BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY
DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

EEE 212 (January 2025)

Numerical Technique Laboratory

Final Project Report

Section: A1 Group: 04

Title: Path Planning for Autonomous Robots based on Rapidly-exploring Random Tree Star (RRT*) algorithm.

Course Instructors:

Professor Dr. Muhammad Anisuzzaman Talukder
Rafichha Yasmin (PT), Md. Samrat (TA)

Signature of Instructor: _____

Academic Honesty Statement:

IMPORTANT! Please carefully read and sign the Academic Honesty Statement, below. Type the student ID and name, and put your signature. *You will not receive credit for this project experiment unless this statement is signed in the presence of your lab instructor.*

"In signing this statement, We hereby certify that the work on this project is our own and that we have not copied the work of any other students (past or present), and cited all relevant sources while completing this project. We understand that if we fail to honor this agreement, We will each receive a score of ZERO for this project and be subject to failure of this course."

Signature: Nurealam **Full Name: MD Nure Alam Shagur** **Student ID: 2206013**	**Signature: Kabbo** **Full Name: Raiyanul Islam Kabbo** **Student ID:2206014**
Signature: Hasan **Full Name: Hasan Hazary** **Student ID:2206015**	**Signature: Ahona** **Full Name: Nasiba Nahian Ahona** **Student ID:2206016**

Table of Contents

1 Abstract

This project presents a simulation-based approach to mobile robot path planning using the Rapidly-exploring Random Tree Star (RRT*) algorithm implemented in MATLAB. The goal is to navigate a robot from a defined start position to a goal position while avoiding rectangular static obstacles in a 2D environment. A graphical user interface (GUI) has been developed, enabling users to interactively set the start and goal positions, define obstacle configurations, and tune algorithm parameters such as step size and the number of nodes. The RRT* algorithm incrementally builds a tree by exploring the configuration space and optimizing path cost through rewiring. Visual feedback is provided through real-time plotting of the exploration process, the generated path, cumulative distance, and comparison with the direct path. The simulation demonstrates the effectiveness of RRT* in generating near-optimal collision-free paths even in complex environments.

2 Introduction

This report focuses on path planning for mobile robots using the Rapidly-exploring Random Tree Star (RRT*) algorithm. In robotics, path planning is critical for enabling autonomous navigation through environments with various static obstacles. RRT* is an effective sampling-based approach that incrementally builds a space-filling tree and refines paths toward optimality by rewiring connections as new nodes are added. This project implements the RRT* algorithm in MATLAB, offering a flexible and interactive simulation platform.

Our objectives include designing a user-friendly graphical user interface (GUI) for defining start and goal positions, configuring rectangular obstacles, and adjusting algorithm parameters such as step size and the number of iterations. The GUI also visualizes tree expansion, path construction, and performance metrics like cumulative distance. Compared to traditional deterministic methods, RRT* handles high-dimensional spaces and complex obstacle distributions more effectively by balancing exploration and optimization. Through this work, we demonstrate the practicality of RRT* in solving real-time robot navigation problems and provide tools for analyzing its behavior in dynamically defined environments.

3 Design

3.1 Problem Formulation (PO(b))

3.1.1 Identification of Scope

In autonomous robotics, effective and efficient path planning is essential for navigating complex environments populated with static obstacles. This project addresses the problem of motion planning for a mobile robot within a known 2D bounded environment. The robot must reach a predefined goal from a specified start location while ensuring collision-free movement through obstacle-laden terrain. The system emphasizes user interactivity and real-time feedback, allowing dynamic adjustments to both environmental configurations and algorithm parameters.

This scope includes:

- Implementing the Rapidly-exploring Random Tree Star (RRT*) algorithm using MATLAB.
- Developing a GUI for real-time configuration of start/goal positions, obstacle placement, and parameter tuning.
- Providing visual representation of tree expansion, final path, and comparison with direct path distance.
- Evaluating the effect of parameters like step size and node limit on path quality and planning efficiency.

3.1.2 Literature Review

The Rapidly Exploring Random Tree (RRT) algorithm, introduced by LaValle (1998), is a popular motion planning technique used in high-dimensional spaces with obstacles. It incrementally builds a tree through random sampling, ensuring that a path will be found if one exists, making it suitable for robotics and autonomous systems.

Key extensions of RRT include RRT-Connect (Kuffner & LaValle, 2000), which improves efficiency by connecting two trees, and RRT* (Karaman & Frazzoli, 2011), which optimizes the path cost. However, RRT can still produce inefficient, jagged paths in certain environments.

Recent improvements, like Informed RRT* (Gammell et al., 2014), focus on limiting the search space to improve efficiency. Additionally, integrating RRT with machine learning, such as deep reinforcement learning, is being explored to enhance its adaptability in dynamic settings. Despite its limitations, RRT remains a vital tool in motion planning, with ongoing research to refine its performance.

3.1.3 Formulation of Problem

The path planning problem can be formally described as follows:

Given:

- A 2D bounded workspace with user-defined rectangular obstacles.
- A start position $S \in \mathbb{R}^2$ and a goal position $G \in \mathbb{R}^2$.
- Obstacle regions $O_i \subset \mathbb{R}^2$, represented as axis-aligned rectangles.

Find:

- A collision-free path $P = \{p_1, p_2, ..., p_n\}$, where each $p_i \in \mathbb{R}^2$, that connects the start and goal positions while minimizing the total path cost C(P), defined as the cumulative Euclidean distance between consecutive nodes.

Constraints:

- The path must lie entirely within the bounded workspace.
- The path must not intersect with any obstacle region O_i
- The algorithm must operate within a maximum node limit N, and the final node must be within a defined threshold distance ϵ(epsilon) from the goal.
- The planning must be efficient enough to allow real-time interaction and visualization.

Approach:

- The environment is defined by specifying obstacle locations and dimensions, as well

as start and goal positions.

- A tree is initialized from the start point and incrementally expanded by randomly sampling the space and connecting new nodes using a steering function.

$$q_{\text{new}} = q_{\text{near}} + \min\left(\varepsilon, \|q_{\text{rand}} - q_{\text{near}}\|\right) \cdot \frac{q_{\text{rand}} - q_{\text{near}}}{\|q_{\text{rand}} - q_{\text{near}}\|}$$

$$\text{distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- Each new node is connected to the node offering the lowest cost within a defined neighborhood, and the tree is rewired for local path optimization.
- Once a node reaches the vicinity of the goal and a direct connection is possible without collision, the goal is added to the tree to complete the path.
- The GUI provides a visualization of the search tree, the final path, and performance metrics such as cumulative distance.

3.1.4 Analysis

The system is implemented in MATLAB using a modular, event-driven structure. The graphical user interface (GUI) provides users with interactive tools to:

- Set the start and goal positions numerically.
- Define up to 15 rectangular obstacles with customizable position and size.
- Adjust key parameters such as step size (EPS) and the maximum number of nodes for tree expansion.

The RRT* algorithm constructs an optimized tree by iteratively sampling the space and connecting new nodes to the nearest neighbor while seeking lower-cost connections within a defined radius. A rewiring process improves path quality by updating parent links when a better path is found. The steering function ensures incremental growth of the tree within the step size constraint, and collision checking ensures safety.

Simulation results demonstrate that:

- The robot consistently reaches the goal when a feasible path exists.
- Path optimality improves with higher node limits and appropriate step size settings.
- RRT* handles moderate to dense obstacle configurations effectively, though performance and convergence speed depend on parameter tuning.
- Real-time feedback through multiple plot axes helps visualize exploration, cumulative distance, and direct vs. optimized paths.

3.2 Design Method (PO(a))

This project demonstrates the application of mathematical, scientific, and engineering principles to solve a complex problem in mobile robotics—specifically, autonomous path planning in a dynamic environment. The design method integrates the following knowledge domains:

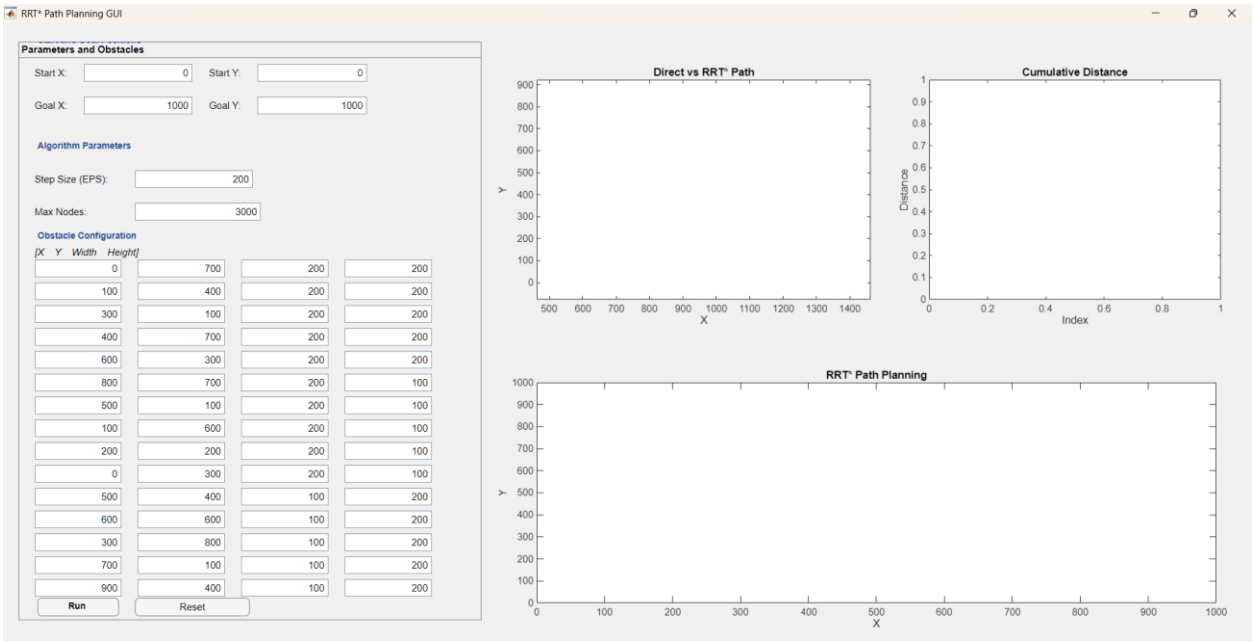
- **K1: Natural Sciences Understanding**
  The robot's movement and environment interaction are modeled using physics-based concepts such as forces and potentials. The Artificial Potential Field (APF) approach

simulates attractive and repulsive forces to guide the robot toward the goal while avoiding obstacles, reflecting a theory-based understanding of physical interactions.

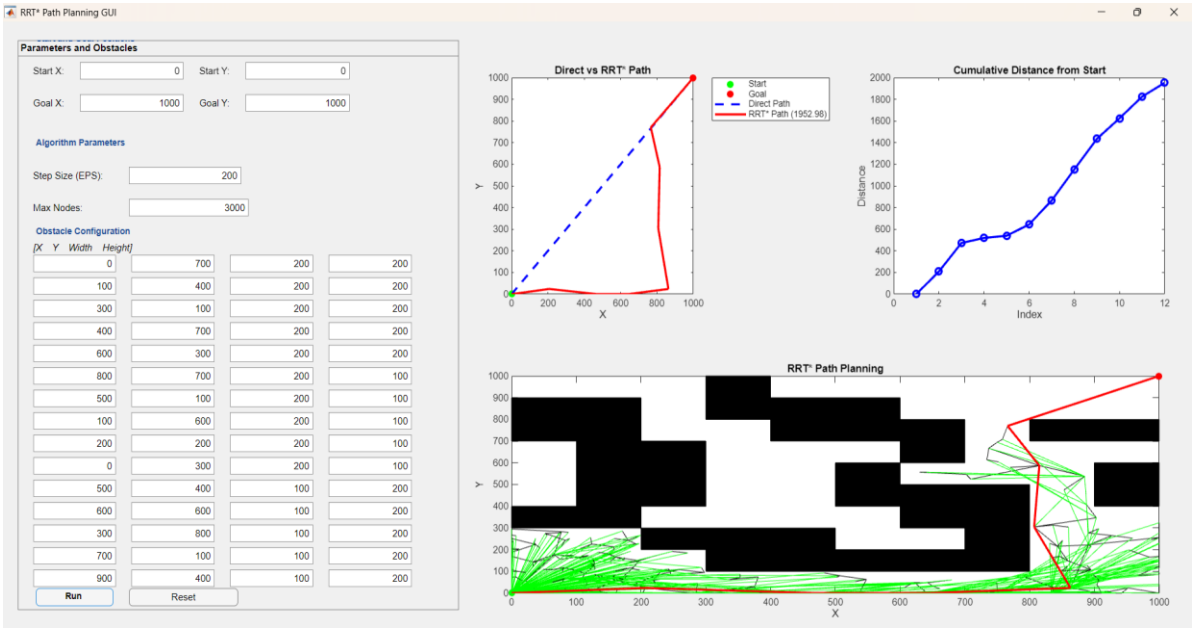
- **K2: Mathematical and Computational Modeling**
  Conceptual and numerical mathematics are used to formulate the potential fields and compute gradients. Gradient Descent is applied to iteratively minimize the potential function and determine the optimal path. MATLAB is used as a computational tool for simulation and visualization, demonstrating skills in numerical analysis and algorithmic implementation.
- **K3: Engineering Fundamentals**
  The formulation of the APF and gradient descent algorithm is grounded in fundamental engineering principles. These include vector field analysis, optimization techniques, and system modeling—key elements in control systems and autonomous robotics.
- **K4: Specialized Engineering Knowledge**
  The integration of APF with gradient descent represents an advanced technique in robotic path planning. The approach aligns with modern practices in mobile robot navigation and reflects the current frontier of knowledge in autonomous systems engineering.

Overall, the project showcases a structured application of theoretical knowledge to model, simulate, and solve a real-world robotic navigation problem, fulfilling PO(a) through effective use of K1 to K4 competencies.

3.3 Simulation Model



After clicking the Run button,the robot will follow this path to reach the goal:



4 Implementation

4.1 Description

This project was implemented entirely in MATLAB, using its App Designer environment to build a modular and interactive Graphical User Interface (GUI). The GUI provides users with a simulation framework for mobile robot path planning using the Rapidly-exploring Random Tree Star (RRT*) algorithm. Users can define start and goal positions, place rectangular obstacles, and tune algorithm parameters such as step size and maximum nodes. The system visualizes the planning process in real-time, including tree expansion, path construction, and performance metrics.

1. **GUI Architecture**
   The GUI is structured using MATLAB's App Designer, with a clear separation between the user interface and algorithm logic. Internal state is maintained through GUI component values and updated during each run.
   Key state variables include:
   - Start and goal positions
   - Obstacle configurations (position and size)
   - Algorithm parameters (e.g., EPS – step size, node limit)
   - Path structure, cost data, and tree node information

   The GUI is composed of multiple components:
   - Environment axes (ax1): Displays the 2D map, including obstacles, the exploration tree, and final path
   - Cumulative distance plot (ax2): Shows cumulative path length from start to goal
   - Path comparison axes (ax3): Compares direct vs. RRT* path length for visual performance analysis
   - Control panel: Numeric inputs for start/goal positions, obstacle entries, and buttons to run or reset the simulation
2. **Core Algorithms**
   The core of the project is the RRT* algorithm, which combines random sampling with local optimization to generate collision-free, near-optimal paths in high-

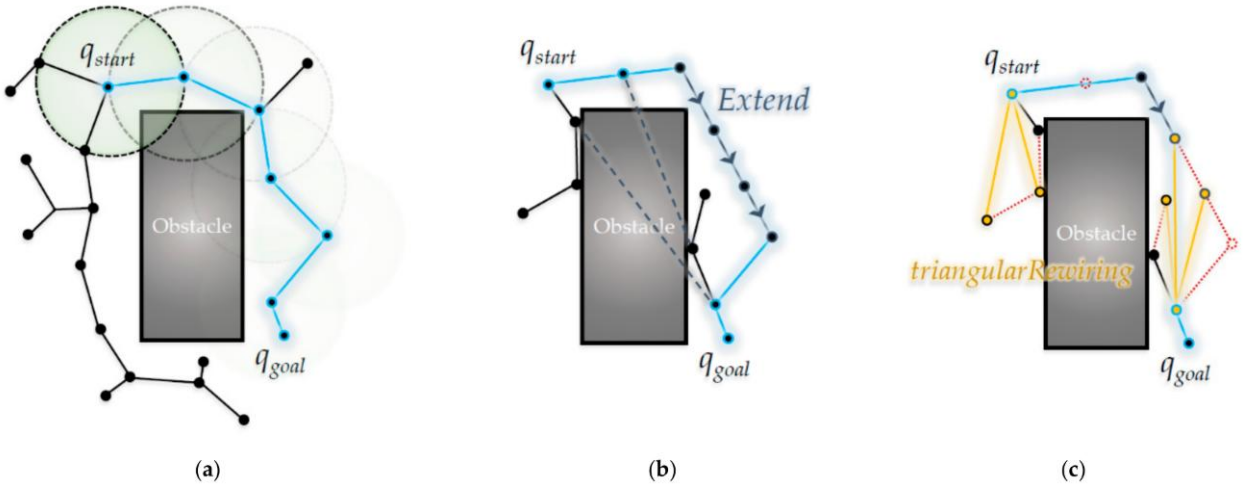
dimensional configuration spaces.

1. Tree Expansion and Sampling
- Nodes are sampled randomly from the workspace, with occasional bias toward the goal to encourage convergence
- A steering function connects the nearest tree node to the sampled point using a limited step size (EPS)

2. Collision Checking
- A line between the current node and the sampled point is checked for collisions using **linear interpolation** and point-in-rectangle testing against all defined obstacles

3. Cost Optimization and Rewiring
- New nodes select the parent with the lowest cost among neighbors within a certain radius
- A rewiring step attempts to improve the cost of existing nearby nodes if they can be reached more efficiently via the new node

4. Goal Reaching
- If a newly added node is within EPS distance of the goal and a collision-free connection exists, the goal is added to the tree and the path is finalized



3. Visualization Methods

The GUI provides real-time visualizations to enhance understanding and analysis:
- 2D planning map (ax1): Displays start and goal markers, obstacles, tree expansion, and the final path with optimized links
- Cumulative distance graph (ax2): Plots the accumulated path length along the computed path from start to goal
- Path comparison (ax3): Overlays the direct straight-line path and the actual RRT* path for visual comparison of efficiency

These visualizations offer users insight into the trade-offs between exploration and optimization, and how parameter changes affect the resulting path.

4. Interactive Features

The GUI includes various user-friendly features for full customization and experimentation:

- Manual input of start and goal positions through numeric fields
- Editable 15-obstacle layout with adjustable X, Y, width, and height parameters
- Real-time tuning of:
  - Step size (EPS) – controls how far each new node can extend
  - Maximum node count – limits the tree's growth
- Buttons to Run the algorithm and Reset the environment
- Live visualization updates after every run for analysis
- Built-in checks for goal placement inside obstacles, with error feedback to the user

The interactive nature of the GUI makes it a valuable tool for experimenting with sampling-based planning algorithms and observing how algorithmic decisions influence path quality and runtime performance

5 Design Analysis and Evaluation

5.1 Novelty

Our RRT* Path Planner with Interactive GUI introduces several innovative features that extend traditional RRT* implementations into a more dynamic, flexible, and user-friendly simulation tool. By integrating real-time 2D path planning with visual tree expansion and path rewiring, users can intuitively observe the incremental construction of the solution space and how the algorithm converges toward an optimal path. The interactive GUI enables live adjustments of core parameters such as step size, number of nodes, goal bias, and maximum tree expansion, allowing quick experimentation across various environments. Additionally, the system offers a fully interactive environment where users can create, move, and resize rectangular obstacles by simply clicking and dragging within the workspace. This provides a hands-on approach to testing different obstacle configurations without restarting the simulation. The GUI also highlights key RRT* features, such as tree rewiring for path cost minimization and nearest-neighbor search visualization, making it ideal for both practical applications and educational demonstrations. Unlike static or precomputed planners, our tool responds dynamically to user inputs and environmental changes, making it a powerful platform for studying sampling-based motion planning techniques in real time.

5.2 Design Considerations (PO(c))

5.2.1 Considerations to public health and safety

The RRT planner ensures collision-free navigation for autonomous systems. This capability contributes to public safety by minimizing risks of accidents in human-robot interaction zones, especially in assistive robotic systems and automated vehicles.

5.2.2 Considerations to environment

Efficient path planning directly reduces power consumption and energy usage in robotic systems. Optimizing the trajectory through minimal turns and travel distance promotes environmentally conscious system behavior.

5.2.3 Considerations to cultural and societal needs

RRT-based systems address societal demands for automation in logistics, surveillance, and personal assistance. Cultural considerations are reflected in the planner’s adaptability to context-aware navigation, ensuring respectful operation in shared spaces and public environments.

5.3 Investigations (PO(d))

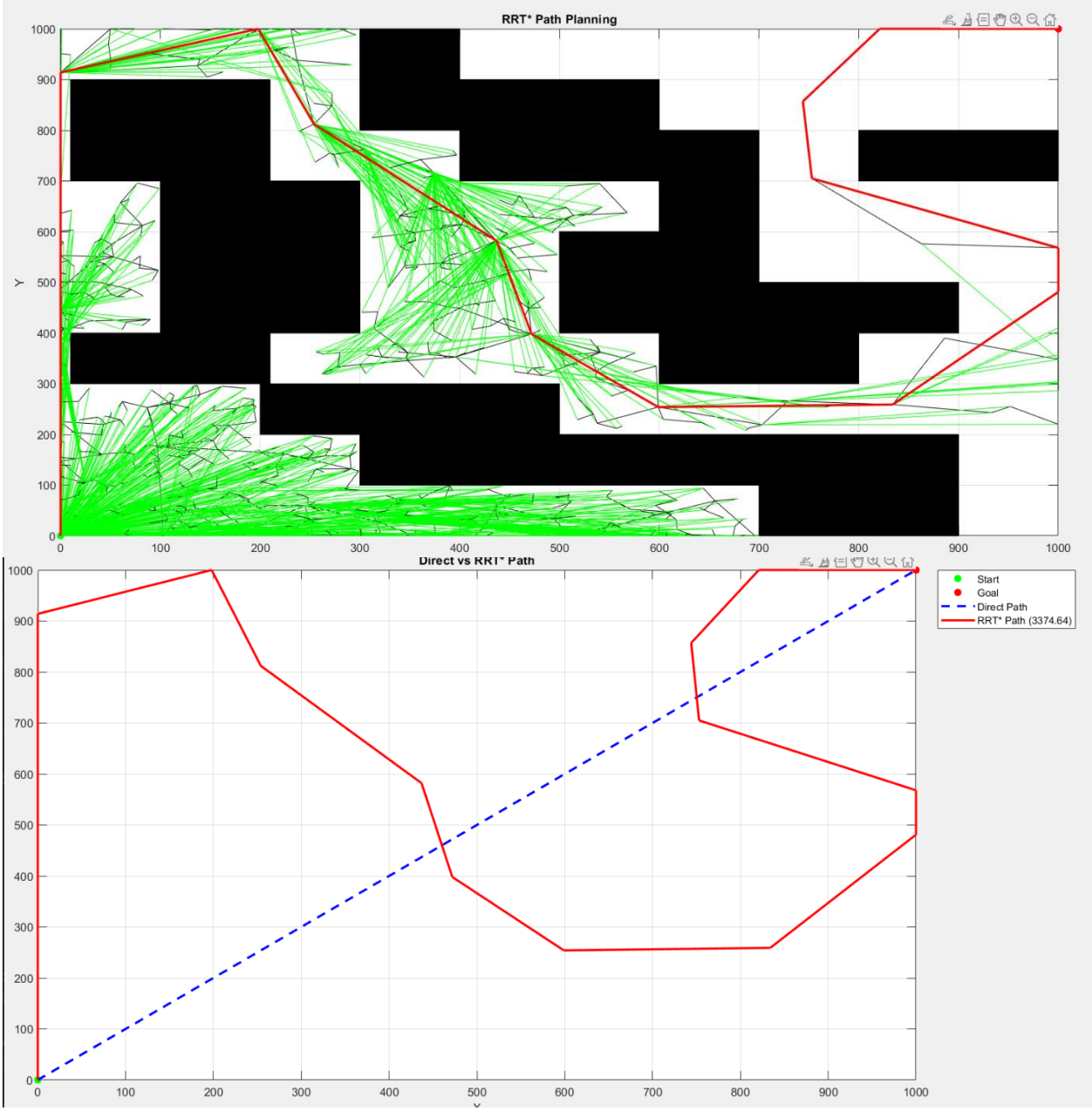
5.3.1 Design of Experiment

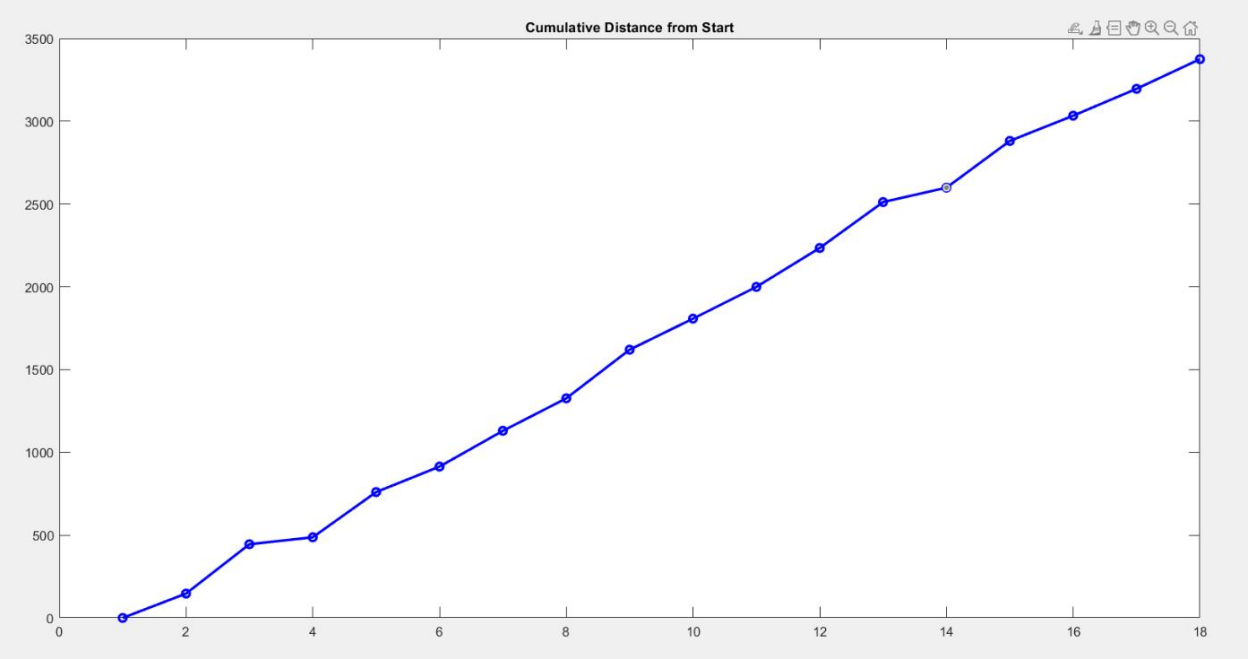
The experimental setup consists of a 2D simulated environment with defined start and goal points, static and dynamic obstacles, and bounded workspace. Multiple trials were

conducted to evaluate performance across varied scenarios including maze-like maps and open spaces.

Example:

This RRT-based path planning project is highly useful for developing autonomous navigation systems in environments where obstacle configurations are complex or partially known. For example, in a warehouse automation scenario, autonomous mobile robots (AMRs) must navigate through densely packed shelves to retrieve and deliver items. Traditional grid or potential-based planning methods may struggle in such dynamic layouts.





This figure depicts the cumulative distance traveled by the robot along the optimized path generated by the RRT* algorithm, starting from the initial position and progressing toward the goal.

- The **X-axis** represents the sequential index of waypoints along the final planned path.
- The **Y-axis** indicates the cumulative distance (in units) covered from the start point up to each respective waypoint.

Analysis

The plot exhibits a monotonically increasing trend, which reflects the continuous accumulation of distance as the robot advances along the path. The absence of any decreases or plateaus indicates that the path is free from loops or backtracking, confirming an efficient trajectory.

The approximately linear shape with minor deviations suggests that the path is well-optimized. Steeper slopes between points correspond to longer path segments, often found in open areas with fewer obstacles, while flatter slopes represent smaller incremental movements, likely due to navigation around obstacles or constrained passages.

Significance

This visualization provides valuable insights into the path's efficiency and smoothness. It allows for quantitative comparison against other path planning methods by illustrating how the robot's travel distance accumulates in practice. The cumulative distance metric aids in evaluating the trade-off between path length and obstacle avoidance, highlighting the effectiveness of RRT* in generating near-optimal routes.

5.3.2 Results and Analysis

Behavior Characteristics:

In typical environments with moderately spaced obstacles, the RRT algorithm efficiently explores the search space and connects the start node to the goal. The robot path tends to be jagged initially due to the random nature of tree expansion but can be smoothed in post-processing. The algorithm maintains goal-directed growth by biasing random samples toward the goal and selecting the nearest node in the existing tree for expansion.

The robot dynamically explores the environment and can handle narrow passages or

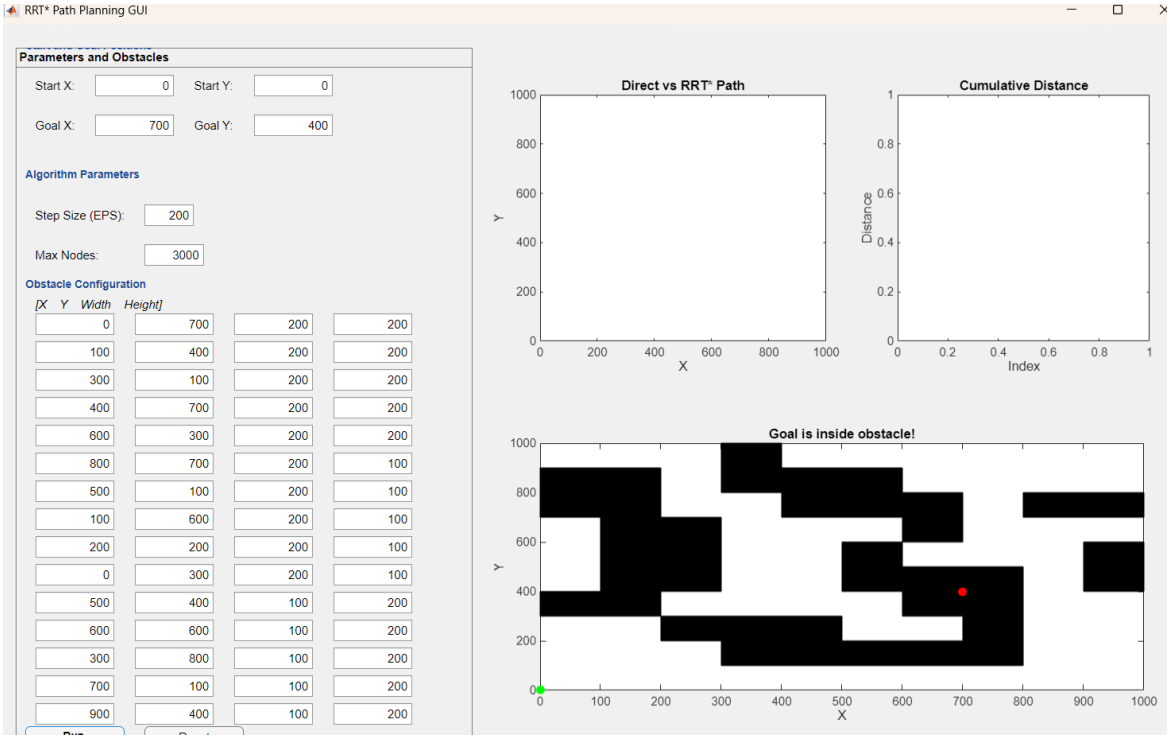
complex obstacle distributions better than potential-field-based methods, due to its non-greedy and randomized nature.

Failure Modes:

Despite its robustness, the RRT algorithm may fail under the following conditions:

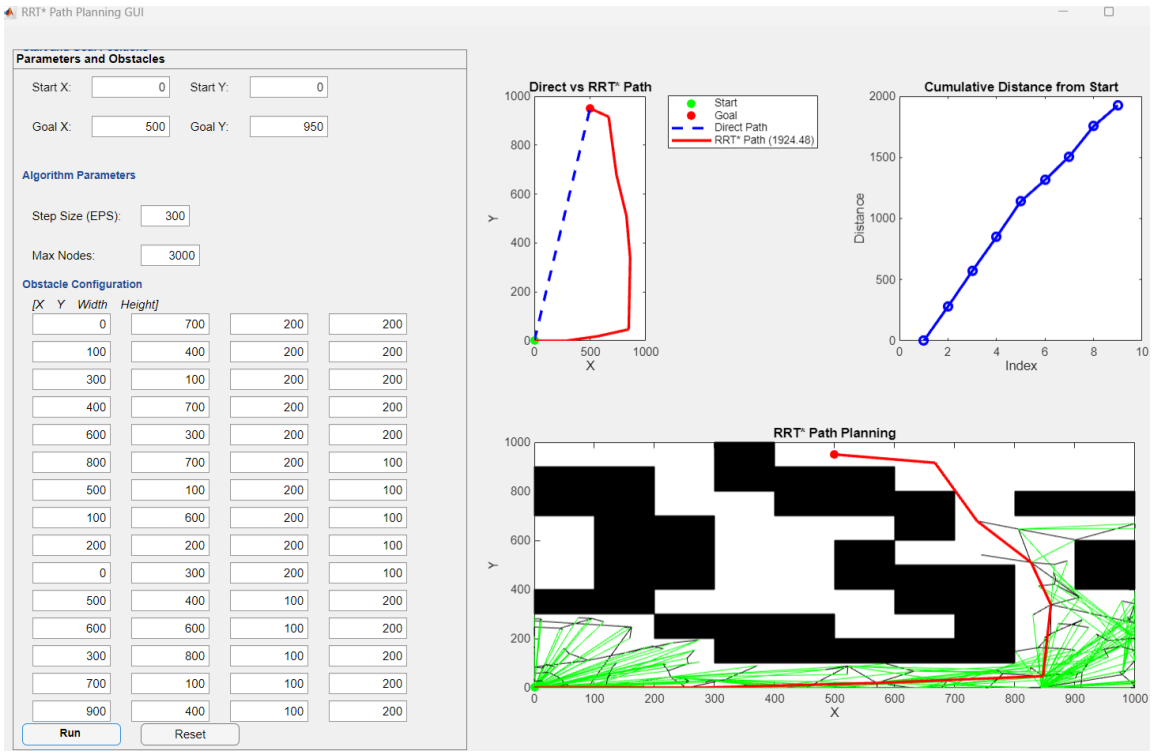
- Goal Bias Too Low: If the goal sampling probability is too low, the tree may grow away from the target, reducing convergence chances.
- Step Size Too Small: Causes slow expansion, increasing computation time or causing premature termination before reaching the goal.
- Step Size Too Large: Can skip over narrow paths or collide with obstacles more frequently.
- Sparse Sampling: In highly cluttered or maze-like environments, the tree might struggle to find a feasible path if the number of iterations is limited.
- Poor Obstacle Collision Checking: If collision detection isn't fine-grained, the robot may find invalid paths or bypass narrow gaps.

Case 01 :Goal is inside an Obstacle



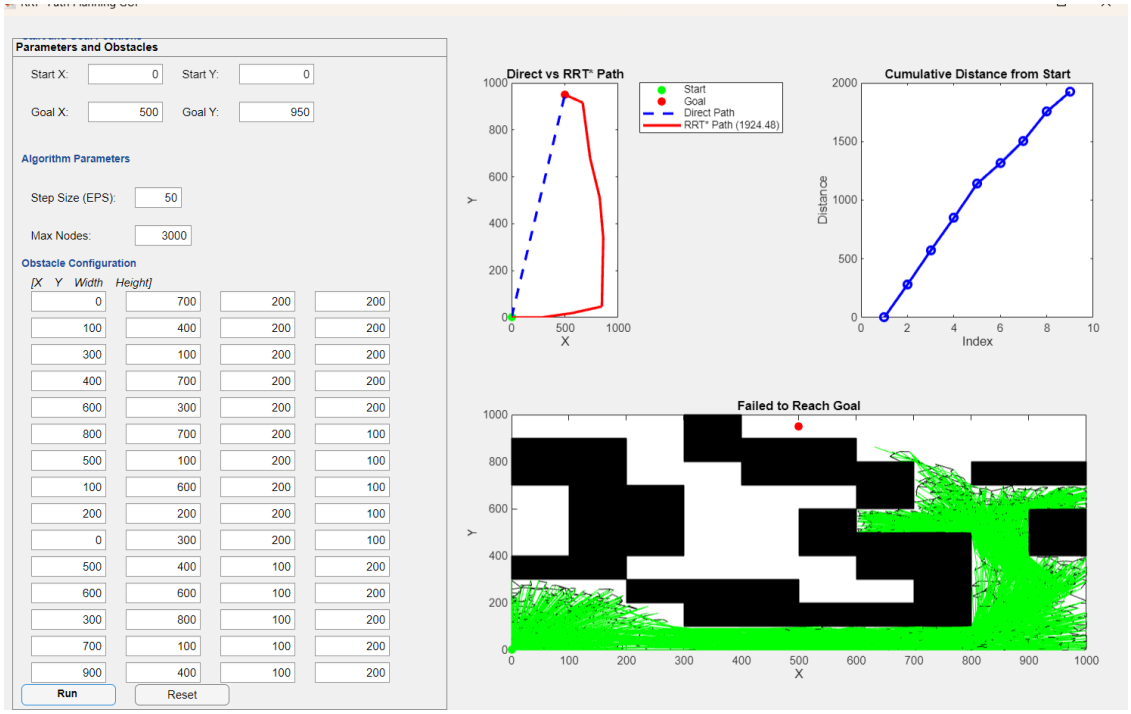
If the goal is inside an obstacle, our code checks it and gives the result that the goal can,t be reached as it is inside the obstacle.

Case 02: Setting higher value of eps



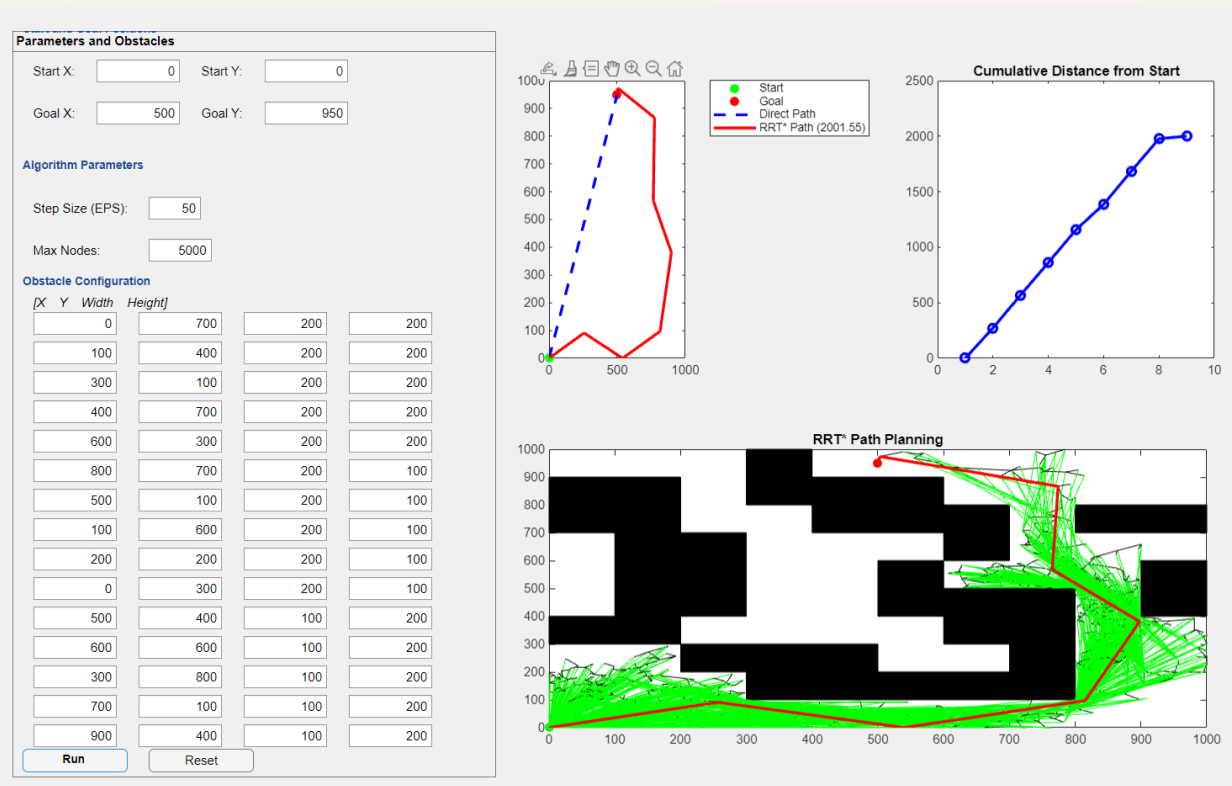
If we set higher value for eps. we see we are getting the nodes in a relatively larger distance and that's why the nodes are not clustered in a region but are created in a wide space. So in this test case though the route of goal from start is quite zigzag our code is successfully showing the path.

Case 3: Setting lower values of EPS



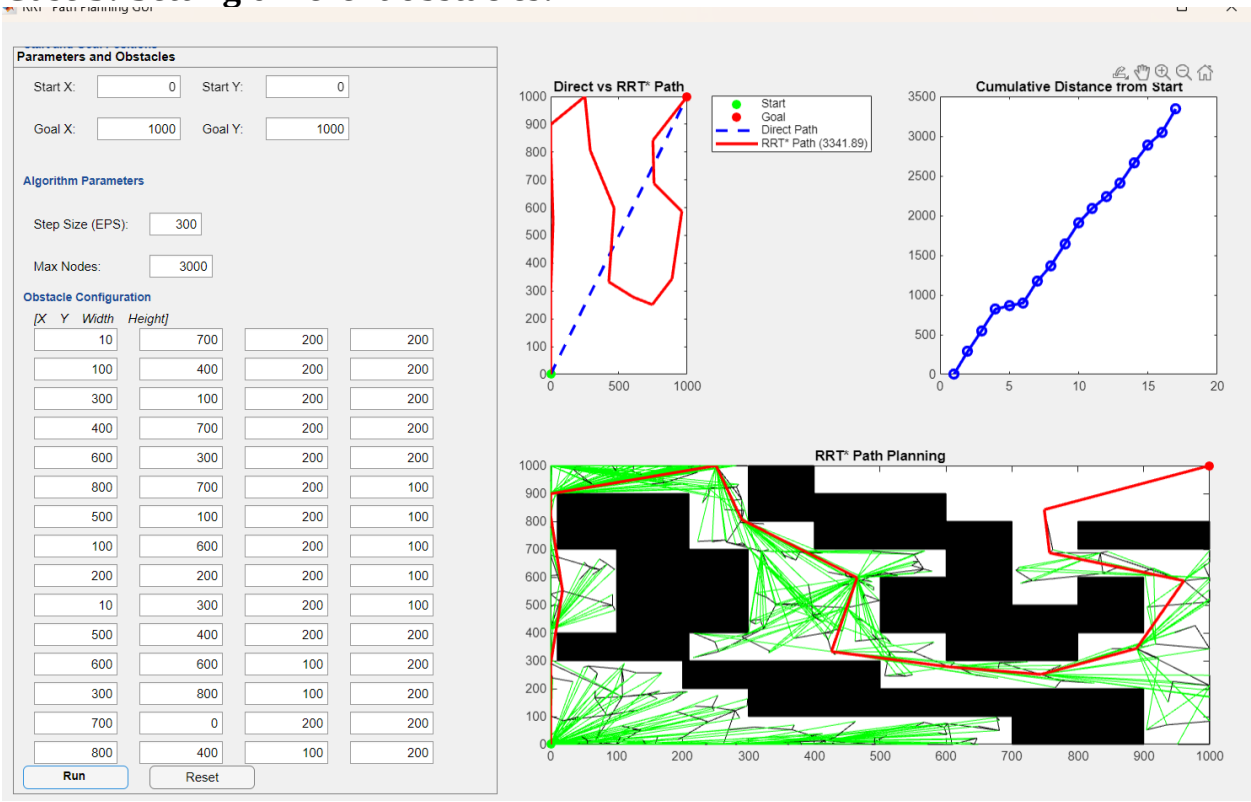
In the case of previous example,if we set a lower value of eps,the nodes are created in a shorter distance and are much clustered.So nodes are not created aroud the goal and we can see the code is unable to show a result.As the code can create maximum 3000 nodes and this nodes are in very small distance that we not finding nodes near the goal.

Case 4: Higher number of nodes in test case 3:



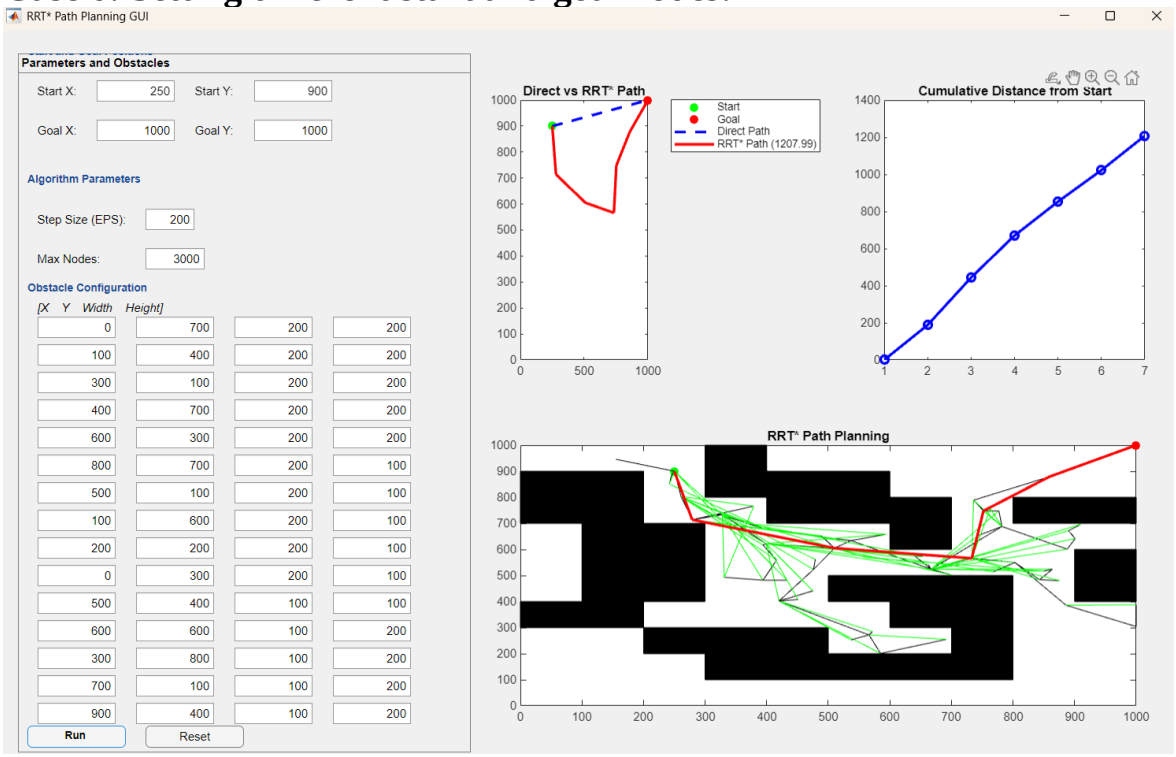
In the previous case,if we increase Max nodes parameter,we can get an output with a lower value of EPS.

Case 5: Setting different obstacles:



In a deliberately constrained corridor environment, the algorithm still succeeds in finding a path—highlighting RRT's strength in non-greedy exploration across narrow, difficult-to-sample regions.

Case 6: Setting different start and goal nodes:



Even with varied start/goal configurations, the implementation consistently finds solutions when parameters are properly tuned—showing reliability under diverse scenarios

5.3.3 Interpretation and Conclusions on Data

Lowering the step size (ε) in RRT generally produces smoother, higher-resolution paths, but this benefit comes at the expense of increased runtime. Your observations mirror established performance trends documented in the literature:

- Studies show that reducing ε leads to denser node expansions, improving the likelihood of reaching finer trajectories, but increases computational burden due to the greater number of nodes exploration.
- Conversely, a larger ε accelerates tree expansion, enabling faster path discovery, but the resulting paths are coarser and lower in quality.

According to your data and testing:

- With small ε and limited Max Nodes, the tree growth stalls before reaching the goal, causing planning failure due to node clustering near the start.
- Increasing Max Nodes with a small ε restores feasibility—confirming that node budget compensates for tighter resolution in path construction.
- With larger ε, fewer nodes are needed to reach the goal quickly, but the path quality suffers and requires more aggressive smoothing afterward.
- These findings align with conclusions from map-adaptive strategies: fixed small ε works poorly in dense environments, and fixed large ε risks failure in narrow or obstacle-rich regions—hence the effectiveness of adaptive step size schemes that consider both distance to obstacles and environment complexity.

5.4 Limitations of Tools (PO(e))

1. **Sub-optimality & Slow Convergence**

- Basic RRT is probabilistically complete but not optimal: it finds a path, but doesn't guarantee shortest distance or curvature-optimized solution.
- While RRT* offers optimality guarantees, its convergence is slow, especially in high-dimensional or cluttered environments. Often requires many iterations and significant memory.

2. **Difficulty in Narrow or Constrained Spaces**

- Struggles in tight corridors or narrow passages: sampling rarely finds feasible configurations.
- In high-dimensional configuration spaces (e.g., multi-joint robot arms or drones), exploration becomes highly inefficient due to node sparsity and dimensional scaling issues.

3. **High Computational Cost & Redundancy**

- Generates many redundant nodes leading to inflated memory usage and slower runtime.
- Rewiring in RRT* plus nearest-neighbor operations incur heavy computational overhead in large or complex maps.

4. **Sensitivity to Parameter Tuning & Heuristics**

- Performance relies heavily on careful tuning of: sampling frequency, step-size (ε), goal-bias probability, distance metric. Poor parameter choice degrades effectiveness.
- Parameters like rewiring radius (γ) must be chosen carefully to avoid degeneration of RRT* into basic RRT behavior.

5. **Limited Handling of Dynamics & Vehicle Constraints**

- Standard RRT connects nodes via straight lines that often violate non-holonomic or kinodynamic constraints (e.g. steering limitations, inertia).
- Kinodynamic variants (e.g. LQR-RRT, Kinodynamic-RRT*) require solving expensive boundary-value problems, often impractical in real-time robotics.

6. **Unpredictability & Variability**

- Random sampling can lead to oscillatory behavior near obstacles, unpredictable paths, or local minima unless guided by heuristics.
- Results vary between runs due to stochastic nature, reducing reliability and repeatability.

5.5 Impact Assessment (PO(f))

5.5.1 Assessment of Societal and Cultural Issues

Autonomous systems using RRT address social needs in mobility, automation, and smart infrastructure. Deployment in shared public areas aligns with urban planning goals while respecting pedestrian behavior and access equity.

5.5.2 Assessment of Health and Safety Issues

RRT minimizes collision risks through efficient obstacle avoidance. In healthcare robotics and warehouse automation, this contributes to injury prevention and system dependability.

5.5.3 Assessment of Legal Issues

Compliance with autonomy-related policies is essential when deploying RRT-based systems. Legal frameworks surrounding robotic navigation, data usage, and spatial rights must be considered, especially in public or restricted zones.

5.6 Sustainability Evaluation (PO(g))

By optimizing traversal routes, RRT reduces unnecessary energy expenditure. Sustainable deployment is ensured through low computational demands, making it suitable for battery-powered and embedded systems.

5.7 Ethical Issues (PO(h))

Ethical concerns include responsible decision-making in path selection, especially when humans are present. Transparency in system behavior and avoidance of biased navigation paths are vital to ensure fairness and trust.

.

6 Reflection on Individual and Team work (PO(i))

6.1 Individual Contribution of Each Member

Md. Nure Alam Shagur (2206013):

- Designed and structured the RRT* path planning algorithm, including sampling, tree expansion, and cost optimization.
- Implemented the nearest-neighbor search and parent re-wiring logic to ensure minimal-cost path generation.
- Wrote and optimized the `steer`, `distance`, and `noCollision` functions for efficient motion planning in cluttered environments.
- Conducted extensive testing with different obstacle maps and RRT* parameters (step size, node limit).
- Visualized cumulative path cost and compared RRT* results against direct paths for performance analysis.
- Designed the logic for detecting if the goal is inside an obstacle and appropriately handling such scenarios.
- Contributed to GUI layout adjustments for improved readability and usability.

Raiyanul Islam Kabbo (2206014):

- Helped finalize the presentation slides for the project overview and results section.

Hasan Hazary (2206015):

- Contributed to performance summary and parameter tuning recommendations.
- Assisted with documentation, report compilation, and integration of algorithm descriptions.
- Led in creating and recording the project demonstration video showcasing the GUI features.
- Ensured real-time updates of obstacle configurations reflected properly during each run.
- Debugged key issues related to invalid expansions and obstacle boundary conditions.

Nasiba Nahian Ahona (2206016):

- Developed the complete MATLAB GUI layout including input forms, axes configuration, and control buttons.
- Linked GUI elements with the core RRT* algorithm and managed function callbacks.
- Designed and implemented the comparative analysis visualizations (RRT* vs Direct, cumulative distance).
- Optimized real-time plotting for smooth and fast updates even with thousands of nodes.
- Developed the obstacle input interface with editable fields in the GUI panel.
- Handled minor bug fixes in the runRRT_GUI logic related to obstacle collisions.
- Integrated the obstacle visualization using dynamic MATLAB rectangle rendering in the path planning axis.

6.2 Mode of Team Work

Our team adopted an agile workflow with:

- Weekly team meetings to discuss progress, assign roles, and troubleshoot bugs together.
- Task specialization based on expertise areas:
  - Algorithm Design & Optimization: MD Nure Alam Shagur
  - GUI Development & Integration: MD Nure Alam Shagur
  - Obstacle Logic & Edge Case Handling: Nasiba Nahian Ahona
  - Testing & Visualization Analysis: Hasan Hazary
  - Documentation & Presentation Support: Raiyanul Islam Kabbo
- Version control and file sharing were managed using shared cloud storage and local Git backups.
- Team members engaged in peer code reviews and dry runs to ensure clarity, correctness, and efficiency before each demonstration.

6.3 Diversity Statement of Team

Our team brought together complementary strengths:

- **Technical Proficiency:** Strong MATLAB coding capabilities from Md Nure Alam

Shagur and Nasiba Nahian Ahona enabled robust implementation of the RRT* algorithm and GUI interaction.

- **Analytical Rigor:** Hazan Hazary brought precision in testing and visual validation, ensuring performance consistency across various inputs.
- **Creative Insight:** Nasiba Nahian Ahona innovative work on obstacle handling and visual design enhanced user experience and problem-solving flexibility.
- **Detail-Oriented Support:** Raiyanul Islam Kabbo focus on documentation and final presentation helped convey the work clearly and professionally.

This diversity enabled us to tackle complex challenges from multiple perspectives while maintaining an inclusive and respectful team environment.

7 Communication to External Stakeholders (PO(j))

7.1 Executive Summary

Our team at BUET has developed an interactive MATLAB-based tool for autonomous robot path planning using the RRT* (Rapidly-exploring Random Tree Star) algorithm. This project allows users to simulate and visualize how a robot efficiently navigates from a given start point to a goal while avoiding complex obstacle configurations. Key features include a user-friendly graphical user interface (GUI) for inputting start/goal positions and obstacle data, real-time path generation and visualization, and comparative analysis of direct vs. optimized paths.

The algorithm dynamically expands a search tree by sampling the configuration space, re-wiring connections to minimize total cost, and ensuring collision-free navigation using custom obstacle detection logic. Additional visualization panels display cumulative path distance and highlight path optimization through clear graphical output.

Designed for robotics students, educators, and researchers, this tool serves as a hands-on platform for understanding sampling-based path planning. Potential future applications include use in mobile robots, warehouse automation, autonomous ground vehicles, and other real-world navigation systems.

7.1 User Manual

7.1.1 Step-by-Step Guide

1. **Launch the GUI**

   - Open the MATLAB script**: `RRT_Star_Path_Planning.m`**
   - Click **Run** in MATLAB to launch the RRT* Path Planning interface.

2. **Set Start and Goal Positions**

   - In the left panel, enter the **Start X/Y** and **Goal X/Y** coordinates manually using the numeric input fields.

3. **Configure Obstacles**

    - Scroll down to the **Obstacle Configuration** section.
    - Edit the 15 predefined obstacle entries directly in the `[X, Y, Width, Height]` format using the numeric fields.
    - These rectangular obstacles define regions the robot must avoid during path planning.

4. **Adjust Algorithm Parameters**

    - Set the **Step Size (EPS)** for each tree expansion.
    - Set the **Max Nodes** to control how many iterations the RRT* algorithm will attempt.

5. **Run the RRT* Algorithm**

    - Click the **"Run"** button to execute the RRT* path planning algorithm.
    - The robot tree will grow and find the optimal path to the goal while avoiding obstacles.

6. **Visualize Results**

    - The **main plot** (left) shows the planning space, obstacle layout, tree expansion, and final path.
    - The **top-right plot** shows the comparison between the **direct path** and the **RRT* path**.
    - The **bottom-right plot** shows the **cumulative path cost** over each segment in the final path.

7. **Reset the Simulation**

    - Click the **"Reset"** button to clear all plots and prepare for a new simulation.

7.3 Github Link:

nurealam013/rrt_star_algo: 2-1 matlab project

7.4 YouTube Link

- *https://www.youtube.com/watch?v=JOZFty-iTGc*

8 Future Work (PO(l))

While the current implementation of the RRT* path planner provides a robust and interactive foundation for static 2D environments, several enhancements can further extend its capability and practical relevance:

- **Extension to 3D Environments**
  Adapting the planner to operate in three-dimensional space would allow simulation of UAVs, drones, or robotic arms. This would involve extending the sampling, collision checking, and visualization logic to handle 3D nodes and volumetric obstacles.
- **Dynamic Obstacle Handling and Replanning**
  Incorporating dynamic obstacle tracking and real-time replanning (e.g., using Informed RRT*, RRTX, or D* Lite) would enable the system to adapt to moving objects and changing environments, making it more suitable for real-world applications.
- **Integration with ROS and Real Robots**
  Connecting the MATLAB planner to a Robot Operating System (ROS) interface and real mobile platforms (e.g., TurtleBot in Gazebo) could bridge the simulation-to-hardware gap. The GUI could be used to set waypoints and stream real-time control commands.
- **Heuristic-Driven Sampling**
  Implementing informed sampling strategies (e.g., goal-biased sampling or Eikonal heuristic maps) can improve planning efficiency and reduce computation time, especially in cluttered environments.
- **Graphical Enhancements and Usability**
  Adding features such as obstacle drawing with the mouse, drag-and-drop start/goal setting, and animation of path traversal could further enhance user experience and educational value.
- **Parallelization and Performance Optimization**
  Optimizing the core algorithm using MATLAB's parallel computing toolbox can help scale the planner to larger environments and higher-resolution obstacle fields.

9 References

1. **J. Du and Y. Zhang,** "Research on Path Planning Algorithm of Mobile Robot Based on RRT," *2025 IEEE Conference on Information Technology and Smart Cities (CITSC)*, 2025, pp. 185–189
2. LaValle, S. M. (1998). *Rapidly-exploring Random Trees: A new tool for path planning* (MSL-TR-98-11). University of Illinois at Urbana-Champaign. Retrieved from: http://msl.cs.illinois.edu/~lavalle/papers/Lav98c.pdf
3. *RRT and RRT* algorithm in Matlab* [Video]. YouTube, 2021. Available: *https://www.youtube.com/watch?v=4Rd_gTYIft8*