A picture containing text

Description automatically generated

Using Modern Mapping and Graph Techniques to Traverse Mountainous Regions.

Submitted by: Daniel Jameson

T-Number: T00158237

Computing with Software Development

Date Submitted: XX/YY 2025

Github Repository: <https://github.com/DogPope/FYP-Development>

Submitted in part-fulfillment of the requirements for the award of

Bachelor of Science (Honours) in Computing with Software Development

Munster Technological University

Table of Contents

Contents

[Chapter 1: Abstract 4](#_Toc184562929)

[Chapter 2: Introduction 5](#_Toc184562930)

[Chapter 3: Literature Review 6](#_Toc184562931)

[3.1 Search and Rescue(SAR) 6](#_Toc184562932)

[3.2 Terminology Relating to Missing People 6](#_Toc184562933)

[3.3 Defining Search Areas 6](#_Toc184562934)

[3.4 Researching and Defining a Sample Area 7](#_Toc184562935)

[3.5 Investigating Navigation Techniques 7](#_Toc184562936)

[3.5.1 Pathfinding in 3D Environments 7](#_Toc184562937)

[3.5.2 Global Positioning System 8](#_Toc184562938)

[3.5.3 Graphs and Corresponding Algorithms 8](#_Toc184562939)

[3.6 Methods of Assessing Physical Terrain 8](#_Toc184562940)

[3.6.1 Photogrammetry 9](#_Toc184562941)

[3.6.2 LiDar(LIght Detection And Ranging) 9](#_Toc184562942)

[3.6.3 Haversine Formula 9](#_Toc184562943)

[3.6.4 Google Elevation API 10](#_Toc184562944)

[3.6.5 AI / Machine Learning to Improve Pathfinding 10](#_Toc184562945)

[3.6.6 Carrauntoohil 11](#_Toc184562946)

[3.7 Hardware and Technology 11](#_Toc184562947)

[3.7.1 Python 11](#_Toc184562948)

[3.7.2 NumPy 11](#_Toc184562949)

[3.7.3 A\* Algorithm 11](#_Toc184562950)

[3.7.4 Lazy Theta\* Algorithm 13](#_Toc184562951)

[Chapter 4: Methodology 14](#_Toc184562952)

[4.1 Research Questions and Objective 14](#_Toc184562953)

[4.2 Gathering Necessary Data 14](#_Toc184562954)

[4.2.1 Acquiring Elevation of Selected Search Area 14](#_Toc184562955)

[4.2.2 Displaying Plots 15](#_Toc184562956)

[4.3 Determining the Distance and Points to be Displayed 15](#_Toc184562957)

[4.3.1 Determining a Bounding Box using the Haversine Formula 16](#_Toc184562958)

[Chapter 5: Design 18](#_Toc184562959)

[5.1 Design Justification – User Stories 18](#_Toc184562960)

[5.1.1 Emergency Rescue Assist 19](#_Toc184562961)

[5.1.2 Outdoor Recreational User 20](#_Toc184562962)

[5.1.3 Environmental Monitoring 21](#_Toc184562963)

[5.1.4 Educators and Trainers 22](#_Toc184562964)

[5.2 Prototype 23](#_Toc184562965)

[5.2.1 Set up Working Environment 23](#_Toc184562966)

[5.2.2 Import the Necessary Libraries into VSCode 23](#_Toc184562967)

[5.2.3 Using Matplotlib to Display the Data 24](#_Toc184562968)

[5.2.4 Finding a Simulated Path to Traverse. 25](#_Toc184562969)

[5.2.5 Path Between Two Points 26](#_Toc184562970)

[Appendix - Glossary 29](#_Toc184562971)

[Appendix – External Links and Information 30](#_Toc184562972)

[References 31](#_Toc184562973)

**Table Of Figures**

[Figure 1: Haversine Formula 9](#_Toc184556972)

[Figure 2: Elevation API 10](#_Toc184556973)

[Figure 3: A\* Algorithm (Nash, 2010) 12](#_Toc184556974)

[Figure 4: Lazy Theta\* Algorithm 13](#_Toc184556975)

[Figure 5: Producing Elevation from Points 15](#_Toc184556976)

[Figure 6: Verifying Summit Coordinates 16](#_Toc184556977)

[Figure 7: Inverse Haversine 17](#_Toc184556978)

[Figure 8: JIRA Board 18](#_Toc184556979)

[Figure 9: Work Environment 23](#_Toc184556980)

[Figure 10: Environment Variables 23](#_Toc184556981)

[Figure 11: MatPlotLib Code 24](#_Toc184556982)

[Figure 12: Data Visualisation 25](#_Toc184556983)

[Figure 13: Lowest and Highest Coordinates 25](#_Toc184556984)

[Figure 14: Distance Between Lowest and Highest Points 26](#_Toc184556985)

[Figure 15: Folium Code 27](#_Toc184556986)

[Figure 16: Basic Path Between Points 28](#_Toc184556987)

# Chapter 1: Abstract

The subject of this report is going to concern the use of modern data analysis and gathering techniques and to investigate how these techniques may be used to improve and enhance terrain navigation going forward. Traditionally, sight-based methods were used to accomplish this goal, over time humanity upgraded to using different types of maps to design navigation routes. These techniques are old and outdated, as modern technology proposes opportunities to refine the pathfinding process.

Weather, geographical change and daylight are just a few of the factors that influence how dangerous any given hiking adventure may play out. In modern times, humanity is better able to estimate the worth of human capital, and we as a species have realised the importance of our own safety.

Modern technology such as drones, and mapping techniques could be combined to create a better and less costly solution that improves both the time taken to carry out these operations and reduce the associated human cost that goes with such operations.

# Chapter 2: Introduction

The subject of gathering the best possible route to a destination is a question that humanity has grappled with for as long as we have been documenting our journeys. This project aims to detail modern techniques and technology that could be used for the purpose of optimising navigation routes.

Over the course of time, we have come up with newer, more clever ways to make our journeys more successful. The first major advancement humanity made was maps. The earliest example of a map was found carved into the tusk of a woolly mammoth, later discovered in Ukraine (Norman, 2024). The map was later interpreted to show a series of dwellings alongside a river.

Later, we developed more advanced methods for getting from A to B. Improved cartography techniques allowed us to gather more detail on our surrounding environment than ever before. Physical maps, terrain maps and many more types of maps come together to give us the most possible amount of information to gather these routes.

Modern computing power allows us to make calculations at a speed never thinkable before. This report will aim to detail how that computing power can be leveraged for the benefit of route-finding and pathing over complex environments that would otherwise be difficult to navigate.

In the case of this report, the challenge lies in navigating complex, 3D environments that is beholden to unpredictable factors, such as the weather and the changing environment, making for conditions that are difficult to anticipate. Advanced graphing and pathing techniques will be used to solve this problem to the best of the ability of modern technology.

# Chapter 3: Literature Review

## 3.1 Search and Rescue(SAR)

Search and Rescue is a term that refers to the location of people to be considered missing in areas of hazardous environments. The process of SAR generally involves sending a search party out to the general location of the missing party and dividing manpower out into smaller parties and deciding to divide out the search area into more manageable sections.

The first major advantage in the field of SAR was the use of dogs to locate people. Dogs have a heightened sense of smell with significantly more smell receptors than humans (Agata Kokocińska-Kusiak, 2021), allowing them to tie a smell to the person they are searching for, facilitating faster location of the subject.

In modern times, this process takes advantage of numerous advanced technologies such as drones and helicopters to aid in the discovery of potential victims. Due to the increasing technology in recent years, the manufacturing of certain types of helicopters and drones specifically for the purpose of SAR, such as the S-70i – Search and Rescue Helicopter. The process of using drones has gained massive popularity in recent years, as it is a safer option than sending helicopters or people to check for the presence of the subject, due to it not being necessary to send living beings to do the work. These drones often come equipped with thermal imaging cameras to further speed up the process, such as the DJI M300RTK.

## 3.2 Terminology Relating to Missing People

The terminology associated with missing people is an area that needs advancement. There are not many terms associated with missing people that work functionally. There is no shorthand for a missing person, they just have to be called “Missing Person” or “subject”, or any other suitable word that just applies to someone who may be missing.

The problem with the lack of terminology in relation to missing persons is that time needs to be spent unnecessarily on defining terms each time an instance of missing people happens. Situations need to be described in more detail for each circumstance, and adds time to the investigation process, and possibly adds time to any rescue that the subject might not possibly have.

This means that any attempt to search on the subject is unnecessarily hindered by a lack of search terms that relate to specific circumstances and any attempt at really taking advantage of modern technologies in the acquisition of missing people is hindered by bureaucracy and red tape, as all these circumstances must be described in greater detail than initially anticipated. More work in this area is needed, although is outside the scope of this report.

## 3.3 Defining Search Areas

Typically, on a global scale, the world is divided into different regions where bodies are responsible for defining the areas under the authority of each organisation. This is a primary division of search areas. There exist 13 main areas of consideration under the SAR.

When an area needs to be searched, the area needs to be assessed where the approximate location of the missing person needs to be gathered. The process of dividing a search area falls under the jurisdiction of two United Nations organisations, the International Maritime Organization(IMO) for sea SAR missions, and the International Civil Aviation Organization(ICAO) for ariel missions.

This process is initially done by defining the main area that someone may have gone missing. This is accomplished using mapping techniques. Then, the area is broken down into smaller, more manageable sub-sections that are more easily searched.

In Ireland for sea SAR operations, the Irish Coast Guard is responsible for abiding by the guidelines set out by the National Search and Rescue Plan, and they have bases in Dublin, Valentia Island and Malin Head. (Gov.ie, 2019)

For ground based SAR operations, the National Search and Rescue Plan does not clearly define who is responsible, alluding to An Garda Siochana having a duty to safety. It states that: “There are no international conventions governing land search and rescue. However, legislation governing policing activity places an obligation on An Garda Síochána to protect life and property, and the provision of land SAR services derives from this requirement” (Gov.ie, 2019).

## 3.4 Researching and Defining a Sample Area

The area in the surrounding where this report was written is mainly mountainous and has jagged, rough terrain. This terrain is naturally difficult to design infrastructure around and to navigate correctly. In the year 2022, a hiker named John Dunne went missing around the mountain of Carrauntoohil, Co. Kerry (Irish Examiner, 2022).

The article describes Mr. Dunne as an experienced walker. Cases like the one of Mr. Dunne demonstrate the need for advanced search and rescue techniques.

The area this project is going to focus on is Carrauntoohil. The mountain, located in the MacGillycuddy Reeks Mountain range, is a formidable area that presents a variety of challenges in the terrain and areas to be assessed. There are lots of jagged rocks, steep inclines and other hazards that may need to be mapped and accounted for when deciding what areas to assess and what kind of weighting may need to be applied in which given circumstance. (Kerry Mountain Rescue, N/A).

## 3.5 Investigating Navigation Techniques

Modern technology can be used to improve the efforts of Search and Rescue services by making parties more efficient and reducing the elements of human liability that are inherent to old fashioned techniques. Using people to do this requires a high amount of human capital to be invested, as the people involved are required to be strong and fit enough to be able to access the areas someone may get lost in.

The obvious solution to this problem is to use forms of technology as a substitute for this human capital. The use of tech such as drones allows humans, and by extension, associated liabilities to be limited. This section is going to detail some of that technology.

### 3.5.1 Pathfinding in 3D Environments

In 3D spaces, the options for pathfinding are steadily improving, as the objective of 3D spaces are naturally more complex than in a 2D environment. Therefore, the challenges need to be met appropriately with the correct mix of modern technology and their applications. For 2D applications, Graph theory and the A\* algorithm can be used to ascertain the shortest path between two points (Alex Nash, 2010). For 3D spaces, the process of pathfinding becomes more complicated, requiring different algorithms or modifications to existing algorithms to gather the shortest path.

### 3.5.2 Global Positioning System

The term Global Positioning System(GPS) refers to any system that sends broadcast navigation pulses to users on earth (Logsdon, 2022). It is a useful system for locating and guiding a given device with the necessary information to make informed decisions as to how to get from one location to another. These systems are then work with different pathfinding algorithms to reach different points for different purposes. For example, Shortest Path First would allow the user to demonstrate a path that takes the least distance between two points. Whereas something like the A\* algorithm that would take a weighted approach to solving the least costly method of reaching a given point.

The first instance of Global Positioning System was in the era of the Sputnik satellite in 1957, when scientists discovered that they could track satellites using shifts in their radio signals. This uses the principal that any receiving device could determine its position based on the strength of the signal. (Aerospace, 2024)

### 3.5.3 Graphs and Corresponding Algorithms

In computing, graphs are an abstract data type that connect nodes(vertices) via edges. Graphs come in many different forms such as directed or undirected and weighted or non-weighted. Graphs are commonly used in Social Networks to determine which users are likely to know each other based on mutual contacts, and weighted graphs are commonly used in mapping projects as they provide a weighting between vertices to make their calculations (Thomas H. Cormen, 2009) P589.

Pathfinding algorithms will have to be implemented into the project to decide how to navigate over a search area. There exists a multitude of algorithms to use for this purpose. Dijkstra's algorithm, Breadth first and Depth First will all receive consideration in factoring in which is the likely way to reach people first from a given location. There are promising results posted from studies involving these algorithms (Pajaziti Arbnor, 2024). The main area to consider researching here is how existing navigation systems utilise these algorithms and use a similar system for this report.

The type of graph used by this project should depend on the type of pathfinding that needs to be done. The terrain of mountains provides a unique challenge in that it provides a third dimension to the proceedings. This will impact pathfinding such that the differences in elevation must be accounted for more so than if one were to design a standard pathfinding system, such as satellite navigation.

## 3.6 Methods of Assessing Physical Terrain

As time goes on, the options available in terms of assessing geographical data become more varied and more impressive. Even the public now has access to technology such as Unmanned Arial Vehicles (Drones) that allow the user to assess any number of objects or terrain of their choosing. At a wide range of technologies and prices, these options need to be carefully considered for their purpose so that the objective is achieved in the most reasonable manner possible. This section aims to detail some of those technologies to decide the correct tool for the job. (Liyang Xiong, 2022)

### 3.6.1 Photogrammetry

Photogrammetry is the process of using various techniques and formulas on photos to assess a physical object and gather measurements from them (3DSourced, 2023). When these areas are considered, the user should then be able to map a 3D space and plan out a route according to graph algorithms.

Photogrammetry breaks down into multiple smaller techniques based on the angle of photo that is taken. If using vertically taken photos, the horizontal axis is pronounced and if using oblique photogrammetry, the vertical axis makes itself more pronounced.

Photogrammetry could be implemented on a completed bounding box to gather photo data on any selected bounding box. This could allow any application to assess the geography in other ways, such as marking out forests and rougher terrain (SPH Engineering, 2017).

### 3.6.2 LiDar(LIght Detection And Ranging)

LiDar is a technology involving the use of lasers to assess the distance between a sensor and an object. LiDar works by sending a laser to an object and measuring the time it takes to return to the source. Lidar is particularly useful in automobile applications, as it is often used for parking sensors to determine the distance between the sensor and any detected object (You Li, 2020).

LiDar could be used here, as a method of determining positioning in relation to given objects, nodes and vertices on a graph, in practice this could be useful in implementation to maintain distance from objects.

A LiDar service could in theory be used from a drone to gather necessary information for making a suitable path to a given point.

### 3.6.3 Haversine Formula

The Haversine Formula is used to calculate distance between two points on a sphere (Kettle, 2017). The Haversine Formula uses two points of latitude and longitude and returns a distance in Radians when calculated. The Haversine Formula takes in Latitude and Longitude belonging to two points and returns the distance.

A white background with black text

Description automatically generated

Figure : Haversine Formula

This project will need to use this formula to ascertain the size of any bounding box that may need to be created around the selected area. In this case, distance must be calculated from the source point to determine the size of a given bounding box.

The Python Programming language has a library for using the Haversine formula specifically, predictably named Haversine. This library will be of particular use, as it holds a function called “Inverse Haversine” that will be use a vector and source coordinate to determine a point that is a selected distance away from the source. (PyPi, 2024).

### 3.6.4 Google Elevation API

Google hosts an API for returning the elevation of any point that is indexed on one of their maps. This information could prove to be useful in assessing the surrounding terrain and provide information to incorporate into the pathfinding route. The API receives an input of longitude and latitude, and returns an elevation to the user at the given coordinates. This returns either a JSON or an XML response in Meters. It should be noted that this a paid API, although usage can be forcibly limited. It must be used sparingly, or a free solution could be found. (Google, 2024) This report is going to assume the use of Googles Elevation API, and a return type of JSON.

If the bounding box for the project was to be extended to a more serviceable range, a range of 5 metres per box section, the project would be rendered prohibitively expensive. This would produce 160,801 different requests to the Google Maps API, at a cost of 4 cents each as it falls within the second category. This would produce a total number of $643.20 Dollars. For the purposes of this report, that would be considered prohibitively expensive.

Figure : Elevation API

This leaves the project with two options, using an alternative to Google Maps Elevation and using the bounding box with more divisions, which would be more accurate but expensive, or using a bounding box with less divisions, therefore, accuracy would suffer as a result, but would be cheaper to make.

### 3.6.5 AI / Machine Learning to Improve Pathfinding

Artificial Intelligence has a range of definitions including capability to perform tasks comparable to human intelligence and computing systems having the ability to think for themselves (Martinez, 2019). Artificial intelligence is a massively growing industry with the possibility of integration into pathfinding algorithms.

Artificial Intelligence could be used here to increase the efficiency of any route if given information on changing factors in the route. Weather, terrain changes and the possibility of landslides demonstrate the need for an updated data set that would allow users to change their route accordingly. AI has been used to reroute paths in applications such as Satellite Navigation and has a history of usage in similar applications.

### 3.6.6 Carrauntoohil

Carrauntoohil is a part of the MacGillycuddy Reeks Mountain Range, in Co. Kerry, Ireland. It is Irelands highest summit and is considered a formidable challenge for experienced hikers. It is often the source of Search and Rescue, as it is a huge area with lots of potential for getting lost in. The mountain range mainly consists of red sandstone. This means that the geology of the area is treacherous and hard ground.

The summit of Carrauntoohil is located at the following co-ordinates: Lat: 51.99904, Long: -9.74324 (Views, 2024). The task here should be to manufacture a bounding box around this area and calculate the elevation based on the latitude and longitude of the selected point. A bounding box could be broken into several different points to create a 3d space from the given co-ordinates.

This report is going to use the summit of Carrauntoohil as a base point for its pathfinding purposes, as this provides a tall summit and an area of much concern for hikers and Search and Rescue teams alike.

## 3.7 Hardware and Technology

### 3.7.1 Python

The Python Programming Language is a high-level, general purpose, object-oriented programming language that is easy to use and excels at a wide range of things (Python.org, 2022). It can be used for GUI programming, embedded systems and web development and everything in between, as it hosts a wide number of libraries and frameworks for any such purpose. It is easily run on command lines, used through scripts and Notebook type applications, such as Jupyter or Googles Co-Lab service.

It is a language with a considerable skill curve and is used by beginners and experts alike. Being an interpreted language, it is easy to work with, but possibly slower given that the code cannot be pre-compiled. Although it makes up for being slow computationally by being more concise, less verbose and faster to develop with, making it an ideal choice for applications that need to be developed quickly.

### 3.7.2 NumPy

Numerical Python(NumPy) is an open source maths library for the Python programming language. NumPy adds functionality over the base mathematical functions available as part of the main Python package. NumPy contains several advanced methods of working with arrays, fourier transformation and linear algebra.

The NumPy library is faster operationally when used on computations than its contemporary libraries and competition because it uses vectors and parallel processing on a C implementation of arrays to process them up to 50 times faster than standard libraries. (NumPy, 2024)

### 3.7.3 A\* Algorithm

The A\* Algorithm is an Any-Angle pathfinding algorithm that tends towards finding the shortest path between a source node and a destination node, traversing many adjacencies in the process. A\* is regarded as a smart algorithm that uses many factors such as weighting to decide which direction to prioritise in its quest to efficiently find the shortest path between nodes.

The algorithm works by assigning a weighting to different edges and how weighting these edges impacts the ability to reach different nodes. Where multiple choices of nodes are available, the algorithm picks the lowest cost estimate of travel itinerary (Cox, 2024). It can be summed up as trying to find the lowest cost between each node to reach the destination node.

In its implementation, it uses a lot of memory, as it has to store each node in memory as it makes its’ calculations. For this reason, it may be more prudent to research many algorithms of similar type to ascertain which one may be the best for the pathfinding purpose. Another drawback of this algorithm is that it only increments in angles of 45 degrees, meaning that it would have limited purpose here, as while it is easy to implement, it should not provide the optimal route over a 3D space, as it would have to consider more angles (Nash, 2012).

A table of math equations

Description automatically generated with medium confidence

Figure : A\* Algorithm (Nash, 2010)

### 3.7.4 Lazy Theta\* Algorithm

Lazy Theta is a pathfinding algorithm that also uses nodes to gather information relating to the travel cost to other nodes, and the target node does not necessarily have to be adjacent to the current node to be traversed.

The Lazy Theta\* Algorithm is an improvement over the A\* Algorithm. The key difference here is that the Lazy Theta\* algorithm can use any visible vertex, whereas the A\* Algorithm can only use neighbouring vertices (Nash, 2012). This should allow the algorithm to use less vertices in theory over its’ predecessor.

These algorithms could, in theory be used to path find over 3D spaces, as both algorithms have modifications that allow them to work in more complex environments. Lazy Theta should also need less memory, considering it does not need to store as many nodes as its predecessors.

A white sheet with black text

Description automatically generated

Figure : Lazy Theta\* Algorithm

As can be seen from the diagram, the Lazy Theta\* algorithm makes changes to the cost calculation and vertex setting methods to improve the algorithm based on application.

# Chapter 4: Methodology

The research above would help to design and implement a suitable system for the purposes of locating people who went missing on mountains.

## 4.1 Research Questions and Objective

The purpose of the research is to determine what kind of technology would be best suited to the task of mountain navigation, and how to apply with modern technologies.

The objective of the research undertaken for this project is to see if modern navigation techniques and tools can enhance SAR operations. The topic will take a deep dive into traditional and more modern techniques, such as graph algorithms and GPS techniques, and how these techniques may be implemented into a modern application.

## 4.2 Gathering Necessary Data

For the purposes of this report, the data on a selected search area will have to be acquired. The data will involve collecting the Latitudes, Longitudes and Elevations of any given search area. The terrain will then have to be assessed and the appropriate routes and areas gathered.

### 4.2.1 Acquiring Elevation of Selected Search Area

The surrounding area has coordinates that can easily be gathered by using Google Maps and recording the distance away from the summit the search area will have to be defined. When these coordinates are gathered, a bounding box can be created. The bounding box will have minimum and maximum values for longitude and latitude, and these coordinates will be sent to the Google Elevation API to determine the elevation at each individual coordinate.

The area can be broken down by determining the difference between each set of coordinates of the bounding box. This difference can be divided however many times the user requires. For the purposes of designing a sensible search area, a balance must be struck between detail and usability. A very small search area would be far easier to design a set of coordinates for because it should divide into manageable sections very easily due to a reduced number of API calls to Elevation API at the cost of making the search area too small. A very large search area would be prohibitively expensive as it would have to be broken down into many more sets of coordinates.

For the purposes of this report, a search area of 2 kilometres will provide a compromise between detail and usage. This 2km squared search area will then have to be broken down into a smaller grid for use in mapping across three dimensions. If broken down into 100 boxes of 100m2 , the area should be a combination of manageable and detailed, while the calculations should not be prohibitively expensive, hitting the Elevation API 441 times. This falls below Googles maximum number of requests per use, and can be run all at once. If the box was broken down into smaller boxes requiring more points, the API would have to sleep to prevent the API from timing out.

The bounding box has a minimum latitude of 51.99045 and a maximum latitude of 52.00844.

The bounding box has a minimum longitude of -9.7573 and a maximum longitude of -9.728086.

The distance between each of the latitudes and longitudes for coordinates is 2 kilometres, giving a 4-kilometre squared search area around the summit. With this information, the user should be able to use maths to gather a set of points that correspond to points in the bounding box to break areas down in terms of geographical importance. A suitable algorithm is shown below, implemented in the Python programming language and a sample output is shown in the screenshot below. (API Key ommitted for security reasons.)A screen shot of a computer

Description automatically generated

Figure : Producing Elevation from Points

This method returns the elevation at each point that is entered. It is run through a loop, for each of the steps given. The JSON needs to be parsed from the returned request, and it is accessed at various points using the JSON library to access it. The elevation, Latitude and Longitude are all accessed and returned from where they were located in the response.

In the case of the code, it returns elevation for each point, using nested loops of equal value to produce a box of coordinates.

The sets of Longitude and Latitude were then exported to an excel Comma Separated Values(.csv) file for manipulation later.

### 4.2.2 Displaying Plots

When it comes to working with 3D spaces, Python has various libraries available to assist a user with that task. The data can be viewed using a plot from MatPlotLib. The data needs to be extracted from the CSV file from the previous step. This is accomplished using the Pandas library, as it has methods available for reading values from these files, in the form of a DataFrame. The values for Latitude, Longitude and Elevation can then be determined and inserted into a suitable MatPlotLib figure for displaying the data.

## 4.3 Determining the Distance and Points to be Displayed

The summit of Carrauntoohil is disputed by various sources. There were three sources compared for this project, and the highest elevation result was used to determine the summit for the purposes of this project. It should be noted that this cannot be taken as entirely accurate, as sources on the exact coordinate of the summit produces different elevations. Plugging the given sources into a python notebook script, the following sources were compared:

A screenshot of a computer program

Description automatically generated

Figure : Verifying Summit Coordinates

As can be seen from the graphic, the High Road Ireland source was the most accurate, although not very close to the claimed reported elevation of the summit of the mountain, which is 1039 metres. (High Point Ireland, 2024). When clicking on Carrauntoohil in the link provided, the returned coordinates are 51.9991, -9.74308. This returned a value that was 30 metres away from the highest reported value.

### 4.3.1 Determining a Bounding Box using the Haversine Formula

From the coordinates provided above, an arbitrary distance must be decided to gather a suitable bounding box. A bounding box must be calculated here to ascertain the possible area to work with. As referenced in research above, the boundaries of the map can be gotten from using the Haversine Formula.

This section was cheated slightly, as the points gathered were not made by inverting the Haversine formula to gather a point, but was done by guessing the distance based on the returned distance value like so, and making incremental changes based on the result until a distance close to 1km was found: A screen shot of a computer program

Description automatically generated

Figure : Inverse Haversine

# Chapter 5: Design

## 5.1 Design Justification – User Stories

The report will need use cases to ascertain viability for the project in the form of user stories. In cases where people go missing, there is an imperative related to finding the person quickly. In this regard, the program needs justification. For this reason, the following user stories can be implemented with the program.

This report will use a series of User stories to gather the necessary information to tell whether certain features will or will not be implemented. This would provide a method of prioritising features based on the need for said features.

The following section describes the program from the point of view of the users, in the form of epic stories. E.G. each heading represents one epic.

This section was developed in conjunction with JIRA, and links to the completed JIRA page are included down in the second part of the appendix of this paper and the following is a demonstration of the JIRA user stories: A screenshot of a computer

Description automatically generated

Figure : JIRA Board

### 5.1.1 Emergency Rescue Assist

The following user stories will focus on the point of view of rescue services, who have safety related reasons for requiring a route on mountains.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Epic | User Stories | | Priority | Estimate | Acceptance Criteria |
| Epic 1:  Mountain Rescue | User Story 1 | As a Rescue Operative, I want to locate missing persons. | High | 30 hours | Program Locates possible missing person locations. |
| User Story 2 | I want to identify optimal evacuation routes. | High | 25 hours | Program designs an easier route for descending than the ascending route. |
| User Story 3 | I want to design an optimal route. | High | 20 hours | Program Locates the fastest route for ascending. |

### 5.1.2 Outdoor Recreational User

The following section will focus on the point of view of recreational users.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Epic | User Stories | | Priority | Estimate | Acceptance Criteria |
| Epic 2:  Hiker | User Story 1 | As a Hiker, I want to gather a safe route to climb mountains. | High | 10 hours | Program displays the optimal route for ascending the mountain to the user. |
| User Story 2 | I want to identify a trail route for bikes or other vehicles. | Medium | 15 hours | Program designs a separate route that would be more suitable for vehicles. |
| User Story 3 | I want to design a more difficult route around rock climbing. | High | 10 hours | Program locates the route with the most verticality. |

### 5.1.3 Environmental Monitoring

The following epic corresponds to the point of view of ecological surveying and data gatherers. This could be important to the changing scope of the project, as natural disasters could affect terrain and require new approaches to be taken with regards to the changing environment.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Epic | User Stories | | Priority | Estimate | Acceptance Criteria |
| Epic 3:  Environmental Monitor | User Story 1 | As an environmentalist, I want to assess the changing environment. | Low | 5 hours | Program refreshes its’ data set to determine changes in environment. |
| User Story 2 | I want to predict potential natural disasters, such as landslides. | Low | 5 hours | Program identifies weaker geological structures, rock types, etc. |
| User Story 3 | I want to assess changing vegetation over a period. | Low | 10 hours | Program generates information regarding local wildlife and vegetation. |

### 5.1.4 Educators and Trainers

This epic will concern the point of view of educators and trainees who may look to educate people on mountaineering.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Epic | User Stories | | Priority | Estimate | Acceptance Criteria |
| Epic 4:  Educators | User Story 1 | As an educator, I want to simulate climbing the mountain. | Low | 5 hours | Program will generate a 3D simulation of the environment. |
| User Story 2 | I want to demonstrate a virtual mountainous environment. | Low | 5 hours | Program generates a visual representation of the mountain data, with physical detail. |
| User Story 3 | I want to have tools for assessing changes to the environment. | Low | 5 hours | Program generates information regarding changes between previous datasets. |

## 5.2 Prototype

Then, a prototype is required to demonstrate basic functionality of the application. The prototype is not going to be functional and will just be used for the purposes of conveying a specific design.

### 5.2.1 Set up Working Environment

This implementation of the application will use VSCode to gather a basic working environment to use for displaying Python Notebook files. Some plugins will have to be downloaded for this, such as Python for VSCode and Notebook extensions. The working environment will have Python version 3.13.0.

A screenshot of a computer

Description automatically generated

Figure : Work Environment

### 5.2.2 Import the Necessary Libraries into VSCode

The project will make use of several libraries for use in different areas, such as the Requests library from Python source. This library will allow the program to interact with internet endpoints, retrieving data and displaying in JSON format.

The OS library will be used to access API keys, as storing these in plain text would introduce security vulnerabilities. Storing the Google API key as an environment variable would be a more prudent way to do this, and is accessed like this: (API Key value removed for security.) A screenshot of a computer

Description automatically generated

Figure : Environment Variables

After the API key has been stored, it can be used to access the endpoints from Googles elevation API, returning the values for elevation wherever the latitude and longitude are placed in the script.

### 5.2.3 Using Matplotlib to Display the Data

The MatPlotLib Python library has features that enable it to display the data from the CSV file. In this case, Matplotlib will be used to map the 3 data types into a visualisation for easier viewing. Each of the three location(latitude, long and elevation) types need to be taken from the elevationCalls.csv file and added to the scatter() method to view the data: A screen shot of a computer screen

Description automatically generated

Figure : MatPlotLib Code

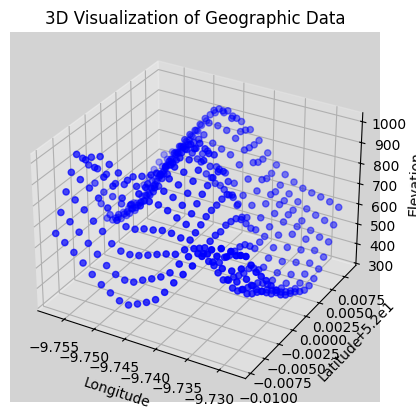
When this is done, the following plot is displayed, which looks roughly like a mountain top: 

Figure : Data Visualisation

### 5.2.4 Finding a Simulated Path to Traverse.

To traverse a sample path, the lowest and highest points on the dataset have to be gathered. The Pandas library for Python will allow for this interaction with the dataset. Calling the maximum value and minimum value for the elevation column will return the points at which the map is at its’ highest and lowest. With this data, we can gather a source and a destination to use for sample path finding. The code and coordinates are ahown below. A computer screen with text and numbers

Description automatically generated

Figure : Lowest and Highest Coordinates

This leaves us with two coordinates, the lowest and highest points in the dataset. When these two points are plugged into the Haversine Formula, the distance between them is roughly 950 meters.

A computer screen shot of a program

Description automatically generated

Figure : Distance Between Lowest and Highest Points

### 5.2.5 Path Between Two Points

Using Pythons Folium (PyPi, 2024) library, a basic visualisation can be drawn on a HTML map. This library provides basic facilities for drawing maps, in this case, the ability to draw a Javascript map, showing the two points with a simple path between them.

A computer screen shot of a program

Description automatically generated

Figure : Folium Code

When this script is run, it produces the following map: A map with a route

Description automatically generated

Figure : Basic Path Between Points

# Chapter 6: Implementation

## 6.0 Sprint Overview

This project will be broken down into six sprints, totalling twelve weeks. This will allow adequate time to implement the main functionality of the project.

|  |  |  |  |
| --- | --- | --- | --- |
| Sprint Iteration | Start Date | End Date | Main Functionality |
| 1 | 24th Jan 2025 | 7th Feb 2025 | Pathfinding Implemented |
| 2 | 7th Feb 2025 | 21st Feb 2025 | Enhance Reusability |
| 3 | 21st Feb 2025 | 7th Mar 2025 | Gather More Data |
| 4 | 7th Mar 2025 | 21st Mar 2025 | Final Data Gathering Sprint |
| 5 | 4th Apr 2025 | 18th Apr 2025 | Implement Algorithms |
| 6 | 18th Apr 2025 | 2nd May 2025 | Finishing Point |

## 6.1 Sprint 1 Timetable Overview

|  |  |  |
| --- | --- | --- |
| Task Number | Description | Status |
| 1 | Create Virtual Environment | Completed |
| 2 | Gather A More Detailed Data Set | Completed |
| 3 | Create Test Path | Completed |
| 4 | Display Updated Map | Completed |

### 6.1.1 Create Virtual Environment

The working environment is going to be a virtual environment using the Python venv library. The following command creates an environment to be used in the program:

‘python venv env’. This command creates the virtual environment folder, in this case, it is called ‘env’.

From there, the environment can be activated in different ways depending on what you are using to interact with it.

* CMD: The environment can be activated using ‘env\Scripts\activate.bat’
* PowerShell: The environment can be activated using ‘env\Scripts\activate.ps1’
* Bash: The environment can be activated using ‘source env/Scripts/activate’

To deactivate the environment, all that needs to be done is running the command ‘deactivate’.

When libraries are imported, the virtual environment stores them in the ‘env/Lib’ folder. Any source code can be checked and inspected from there. A sample algorithm can be found in the NetworkX directory, ‘\env\Lib\site-packages\networkx\algorithms\shortest\_paths’ directory, and it is displayed here:

### 6.1.2 Gather More Detailed Dataset

First, the data gathering process from the section 4.2.1 will have to be refined to produce a more detailed dataset. This can be accomplished by increasing the number of steps to 200. This means that the loop involved will run 40400 times, representing 40000 boxes, each 10 meters apart, and the points connecting these boxes. When run through MatPlotLib, the graph looks like the following: 3D view of Carrauntoohil
Uses the MatPlotLib library to display the plot.

Figure : MatPlotLib Plot with Advanced Data Set

This version produces a much more detailed dataset, at the expense of readability when viewed with MatPlotLib. It is now less clear what is being displayed in the graph.

The first sprint of this project will entail getting the data from the CSV file to display and find a path from the lowest point to the highest. At the current point in time, the CSV file contains Latitude, Longitude and Elevation(X,Z,Y) coordinates that roughly correspond to a 1 kilometre box around the mountain of Carrauntoohil.

### 6.1.3 Application of Pathfinding Algorithms

As of the first sprint, the pathfinding process will be implemented in a basic capacity. Using the SciPy and KDTree libraries, the information can be stored and manipulated accordingly.

SciPy can be used to store the Latitudes, Longitudes and Elevations as a graph, and NetworkX can be used to apply pathfinding algorithms to the graph.

In the case of a basic example, the lowest and highest points in the graph will be found using Python data science libraries. The data from the CSV is stored in a Pandas dataframe and the lowest and highest values found using the min() and max() methods. They are returned like so: A screen shot of a computer code

AI-generated content may be incorrect.

Figure : Returning the Lowest and Highest Values

Then the distance between the two points can be gathered using the Haversine formula. In Python, there is a library for installing Haversine, and it can be used by inserting the coordinates into a haversine() method and specifying a distance unit to return the result.

### 6.1.4 Deciding on Path for Testing

The algorithms need a path to find, therefore an appropriate path needs to be ascertained and traversed via the graph. The start and finish points were chosen arbitrarily, using the lowest and highest points available in the data structure. This presents a unique issue. The lowest point in the set is in the middle of a lake, roughly 335 meters above sea level. Obviously, the intention is not to have the subject start their traversal from the lake as a starting point, and leaves the obvious direction going forward, in a later sprint, to overlay the map with already existing paths to gather possible entry points for the area in question. The lowest and highest points will be used for now, as a placeholder, to test the algorithms over the selected path.

### 6.1.5 Create a Graph of Points

First, the graph needs to be produced and the algorithms applied to that. Application of the Graph and path are done with the NetworkX library, while the SciPy KDTree is used for storing the information in a binary tree and traversing. A screenshot of a computer program

AI-generated content may be incorrect.

Figure : SciPy and NetworkX and KDTree for queries

### 6.1.6 Display Updated Map

Again, using the Folium library, the map can be saved locally as a HTML file. When the file is viewed, a Javascript plugin displays the map. The Javascript library in question uses resources from the OpenStreetMap project. The script uses Folium Map method to design a map and adds the necessary resources to it from there, using several options, such as which marker corresponds to which point on the map. Highest and Lowest, in this case.

A screen shot of a computer code

AI-generated content may be incorrect.

Figure : Folium to Display the Sample Path

When displayed using Folium, the following map is created and stored in the folder in the form of a HTML file, displaying the lowest point marker, located in Lough Gouragh.

A map with a route

AI-generated content may be incorrect.

Figure : Map with Shortest Path Applied

### 6.1.7 Sprint Retrospective

This sprint covered a lot of ground. It implemented pathfinding on a basic level, and covered some aspects of reusability for the project. If the aim is to have end users gather data on their own selected areas to find, there is a good starting point here.

Going forward, the next sprints are going to have to make the program more reusable and allow for more direct actions from the user. Reusability will have to take more focus going forward, as gathering their own bounding boxes and points of interest would be too much of a barrier to entry for most people. The application has the potential to operate with only a single point being inserted by the user and from there, the application could generate it’s own paths based on that data.

## 6.2 Sprint 2 Timetable Overview

|  |  |  |
| --- | --- | --- |
| Task Number | Description | Status |
| 1 | Refine Data Gathering Script | Completed |
| 2 | Gather GeoJSON Data for Existing Paths | Completed |
| 3 | Overlay Paths on Map | Completed |
| 4 | Display Updated Map | Completed |

### 6.2.1 Refine Data Gathering Script

The script used to gather the data set for the plot could be reused by potential users. To this end, a new file will be created, containing methods for taking in a point and generating a bounding box based on the given point. The idea is to produce a script that would allow the user to insert a single point that would produce the necessary elevation data using only two inputs, Longitude and Latitude. It uses the same basic idea as before, with refactored code to make the process easier. It produces a CSV file with the elevations added to the data set.

The script has been broken down into functions based on their usage. The first set of methods concern validation of the given inputs, decimal points with roughly 6 points of precision. This means that standard number validation methods such as is\_numeric() won’t work for this purpose.

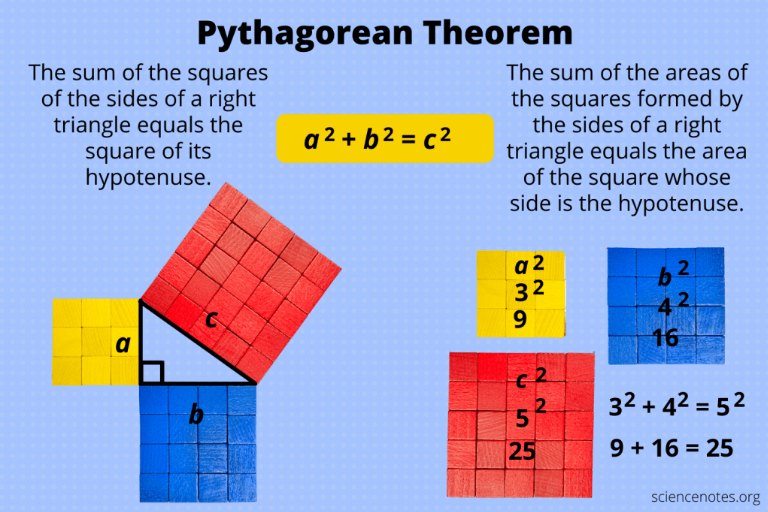
Then the valid data inputted into the script gets fed through a for loop that uses the inverse\_haversine() method to return a distance away from the points in each diagonal direction. The size of the bounding box is hard-coded to 1 kilometre for cost-saving purposes, meaning that the inverse\_haversine method will always be taking in the square root of two as its distance parameter. The Pythagorean Theorem was used to calculate this distance. Each side is 1, meaning that the diagonal distance will always be root 2. Proof of the theorem can be found below: 

Figure : Pythagoras Theorem

The finished method produces a bounding box using the points that exist on the northwestern, northeastern, southwestern and southeastern points of the map using a loop as seen below: A screen shot of a computer code

AI-generated content may be incorrect.

Figure : Using Inverse\_Haversine()

When this step is complete, the data can be broken into a 2D array and exported to a CSV file. This process is mostly a retread of the data gathering method, but with this script, the data gets written automatically, instead of having to take the data from the Jupyter Notebooks output directly from the text editor. It does this using Pythons CSV library. The elevation data gets stored in an empty list, and for each iteration of the loop, adds the list item to a file called “custom\_data\_set.csv”. A sample of the scripts output can be seen here, with longitude and latitude points corresponding to the summit of Mount Everest: A screen shot of a computer program

AI-generated content may be incorrect.

Figure : Mt. Everest Elevation Data Sample

### 6.2.2 Gather GeoJson Data for Existing Paths

There are several already existing paths to reach the top of the mountain. They exist in different flavours that depend on each subjects use case. The easiest way to assess these routes would be to start at a common point, and all routes start from the same position. The starting area is a local area called “Cronin’s Yard”. This point is outside the data set, so the problem now becomes a matter of combining a common entry point into the already produced data.

A script can be made for that using the Folium library and OpenStreetMaps. Using the osmnx library, a map of the paths can be exported in GeoJson format.

### 6.2.3 Overlay Bounding Box on Map

The Folium library will have to be updated from past iterations to display the new data that has been gathered. This is a relatively simple process, involving gathering the borders of the map generated by the CSV file. This can be done using the haversine method taken from the data gathering script. Foliums PolyLine() method can be used to display the points, because there is no pre-existing method for drawing a box over a map. This can be done using the inverse\_haversine method, although the implementation from the script cannot be reused for this, as it would have an incompatible return type for the location parameter of the PolyLine() method, so each border would have to be gathered manually. Here is the implementation: A screen shot of a computer program

AI-generated content may be incorrect.

Figure : Gathering the Border Data

### 6.2.4 Display Updated Map

The newly gathered information can then be inserted into map via the Folium PolyLine() method like so: A screen shot of a computer program

AI-generated content may be incorrect.

Figure : Using Folium to Display the Borders

This updates the map to display more relevant information, including the placeholder path from before, all the borders of the data set and the pre-existing paths gathered from the GeoJson data. This makes assessment of the algorithm performance much easier. Local landmarks can now be viewed together with the path data, and viewers should be able to see the starting points and end points roughly now. This is the completed map, with Cronin’s Yard being the point at the top right where the path begins: A map with a black square

AI-generated content may be incorrect.

Figure : Map with Bounding Box and Paths

### 6.2.5 Sprint Retrospective

This sprint went to plan, and it supplemented the data set and ran some tests that verified the processes already used. The map created by OpenStreetMaps library produced a list of the paths in the surrounding area to be integrated into the path finding process. This provided the project with much needed data that would allow for the comparison to be made between existing paths and the ones that will be created in the path finding process.

At the point here, it was considered that maybe sticking to Jupyter Notebook files alone would not be a great method of dealing with data. The functions are all there, but this is an approach that does not lend itself too well with reusability. At this point, the decision should probably be made to put more code in functions, enhancing the code functions and allowing for easier script integration.

## 6.3 Sprint 3 Timetable Overview

|  |  |  |
| --- | --- | --- |
| Task Number | Description | Status |
| 1 | Transfer GeoJSON to Elevation Data | Completed |
| 2 | Add Path Values to Data Set | Completed |

### 6.3.1 Transfer GeoJSON to Elevation Data

All the coordinates thus far have been listed as Latitude, then Longitude, and there is no real universal standard. This means that various data sources may have these attributes in different orders. The GeoJSON described above, for example, is listed as Longitude and Latitude, as opposed to the standard that this project uses. This also must be accounted for.

The reason for this discrepancy is that Google takes coordinates in the “Latitude, Longitude” and GeoJSON stores its coordinates the other way around. This led to the creation of a dataset for the paths being made in the Indian Ocean, near Madagascar. Here is what that coordinate looks like on Google Maps: A screenshot of a computer

AI-generated content may be incorrect.

Figure : Madagascar Coordinates

This step of the project ran into numerous problems, as extracting the path data from the GeoJSON file turned out to be more difficult than expected. The first problem associated with this is extracting the data. This could not be done with a basic script, as the LineString data structure that holds the data is held at varying depths of the JSON structure, meaning that no one approach of assessing the JSON will return the path data for all the paths. This problem will have to be rectified by finding a different library or approach to extracting the necessary values.

The third problem is producing these paths from the OpenStreetMap data set only produces paths in the given area where it can reach the start and end of a path. What this means from a practical perspective is that you need to cover a much wider area that the bounding box described above, as it will not return all available paths within the 1 Kilometre radius, as they don’t start within the box. This was adjusted by increasing the distance of the check to 3,500 metres instead of 1 Kilometre.

This leads to problems with programmatic variables, as it means that no one approach will work for every area that might be considered to gather data on. The path data for something like Mount Everest, for example, is going to look very different than it would here. Producing a set regarding this area would have to be handled differently, as all the paths might not be returned with this approach.

This problem was eventually solved using the GeoPandas library to parse the data. GeoPandas has methods for assessing the individual Geometry features returned into the file. It is assessed like so: A screen shot of a computer program

AI-generated content may be incorrect.

Figure : Extract Path Data

Using Shapely, GeoPandas can store various geometry features as attributes that can be more easily assessed by using simple loops.

The geom\_type being searched for is a LineString, the type that is returned as a path in a GeoJSON file.

The sorting function here deserves special attention, as there is a lot going on in this line in particular. It is sorting a list, converting it to a set, just for the purpose of removing duplicates, then returning it as a list at the end. It sorts by the first value, the Latitude, then Longitude and returns a list for easier access.

So what is returned here afterwards, is a list of coordinates in the same format that the carrauntoohil\_data\_set.csv file was before adding the elevations to it.

### 6.3.2 Add Path Values to Data Set

The values taken above from the paths can have the same elevation gathering process applied to it, making it take the same format as the Carrauntoohil\_data\_set.csv file. This process will allow for consolidating the data into a single Graph using a simplified version of the elevation gathering method described in the steps located in above sprints. When run through the method, it produces a similar CSV file called ‘path\_terrain\_set.csv’.

This produces a nice data set that should have all the necessary values for traversal of the data set concerning the mountain. From here, it should be possible to implement the pathfinding methods and see if any improvements can be made over pre-existing paths, or whether new approaches are needed.

### 6.3.3 Sprint Retrospective

This sprint did not quite go to plan, as the parsing of GeoJSON data into a more usable format was fraught with issues. Deciding the best approach for data formats here was a time sink that involved spending a lot of time on gathering the wrong data, as it was wrongly assumed that the GeoJSON data would be in roughly the same format as the rest of the sets, as standard convention dictates that it should have been. Taking the wrong approach here led to this sprint taking longer than expected and could have been prevented with more research, although this area is quite niche and may not have been uncovered even by experienced developers.

## 6.4 Sprint 4 Timetable Overview

|  |  |  |
| --- | --- | --- |
| Task Number | Description | Status |
| 1 | Consolidate Data into One Graph | Completed |
| 2 | Create MatPlotLib Graphical Representation | Completed |

### 6.4.1 Consolidate Data into One Graph

The data gathered above could be refactored into a single Graph using the NetworkX and SciPy.Spatial libraries. This would allow traversal between all nodes available in the Graph and allow for more mathematical calculations made between them. The NetworkX library has functions available for offloading a Graph object into a file for parsing as Extensible Markup Language (XML). This means that any Graph can be created and destroyed as necessary, and provides hard storage for the Graph data.

### 6.4.2 Create a MatPlotLib Graphical Representation

The MatPlotLib library can be used here to visualise the data of both data sets. This can be done by using a surface plot, with the y-axis being used to represent the Elevation column, and x and z-axes are representing the latitudinal and longitudinal values.

Note that the trailing set of nodes are coming from outside the main plot, which is why they appear to be floating in mid-air. This is what is supposed to be output from the plot. Overall, this would appear to be correct as for this point in the project.

### 6.4.3 Sprint Retrospective

This sprint allowed the progress made so far to be visualised and interpreted. The best form of testing is a visual test for a project like this. The aspect of

The finished plot can be viewed here: A graph with a colorful graph

AI-generated content may be incorrect.

Figure : View of Both Graphs

This plot contains all the gathered path data from the script, with the elevations displayed and the paths interspersed within the same plot. This gives the area a more detailed feature set, and allows for easier traversal of the area.

## 6.5 Sprint 5 Timetable Overview

|  |  |  |
| --- | --- | --- |
| Task Number | Details | Status |
| 1 | Design a Cost Function | Done |
| 2 | Implement Custom Pathfinding Algorithm | Done |

### 6.5.1 Design a Cost Function

In this case, a sharp elevation should be penalised as the goal is to traverse the environment easily. In this case, the elevation difference is used to calculate the slope between source and destination, with a penalty of fifty percent for every ten percent slope. Here, the slope can be negative or positive.

The slope\_percent variable is calculated by converting the distance to Kilometres and multiplying it by one-hundred. The power operator here allows the cost to exponentially increase in proportion to the increase in the slope.

A small downward slope here is considered beneficial, as a small decline is easier to traverse than an incline. As the value of decline increases, so does the penalty. The steep\_factor variable is a normalised variable that can range in value between zero and one. When a lower value is received in this variable, the output will be a value relating to the percentage of the slope.

|  |  |  |
| --- | --- | --- |
| steep\_factor value | Slope | Difficulty |
| -0.05 | -5% | 13% Easier |
| -0.1 | -10% | 3% Easier |
| -0.2 | -20% | 23% Harder |
| -0.3 | -30% | 50% Harder |

Here is the completed cost function: A computer screen shot of a code

AI-generated content may be incorrect.

Figure : Cost Function

### 6.5.2 Implement Custom Pathfinding Algorithm

This step involves the dissection of the A\* algorithm and implementing a custom version of it that penalises major changes in elevation. This is going to be based on the NetworkX implementation of the algorithm (NetworkX, 2024). This means that it will need to mimic some of the behaviours of the original, with added functionality allowing for assessing the elevation of the nodes within the Graph. The first step here is to define the method and its parameters. It will need to take in a NetworkX Graph, a source and destination, and a weighting for the elevation. If successful, it should return a series of points that ends at the destination point.

The heuristic parameter in the original library corresponds to the haversine distance between two points. There is already a library for returning this value, in the form of the Haversine library (PyPi, 2024). This function returns the Great-Circle distance between two points and serves as the heuristic in this case. This is implemented in the form of importing the library and returning the distance between the source and destination.

Breaking down the method, the first thing that needs to be done is validating the input. This can be done with a simple check if the source node and destination node are in the Graph. If not, then return a ValueError message.

This algorithm will continually assess the nodes it reaches until one of two conditions are met. Either the destination is reached, or the nodes are fully explored. If no path is found, the resulting value that will be returned is none.

The algorithm, if designed correctly, should take in parameters in the same way the NetworkX path algorithms do, with a source and destination Latitude and Longitude, a Graph object, and an elevation weighting that will allow the relative penalties to be applied.

The implementation above allows for easy comparisons with similar algorithms, making testing an easier process and ensuring that it works the same way as the others, abstracting away the function itself, and the costs associated with it.

### 6.5.3 Sprint Retrospective

This sprint has successfully implemented an algorithm that mimics the usage of the A\* algorithm with a cost function. The resulting path generated by this algorithm will ensure that the more steep path options are avoided for any potential climbers, and traversal is made easier for any who attempt to use it.

The algorithm appears to work to a more optimal degree than the previously tested A\* method, and the path is longer to ascend the mountain than it is with the A\* algorithm. A sample of the algorithms output can be seen below. A screenshot of a computer program

AI-generated content may be incorrect.

Figure : Sample Algorithm Output

# Chapter 7: Project Retrospective / Possible Future Implementations

## Chapter 7.1 Technology

In future implementations of the project, better technology could definitely be leveraged to gather a better data set. There exist programs such as QGIS and ArcGIS that could provide better forms of map for the information gathering process, but they were discovered too late to really be implemented within the bounds of this project.

Other types of data such as land cover and general precipitation maps could enhance the path finding process, and may be implemented in the future in a more live service environment that could give real-time data on the conditions surrounding the mountain. As these conditions often change, this real-time data could give users the necessary information as to whether the route they are choosing is appropriate for their application. Adverse weather may affect the users choice of route. But that would be more appropriate for a satellite navigation system, although maybe future iterations of this project may go in that direction.

## Chapter 7.2 Funding and Resources

One of the main problems that kept coming up was the sourcing of good data sets. Most of the relevant sets were held behind paywalls and strictly controlled. Accessing these sets could have greatly improved the project data set and allowed for more in the way of assessment over the covered area.

Accessing real-time data would also have a cost associated with it. This data would enhance the project, but the funding was limited, therefore accessing this data was unfeasible for this implementation.

# Chapter 8: Results

This report has shown that it is feasible to take rough metrics from an area, transform it into a viable data set and gather data on traversing the area with programmatic methods such as Graphs and pathfinding algorithms. The area of assessment is probably the most interesting part of the process, and the ability to create a data set from very small input is impressive. The user needs to insert a Latitude and Longitude into the program and data sets are produced for the given area.

The main challenge with assessing the results of a project like this is subjectivity. The effectiveness of any given path finding algorithm is ultimately up to the end user, as all people are different. This means that the optimal route will always be regarded as different. A professional climber may search for a harder route as they are looking for a challenge, and an elderly climber may prefer an easier route.

Because the cost function in the algorithm is averse to higher elevations, this algorithm, when tested, should produce a higher number of nodes when traversing from the source at the bottom of the graph to the top. This MatPlotLib plot shows a theoretical path returned by this algorithm, with a weighting factor of 3.0: A graph showing a red line and a green star

AI-generated content may be incorrect.

Figure : Ariel View of Path

In comparison, when the weighting factor is set to 0.0, in other words, the algorithm behaves the same way that A\* does, it produces the following, slightly shorter path: A hand pointing to a graph

AI-generated content may be incorrect.

Figure : Ariel View with Weighting

Looking more in-depth at the results, the paths come back with a different number of nodes traversed depending on the weight factor. The following source and destination points are true for both algorithms.

Method\_name(G, source=(51.99895007, -9.74338607), target=(52.025803, -9.7086641))

|  |  |  |
| --- | --- | --- |
| Algorithm | Number of Nodes Traversed | Algorithm Weighting |
| A\* | 215 | Elevation |
| Shortest Path (Dijkstras) | 215 | N/A |
| Elevation Aware Pathfinding | 215 | 0.0 |
| Elevation Aware Pathfinding | 255 | 1.0 |
| Elevation Aware Pathfinding | 276 | 2.0 |
| Elevation Aware Pathfinding | 425 | 3.0 |

This table shows the results of testing the algorithm on a small scale. The algorithm would appear to perform as intended, using many more steps when the elevation requirement is increased, meaning that the algorithm is searching for paths that are less steep on a vertical incline.

With regards to testing these results, the best method would be physically, as the area is best assessed in person. Methods for assessing physical strain could be utilised, and could be tested by relevant people, such as climbers and other recreational users.

# Chapter 9: Conclusions

The project has shown that it is indeed viable to gather data on a given area and transform it into a traversable path. Basic methods can be used with varying degrees of success, such as A\* and shortest path algorithm. The data gathering process on display here should work in a variety of environments and should be suited to any rough terrain where elevation is a concern. When applied to the mountain Carrauntoohil, it has produced paths that are roughly equal to the already existing paths, and when an algorithm considering elevation differences was applied, it made the necessary changes to the path, allowing for easier traversal of the terrain.

This project has produced a framework for the generation and assessment of this terrain, and should allow future iterations of this software to build on and improve the already existing processes for even greater terrain navigability going forward.

The most important aspect of this project, up to this point, is probably the automation factor. It should be considered impressive that through abstraction and external libraries, this entire process can be run in just a few scripts. What is as impressive here, is what is kept hidden even from the developer. The use of libraries has contributed to visualising the data and assessing it in a manner that is much easier to convey to any possible end user.

# Appendix – Glossary

This section contains some commonly used terms and concepts with regards to this report.

API – Application Programming Interface – Describes how parts of a computer program interact with each other.

LiDar – LIght Detection And Ranging – Technique that uses lasers to gather distance information.

SAR – Search And Rescue – Term describing search and rescue operations.

GPS – Global Positioning System – Satellite system for gathering coordinate values.

HTML – HyperText Markup Language – A language for displaying information to web pages.

Lidar – LIght Detection And Ranging – Laser technology used to gather distances.

IMO – International Maritime Organization.

ICAO – International Civil Aviation Organization.

Subject – In this report, subject will refer to end user or customer.

Summit – The highest point of elevation in relation to any adjacent points.

UAV – Unmanned Ariel Vehicle, otherwise known as a drone.

XML – Extensible Markup Language – XML is a document format that is designed to be readable by machines and humans. Similar to HTML, but does not use pre-defined tags.

# Appendix – External Links and Information

Github Repository – The public Github repository for this project up to chapter 6 can be found at the following address:

<https://github.com/DogPope/FinalYearProject.git>

It was then split into another repository for the application development cycle and another Github repository created for the final product:

<https://github.com/DogPope/FYP-Development.git>

# References

3DSourced, 2023. *Photogrammetry Guide 2023 – Definition, Advantages and Uses Explained.* [Online]   
Available at: https://www.3dsourced.com/guides/photogrammetry-guide/  
[Accessed 23 October 2024].

Aerospace, 2024. *A brief history of GPS.* [Online]   
Available at: https://aerospace.org/article/brief-history-gps  
[Accessed 05 October 2024].

Agata Kokocińska-Kusiak, M. W. M. Z. J. M. K. B. M. D., 2021. *Canine Olfaction: Physiology, Behavior, and Possibilities for Practical Applications.* [Online]   
Available at: https://pmc.ncbi.nlm.nih.gov/articles/PMC8388720/#sec2-animals-11-02463  
[Accessed 12 October 2024].

Alex Nash, S. K. C. T., 2010. *Lazy Theta\*: Any-Angle Path Planning and Path Length Analysis in 3D.* [Online]   
Available at: http://idm-lab.org/bib/abstracts/papers/aaai10b.pdf  
[Accessed 27 November 2024].

Cox, G., 2024. *A\* Pathfinding Algorithm.* [Online]   
Available at: https://www.baeldung.com/cs/a-star-algorithm  
[Accessed 1 November 2024].

Google, 2024. *Google Elevation API Documentation.* [Online]   
Available at: https://developers.google.com/maps/documentation/elevation/start  
[Accessed 06 October 2024].

Gov.ie, 2019. *National Search and Rescue Plan.* [Online]   
Available at: https://www.gov.ie/pdf/?file=https://assets.gov.ie/22353/7aec819c614e4b9ba948fc06b52801d9.pdf#page=null  
[Accessed 10 October 2024].

High Point Ireland, 2024. *High Point Ireland - Carrountoohil.* [Online]   
Available at: https://www.highpointireland.com/major-island-high-points.html  
[Accessed 23 November 2024].

Irish Examiner, 2022. *Tributes paid to Tipperary man who died on Carrauntoohill in Kerry.* [Online]   
Available at: https://www.irishexaminer.com/news/munster/arid-40887185.html  
[Accessed 17 October 2024].

Kerry Mountain Rescue, N/A. *Carrauntoohil Route Descriptions.* [Online]   
Available at: https://kerrymountainrescue.ie/carrauntoohil-route-descriptions  
[Accessed 10 October 2024].

Kettle, S., 2017. *Esri Community.* [Online]   
Available at: https://www.math.ksu.edu/~dbski/writings/haversine.pdf  
[Accessed 27 November 2024].

Liyang Xiong, S. L. G. T. J. S., 2022. *Geomorphometry and terrain analysis: data, methods, platforms and applications.* [Online]   
Available at: https://www.sciencedirect.com/science/article/pii/S0012825222002756  
[Accessed 08 October 2024].

Logsdon, T. S., 2022. *Britannica GPS.* [Online]   
Available at: https://www.britannica.com/technology/GPS  
[Accessed 06 October 2024].

Martinez, R., 2019. *Artificial Intelligence Types and Definitions.* [Online]   
Available at: https://scholars.law.unlv.edu/cgi/viewcontent.cgi?article=1799&context=nlj  
[Accessed 12 November 2024].

Nash, A., 2012. *IDM-Lab.* [Online]   
Available at: http://idm-lab.org/bib/abstracts/papers/dissertation-nash.pdf  
[Accessed 14 November 2024].

NetworkX, 2024. *NetworkX Algorithm Shortest Path.* [Online]   
Available at: https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.shortest\_paths.astar.astar\_path.html  
[Accessed 25 April 2025].

Norman, J., 2024. *Perhaps the Oldest Map in the World.* [Online]   
Available at: https://www.historyofinformation.com/detail.php?id=4  
[Accessed 01 December 2024].

NumPy, 2024. *Why is NumPy so Fast?.* [Online]   
Available at: https://numpy.org/doc/stable/user/whatisnumpy.html  
[Accessed 1 November 2024].

Pajaziti Arbnor, B. X. F. A., 2024. *Sciendo, Journal of Mechanical Engineering.* [Online]   
Available at: https://sciendo.com/article/10.2478/scjme-2024-0011  
[Accessed 04 October 2024].

PyPi, 2024. *Haversine Python Library.* [Online]   
Available at: https://pypi.org/project/haversine/  
[Accessed 1 December 2024].

PyPi, 2024. *Make beautiful maps with Leaflet.js & Python.* [Online]   
Available at: https://pypi.org/project/folium/  
[Accessed 5 December 2024].

Python.org, 2022. *Python Overview.* [Online]   
Available at: https://wiki.python.org/moin/BeginnersGuide/Overview  
[Accessed 18 October 2024].

SPH Engineering, 2017. *Photogrammetry Technique for UAV Land Surveying Missions.* [Online]   
Available at: https://robohub.org/ugcs-photogrammetry-technique-for-uav-land-surveying-missions/  
[Accessed 17 November 2024].

Thomas H. Cormen, C. E. L. R. L. R. a. C. S., 2009. *Introduction to Algorithms.* 3rd ed. Cambridge: MIT Press.

Views, M., 2024. *Macgillycuddy Reeks.* [Online]   
Available at: https://mountainviews.ie/summit/1/  
[Accessed 29 October 2024].

Yan, Y., 2022. *Research Gate.* [Online]   
Available at: https://www.researchgate.net/publication/376719056\_Research\_on\_optimal\_path\_planning\_technology\_for\_vehicle\_positioning\_and\_navigation\_system  
[Accessed 6 October 2024].

You Li, J. I.-G., 2020. *IEEEExplore.* [Online]   
Available at: https://ieeexplore.ieee.org/document/9127855  
[Accessed 2024 November 23].