Rapid Topographic Path planning with Human-In-Loop exploration strategy

- Authors:
  - Balajee Devesha Srinivasan (basrini@iu.edu),
  - Ashley FNU (ashlnu@iu.edu),
  - Ankur Jayant (ajayant@iu.edu)

Introduction

Problem Statement

In autonomous robotics, optimizing energy consumption for robots navigating diverse and confined terrains is a critical challenge. This project aims to develop a simulation framework for an autonomous robot, emphasizing mapping the topography of its environment. Such optimization is vital for real-world applications like surveillance, exploration, and environmental monitoring.

Significance

Efficient path planning is crucial for the success of autonomous systems in practical scenarios. This project emphasizes energy-efficient path planning and human-in-the-loop (HITL) interaction, enhancing autonomous robots' capabilities in varied and complex terrains. This approach aligns with developing autonomous systems that perform optimally in resource-constrained environments.

Technical Details

Objective Fulfillment

The implemented solution achieves the outlined objectives through:

1. **Environment Simulation and Point Specification**
   - **Terrain Generation**: Using `TerrainGenerator` class to model confined environments with topographical features.
   - **Point Specification**: Specifying two points for robot navigation.

2. **Robot Modeling and Sensor Simulation**
   - **Sensor Models**: Equipping the robot with sensors for measuring elevation changes.
   - **Data Acquisition and Terrain Mapping**: Implementing terrain mapping algorithms.

3. **Rapid Localization and Topographic Mapping**
   - **Localization Algorithm**: Utilizing a localization algorithm for positional awareness.
   - **Topographic Mapping**: Implementing occupancy grid mapping for topographic representation.

4. **Energy-Efficient Path Planning and HITL**
    - **Path Planning**: Employing the RRT algorithm (`RRT_Exp` class) for energy-efficient path planning.
    - **Human-in-the-Loop Interaction**: Integrating HITL for dynamic path adjustment and decision-making support.

Class Descriptions

TerrainGenerator Class

- **Purpose**: `TerrainGenerator` class encapsulates the functionality to create simulated terrains with customizable features. Its main purpose is to generate elevation matrices representing terrains, apply modifications such as adding trees, and provide methods for visualizing the generated terrains.
- **Key Features**:
    - Fractal Noise Generation: The class utilizes fractal noise generation techniques to create realistic and diverse elevation patterns, simulating the natural complexity of terrains.
    - Customizable Parameters: Users can customize parameters such as random seed, maximum and minimum elevations, and the number of trees to control the characteristics of the generated terrain.
    - Tree Placement: The class includes functionality to randomly place a specified number of trees on the terrain, enhancing its visual realism and providing points of interest.
    - Visualization: The class offers a method to visualize the generated terrain with colored contours, providing insights into the elevation variations and the distribution of added trees.

RRT_Exp Class

- **Purpose**: The `RRT_Exp` class implements the Rapidly-exploring Random Trees (RRT) exploration algorithm. It facilitates the generation of a tree-based exploration path in a given terrain matrix from a specified start to a goal position. The exploration is performed by an agent, and the class includes methods for tree expansion, path generation, and visualization of the exploration process.
- **Key Features**:
    - Initialization:
        - Parameters: terrain_matrix, start, goal, agent.
        - Initializes the RRT exploration algorithm with the terrain matrix, start and goal positions, and an agent object.
    - Collision Checking:
        - `is_collision_free`(point): Checks if a given point is collision-free, indicating a traversable terrain.
    - Sampling and Steering:
        - `sample_free_space`(): Samples a random point in the free space of the terrain.
        - `steer`(from_point, to_point): Steers from one point to another, considering a specified step size.
    - Tree Expansion:
        - `expand_tree`(iterations=10000, goal_bias=0.1): Expands the RRT tree in the terrain space, biased towards the goal.
    - Path Finding:
        - `find_path`(): Retrieves the path from the tree structure.
    - Visualization:
        - `plot_tree`(ax): Plots the RRT tree on a given matplotlib axis.
        - `visualize_agent_traversal`(path, ax): Animates the agent's traversal along the optimal path and visualizes the exploration process.

Agent Class

- **Purpose**: The Agent class serves as the representation of an autonomous agent in the simulation, embodying the key functionalities required for the exploration and traversal of the simulated environment. Its primary purposes are as follows:
- **Key Features**:
  - Velocity and Movement Control: The `change_velocity` and `change_angular_velocity` methods allow dynamic adjustments to the agent's linear and angular velocities, ensuring they stay within predefined ranges.
  - Agent Movement: The move method facilitates the movement of the agent based on its current velocity and orientation. It computes the new position of the agent in the simulation.
  - Terrain Exploration and Cost Matrix Update: The `initialize_terrain_and_cost` and update_explored_area_and_cost methods are responsible for initializing the terrain and cost matrix for exploration and updating them based on the agent's position, respectively.
  - Event Handling: The `on_key` and `on_mouse_click` methods handle key and mouse click events, respectively. They interpret user inputs to control the agent's movement, trigger the end of exploration, and initiate the RRT algorithm.
  - Visualization Update: The `redraw_plot` method is responsible for updating the visualization of the plot with the latest information, including the agent's position and the current cost matrix.

Exploration and Path Planning Results

Exploration Phase

- **Capability**: Navigating through simulated environments and capturing elevation data.
- **Terrain Traversal**: Controlled via keyboard inputs for systematic exploration.
- **Mapping**: Real-time topographic map creation using sensor models.

Path Planning Phase

- **Outcome**: Energy-efficient path planning between specified points.
- **Algorithm**: RRT-based planning considering terrain characteristics.
- **Visualization**: Real-time path planning insights.

Discussion

Achievements

- **Energy-Efficient Path Planning**: Meeting the project's fundamental goal if performing exploration of unknown terrain and then planning an optimal path through the terrain.
- **Effective Terrain Mapping**: Laying the foundation for optimal path planning.
- **Human-in-the-Loop Integration**: Enhancing decision-making and adaptability.
- **Terrain based path planning**: In the result GIfs we can clearly see that wherever the Brown-whitish peaks are observed the RRT successfully avoid or tries to circumvent the area and wherever we see deep blue trenches due to our currently limited negative modelling, the RRT prioritizes takin in the path to optimize the overall path thus achieving dynamic path planning based on the terrain.

Future Work

- **Refining Energy Models**: Incorporating more complex modelling for the cost for elevation/depression and terrain roughness.

- **Advanced Path Planning**: Exploring beyond RRT for intricate navigation and adding fuel based modelling for exploration etc.

Challenges and Lessons

- **Continuous Improvement**: Emphasizing adaptation in complex terrains.
- **Dynamic RRT parameter tuning**: The RRT parameters need to be more dynamically optimizable instead of use trying to run it again from scratch if it fails..

Conclusion

This project successfully addresses efficient path planning in autonomous robotics, with a focus on HITL interaction and terrain adaptability. It serves as a foundation for future developments in autonomous navigation and path planning systems for terrain based cost of exploration optimization.

Project Runtime Documentation: (How to run the code)

Introduction

This guide helps you run through the Autonomous Robotics project, which involves terrain generation, autonomous exploration using an agent, and path finding using the Rapidly-exploring Random Tree (RRT) algorithm. Follow these steps to run the simulation and achieve the outputs.

Prerequisites

- Python environment setup with necessary libraries (`numpy`, `matplotlib`, `scipy.ndimage`, etc.).
- The project consists of four main files: `Agent.py`, `TerrainGen.py`, `RRT.py`, and `main.py`.
- Ensure all these files are located in the same directory.

Step-by-Step Instructions

Step 1: Terrain Generation

1. **Start with Terrain Generation**: The `TerrainGen.py` file is responsible for generating the terrain.
    - The `TerrainGenerator` class is initialized with parameters like seed, max and min elevation, and the number of trees.
    - For consistency, we are keeping seed = 125 for out experimentation and results. You can also change the max and min elevation in `TerrainGenerator` class parameters.
    - The `generate_terrain_with_trees()` method creates a terrain matrix that represents the generated terrain with elevations and obstacles (trees).

Step 2: Agent Initialization

2. **Initialize the Agent**: In the `Agent.py` file, an `Agent` class is defined.
    - Create an instance of the `Agent` class, specifying the start position, step size, and other parameters.
    - The Agent has init parameters like the starting point of the agent that can be edited, and also the details regarding the step size for the RRT and also the Iteration limit for the RRT and the Agents Discovery radius during exploration phase.
    - The agent is responsible for exploring the terrain and interacting with the environment.

Step 3: Path finding with RRT

3. **Implemented RRT for Path finding**: The `RRT.py` file contains the `RRT_Exp` class.
    - Initialize this class with the terrain matrix, start and goal points, and the agent.

- Use methods like `expand_tree()` and `find_path()` to generate and find a path from the start to the goal.
- There are multiple other functions that help in the rrt implementation.

Step 4: Running the Main Simulation

4. **Execute the Main Simulation**: The `main.py` file integrates all components.
   - Run this file to start the simulation.
   - It initializes the terrain and the agent, sets up the simulation environment, and handles GUI interactions.
   - Initially, a plot will appear with the entire terrain as Figure 1.

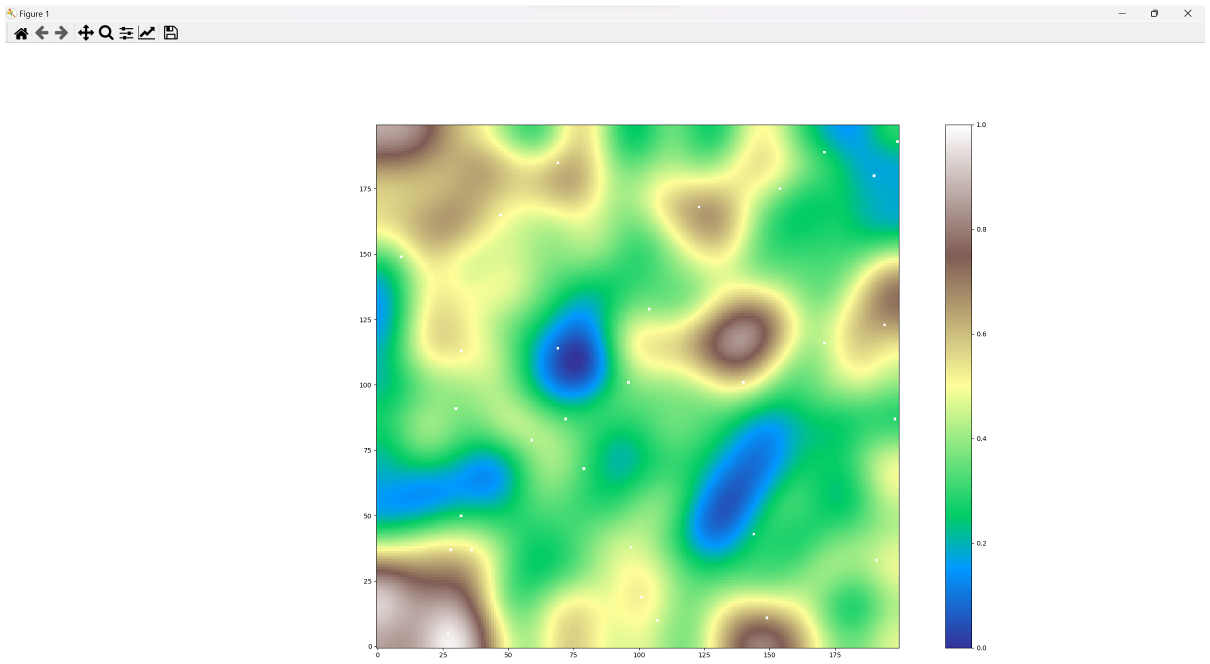


Figure 1: Terrain Generated

- Next if you press any of the arrow keys it will go into the exploration mode with the point Agent at the starting location as red dot with its discovery radius as initialized.
- Pressing Up key drives the car up but the updation is key based so the velocity to the agent is provided but unless you hold the key or keep pressing it the updated position will not appear as Figure 2.
- The boundaries doesn not stop the agent but don't add onto the exploration matrix to avoid cost matix bounds and still allow for bette ragent control to avoid losing the agent in exploration.
- Watch the agent as it explores the terrain.
- Once you have explored as needed, press "x" key on the keyboard to end the exploration phase (Figure 3).

Step 5: Interaction and Visualization

5. **Interact with the Simulation after exploration**:

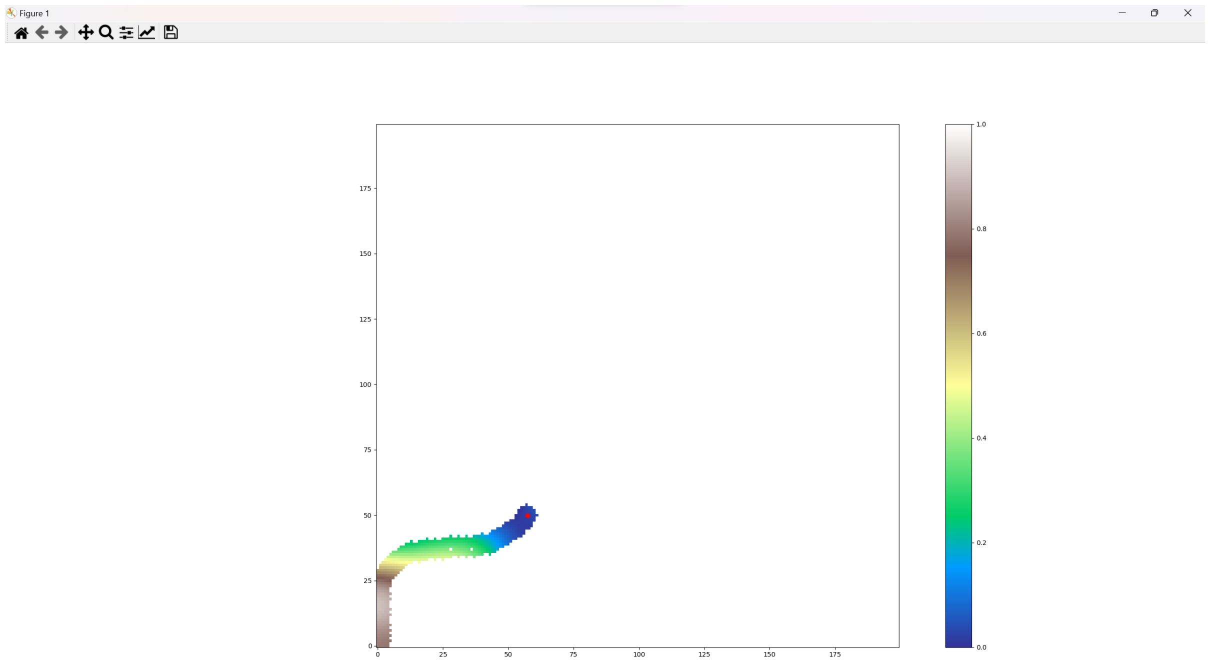
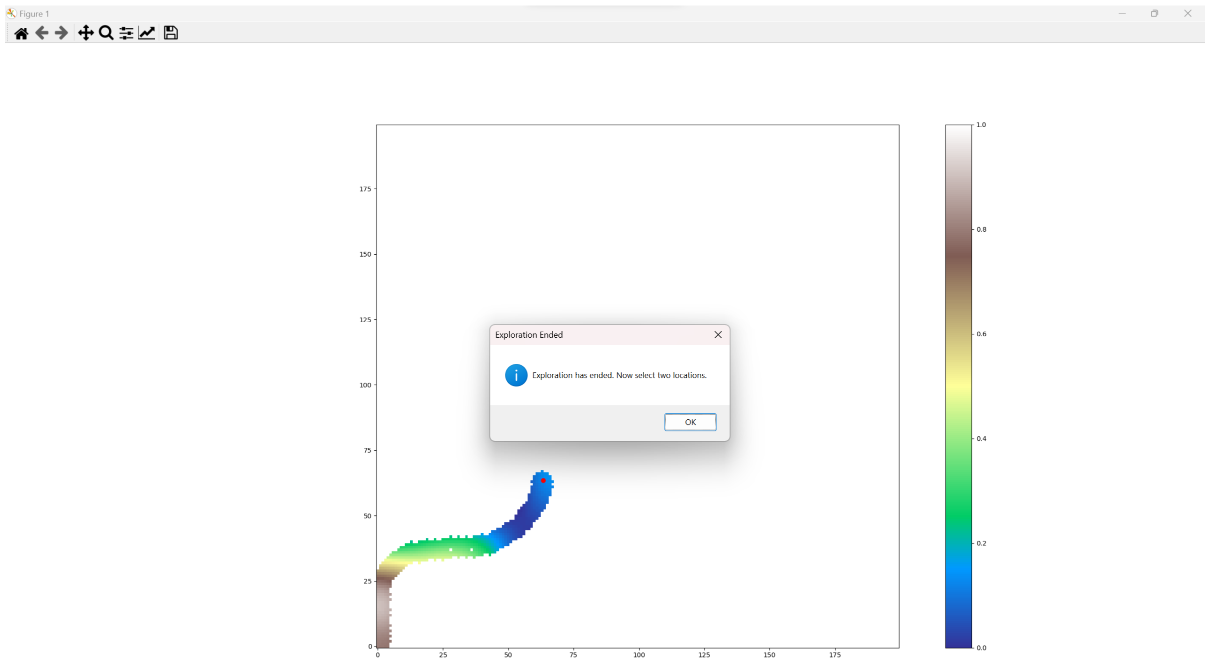


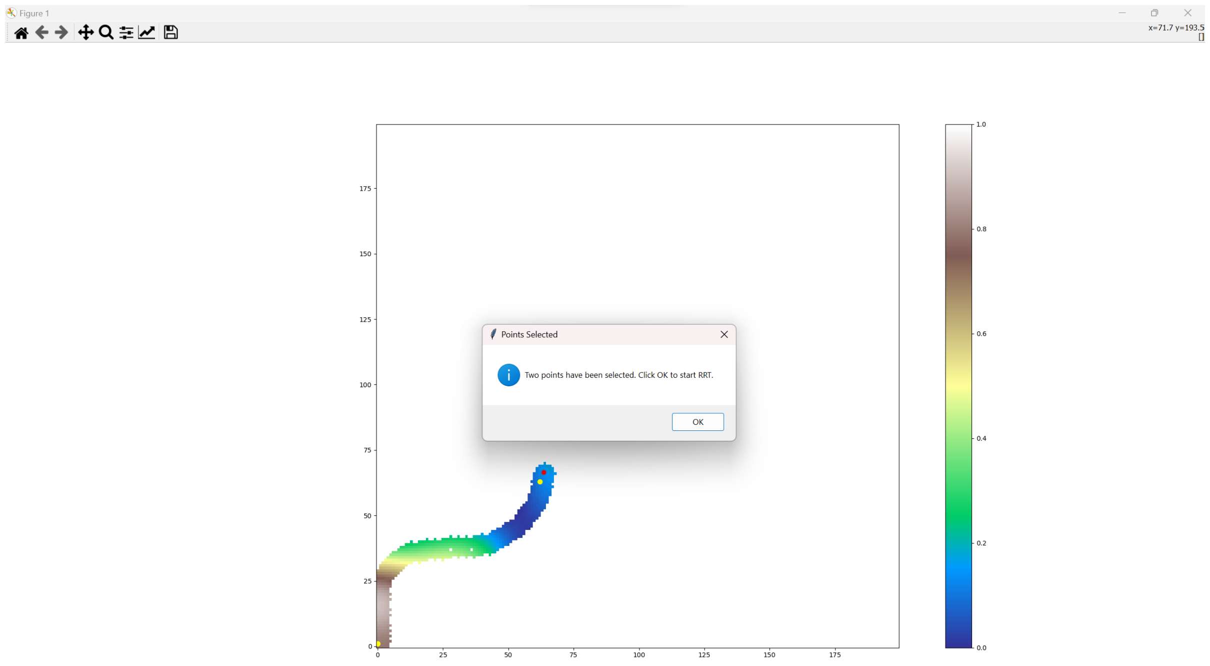
Figure 2: Terrain Exploration



Exploration has ended. Now select two locations.

Figure 3: Exploration Ended

- Once the exploration has ended and you click Ok on the message box, you can then select 2 point using mouse pointer on the explored area which will be indicated on the Plot as yellow dots.
- These can only be selected on the explored region and not the unexplored regions and once these are selected, it triggers the RRT functionality (Figure 4).



Two points have been selected. Click OK to start RRT.

Figure 4: Start and End locations

- Select start and end points, and observe the agent's pathfinding in action (depending on the path and locations the pathfinding may take a while and also reruns as the iterations and step size may need tweaking which is a sort of limitation we need to address later).

Step 6: Analyzing the Output

6. **Analyze the Results**:
    - The final output is a visualization of the agent's exploration path and the path planned using RRT.
    - The Original plot gets updated with the explored Tree paths in blue and in the same location as the files it saves a file called `Exploration.gif` which has the animation of the agent traversal from the start to end in magenta colored path (Figure 5).
    - Analyse how different parameters (like terrain complexity, agent's step size, etc.) affect the path finding i.e. path avoid the brown-white peaks and prioritizes visiting the deep blue depths as marked in color bar (Figure 6).

Step 7: Modifications, Experimentation and troubleshooting

7. **Experimentation with Different Scenarios**:
    - Modify parameters in the `TerrainGenerator`, `Agent`, and `RRT_Exp` classes to see influence on the agent's behavior and path finding efficiency.
    - Experiment with different terrain types, agent capabilities, and goal locations.

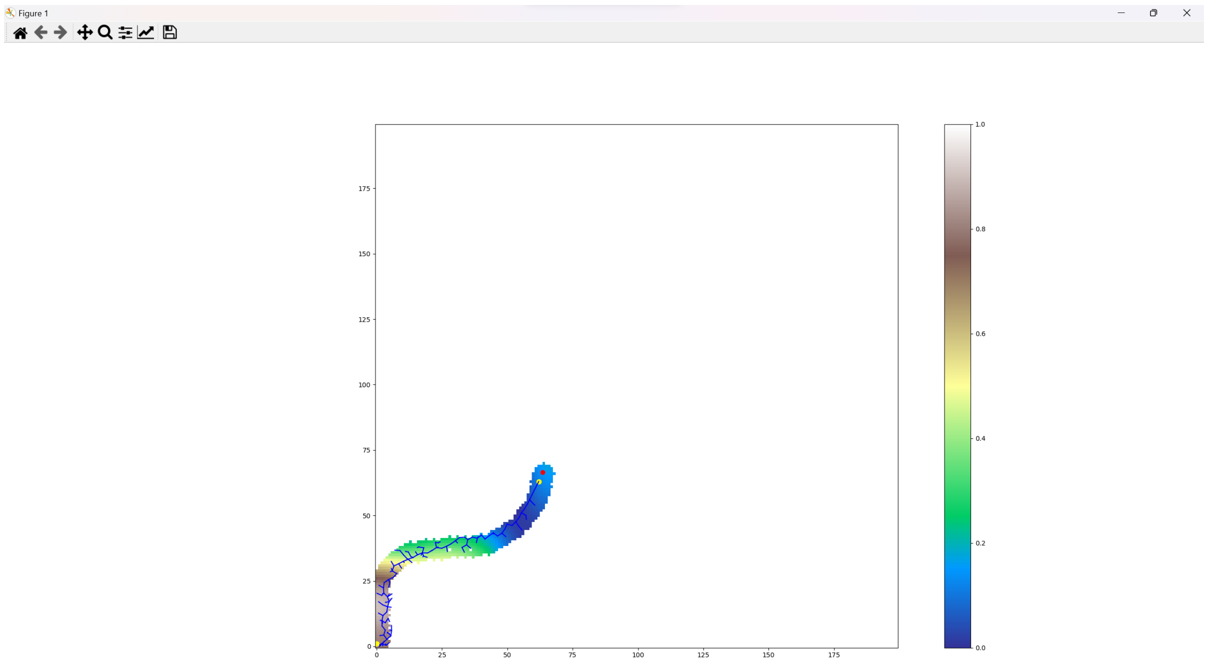


Figure 5: RRT path

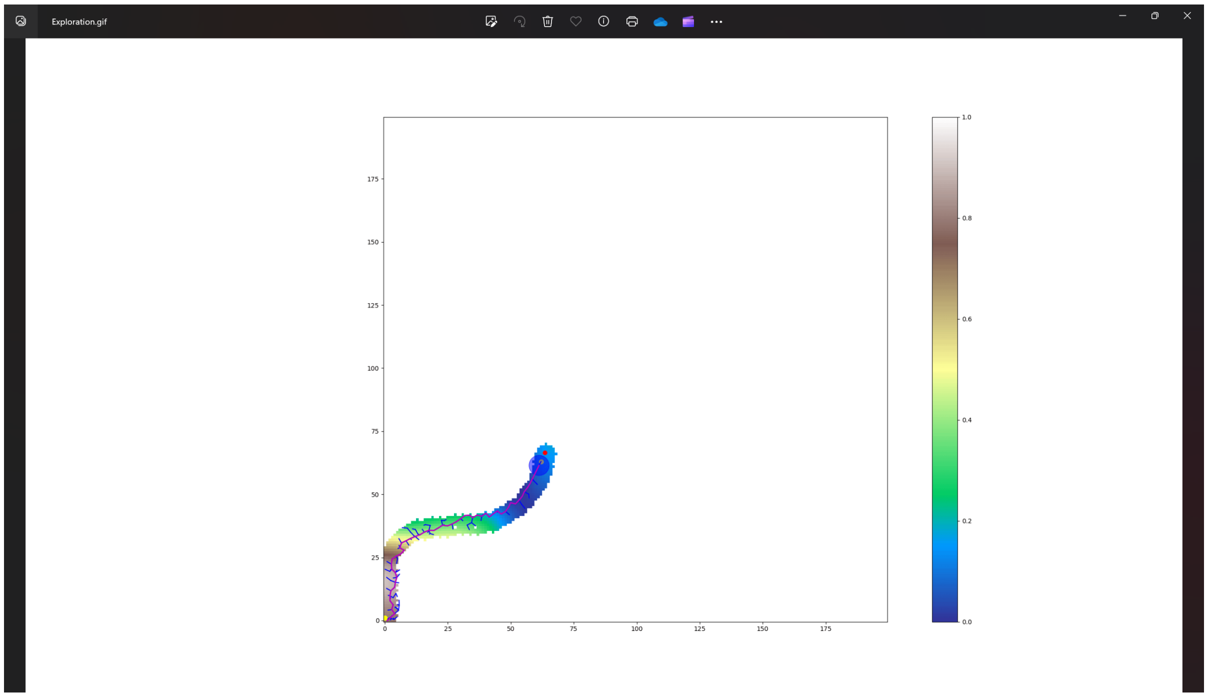


Figure 6: RRT Final path

- If at any point the code starts to only graph half the path or doesn't converge, please increase the step size and/or Iteration count for it to allow more processing capability for it to converge. Given the run time will also increase by a significant chunk by doing this.

Conclusion

By following these steps, we can effectively run and interact with the Autonomous Agent for terrain exploration. Each component plays a crucial role in simulating a realistic exploration and path finding scenario. Experiment with different configurations to explore the capabilities of your autonomous agent in varying environments.