# ABSTRACT

The increasing number of vehicles on roads has led to a significant demand for intelligent driving systems that enhance safety and driver convenience. One such innovation is Adaptive Cruise Control (ACC), a technology that automatically adjusts a vehicle's speed to maintain a safe distance from the vehicle ahead. This project aims to simulate an ACC system using cost-effective and readily available components such as the Arduino UNO, HC-SR04 ultrasonic sensor, L298N motor driver, and a DC motor. The core functionality of this system lies in its ability to measure the distance to the object in front and accordingly regulate the speed of the vehicle. The ultrasonic sensor continuously senses the distance to an obstacle, and the Arduino processes this data to generate pulse-width modulation (PWM) signals. These PWM signals are used to control the speed of the motor through the L298N driver module. A 16x2 LCD module displays real-time values of distance and motor speed. This prototype replicates the basic principles of modern adaptive cruise control systems found in automobiles. It demonstrates real-time response to varying traffic conditions by reducing or increasing speed based on proximity to the object ahead. The system enhances road safety and fuel efficiency by ensuring optimal speed without requiring manual throttle control. The project not only serves as a practical implementation of sensor-based automation but also provides a foundational understanding of embedded systems, motor control, and intelligent transportation technologies. It is a valuable contribution for learners and engineers aiming to work in automotive electronics, robotics, or automation fields.

# TABLE OF CONTENTS

# Table of Figures

# CHAPTER 1: INTRODUCTION

## 1.1 Introduction

With the rapid advancement of technology in the automotive sector, intelligent transportation systems have become a focal point of modern engineering design. One of the most impactful innovations in this domain is the Adaptive Cruise Control (ACC) system. This system is a smarter evolution of conventional cruise control technology and plays a crucial role in enhancing vehicle safety and improving the driving experience. Unlike traditional cruise control, which maintains a constant speed set by the driver, an adaptive cruise controller can automatically adjust the vehicle's speed based on the surrounding traffic conditions, especially the distance to the vehicle ahead.

The primary motivation behind developing ACC systems is the growing need for collision avoidance, traffic congestion reduction, and fuel efficiency in modern transportation. According to recent studies, driver fatigue and inattentiveness are major contributors to road accidents. ACC systems help mitigate this by providing semi-autonomous control over vehicle speed, thereby reducing the workload on the driver and enhancing reaction times in dynamic traffic environments.

This mini project aims to demonstrate the working principle of an adaptive cruise controller using basic and cost-effective hardware components such as an Arduino UNO microcontroller, an ultrasonic distance sensor (HC-SR04), an L298N motor driver, and a DC motor that simulates the vehicle’s motion. The system operates by continuously measuring the distance to any object in front using the ultrasonic sensor. Based on the distance data, the Arduino processes the input and controls the speed of the motor accordingly using PWM (Pulse Width Modulation) signals. If the object ahead is too close, the system automatically reduces the motor speed or halts it, and if the path is clear, the motor returns to its full speed. A 16x2 LCD display is integrated to show the real-time distance and speed percentage, giving users immediate feedback on system behavior.

This project not only illustrates an essential function found in modern vehicles but also offers an excellent platform for students to understand concepts related to embedded systems, sensor integration, real-time control, and automation. It bridges the gap between theoretical knowledge and real-world applications, encouraging innovation in the field of smart mobility and autonomous driving technologies. Through this project, students gain practical insight into how microcontrollers can be employed to build intelligent systems that react to environmental stimuli, ultimately preparing them for more complex applications in automotive engineering and the Internet of Things (IoT).

## 1.2 Literature Survey

**1. Title: A hybrid MPC approach to the design of a Smart adaptive cruise controller**
**Authors:** Daniele Corona, Mircea Lazar, Bart De Schutter, Maurice Heemels
**Technology Used:** Hybrid Model Predictive Control (Hybrid MPC), PieceWise Affine (PWA) system modeling, Terminal cost and constraint set adaptation
**Work Done:** Designed a Smart car adaptive cruise controller using Hybrid MPC to track a moving reference (leading vehicle), ensuring safety under physical and computational constraints; modified terminal set MPC from fixed-point control to suit trajectory tracking
**Drawback:** Computational complexity remains a challenge for real-time execution; limited testing under diverse real-world driving conditions; relies on accurate system modeling
**Published In:** 2006 IEEE Conference on CACSD

**2. Title: Researches on Adaptive Cruise Control system: A state of the art review**
**Authors:** Liangyao Yu, Ruyue Wang
**Technology Used**: ACC, ADAS, control theories (PID, fuzzy logic, MPC), HiL, human-machine interaction
**Work Done:** Reviewed ACC algorithms, validation methods, real-world applications, and user interaction studies
**Drawback:** No new methods proposed; broad overview with limited depth in specific areas
Published In: Journal of Automobile Engineering, Vol. 236.

**3. Title: Stop and go controller for adaptive cruise control**
**Authors:** M. Persson, F. Botling, E. Hesslow, R. Johansson
**Technology Used:** Adaptive Cruise Control (ACC), autonomous car-following algorithms, onboard vehicle computers
**Work Done:** Developed and tested an ACC system capable of functioning in both low-speed stop-and-go and high-speed driving scenarios
**Drawback:** Early-stage implementation; may lack advanced optimization for real-world variability in urban driving
**Published In:** 1999 IEEE International Conference

**4. Title: Development and testing of a fully Adaptive Cruise Control system**
**Authors:** Gennaro Nicola Bifulco, Luigi Pariota, Fulvio Simonelli, Roberta Di Pace
**Technology Used:** Linear car-following model, multilayer architecture (sampler, profiler, tutor, performer), human-like ACC logic
**Work Done:** Proposed a fully-adaptive ACC system that learns and applies individual driver preferences in real time through a learning mode and multi-layer control structure
**Drawback**: Excludes integration with V2V; limited real-world testing on performer module; tutor and performer layers not deeply explored
**Published In**: Transportation Research Part C: Emerging Technologies

**5.Title: Cooperative Adaptive Cruise Control: A Reinforcement Learning Approach**
**Authors:** Charles Desjardins, Brahim Chaib-draa
**Technology Used:** Cooperative Adaptive Cruise Control (CACC), Reinforcement Learning, Policy-Gradient Methods, Vehicle-to-Vehicle (V2V) communication
**Work Done:** Developed a CACC system using reinforcement learning with policy-gradient methods and validated it through simulations.
**Drawback:** Limited to simulation; lacks real-world testing for scalability, safety, and data requirements.
**Published In:** IEEE Transactions on Intelligent Transportation Systems

6**. Title: An IMM Algorithm for Tracking Maneuvering Vehicles in an Adaptive Cruise Control Environment**
**Authors:** Yong-Shik Kim, Keum-Shik Hong
**Technology Used:** IMM algorithm, Unscented Kalman Filter (UKF), stochastic hybrid system, radar-based vehicle tracking
**Work Done:** Developed a vehicle tracking method using IMM-UKF to accurately detect linear and curvilinear maneuvers in ACC systems.
**Drawback:** Complexity is high and results are limited to simulation without real-world validation.
**Published In:** International Journal of Control, Automation, and Systems.

## 1.3 Gap Identified

1. **Limitations of Traditional Cruise Control Systems**
- Conventional cruise control systems only maintain a constant speed set by the driver.
- These systems do not respond to changes in traffic conditions, road curvature, or sudden braking of the vehicle ahead.
- Lack of adaptability increases the risk of collisions and driver fatigue in dense traffic.

2. **High Cost and Complexity of Commercial ACC**
- Modern Adaptive Cruise Control systems use radar, LiDAR, and camera modules, making them expensive and complex.
- Integration of such systems is limited to luxury and premium vehicles.
- Budget and educational projects cannot afford these advanced sensor modules, limiting accessibility.

3. **Lack of Affordable Educational Prototypes**
- Most academic or DIY models focus on open-loop motor control or basic obstacle detection.
- Very few models demonstrate closed-loop control with real-time sensor feedback, especially related to vehicle safety.
- Students and hobbyists lack a low-cost, hands-on prototype to understand adaptive systems in transportation.

4. **Underutilization of Open-Source Platforms**
- Platforms like Arduino and Raspberry Pi are capable of simulating intelligent systems but are underused in automotive safety project contexts.
- Few documented implementations of ACC using microcontrollers exist for educational demonstration purposes.
- There is a need for simple, open-source solutions that bridge embedded systems with intelligent control logic.

5. **Limited Exposure to Real-Time Decision-Making in Student Projects**
- Many student projects do not involve real-time decision-making based on sensor inputs.
- Projects typically focus on pre-programmed or reactive systems without dynamic adaptation.
- A gap exists in integrating sensors, controllers, and actuators in a single functional feedback loop for autonomous control.

# 1.4 Problem statement & objectives

## Problem Statement

- Traditional cruise control systems lack the ability to adjust speed based on surrounding traffic.
- Existing ACC systems are costly and not feasible for low-budget or educational use.
- There is a gap in affordable, real-time prototypes that demonstrate adaptive speed control.
- Most student projects lack integration of sensor-based dynamic decision-making.
- A low-cost embedded system is needed to simulate ACC using Arduino and basic sensors.
- The system must ensure safe following distance by controlling motor speed based on object proximity..

## Objectives

1. Design a low-cost Arduino-based system to simulate adaptive cruise control functionality.
2. Control motor speed in real-time using ultrasonic distance sensing and PWM signals.
3. Display live distance and speed data on an LCD for real-time monitoring and analysis

# CHAPTER 2: METHODOLOGY

## 2.1 Methodology

### 2.1.1. System Overview

- The Adaptive Cruise Controller system uses an ultrasonic sensor to detect the distance to obstacles ahead and adjusts the motor speed accordingly using an Arduino and L298N driver. It ensures automatic speed control based on real-time sensor data, simulating safe vehicle behavior..

### 2.1.2. Sensor Integration and Data Collection

- Ultrasonic Sensor (HC-SR04):
  - Used to detect obstacles or vehicles in front of the car.
  - Emits ultrasonic waves and measures the echo time to calculate distance.
  - Provides continuous feedback for decision-making.
- Real-Time Monitoring:
  - The sensor collects data every few milliseconds.
  - Enables quick reaction to any changes in distance or obstacle appearance.

### 2.1.3. ESP32 as Central Controller:

  - Reads input from the ultrasonic sensor.
  - Processes data using conditional statements
  - If the distance is greater than 50 cm → the motor runs at full speed.
  - If the distance is between 30 cm and 50 cm → the motor runs at medium speed.
  - If the distance is between 15 cm and 30 cm → the motor runs at low speed.
  - If the distance is less than 15 cm → the motor is stopped for safety

### 2.1.4. Motor Speed Control and Actuation

- Arduino sends PWM signals to control motor speed.
- L298N motor driver receives PWM and powers the DC motor.
- Motor simulates vehicle movement by rotating wheels.
- As distance to an obstacle decreases, PWM is reduced, slowing the motor.
- The car moves forward only; direction pins (IN1/IN2) can be changed to reverse if needed.

### 2.1.5. Display and Feedback Mechanism

- 16x2 LCD displays current distance and motor speed percentage.
- Helps users observe system behavior during operation.
- Optional buzzer or LED can alert when the car is too close to an obstacle.
- Adds safety and improves user interaction during testing.

### 2.1.6. Algorithm and Decision-Making Logic

- The ESP32 code uses if-else conditions to compare measured distances with predefined thresholds.
- Based on the result, it adjusts the motor speed or stops the vehicle to maintain a safe following distance.
- The main loop function runs continuously, ensuring real-time response to changes in obstacle distance.
- This enables dynamic and automatic speed control without any manual intervention..
- This allows the system to be upgraded for more advanced autonomous vehicle applications.

### 2.1.7.System Integration and Testing

- All hardware components are connected and mounted securely on the vehicle chassis.
- Arduino code is uploaded, and the system is powered on for real-world testing.
- The system is tested by placing obstacles at various distances to simulate traffic conditions.
- The motor responds accordingly, varying its speed based on proximity to obstacles.
- LCD output and system behavior are observed to verify if the logic executes correctly.
- The complete test confirms successful integration of sensors, controller, motor driver, and feedback

## 2.2 Flow chart

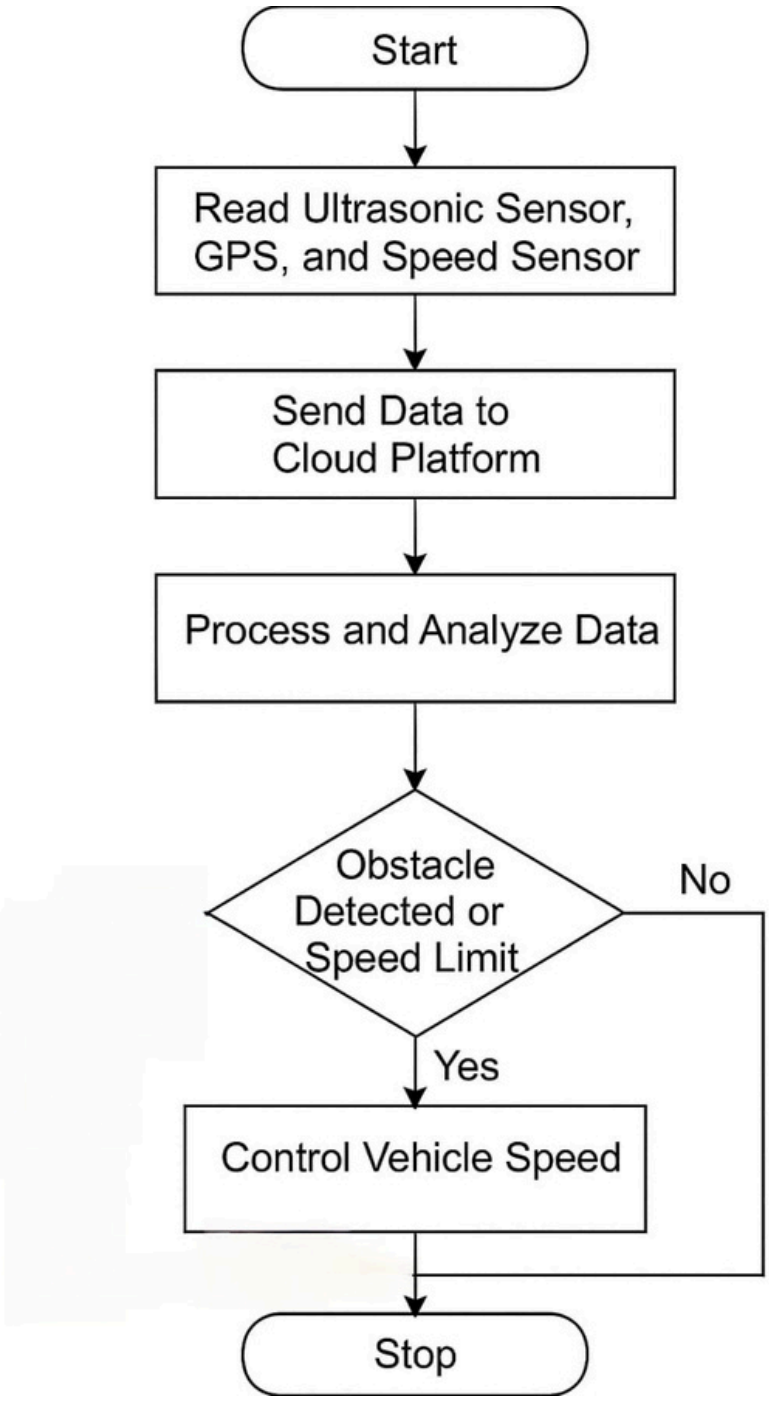



fig 2.2.1

# CHAPTER 3: HARDWARE IMPLEMENTATION

## 3.1 Hardware description

### 3.1.1 ESP32

fig 3.1.1

- Acts as the main controller for the system.
- Processes sensor data and controls motor speed using PWM signals.
- Chosen because it offers built-in Wi-Fi and Bluetooth, higher processing power, and more GPIO pins compared to Arduino UNO.
- Not using Raspberry Pi or STM32 as they are either overpowered or more complex for this specific application, and ESP32 offers a good balance between performance, features, and cost

### 3.1.2 HC-SR04 Ultrasonic Sensor

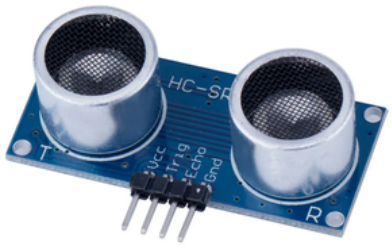



fig3.1.2

- Measures distance to the object in front using ultrasonic waves.
- Sends distance data to Arduino in real-time for speed adjustment.
- Selected for its accuracy, reliability, and simplicity of integration with Arduino.
- IR sensors were avoided due to poor accuracy and light sensitivity; radar was too costly.

### 3.1.3 L298N Motor Driv

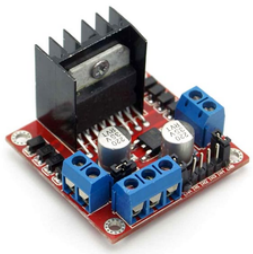


fig3.1.3

- Receives PWM signals from Arduino to control motor speed and direction.
- Includes built-in heat sink and dual H-bridge configuration.
- Preferred over MOSFET circuits for ease of setup and over L293D for better power handling.

### 3.1.4 **DC Motor**

fig3.1.4

- Represents the vehicle's motion; speed is varied as per obstacle distance.
- Easy to control using PWM for demonstration purposes.
- Simple and cost-effective motor with visible speed variation.
- Stepper and servo motors were avoided due to unnecessary complexity.

### 3.1.5. **16x2 LCD Display**

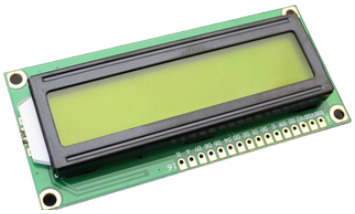


fig3.1.5

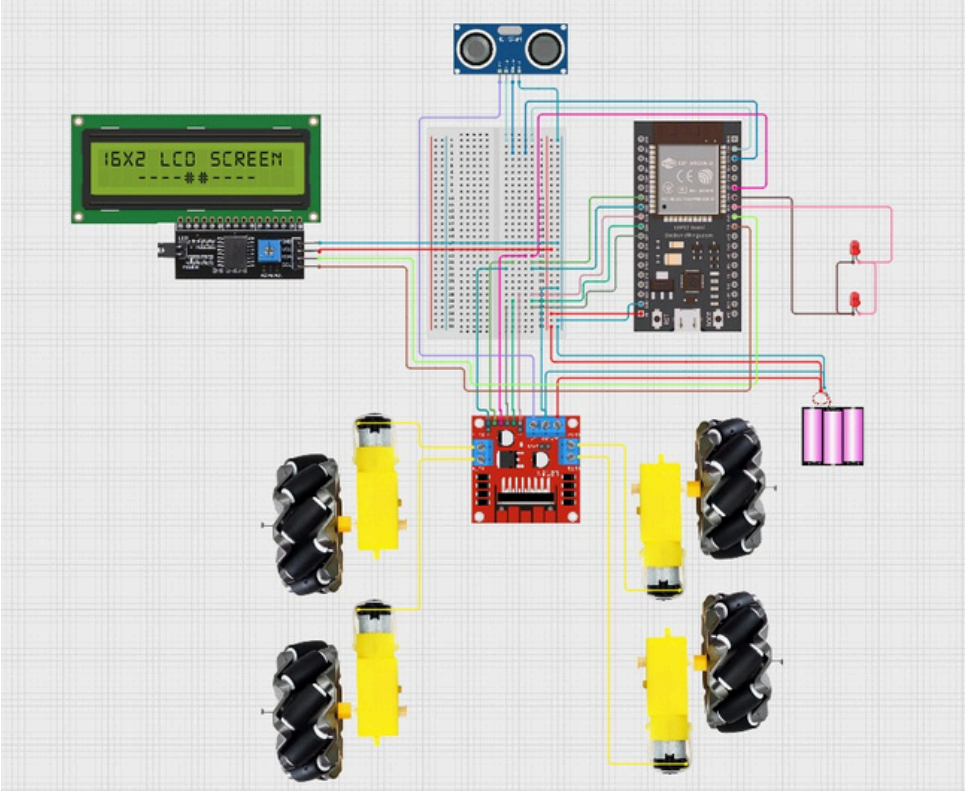
- Displays real-time distance and motor speed percentage.
- Enhances user interaction and system transparency during operation.
- Easy to interface with Arduino (via I2C or parallel connection).
- OLED/TFT displays were not used to keep cost and power consumption low.

### 3.1.6.**Power Supply (Batterv )**

fig3.1.6

- Provides power to Arduino, motor driver, and motor.
- Chosen for portability and simplicity during demonstration.
- Can be 9V battery or 12V adapter depending on motor specs.
- Regulated to ensure stable voltage for all components.

## 3.2 Circuit Connection




- The ESP32 microcontroller is used as the central controller for all components.
- The HC-SR04 ultrasonic sensor is connected to ESP32 for object distance measurement.
- A 16x2 LCD display (I2C module) is used to show values like speed and distance.
- The I2C LCD connects via SDA and SCL pins to the ESP32 (usually GPIO 21 and 22).
- The L298N motor driver is connected to ESP32 to control four DC gear motors.
- Each motor is connected to the output terminals of the L298N driver.
- Power supply (battery) is connected to the motor driver and shared with ESP32 (via Vin/GND).
- Two LED indicators are connected to GPIO pins of ESP32 to show system or sensor status.
- All grounds (GND) of ESP32, sensors, driver, and power supply are connected together

# CHAPTER 4: SOFTWARE AND TOOLS

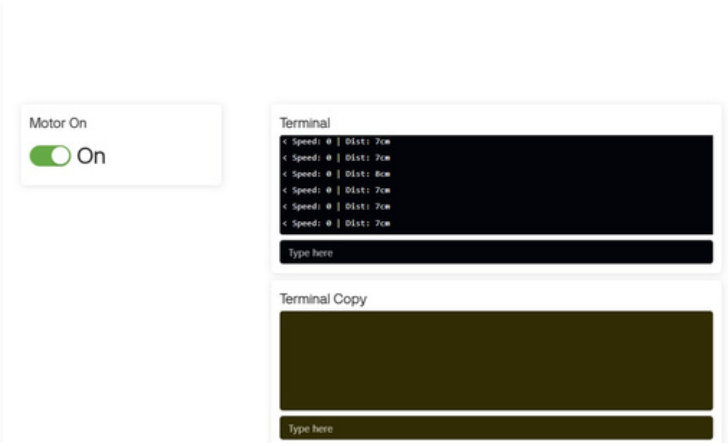
## 4.1 Arduino IDE:

- Arduino IDE is an open-source software used to write and upload code to microcontroller boards.
- It supports the ESP32 board by installing the required board manager files.
- The interface is simple and user-friendly, making it ideal for beginners.
- It uses a simplified version of C/C++ for programming embedded applications.
- A built-in Serial Monitor helps in real-time data viewing and debugging.
- The IDE allows easy installation of libraries for sensors and modules.
- It is lightweight and works on Windows, macOS, and Linux platforms.
- Arduino IDE was used in this project for its simplicity and strong community support

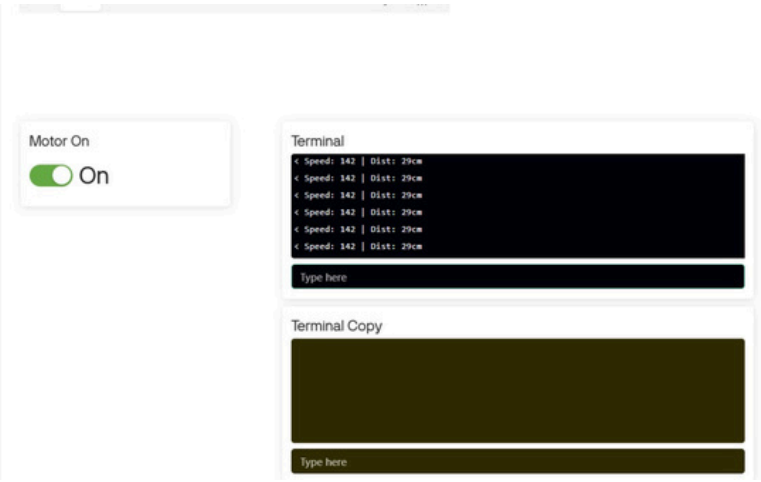
## 4.2 Blynk IoT Platform:

- Blynk is a platform that allows you to build IoT applications and control hardware from a smartphone.
- It supports ESP32 and many other boards through Wi-Fi, Bluetooth, or internet connectivity.
- The Blynk mobile app provides widgets to display data like speed, distance, and object detection.
- It uses virtual pins to communicate between the hardware and the app.
- Real-time data can be monitored and visualized on a user-friendly dashboard.
- It requires the Blynk library to be installed in the Arduino IDE for ESP32 communication.
- Blynk helps create wireless control and monitoring systems without complex coding.
- In this project, Blynk was used to display motor speed, sensor distance, and object detection status.

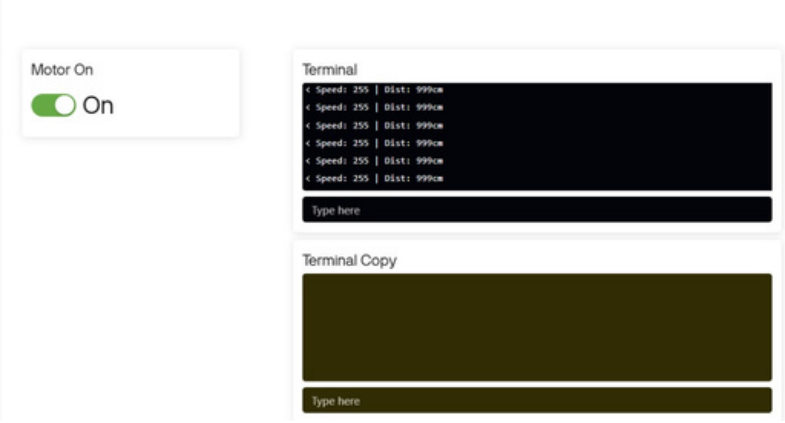
Case 1: Minimum Distance




**Case 2: Medium Distance**




Case 3: Maximum Distance:



# CHAPTER 5: RESULTS AND IDENTIFICATION

## 5.1 Results and Discussions

In this project, the ESP32 was successfully programmed to control motor speed based on the object's distance using an ultrasonic sensor. The system accurately detected distance values and adjusted the motor speed accordingly using PWM signals. The Blynk IoT platform was used to display the real-time values of both distance (in cm) and motor speed (in RPM).

**Case 1: Minimum Distance**

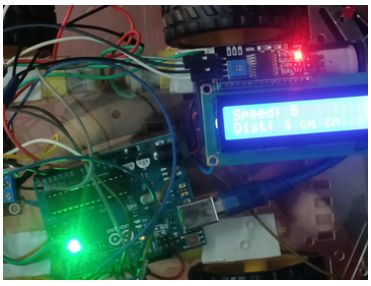


fig 5.1.1

Distance: 15 cm
Speed: 0 RPM
Blynk Output: Distance shown as 15 cm, Speed shown as 0 RPM
Observation: The object was very close, and the motor stopped automatically for safety.

**Case 2: Medium Distance**

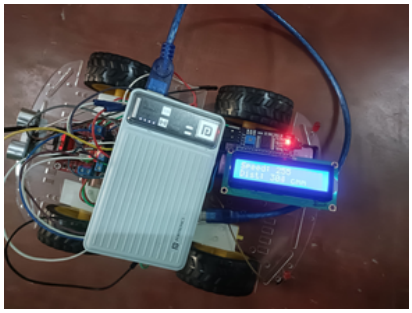


fig 5.1.2

Distance: 25 cm
Speed: 155 RPM
Blynk Output: Distance shown as 25 cm, Speed shown as 155 RPM
Observation: Object was at a moderate distance, and the motor ran at controlled speed

Case 3: Maximum Distance:

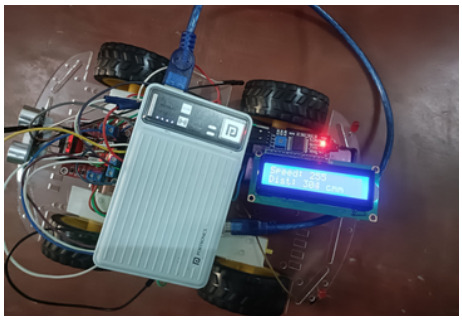


fig 5.1.3

Distance: 40 cm
Speed: Full Speed (255 RPM)
Blynk Output: Distance shown as 40 cm, Speed shown as 255 RPM
Observation: No nearby object detected; motor ran at full speed without restriction.

## 5.2 Advantages

1. Improved Safety
- Automatically maintains a safe distance from obstacles, reducing collision risk.
- Immediate response to sudden obstacles helps prevent accidents.
2. Hands-Free Speed Control
- Maintains speed without constant user input.
- Reduces driver workload in repetitive driving conditions (e.g., slow traffic).
3. Real-Time Obstacle Detection
- Continuously monitors surroundings using ultrasonic sensors.
- Responds instantly to any object in front of the vehicle.
4. Cost-Effective Implementation
- Built using low-cost components like Arduino and ultrasonic sensors.
- Ideal for academic, prototype, or educational use without expensive systems.
5. Modular and Scalable Design
- Can be expanded by adding GPS, speed sensors, or camera modules later.
- Easy to integrate with IoT platforms for advanced versions.
6. Autonomous Control
- Vehicle adjusts speed on its own based on environment conditions.
- Enables basic automation features, similar to real-world smart vehicles.

## 6.3 Conclusion

This project successfully demonstrates a smart motor control system using the ESP32 microcontroller, an ultrasonic sensor, and the Blynk IoT platform. The system was designed to automatically adjust the speed of a DC motor based on the distance of an object. When an object comes closer to the sensor, the motor slows down or stops completely. As the object moves away, the motor gradually resumes and eventually reaches full speed.

Real-time monitoring was achieved using the Blynk app, which displayed both the distance (in cm) and the motor speed (in RPM) using virtual pins. This made it easier to observe and analyze the system's performance without needing to connect additional physical displays. The three different test cases clearly illustrated the intended behavior of the system at minimum (15 cm, 0 RPM), medium (25 cm, 155 RPM), and maximum distance (40 cm, full speed).

## References


1. Kumar A. & Rani S., "**Studied speed control methods in highway work zones**," International Journal of Transportation Engineering, vol. 7, no. 2, pp. 85–91, 2021.
2. Sharma R., Gupta M., & Verma T., "**Developed an IoT-based vehicle tracking system using GPS and GSM,**" Journal of Emerging Trends in Computing and Communication, vol. 10, no. 4, pp. 233–239, 2020.
3. Ahmed M.T., Khan A., & Syed N., "**Proposed a smart traffic control system using IoT technology,**" IEEE International Conference on Smart Cities (ICSC), pp. 102–107, 2022.
4. Patel D.A. & Desai K.D., "**Implemented speed control using GPS for zone-based monitoring**," Journal of Intelligent Transport Systems and Applications, vol. 6, no. 3, pp. 144–150, 2021.
5. Saxena A. & Joshi R., "**Designed adaptive cruise control using radar for safe distances,**" Proceedings of the National Conference on Intelligent Mobility Systems, pp. 56–61, 2023.
6. Raj S. & Mehta A., "**Built an IoT-based accident prevention system using sensors,**" International Journal of Embedded Systems and IoT, vol. 9, no. 1, pp. 25–30, 2022
7. Mishra P. & Rao V., "**Design and implementation of a real-time vehicle speed monitoring system,**" International Journal of Electronics and Communication Systems, vol. 8, no. 2, pp. 98–104, 2021.
8. Ali N., Hussain F., & Shaikh M., "**IoT-enabled vehicle safety system for urban traffic,**" IEEE International Conference on Internet of Things and Applications, pp. 210–215, 2022.
9. Bose A. & Naik R., "**Smart braking system using distance sensors and embedded control**," Journal of Mechatronics and Automation, vol. 5, no. 1, pp. 33–39, 2020.
10. Tiwari D., Prasad R., & Kaur H., "**Real-time monitoring and speed regulation using ultrasonic sensing and microcontrollers,**" International Journal of Embedded Computing and Smart Systems, vol. 6, no. 4, pp. 172–178, 2023.

# ANNEXURE

```
#define BLYNK_TEMPLATE_ID "TMPL344KpYuQY"
#define BLYNK_TEMPLATE_NAME "Cruise Controller"
#define BLYNK_AUTH_TOKEN "avYC7BNdX9pyZ7lZfAUonfcfdCbLRk0i"
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <WiFi.h>
#include <BlynkSimpleEsp32.h>
char ssid[] = "manoj";
char pass[] = "manoj@123";
#define TRIG_PIN 23
#define ECHO_PIN 22
// Motor driver (L298N)
#define ENA 25
#define IN1 26
#define IN2 27
#define ENB 33
#define IN3 32
#define IN4 21
// Brake LED
#define LED_WARN 19
// I2C LCD pins
#define I2C_SDA 18
#define I2C_SCL 5
LiquidCrystal_I2C lcd(0x27, 16, 2);
// Logic variables
int distance = 0;
int targetSpeed = 0;
float currentSpeed = 0;
float smoothingFactor = 0.1;
bool systemOn = true;
bool ledState = false;
unsigned long lastLCDUpdate = 0;
unsigned long lastBlink = 0;
unsigned long tooCloseStart = 0;
bool forceStopped = false;
bool dangerNotified = false;
else {
 digitalWrite(IN1, LOW); digitalWrite(IN2, LOW);
 digitalWrite(IN3, LOW); digitalWrite(IN4, LOW);
 }
 analogWrite(ENA, pwm);
 analogWrite(ENB, pwm);
}
```

```
void updateLED(int pwm, int prevPwm) {
 if (pwm == 0) {
 digitalWrite(LED_WARN, HIGH); // Solid ON
 } else if (prevPwm > pwm) {
 if (millis() - lastBlink >= blinkInterval) {
 ledState = !ledState;
 digitalWrite(LED_WARN, ledState);
 lastBlink = millis();
 }
 } else {
 digitalWrite(LED_WARN, LOW); // OFF
 }
}
void updateLCD(int pwm, int dist) {
 lcd.setCursor(0, 0); lcd.print("Speed: ");
 lcd.print(pwm); lcd.print(" ");
 lcd.setCursor(0, 1); lcd.print("Dist: ");
 lcd.print(dist); lcd.print("cm ");

 Blynk.virtualWrite(V1, "Speed: " + String(pwm) + " | Dist: " + String(dist) + "cm");
}
void loop() {
 Blynk.run();

 if (!systemOn) {
 setMotors(0);
 digitalWrite(LED_WARN, LOW);
 return;
 }
 distance = readDistanceCM();
const unsigned long lcdInterval = 100;
const unsigned long blinkInterval = 200;
// Blynk ON/OFF button
BLYNK_WRITE(V0) {
 systemOn = param.asInt();
}
void setup() {
 Serial// Pins
 pinMode(TRIG_PIN, OUTPUT);
 pinMode(ECHO_PIN, INPUT);
 pinMode(ENA, OUTPUT);
 pinMode(IN1, OUTPUT);
 pinMode(IN2, OUTPUT);
 pinMode(ENB, OUTPUT);
 pinMode(IN3, OUTPUT);
 pinMode(IN4, OUTPUT);
.begin(115200);
```

```
 pinMode(LED_WARN, OUTPUT);
digitalWrite(LED_WARN, LOW);
// LCD
Wire.begin(I2C_SDA, I2C_SCL);
lcd.init();
lcd.backlight();
lcd.setCursor(0, 0); lcd.print("Adaptive Cruise");
lcd.setCursor(0, 1); lcd.print("Connecting WiFi");

Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);
lcd.clear();    }
int readDistanceCM() {
digitalWrite(TRIG_PIN, LOW); delayMicroseconds(2);
digitalWrite(TRIG_PIN, HIGH); delayMicroseconds(10);
digitalWrite(TRIG_PIN, LOW);
long duration = pulseIn(ECHO_PIN, HIGH, 30000);
if (duration == 0) return 999;
return duration * 0.034 / 2;   }
void setMotors(int pwm) {
pwm = constrain(pwm, 0, 255);
if (pwm > 0) {
  digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);
  digitalWrite(IN3, HIGH); digitalWrite(IN4, LOW);    }

if (distance <= 10 && !dangerNotified) {
Blynk.logEvent("too_close", "🚨 Obstacle too close!");
dangerNotified = true;
} else if (distance > 15) {
dangerNotified = false;
}
// Force-stop
if (distance <= 10) {
if (tooCloseStart == 0) tooCloseStart = millis();
else if (millis() - tooCloseStart > 2000 && !forceStopped) {
forceStopped = true;
currentSpeed = 0;
targetSpeed = 0;
setMotors(0);
Blynk.logEvent("force_stop", "🛑 Object blocked >2s - Motor stopped");
}
} else {
tooCloseStart = 0;
forceStopped = false;
}
if (forceStopped) return;
```

```
// Auto speed
 if (distance > 40) targetSpeed = 255;
 else if (distance > 20) targetSpeed = map(distance, 21, 40, 100, 200);
 else if (distance > 10) targetSpeed = map(distance, 11, 20, 50, 100);
 else targetSpeed = 0;
 int prevSpeed = round(currentSpeed);
 currentSpeed = (targetSpeed == 0) ? 0 : currentSpeed + (targetSpeed - currentSpeed) *
smoothingFactor;
 int pwm = round(currentSpeed);
 setMotors(pwm);
 updateLED(pwm, prevSpeed); if (millis() - lastLCDUpdate >= lcdInterval) {
 updateLCD(pwm, distance);
Serial.print("Distance: "); Serial.print(distance);
Serial.print(" cm | Speed: "); Serial.println(pwm);
 lastLCDUpdate = millis();
 }
}
```