# Detailed Roadmap of the Entire Process:

## ( Calibration – Detection – Pathfinding – Visualization )

- As the Orbiter High Resolution Camera (OHRC) images are not Georeferenced, they are first georeferenced using GDAL and reprojected with appropriate CRS - Lunar South Polar Stereographic
- The Dataset also includes Calibrated OHRC images which along with the .IMG files includes a .CSV file which consists of Latitude, Longitude for each Pixel value in the OHRC image. This helps in determining the coordinates of Craters and Boulders that we would be detecting further.

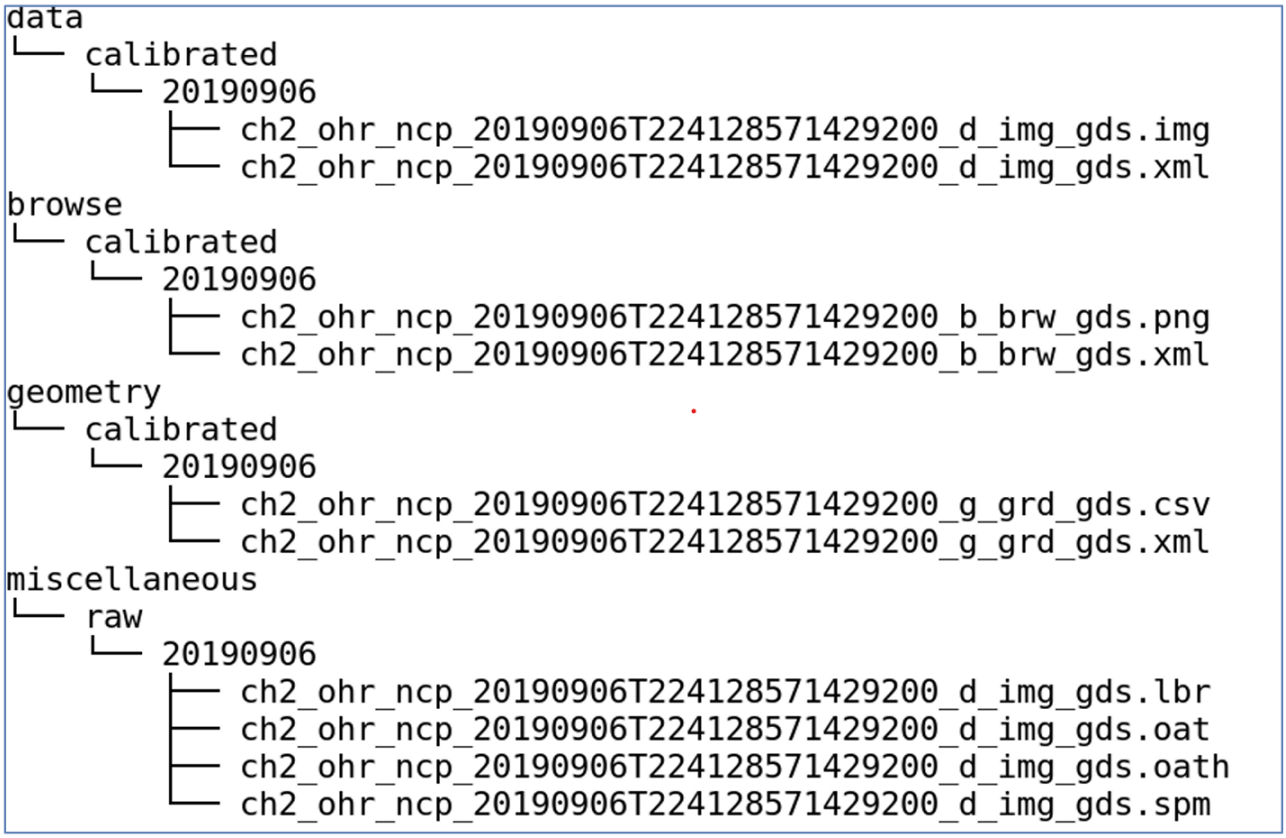
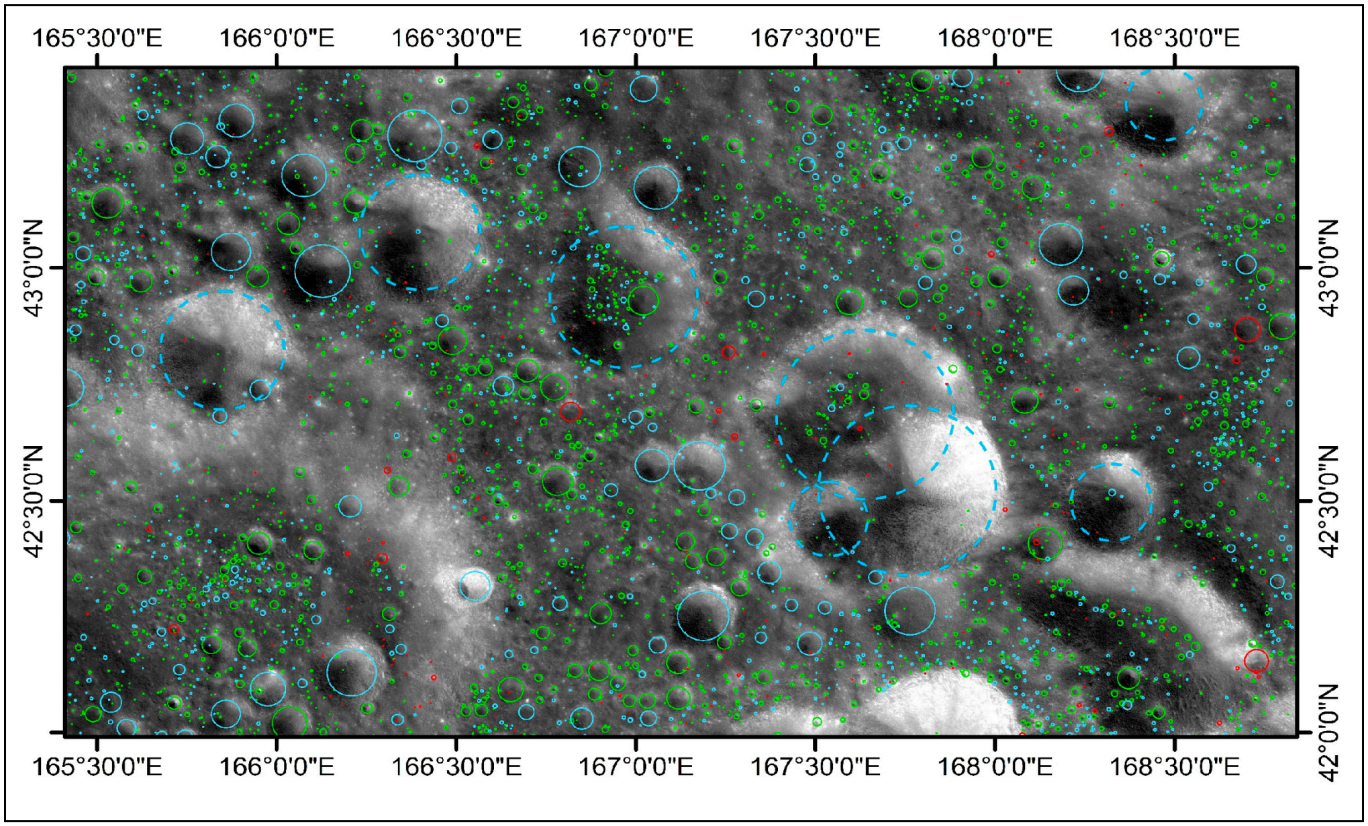



**Figure 3 OHRC Calibrated Data Archive**

- Furthermore, after the Georeferencing and Reprojecting of the OHRC image, we would be clipping the whole dataset (The OHRC, DTM and the Orthophoto) so that it only includes the interested region(s) / tentative landing coords.
- This clipped OHRC is feeded into the Detection Pipeline, which includes preprocessing (Normalization, Denoising)

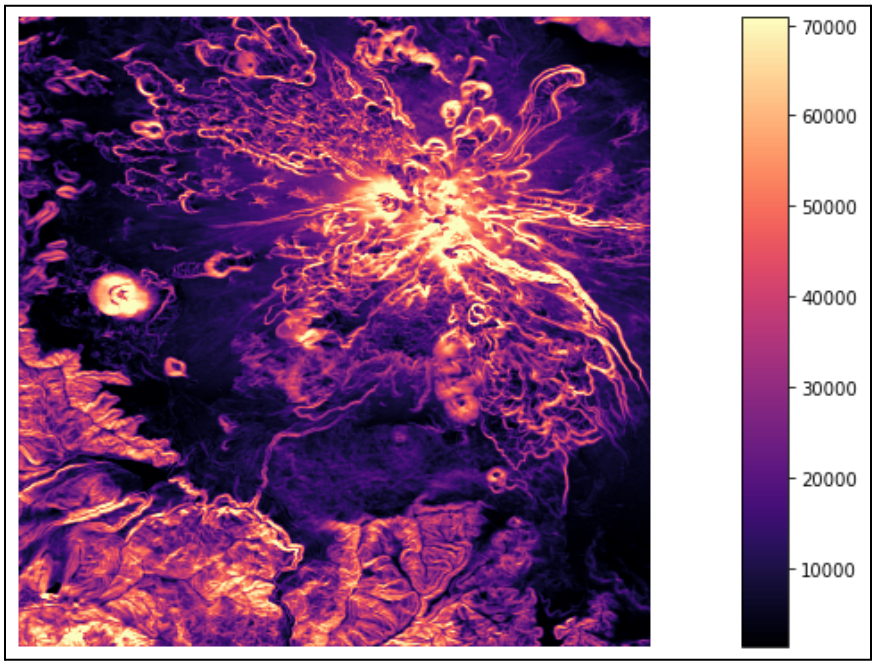
- **Approach A**:
    a. The Clipped OHRC image would be parallel processed for YOLOv5 and Traditional Computer Vision (Circle Hough Transform, Blob Detection)
    b. The output of YOLOv5 would be merged with the output of parallel sub-pipeline, i.e detections using Traditional Computer Vision

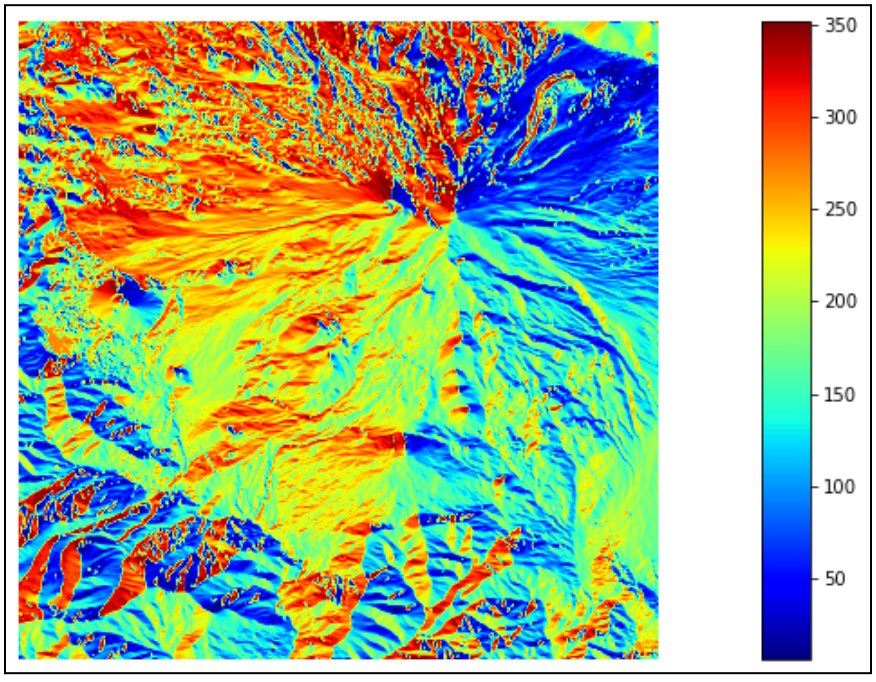



- **Approach B**:
    a. Instead of strictly following parallel processing in the pipeline, we would implement two models, which would be Detection using only YOLOv5 and Detection using Traditional CV.
    b. Providing them with the analysis of both the separate pipelines, and their latency when they are implemented solo, and latency when combined.
    c. This provides them with an option to either choose a Faster but less accurate (Solo YOLOv5 and Solo CV) implementation or a comparatively Slower but more accurate (Dual Result Merging) implementation.
- After successful detection of obstacles (Craters and Boulders), they would be saved in a .TXT or .JSON which would include their center latitude and longitude and radius of each obstacle.

- The Ant Colony System, a variant of Ant Colony System, would be used for Path Finding to traverse between individual points rather than all 10 points altogether.

- Which will consider the Elevation data from the DTM which would be calculated. The Slope (Slope is the steepness or the degree of incline of a surface) and Aspect (Aspect is the orientation of slope, measured clockwise in degrees from 0 to 360, where 0 is north-facing, 90 is east-facing, 180 is south-facing, and 270 is west-facing) will be measured from the DTM itself.



- **Pseudocode for Georeferencing, OHRC Image Calibration, Clipping, and Obstacle Detection Pipeline:**

```
# Georeferencing and Reprojecting OHRC Images
1. Use GDAL to georeference the OHRC images
2. Reproject the georeferenced OHRC images to Lunar South Polar
Stereographic CRS

# Extracting Coordinates from Calibrated OHRC Images
1. Load the calibrated OHRC images and the accompanying .CSV file
2. Extract the latitude and longitude for each pixel value in the OHRC
image

# Clipping the Dataset
1. Clip the OHRC image, DTM, and Orthophoto to the interested region(s)
/ tentative landing coordinates

# Obstacle Detection Pipeline
## Preprocessing
1. Normalize the clipped OHRC image
2. Denoise the clipped OHRC image

## Approach A: Parallel Processing
1. Process the preprocessed OHRC image through YOLOv5 and a traditional
computer vision pipeline (Circle Hough Transform, Blob Detection) in
parallel
2. Merge the outputs from the two sub-pipelines

## Approach B: Dual Model Approach
1. Implement two separate models: YOLOv5 for detection and traditional
computer vision for detection
2. Analyze the latency and accuracy of the two separate pipelines and
the combined pipeline
3. Provide the option to choose between a faster but less accurate (solo
YOLOv5 and solo CV) implementation or a slower but more accurate (dual
result merging) implementation

# Saving the Detected Obstacles
1. Save the center latitude, longitude, and radius of each detected
obstacle (crater and boulder) in a .TXT or .JSON file
```

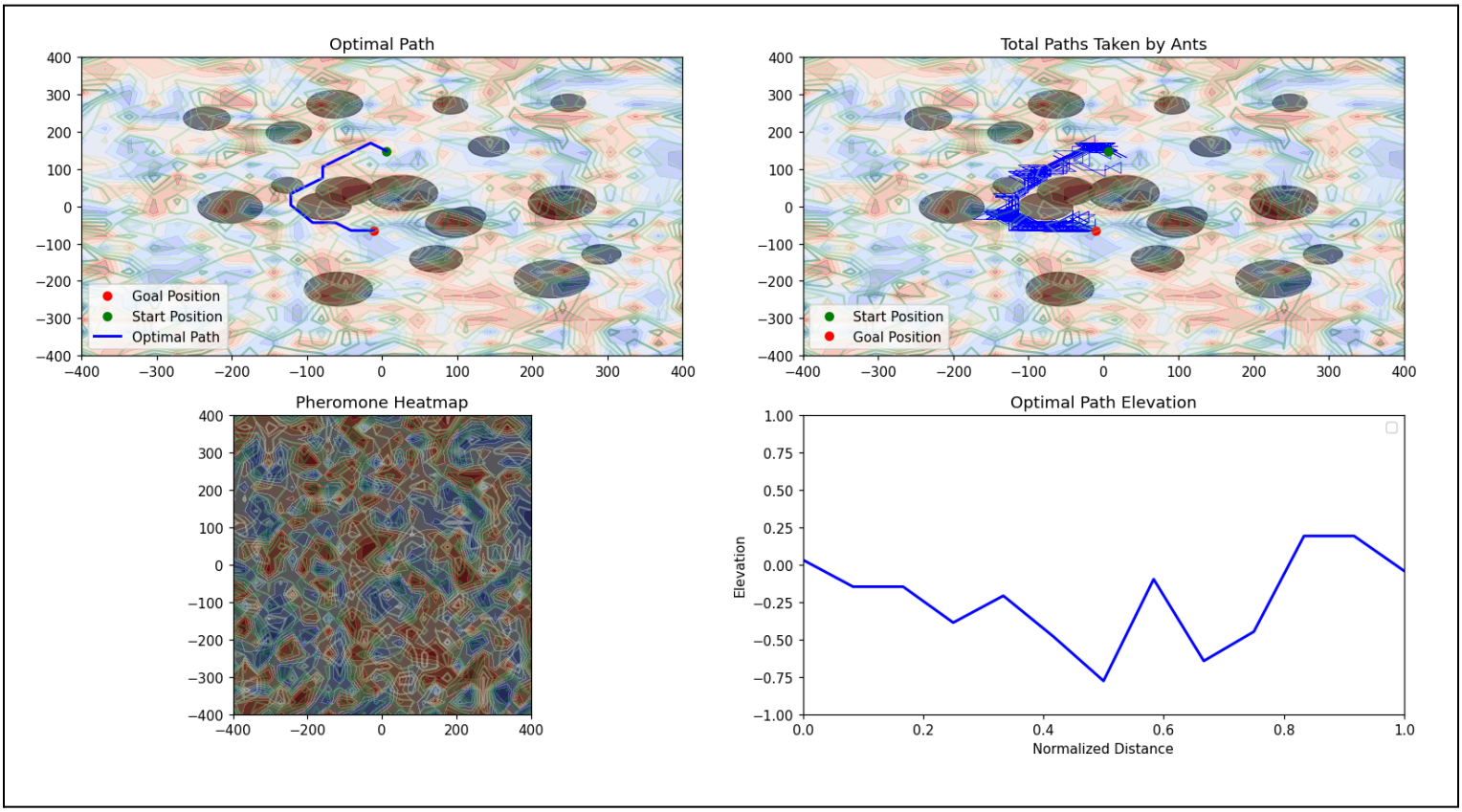
## Underlying Mechanism of ACS

- When exiting home, Ants randomly wander and leave "To Home" Markers
- When ants find the food they follow the "To Home" Markers, and leaving "To Food" Markers when found food (While returning), leading the others to the food i.e when other

ants find the "To Food" Markers they start following the markers instead of randomly wandering.

- This naturally forms a path, followed by more and more ants.
- Markers are nothing but Pheromones which are spatial dots on the map, they store a position, a type and an intensity Blue one leads to Home and green one leads to food.
- Each ant has a freedom coefficient, It determines the ants tendency to get out of the already found path and find more paths (May or may not be optimal) Potentially discovering more paths.

The above shows our Ant Colony System on an example elevation data




- Ants have been classified on the basis of their selection of starting points.
- Optimal Path is calculated when the ants from one side intersects the ants/ant's path from another side.
- The following figure shows the Best path length found after each updation of Optimal path found
- The Best path length is designed to also update after every 30-50 iterations, to avoid path stagnation.

```
Frame 17: Best path length: 1116.66
Frame 28: Best path length: 1116.66
Frame 41: Best path length: 792.57
Frame 43: Best path length: 765.68
Frame 44: Best path length: 755.76
Frame 45: Best path length: 645.68
Frame 58: Best path length: 645.68
Frame 67: Best path length: 564.04
Frame 72: Best path length: 529.30
Frame 73: Best path length: 511.73
Frame 88: Best path length: 511.73
Frame 96: Best path length: 473.65
Frame 100: Best path length: 469.30
Frame 118: Best path length: 469.30
Frame 148: Best path length: 469.30
Frame 160: Best path length: 441.35
Frame 178: Best path length: 441.35
Frame 185: Best path length: 430.80
Frame 186: Best path length: 424.11
Frame 187: Best path length: 413.65
Frame 208: Best path length: 413.65
Frame 238: Best path length: 413.65
Frame 268: Best path length: 413.65
Frame 298: Best path length: 413.65
Frame 328: Best path length: 413.65
Frame 338: Best path length: 404.82
Frame 358: Best path length: 404.82
Frame 388: Best path length: 404.82
Frame 418: Best path length: 404.82
Frame 448: Best path length: 404.82
Frame 478: Best path length: 404.82
```

- Position of ants reset with the iteration interval that is provided. This Helps other ants to calculate a more optimal path since every time ants reset their position, pheromone level of the latest optimal path is increased.
- For this example the Initial path found is calculated of length 1116.66. This decreases as iterations proceed with finding optimal paths and avoiding obstacles at the same time.

- Dark Circles are represented for Obstacles such as Craters & Boulders. Ants locate optimal paths by avoiding them.
- To deal with Elevation topography, 3 parameters vary to optimize the path :
    - Distance - For Elevated Regions, Distance required to traverse that area is increased. For Demoted Regions, Distance required to traverse that area is decreased.
    - Speed - For Elevated Regions, Speed required to traverse that area is decreased. For Demoted Regions, Speed required to traverse that area is increased.

- Heuristic Value - For Elevated Regions, Heuristic Value is increased. For Demoted Regions, Heuristic Value is decreased.

By Balancing these three values, ants optimize their path when dealing with elevation topography.

ACO is able to find the optimal path while dealing with factors that can be hazardous for the rover's traversal path.
Later in the stage, ACO will be deployed on the co-ordinates of stop points thus calculating optimized traversal route to its next adjacent stop. The Traversal Route will be optimized by Elevation topography extracted from a dataset of DTM belonging to the lunar surface.

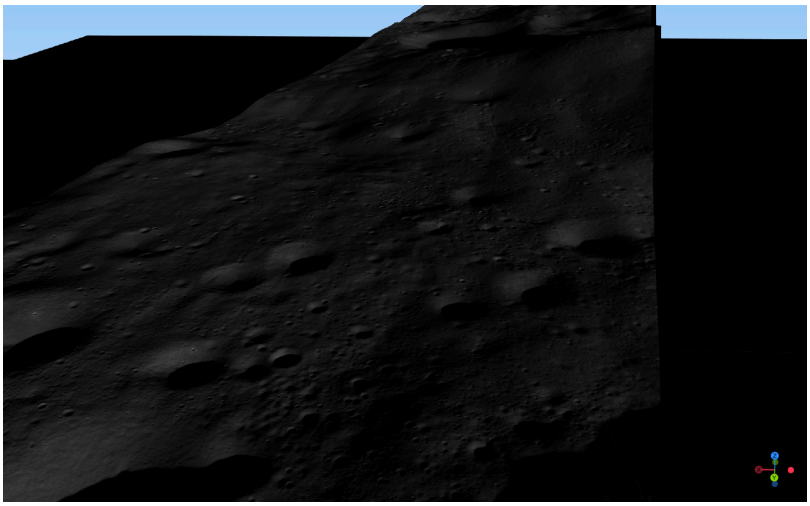
- **Pseudocode for the ACS algorithm:**

```
# Initialize constants and data structures
NUM_ANTS, MAX_ITERATIONS, ALPHA, BETA, RHO, Q, GRID_SIZE,
PHEROMONE_INITIAL, RESET_INTERVAL
Initialize goal position, start position, pheromone grid, obstacles,
elevation map, and slope map

# Define helper functions
function get_pheromone(x, y)
function update_pheromone(x, y, value)
function evaporate_pheromones()
function get_elevation(x, y)
function get_slope(x, y)

# Define Ant class
class Ant:
    function __init__(start, goal, follow_pheromone=False)
    function move()
    function get_possible_moves()
    function is_valid_move(x, y)
    function select_next_move(possible_moves)

# Initialize ants
function initialize_ants(follow_pheromone=False)
forward_ants, backward_ants = initialize_ants()

# Set up the plot
Set up the subplots for optimal path, total paths taken, pheromone
heatmap, and optimal path elevation

# Main animation loop
function update(frame):
    for each ant in forward_ants and backward_ants:
        ant.move()
```

```
    # Update paths and check for the best path
    for each ant in forward_ants and backward_ants:
        update the path for the ant

    # Update the pheromone heatmap
    update the pheromone heatmap

    # Check for and update the best path
    for each forward ant and backward ant:
        if update_best_path(forward_ant, backward_ant):
            path_found = True

    # Reset ants if needed
    increment iteration_count
    if iteration_count >= RESET_INTERVAL:
        reset_ants()

    evaporate_pheromones()

    # Print the best path length every RESET_INTERVAL iterations
    if iteration_count == 0 or path_found:
        print the best path length

    return the updated artists

# Create the animation
Create the animation using FuncAnimation and display it
```

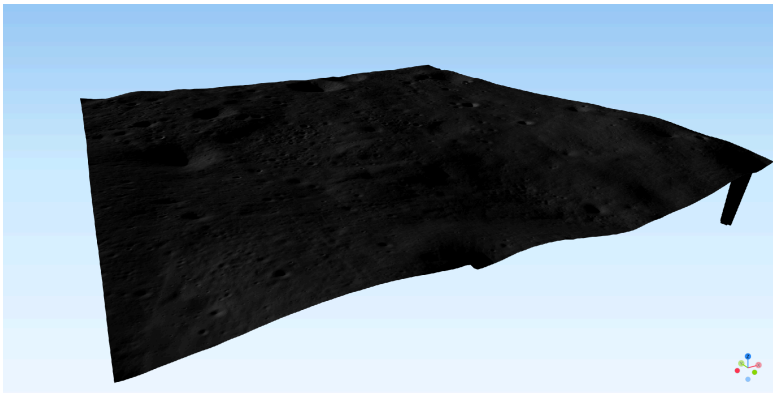
**Visualization (Virtually Simulating Rover on Optimal Traversal path)**

Simulation of the rover on the optimal path calculated by the Ant Colony System is achieved with the help of Unity Software.



- With datasets such as DTM and DEM, a mesh can be obtained using the QGIS Software.
- After extracting wireframe and topological elevation data from this dataset, the mesh can be loaded into Unity Software.
- A sample design of the rover available on the web can be considered for simulating the rover's movements.

- On this DTM dataset, 10 scientific stop points are located with the help of coordinates.
- Starting from the landing position up to the 10th point, the optimal traversal path is calculated by ACO, considering each pair of adjacent points as the starting point and destination point.
- After calculating the optimal path, it is deployed in the script that controls the movements of the rover within Unity Software.



Through such a process, the simulation of the optimal path taken by the rover on the lunar surface is virtually achieved.

- **Unity Script Pseudocode for Rover Pathfinding and Stop Point Traversal:**

```
// Variables
optimalPath = []  // List of waypoints (coordinates) for the rover to
follow
stopPoints = []   // List of 10 scientific stop points
currentWaypointIndex = 0  // Index to track the current waypoint

// Initialization
function Start() {
    // The optimal path has been loaded from the ACO result
    optimalPath = LoadOptimalPathFromACO()

    // The stop points have been loaded from the ACO result
    stopPoints = LoadStopPointsFromACO()

    // The rover's initial position has been set
    SetRoverPosition(stopPoints[0])

    // The initial waypoint has been set to the first in the optimal
path
    currentWaypointIndex = 1  // Starting with the next waypoint after
the first stop point
}
```

```
// Update function runs every frame
function Update() {
    // The system checks if the rover has reached the current waypoint
    if (HasReachedWaypoint()) {
        // The rover moves to the next waypoint in the path
        currentWaypointIndex += 1

        // The system verifies if all waypoints have been traversed
        if (currentWaypointIndex >= length(optimalPath)) {
            // The movement is stopped or simulation ends
            StopRover()
            return
        }

        // The next waypoint is set as the target
        SetRoverTarget(optimalPath[currentWaypointIndex])
    }

    // The rover moves towards the target waypoint
    MoveRoverTowardsTarget()
}

// Function to load optimal path from ACO
function LoadOptimalPathFromACO() {
    // The optimal path coordinates are retrieved from the ACO result
    return optimalPathData
}

// Function to load stop points from ACO
function LoadStopPointsFromACO() {
    // The stop points coordinates are retrieved from the ACO result
    return stopPointsData
}

// Function to set the rover's position
function SetRoverPosition(position) {
    // The rover's position is set to the given coordinates
    rover.transform.position = position
}

// Function to check if the rover has reached the current waypoint
function HasReachedWaypoint() {
    // The system determines if the rover has reached the target
waypoint
    return distanceToWaypoint < threshold
}

// Function to set the rover's target waypoint
function SetRoverTarget(target) {
    // The target position for the rover is updated
    roverTarget = target
}
```

```
// Function to move the rover towards the target waypoint
function MoveRoverTowardsTarget() {
    // The rover is moved towards the target waypoint
    // This involves physics, pathfinding, or other movement techniques
}

// Function to stop the rover
function StopRover() {
    // The rover's movement is halted
    rover.velocity = Vector3.zero
}
```