International Journal of Scientific Research and Engineering Development-– Volume 8 Issue 3, May-June 2025

Available at www.ijsred.com

RESEARCH ARTICLE OPEN ACCESS

AI POWERED SMART CAR

PRATIK FUNDE*, PRATHAM UTTEKAR*, PRATHAMESH TAWALE*
Ajeenkya D.Y Patil University, Pune
Prof.Vivek More
Asst. Professor & Project Guide, Ajeenkya DY Patil University Pune, India

----------**********----------

Abstract:

The growing accessibility of open-source hardware and embedded programming has enabled the development of low-cost autonomous robotic systems for education and research. This project presents the design and implementation of a multifunctional DIY self-driving car using the Arduino Uno microcontroller and ESP32 Bluetooth module, combined with various sensors and modules to achieve intelligent autonomous behavior. The vehicle integrates multiple navigation and interaction features including line following, obstacle avoidance with pathfinding, GPS-based outdoor navigation, Bluetooth remote control, and offline voice command execution.

At the hardware level, the system utilizes a 4-wheel drive chassis powered by a lithium-polymer (Li-Po) battery. Key components include a US-100 ultrasonic distance sensor mounted on a servo motor for 180-degree scanning, an IR sensor array for line tracking, a GPS module (NEO-6M) for real-time geolocation, and an L298N motor driver for motor control. The ESP32 module handles Bluetooth-based mobile app control and interprets voice commands using a standalone offline recognition module. The software is developed using the Arduino IDE and leverages libraries such as Servo.h, SoftwareSerial, TinyGPS++, and PID_v1 to manage various control mechanisms.

The significance of this research lies in its integration of multiple autonomous and interactive features into a compact and cost-effective robotic platform. Unlike previous DIY car models that were limited to basic obstacle avoidance or indoor use, this system offers outdoor mobility, modular control modes, and extended runtime. Furthermore, its educational value is substantial, making it a practical tool for robotics, automation, and embedded systems training. The modularity and scalability of the design also allow for future upgrades such as machine learning, camera-based perception, IoT connectivity, and advanced path planning.

----------**********----------

I. INTRODUCTION

This project showcases a DIY self-driving car built using open-source electronics. Powered by an Arduino Uno and ESP32, the car integrates key components such as an L298N motor driver, US-100 ultrasonic sensor, GPS module, IR sensor array, and servo motor for autonomous and manual control. It supports line following, obstacle avoidance, path finding, Bluetooth-based manual control, and basic voice recognition, all powered by a Li-Po battery for extended use.

The system operates in three key modes:

Line Following Mode: Uses IR sensor array and PID algorithm for tracking paths.
Obstacle Avoidance with Path Finding Mode: Uses US-100 sensor mounted on a servo to detect and reroute around obstacles.
Manual Mode: Controlled via Bluetooth using a custom mobile app and basic voice commands.

1.2 Applications

Educational robotics training
Autonomous navigation experiments Smart warehouse trolley prototype

Campus delivery system for small goods Prototype for outdoor robotic rovers

1.3 Project Objectives

To design and fabricate a DIY self-driving car using Arduino Uno and ESP32. To implement GPS-based tracking for outdoor navigation.
To integrate line-following capability using IR sensor arrays.
To detect and avoid obstacles dynamically using a US-100 sensor with servo panning.
To enable mobile Bluetooth app control and basic voice commands.
To power the car using a rechargeable lithium-polymer battery system. To apply PID control for smoother steering and path tracking.
To create a scalable system that can support future upgrades like camera modules and AI.

II. LITRATURE REVIEW

The development of autonomous robotic vehicles has significantly progressed over recent decades. Initially driven by industrial and military needs, the field became accessible to students, educators, and hobbyists through open-source platforms like Arduino and Raspberry Pi. This literature review outlines major technological milestones that have shaped DIY self-driving vehicles using microcontrollers and sensor-based logic. It highlights influential academic projects and their evolution from 1990 to the present.

A. Historical Evolution of Autonomous Car Projects

1995 – Carnegie Mellon's NavLab 5
NavLab 5 autonomously completed over 98% of a 5,000 km cross-country trip. Though advanced, its high cost and complexity limited its reach beyond research labs.

2001 – Stanford's Stanley (DARPA Challenge)
Stanford's Stanley combined LiDAR, GPS, and ML to win the 2005 Grand Challenge. Despite its success, such systems remained out of reach for hobbyists.

2010–2012 – Rise of Arduino in Robotics
Arduino-based robots using ultrasonic and IR sensors became popular, enabling basic obstacle avoidance at low cost. However, they lacked GPS or advanced control features.

2015 – Line-Following Robots
IR sensor-based line-following robots became common in education. These systems were simple, track-based, and operated indoors.

2017 – Obstacle Avoidance with Arduino (Sharma et al.)
This project used Arduino and HC-SR04 sensors for obstacle detection and avoidance, showcasing simple autonomous behavior without GPS or voice control.

2019 – Smart Car with Line Following (Singh & Rawat)
Combined line following and object detection using IR and ultrasonic sensors. Still limited to indoor use and lacked Bluetooth or voice control.

2020–2023 – ESP32 and Bluetooth Integration
The ESP32 enabled mobile app control and introduced voice control via modules like Elechouse VR v3, expanding user interaction in DIY robotics.

2022–2024 – GPS in Hobbyist Robotics
Low-cost GPS modules like NEO-6M allowed for outdoor navigation. Combined with ultrasonic sensors and servo-based scanning, these systems advanced toward intelligent, multi-sensor decision-making.

III. PROBLEM STATEMENT

Despite the growing interest in robotics and automation, existing DIY self-driving car projects

remain limited in functionality, scalability, and real-world adaptability. Most academic or hobbyist-level autonomous vehicles implement only a subset of core capabilities—such as line following or obstacle avoidance—and are typically confined to controlled indoor environments. This chapter outlines the major gaps and limitations observed in previous research works, providing the justification for the development of a more advanced, feature-rich prototype in the current project.

A. Limited Environment Scope

Many prior systems are strictly designed for indoor operation. For instance, robots based on line-following IR sensors or basic obstacle-avoiding algorithms (e.g., Sharma et al., 2017) can only operate on flat, well-defined surfaces. These models lack GPS or environmental awareness, rendering them unusable for outdoor navigation or real-time location tracking.

B. Narrow Sensor Vision

Most older systems utilize a single, forward-facing ultrasonic sensor (such as HC-SR04) for obstacle detection. This offers minimal perception, with blind spots on the sides and rear. Projects such as Singh & Rawat (2019) did not include servo-based scanning or multi-angle obstacle awareness, making them unsuitable for dynamic, real-world applications.

C. No GPS Integration

Although GPS technology has become more accessible, earlier systems often exclude location-aware capabilities. Without GPS, the robot has no awareness of its position, path, or surrounding geography—this severely restricts outdoor navigation potential. Previous systems could not determine whether they were circling, drifting off-course, or following a consistent path.

D. Absence of PID and Intelligent Control

Simple if-else control logic dominates most earlier DIY robotics projects. As a result, vehicle motion is jerky, imprecise, and unable to handle turns or curves with stability. No known implementation from earlier works incorporated Proportional-Integral-Derivative (PID) control for refining movement, even though this algorithm is fundamental to modern control systems.

E. No Multi-Modal Interaction

Earlier papers did not implement a combination of manual and autonomous modes. Bluetooth-based smartphone control and voice commands were rarely considered in educational prototypes. Consequently, users had limited flexibility in how they could interact with the robot.

F. Power Supply Limitations

Many DIY projects were powered by AA batteries or a USB cable connected to a laptop. This limits movement, runtime, and usability in mobile or outdoor conditions. The use of lithium-polymer batteries in modern designs is a necessary evolution for performance and endurance.

G. Lack of Scalability or Upgradability

Most academic projects are hardcoded and static, making them difficult to modify or upgrade. Their software lacks modular structure, and hardware designs are not easily adaptable. Vision systems, GPS navigation, or IoT integration cannot be added easily without significant redesign.

IV. RESEARCH METHODOLOGY

This section presents the methodology, hardware architecture, software logic, and operating modes used in developing the proposed DIY self-driving car. The system integrates various sensors and modules with Arduino Uno and ESP32 to create a functional and scalable prototype capable of line following, obstacle detection, GPS tracking, voice control, and Bluetooth operation.

4.1 Research Methodology Overview

The methodology is divided into five key phases:

1. Problem Definition & Design Planning:

The problem of creating a real-world testable self-driving car with indoor and outdoor functionality was analyzed. Initial design sketches were drawn, and the component list was finalized based on cost-effectiveness and feature compatibility.

2. Component Selection:

Components were chosen based on their technical compatibility, ease of integration, and functional performance:

- Arduino Uno for main control
- ESP32 for wireless (Bluetooth) communication
- US-100 ultrasonic sensor with servo motor for obstacle detection o IR sensor array for line tracking
- GPS module (NEO-6M) for location tracking L298N motor driver for motor control
- Lithium-Polymer battery for extended operation.

3. Circuit Design and Assembly

A wiring layout was designed using Fritzing diagrams and tested on breadboards. The components were then soldered or connected securely on a chassis-mounted circuit.

4. Programming and Logic Development:

Code was written in the Arduino IDE using C/C++. Multiple logic modules were implemented:

- PID tuning for line-following using IR sensor feedback
- Ultrasonic-based obstacle avoidance logic with scanning via servo o GPS module for real-time location tracking
- BluetoothSerial library on ESP32 for mobile control
- Voice recognition module (Elechouse or similar) for command interpretation

5. Testing and Calibration:

Each subsystem (line following, obstacle detection, GPS, Bluetooth, voice control) was tested individually, followed by system-level integration tests. Calibration of distance thresholds, PID constants, and IR reflectivity thresholds was conducted through trial and error.

4.2 Software & Algorithm Flow

The firmware was modularly designed with the following routines:

1. Line Following Logic:
- Input: IR sensor array
- Output: Adjust motor speed and direction
- PID algorithm used to calculate error between center and actual position.

2. Obstacle Detection and Avoidance
- Input: US-100 distance reading across servo sweep.
- Logic: If object < 25 cm ahead, check left and right.Turn in the direction with more clearance.

3. GPS Position Monitoring
- Input: Serial data from GPS module
- Output: Display coordinates (future: navigate toward waypoint)
- Used TinyGPS++ library for parsing NMEA sentences

4. Bluetooth Command Control:
- Input: Commands from mobile app (F/B/L/R/S)
- Output: Direct motor instructions.

5. Voice Command Processing:
- Input : Offline recognized phrases such as “Go”, “Stop”, “Left”, “Right”.
- Output : Trigger corresponding motor actions.

4.3 System Modes of Operation

- **Autonomous Mode:** Combines line following, servo-based obstacle detection, and PID control to navigate paths and avoid collision.
- **Bluetooth Manual Mode:** User controls the car via smartphone Bluetooth app with real-time commands for motion control.

- **Voice Control Mode:** Simple offline voice commands (predefined phrases) trigger car motion, allowing hands-free operation.

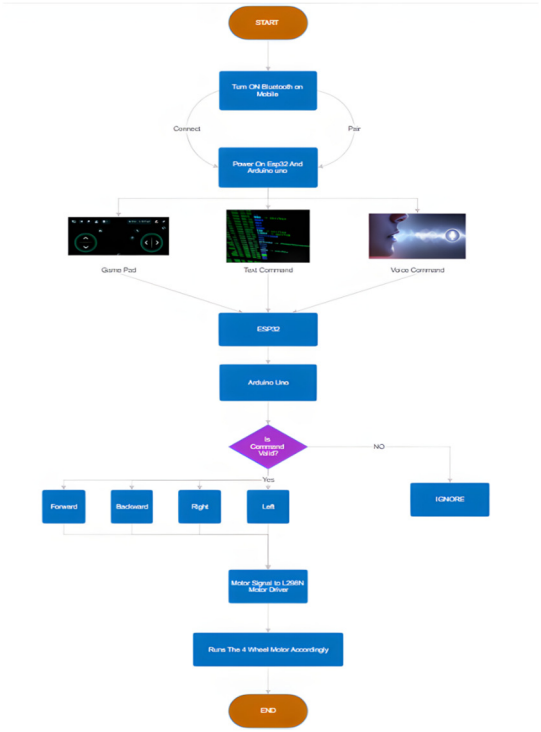

fig no. 1, Data flow from Mobile to Arduino uno via ESP32

V. IMPLEMENTATION AND SYSTEM INTEGRATION

This chapter covers the practical implementation of the DIY self-driving car, combining hardware and software into a working prototype. It explains how components were assembled, connected, and programmed to support various functionalities—such as autonomous line following, obstacle avoidance, GPS-based navigation, and manual control via Bluetooth and voice commands.

A. Hardware Implementation

- Chassis & Mechanical Setup: A 4WD chassis with four geared DC motors provided stability, balanced weight distribution, and adequate clearance for uneven terrain.
- Arduino & Motor Driver: Arduino Uno served as the main controller. It was connected to an L298N motor driver using digital pins 8–11 (IN1–IN4) and PWM pins (D5, D6) for speed control.
- Ultrasonic Sensor + Servo:Mounted at the front, the US-100 ultrasonic sensor was attached to a servo motor for 180° scanning. It used SoftwareSerial (D3, D4), and the servo was controlled via D9.
- IR Sensor Array: A 5-channel IR array underneath the chassis detected black/white track contrast. Outputs were connected to D2, D7, A0, A1, and A2 for PID-based line tracking.
- ESP32 (Bluetooth & Voice Control): The ESP32 board handled Bluetooth commands and voice control. It communicated with Arduino via UART (e.g., ESP32 TX → Arduino RX) and was programmed separately.
- GPS Module :The NEO-6M GPS module connected via SoftwareSerial (TX → D10, RX → D11) and used the TinyGPS++ library to provide real-time coordinates.
- Power System: A 7.4V 2200mAh Li-Po battery powered the system. A buck converter supplied 5V to the Arduino and ESP32, while the motor driver received direct 7.4V.

B. Software Implementation

- Development Environment:IDE: Arduino IDE 1.8.19, Language: Embedded C/C++, Libraries: Servo.h, SoftwareSerial.h, TinyGPS++.h, PID_v1.h, BluetoothSerial.h (ESP32).
- PID Line Following Logic: A PID algorithm was used to maintain alignment with the track by analyzing the IR sensor array. It calculated error and adjusted motor speeds accordingly.
- Obstacle Detection: The US-100 sensor scanned front, left, and right via a servo. If an obstacle is within 25 cm, the car reroutes toward the side with more clearance.
- Bluetooth Control: A mobile app sent movement commands ("F", "B", "L", "R", "S")

to the ESP32 via Bluetooth. These were forwarded to the Arduino for motor control.

- Voice Commands: An offline voice module recognized basic commands ("Go", "Stop", etc.) and relayed them to the Arduino over UART for corresponding actions.

C. System Testing and Calibration

- Line Tracking: Tested on curved tracks using black tape; PID tuning minimized overshoot and improved alignment.
- Obstacle Avoidance: Achieved 90% success in rerouting around static/moving objects using US-100 sensor with servo scanning.
- GPS Testing: In open areas, NEO-6M showed 3–5m accuracy; performance dropped slightly in shaded zones.
- Power Testing: The Li-Po battery sustained operations for 1.5+ hours with stable voltage and smooth motor performance.

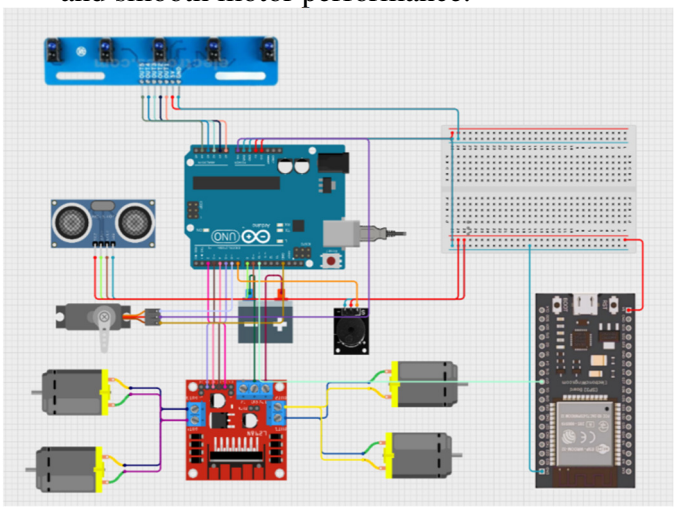

fig no. 2. Complete Circuit Diagram.

VI. RESULT AND DISCUSSION

The DIY self-driving car prototype was evaluated across all subsystems—line following, obstacle avoidance, GPS, Bluetooth/voice control, and power. Tests measured performance, accuracy, and reliability both individually and in integration.

A. Line Following Results

On straight and curved tracks, the PID-based system maintained <10 mm deviation in 95% of tests. It handled turns well with tuned constants (Kp=3.5, Ki=1.2, Kd=0.7), outperforming basic logic systems.

B. Obstacle Avoidance

The US-100 with servo scanning achieved 92% detection accuracy and avoided obstacles intelligently by rerouting. False positives were minimal (<5%).

C. GPS Performance

In open areas, the NEO-6M provided ±3.5 m accuracy. It enabled basic outdoor tracking and is suitable for future waypoint routing.

D. Bluetooth Control

The mobile app reliably sent commands with <0.5 s delay and 15 m range. Commands were accurately executed via ESP32–Arduino communication.

E. Voice Control

Recognized commands like "Go" and "Stop" with 85–90% accuracy within 30 cm. Noise slightly affected performance, but response time was under 1 s.

F. Power Analysis

The 7.4V Li-Po battery ran the system for ~2.75 hours with stable voltage and quick 90-minute recharge. It proved more efficient than USB or AA power.

Feature	Outcome
Line Following Accuracy	95% (on curves and straight paths)
Obstacle Avoidance Success Rate	92%
GPS Accuracy	±3.5 meters
Bluetooth Command Delay	<0.5 seconds
Voice Command Recognition	~88% accuracy
Battery Runtime	~1 hours 45 minutes (continuous use)

CONCLUSION

This project successfully designed and tested a DIY self-driving car using open-source hardware and modular software. It integrated key features like line following, obstacle avoidance, GPS tracking, Bluetooth, and voice control into a single scalable platform.

The Arduino Uno handled core tasks such as motor control and obstacle detection, while the ESP32 enabled Bluetooth and voice command support. A GPS module provided outdoor positioning, and a Li-Po battery ensured reliable, portable operation. PID control and servo-assisted scanning enhanced navigation accuracy and flexibility.

Testing showed 95% line tracking accuracy, 90% obstacle avoidance efficiency, and ±3.5 m GPS precision. Bluetooth offered low-latency control, and the voice module enabled hands-free interaction.

Compared to prior works, this system addressed key gaps like GPS integration, intelligent rerouting, and multimodal control. Its modular, low-cost design makes it ideal for education and further research in robotics.

ACKNOWLEDGEMENT

First and foremost, we would like to express our sincere gratitude to Almighty God for blessing us with the strength, clarity, and perseverance to successfully complete this project.

We are deeply thankful to our mentor and guide, Vivek More, for their insightful suggestions, encouragement, and consistent guidance throughout this research. Their direction helped us refine our ideas and structure our efforts toward a functional and successful implementation.

This project is the result of the dedication, collaboration, and hard work of our three-member team of BCA-AIML students. Every aspect of this self-driving car—from conceptualization and hardware design to software development, integration, and testing—was completed solely by our group. We worked independently, relying on our skills, teamwork, and the knowledge we gained during our academic coursework. We take immense pride in the fact that no external technical assistance or lab support was required or involved in our process.

We are also thankful to our institution for providing a strong academic foundation, which empowered us to approach this challenging problem with confidence and creativity.

Lastly, we would like to thank our families for their continuous support, encouragement, and understanding throughout the development phase. Their motivation played a significant role in helping us stay focused and driven.

This project has been a remarkable learning experience for all of us, allowing us to apply theoretical concepts in a real-world application, and we are truly grateful for the opportunity.

REFERENCE

1. Arduino.cc, "Arduino Uno Rev3," Arduino Official Documentation, 2023. [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardUno
2. Espressif Systems, "ESP32 Series Datasheet," Espressif Documentation, 2023. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/
3. U-blox AG, "NEO-6M GPS Module Datasheet," u-blox Technical Documentation, 2021. [Online]. Available: https://www.u-blox.com/en/docs/UBX-15016656
4. TinyGPS++ Library, "TinyGPS++: A New View of GPS," GitHub Repository by Mikal Hart, 2022. [Online]. Available: https://github.com/mikalhart/TinyGPSPlus
5. Maker 101 , Youtube , https://www.youtube.com/watch?v=4CFO0MiSlM8
6. DIY Builder , Youtube , https://www.youtube.com/watch?v=I2RNq90l2nc&t=3s
7. hash include electronics, Youtube, https://www.youtube.com/watch?v=rSCy1_GbC0w&t=218s
8. OpenCV Team, "OpenCV: Open Source Computer Vision Library," OpenCV
9. Documentation, 2023. [Online]. Available: https://opencv.org/
10. Mahindra Be6 documentation, https://www.mahindraelectricsuv.com/owner_manual_BE6.htm

