HIGH SCHOOL EDITION
Journal of Student Research
Volume 13 Issue 1 (2024)

An Automated Wayfinding Algorithm to Find Rovers' Path on the Lunar South Pole

Piyali Bhattacharya[1], Shaayan Bhattacharya[2] and Robert Chin[#]

[1]Liberal Arts and Science Academy, USA
[2]Liberal Arts and Science Academy, Austin, TX, USA
[#]Advisor

ABSTRACT

As a part of the upcoming Artemis mission, NASA released a large satellite dataset (latitude, longitude, height, and slope) of the lunar south pole, around the area of the lunar module landing site and rover exploration site near the Shackleton Crater. While designing the rover, it's important to understand the minimum power the rover needs (the amount of slope it can overcome) to navigate the terrain. Our goal was to process the data, visualize the data in 2-D and 3-D, and develop an algorithm that would automatically map the shortest possible path (or no possible path) between two coordinates according to the lunar terrain and rover parameters. To accomplish that, we processed a large unorganized dataset into an equally spaced and sequential matrix of latitude, longitude, height, and slope. The matrix was then converted into a go/no-go maze of the terrain and the Breadth First Search (BSF) and A* algorithms were applied to the maze to find the optimal path. The terrain and the optimal path were plotted in 2-D using Matplotlib and interactively displayed in 3D using Plotly. We developed a Jupyter Notebook web-based app that will allow a user to interactively use the algorithm and visualize the lunar terrain. The results identify the minimum rover slope and various paths depending on higher rover slopes. The algorithm and the rover paths were validated by the NASA SCaN (Satellite Communication and Navigation) team of engineers and the application can be used for design, validation, and training purposes.

Introduction

As a part of the upcoming Artemis mission to the moon, NASA released a large satellite data set (latitude, longitude, height, and slope) of the lunar south pole, around the area of the lunar module landing site and rover exploration site near the Shackleton Crater to explore possible ice. While designing the rover, it is important to understand the minimum power the rover needs (the amount of slope it can overcome) to navigate the terrain. A heavier rover with higher power can increase the payload cost significantly. We wanted to develop an algorithm that would automatically find and map the shortest possible path (or no possible path) between two coordinates given the slope the rover can overcome.

Project Goals

Our hypothesis was, that given lunar terrain data and the slope the rover can overcome, it is possible to develop an automated algorithm to find the shortest rover path between a start and destination coordinates. The engineering goals were:

1. Process the large lunar terrain unorganized data file to create a reduced sequential and equally spaced matrix that is representative of the lunar terrain.
2. Develop an algorithm to automatically identify the optimal path (or no possible path) between the two coordinates using the reduced matrix.

3. Analyze the efficiency and effectiveness of the algorithms. Analyze the results to draw conclusions between the different paths.
4. Visualize the lunar terrain latitude and longitude against the rover parameter (slopes) in 2D and 3D plots.

The Lunar Terrain Dataset

The dataset for the lunar south pole terrain, captured by a satellite orbiting the moon, was provided by NASA. It was in the form of a CSV (comma-separated values) file with approximately 1.5 million lines of data. The data was non-sequential, not equally spaced, and unorganized due to the way the satellite scans the data. The latitude and the longitude data were in spherical coordinates. The height was meters above or below the lunar radius (1737400 meters).

Table 1. Lunar dataset provided by NASA.

Latitude (degree)	Longitude (degree)	Height (m)	Slope
-89.22993678059875	-30.9805884292136	-1345.5	24
-89.23106736255683	-31.03118313173637	-1338.5	26
--1.5 million rows of data --			
-88.35705884322554	128.4478820065164	-1091.5	24
-88.35623845806803	128.4838896893954	-1092.5	24

Spherical Versus Cartesian Coordinates

The lunar dataset was a square area around the lunar south pole (Figure 1A). It contained the coordinates of the Artemis landing site and possible exploration site of the Shackleton crater. The data was in spherical coordinates that needed to be converted to cartesian coordinates for plotting using Matplotlib and Plotly libraries. The formula for conversions between the spherical (latitude, longitude, and radius) and the cartesian coordinates (x, y, and z) were used, as shown in Figure 1C.

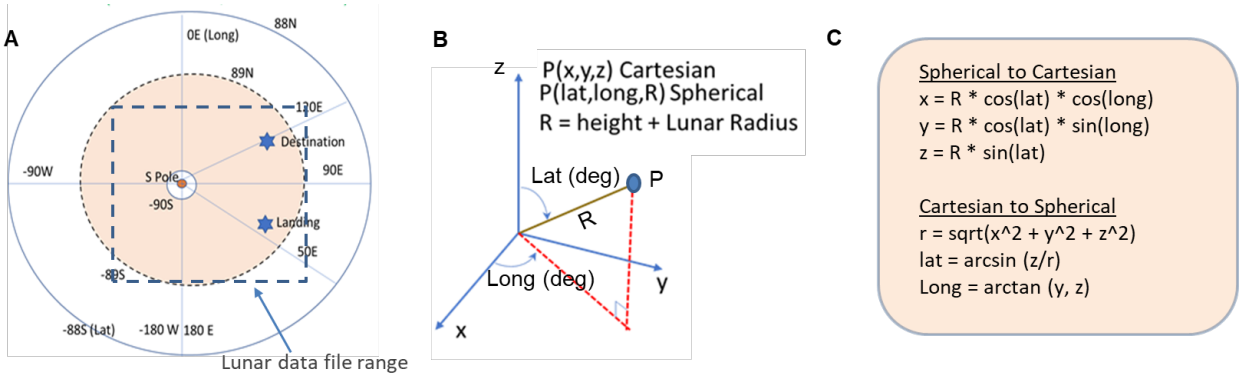

Figure 1. A) Lunar data file coordinates of the south pole. B) Spherical versus cartesian coordinates C) Coordinate conversion

Processing the Lunar Data

The large dataset (~1.5 million rows) was unorganized, non-sequential, and took a long time to process using Python. The maze-solving algorithm requires a sequential and equally spaced matrix, and the Plotly library for 3D rendering of the data could only take a maximum of 100K data points. So, the goal was to reduce the size of the dataset and make it a sequential and equally spaced 300x300 (less than 100K) matrix. To achieve that, a smaller area around the landing and destination site can be processed from the larger dataset. The amount of data points can be further reduced by interpolating the data and reducing the data to a smaller matrix of sequential latitude and longitude.

Maze Solving Algorithms

Once the data is processed, standard maze-solving algorithm libraries can be used to find the shortest path between any start and end point inside the matrix. Both the Breadth First Search (BFS) and A* algorithms are commonly used to find the shortest path between start and end points in a maze. Breadth First Search (BFS) creates nodes and traverses all paths simultaneously, finding the shortest possible path. The A* algorithm is an informed (directional) algorithm that always moves towards the target. Typically, A* is faster, but it depends on the maze.

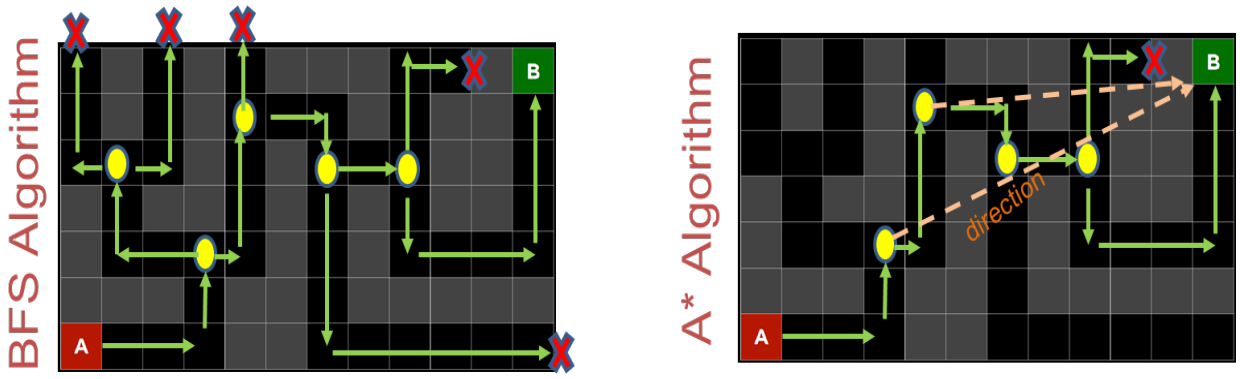

Figure 2. BFS versus A* Algorithm that finds the shortest path between a start (A) and end (B) point. The green arrows show the path it traverses to go to the destination.

Methods

Project Design

The algorithm was developed using Python with standard BSF / A* search libraries. Matplotlib library was used for 2D graphs and Plotly library was used for 3D rendering of the terrain. Jupyter Notebook was used to develop the interactive web-based app.

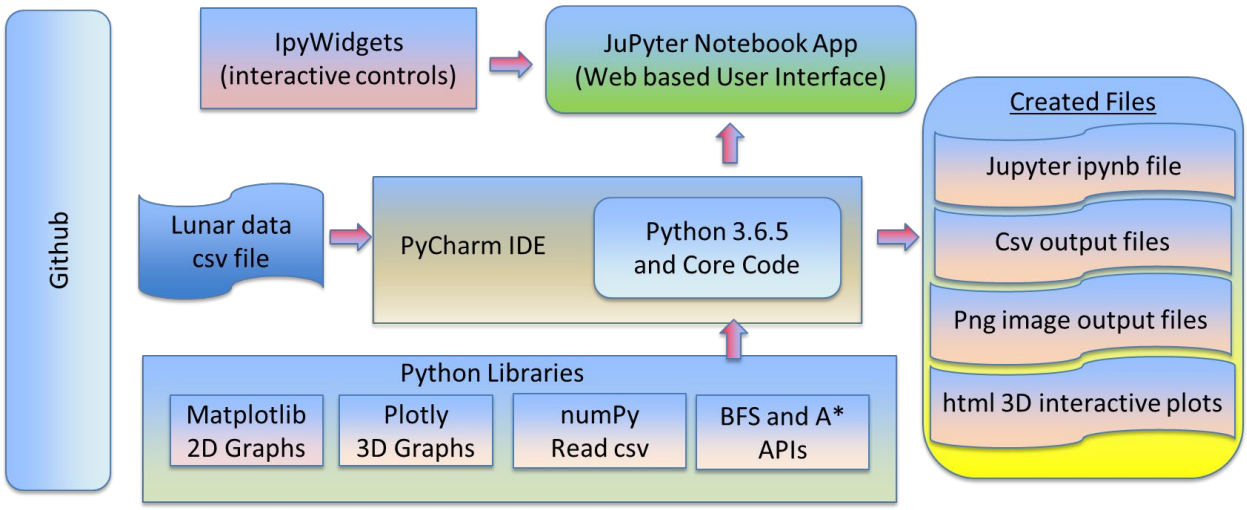

Figure 3. Project design. Source Code:https://github.com/1piyalib/rover_pathfinder

Development Steps

The following three steps describe the program. To start, a reduced sequential matrix was processed from the large lunar data file. Pathfinding algorithms of BFS and A* were applied to the reduced matrix that created the list of rover path coordinates. The matrix and the rover path were overlaid in the graphing tools to generate 2D and 3D images.

Figure 4. Three steps for program development

Step 1: Create Reduced Matrix Files

The first step was to create a sequential, equally spaced, and reduced matrix file from the large lunar data file that can be used for plotting and the BFS/A* algorithms. An area around the landing and destination site (latitude of -89.3 to -88.7 and longitude of 50 to 150) was selected that reduced the matrix size to ~450K rows from ~1.5 million rows of the original lunar data file. An equally spaced 300x300 latitude and longitude list was created from the reduced matrix. For each latitude and longitude pair in the 300x300 matrix, the closest data point in the large lunar data file was selected by using the distance formula between two coordinates. This process reduced the large unorganized lunar data file to a representative 300x300 (90K) equally spaced maze that can be used for the maze-solving algorithm.

The original file was also converted from spherical to cartesian coordinates and was reduced to a 300x300 cartesian matrix. This second matrix could be used for plotting the terrain. The 90K data points were selected because that is the maximum amount of data that Plotly libraries can use for 3D rendering.

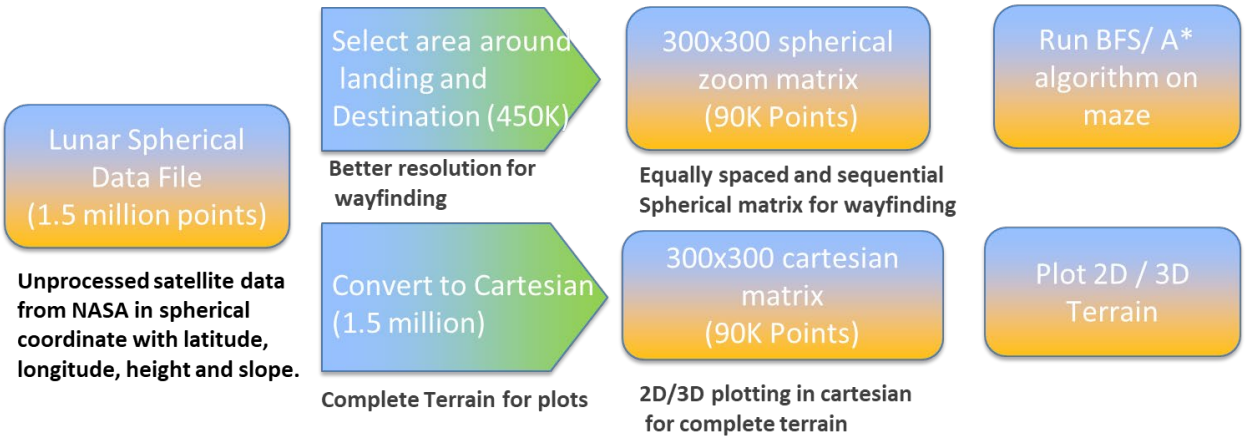

Figure 5. Create a reduced matrix file

Step 2: Run BFS and A to Find Rover Path*

The 300x300 matrix created in Step 1 can be converted into a two-dimensional matrix of go (1) and no-go (0) cells. The slope the rover can overcome is used as an input. Any cell with a terrain slope greater than the rover slope (15) is converted into a no-go cell, otherwise it's assigned as a go cell. The Breadth First Search (BSF) and the A* algorithm library take the index of start and end coordinates with the go/no-go matrix. The algorithm finds the shortest path (or no possible path) between the start and the end and returns the indexes of the possible rover path. The indexes are then converted to corresponding latitude and longitude and then converted to cartesian coordinates for plotting.

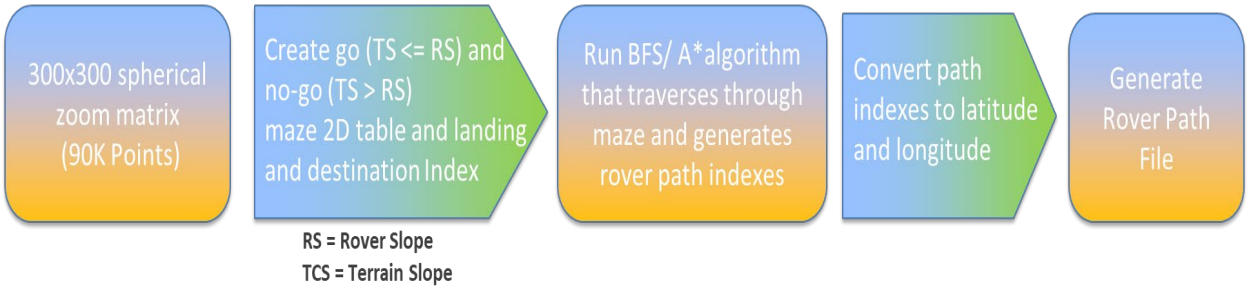

Figure 6. Run BFS / A* Algorithm

Step 3 Generate 2D and 3D Terrain Graphs with Rover Path

The terrain 300x300 cartesian matrix file from step 1 is plotted using Matplotlib scatter plot (2D visualization) and Plotly (3D rendering) to show the lunar terrain graphically. The rover's cartesian coordinates are superimposed on the plots to show the rover's path on the lunar terrain.

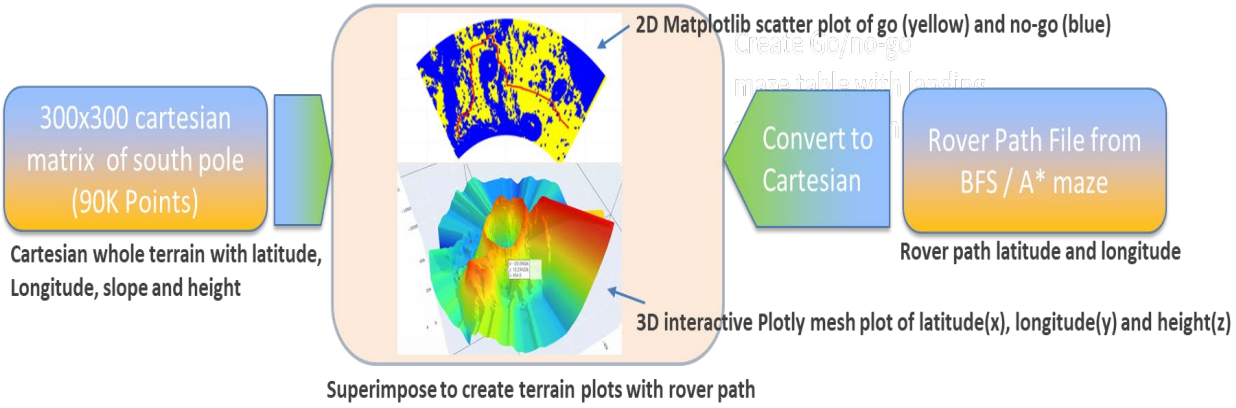

Figure 7. Generate graphs.

Results

Data Integrity

The reduced matrix files have a lower resolution but are a good representation of the large lunar data file for wayfinding and plotting. The scatter plots in Figure 8 have x as latitude, y as longitude, blue as rover no-go (terrain slope > rover slope), and yellow as go (terrain slope < rover slope) points. Figure 8A shows the original lunar data file (1.5 million points) plotted using the Matplotlib library. Figure 8B is the 300x300 reduced matrix that is a good approximation of the large lunar data file. Figure 8C shows the area around the landing and destination site that was used for the maze. It has better resolution (due to the zooming effect) than the reduced matrix of the whole region.

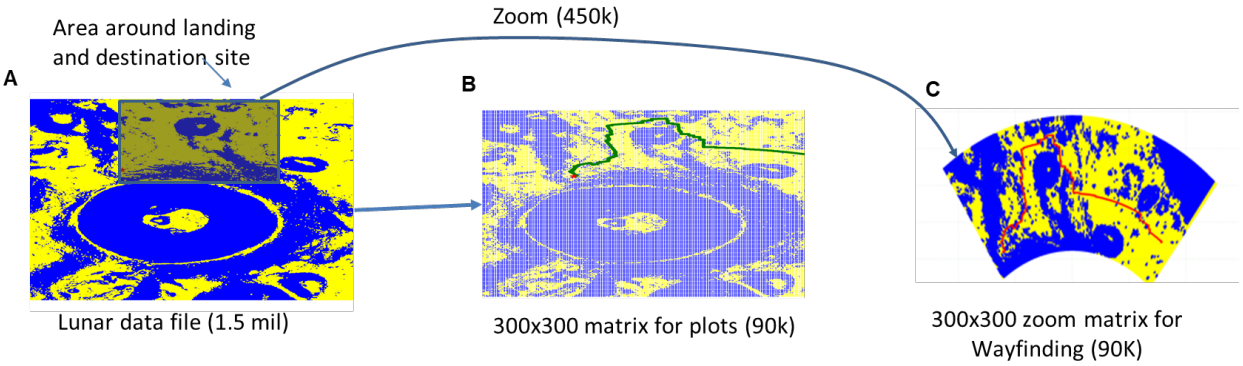

Figure 8. Data Integrity

Automated Rover Path

The 2D terrain was plotted using Matplotlib scatter plots which represent cartesian values of latitude (x-axis) and longitude (y-axis). The blue color represents the coordinates that are too steep for the rover (no-go) and the yellow color represents the coordinates that are the go paths for the rover. The green color shows the calculated rover path. The slope is the slope that the rover can overcome.

The data shows that a rover slope of 14 cannot reach the destination due to the lack of a continuous terrain path of slope 14 between the start and the endpoint. As the rover's slope increases, the more terrain path it can overcome (yellow color), and the rover can take a shorter path to reach the end destination.

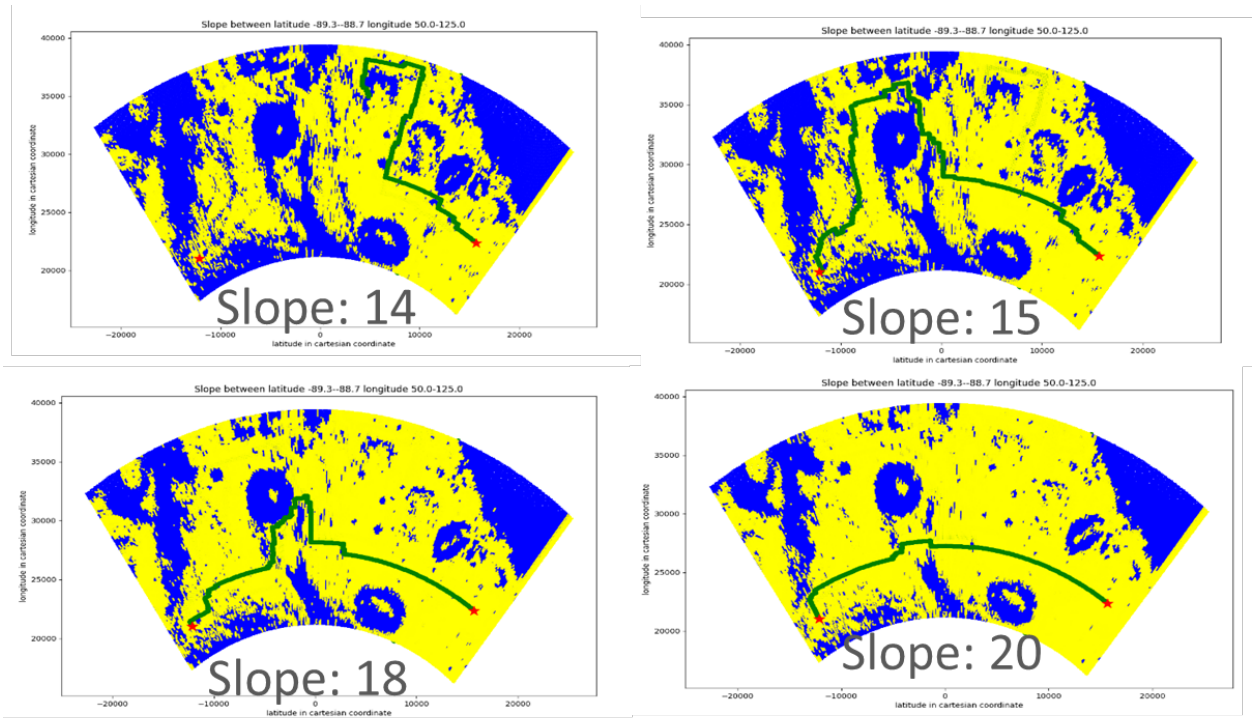

Figure 9. Automated rover path. The results show that from the landing site to the Shackleton crater, the rover needs to be designed to overcome a minimum slope of 15. The path gets shorter as rover power increases (the more slope it can overcome).

BFS vs A* Algorithm

The BSF algorithm traversed all possible nodes and always found the optimal rover path for rover slopes greater than 15. While BFS was always able to find the rover path, the A* algorithm (with path approximation heuristics of Euclidean Distance) was not able to traverse paths with lower rover slopes and longer distances. The directional A* algorithm tried to move towards the target and found directional nodes but was not always able to traverse the paths.

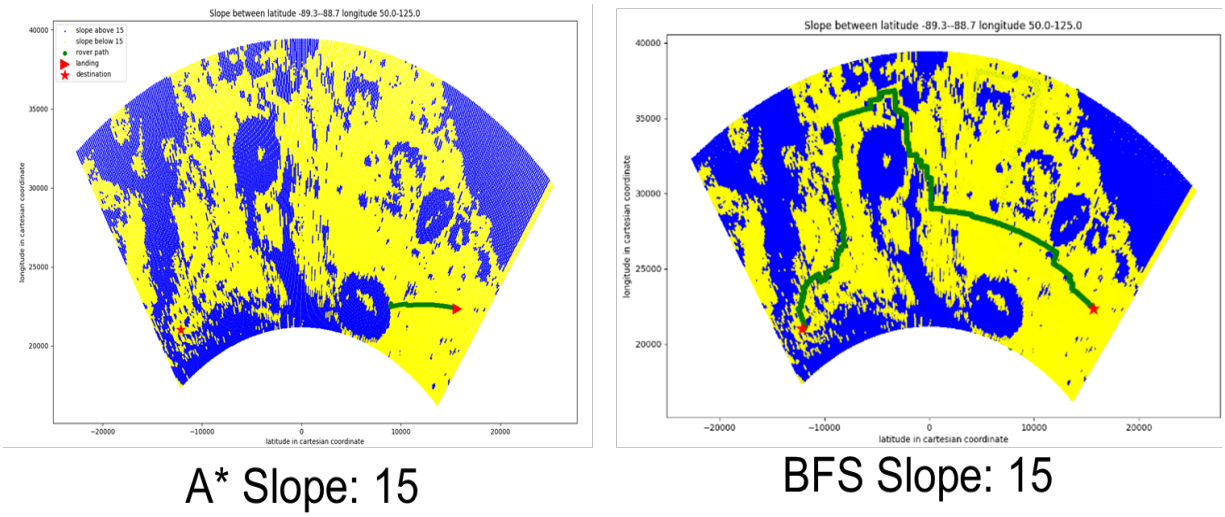

Figure 10. BFS vs A* algorithm. BFS was able to find the rover path for a rover slope of 15, whereas A* was not able to find the path for the lower rover slope.

When it was able to find the path, the A* algorithm traversed a smaller number of nodes and took less time to calculate than BFS, showing the efficiency of the A* algorithm.

Table 2. BFS vs A*: the comparison between the BFS and A* algorithms in terms of the number of nodes it traverses and the time it takes to go from start to end coordinates.

Start Coordinate (latitude, longitude)	End Coordinate (latitude, longitude)	BSF Total Time (ms)	BFS Number of nodes	A* Total Time (ms)	A* Number of nodes
(-89.099,55.017)	(-89.199,119.983)	2060.024	43356	No Path	NA
(-89.299,55.017)	(-89.299,80.100)	2120.04	44006	260.54	100

3D Terrain

The image Figure 11 is the 3D interactive view of the lunar south pole with the rover path. The 300x300 cartesian matrix created was used with the Plotly library to create the 3D terrain. The calculated rover path with slope 15 was also plotted to show the path on a 3D terrain. The x coordinate is the latitude, y is the longitude, and z is the height. The color map shows the different heights of the terrain. This can be compared to the landing site image released by NASA.

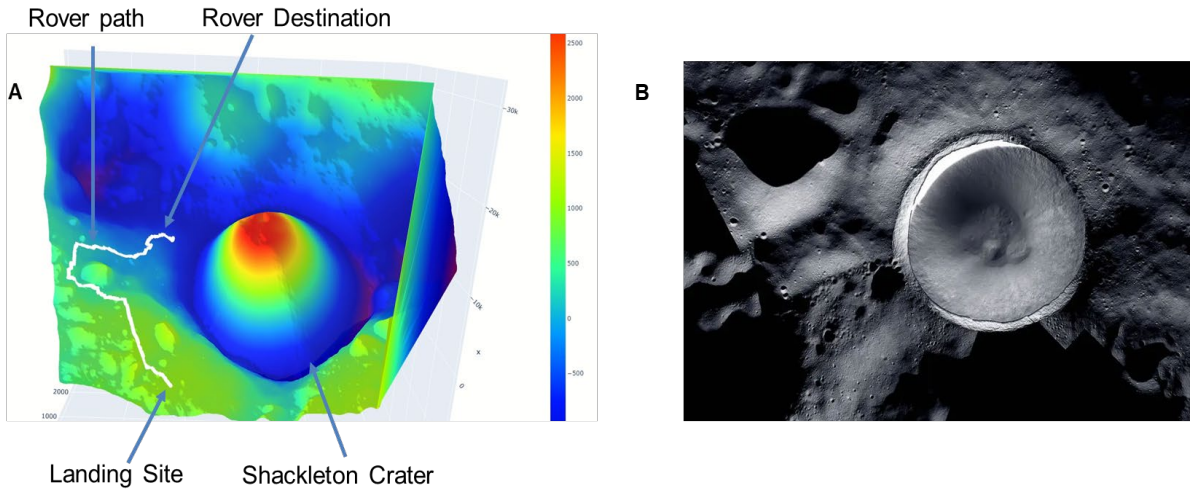

Figure 11. Comparison of the rendered 3D plot using the 300x300 matrix in Plotly (A) and the image of the south pole landing site released by NASA (B).

The NASA image (B) was created by LROC (Lunar Reconnaissance Orbiter Camera), and from ShadowCam, a NASA instrument on board a KARI (Korea Aerospace Research Institute) spacecraft called Danuri, which launched in August 2022 https://www.nasa.gov/missions/lro/nasa-moon-camera-mosaic-sheds-light-on-lunar-south-pole/

Discussion

Given the reduced sequential matrix derived from the lunar data file and the start and destination coordinates, the Breadth First Search (BFS) algorithm can always find the shortest rover path (or no possible path). For the path between the landing site and the Shackleton crater, the rover slope of less than 15 cannot find a path. So, for this specific path, the rover needs to be designed with enough power to overcome a minimum slope of 15. As the amount of slope the rover can overcome becomes greater, the path between the start and the endpoint gets shorter, as more terrain becomes available to be overcome by the rover.

The A* algorithm (widely used in maze solving), due to its directional nature, can be faster with fewer traversed nodes for most mazes. But for the lunar pathfinding maze, A* could not always find a path while BFS was always good at finding it. The BSF algorithm traverses all possible nodes and finds the optimal path. The A* algorithm tries to move in the direction of the endpoint which is not necessarily an optimal solution for the complex lunar terrain.

The 3D plot provided a good visualization of the landing site, Shackleton crater, and rover path and compared well with the landing site image released by NASA.

Conclusion

The results from our fully functional Application met our engineering goals. The BFS algorithm can always identify the shortest possible path between any two coordinates, given the rover slope. Our analysis shows the counterintuitive result that BSF is a better choice for lunar terrain than A*. The interactive application can be very useful to find the slope the rover needs to overcome to go through different paths in the lunar south pole.

It can be a useful tool for designing the rover as well as charting paths for the rover to go between any two coordinates.

The algorithm was applied on the lunar data file with the NASA official landing and destination coordinates on the lunar surface. The algorithm and the resulting rover paths were validated by the NASA SCaN (Satellite Communication and Navigation) team of engineers. The application can be used for rover design requirements and validation purposes.

Limitations

The reason behind the failure of the A* algorithm to find the optimal path consistently is not decisively understood. It may be due to the complex nature of lunar terrain. Further research needs to be done with different Heuristics of the A* algorithm to find if this faster algorithm can be used consistently to solve this problem.

Acknowledgments

We would like to thank our mentor and team members who worked on other aspects of this project.

References

Space Communication and Navigation Team, NASA (2022), Moon SPARX Mission Commander Handbook https://www.nasa.gov/stem

A. M. J. Sadik, M. A. Dhali, H. M. A. B. Farid, T. U. Rashid and A. Syeed (2010). A Comprehensive and Comparative Study of Maze-Solving Techniques by Implementing Graph Theory. 2010 International Conference on Artificial Intelligence and Computational Intelligence, Sanya, China https://ieeexplore.ieee.org/document/5656597

Gilbert Strang & Edwin Herman, Libretexts (2015). Calculus (OpenStax) : Cylindrical and Spherical Coordinates. https://math.libretexts.org/Bookshelves/Calculus/Book:_Calculus_(OpenStax)/12:_Vectors_in_Space/12.07:_Cylindrical_and_Spherical_Coordinates

Soner Yıldırım (2010). How to Create Dynamic 3D Scatter Plots with Plotly. Towards Data Science https://towardsdatascience.com/how-to-create-dynamic-3d-scatter-plots-with-plotly-6371adafd14