IE415

Control Of Autonomous Systems

:: Project ::

Algorithms For Path Planning Of the Robot

Name: Heman Chauhan

Student ID: 202201267

Abstract

The project "Algorithms for Path Planning of the Robot" explores various path planning algorithms for autonomous robots in a grid-based environment. The primary focus is on implementing and visualizing different techniques to navigate robots from a start point to a goal while avoiding obstacles. Four different algorithms are implemented and compared: Dijkstra's algorithm, A*, Rapidly Exploring Random Tree (RRT), and Potential Field Path Planning. Each algorithm is designed to offer solutions for optimal and collision-free navigation, with visualizations provided to demonstrate their performance in real-time. The project aims to contribute to enhancing the efficiency and flexibility of robot path planning in dynamic environments, making it relevant for autonomous robotics applications in fields like navigation, exploration, and automation.

Algorithms

1. A* Algorithm

Overview:

A* (pronounced "A-star") is a popular and widely-used algorithm in pathfinding and graph traversal, known for its efficiency and optimality. It is typically employed in scenarios where a robot, game agent, or AI needs to find the shortest path between a starting point and a goal while avoiding obstacles. The algorithm combines the benefits of both Dijkstra's Algorithm (which guarantees the shortest path) and Greedy Best-First Search (which prioritizes reaching the goal quickly).

Key Concepts:

1. Nodes and Graph:

- **The environment is represented as a grid or graph, where each point is a "node."**
- **The goal is to find the least-cost path from a start node to a goal node.**

2. Cost Functions:

- **A* uses a cost function, f(n), to determine the most promising node to explore at each step. The function combines two components:**
  - **g(n): The cost of the path from the start node to the current node n. This value represents the known distance from the start.**
  - **h(n): The heuristic estimate of the cost from the current node n to the goal. It provides an estimate of the remaining distance to the goal and is problem-specific (e.g., Euclidean or Manhattan distance).**

Thus, the total cost function is:

$$f(n)=g(n)+h(n)$$

A* aims to minimize this total cost and prioritize nodes that are likely to lead to the shortest path.

3. Heuristic Function:

- **The heuristic function h(n) estimates the remaining cost from node n to the goal. It must be admissible, meaning it never overestimates the true cost.**
- **Common heuristics include:**
  - **Manhattan distance (for grids with only horizontal and vertical movements).**
  - **Euclidean distance (for diagonal movements and when movement is allowed in any direction).**

4. Open and Closed Lists:

- **Open list: A priority queue (often implemented using a min-heap) that stores nodes to be evaluated, based on their f(n) values. Nodes in the open list are candidates for expansion.**
- **Closed list: A set of nodes that have already been evaluated and expanded. Once a node is moved to the closed list, it is not revisited.**

5. Algorithm Process:

- **Initialization: Add the start node to the open list with an initial cost of 0.**
- **Expansion: Continuously expand nodes by exploring their neighbours, calculating their f(n) values, and placing them into the open list.**
- **Goal Check: If the current node is the goal, the algorithm has successfully found the path, and the path is reconstructed by tracing the parent nodes back to the start.**
- **Termination: The algorithm terminates when the goal is reached or the open list is empty, in which case no path exists.**

6. Optimality and Completeness:

- **A* is optimal (i.e., it guarantees finding the shortest path) if the heuristic function is admissible (it never overestimates the true cost) and consistent (the**

estimated cost between two nodes is always less than or equal to the cost of reaching the second node via the first node).

- **A* is also complete, meaning it will always find a solution if one exists, provided there are no obstacles blocking the path and the search space is finite.**

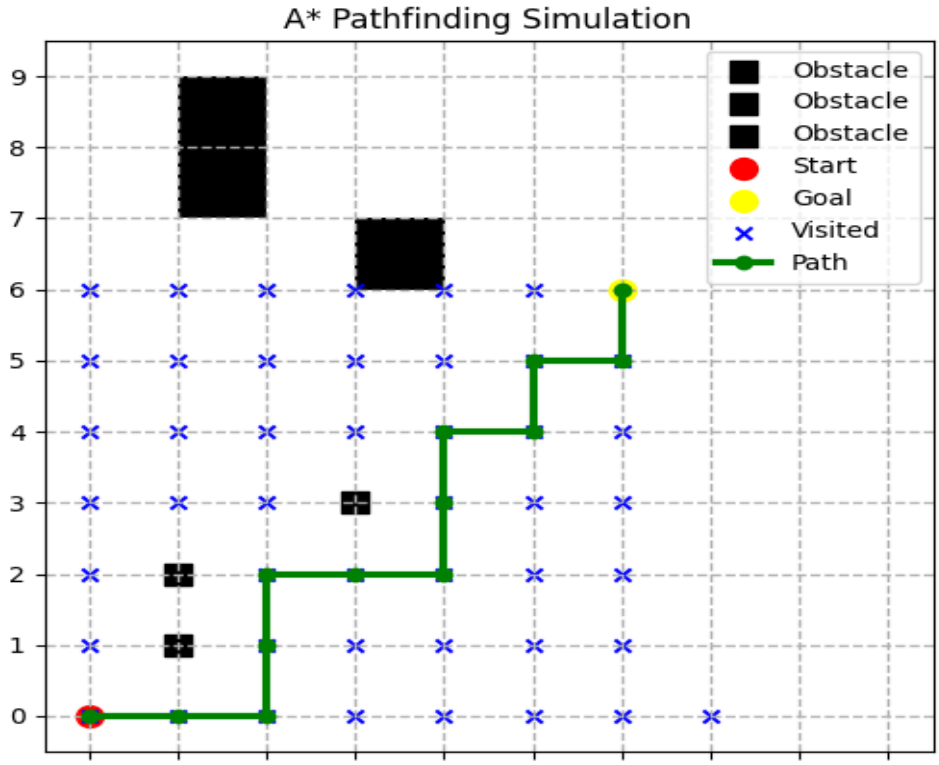
Advantages:

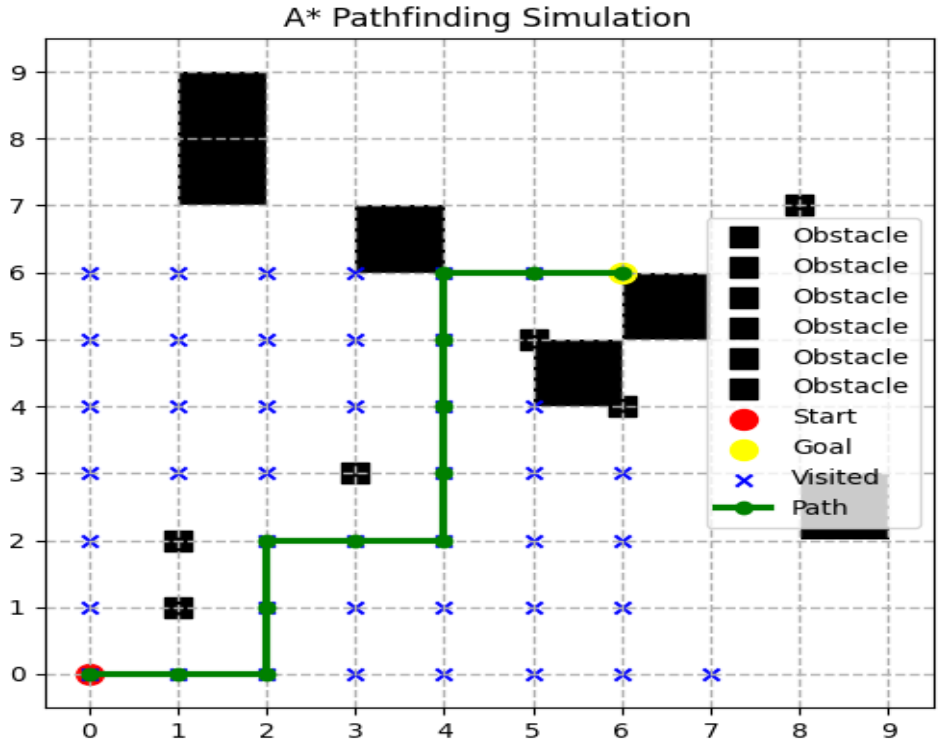
- **Optimality: When the heuristic is admissible and consistent, A* guarantees the shortest path.**
- **Flexibility: It can be applied to various grid-based or graph-based pathfinding problems, including those with obstacles and dynamic environments.**
- **Efficiency: A* is more efficient than exhaustive search methods, such as Dijkstra's algorithm, because it prioritizes exploring nodes that are more likely to lead to the goal, based on the heuristic.**

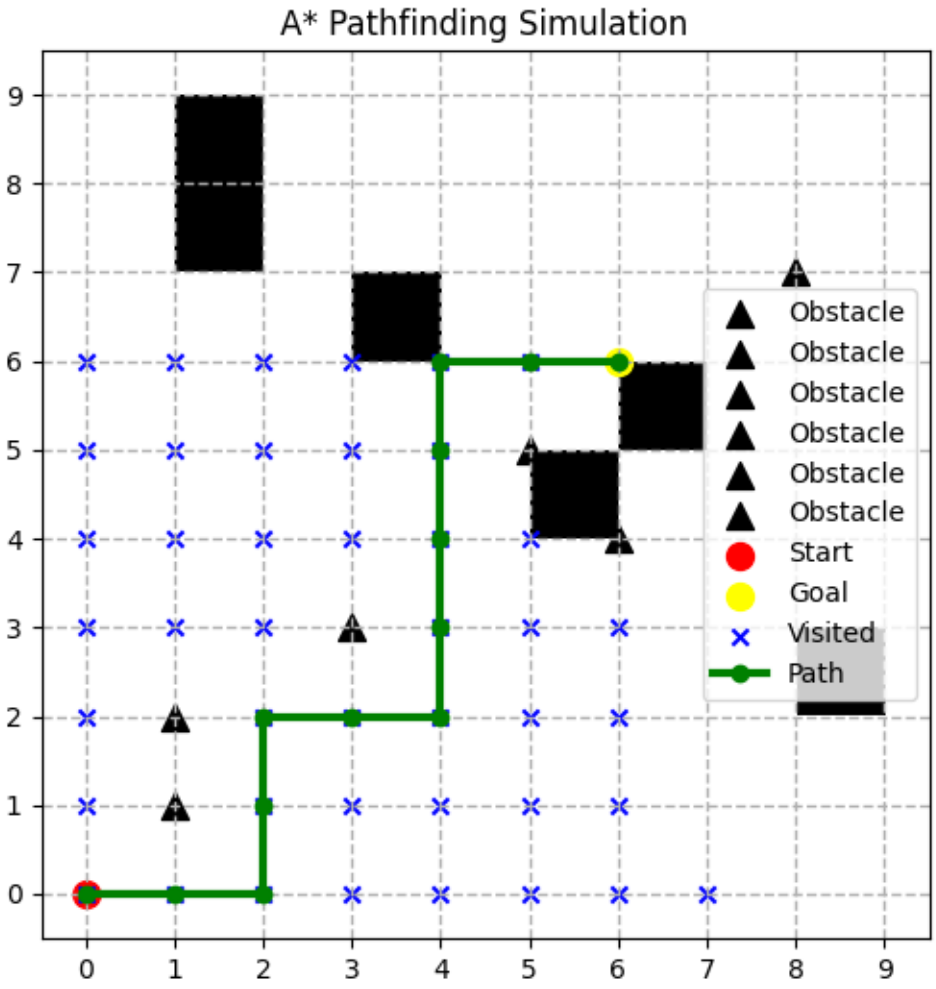
Limitations:

- **Heuristic Dependency: The performance of A* is highly dependent on the heuristic function. A poor heuristic can result in excessive exploration, leading to slower performance.**
- **Memory Usage: A* can be memory-intensive, especially in large environments, as it needs to store both the open and closed lists of nodes.**

Stimulations:







2.RRT Algorithm

Overview:

The Rapidly-exploring Random Tree (RRT) algorithm is a popular motion planning technique used primarily for robotic pathfinding in high-dimensional spaces, such as in environments with obstacles or in continuous spaces. It is particularly useful for applications where the robot needs to navigate through a complex environment with

many obstacles while seeking to find a feasible path from a start to a goal configuration.

Key Concepts:

1. Tree-Based Structure:

- **RRT is a tree-based algorithm, meaning it builds a tree starting from an initial node (usually the robot's starting position). The tree grows incrementally by randomly selecting points in the space and expanding the tree toward those points.**
- **Each node in the tree represents a configuration of the robot (position and orientation), and the edges between nodes represent valid movements.**

2. Random Sampling:

- **RRT uses random sampling to explore the space. At each iteration, it randomly selects a point in the configuration space (either near the goal or in an arbitrary direction) and attempts to grow the tree toward it.**
- **The randomness helps in rapidly exploring large areas of the space, particularly in complex environments with obstacles.**

3. Expansion:

- **The algorithm iteratively expands the tree by finding the closest node in the tree to the randomly selected point.**

Then, it attempts to generate a new node that is a small step from the selected node toward the randomly chosen point.

- **The new node is added to the tree if the movement is valid (i.e., if it doesn't collide with obstacles or violate constraints).**

4. Goal Biasing (optional):

- **Goal Biasing is a common technique used to accelerate the algorithm's convergence toward the goal. Instead of selecting random points uniformly throughout the entire space, the algorithm occasionally selects the goal position as the random point, encouraging the tree to grow toward the goal more quickly.**

5. Collision Checking:

- **A critical step in RRT is ensuring that the path between nodes does not intersect with any obstacles. This is achieved by performing collision checking during the tree expansion process.**
- **If an edge between two nodes intersects an obstacle, it is discarded, and the tree does not expand in that direction.**

6. Path Construction:

- **Once the tree reaches or connects to the goal configuration (or a state close to the goal), the algorithm constructs the path by tracing the nodes from the goal to the start node.**

- **The path is typically smoothed or optimized after construction to reduce sharp turns or inefficiencies.**

Algorithm Process:

Algorithm 1: RRT* pseudo code

Input : x_{init}, X_{goal}
Output: Tree $T = (V, E)$

$V \leftarrow x_{init}$, $E \leftarrow \emptyset$;
for $i = 1, ..., N$ **do**
  $x_{rand} \leftarrow$ RandomSample(i);
  $x_{nearest} \leftarrow$ Nearest(T, x_{rand});
  $x_{new} \leftarrow$ Steer($x_{nearest}, x_{rand}$);
  if ObstacleFree($x_{nearest}, x_{new}$) **then**
    $V \leftarrow V \cup \{x_{new}\}$;
    $X_{near} \leftarrow$ Near(T, x_{new}, r);
    ChooseParent ($X_{near}, x_{nearest}, x_{new}, E$);
    Rewiring (X_{near}, x_{new}, E);
  end
end

1. Initialization:

- **Start with an initial node representing the robot's starting position.**
- **Add this node to the tree.**

2. Tree Expansion:

- **Randomly sample the configuration space to select a point.**
- **Find the closest node in the tree to this point.**
- **Attempt to expand the tree by moving a small step toward the random point and add this new node to the tree.**

3. Collision Checking:

- **Verify that the movement from the closest node to the new node does not intersect any obstacles. If it does, discard that movement and try another direction.**

4. Goal Reached:

- **If the tree reaches the goal configuration or gets sufficiently close, stop and construct the path from the goal to the start node.**

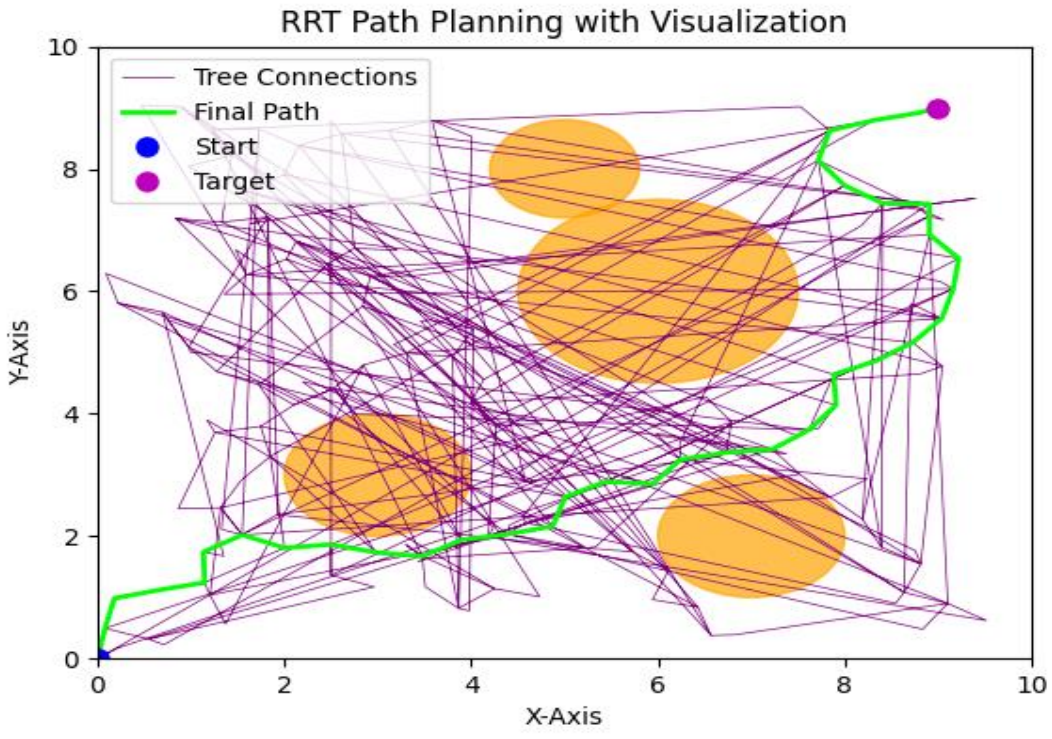
Advantages:

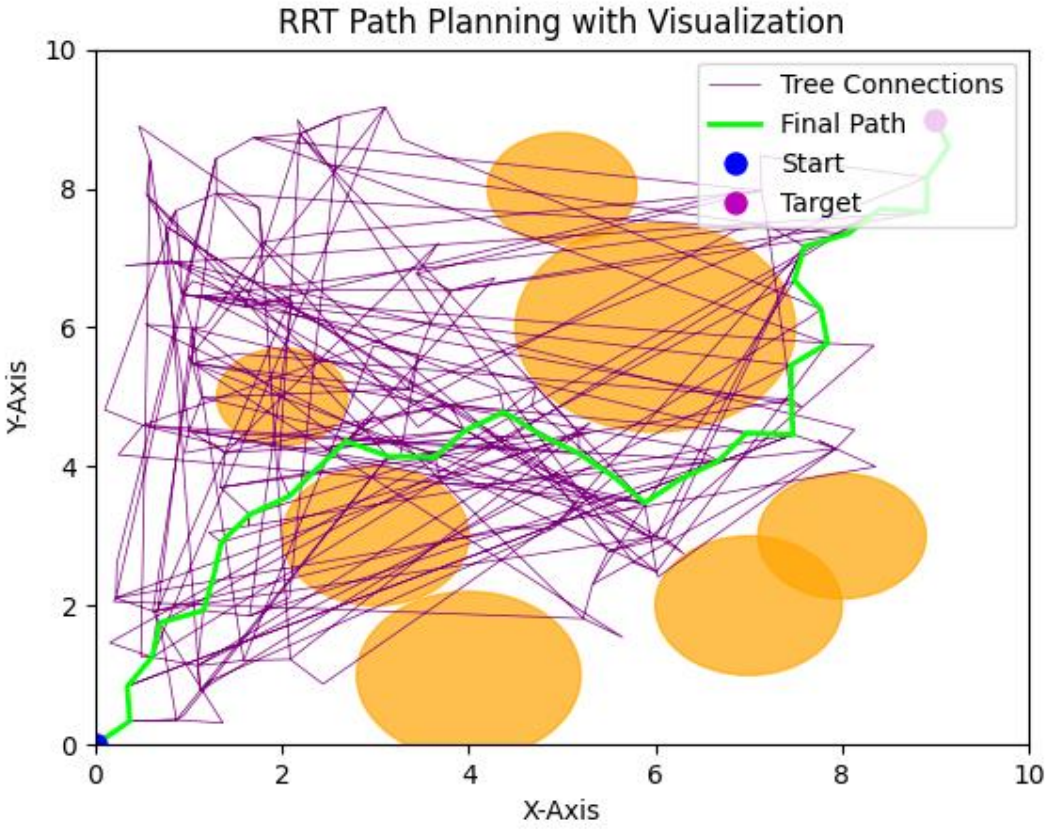
- **Efficient Exploration: RRT is capable of efficiently exploring large and high-dimensional spaces, making it well-suited for complex environments.**
- **Flexibility: It can handle non-holonomic constraints (e.g., vehicle dynamics) and arbitrary environments.**
- **Simplicity: The basic RRT algorithm is simple to implement and does not require explicit knowledge of the goal location.**

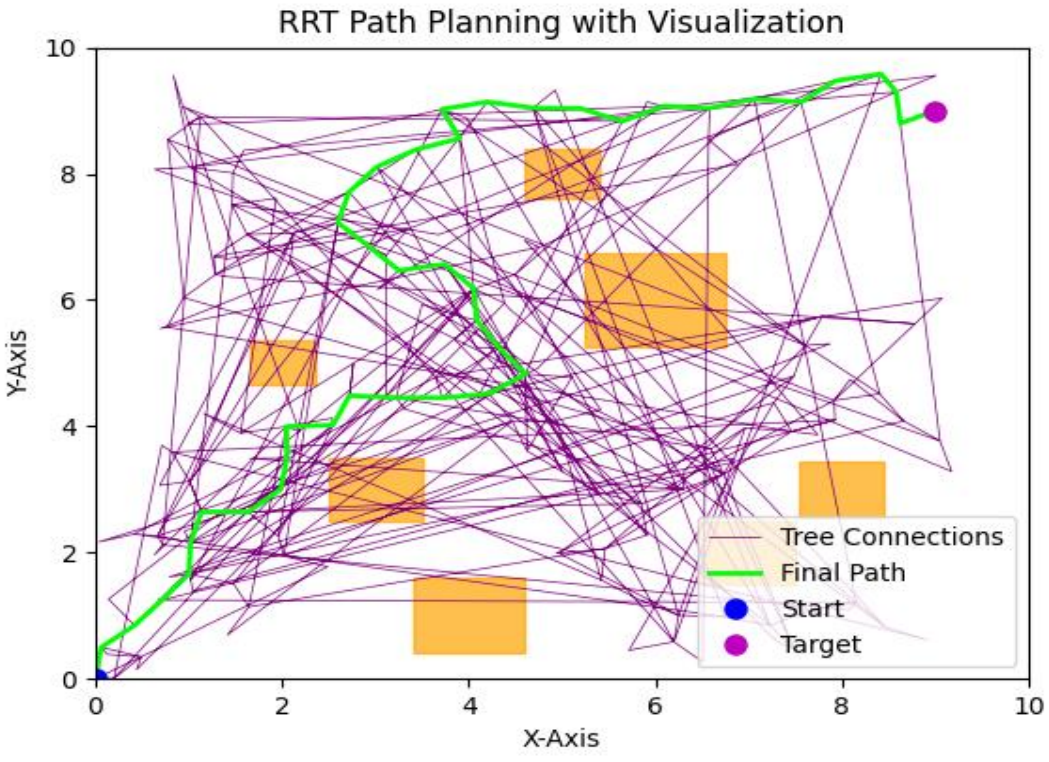
Limitations:

- **Suboptimal Paths: The paths generated by RRT are often not optimal. The algorithm focuses on exploring space quickly, rather than minimizing path length or smoothness. As a result, the paths can be jagged and inefficient.**
- **High-Dimensional Spaces: RRT struggles in very high-dimensional spaces due to the curse of dimensionality. The random sampling can become sparse in higher dimensions, making it difficult to explore the space effectively.**
- **Limited Convergence to Goal: The standard RRT does not guarantee that the goal will be reached unless the goal is biased toward in the sampling process. Without goal biasing or other techniques, the tree might never reach the goal.**

Stimulations:




RRT Path Planning with Visualization
Tree Connections
Final Path
Start
Target
Y-Axis
X-Axis
0
2
4
6
8
10


RRT Path Planning with Visualization
Tree Connections
Final Path
Start
Target
Y-Axis
X-Axis
0
2
4
6
8
10

3. Potential Field Algorithm

Overview:

The Potential Field Algorithm is a popular approach used for robot path planning and navigation, where the robot is guided by artificial forces that are generated from the environment. The algorithm is inspired by the idea of a "field" where the robot behaves like a particle affected by forces (both attractive and repulsive) within that field. These forces influence the robot's movement, allowing it to find a path toward the goal while avoiding obstacles. The algorithm is simple, intuitive, and works in continuous spaces, making it widely used in real-time applications.

Key Concepts:

1. Attractive Force (Goal Attraction):

- **This force attracts the robot toward the goal or target.**
- **It is typically modelled as an inverse distance force, meaning the closer the robot is to the goal, the stronger the force pulling it towards the goal.**
- **Mathematically, the attractive force is often computed as:**

$$F_{\text{attraction}} = -k_{\text{attraction}} \cdot (\text{position} - \text{goal})$$

➔ **where is $k_{\text{attraction}}$ the attraction gain, and position and goal are the current robot position and target position.**

2. Repulsive Force (Obstacle Avoidance):

- **This force repels the robot away from obstacles or barriers in the environment.**
- **It is modelled to be very strong when the robot is close to an obstacle and weaker as the robot moves away from the obstacle.**
- **The repulsive force is usually computed based on the inverse square of the distance from the obstacle and is limited to a certain range (repulsion radius).**
- **Mathematically, the repulsive force is often given by**

$$F_{\text{repulsion}} = k_{\text{repulsion}} \cdot \left(\frac{1}{\text{distance}} - \frac{1}{\text{repulsion radius}} \right) \cdot \frac{(\text{position} - \text{obstacle})}{\text{distance}^2}$$

➔ **where $k_{\text{repulsion}}$ is the repulsion gain, and the repulsion radius determines the effective range of the repulsive force.**

3. Total Force:

- **The total force acting on the robot at any point is the vector sum of the attractive and repulsive forces:**

$$F_{\text{total}} = F_{\text{attraction}} + F_{\text{repulsion}}$$

- **This total force is used to determine the robot's movement by updating its position at each step.**

Algorithm Steps:

1. **Initialize the robot's starting position and goal position.**
2. **Iterate:**

- **At each step, calculate the attractive force pulling the robot toward the goal.**
- **Compute the repulsive forces from nearby obstacles.**
- **Sum the attractive and repulsive forces to determine the total force.**
- **Move the robot in the direction of the total force (the robot's new position is updated).**
- **If the robot is within a threshold distance of the goal (goal tolerance), stop the algorithm as the goal has been reached.**

3. **Termination:**
    - **The robot reaches the goal if the total force causes it to stop within the goal's tolerance range.**

Advantages:

- **Simplicity: The algorithm is straightforward to implement, as it requires only basic force calculations and updates.**
- **Real-time: It can run in real-time with continuous adjustments to the robot's path as it moves.**
- **Flexibility: It can be adapted to different environments by adjusting the gains and distances for attraction and repulsion.**

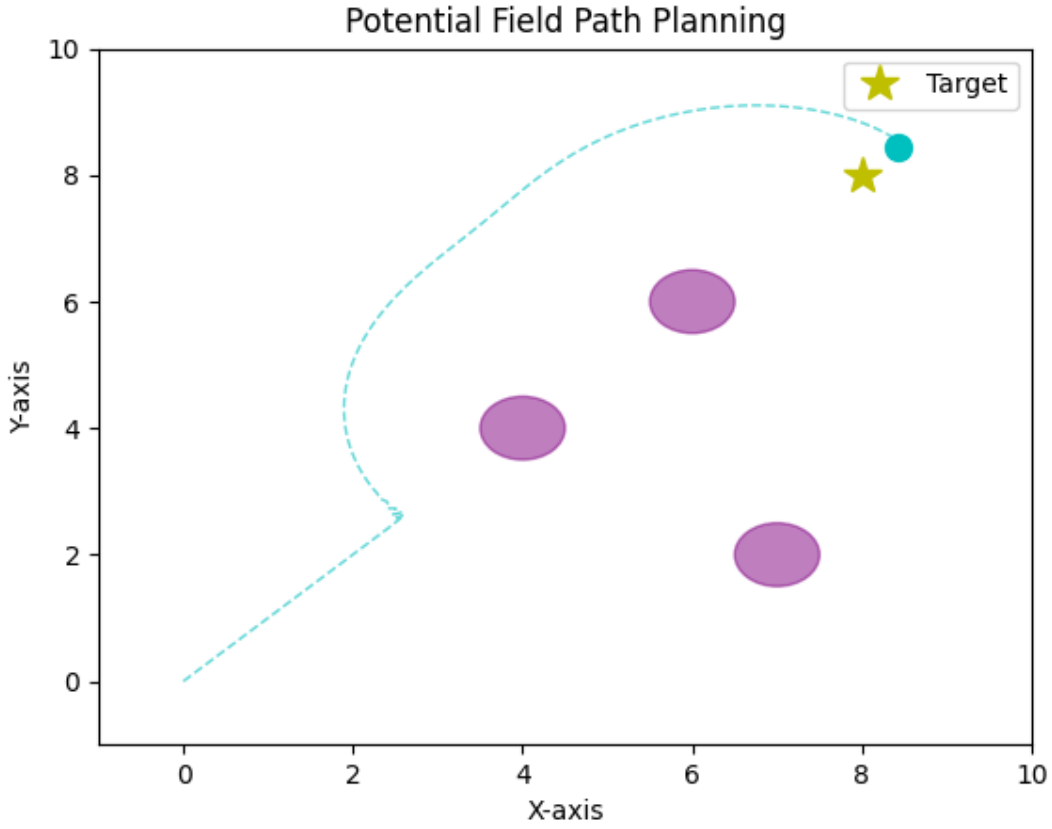
Disadvantages:

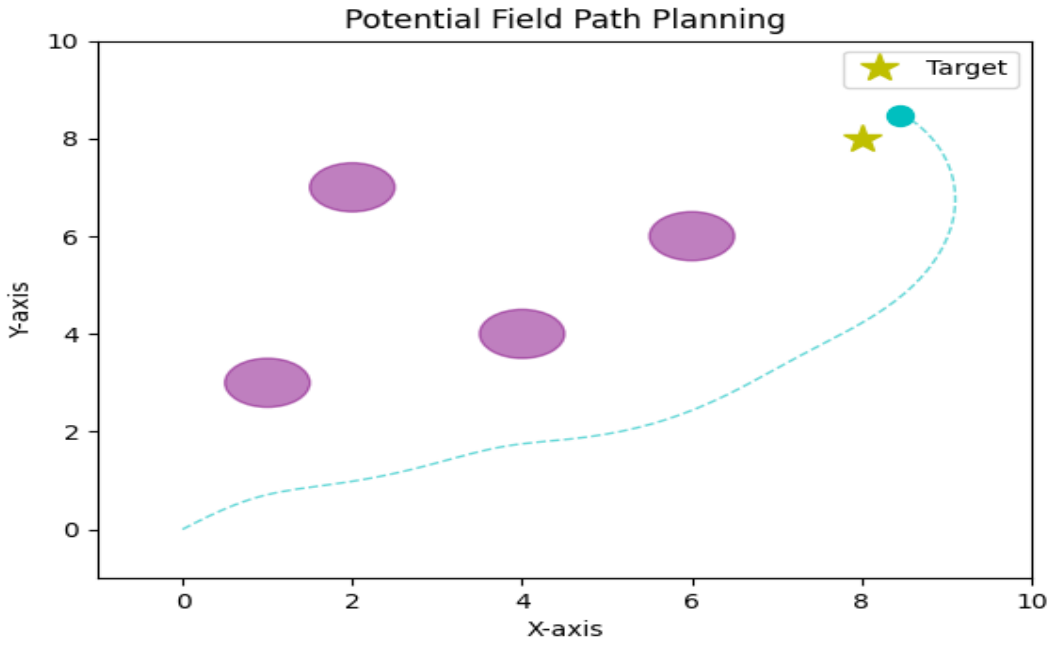
- **Local Minima Problem: The robot can get stuck in local minima (areas where the robot is attracted to a point that is not the goal, such as being trapped between two obstacles).**

- For example, if the robot is equidistant between two obstacles and they both repel the robot, the attractive force may be too weak to pull the robot out of the trap.

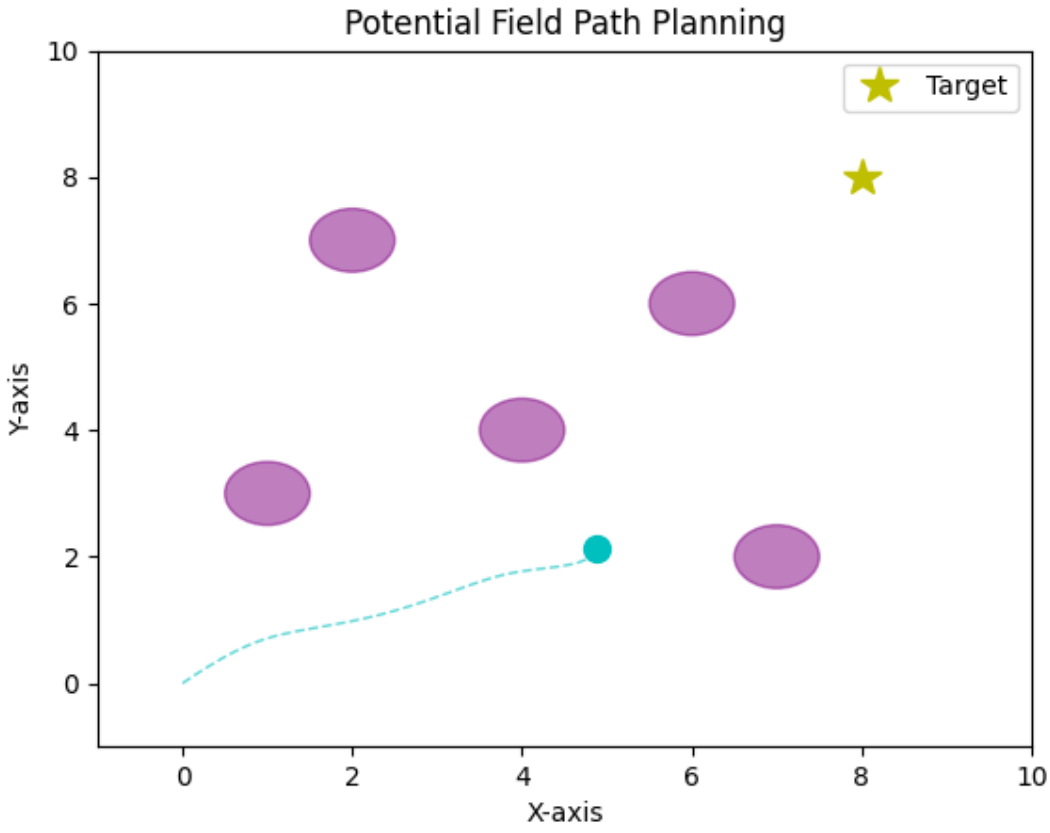
- **Poor Global Path Planning: The algorithm doesn't explicitly plan a path but reacts to local forces. This can sometimes lead to suboptimal paths, especially in complex environments.**
- **Sensitivity to Parameters: The choice of the attraction and repulsion gains, as well as the repulsion radius, can significantly affect performance and behaviour.**

Stimulations:

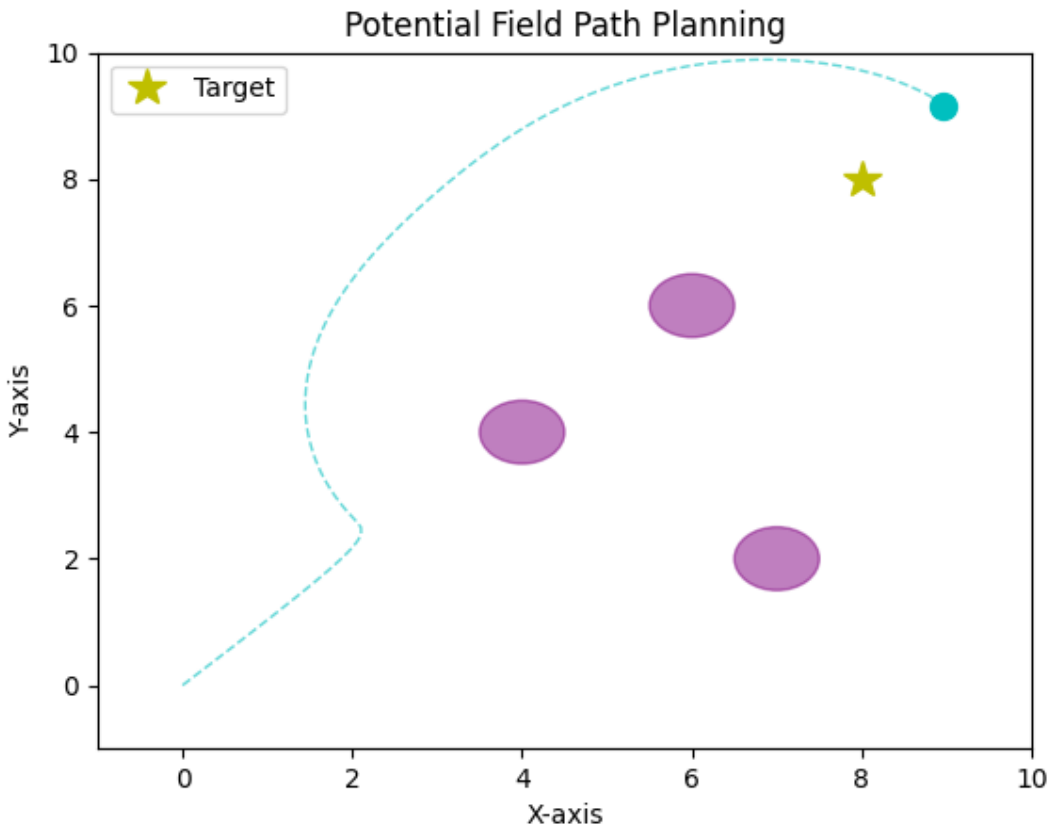




Here robot cannot reach goal if there are many obstacles because there will be repulsion from each obstacle so the robot won't have any path to move to the goal



If the Repulsion Radius is more than more repulsion will be there so therefore the chances of the robot to get to the goal will be very less if there is any obstacle near the goal



4. Dijkstra's Algorithm

Overview:

Dijkstra's algorithm is a well-known graph search algorithm used for finding the shortest path between nodes in a graph. It is particularly useful in grid-based pathfinding problems, such as robot navigation in environments with obstacles. Below is a breakdown of how it works in the context of a grid world

Key Concepts:

1. Grid Representation:

- **The environment is represented as a grid with obstacles, where the start and goal positions are predefined.**
- **Each grid cell is treated as a node in a graph, and the edges between nodes represent possible moves (up, down, left, right).**
- **Obstacles are treated as impassable nodes, and the algorithm ensures the path avoids these.**

2. Algorithm Description:

- **Initialization: The algorithm starts with the start node, setting its cost to 0, and all other nodes' costs are initially set to infinity. The start node is added to a priority queue with a cost of 0.**
- **Priority Queue: The algorithm uses a min-heap (priority queue) to always expand the node with the lowest accumulated cost. The queue stores nodes with their associated path costs.**
- **Node Expansion: In each iteration, the algorithm:**
  1. **Pops the node with the least cost from the priority queue.**
  2. **Explores its neighbours (adjacent nodes), checking if they are within bounds and not blocked by obstacles.**

3. **For each valid neighbour, it calculates the new cost (current cost + 1) and updates the cost if the new cost is lower than the previously recorded cost.**
4. **Each neighbour is added to the priority queue for further exploration.**

- **Termination: The algorithm stops when the goal node is reached, or if all nodes have been visited.**

3. Path Reconstruction:

- **Once the goal is found, the algorithm traces back from the goal node to the start node using a parent dictionary, which records the node from which each node was reached.**
- **This process reconstructs the shortest path by following the parent links from goal to start, and then reversing the path.**

Steps in the Code:

1. **Grid Initialization: The grid size and obstacles are defined, and the start and goal positions are specified.**
2. **Dijkstra's Algorithm: The Dijkstra function performs the algorithm, returning the path and the set of visited nodes.**
3. **Visualization: The FuncAnimation from Matplotlib is used to animate the algorithm's progress, showing the order in which nodes are visited and the path found by Dijkstra's algorithm.**

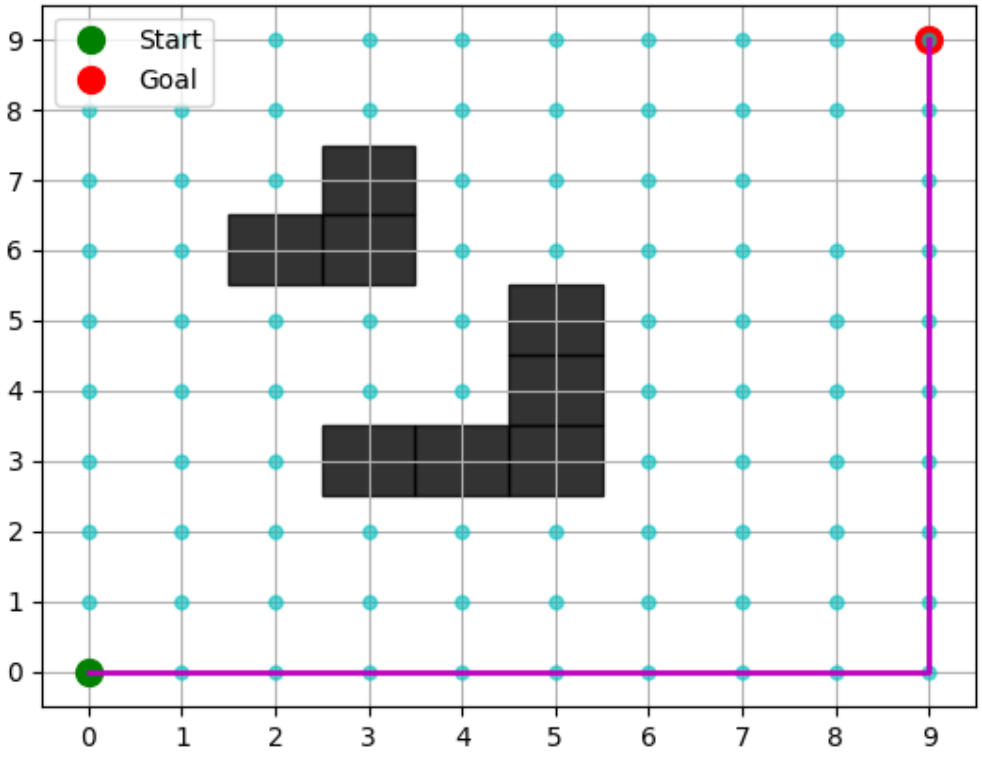
Advantages:

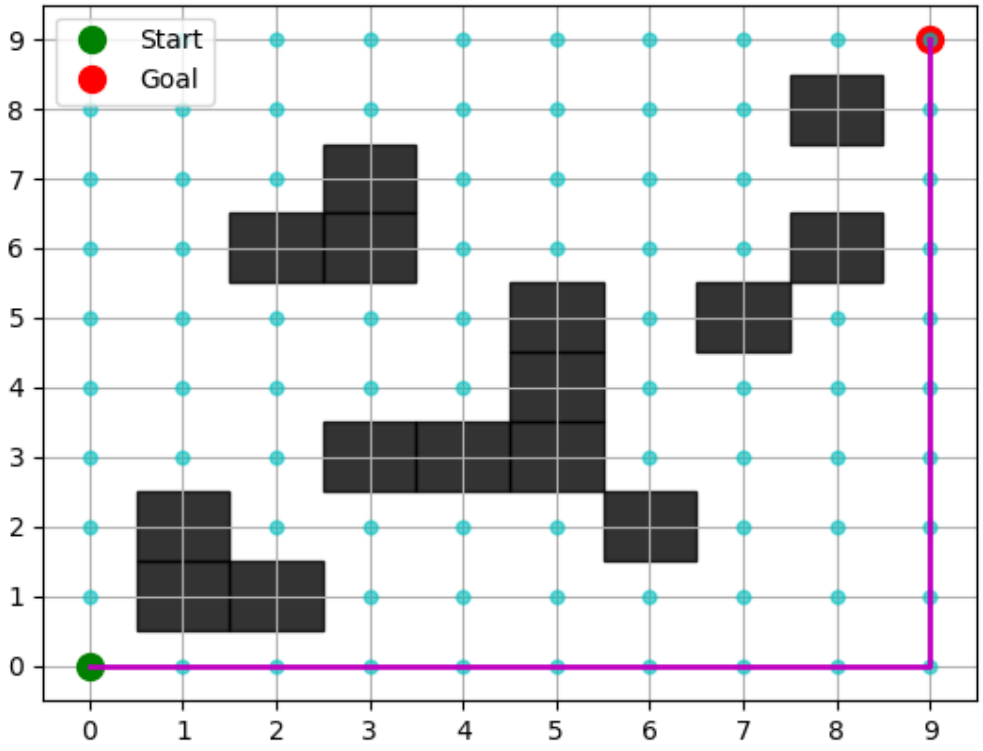
- **Guarantees the shortest path in a grid world, provided that all edge weights are non-negative (which is the case here with uniform costs).**
- **It's optimal and complete, meaning it will always find a path if one exists.**

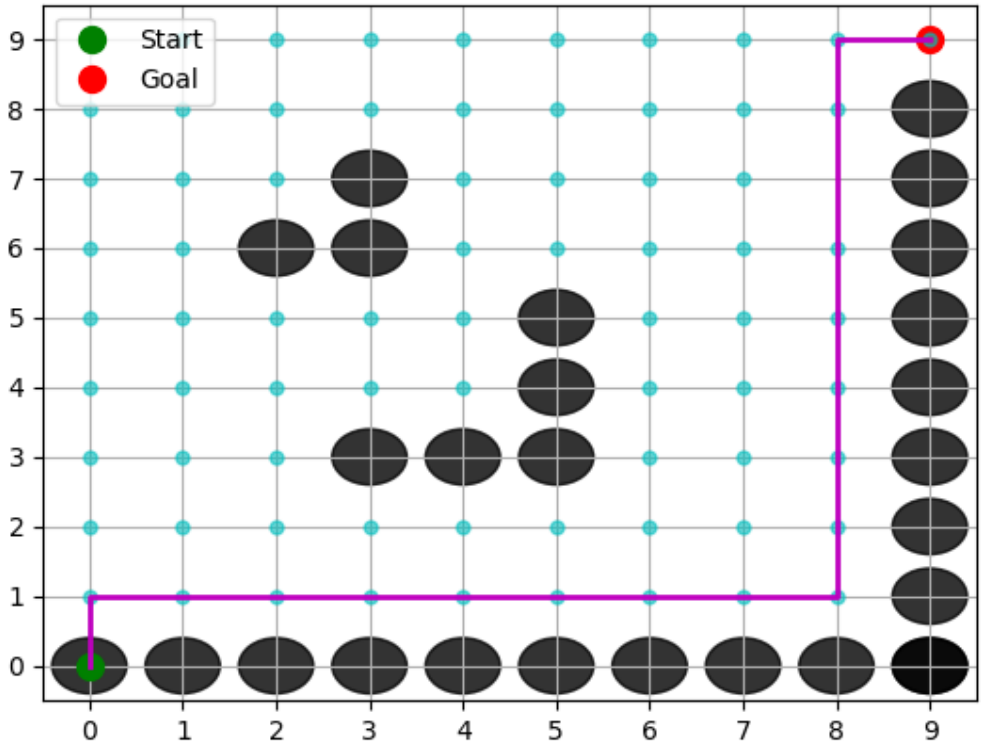
Limitations:

- **The algorithm may be slow for very large grids because its time complexity is O(E log V), where E is the number of edges, and V is the number of nodes.**

Stimulations:




Start
Goal
0
1
2
3
4
5
6
7
8
9


Start
Goal
0
1
2
3
4
5
6
7
8
9

References

1. Path Planning of Mobile Robot Based on A* Algorithm | IEEE Conference Publication | IEEE Xplore Path Planning of Robots Based on an Improved A-star Algorithm | IEEE Conference Publication | IEEE Xplore

2. A new method for robot path planning based artificial potential field | IEEE Conference Publication | IEEE Xplore

3. https://www.researchgate.net/publication/301463753_RT-RRT_a_real-time_path_planning_algorithm_based_on_RRT

4. Path Planning Based on Mixed Algorithm of RRT and Artificial Potential Field Method | IEEE Conference Publication | IEEE Xplore

5. https://www.researchgate.net/publication/348035882_DIJKSTRA_ALGORITHM_USING_UAV_PATH_PLANNING

6. LLM model (Ai Models): Assisted in the Implementation of Algorithms and Research for the project