Deterministic Optimal Control Using Geographic Point Cloud Datasets

Thomas Joyce

Department of Mathematics, University of Arizona

Github repository

1 Introduction

Devising an optimal path through a terrain sounds simple enough, as we humans do this regularly in our everyday lives. Yet, a machine can do so with greater efficiency and accuracy, as long as it is provided the correct information. For the purpose of finding an optimal path through a landscape or terrain, the primary focus will be on the gradient, or change in position of locations along the terrain. Studies have pursued optimal trajectory problems like this with AI and machine learning algorithms, but they lack robustness and vary in their efficacy when weighing crucial decisions [1] [2].

Another way to tackle this challenge is through optimal control theory. This has been done previously using methods outlined in [3] and [4] for solving the Hamilton-Jacobi-Bellman equation. However for this interpretation of the problem, we will be evaluating movement deterministically, where each point is specified and has an associated weight to it. This weight will come in the form of the speed associated with traversal and how favorable it is towards the overall objective, that being the destination.

This work is important to the advancement of path-dependent knowledge. For example, it has weight in ecological studies, where an animal must traverse a terrain, revealing its migration patterns. Additionally, there the rise in autonomous vehicles, requiring the most efficient path through their respective environments as investigated in [5]. This could even have ramifications for navigating harsh environments, like the Curiosity or Perseverance rovers on Mars [6] [7].

2 Methodologies

2.1 Real-world Data Set

The data utilized within the algorithm is a real-world selection of terrain coordinates publicly available through the United States Geological Survey (USGS). The terrain data comes in the form of raster points made from LiDAR (Light Detection And Ranging) imaging maps of a particular region stored within a Geographic Tagged Image File Format (GeoTIFF). An example of a few GeoTIFF files can be found in my Github repository under example data, but more files can be downloaded directly from the USGS website available here. To acquire a terrain file that differs from the example data, use the database and select a particular region that is available under the tab "Elevation Products (3D Elevation Program Products and Services)." Make sure that the corresponding file type is a GeoTIFF so it can be processed through the Python pipeline properly.

Next, the raster data in the form of a GeoTIFF is passed into the script Geotiff-xyz, which converts the longitudinal and latitudinal raster points within the image into cartesian x,y,z coordinates measured in meters. This is done by first projecting a cartesian plane onto the surface of the globe evaluated at a particular spot, giving the initial coordinates. Then, through this location-based projection, the raster image can be multiplied by a transformation matrix, converting the cartesian points into meters relative to the zero point in the coordinate reference system. Through this process, the data can be represented as a point cloud.

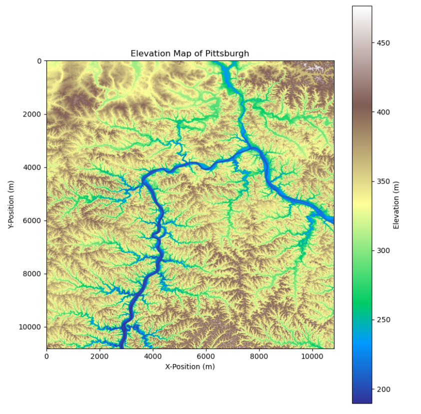
Using this point cloud, other parameters can be determined, such as the gradient. By utilizing adjacent points in the cloud, the gradient can be established at each point by the combination of the horizontal and vertical slopes, as seen below in the equation:

$$Slope = \sqrt{(\frac{dz}{dx})^2 + (\frac{dz}{dy})^2} \tag{1}$$

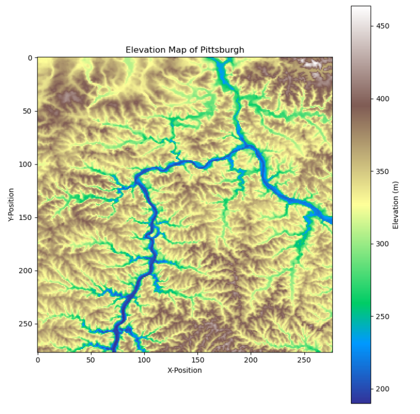
However, for this investigation and ease of computation, a large point cloud can become a hassle. One way to combat this is through binning the real-world data to create a subsample of the landscape. The program can also intake binning parameters through two methods, median and sampling, along with a user-specified bin factor (ex: 4, 10, 40, etc). The sampling option takes into consideration one point for every bin (ex: every 4th, 10th, 40th, etc). The median functionality takes the median over a ((bin factor * 2) - 1) range and uses that value to associate with the specific point. For example, in a bin factor of 4, the median would be taken over points 1 - 7, and the median would be assigned to the value of point 4. Keep in mind that the binning is done in both dimensions x and y. The binned data is then written to a text file with the x,y,z positions and the slope evaluated at each point.

An important mention is that for large bin factors (> 10), the user should refrain from the sampling setting, as the gradient is being taken over this binned range, potentially skewing it to larger values in some cases and smaller values in others, skewing the perception of the terrain. However, the median method helps to smooth out the slope so that it does not experience as drastic of changes, making the data easier to work with.

An example of the resulting output and binning process of the Geotiff-xyz can be seen in Figure 1.a and 1.b below:



(a) Unbinned elevation map of Pittsburgh



(b) 40x median binned elevation map of Pittsburgh

Figure 1: Results of Geotiff-xyz, showing the result of the binning process.

2.2 Speed and Favorability Functions

Using the binned point cloud for traversal and decision-making requires a metric to weigh against when deciding how to move forward. This comes in the form of a favorability function, dependent upon the speed of movement from one pixel to another. After given the initial and final coordinates, the traveler should be able to access these functions at each point and determine the optimal path through the maximum of the favorability function.

The speed function is in the form of a matrix (corresponding to the terrain) depicting a value of the maximum speed the traveler can go when traversing a pixel. For example, the speed function could be 0.45 at position (1,1) and 0.89 at position (4,7), and so on. The speed function is determined point by point, unique to each entry, and contains two components: the speed multiplier and the gradient-based functional form.

The speed multiplier is determined first by processing the gradient data and determining which points will be unreachable due to a substantially large slope. The current insurmountable limit is whenever the slope of a point reaches 50°. This is the level at which most travelers either find it too steep or too unsafe to proceed, halting their motion. A sharp limit before this cut-off is at 45°, where the traveler would most likely have to begin clamoring with their hands to achieve any gain but can do so very slowly. At any point if the value is 50°(corresponding to a slope value of $\tan(50°) = 1.19$), the speed multiplier is 0, meaning the terrain is impassable. At a slope value of 1 (45°), the speed multiplier is 0.1, passable but just at 10 % of the maximum obtainable speed. Any other value less than a slope of 1 is multiplied by 1, meaning the terrain is passable at 100 % max speed. The speed multiplier for each point takes the form of the variable S_{m_i}.

The second component of the speed function is the gradient-based speed functional form (e^{-Grad_i}). This accounts for steeper slopes being exponentially more difficult to traverse. It should be noted that this same form is used for downward and upward traversals. If the traveler must traverse uphill, it will be slower with steeper slopes due to the extra work put into each movement. However, if the traveler must traverse downhill, it will be slower by the same degree as the traveler will have to be safe and secure their footing. The total speed function is modeled below, representing the speed calculated at each point.

$$S_i = S_{m_i} e^{-Grad_i} \tag{2}$$

This speed function is similar to that used in [3], but simplified to fit within the confines of this study and how the slope is specifically measured. An example of the speed function evaluated for the terrain of Pittsburgh can be seen in Figure 3, dependent upon the gradient shown in Figure 2.

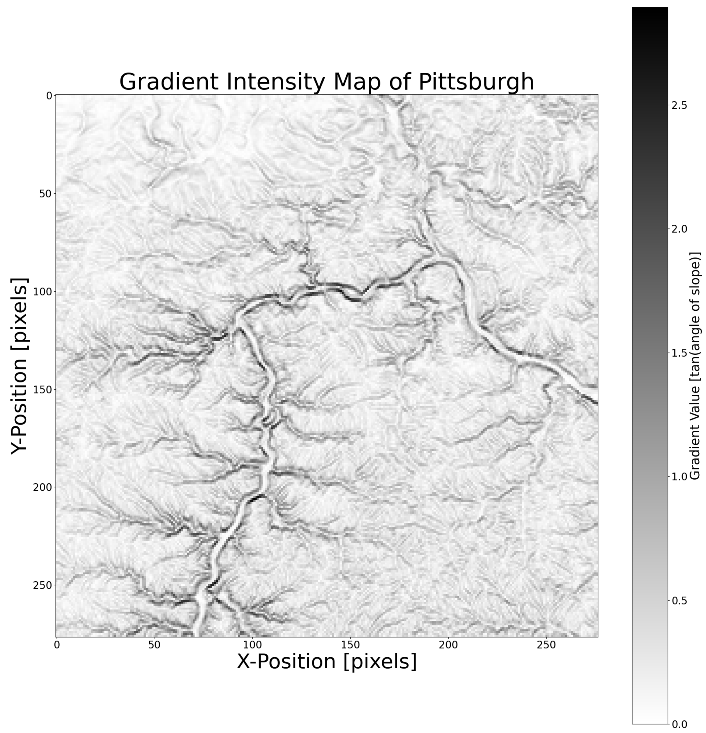

Gradient Intensity Map of Pittsburgh
Y-Position [pixels]
X-Position [pixels]
Gradient Value [tan(angle of slope)]
0
50
100
150
200
250
0.0
0.5
1.0
1.5
2.0
2.5

Figure 2: Uncolored gradient intensity map of Pittsburgh, showing the magnitude of the fractional slope of each point available from the terrain in Figure 1.b.

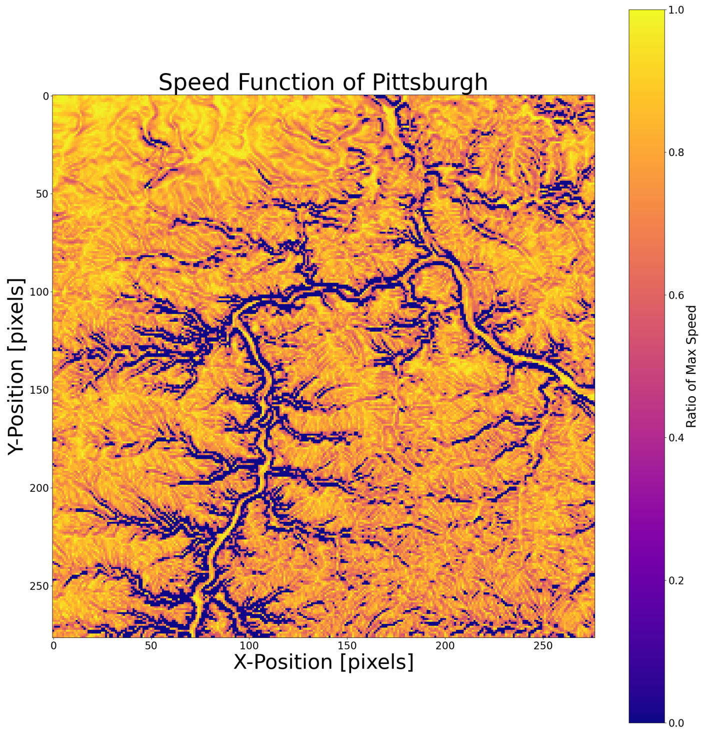


Figure 3: Color-coded speed function of Pittsburgh. Showing the magnitude of speed evaluated at each point in the terrain of Figure 1.b.

This speed function is valuable to determine small changes, optimizing the local pathing. However, to consider the global objectives, we must introduce a favorability function (F) that takes into account reward factors, such as reaching the destination, wanting to travel downhill, or even the addition of other locations or "nodes" the traveler might want to reach on their way to the destination. This favorability function will be the function that is being maximized as a result of the path planning process. The functional form of the favorability is as follows:

$$F_i = \begin{cases} CS_i(r_i - r_f)^{-2}, & \text{for} + \Delta Z_i \\ 1.1CS_i(r_i - r_f)^{-2}, & \text{for - } \Delta Z_i \end{cases} \tag{3}$$

In this example, there is one node being considered (the destination), as defined by the $(r_i - r_f)^{-2}$ term. However, if there were many destinations, we could consider a summation of many $W_n(r_{i_n} - r_{f_n})^{-2}$ terms. Where each n is a node of some weight (W_n) specifying how important reaching that location is. Once the location is reached, the weight can go to zero, as this node is no longer "active."

The favorability depends on the arbitrary constant (C) used to ensure numerical stability, the speed associated with each point(S_i), the change in elevation (as seen by a slight favorability for downhill movements), and the destination term, whether it be single or a collection of nodes. This function is evaluated specific to each movement as it depends on the elevation change. So this maximization is done movement per movement.

2.3 Optimal Control and Path Planning

With the favorability function defined for each step, it can be used as an evaluation for each pixel movement. To devise the best decision on how to proceed, we must consider a few steps in advance to fully utilize the contribution of the node/destination term in the favorability function. This is done by first planning a route and then choosing a step in the direction of the best route.

Beginning with the actual position of the traveler in the form of index x and y (x_i, y_i), we must plan a route for this starting point. Within the plan route function, the program starts 4 paths with a predefined first step (a step in the northward, eastward, southward, and westward directions). This predefined step is the first step in the series of the route, but after this first step, the program is free to move in any direction (north, east, south, west) in each subsequent step. The user can define a number of "weighing steps," which correlates to how many steps are considered within the plan route function. For example, if weighing steps = 7, then each of the 4 paths will consider the first predefined step in their respective directions and will then decide to move in whatever the following directions are by 6 additional steps. After plan route is run, there will be 4 routes of 7 individual steps each to weigh when deciding which direction to move.

With each iteration of the plan route, we have a plan step function, which considers the position of the traveler and decides which movement (northward, eastward, southward, or westward) the traveler should make based upon the maximum of the favorability function. This function then saves the favorability value associated with the optimal movement and passes it to the plan route. Whenever the plan route decides which path is best, it will find the maximum sum of each plan step's favorability output and decide to move in the direction of the first step in that path. An example can be seen below, and a visual representation of the process is also shown below in Figure 4.

Example:

- Weighing Steps = 3
- Starting at position (2,2)
- Ending at position (5,5)

When running plan route, it will execute the first step in each direction (northward, eastward, southward, and westward). Then, it will initiate the plan step function to determine the other two movements, giving the routes below:

- Northward route:
  - Route movements: (2,1), (plan step), (plan step)
  - Favorability: F = 3 + 4 + 6 = 13
- Eastward route:
  - Route movements: (3,2), (plan step), (plan step)
  - Favorability: F = 4 + 5 + 7 = 16
- Southward route:
  - Route movements: (2,3), (plan step), (plan step)
  - Favorability: F = 6 + 5 + 9 = 22
- Westward route:
  - Route movements: (1,2), (plan step), (plan step)
  - Favorability: F = 2 + 4 + 6 = 12

The plan route function will then take the summation of each favorability value for each movement and find the maximum. In this case, it would be the southward route. The traveler will then take a step southward (in the direction of the most favorable route). The program will then begin again planning out 3 steps in each direction relative to this new starting point at (2,3).

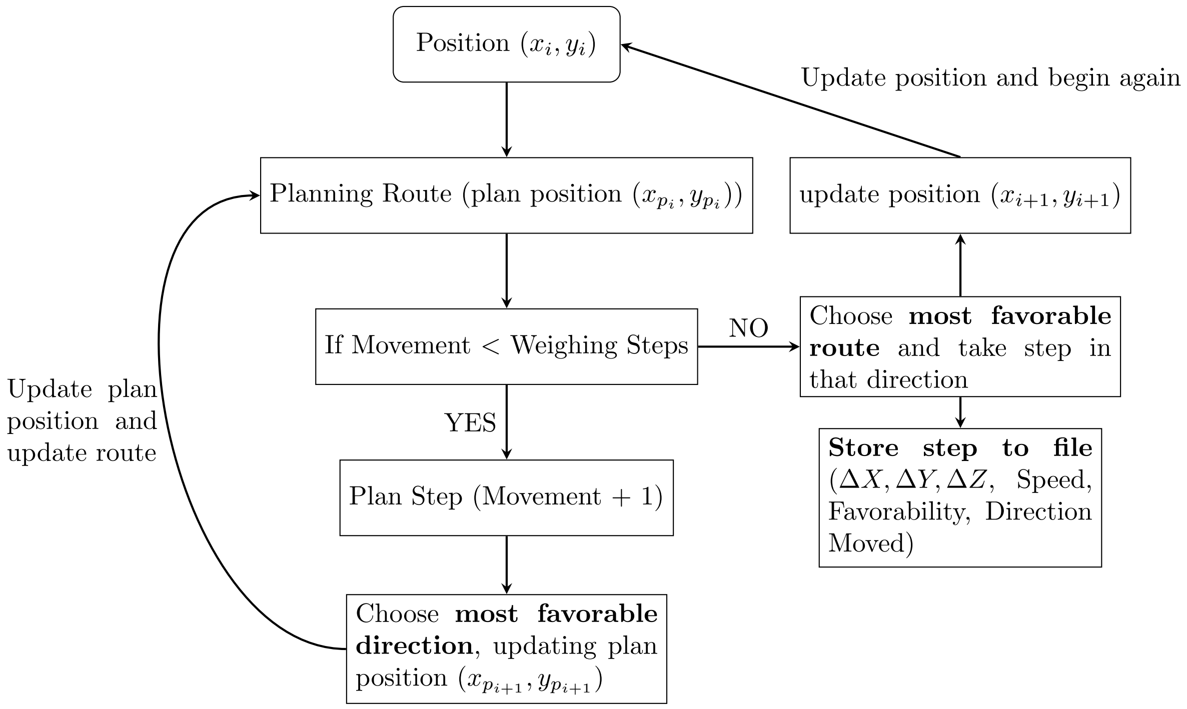

Position (x_i, y_i)
Planning Route (plan position (x_{p_i}, y_{p_i}))
If Movement < Weighing Steps
YES
Plan Step (Movement + 1)
Choose **most favorable direction**, updating plan position $(x_{p_{i+1}}, y_{p_{i+1}})$
Update plan position and update route
NO
Choose **most favorable route** and take step in that direction
Store step to file ($\Delta X, \Delta Y, \Delta Z$, Speed, Favorability, Direction Moved)
update position (x_{i+1}, y_{i+1})
Update position and begin again

Figure 4: Flowchart of the Optimal Control Movement Process.

3 Future Work

To take this work further, several future steps need to be implemented to build a fully functional optimal control problem. For instance, the process outline in Figure 4 has yet to be fully implemented. So far the data has been manipulated, and the speed and favorability functions have been implemented. However the decision-making and path routing needs to be fully functional to have a tangible research product. This will be the primary goal for the remainder of the semester.

Another important consideration is how the traveler might react to a local minimum. It could get stuck in a zone that has "no good movement options" or a region of similar favorability. In this case, a special condition can be used where the weighing steps are increased if the traveler stays within a certain area for too long. By increasing the weighing steps, it puts greater influence on the effect of the node term (governing large-scale changes) in the favorability function.

A component that would be nice to implement, but not necessary would be the implementation of additional degrees of motion, like taking into account movements in the northeast, northwest, southeast, and southwest directions, providing a more realistic picture of terrain traversal. In this vein, it may be a sacrifice of computational efficiency to get a more realistic motion. The program will have to be operating in an optimized capacity to handle this additional computational load. This will be a supplementary goal to have if extra time exists.

For now, the developments to the program will mainly come down to plenty of "fiddling" to get the equations, coefficients, and special conditions just right in order for the traveler to make a reasonable path. Through testing of the binned Pittsburgh data set currently available, a benchmark can be established for future improvements and additional functionality! For a complete road map of version changes and improvements to the code, reference my Github repository available here.

References

[1] E. Commission, J. R. Centre, I. Sanchez, H. Junklewitz, and R. Hamon, *Robustness and explainability of Artificial Intelligence – From technical to policy solutions.* Publications Office, 2020.

[2] L. Chen, Z. Jiang, L. Cheng, A. C. Knoll, and M. Zhou, "Deep reinforcement learning based trajectory planning under uncertain constraints," *Frontiers in Neurorobotics*, vol. 16, p. 883562, 2022.

[3] C. Parkinson, D. Arnold, A. L. Bertozzi, Y. T. Chow, and S. Osher, "Optimal human navigation in steep terrain: a hamilton-jacobi-bellman approach," *arXiv preprint arXiv:1805.04973*, 2018.

[4] C. Parkinson, D. Arnold, A. Bertozzi, and S. Osher, "A model for optimal human navigation with stochastic effects," *SIAM journal on applied mathematics*, vol. 80, no. 4, pp. 1862–1881, 2020.

[5] C. Parkinson and I. Boyle, "Efficient and scalable path-planning algorithms for curvature constrained motion in the hamilton-jacobi formulation," *Journal of Computational Physics*, vol. 509, p. 113050, 2024.

[6] NASA, "Mars science laboratory: Curiosity rover," 2013.

[7] K. A. Farley, K. H. Williford, K. M. Stack, R. Bhartia, A. Chen, M. de la Torre, K. Hand, Y. Goreva, C. D. K. Herd, R. Hueso, Y. Liu, J. N. Maki, G. Martinez, R. C. Moeller, A. Nelessen, C. E. Newman, D. Nunes, A. Ponce, N. Spanovich, P. A. Willis, L. W. Beegle, J. F. Bell, A. J. Brown, S.-E. Hamran, J. A. Hurowitz, S. Maurice, D. A. Paige, J. A. Rodriguez-Manfredi, M. Schulte, and R. C. Wiens, "Mars 2020 Mission Overview," , vol. 216, p. 142, Dec. 2020.