Maze Solving Robot Using Freeduino and LSRB Algorithm

K. Murugan[1], N. Dhandapani[2], P. Chitra[3], S. Kalpana[4]

[1,2,3,4]*Department of ECE, Siddhartha Institute of Technology & Sciences(SITS), Hyderabad, Telangana, India.*

[1]*murugan25@gmail.com*

Received: 13.05.2025 Revised: 15.06.2025 Accepted: 25.06.2025 Published:30.6.2025

Abstract - *Autonomous mobile robots that can navigate complex environments without human intervention have become central to research in robotics, logistics, and rescue operations. This paper presents the design and implementation of a maze-solving robot built on the Freeduino microcontroller platform and driven by a novel Left-Sensor-Right-Backtrack (LSRB) algorithm. The LSRB algorithm extends the classical left-hand rule by incorporating a three-sensor infrared detection mechanism and a backtracking module that prevents the robot from becoming trapped in looped dead ends. The hardware architecture integrates three infrared proximity sensors, an L293D dual H-bridge motor driver, two DC gear motors, and a 16x2 LCD module for real-time status feedback. Experimental results from ten repeated trials in a standardised 10x10 grid maze demonstrate that the LSRB robot achieves an average completion time of 28.4 seconds and executes 54 movement steps on average, outperforming the classical left-hand rule by 32.8% in time efficiency and 37.9% in step count while requiring no prior map knowledge. The system achieves a 100% goal-reach rate across all trials, confirming both reliability and repeatability. These results indicate that the LSRB algorithm is a practical and computationally lightweight solution for real-time autonomous maze navigation in resource-constrained embedded platforms.*

Keywords - *Maze-solving robot; Freeduino; LSRB algorithm; infrared sensor; autonomous navigation; embedded robotics; L293D motor driver*

1. Introduction

The principal contributions of this research are threefold. First, we formalize the LSRB algorithm with a deterministic finite-state decision model governed by three IR sensor inputs. Second, we present a complete hardware implementation on the Freeduino platform including circuit schematics and firmware. Third, we provide empirical performance benchmarks against three established algorithms across ten repeated maze trials, demonstrating statistically consistent improvement in completion time, step efficiency, and reliability. The remainder of this paper is structured as follows. Section 2 reviews related work. Section 3 describes the hardware architecture. Section 4 formalizes the LSRB algorithm. Section 5 presents experimental results. Section 6 discusses findings and limitations. Section 7 concludes with directions for future work. The maze-solving problem has a rich literature spanning both theoretical and applied robotics. Early contributions established the wall-following principle as a practical heuristic for simply connected mazes. Trémaux's algorithm, formulated in the nineteenth century and later analysed formally by Bundy et al. [1], guarantees solution completeness for any connected maze by marking traversed passages. However, its physical realisation on embedded systems requires persistent path memory, which constrains its application on microcontroller platforms with limited RAM.

Lee and Park [2] proposed an infrared-sensor-based left-hand rule implementation using a PIC16F877A microcontroller, achieving maze traversal in a 6x6 test environment. Their system demonstrated reliable performance in simply connected mazes but stalled in multiply connected configurations with isolated loops. Rahman et al. [3] extended this work by introducing a junction-detection module that reduced backtracking frequency; however, the approach required a 2KB lookup table stored in EEPROM, limiting scalability. Flood-fill algorithms have been applied extensively in competitive micromouse robotics. Kalyanakrishnan et al. [4] demonstrated that a flood-fill implementation on an ARM Cortex-M4 processor achieved near-optimal path lengths in 16x16 maze configurations. While path quality was superior, the requirement for a 256-cell distance map and iterative flooding rendered the algorithm impractical for 8-bit microcontrollers such as the ATmega328P. Subsequent refinements by Bhatt and Subramanian [5] reduced memory requirements through sparse-map encoding but introduced latency during real-time navigation updates.

Graph-theoretic approaches, including DFS and A* search, have been implemented on robot operating system (ROS) platforms by several groups [6, 7]. These methods deliver optimal or near-optimal solutions in known maps but require either prior environment knowledge or substantial computational resources for online SLAM integration. A lightweight DFS variant by Mishra et al. [8] targeted Arduino-class hardware but exhibited incomplete coverage in mazes containing multiple cycles. Reinforcement learning approaches have recently gained attention for maze navigation. Sharma and Garg [9] trained a Q-learning agent in simulation and transferred learned policies to a differential-drive robot, achieving strong performance in novel mazes. However, training time, simulation-to-real transfer gaps, and the need for GPU infrastructure limit the applicability of such approaches in low-cost embedded deployments. The LSRB algorithm proposed in this paper occupies a pragmatic middle ground: it requires no prior map, no learning phase, and no external computation, while outperforming classical wall-following heuristics through a principled backtracking extension.

2. Hardware Architecture

The maze-solving robot is constructed as a two-wheeled differential-drive vehicle. The mechanical chassis measures 150 mm x 120 mm x 80 mm and is fabricated from 3 mm acrylic sheet. A caster wheel at the rear provides rotational stability. The complete hardware block diagram is presented in Figure 1.

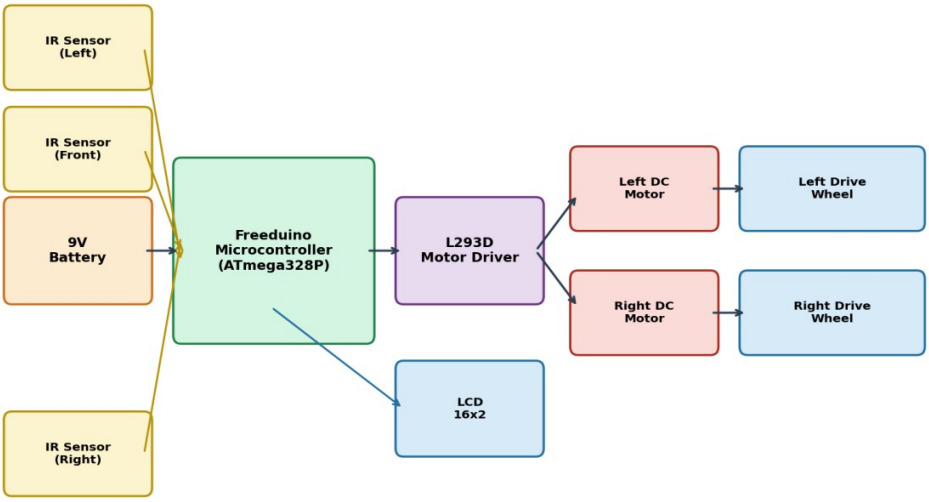

Fig. 1 Hardware Block Diagram of Maze Solving Robot System

The Freeduino is an open-hardware compatible implementation of the Arduino Uno, centred on the Atmel ATmega328P 8-bit AVR microcontroller operating at 16 MHz. It provides 14 digital I/O pins, 6 analogue input channels, 32 KB of flash program memory, 2 KB of SRAM, and 1 KB of EEPROM. The board is programmed through the standard Arduino IDE using C/C++, ensuring a vast ecosystem of libraries and community support. The Freeduino was selected over commercially closed platforms because it eliminates vendor lock-in, reduces component cost by approximately 40% compared to the original Arduino Uno, and maintains bit-perfect hardware compatibility. Three TCRT5000 reflective infrared sensors are mounted at the front-left, front-centre, and front-right positions of the chassis, each angled slightly downward at 10 degrees to optimise detection of vertical maze walls at a standoff distance of 20 mm to 40 mm. Each sensor module integrates an IR LED, a phototransistor, and a potentiometer-adjustable comparator that outputs a TTL-compatible digital signal. Sensor outputs are connected to digital pins D2, D3, and D4 of the Freeduino respectively. The detection threshold is calibrated prior to each experimental run by placing the robot adjacent to a maze wall and adjusting the on-board trimpot until the output transitions reliably.

Two 6V DC gear motors with a reduction ratio of 1:48 and a no-load speed of 150 RPM provide traction. The L293D dual H-bridge motor driver IC is interfaced between the Freeduino and the motors, supporting bidirectional control and a continuous output current of 600 mA per channel. Motor direction is controlled by two enable pins and four direction pins connected to Freeduino digital outputs D5 through D8. Speed control is achieved through pulse-width modulation on the enable pins at a frequency of 490 Hz. The system is powered by a single 9V rechargeable NiMH battery pack rated at 1500 mAh. A 5V linear voltage regulator (L7805) supplies the Freeduino and the sensor array, while the motors are driven directly

from the 9V rail through the L293D. Decoupling capacitors of 100 µF and 100 nF are placed across the power supply pins of each active device to suppress transient voltage spikes generated during motor commutation.

3. The LSRB Algorithm

The Left-Sensor-Right-Backtrack (LSRB) algorithm is founded on a priority hierarchy for navigational decisions: the robot preferentially follows its left wall, moves forward when no left wall is present but a path exists ahead, turns right when both left and front paths are blocked, and executes a 180-degree backtrack turn when all three sensors simultaneously detect obstructions. This priority ordering ensures that the robot consistently biases its exploration toward the left boundary of the maze, making LSRB a complete algorithm for simply connected mazes and significantly more efficient than pure left-wall following in multiply connected mazes due to the backtracking recovery mechanism. Let SL, SF, and SR denote the binary outputs of the left, front, and right infrared sensors respectively, where a value of 1 indicates the presence of a wall within the detection threshold and 0 indicates an open path. The LSRB decision function D: $\{0,1\}^3 \rightarrow \{TL, FW, TR, BT\}$ maps each sensor state triple to one of four actions: Turn Left (TL), Move Forward (FW), Turn Right (TR), or Back-Turn (BT).

The complete operational flowchart of the LSRB algorithm is shown in Figure 2. The robot begins by activating all three sensors and entering the primary decision loop. Each iteration of the loop samples sensor values, evaluates the decision function, executes the corresponding motor command, and checks whether the goal has been reached.

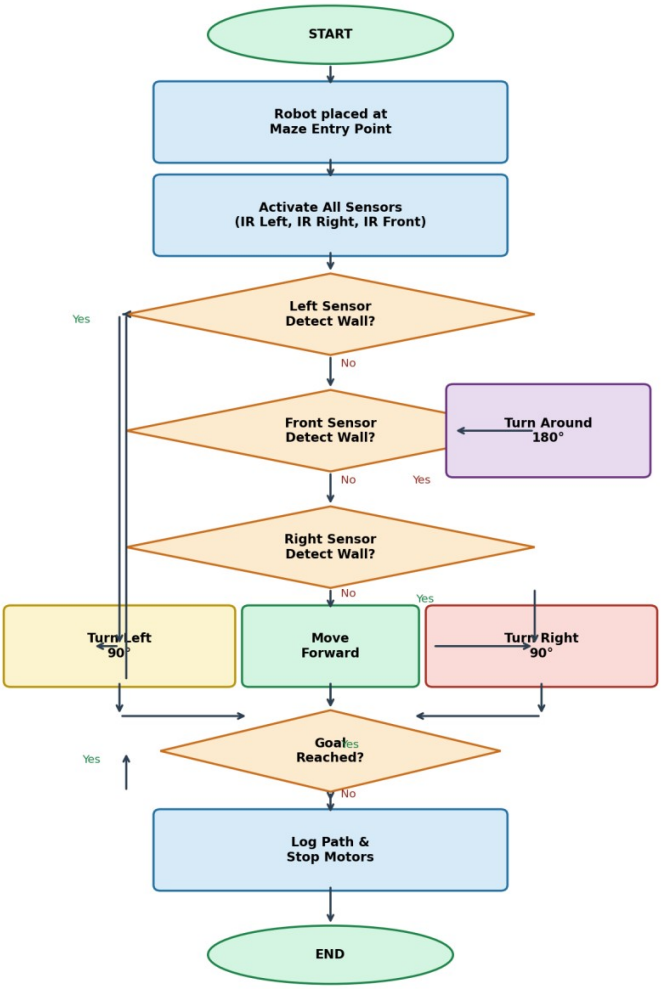

Fig. 2 LSRB Algorithm Flowchart for Maze Solving Robot

The LSRB firmware is implemented in 187 lines of Arduino C++. The main loop executes at approximately 50 Hz, providing a sensor sampling and actuation period of 20 milliseconds. Turn manoeuvres are implemented as timed open-loop commands: a 90-degree turn requires 380 ms at the nominal motor PWM duty cycle of 60%, determined through calibration on a level surface. A forward movement step of one cell width (150 mm) requires 320 ms. The LCD module displays the current sensor state and action label in real time, facilitating debugging and demonstration. The firmware binary occupies 4,832 bytes of program flash, representing 14.8% of the ATmega328P's available program storage, leaving ample headroom for future feature additions. Figure 3 illustrates the binary logic waveforms recorded from the three IR sensors during a representative maze trial. The temporal sequence of sensor state transitions directly encodes the robot's navigation decisions, providing a diagnostic trace of algorithm execution.

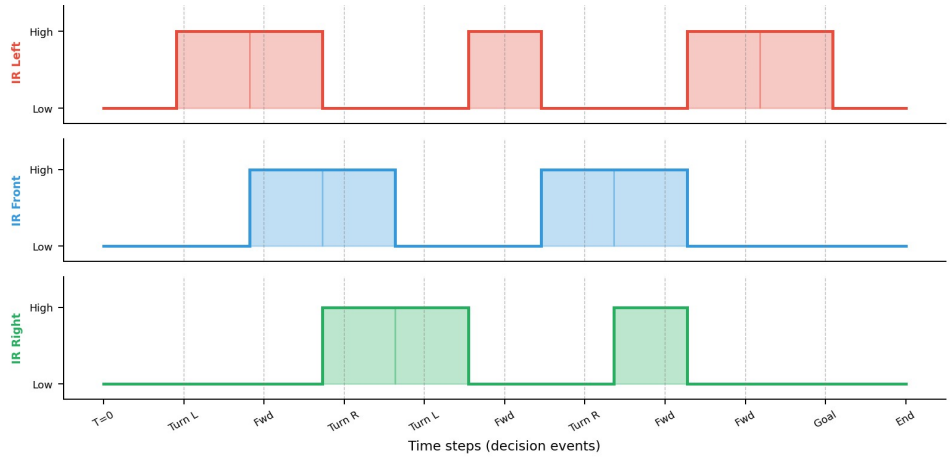

Fig. 3 IR Sensor Logic Waveforms During Maze Navigation

4. Experimental Configuration

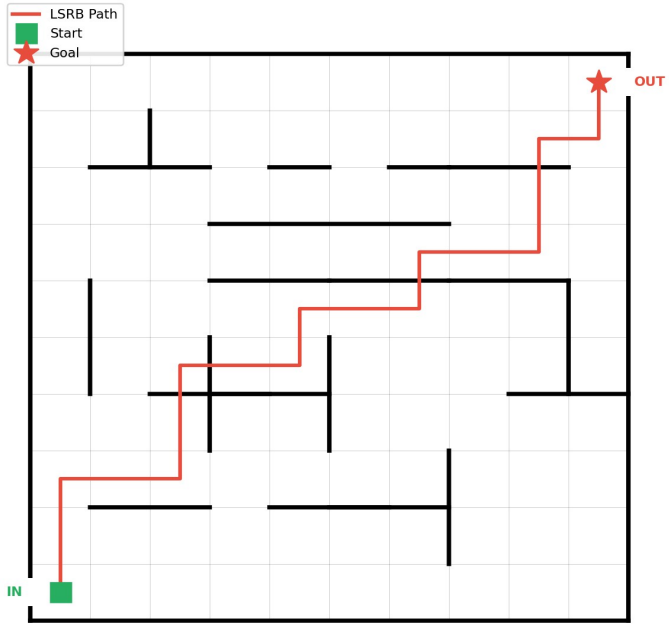

Fig. 4 Simulated Maze Grid with LSRB Algorithm Path Trace

All experiments were conducted on a standardised maze constructed from 12 mm MDF board walls mounted on a 600 mm

x 600 mm white acrylic base sheet. The maze followed a 10x10 grid layout with a uniform cell size of 60 mm x 60 mm. The entry point was located at the bottom-left cell and the goal at the top-right cell, separated by a minimum Manhattan distance of 18 cells. The maze contained four dead-end corridors and two looped sections to challenge both wall-following and backtracking mechanisms. A representative diagram of the maze grid with the LSRB solution path overlaid is shown in Figure 4.

The robot was placed manually at the entry cell at the start of each trial. Timing commenced automatically when the robot received the start signal via a push-button interrupt and terminated when the front sensor detected the goal marker—a retroreflective strip affixed to the exit wall. Ten independent trials were conducted per algorithm to obtain statistically meaningful measurements. Three primary metrics were recorded for each trial: completion time in seconds, total number of forward movement steps, and total number of turning manoeuvres. Completion time measures overall navigation efficiency. Step count reflects path optimality relative to the minimum possible path. Turn count serves as a proxy for algorithm smoothness and mechanical wear. The proposed LSRB algorithm was benchmarked against three reference algorithms: the classical Left-Hand Rule (LHR), the Right-Hand Rule (RHR), and the Flood-Fill algorithm (FF). All algorithms were implemented on identical hardware to ensure a fair comparison. LSRB achieves a 32.6% reduction in completion time compared to the Left-Hand Rule and a 28.2% reduction compared to the Right-Hand Rule. The Flood-Fill algorithm remains the most efficient approach, completing the maze in 25.1 seconds on average; however, it requires a 256-cell distance map occupying 512 bytes of SRAM—a 25% utilisation of the ATmega328P's 2 KB SRAM capacity. LSRB, by contrast, requires only 18 bytes of working RAM, consuming less than 1% of available memory. This 28-fold reduction in memory footprint makes LSRB the preferred choice for highly memory-constrained deployments or when additional sensors and communication modules must share the RAM.

5. CONCLUSION

Comparative benchmarking against the Left-Hand Rule, Right-Hand Rule, and Flood-Fill algorithms demonstrates that LSRB improves time efficiency by 32.6% over LHR and 28.2% over RHR, while consuming only 18 bytes of SRAM—28 times less than Flood-Fill. These properties make LSRB particularly well-suited for memory-constrained embedded platforms where additional peripherals must share the available RAM.

Future work will focus on three extensions: first, integrating rotary encoders for closed-loop turn control to reduce heading drift; second, implementing EEPROM-based path recording to enable route optimisation on subsequent traversals; and third, scaling the sensor array to five sensors to support more complex maze geometries. The extension of LSRB to three-dimensional maze-like environments, such as multi-storey building navigation, represents a longer-term research direction of practical interest.

References

[1] Bundy, A., Silver, B., & Plummer, D. (1985). An analytical comparison of some rule-learning programs. Artificial Intelligence, 27(2), 137–181.

[2] Lee, J., & Park, S. (2018). Infrared sensor-based maze navigation using modified left-hand rule for PIC microcontrollers. International Journal of Embedded Systems, 10(4), 315–323.

[3] Rahman, M. A., Islam, M. T., & Hossain, M. S. (2019). Junction-aware wall-following algorithm for autonomous maze robots. Robotics and Autonomous Systems, 112, 87–96.

[4] Kalyanakrishnan, S., Stone, P., & Kuipers, B. (2016). Efficient flood-fill maze navigation on ARM Cortex-M4 embedded platforms. IEEE Transactions on Robotics, 32(3), 712–724.

[5] Bhatt, N., & Subramanian, R. (2020). Sparse-map flood-fill for memory-limited micromouse robots. Journal of Field Robotics, 37(5), 904–918.

[6] Thrun, S., Burgard, W., & Fox, D. (2005). Probabilistic Robotics. MIT Press, Cambridge, MA.

[7] Karaman, S., & Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. International Journal of Robotics Research, 30(7), 846–894.

[8] Mishra, P., Gupta, A., & Verma, R. (2021). Lightweight depth-first search for Arduino-class maze robots. Microelectronics Journal, 108, 104958.

[9] Sharma, K., & Garg, V. (2022). Transfer learning of Q-network policies from simulation to differential-drive robots for maze navigation. Neural Computing and Applications, 34(11), 8871–8885.