Graph-Based Motion Planning for Autonomous Vehicle with Satellite Communication

Exercise 1

EE60216: Motion Planning and Control of Unmanned Vehicles

Full Name	Roll No.
Allen Emmanuel Binny	21EC30067

Professor-in-charge: Prof. Bhargav Jha

Department of Electrical Engineering
Indian Institute of Technology Kharagpur
Kharagpur, West Bengal, India

February 7, 2025

Contents

1 | Problem Statement

An autonomous vehicle needs to navigate through a binary occupancy grid, with assistance from two satellites. The environment is represented as a 60x50 grid where each cell has a value of 0 (unoccupied) or 1 (obstacle).

1.1 | Part A: Satellite Path Planning

1. Two satellites, Sat1 and Sat2, observe the grid:
   - Sat1 can access grid points with x-coordinates in the range 0-40
   - Sat2 can access grid points with x-coordinates in the range 30-60
2. Satellites must compute the shortest path for the vehicle from its initial position to a specified final position.
3. Constraints:
   - Satellites cannot share complete maps
   - They can only share visited nodes from the overlapping part of the grid (x-coordinates 30-40)
   - Both satellites assume 8-point connectivity on the grid
4. Algorithm requirements:
   - Sat1 and Sat2: Implement Dijkstra's algorithm

1.2 | Part B: Autonomous Vehicle Path Planning

1. Vehicle State Space and Dynamics:
   - Define the state space, action space, and state-transition equation for the vehicle
   - The vehicle can move only in forward, backward, left, and right directions
2. A* Algorithm Implementation:
   - Account for transmission delay (vehicle can move up to 30 units Manhattan distance during delay)
   - Implement A* algorithm for the vehicle to find the shortest path from its new position to the goal
3. Heuristics Analysis:
   - Implement and analyze three heuristics:
     a) Manhattan distance to the goal point
     b) Euclidean distance to the goal point
     c) Custom heuristic using Euclidean distance to the satellite-reported path and cost-to-reach the goal
   - Explain the validity of each heuristic
4. Performance Evaluation:
   - Report the efficiency of all implemented search algorithms
5. Visualization:
   - Provide Matlab plots highlighting the computed paths by each satellite and the vehicle's final path
   - Implement failure reporting if no path is found

2 | Part A: Satellite Path Planning

2.1 | Environment Setup

We implement a binary occupancy grid as described in the problem statement, using a 60x50 grid with randomly placed obstacles. We obtain the following as user inputs:

1. Number of Obstacles in the path
2. Initial Position
3. Final Position

2.2 | Satellite Algorithms

Both satellites assume 8-point connectivity on the grid and uses Dijkstra's Algorithm.
For implementation purpose, we have used the same function but provided both the functions with different x-limits to denote if its satellite 1 or 2.

2.3 | Information Sharing

Satellites can only share visited nodes from the overlapping part of the grid (x-coordinates 30-40). We have created a queue structure for our implementation that stores the visited nodes in the order they are visited. This order is considered as the nodes visited earlier in the queue can be associated with having lower costs. We store all these visited nodes in order and search for the lowest possible combination. This will guarantee that a possible path is found and the most optimal possible path is found.

Due to implementation limitations in MATLAB, this was not implemented; however, one possible method was to communicate the constant stream of the queue with each other and stop as soon as we found a common node.

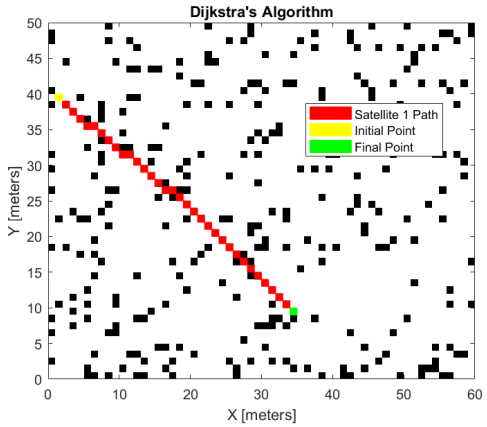
2.4 | Path Computation

We implement a custom function that combines the partial paths computed by each satellite, ensuring a continuous path from the initial position to the final position. This is done in 4 cases:

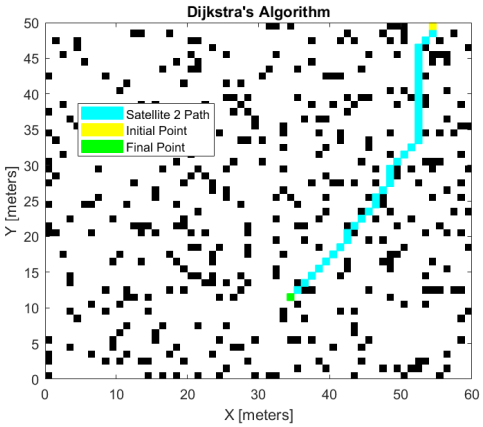
1. **Start and Goal within Sat1:** We compute the path in Dijkstra from start to goal.
2. **Start and Goal within Sat2:** We compute the path in Dijkstra from start to goal.
3. **Start within Sat1 and Goal within Sat2:** We start 2 Dijkstra paths from Start and Goal using the two satellites and find the point of intersection as mentioned in (2.3).
4. **Start within Sat2 and Goal within Sat1:** We start 2 Dijkstra paths from Start and Goal using the two satellites and find the point of intersection as mentioned in (2.3).

2.5 | Plots for Dijkstra

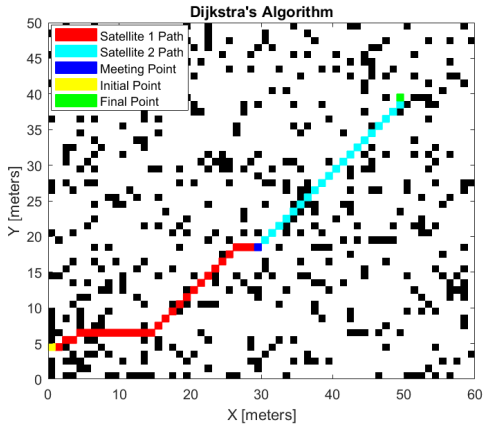
1. Case 1:
   - **No of Obstacles:** 300
   - **Initial Point:** [2, 40]
   - **Final Point:** [35, 10]
2. Case 2:
   - **No of Obstacles:** 400
   - **Initial Point:** [55, 50]
   - **Final Point:** [35, 12]
3. Case 3:
   - **No of Obstacles:** 500
   - **Initial Point:** [1, 5]
   - **Final Point:** [50, 40]
4. Case 4:
   - **No of Obstacles:** 600
   - **Initial Point:** [50, 50]
   - **Final Point:** [12, 14]



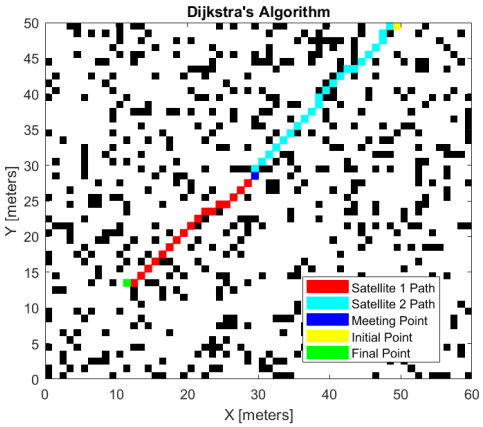
(a) Case 1



(b) Case 2



(c) Case 3



(d) Case 4

Figure 2.1: Four cases of obstacle avoidance

3 | Part B: Autonomous Vehicle Path Planning

3.1 | Vehicle State Space and Dynamics

- **State space**($\mathbb{X}$): The state space $\mathbb{X}$ represents all possible positions of the vehicle:

$$\mathbb{X} = \{(x, y) \in \mathbb{N}^2 : 1 \leq x \leq 60, 1 \leq y \leq 50\}$$

- **Action space**($\mathbb{U}$): The action space $\mathbb{U}$ consists of four possible movements:

$$\mathbb{U} = \{(1, 0), (-1, 0), (0, 1), (0, -1)\}$$

  representing right, left, up, and down movements respectively.

- **State-transition equation**: For a state $\mathbf{x}_t = [x_t, y_t]^\top$ and action $\mathbf{u}_t \in \mathbb{U}$, the state-transition equation is:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{u}_t$$

  subject to the constraints:

$$\mathbf{x}_{t+1} \in \mathbb{X} \quad \text{(i.e., the vehicle cannot move outside the grid or onto an obstacle)}$$

  where:

$$\mathbf{x}t, \mathbf{x}t + 1 \in \mathbb{X}, \quad \mathbf{x}_{t+1} \notin \mathbb{O}$$

3.2 | Heuristics

We implement and analyze three heuristics:

1. Manhattan distance to the goal point
2. Euclidean distance to the goal point
3. Custom heuristic using Euclidean distance to the satellite-reported path and cost-to-reach the goal

3.3 | Analysis of Heuristics

3.3.1 | Manhattan Distance(MD)

Valid heuristic as it never overestimates the true cost to the goal in a grid-based environment with 4-directional movement.
If current point $(\mathbf{x_n}) = [x, y]$, goal point $(\mathbf{x_f}) = [x_g, y_g]$

$$\begin{aligned} MD &= \|\mathbf{x_n} - \mathbf{x_f}\| && (3.1)\\ &= |x - x_g| + |y - y_g| && (3.2) \end{aligned}$$

3.3.2 | Euclidean Distance(ED)

Valid heuristic as it also never overestimates the true cost, but may be less informative than Manhattan distance for grid-based movement. If current point $(\mathbf{x_n}) = [x, y]$, goal point $(\mathbf{x_f}) = [x_g, y_g]$

$$\begin{aligned} MD &= \|\mathbf{x_n} - \mathbf{x_f}\|_2 && (3.3)\\ &= \sqrt{(x - x_g)^2 + (y - y_g)^2} && (3.4) \end{aligned}$$

3.3.3 | Custom Heuristic

The Heuristic is valid as the first part of the cost is to take the Euclidean Distance to the closest point in the a-star path, which is an underapproximation. We then add the cost to the goal which is the exact length of the path taken by A-star algorithm. Thus the sum of these costs are also an underapproximation of the total path that can be taken, making it a valid heuristic.

3.4 | Algorithm Implementation

3.4.1 | Exploration Phase

- The Algorithm conducts the exploration in accordance with the heuristic. This means that the movement to the goal has been sorted according to how the satellite calculates the path.
- This is essential in the path rejoining step as it ensures that the overall path we take will be as close to the one supplied by the satellite and can be easily be reached.
- In case when we are using the custom heuristic for the reaching phase, we use the Euclidian distance as a heuristic as there is no path from satellite obtained so far.
- We noticed in cases of a high number of obstacles when some nodes were found to be blocked by obstacles on 3 sides. In such cases, if the robot reaches on of these nodes, it will keep oscillating between these nodes which is not ideal. We cannot include the condition that the robot cannot go to the visited node as this will make the robot terminate in this position. Hence we add weight according to the number of times of node is visited to reduce the likelihood of visiting an already visited node.
- An interesting edge case to consider here is the fact that we cannot observe the obstacles which are diagonal position although they are neighbours. This limits the path in certain cases where the obstacle in the diagonal position blocked a more optimal path that could have been taken.

3.4.2 | Planning Algorithm from Satellite

- We plan the path from the start point to the goal using the A* algorithm. The heuristic here is Manhattan or Euclidean distance to the goal point(in the case of custom heuristics, we also use the Euclidean distance).
- The cost of each node will be calculated as:
  Total Cost = Cost to go + Cost to come
- Each path is taken from the initial point to the goal and passed on to the robot after taking 30 Manhattan distances.

3.4.3 | Reaching the A-star Path

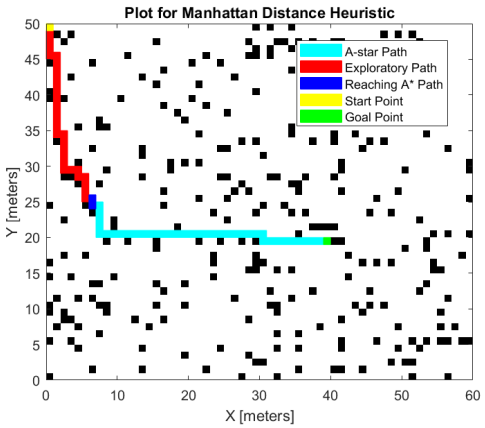
- Here is where we use the heuristic method for a greedy-based approach to reach the A-star algorithm.
- For the Manhattan and Euclidean distances, we find the closest point on the A-Star Path and navigate to that point greedily until we reach that point.
- The custom heuristic should ideally perform well here. This is because we have a globally optimal path obtained from the satellite and want to get to this globally optimal path. However, the greedy algorithm is only locally optimal and may not converge. However, with this heuristic, we aim to optimally converge to the optimal path. Especially in cases when the offset from the A-star algorithm is observed, then this heuristic will perform very well.

3.5 | Plots for A-Star Algorithm

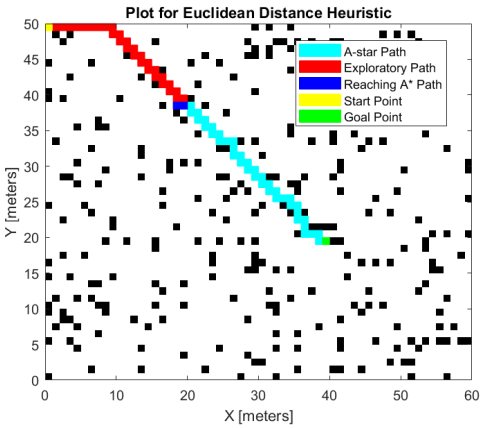
1. Case 1:
   - **No of Obstacles:** 300
   - **Initial Point:** [1, 50]
   - **Final Point:** [40, 20]
2. Case 2:
   - **No of Obstacles:** 750
   - **Initial Point:** [3, 15]
   - **Final Point:** [50]

```
Enter numberer of obstacles (positive integer): 300
Enter initial point [x y] (1 ≤ x ≤ 60, 1 ≤ y ≤ 50): [1 50]
Enter final point [x y] (1 ≤ x ≤ 60, 1 ≤ y ≤ 50): [40 20]
Initalised a proper map
Using Manhattan Distance as Heuristic
Path Cost for the current method: 72
Path Cost for the Ideal ASTAR Algorithm: 70
############################################################
Using Euclidian Distance as Heuristic
Path Cost for the current method: 72
Path Cost for the Ideal ASTAR Algorithm: 70
############################################################
Using Custom Heuristic
Path Cost for the current method: 72
Path Cost for the Ideal ASTAR Algorithm: 70
############################################################
```

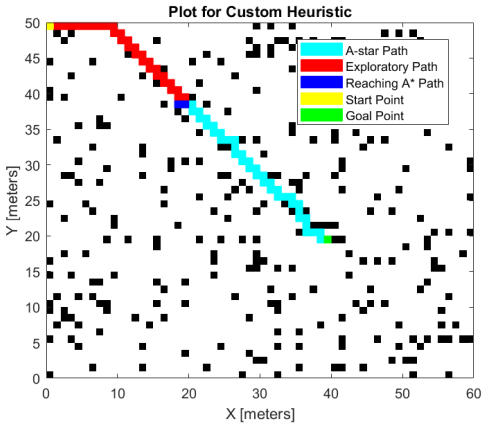
Figure 3.1: Case 1 Statistics with 300 Obstacles



(a) Manhattan Distance



(b) Euclidean Distance

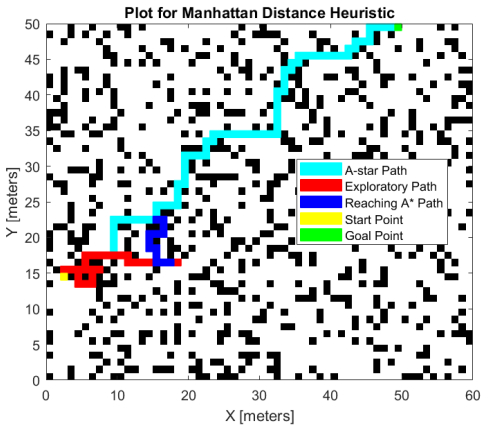


(c) Custom Heuristic

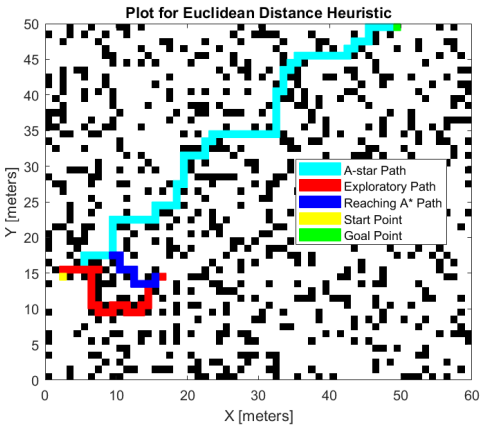
Figure 3.2: Case I with 300 obstacles

```
Enter number of obstacles (positive integer): 750
Enter initial point [x y] (1 ≤ x ≤ 60, 1 ≤ y ≤ 50): [3 15]
Enter final point [x y] (1 ≤ x ≤ 60, 1 ≤ y ≤ 50): [50 50]
Initalised a proper map
Using Manhattan Distance as Heuristic
Path Cost for the current method: 105
Path Cost for the Ideal ASTAR Algorithm: 83
############################################################
Using Euclidian Distance as Heuristic
Path Cost for the current method: 115
Path Cost for the Ideal ASTAR Algorithm: 83
############################################################
Using Custom Heuristic
Path Cost for the current method: 115
Path Cost for the Ideal ASTAR Algorithm: 83
############################################################
```

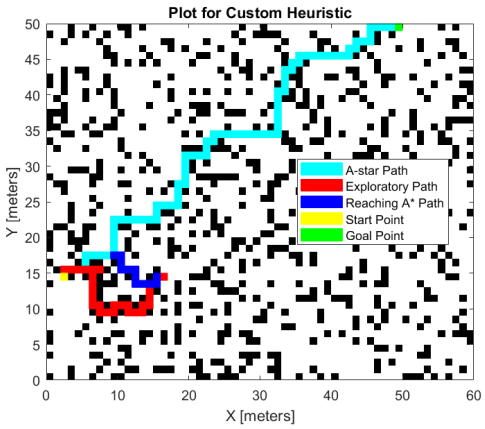
Figure 3.3: Case 2 Statistics with 750 Obstacles



(a) Manhattan Distance



(b) Euclidean Distance



(c) Custom Heuristic

Figure 3.4: Case II with 750 obstacles

4 | Efficiency Comparison

4.1 | Dijkstra Algorithm

Algorithm Design

- Implementation uses grid-based Dijkstra's algorithm with 8-point connectivity
- Optimized for satellite-specific regions (x=[1,40] for Sat1, x=[30,60] for Sat2)
- Shared region handling (x=[30,40]) minimizes redundant computations

Key Implementation Features

- **Region-Based Processing:**
  - Restricts search space to satellite-specific regions
  - Reduces computational complexity by limiting node exploration
  - Implements efficient boundary checking for valid moves
- **Path Construction:**
  - Uses parent node tracking for efficient path reconstruction
  - Implements reverse path construction for bidirectional searches
  - Optimizes memory usage by storing only necessary path information
- **Shared Region Optimization:**
  - Implements efficient node sharing between satellites
  - Uses joinPaths function for optimal connection point selection
  - Minimizes communication overhead between satellites (Can be made more effective use parallel processing of the two satellites and communication of visited nodes in shared region)

Computational Complexity

- Time Complexity: $O(V \log V + E)$ where:
  - V: Number of vertices in satellite's region
  - E: Number of edges (8-connectivity)
- Space Complexity: $O(V)$ for storing:
  - Parent nodes
  - Distance values
  - Visited nodes

Optimization Techniques

- Early termination when target is reached
- Efficient obstacle checking using occupancy grid
- Optimized path merging in shared regions
- Memory-efficient parent node tracking

4.2 | A* Algorithm

Algorithm Design

- Implements A* with dual heuristic options:
  - Manhattan distance heuristic
  - Euclidean distance heuristic
- Handles 30-step initial movement phase
- Integrates with satellite-provided path information
- Uses 4-directional connectivity: $U = \{[-1, 0], [0, -1], [1, 0], [0, 1]\}$

Key Implementation Features

- **Movement Control:**
  - Implements boundary checking: $1 \leq x \leq 60$, $1 \leq y \leq 50$
  - Real-time obstacle avoidance using occupancy grid
  - Handles transmission delay with 30-step initial movement phase
- **Heuristic Functions:**
  - Manhattan Distance: $h_1(n) = |g - s|$
  - Euclidean Distance: $h_2(n) = \|g - s\|_2$, where g is the goal point, s is the start point
  - Cost function: $f(n) = g(n) + h(n)$ where:
    - $g(n)$: Cost from start to current node
    - $h(n)$: Estimated cost to goal using chosen heuristic
- **Path Integration:**
  - Dynamic merging of initial movement path with A* computed path
  - Closest point computation using squared Euclidean distance
  - Smooth transition between movement phases

Computational Complexity

- **Time Complexity:** $O(b^d)$ where:
  - b: Branching factor (4 for 4-directional movement)
  - d: Depth of solution
- **Space Complexity:** $O(b^d)$ for storing:
  - Open list
  - Closed list
  - Path information

Optimization Techniques

- Early termination upon reaching goal state
- Efficient obstacle checking using binary occupancy grid
- Priority-based movement selection using sorted indices
- Dynamic path merging with satellite-provided information

Trade-offs

- **Manhattan vs Euclidean heuristic:**
  - Manhattan: Better suited for grid-based movement, faster computation. For a large number of obstacles, we observe, on average, that the path cost is minimal. This behaviour however is not observed at low number of obstacles.
  - Euclidean: More accurate distance estimation, higher computational cost
- The Initial 30-step movement phase may lead to suboptimal paths but ensures real-time response
- Memory usage increases with map size (60×50) and obstacle count

5 | Conclusion

Through these methodologies, we were able to understand the complexities of BFS, Dijkstra's algorithm with the intercacies involved with their implementation.