ORIGINAL ARTICLE

Implementation of Coverage Path Planning Based on Modified NSGA-II and Kalman Filter on GoPiGo3 Robot

Monex Sharma[1], Hari Kumar Voruganti[1,*]

[1]*Department of Mechanical Engineering, National Institute of Technology, Warangal, Telangana State, 506004, India*

(**Received** 12 September 2025, **Revised** 12 December 2025, **Accepted** 15 December 2025)
*Corresponding author: harikumar@nitw.ac.in
DOI: 10.5875/czmm9z84

Abstract: Coverage path planning (CPP) is essential in mobile robot tasks such as cleaning and exploration, where paths must balance area coverage, path length, number of turns, and energy consumption. This work presents a complete framework, from multi-objective CPP using a modified NSGA-II to real-world deployment on a GoPiGo3 differential-drive robot. Pareto-optimal paths are generated to address conflicting objectives, while an adaptive Kalman filter and PD controllers ensure accurate localization and control. IMU noise variances are derived from power spectral density analysis, and odometry noise is modeled based on slip detection. PD gains are auto-tuned using a two-stage Ziegler-Nichols method followed by L-BFGS-B optimization. The full system is implemented on a Raspberry Pi 3B+, running at 10 Hz with cycle times around 30 milliseconds and under 5 MB RAM usage in Python. Ablation studies (full system vs. odometry-only; full system vs. no PD) highlight the impact of each module. Kalman filter covariance evolution shows increasing confidence over time. Real-world experiments conducted in indoor kitchen and living room layouts demonstrate 100% coverage, 15–30% energy savings compared to a hybrid genetic algorithm, and 20–30% better coverage than spiral-STC. The results validate the framework's ability to translate optimized plans into robust physical execution.

Keywords: Coverage Path Planning; Gopigo3 Mobile Robot; Modified-NSGA-II**;** Multi-Objective Optimization; Sensor Fusion.

Introduction

Path planning in robotics has been widely studied for enabling robots to move from a start to a goal position while avoiding collisions against obstacles [1]. Within this domain, two important categories are distinguished: shortest path planning (SPP), which focuses on optimizing distance or time to reach a target [2], and coverage path planning (CPP), which aims to generate paths that ensure complete coverage of a workspace [3]. CPP is vital for applications where thorough surface exploration is required, including floor cleaning [4], agricultural field operations [5], planetary exploration [6], landmine detection [7], and underwater inspection [8].

CPP inherently involves multiple and often conflicting objectives. A path must not only guarantee complete overage but also minimize travel distance, reduce the number of turns, and limit energy consumption. Multi-objective optimization has therefore become crucial to CPP research. Among the available approaches, the Non-dominated Sorting Genetic Algorithm II (NSGA-II) [9] has been widely adopted due to its efficiency in generating diverse Pareto-optimal solutions that balance competing objectives.

Although significant progress has been made in simulation studies of CPP, experimental validation on real robots remains relatively scarce. Most works in this

area still focus on algorithmic performance in simulation, with limited attention to practical issues such as calibration, localization, and execution accuracy. This leaves a persistent gap between theoretical optimization and real-world deployment. To bridge this gap, experimental studies are essential, as they account for factors like wheel slip, sensor noise, and obstacle configurations that cannot be fully captured in simulation.

The main contribution of this paper is the experimental validation of a multi objective evolutionary coverage path planning approach. The previously proposed modified NSGA II algorithm [10] is implemented on a physical differential drive robot, GoPiGo3, and evaluated in real indoor environments. The results show that Pareto optimal solutions generated in simulation can be executed reliably on hardware. In addition to the planning algorithm, a modified Kalman filter with absolute sensor updates is developed, along with proportional derivative controllers for orientation and centering correction. This integrated framework enables robust localization and accurate trajectory tracking, effectively addressing key challenges encountered during experimental deployment.

To our knowledge, this work is among the first to experimentally validate a modified NSGA II based coverage path planning framework on real robotic hardware. The proposed approach achieves complete area coverage while reducing energy consumption compared with existing methods. By integrating evolutionary optimization with reliable localization and control, the study moves coverage path planning beyond simulation toward practical deployment in applications such as domestic cleaning and autonomous inspection.

Monex Sharma received his bachelor's degree in Mechanical Engineering from Shankara Institute of Technology, Jaipur, in 2015, and his master's degree in Manufacturing Engineering from the Department of Mechanical Engineering at the National Institute of Technology, Warangal, in 2019. During 2019, he completed an internship at TAL Robotics, Tata Motors, Pune. He is pursuing a PhD in Mechanical Engineering at the National Institute of Technology, Warangal. His research interests include mobile robot path planning and optimization techniques. He works as a Robotics Software Engineer at WCB Robotics India Pvt. Ltd.

Hari Kumar Voruganti received his bachelor's degree in Mechanical Engineering from Kakatiya Institute of Technology and Science, Warangal, in 2001, and his master's degree in Mechanical Engineering from Osmania University in 2003. He earned his PhD in Mechanical Engineering from the Center for Robotics at the Indian Institute of Technology, Kanpur, in 2009. He is a Professor in the Department of Mechanical Engineering at the National Institute of Technology, Warangal. His research areas include shape and topology optimization; isogeometric analysis, finite element analysis, geometric modelling, and CAD; robotics with focus on kinematics and path planning; design thinking and product design and development; and artificial intelligence for design. He has authored 61 publications, holds 8 patents, and has led or contributed to 5 research projects.

Literature Review

Several approaches have been proposed to address the coverage path planning (CPP) problem in mobile robots. Wang et al. introduced the artificial potential field (APF) method [11], which provides fast responsiveness, adapts to random initial placements, and supports real-time planning. However, it often falls into local optima and struggles to cover regions near obstacles due to repulsive forces. Spanning tree coverage (STC) methods have also been widely applied [12, 13], guaranteeing complete coverage of cells and performing well in environments with arbitrary obstacles. Their main limitation is difficulty in handling small free spaces and a sharp increase in the number of turns as complexity grows. Greedy search algorithms such as A* and D* [14, 15] offer low computational cost and adaptability in dynamic environments, but they do not guarantee globally optimal paths and become inefficient in highly cluttered spaces.

Metaheuristic approaches have also been widely studied for CPP. Yakoubi et al. applied genetic algorithms (GA) [16], which offer strong global search capability but suffer from poor stability and uncertain convergence. Schäfle et al. proposed a hybrid GA with improved initialization strategies to accelerate convergence [17]. Particle swarm optimization (PSO) [18] is easier to implement than GA and requires fewer parameters, yet it is prone to premature convergence. Ant colony optimization (ACO) [19] benefits from distributed computation and avoids early convergence, but performs slowly in the initial phase and scales poorly in large environments. While these heuristic and evolutionary approaches are effective in simulation, their experimental validation on physical robots remains limited, especially when multiple objectives are considered simultaneously.

Probabilistic approaches such as RRT and RRT* have also been investigated [20, 21], enabling robots to explore complex environments but often producing suboptimal dead-end paths due to randomness. Adhipatiunus et al. improved coverage by incorporating entropy into a waypoint heuristic using Octomap [22]. Kamalova et al. applied multi-objective grey wolf optimization (GWO) [23], but their results showed repeated revisits and difficulties in handling non-convex obstacles.

Experimental validation of CPP remains relatively scarce. Guruprasad et al. tested STC and extended STC using Corobot robots in Webots simulation [24], while

Schäfle et al. used a hybrid GA with a differential-drive robot for floor cleaning [25]. These studies moved toward real-world deployment but were restricted to simplified environments and did not evaluate multi-objective optimization. Moreover, localization and control remain persistent challenges in CPP experiments. Traditional Kalman filtering has been widely used to reduce odometry drift and sensor noise [26], and feedback controllers help stabilize motion. However, these methods are rarely integrated with advanced optimization-based CPP frameworks. For CPP in particular, errors in localization directly reduce coverage efficiency, while unstable control increases unnecessary turns and energy use. This underscores the need to integrate robust localization and control with multi-objective planning.

In summary, although a wide range of CPP algorithms has been developed, most remain confined to simulation or focus on single-objective criteria. Multi-objective approaches such as NSGA-II [9] are better suited to balance conflicting goals of path length, number of turns, energy consumption, and coverage area. In our earlier work, we proposed a modified NSGA-II for CPP using a grid-based environment representation, formulating the task as a discrete optimization problem [10]. The present study extends that contribution by providing experimental validation on a physical GoPiGo3 robot, supported by a modified Kalman filter with absolute sensor updates and proportional–derivative (PD) controllers for orientation and centering correction. By addressing both path optimization and execution reliability, this integration bridges the gap between simulation-based research and practical robotic deployment. The remainder of this paper is organized as follows. Section III describes the environment setup. Section IV details the implementation of path planning on the robot. Section V presents the experimental results and discussion, and Section VI concludes the work and future scope.

Environment Setup

Environment setup in robotics involves preparing the hardware and software, as well as configuring the test environment to ensure the robot performs tasks effectively and reliably. The hardware setup includes the robot, sensors, actuators, and other accessories required for the CPP task. The software setup involves installing the necessary frameworks, libraries, and control algorithms to integrate with the hardware. The test environment is designed to mimic real-world conditions and consists of a physical workspace with defined layouts, obstacles, and boundaries. This setup is critical for validating robotic systems under practical constraints and ensures accurate, repeatable testing for CPP.

Hardware Description

The GoPiGo3 robot employs a differential-drive configuration with two active rear wheels and a spherical passive front wheel, enabling full rotation about its axis and providing high maneuverability for path planning tasks. Its compact size, programmability, and low cost make it a suitable platform for this research, where path planning and coverage experiments are validated on real hardware. Built around a Raspberry Pi 3 Model B+, the GoPiGo3 is lightweight and equipped with motorized wheels driven by gear-boxed DC motors and quadrature Hall-effect encoders for odometry. Additional sensing includes a pair of front-mounted ultrasonic (HC-SR04) distance sensors, positioned at the left and right chassis edges for orientation and centering correction, and an MPU-6050 inertial measurement unit (IMU) sensor that provides linear acceleration and angular rate information for motion estimation. Together with the wheel encoders, these sensors enable both relative and absolute localization within the grid-based environment. Robot control is implemented using the EasyGoPiGo3 Python library, which facilitates motor actuation, sensor integration, and data acquisition. Designed for both education and research, the GoPiGo3 provides a versatile platform for investigating robotic tasks such as path planning, localization, obstacle avoidance, and automation. Figure 1 illustrates the main components of the GoPiGo3 robot.

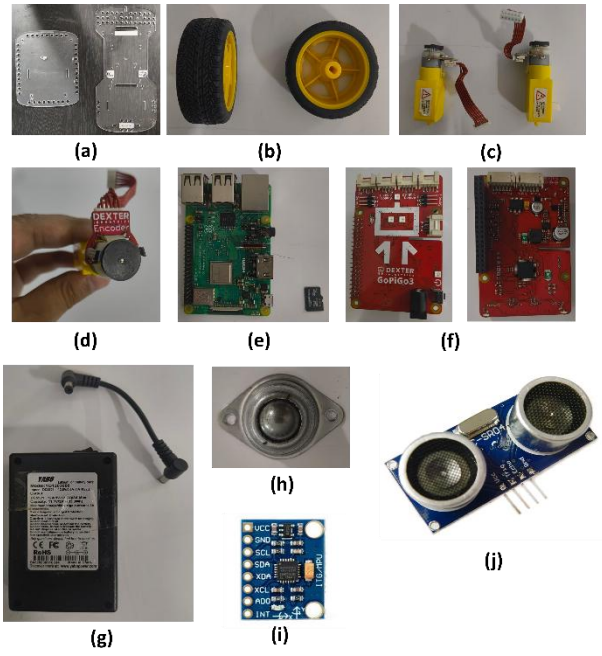


Figure 1. Components of GoPiGo3 robot, (a) chassis and canopy of acrylic material, (b) wheels of the robot, (c) gear-boxed brushless DC motors, (d) quadrature hall effect encoders, (e) Raspberry Pi 3 model B+ with 32 GB memory card, (f) top and bottom view of GoPigo3 control board, (g) rechargeable 8x AA alkaline batteries with connector, (h) caster wheel, (i) IMU MPU-6050 sensor, (j) ultrasonic sensors (HC-SR04).

Grid-Based Representation of Environment

In this study, the map of the environment is assumed to be prebuilt and provided as input. The environment for path planning is discretized into a grid-based model. For CPP validation, a rectangular area is defined by its length L and breadth B, representing the coverage region. The environment is divided into grid cells based on the maximum dimension of the GoPiGo3 robot, denoted l (the longer side of the robot), such that each cell corresponds to the space occupied by the robot. This ensures systematic traversal, with the robot occupying one cell at a time.

To support both simulation and experimental testing, environments were initially created using Coohom 3D interior design software. Coohom allows precise modeling of indoor spaces, which can then be physically reproduced in the laboratory with matching dimensions. This approach ensured consistency between simulated and real-world experiments, enhancing the reliability of the validation.

The discretization results in a grid layout with the number of cells along the length ($n_L = \frac{L}{l}$) and breadth ($n_B = \frac{B}{l}$). This approach simplifies path planning while ensuring complete and efficient coverage of the area. The layout of the environment and its grid-based discretization are illustrated in Figure 2, providing a visual representation of the defined dimensions and grid structure.

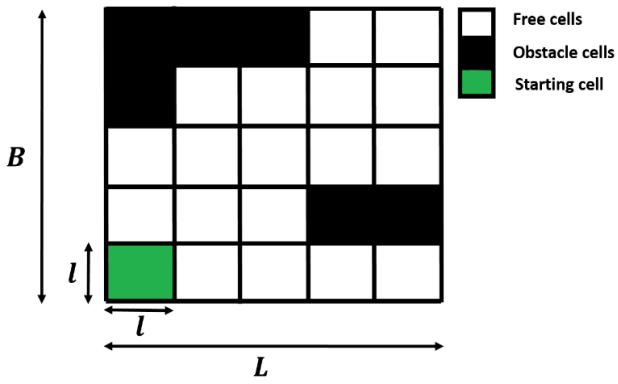


Figure 2. Pictorial representation of grid-based model.

Each cell, C is either completely occupied by an obstacle, C_o or completely free of an obstacle, C_f. The free cells and obstacle cells are represented by white and black colors in the grid map respectively, as shown in Figure 2. The robot is permitted to move in, C_f. C_f cells that the robot has visited are known as visited cells C_v, whereas C_f cells that the robot has not visited are known as unvisited cells C_u. The starting point of the robot for a given environment is denoted by P_s which is represented by green color. When a robot passes through a cell, it is considered that the cell is completely occupied by the robot.

Implementation Of Path Planning on the Robot

In our previous work [10], a modified version of the NSGA-II algorithm was proposed for the CPP of a mobile robot. The algorithm was adapted for a grid-based environment to optimize coverage-specific objectives, primarily minimizing path length and reducing the number of turns. While the algorithm was previously validated in simulation, the current study focuses on its experimental validation using the GoPiGo3 mobile robot in a controlled environment.

Modified NSGA-II method

The modified NSGA-II approach is used as a multi-objective optimization method in this work. Two objectives are considered: the total distance traveled by the mobile robot and the total number of turns executed during coverage.

The total distance f_1 is computed using the Manhattan distance, shown in Eq. (1). This measure adds the absolute differences between the x and y coordinates of consecutive grid points, giving the path length the robot travels along the grid.

$$\mathrm{f_1} = \sum_{i=1}^{N_{gc}-1} (|x_{i+1} - x_i| + |y_{i+1} - y_i|). \qquad (1)$$

where N_{gc} denotes the number of waypoints visited by the robot. The robot's position at the i^{th}waypoint is given by $p_i(x_i, y_i)$.

The robot's turns (T) and U-turns (U) are computed from its previous, current, and next positions: p_i, p_{i-1} and p_{i+1}. Since the robot moves only in four directions, it can make either a right or left 90-degree turn, or a 180-degree U-turn.

A 180-degree turn requires more energy than a 90-degree turn. So, the effective number of turns, f_2, is computed as:

$$\mathrm{f_2} = \mathrm{T} + \mathrm{w} \times \mathrm{U} \qquad (2)$$

where w = 1.5, based on the assumption that a U-turn requires about 50 percent more energy than a 90-degree turn.

The goal is to minimize the two objective functions, $f_1$and f_2, while ensuring that the robot achieves 100 percent coverage of the free area. The multi-objective optimization problem is defined as follows:

$$\text{Minimize } f = [f_1\ f_2] \quad (3)$$

Subject to:

p_i in $C_v \in C_f$,

$C_u = \Phi$.

For example, for the path in Figure 3, $f_1 = 24$ and $f_2 = 7 + 1.5 \times 1 = 8.5$. The objective values for the MOO problem are therefore $f = [24, 8.5]$. The comprehensive study of the modified NSGA-II is explained in [10].

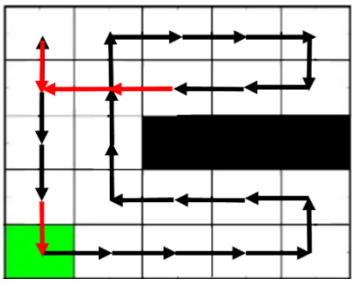


Figure 3. Example of a mobile robot CPP result using the modified NSGA-II method.

Path translation

In this paper, the experimental validation of the proposed modified NSGA-II method using the GoPiGo3 mobile robot is presented and compare its performance with HGA and spiral-STC methods. From the Pareto-optimal solutions generated by the proposed modified NSGA-II algorithm, a specific path was selected based on high-level criteria prioritizing minimum energy consumption, which can be represented as a sequence of waypoints or path segments. Each waypoint is typically defined by coordinates (x, y) or by relative movement instructions such as "move 10 cm forward" or "turn 90^0." These waypoints are then converted into commands that the GoPiGo3 robot can understand, such as encoder counts with desired or speed and turn instructions.

Path execution

Path execution refers to the process by which the GoPiGo3 robot physically follows the planned path generated by the modified NSGA-II algorithm within the grid-based environment. This step translates the algorithmic solution into real-world motion, enabling experimental validation of the proposed CPP method.

The planned path is represented as a sequence of waypoints or relative movement instructions, which are converted into motor commands to control the robot's linear and angular motion accurately.

- **Forward Movement:** The robot calculates the required encoder counts to traverse a specified distance, with both wheels moving in the same direction. Encoder feedback ensures precise travel, while velocity is regulated to prevent slippage or overshooting.

- **Turning:** To achieve a desired rotation (e.g., 90° or 180°), encoder counts determine the wheel movements necessary for the turn. Using differential drive mechanics, the robot can rotate in place by moving one wheel forward and the other backward, allowing accurate in-place turning.

By executing these commands, the robot follows the planned path reliably, allowing evaluation of trajectory accuracy, coverage completeness, and energy efficiency during experimental trials.

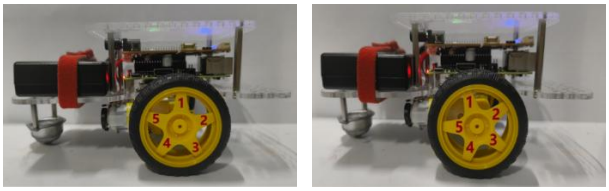
Calibration of encoders

Calibration is a crucial step for ensuring accurate sensor performance on mobile robots. In this study, the GoPiGo3 robot, equipped with quadrature Hall-effect encoders on each motor, is used for odometry. Ideally, each encoder should register 360 counts per full wheel rotation. However, factors such as mechanical slippage, electrical noise, and signal interference can cause discrepancies, leading to undercounting or overcounting. To correct this, an encoder calibration factor (CF) is calculated and applied to align the measured counts with the theoretical value. As shown in Table 1, encoder counts were measured over five repeated runs. The observed average was 371 counts per rotation instead of the expected 360, resulting in an overcount of about 3%. A calibration factor of 0.97 was therefore applied to ensure accurate odometry, which is critical for reliable coverage path planning.

Table 1. Encoder calibration results for five repeated runs

Run	Measured counts	Expected counts	Error (%)	Calibration factor
1	372	360	+3.33	0.967
2	370	360	+2.78	0.973
3	369	360	+2.50	0.975
4	371	360	+3.06	0.970
5	373	360	+3.61	0.964

Figure 4 illustrates the wheel position before rotation (a) and after the encoder records 360 counts (b). While 360 counts should represent a full revolution, the wheel does not complete the turn because of overcounting. This error highlights the effect of uncalibrated encoder measurements, where even small per-rotation discrepancies can accumulate into noticeable odometry drift during extended operation.



(a) (b)

Figure 4. Position of the wheel before rotation (a) and after rotation (b) without calibration.

The average CF, calculated as:

$$CF = \frac{Expected\ counts}{Measured\ counts} \quad (4)$$

The calibration factor is applied to the encoder readings to compensate for counting errors. After applying the CF as defined in Eq. [4], the encoder counts were corrected to match the expected 360 counts, ensuring that one full wheel rotation was accurately represented, as illustrated in Figure 5. Using the wheel diameter d = 66.5 mm, the linear distance per rotation C was calculated as:

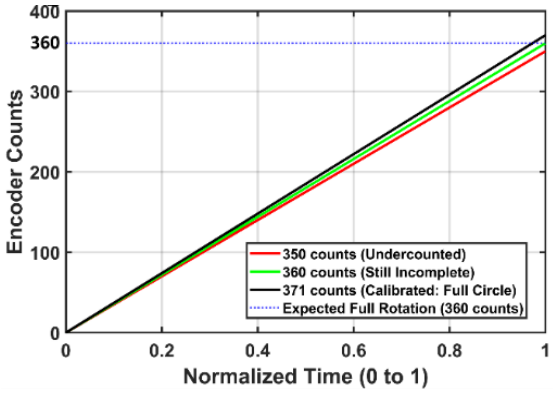
$$C = \pi \times d \quad (5)$$

This adjustment ensured that the calibrated encoder readings accurately matched the robot's actual movement, minimizing deviations and enhancing the precision of the robot's motion.

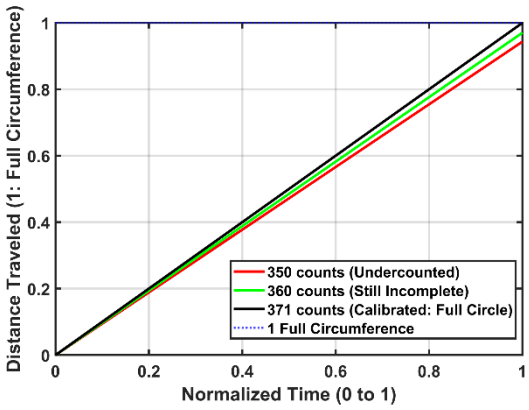
(a) (b)

Figure 5. Position of the wheel before rotation (a) and after rotation (b) with calibration.

Figure 6 shows how the encoder was calibrated to reflect a full wheel rotation. In theory, one turn should give 360 counts, but in practice small errors such as wheel slip caused the reading to drift. As a result, the robot was not measuring distance correctly. In Figure 6(a), the encoder reached 360 counts before the wheel had actually completed a full turn. To correct this, a calibration factor was applied so that a full rotation matched 371 counts, as shown in Figure 6(b). With this adjustment, the encoder output now tracks the real wheel motion, giving more reliable distance and better motion control.



(a)



(b)

Figure 6. Calibration of encoder counts to obtained one complete circumference: (a) shows more than 360 counts are required due to noise, (b) represents one complete circumference with 371 encoder counts.

Localization and Control Integration

To execute the coverage paths from the modified NSGA-II algorithm accurately, several localization and control strategies were evaluated. The process started with pure odometry, then added Kalman filtering using two relative sensors: odometry and an inertial measurement unit [26]. A modified Kalman filter was then applied, combining these relative sensors with an absolute sensor that was used only before and after turns. Finally, a proportional-derivative (PD) controller was applied for position correction to improve orientation and centering [27].

Strategy 1: Pose Estimation with Odometry

The GoPiGo3 encoders provide wheel rotation counts, which can be converted into linear and angular

displacements. Given the wheel diameter d and grid cell width l, the forward and rotational increments at each time step are expressed as:

$$\Delta s = \frac{d}{4}(\Delta\phi_R + \Delta\phi_L) \quad (6)$$

$$\Delta\theta = \frac{d}{2l}(\Delta\phi_R - \Delta\phi_L) \quad (7)$$

where $\Delta\phi_R$ and $\Delta\phi_L$ are the right and left wheel rotations. The pose (x, y, θ) is updated by:

$$x_{k+1} = x_k + \Delta s\, cos(\theta_k + \frac{\Delta\theta}{2}) \quad (8)$$

$$y_{k+1} = y_k + \Delta s\, sin(\theta_k + \frac{\Delta\theta}{2}) \quad (9)$$

This method is computationally simple but accumulates drift due to wheel slip and encoder errors. Hence it results inaccurate localization result.

Strategy 2: Kalman Filter with Relative Sensors

The Kalman filter operates in three main steps: prediction of the next state from current state, measurement using available sensor data, and update by combining the prediction with the measurement to reduce uncertainty using weighted sum method based on Kalman gain.

In this research, a seven-state linear Kalman filter is formulated to estimate the localization of mobile robot. The state vector is:

$$\mathbf{X_k} = \begin{bmatrix} x_k \\ v_{x,k} \\ a_{x,k} \\ y_k \\ v_{y,k} \\ a_{y,k} \\ \theta_k \end{bmatrix} \quad (10)$$

where x_k and y_k denote the position of the robot in the world frame (m), $v_{x,k}$ and $v_{y,k}$ are the translational velocities (m/s), and $a_{x,k}$ and $a_{y,k}$ are the linear accelerations (m/s^2) along the world X and Y axes, respectively. The final element, θ_k, represents the robot heading in radians, measured counter - clockwise from the world X axis to the robot's forward axis. The sampling interval is denoted by Δt.

The Kalman filter uses a measurement matrix built from the processed sensor data. In this work, the measurement matrix is defined as:

$$\boldsymbol{Z_k} = \begin{bmatrix} v_{x,k}^{odo} & v_{y,k}^{odo} & a_{x,k}^{imu} & a_{y,k}^{imu} & \theta_k^{imu} \end{bmatrix}^{\boldsymbol{T}} \quad (11)$$

The measurement vector includes five terms that capture the robot's motion state at each time step. The odometry-based velocities $v_{x,k}^{odo}$ and $v_{y,k}^{odo}$ describe how fast the robot is moving along the world X and Y directions, with the first capturing forward motion and the second indicating any sideways drift caused by slip or uneven contact. The IMU accelerations $a_{x,k}^{imu}$ and $a_{y,k}^{imu}$ provide short-term motion data points after preprocessing, helping the filter adjust velocity estimates during changes in speed or direction. The final term, θ_k^{imu}, is the robot's heading angle (yaw angle) from the IMU, which stabilizes orientation estimates and keeps the motion aligned with the world frame.

Strategy 3: Modified Kalman filter with PD control

The prediction step and relative-sensor updates with odometry and the IMU remain unchanged. The modification is the addition of a sparse absolute update from the ultrasonic sensor, applied only in two cases: (i) immediately before a turn and (ii) immediately after completing the turn. At these points, the robot is expected to be at the centre of the active grid cell. If it is not, the ultrasonic measurement is used to correct (x, y) to the known cell centre. Achieving this correction requires two actions: orientation correction and centering correction.

Correction 1: Orientation correction

Before executing a turn, the robot heading is aligned with the global grid axes using two front-mounted ultrasonic sensors positioned at the left and right edges of the chassis. Let the measured distances from the left and right sensors to the wall be $d_L(t)$ and $d_R(t)$ at time t, respectively, and let w denote the lateral separation between the sensors. If the robot is perfectly parallel to the wall, the two readings are equal. The orientation error is estimated as:

$$e_\theta(t) \approx arctan\left(\frac{d_R(t) - d_L(t)}{w}\right) \quad (12)$$

A (PD) controller is applied for orientation correction because the task is to drive the angular error to zero quickly when the robot faces a wall. The proportional term $K_p^o e_\theta(t)$ generates a corrective response proportional to the heading error, while the derivative term $K_d^o \dot{e}_\theta(t)$ damps the motion and suppresses oscillation at time t. This combination ensures fast convergence and stability. A PD control law is applied to generate an angular velocity command as given below:

$$\omega_{cmd}(t) = K_p^o e_\theta(t) + K_d^o \dot{e}_\theta(t) \quad (13)$$

This angular command is converted into differential wheel velocities for in-place rotation:

$$v_L = -\frac{\omega_{cmd} l}{2},$$

$$v_R = +\frac{\omega_{cmd} l}{2} \quad (14)$$

where l is the track width (distance between wheels) which is equal to dimension of cell.

The wheel linear speed v (m/s) is converted into motor command in degrees per second (dps) used by the GoPiGo mobile robot as $dps = v.\frac{180}{\pi r}$. The correction motor command is stopped when $|d_R - d_L| < \varepsilon_d$ (very small threshold).

Correction 2: Centering correction

The robot workspace is modeled as a rectangular environment of dimensions L × B, bounded by walls on all sides. The area is divided into square cells of side length l. Consider (x_0, y_0) denote the coordinates of the bottom-left corner of the environment. If the robot is in cell (i, j), with i = 0, 1, …, $\frac{L}{l-1}$ and j = 0, 1, …, $\frac{B}{l-1}$, the centre of that cell is

$$x_c(i) = x_0 + \left(i - \frac{1}{2}\right) l$$
$$y_c(j) = y_0 + \left(j - \frac{1}{2}\right) l \quad (15)$$

The robot is aligned with the global axes using orientation correction, the expected distance from the robot centre to the facing wall is:

$$d^{ref} = \begin{cases} x_0 + L - x_c(i) + \delta, & if\ robot\ is\ facing + X \\ x_c(i) - x_0 - \delta, & if\ robot\ is\ facing - X \\ y_0 + B - y_c(i) + \delta, & if\ robot\ is\ facing + Y \\ y_c(i) - y_0 - \delta, & if\ robot\ is\ facing - Y \end{cases} \quad (16)$$

where δ is the sensor offset (location of sensor from the center of the robot along the +X of robot frame).

Once the expected distance to the facing wall d^{ref} is computed for the active cell (i, j) (accounting for sensor offset δ), a PD controller is used to drive the robot to the desired position along the facing axis. The instantaneous ultrasonic sensor measurement is considered as d^{US}(t).

Then the centering error is defined as

$$e_c(t) = d^{US}(t) - d^{ref} \quad (17)$$

A PD law generates a forward velocity command:

$$v_{cmd}(t) = K_p^c e_c(t) + K_d^c \dot{e_c}(t) \quad (18)$$

where K_p^c and K_d^c are the centering proportional and derivative gains. The same linear velocity is applied to both wheels for pure translation, and converted to motor units (degrees per second).

$$v_L(t) = v_R(t) = v_{cmd}(t)$$
$$dps = v.\frac{180}{\pi r}$$

The correction motor command is stopped when $|e_c(t)| < \varepsilon_c$ (very small threshold).

Orientation correction and centering correction are applied only at discrete events, specifically immediately before and immediately after each turning maneuverer. This ensures that the robot is properly aligned with the global axes and positioned at the centre of the active grid cell prior to executing a turn, and that it is recentred again once the turn is complete. Continuous application of these corrections is avoided, as it would unnecessarily increase computation cost without improving performance during straight-line motion. During straight traversal between turns, localization relies solely on the Kalman filter with relative sensors (odometry and IMU), which provides sufficient accuracy while keeping the estimation process efficient.

Parameter Settings

The process and measurement noise covariances were fixed prior to all experiments. The process noise covariance Q captures uncertainty in the robot motion model, including unmodeled wheel slip, actuator nonlinearities, and small errors in acceleration integration. It was defined as:

$$Q = diag\left(\sigma_x^2, \sigma_{vx}^2, \sigma_{ax}^2, \sigma_y^2, \sigma_{vy}^2, \sigma_{ay}^2, \sigma_x^2, \sigma_{vx}^2, \sigma_\theta^2\right) \quad (19)$$

where:

$$\sigma_x^2 = \sigma_y^2 = 1.0 \times 10^{-4}\ m^2$$
$$\sigma_{vx}^2 = \sigma_{vy}^2 = 2.5 \times 10^{-4}\ (m/s)^2$$
$$\sigma_{ax}^2 = \sigma_{ay}^2 = 1 \times 10^{-3}\ (m/s^2)^2$$
$$\sigma_\theta^2 = 2.5 \times 10^{-4}\ rad^2$$

The measurement noise covariance R models uncertainty from the datasheets of the odometry (quadrature Hall effect encoders) and IMU sensor (MPU-6050).

At 10 Hz update, velocity uncertainty ≈ ±0.04 m/s for typical quadrature Hall effect encoders during no slippage condition. Hence variance of velocity of odometry sensor for ideal and pure rotation condition is given as:

$$\sigma_{vx,odo}^2 = \sigma_{vy,odo}^2 \approx 0.04^2 = 1.6 \times 10^{-3}(m/s)^2.$$

Wheel slippage is detected by comparing the displacement estimated from odometry with the

displacement measured by the ultrasonic sensors. The disagreement between the two sensors is determined as

$$\Delta_R = \Delta S_R - \Delta d, \quad \Delta_L = \Delta S_L - \Delta d \tag{20}$$

Where $\Delta S_R = \frac{d}{2}(\Delta\phi_R)$ and $\Delta S_L = \frac{d}{2}(\Delta\phi_L)$, are the incremental displacements of the right and left wheels, and Δd is the change in distance measured by the ultrasonic sensors between time steps t and t-1 as given below.

$$\Delta d = \frac{d_R(t)+d_L(t)}{2} - \frac{d_R(t-1)+d_L(t-1)}{2} \tag{21}$$

Slippage is detected when odometry indicates motion while the ultrasonic readings remain nearly constant, or when the disagreement values Δ_R *and* Δ_L become large. In such cases, the confidence in odometry is reduced by increasing its measurement noise. The updated measurement noise covariance along the slipping direction (y-axis) is given by:

$$\sigma^2_{\text{vy,odo}} = \sigma^2_{\text{vy,odo}} + (\Delta_R{}^2 + \Delta_L{}^2) \tag{22}$$

This adaptive update raises $\sigma^2_{\text{vy,odo}}$ when the sensors disagree, allowing the Kalman filter to reduce reliance on odometry during wheel slip.

The noise parameters for the IMU-6050 sensor are taken from the manufacturer's datasheet [28]. For IMU MPU-6050 sensor, accelerometer noise density ≈ $400\ \mu g/\sqrt{Hz}$. At 10 Hz effective bandwidth ≈ $0.004\ g = 0.04\ m/s^2$. While gyroscope noise density ≈ $0.005\ °/s\sqrt{Hz}$. At $10\ Hz \approx 0.016\ dps = 2.8 \times 10^{-4}\ rad$. Over 0.1 s sample interval, heading uncertainty ≈ 2.8×10^{-5} rad. Hence the variances for IMU sensor of accelerometer and gyroscope are given below:

$$\sigma^2_{ax,imu} = \sigma^2_{ax,imu} = 0.04^2 = 1.6 \times 10^{-3}(m/s^2)^2.$$
$$\sigma^2_{\theta,imu} = 10^{-4}\ rad^2.$$

To obtain better IMU noise parameters for the Kalman filter, power spectral density (PSD) [29] analysis is used to quantify noise energy and extract variance for each axis.

Let $x[n]$ be the IMU sample sequence (one axis) with sampling frequency f_s and number of samples N. The zero-mean signal and its discrete Fourier transform in terms of frequency are computed:

$$x_{zm}(n) = x(n) - \frac{1}{N}\sum_{i=1}^{N} x(i) \tag{23}$$
$$X(k) = FFT(x_{zm}(n)) \tag{24}$$

The FFT provides *both* positive and negative frequencies. But real signals of IMU data are symmetric, so the one-sided PSD is estimated below:

$$PSD[k] = \frac{|X(k)|}{N f_s} \tag{25}$$

where f_s is the sampling frequency. Noise variance for the axis is the integrated PSD:

$$\sigma^2 = \sum_{k=0}^{N/2} PSD(k)\Delta f \tag{26}$$
where $\Delta f = \frac{f_s}{N}$

Hence, the variance is computed separately for each IMU axis: $\sigma^2_{ax,imu}, \sigma^2_{ay,imu}, and\ \sigma^2_{\theta,imu}$.
The Figure 7 shows the IMU noise recorded during a one-minute test. This noise must be accounted for in the measurement model to prevent error buildup during long-duration operation.

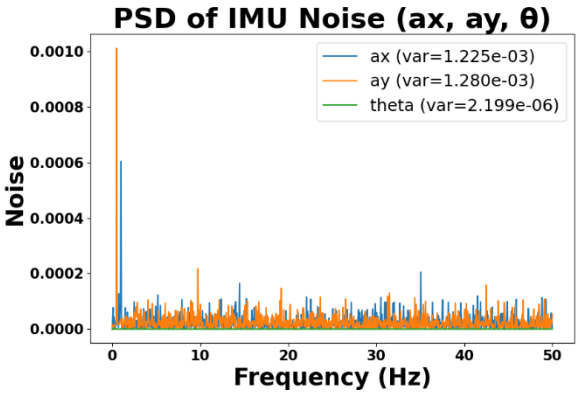


Figure 7. PSD Analysis of IMU Acceleration and Gyroscope Noise

Although the noise in Figure 7 appears small, integrating it over time leads to significant drift, which can cause large errors during extended operation.

Automated tuning algorithms for PD gains

In this paper, the PD gains were refined using a two-stage tuning process. To obtain the initial controller gains, Ziegler-Nichols closed-loop tests were carried out separately for the orientation and centering loops [30]. In each test, the derivative term was disabled by setting $K_d = 0$, and a small step correction was applied to excite the system. The proportional gain K_p was then increased gradually until the robot response exhibited sustained oscillations with nearly constant amplitude. The proportional gain at this point was recorded as the ultimate gain K_u, and the period of these oscillations was measured as T_u. Using the standard Ziegler-Nichols

tuning rules for PD control, the initial gains were computed as:

$$K_p = 0.8\,K_u \text{ and } K_d = 0.1\,K_u\,T_u$$

This procedure provided starting values (K_p^o, K_d^o) for the orientation controller and (K_p^c, K_d^c) for the centering controller. These values served only as initial predictions and were later refined through a gradient-based optimization.

To evaluate each candidate set of PD gains, a performance cost was defined using the Integral of Time-weighted Absolute Error (ITAE),

$$J = \int_0^T t\,|e(t)|dt \tag{27}$$

where $e(t)$ denotes the instantaneous control error at time t. For the orientation controller, $e(t)$ is the difference between the desired heading and the measured heading, while for the centering controller it represents the lateral offset from the lane or cell centre. The time-weighted term penalizes errors that persist later in the response, making ITAE well suited for assessing settling behavior. For each gain set, ITAE was computed separately for orientation and centering corrections, and the values were averaged over five repeated trials to obtain a consistent and noise-robust measure.

In the second stage, a gradient-based optimization was adopted to refine the PD gains. The optimization started from the Ziegler-Nichols values and searched for (K_p, K_d) that minimized the cost. A gradient-based solver Limited-memory Broyden Fletcher Goldfarb Shanno with Bounds (L-BFGS-B) was employed [31], and the gradients of the cost with respect to the controller gains were approximated numerically using finite differences. This required evaluating the partial derivatives

$$\frac{\partial J}{\partial K_p} \text{ and } \frac{\partial J}{\partial K_d},$$

Each update was restricted to safe bounds to prevent overly aggressive controller behavior. The solver iteratively evaluated the cost, estimated the gradients, and adjusted the gains until further reduction to reach negligible variation.

Calculation of Energy Consumption for Gopigo3 Mobile Robot

The energy consumption of the GoPiGo3 mobile robot can be evaluated in two ways: through theoretical modeling based on component specifications and through experimental estimation from the robot's operating parameters. In this study, both approaches are considered. The theoretical model provides a baseline, while the experimental estimates are taken as the reported values.

Theoretical Estimation

The power demand of the robot arises from two main contributions: (i) mechanical requirements for wheel rotation and robot translation, and (ii) static consumption of onboard electronics such as the Raspberry Pi (3 W) and motor driver (0.2 W).

For each type of motion (starting, stopping, straight-line motion, and turning), the energy can be derived using classical mechanics:

Acceleration/deceleration:

$$v^2 = u^2 + 2as$$
$$F = ma, \quad \tau = \frac{F.d}{2}$$
$$P_{rot} = \tau.\omega, \quad P_{motor} = \frac{P_{rot}}{\eta}$$
$$E_{start} = P_{motor}.t \tag{28}$$

For each type of motion (starting, stopping, straight-line motion, and turning), the energy can be derived using classical mechanics:

Acceleration/deceleration:

$$v^2 = u^2 + 2as$$
$$F = ma, \quad \tau = \frac{F.d}{2}$$
$$P_{rot} = \tau.\omega, \quad P_{motor} = \frac{P_{rot}}{\eta}$$
$$E_{start} = P_{motor}.t \tag{29}$$

where u and v are initial and final velocities, a is acceleration, s is displacement, F is force, τ is torque, d is wheel diameter, ω is angular velocity, and η is motor efficiency.

Constant velocity motion:

$$t_{cv} = \frac{s}{v}$$
$$E_{cv} = P_{motor}.t_{cv} \tag{30}$$

Turning motion:

$$\omega = \frac{v_t}{r_t}, \quad F_c = \frac{mv_t^2}{r_t}, \quad \tau = \frac{F_c.d}{2}$$

$$P_{rot} = \tau.\omega, \quad P_{motor} = \frac{P_{rot}}{\eta}$$

$$E_{turn} = P_{motor}.t \tag{31}$$

These theoretical calculations provide approximate values and highlight the relative costs of different motion types. A summary of theoretical estimates is presented in Table 2.

Table 2. Energy consumption for each type of motion by GoPiGo3 robot

Motion type	*Total power consumption (W)*	*Time required (seconds)*	*Total energy consumption (J)*
Start	3.20176	2.5	8.0044
Stop	3.20176	2.5	8.0044
Straight	3.20176	$12.5 \times s_2$	$40.002\times s_2$
Turn	3.200628	2.12	6.786

Energy Consumption Estimation

Energy consumption of the GoPiGo3 robot was estimated during experiments based on the operational parameters of the mobile robot. At each run, instantaneous power was calculated as

$$P(t) = V(t) \times I(t) \tag{32}$$

where V(t) and I(t) represent the voltage and current at time t. The total energy for a run was then calculated as

$$E = \sum_{t=1}^{N} P(t).\Delta t \tag{33}$$

with Δt representing the sampling interval of the onboard measurement. Mean and standard deviation of energy consumption were computed from five repeated trials for each environment and planning method.

While Table 2 presents theoretical energy estimation of GoPiGo mobile robot, the actual energy consumption is determined from operational parameters of the robot. These experimental measurements complement the theory by capturing the combined effects of real-world operation and unavoidable inefficiencies.

Computational Analysis on Raspberry Pi

To verify that the proposed system can run on lightweight embedded hardware, the CPU load, and memory usage were profiled on a Raspberry Pi 3B+ while the full localization and control loop was running. The estimator, path follower, and sensor interfaces operated at 10 Hz.

The combined memory footprint of the Kalman filter (state, covariance, noise matrices), odometry variables, two ultrasonic sensor readings, and the PD controller remains small, using roughly 0.8-1.0 KB of float storage in total. Even when auxiliary buffers are included, the footprint stays well below 2 KB, which is negligible for the Raspberry Pi. The full control-cycle cost is small. The Kalman filter needs about 5k floating-point operations per update, which takes roughly 1.5 ms in Python on a Raspberry Pi. Odometry and PD control add only a few hundred operations and remain below 1 ms. The main delay comes from sensor input and output: ultrasonic and I²C reads can add upto 30 ms depending on the sensor and bus load. In practice, a complete cycle typically falls upto 30 ms, which is well within the limits for a 10 Hz control loop on the Raspberry Pi.

The full program is observed to be used approximately 5 MB of RAM on a Raspberry Pi 3B+. This includes the Python interpreter, sensor drivers, I/O libraries, thread stacks, and the runtime environment. The Python process itself dominates memory usage, not the float variables. So, in practice the process usually settles near 5 MB.

Results and Discussion

The experimental workflow began with the design of two indoor layouts in Coohom, an online 3D modeling and visualization tool: a kitchen (Figure 8) and a living room (Figure 9).

(a) (b)

Figure 8. Real environment of kitchen room created based on Coohom software.

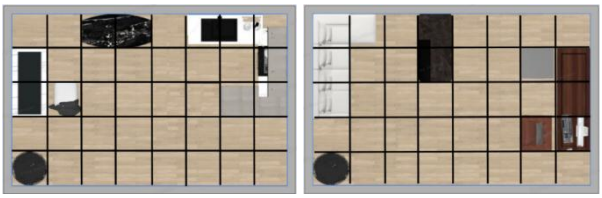
(a) (b)

Figure 9. Real environment of living room created based on Coohom software.

Each layout included fixed obstacles such as walls, furniture, and passageways to reflect realistic indoor spaces. The models were exported in both top and isometric views and used for simulation as well as for physical recreation in the laboratory. This ensured that

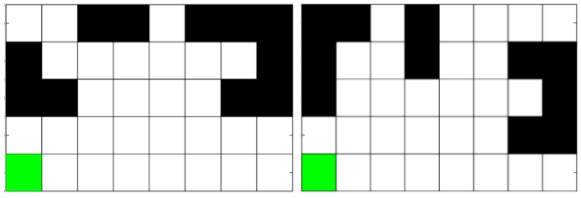
the robot was tested in environments that closely resembled practical indoor conditions while maintaining consistency between simulated and experimental trials.

The layouts designed in Coohom were then converted into grid-based representations for use in both simulation and real-world execution. The laboratory recreation of the environments was discretized into uniform grid cells aligned to the robot's footprint, producing the real-environment grid shown in Figure 10. A corresponding grid for simulation was generated to test the planned paths virtually before execution, as illustrated in Figure 11. These two grids ensured consistency between simulation and physical testing, allowing direct comparison of planned and executed results.



(a) (b)

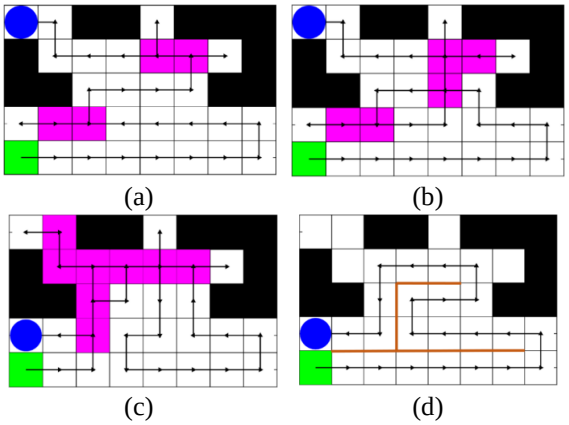
Figure 10. Real environments divided into cells using grid-based method.



(a) (b)

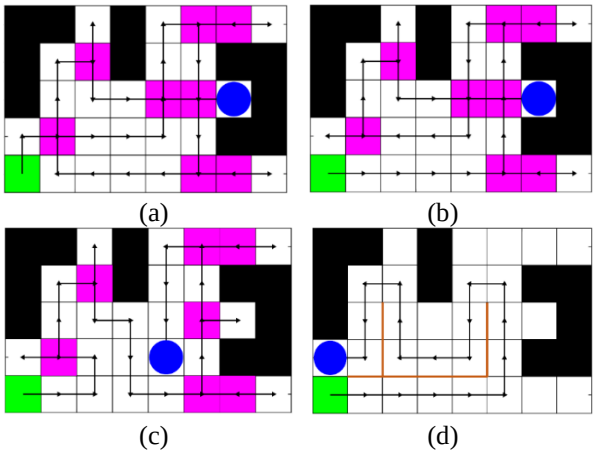
Figure 11. Simulation environment in grid-based model.

Pareto fronts were generated in simulation with the modified NSGA-II algorithm, and representative solutions were chosen for hardware execution based on least energy consumption (Figures 12 and 13). In Figure12, subfigures (a) and (b) show two such Pareto solutions that both minimize energy use but differ in how they balance path length and number of turns. Both still achieve full coverage. Subfigure (c) shows the Hybrid Genetic Algorithm (HGA) path, which also covers the area but requires more turns and a longer path. Subfigure (d) shows the spiral-STC path, which is shorter overall but leaves boundary regions uncovered. Figure 13 presents the same comparison for the second environment, with subfigures (a) and (b) showing two Pareto solutions, (c) the HGA result, and (d) the spiral-STC result.



(a) (b) (c) (d)

Figure 12. Path generated in the first simulation environment (kitchen) using proposed method (a, and b), HGA (c), and spiral-STC methods (d).



(a) (b) (c) (d)

Figure 13. Path generated in the second simulation environment (living room) using proposed method (a, and b), HGA (c), and spiral-STC methods (d).

Selected planned paths were converted to motor commands by translating waypoint displacements and rotations into encoder counts, using the calibrated encoder factor and wheel circumference described in Section 4.4 and illustrated in Figures 4-5. The IMU provided linear acceleration data, while the ultrasonic sensors were used for both orientation alignment and centering corrections before and after turns. These inputs, together with encoder-based odometry, were fused through a modified Kalman filter, and a PD controller was applied to refine orientation and centering during corrections. The experiments were conducted with the parameter values given in Section 4.4.4. For each run, the GoPiGo3 robot was placed at a repeatable start pose (Figure 14).

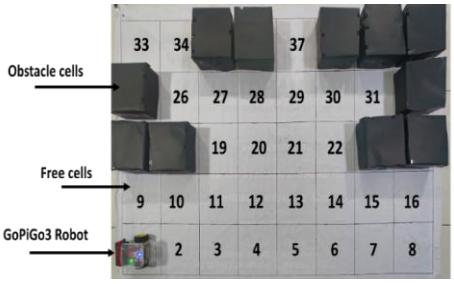


Figure 14. Pictorial representation of the GoPiGo3 robot before the experiment.

For each path planning and environment combination, five independent runs were carried out. Instantaneous power was calculated using Eq. [32], and the total energy for each run was obtained by numerical integration as shown in Eq. [33]. The theoretical per motion estimates are provided in Table 2 for reference, while the experimental values obtained during robot operation are presented in Table 3 and Table 4 for the first and second environments, respectively.

Table 3. Energy consumption and coverage area for the proposed and two existing methods in the first environment.

Methodology	*Distance traveled (cells)*	*Number of turns*		*Energy (J)*	*Coverage area in %*
		90^0	180^0		
modified NSGA-II (solution 1)	32	10	3	1438.5 ± 48.5	100
modified NSGA-II (solution 2)	34	9	3	1527.4 ± 52.6	100
HGA	37	18	3	1874.6 ± 53.3	100
Spiral-STC	23	8	0	1231.5 ± 43.7	82.75

Table 4. Energy consumption and coverage area for the proposed and two existing methods in the second environment.

Methodology	*Distance traveled (cells)*	*Number of turns*		*Energy (J)*	*Coverage area in %*
		90^0	180^0		
modified NSGA-II (solution 1)	36	9	3	1388.3 ± 47.3	100
modified NSGA-II (solution 2)	36	8	4	1432.2 ± 48.2	100
HGA	35	11	5	1784.6 ± 52.6	100
Spiral-STC	19	8	0	1032.4 ± 41.8	68.96

Convergence and diversity analysis

Hypervolume measures how much objective space is dominated by the Pareto front, using a reference point that represents the worst acceptable values of all objectives. For each solution, a rectangular area is formed between its two objective values and this worst point, and these areas are summed to get the hypervolume of that generation. When this value grows over iterations, it shows that the algorithm is finding better and more diverse solutions. Tracking this trend across generations helps demonstrate the strength and steady improvement of the proposed method.

In the initial generation, the hypervolume of the objective values begins very close to the worst-case reference point. As illustrated in Figure 15, the hypervolume increases rapidly and demonstrates substantial convergence within the first 70 generations. Beyond this point, the improvement saturates, with only minor incremental gains observed. Moreover, Figure 15 shows that the final multi-objective optimal value approaches the ideal solution, with the hypervolume nearing 1.

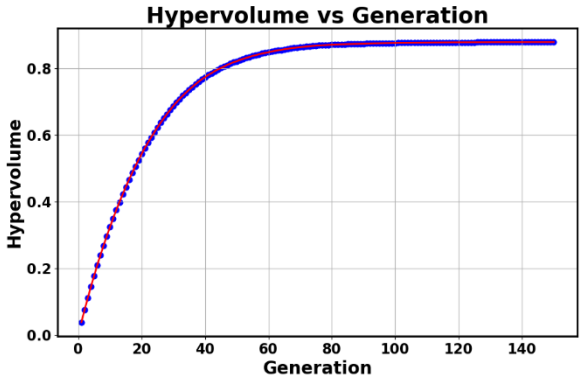


Figure 15. Hypervolume growth during modified NSGA-II evolution

Localization and Control Results completed till here

All coverage experiments were executed using the localization and control framework described in Section 4.4. A modified Kalman filter was used, combining odometry and IMU data for continuous state estimation during straight-line traversal. Sparse absolute corrections from the ultrasonic sensors were applied immediately before and after turns, with a PD controller refining orientation and centering. This approach ensured that the robot remained cantered within each grid cell and aligned with the global axes. By comparing cases with and without the Kalman filter or PD correction, the results highlight how much each component contributes to path tracking quality and overall energy use.

The ablation is examined in two parts to show how each component influences the robot's performance:

i. Full system vs. odometry only
ii. Full system vs. no PD correction

Full system vs. odometry only

The robot's position is estimated using the odometry model Eq. [6-7] and the Kalman filter. For comparison, the robot moves from the initial point $P_1(0,0)$ to the target point $P_2(50,50)$cm. Figure 16 shows the resulting (x, y) trajectories. The Kalman filter estimate is shown in red, while the odometry-only estimate appears in blue. The deviation in the odometry-based trajectory is mainly due to wheel slippage.

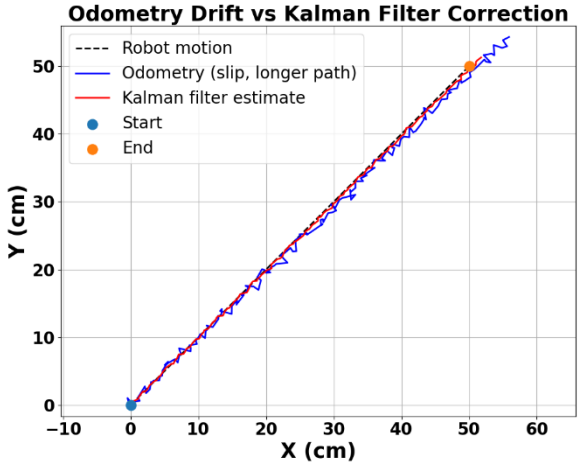


Figure 16. Comparison of odometry slip and filtered position

The RMS path deviation between the odometry-based position and the Kalman filter estimate is computed as follows.

$$RMS_{odo-kf} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} ||odom_i - kf_i||^2} \quad (34)$$

For this experiment, the RMS path deviation between the odometry estimate and the Kalman filter output is $RMS_{odo-kf} = 2.08cm$. This indicates that, over the full trajectory, odometry deviates from the Kalman filter-corrected path by an average of 2.08 cm due to wheel slip. The RMS_{odo-kf} value depends the amount of wheel slippage during the robot traverse.

Full system vs. no PD correction

To isolate the effect of PD correction during turns, a single-turn ablation is performed. For a standard 90° turn the GoPiGo3 robot executed $K = 10$ trials per condition:

i. Kalman + PD (full) and
ii. Kalman only (PD disabled).

Orientation correction is the measurement of the heading error as the absolute difference between the robot's final orientation and the desired target as explained in Eq. [12]. Centering error is computed as the deviation from the expected grid centre after the turn as given in Eq. [16]. For each metric, the mean and standard deviation over all trials is reported for both conditions. A paired statistical comparison is then used to determine whether disabling the PD controller leads to a significant degradation in turning accuracy.

The quantitative effect of the corrections is summarized in Table 5. The PD controllers used for orientation and centering corrections consistently converged within about 2 s in all trials. Orientation errors, measured as the difference between the left and right ultrasonic sensors, were reduced from initial offsets of 3-4 cm to less than 0.5 cm, corresponding to a final angular error below 2 degrees. Centering corrections reduced distance error relative to the wall reference from 2-3 cm to less than 1 cm.

Table 5. Orientation and centering correction performance (average over five trials)

Correction type	*Metric (unit)*	*Without PD controller*	*With PD controller*
Orientation	Left-right ultrasonic sensor difference (cm)	3-4 cm	< 0.5 cm
	Alignment error (deg)	$5-7^0$	$< 2^0$
Centering	Distance error to wall (cm)	2-3 cm	< 1 cm
	Settling time (s)	NA	~ 2 s

Kalman filter covariance evolution

This section examines how the Kalman filter behaves over extended operation by tracking the evolution of the position covariance terms P_{xx} and P_{yy} as shown in Figure 17.

It can be observed from the Figure 17 that the covariance gradually decreases over time, which indicates growing confidence in the estimated robot position and closer agreement with the actual trajectory.

It can be seen in Figure 17 that the zoomed part of the figure shows an increase in covariance during straight motion and while turning. The covariance drops immediately after each turn because the PD controller corrects the robot's motion.

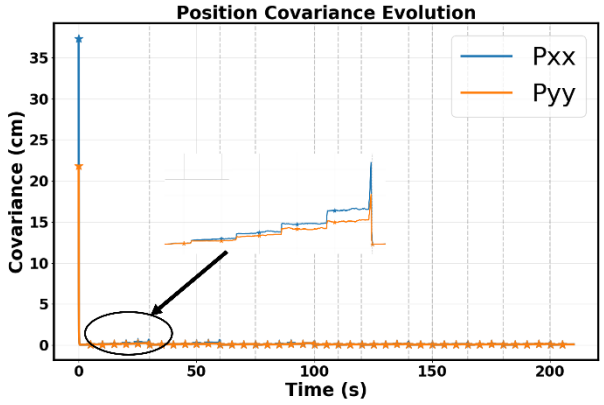
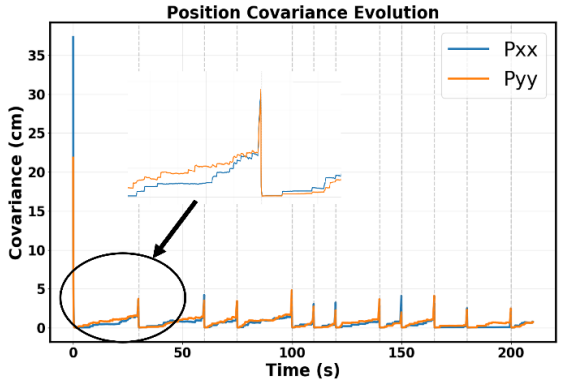
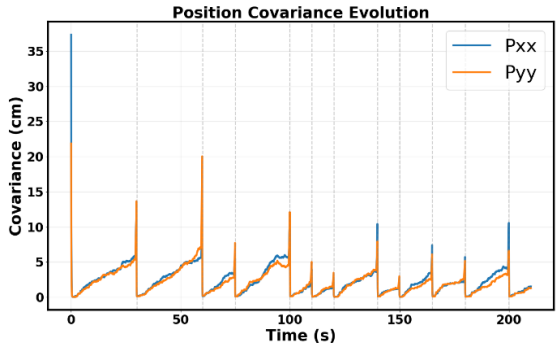


Figure 17 Evolution of position covariance over time

In addition, Monte Carlo simulations were performed using multiple initial conditions to evaluate consistency and to verify that the filter converges reliably under different starting uncertainties. To study the stability of the filter, the IMU and odometry noise levels were increased separately, and the resulting covariance traces were compared as illustrated in Figure 18.



(a)



(b)

Figure 18. Comparison of position covariance under elevated IMU noise (a) and odometry noise (b)

Figure 18(a) shows that increasing the IMU noise causes the covariance to rise in some cases but remain unchanged in others, mainly because the large IMU variance reduces the filter's sensitivity to wheel-slip effects.

When wheel slippage occurs, the odometry readings become inaccurate and the covariance increases as the motion estimate drifts. If there is no slippage, the odometry alone is sufficient to keep the estimate close to the true trajectory. In contrast, increasing the odometry noise consistently leads to larger long-term covariance, since even small errors of IMU data accumulate during extended motion as shown in Figure 18(b).

The analysis shows how the estimator responds to different uncertainty conditions, including changes in sensor noise and variation in the initial covariance as shown in Figure 19. These results help confirm whether the estimator remains stable during long-duration runs and whether it can maintain bounded uncertainty in realistic operating conditions.

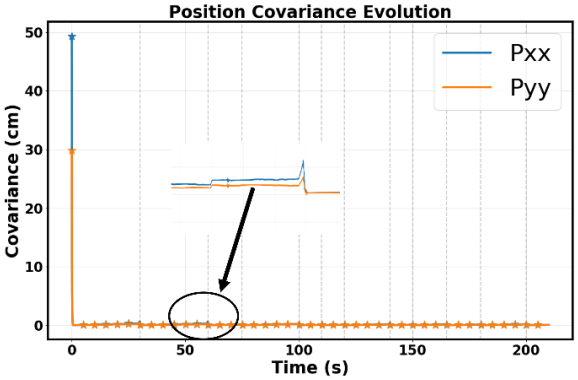


Figure 19. Effect of initial covariance on filter convergence

The initial covariance values were set higher than in the previous cases, allowing the Kalman filter to naturally reduce them as it gains confidence during the early stages of motion. A slight rise in covariance appears before each turn, but the ultrasonic sensor provides an absolute correction and the PD controller aligns the robot during the turn, which brings the covariance back down once the turning is completed.

Coverage performance

Both selected modified NSGA-II solutions achieved full coverage in the two test environments (kitchen and living room), as shown in Table 3 and Table 4 confirmed by the executed cell-visitation sequences in Figures 20 and 21.

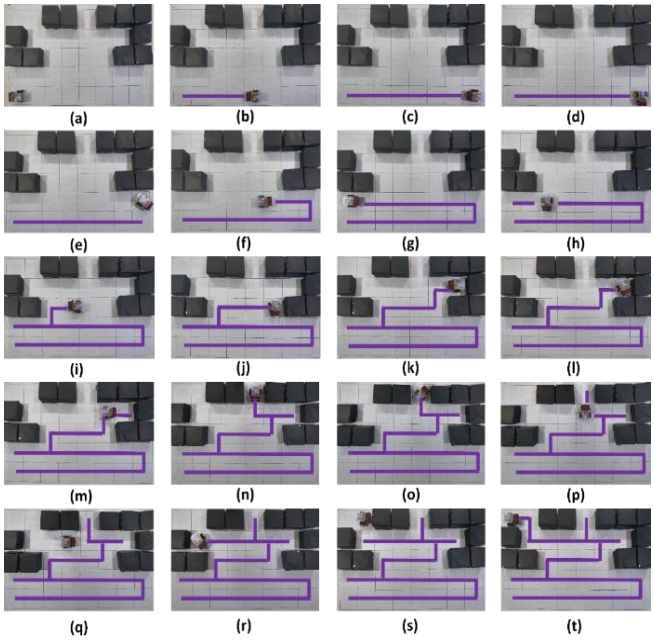


Figure 20. Sequences of the CPP of the robot motion in the first environment using GoPiGo3 robot.

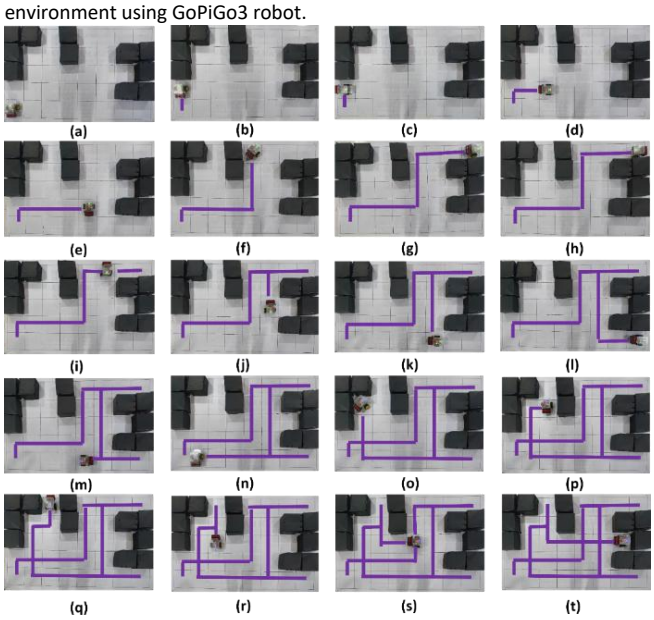


Figure 21. Sequences of CPP of the robot motion in the second environment using GoPiGo3 robot.

By contrast, the spiral-STC baseline produced incomplete coverage in both layouts, with coverage values of 82.75 percent and 68.96 percent for Environments 1 and 2 respectively (Table 2 and Figures 20-21). Visualization results of the executed paths shows that the uncovered areas generated by spiral-STC are clustered near boundaries and narrow passages in the Coohom layouts (Figures 8-9). The HGA method reached full coverage but consumes large distance travels and number of turns, as given in Table 2.

The energy results in Table 2 were obtained directly from the experiments. In Environment 1, the two modified NSGA-II solutions consumed 1438.5 J and 1527.4 J, compared to 1874.6 J for HGA and 1231.5 J for spiral-STC. In Environment 2, the modified NSGA-II solutions used 1388.3 J and 1432.2 J, while HGA consumed 1784.6 J and spiral-STC 1032.4 J. The lower energy values for spiral-STC are mainly due to its shorter travel distance, but that efficiency came at the cost of poor coverage, as seen in Figures 12-13 and Table 3 and Table 4. By contrast, the modified NSGA-II solutions offered a more practical trade-off: full coverage with significantly lower energy use than HGA. Depending on the environment and the selected Pareto solution, the energy reduction relative to HGA was in the range of about 15 to 30 percent.

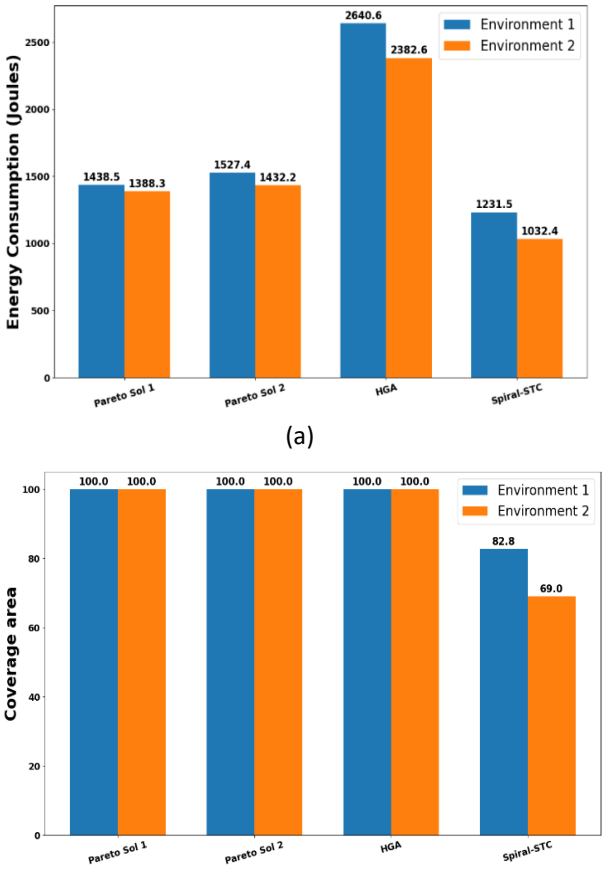


Figure 22. Experimental results in kitchen (environment 1) and living-room (environment 2): (a) Energy consumption comparison, (b) Coverage comparison.

To complement the tabulated results in Table 2, bar charts were created to provide a clearer visual comparison of the three planners across both environments (Figures 22(a) and 22(b)). Figure 22(a) summarizes the energy consumption, while Figure 22(b) shows the coverage performance. In both environments, the Pareto-optimal solutions achieved lower energy consumption than the HGA, with reductions of 15-30 percent. At the same time, the

Pareto solutions maintained full coverage in all runs, whereas Spiral-STC showed noticeable coverage loss, particularly in the second environment. These plots highlight the consistency of the experimental validation and make it easier to compare the relative performance of each method.

Path length and turn-count analysis

Path length and turn counts help explain the observed energy differences. In Environment 1, the two modified NSGA-II solutions traversed 32 and 34 cells with 10 and 9 ninety-degree turns plus three 180-degree pivots, while HGA required 37 cells and 18 ninety-degree turns (Table 2). The higher turn count for HGA caused more frequent acceleration and deceleration, leading to higher instantaneous power peaks. In Environment 2, both modified NSGA-II solutions traversed 36 cells with fewer or comparable turns relative to HGA, and the energy values followed the same trend (Table 2 and Figures 20-21). With the modified Kalman filter, overlays of the planned and executed trajectories (Figures 12-13 and 20-21) show that the robot stayed close to the intended paths. Encoder calibration (Figures 4-5) and sensor fusion were key to this, ensuring that wheel rotations matched the planned distances and turns. Overall, jointly optimizing distance and turns in the multi-objective framework delivered measurable energy savings when Pareto-optimal solutions were run on the physical robot.

Robustness Evaluation under Sensor Disturbances

The robustness of the localization and control system was tested by adding controlled disturbances to the IMU and wheel encoders. The evaluation included IMU bias and simulated wheel slip to reflect realistic sensing degradation. The analysis focuses on how these disturbances influence navigation accuracy and system stability, especially as the robot approaches the point where it initiates a turn and the noise level rises. All tests were carried out during straight-line motion up to the moment just before turning.

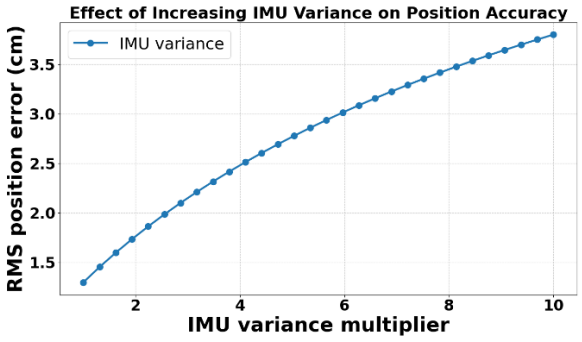


Figure 23. RMS Position Error Under Increasing IMU Variance.

With the proposed IMU variance setting, the RMS position error of 1.3 cm is generated. As the variance of the IMU acceleration along the y-axis is increased, the Kalman filter relies less on the IMU and more on the odometry estimate as shown in Figure 23. Because odometry accumulates slip during forward movement, the position error rises steadily and reaches 3.8 cm when the IMU variance multiplier is set to 10. This shows that accurate IMU measurements are important for correcting wheel slip and preserving localization accuracy.

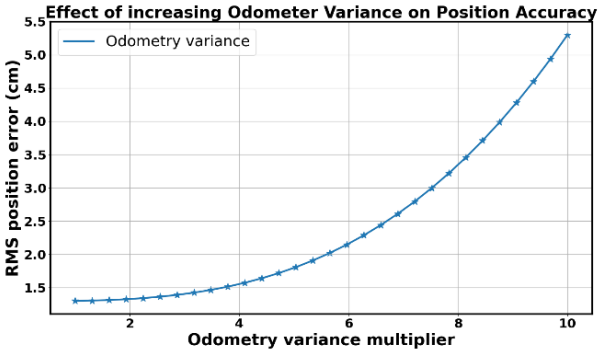


Figure 24. Impact of increasing odometry variance on Kalman filter position accuracy

As shown in Figure 24, increasing the odometry variance multiplier reduces the weight of odometry in the Kalman update, so the IMU dominates the position estimate. Small IMU biases and noise then accumulate and create significant drift. Minor yaw errors from the gyroscope can bend the estimated path even during straight-line motion. With odometry down-weighted, the filter also loses reliable displacement information, allowing the position to drift even when the robot is stationary. As this accumulation grows over time, the RMS position error rises more sharply at higher multipliers, showing that both the error and its growth rate increase as odometry uncertainty grows.

A balanced variance setting for the IMU and odometry is essential for stable and accurate localization. When the IMU is assigned a high variance value, the filter reduces its trust in acceleration and yaw measurements, which can expose slip-related errors in the odometry. When odometry is given a high variance value, the filter leans heavily on the IMU, and its bias and noise begin to accumulate and cause drift. The results in Figures 23 and 24 show that each sensor offsets the other's limitations: the IMU corrects slip, and odometry constrains long-term drift. Using realistic variance values for both sensors allows the Kalman filter

to balance their information and maintain consistent performance over longer runs.

Conclusion and Future Work

This study presented the first experimental validation of our earlier proposed work using modified NSGA-II algorithm for multi-objective coverage path planning using the GoPiGo3 mobile robot. The approach combines odometry and IMU inputs through a Kalman filter, with sparse ultrasonic corrections applied before and after turns, and PD controllers to maintain orientation and centering. This integration ensured that the robot stayed close to its intended path, bridging the gap between our earlier simulation work and practical real-world execution. Across both case study environments, a kitchen and a living room, the modified NSGA-II consistently achieved 100% coverage while optimizing for energy consumption, distance, and number of turns. In the kitchen environment, the selected Pareto-optimal paths traversed 32-34 cells with 9-10 ninety-degree turns and three 180-degree pivots, consuming about 1438-1527 J. By contrast, the Hybrid Genetic Algorithm required 37 cells with 18 turns, consuming roughly 1875 J. The spiral-STC path achieved shorter travel distance but left 20-30% of boundary regions uncovered, undermining coverage completeness. In the living room, the modified NSGA-II again balanced distance and turns effectively, traversing 36 cells with fewer or comparable turns relative to HGA, and with energy consumption reduced by 15-25%. Spiral-STC showed similar limitations, with incomplete coverage despite lower travel distance. The modified NSGA-II eliminates gaps in coverage that spiral-STC leaves behind, while producing smoother, more regular paths than HGA. The algorithm reduces energy use by 15-30% and turn counts by up to 50% compared with HGA, while maintaining complete coverage. Together, the experiments confirm that the modified NSGA-II, when paired with the proposed localization and control framework, delivers practical gains in efficiency and reliability.

Future work can evaluate the modified NSGA-II algorithm in larger and more complex indoor environments, including spaces with moving obstacles. Testing on uneven outdoor terrains like fields or construction sites will help assess real-world performance. Integrating richer sensor fusion, such as LiDAR or vision-based SLAM, can improve localization. Developing advanced controllers may enhance stability and reduce drift during longer or faster operations. These improvements will support more autonomous, reliable, and scalable robot deployments.

References

[1] Ju. Ming-Yi, Y. J. Chen, and W. C. Jiang, "Implementation of odometry with EKF in hector SLAM methods," *International Journal of Automation and Smart Technology*, vol. 8(1), 2018. doi: 10.5875/ausmt.v8i1.1558

[2] C. H. Kuo, "Uav parcel delivery system with deep reinforcement learning based collision avoidance and route optimization," *International Journal of Automation and Smart Technology*, vol. 14(1), pp. 8-p18, 2024. doi: 10.5875/p25fht10

[3] S. C. Tan, M. M. Rosmiwati, and M. R. Arshad, "A comprehensive review of coverage path planning in robotics using classical and heuristic algorithms," *IEEE Access*, vol. 9, pp. 119310-119342, 2021. doi: 10.1109/ACCESS.2021.3108177

[4] S. Kuo-Lan, Y. L. Liao, S. P. Lin, and S. F. Lin, "An interactive auto-recharging system for mobile robots," *International Journal of Automation and Smart Technology*, vol. 4(1), pp. 43-53, 2014. doi: 10.5875/ausmt.v4i1.197

[5] C. S. Luís, N. S. Filipe, E. J. P. Solteiro, A. Valente, P. Costa, and S. Magalhães, "Path planning for ground robots in agriculture: A short review," in *IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, Ponta Delgada, Portugal, May 19, 2020, pp. 61-66. doi: 10.1109/ICARSC49921.2020.9096177

[6] W. Yu, W. Shaobo, and H. Xinting, "Cooperative path planning of UAVs & UGVs for a persistent surveillance task in urban environments," *IEEE Internet of Things Journal*, vol. 8(6), pp. 4906-4919, 2020. doi: 10.1109/JIOT.2020.3030240

[7] B. Ahmed, K. Kumar, N. Kumar, N. Thakur, B. Alzahrani, and A. Almansour, "Unmanned aerial vehicle (UAV) path planning for area segmentation in intelligent landmine detection systems," *Sensors*, vol. 23(16), pp. 7264, 2023. doi: 10.3390/s23167264

[8] T. S. Hsu, and T. C. Wang, "An improvement stereo vision images processing for object distance measurement," *International Journal of Automation and Smart Technology*, vol. 5(2), pp. 85-90, 2015. doi: 10.5875/ausmt.v5i2.460

[9] K. Deb, A. Pratap, S. Agarwal, and T. A. M. T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on*

Evolutionary Computation, vol. 6(2), pp. 182-197, 2002.
doi: 10.1109/4235.996017

[10] M. Sharma and H. K. Voruganti, "Multi-objective optimization approach for coverage path planning of mobile robot," *Robotica*, vol. 42(7), pp. 2125-2149, 2024.
doi: 10.1017/S0263574724000377

[11] W. Zikai, W. Zhao, J. Zhang, N. Yang, P. Wang, J. Tang, J. Zhang, and L. Shi, "APF-CPP: An artificial potential field based multi-robot online coverage path planning approach," *IEEE Robotics and Automation Letters*, vol. 9(11), pp. 9199-9206, 2024.
doi: 10.1109/LRA.2024.3432351

[12] G. Yoav and E. Rimon, "Spanning-tree based coverage of continuous areas by a mobile robot," *Annals of Mathematics and Artificial Intelligence*, vol. 31, pp. 77-98, 2001.
doi: 10.1023/a:1016610507833

[13] K. R. Guruprasad and T. D. Ranjitha, "ST-CTC: A spanning tree-based competitive and truly complete coverage algorithm for mobile robots," in *Proceedings of the 2015 Conference on Advances in Robotics*, vol. 43, pp. 1-6, 2015.
doi: 10.1145/2783449.2783492

[14] C. Zengyu, S. Li, Y. Gan, R. Zhang, and Q. Zhang, "Research on complete coverage path planning algorithms based on A* algorithms," *The Open Cybernetics & Systemics Journal*, vol. 8(1), pp. 418-426, 2014.
doi: 10.2174/1874110X01408010418

[15] D. Marija, S. Horvatić, and I. Petrović, "Complete coverage D* algorithm for path planning of a floor-cleaning mobile robot," *IFAC Proceedings*, vol. 44(1), pp. 5950-5955, 2011.
doi: 10.3182/20110828-6-IT-1002.03400

[16] A. Y. Mohamed and M. L. Tayeb, "The path planning of cleaner robot for coverage region using genetic algorithms," *Journal of Innovation in Digital Ecosystems*, vol. 3(1), pp. 37-43, 2016.
doi: 10.1016/j.jides.2016.05.004

[17] T. R. Schäfle, M. Mitschke, and N. Uchiyama, "Generation of optimal coverage paths for mobile robots using hybrid genetic algorithm," *Journal of Robotics and Mechatronics*, vol. 33(1), pp. 11-23, 2021.
doi: 10.20965/jrm.2021.p0011

[18] C. Hongyue, J. Yi, and F. Yang, "Chaos particle swarm optimization enhancement algorithm for UAV safe path planning," *Applied Sciences*, vol. 12(18), pp. 8977, 2022.
doi: 10.3390/app12188977

[19] M. Karthikeyan, A. Shrivastava, and P. Rajagopal, "An optimal coverage path plan for an autonomous vehicle based on polygon decomposition and ant colony optimisation," *Ocean Engineering*, vol. 252, pp. 111101, 2022.
doi: 10.1016/j.oceaneng.2022.111101

[20] H. I. Lin, and C. S. Yang, "2D-span resampling of bi-RRT in dynamic path planning." *International Journal of Automation and Smart Technology*, vol. 5(1), pp. 39-48, 2015.
doi: 10.5875/ausmt.v5i1.837

[21] Q. Gao, Y. Qingni, S. Yu, and X. Liangyao, "Path planning algorithm of robot arm based on improved RRT* and BP neural network algorithm," *Journal of King Saud University - Computer and Information Sciences*, vol. 35(8), pp. 101650, 2023.
doi: 10.1016/j.jksuci.2023.101650

[22] M. Adhipatiunus, R. R. A. Setiadji, R. M. Widyaputra, and W. Adiprawita, "Implementation of informed rapidly-exploring random tree* on a pre-mapped Octomap generated by real time appearance based map," in *AIP Conference Proceedings*, vol. 2366(1), pp. 060012, 2021.
doi: 10.1063/5.0060189

[23] A. Kamalova, D. K. Ki, and G. L. Suk, "Waypoint mobile robot exploration based on biologically inspired algorithms," *IEEE Access*, vol. 8, pp. 190342-190355, 2020.
doi: 10.1109/ACCESS.2020.3030963

[24] K. R. Guruprasad, "X-STC: An extended spanning tree-based coverage algorithm for mobile robots," in *Proceedings of the 2019 4th International Conference on Advances in Robotics*, Art. no. 52, pp. 1-6, 2019.
doi: 10.1145/3352593.3352662

[25] T. R. Schäfle, M. Marcel, and N. U., "Generation of optimal coverage paths for mobile robots using hybrid genetic algorithm," *Journal of Robotics and Mechatronics*, vol. 33(1), pp. 11-23, 2021.
doi: 10.20965/jrm.2021.p0011

[26] M. Brossard and S. Bonnabel, "Learning wheel odometry and IMU errors for localization," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 291-297, 2019. doi: 10.1109/ICRA.2019.8794237

[27] V. M. Hernández-Guzmán, R. Silva-Ortigoza, and C. Márquez-Sánchez, "A PD path-tracking controller plus inner velocity loops for a wheeled mobile robot," *Advanced Robotics*, vol. 29(16), pp. 1015-1029, 2015. doi: 10.1080/01691864.2015.1033459

[28] https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf

[29] Nirmal, K., and A. G. Sreejith. "Noise modeling and analysis of an IMU-based attitude sensor: Improvement of performance by filtering and sensor fusion." In *Advances in optical and mechanical technologies for telescopes and instrumentation II*, vol. 9912, pp. 2138-2147. SPIE, 2016. doi: 10.1117/12.2234255

[30] S. M. Othman, A. H. Noorhazirah Sunar, M. N. Ismail, M. S. Ayob, M. S. M. Muhamad Azmi, and M. S. M. Hashim, "Position tracking performance with fine-tuned Ziegler–Nichols PID controller for electro-hydraulic actuator in aerospace vehicle model," *Journal of Physics: Conference Series*, vol. 2107, no. 1, Art. no. 012064, 2021. doi: 10.1088/1742-6596/2107/1/012064

[31] X. Li, Y. Wang, and D. Liu, "Research on extended Kalman filter and particle filter combinational algorithm in UWB and foot-mounted IMU fusion positioning," *Mobile Information Systems*, vol. 2018, Art. no. 1587253, 2018. doi: 10.1155/2018/1587253