Preference-Based Path-Planning for Autonomous Robots

Gabriel Kepets
The Cooper Union
Department of Electrical Engineering
New York, USA
gkepets@gmail.com

Ayden Shankman
The Cooper Union
Department of Electrical Engineering
New York, USA
aydenshankman1@gmail.com.com

Netanel Fiorino
The Cooper Union
Department of Mechanical Engineering
New York, USA
netanelmf@gmail.com

***Abstract*—Efficient travel for an autonomous robot requires a path-planning algorithm to determine the optimal path to take based on certain criteria. The most common criteria are speed, safety, and energy, as most autonomous robots are designed to operate in remote and/or difficult terrain where recharging, repair, or replacement is difficult or impossible. Most path-planning algorithms are catered toward specific robots, environments, or tasks, so the algorithms prioritize a static set of criteria, typically one or more of the three criteria previously listed. Many robots would benefit from having a path-planning algorithm that can dynamically prioritize a combination of the three criteria while also being generalizable to any type of autonomous robot. We present a novel path-planning algorithm that accounts for a robot's specifications, including its mass, dimensions, energy usage, and climbing abilities, and allows the user to set preferences for how much to prioritize speed, safety, and energy when calculating paths on any type of terrain. INCLUDE RESULTS**

***Index Terms*—Autonomous robots, path-planning, A*, LiDAR**

I. Introduction

Autonomous robotics is a field of study that focuses on the development of robots that are able to operate independently of human control. Today, autonomous robots are used for a variety of applications, including industry, search and rescue, surveillance, and military operations [1]. These robots require algorithms and systems that enable them to perceive their environment and make decisions based on gathered information. One of the most important tasks for a functional autonomous robot is path-planning. Path-planning is the task of finding the optimal path for a robot to follow between points in a defined space (often a physical space). Paths are considered optimal based on specific criteria; the most common criteria for autonomous robots are speed, safety, and energy [1]. Speed relates to how much time it takes for a robot to get to a destination. Safety relates to how risk-averse or traversable a path is for a given robot. Energy relates to how energy-efficient a path is for a given robot.

However, the vast majority of path-planning algorithms are catered to very specific robots, environments, and tasks, so they are designed to only optimize for a static set of one or two of the three criteria mentioned [1]. An example of such a path-planning algorithm is the one deployed on the Mars rovers known as Spirit, Opportunity, and Perseverance, which strictly prioritize safety when generating the path they should take [2]. Additionally, most path-planning algorithms are designed to account for the exact specifications and capabilities of a particular robot, such as its dimensions and battery capacity. However, if a robot's objective or specifications change, it is possible that the criteria or decision-making for its path-planning algorithm would need to change as well. If a Mars rover, which usually prioritizes only safety when calculating paths, suddenly becomes low on power, it may want to switch to prioritizing energy conservation when calculating new paths. To the best of our knowledge, there is currently no path-planning algorithm that allows for adjustable levels of prioritization of the three main criteria of speed, safety, and energy, and that is also generalizable to any type of autonomous robot.

In this paper, we present a novel path-planning algorithm that addresses this issue. A user inputs various specifications of a robot, such as its mass, dimensions, energy usage, and climbing abilities. The algorithm then converts point cloud data collected by a 3D LiDAR (also provided by the user) and converts it to a 2.5D map, which is a 2D grid that represents a 3D surface or terrain by mapping the height and normals information of each point on the surface to the grid. A probabilistic road map (PRM), which is a graph comprised of a set of randomly sampled points that are each connected to a fixed amount of their nearest neighbors, is then generated on top of the height map. The path-planning algorithm A* is then applied to calculate the optimal path from one point on the PRM to another, where the cost of each edge between two points is calculated based on the data from the underlying map and the robot's specifications. Separate costs are calculated for speed, safety, and energy and individually weighted based on the user's specified preference and then summed together. Finally, a check is done to determine if the edge is impossible for the robot to traverse, and if it is impossible, a relatively high cost, denoted as the limitation cost, is added to the edge to ensure it is not considered in the path-planning process.

To demonstrate our algorithm, we have created a program with a user interface (UI) that allows a user to input the various specifications of a robot mentioned previously, along with point cloud data from a 3D LiDAR. Following the user input, the program displays a 2.5D height map representing the surrounding terrain of the robot, which can be randomly generated for demonstration purposes or based on LiDAR data inputted by the user. The user can then input a start and goal location for the robot to navigate, as well as their preferences for speed, safety, and energy. The program then uses our path-planning algorithm to calculate the optimal path from the start point to the goal point and displays it on the map for the user to see. Using the same map, the user is able to change their speed, safety, and energy preferences, robot specifications, or start and goal locations to generate new paths, to see how each of these metrics may affect the generated path. To simulate a robot, a 3D LiDAR sensor, and varying terrain, the Unity game engine is used. Unity's "Unity Industry" plugin is used to simulate high-fidelity and real-time simulations of LiDAR sensors and autonomous robots. We have also tested our algorithm on terrain generated by readings from a physical 3D LiDAR to evaluate the algorithm within real-world environments.

II. Background and Related Work

A. A* Algorithm

Our path-planning algorithm utilizes the A* algorithm, which is a widely-used algorithm for finding the least costly path between two nodes in a graph or network. It uses a heuristic to efficiently search for the optimal path. The heuristic is used to estimate the cost of the remaining path, which is combined with the current cost of the path, which drives the algorithm toward picking the most optimal path. The algorithm evaluates nodes in a priority queue, and at each iteration, it selects the node with the lowest total estimated cost to continue exploring. The cost is calculated using the function:

$$f(n) = g(n) + h(n) \tag{1}$$

where n is the next node on the path, $g(n)$ is the cost of the path from the start node to n, and $h(n)$ is a heuristic function that estimates the cost of the path from n to the goal node. A* is guaranteed to find the optimal path without evaluating any point more than once and is commonly used in autonomous robotics, computer games, and mapping [4].

B. Probabilistic Road Map

Many path-planning algorithms rely on an underlying graph that represents the possibilities for movement. Graphs are a collection of points, called nodes, that are connected to each other by edges. A path-planning algorithm that uses a graph finds an ordered collection of nodes to traverse in order to get from the start position to the goal position. Nodes in a graph can represent anything from states that the object is in to physical locations.

The case of a graph that represents physical space will be considered. The simplest type of graph is a grid. A grid has the advantage of being a simple graph that can accurately represent the underlying space, but it may be inefficient due to the number of nodes and edges. The more nodes and edges there are, the more possible paths there are, which means that the path planning algorithm will take a longer time to find the optimal path. Fewer nodes and edges mean that an optimal path will be quicker to find, but the resulting path may be relatively inefficient. In order to remedy this, a Probabilistic Road Map (PRM) is introduced. A PRM is a graph that is generated by randomly sampling nodes [3]. The randomly selected points in a PRM can be connected in a variety of ways; a popular method of connecting them is with K-Nearest-Neighbors. While PRMs will not provide the most optimal path, they can come close in a fraction of the time.

C. 2.5D Mapping, Pathfinding and Path Following In Uneven Terrain

[THIS SECTION NEEDS REVISION]
Researchers at the Federal Research Center in Moscow detail how they implemented a path-planning algorithm for a robot that is meant to traverse uneven terrain. The terrain surrounding the robot is divided into a grid, and each cell is assigned a height that best represents the terrain within it. They present three main procedures in their system: elevation mapping, path planning, and path following [5].

Elevation mapping is the process of sensing the terrain and converting it to a 2.5D height map. In this paper, the authors used an open-source solution [6]. Path planning is the process of finding a path from a start point to an end point. The authors suggest using Theta* or A* as the path-planning algorithm. Path following is the process of controlling a robot to follow a desired path. They use a Model Predictive Path Integral algorithm for path following, which is an iterative algorithm that optimizes trajectories.

The novelty this paper provides is using the 2.5D height map in their path-following algorithm. Slope-roughness and slope-traversability are calculated for the areas of the terrain that the robot may traverse, and those factors are used in the path-following algorithm.

D. NASA's Self-Driving Perseverance

The inspiration for how a user interacts with our path-planning algorithm came from an article from NASA about how they control the Perseverance rover [7]. They explain that an operator is able to observe the rover's surroundings, which are based on readings from onboard cameras and sensors, and can select a destination for the rover to travel to. The rover uses a path-planning algorithm that avoids obstacles and dangerous terrain to find a safe path to the destination, which the user can see projected onto the simulated terrain [7]. Similarly, in the program that demonstrates our algorithm, a user can view the terrain that surrounds a robot, which is based on 3D LiDAR data, and is able to specify a destination, the robot's specifications, as well as their preference for how much to prioritize speed, safety, and energy. Our algorithm then calculates the optimal path based on the inputted settings and displays it on the map of the terrain for the user to see.

E. Continuous-field Path Planning with Constrained Path-dependent State Variables

There has been previous work done on path-planning algorithms that highlight the need to reason about time, risk, and energy when generating paths for autonomous robots, such as the algorithm designed by NASA's Jet Propulsion Laboratories [8]. However, NASA's algorithm is specifically designed for long-range autonomous journeys and does not allow for the dynamic adjustment of the prioritization of multiple criteria when generating paths. Furthermore, NASA's algorithm does not take into account any specifications of a robot, such as its dimensions or climbing capabilities, and only incorporates battery capacity during path-planning. That being said, NASA's algorithm is able to change the cost of a given position when calculating paths based on the characteristics of the path leading to it and is also able to dynamically update and re-plan the path if any changes occur [8]. Our algorithm is currently unable to accomplish either of these tasks.

III. Approach

Our proposed path-planning algorithm takes in four sets of inputs: a 2.5D height map representing the surrounding terrain of the robot, a start and goal location within the map, the specifications of the robot, and the user's preferences of how much to prioritize speed, safety, and energy. Following the inputs, the algorithm generates a PRM from the inputted map to enable faster path-planning. The PRM comprises randomly sampled points within the map with a set amount of edges connecting each point to its K-nearest points. The algorithm then uses the A* path-planning algorithm on the PRM graph to find the optimal path from the start location to the goal location that were inputted.

During the A* search, to calculate the cost of traversing a given edge between two points, four separate costs are calculated and combined. The four costs consist of the speed cost, which represents how much time it would take the robot to traverse an edge; the safety cost, which represents how safe an edge is for the robot; the energy cost, which represents how energy-efficient an edge is for the robot; and the limitation cost, which will add a relatively high cost if the edge is considered impossible for the robot to traverse. The speed, safety, and energy costs are each normalized to range between 0 and $\sim$1 and multiplied by different weights corresponding to the user's inputted preferences. Finally, the normalized costs are added together along with the limitation cost to get the total cost of an edge. The heuristic cost $h(e)$ for a given edge is calculated by finding the Euclidean distance between the edge's end point e and the goal point g and dividing it by the distance between the start point s and g to normalize $h(e)$ between 0 and $\sim$1.

$$h(e) = \frac{\sqrt{(x_g - x_e)^2 + (y_g - y_e)^2}}{\sqrt{(x_g - x_s)^2 + (y_g - y_s)^2}} \tag{2}$$

A. PRM Generation

To generate the PRM, the user can specify the number of points to select and the number of edges each point should have. Once the points are selected, the K-Nearest Neighbors algorithm is used to connect each point to its respective closest neighbors. Typically, the PRM performs best when about 75% of the total points are selected, with between 5 and 7 neighbors per point.

ADD FIGURE

B. Finding the Cells an Edge Travels Through in a Grid

In the PRM generated from the height map, an edge that connects one point to another, where both points lie at the centers of different cells, might actually travel through several cells that are in between the start and end cells, as shown in Figure 1. To accurately calculate the total cost of an edge, we calculate not only what cells on the height map grid the edge travels through, but also the distance that the edge travels within each of those cells. We created a novel algorithm to determine the cells an edge travels through in order from the start cell to the end cell and the respective lengths traveled within each cell.

The algorithm takes a start and end point as input, where the coordinates of each point must be integers (due to every point on the PRM being at the center of a square cell with side length 1 in a uniform-size grid), and outputs a list of tuples that each have three elements. The first two elements are the x and y coordinates of a cell the edge passes through and the third is the Euclidean distance traveled within it.

When the start point (0,0) and end point (3,2) were inputted, the algorithm calculated the cells traveled through and their respective lengths to be the following: [(0, 0, 0.600925), (1, 0, 0.300462), (1, 1, 0.901387), (2, 1, 0.901387), (2, 2, 0.300462), (3, 2, 0.600925)]. These cells match what is expected, as seen in Figure 1, and the sum of their lengths is equal to the Euclidean distance between the start and end points.

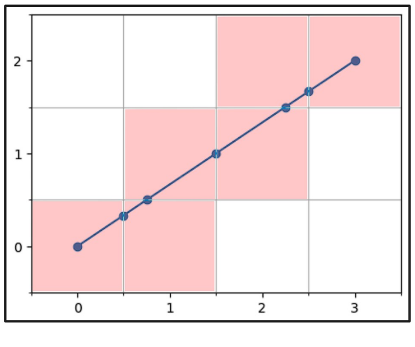

Fig. 1. Example of a line (blue) that starts at cell (0,0) and ends at cell (3,2). The line passes through each red cell and the blue line within each red cell is the portion of the total line that travels through that cell. The blue dots are points on the line that lie on cell borders.

C. Calculating Speed Cost

The speed cost of an edge is exclusively based on the length of the edge due to the assumption that the robot tries to travel at a constant speed at all times. The total length of a given edge is approximated by assigning a right triangle to each cell (besides the last) that the edge goes through, where the base of each triangle is the distance that the edge travels through the cell d_{cell_i} and the height is the height difference between the cell and the cell that immediately follows it $(h_{cell_{i+1}} - h_{cell_i})$. The total length of the edge is calculated by summing the hypotenuses of every triangle in the edge and then adding the distance d_{cell_n} that the edge travels through the last cell to that summation (Figure 2). This approximation is calculated using the following formula:

$$EdgeLen = d_{cell_n} + \sum_{i=1}^{n-1} \sqrt{(h_{cell_{i+1}} - h_{cell_i})^2 + d_{cell_i}^2} \quad (3)$$

where n is the total number of cells that the edge travels through.

To normalize the $SpeedCost$ for an edge to range between 0 and ~1, $EdgeLen$ is divided by the theoretical distance the robot would traverse if it were to cross the length of the longest edge in the PRM $d_{longest}$ at the maximum upwards incline it can climb $maxIncline$.

$$SpeedCost = \frac{EdgeLen}{\frac{d_{longest}}{\cos(maxIncline)}} \quad (4)$$

where $d_{longest}$ is the length of the longest edge in the PRM and $maxIncline$ is the maximum angle (in radians) that the robot can climb.

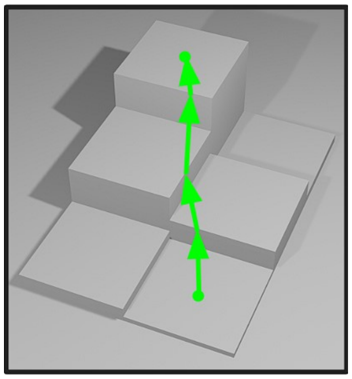

Fig. 2. Visualization of how the $SpeedCost$ of a given edge is measured. The green arrows represent the hypotenuses of the triangles that are in each cell.

D. Calculating Safety Cost

The safety cost for an edge is based on four separate costs that range between 0 and ~1: step safety, turn sharpness, height variance, and normal variance. To calculate the step safety cost $StepSafety$, the cells that an edge travels through are looped through, and the height difference d_h between one cell $cell_1$ and the next $cell_2$ is calculated as $d_h = height_{cell_2} - height_{cell_1}$. If $d_h > 0$, meaning the cell goes from a lower to a higher height, then the variable $maxVal$ is set to the robot's maximum step height up, but if $d_h <= 0$, $maxVal$ is set to the robot's maximum step height down. The $StepSafety$ cost of the step between one cell and the next is exponentially related to how close the step is to the robot's max step-up or down capabilities and is calculated using the following formula:

$$StepSafety = 2^{\frac{|d_h|}{maxVal}} - 1 \quad (5)$$

As it loops through each cell the edge travels through, the algorithm keeps track of the highest calculated $StepSafety$ cost, which represents the most unsafe step in the edge being evaluated.

The turn sharpness cost $TurnSharpness$ is based on how sharp of a turn the robot would have to make to get to the current edge being evaluated from the edge that came before it as seen in Figure 3. The closer a turn is to a full 180°, the more unsafe it is considered. To calculate turn sharpness, the angle between the current edge and the previous edge is calculated using the start point a of the previous edge and the start point b and end point c of the current edge using the following formula:

$$angle = \arccos(\frac{ba \cdot bc}{||ba|| * ||bc||}) \quad (6)$$

where $ba = b - a$ and $bc = c - b$. After $angle$ is calculated and converted to degrees, $\frac{|angle|}{180}$ is used as the $TurnSharpness$ cost, which will range between 0 and 1 depending on how close the turn angle between an edge and the edge that led to it is to a full 180°. If an edge has no previous edges leading to it, the $TurnSharpness$ cost is 0.

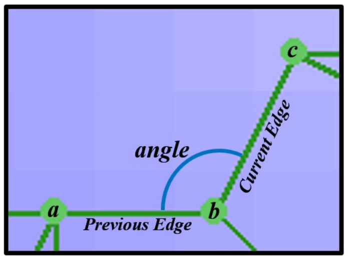

Fig. 3. Example of two edges where the angle between them is calculated using the previous edge's start point a and the current edge's start point b and end point c.

The height variance cost $HeightVar$ is based on the variance of the height of the terrain surrounding the edge being evaluated. The more height variance there is, the less safe it is considered. **EXPLAIN DILATION**

The normal variance cost $NormVar$ is based on the variance of the normal vectors that are associated with each cell that the edge being evaluated travels through. The higher the variance of the cell normals along an edge, the more unsafe it is considered. Obtaining the normal of each cell in the height map is explained later in the section on converting 3D LiDAR data to a 2.5D height map. To measure the variance of the normals the following calculation was used:

$$NormVar = \frac{\Sigma_{i=1}^{n}[(x_i - \overline{x})^2 + (y_i - \overline{y})^2 + (z_i - \overline{z})^2]}{n-1} \quad (7)$$

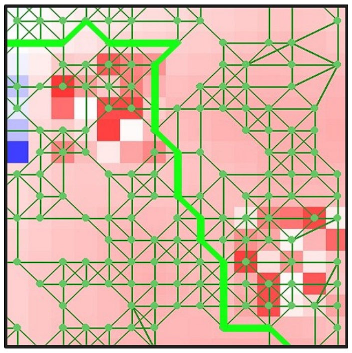

Fig. 4. Example of a generated path (green) avoiding areas with high variance.

where x, y, and z represent the three axes of a normal vector, $\overline{x}$, $\overline{y}$, and $\overline{z}$ represent the mean of all points within a cell on the three axes, and n is the total number of points within a cell. After calculating the four costs, the average between them is calculated to represent the total safety cost of a given edge.

E. Calculating Energy Cost

To calculate the energy cost a simplified model was used. It was assumed that the motor torque would provide 100% of the required power needed to drive the robot up the incline. Using the conservation of energy equation:

$$\Delta KE + \Delta PE = Work \tag{8}$$

where $\Delta KE = 0$ due to the robot moving at a constant speed and $\Delta PE = m * g * (h_2 - h_1)$. Using the equations:

$$Work = \tau * \theta \tag{9}$$

$$\theta = d_{traveled}/r_{wheel} \tag{10}$$

$$d_{traveled} = \sqrt{l_{cell_1}^2 + (h_{cell_2} - h_{cell_1})^2} \tag{11}$$

$$Power = \tau * \omega \tag{12}$$

$$\omega = v_{robot}/r_{wheel} \tag{13}$$

where τ is the applied torque, $d_{traveled}$ is the total distance traveled, r_{wheel} is the wheel radius, the following equation can be derived:

$$InclinePower = \sum_{i=1}^{n_{cells}} \frac{v_{robot} * m_{robot} * g * (h_2 - h_1)}{(\sqrt{l_{cell_1}^2 + (h_{cell_2} - h_{cell_1})^2}} \tag{14}$$

$$EdgePower = cell_{size} * EdgeLen * power_{min} + InclinePower \tag{15}$$

where $power_{min}$ is the power required to traverse a unit length on the map, and $EdgeLen$ is the total distance travelled as calculated in eqn(3).

To normalize this value to be between 0 and 1, $EdgePower$ is divided by the theoretical power needed to traverse the longest edge in the PRM, $d_{longest}$, at the maximum upwards incline it can climb, $maxIncline$. The theoretical energy cost of the maximum edge is the combination of the energy required to traverse the longest distance and the required energy to climb the greatest incline.

$$InclinePower_{max} = v_{robot} * m_{robot} * g * \sin(maxIncline) \tag{16}$$

$$EnergyDen = \frac{d_{longest} * cell_{size} * power_{min}}{\cos(maxIncline)} + InclinePower_{max} \tag{17}$$

$$EnergyCost = \frac{EdgePower}{EnergyDen} \tag{18}$$

F. Calculating Limitation Cost

To ensure that our algorithm never considers a path that has an edge that is impossible for a robot to traverse given its specifications, a limitation measurement is employed. The limitation measurement conducts three tests on a given edge. The first test is iterating through each cell that the edge travels through and checking if the step-up or down required to get from one cell to the next exceeds the robot's max step-up or down capabilities. The second test is iterating through each cell that the edge travels through and checking if the incline up to get from one cell to the next exceeds the max incline up that the robot is able to traverse. The following calculation is used to calculate the incline up between two cells:

$$incline = \arctan(\frac{height_{cell_2} - height_{cell_1}}{cellSize * \sqrt{2}}) \tag{19}$$

where $cellSize$ is the length, in meters, of the diagonal of a cell on the height map. The third test is to check the clearance of the edge. The clearance is based on if the robot is able to traverse the edge without touching any cells that are considered obstacles that may not be directly along the edge but are less than the robot's width away from the edge where the robot might bump into as seen in Figure 5. To check clearance, we used dilation. **EXPLAIN DILATION**

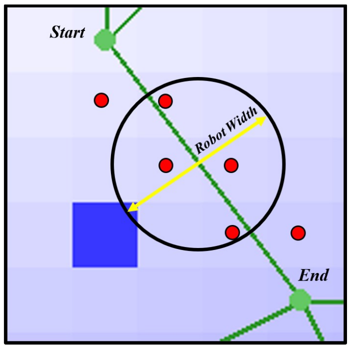

Fig. 5. Example scenario where robot traversing an edge (green line) connecting $node_{start}$ to $node_{end}$ would think that it is traversable if it only considers the cells that the edge travels through (cells with red circles at center). However, due to the width of the robot (yellow line), it will collide with a cell that is considered an obstacle (dark blue square). Therefore, the heights of cells surrounding an edge (in addition to the cells that it travels through) must be checked to ensure the robot can traverse an edge.

If an edge fails any of these three tests, the edge is considered impossible to traverse and a $LimitationCost$ of 10,000 is returned and added to the combined speed, safety, and energy costs (each ranging between 0 and $\sim$1). This ensures that the edge is never considered in the path-planning process. If the edge passes all three tests, a $LimitationCost$ of 0 is returned.

G. Converting 3D LiDAR Data to Height Map

A function was created to convert point cloud data collected from a 3D LiDAR to a 2.5D height map that can be used as input to our path-planning algorithm. The function takes a point cloud data (PCD) file comprised of (x, y, z) coordinates as input. Each point's normal is then calculated based on all surrounding points within a set radius. Then, each point, based on its x and y coordinates, is assigned to a cell that would theoretically be in a 2D grid that lies parallel to the ground. After each point is assigned to a cell, we loop through each cell that has at least one point and calculate the maximum, minimum, mean, and variance of the heights (z coordinates) of the points within the cell. Additionally, we calculate the mean of the points within each cell as well as the variance of the normals of those points using Equation 7.

Because current 3D LiDARs can only scan at a limited amount of vertical angles, there are usually gaps in the point cloud where flat ground should be, as seen in the left of Figure 6 between the dark blue circles around the center. Consequently, there would be cells within the grid that don't have any points and the grid would be incomplete. To obtain a full grid that approximates the terrain, we consider only the cells on the grid that would be within a set square boundary surrounding the center of the point cloud. We loop through each of the cells within that grid, and if a cell has no points, we use the algorithm described in Section IIB to find what cells would be between the empty cell and the LiDAR at the center of the grid. If there are any cells that are considered obstacles (based on the max height of the cell) between the empty cell and the center, the empty cell is given a max height that classifies it as an obstacle as well. If there are no such obstacle cells between the empty cell and the center, it is given a max height that is close to flat ground. Using this method, we were able to convert point cloud data collected from a VLP-16 LiDAR to a 2.5D height map we could use as input to our path-planning algorithm, as shown in Figure 6, where the height and normals of each cell are based on the max height and average normal of each cell calculated from the point cloud data.

IV. Experiments and Results

To ensure the effectiveness of the algorithm, three verification methods were used:

1) An increase in preference for speed, safety, or energy resulted in a reduced cost
2) The algorithm performed comparably to a crow's flight A* algorithm
3) In a known environment with three distinct paths, the algorithm will follow expectations

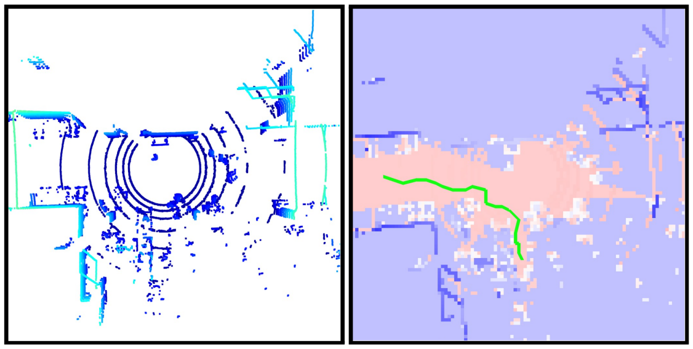

Fig. 6. Top-down view of point cloud data collected from VLP-16 LiDAR where darker is lower (left). Resulting 2.5D height map (red = lower, blue = higher) after conversion with a path (green) generated from the bottom of the map to left (right).

V. Conclusions and Future Work

Acknowledgments

We would like to thank Professor Carl Sable, Professor Michelle Rosen, and Michael Giglia for their support and guidance on this project.

References

[1] Lin, S.; Liu, A.; Wang, J.; Kong, X. Path-Planning Approaches for Multiple Mobile Robots. Encyclopedia. Available online: https://encyclopedia.pub/entry/27164 (accessed on 05 April 2023).

[2] J. Carsten, A. Rankin, D. Ferguson, A. Stentz (2007). "Global Path Planning on Board the Mars Exploration Rovers," 2007 IEEE Aerospace Conference, doi: 10.1109/AERO.2007.352683

[3] "Open Motion Planning Library: A Primer," 31-Jan-2021. Available: https://ompl.kavrakilab.org/OMPL_Primer.pdf

[4] Hart, P. E., Nilsson, N. J., Raphael, B. (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," IEEE Transactions on Systems Science and Cybernetics. 4 (2): 100–107. doi:10.1109/TSSC.1968.300136.

[5] Stepan Dergachev,Kirill Muravyev and Konstantin Yakovlev (2022) "2.5D Mapping, Pathfinding and Path Following For Navigation Of A Differential Drive Robot In Uneven Terrain," 2002, doi: 10.48550

[6] P. Fankhauser, M. Bloesch and M. Hutter, "Probabilistic Terrain Mapping for Mobile Robots With Uncertain Localization," in IEEE Robotics and Automation Letters, vol. 3, no. 4, pp. 3019-3026, Oct. 2018, doi: 10.1109/LRA.2018.2849506.

[7] T. Greicius, "NASA's self-driving perseverance Mars Rover 'takes the wheel'," NASA, 01-Jul-2021. [Online]. Available: https://www.nasa.gov/feature/jpl/nasa-s-self-driving-perseverance-mars-rover-takes-the-wheel/.

[8] G. Mills-Tettey, Anthony Stentz, M. Dias (2008). "Continuous-field path planning with constrained path-dependent state variables," Carnegie Mellon University