Mathematical Investigation

Path Planning Algorithms

Internal Assessment

Mathematics

International Baccalaureate Programme

Word Count: 3000

2021-2023

Table of Contents

Introduction 3

Rationale 3

Aim 3

2d Path Planning 4

Basic Of Path Planning 4

Dijkstra Algorithm 5

Star searches 7

D* Lite Algorithms 9

3d Path Planning 16

D* Lite algorithm on n-dimensional planes 16

Final thoughts 22

Evaluations and Extensions 22

Conclusion 22

Bibliography 23

Introduction

The development of a sustainable world in which we can live in harmony with the environment will undoubtedly require a complete re-invention of the mobility sector[1]. Developing new technologies that will allow us to move between two points as quickly as possible will likely require the use of free air above us[2].

We have already seen many examples of autonomous ground vehicles[3] in recent years, but I believe that aerial mobility will take a new turn when autonomous vehicles can move people or objects in the sky[4].

Rationale

I've always had a strong interest in engineering, cars, and anything that could move. I will never forget the times, sitting on the sofa with my grandfather watching each Formula 1[5] race, getting exited about a victory or sad for a missed pit stop.

I am also aware of the importance of addressing mobility issues, which, particularly in my country, are the source of high levels of pollution, an increased number of road deaths, and massive logistics delays. Being able to study technologies that contribute to the development of a better future truly excites me; I'm confident that this paper is only the beginning of a long journey in my learning process of innovative technologies that may revolutionise the world.

Aim

The aim of this paper is to investigate the algorithms used by autonomous vehicles (Drones, UAVs, underwater submarines …) to move in a three dimensional space. Because I am

---

[1] "The Future of Transport: Driving Change in the next 10 Years." *National Grid Group*, https://www.nationalgrid.com/stories/journey-to-net-zero-stories/future-transport-driving-change-next-10-years.

[2] "The Future of Air Mobility: Electric Aircraft and Flying Taxis | the next Normal | McKinsey & Company." *Www.mckinsey.com*, www.mckinsey.com/featured-insights/the-next-normal/air-taxis.

[3] "The Future of Transport: Driving Change in the next 10 Years." *National Grid Group*, https://www.nationalgrid.com/stories/journey-to-net-zero-stories/future-transport-driving-change-next-10-years.

[4] Neon. "The Future of Aerial Transport." *The Atlantic*, 1 Jan. 1928, www.theatlantic.com/magazine/archive/1928/01/the-future-of-aerial-transport/306535/. Accessed 24 Oct. 2022.

[5] "The Official Home of Formula 1® | F1.com." *Formula 1® - the Official F1® Website*, www.formula1.com.

interested in determining how to increase mobility in my country, I have specifically chosen to investigate drones for the purposes of this paper. We'll examine how path planning and decision-making systems are shaping up to be crucial components of the future[6].

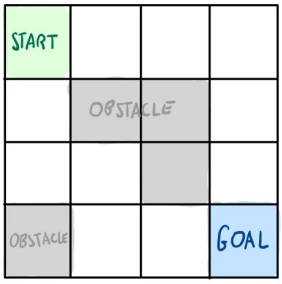
The goal is to reach conclusions using critical reasoning in order to fully understand the deep functioning of these algorithms and especially how they were ideated.

2d Path Planning

Basic Of Path Planning

Before getting to understand the movement of vehicles on a 3d plane we need to master the concept of 2d path planning. Autonomous robots operate on the premise of mapping the environment and representing it as a grid; Figure 1 depicts a simple map of a 2D environment. We can see that the map has a starting point (the robot's current location) and a goal location.

Figure 1



Author: hand drawn by me

It also contains important information on nearby obstacles (gray blocks), which are required to develop an algorithm that will provide the shortest path. An important point to note is that our drone could either have direct access to the map (as provided by an external source) or map it himself, using sensors and lasers to detect the area of free space[7]. We should also consider how environmental changes may affect our algorithms: in real life, there will be cars

[6] "Path Planning." *Www.mathworks.com*, www.mathworks.com/discovery/path-planning.html. Accessed 24 Oct. 2022.

[7] "How Many Sensors Are in a Drone, and What Do They Do?" *FierceElectronics*, www.fierceelectronics.com/components/how-many-sensors-are-a-drone-and-what-do-they-do.

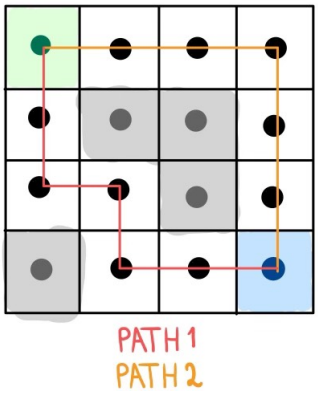
moving, people walking, and animals changing positions, all factors that will affect our algorithm and should be taken in consideration. These models will be described in further sections as for this one, we will assume that no environmental changes will occur.

The fundamental concept behind 2D path planning is to establish a framework around the path that has the shortest distance between two adjacent nodes (two connected points). Graph theory, the foundation of path planning algorithms, makes this possible. By calculating the cost of each movement (the amount of energy needed to move from point A to point B) and choosing the most effective option, will establish the shortest path. Specific algorithms as Dijkstra[8] provide a simple and efficient method to calculate the shortest path on a 2d map.

Dijkstra Algorithm

It might be interesting to know that the inspiration for writing this essay came to me as I watched a robot vacuum move around a house without the aid of a camera. I then gained a deeper understanding of the Dijkstra algorithm, the most common 2D algorithm to later understand it's limitations on a 3d model.

Figure 2



Author: Hand drawn by me

Understanding the functioning of this algorithm is key for further researches and we will do so by taking our figure 1 as sample and calculate its shortest path. Figure 2 illustrates two

[8] "Dijkstra." *Wikipedia*, 26 Oct. 2019, en.wikipedia.org/wiki/Dijkstra.

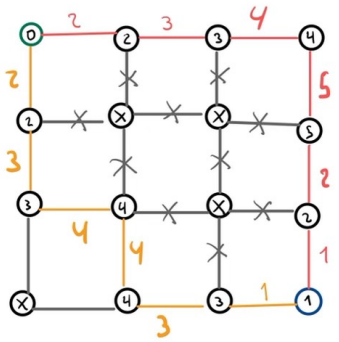
potential routes that offer the quickest way to get from the starting point to the destination while avoiding the obstacles. These paths are identical in terms of cost because they have the same number of nodes (in our case, since the graph is unweighted), as we assume that the cost of each node is equal to 1. This example was made to be mentally calculable and simple, but problems can arise when more complicated maps are used. At this stage the Dijkstra algorithm will start playing a role in mapping the shortest path

For the Dijkastra algorithm to function The nodes must be labeled with distance labels (weight of travel of previous node) and it's important that each edge will include its a travel cost. These will be named as

n = node distance label

$c(n, n')$= weight to travel edge where n is the current node and n' the adjacent node

Figure 3



Author: hand drawn by me

The distance label which is drawn inside the circles is calculated by looking at the previous edge travel cost. We know for sure that our starting point is at n=0 at this point we will compare the adjacent Distance labels[9] and select the smallest one, as it will represent our "cheapest" node to travel to. We will then store it's travel cost and repeat the process until we get to the node that will be indicated as the goal.

[9] Shortest Path "users.wpi.edu/~msarkis/MA2210/Dijkstra.pdf"

As seen by figure 3 we have labelled the values inside our boxes (n) and the weight of each edge $c(n, n')$. At this point we are able to add all the Distance labels and therefore find the overall cost to travel both the paths. We can also note that this specific map has 2 paths that will add up to have the same overall cost value.

$P1 = (0 \rightarrow 2) + (2 \rightarrow 3) + (3 \rightarrow 4) + (4 \rightarrow 5) + (5 \rightarrow 2) + (2 \rightarrow 1) = 2 + 3 + 4 + 5 + 2 + 1 = 17$

$P2 = (0 \rightarrow 2) + (2 \rightarrow 3) + (3 \rightarrow 4) + (4 \rightarrow 4) + (4 \rightarrow 3) + (3 \rightarrow 1) = 2 + 3 + 4 + 4 + 3 + 1 = 17$

Mathematicians have recently come across some Dijkstra algorithm limitations[10]. The main issue was that this algorithm wasted a lot of energy and took a long time to compute because it developed in every direction without thinking about how to concentrate on specific graph regions. Due to this, a group of researchers[11][12][13] in the early 2000s began introducing a novel method of path planning that was subsequently known as * searches[14]. We will analyse the A* and D* searches in later sections, with the latter being the most sophisticated and the one we ultimately decided to develop on a three-dimensional plane.

Star searches

The main distinction between the algorithms used in multidimensional fields and two-dimensional planes is how well they plan articulated sets of nodes. With a straightforward Dijsktra algorithm, planning a lengthy path would be incredibly slow. Because of this, it will be essential to comprehend star searches when creating an algorithm to plan a drone's path.

The fundamental principle of a star search is that we direct our search toward nodes that we think are closer to our desired state. In essence, we require a function that, at any point, will

[10] Jaliparthi, Ravikiran. PATH FINDING -Dijkstra's Algorithm. 2014.

[11] "Nils John Nilsson." *Wikipedia*, 2 Dec. 2022, en.wikipedia.org/wiki/Nils_John_Nilsson. Accessed 4 Dec. 2022.

[12] "Peter E. Hart." *Wikipedia*, 9 Nov. 2022, en.wikipedia.org/wiki/Peter_E._Hart. Accessed 4 Dec. 2022.

[13] "Bertram Raphael." *Wikipedia*, 25 May 2022, en.wikipedia.org/wiki/Bertram_Raphael. Accessed 4 Dec. 2022.

[14] "A* Search | Brilliant Math & Science Wiki." *Brilliant.org*, 2016, brilliant.org/wiki/a-star-search/.

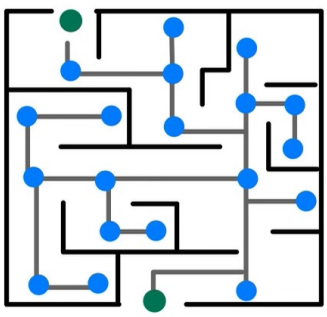
inform us of our estimated distance from the target and must be simpler and more practical than a standard search. An illustration using an example will make it simple to convey.

I like to picture a mouse getting out of a maze and going to the other side to find some cheese. The mouse will be forced to systematically explore every path until it exits by simple algorithms, but this is time-consuming and inefficient. The mouse will therefore use his nose to find the cheese and direct himself by smelling it when performing star searches. The so-called heuristic function[15] of our issue is the smell of the cheese.

Therefore with star searches the mouse will explore the maze creating a priority of areas in which he thinks it might be possible to reach the goal location.

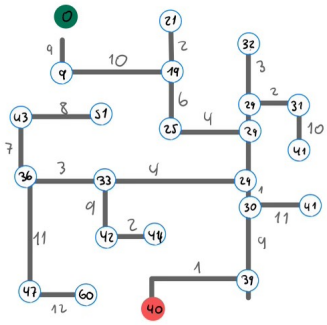
The maze with unweighted edges is depicted in Figure 4. Now that we have the options in front of us, we can draw a weighted diagram of the heuristic functions to help us prioritise the areas to explore and get a general idea of how close we are to the objective. I began to plot every weighted travel expense along each edge. At this point, I measured my distance of each node to the start, figuring out how long it would take me to travel there (values inside circles). This leaded me to develop a heuristic that, at any point, is the sum the edge costs to the starting point.

Figure 4



Author: hand drawn by me

Figure 5



Author: hand drawn be me

[15] Ferguson, Dave, et al. A Guide to Heuristic-Based Path Planning.

We can therefore plot an equation for our heuristic function.

Heuristic function:

$$h(n) = \sum n[n_s, n_p]$$

Which in words

Heuristic function= sum of all nodes from starting point to current node.

For example if we find ourselves in figure 5 at the node with 25 in the circle, the heuristic function will be calculated as:

$$h(n) = ((0 \rightarrow 9) + (9 \rightarrow 19) + (19 \rightarrow 25) = 9 + 10 + 6 = 25$$

We have now gone through different methods to determine the shortest path between two nodes, from critical reasoning to Dijkstra we have passed to simple star algorithms. It will now be helpful to understand our last 2d algorithm before extending our research to a 3d plane.

The D* Lite[16] picked the best methods from the previous methods and combined them togheter, creating an algorithm that enables re-planning at the movement of an obstacle but especially uses the heuristic functions to determine it's path.

D* Lite Algorithms

Although my initial thought was to figure out how to find the shortest path in a 2D plane, this doesn't satisfy my current research because my ultimate objective is to create an algorithm that will operate on a 3D plane[17]. Now, we'll look into specific algorithms (the D* and D* Lite algorithms) that address the problem of locating and constructing a short path between two nodes. These algorithms are based on A* searches, which are themselves constructed on the Dijkstra algorithm with a single heuristic function variation. As a result, A* algorithms are just Dijkstra with the estimated cost added. I purposely chose to look into a more sophisticated

[16] Wikipedia Contributors. "D." *Wikipedia*, Wikimedia Foundation, 18 Dec. 2018, en.wikipedia.org/wiki/D.

[17] Krafft, Carina. Implementation and Comparison of Pathfinding Algorithms in a Dynamic 3D Space Bachelor Thesis for Attainment of the Academic Degree of B.Sc.

form of A* because it supports real-time replanning and is 8 times faster than the standard A* algorithm. Dynamic A* search, also known as D* will enable us to re-route our path in response to environmental changes, while also reducing computation times and enabling the development of the path on a three-dimensional plane[18].

Defining notation will be a fundamental aspect for our working:

$$n \rightarrow \text{node/state}$$

$$c(n_1, n_2) \rightarrow \text{the weight of the edge between two nodes.}$$

$$g(n) \rightarrow \text{cost from start location to current node}$$

$$\text{b(n1)} = \text{n2} \rightarrow \text{backpointer of a node n1 to a node n2.}$$

$$h(n) \rightarrow \text{heuristic function: estimated cost to goal node}$$

These algorithms are easier to understand when analysed with a real-world example. Despite the fact that our example is simple, it will assist you in fully comprehending this algorithm's features so that you can use it to analyse a three-dimensional path.

The essential concept behind D* algorithm is about building an open list which is basically a queue of nodes that should be visited. These nodes are ordered in a priority determined by a key $k(n)$. This key is based on the objective function value and the rhs function[19]:

$f(n)$: the objective function value

$$f(n) = g(n) + c(n, n')$$

$g(n)$ is there cost from the start node to the current node

$c(n, n')$ is the cost of the current node to there adjacent node

$rhs(n)$: one- step look ahead of the objective function value

[18] "Intro to Path Planning: D* Lite vs. A*." *Www.youtube.com*, www.youtube.com/watch?v=skK-3UfcXW0&t=87s. Accessed 4 Dec. 2022.

[1919] Krafft, Carina. Implementation and Comparison of Pathfinding Algorithms in a Dynamic 3D Space Bachelor Thesis for Attainment of the Academic Degree of B.Sc.

At this point the last aspect is to take in account the consistency of the nodes. These two functions that we will later mathematically analyse help us to define the consistency of the nodes that therefore determine the position in the open list queue of nodes.

$$g(n) = rhs(n) \text{ is consistent}$$

$$g(n) \neq rhs(n) \text{ is inconsistent}$$

Inconsistency falls into two categories: Under-consistency and Over-consistency[20]

$g(n) > rhs(n)$: over consistent (path is less expensive)

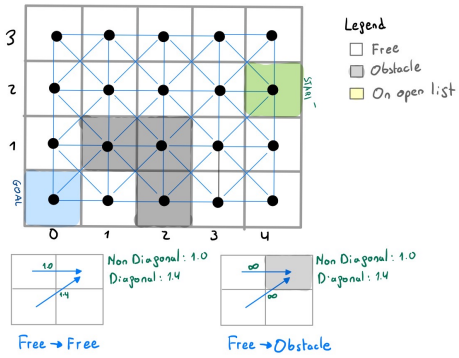
$g(n) < rhs(n)$: under consistent (path is more expensive)

If all vertices are locally consistent then g values of all vertices are equal to their respective start distances. We can therefore plot the shortest path for the map. If instead not all nodes are consistent we should make them consistent and consider their priority in the queue using the heuristic function.

Now that we have understood the theoretical reasoning behind these aspects it will be helpful to analyse them in terms of mathematical equations.

The $rhs(n)$ value is calculated based on the g values of its successors in the graph and the transition costs of the edges to those successors.

Figure 6



Author: Hand drawn by me

[20] Hayne, Rafi, et al. "Considering Avoidance and Consistency in Motion Planning for Human-Robot Manipulation in a Shared Workspace." *IEEE Xplore*, 1 May 2016, ieeexplore.ieee.org/abstract/document/7487584. Accessed 24 Oct. 2022.

rhs equation $rhs(n) = min_{n' \in Succ(n)} \ (g(n') + c(n, n'))$	The minimum of the adjacent nodes (n') contained in the successive of n adds the cost to travel from n to his adjacent + the cost to the adjacent from the start.
Priority list equation on open list $k(n) = [min(g(n), rhs(n)) + h(n_{start}, n); min(g(n), rhs(n))$	The minimum between the g and rhs values plus a focusing heuristic function.

The key of the priority list will be to analyse the consistency of the nodes and therefore decided the path through over-consistency and under consistency. This works perfectly until we reach the point that both nodes are equally under or over consistent, at this point we will therefore use the secondary key in the equation which will involve the use of the heuristic function to estimate the shortest path.

Therefore to compute the shortest path we must analyse the consistency between the $g(n)$ and $rhs(n)$ if the value is over consistent (less expensive) we know it will more likely to be the correct path and therefore we will set the $rhs(n)$ value equal to the $g(n)$. On the other hand if the value is under consistent it's less likely to be the correct path and therefore will set the $rhs(n)$ value to infinite.

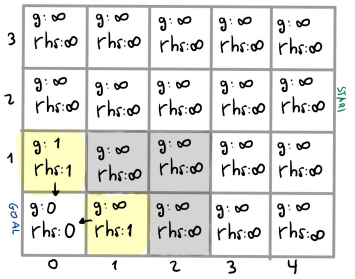
Now knowing this we can develop an example

Figure 6 will be our model diagram where the green box will be our starting point and the blue box our goal location. The image also depicts the cost of movement in a diagonal and non-diagonal manner also including a legend as reference to the colours of the states.

To calculate the shortest path as previously discussed we must compare the $rhs(n)$ and $g(n)$ values and their consistency to develop an open list of values to develop. We know by

definition that our g value has the $g(n) = rhs(n) = 0$ and therefore will start from there. For simplicity we will also set all the other values to infinite and as we start computing them will add their specific costs. (Figure 7)

Figure 7



Author: hand drawn by me

As seen in Figure 7 we will start computing the open list queue from the goal node. We will expand our starting node meaning that we will compute their rhs values known by the cost index shown in figure 6.

$$rhs(n) = min_{n' \in Succ(n)} \, (g(n') + c(n, n'))$$

$$rhs((0,1);(1,0)) = min_{n' \in Succ(n)} \, (0 + 1) = 1$$

As both of the values have the same rhs value we will look at the heuristic function which might differ. The heuristic for the point (1,0) is given by as there is an obstacle:

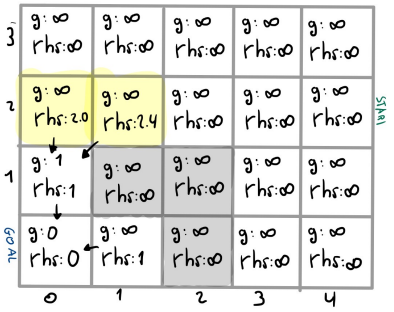
$$h(n) = \sum n[n_s, n_p] \qquad h(1,0) = \sum \infty + 1 = \infty$$

While the heuristic for the point (0,1) is given by

$$h(n) = \sum n[n_s, n_p] \qquad h(0,1) = \sum 1 + 1 = 2$$

We will therefore expand our (0,1) node.

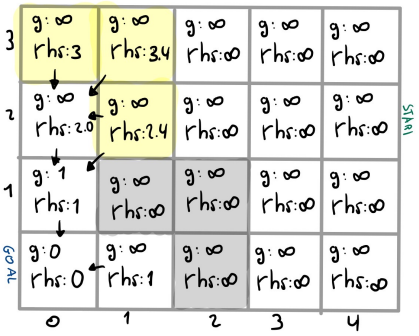
Figure 8



Author: Hand drawn by me

As seen in figure 8, the expansion of the node (0,1) will follow the exact same procedure as the previous node and therefore will decide the minimum node as (0,1) as contains lower rhs values. I will therefore not show the calculations as the following nodes will be more important.

Figure 9



Author: hand drawn by me

At this point we will expand the node (0,2) and set in the open list the three different nodes. (Figure 9) The rhs values will be respectively

$$rhs((0,3)) = min_{n' \in Succ(n)} (0 + 1 + 1 + 1) = 3$$

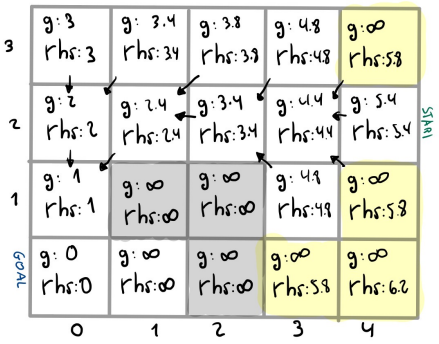
$$rhs((3,1)) = min_{n' \in Succ(n)} (0 + 1 + 1 + 1.4) = 3.4$$

$$rhs((2,1)) = min_{n' \in Succ(n)} (0 + 1 + 1.4) = 2.4$$

Therefore the minimum node will be (2,1) and decide to expand it.

As shown in figure 10 we will continue this process until we reach our starting node and therefore have computer the shortest path using a D* Lite algorithm.

Figure 10



Author: hand drawn by me

As a result, we have successfully followed the steps in our hypothetical example and have a complete understanding of how these sophisticated algorithms work. It is impressive to understand the various applications of these systems and how they have advanced society. D* lite algorithms are one of the most sophisticated path planning technologies, used in everything from space-based autonomous robots to self-driving cars[21].

We are all aware of the harm that inefficient transportation causes to the environment and how finding a substitute for cars might alter our outlook on the world. Being a self-professed owner of an FPV drone with which I enjoy taking pictures and videos, I wondered how these technologies might be developed to create a sustainable and effective mode of transportation. I've discovered through my research that autonomous drones may be beneficial for people and logistics in particular. To fully comprehend how an autonomous drone might operate, I want to extend the D* lite algorithm on a 3D plane as part of my investigation.

[21] Krafft, Carina. Implementation and Comparison of Pathfinding Algorithms in a Dynamic 3D Space Bachelor Thesis for Attainment of the Academic Degree of B.Sc.

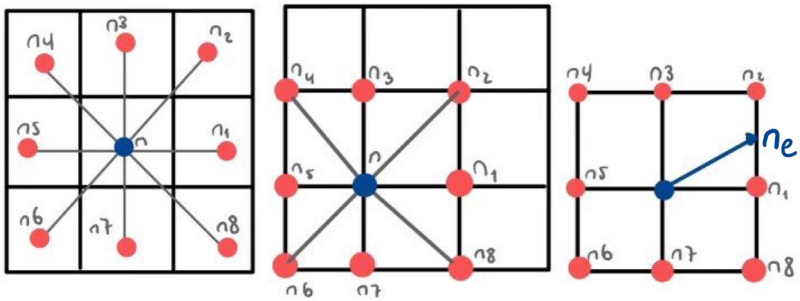
3d Path Planning

D* Lite algorithm on n-dimensional planes

When considering how to generate a smooth path between two points[22], the limitations of using basic path planning algorithms become evident. The entire environment will be mapped as a grid (previously described), with the only transition points being on the grid's nodes (red points in figures), limiting the ability to create a smooth path and movement[23].

It would not be possible to move autonomously a vehicle on a 3d plane limiting its movements to only nodes of a grid.

Figure 6



Author: Hand drawn by me

Therefore, the D* lite algorithm enables the vehicle to create a path of infinite nodes on a plane not only limiting itself on nodes but giving the possibility of movement on a whole edge

As Seen in figure 6 the starting point (blue) can transition itself to any section of the bold line, not only limiting itself to the red points. Interpolation[24] is used to plan paths from a node n to any point N_e on an adjacent edge. In the figure 6 the edge between nodes n_1 and n_2 is being considered rather than just the endpoints of these edges.

[22] [22] Song, Baoye, et al. "An Improved PSO Algorithm for Smooth Path Planning of Mobile Robots Using Continuous High-Degree Bezier Curve." *Applied Soft Computing*, vol. 100, 1 Mar. 2021, p. 106960, www.sciencedirect.com/science/article/abs/pii/S156849462030898X, 10.1016/j.asoc.2020.106960. Accessed 25 Oct. 2022.

[23] [23] Carsten, Joseph, et al. 3D Field D*: Improved Path Planning and Replanning in Three Dimensions.

[24] "What Is Extrapolation and Interpolation? - Definition from WhatIs.com." *WhatIs.com*, www.techtarget.com/whatis/definition/extrapolation-and-interpolation.

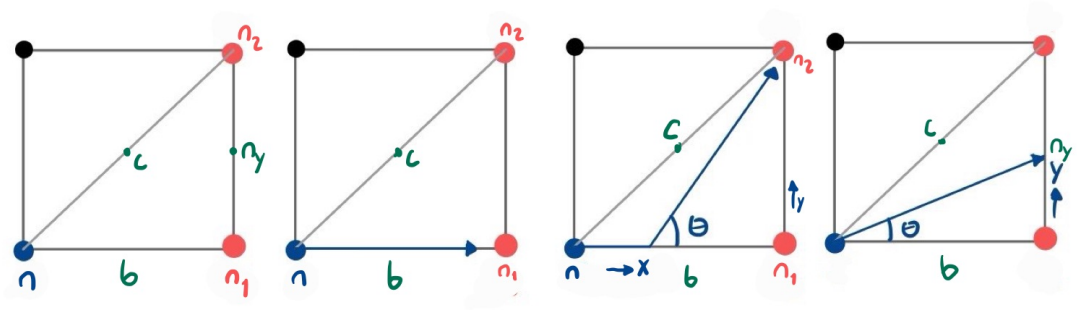
Let's name the edge from n_1 to n_2 as $g(n_e)$ in which every point on the edge can be a new node. We could deduce the real cost from our starting node n to every point as we have previously done:

$$g(n_e) = c(n, n_y) + g(n_y)$$

It is impossible to calculate the cost of because there are an infinite number of points on n_y,

For this reason we use linear interpolation. To do so, it is assumed that every value $g(n_y)$, with n_y being a point on the edge connecting n_1 and n_2, is a linear combination of $g(n_2)$ and $g(n_1)$.

Figura 8



Author: hand drawn by me

We can therefore create a cost equation using interpolation as visualised in figure 8 where $g(n_y)$ is the midpoint n_y between s_1 and s_2

$$g(n_y) = y \cdot g(n_2) + (1 - y) \cdot g(n_1)$$

This alternative option provides the possibility to access different cells at any position and not only on the edges of the cell. The interpolation method can hence be used to calculate the cost from any given point given that it's not on an edge point.

We can now plug in our specific cost equation for $g(n_y)$ into our initial equation to get our cell cost equation:

$$g(n) = min(bx + c\sqrt{(1-x)^2 + y^2} + yg(n2) + (1-y)g(n1))$$

It might happen that it is not always the case to pass through the central cell as it might be a more expensive path compared to the path that will go through the n1 node. If its cheaper to get the traditional path through n1 then we can say that $y* = 0$ if instead it is cheaper to pass through the central cell we say that either $x* = 1 \mid y* = 0$.

The most optimal depends on the relative sizes of cell costs c and b, as well as the "difference of path cost between n1 and n2". We can therefore define the f function as:

$$f = g(n_1) - g(n_2)$$

If g(n1) < g(n2) it is cheapest to travel along the traditional path without crossing the cell. g(n) is not dependent on y and can simply be calculated by looking at figure 8 as:

$$g(n) = min(c, b) + g(n1)$$

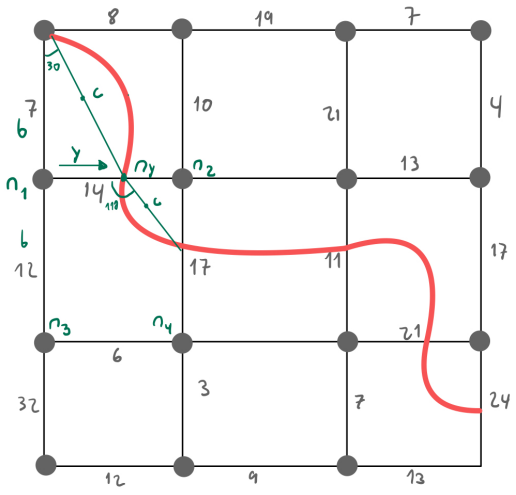
Instead if f < b we can say that it is more expensive to travel through the bottom edge, we can state that. In this case k is set to k = f

$$g(n) = c\sqrt{1 - y^2} + k(1 - y) + g(n_2)$$

Knowing this we can therefore develop the D* lite algorithm that will determine a smooth path for a given set of cells.

An example will make it all easier to understand.

Figure 9



Author: hand drawn by me

Figure 9 illustrates a straightforward grid that will be used as a model for creating an interpolation for the provided red route. I've decided that the red path is the shortest for the purposes of the inquiry and am interested in computing its cost.

The first step is to decide whether it is more practical to pass through the center of the grid or along the regular grid edges. For this, we must compare the costs of n1 and n2 in accordance with the earlier descriptions.

$$g(n_1) = 7 \mid g(n_2) = 14$$

$$\therefore$$

$$g(n_1) < g(n_2)$$

Looking at figure 9 we know the cost of the n_1 and n_2 and can determine which will be the more expensive.

If n1 is less expensive than n2 we know that the more convenient route will be along the edges, the traditional path. As shown in figure 9 the letters b and c are the corresponding values for the side path cost and the central path cost, these can be determined using simple trigonometric equations and will be needed for computing the general cost of the first cell.

$$b = 7 \mid c = \frac{7}{tan 30}$$

We know that the b value is equivalent to the cost between the n_0 and the node n_1. Using simple trigonometry we can also calculate the c value.

As previously stated we know that the shortest path will be along the edges and therefore use the equation described previously to compute the general cost of the cell.

$$g(n) = \min(b, c) + g(n_2)$$

$$\therefore$$

$$g(n) = \min(7, \frac{7}{tan30}) + 14 = 7 + 14 = 21$$

The general cost is given by the minimum value between b and c and the cost of the second node.

We have concluded that the general cost for the cell will be of 21, this could have also been done using simple addition of the two node costs, but was fundamental to understand the method of interpolation for further investigations.

Following our red line, we can do the same for the second cell. As with the same cell we need to compare the cost of n3 with n4

$$g(n_3) = 12 \,|\, g(n_4) = 6$$

$$\therefore$$

$$g(n_3) > g(n_4)$$

$$\therefore$$

$$f = g(n_3) - g(n_4) = 12 - 6 = 6$$

Looking at figure 9 we know the cost of the n_3 and n_4, and by comparing them know that the middle path will be less expensive.

Knowing this we will need a value for f which will be needed for further calculations.

From these point we know that the middle path will be the less expensive and will therefore use the equation for interpolation

$$g(n) = c\sqrt{(1 - y^2)} + k(1 - y) + g(n_4)$$

$$\therefore$$

$$g(n) = 10\sqrt{(1 - (tan110 \cdot 12)^2)} + 6(1 - (tan110 \cdot 12)) + 6$$

From previous explanations we know that f=k and can also compute the cost of y using trigonometric equations.

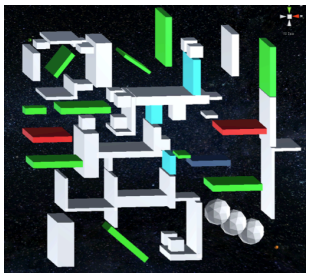
As for the previous section we know the values of c and b by simple trigonometric functions.

For the purposes of this investigation I will not continue investigating the total cost of the grid as 2 cells were necessary to understand the method and procedure required to calculate the cost through interpolation.

As previously described interpolation method with D* algorithm is used to determine a smooth path between a given set of nodes. As happened to be, I started wandering how this method could be helpful to investigate the shortest path in a three dimensional planes.

This was simpler than what as I expected as the D* algorithm can be used to compute the shortest path on 3d plane by breaking down the different xyz planes and calculating a smooth path for each of them to later add them and determine the shortest path on a 3d plane that will avoid obstacles and not require excessive amounts of computational energy.[25] The mathematical calculations required to compute the shortest path in any given 3d grid will be too long but identical to the ones showed in this section. For this reason I decided to not manually find the shortest path on a 3d plane but use some code that literally explains all the steps I have developed with the only difference that it has the possibility to easily repeat the process for all the given cells. This process is shown in the following figures.

Figure 10



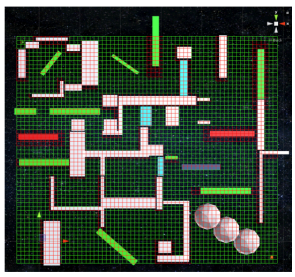
Author: external source

Figure 12

ComputeCost(s, s_a, s_b)
01. if (s_a is a diagonal neighbor of s)
02. $\quad s_1 = s_b$; $s_2 = s_a$;
03. else
04. $\quad s_1 = s_a$; $s_2 = s_b$;
05. c is traversal cost of cell with corners s, s_1, s_2;
06. b is traversal cost of cell with corners s, s_1 but not s_2;
07. if $(\min(c, b) = \infty)$
08. $\quad v_s = \infty$;
09. else if $(g(s_1) \leq g(s_2))$
10. $\quad v_s = \min(c, b) + g(s_1)$;
11. else
12. $\quad f = g(s_1) - g(s_2)$;
13. $\quad$ if $(f \leq b)$
14. $\quad\quad$ if $(c \leq f)$
15. $\quad\quad\quad v_s = c\sqrt{2} + g(s_2)$;
16. $\quad\quad$ else
17. $\quad\quad\quad y = \min(\frac{f}{\sqrt{c^2 - f^2}}, 1)$;
18. $\quad\quad\quad v_s = c\sqrt{1 + y^2} + f(1 - y) + g(s_2)$;
19. $\quad$ else
20. $\quad\quad$ if $(c \leq b)$
21. $\quad\quad\quad v_s = c\sqrt{2} + g(s_2)$;
22. $\quad\quad$ else
23. $\quad\quad\quad x = 1 - \min(\frac{b}{\sqrt{c^2 - b^2}}, 1)$;
24. $\quad\quad\quad v_s = c\sqrt{1 + (1 - x)^2} + bx + g(s_2)$;
25. return v_s;

Author: Krafft, Carina. Implementation and Comparison of Pathfinding Algorithms

Figure 11



Author: external source

[25] Krafft, Carina. Implementation and Comparison of Pathfinding Algorithms in a Dynamic 3D Space Bachelor Thesis for Attainment of the Academic Degree of B.Sc.

Final thoughts

Evaluations and Extensions

The goal of this investigation has been to deepen my knowledge on path planning algorithms and how they could help future generations to develop autonomous mobility systems which could revolutionise the way we conceive transports in our cities.

Two dimensional path planning algorithms found its application in several different fields such as satellite navigators or more modern autonomous ground vehicles, the challenge though was to extend the investigation on a three dimensional field where much more complicated circumstances could occur. I am aware of the fact that algorithms can become extremely more complicated but my interest was in understanding the basic functioning of these amazing masterpieces of technology.. It has been extremely important and challenging to not loose myself in extremely interesting topics that yet didn't serve to reach the goal of my investigation. I am also aware of much further I could develop the topics discussed in the paper, yet conciseness was also a key aspect to meet and not falling in the trap of further extending the investigation has revealed to be challenging.

Conclusion

I can't deny how much effort and struggle I had to put in order to successfully complete this paper. Understanding and explaining in easy terms the topics covered is not easy at all. The most challenging aspect beside understanding the topic itself was to express what I meant in a clear and concise manner. I really wanted that everyone who reads it could understand the reasoning behind these algorithms.

Beside being an exhausting work, I feel extremely satisfied and happy with what I was able to complete. This is why math and scientific subjects in general have became my passion, I really enjoy the satisfaction when you complete a work of this kind especially having so much fun.

Bibliography

Carsten, Joseph, et al. 3D Field D*: Improved Path Planning and Replanning in Three Dimensions.

Har-Peled, Sariel. “Constructing Approximate Shortest Path Maps in Three Dimensions.” Proceedings of the Fourteenth Annual Symposium on Computational Geometry - SCG ’98, 1998, 10.1145/276884.276927.

Krafft, Carina. Implementation and Comparison of Pathfinding Algorithms in a Dynamic 3D Space Bachelor Thesis for Attainment of the Academic Degree of B.Sc.

“A* (a Star) Search and Heuristics Intuition in 2 Minutes.” Www.youtube.com, www.youtube.com/watch?v=71CEj4gKDnE. Accessed 1 Dec. 2022.

“Advanced 1. Incremental Path Planning.” Www.youtube.com, www.youtube.com/watch?v=_4u9W1xOuts&t=591s. Accessed 1 Dec. 2022.

Autoren der Wikimedia-Projekte. “Lateinischer Buchstabe.” Wikipedia.org, Wikimedia Foundation, Inc., 30 Aug. 2002, de.wikipedia.org/w/index.php?title=%20A. Accessed 1 Dec. 2022.

BlueRaja. “High Speed Priority Queue for C#.” GitHub, 1 Sept. 2022, github.com/BlueRaja/High-Speed-Priority-Queue-for-C-Sharp.

Booth, Michael, et al. The Official Counter-Strike Bot.

Botea, Adi, et al. Near Optimal Hierarchical Path-Finding.

Carsten, J., et al. “3D Field D: Improved Path Planning and Replanning in Three Dimensions.” Undefined, 2006, www.semanticscholar.org/paper/3D-Field-D%3A-Improved-Path-Planning-and-Replanning-Carsten-Ferguson/a671c91945b6da3af697b83f7f4f9d39a0732cfb. Accessed 1 Dec. 2022.

Choset, Howie. Robotic Motion Planning: A* and D* Search.

"Compare A* with Dijkstra Algorithm." Www.youtube.com, www.youtube.com/watch?v=g024lzsknDo&t=14s. Accessed 1 Dec. 2022.

"Donald Bren School of Information and Computer Sciences @ University of California, Irvine." Www.ics.uci.edu, www.ics.uci.edu.

"DStarLite." Www.youtube.com, www.youtube.com/watch?v=wsHiqMX1EQ8. Accessed 1 Dec. 2022.

"Heuristics." Theory.stanford.edu, theory.stanford.edu/~amitp/GameProgramming/Heuristics.html.

"Introduction to Artificial Intelligence Mitch Marcus CIS391 Fall, Ppt Download." Slideplayer.com, slideplayer.com/slide/8333201/.

Koenig, S., and M. Likhachev. "Fast Replanning for Navigation in Unknown Terrain." IEEE Transactions on Robotics, vol. 21, no. 3, June 2005, pp. 354–363, 10.1109/tro.2004.838026. Accessed 9 June 2022.

Koenig, Sven, and Maxim Likhachev. D* Lite.

Nash, Alex, and Sven Koenig. "Any-Angle Path Planning." AI Magazine, vol. 34, no. 4, 18 Sept. 2013, p. 85, 10.1609/aimag.v34i4.2512.

"OpenReview." OpenReview, openreview.net/profile?id=~Max_Welling1. Accessed 1 Dec. 2022.

Wikipedia Contributors. "A* Search Algorithm." Wikipedia, Wikimedia Foundation, 10 Mar. 2019, en.wikipedia.org/wiki/A.