

Advanced Robotics Control for Automated Item Delivery

Srijon Mandal*

Department of Electrical and Electronics Engineering, Amador Valley High School, Pleasanton, California, USA

*Corresponding author: Srijon Mandal, Department of Electrical and Electronics Engineering, Amador Valley High School, Pleasanton, California, USA; E-mail: mandal.srijon@gmail.com

Citation: Mandal S (2026) Advanced Robotics Control for Automated Item Delivery. J Electr Eng Electron Technol 15: 1027

Received: 06-Dec-2024, Manuscript No. JEEET-24-154363; **Editor assigned:** 11-Dec-2024, PreQC No. JEEET-24-154363 (PQ); **Reviewed:** 25-Dec-2024, QC No. JEEET-24-154363; **Revised:** 17-Feb-2026, Manuscript No. JEEET-24-154363 (R); **Published:** 24-Feb-2026, DOI: 10.4172/2325-9838.10001027

Abstract

This paper describes the development of a low-cost robotic car designed for automated delivery of items within indoor environments. The system leverages a Raspberry Pi-based platform, integrating line detection algorithms, object detection techniques and PID control for smooth and accurate navigation. By combining hardware and software solutions, the robotic car can follow predefined paths, detect intersections and handle multi-point deliveries with real-time adaptability. The design ensures affordability, reliability and ease of deployment, making it suitable for applications in healthcare, retail and logistics. Detailed implementation, testing and validation results are provided to demonstrate the system's effectiveness.

Keywords: Microcomputer; Robotics; Raspberry Pi; Yolo; OpenCV; Image processing; Python

Introduction

Automation and robotics are transforming industries by improving efficiency and accessibility. This paper presents a cost-effective robotic car designed for automated item delivery in indoor spaces such as elderly homes, warehouses and retail stores. The system addresses the need for timely, reliable deliveries using low-cost technology [1].

The robotic car uses a Raspberry Pi, camera, motors and sensors to follow a defined path, detect objects and perform multi-point deliveries. Advanced algorithms, including YOLO for object detection and PID for path-following, ensure smooth operation. The design focuses on affordability, adaptability and reliability, making it suitable for real-world applications. This paper details the design, system setup, algorithms, testing and results [2].

Dataset

The system relies on datasets capturing the spatial layouts of delivery areas, object detection samples and various movement patterns captured over multiple iterations. These datasets are essential for training the robot to follow paths and detect obstacles [3].

Preprocessing of data

It is essential to preprocess the data to ensure that it is suitable for statistical operations:

- **Normalizing spatial data:** The regular features are normalized.
- **Anomaly removal:** Anomalies caused by environmental factors like lighting conditions are removed [4].

Materials and Methods

Process flow

- Configure the robot car with a chassis, wheels, power source and single board microcomputer.
- Setup and test the Thunderborg motor controller and Raspberry Pi 4 with Pi-Camera, LED strips and other components.
- Program the motor controller to ensure precise speed, direction and stopping for deliveries.
- Implement line-following and obstacle-detection algorithms using the camera and sensors [5].

Key features

- The robotic car can work on a predefined schedule and adapts to any indoor layout by following a specified path.
- It also can perform multi-point deliveries while providing audio alerts during each of them.
- It can work for extended periods of time.

System configuration

- Configure the robot car with a chassis, wheels, power source and single board microcomputer (Figure 1).

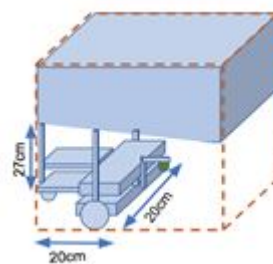


Figure 1: Schematic diagram.

- Configure the Raspberry Pi 4 with following components: a) Pi-Camera, b) 2 LED strips, c) a Thunderborg motor controller with 2 motors, d) a speaker (Figure 2) [6].

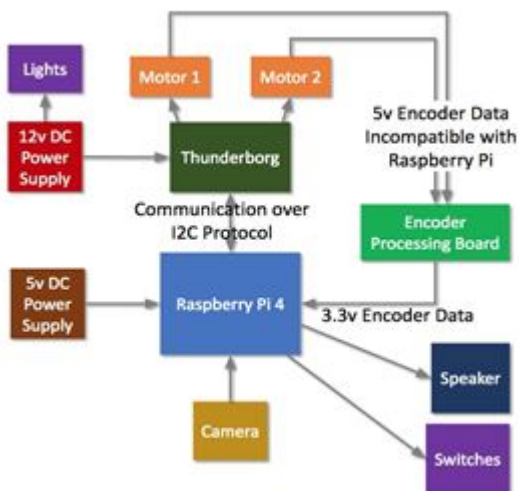


Figure 2: Hardware component design.

- Programming a motor controller: a) Setup ThunderBorg motor controller and verify if the controller is able to connect properly with the board. B) A DC motor controller manipulates the position, speed or torque of a DC-powered motor and easily reverses, so that DC motor drive current runs in the opposite direction [7].
- PID has three terms, proportional, integral and derivative. They are used to nullify the oscillation in the robot as it follows the line. It varies greatly depending on the mass and other variables.
- Next, we set battery monitoring limits to see the LED status on the controller itself.
- The show color function is used to change the color of the LED in different scenarios.
- Some crucial ones are the pin that is designated to the hall effect sensor which is activated when a magnet field is detected. There is a magnetic flap in the car.
- All the hardware settings can be configured by users.
- The encoders for both motors are initialized, whose function is to adjust to different surfaces when the motors are running.
- End program function is simply used to end and close all processes when a run is finished [8].
- Next the camera management module was created, which spots the line and detects intersections. Based on what it detects, it takes corresponding actions. a) If the line isn't found, the robot backtracks until the line is back in sight of the camera which is what the function is for. b) Additionally, when the hall effect sensor is detected, the robot stops to either drop a ball or rotate around to return to home base.
- Finally, there are physical switches on the robot itself for which there are specific functions to carry out those actions the switches perform when pressed.

Hardware interface design

Setup and configure python libraries to integrate various hardware components (Figure 3) [9].

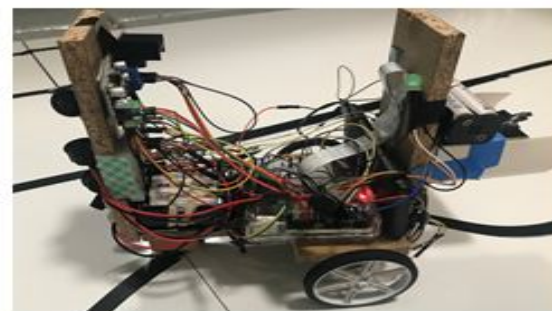


Figure 3: Internals of the robot car.

Equations

The PID control adjustment equation is defined as:

$$\text{output} = K_p \cdot \text{error} + K_i \cdot \int \text{error} \, dt + K_d \cdot \frac{(\text{error})}{dt}$$

where, K_p is the proportional gain, K_i is the integral gain and K_d is the derivative gain.

Algorithm for object detection

Develop a custom algorithm to apply the dynamic inference rules and tune object detection machine learning models and image processing techniques.

Algorithm steps:

- The speed of the left and right motors are constantly being adjusted depending on the steps for each motor. After that, the PID is updated with the values constantly so that the robot runs smoothly on the line. Finally, the speed left and right are fed to the Thunderborg motor controller.

Libraries and frameworks

The following Python libraries and frameworks were used for the implementation of the machine learning models:

- **Raspberry Pi OS:** The operating system for the Raspberry Pi, allows one to run Python and other necessary libraries in the microcomputer.
- **Motor control libraries:** For interfacing the Thunderborg motor controller with the Raspberry Pi, use libraries such as the Thunderborg Python library for motor control and status monitoring.
- **OpenCV:** It is used for Image processing tasks such as object detection and line following. OpenCV will help process camera inputs to detect lines, intersections and obstacles.
- **NumPy:** A library for numerical computing used for matrix operations and handling arrays.
- **Matplotlib:** A plotting library used for visualizing the results and performance of the models.

Results and Discussion

- Path following functionality was tested over a long period of time under different lighting conditions (Figure 4).

- The robot car started oscillating and going off track. So, we tuned the PID through trial and error to ensure the robot car delivered items smoothly.
- Voltage fluctuation caused by LED strips with one power source resulted in malfunctioning. It was resolved by maintaining 2 separate power sources (Figure 5).

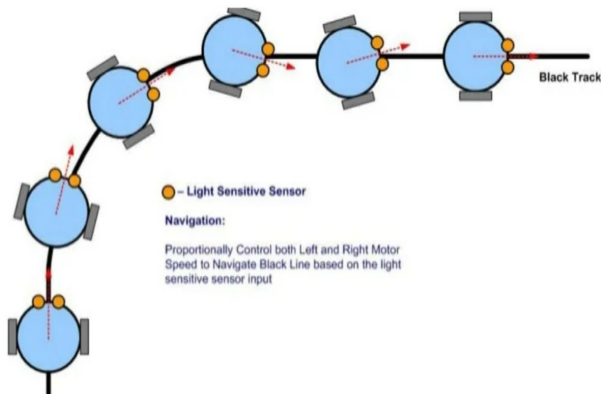


Figure 5: Robot car initial PID before tuning.

- Heatmaps and Voltage Patterns were plotted and validated (Figures 6 and 7 and Tables 1 and 2).



Figure 4: Path following process.

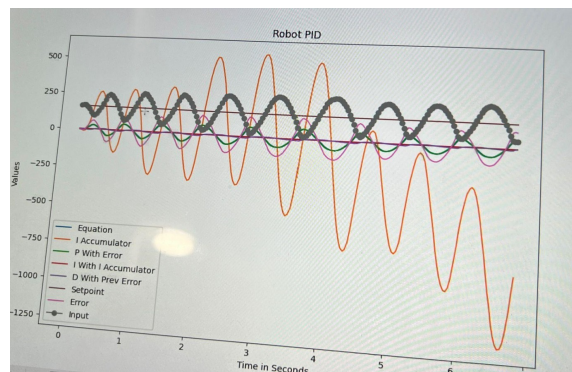


Figure 6: Robot car PID after tuning.

Scenario	Result
Given a 12 Volts DC, it must be able to follow the line	Connect to Raspberry Pi to breadboard and computer and it works
It must detect object in the range of 2-100 cm	With measure tape added obstacles and values are printed on screen/ programming logs
Find intersections and move around	It was able to detect intersections based on the PID configurations
Follow the indoor path	Device will detect "line follow" methodology and move around
deliver item	ARCAID reached its destination within time
Audio alert	Notified user at destination point

Table 1: Test result.

	Cost	Maneuverability	Safety	Speed	Stability	Battery life	Complexity	Movement
Cost		1	1	0	1	0	0.5	1
Maneuverability	0		0	0	0	0	0	0
Safety	0	1		0	0.5	0	0	0

Stability	1	1	1		1	1	1	1
Speed	1	1	0.5	0		0	0	1
Battery life	0	1	1	0	1		0.5	0.5
Complexity	0.5	1	1	0	1	0.5		0.5
Movement	0	1	1	0	0	0.5	0.5	
Total	2.5	7	5.5	0	4.5	2	2	4

Table 2: Value proposition table.

Limitations and future work

This study has some limitations like reliance on stable lighting conditions and challenges in adapting to complex layouts. Future work will integrate advanced object detection algorithms and optimize battery performance.

Conclusion

In this paper, we have successfully implemented a robotic car for automated item delivery using advanced robotics control. This solution demonstrated feasibility for real-world applications in healthcare, retail and logistics.

References

1. Joseph Z. Ben-Asher, Elon D. Rimon (2022) Time optimal trajectories for a car-like mobile robot. IEEE Transactions on Robotics.

2. Hao Yu, Gong-You Tang, Hao Su, Chun-Peng Tian, Jian Zhang (2024) Trajectory tracking control of wheeled mobile robots via

fuzzy approach. Proceedings of the 33rd Chinese Control Conference.

3. Wang Z (2017) Autonomous robotic cart for food delivery on airplane. Master of Science (MS) in Mechatronics and Robotics.

4. Li TH, Chang SJ (2003) Autonomous fuzzy parking control of a car-like mobile robot. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans 33: 451-465.

5. Demirli K, Khoshnejad M (2009) Autonomous parallel parking of a car-like mobile robot by a neuro-fuzzy sensor-based controller. Fuzzy Sets Sys 160: 2876-2891.

6. Hwang CL, Chang LJ, Yu YS (2007) Network-based fuzzy decentralized sliding-mode control for car-like mobile robots. IEEE Trans Ind Electron 54: 574-585.

7. Li H, Luo X, Skitmore M (2020) Intelligent hoisting with car-like mobile robots. J Const Eng Manag 146: 04020136.

8. Mandal S (2024) Advanced robotics control for automated item delivery. Authorea Preprints.

9. Nguyen-Tuong D, Peters J (2011) Model learning for robot control: A survey. Cogn Process 12: 319-340.