PB*: 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

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

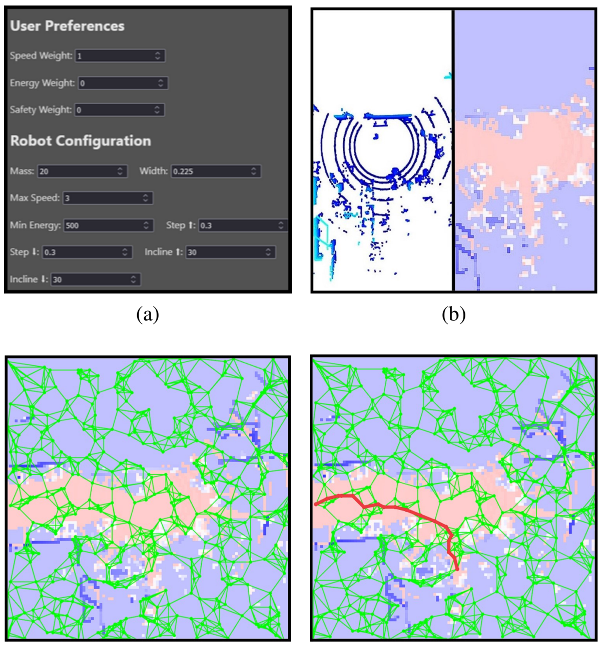
***Abstract*—Efficient travel for an autonomous robot requires a path-planning algorithm to determine the optimal path to take based on certain criteria. The most common criteria are speed, safety, and energy, as most autonomous robots are designed to operate in remote and/or difficult terrain where 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, called PB*, 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. After running various tests, we demonstrated that PB* can provide accurate paths that respond appropriately to user inputs within a reasonable run-time.**

***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] [2]. 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 path criteria for autonomous robots are speed, safety, and energy [1] [2]. Speed relates to how much time it takes for a robot to reach a destination, safety relates to how risk-averse or traversable a path is, and energy relates to the energy efficiency of the path.

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] [2]. 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 paths [3]. 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 [1] [2]. 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. For example, if a robot that 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


User Preferences
Speed Weight: 1
Energy Weight: 0
Safety Weight: 0
Robot Configuration
Mass: 20 Width: 0.225
Max Speed: 3
Min Energy: 500 Step ↑: 0.3
Step ↓: 0.3 Incline ↑: 30
Incline ↓: 30

(a) (b) (c) (d)

Fig. 1: Four steps of PB*: (a) User input collection. (b) Map generation (blue cells are higher, red cells are lower). (c) PRM generation. (d) Path-planning.

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, called PB*, that addresses this issue. A user inputs various specifications of a robot, such as its mass, dimensions, energy usage, and climbing abilities (Figure 1a). A function then converts point cloud data collected by a 3D LiDAR (also provided by the user) and converts it to a 2.5D height map, which is a 2D grid that represents a 3D surface or terrain by mapping the height and normals information to the surface to the grid (Figure 1b). 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 (Figure 1c). The path-planning algorithm A* is then applied to calculate the optimal path from one point on the PRM to another (Figure 1d), where the cost of each edge between two points is calculated based on the data from the underlying height 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. Finally, a check is done to determine if the edge is impossible for the robot to traverse, and if it is, a relatively high cost, denoted as the limitation cost, is added to the edge to ensure it is heavily avoided in the path-planning process.

To demonstrate PB*, 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 PB* to calculate the optimal path from the start point to the goal point and displays it on the height map for the user to see. Using the same height 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. PB* was also tested on terrain generated by readings from a physical 3D LiDAR to be evaluated within real-world environments.

II. Approach

A. User Input Collection

For PB* to return an optimal path, various inputs must be collected from the user. The most important inputs are the user's criteria preferences, which are weights for how important speed, safety, and energy are relative to each other. Each preference is converted to be a percentage of the total three weights. In addition to the criteria preferences, the user must input the configuration of their robot. Every robot specification input is listed in Table I.

The inputs specified in Table I are important for ensuring that the generated path is optimal for the user's robot, and

Input	**Explanation**
Mass	The mass of the robot
Width	The width of the robot
Minimum Energy Usage	The energy usage of the robot while traversing on flat ground
Step Height Up	The maximum height the robot can step up
Step Height Down	The maximum height the robot can step down
Incline Up	The maximum upwards incline the robot can traverse
Incline Down	The maximum downwards incline the robot can traverse

TABLE I: Explanations of user inputs for robot specifications.

will be used during the cost calculation process of PB*. The user also has the ability to input their own point cloud data or randomly generate a height map.

B. Map Generation

PB* requires a 2.5D height map as an input, which is a 2D grid that represents a 3D terrain where each grid cell is assigned a height that best represents the portion of the terrain within it. Using 2.5D height maps is an intuitive method for mapping life-like terrain to a discrete space and is commonly used for path-planning for autonomous robots [4] [5].

It is assumed that the data representing the surrounding terrain of a user's robot is collected using a 3D LiDAR, so the option to convert point cloud data to a 2.5D height map is provided to the user. We created a function to perform this conversion which 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 using a built-in function of Open3D, which is an open-source 3D data processing library [6]. 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, the function loops through each cell that has at least one point and calculates the maximum, minimum, mean, and variance of the heights (z coordinates) of the points within the cell. Additionally, the mean and variance of the normals of the points within each cell are calculated. To measure the variance of the normals, $NormVar$, the following calculation is used:

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

where x, y, and z represent the three axes of a normal vector, $\overline{x}$, $\overline{y}$, and $\overline{z}$ represent the mean on the three axes of all points within a cell, and n is the total number of points within a cell.

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 would be, as shown in Figure 2a between the dark blue circles around the center. Consequently, there would be cells within the height map that don't have any points and the height map would be incomplete. To obtain a full height map that approximates the terrain, only cells within a set square boundary surrounding the center of the point cloud are considered. The function then loops through each cell within that square boundary, and if a cell has no points, the algorithm described in Section IIE is used to find which cells would be between the empty cell and the center. If there are any cells that are considered obstacles (based on the maximum height of the cell) between the empty cell and the center, the empty cell is given a maximum 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 maximum height that is close to flat ground. Using this method, point cloud data collected from a VLP-16 LiDAR was converted to a 2.5D height map that could be used as an input to PB*, as shown in Figure 2b.

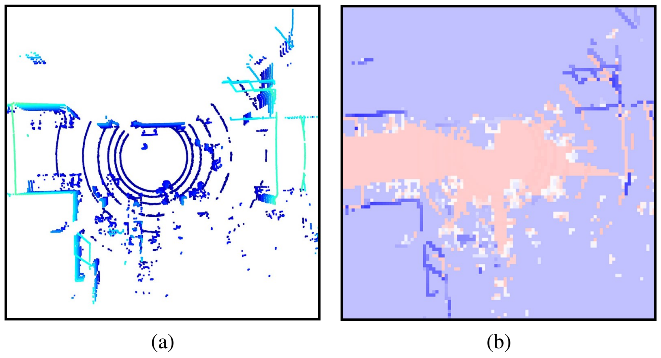


Fig. 2: (a) Top-down view of point cloud data collected from VLP-16 LiDAR (darker colors are lower). (b) 2.5D height map after conversion (blue cells are higher, red cells are lower).

C. Probabilistic Road Map

Many path-planning algorithms rely on an underlying graph that represents the possibilities for movement [1]. 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 a path comprised of an ordered sequence of nodes between the start position and the goal position. A simple graph that can be used is a grid, which has the advantage of accurately representing an underlying space. However, 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 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 [7]. The randomly selected points in a PRM can be connected in a variety of ways; the most commonly used method to connect them, which is what is used in PB*, is the K-Nearest-Neighbors algorithm, which is an algorithm that connects each point to K of the closest points in the space, where K is a predetermined positive integer [7]. After conducting experiments that compared a full grid to a graph generated by a PRM using 50% of the points on a 64×64 height map, it was determined that, on average, paths on a PRM were generated 14× faster than paths on a full grid and were only 1.5× as costly.

In our UI, the user can specify the number of points in the PRM and the number of edges per point. The PRM is superimposed on top of the height map so that paths can be generated from one point on the map to another, as shown in Figure 1c.

D. Path-Planning

To find a path on the PRM, the A* search algorithm is employed, which is an algorithm for finding the least costly path between two nodes in a graph [8]. We decided to use A* due to its simplicity, computational efficiency, and wide use in autonomous robotics, computer games, and environmental mapping [1]. A* utilizes a heuristic to estimate the distance between a given node and the destination node, enabling the algorithm to find the shortest path more efficiently.

The algorithm evaluates nodes in a priority queue containing all previously examined nodes, and at each iteration, it selects the node with the lowest total estimated cost to continue exploring. The cost is calculated using the function:

$$f(n) = g(n) + h(n) \tag{2}$$

where n is the current 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. As long as $h(n)$ is admissible, meaning it never overestimates the minimum cost to the goal, A* is guaranteed to find the optimal path without evaluating any point more than once.

During the A* search, to calculate the cost of traversing a given edge between two points, separate costs for speed, safety, energy, and limitation are calculated. The speed, safety, and energy costs are each normalized to range between 0 and ∼1, while the limitation cost is a relatively high cost (10,000) if the edge is considered impossible for the robot to traverse, and zero if it is possible. The total cost for an edge, $g(n)$, is the linear combination of the four costs where the coefficients of the speed, safety, and energy costs are the weights determined by the user preferences.

$$\begin{aligned} g(n) = w_{speed} \cdot g_{speed}(n) + w_{energy} \cdot g_{energy}(n) \\ + w_{safety} \cdot g_{safety}(n) + g_{limitation}(n) \end{aligned} \tag{3}$$

At each node, separate heuristic costs are calculated for speed, safety, and energy that each range between 0 and 1. Similar to Equation 3, the total heuristic cost for an edge, $h(n)$, is the linear combination of the three heuristics where the coefficients are the weights determined by the user preferences.

$$\begin{aligned} h(n) = w_{speed} \cdot h_{speed}(n) + w_{energy} \cdot h_{energy}(n) \\ + w_{safety} \cdot h_{safety}(n) \end{aligned} \tag{4}$$

E. *Finding the Cells an Edge Travels Through in a Grid*

Before being able to calculate the cost for an edge, the cells on the underlying map that an edge passes through must be determined. 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 3. To accurately calculate the total cost of an edge, the cells that the edge travels through, as well as the length traveled within each cell, must be determined. We created an 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 cells and the distance the edge travels through it.

When the start point (0,0) and end point (3,2) were chosen, the algorithm returned the following list, where each element the coordinates of a cell and the distance traveled through it: [(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 shown in Figure 3, and the sum of their lengths is equal to the Euclidean distance between the start and end points.

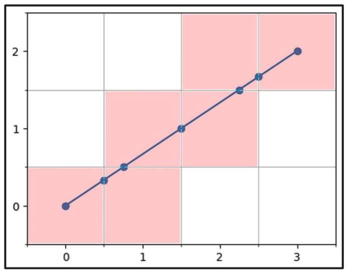


Fig. 3: Example of an edge (blue) that starts at cell (0,0) and ends at cell (3,2). The edge passes through each red cell and the blue line within each red cell is the portion of the edge that travels through that cell. The blue dots are points on the edge that lie on cell borders.

F. *Calculating Speed Cost*

The speed cost of an edge, $g_{speed}(n)$, is exclusively based on the length of the edge, due to the assumption that the robot travels at a constant speed. The total length of an edge is approximated by assigning a right triangle to each cell that the edge travels 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 in the last cell. This approximation is calculated using the following formula:

$$EdgeLen = d_{cell_n} + \sum_{i=1}^{n-1} \sqrt{(h_{cell_{i+1}} - h_{cell_i})^2 + d_{cell_i}^2} \quad (5)$$

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

To normalize $g_{speed}(n)$ 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, $incline_{max}$.

$$g_{speed}(n) = \frac{EdgeLen}{\frac{d_{longest}}{\cos(incline_{max})}} \quad (6)$$

To obtain the heuristic cost for speed, $h_{speed}(n)$, the Euclidean distance from the edge's end point, n, to the goal point, g, is measured and divided by the length of the diagonal of the height map, $l_{diagonal}$, in order to normalize it between 0 and 1. Because the path from an edge to the goal with the minimum speed cost is a straight line, this calculation never overestimates the minimum cost to get from the node being evaluated to the goal, which makes it an admissible heuristic.

$$h_{speed}(n) = \frac{\sqrt{(x_g - x_n)^2 + (y_g - y_n)^2}}{l_{diagonal}} \quad (7)$$

G. *Calculating Safety Cost*

The safety cost, $g_{safety}(n)$, for an edge is based on the average of four separate costs that range between 0 and ~1: step safety, turn sharpness, height variance, and normal variance.

$$g_{safety}(n) = (StepSafety + TurnSharpness + HeightVar + NormVar)/4 \quad (8)$$

To calculate $StepSafety$, 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 $d_h = h_{cell_2} - h_{cell_1}$. If $d_h > 0$, meaning the cell goes from a lower to a higher height, then the maximum step height, h_{max}, is set to the robot's maximum step height up, but if $d_h <= 0$, h_{max} is set to the robot's maximum step height down. The $StepSafety$ cost of the step between one cell and the next increases exponentially as the step gets closer to the robot's maximum step height up or down and is calculated using the following formula:

$$StepSafety = 2^{\frac{|d_h|}{h_{max}}} - 1 \quad (9)$$

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

$TurnSharpness$ is based on the turn angle the robot would have to make to get to the current edge being evaluated from the edge that came before it, as shown in Figure 4a. The closer a turn is to a full 180°, the more unsafe it is considered. To calculate the turn angle, $TurnAngle$, the angle between the current edge and the previous edge is calculated using the start point, a, and end point, b, of the previous edge, and the end point, c, of the current edge, using the following formula:

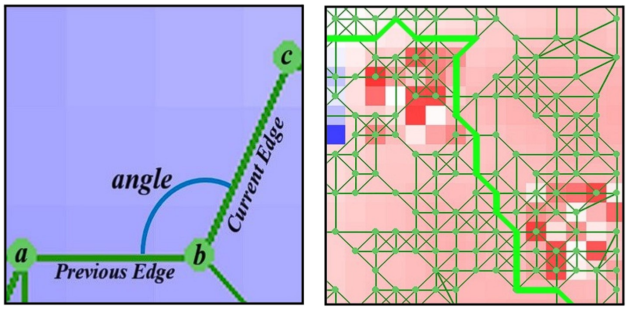
$$TurnAngle = \arccos(\frac{[b-a]\cdot[c-b]}{||[b-a]|| * ||[c-b]||}) \tag{10}$$

After $TurnAngle$ is calculated and converted to degrees, it is divided by 180 to normalize it between 0 and 1. If an edge has no previous edges leading to it, the $TurnSharpness$ cost is 0.

$$TurnSharpness = \frac{|TurnAngle|}{180^\circ} \tag{11}$$

$HeightVar$ is based on the variance of the heights of the cells surrounding the edge being evaluated. The more height variance there is, the less safe it is considered and should be avoided as shown in Figure 4b, as high variance areas represent uneven terrain. The width of the robot is converted to the number of cells it spans, divided by two, and rounded to the nearest integer which is denoted as R. For each cell, $cell_i$, that the edge travels through, the cells that lie within the circle with a radius of R that surrounds $cell_i$ are found. The mean of the heights of those cells is then calculated and used to calculate the variance of the heights of the cells that surround the edge.

$NormVar$ 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. A lower normal variance represents terrain, such as a ramp or flat ground, while a higher variance can represent stairs or uneven terrain. Obtaining the normal of each cell in the height map is explained in section IIA. To measure the variance of the normals, $NormVar$, Equation 1 is used.



(a) (b)

Fig. 4: (a) Example of two edges where the angle between them is calculated using the previous edge's start point, a, the end point, b, and the current edge's end point, c. (b) Example of a generated path avoiding areas with high height variance.

The heuristic cost for safety, $h_{safety}(n)$, is always equal to 0. The reason for this is that the minimum safety cost of a path can be zero even if the length of the path is more than zero. Therefore, $h_{safety}(n)$ must be equal to 0 for it to be considered admissible.

H. 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 an incline, and the robot would be traveling at a constant speed at all times. Using the conservation of energy equation:

$$\Delta KE + \Delta PE = Work \tag{12}$$

where $\Delta KE = 0$ due to the robot moving at a constant speed and $\Delta PE = m * g * \Delta h$ and using the equations:

$$Work = \tau * \theta \tag{13}$$

$$\theta = d_{traveled}/r_{wheel} \tag{14}$$

$$d_{traveled} = \sum_{i=1}^{n-1} \sqrt{(h_{cell_{i+1}} - h_{cell_i})^2 + d_{cell_i}^2} \tag{15}$$

$$Power = \tau * \omega \tag{16}$$

$$\omega = v_{robot}/r_{wheel} \tag{17}$$

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 for incline power, IP:

$$IP = \sum_{i=1}^{n_{cells}} \frac{v_{robot} * m_{robot} * g * (h_{cell_{i+1}} - h_{cell_i})}{(\sqrt{(h_{cell_{i+1}} - h_{cell_i})^2 + d_{cell_i}^2}} \tag{18}$$

In addition to using energy to overcome a height change, baseline energy must be expended to travel any flat distance. The energy consumption per distance traveled is highly dependent on the specifications of the robot and is therefore left as a user input. The total power required to traverse an edge is the sum of the required energy to traverse the total distance of the edge and the energy required to overcome the potential energy change due to height difference along the edge. The total required edge power is given by

$$EdgePower = cell_{size} * EdgeLen * power_{min} + IP \tag{19}$$

where $cell_{size}$ is the length, in meters, of the width of a cell in the height map, where $power_{min}$ is the power required to traverse a unit length on the height map, and $EdgeLen$ is the total distance traveled as calculated in equation(5).

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 angle the robot can climb, $incline_{max}$, represented by the term ED. Equation 22 is used to calculate the normalized energy cost of an edge.

$$IP_{max} = v_{robot} * m_{robot} * g * \sin(incline_{max}) \tag{20}$$

$$ED = \frac{d_{longest} * cell_{size} * power_{min}}{\cos(incline_{max})} + IP_{max} \tag{21}$$

$$g_{energy}(n) = \frac{EdgePower}{ED} \tag{22}$$

The heuristic cost for energy, $h_{energy}(n)$, is the same as $h_{speed}(n)$ in Equation 7 because the path from an edge to the goal with the minimum energy cost is also a straight line.

I. *Calculating Limitation Cost*

To ensure that the generated path heavily avoids edges that are impossible for the user's robot to traverse, 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 height up or down required to get from one cell to the next exceeds the robot's maximum step height up or down. If so, the edge fails the first test.

The second test is iterating through each cell that the edge travels through and checking if the upwards incline to get from one cell to the next exceeds the maximum upwards incline that the robot can traverse. If so, the edge fails the second test. The following calculation is used to calculate the upwards incline between two cells.

$$UpIncline = \arctan(\frac{h_{cell_2} - h_{cell_1}}{d_{cell_1}}) \tag{23}$$

The third test is to check the clearance of the edge. The clearance test checks whether or not the robot is able to traverse the edge without touching any cells that are considered obstacles. This is based on the robot's width, and checking if the robot might bump into a cell that is not directly along the edge. In Figure 5, a robot traversing an edge (green line) connecting $node_{start}$ to $node_{end}$ would assume 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. The same surrounding cells checked during the $HeightVar$ calculation in Section IIG are used for the clearance test. If the height difference between a cell the edge travels through and a cell that surrounds it is greater than the robot's maximum step height up or down, the cell is classified as an obstacle, and the edge fails the clearance test.

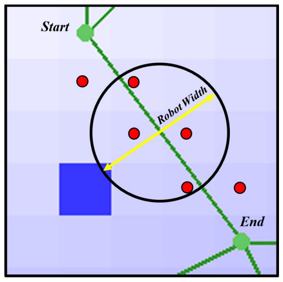


Fig. 5: Example scenario where a robot would experience a collision with a cell (dark blue) that the edge doesn't travel through.

If an edge fails any of these three tests, the edge is considered impossible to traverse, and $g_{limitation}(n) = 10,000$. This ensures that the edge is heavily avoided in the path-planning process since 10,000 is a very high value relative to the linear combination of the speed, safety, and energy costs. If the edge passes all three tests, $g_{limitation} = 0$.

III. Experiments and Results

To ensure the effectiveness of PB*, three statements need to be confirmed:

1) An increase in preference for speed, safety, or energy results in a reduced cost for the respective preference
2) PB* performs comparably to a basic A* algorithm in computation time
3) In a known environment with distinct paths for speed, safety, or energy, the generated paths will follow expectations

A. *User Preference Tests*

One expected behavior of PB* is that the cost of the calculated path should reflect the user's preferences. Therefore, as the user increasingly prioritizes a preference, the resulting preference cost for the generated path should decrease. To demonstrate this, PB* was tested in a variety of randomized scenarios, and the costs and preferences were recorded. In total, 100 trials were run on 10 random height maps, with the preference weights set to random values between 1 and 100.

For each map, the path costs for each preference were normalized to values between 0 and 1, where 0 is equivalent to the lowest path cost on that map, and 1 is equivalent to the highest path cost on that map. This was done using the following equation:

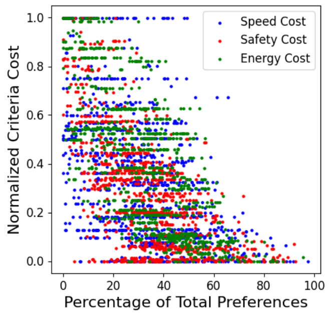
$$NormCost_{pref} = \frac{C_{pref} - \min(C_{pref})}{\max(C_{pref}) - \min(C_{pref})} \tag{24}$$

where $NormCost_{pref}$ is the normalized cost for a given preference, C_{pref} is the total preference cost for a path, and $max(C_{pref})$ and $min(C_{pref})$ represent the maximum and minimum preference cost on a given map for a path. As expected, as a preference becomes more important to the user, the corresponding cost for that preference decreases.

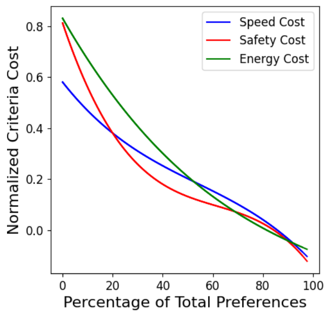
In Figure 6a, each dot represents the cost for an individual path, where the blue points represent the speed costs, the green points represent the energy costs, and the red points represent the safety costs. As the user increases their preference for speed, safety, or energy, the respective criterion cost of the generated path goes down, as shown by the trend lines in Figure 6b. This demonstrates that the user's preference for speed, energy, and safety actually affects the generated path and its respective costs.

B. *Runtime Tests*

PB*'s computation time was compared to basic A* on increasing height map sizes. Basic A* only accounts for the 2D distance between nodes and ignores the underlying height map. For this test, 50 trials were run on 8 random height maps, and the computation times to generate paths were recorded. For each trial, five different paths were generated: one using basic



(a)

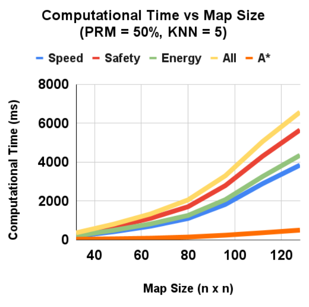


(b)

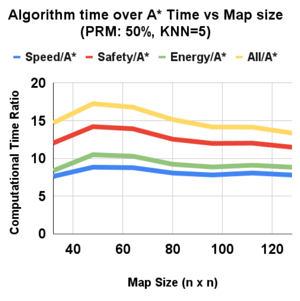
Fig. 6: (a) Individual criterion costs for paths generated with varying user preferences. (b) Trend lines for each criterion cost for generated paths as the user preferences increase.

A*, and four using PB*. The first path only prioritized speed, the second only prioritized safety, the third only prioritized energy, and the fourth prioritized them all equally. As the map size increased, so did the computation time, as shown in Figure 7a. In all tests, basic A* performed the fastest, followed by the path prioritizing speed, then energy, then safety, and then the path that prioritized all three.

Even as the map size increased, the ratios of PB*'s computation times compared to basic A* stayed relatively constant, shown in Figure 7b. On average, with all preferences considered during path generation, PB* took ∼15× as long to run. Out of the three criteria, the safety cost took by far the longest to compute, taking 12.61× as long as basic A*. Comparatively, calculating the speed and energy costs took 8.15× and 9.32× as long respectively. Although PB* is slower than basic A*, PB*'s time to generate a path is minuscule compared to the time it would take for a robot to traverse the full length of the path, which is assumed to be satisfactory for most autonomous robots.



(a)

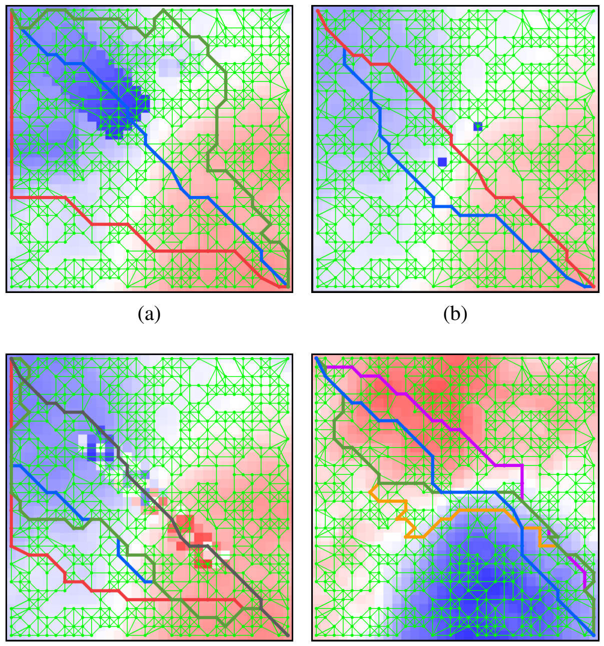


(b)

Fig. 7: (a) Computation Time to generate a path with different preferences as the height map size increases. (b) PB* computation time divided by basic A* computation time as the height map size increases.

C. Manual Tests

In addition to testing the computation time and the user preferences, PB* was tested on a number of handmade height maps where the correct paths were obvious, in order to ensure that it is making the correct decisions.



(a) (b) (c) (d)

Fig. 8: Tests run on four handmade maps, with the start at the top left and the goal at the bottom right. (a) Map for testing user preferences. (b) Map for testing clearance. (c) Map for testing against basic A*. (d) Map for testing robots of various masses.

For the first test, a height map was handmade with three obvious opportunities in mind; the map in Figure 8a has a fast but mountainous path, a downhill but dangerous path, and an uphill but safe and long path. The robot was configured so that it could climb large inclines and traverse off of large drops. As shown in figure 8a, each path (with speed in blue, safety in red, and energy in green) took its respective opportunity.

Another map was tested in order to verify that the robot configuration affects the resulting path. In Figure 8b, there are two tall obstacles near the center of the map. Two different robots were tested, both optimized for speed; the first robot was too wide to fit between the obstacles, while the second robot was narrower, and could fit between the obstacles.

Another notable test was to show that PB* is equipped to make better decisions than a simple, basic A* algorithm. The height map in Figure 8c is relatively smooth, with the exception of a mountainous area directly between the start and goal. The speed, safety, and energy paths (blue, red, and green paths) avoid the high variance area and take relatively similar paths around it. A simple, basic A* path (gray) cuts right through the middle, as it is the quickest way to traverse from the start node to the end node. While the basic A* appears to be shorter, when taking into account changes in elevation, the speed path generated using PB* is actually quicker to traverse. The basic A* would take this particular robot $\sim$62 seconds to traverse, while PB*'s path would take $\sim$51 seconds to traverse.

Finally, to test that the energy preference works, three different robots were tested on the same height map in Figure 8d. The first robot had a mass of 2 and took a direct route (purple). The second robot had a mass of 200 and avoided any uphill climbs, taking a longer path instead (orange). The third robot had a mass of 20 and followed a balanced path that involved some climbing and some directness (green). These tests demonstrate that as the robot's mass increases, it becomes increasingly likely to avoid uphill travel because it requires more energy to ascend for heavier robots. It's worth noting that the speed path (blue) still differs from any energy path.

IV. Conclusions and Future Work

Based on the presented work, we conclude PB* can produce optimal paths based on dynamic user preferences and various robot configurations within a sufficient time frame. PB* can be useful for users that have robots in dynamic environments or a variety of robots with different specifications.

There are many areas in which to improve the method of path-planning used in PB*. Most importantly, PB* makes a number of assumptions; for example, it assumes the robot traverses at a constant speed, allowing the energy equation to be simply the potential energy equation. Ruling out assumptions would result in more complex calculations and would require more user inputs, but would result in a more accurate path.

Another useful feature that could be implemented is constraint-based optimization; for example, the user could specify to take the safest path that takes a maximum of 30 seconds to traverse. Additionally, PB* could be applied to more than just autonomous, ground-traversing robots, as the cost calculations could be generalized for 3D space and used for autonomous robots in the air, underwater, or in space.

Finally, the code could be further optimized to improve computation time and accuracy. Other path-planning algorithms, such as Theta* [9] or Field D* [10] could be used in place of A*, as they generate paths that are more efficient for practical robots and are commonly used in the field of autonomous robotics [1] [2] [3]. PB* could also be packaged into a developer kit, allowing any user to easily implement it on their robot. While PB* already allows for many configurations, it could easily be expanded upon and generalized further.

A simple interactive demonstration of PB* can be viewed at pathplanning.online.

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] K. Karur, N. Sharma, C. Dharmatti, and J. E. Siegel, "A Survey of Path Planning Algorithms for Mobile Robots," Vehicles, vol. 3, no. 3, pp. 448–468, Aug. 2021, doi: 10.3390/vehicles3030027.

[2] S. Lin, A. Liu, J. Wang, and X. Kong, "Path-Planning Approaches for Multiple Mobile Robots," Machines, vol. 10, no. 773, Sep. 2022. Available: https://doi.org/10.3390/machines10090773Encyclopedia.

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

[4] S. Dergachev, K. Muravyev, and K. Yakovlev, "2.5D Mapping, Pathfinding and Path Following For Navigation Of A Differential Drive Robot In Uneven Terrain," Sep. 2022. Available: https://doi.org/10.48550/arXiv.2209.07252

[5] P. Fankhauser, M. Bloesch, and M. Hutter, "Probabilistic Terrain Mapping for Mobile Robots With Uncertain Localization," IEEE Robotics and Automation Letters, vol. 3, no. 4, pp. 3019-3026, Jun. 208, doi: 10.1109/LRA.2018.2849506.

[6] Q. Zhou, J. Park and V. Koltun "Open3D: A Modern Library for 3D Data Processing," Jan. 2018, arXiv:1801.09847. Available: http://www.open3d.org/docs/release/index.html

[7] "Open Motion Planning Library: A Primer," Kavraki Lab, Rice University, Jan. 2021. Available: https://ompl.kavrakilab.org/OMPL_Primer.pdf

[8] P. Hart, N. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," IEEE Transactions on Systems Science and Cybernetics. vol. 4, no. 2, pp. 100–107, Jul. 1968, doi:10.1109/TSSC.1968.300136.

[9] K. Daniel, A. Nash, S. Koenig, and A. Felner, "Theta*: Any-Angle Path Planning on Grids," Journal of Artificial Intelligence Research, vol. 39, pp. 533–579, Oct. 2010, doi: 10.48550/arXiv.1401.3843.

[10] D. Ferguson, A. Stentz, "Using Interpolation to Improve Path Planning: The Field D* Algorithm," Journal of Field Robotics, vol. 23, no. 2, Feb. 2006, pp. 79-101, doi: 10.1002/rob.20109.