# Maze Solver Robot: Detailed Project Document

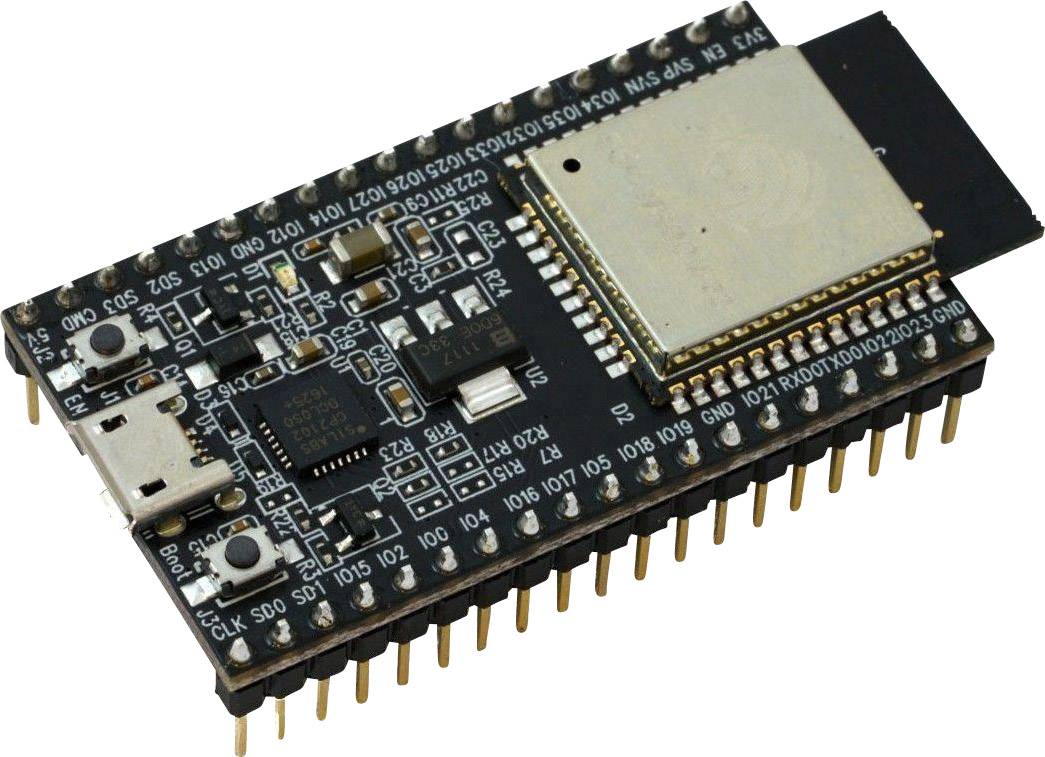
# Project Title: Maze Solver Robot Team Name: Team Techno Author: Ahmad Azzam Submission Date: 29 July,2025

**1. Introduction**

Maze-solving robots, often referred to as micromice, are autonomous machines designed to explore and find optimal paths in a structured environment, typically a maze. These robots are used for educational purposes, algorithm testing, and even in real-world scenarios like pathfinding in disaster zones. This document details the complete architecture of a maze-solving robot including hardware components, software architecture, and the pathfinding algorithm implemented.

**2. Hardware Components**

The hardware section of the robot is composed of key modules that allow for processing, sensing, locomotion, and energy storage. Each component has been selected based on size, performance, energy efficiency, and integration potential. Below is an in-depth explanation of each component, its purpose, and physical/electrical structure.



**Microcontroller: ESP32**

**Purpose:** Acts as the central processing unit, controlling the robot's logic, sensor input, motor output, and algorithm execution.

**Features:**

* Dual-core Tensilica Xtensa LX6 processor (up to 240 MHz)
* 520 KB SRAM, with external PSRAM option
* Integrated Wi-Fi and Bluetooth
* Multiple ADC, PWM, I2C, SPI, UART interfaces
* Deep sleep mode for power efficiency

**Structure:**

* Typically in a 38-pin DevKit or WROOM module form factor
* Compact PCB board (~25mm x 50mm)
* USB port for programming
* Exposed GPIO pins for peripheral connection

**Why ESP32?**

* Rich feature set with wireless capabilities
* Faster prototyping due to Arduino IDE support
* Energy efficient compared to traditional options

**Future Upgrade:** STM32 microcontroller with a custom-designed PCB to improve size, power efficiency, and reduce cost.



**Sensors: Sharp IR Sensor (GP2Y0A41SK0F)**

**Purpose:** To detect distance from walls and obstacles to aid in maze navigation.

**Features:**

* Analog output based on detected distance
* Detection range: 4 cm to 30 cm
* Low power consumption (~30 mA typical)

**Structure:**

* Rectangular compact plastic housing (~44mm x 18mm x 13.6mm)
* 3-pin JST connector: VCC (5V), GND, Vo (Analog output)
* Internal IR LED and phototransistor for triangulation

**Placement:**

* Front-facing for forward obstacle detection
* Side-facing (left/right) to detect walls

**Why Sharp IR?**

* Reliable analog signal
* Fast response time
* Easy integration with ESP32 ADC pins



**Motor Driver: TB6612FNG**

**Purpose:** Controls the speed and direction of the motors using low-power signals from ESP32.

**Features:**

* Dual H-Bridge output for 2 motors
* Maximum output current: 1.2A per channel (peak 3.2A)
* Standby mode for power saving
* PWM speed control support

**Structure:**

* Breakout board (~20mm x 15mm)
* 12-pin configuration:
  + Inputs: AIN1, AIN2, BIN1, BIN2 (for motor direction)
  + PWM: PWMA, PWMB
  + Power: VCC (logic, 3.3V/5V), VM (motor, 6–12V), GND
  + STBY pin to enable/disable the driver
  + Outputs: A01/A02 and B01/B02 for motors

**Why TB6612FNG?**

* More efficient and compact than L298N
* Minimal heat dissipation
* Low quiescent current



**Batteries: Lithium Polymer (Li-Po)**

**Purpose:** Provides portable and high-density power supply to the entire robot.

**Features:**

* High energy density
* Rechargeable and lightweight
* Nominal voltage: 3.7V per cell (commonly used: 2S → 7.4V)
* Capacity range: 500mAh to 1000mAh (depending on desired runtime)

**Structure:**

* Flat, rectangular pack in soft polymer casing
* Includes balance charging wires and discharge leads
* Protection circuit module (PCM) required for overcharge/discharge safety

**Why Li-Po?**

* Lightweight and compact form factor
* Suitable for high-current motors
* Easy to integrate in chassis



**Motors: N20 Gear Motors with Encoders**

**Purpose:** Provides the movement mechanism for the robot. Encoders allow for precise speed and distance tracking.

**Features:**

* Gear reduction: 30:1 or 50:1 for higher torque
* Operating voltage: 6V (optimal between 3V–12V)
* Speed: ~200–300 RPM (gear dependent)
* Torque: Higher than standard toy motors
* Shaft encoders provide two-channel (quadrature) output for RPM and direction

**Structure:**

* Cylindrical brushed DC motor (diameter: 12mm)
* Attached rectangular gearbox (10mm x 12mm x 26mm)
* Dual-channel magnetic encoder disk with optical sensors mounted at rear
* 6 wires: 2 for motor power, 4 for encoder signal

**Why N20 with Encoders?**

* Compact and modular
* High torque for quick turns and acceleration
* Feedback control for improved algorithm performance

**3. Software Architecture**

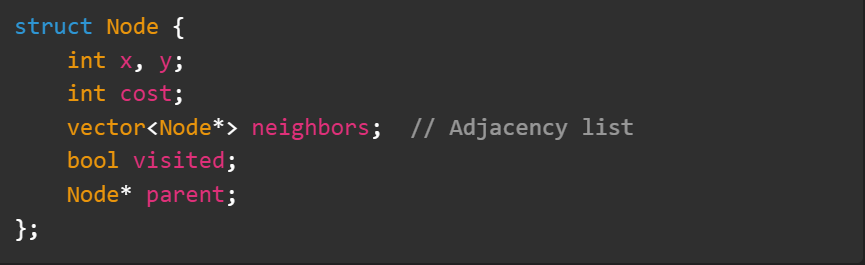
**Programming Language**

The firmware is developed using C++ within the Arduino IDE framework for ESP32. This allows faster prototyping and easier transition to bare-metal STM32 development later. **Data Structures and Algorithms**

A core part of maze-solving is managing spatial data efficiently. Here, graph-based data structures are implemented to represent the maze.

**Primitive vs User-Defined Data Structures**

Primitive types like arrays are used to store sensor data. However, a user-defined data structure, such as a struct or class, is used to represent the nodes in the maze. Example:



This structure mimics a graph where each cell in the maze is a node, and walls define the absence of edges between cells.

**Linked List to Graph Analogy**

* In linked lists, nodes are connected linearly via a next pointer.
* In graphs, each node contains a list of pointers (neighbors) to adjacent nodes.

Thus, to represent a maze:

* Each cell is a vertex (Node)
* Paths between cells are edges
* The entire maze is a graph structure where DFS/BFS/A\* can be applied.

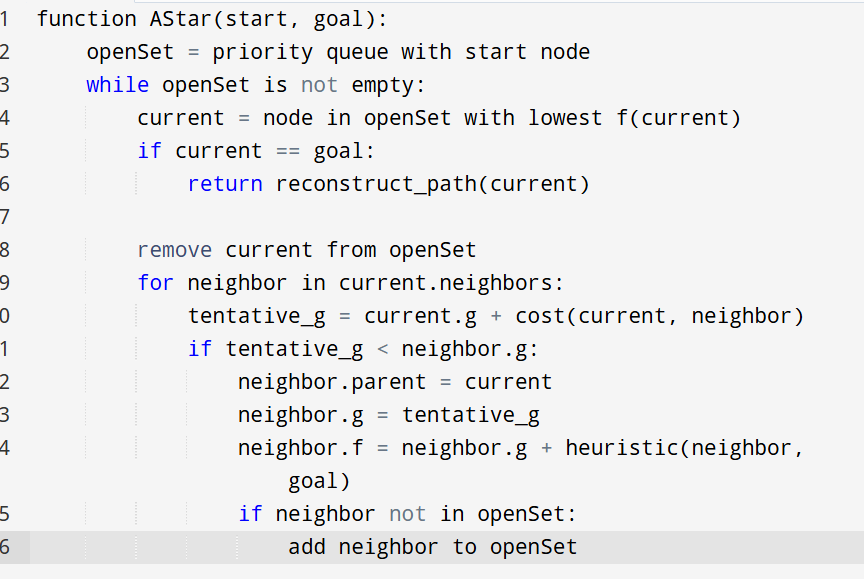
***4. Algorithm: A Pathfinding Algorithm*\***

* While DFS and BFS are simple to implement, A\* (A-Star) is used for optimal performance due to its heuristic approach, combining the strengths of Dijkstra's algorithm and Greedy Best-First Search.

A\* uses the formula:

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

* g(n) = cost from start to current node
* h(n) = heuristic estimate (e.g., Manhattan distance) from current to goal
* f(n) = total estimated cost



**Implementation Notes**

* Use a priority queue to always fetch node with lowest f(n)
* Manhattan distance is a good heuristic for grid-based mazes
* Path reconstruction is done by backtracking from goal to start using parent pointers

***Why A is Chosen*\***

* **Efficient:** Finds the shortest path with minimal computation
* **Flexible:** Heuristic function can be modified to suit maze layouts
* **Real-Time Friendly:** With proper optimizations, can work even in real-time maze exploration

**5. Final Notes & Future Improvements**

* **STM32 Custom PCB:** Transitioning to STM32 allows better real-time performance, reduced power, and production scalability.
* **Advanced Sensor Fusion:** Combining IR sensors with IMU or ultrasonic sensors for better obstacle avoidance.
* **Dynamic Maze Updating:** Real-time updates of the graph when new paths are discovered.
* **GUI Visualization:** Optional Bluetooth-based live visualization of maze exploration on PC or mobile.

**6. Conclusion**

This maze-solving robot is not only a tool for learning embedded systems and algorithms but also a prototype for autonomous exploration and pathfinding robots. Using efficient data structures, robust hardware, and a powerful algorithm like A\*, the robot can autonomously find and traverse optimal paths in any grid-based environment.

Future iterations with a custom PCB and improved sensors will make it more compact, faster, and production-ready.