UAV Path Planning using Aerially Obtained Point Clouds

Auburn REU on SMART UAVs 2022

Pugh, Alec
University of North Carolina at Chapel Hill
alecjp@unc.edu

Bower, Luke
The University of Alabama in Huntsville
lcb0035@uah.edu

Chapman, Richard
Auburn University
chapmro@auburn.edu

Biaz, Saad
Auburn University
biazsaa@auburn.edu

July 6, 2022

Abstract

With the growing use of unmanned aerial vehicles (**UAVs**) for commercial and military operations, path efficiency remains an utmost concern for battery and time preservation. This paper presents a method for three dimensional (**3D**) path planning using point clouds obtained from the USGS 3DEP (**United States Geological Survey 3D Elevation Program**) dataset via OpenTopography. The path itself is obtained using the *A* algorithm*, with additional modifications implemented to account for path smoothing, UAV size, and energy consumption. We also introduce a collision avoidance method using the precomputed data to account for unforeseen obstacles not rendered within the point cloud. The method presented is designed specifically for point clouds obtained via LiDAR (**Light Detection and Ranging**) scans from aircraft, where cavities may be present underneath the surface layer. Simulations show the validity of this method.

1 Introduction

The importance of research towards efficient and reliable path finding algorithms has steadily increased with the growing popularity and use of autonomous UAVs. There are many benefits around the development and use of UAV path finding, and each process developed mainly strives to satisfy each of the following requirements: time preservation, path optimality, and danger avoidance. The consequences of these requirements are the most apparent in military and commercial operations: provide a reduction in the cost spent towards training UAV pilots, a reduced amount of human manpower needed to pilot the UAVs, and a reduction of accidents due to human error or incorrect decision making. Technological developments have allowed humans to power cars with batteries for hundreds of miles at a time. While such batteries are viable for larger vehicles, versatile UAVs are compact and cannot carry large batteries or payloads due to their size and thrust output. As such, a typical commercially viable UAV has an average flight time of approximately 15-25 minutes [11]. With flight times this short, being able to accomplish more during a mission without

recharging is critical to success. This correlates directly with the time spent travelling between points of interest and energy consumption of maneuvers, which path finding methods help reduce.

The USGS 3D Elevation Program is a developing program that aims to provide topographical data for the entire United States. Beginning production in 2016, 84% of the nation has topographical data that has been obtained from aircraft LiDAR scans. USGS also plans to have the entire US mapped by the end of 2023. These LiDAR scans are displayed in the form of a point cloud, which is a set of 3D points (containing X, Y, and Z coordinates, among other types of data) in space relating to their physical GPS location. The data available through USGS 3DEP can be seen as undoubtedly instrumental in providing accurate path finding for UAV operations, especially for locations that may be inaccessible to humans or ordinary UAV flight operation due to dangerous conditions, Federal Aviation Administration (**FAA**) regulations, or various other impractical reasons. Furthermore, if this information is confined to a specific operating environment and is obtained prior to a mission launch, optimal paths can be found with less computational intensity.

To this end, the main purpose of this paper hopes to leverage the vast amount of LiDAR point cloud data obtained through USGS 3DEP and show a concise method to provide path finding using this data. In addition, a secondary objective is to present a method of collision avoidance that also leverages the precomputed data, creating an all-inclusive program. This paper is outlined as follows: Section 2 displays the purpose of the research presented. Section 3 presents a literature review for the previous research done in this field. We outline our proposed method for UAV path planning using aerially obtained point clouds in Section 4. Simulated testing is performed and documented in Section 5. Our results follow in Section 6 with a brief conclusion and outline for future work in Section 7.

2 Problem Statement and Motivation

The advantages that UAVs withhold over manned aerial vehicles are apparent; however, there are still many limitations with the currently available technology. One main disadvantage comes from limited battery life that UAVs posses, so research into battery improvements for UAVs has been vast. While various methods for increasing battery life has been tested, such as unlimited endurance (laser-beam in-flight recharging, tethering, and swapping), along with alternative fuel sources (such as hydrogen, methanol, and hydrocarbons) [4], few of these methods or alternatives have become common practice for use in commercial and consumer UAVs as of today. This is often due to the limited weight tolerance that UAVs allow, which naturally prevents the trivial solution of equipping a UAV with additional batteries. Therefore, off-board optimization techniques can be viewed as a solution to help optimize the use of the limited battery life available while also preventing weight additions on UAVs with already limited payload tolerances.

As stated previously, obtaining point cloud scans of a given area can be difficult or impossible due to lack of proper equipment, state/governmental regulations, or safety concerns. By taking advantage of the datasets already available, this eliminates the need for expensive equipment and the inaccessibility of obtaining topographical information otherwise. However, the accuracy of path planning using point cloud data is proportional to the quality of a given point cloud scan. With USGS 3DEP datasets, LiDAR scan quality is standardized through QL (quality level) specifications which are detailed enough to be trusted for use for path finding operations. In addition, USGS has plans to continue to increase the overall scan quality overtime through dataset updates [27]. This shows that the accuracy of the methodology presented will increase linearly with the progression of future processes and technologies used to obtain these scans. Therefore, a streamlined 3D path planning approach using this data will be useful for not only

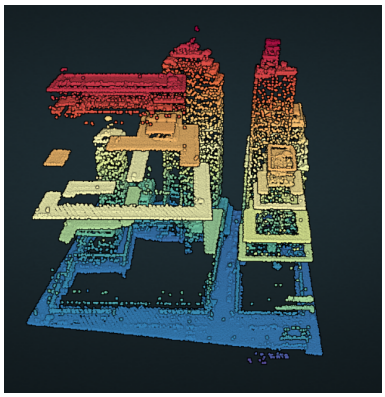

Figure 1: Example of point cloud with cavities present. (Potree) [3]

present UAV operations, but also the future.

The primary objective of our research is to create a reliable and efficient 3D path that works seamlessly with 3DEP scans. Due to the nature of these scans being obtained through aircraft flying above an area, only the surface points of an environment may be scanned accurately. Thus, features that exist directly perpendicular to a surface (such as building faces underneath a roof, or steep cliffs) may be represented as empty space in a 3DEP scan, as shown in Figure 1. The process presented accounts for this problem. Our research simulates a multi-rotor UAV for its quick movement and directional freedom. Due to the path being generated off board from the UAV, on-board equipment can be limited to a 2D LiDAR sensor, GPS, and transmitter as shown in our simulations. A secondary objective of this research is to have systems in place to provide collision avoidance procedures. By using the precomputed LiDAR data for collision avoidance, we once again do not require additional onboard equipment, which provides greater flexibility for deployment applications. The final objective is to incorporate API integration to create a seamless path generation process from input to output.

3 Literary Analysis

There has been many significant contributions to the field of UAV research over the years. To provide a top-down approach in topic discussion, this section will first provide a classification for the different types of mapping systems within UAV research. Then, we provide definitions and related work with UAV and LiDAR, path planning, and collision avoidance techniques with respect to the type of mapping that reflects this paper and process.

3.1 Mapping Classifications

In general, there are three main types of mappings used in autonomous UAV navigation and path planning: map-building, map-less, and map-based systems [19]. *Map-building* systems use a UAV to simultaneously build a map of the environment while flying within it. This is referred to as the "simultaneous localization and mapping" (**SLAM**) computational problem. Work in this area commonly uses onboard sensors (such as LiDAR) to perform both the mapping and localization. Algorithms used for SLAM operate with probabilistic quantities (a dimensionless set of values that build the environment), with two major implementations being the extended Kalman filter (**EKF**) SLAM and GraphSLAM [28]. The benefits of SLAM can be clearly seen for use in environments with unreliable GPS navigation [13], environment reconstruction [25], or as a means to perform autonomous navigation as a whole.

A *map-less* system seeks to perform navigation without using any maps. This is often accomplished using computer vision and reinforcement learning techniques [15], in which information about an environment is constantly being given to a UAV and decisions are made using that data. One method uses a computer vision algorithm to identify the features of obstacles in a process known as *feature tracking*, and the position of a UAV within its environment is calculated based on such features [22]. This method of path planning is great for unknown locations; however, due to the increasing amount of LiDAR data collected from USGS that covers the majority of the United States, there are less computer intensive methods of path planning.

Finally, the *map-based* system, which is the type of system that this paper is based upon, uses a predefined map layout and allows a UAV to navigate based on that information. One way obstacles are defined in a map-based system is through an obstacle grid, which is a 2D grid that stores information on whether a cell is an obstacle or not by giving either a value of 0 or 1, with one representing an obstacle and zero representing free space. Dryanovski *et al.* developed a method for producing multi-volume occupancy grids, which store three-dimensional information about about both free and occupied space of an environment and continuously corrects a map when given updated sensor information [9]. Another common method of mapping within a map-based system uses *octrees*. With octrees, a node of space is represented as a 3D cube, called a voxel. Through recursion, each voxel is further subdivided into eight children until a minimum size is obtained, which is predefined and also dictates the resolution of the octree. Because octrees are represented as a tree

data structure, previous (lower) resolutions can be easily accessed at any time [14]. However, octrees by nature can only provide approximations of an environment, as voxels are represented as cubes. As such, an approximation of an object with surface curvature (i.e. a sphere or dome) will never be perfectly represented regardless of octree depth.

3.2 LiDAR and UAVs

LiDAR is a remote sensing method that uses pulsed lasers to emit light waves towards an environment. The time it takes for these pulses to bounce off of objects in the environment and return to the sensor is measured, and that distance is used to determine the height of the objects. LiDAR points are initially returned as spherical coordinates, but can be converted into cartesian using mathematical formulae, which are displayed below with radius ρ, polar angle θ, and azimuth φ.

$$
\begin{aligned}
x &= \rho \sin\theta \cos\varphi \\
y &= \rho \sin\theta \sin\varphi \\
z &= \rho \cos\theta
\end{aligned}
$$

Alternatively, an onboard Inertial Processing Unit (**IMU**), often equipped on aircraft and used to report the orientation of the vessel, can be used to automatically convert LiDAR data from spherical to cartesian coordinates.

The applications of LiDAR scanners often mesh well with the operational capabilities of UAVs, making their usage together well documented in research. Specifically, there have been many applications resulting from equipping a UAV with a LiDAR scanner for mapping, inspection, or measuring purposes. The general process for direct mapping and some inspection applications is to obtain a point cloud via a LiDAR-equipped UAV, and then perform a feature extracting technique to remove noise and enhance the data related to the problem at hand. G. E. Teng *et al.* used this process to inspect the features and security of a power line [26]. Measuring applications follow a similar process: develop post-processing algorithms that interpret the obtained LiDAR data for desired information. Chisholm *et al.* have recently used a LiDAR equipped UAV to estimate the diameter of trees below canopies [8], which shows a promising future for remote sensing using LiDAR in GPS-denied environments. It should be mentioned that most applications with LiDAR and UAVs involve obtaining a scan using the UAV, and then afterwards (or simultaneously in the case of SLAM) performing some processing on the data. This paper focuses on performing UAV flight operations on LiDAR scans obtained via another source (from aircraft).

3.3 UAV Path Planning

Path planning techniques all exist to solve the same problem: given an environment with obstacles and a start/goal node, how can a path be constructed between these two nodes while avoiding all obstacles? As described in [19], there are two main types of UAV path planning: global and local. *Global path planning* techniques find a path from the start to the end node within a global (static) map environment. *Local path planning* works to dynamically find subsets of a global path based on UAV state and updated stimuli. It can be seen how global and local path planning are related. A global path remains constant, but can be adjusted with intermediate local paths based on new information obtained from sensors (i.e. appearance of an obstacle, environment that differs from expected, etc.). There are many different algorithms that achieve these goals, with each belonging to a certain paradigm that describes its process at a categorical level [29]. This review will mostly cover research involving UAV path finding for node and sample-based algorithms with a focus on real-word environments.

Sampling-based algorithms, such as the Rapidly-exploring Random Tree (**RRT**) and RRT* algorithms operate by *sampling* points randomly in some $\mathbb{R}^n$ space. RRT then uses these points as seeds to grow trees from, and connects the grown trees together to form a path between a start and goal point. Many variations have been made to the base RRT algorithm for UAV path planning purposes, such as introducing

a bidirectional variant [20] or reducing the search space by growing trees from the start and end points instead of just the initial point [31] to reduce computation time. 3D UAV navigation using sampling-based algorithms and point clouds has also been documented [32], however these algorithms may fail if cavities are present in the point cloud scans.

Node-based algorithms, such as the *A* algorithm* have been widely used for UAV path planning purposes as well. This is due to its ability to be easily modified to fit the subjects of specific papers [10]. It works by calculating the total cost $F(n)$ of each node using two components, a metric for node distance travelled so far $g(n)$ and a heuristic function $h(n)$. The heuristic function chosen can vary the resulting path greatly, and is itself a subject of many papers. By processing the nodes with the lowest cost first using a priority queue, the algorithm prevents unnecessary exploration of too costly nodes, which reduces the computation time. The formula to calculate a node's cost value is shown below.

$$F(n) = g(n) + h(n)$$

The A* algorithm has been shown to generate almost optimal paths for 3D UAV path planning, with shorter lengths and less computation time compared to the popular sampling-based algorithm, RRT/RRT* [30, 6].

3.4 Collision Avoidance

Collision avoidance is necessary for safe and fully-functional autonomous flight, and the different systems used to accomplish this have undoubtedly contributed to the vast amount of research that has been done in this area. The two key concepts that make up collision avoidance systems are sensing and detection, and maneuver approaches [24]. Sensing and detection, as can be implied from the name, constitute devices or methods that allow a UAV to position itself relative to other obstacles. One technique is optic flow, where motion is detected from brightness patterns of an image and used to calculate an optical flow field, consisting of vectors for each pixel [7, 21, 12]. Stereo-vision techniques use the triangulation of two sensors to obtain depth maps, which is used to perform an avoidance [23]. Laser sensor techniques use sensors such as LiDAR scanners to perform detection and avoidance, and can be seen in [18, 16].

Maneuver approaches are more varied, as the optimality of an approach would depend on the sensing method used and environment that the UAV is operating in. Many different techniques have been developed, and are covered by [24].

4 Approach

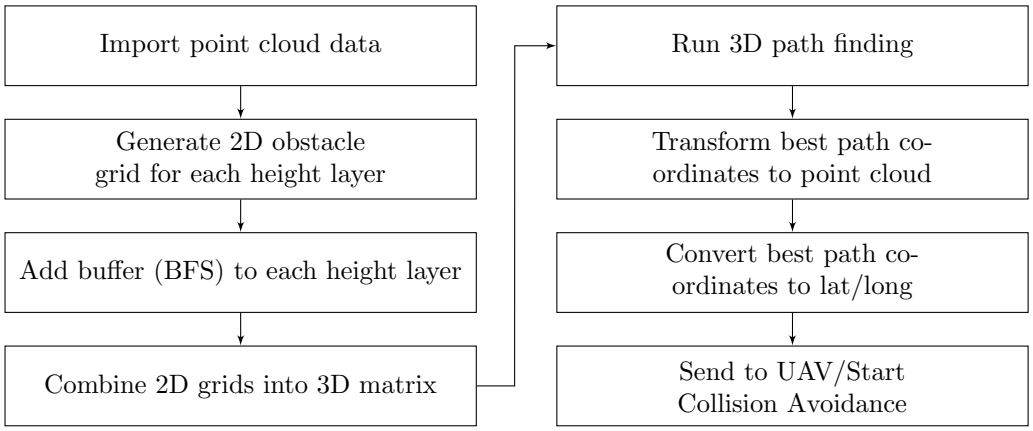

Figure 2: Overview of process from importing data to sending way points to UAV.

4.1 Software and Equipment Used

Software/Equipment	Reasoning
OpenTopography	Used to access all point cloud datasets for testing purposes. OpenTopography API was used to access datasets remotely.
UTM	Python library used to convert WGS84 encoded coordinates to latitude and longitude to send to the UAV.
LASzip	Used initially to convert .las files to ASCII file formats.
Open3D	Python visualization library used due to its available functionality for point cloud operations.
ARDU Pilot	Used for communication from computer to UAV and for simulated UAV testing purposes.
Mission Planner/MAVProxy	Simulation and transmission software that can control UAVs for realistic simulation testing connecting to ARDU Pilot over IP addresses.
Selenium	Python library used with Chrome to remotely navigate OpenTopography.

4.2 Process

Our process for computing path planning and collision avoidance for aerially obtained point clouds is shown in Figure 2 and outlined in detail below. The process has been subdivided into multiple sections for better comprehension. Each part will discuss the methodology used to accomplish the goal outlined in the title of the section.

Figure 3: Simulated path in point cloud vs. satellite view using ARDU Pilot. [2]

4.2.1 Matrix Generation

To generate the 3D matrix M that is used for path planning, we first must import the .las file that contains the desired point cloud into *laspy*, a Python library. Additionally, we use *laspy* commands to extract the maximum and minimum point cloud points from the data. The difference between the maximum and minimum Z point (Z_{max} and Z_{min}) defines the depth of the matrix. Similarly, the difference between the maximum and minimum Y (Y_{max} and Y_{min}) and X points (X_{max} and X_{min}) define the rows and columns, respectively.

$$M_x = X_{max} - X_{min}$$
$$M_y = Y_{max} - Y_{min}$$
$$M_z = Z_{max} - Z_{min}$$

An elevation grid is then created from the imported .las file. In this 2D grid, the X and Y coordinates represent the relative point cloud location, and the value within that cell represents the height of that section in meters (1 cubic meter per cell by default). This is done using code modified from [17], and contains an interpolation gradient to account for data loss from the point cloud data.

Figure 4: Two cross-sectional 2D grids displaying different heights generated using Algorithm 1. [1]

To convert the 2D grid into a 3D occupancy grid, we loop using height values that start from the minimum Z point to the maximum Z point in the point cloud, creating a temporary 2D grid and assigning a value of 1 or 0 depending on whether or not the cell is an obstacle or not, respectively. This is determined by subtracting the current height value from the height found in the elevation grid at the specified pixel. A threshold value is used to create a *vertical* buffer for object classification within the matrix. The algorithm is displayed as Algorithm 1. A *horizontal* buffer is created using an implementation of the Breadth First

Algorithm 1 create3DMatrix($height\_values$, $buffer\_size$, $threshold$)

$M \leftarrow matrix$
$height\_values \leftarrow H$
for $cur\_height = Z_{min}$ to Z_{max} **do**
    $G \leftarrow grid$
    for $x, y \in G$ **do**
        if $cur\_height - H[x, y] \leq threshold$ **then**
            $G[x, y] \leftarrow obstacle$
        else
            $G[x, y] \leftarrow free$
        end if
    end for
    $G \leftarrow bfs(G, buffer\_size)$
    $M \leftarrow append(G)$
end for
return M

Search (**BFS**) algorithm. Obstacles groupings are identified and expanded outwards by a predefined amount of meters. The inclusion of the vertical and horizontal buffers were implemented for three main reasons: to help prevent potential collisions due to noise within the point cloud, to account for the size of the UAV itself, and to provide a safe distance between the UAV and the obstacle for path finding and smoothing purposes.

4.2.2 Path Planning and Conversions

To develop the path planning portion of this project, we needed something easily modifiable that would work smoothly with the matrix generation technique discussed in the previous section. For these reasons, the A* path finding algorithm was selected and used as a base for further modification. To account for all directions in a cubic grid, our A* will evaluate 26 neighbors at every current node. This accounts for the four cardinal directions and diagonals on the same Z plane, as well as the same for one unit above and below, and straight up/straight down. The path obtained from A* consists of 3D points in space in relation to the matrix, with the origin at $(0, 0, 0)$. This path cannot be directly related to the point cloud, as the coordinates of the point cloud are encoded using some reference system (such as WGS84 Web Mercator), so it must first be

transformed. Algorithm 2 presents the method for transforming the path coordinates to point cloud space using the best path and the maximum/minimum point values from the .las file.

Algorithm 2 transformToPointCloud(*best_path*, *las_extremes*)

for $(x, y, z) \in best\_path$ **do**
    $x \leftarrow x + X_{min}$
    $y \leftarrow Y_{max} - y$
    $z \leftarrow Z_{min} + z$
end for
return *best_path*

With the best path now encoded in WGS84 to coincide the point cloud data, we can apply path smoothing using Bézier curves, the formula of which is presented below as a function of time t.

$$\mathbf{P}(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i \mathbf{P}_i$$

Bézier curves produce smooth paths through n control points. This defines the degree of the curve (1 point for linear, 2 for quadratic, etc.). By using an n-order Bézier curve, we create a smoothed line using each point from the best path as a control point. This provides for the most accurate representation of the non-smoothed path, which is shown in Figure 5.

We also use this curve to produce a more realistic flight path of a UAV within the simulation, and this in turn shortens the overall distance of the path. However, with the Bézier path smoothing applied, the number of points within the path increases dramatically, the higher amount of points results in a more accurate path to be generated due to greater amount of interpolation between each point. Because the focus of this project is for real-life operation, additional conversions must be made to the best path coordinates. Namely, the previously transformed path must be converted to latitude and longitude so that a UAV may receive the path using an onboard GPS. To do this, the Python library *utm* is used. The conversion function must be supplied with the UTM zone of the physical location containing the point cloud scan and whether it exists in the southern or northern hemisphere both gathered from the evaluation of the starting node.

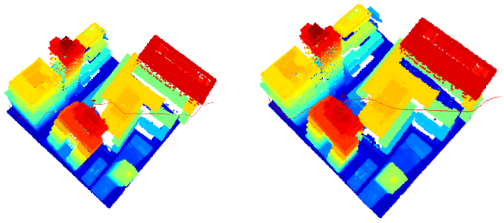

Figure 5: Comparison of non-smoothed and smoothed path. [3]

The final path F is a 2D array that contains the latitude and longitude coordinates, and the unconverted height (Z value) for each point in the smoothed path. This path is now able to be sent to a UAV, and the accuracy of the conversions from point cloud to real world GPS way points can be shown in Figure 3.

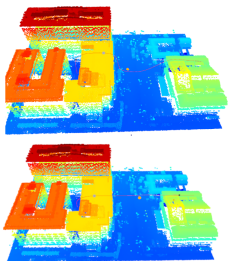

Figure 6: Comparison of path before and after energy model. [2]

Controlling the energy consumption of UAVs is also an important part of battery and resource management, and a primitive model for this is included within our path planning algorithm based off the research done in [5]. The weight of each node $F(n)$ is not only calculated using the cost to the current node $g(n)$ and the heuristic distance to the ending node $h(n)$, but also an energy consumption factor $e(n)$, which weights neighboring nodes differently based on their direction from the current node.

$$\mathbf{F(n)} = \mathrm{g(n)} + \mathrm{e(n)} + \mathrm{h(n)}$$

Table 3 shows the values used for this model. When including this model, the generated path does not make unneeded up-

wards movement and will only move upwards if absolutely necessary to reach the goal node, as shown in Figure 6.

4.2.3 Collision Avoidance

The collision avoidance procedure is currently simulated by measuring the distance between the UAV and an obstacle using euclidean distance, the formula of which is given below. We assume a working sensor is onboard the UAV.

$$\mathbf{d}(p, q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2 + (p_z - q_z)^2}$$

After the best path has been found using the methodology discussed in Section 4.2.2, an avoidance trajectory is calculated for each way point in the best path. This is done by evaluating the 3D obstacle matrix in a set of predefined directions by continuously extending a line until an obstacle is met, or a distance threshold is reached. The set of potential avoidance directions can be decided based on the type of obstacle encountered. For example, a tall object could only include horizontal movement, while a small object could include the possibility of going over or under it. These directions are all oriented the same regardless of UAV heading except for horizontal movement, which is based on the heading H of the UAV (obtained from the best path it is following) and calculated as follows:

$$H = \arctan \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$$

This calculation is done for each way point, and the lines that meet the distance threshold are inserted into a priority queue which intrinsically sorts based on the euclidean distance between the avoidance trajectory and the ending (goal) node. The line with the shortest distance to the end is selected as it will be at the front of the priority queue. The idea behind pre-computing the avoidance trajectory is to reduce the computation time spent on maneuvering during the event of collision avoidance, and allow more time to recalculate the path. If an obstacle is met, the general flow of operation is: move UAV to precomputed avoidance trajectory, then recalculate A* from that avoidance trajectory. Algorithm 3 shows how the trajectory calculation is done.

Algorithm 3 findAvoidances(*matrix*, *best_path*, *threshold*)

```
A ← all_avoidance
for p ∈ best_path do
    Q ← priority_queue()
    for d ∈ directions do
        t ← threshold
        while in_bounds(d, matrix) and threshold > 0 do
            p_x ← p_x + d_x
            p_y ← p_y + d_y
            p_z ← p_z + d_z
            t ← t − 1
        end while
        if t == 0 then
            Q ← push(euclidean([p_x, p_y, p_z], end_node), [p_x, p_y, p_z])
        end if
    end for
    A ← append(Q_front)
end for
return A
```

After the avoidance trajectories have been generated, they are ran through the same transformations and conversions discussed in Section 4.2.2. Therefore, the avoidance trajectory and the new path can be sent to the drone with no issues. However, there are some current limitations with our avoidance method. Limited movement options could prevent a successful avoidance or cause the UAV to bounce between two trajectories, becoming stuck. To combat this, we use a hash table that stores an avoidance direction for each obstacle encountered. If a direction has already been selected and used, and the obstacle still needs to be avoided, the same direction will be used again. If there hasn't been a successful avoidance after x tries, the key is deleted and a new direction is picked and tried.

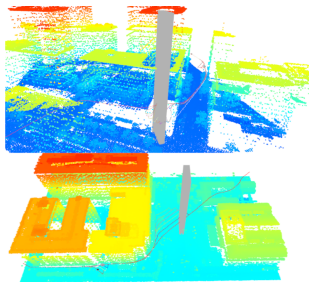

Figure 7: Example of collision avoidance for two obstacle types. [2]

4.2.4 Automation using API

A final goal of this research project is to create a seamless process from the input of GPS coordinates to the UAV flying the generated path. Thus, we needed to utilize OpenTopography's API to allow for remote file downloading. This API allows users to input a search polygon (in the form of minimum and maximum longitude/latitude coordinates) and returns a list of datasets and other information in .json format that contain points in the search area. This .json file also holds critical information regarding each dataset that is extracted from the used dataset. However, we found that not every dataset returned by the API contained points within the search area, thus additional code was used to prune returned datasets from the parsed .json until only the working ones remained. Using *Selenium*, we then automated the process of logging into and downloading a .las file obtained from the API and using that file for the rest of the program remotely, completely automating the process after initial coordinates are given.

5 Testing

5.1 Simulation

All tests were computed using the following specifications: 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 2.70 GHz Processor, RAM: 32.0 GB (31.7 GB usable), System Type: 64-bit, x64-based processor, Graphics Card: Intel iRISxe. The first testing area used is located in Seattle, Washington (137,479.904 square feet, coordinates: 47.60002984, -74.32896467 to 47.59862772, -74.32769222) with a maximum height of 59 m. The secondary testing area used was New York, New York (202,476.83 square feet, coordinates: 40.70502297, -74.01328363 to 40.70357673, -74.01165960) with a maximum height of 156 m. The starting point and ending point used were the top left corner and the bottom right corner and started from the lowest non-obstructed height and ended 10 m above the starting point. The path was generated by our adapted A* algorithm with a buffer size of 3. Each path was generated using the following settings: 26 nodes (3D movement including diagonals in the Z directions) or 10 nodes of freedom (3D movement without diagonals in the Z directions). In addition, we tested the effects of path smoothing and using euclidean distance for the $g(n)$ cost as well as the $h(n)$ cost (called sqrt in the tables). We run these tests with and without the energy function to compare the difference. Path length was measured in units using euclidean distance between each way point. All paths generated were possible to fly without collisions.

5.1.1 Without Energy Function

For our baseline testing we compared the distance of each path, computational time for the path generation, and the change of elevation in that path for 26 node or 10 node paths with various levels of path smoothing

added. This section was also ran using euclidean distance and without the use of our energy function. This test was used to create a standard set of data to compare among itself and with the addition of different factors as displayed in this section.

Testing Settings	Distance	A* Computation Time	Elevation Change(m)
26 Nodes (path smoothing, no sqrt)	18868.6986	0.0148	20.7395
10 Nodes (path smoothing, no sqrt)	18974.0368	**0.005**	11.9932
26 Nodes (path smoothing, sqrt)	**18589.648**	2.8257	**11.779**
10 Nodes (path smoothing, sqrt)	18665.4663	0.9536	11.9952
26 Nodes (no path smoothing, no sqrt)	21426.6782	0.0117	44
10 Nodes (no path smoothing, no sqrt)	20762.2366	0.0178	12
26 Nodes (no path smoothing, sqrt)	19192.2423	2.8640	12
10 Nodes (no path smoothing, sqrt)	19913.7084	0.9038	12

Table 1: Data found on Seattle simulation without energy model, euclidean distance heuristic. [2]

Testing Settings	Distance	A* Computation Time	Elevation Change(m)
26 Nodes (path smoothing, no sqrt)	28135.3111	0.0602	87.1705
10 Nodes (path smoothing, no sqrt)	30308.2202	**0.026**	100.868
26 Nodes (path smoothing, sqrt)	**25340.025**	50.364	**46.999**
10 Nodes (path smoothing, sqrt)	26282.4647	11.167	46.9999
26 Nodes (no path smoothing, no sqrt)	43521.8591	0.0576	219
10 Nodes (no path smoothing, no sqrt)	46636.2481	0.0320	155
26 Nodes (no path smoothing, sqrt)	27195.7931	49.713	47
10 Nodes(no path smoothing, sqrt)	30161.0173	11.121	47

Table 2: Data found on New York simulation without energy model, euclidean distance heuristic. [3]

5.1.2 Energy Function Added

For our energy consumption model testing we compared the distance of each path, computational time for the path generation, and the change of elevation in that path for 26 node or 10 node paths with various levels of path smoothing added. The energy model derived from [5] was also included in the path planning algorithm for these tests. Our modifications to the weight of the nodes based on their respective direction is as shown in Table 3. The energy function was implemented using a hash table where the direction is the key, and the weight is the value. Each neighboring node is associated with a direction and the corresponding energy weight value is added when calculating the $F(n)$ cost of the node. This addition was created to try and limit the amount of strenuous maneuvers the UAV preforms in the flight and optimise battery life.

Direction	Energy Weight
horizontal	1.0
horizontal angle	1.3
down	1.3
down cardinal	1.5
down angle	1.7
up	8.0
up cardinal	8.4
up angle	8.8

Table 3: Energy weights used in A* algorithm.

Testing Settings	Distance	A* Computation Time	Elevation Change(m)
26 Nodes (path smoothing, no sqrt)	**19631.779**	**0.9207**	**10**
10 Nodes (path smoothing, no sqrt)	20235.2898	13.1873	**10**
26 Nodes (path smoothing, sqrt)	19644.8975	54.8695	**10**
10 Nodes (path smoothing, sqrt)	20242.1704	14.5067	**10**
26 Nodes (no path smoothing, no sqrt)	20245.7593	1.03158	**10**
10 Nodes (no path smoothing, no sqrt)	20927.9220	14.2147	**10**
26 Nodes (no path smoothing, sqrt)	20245.7593	55.7438	**10**
10 Nodes(no path smoothing, sqrt)	20927.9220	14.1579	**10**

Table 4: Data found on Seattle simulation with energy model, euclidean distance heuristic. [2]

Testing Settings	Distance	A* Computation Time	Elevation Change(m)
26 Nodes (path smoothing, no sqrt)	**26378.044**	57.2290	**43**
10 Nodes (path smoothing, no sqrt)	29202.0732	66.5170	**43**
26 Nodes (path smoothing, sqrt)	26563.8851	247.682	**43**
10 Nodes (path smoothing, sqrt)	29017.7867	71.9563	**43**
26 Nodes (no path smoothing, no sqrt)	28146.0992	**52.457**	**43**
10 Nodes (no path smoothing, no sqrt)	30643.8600	68.3621	**43**
26 Nodes (no path smoothing, sqrt)	28146.0992	247.149	**43**
10 Nodes(no path smoothing, sqrt)	30643.8600	61.6462	**43**

Table 5: Data found on New York simulation with energy model, euclidean distance heuristic. [3]

5.1.3 Collision Avoidance

To test collision avoidance, we performed 20 tests each on various point clouds obtained from USGS 3DEP scans in urban environments (cities, downtown areas, etc.). Each simulation contained three different types of obstacles (static sphere, dynamic sphere, and vertical); the purpose of including all at once is to account for potential worst case scenarios. The obstacle positions were randomly placed along the generated path for each test, and the successes and failures were recorded and documented in Table 6. Selected examples of various successes and failures (on the top and bottom of the figure, respectively) can be seen in Figure 8. The point cloud map was omitted for better visibility, but the point cloud used can be seen in the lower left corner.

Test Map	# of Successes	Percentage	Size(m)
New_York_2	12 out of 20	**60%**	168x186
Seattle	10 out of 20	**50%**	98x156
Statue_of_Liberty	10 out of 20	**50%**	127x107
AuburnU	15 out of 20	**75%**	401x182

Table 6: Collision avoidance testing on various maps using randomly placed obstacles along the path. [1, 3, 2]

6 Results

A few observations can be made from the tables shown above. To begin, the results without the energy function will be discussed. In all cases without the energy model, 26 nodes outperformed 10 nodes in distance at the expense of more computation time. The path smoothing greatly shortened each path compared to the non-smoothed version with the same settings. In addition, the additional euclidean measurement shortened each path even more at the expense of more computation time. Therefore, the overall shortest path settings without the energy model were 26 nodes, including path smoothing and the added square root function.

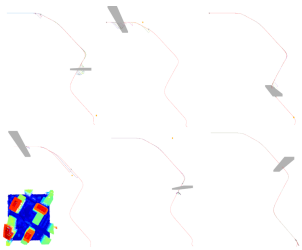

Figure 8: Selected collision avoidance simulations during testing on New_York_2.

However, due to the vastly shorter computation time and only minor increase in distance, the best overall path settings is arguably the 26 Nodes (path smoothing, no sqrt) setting. With the energy function, the results vary slightly. Computation time is increased significantly for each option, and the shortest path is generated using 26 nodes with path smoothing and no added square root. This is because the energy function weights eliminates the need for it. The elevation change also remains the same for each path. Overall, the addition of the energy model makes the paths slightly longer in distance, much longer in computation time, and decreases the change in elevation. The reason for the longer path is because of the drone preferring to go straight down/up instead of at an angle making the path extend slightly. These paths will conserve much more battery power due to the path being the least strenuous on the UAV. Our proposed collision detection algorithm successfully avoided all obstacles on 50% to 75% of the tests performed, and the average of success from all tests is 59%. The higher success rates were attributed to larger point clouds with more flat terrain and less building density. Avoidance failures were most commonly the result of an avoidance needing to go out of bounds, no valid avoidance trajectory generated at a necessary way point, and sub-optimal direction selection between two open choices.

7 Conclusion

With the growing topographical data available through programs like USGS 3DEP, path planning for UAVs is more accessible than ever before. We have developed and shown the validity of our process for 3D path planning using aerially obtained point clouds from USGS 3DEP databases with various simulations. In addition, we have added necessary modifications to account for UAV size, added path smoothing, and reduced energy consumption. We have also shown a primitive collision detection algorithm that uses the point cloud data to pre-compute avoidance trajectories along the flight path to be used if needed. The process for obtaining point clouds for a particular area has been fully automated with the use of Selenium and the OpenTopography API. Through testing different settings of our path generation process, we have found the settings (path smoothing, sqrt function, # of nodes checked) that will most likely lead to the shortest path being generated with and without the use of our energy function. Finally, we have shown the precision of our method in converting the path into latitude and longitude to be used with a GPS-equipped UAV via ARDU Pilot.

While our process works well in simulation, real-life testing and the continuation of testing different factors to improve optimality should be tested. Further work to improve the collision detection algorithm must be done to increase avoidance success rates. The addition of changing some values like grid resolution should also be implemented in the future.

References

[1] USGS 3DEP (2021). *GA Central 2 2018. U.S. Geological Survey 3D Elevation Program, distributed by OpenTopography.*

[2] USGS 3DEP (2021). *WA KingCo 1 2021. U.S. Geological Survey 3D Elevation Program, distributed by OpenTopography.*

[3] USGS 3DEP. *NY NewYorkCity. U.S. Geological Survey 3D Elevation Program, distributed by OpenTopography.*

[4] "A critical review on unmanned aerial vehicles power supply and energy management: Solutions, strategies, and prospects". In: *Applied Energy* 255 (2019), p. 113823. ISSN: 0306-2619. DOI: https://doi.org/10.1016/j.apenergy.2019.113823.

[5] Hasini Viranga Abeywickrama et al. "Comprehensive Energy Consumption Model for Unmanned Aerial Vehicles, Based on Empirical Studies of Battery Performance". In: *IEEE Access* 6 (2018), pp. 58383–58394. DOI: 10.1109/ACCESS.2018.2875040.

[6] João Braun et al. "A Comparison of A* and RRT* Algorithms with Dynamic and Real Time Constraint Scenarios for Mobile Robots". In: Jan. 2019, pp. 398–405. DOI: 10.5220/0008118803980405.

[7] Haiyang Chao, Yu Gu, and Marcello Napolitano. "A survey of optical flow techniques for UAV navigation applications". In: *2013 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2013, pp. 710–716. DOI: 10.1109/ICUAS.2013.6564752.

[8] Ryan A. Chisholm et al. "UAV LiDAR for below-canopy forest surveys". In: *Journal of Unmanned Vehicle Systems* 01.01 (2013), pp. 61–68. DOI: 10.1139/juvs-2013-0017. eprint: https://doi.org/10.1139/juvs-2013-0017. URL: https://doi.org/10.1139/juvs-2013-0017.

[9] Ivan Dryanovski, William Morris, and Jizhong Xiao. "Multi-volume occupancy grids: An efficient probabilistic 3D mapping model for micro aerial vehicles". In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2010, pp. 1553–1559. DOI: 10.1109/IROS.2010.5652494.

[10] Daniel Foead et al. "A Systematic Literature Review of A* Pathfinding". In: *Procedia Computer Science* 179 (2021). 5th International Conference on Computer Science and Computational Intelligence 2020, pp. 507–514. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2021.01.034. URL: https://www.sciencedirect.com/science/article/pii/S1877050921000399.

[11] Boris Galkin, Jacek Kibilda, and Luiz A. DaSilva. "UAVs as Mobile Infrastructure: Addressing Battery Lifetime". In: *IEEE Communications Magazine* 57.6 (2019), pp. 132–137. DOI: 10.1109/MCOM.2019.1800545.

[12] Zdzislaw Gosiewski, Jakub Ciesluk, and Leszek Ambroziak. "Vision-based obstacle avoidance for unmanned aerial vehicles". In: *2011 4th International Congress on Image and Signal Processing*. Vol. 4. 2011, pp. 2020–2025. DOI: 10.1109/CISP.2011.6100621.

[13] Sebastian Hening et al. "3D LiDAR SLAM Integration with GPS/INS for UAVs in Urban GPS-Degraded Environments". In: *AIAA Information Systems-AIAA Infotech @ Aerospace*. DOI: 10.2514/6.2017-0448. eprint: https://arc.aiaa.org/doi/pdf/10.2514/6.2017-0448. URL: https://arc.aiaa.org/doi/abs/10.2514/6.2017-0448.

[14] Armin Hornung et al. "OctoMap: an efficient probabilistic 3D mapping framework based on octrees". In: *Autonomous Robots* 34 (2013), pp. 189–206.

[15] Sunggoo Jung and David Hyunchul Shim. *Mapless Navigation: Learning UAVs Motion forExploration of Unknown Environments*. 2021. DOI: 10.48550/ARXIV.2110.01747. URL: https://arxiv.org/abs/2110.01747.

[16] Cezary Kownacki. "A concept of laser scanner designed to realize 3D obstacle avoidance for a fixed-wing UAV". In: *Robotica* 34.2 (2016), pp. 243–257.

[17] Joel Lawhead. *Learning Geospatial Analysis with Python*. Birmingham, United Kingdom: Packt Publishing, 2019.

[18] Liang Lu et al. "Laser-based Collision Avoidance and Reactive Navigation using RRT and Signed Distance Field for Multirotor UAVs". In: *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2019, pp. 1209–1217. DOI: 10.1109/ICUAS.2019.8798124.

[19] Yuncheng Lu et al. "A survey on vision-based UAV navigation". In: *Geo-spatial Information Science* 21.1 (2018), pp. 21–32. DOI: 10.1080/10095020.2017.1420509. eprint: https://doi.org/10.1080/10095020.2017.1420509. URL: https://doi.org/10.1080/10095020.2017.1420509.

[20] Li Meng, Song Qing, and Zhao Qin Jun. "UAV path re-planning based on improved bidirectional RRT algorithm in dynamic environment". In: *2017 3rd International Conference on Control, Automation and Robotics (ICCAR)*. 2017, pp. 658–661. DOI: 10.1109/ICCAR.2017.7942779.

[21] Paul Clark Merrell, Dah-Jye Lee, and Randal W Beard. "Obstacle avoidance for unmanned air vehicles using optical flow probability distributions". In: *Mobile Robots XVII*. Vol. 5609. International Society for Optics and Photonics. 2004, pp. 13–22.

[22] Ivan Fernando Mondragon et al. "Visual Model Feature Tracking For UAV Control". In: *2007 IEEE International Symposium on Intelligent Signal Processing*. 2007, pp. 1–6. DOI: 10.1109/WISP.2007.4447629.

[23] Jongho Park and Youdan Kim. "Stereo vision based collision avoidance of quadrotor UAV". In: *2012 12th International Conference on Control, Automation and Systems*. 2012, pp. 173–178.

[24] Hung Pham et al. "A survey on unmanned aerial vehicle collision avoidance systems". In: *CoRR* abs/1508.07723 (2015). arXiv: 1508.07723. URL: http://arxiv.org/abs/1508.07723.

[25] Zhexiong Shang and Zhigang Shen. "Real-time 3D Reconstruction on Construction Site using Visual SLAM and UAV". In: *CoRR* abs/1712.07122 (2017). arXiv: 1712.07122. URL: http://arxiv.org/abs/1712.07122.

[26] GE Teng et al. "Mini-UAV LiDAR for power line inspection". In: *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci* (2017), pp. 297–300.

[27] *USGS 3DEP Topographic Data Quality Levels (QLs)*. URL: https://www.usgs.gov/3d-elevation-program/topographic-data-quality-levels-qls.

[28] Wikipedia. *Simultaneous localization and mapping — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Simultaneous%20localization%20and%20mapping&oldid=1092128572. [Online; accessed 08-June-2022]. 2022.

[29] Liang Yang et al. "A literature review of UAV 3D path planning". In: *Proceeding of the 11th World Congress on Intelligent Control and Automation*. 2014, pp. 2376–2381. DOI: 10.1109/WCICA.2014.7053093.

[30] Christian Zammit and Erik-Jan Van Kampen. "Comparison between A* and RRT algorithms for UAV path planning". In: *2018 AIAA guidance, navigation, and control conference*. 2018, p. 1846.

[31] Denggui Zhang, Yong Xu, and Xingting Yao. "An Improved Path Planning Algorithm for Unmanned Aerial Vehicle Based on RRT-Connect". In: *2018 37th Chinese Control Conference (CCC)*. 2018, pp. 4854–4858. DOI: 10.23919/ChiCC.2018.8483405.

[32] Zhaoliang Zheng, Thomas R. Bewley, and Falko Kuester. "Point Cloud-Based Target-Oriented 3D Path Planning for UAVs". In: *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2020, pp. 790–798. DOI: 10.1109/ICUAS48674.2020.9213894.