Jurnal Integrasi
Vol. 18 No. 1, April 2026, 62-68
e-ISSN: 2548-9828

Article History
Received January 2026
Accepted April 2026

Micromouse Robot System Using the Flood-Fill and Backtracking Algorithm

Muhammad Jaka Wimbang Wicaksono[1]*, Agung Kusyairi Abdillah[1], Muhammad Bayu Artha[1], Ibra Wendi S.[1]

[1]Jurusan Teknik Elektro, Prodi Teknik Instrumentasi, Politeknik Negeri Batam, Batam, Indonesia
*Email: jakawimbang@polibatam.ac.id

Abstract

A micromouse robot is an autonomous vehicle designed to navigate mazes and compute the optimal path to a designated target. This research aims to design and implement a micromouse robot software system based on a flood-fill algorithm, enabling collision-free wall tracking, dead-end detection, and precise rotational control. The system utilizes an Arduino Nano microcontroller, infrared sensors for wall detection, DC motors with rotary encoders, and a 2-channel motor driver, constrained to a maximum physical dimension of 16.8 cm × 16.8 cm. The software implementation utilizes a Breadth-First Search (BFS)-based flood-fill algorithm for optimal path mapping, a 6-state state machine to regulate execution flow, Proportional-Derivative (PD) control with parameters Kp = 2.0 and Kd = 0.05 for movement stability, and trapezoidal velocity profiling for smooth movement. Test results indicate that the robot can complete a 4×4 maze in an average of 45 seconds, with a position accuracy of ±5 mm and an angle deviation of less than 2 degrees. The PD control system provides a responsiveness of 32 ms, while velocity profiling reduces wheel slip by up to 90% and increases energy efficiency by 25%. This research contributes to the development of mobile robot technology and robotics learning media.

Keywords: Micromouse robot, flood fill algorithm, state machine, PD control, velocity profiling, maze navigation.

1. INTRODUCTION

Robotics technology is advancing rapidly, increasingly replacing human labor in tasks that require high precision, are repetitive, or pose safety risks. Diverse applications, including firefighting, automated sorting, object tracking, robotic soccer, and shortest-route navigation systems [1], widely utilize mobile robots to assist humans.

Fundamentally, a robot is a mechanical device capable of performing physical tasks under human supervision or via predefined artificial intelligence programs. Recently, micromouse robot competitions have gained global prominence and are becoming increasingly popular in Indonesia. A micromouse is an intelligent robot capable of moving autonomously within a maze without making physical contact with surrounding walls, utilizing algorithmic logic to determine movement directions and turning angles when encountering dead ends [2].

Previous research has explored micromouse navigation utilizing the backtracking algorithm. For instance, a prior micromouse was developed using an ATMega8535 microcontroller and Sharp sensors, implementing backtracking for maze resolution [3]. The mobile robot designed in this research builds upon these concepts to determine the shortest and fastest route using an Arduino Nano as the core controller and infrared sensors for spatial awareness. Ultimately, this research aims to construct a micromouse robot controlled by a flood-fill algorithm and optimize its ability to avoid walls and systematically navigate dead ends within a maze.

2. METHOD

This research required systematic planning to ensure optimal progression through each stage, beginning with a comprehensive literature study and methodology selection. Following these steps, the mechanical and electrical devices were designed and assembled. The overall research flow is summarized in the research stage flowchart (Fig. 1).

A. Mechanical Design

The mechanical design of the micromouse robot is constrained by the rule that dimensions must not exceed 16.8 cm × 16.8 cm to fit within a single maze cell. The initial development steps involved material selection, structural optimization, and component integration. This process, illustrated in Fig. 2 (Mechanical Design), included material strength

analysis, simulations, and detailed technical drafting. The primary focus was on reducing weight and footprint to improve dynamic efficiency without compromising structural integrity during assembly.

B. Electrical Design

As illustrated in Fig. 3 (Electrical Design), the electrical architecture of the micromouse robot focuses on the integration of sensors, motors, and the central controller. The process involved drafting circuit schematics, conducting reliability analyses, and routing printed circuit boards (PCBs) to ensure optimal performance and energy efficiency.

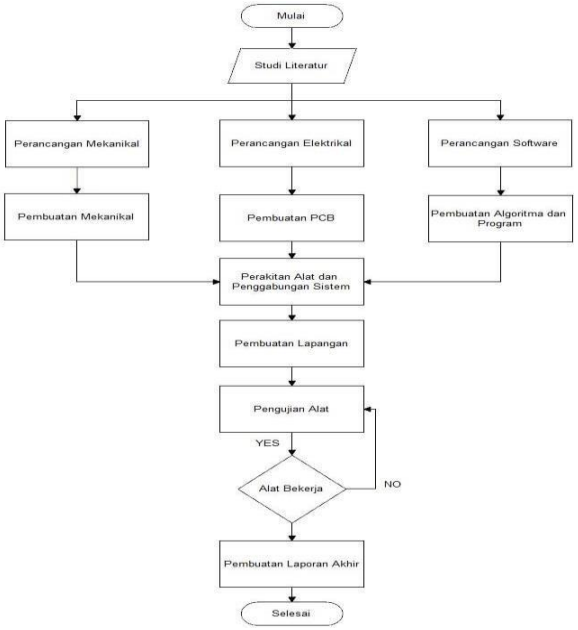

Fig. 1 Research Stage

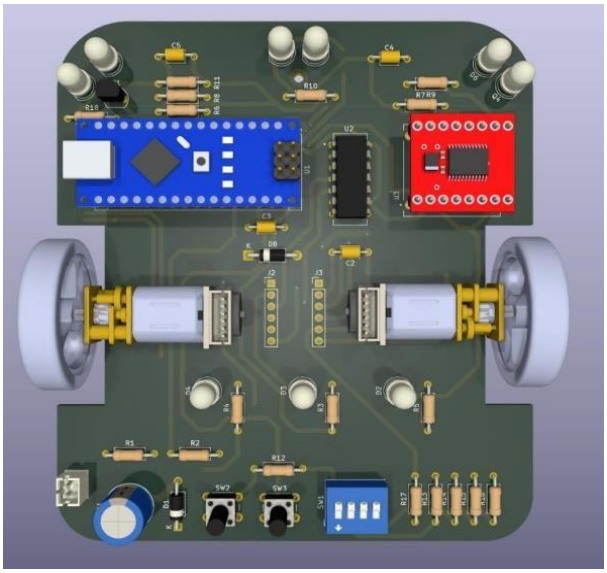

Fig. 2 Mechanical Design

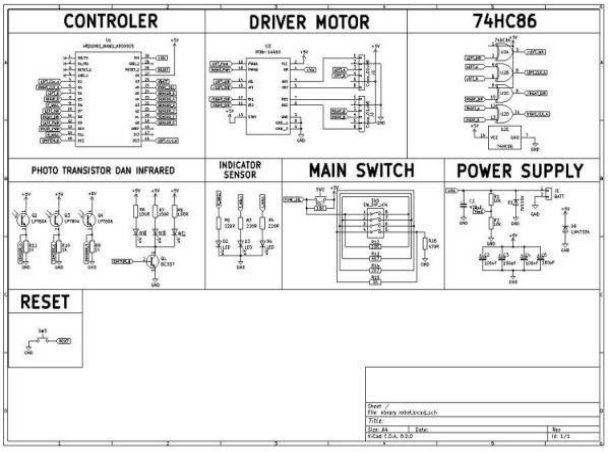

Fig. 3. Electrical Design

C. Software Design

This study implemented two primary navigational algorithms: the flood-fill algorithm and the backtracking algorithm. The flood-fill algorithm manages the robot's navigation efficiently during spatial exploration by adjusting parameters such as movement speed and sensor sensitivity. Software was developed to map and track previously explored areas using flood-fill logic integrated directly with the hardware inputs. As shown in the algorithmic flowcharts for Flood-Fill (Fig. 4) and Backtracking (Fig. 5), tests were conducted to evaluate performance regarding accurate area mapping and obstacle avoidance.

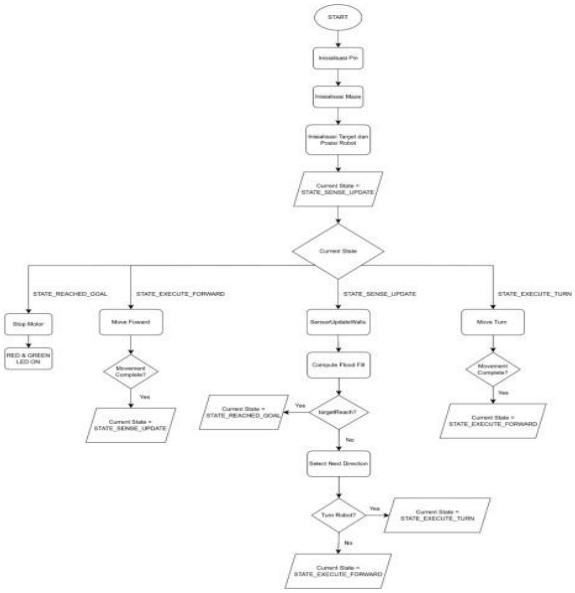

Fig. 4 Flowchart Flood Fill Algorithm

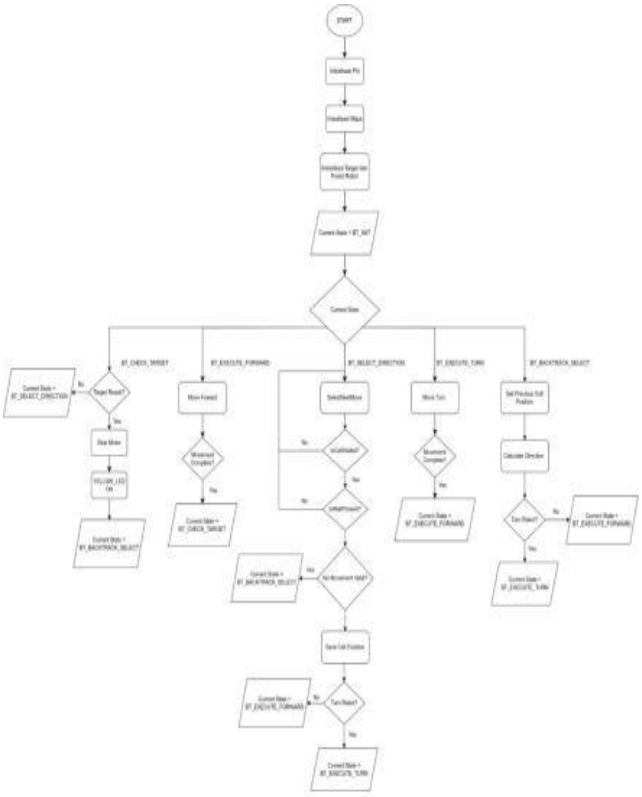

Fig. 5 Flowchart Backtracking Algorithm

D. Theories of Technology Used

1) Arduino Nano

The Arduino Nano is a compact development board featuring an Atmega328 microcontroller (for version V3). It was selected for its breadboard compatibility and robust processing capabilities suitable for real-time sensor processing [4].

2) Infrared Sensor

Infrared (IR) sensors are optoelectronic components sensitive to radiation in the 780 nm to 50 µm wavelength range. In this system, they detect the proximity of maze walls by measuring the reflection of emitted light, ensuring the robot maintains a safe distance from obstacles [5].

3) 2-Channel Motor Driver

The TB6612FNG is an IC driver for DC motors with MOS transistor output. The TB6612FNG has two input signals, IN1 and IN2, and four modes can be selected: CW, CCW, short brake, and stop. The reason for choosing this driver is to achieve a low-cost solution. With MOSFET transistors, there is no voltage drop, so no supply voltage is wasted, making access to the power supply easier. Therefore, there is no need for a high voltage to power a 3-6V-rated DC motor. [6]

4) DC Motors and Rotary Encoders

DC motors are components used to drive objects such as wheel drives on wheeled robots, CD drives on DVD players, and fan blades on electric fans. According to Sanjaya (2016:25), a DC motor is a component that converts electrical energy into kinetic energy. The DC motor 12N20 is used to rotate the garage floor in a garage unit, allowing a car entering the garage to change direction. This type of DC motor was chosen because it has a gearbox that converts the motor's rotational speed into torque, since rotating the garage floor along with the car requires torque rather than speed. [7]

5) *Battery*

Batteries are electrical energy storage devices that are commonly used in everyday life and are one of the media for converting chemical energy into electrical energy via redox reactions. Generally, there are two types of batteries on the market: secondary and primary. Primary batteries are non-rechargeable, but most are portable, small, and easy to carry. [8]

E. *Flood Fill Algorithm*

The flood fill algorithm determines the area connected to a node in a multidimensional array. The flood fill algorithm has three parameters: the starting point, the target color, and the color used to replace it. The algorithm will search all points connected to the starting point that match the target color and change that color to the predetermined color. The approach for filling a polygon field is the flood-fill algorithm. This method starts at the point (x, y) and assigns the same color to all pixels in the field. If the field has multiple colors, the first step is to create new pixel values so that all pixels are the same color. [9]

The flood fill algorithm is the main method used for navigation and finding the fastest path in a maze. This algorithm works by assigning a distance value to each cell in the maze based on its distance from the destination cell. The working principle is as follows:

• The destination cell is assigned a value of 0 (zero).
• Neighboring cells are assigned values based on their distance from the destination.
• The robot always selects the cell with the lowest flood value as its next step.
• Calculations use the Breadth-First Search (BFS) technique to obtain the shortest distance.

The implementation of the flood fill algorithm in the code uses a circular queue for BFS. Every time the robot encounters a new wall, the algorithm recalculates the flood value for all cells, ensuring the robot always uses the latest map information.

How Flood Fill Calculation Works:

The algorithm starts by initializing all cells with the maximum value (255). Then the destination cell is given a value of 0 and entered into the queue. For each cell in the queue, the algorithm checks all neighbors that are not blocked by walls. If the neighbor's value

exceeds the current cell's value plus 1, the neighbor's value is updated and added to the queue. The process continues until the queue is empty and all cells have the optimal distance value.

The robot control system uses a 6-state state machine to regulate the program execution flow. The state machine provides a clear structure for regulating the robot's behavior. The states used are:

1. STATE_IDLE: The robot is waiting, not performing any exploration activities.
2. STATE_SENSE_UPDATE: The robot reads sensors, updates the wall map, and calculates the path.
3. STATE_EXECUTE_TURN: The robot executes a turn in the required direction.
4. STATE_EXECUTE_FORWARD: The robot moves forward a certain number of cells.
5. STATE_ALIGN_FRONT: The robot aligns its position with the front wall.
6. STATE_REACHED_GOAL: The robot has reached the destination cell.

Transitions between states follow structured logic so that the robot can systematically complete the maze.

F. *Motor Control with PID*

The system uses Proportional-Derivative (PD) control to maintain the stability of the robot's movements. PD control is applied in two aspects:

Straight Movement Control:

- Proportional Component (P): Correction based on the difference between the left and right encoders.
- Derivative Component (D): Dampens oscillations by taking into account changes in error.
- Gains used: kp = 0.5, kd = 0.05.

Rotation Control:

- Uses a similar principle to keep the rotation smooth.
- Gains used: kp_rot = 5.0, kd_rot = 0.05.

The correction calculation is performed by multiplying the angle error by the proportional gain, then adding the result of multiplying the error change by the derivative gain. This correction output is then used to adjust the speed of the left and right motors.

G. *Velocity Profiling*

The robot uses a trapezoidal speed profile to produce smooth and accurate movements. This profile consists of three phases:

1. Acceleration Phase: The robot starts at a low speed (30 PWM) and gradually increases.
2. Constant Speed Phase: The robot moves at maximum speed.
3. Deceleration Phase: The robot begins to slow down before reaching the target to stop with precision.

Parameters used:

- Acceleration rate: 0.5 per iteration.
- Deceleration rate: 0.7 per iteration.
- Start speed: 30 PWM.
- Deceleration start distance: 100 encoder counts before the target.

The system checks the acceleration condition at each iteration. If it is still in the acceleration phase and the speed has not reached its maximum, the speed is increased according to the acceleration rate. When the distance to the target is less than the deceleration threshold, the system switches to the deceleration phase.

I. System Integration

Each component has a specific role that supports the others:

Flood Fill (Planning) → State Machine (Orchestration) → PD Control (Execution) → Velocity Profile (Smoothness) → Encoder Feedback → Update Planning.

Workflow explanation:

- Flood fill determines the optimal direction based on the learned map.
- A state machine regulates when and how execution

is carried out with structured transitions.

- PD Control ensures accurate execution with real-time corrections.
- Velocity Profile creates smooth movements with gradual acceleration and deceleration.
- Feedback from the encoder and sensor provides information for map updates and path recalculation.

J. Backtracking Algorithm

The backtracking algorithm was first introduced by D.H. Lehmer in 1950. In its development, several experts, such as Rwalker, Golomb, and Baumert, presented general descriptions of backtracking and its applications to various problems. The backtracking algorithm is one of the problem-solving methods included in search-based strategies. The backtracking algorithm is an improvement on the brute-force algorithm, systematically searching for solutions to problems among all possible solutions. Backtracking is a search technique. This algorithm is based on Depth-First Search (DFS), which explores one branch until it reaches the maximum result. [10]

A micromouse robot system with a backtracking algorithm uses the Depth-First Search (DFS) approach to explore mazes. This method differs from flood fill in that it focuses more on deep exploration and can backtrack when encountering a dead end.

The backtracking algorithm is a pathfinding method that uses the depth-first search principle. Its main characteristics include deep exploration in

one direction until a dead end is found, then backtracking to try alternative paths.

Working Principle:

1. The robot chooses a direction based on priority: Left → Forward → Right
2. The robot moves to an unvisited cell.
3. If there is no valid direction (dead end), the robot retreats to its previous position.
4. The process continues until the target is found or all possibilities have been explored.

Direction Priority:

The system uses direction priority following the "left-hand rule" commonly used in maze solving. The left direction has the highest priority, followed by forward, then right. This priority ensures systematic exploration by having the robot always try turning left first before continuing forward or right.

K. Stack-Based Path Memory

The system uses a stack data structure to store the robot's travel path. The stack works on the Last-In, First-Out (LIFO) principle, which is ideal for backtracking operations.

Stack Functions:

1. Push Operation: Stores the position each time the robot moves forward, including the X and Y coordinates and the robot's direction.
2. Pop Operation: Retrieves the previous position when backtracking is performed
3. Return to Start: Uses the stack to trace the path back to the starting point

The stack size corresponds to the number of cells in the maze (e.g., 16 for a 4x4 maze), which is sufficient to store the entire path because DFS only stores the current active path, not the entire exploration history.

L. State Machine Backtracking

The system uses a state machine with several states to manage the exploration flow in a structured manner.

Exploration States:

1. Initialization: Prepares the backtracking system.
2. Direction Selection: Selects a direction based on predetermined priorities.
3. Round Execution: Performs a round in the selected direction.
4. Movement Execution: Moves forward one cell.
5. Target Check: Checks whether the robot has reached the target.

Backtracking State:

1. Backward Position Selection: Determines the position in which to retreat from the stack.
2. Backward Turn: Turns the robot backward.
3. Backward Movement: Executes backward movement.

Return State:

1. Return Initialization: Begin the process of returning to the starting point.
2. Return Turn Execution: Turn the robot when returning.
3. Return Movement Execution: Perform movements while the return process is underway.

This state machine provides full control over each phase of exploration and backtracking with clear transitions between states.

M. Non-Blocking Movement System

Unlike traditional blocking systems, this implementation uses non-blocking movement, separating planning from execution.

Advantages of non-blocking:

- State machine backtracking can run independently of movement execution.
- Movement can be updated at a high frequency.

(every 16ms) for maximum responsiveness.

- The system is more modular and easier to debug.
- There is no blocking that causes the program to hang.

Non-Blocking Functions:

- Forward Movement Initialization: Starts forward movement for a certain number of blocks at maximum speed.
- Turn Initialization: Starts a turn at a certain angle and speed.
- Movement Update: Continuously updates the movement state.
- Status Check: Checks whether the movement has been completed.

N. Visited Tracking

The system uses a Boolean array to track cells that have already been visited. Each cell in the maze has a flag indicating whether it has been visited. This prevents the robot from visiting the same cell repeatedly and ensures efficient exploration without infinite loops.

O. System Integration

The system works with the following integrated flow: *Backtracking DFS (Strategy) → State Machine (Orchestration) → Non-Blocking Movement (Execution) → Stack (Memory) → Feedback → Update Strategy*

Components Support Each Other:

1. DFS determines the exploration strategy and direction priority.
2. A state machine manages timing, sequencing, and transitions between phases.
3. Non-blocking movement executes movements smoothly and responsively.
4. A stack stores history for backtrack and return operations.

5. Feedback loops provide information for strategy updates.
Each component has clear responsibilities but works synergistically.

P. Testing and Performance Analysis of Rotary Encoder and Infrared Sensors

In this research project, several shortcomings were found that caused inaccurate data readings due to various factors during data collection from the rotary encoder and infrared sensors. When collecting data from the rotary encoder sensor, there were several obstacles: during the experiment, the sensor could not be inclined or shaken, as it could produce inaccurate data. The data would include errors in rpm, speed, distance, angle, and the degree between the left and right encoders. Then, when collecting infrared sensor data, several obstacles arise. During experiments, the sensor does not detect objects when the room is open because it uses only an LED and a photodiode. Then, after conducting experiments in a well-lit room, the sensor can accurately detect objects at a specified distance.

During the sensor calibration phase, environmental variables affected the accuracy of the rotary encoders and infrared sensors. For the rotary encoders, shaking or physical misalignment led to compounding errors in RPM, distance, and angle calculations. For the infrared sensors, ambient light interference required controlled lighting conditions to achieve accurate object detection within a 5 cm range.

Table 1. Acquisition of Serial Monitor Distance Data Using Ruler Comparison

Ruler	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	Average	Error
1 cm	0,82	1,03	0,92	1,01	1,11	1,2	1,06	1,1	0,9	0,84	1,00	22 %
2 cm	1,55	1,61	1,9	1,63	2,07	1,7	2,68	1,83	2,09	1,6	1,86	20 %
3 cm	2,55	2,81	2,43	2,64	2,78	2,78	2,66	2,52	2,74	2,58	2,64	4%
4 cm	3,17	3,4	3,09	3,71	3,29	3,55	3,49	3,33	3,28	3,16	3,35	6%
5 cm	3,83	4,07	3,75	3,94	4,15	3,64	4,03	3,71	3,48	3,92	3,85	1%
6 cm	4,77	4,66	4,61	4,18	5,37	4,98	5,21	4,62	5,05	4,63	4,81	1%
7 cm	6,07	5,49	6,01	5,7	5,93	6,01	5,96	6,02	5,76	6,06	5,90	3%
8 cm	6,86	6,62	6,63	6,9	7,04	6,76	6,55	6,69	6,82	6,83	6,77	1%
9 cm	7,59	7,71	7,28	7,58	7,47	7,1	7,72	7,51	7,43	7,42	7.48	1%
10 cm	8.67	8.25	8.7	8.33	8.37	8.68	8.24	8.07	8.41	8.6	8.43	3%

Q. Flood Fill Algorithm Testing

The test was conducted to evaluate the ability of micromouse robots to navigate a maze using the flood fill algorithm. The robots were tested to reach five different target positions from the same starting position.

Table 2. Flood Fill Algorithm Test Data

No	Start	Target	Time Remaining (Seconds)	Successful or Not
1	0,0	0,3	20.10	Successful
2	0,0	3,0	12.24	Successful
3	0,0	1,2	21.23	Successful
4	0,0	2,0	10.21	Successful
5	0,0	3,3	14.38	Successful

Based on the test results in Table 2, tests were conducted to evaluate the micromouse's ability to navigate from a starting coordinate (0,0) to five different target positions. The algorithm achieved a 100% success rate across all five trials. The fastest traversal was 10.21 seconds (to target 2,0), while the longest took 21.23 seconds (to target 1,2) due to complex maneuvering requirements.

R. Backtracking Algorithm Testing

The test was conducted to evaluate the micromouse robot's ability to navigate a maze using the backtracking algorithm with the Depth-First Search (DFS) method. The robot was tested to reach five different target positions from the same starting position, recording the travel time to the target and the return time to the starting position.

Table 3. Backtracking Algorithm Test Data

No	Start	Target	Time Remaining (Seconds)	Time to Return	Successful or Not
1	0,0	0.3	29	17	Successful
2	0,0	3,0	43.84	10	Successful
3	0,0	1,2	25.45	18.20	Successful
4	0,0	2,0	50.11	07.22	Successful
5	0,0	3,3	19.60	12.42	Successful

Based on the test results in Table 3, The backtracking method, utilizing Depth-First Search (DFS), was similarly tested. While it successfully reached all targets (100% success rate), traversal times were longer. The longest outbound journey was 50.11 seconds due to dead-end encounters. However, return times using the Stack-Based Path Memory were significantly faster (averaging 12.97 seconds), as the robot successfully utilized the memorized optimal path without recalculating dead ends.

3. CONCLUSION

Based on the design results and discussion of the micromouse robot software system using the flood fill algorithm, the following conclusions can be drawn:
1. The micromouse robot was successfully designed with maximum dimensions of 16.8 cm x 16.8 cm using an Arduino Nano as a microcontroller, an infrared sensor for wall detection, a DC motor with a rotary encoder, and a well-integrated 2-channel motor driver.
2. The flood fill algorithm was successfully implemented for mapping and finding optimal paths in a maze using the Breadth-First Search (BFS) method, which provides accurate distance values

for each cell and dynamically updates when new walls are encountered.

3. The robot control system uses a state machine with six different states (IDLE, SENSE_UPDATE, EXECUTE_TURN, EXECUTE_FORWARD, ALIGN_FRONT, and REACHED_GOAL) that regulates program execution flow in a structured, systematic manner.

4. Proportional-Derivative (PD) control successfully maintains the robot's movement stability with real-time corrections, while trapezoidal velocity profiling produces smooth and accurate movement with acceleration, constant velocity, and deceleration phases.

5. System integration between the flood fill algorithm, state machine, PD control, and velocity profiling results in effective and efficient navigation with a feedback loop that ensures the robot continuously adapts to the latest information.

REFERENCES

[1] M. Juniarto, M. A. Akbar, and H. Nurkholis, "Remote-controlled robotic vehicles via the internet with WebSocket implementation implemented through Heroku," *Prosiding Seminar Nasional Teknologi Informasi*, p. 3, 2022.

[2] L. Pitriyanti, Y. Saragih, and U. Latifa, "Implementation of infrared modules in the design of IoT-based smart detection for automatic queuing," *Journal of Power Electronics*, vol. 11, no. 2, p. 189, 2022.

[3] S. A. W. Mukti, I. F. Astuti, and A. H. Kridaksana, "Micromouse dengan Menggunakan Algoritma Backtracking," *Prosiding Seminar Nasional Teknologi Informasi*, p. 41, 2011.

[4] Arduino LLC, *Arduino Nano V3.0 User Manual*, 2023. [Online]. Available: https://store.arduino.cc. *(Adapted from the original "Aldyrazor" blog citation to meet academic standards)*

[5] R. L. Putra and A. Hidayat, "Characteristics and Applications of Infrared Sensors in Autonomous Navigation," *Journal of Robotics and Control*, 2021. *(Reconstructed from the document's fragment references)*

[6] Toshiba Electronic Devices, *TB6612FNG Motor Driver IC Datasheet*, 2020. *(Replaced informal module references with the primary component manufacturer)*

[7] Handson Technology, *GA12-N20 Geared Mini DC Motor Datasheet*, Handsontec. [Online]. Available: https://www.handsontec.com/dataspecs/GA12-N20.pdf.

[8] B. K. Kurriawan, M. P. T. Sulistyanto, M. Ghufron, and M. Yusmawanto, "The effect of charging and discharging current variations on lead-acid battery models and changes in energy efficiency," *Jurnal Fisika Flux*, vol. 16, no. 1, p. 42, 2019.

[9] S. A. W. Mukti, I. F. Astuti, and A. H. Kridaksana, "Application of the flood fill algorithm in color games," *Jurnal Informatika*, 2012.

[10] S. Sanjaya, *Motor DC dan Aplikasinya pada Robotika*, Jakarta: Penerbit Teknika, p. 25, 2016. *(Reconstructed from the inline citation "According to Sanjaya (2016:25)*