the global scale uses a second voxel grid covering the environment, run only once before the navigation. This uses an implementation similar to the A* algorithm (Zeng & Church, 2009), where only the probability densities in the voxels adjacent to those in the open set are updated. This process terminates if the changes to the probability densities in the voxel containing A are smaller than a threshold.

In the case that a prior map is unavailable, we can alternatively use a heuristic function to compute $p_B(\xi_i)$ in (6):

$$p_B(\xi_i) = \begin{cases} -|\Delta y_i|, & \text{2D case}, \\ -|\Delta p_i \Delta y_i|, & \text{3D case}, \end{cases} \tag{12}$$

where Δp_i and Δy_i are the relative angles between ξ_i and the goal direction in pitch and yaw, respectively. Equation (12) essentially biases the path planner toward the goal direction. Note that the same function can be used to process human input, for example, from a joystick controller. The result is that the vehicle is guided by an operator while itself conducts collision avoidance during the navigation.

5 | EXPERIMENTS

5.1 | Simulation

We first validate the method in simulation. We use a prior map in these experiments and we compare the time of probability propagation in our method to the state-of-the-art planning methods. A 3.1 GHz i7 computer

Algorithm 1: Online Processing Algorithm

input : path groups, adjacency list, $p_B(x_j^k)$ for 2D or $p_B(x_j^{l,k})$ for 3D, $j, k, l \in Z$
output : x_s^* or no-path-found
begin
  Initialize all paths as unoccluded; // $\mathcal{O}(nh)$
  for each perception sensor data point **do** // $\mathcal{O}(mnh)$
    Label corresponding paths as occluded using the adjacency list; // $\mathcal{O}(nh)$
  end
  if not all paths are occluded **then** // $\mathcal{O}(1)$
    for each path group **do** // $\mathcal{O}(nh)$
      Compute $P_B(x_s)$ based on (6); // $\mathcal{O}(n)$
    end
    Find the path group with $\max P_B(x_s)$ and return x_s^*; // $\mathcal{O}(h)$
  end
  else
    Return no-path-found; // $\mathcal{O}(1)$
  end
end

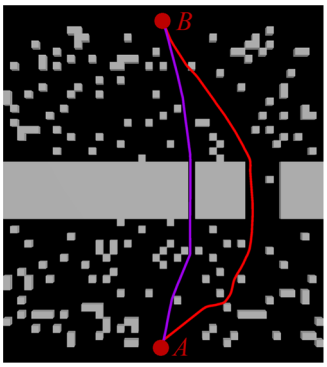

FIGURE 8 Narrow pathway test result. The environment contains a narrow pathway on the left and a wide pathway on the right. The widths of the narrow and wide pathways are 5 m and 20 m, respectively. Existing planning methods, such as RRT* (magenta curve) mostly seek the shortest path regardless of the width of the pathway. Our method (red curve) maximizes the probability to reach the goal and therefore prefers the wider pathway. Wider pathways help keep the vehicle safe during the flight and leave more choices for obstacle avoidance in the presence of dynamic obstacles. RRT, Rapidly exploring Random Tree [Color figure can be viewed at wileyonlinelibrary.com]

is used to record the CPU processing time. Figure 8 shows the result of a test containing a narrow pathway and a wide pathway. Common planning methods mostly find the shortest path connecting A to B regardless of the width of the pathway. In Figure 8, the orange curve is generated by RRT* (Karaman & Frazzoli, 2011). Our method seeks the highest probability to reach the goal and therefore prefers the wider pathway. Wide pathways are preferable in aerial navigation keeping the vehicle safe as well as leaving more choices for obstacle avoidance. Note that this behavior can be produced by other methods if considering the distance from a node to the closest obstacle in the cost function. Our method incorporates such behavior as its inherent characteristic.

Figure 9 shows the result of a test involving an environment change between the prior map and actual world. Figure 9a shows the prior map where an opening is available to the right. Our method uses the prior map for probability propagation before the navigation starts. Obstacles on the prior map have the traversability set at $r_j = 0.01$. Figure 9b shows the actual world where the opening on the prior map is closed. Another opening is now available to the front. After the navigation starts, our method generates a path curving to the right as an effect of the opening on the prior map. As the vehicle approaches, the method realizes the environment change from the perception sensors and then guides the vehicle toward the opening to the front. Thanks to the minor probabilities propagated through the obstacles on the prior map. On the other hand, traditional methods based on deterministic representations of the environment encounter difficulty. In Figure 9c RRT* generates a path using the prior map when the navigation starts. As the vehicle travels, in Figure 9d, the environment seen by the perception sensors is updated indicating that the opening to the right is unavailable. However, the opening to the front has not been seen. As a result, RRT* finds no path from A to B.

The proposed method is further tested in 2D random world environments. The result is in Figure 10, where our method is compared with RRT (LaValle, 2006), RRT-Connect (Kuffner & LaValle, 2019), RRT*, and batch informed tree (BIT*; Gammell et al., 2015) planners. Figure 10a shows an example environment and representative paths generated by all methods. The obstacle ratio is set at 20%. Figure 10b compares the runtime and corresponding success rate. Each method is tested in 50 randomly generated environments and executed 10 times

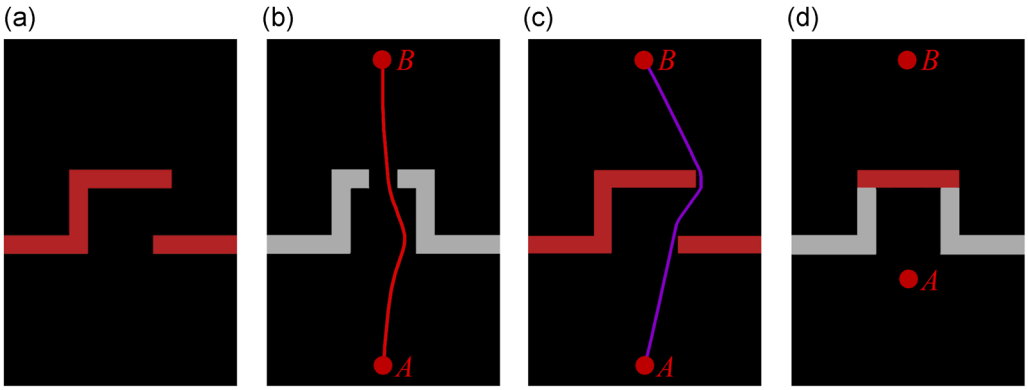

FIGURE 9 Outdated prior test result. The test involves an environment change between the prior map and actual world. (a) shows the prior map where an opening is available to the right. Our method uses the prior map for probability propagation. Obstacles on the prior map have the traversability set at $r_j = 0.01$. (b) shows the actual world where the opening is to the front due to the environment change. Upon the navigation starts, the vehicle curves to the right as an effect of the opening on the prior map. As the vehicle approaches, perception sensor data indicates the environment change and the vehicle is then guided toward the opening to the front because of minor probabilities propagated through the obstacles on the prior map. On the other hand, existing methods based on deterministic representations of the environment encounter difficulty. In (c), at the start of the navigation, RRT* uses the prior map to plan a path. During the navigation, the environment within $\mathcal{S}$ is updated continuously by the perception sensors. In (d), the vehicle realizes the opening to the right is unavailable. However, the opening to the front has not yet been seen. RRT, Rapidly exploring Random Tree [Color figure can be viewed at wileyonlinelibrary.com]

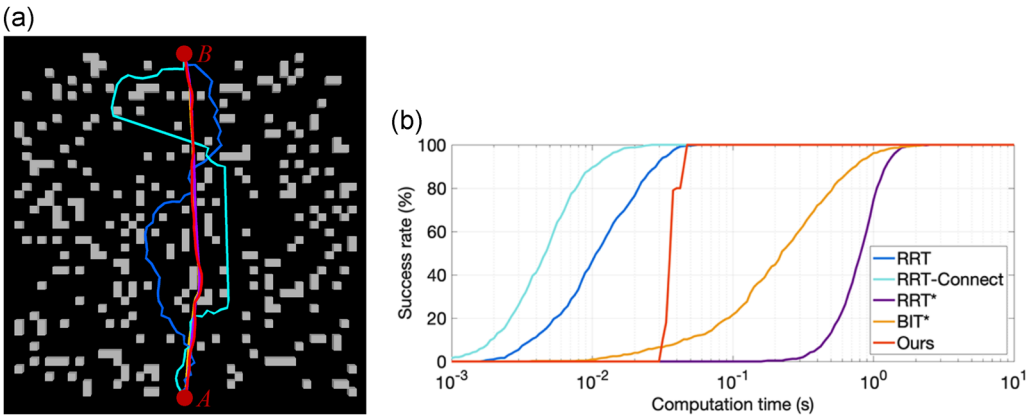

FIGURE 10 Two-dimensional random world test result. The proposed method is compared with RRT, RRT-Connect, RRT*, and BIT* in random world environments. The obstacle ratio is set at 20%. (a) An example environment and the path from each method. (b) A comparison of the runtime by testing in 50 environments. Note that RRT and RRT-Connect terminate after finding the first path. Their runtime is faster than our method but the paths are not practically useful. RRT* and BIT* terminate after finding the near-optimal path. Their paths are similar to our method but the runtime is slower. BIT, batch informed tree; RRT, Rapidly exploring Random Tree [Color figure can be viewed at wileyonlinelibrary.com]

in each of the environments. Note that RRT and RRT-Connect terminate after finding the first path. Given the same success rate, their runtime is faster than ours but the resulting paths are not practically useful. RRT* and BIT* are configured to terminate after finding the near-optimal path. Their runtime is much slower than ours given the same success rate.

We also test the method in 2D maze environments. As shown in Figure 11, the size of the maze is set at 45×45. Figure 11a shows an example environment. All methods generate similar paths in the maze environments because of constrained space for navigation. Figure 11b compares the runtime. Our method runs more than 10 times faster than RRT-Connect and about 100 times faster than RRT, RRT*, and BIT* while producing the same success rate.

The proposed method is tested in 3D cases. Figure 12 presents the result in 3D random world environments. Figure 12a gives an example environment with a representative path from each method. The obstacle ratio is set at 20%. Figure 12b shows the runtime and corresponding success rate, tested in 50 randomly generated environments and executed 10 times in each environment. Similar to the result in Figure 10, RRT and RRT-Connect terminate after finding the first path. Even though their runtime is faster than ours, the resulting paths are practically useless. compared with RRT* and BIT*, our method consumes much less runtime at the same success rate while the three methods produce similar paths.

Finally, we evaluate the method in 3D maze environments. The result is in Figure 13. The size of the maze is set at $25 \times 25 \times 25$. Figure 13a shows an example environment. Similar

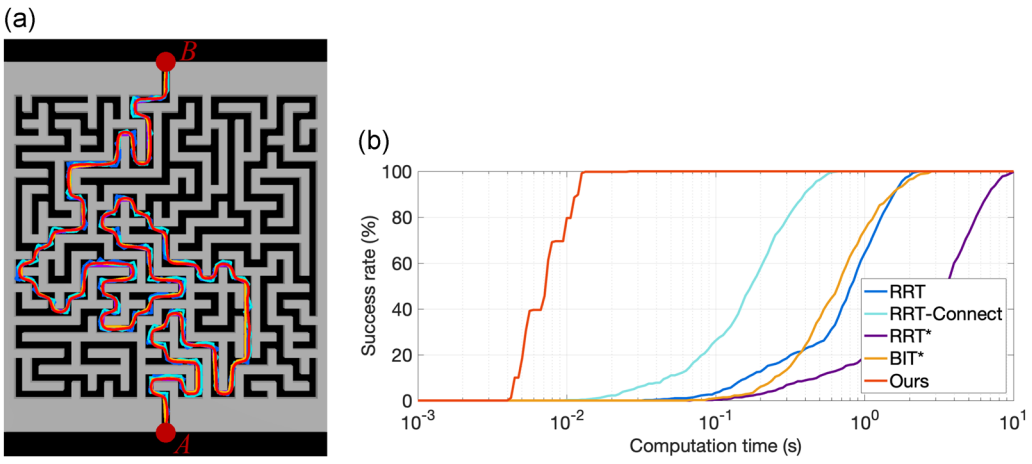

FIGURE 11 Two-dimensional maze test result. The proposed method is compared with RRT, RRT-Connect, RRT*, and BIT* in maze environments. The size of the maze is set at 45×45. (a) An example environment and the paths where all methods generate similar paths. (b) A comparison of the runtime by testing in 50 environments. Our method is more than 10 times faster than RRT-Connect and about 100 times faster than RRT, RRT*, and BIT*. BIT, batch informed tree; RRT, Rapidly exploring Random Tree [Color figure can be viewed at wileyonlinelibrary.com]

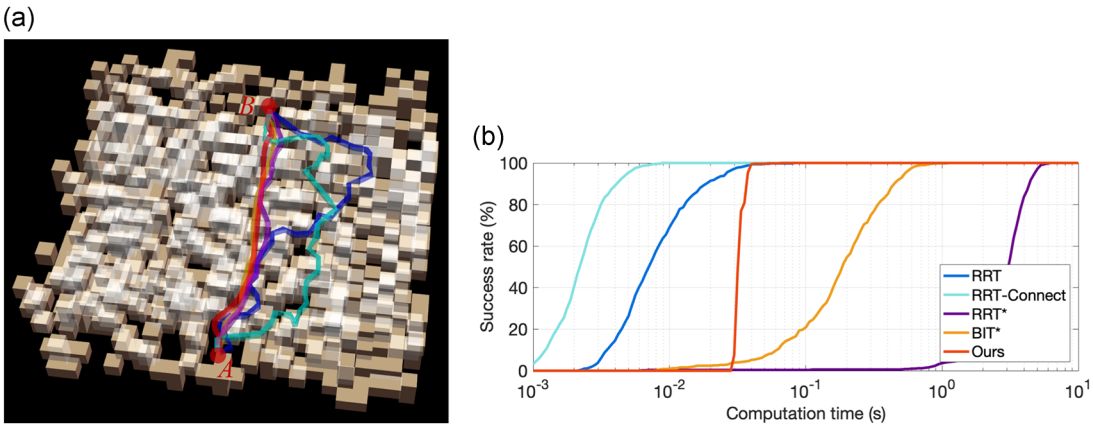

FIGURE 12 Three-dimensional random world test result. The obstacle ratio is set at 20%. (a) An example environment and the path from each method. (b) A comparison of the runtime by testing in 50 environments. Similar to Figure 10, RRT and RRT-Connect terminate after finding the first path. Their runtime is faster than our method but the paths are practically useless. RRT* and BIT* are set to find the near-optimal path. Their paths are similar to our method but the runtime is slower given the same success rate. BIT, batch informed tree; RRT, Rapidly exploring Random Tree [Color figure can be viewed at wileyonlinelibrary.com]

to Figure 11, all methods produce similar paths in the maze environments. Figure 13b compares the runtime. Our method outperforms the other methods by a factor of 10 in terms of runtime while producing the same success rate.

5.2 | Unmanned aerial vehicle (UAV) experiments

The UAV experiment platform is shown in Figure 14. 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 micro-electromechanical system (MEMS)-based Inertial Measurement Unit (IMU). A 3.1 GHz i7 embedded computer carries out all onboard processing. The state estimation is based on our previous work (Zhang & Singh, 2018), which integrates data from the three sensors to provide vehicle poses and registered laser scans. The tests use the path groups described in Section 4.2. Sensor range $\mathcal{S}$ is set at 30 m in front of the vehicle and the collision check uses a voxel grid overlaid with $\mathcal{S}$ at 0.1 m resolution.

The site of UAV Test 1 is in a forest as shown in Figure 15. The flight test does not use a prior map but the heuristic function in (12) to guide the navigation. Figure 15a shows an aerial overview of the test site. Figure 15b presents an image logged by an onboard camera during the flight. Figure 15c shows a render of the map built during the flight with the executed path overlaid on the map, from the same viewpoint as in Figure 15b. Figure 15d presents the registered scans as the perception sensor data during the flight (colored points) and the determined collision-free paths (white curves). The yellow curve is the selected path for the vehicle to execute. The vehicle pose in Figure 15d is the same as in Figure 15b. Figure 15e shows the entire map and overall path of the

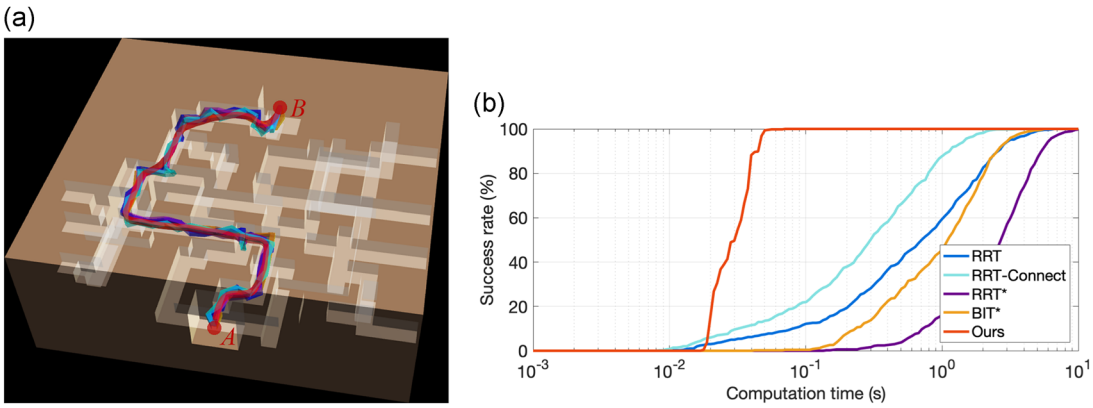

FIGURE 13 Three-dimensional maze test result. The size of the maze is set at 25 ×25 × 25. (a) An example environment and the paths where all methods generate similar paths. (b) A comparison of the runtime by testing in 50 environments. Given the same success rate, our method outperforms RRT, RRT-Connect, RRT*, and BIT* by a factor of 10 in terms of runtime. BIT, batch informed tree; RRT, Rapidly exploring Random Tree [Color figure can be viewed at wileyonlinelibrary.com]

FIGURE 14 UAV experiment 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 an MEMS-based IMU. An i7 embedded computer carries out all onboard processing. Note that GPS data are unused in the test. GPS, Global Positioning System; IMU, Inertial Measurement Unit; MEMS, micro-electromechanical system; UAV, unmanned aerial vehicle [Color figure can be viewed at wileyonlinelibrary.com]

flight. The vehicle position in Figure 15b,d is labeled with number 1 in Figure 15e. The canopy of the forest is manually cropped to reveal the flight path. The traveling distance is approximately 300 m and the maximum speed during the flight reaches 10 m/s as indicated in Figure 16.

Further, let us inspect some metrics from UAV Test 1. Different from the simulation tests, the test does not use a prior map or

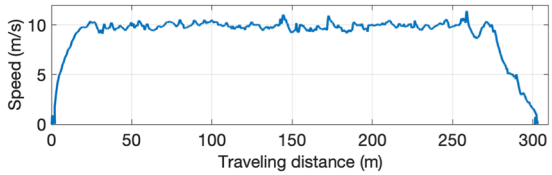

FIGURE 16 Speed in UAV Test 1. The maximum speed of the UAV reaches 10 m/s while flying in a forest environment as presented in Figure 15. UAV, unmanned aerial vehicle [Color figure can be viewed at wileyonlinelibrary.com]

propagate the probabilities. At initialization, the onboard navigation system reads the paths and adjacency list (described in Section 4.4) into the computer memory. The onboard processing time is listed in Table 1. Collision check first processes all perception sensor data points to determine the collision-free paths, taking 213.7 μs on average. Then, the processing traverses all paths to compute $P_B(x_s)$ for each group and select the path group with the highest $P_B(x_s)$, taking 38.4 μs on average. The method runs at 5 Hz. The resulting CPU load is <5% of a single thread.

UAV Test 2 is conducted in an orchard as shown in Figure 17. In this test, an operator uses a joystick controller to guide the navigation. The system uses (12) to interpret the directional input from the joystick controller. Figure 17a gives an aerial overview of the test site with the start point and target. Figure 17b shows a

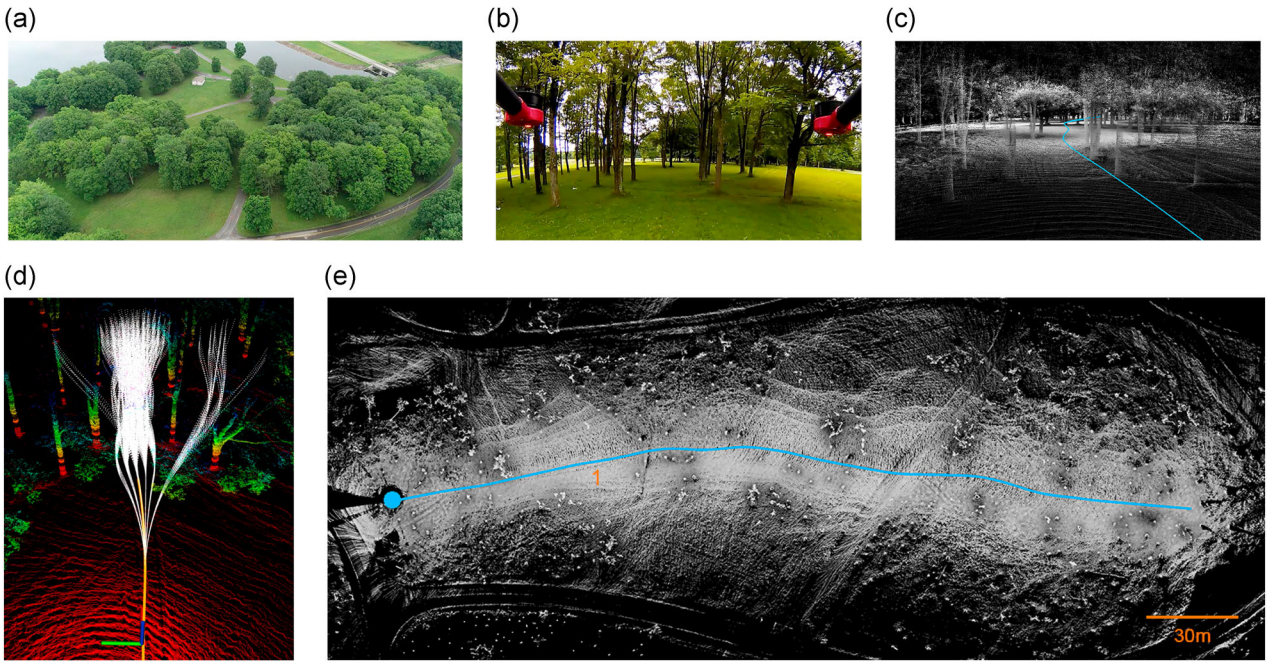

FIGURE 15 Result of UAV Test 1. The flight test is conducted in a forest environment. No prior map is used in the test. The method uses the heuristic function in (12) to guide the navigation. (a) An aerial overview of the test site. (b) An image from an onboard camera captured during the flight. (c) A render of the map built during the flight and the executed path overlaid on the map. (d) The perception sensor data during the flight as the colored points and the corresponding collision-free paths as the white curves. The yellow curve is the selected path for the vehicle to execute. The vehicle pose is the same as in (b). (e) The entire map and overall path of the flight. The vehicle position in (b) and (d) is labeled with number 1 in (e). The canopy of the forest is removed to reveal the path. The flight has 300 m of travel and the vehicle speed is at 10 m/s through the course of the flight. UAV, unmanned aerial vehicle [Color figure can be viewed at wileyonlinelibrary.com]

TABLE 1 Online processing time in UAV Test 1

Collision check		Path selection		Overall	
Mean (μs)	Worst (μs)	Mean (μs)	Worst (μs)	Mean (μs)	Worst (μs)
213.7	286.2	38.4	41.3	252.1	327.5

Abbreviation: UAV, unmanned aerial vehicle.

photo of the UAV-operator setup. The operator uses goggles to see first-person-view images from the UAV. Figure 17c is an example first-person-view image when the UAV is approaching the target. Figure 17d shows a photo of the UAV during the flight when the UAV is passing underneath a wire. Figure 17e presents the perception sensor data as the colored points and the corresponding collision-free paths as the white curves. Figures 17f,g, are two different views of the map built during the flight and the flight path. The vehicle position in Figure 17d,e is labeled with number 1 in Figure 17f. The target is labeled in red. Figure 17g gives a zoomed-in view of the flight path underneath the wire. After approaching the target, the UAV follows the same path back to the start point. The flight speed is 5 m/s over 284 m of travel.

FIGURE 17 Result of UAV Test 2. The flight test is conducted in an orchard. Here, an operator uses a joystick controller to guide the flight. (a) An aerial overview of the test site with the start point and target. (b) A photo of our UAV-operator setup. The operator uses goggles to see first-person-view images from the UAV. (c) An example first-person-view image when the UAV is approaching the target. (c) A photo of the UAV during the flight while passing underneath a wire. (e) The perception sensor data as the colored points and the corresponding collision-free paths as the white curves. A photo of the joystick operation is shown at the bottom-right corner. (f, g) Two views of the map and flight path. The vehicle position in (d) and (e) is labeled with number 1 in (f). The target is labeled in red. (g) A zoomed-in view of the flight path underneath the wire. The UAV follows the same path back to the start point after approaching the target. The flight speed is 5 m/s over 284 m of travel. UAV, unmanned aerial vehicle [Color figure can be viewed at wileyonlinelibrary.com]

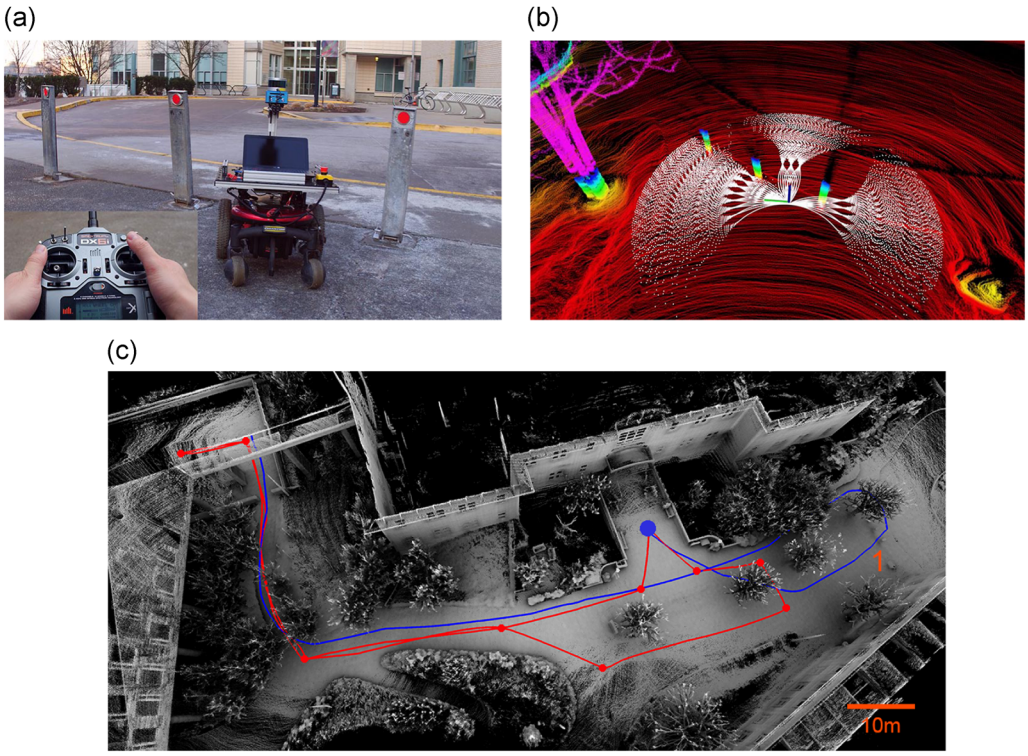

FIGURE 18 Result of Ground Vehicle Test 1. (a) A photo of the ground vehicle navigating on the university campus. The vehicle shares the same sensor configuration with the UAV in Figure 14. (b) The corresponding data render where the colored points are from registered scans and the white curves show the collision-free paths. (c) The resulting map and navigation paths. The test consists of two runs. The blue path is from the first run where the vehicle is guided by an operator. Upon finishing the first run, a map is built. Way-points are selected based on the map as the orange dots. The orange path is from the second run where the vehicle follows the way-points. Both runs start from the blue dot. The speed is 1 m/s. The vehicle position in (a) and (b) is labeled with number 1 in (c). UAV, unmanned aerial vehicle [Color figure can be viewed at wileyonlinelibrary.com]

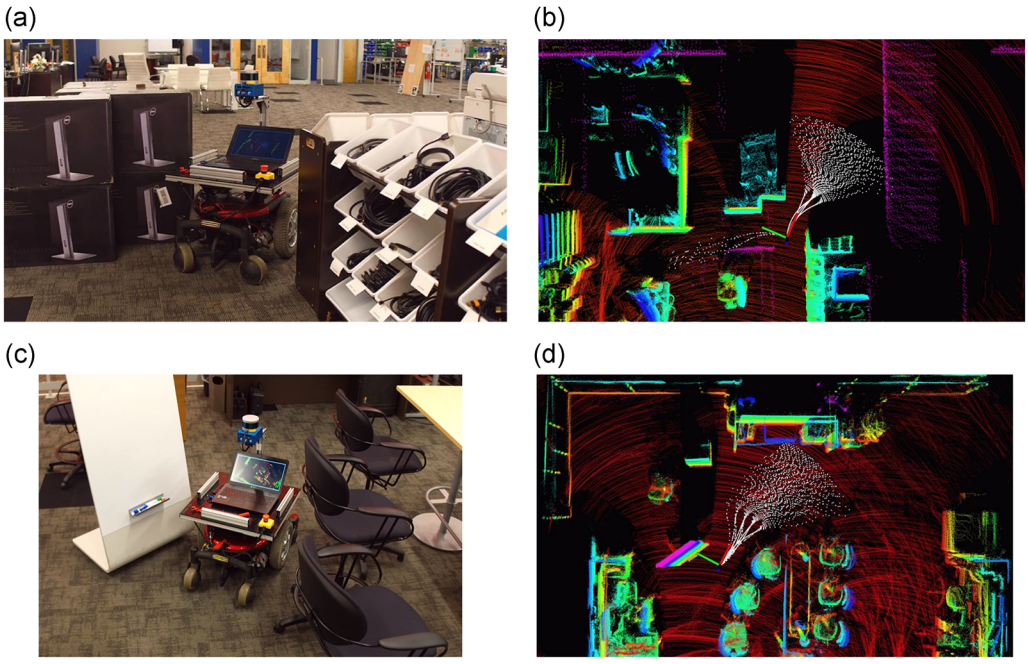

FIGURE 19 Result of Ground Vehicle Test 2. The vehicle passes through two tight openings as in (a) and (c). (b, d) The corresponding data render where the colored points show the registered scans and the white curves are collision-free paths [Color figure can be viewed at wileyonlinelibrary.com]

5.3 | Ground vehicle experiments

The ground vehicle experiments use a wheelchair-based vehicle with the same sensor configuration as the UAV in Figure 14. Figure 18 shows the result of Ground Vehicle Test 1. A photo of the vehicle is present in Figure 18a while the vehicle is navigating on the university campus. Figure 18b shows the corresponding data render where the colored points are from registered scans and the white curves show the collision-free paths. Different from the UAV tests, the paths used in the ground vehicle tests are in 2D and spread out in all directions except to the back. Sensor range $\mathcal{S}$ is set at 3 m around the vehicle and the collision check uses a voxel grid overlaid with $\mathcal{S}$ at 0.02 m resolution. Figure 18c shows the map built during the test and the navigation paths. The test consists of two separate runs. In the first run, the vehicle is guided by an operator with a joystick controller. The navigation path is in blue. Upon finishing the first run, a map is built. Way-points are then defined based on the map as the orange dots. In the second run, the vehicle follows the way-points autonomously. The navigation path is in orange. Both runs start from the blue dot. The speed is 1 m/s. The vehicle position in Figure 18a,b is labeled with number 1 in Figure 18c.

Further, Ground Vehicle Test 2 contains two tight openings in an indoor environment as shown in Figure 19. Figures 19a,c are two photos of the vehicle passing through the tight openings. Figures 19b,d are the corresponding data renders. Note that all ground vehicle experiments use the same set of parameters for planning and collision avoidance.

6 | CONCLUSION AND FUTURE WORK

The paper proposes a planning method to enable fast autonomous flight in complex environments. The environment is modeled to be deterministically known within the sensor range where obstacle information is from the perception sensors, and probabilistically known beyond the sensor range. Instead of searching for the path with the lowest cost, the method maximizes the likelihood to successfully reach the goal in determining the immediate next step for execution of the navigation. If a prior map is available, probabilities are propagated offline through the environment. If without a prior map, the method takes a directional input either from the goal point or a human commander as the guidance for the navigation. The online method realized by a trajectory library determines a path within 0.2–0.3 ms using a single CPU thread on a modem embedded computer. In experiments, it enables a lightweight UAV to fly at 10 m/s in a cluttered forest environment.

ORCID

Ji Zhang http://orcid.org/0000-0002-3122-3106

REFERENCES

Akgun, B., & Stilman, M. (2011). Sampling heuristics for optimal motion planning in high dimensions. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. San Francisco, CA.

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

Chung, J., Smith, A., Skeele, R., & Hollinger, G. (2019). Risk-aware graph search with dynamic edge cost discovery. *The International Journal of Robotics Research*, *38*(2–3), 182–195.

Droeschel, D., Nieuwenhuisen, M., Beul, M., Holz, D., Stuckler, J., & Behnke, S. (2016). Multi-layered mapping and navigation for autonomous micro aerial vehicles. *Journal of Field Robotics*, *33*(4), 451–475.

Fraichard, T., & Mermond, R. (1998). Path planning with uncertainty for car-like robots. In *IEEE International Conference on Robotics and Automation (ICRA)*. Leuven, Belgium.

Gammell, J. D., Srinivasa, S., & Barfoot, T. (2015). Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *IEEE International Conference on Robotics and Automation (ICRA)*. Seattle, WA.

Gammell, J. D., Srinivasa, S. S., & Barfoot, T. D. (2014). Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Chicago, IL.

Gonzlez, D., Prez, J., Milans, V., & Nashashibi, F. (2016). A review of motion planning techniques for automated vehicles. *IEEE Transactions on Intelligent Transportation Systems*, *17*(4), 1135–1145.

Heiden, E., Hausman, K., Sukhatme, G., & Agha-mohammadi, A. (2017). Planning high-speed safe trajectories in confidence-rich maps. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Vancouver, Canada.

Hsu, D., Latombe, J.-C., & Kurniawati, H. (2006). On the probabilistic foundations of probabilistic roadmap planning. *The International Journal of Robotics Research*, *25*(7), 627–643.

Kala, R., & Warwick, K. (2013). Multi-level planning for semi-autonomous vehicles in traffic scenarios based on separation maximization. *Journal of Intelligent and Robotic Systems*, *72*(3/4), 559–590.

Karaman, S., & Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, *30*(7), 846–894.

Kavraki, L. E., Kolountzakis, M. N., & Latombe, J.-C. (1998). Analysis of probabilistic roadmaps for path planning. *IEEE Transactions on Robotics and Automation*, *14*(1), 166–171.

Kuffner, J. J., & LaValle, S. M. (2019). RRT-connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation (ICRA)*. San Francisco, CA.

LaValle, S. M. (2006). *Planning algorithms*. New York, NY: Cambridge University Press.

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

Melchior, N. A., & Simmons, R. (2007). Particle RRT for path planning with uncertainty. In *IEEE International Conference on Robotics and Automation (ICRA)*. Roma, Italy.

Otte, M., & Correll, N. (2013). C-FOREST: Parallel shortest path planning with superlinear speedup. *IEEE Transactions on Robotics and Automation*, *29*(3), 798–806.

Pereira, G. A. S., Choudhury, S., & Scherer, S. (2016). A framework for optimal repairing of vector field-based motion plans. In *International Conference on Unmanned Aircraft Systems (ICUAS)*. Arlington, VA.

Robert, C. (2004). *Monte Carlo methods*. John Wiley & Sons Ltd.

Rufli, M., & Siegwart, R. Y. (2009). On the application of the D search algorithm to time-based planning on lattice graphs. In *The European Conference on Mobile Robots (ECMR)*. Dubrovnik, Croatia.

Scherer, S., Singh, S., & Chamberlain, L. (2008). Flying fast and low among obstacles: Methodology and experiments. *The International Journal of Robotics Research*, *27*(5), 549–574.

Van Den Berg, J., Abbeel, P., & Goldberg, K. (2011). LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information. *The International Journal of Robotics Research*, *30*(7), 895–913.

Zeng, W., & Church, R. L. (2009). Finding shortest paths on real road networks: The case for A*. *International Journal of Geographical Information Science*, *23*(4), 531–543.

Zhang, J., Chadha, R. G., Velivela, V., & Singh, S. (2018). P-CAP: Pre-computed alternative paths to enable aggressive aerial maneuvers in cluttered environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Madrid, Spain.

Zhang, J., Chadha, R. G., Velivela, V., & Singh, S. (2019a). P-CAL: Pre-computed alternative lanes for aggressive aerial collision avoidance. In *The 12th International Conference on Field and Service Robotics (FSR)*. Tokyo, Japan.

Zhang, J., Hu, C., Chadha, R. G., & Singh, S. (2019b). Maximum likelihood path planning for fast aerial maneuvers and collision avoidance. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Macau, China.

Zhang, J., & Singh, S. (2018). Laser-visual-inertial odometry and mapping with high robustness and low drift. *Journal of Field Robotics*.

How to cite this article: Zhang J, Hu C, Chadha RG, Singh S. Falco: Fast likelihood-based collision avoidance with extension to human-guided navigation. *J Field Robotics*. 2020;37:1300–1313. https://doi.org/10.1002/rob.21952