# A Fast & Efficient Mission Planner for Multi-rotor Aerial Vehicles in Large, High-resolution Maps of Cluttered Environments

David Butterworth

CMU-RI-TR-17-07

May 2017

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania

Thesis committee:
Sanjiv Singh (Advisor)
Stephen Nuske
Sanjiban Choudhury

## Abstract

Autonomous multi-rotor aerial vehicles have many potential applications in urban environments, such as inspecting infrastructure, creating 3D maps from the air, or delivering packages. However this involves flying close to obstacles like trees and buildings, or flying under overhanging structures like bridges, powerlines and doorways. Flying safely in environments like this will require perception and planning with respect to a full 3D world model.

Recent work has demonstrated that multi-rotor UAVs can robustly localize to static point cloud maps of urban environments using LiDAR- and VO-based methods. However it is difficult to plan directly in large 3D maps due to the memory requirements and size of the state space. We investigate the problem of planning paths in large high-resolution point clouds, for missions that include flying around obstacles and under overhanging structures.

Our approach is to plan offline a complete global mission path in the static map, that is collision-free and satisfies altitude constraints. We use a random-sample based planning algorithm to find approximate shortest paths, with rejection-sampling to satisfy constraints. We also introduce a novel approach for avoiding obstacles that may appear on the mission path by pre-planning a network of alternative paths.

We present results showing a comparison of various path planning algorithms and choose BIT* because it finds the approximate shortest path in the fastest time. For planning alternative paths, we use geometric path primitives based on splines or revert to BIT*. We also compare various data structures for storing the 3D world representation and use a sparse Octree to store occupied voxels because it offers the best trade-off between storage memory and collision-checking speed. We present qualitative results showing the resulting mission paths for multiple environments, including an industrial site and a tree-filled area.

# Acknowledgements

This research was conducted as part of the “Safe 50” project, working with:
Sanjiv Singh - **PI**
Marcel Bergerman - **PI**
Ji Zhang - **Team Lead**
Rushat G. Chadha
Vivek Velivela
David Murphy
Dave Wright
Kimber Rugg

Thanks to the staff at Near Earth Autonomy and Kaarta for collecting the point cloud data used to conduct the experiments in this report.

This work was partially funded by the NASA Small Business Innovation Research Award for “Autonomous, Safe Take-Off and Landing Operations for Unmanned Aerial Vehicles in the National Airspace”, contract No. NNX16CA03C.

# Contents

# Nomenclature

| | |
|---|---|
| **AABB** | Axis-Aligned Bounding Box |
| **BVH** | Bounding Volume Hierarchy |
| **DEM** | Digital Elevation Map |
| **DSM** | Digital Surface Map (includes the ground, trees & buildings) |
| **DTM** | Digital Terrain map (the bare earth surface) |
| **FAA** | United States Federal Aviation Administration |
| **FCL** | Flexible Collision Library |
| **GIS** | Geographic Information System |
| **GPS** | Global Positioning System |
| **LiDAR** | Laser-based radar (distance measuring technique) |
| **MAV** | Micro Aerial Vehicle |
| **NASA** | National Aeronautics and Space Administration |
| **Octree** | A data structure where one parent cell gets divided into 8 child cells |
| **PCL** | PointCloud Library |
| **SfM** | Structure from Motion (a technique to create 3D models from 2D photographs) |
| **TIN** | Triangular Irregular Network |
| **TPS** | Thin Plate Splines |
| **UAS** | Unmanned Aerial System |
| **UAV** | Unmanned Aerial Vehicle or "drone" |
| **UTM** | UAV Traffic Management |
| **VO** | Visual Odometry |

# 1 Introduction

## 1.1 UAVs in Urban Environments

Civilian UAVs have potential applications in search and rescue, monitoring agricultural crops, and inspecting utility infrastructure such as railways and powerlines. The potential benefits are greater if the UAV is semi- or fully-autonomous, requiring only occasional human oversight or none at all. This would allow the above use cases to be performed at lower cost, during any time of day, or enable new possibilities such as autonomous package delivery.

However there is significant work required to improve the current technology: battery life, safety control systems, beyond line of sight control, and the computation-to-weight ratio. At present, electrically powered multi-rotor

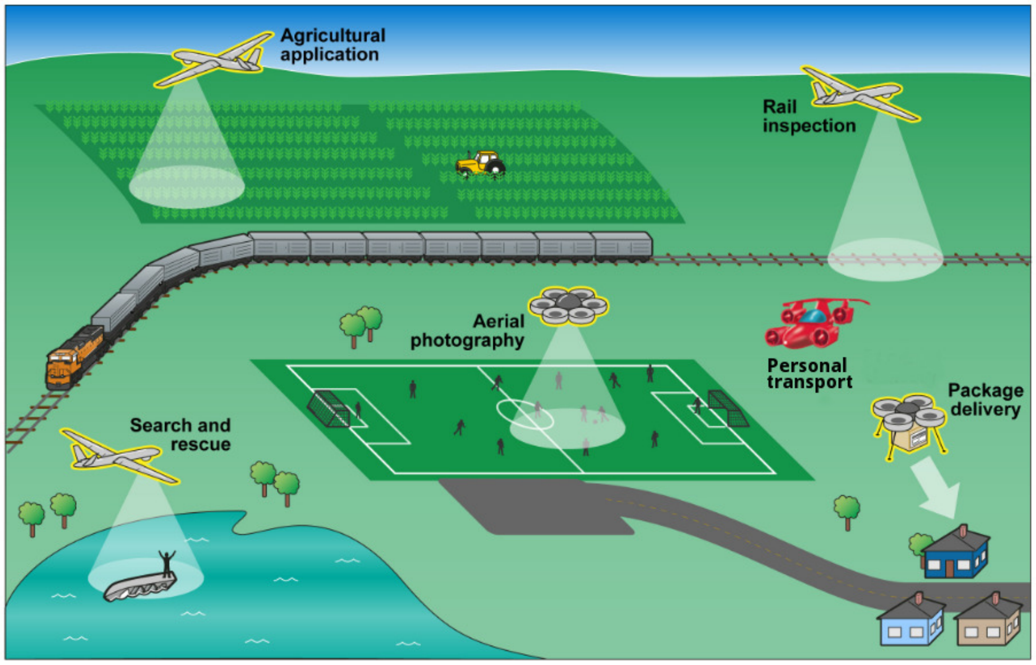


**Figure 1:** Potential applications for UAVs. Image adapted from [1].

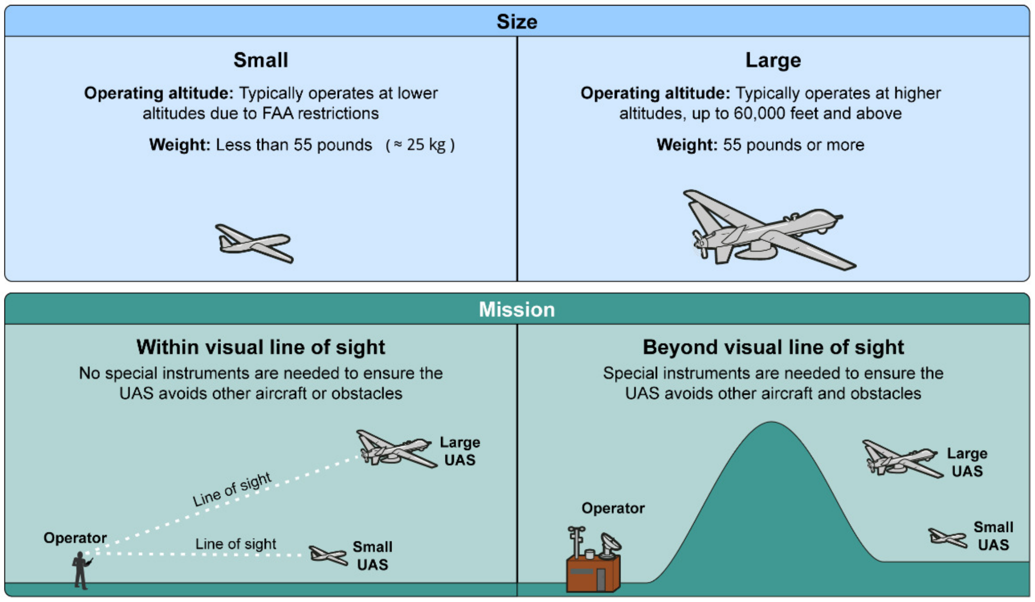

Size
Small
Operating altitude: Typically operates at lower altitudes due to FAA restrictions
Weight: Less than 55 pounds (≈ 25 kg)
Large
Operating altitude: Typically operates at higher altitudes, up to 60,000 feet and above
Weight: 55 pounds or more
Mission
Within visual line of sight
No special instruments are needed to ensure the UAS avoids other aircraft or obstacles
Large UAS
Line of sight
Operator
Line of sight
Small UAS
Beyond visual line of sight
Special instruments are needed to ensure the UAS avoids other aircraft and obstacles
Operator
Large UAS
Small UAS

**Figure 2:** The FAA categorizes UAVs based on their weight and if they can be operated beyond visual range of a human operator. In this work we test our planning and obstacle avoidance system with a small UAV, that was controlled autonomously but within visual range of a safety operator. Image adapted from [1].

UAVs have a flight time of less than 30 minutes, which is not long enough for most of the above applications. Flight safety control systems need to be developed so the UAV won't fall out of the sky if something unexpected happens. Mission control systems must be developed so the ground station can send occasional commands to the UAV and receive telemetry, beyond line-of-sight or even in a different geographical location. And all of this autonomy equipment, plus the mission payload, needs to be light enough to be carried by the UAV.

The FAA is investigating the concept of a UTM system. If you want to fly a regular helicopter or airplane, you can submit a request via the internet and receive instant approval for a flight plan. There are laws about what altitude the craft must fly and what to do if another craft is encountered.

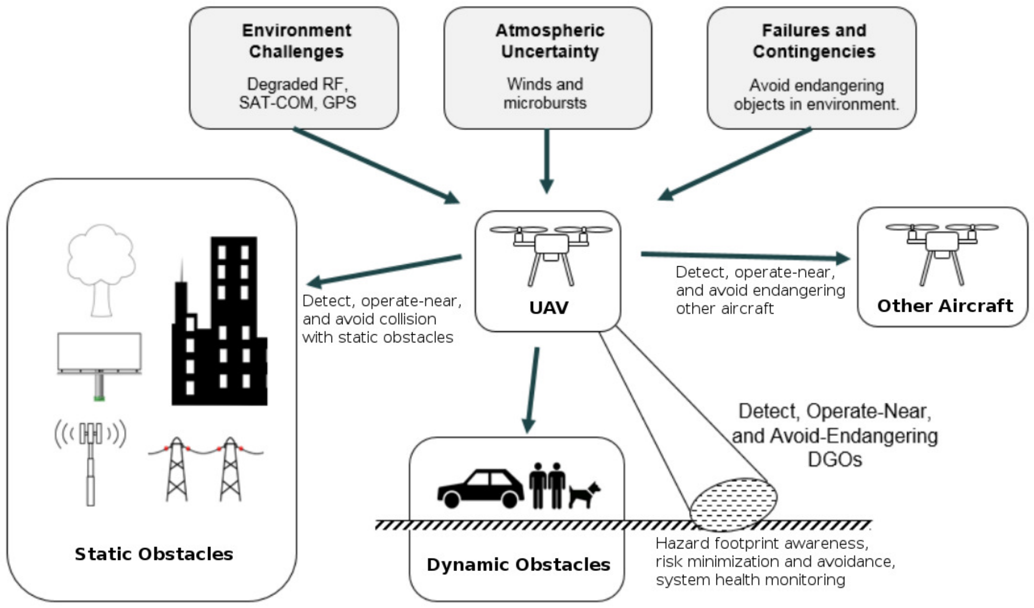

Environment Challenges
Degraded RF, SAT-COM, GPS
Atmospheric Uncertainty
Winds and microbursts
Failures and Contingencies
Avoid endangering objects in environment.
UAV
Detect, operate-near, and avoid collision with static obstacles
Detect, operate-near, and avoid endangering other aircraft
Other Aircraft
Detect, Operate-Near, and Avoid-Endangering DGOs
Static Obstacles
Dynamic Obstacles
Hazard footprint awareness, risk minimization and avoidance, system health monitoring

**Figure 3:** The hazards that a small UAV must handle when operating in cluttered urban environments. Image adapted from [2].

When the aircraft is nearing controlled airspace then it must communicate with the local aircraft control tower. Before allowing UAVs to operate in the national airspace, they will need to be capable of avoiding obstacles, and safely operating in the event of loss of signal in the radio data link or interference in GPS.

NASA is investigating the feasibility of UAVs operating in cluttered urban environments. For small-sized UAVs they have identified that the 50 feet near the earth's surface is the most dangerous, due to dynamic obstacles, uncertain wind currents in urban landscapes, and safety issues related to the vehicle's power limitations or a complete system failure. Therefore NASA is leading a research program called SAFE50, to develop a Safe Autonomous Flight Environment for the 55 lb class of UAVs operating in the lowest 50 feet of airspace [2].

In developing autonomous UAVs, some of the open research problems are similar to those with ground-based robots, whereas other problems are unique. For example, the general object recognition problem and current approaches are similar for all types of robots, however the specific challenge is developing perception algorithms that can be run on the sort of low-power, small form-factor computer that will physically fit on a UAV. The robot localization problem is similar to that for self-driving cars, using LiDAR or camera-based odometry techniques, however the sensors on a UAV must be more powerful if the vehicle is operating high above the earth surface. Also obstacle avoidance algorithms are similar to those used by high-speed self-driving vehicles. One unique challenge is in developing control algorithms for autonomous UAVs because these vehicles are inherently unstable. Whilst a ground robot can safely come to a sudden stop, with the risk of hitting another object, a UAV also risks falling out of the sky and becoming a projectile.

## 1.2 Problem Motivation

The typical approach for autonomous UAV navigation in outdoor environments is to use GPS localization and to plan paths comprising of sparsely-located waypoints [3, 4]. The assumption is that the environment is mostly free of obstacles, and obstacle-avoidance is achieved using a reactive algorithm.

Recent work has shown that LiDAR- and VO-based methods can enable GPS-denied localization for multi-rotor MAVs [5] and larger UAVs [6].

Also Zhang et al. have combined a LiDAR-based odometry system with waypoint-following to enable a light-weight industrial UAV to autonomously fly between sparse waypoints, without using a GPS [7].

We extend the work of Zhang et al. [7] and develop a method to plan paths with dense waypoints and continuously-varying curvature, to automatically generate paths that enable the UAV to fly around obstacles and through overhanging structures. The LiDAR odometry and mapping system creates a static map of dense 3D points (a point cloud), for environments of more than 500 $m^2$. We propose to exploit the rich detail of the static map, and to reduce online computation time, by pre-planning a global path directly in the point cloud.

The problem of planning paths through large point clouds is interesting because not many research groups have access to large 3D maps and so there has been little work in this area. It is challenging because storing, manipulating or visualizing large point clouds requires a large amount of computer memory and specialized algorithms. And typical approaches to

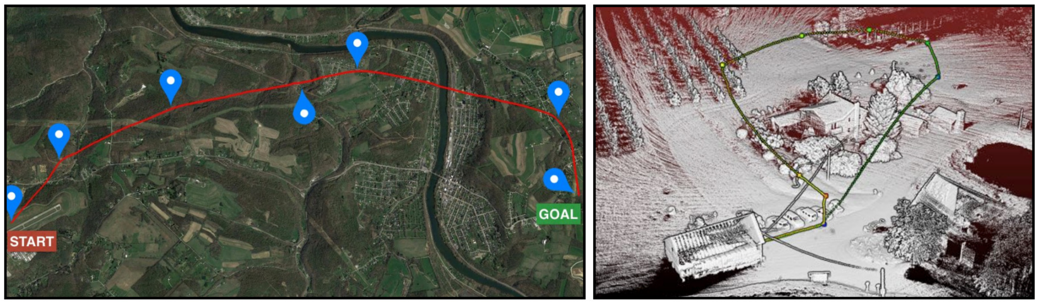


**Figure 4:** Typical approaches to autonomous UAV navigation in outdoor environments use paths planned through sparse waypoints.
(left) A path of [lat.,long.] waypoints in a 2D map, using GPS localization [4].
(right) A path of [x,y,z] waypoints in a 3D map, using LiDAR/VO localization [7].

robot motion planning don't scale to large state spaces.

In this work we investigate various methods for storing dense 3D maps and algorithms for path planning. We describe a hierarchical planning system that generates smooth take-off and landing motions for the vehicle and adheres to altitude constraints. Additional post-processing adds velocity information to the path, to create a trajectory. To achieve obstacle-avoidance, the system pre-computes a road-map of alternative paths so that when the UAV is flying it only has to check which paths are safe to travel.

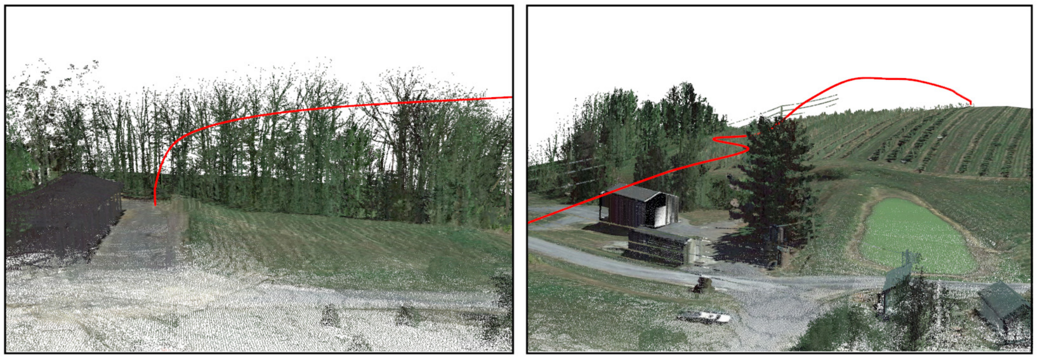


**Figure 5:** Our mission planner produces paths with continuously-varying curvature and densely-located waypoints. The red line shows an example of a main path generated by our planner for a point-to-point mission flying at a fixed altitude. Not shown are the alternative paths used for obstacle avoidance.
(left) The take-off phase consists of a short vertical segment and a smooth transition into the mission phase.
(right) The path avoids obstacles, like the tree, and moves over the ground terrain at a fixed altitude.

## 1.3 Problem Statement

The specific problem we are trying to solve is:

We have a multi-rotor UAV that we want to fly autonomously from a start position to a goal position in a static map. It should follow a continuously-varying curved path (a global path) consisting of densely-located waypoints at 0.10m separation. We need to model the environment with sufficient detail in 3-dimensions, to plan paths through overhanging structures like doors or passageways. The path should be collision-free with respect to the static map. The path should be as smooth as possible, without sharp changes in curvature, so the UAV can fly at maximum forward speed.

For system modeling, we treat the low-level flight controller that is built-in to our experimental UAV platform as a "black box", so we don't consider rotor torque or thrusts. We model the UAV as a point mass with position $(x, y, z)$ and attitude $(forward\ velocity, altitude, roll, yaw)$. However, for planning, we decouple this model and plan in a hierarchical manner:

1. Plan a path in position space
2. Add velocity information to path
3. Follow the path using a trajectory-tracking controller

The discrete path planning problem (1) can be formally defined as follows:

We have **a path** $p$ of waypoints,

where $p_i = (x_i, y_i, z_i)$ for $i = 0$ to $n$, and $\|p_i, p_{i+1}\| = 0.10\text{m}$

**A world model:** obstacles $w$, elevation $e$

**Find the shortest path:**

$$\min \sum_{i=0}^{n} \|p_i, p_{i+1}\|$$

s.t.

$p_0 =$ start position

$p_n =$ goal position

curvature$(p_{i-m}, p_{i+m}) < c$

$altitude_{min} <$ getAltitude$(p_i,\ e) < altitude_{max}$

inCollision$(p_i,\ w) =$ false

# 2 Related Work

There is a large body of work related to motion planning for various types of robots, as well as algorithms specifially for planning and obstacle avoidance for UAVs.

## 2.1 Motion Planning Algorithms

The general motion planning problem is to make the vehicle smoothly fly between two poses, without hitting any obstacles. Ideally the robot is represented by a 3D model, the world and obstacles are all 3D surfaces, and we want to find the shortest path to the goal whilst adhering to constraints such as smooth acceleration. However it is widely known that continuous motion planning problems like this are computationally intractable [8]. Therefore, two practical approaches are to discretize the problem and solve it within that discrete resolution, or to decompose the problem into sub-problems that can be solved in a hierarchical fashion.

### 2.1.1 Graph-based Search

When a planning sub-problem can be represented by a graph then we can use graph search to find an optimal solution. Examples include topological maps or roadmaps (discussed below). Algorithms include Dijkstra [9] which is a type of brute-force search, and A* [10] that uses a heuristic to avoid searching the entire space.

Some key benefits of this type of planner are that they are "complete" and will return an exact solution. They will also report if there is no solution.

For example, A* can be used to find the shortest path for a UAV from a set of waypoints. Note that there are many variants of A* that have other properties. For example, Anytime Repairing A* [11] trades-off planning time for optimality, by iteratively planning a sub-optimal solution that gets better over time.

One drawback is that a lot of memory is required to store the vertices and edges. It is possible to use dynamic memory allocation or store the graph out-of-core, however this greatly increases the time to search the graph.

An example of such a problem is finding the shortest path between two geographical locations on a graph of the US road network, which contains approximately 24 million vertices and 58 million edges. Bast et al. developed an algorithm called TRANSIT [12] to pre-process this graph into a hierarchical structure in order to increase lookup times. They demonstrated shortest path queries in 10 microseconds, after taking 20 hours to pre-process the graph.

### 2.1.2 Discrete, grid-based planning

A typical path planning problem for a robot can be solved by discretizing the world model into a 2D grid of cells or a 3D grid of voxels. Grid-based methods are related to graph-based methods in that a graph can be constructed from the grid. We call this an explicit graph, which can be searched using various searched-based planning algorithms. In contrast, grid-base planning algorithms typically use an implicit graph structure where the planner operates directly on the grid, without using the time or memory required to build an actual graph.

The grid must have an associated connectivity that defines how the robot is allowed to move from one cell to the next. A 2D grid of square cells can be 4-connected or 8-connected between adjacent cells. A 3D grid of cubes can be 6-connected or 26-connected, if motion is allowed between all cells that have a neighbouring face, edge or corner. The time to plan through these grid words increases with the connectivity and number of grid cells.

Most approaches for grid-based planning are improvements based on the original A* algorithm. For example, Field D* [13] creates smoother paths by moving the vertices from the center of grid cells to the corners and performing an interpolation step after edge expansion to short-cut linear segments between the vertices. There is also a variant for 3D voxel grids called 3D Field D* [14].

Removing this restriction on strictly planning 90°and 45°movements has created a sub-class of methods called any-angle path planning. For example, Theta* [15] produces shorter paths than Field D* or A*-with-smoothing, in faster time and with less vertex expansions.

However, it is important to note that any path made on a grid-based lattice will be sub-optimal. Any-angle path planning algorithms are trading off between planning time versus optimality when compared to A* with smoothing.

### 2.1.3 Roadmap Methods

Roadmap-based planning methods are used to generate a graph that represents the robot's environment or state-space that can be searched using algorithms like A*. Roadmap methods are most useful for multi-query plan-

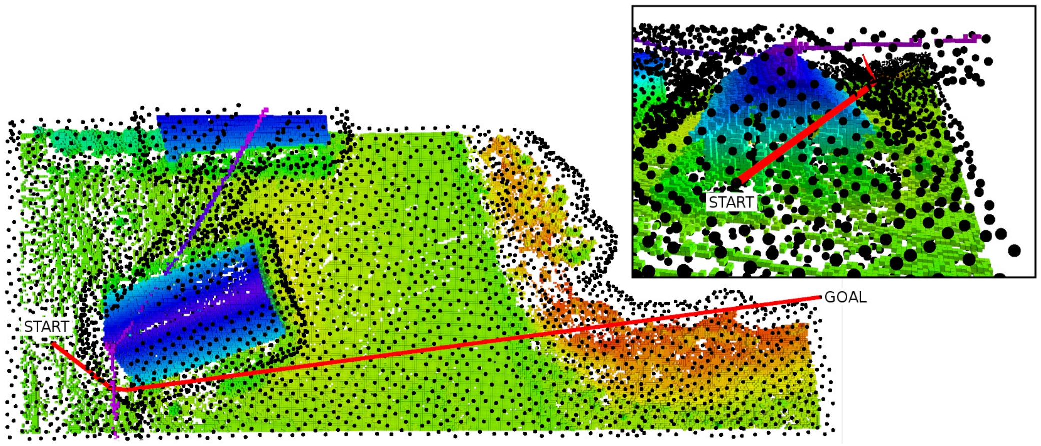


**Figure 6:** A path produced by SPARTAN [16], which is a type of discretized Visibility Graph. The voxel grid map is shown colored by height. The black dots are the graph vertices which are at a fixed distance tangent to the surface of obstacles. Image by the author.

ning, where a single roadmap is generated in advance and later queried using varying start and goal configurations. For example, in games programming, a roadmap called a Navigation Mesh or Graph is pre-computed and used to generate paths for characters directly on the 3D mesh that represents that game world [17].

A visibility graph is a method that can be used to find the shortest path between obstacles represented by polygon shapes [18]. The graph is created by treating the obstacle vertices as the nodes in the graph, and connecting edges between any pair of vertices that are visible to each other. The shortest path can be imagined like pulling a string tight between the polygonal obstacles. For example, Schøler used a 3D visibility graph to plan shortest paths around polygonal obstacles for an autonomous helicopter [19].

One problem with the original visibility graph method is that it quickly uses a lot of memory. Instead of expanding all edges to create a complete visibility graph, Kneidl et al. use a cone heuristic to create a reduced visibility graph and search only nearby vertices [20].

In modern robotics, the world is modelled using point clouds, and obstacles can not easily be represented as polygons. Cover et al. created a type of discretized visibility graph called SPARTAN (Sparse Tangental Network) [16] where the vertices are evenly distributed by performing a distance transform on the point cloud. Planning time is improved by using an angle-based heuristic that determines which edges are expanded. However, when the distance to the next set of vertices is large, a small search-angle heuristic still results in too many edge expansions. Also, the map data structure is a 3D voxel grid, which like many methods limits the size of the map.

A topological roadmap is a sparse representation of the world using a graph of distinct places. Each vertex is located in the free space between obstacles and if multiple neighbouring vertices are visible, edges are only added that give the best representation of the map's topology. Beeson et al. developed an approach for identifying unique places on a map, like intersections, corners or dead-ends in corridors [21].

A voronoi diagram is a raster image of points that are located in the middle of the free-space between obstacles. One benefit of using a voronoi-based roadmap is that it represents all of the toplogically-distinct paths, or alternate routes, between obstacles. It is also called the medial axis or skeleton of the free space of a map. Lau et al. developed a voronoi based map system that can be updated in real-time [22].

### 2.1.4 Probabilistic Sampling-based Methods

Instead of trying to explicitly model a robot's configuration space and the free space in the environment, probabilistic planning methods find a solution by sampling from the state-space to build a graph of valid states. These methods are probabilistically complete, which means that as the planning time increases, it is more likely to find a solution. However there is no guarantee that the plan will be optimal and the planner can't tell you if there's no solution. For example, the PRM (Probabilistic RoadMap) [23] samples states first and then connects them to create a roadmap. In contrast, the RRT (Rapidly-exploring Random Tree) [24] builds a graph as it goes by interleaving state sampling and edge expansion. In practice these methods produce a solution very quickly, but the solution has sporadic random motions and requires post-processing to smooth the robot's behavior.

A further improvement is asymptotically optimal planners, which will also converge towards the optimal solution as planning time increases. For example, RRT* [25] adds an optimization step that tries to prune any non-optimal sub-trees in the vicinity of new vertices. In practice, RRT* is slower than using RRT with post-processing, when applied to problems like path planning or manipulator motion planning. Gammell et al. developed Informed RRT* [26] which reduces the planning time required to find an optimal solution by only optimizing within an ellipsoidal space between the start and goal.

Currently the state-of-the-art combines probabilistic methods in a hierachical approach. For example, BIT* [27] combines Informed RRT* and dynamic programming, resulting in an algorithm that can produce optimal solutions 3x faster. BIT* is also an "anytime" algorithm, which means that it can produce a sub-optimal solution quickly and that solution gets better by letting the planner run for more time. Choudhury et al. developed RABIT* [28] with combines BIT* with a local optimizer, and can more quickly find paths through thin passages of free space.

## 2.2 Autonomous UAV Navigation

A typical approach to achieving autonomous navigation of a UAV is to use a hierarchical planning scheme. First, a global planner is used to find a path to the goal position in a large low-resolution map of the environment. The assumption is that the world is static and the global planner prevents the UAV getting stuck in dead-ends. Then a local planner is used with a small high-resolution map to plan short maneuvers or avoid obstacles.

Scherer et al. [29] use a hierarchical planning scheme for a small unmanned helicopter. A probabilistic occupancy map is maintained in real-time during flight and a potential fields method is used for planning. A global planner finds a feasible path to pass through a set of sparse GPS waypoints that are located hundreds of meters apart. Then a local, reactive planner is used to avoid obstacles that appear on the current heading direction.

Dryanovski et al. [30] developed a navigation system for a micro-sized quad-rotor aerial vehicle. A 3D model of the world is built as the UAV flies around and this is projected down onto a 2D costmap which is used for planning. The costmap obstacles are inflated by the robot's radius so that planning can be done by assuming the robot is a point mass. A global path is planned on the grid using Dijkstra and an algorithm converts this grid-based path into a set of sparse waypoints. If an obstacle appears then the planner must re-plan the path. A position-based controller is used to reach the next waypoint.

Various work has used motion primitives as part of the planning system. MacAllister et al. [31] implemented path planning for a non-circular quadcopter by using motion primitives with AD* to find the optimal path on a state lattice. Fang et al. [32] demonstrated autonomous navigation on a micro aerial vehicle, using a 2D costmap similar to Dryanovski. A global path is found by running A* on a voronoi diagram, which is then sampled at 5m intervals to product a set of sparse waypoints. The local planner is a receding-horizon method that both keeps the robot headed on the nominal path and also avoids obstacles. At each time step, a path is chosen from a path library, which is a set of pre-generated motion primitives of varying curvature and direction. The chosen path primitive is the one which minimizes the distance to the next waypoint in the global path. Lastly the path is optimized to minimize smoothness and maximize distance from obstacles.

Droeschel et al. [33] utilize a multi-layer planning system for a micro aerial vehicle used for mapping buildings. A global planner is used to determine optimal viewpoints of a target building and to plan the shortest path that loops around the waypoints. In the next layer, a local planner is used to find the shortest path using a multi-resolution voxel grid, to speed-up planning time. At the low level, another local planner is used to avoid obstacles, based on the potential field method.

In summary, the general approach to navigation over long distances is to plan a path between sparse GPS waypoints on a 2D map. For planning through cluttered scenes, a dense 3D occupancy map is typically used. However, due to the large amount of memory required to store occupancy grids, this method does not scale-up to large environments. In contrast to this, our work both demonstrates planning over long distances and through cluttered environments, using a path with waypoints at 0.10m separation.

## 2.3 Terrain Classification

For mission planning it is useful to have a world model labelled with semantic information. Having knowledge of the ground terrain, roads, overhead power lines and buildings means that we can plan safer paths. In this work we classify the ground terrain so we can determine the relative altitude of the UAV's position. We also experiment with using road data overlays obtained from a GIS database.

In the GIS field, a ground terrain map is called a DEM or DTM, and there is a large body of work related to generating ground terrain models. For example, Hu et al. demonstrated an online system that can analyse and classify streaming data from LiDAR sensors [34]. However, because we are doing mission planning offline, we use a simpler offline method for classifying the ground surface in a point cloud.

Axelsson [36] developed an algorithm for detecting the true earth surface from airborne or terrestrial LiDAR. The mesh surface is represented by a TIN (Triangular Irregular Network), and is fitted to the point cloud from below. Initially a coarse set of seed points is chosen to define the TIN, which is then iteratively refined such that some parameters are within tolerance. The tunable parameters are the distances to the facet planes and angles to the nodes.

MCC (Multi-scale Curvature Classification) by Evans and Hudak [37] is an iterative algorithm that classifies points as ground or non-ground, based on the curvature of that region. A coarse raster surface is interpolated using thin-plate splines and the 12 neighbours for a specific point. If the curvature exceeds a tolerance then the point is classified as non ground. The algorithm interpolates a surface at multiple scales and is run until convergence in each scale domain.

Brodu and Lague [38] created an algorithm for classifying complex scenes using terrestrial LiDAR. They use a multi-scale feature that measures the dimensionality of the point cloud data relative to a specific point, combined with supervised machine learning. Each feature is a sphere located at one point, with the radius being the scale. The feature characterizes if the neighboring points contained within the sphere are predominantly on a 1D surface (a line), a 2D surface (a plane), or are 3D. In the training phase, the user

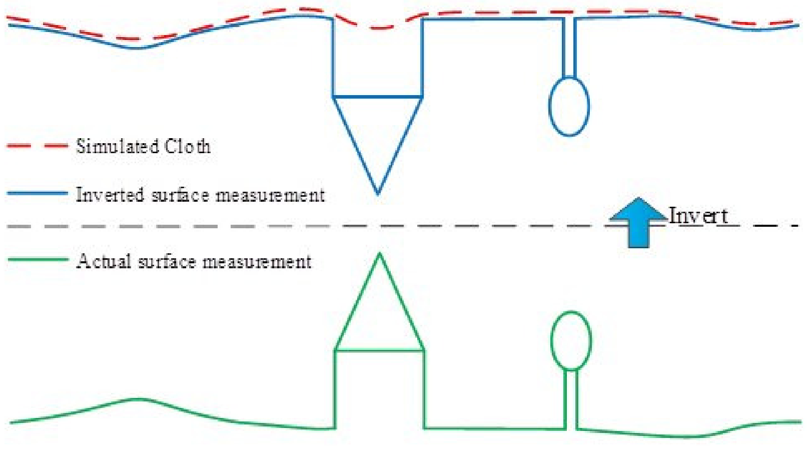


**Figure 7:** One method of ground point classification from LiDAR data is to fit a model of a cloth surface to the point cloud from underneath [35]. The accuracy of the detected ground surface is dependent on how closely the model touches the points or sags in-between them.

manually segments the training data and the learning system automatically calculates the set of scaled features which will best classify the data.

Zhang et al. [35] developed a method for segmenting the ground terrain from airborne LiDAR data by using a cloth simulation. The point cloud is turned upside down and a virtual model of a cloth is draped over the points., which becomes the detected terrain surface. The amount that the cloth deforms to fit the points is dependent on parameters which define the virtual forces between the nodes of the simulated cloth. A post-processing step makes the cloth fit closer to points that define any steep gradients.

# 3 Mapping

## 3.1 Creating a 3D map

First we create a large and detailed, static 3D map of the world which is used for planning a path for the UAV. To create this point cloud map we use the "Stencil" LiDAR- and VO-based mapping device [39], which is manufactured by Kaarta and based on Zhang and Singh's work called LOAM [40] and VLOAM [41].

The Stencil mapping device produces dense 3D point clouds and is quite accurate as an odometry system for relative motion in outdoor environments. Using a global map, it can localize to within 0.16m [42]. Fig. 8 shows the Stencil in hand-held configuration, for 3D mapping, and also mounted to our experimental UAV platform, for localization.

The static map is created by walking the Stencil through the entire en-

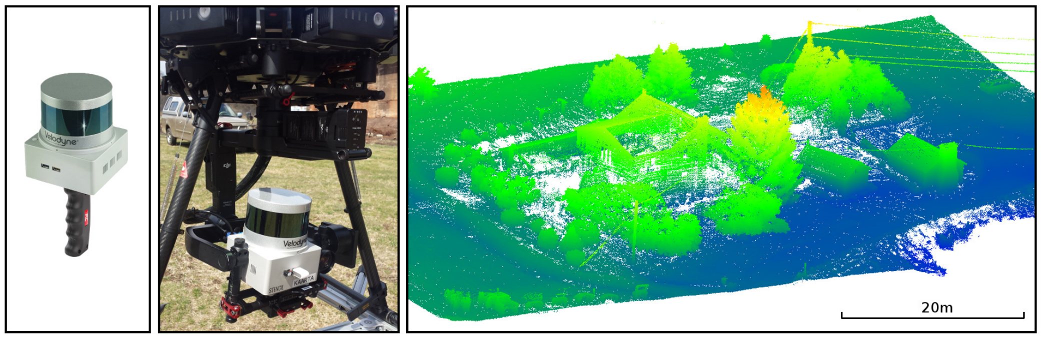


**Figure 8:** (left) The Stencil mapping device in hand-held configuration, which we use to map an environment.
(center) The Stencil is mounted to our UAV platform, a DJI Matrice 600.
(right) The resulting point cloud. Note the points are dense where the mapping operator used the hand-held device, and less dense in the middle of the scene. The LiDAR detects powerlines (image top-right) but fails to detect the body of water (image bottom-right).

vironment in which the UAV will eventually be flown. It is not possible to create an airborne map, or for the UAV to localize, without creating at least a small ground-based map first. This is because when the UAV is in the air, the fast motion and lack of 3D structure cause problems for the LiDAR scan matching. So the ground-based static map acts as an "anchor" for the LiDAR scan-matching and additional motion estimation is provided by VO with a camera pointed down at the ground.

The resulting point cloud is a list of 3D points with the z-axis is pointing "upwards", as defined to be opposite to the gravity vector from the Stencil's IMU sensor. The origin of the point cloud where $x = y = z = 0$ is the initial position where the Stencil was initialized.

## 3.2 World Modeling

After creating a map of the world, the resulting 3D point cloud is converted into a representation that can be used by the path planner. This world model consists of two data structures: an Octree containing the 3D points of obstacles, and a 2D grid representing the height of the ground. This section describes these in more detail.

In robot mapping and planning, it is typical to create an explicit representation of the occupied space (the obstacles), the free space and the unknown space. For example, one approach is to discretize the space into a grid at some resolution and store information about the status of each block of space in one of the following data structures:

- Elevation/height map
- Multi-level surface map

- Voxel grid
- Multi-resolution voxel grid (octree)

These approaches are commonly used with discrete, grid-based planning methods, but can also be queried by probabilistic planning algorithms. The benefits of a grid-based map are that it can:

- Store data for each cell, like "cost to go" or "visibility to object"
- Use shortest path algorithms
- Compute distance transforms, used for distance-to-obstacle queries
- Compute a voronoi graph, for knowledge of maximum distance to obstacles

However, storing a complete representation of the world uses a lot of memory.

An alternative approach is to only store information about the occupied space or obstacles. The assumption is that all the other unknown space is free-space. Such data structures include:

- A surface mesh
- List of points (a "point cloud")
- List of solid voxels
- Tree of points (octree, kd-tree , binary space partition)

We are concerned with planning paths for UAVs through cluttered environments, including through doorways or overhanging structures, therefore we need a high resolution map that can sufficiently represent the scene with sufficient detail. Fig. 9 shows the result of using various voxel sizes to represent an environment, so we use a resolution of 0.10m which allows the planner to find a path through tight spaces.

We are also concerned about planning through point clouds of more than

1 km$^2$, therefore we need to choose a data structure that has low memory usage. Fig. 10 compares the memory requirements of various data structures for 3 point clouds of varying size. We define a "small" point cloud as one that can be stored in a dense map representation, such as a 3D cost map or voxel grid; a "large" sized point cloud to be one that is large enough that it must be stored in a sparse map representation, such as a point cloud or octree; a "massive" point cloud is one that is so large it must be stored out-of-core (not covered in this work).

The planner also needs to query the world model for collision-checking. Fig. 11 compares the collision-checking speed for various data structures for a sphere shaped robot. It can be seen that the most memory efficient way of storing point data is also the slowest structure to use for collision checking. Therefore we model the world using an efficient Octree representation that allows both fast collision checking and acceptable memory usage.

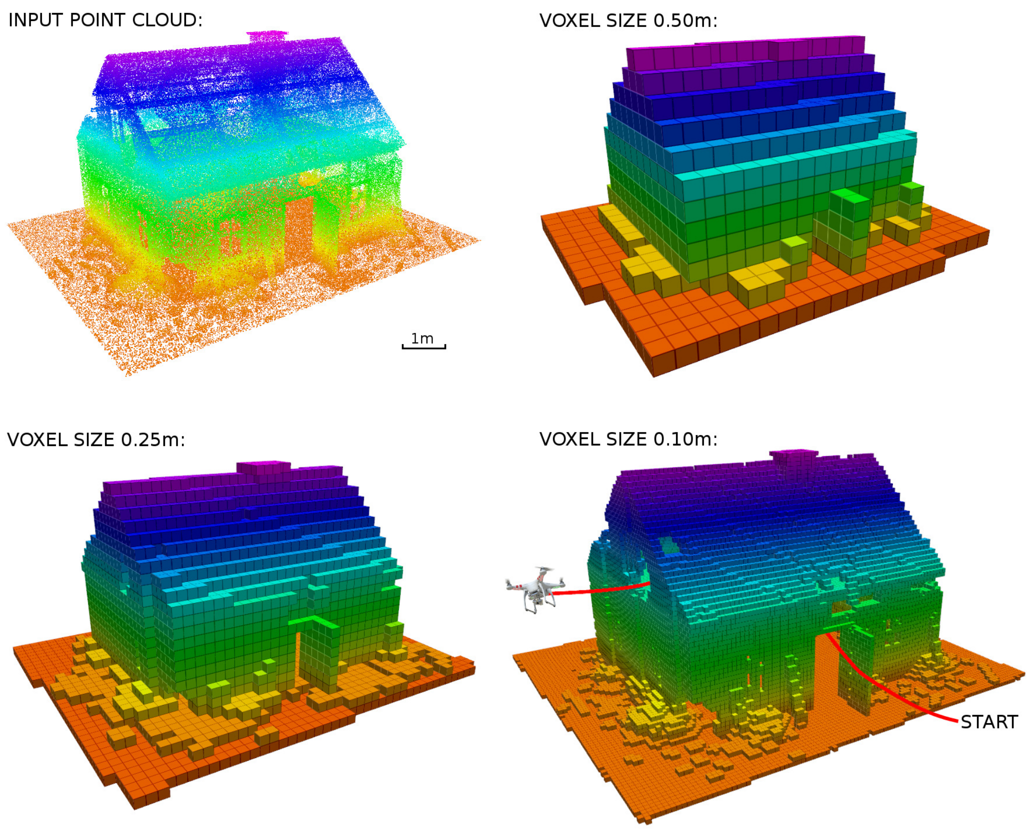
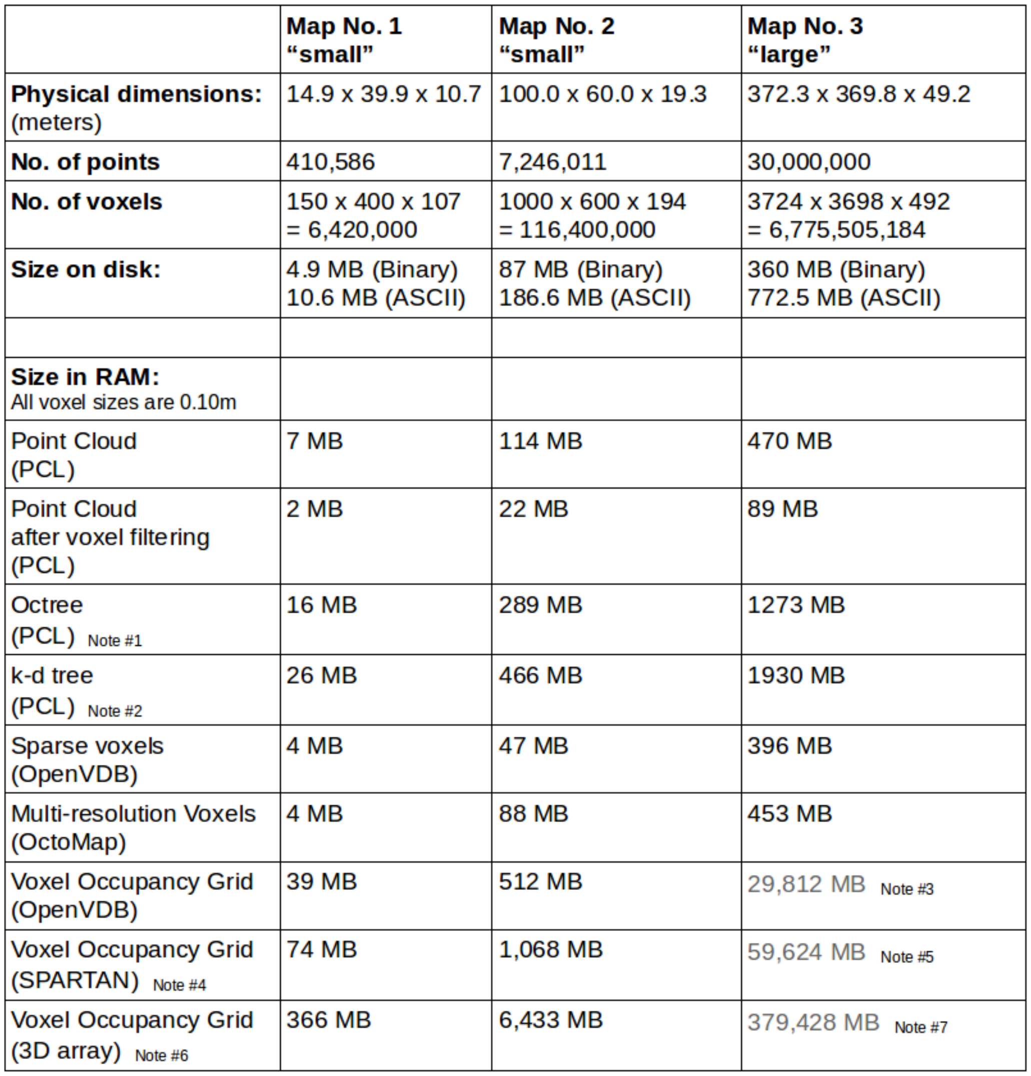


**Figure 9:** A comparison of the world model resulting from various voxel sizes. We take the input point cloud and store it in an Octree with 0.10m resolution, which allows the planner to find safe paths through cluttered environments. The bottom-right image shows a real path generated by our planner, that allows a MAV to explore inside a derelict house. A diameter of 0.60m was used for collision-checking and one intermediate waypoint is located inside the building at the top of the staircase to force the planner to find a through-path.



| | **Map No. 1 "small"** | **Map No. 2 "small"** | **Map No. 3 "large"** |
|---|---|---|---|
| **Physical dimensions:** (meters) | 14.9 x 39.9 x 10.7 | 100.0 x 60.0 x 19.3 | 372.3 x 369.8 x 49.2 |
| **No. of points** | 410,586 | 7,246,011 | 30,000,000 |
| **No. of voxels** | 150 x 400 x 107 = 6,420,000 | 1000 x 600 x 194 = 116,400,000 | 3724 x 3698 x 492 = 6,775,505,184 |
| **Size on disk:** | 4.9 MB (Binary) 10.6 MB (ASCII) | 87 MB (Binary) 186.6 MB (ASCII) | 360 MB (Binary) 772.5 MB (ASCII) |
| | | | |
| **Size in RAM:** All voxel sizes are 0.10m | | | |
| Point Cloud (PCL) | 7 MB | 114 MB | 470 MB |
| Point Cloud after voxel filtering (PCL) | 2 MB | 22 MB | 89 MB |
| Octree (PCL) Note #1 | 16 MB | 289 MB | 1273 MB |
| k-d tree (PCL) Note #2 | 26 MB | 466 MB | 1930 MB |
| Sparse voxels (OpenVDB) | 4 MB | 47 MB | 396 MB |
| Multi-resolution Voxels (OctoMap) | 4 MB | 88 MB | 453 MB |
| Voxel Occupancy Grid (OpenVDB) | 39 MB | 512 MB | 29,812 MB Note #3 |
| Voxel Occupancy Grid (SPARTAN) Note #4 | 74 MB | 1,068 MB | 59,624 MB Note #5 |
| Voxel Occupancy Grid (3D array) Note #6 | 366 MB | 6,433 MB | 379,428 MB Note #7 |

**Figure 10:** A comparison of the memory requirements for various data structures used to store a 3D world model.

Note 1: The Octree structure in PCL is a pointer to the data contained in a point cloud, so the memory usage includes the original point cloud.
Note 2: k = 3 dimensions. The k-d tree structure in PCL is a pointer to the data contained in a point cloud, so the memory usage includes the original point cloud.
Note 3: Estimated size is 30 GB, based on 4 bytes per voxel + 10% overhead = 6,775,505,184 x 4 x 1.1
Note 4: Voxel Occupancy Grid used by SPARTAN uses a memory efficient char* structure, and supports scrolling and indexed incremental distance transform (iidt).
Note 5: Estimated size is 60 GB, based on 8 bytes per voxel + 10% overhead = 6,775,505,184 x 8 x 1.1
Note 6: Voxel Occupancy Grid is a multi-dimensional std::vector of Int, which is not the most memory efficient implementation.
Note 7: Estimated size is 380 GB, based on 56 bytes per voxel = 6,775,505,184 x 56

| | Map No. 1 "small" | | Map No. 2 "small" | | Map No. 3 "large" | |
|---|---|---|---|---|---|---|
| | AVERAGE TIME TO CHECK COLLISION (ms) | | AVERAGE TIME TO CHECK COLLISION (ms) | | AVERAGE TIME TO CHECK COLLISION (ms) | |
| | in-collision | collision-free | in-collision | collision-free | in-collision | collision-free |
| Octree (PCL) | 0.002 | | 0.003 | | 0.004 | |
| | 0.003 | 0.001 | 0.004 | 0.002 | 0.006 | 0.001 |
| k-d tree (PCL and FLANN) | 0.005 | | 0.009 | | 0.003 | |
| | 0.004 | 0.006 | 0.007 | 0.012 | 0.004 | 0.002 |
| Multi-resolution Voxels (FCL Sphere and OctoMap) | 0.010 | | 0.012 | | 0.009 | |
| | 0.015 | 0.005 | 0.018 | 0.018 | 0.014 | 0.003 |
| Multi-resolution Voxels (FCL Sphere BVHModel in DynamicAABBTree and OctoMap) | 0.018 | | 0.019 | | 0.019 | |
| | 0.028 | 0.007 | 0.031 | 0.007 | 0.033 | 0.004 |
| Voxel Occupancy Grid (OpenVDB) | 0.074 | | 0.076 | | --- | |
| | 0.028 | 0.120 | 0.026 | 0.126 | | |
| Voxel Occupancy Grid (3D array) | 0.089 | | 0.098 | | --- | |
| | 0.032 | 0.146 | 0.033 | 0.163 | | |
| Point Cloud (PCL) | 1.104 | | 18.776 | | 91.247 | |
| | 0.406 | 1.801 | 6.056 | 31.496 | 51.464 | 131.031 |

**Figure 11:** A comparison of average time taken (in milliseconds) to collision-check the UAV in one position, for 3D maps stored various types of data structures. This shows that an Octree structure is the fastest for collision-checking.

The robot's geometry is approximated with a sphere of radius 0.75m, which is a good approximation for a multi-rotor UAV, and is the geometric shape which is fastest to collision-check.

This test was conducted by drawing 10,000 random (x,y,z) positions for the robot, then testing each data structure using the same set of positions.

For each type of map there are 3 numbers shown above: the first is the average time for a collision-check, then the average time when the result is "in collision", and the average time when the result is "collision free". Some data structures are much faster when the result is "collision free", such as Octree, whereas others like Point Cloud are much slower.

## 3.3 Ground Classification

The height of the ground terrain surface is required in order to plan paths that are constrained to a specific altitude. This is important because the ground is typically the most important obstacle that aerial vehicles must avoid. We could treat the ground the same as any obstacle, which would result in the UAV flying high over valleys and within the minimum distance over hill tops. However it is safer, and generally required by law, that a UAV be operated at a specific altitude during the mission phase.

In our raw input point cloud data, the z-axis is pointing opposite to the direction of the gravity, and $z = 0$ at the position where the mapping device was initialized. The ground height is the z value of a point representing the ground surface in the point cloud. However we have no prior knowledge of which points represent the ground surface.

The difficulty of segmenting the ground surface depends on how the LiDAR data was collected. A point cloud created using SfM will have almost no points underneath trees of bushes, depending on how many aerial images were used. An aerial LiDAR scan will usually result in some points penetrating through trees and hitting the ground. We use a terrestrial LiDAR scanner, which has a high density of ground returns (points representing the ground surface) which is the easiest type of data for ground segmentation.

We use the following process to create a ground elevation map:

1. Classify the LiDAR points as ground or non-ground using a method based on Axelsson's adaptive TIN algorithm [36].
2. For the ground points, fit a mesh surface using Delaunay Triangulation method with a 2.5D assumption.

3. Sample the mesh surface on an x,y grid with a resolution of 100 points per square meter, which matches the 0.1m resolution of our Octree map.
4. Store the resulting x,y coordinates and elevation in a 2D grid data structure for use by the planner.

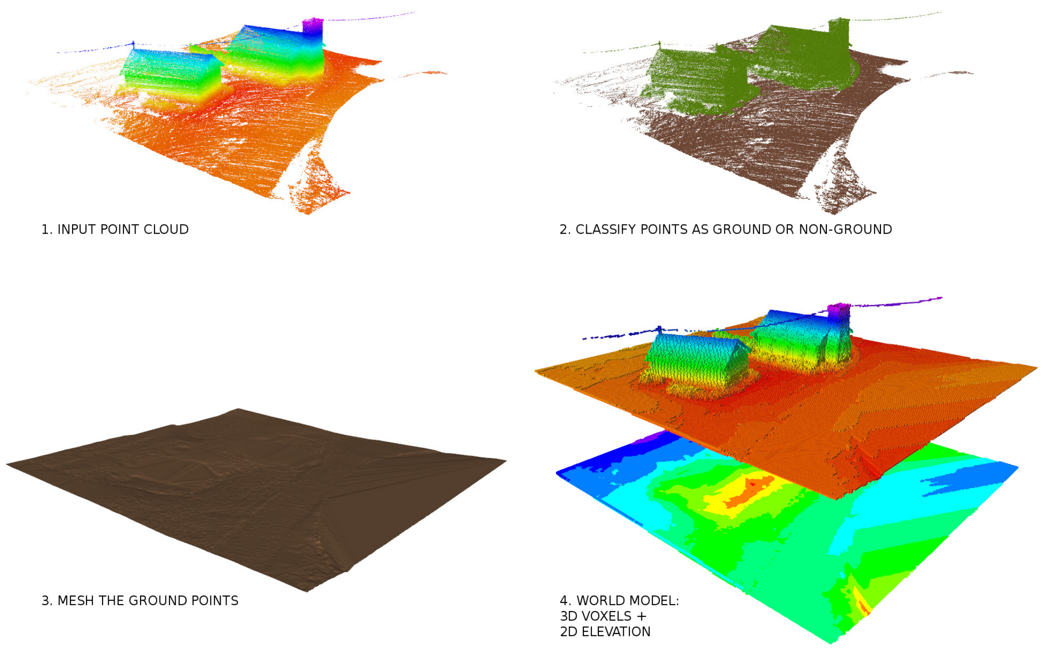

1. INPUT POINT CLOUD
2. CLASSIFY POINTS AS GROUND OR NON-GROUND
3. MESH THE GROUND POINTS
4. WORLD MODEL:
3D VOXELS +
2D ELEVATION

**Figure 12:**
The process of classifying a point cloud to create a ground terrain map.
1: The input point cloud.
2: The points classified as ground and non-ground.
3: The ground points are triangulated into a mesh.
4: The final world model consists of 3D Voxels stored in an Octree (top) and height information stored in a 2D grid.

## 3.4 Public Map Data

Instead of having to manually create a point cloud map of an environment before we fly the drone, it would be convenient to use publicly available maps for localization and planning. For example, various Government agencies occasionally gather aerial LiDAR data, oblique aerial imagery and ortho-photos. Hemann et al. [43] demonstrated improved localization for an autonomous helicopter by matching data from the on-board LiDAR sensor with a publicly available DEM.

In this section we compare the quality of this publicly available map data to that which we collected. Unfortunately we can't use this data for localization because our system requires a much higher density of points, which is only found by terrestrial or very low altitude LiDAR scans. Also, public data is at least 1 year out of date which reduces its usefulness for navigating in urban environments that are constantly changing. However, there are efforts being made to create high quality publicly accessible terrain data using crowd-sourcing [44]. In the future it is likely that citizens and machines will share 3D mapping data of their local city via "The Cloud", to enable autonomous robots to safely operate in constantly changing urban environments.

### 3.4.1 LiDAR DEMs

We examined elevation data of the Pittsburgh area created from a LiDAR survey conducted in 2007 [45]. This data is accessible via an online interactive tool called EarthExplorer [46]. We converted the LAS points data to raster image tiles using LAS Tools. Fig. 13 (top-right) shows 4 DEM tiles overlaid on a map of Pittsburgh, with each tile being approximately 3,000 km$^2$ (10,000 feet$^2$).

The LiDAR data is listed as having a ground resolution of 3.2 feet (1 meter), however by measuring the average density of the points we determined the resolution to be 1.5 meters. Fig. 13 shows a close-up view of the DEM in the area of the Carnegie Mellon Campus, where Hamerschlag Hall is visible as a small red-colored blob.

The ground-return points have already been classified so we can create separate DSM and DTM maps. For each set of points a mesh is created using the Delaunay Triangulation method with the assumption that the data is 2.5D with a vertical z-axis. Fig. 14 (top) shows a surface model, including the buildings and trees, and (bottom) shows a mesh of the bare earth. One problem with public aerial data is that it can be out-of-date. For example, in Fig. 14 (bottom) there is a large depression in the bare earth mesh where the Gates-Hillman building now stands.

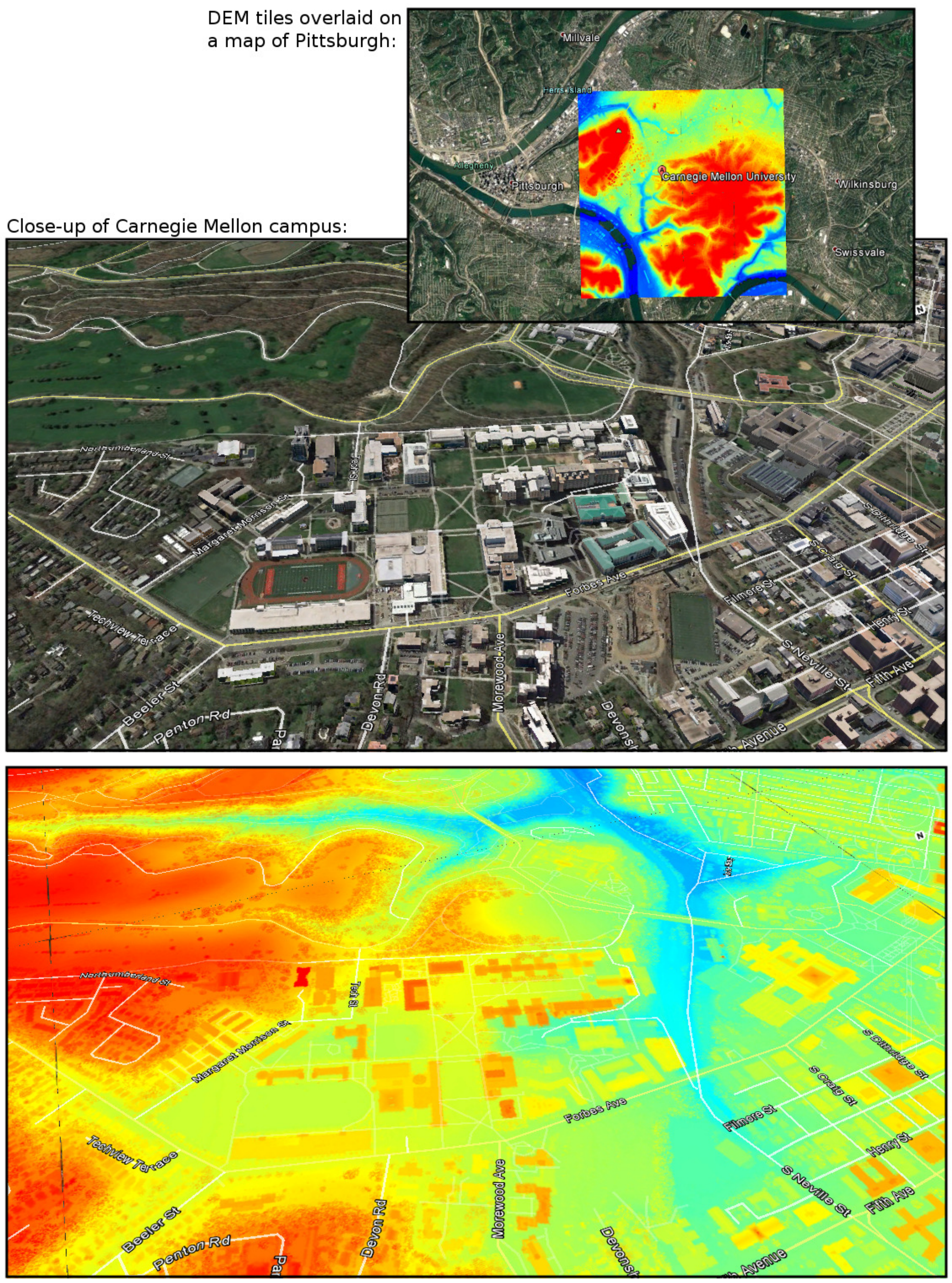


**Figure 13:** The bottom image shows a DEM of the area around the Carnegie Mellon campus, captured using an airborne LiDAR with 1.5m resolution.
The map is colored by height, where dark blue is the lowest altitude of the river bed and red is the highest altitude. This publicly-available data is not useful for air vehicles operating near the ground terrain because it is out-of-date. The Gates-Hillman computer science building is shown in the upper 3D photograph which was captured in 2016, but is missing from the DEM which was captured in 2007.

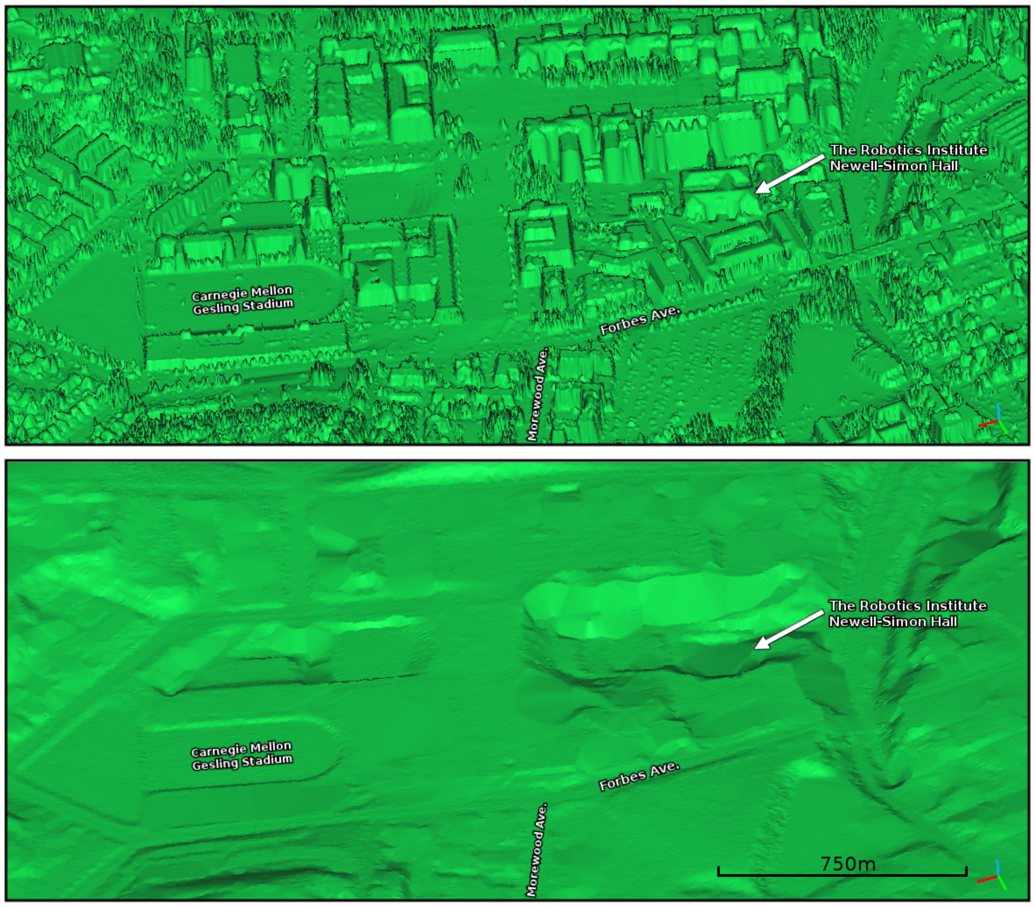


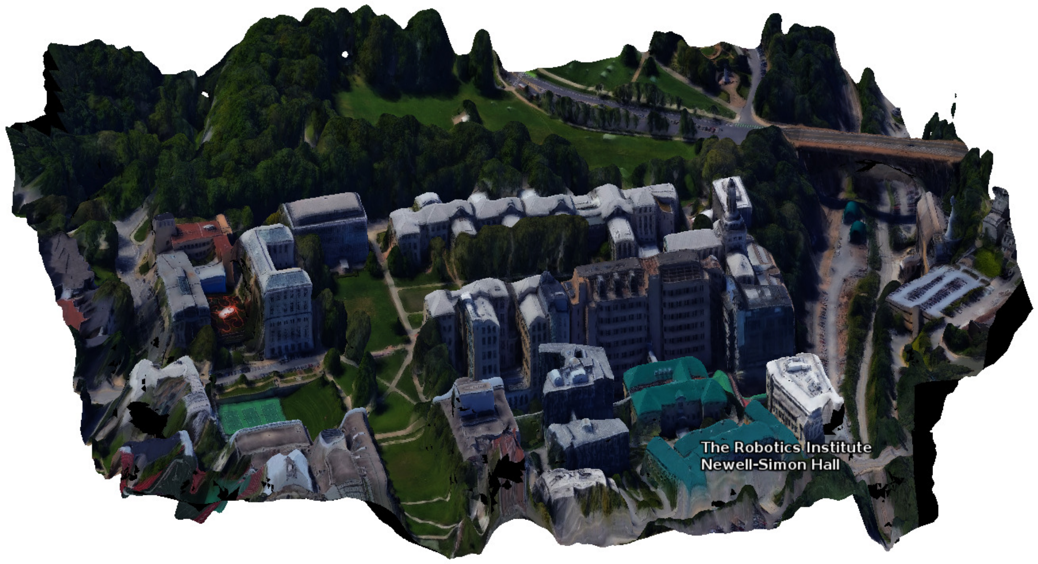
**Figure 14:** This shows the result of converting the 2.5D DEM raster image into a 3D surface. **Top image:** The DSM includes all objects, such as buildings and trees. However, missing is the Gates-Hillman computer science building and the Tepper Quad business building. **Bottom image:** The DTM is the bare earth terrain.

### 3.4.2 Aerial Imagery

An alternative method to create a point cloud map is using aerial images and the photogrammetry technique called Structure from Motion (SfM). Modern aerial imagery is captured from a plane using an array of full-frame cameras that take photographs at 5 angles: nadir (looking down), and 45°oblique angles of north/south/east/west. An example of what this looks like can be seen on Bing Maps "Birdseye View" [47]. Also Sanborn provides a non-free tool for viewing oblique imagery [48].

Using the original, full-frame images it is possible to reconstruct a point cloud of the environment. Fig. 15 (top) shows the result of reconstructing a 3D model of the Carnegie Mellon Campus from oblique aerial images, and (bottom) shows the resulting digital surface model. Note that the result is higher quality than produced from the LiDAR DEM, however there is over-smoothing of sharp edges, which is a common symptom of SfM. This can be improved by using at least two downward angles (30 and 45°) and multiple forward facing angles for each part of the scene.

We can also use aerial images to generate semantic information for specialized mission planning. Potentially we can leverage work in 2D image classification to classify the terrain from aerial images, register that information to our 3D map, and use this to produce "safer" flight paths that follow roads or are biased towards wide-open spaces. Also registering the color information to 3D point cloud maps provides improved visualization and operator awareness.



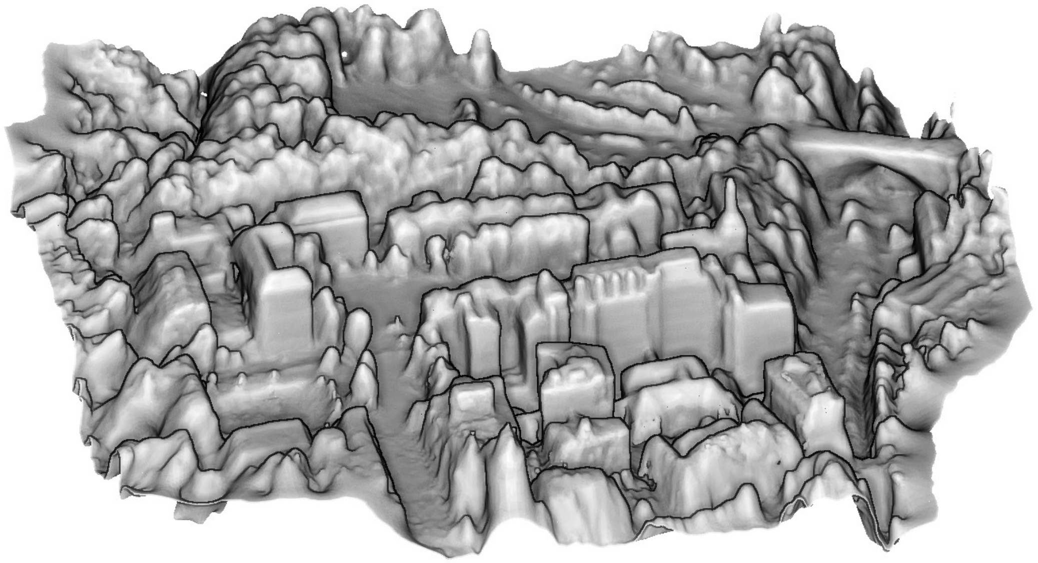


**Figure 15:** A 3D map of the Carnegie Mellon campus, created using photogrammetry.
**Top image:** Multiple aerial images and SfM were used to create a 3D model.
**Bottom image:** The resulting point cloud created from the mesh. The points were sampled very finely such that it is rendered like a smooth surface.
An undesirable result of aerial photogrammetry is that the 3D model does not have any detail around the base of buildings or underneath trees, which restricts our ability to plan through cluttered environments.

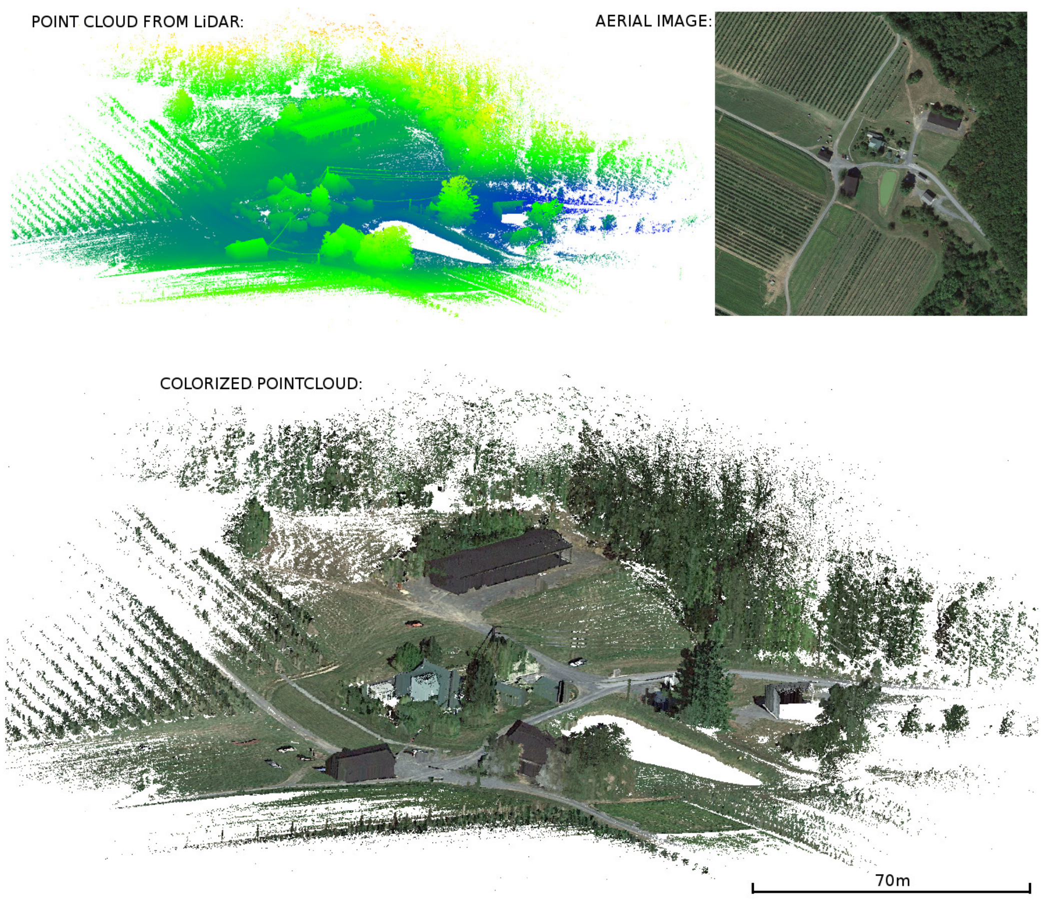


**Figure 16:** We have experimented with using semantic information from GIS databases to enable specialized mission planning, such as knowledge of road networks. Also we can apply recent work in 2D image classification and register that information onto our 3D map.

**Top left:** A terrestrial point cloud captured by the Stencil mapping device.
**Top right:** A set of aerial photographs stitched into one image.
**Bottom:** The result of manually registering the aerial image to the LiDAR data, for visualization purposes.

# 4 Path Planning

The inputs to the planning system are:

- A 3D world model, consisting of an Octree (obstacles) and a 2D grid (ground height map)
- A start and goal position (x,y,z)
- Various constraints, such as altitude and minimum distance to obstacles

## 4.1 Path Planner

To find a path of (x,y,z) positions from point-to-point we use a state-of-the-art sampling-based algorithm called BIT* [27] which is hybrid method combining a RRT and dynamic programming. Like all RRT-based methods, the resulting plans can be "erratic", so we employ a 2-step approach from manipulator motion planning which is find a plan in $t$ seconds of allowable time, and then post-process the path using vertex simplification. This section describes the path planner in more detail.

Whilst plans produced by RRT-based methods can be erratic, a key benefit is rapid exploration of the state space, which is important to quickly return a solution. Even though we use a simple 3-dimensional state space (x,y,z), this space is discretized at a high-resolution and is very large, being more than 500 meters$^3$, which can make planning slow. Fig. 17 compares the performance of various RRT-based planners.

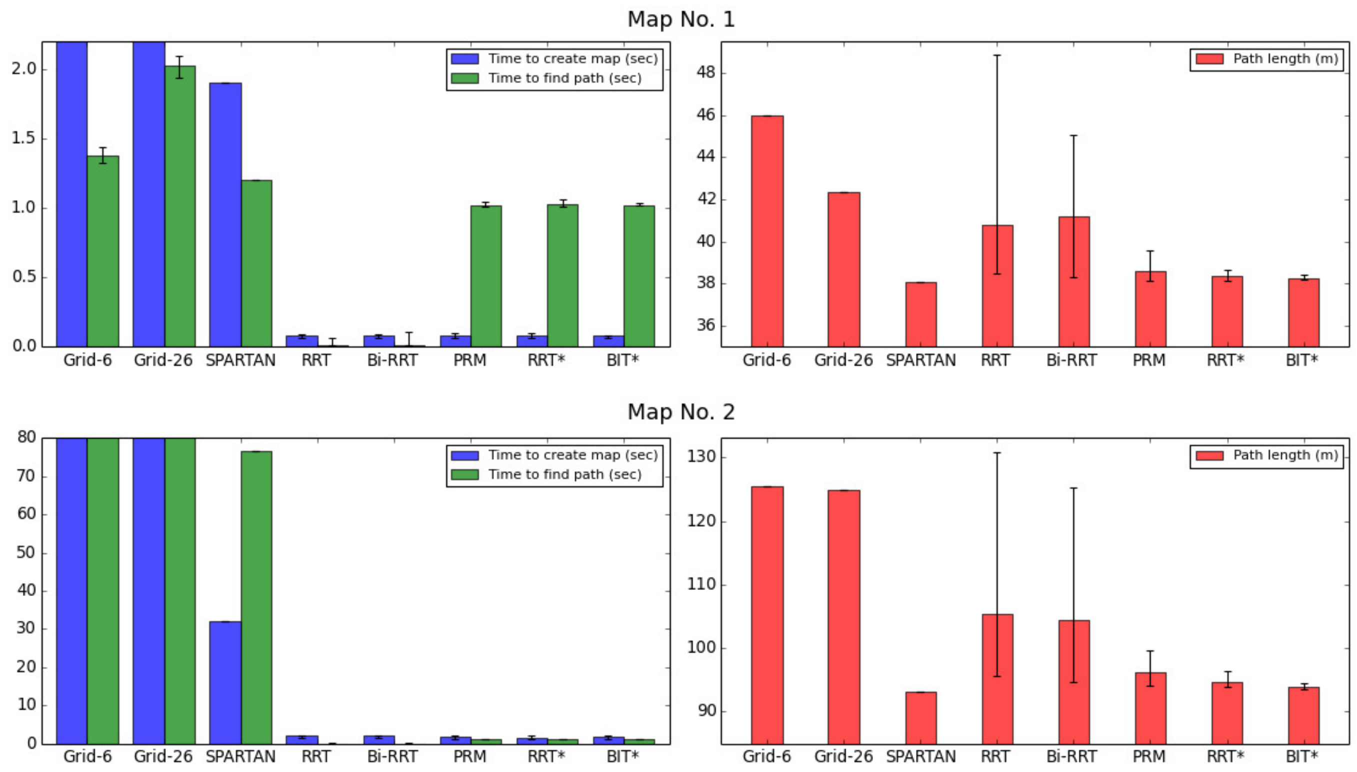
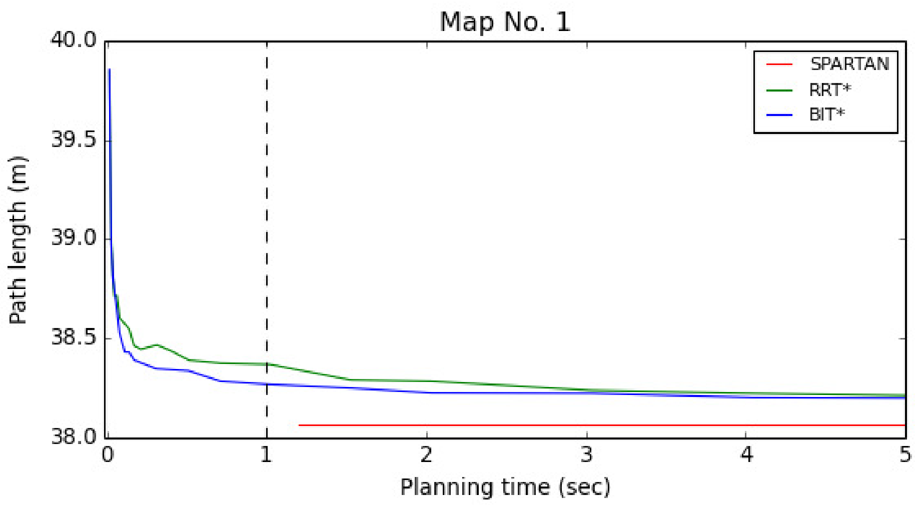


**Figure 17:** This shows the performance of various planning algorithms for point-to-point planning in two sizes of map. Our planning framework uses the BIT* sampling-based algorithm, which generates paths that are within 1% of the shortest path produced using the SPARTAN Visibility Graph algorithm. For large maps, BIT* finds the path and creates the map in significantly less time than SPARTAN.

**Notes:** "Time to create map" is the average time taken to generate the map representation, such as a Voxel Grid or Octree, from the input point cloud.
"Time to find path" is the average time taken to find the shortest path. Except for PRM, RRT* and BIT* which are allowed to run for 1 sec.
For the 5 sample-based algorithms, the time given includes planning + short-cutting.
"Path length" is the average length of the path found by the planner, where the error bars indicate the minimum and maximum paths found.

Map 1: start = (67.0, 2.0, -2.0), goal = (65.0, 39.0, -1.0)
Map 2: start = (13.0, 20.0, -0.5), goal = (97.0, 56.0, 0.0)

The "Grid" method uses a Voxel Occupancy Grid implemented with a 3D array. It takes a long time to create the map because it uses an inefficient method for expanding the occupied cells by the robot's radius. Planning on the grid is done using A* with 6-connected or 26-connected voxels. For small maps the planning time is comparable to other methods.
SPARTAN works well for small maps. However the planning time gets longer as the map size increases due to the larger number of vertices and the longer distance between vertices means the heading-angle heuristic is less effective.
The traditional RRT methods are very quick to find a path but there is high variance in the path length, and no way to optimize the path towards the shortest route.
PRM, RRT* and BIT* are all allowed to run for 1 second and produce similar path lengths. However, BIT* produces the shortest path and with less variance.



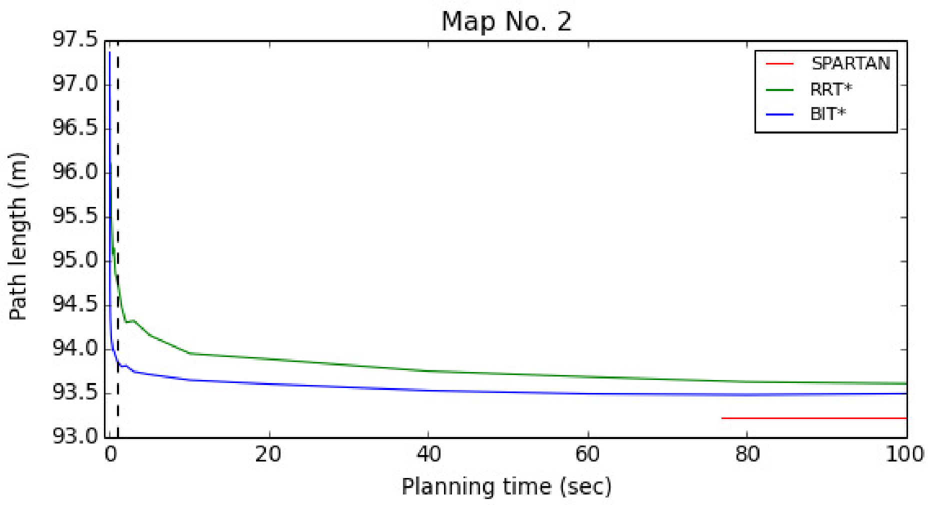


**Figure 18:** This shows the asymptotic performance of RRT* and BIT*, for planning times between "first solution" and infinity.
If allowed to run for 1 second, the sampling-based algorithms achieve a path length which has approximately 1% error relative to path length, compared to the path produced by the SPARTAN Visibility Graph method. If allowed to run for longer, the length of the path approaches that of SPARTAN. The residual difference in path length is because SPARTAN uses a different type of map representation and a different tolerance for distance-to-obstacles.

As shown in Fig. 17, BIT* produces a shorter path that other sampling-based methods, in less time and with less variance in path length. Traditional non-optimizing RRT algorithms often produce a path in the wrong homotopy class and thus that solution can never converge to the shortest path. Fig. 17 also shows the validity of our general approach in using an Octree map representation and sample-based planner, which are faster than a previous state-of-the-art method called SPARTAN [16], that plans on a Visibility Graph generated from a dense 3D Voxel Grid.

BIT* achieves its performance by using a heuristic to focus the state sampling towards the goal and therefore produces a higher-quality solution in less time. BIT* is also an "anytime" planner and will continuously improve the quality of the path when allowed to run longer than finding the initial solution. The final path converges asymptotically to a local minima so only a small amount of optimization provides a nice improvement, as shown in Fig. 18.

The path returned by a sampling-based planner is a set of (x,y,z) waypoints with linear segments, which is further improved using simplification and spline fitting. The simplification process involves examining a pair of non-neighbouring waypoints and trying to fit a linear segment between them. If this segment is collision-free and shorter in distance, then the overall path is updated.

The final step is to fit an interpolating spline between the sparse waypoints of the simplified path and to collision-check the resulting smoothed path. We use a Catmull-Rom spline which is $C^1$ continuous [49, 50], and using a "centripetal" parametrization the resulting has no sharp cusps or selfintersections [51]. The only negative is that a centripetal Catmull-Rom spline has a higher curvature near waypoints than minimum-curvature splines. If the random sampling-based planner produced a path with two waypoints very close to each other, the resulting path after spline-fitting will have high-curvature at this location.

In summary, the overall path planning process is as follows:

1. Load input data (two pointclouds: the 3D point cloud and 2.5D ground surface points).
2. Set start and goal position.
3. Find path using BIT* for $t$ seconds. The vertices and edges of the RRT are collision-checked by doing a fast search for occupied voxels within the robot's radius.
4. Try to simplify path by reducing number of vertices, by short-cutting with linear segments.
5. Fit an interpolating Catmull-Rom spline.
6. Collision check this path again.
7. Sample continuous path at 10cm intervals, to produce discrete path required by control system.

## 4.2 Constraints

To fly the UAV in real-world situations, simple point-to-point planning is not enough. Our planning system also supports various constraints, as listed in Fig. 19. This section describes them in more detail.

Incorporating constraints during planning can be implemented in various ways:

- Intermittently sample from the constraint manifold
- Project samples onto the constraint manifold
- Sample as normal, but reject samples that are not on the constraint

Currently we use rejection-sampling, which has provided acceptable performance, but future work is to use a projection method.

Virtual obstacles are synthetic objects that are added and removed from the world model, and are implemented as a sphere of occupied voxels in the Octree map. This allows the planning system to "hallucinate" different configurations of the world state, and to reason about "what if" an obstacle

| **Constraint Type** | **Constraint Manifold** | **Resulting Motion** |
|---|---|---|
| Virtual obstacles | Sphere | An alternative path |
| Take-off & Landing | Straight line | Linear, vertical motion |
| Ascent (climb) & descent (approach) | Cone (funnel) | Smooth curved transition between vertical & horizontal motion |
| Altitude | Inequality, between min/max surface offset from ground terrain | Primarily forward motion |
| Waypoint | A point | Ends at, or intersects the waypoint |
| Map boundary | Vertical wall | Inside the boundary |
| GIS overlay (roads) | Vertical wall | Inside the boundary |
| No-fly zone | Vertical wall | Outside the boundary |

**Figure 19:** Types of path constraints supported by the planning system.

did appear in some position. This is described in more detail in Section 4.3 (Obstacle Avoidance).

For the take-off and landing motion of the UAV, there are 3 types of path shape that we use, depending on the mission requirements:

1. Vertical motion, joined to horizontal motion for the mission
2. Linear blend between waypoint and desired altitude
3. Smooth curved transition between vertical and horizontal motion

Fig. 20 compares these motion profiles. For (1), the end-points are lifted to the desired altitude, the planner finds a point-to-point path between the end-points, then a linear vertical segment is added between the path end-points and true end-points near the ground. For (2), a funnel shape of voxels is added to the Octree map above the linear segment created by method (1), which results in a smooth blended motion. Method (3) is similar to (1), except the end-points are also offset in the direction of the opposite end-point, which results in a linear varying change in altitude.

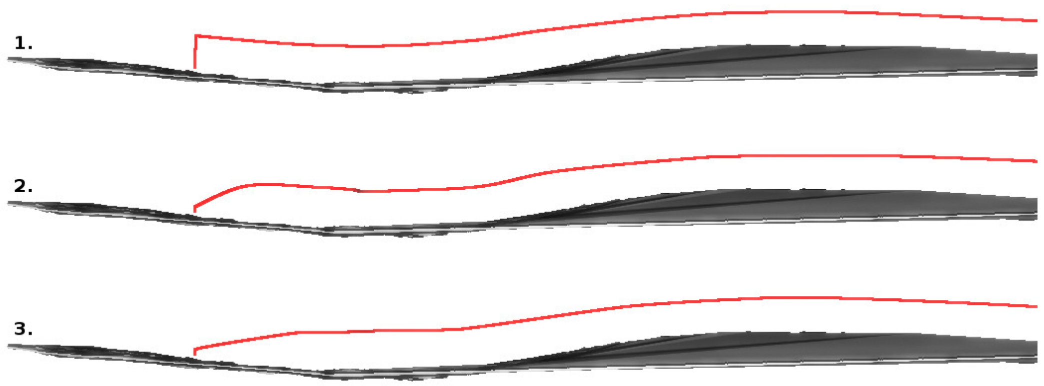


**Figure 20:** Comparison of path profiles used for take-off and landing motions.
1. A linear, vertical take-off motion.
2. A smooth take-off motion resulting from a funnel constraint.
3. A take-off where vertical motion is linearly interpolated to desired altitude.

The altitude constraint is used by the planner when finding the main path. When expanding the RRT, a new (x,y,z) state is sampled, checked for collisions in the 3D Octree map, and its altitude is determined from the 2D height map of the ground surface. If the altitude is within the desired altitude range then the state is valid, otherwise it is rejected.

Waypoints are used to guide the planner to find a path through a specific part of the map. Usually the planner will find the most direct path to the goal, but sometimes the operator may want the UAV to take a longer route towards the goal, or pass through a certain area. For example, with one waypoint, the planner finds a path from the start to the waypoint, then from the waypoint to the goal. Each sub-path consists of linear segments which are concatenated and the entire path is smoothed using an interpolating spline.

Polygon constraints are used to keep the UAV inside, or outside, a specific x-y boundary. The region is defined by a series of (x,y) points that are joined with lines to form an enclosing polygon. A wall of voxels is added to the Octree map, ranging the entire height of the world model. The map boundary constraint is useful when the map is L-shaped or U-shaped, acting like a wall around the world model to stop the planner finding a path across the unknown space. Similarly we can incorporate GIS overlays, like the vector edges of a road, to constrain the UAV to fly along the road. A no-fly zone is implemented in the same way, by adding a wall of voxels to the Octree map in a small region. However, because the UAV starts outside this region then the planner won't find a path through the no-fly zone. One issue with using this solid constraint for GIS overlays is that we would prefer to only bias the path along the road, whilst letting it take an alternate route if required.

Future work is to implement soft constraints or to somehow bias the planner to favor finding a path through certain areas of the map.

## 4.3 Obstacle avoidance

We propose a novel method for obstacle collision avoidance that uses pre-computed sets of alternate routes for the UAV to fly. Normally if a robot encounters an obstacle blocking its path, an online local planner is used to find a collision-free path around the obstacle. In contrast to this, we want to exploit the information available in our high-resolution static map by pre-computing these obstacle avoidance maneuvers.

The first step is to plan the main path from the start to the goal using the point-to-point method described above. Then we imagine "what if" the robot was at some position along this main path and "what if" an obstacle appeared in front of it? The planner "hallucinates" a virtual obstacle in front of the robot and uses this synthetic world model to plan a short, alternate path that goes around the obstacle. This process is iteratively repeated, first along the length of the path, and then recursively along the length of each alternate path. Fig. 21 illustrates the process of generating a roadmap of alternative paths.

The procedure for generating a network of alternative paths is described in Algorithm 1. Up to 5 alternate paths are generated for each virtual obstacle by using a set of 5 spline primitives, oriented at 45°angles relative to each other. These splines are generated between the end-points on the parent path with a 3rd control point located tangent to the mid-point. If a spline representing an alternate path is found to be collision-free in the static map, then it is added to the roadmap of paths. If no alternative paths are found using the spline primitives, a path is found between the required end points using a random-sampling based local planner. This method is effective for finding alternate paths in very cluttered environments, however it can result in paths with sharply varying curvature. The random nature of the planning algorithm also means that the altitude along the length of the alternate path can vary in a non-optimal way.

When the UAV is flying, the perception system checks if the main path is free of obstacles along a receding-horizon. If an obstacle is detected on the current path, the system will check the path roadmap for the next branch point and return a list of the available alternative paths. If this alternative path is free of obstacles then the path switching system tells the controller to follow this branch around the obstacle and back to the main path. If all the alternative paths are also in collision then the UAV must stop. This method of inverse collision-checking reduces the computation required to check the entire world space around the obstacle. The novelty of our approach is that the organization of these pre-computed alternate paths means that the problem of obstacle avoidance is reduced to a simple path switching problem, being “which path do I take next”. This approach makes a trade-off by reducing the required online computation by the CPU, but increases the required memory to store the complex roadmap of paths.

**Algorithm 1:**
For generating a set of alternative paths.

```
Inputs: main_path
        world_model

step_distance = 1.0   // The spacing between alternative paths
robot_to_obs_distance = 25.0   // The distance from which the robot can see an obstacle
n = 1

loop:
   $p_0 = n \times step\_distance$
   $p_{obs} = (n \times step\_distance) + robot\_to\_obs\_distance$
   $p_1 = (n \times step\_distance) + (2 \times robot\_to\_obs\_distance)$

   if ($p_1$ > length of main_path):
      break

   obs = addVirtualObstacle($p_{obs}$, world_model)

   foreach angle in {0, 45, 90, 135, 180}:
      generate path from $p_0$ to $p_1$ using path-primitive
      save path to file
      save path information to index
   if no paths found using primitives:
      find path from $p_0$ to $p_1$ using local planner
      smooth path using spline

   deleteVirtualObstacle(obs)

   n = n + 1
end loop
```

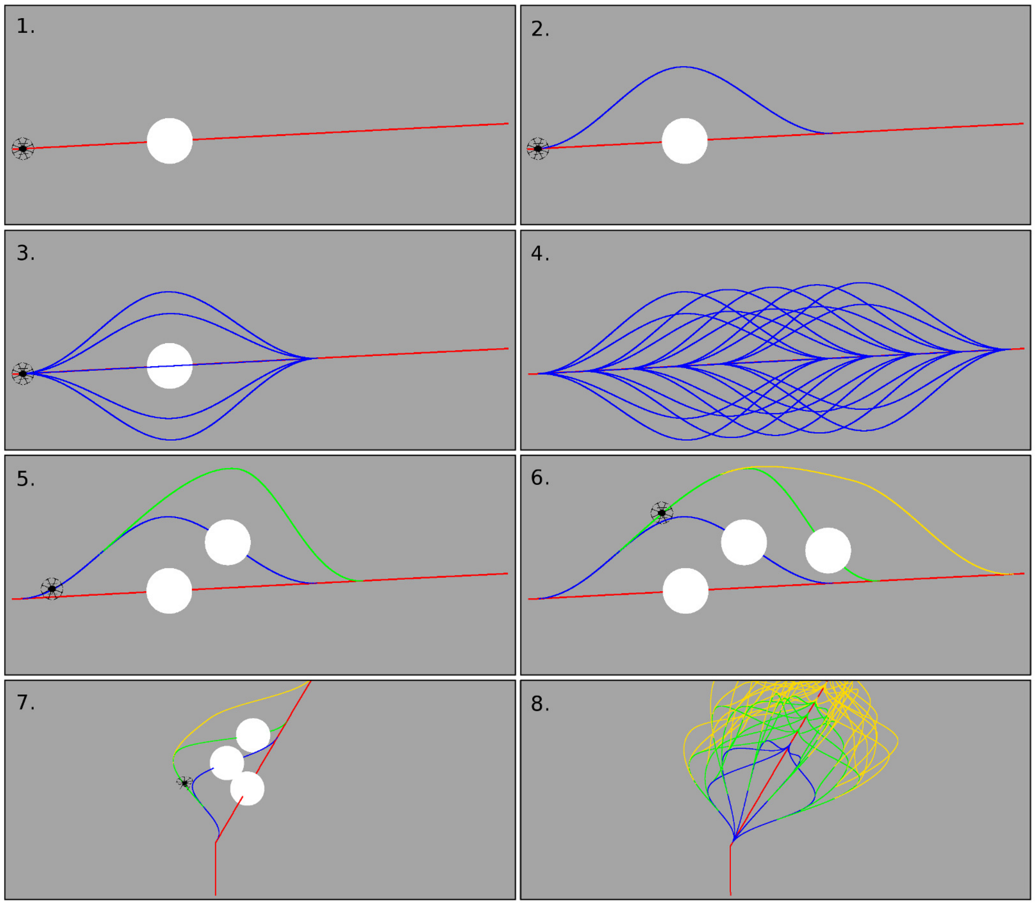


**Figure 21:** The process of generating (offline) a roadmap of alternative paths, that are used by our obstacle-avoidance controller (online).
1. A virtual robot (black-colored object) and virtual obstacle (white-colored sphere) are positioned along the main path.
2. An alternate path is planned around the obstacle, using splines or a RRT as described in our algorithm.
3. Alternate paths are generated at various angles.
4. The resulting set of alternate paths on Level 1 (blue).
5. The process is repeated depth-wise to produce level 2 paths (green), by placing a virtual robot and virtual obstacle on each Level 1 path.
6. The process is repeated again depth-wise to produce level 3 paths (yellow).
7. The same paths from #6, viewed at an angle.
8. The resulting network of alternate paths with 3 levels.

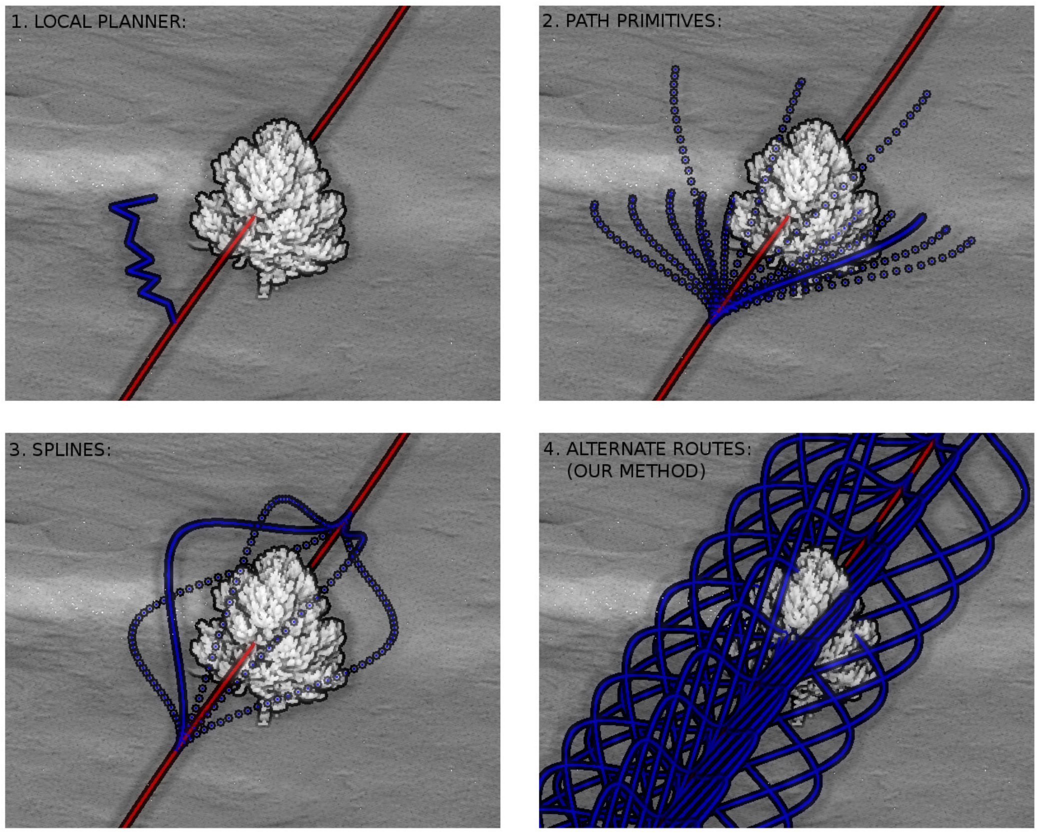


**Figure 22:** This shows 4 methods for obstacle-avoidance
1. A local planner using a RRT.
2. A reactive local planner using path primitives on a receding-horizon.
3. A local planner using splines.
4. Our method, loading pre-planned paths from hard disk.
We compare the planning or computation time required for (1) and (3) with our method (4).

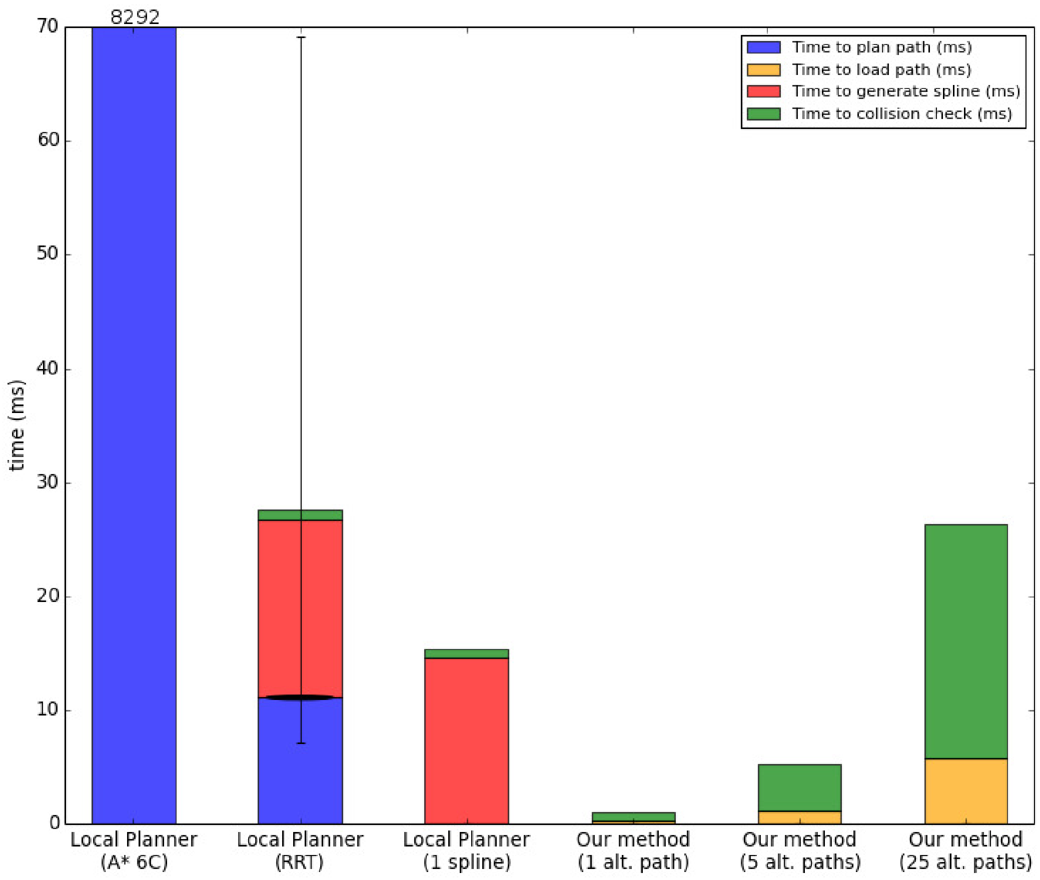


**Figure 23:** This shows the computation time required by our method for collision-avoidance (pre-planned alternative paths) compared to online local planning using an RRT or spline primitives. Our method requires less online computation time. Therefore in cluttered environments the vehicle has more time to switch to an emergency maneuver or to stop completely. However, if our network of alternative paths is too dense, we end up doing more collision-checking work than an online local planner.

## 4.4 Preparing paths for the UAV

The final step is to add velocity information to the roadmap of paths to make them trajectories. Currently we use constant velocities for large segments of the path. For example, if a section of path has low curvature then it is assigned with high velocity. If the path is maneuvering around obstacles then a lower velocity is assigned. We use a simulation of the drone's control system and trial-and-error to determine the maximum safe velocity for each section of the path. The changes in velocity are blended using linear interpolation.

We plan to improve this method in future work. A more robust approach would be to use a dynamics filter to automatically determine the optimal velocities for the path. This would require system identification to build an accurate dynamic model of the drone and to characterize its handling capability in various situations. Using this model, a receding-horizon controller is used offline to filter the path and the optimal control signals are recorded.

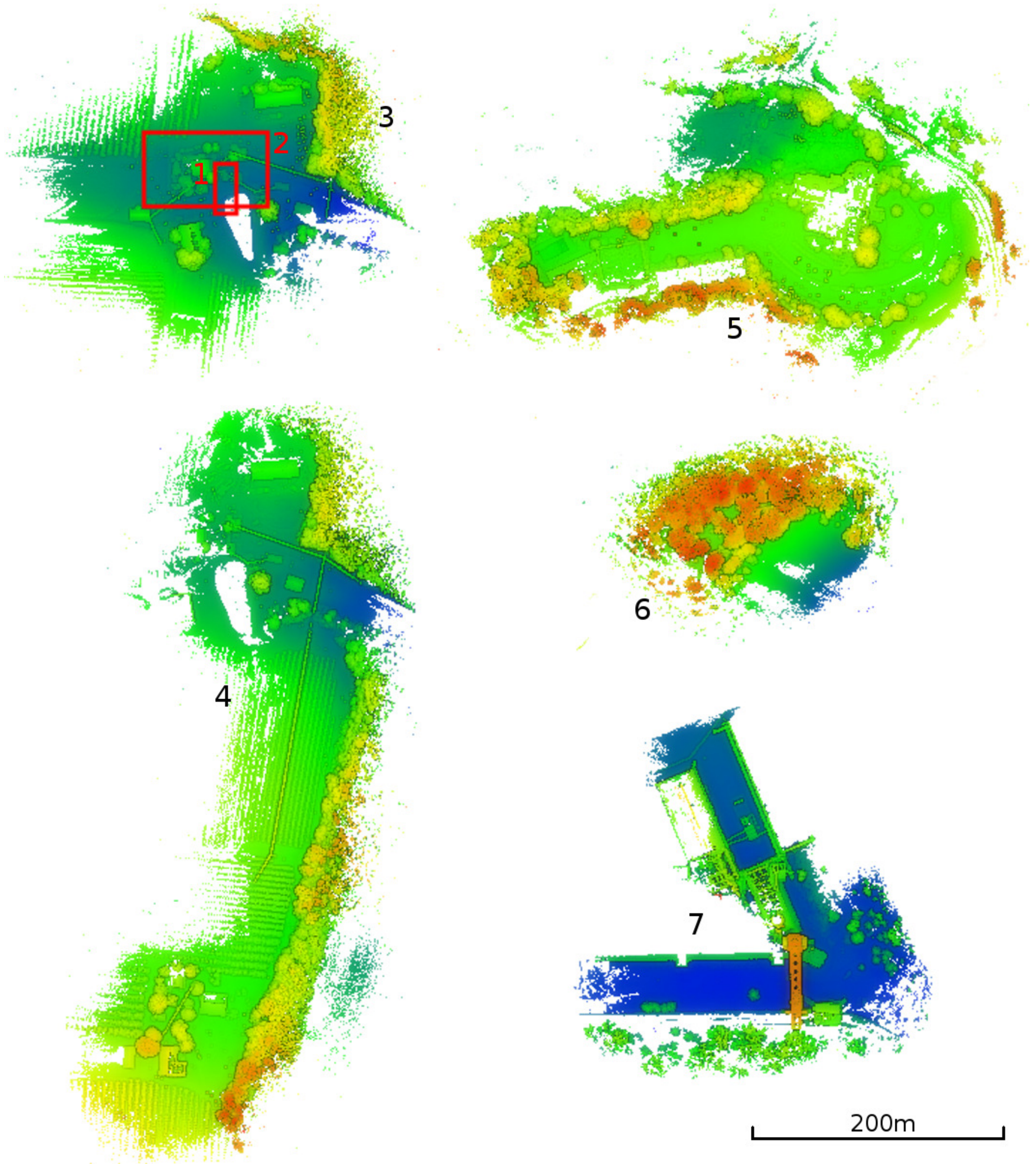


**Figure 24:** We use 7 point clouds from various outdoor environments to test our path planner. Our approach can plan paths in high-resolution using all of these maps, of which the largest is more than 1km long. Previous methods can only use small maps like #1 and #2.

# 5 Experiments

## 5.1 Path Planning

We test our mission planning system using 7 high-resolutions maps of various large outdoor environments, as shown in Fig. 24.
Map 1: The smallest map, this is a slice of map #3.
Map 2: Another slice of map #3 and the largest map that can be stored in a Voxel Occupancy Grid.
Map 3: The Orchard.
Map 4: The Orchard.
Map 5: The Country Club.
Map 6: Trees in Schenley Park.
Map 7: Carrie Furnaces, a derelict former steel plant in Pittsburgh.

Fig. 25 shows the result of planning with and without altitude constraints for Map No. 3.

Fig. 27 shows how the planner prefers to find the shortest path, which for certain maps may not be the safest flight path, or even a reasonable path. By using altitude constraints and defining a boundary around the region which was mapped, the resulting path moves through the map.

Fig. 28 shows the result of using no-fly zones, to steer the path around flying overhead of buildings.

We experimented with using information from GIS databases for path planning, such as the location of roads. The road network for the United States is represented as a geo-referenced vector graphics layer, where each road consists of a series of parallel lines. We use this outline to create a hard constraint that forces the planner to generate a path that follows the road, as shown in Fig. 29.

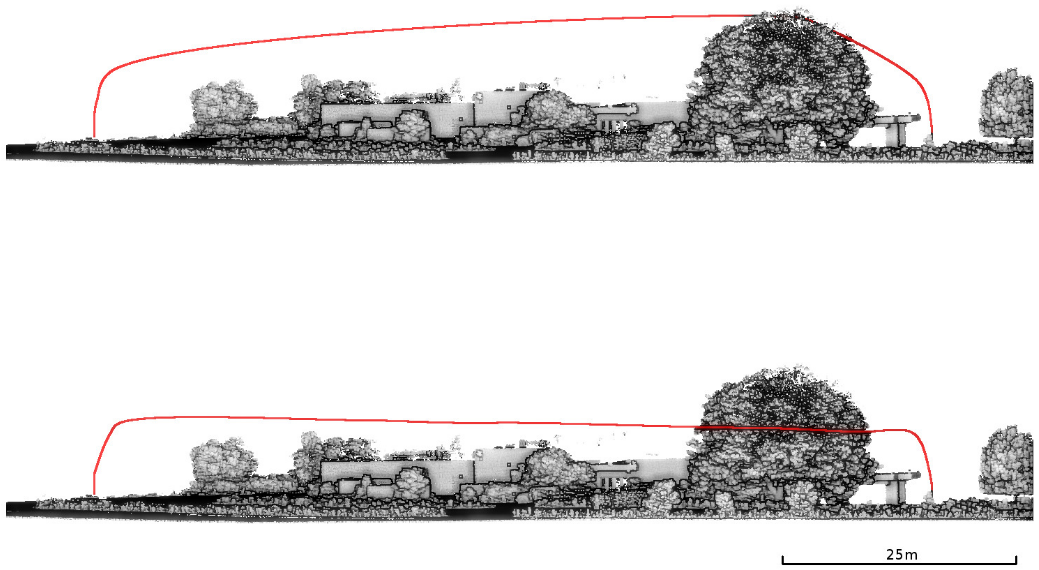


**Figure 25:** This compares path planning with and without altitude constraints.
(top) A standard point-to-point path, with smooth motion for take-off and landing resulting from the funnel constraints.
(bottom) A point-to-point path with funnel constraints and altitude constraint of 9.0m +0.0/-0.5

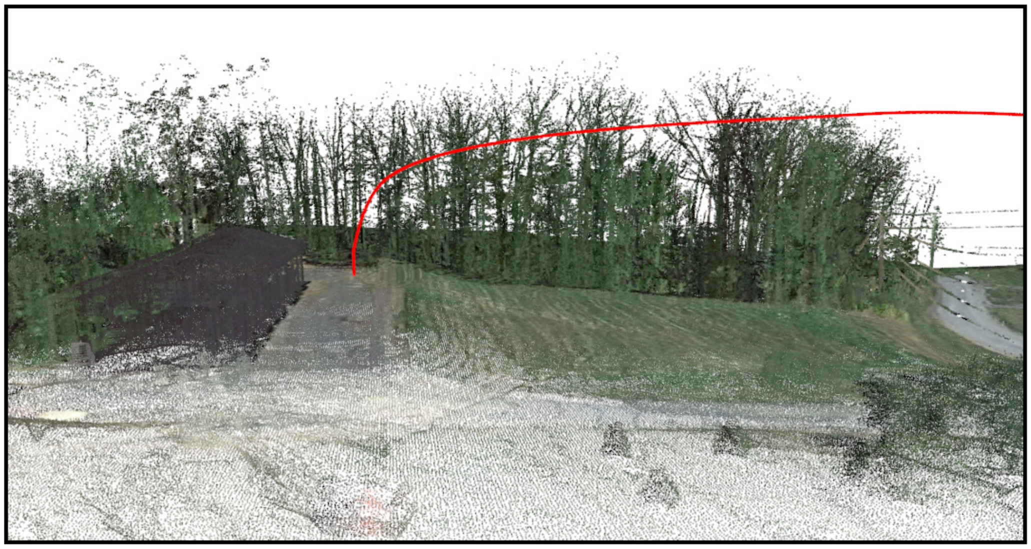
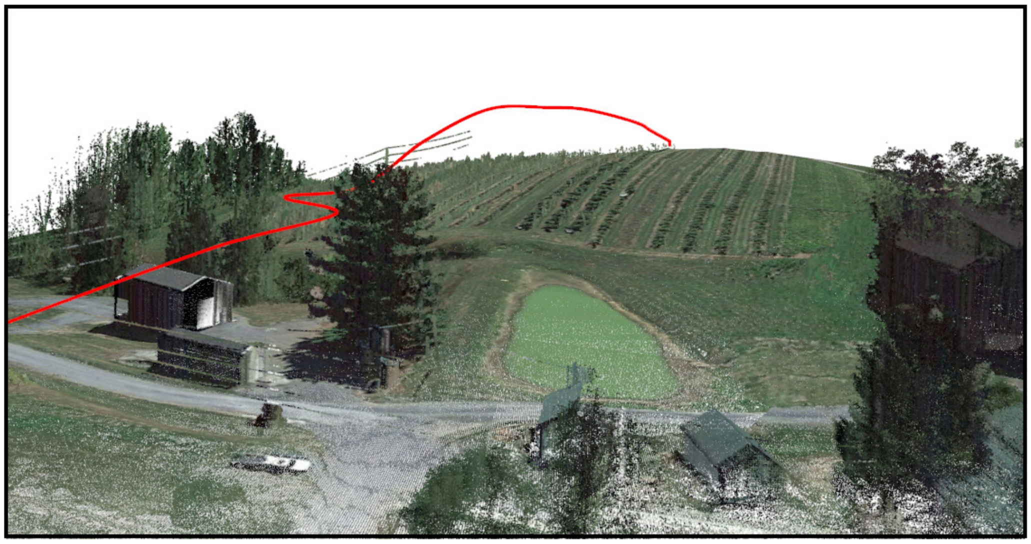



**Figure 26:** This shows a path planned in a large map which is longer than 500m, whilst avoiding obstacles and adhering to altitude constraints. The point cloud is colorized using aerial imagery, for visualization purposes.
(top) The take-off phase of the path is the result of using a funnel constraint. A short vertical section is generated using linear interpolation and the smooth, curved transition comes from using a large-radius funnel-shaped constraint.
(bottom) The path goes around obstacles that exist in the static map, like the big tree. An altitude constraint of 15–17m makes the shape of the path match the ground terrain below it.

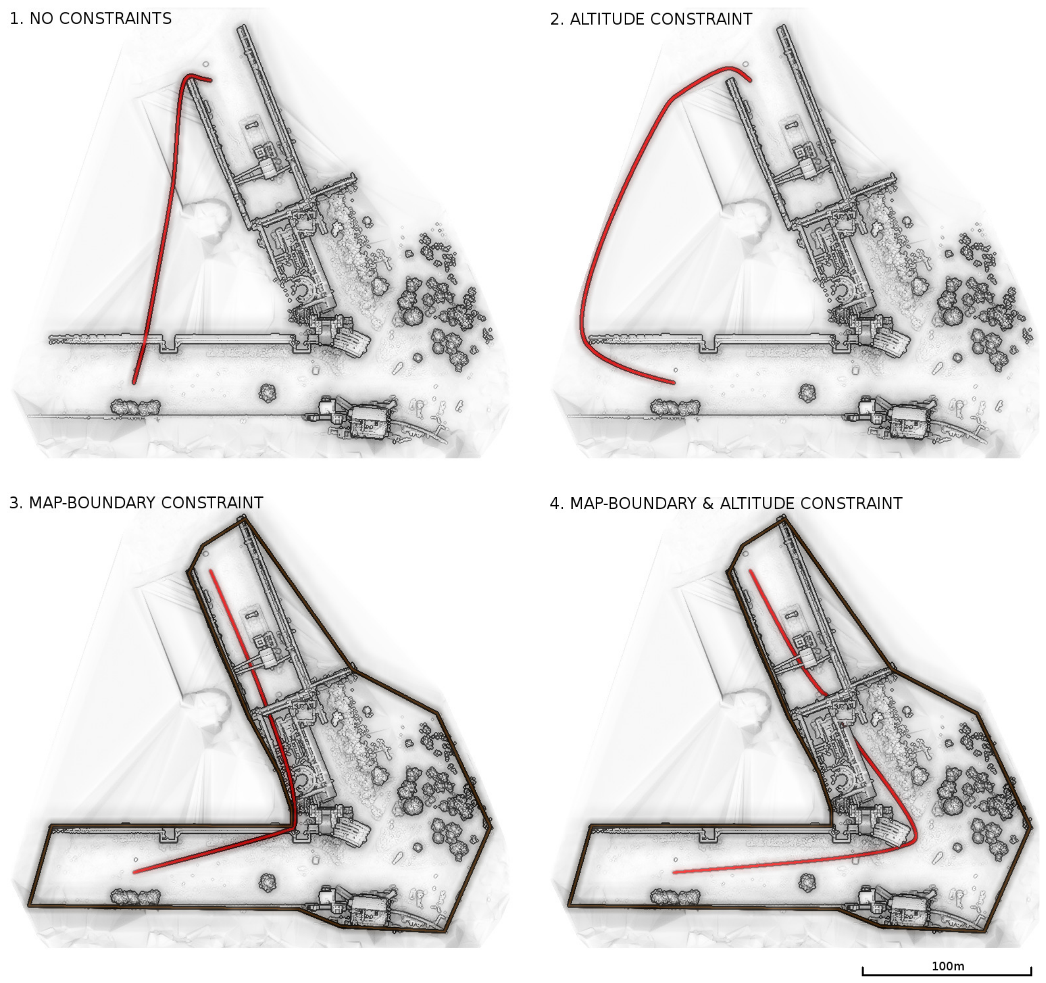

1. NO CONSTRAINTS
2. ALTITUDE CONSTRAINT
3. MAP-BOUNDARY CONSTRAINT
4. MAP-BOUNDARY & ALTITUDE CONSTRAINT
100m

**Figure 27:** A comparison of planning with/without altitude and vector boundary constraints in an L-shaped map, which can cause issues when finding the shortest path.
(top left) A point-to-point plan with no constraints creates a path that goes outside the area captured during the mapping process.
(top right) With altitude constraints, the shortest path still goes outside the map.
(bottom left) A boundary constraint is added along the most extremal points of the map. The shortest path flies close to the inside corner of the map boundary and down through the roof of a building which was not captured during the mapping process.
(bottom right) Combining the map boundary constraint and an altitude constraint of 1–3m results in a path that flies through the environment.

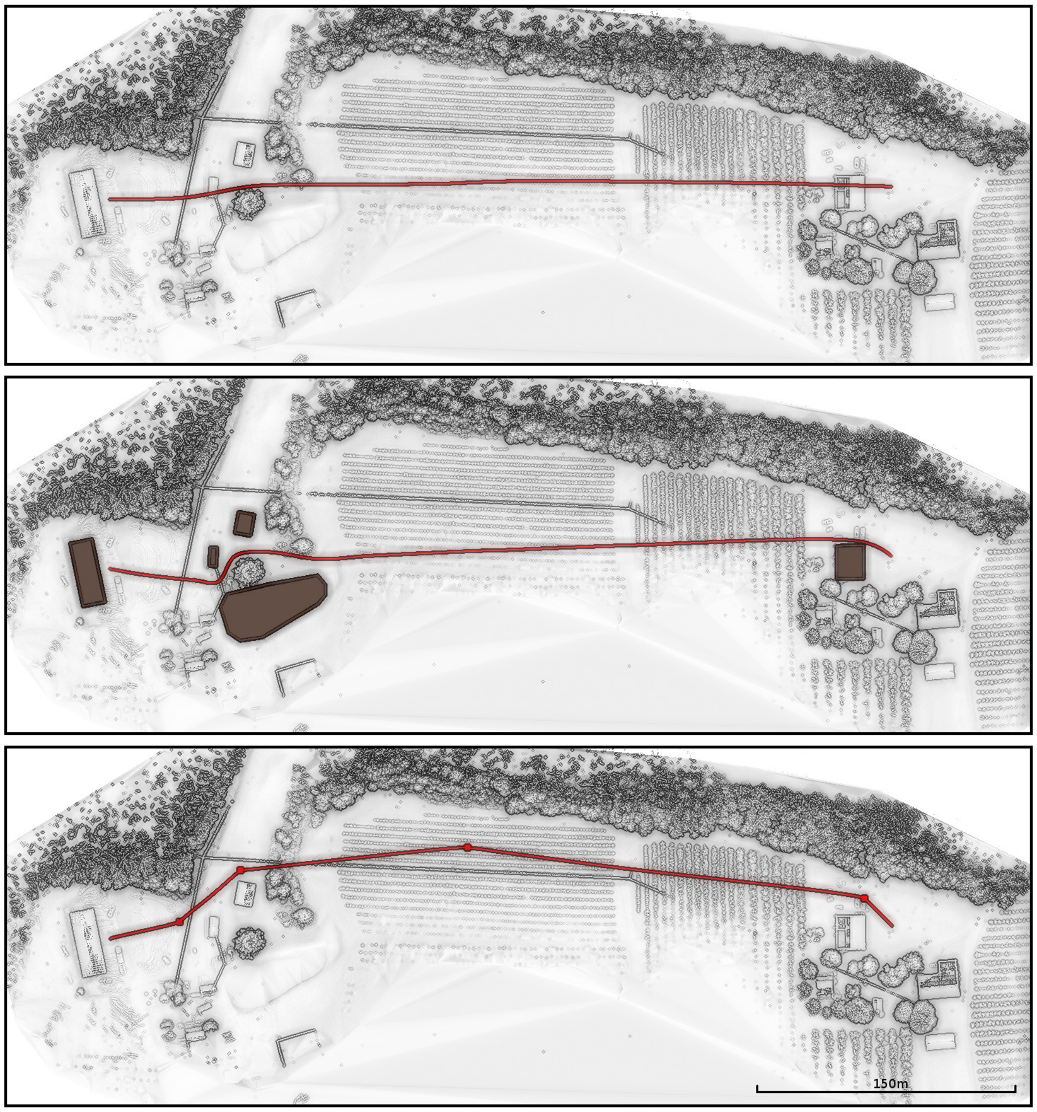


**Figure 28:** This compares the result of path planning using no-fly zones and waypoints.
(top) A point-to-point path using an altitude constraint of 15–17m follows approximately the shortest path, whilst flying around the large tree.
(middle) "No fly" zones are manually defined using vector boundary constraints, shown in brown, which causes the path to fly around a small building near the tree and around the building near the landing zone.
(bottom) Manually defined waypoints are used to force the planner to produce a path along some desired route.

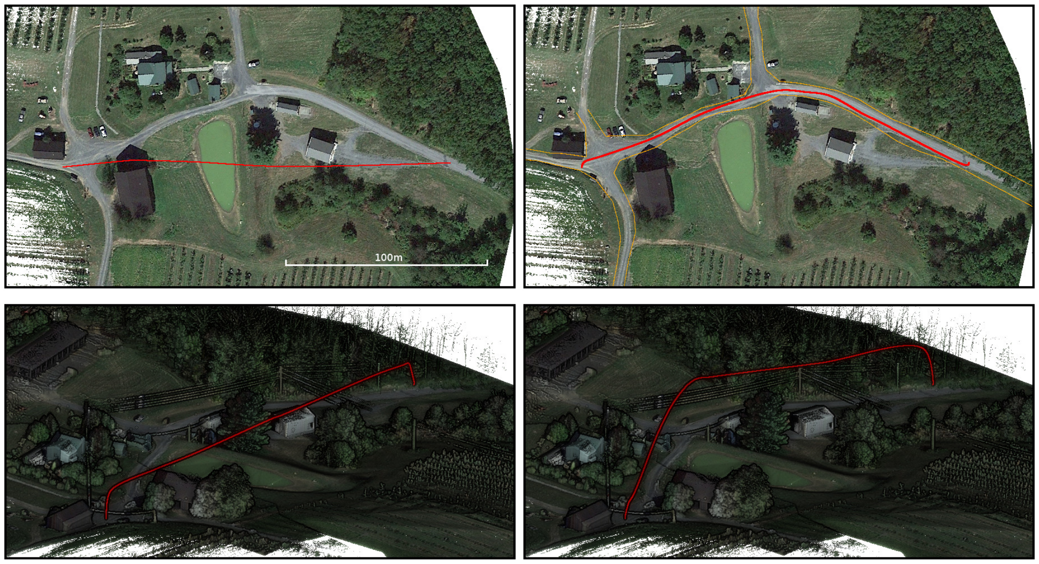


**Figure 29:** This shows how a path can be constrained to follow a road, using vector outlines which are available from GIS databases.
(left images) A point-to-point path using a 12–15m altitude constraint follows approximately the shortest path.
(right images) The road outline vectors are used to generate boundary constraints, shown in orange. The path start and goal positions are located within the road boundary and the resulting path follows the road.

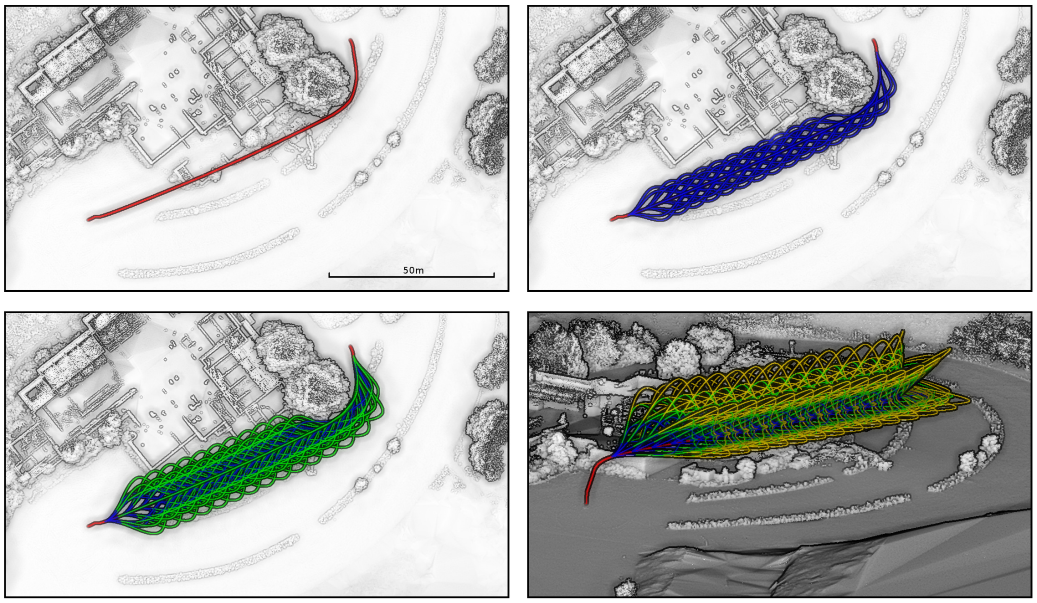


**Figure 30:** A network of alternative paths is pre-planned (offline) and used for obstacle-avoidance (online) by collision-checking one or more paths that would allow the UAV to fly around the obstacle.
(top left) The main mission path (red) has an altitude constraint of 9.0m +0.0/-0.5.
(top right) The level 1 set of alternative paths (blue). Note how these obstacle-avoidance maneuvers fly around the large tree in the static map.
(bottom left) The level 2 set of alternative paths (green).
(bottom right) The smooth tale-off motion (red) and level 1 (blue), level 2 (green) and level 3 (yellow) paths.

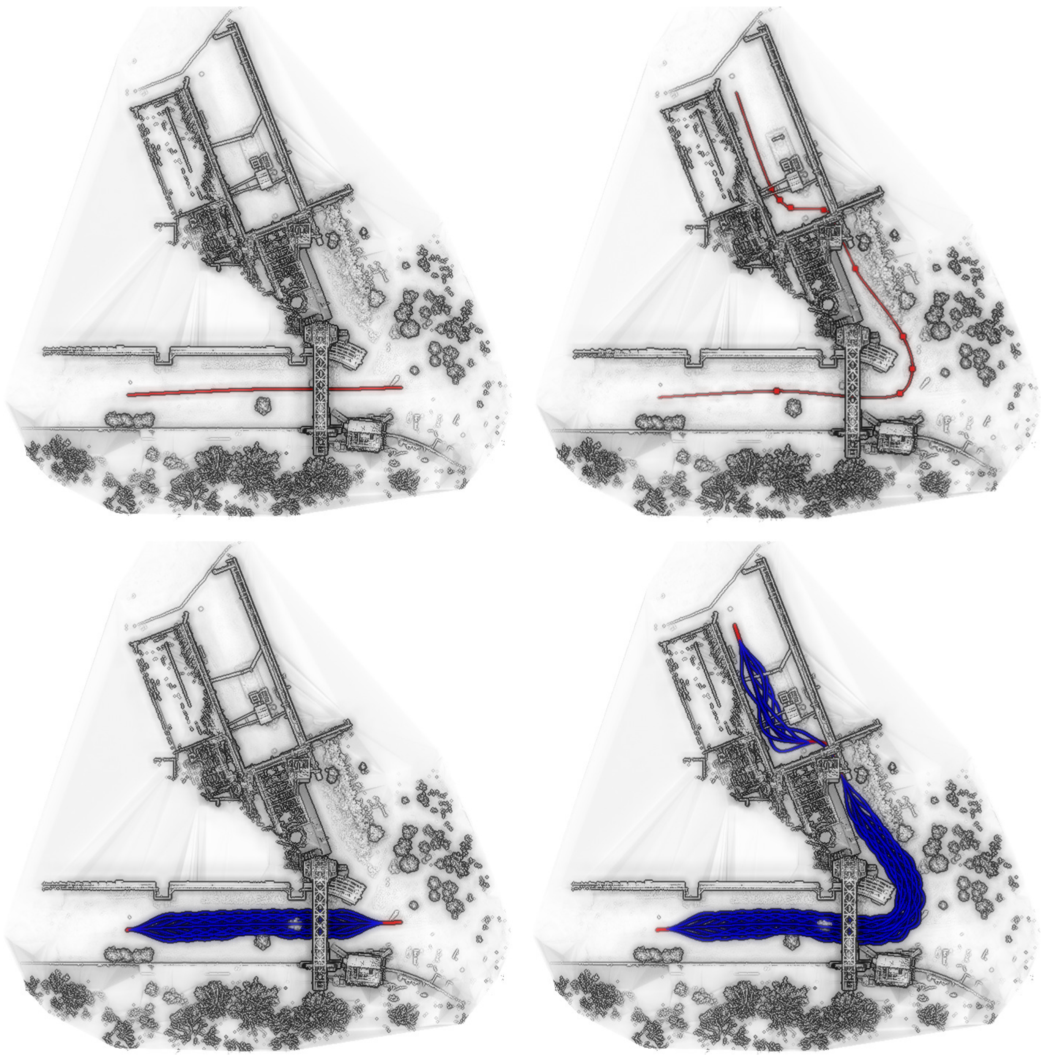


**Figure 31:** Another example of a network of pre-planned alternative paths used for obstacle avoidance.
(top left) A short point-to-point mission path (red).
(top left) A longer mission path (red) is planned using manually defined [x,y,altitude] waypoints to force the path to be further away from obstacles, for testing purposes.
(bottom left) The level 1 alternative paths (blue) for the short main path.
(bottom right) The level 1 alternative paths (blue) for the long main path. The main path flies through a narrow overhanging structure and no collision-free alternative paths exist. Therefore, if an obstacle appeared (online) in this "choke point", the UAV would have to come to a stop.

# 6 Conclusion

We have developed a fast and efficient method for planning missions for multi-rotor UAV. Using a memory-efficient data structure we can create high-resolution models of the world for planning through doorways and overhanging structures in cluttered environments.

This work makes the following contributions:

- An analysis of 3D map data structures and algorithms for robot-to-map collision checking and path planning.
- A fast planner for UAV missions that works directly on large, high-resolution point clouds.
- A path planning system that can utilize multiple constraints: smooth take-off and landing, height or altitude, map-based waypoints, no-fly zones and GIS vector overlays.
- A novel obstacle-avoidance method that reduces online computation to enable faster (and therefore safer) decision making, and improves the predictability of motions.

In future work we hope to address how we can plan paths with would be considered “safer” or have some type of safety bounds. One approach might be to constrain the UAV to a certain altitude and try to stay as far away as obstacles as possible. Also, if there are multiple topologically-distinct paths at a specific altitude, we could choose the path with the maximum distance to obstacles.

# References

[1] "Unmanned Aerial Systems: FAA Continues Progress toward Integration into the National Airspace," US GAO (Government Accountability Office), Tech. Rep. GAO-15-610, August 2015, http://www.gao.gov/products/GAO-15-610.

[2] K. S. Krishnakumar, P. H. Kopardekar, C. A. Ippolito, J. Melton, V. Stepanyan, S. Sankararaman, and B. Nikaido, "Safe Autonomous Flight Environment (SAFE50) for the Notional Last 50 ft of Operation of 55 lb Class of UAS," in *AIAA SciTech Forum, AIAA Information Systems-AIAA Infotech @ Aerospace*. American Institute of Aeronautics and Astronautics, Jan 2017, http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20170000382.pdf.

[3] S. Choudhury, S. Arora, and S. Scherer, "The Planner Ensemble and Trajectory Executive: A High Performance Motion Planning System with Guaranteed Safety," in *AHS 70th Annual Forum, Montréal, Québec, Canada*, May 2014.

[4] V. Dugar, S. Choudhury, and S. Scherer, "Smooth Trajectory Optimization in Wind: First Results on a Full-Scale Helicopter," in *AHS Forum, Vol 73, 2017*, March 2017.

[5] M. Nieuwenhuisen, D. Droeschel, M. Beul, and S. Behnke, "Autonomous Navigation for Micro Aerial Vehicles in Complex GNSS-denied environments," *Journal of Intelligent & Robotic Systems*, vol. 84, no. 1, pp. 199–216, 2016.

[6] S. Singh, H. Cover, A. Stambler, B. Grocholsky, S. Spiker, J. Mishler, B. Hamner, K. Strabala, G. Sherwin, M. Bergerman, M. Kaess, and G. Hemann, "Perception for Safe Autonomous Helicopter Flight and Landing," in *AHS Forum, Vol 72, 2016*, May 2016.

[7] J. Zhang, R. Chadha, and S. Singh, "Realtime State Estimation & Mapping for Autonomous Flight in Complex, GPS-denied, Environments," in *Workshop on Vision-based High Speed Autonomous Navigation of UAVs at IEEE/RSJ International Conference on Intelligent Robots and System (IROS)*, Oct 2016, Paper was prepared for this workshop but not presented. Available online: http://www.seas.upenn.edu/~loiannog/workshopIROS2016uav/.

[8] J. H. Reif, "Complexity of the Mover's Problem and Generalizations," in *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, Oct 1979, pp. 421–427.

[9] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, Dec. 1959.

[10] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems, Science, and Cybernetics*, vol. SSC-4, no. 2, pp. 100–107, 1968.

[11] M. Likhachev, G. Gordon, and S. Thrun, "ARA*: Anytime A* with Provable Bounds on Sub-Optimality," in *Proceedings of the Neural Information Processing Systems Conference (NIPS): Advances in Neural Information Processing Systems 16*, 2003.

[12] H. Bast, S. Funke, and D. Matijevic, "TRANSIT – Ultrafast Shortest-Path Queries with Linear-Time Preprocessing," in *In 9th DIMACS Implementation Challenge Workshop: Shortest Paths*, 2006.

[13] D. Ferguson and A. T. Stentz , "Field D*: An Interpolation-based Path Planner and Replanner," in *Proceedings of the International Symposium on Robotics Research (ISRR)*, October 2005.

[14] J. Carsten, D. Ferguson , and A. T. Stentz , “3D Field D: Improved Path Planning and Replanning in Three Dimensions,” in *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2006, pp. 3381–3386.

[15] A. Nash, K. Daniel, S. Koenig, and A. Feiner, “Theta*: Any-angle Path Planning on Grids,” in *Proceedings of the 22nd National Conference on Artificial Intelligence - Volume 2*, ser. AAAI’07. AAAI Press, 2007, pp. 1177–1183.

[16] H. Cover, S. Choudhury, S. Scherer, and S. Singh, “Sparse Tangential Network (SPARTAN): Motion Planning for Micro Aerial Vehicles,” in *2013 IEEE International Conference on Robotics and Automation*, May 2013, pp. 2820–2825.

[17] I. Borovikov. Navigation Graph Generation. [Online]. Available: http://www.gamedev.net/reference/index.html/_/technical/artificial-intelligence/navigation-graph-generation-r2805

[18] T. Lozano-Pérez and M. A. Wesley, “An Algorithm for Planning Collision-free Paths Among Polyhedral Obstacles,” *Commun. ACM*, vol. 22, no. 10, pp. 560–570, Oct. 1979.

[19] F. Schøler, “3D Path Planning for Autonomous Aerial Vehicles in Constrained Spaces,” Ph.D. dissertation, Aalborg University, Denmark, 2012, http://vbn.aau.dk/files/68652677/thesis.pdf.

[20] A. Kneidl, A. Borrmann, and D. Hartmann, “Generating Sparse Navigation Graphs for Microscopic Pedestria Simulation Models,” *Advanced Engineering Informatics*, vol. 26, no. 4, pp. 669–680, October 2012.

[21] P. Beeson, N. K. Jong, and B. Kuipers, “Towards Autonomous Topological Place Detection Using the Extended voronoi graph,” in *Proceedings of*

*the 2005 IEEE International Conference on Robotics and Automation, ICRA 2005, April 18-22, 2005, Barcelona, Spain*, 2005, pp. 4373–4379.

[22] B. Lau, C. Sprunk, and W. Burgard, “Efficient Grid-Based Spatial Representations for Robot Navigation in Dynamic Environments,” *Robotics and Autonomous Systems*, vol. 61, no. 10, pp. 1116 – 1130, 2013, selected Papers from the 5th European Conference on Mobile Robots (ECMR 2011).

[23] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars, “Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces,” *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, Aug 1996.

[24] S. M. LaValle and J. J. Kuffner, “Randomized Kinodynamic Planning,” in *Proceedings 1999 IEEE International Conference on Robotics and Automation*, vol. 1, 1999, pp. 473–479 vol.1.

[25] S. Karaman and E. Frazzoli, “Sampling-based Algorithms for Optimal Motion Planning,” *International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, Jun. 2011.

[26] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, “Informed RRT*: Optimal Sampling-based Path Planning Focused via Direct Sampling of an Admissible Ellipsoidal Heuristic,” in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sept 2014, pp. 2997–3004.

[27] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot., “Batch Informed Trees (BIT*): Sampling-based Optimal Planning via the Heuristically Guided Search of Implicit Random Geometric Graphs,” in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 3067–3074.

[28] S. Choudhury, J. D. Gammell, T. D. Barfoot, S. S. Srinivasa, and S. Scherer, "Regionally Accelerated Batch Informed Trees (RABIT*): A Framework to Integrate Local Information into Optimal Path Planning," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, Ed., May 2016.

[29] S. Scherer, S. Singh, L. J. Chamberlain, and S. Saripalli, "Flying Fast and Low Among Obstacles," in *Proceedings International Conference on Robotics and Automation*, Pittsburgh, PA, April 2007.

[30] I. Dryanovski, R. G. Valenti, and J. Xiao, "An Open-Source Navigation System for Micro Aerial Vehicles," *Autonomous Robots*, vol. 34, no. 3, pp. 177–188, 2013.

[31] B. MacAllister, J. Butzke, A. Kushleyev, H. Pandey, and M. Likhachev, "Path Planning for Non-circular Micro Aerial Vehicles in Constrained Environments," in *2013 IEEE International Conference on Robotics and Automation*, May 2013, pp. 3933–3940.

[32] Z. Fang, S. Yang, S. Jain, G. Dubey, S. Roth, S. M. Maeta, S. T. Nuske, Y. Wu, and S. Scherer, "Robust Autonomous Flight in Constrained and Visually Degraded Shipboard Environments," *Journal of Field Robotics*, vol. 34, no. 1, pp. 25–52, January 2017.

[33] D. Droeschel, M. Nieuwenhuisen, M. Beul, D. Holz, J. Stückler, and S. Behnke, "Multilayered Mapping and Navigation for Autonomous Micro Aerial Vehicles," *Journal of Field Robotics*, vol. 33, no. 4, pp. 451–475, June 2016.

[34] H. Hu, D. Munoz, J. A. Bagnell, and M. Hebert, "Efficient 3-D Scene Analysis from Streaming Data," in *2013 IEEE International Conference on Robotics and Automation*, May 2013, pp. 2297–2304.

[35] W. Zhang, J. Qi, P. Wan, H. Wang, D. Xie, X. Wang, and G. Yan, "An Easy-to-Use Airborne LiDAR Data Filtering Method Based on Cloth Simulation," *Remote Sensing*, vol. 8, no. 6, p. 501, 2016.

[36] P. E. Axelsson, "DEM Generation from Laser Scanner Data using Adaptive TIN Models," *International Archives of the Photogrammetry and Remote Sensing*, vol. 33, pp. 110–117, 2000.

[37] J. S. Evans and A. T. Hudak, "A Multiscale Curvature Algorithm for Classifying Discrete Return LiDAR in Forested Environments," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 45, no. 4, pp. 1029–1038, April 2007.

[38] N. Brodu and D. Lague, "3D Terrestrial LiDAR Data Classification of Complex Natural Scenes using a Multi-scale Dimensionality Criterion: Applications in Geomorphology," *{ISPRS} Journal of Photogrammetry and Remote Sensing*, vol. 68, pp. 121–134, 2012.

[39] Kaarta. Stencil 3D mapping & localization device. [Online]. Available: http://www.kaarta.com/stencil

[40] J. Zhang and S. Singh, "LOAM: Lidar Odometry and Mapping in Real-time," in *Robotics: Science and Systems Conference*, Pittsburgh, PA, July 2014.

[41] J. Zhang and S. Singh., "Visual-Lidar Odometry and Mapping: Low-drift, Robust, and Fast," in *IEEE International Conference on Robotics and Automation (ICRA)*, Pittsburgh, PA, May 2015.

[42] Results from the microsoft indoor localization competition, held at the 15th acm/ieee international conference on information processing in sensor networks (ipsn). [Online]. Available: http://www.microsoft.com/en-us/research/event/microsoft-indoor-localization-competition-ipsn-2016/

[43] G. Hemann, S. Singh, and M. Kaess, "Long-range GPS-denied Aerial Inertial Navigation with LiDAR Localization," in *IEEE/RSJ International Conference on Intelligent Robots and System (IROS)*, October 2016.

[44] B. Discoe. Virtual Terrain Project. [Online]. Available: http://www.vterrain.org

[45] Pennsylvania Department of Conservation and Natural Resources, Bureau of Topographic and Geologic Survey. PAMAP program. [Online]. Available: http://www.dcnr.state.pa.us/topogeo/pamap/lidar/index.htm

[46] USGS. EarthExplorer. [Online]. Available: http://www.earthexplorer.usgs.gov

[47] Microsoft. Bing Maps. [Online]. Available: http://www.bing.com/maps/

[48] Sanborn. Oblique Analyst. [Online]. Available: http://www.sanborn.com/oblique-analyst/

[49] E. Catmull and R. Rom, "A Class of Local Interpolating Splines," in *Computer Aided Geometric Design 74*. Academic Press, 1974, pp. 317 – 326.

[50] P. J. Barry and R. N. Goldman, "A Recursive Evaluation Algorithm for a Class of Catmull-Rom Splines," in *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '88. New York, NY, USA: ACM, 1988, pp. 199–204.

[51] C. Yuksel, S. Schaefer, and J. Keyser, "On the Parameterization of Catmull-Rom Curves," in *2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling*. New York, NY, USA: ACM, 2009, pp. 47–53.