COLORADO STATE UNIVERSITY

Quinn Peterson

Brendan Geraci

Harshith Reddy Chitreddy

Joshua Weisner

Rover Rescue Optimization

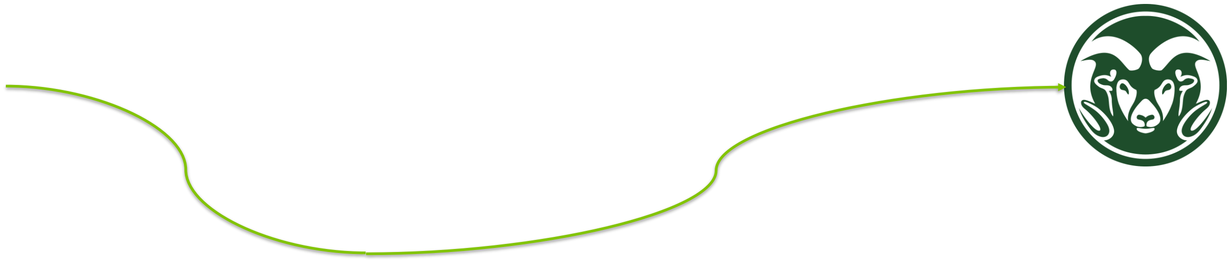
- Functional Requirements:
  - Finds the best way to the end point
  - Avoids steep hills and declines, and prefers roads to hills to save energy
  - Only goes off road if the road path is inefficient and slower
- Code Requirements:
  - Takes in DEM file, hillshade file and trail path
  - Draws optimal path for rover to take
  - Completes the task in optimal time
  - Works well on any other data set

Our Approach

- A* Search:
  - Found the most efficient route to the end point
  - Slower
- Bidirectional A* Search:
  - Found the most efficient route to the end point
  - Three times faster
- Other options:
  - D* Search
    - Like A* but for dynamic data

Bidirectional A* Search Algorithm

- Off road node cost = Offroad weight + (elevation difference * elevation resistance)
- On road node cost = On road weight + elevation difference
- Traverses map from both points until they meet in the middle, decreasing computation time by ~70%

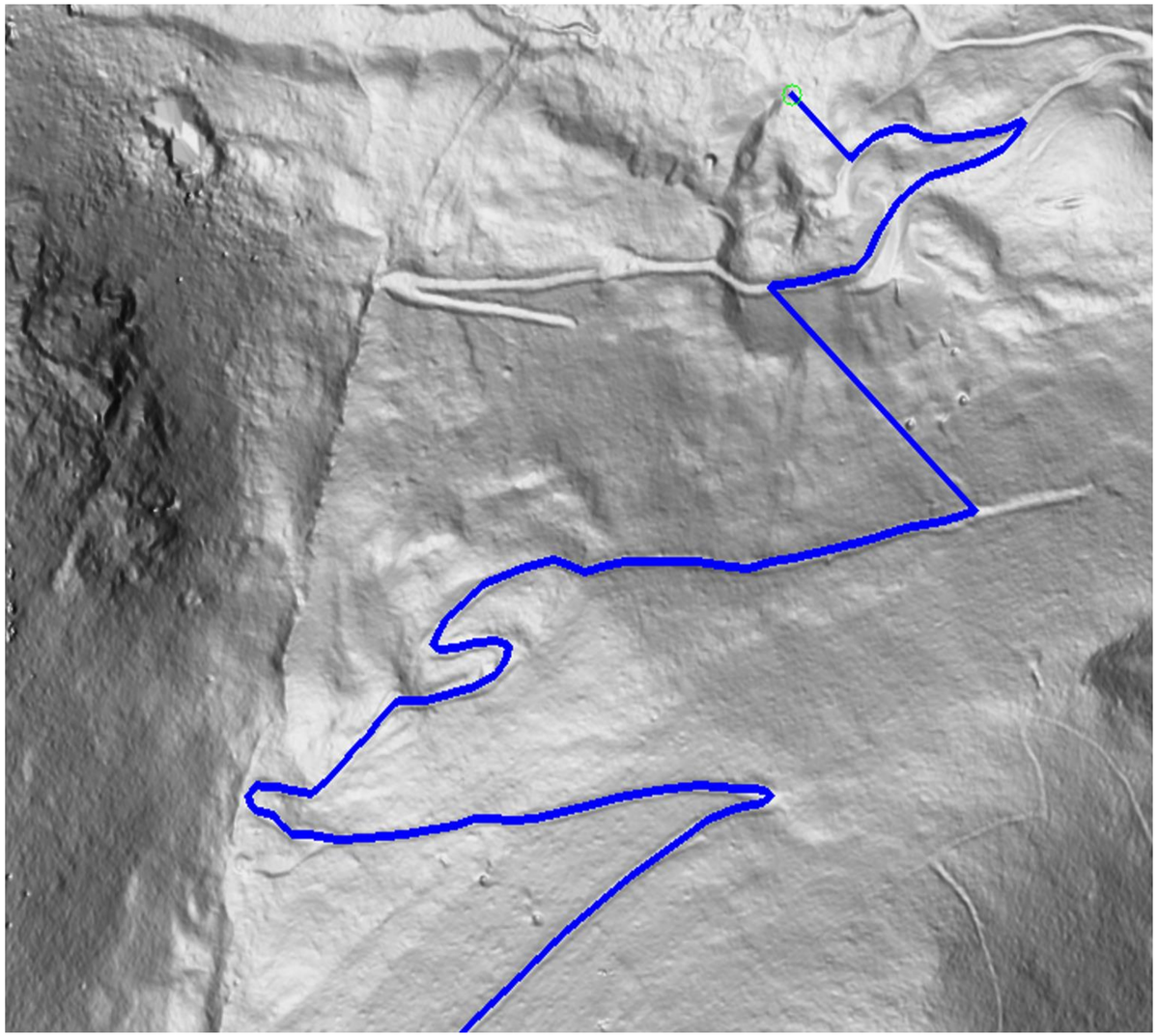


Possible Improvements

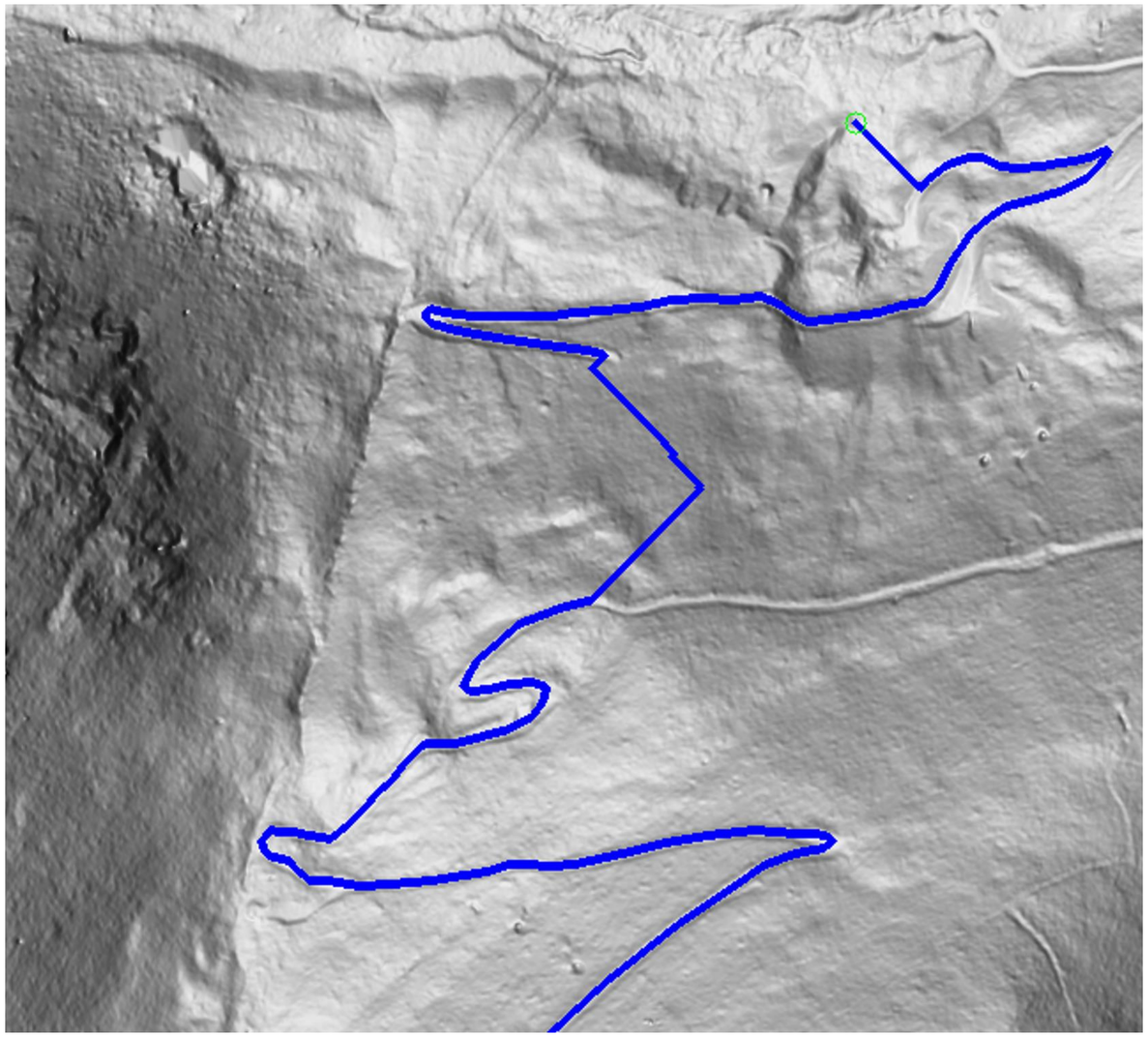
- Dashboard with selectable safety and time trade off values
- Add heuristic for temperature map to avoid water and other extreme conditions

Usability

- Using a graph search algorithm instead of AI reduces cost and time
- Using of Jupyter file makes new user comprehension and additions simple
- Modular outputs allow for unit testing

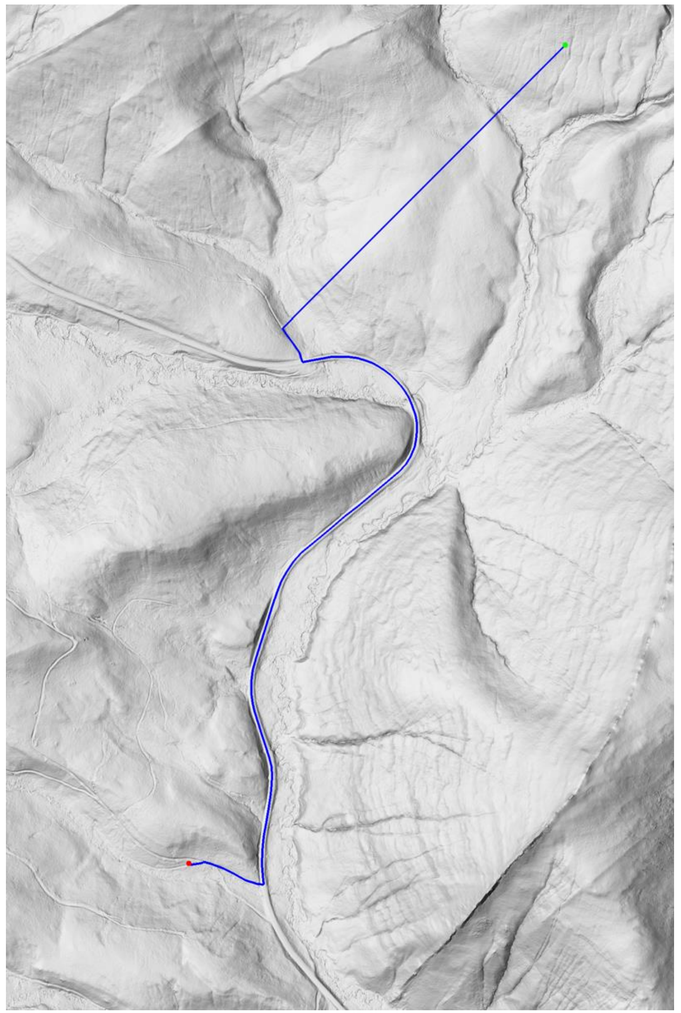


Without Elevation Resistance



With Elevation Resistance

Validation Dataset



Start and goal coordinates selected at random

Code Snippets

```python
def rover_cost(current, neighbor, dem, road_mask, road_weight=0.2, offroad_weight=5.0, elevation_weight=1, elevation_resistance = 2 ):
    elev_diff = abs(dem[neighbor] - dem[current]) * elevation_weight
    if road_mask[neighbor] == 1:
        return road_weight + elev_diff
    else:
        return offroad_weight + elev_diff * elevation_resistance
```

```python
def heuristic(a, b): #Based on Euclidean distance
    return np.linalg.norm(np.array(a) - np.array(b))
```

```python
def bidirectional_search(start, goal, dem, road_mask):
    rows, cols = dem.shape
    open_set = {
        'start': [(0, start)],
        'goal': [(0, goal)]
    }
    came_from = {
        'start': {},
        'goal': {}
    }
    g_score = {
        'start': {start: 0},
        'goal': {goal: 0}
    }
    f_score = {
        'start': {start: heuristic(start, goal)},
        'goal': {goal: heuristic(goal, start)}
    }
```