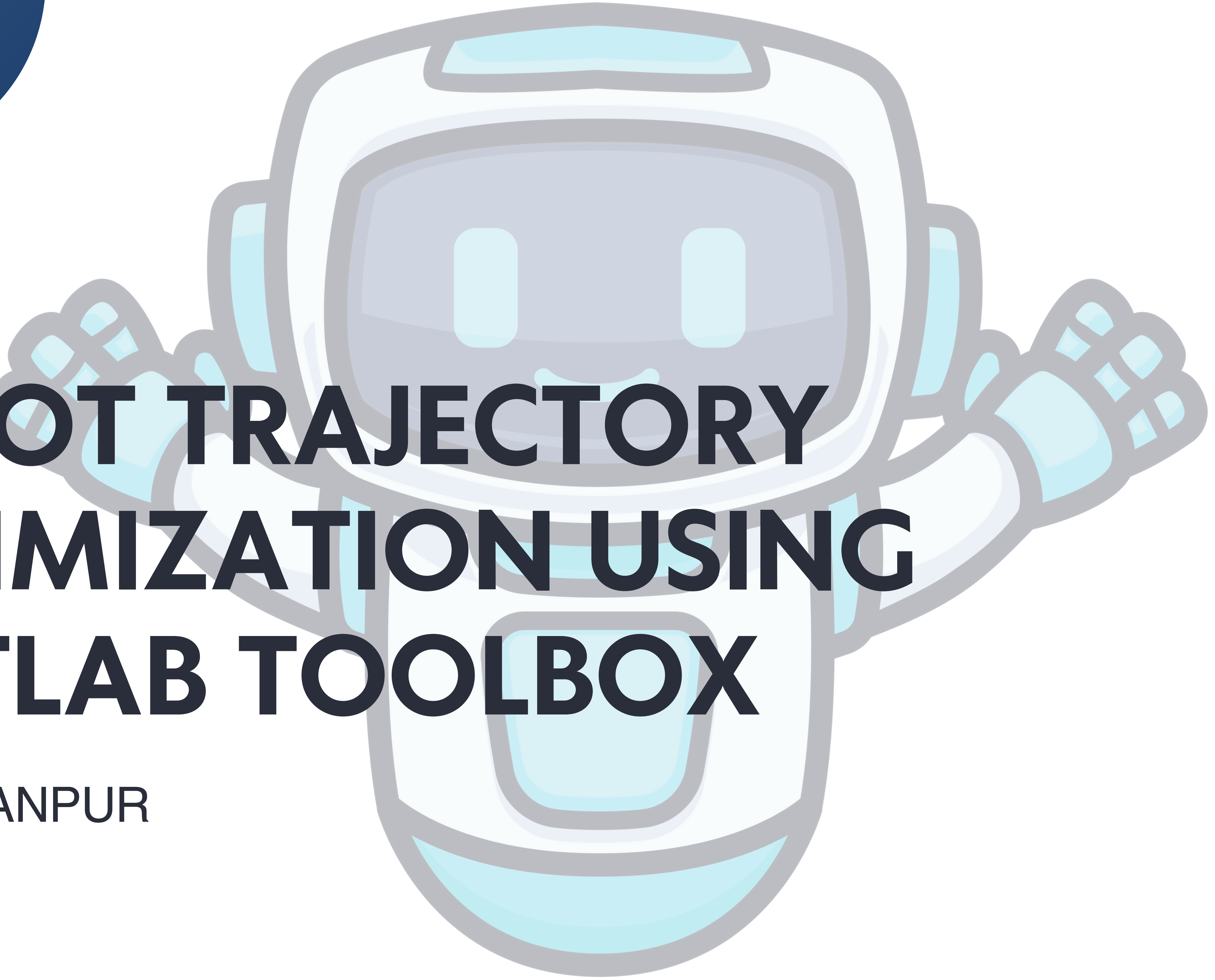


ROBOT TRAJECTORY OPTIMIZATION USING MATLAB TOOLBOX

AME,IIT KANPUR

MENTORS:
SHRUTI
ANTARYA MONDAL

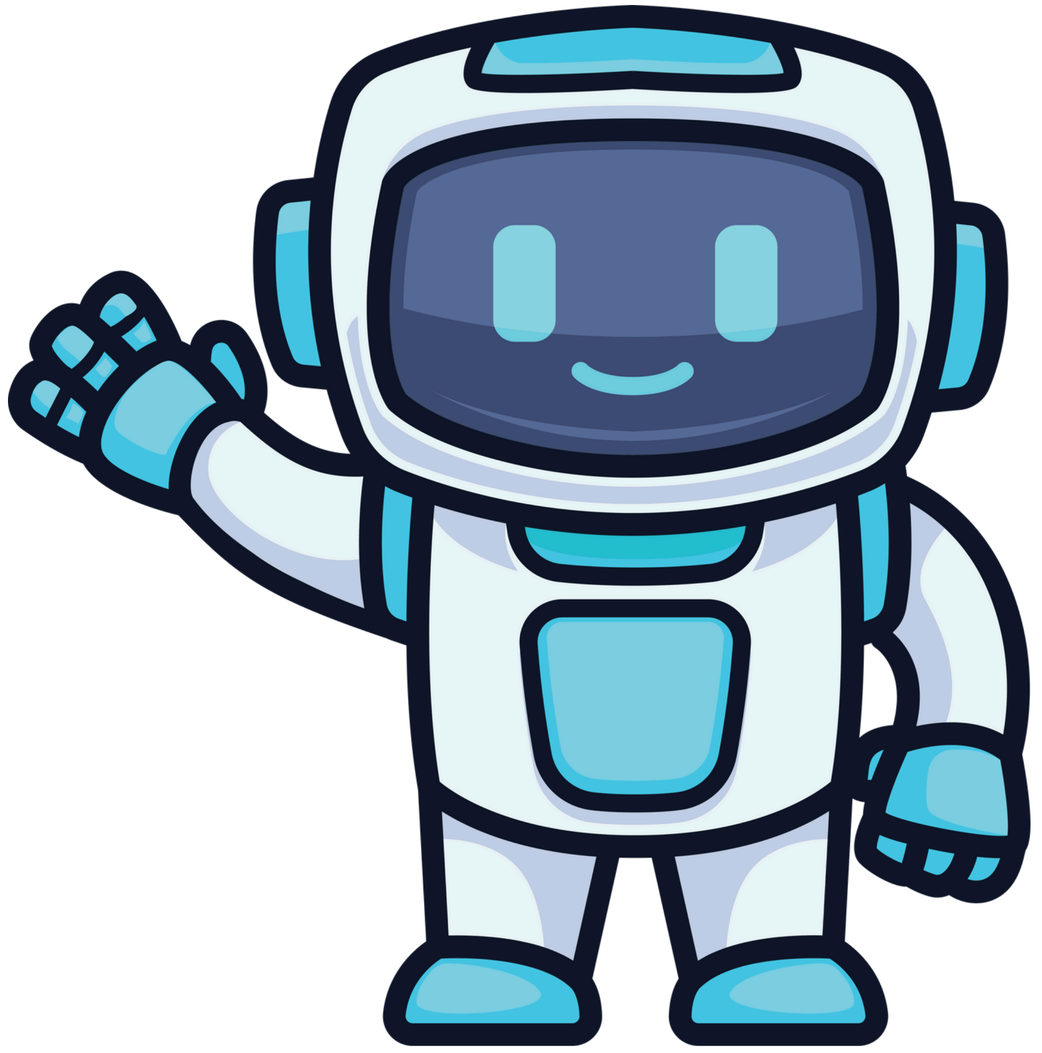
MEMBERS:
DEV SHAKYA
SHIVANSH GUPTA
PRADYUMNA
NIKHIL SONI
AARIN
PRIYANK AGNIHOTRI

WEEK 0

This project looks at trajectory optimization, an important challenge in robotics, which involves finding the most efficient and practical path for a robot to complete tasks while following physical and environmental limits. Using MATLAB's Optimization Toolbox, the project helps users optimize the movement of robotic systems like arms, mobile robots, or drones in a simulated environment.

MATLAB

MATLAB is widely used in robotics for various tasks related to modeling, simulation, control, trajectory planning, and optimization. Its high-level programming environment, combined with specialized toolboxes, provides powerful tools to solve many complex problems in robotics.



MATLAB ONRAMP:

MATLAB Onramp is a free, interactive introductory course that teaches the basics of MATLAB. Designed for beginners, it provides hands-on exercises and immediate feedback, making it an excellent starting point for learning MATLAB quickly and effectively.

BASICS OF MATLAB

Variables & Operations:

- Create variables: x = 5, y = 10
- Basic arithmetic: z = x + y, z = x * y
- Matrix operations: A = [1, 2; 3, 4], B = A * A

If-Else Statements:

```
if a > b
 disp('a is greater')
else
 disp('b is greater')
end
```

```
number = 8;

if mod(number, 2) == 0
   disp('The number is even');
else
   disp('The number is odd');
end
```

For Loop:

```
for i = 1:5
    disp(i)
end
```

a.Use a for loop to calculate and display the squares of numbers from 1 to 10.

```
for i = 1:10
    square = i^2;
    disp(['The square of ', num2str(i), ' is ', num2str(square)]);
end
```

num2str is a MATLAB function that converts a numeric value into a string (character array). This is useful when you want to combine numbers with text, as MATLAB requires both elements to be in string format for concatenation.

WEEK 1 & 2

MATLAB & SIMULINK

Optimization Basics:

- **Covers problem formulation, solver selection, and constraints application using MATLAB's Optimization Toolbox.**
- **Focus on hands-on learning with engineering and robotics examples.**

Trajectory Optimization:

-
- **Designing efficient, feasible robot paths while meeting constraints.**
- **Applications in robotics and autonomous vehicles for energy-efficient and collision-free motion.**
- **Includes simulation of robot dynamics and optimization for stability and energy efficiency.**

WEEK 1 & 2

Adaptive Trajectory Planning:

- **Emphasizes creating paths that adapt to environmental changes in real time.**
- **Utilizes tools like sensors, machine learning, and real-time feedback for dynamic obstacle handling.**

Gait Optimization for Legged Robots:

- **Focuses on creating stable and efficient walking patterns.**

WEEK 3

- We used fmincon for nonlinear optimization in MATLAB.
- Solves constrained optimization problems by minimizing an objective function.
- Handles constraints including equality, inequality, and variable bounds.
- Part of the Optimization Toolbox for tackling complex real-world applications.
- Commonly applied in engineering, robotics, and finance for efficient, precise solutions.
- Supports flexible algorithms such as interior-point and sequential quadratic programming (SQP).

WEEK 3

- We learnt about path planning algorithms in this week.
- Path planning ensures collision-free, optimized navigation for robots.
- It handles both static and dynamic environments.
- Applications include autonomous vehicles, drones, robotic arms, and humanoid robots.
- Focuses on minimizing path length and energy consumption.
- Adapts to real-time changes, such as obstacles or moving targets.
- Enhances reliability, safety, and efficiency in robotics tasks.

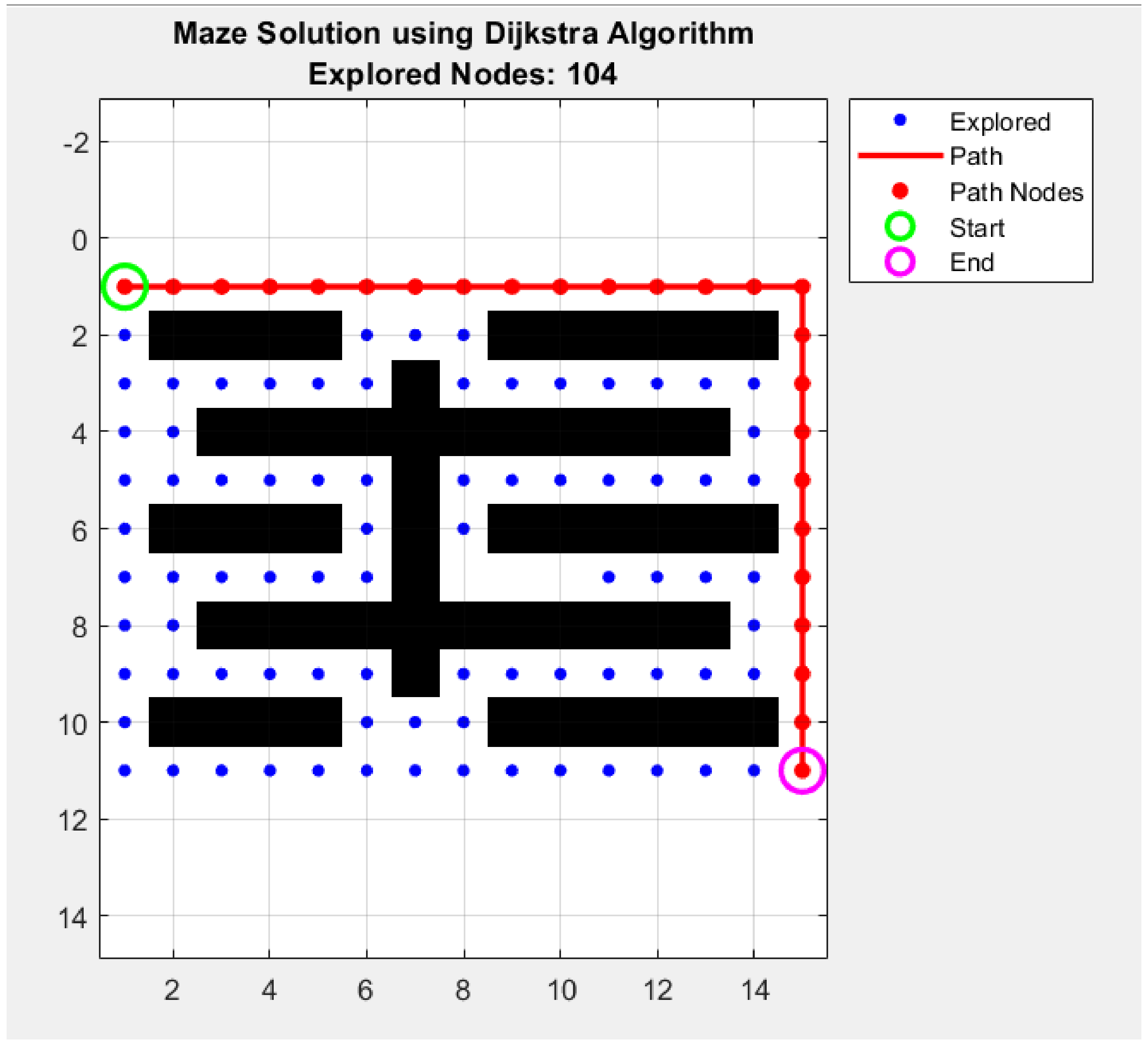
WEEK 3

Dijkstra’s Algorithm

1. **Initialization:**
   - Start from the source node.
   - Set all other nodes' distances to infinity.
2. **Selection:**
   - Choose the unvisited node with the smallest distance.
3. **Distance Update:**
   - Update the distance to its neighbors based on the current node.
4. **Iteration:**
   - Repeat until all nodes have been visited or the shortest paths are found.

WEEK 3

Dijkstra’s Algorithm



WEEK 3

A* Algorithm

1. **Heuristic Function (h(n)):**
- Estimates the cost to reach the goal from node n.
- Common heuristics: Manhattan distance, Euclidean distance, etc.
2. **Cost Function:**
- g(n) + h(n)
- g(n) = cost from start node to current node
- h(n) = heuristic estimate from current node to goal

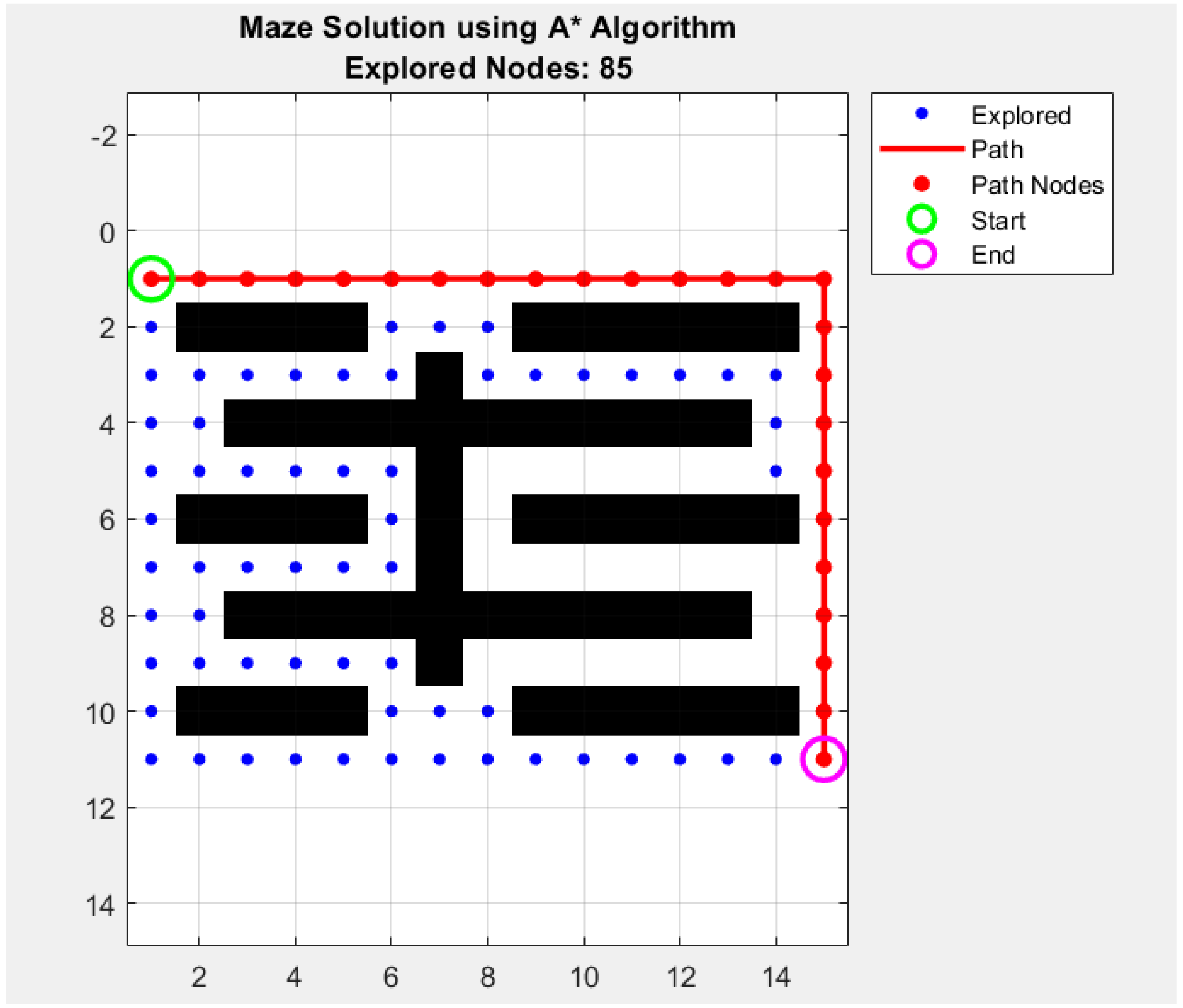
WEEK 3

A* Algorithm

1. **Initialization:**
   - Start from the initial node with a cost of 0.
   - Set all other nodes’ costs to infinity.
2. **Selection:**
   - Choose the node with the lowest combined cost (g(n) + h(n)).
3. **Expansion:**
   - Expand the selected node, updating neighboring nodes’ costs.
4. **Termination:**
   - Stop when the goal node is reached or all paths are explored.

WEEK 3

A* Algorithm



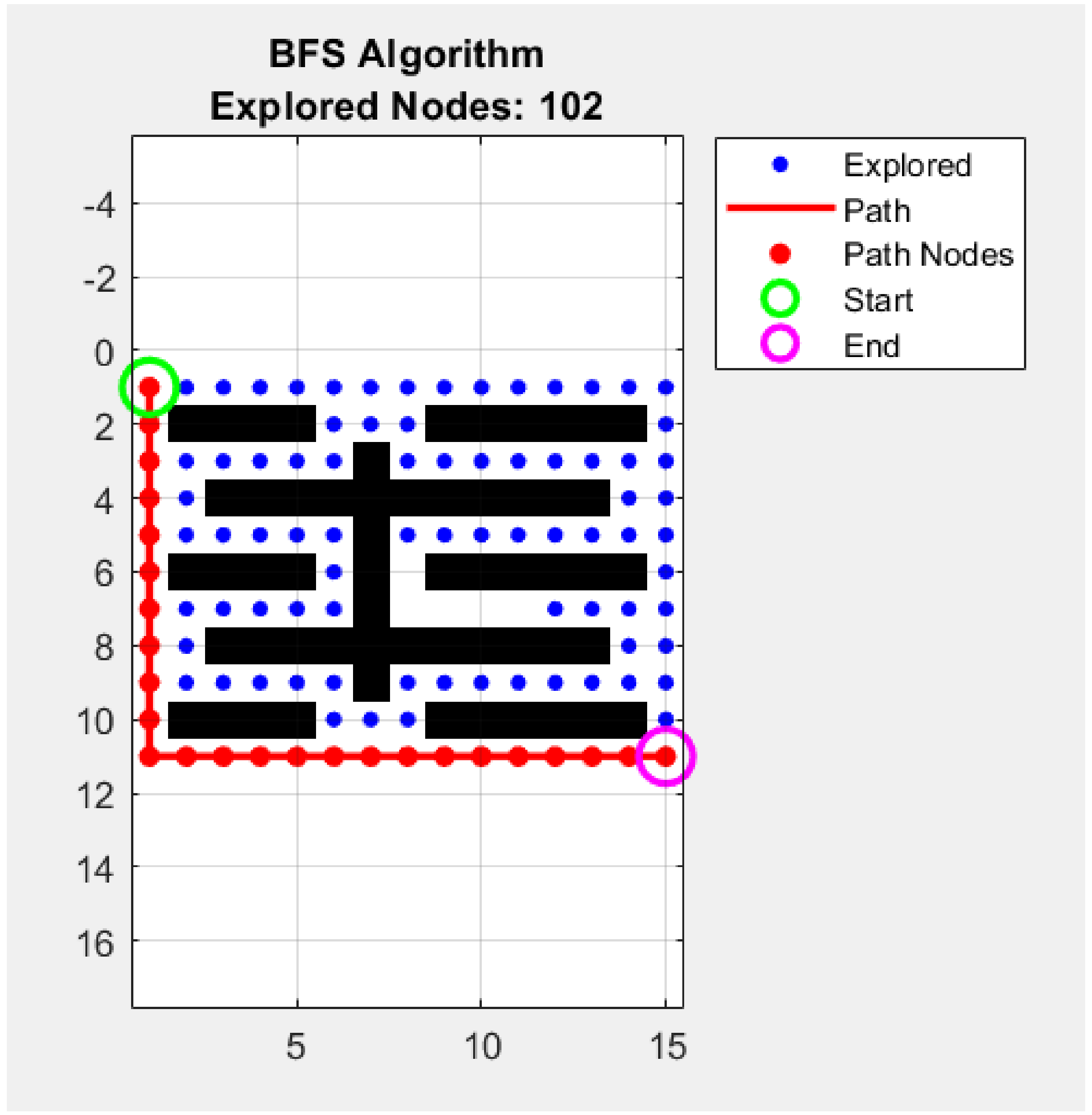
WEEK 3

Breadth First Search

1. **Initialization:**
   - Start from the initial node and mark it as visited.
   - Create a queue to manage exploration order.
2. **Traversal:**
   - Dequeue a node, visit it, and explore its neighbors.
   - If a neighbor hasn't been visited, enqueue it for further exploration.
3. **Iteration:**
   - Continue until all nodes have been visited or the target node is found.

WEEK 3

Breadth First Search



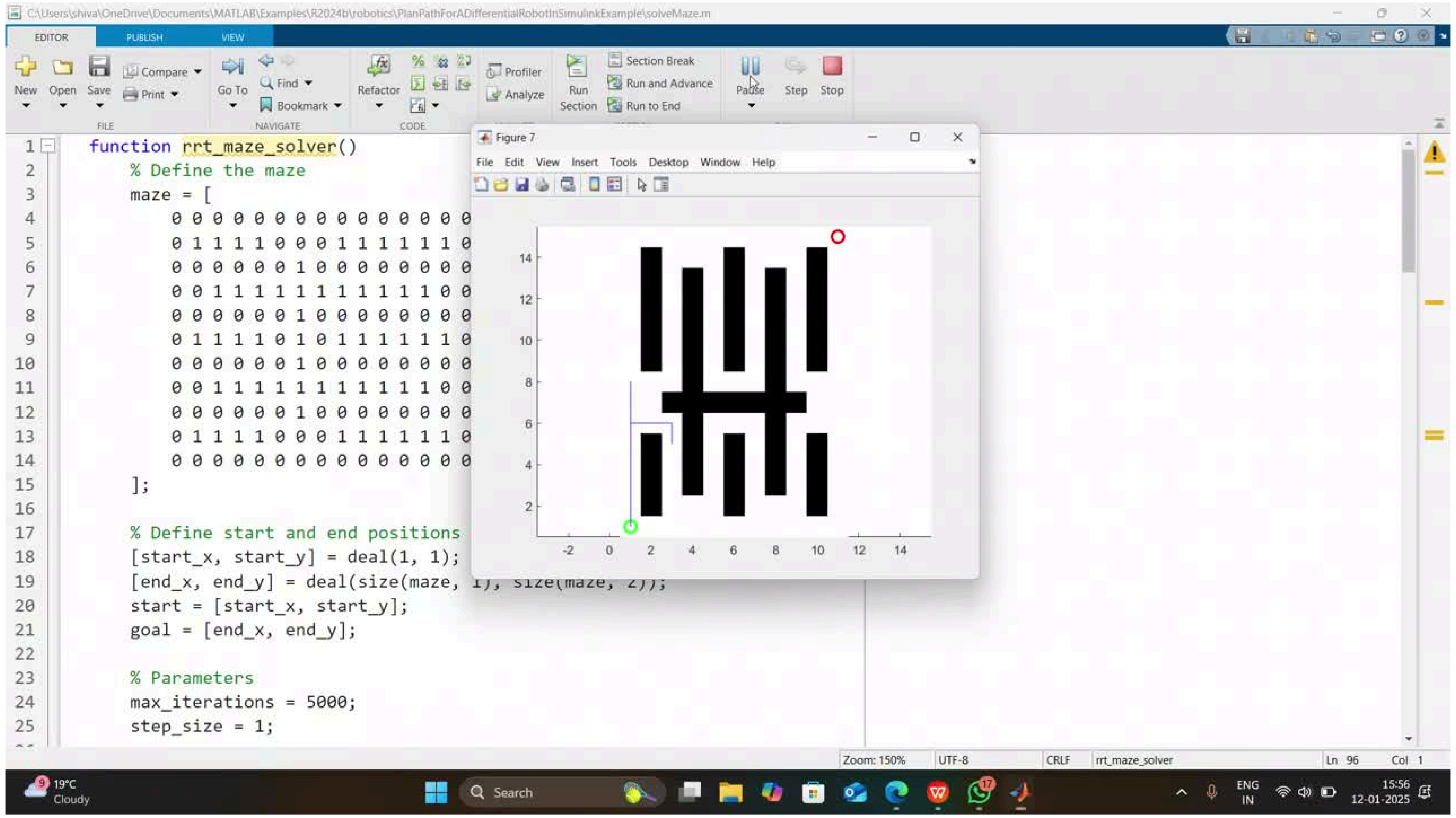
WEEK 3

Rapidly Exploring Random Trees

1. **Initialization:**
- Start with a single node (root) at the initial position.
- Define a goal region or state.
2. **Random Sampling:**
- Generate a random point in the search space.
3. **Nearest Neighbor Search:**
- Find the nearest node in the existing tree to the random point.
4. **Extension:**
- Extend the tree by adding a new node connected to the nearest node.
5. **Iteration:**
- Repeat until a path reaches the goal region or space is fully explored
-

WEEK 3

Rapidly Exploring Random Trees



```
function rrt_maze_solver()
    % Define the maze
    maze = [
        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
        0 1 1 1 1 0 0 0 1 1 1 1 1 1 0
        0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
        0 0 1 1 1 1 1 1 1 1 1 1 1 0 0
        0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
        0 1 1 1 1 0 1 0 1 1 1 1 1 1 0
        0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
        0 0 1 1 1 1 1 1 1 1 1 1 1 0 0
        0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
        0 1 1 1 1 0 0 0 1 1 1 1 1 1 0
        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    ];

    % Define start and end positions
    [start_x, start_y] = deal(1, 1);
    [end_x, end_y] = deal(size(maze, 1), size(maze, 2));
    start = [start_x, start_y];
    goal = [end_x, end_y];

    % Parameters
    max_iterations = 5000;
    step_size = 1;
```

WEEK 4

Refinement And Simulation

This week focuses on improving and testing robot trajectories for smoothness, stability, and efficiency.

1. **Trajectory Refinement:**
- **Fine-tune paths for smoothness, stability, and energy efficiency.**
- **Ensure compliance with constraints like joint limits and obstacle avoidance.**

2. **Simulation in MATLAB/Simulink:**
- **Validate refined trajectories in a controlled environment.**
- **Analyze metrics like energy consumption and stability.**
- **Visualize the robot's motion to ensure it meets physical and environmental constraints.**

WEEK 4

Refinement And Simulation

3. Stability and Balance Analysis:

- **Test balance during dynamic tasks using metrics like Zero Moment Point (ZMP).**
- **Simulate challenging conditions, such as uneven terrain.**

4. Control System Integration:

- **Implement controllers (e.g., PID) for precise path tracking.**
- **Test real-time adjustments to ensure smooth movement.**

5. Error Detection and Correction:

- **Identify and resolve issues like jerky motions, high energy use, or constraint violations.**
- **Iteratively refine trajectories and parameters.**

THANK YOU