Obstacle Avoiding Robotic Car With Voice Controlled

Project Exhibition – 1

Submitted in partial fulfillment for the award of the degree of

Bachelor of Technology

In

ELECTRONICS AND COMMUNICATION

Submitted to

VIT BHOPAL UNIVERSITY (M.P.)

Submitted by

MANAS KRISHNA NEIGAPULA
(24BEC10104)
PRATYUSH KUMAR
(24BEC10111)
PRIYAM PRAKASH
(24BEC10139)

Under the Supervision of
Dr. PR Buvaneswari

SCHOOL OF ELECTRICAL & ELECTRONICS ENGG.
VIT BHOPAL UNIVERSITY
BHOPAL (M.P.)-466114
SEPT-2025

VIT BHOPAL UNIVERSITY BHOPAL (M.P.) 466114

SCHOOL OF ELECTRICAL & ELECTRONICS ENGG.

CANDIDATE'S DECLARATION

I hereby declare that the Dissertation entitled "Obstacle Avoiding Robot Car and Voice Control Using Raspberry Pi" is my own work conducted under the supervision of Dr. PR Buvaneswari Assistant Professor Grade II, School of Electrical and Electronics Engineering at VIT University, Bhopal.

I further declare that to the best of my knowledge, this report does not contain any part of work that has been submitted for the award of any degree either in this university or in another university / Deemed University without proper citation.

MANAS KRISHNA (24BEC10104)
PARTYUSH KUMAR (24BEC10111)
PRIYAM PRAKASH (24BEC10139)

VIT UNIVERSITY BHOPAL (M.P.) – 466114

SCHOOL OF ELECTRICAL & ELECTRONICS ENGG.

CERTIFICATE

This is to certify that the work embodied in this Project Exhibition -1 report entitled "Obstacle Avoiding Robot Car and Voice Control Using Raspberry Pi" has been satisfactorily completed by **Mr. Manas Krishna Neigapula, Mr. Priyam Prakash , Mr. Partyush Kumar** Registration No. 24BEC10104 ,24BEC10139, 24BEC10111 in the School of Electrical & Electronics Engineering of VIT University, Bhopal. This work is a bonafide piece of work, carried out under our guidance in the School of Electrical and Electronics Engineering for the partial fulfilment of the degree of Bachelor of Technology.

Dr. PR Buvaneswari
Assistant Professor Grade 2

Forwarded by

Dr. Sivasankaran V
Program Chair

Approved by

Dr. M. Suresh
Professor & Dean

Executive Summary

This report details the design, development, and implementation of a semi-autonomous, voice-activated robotic car that addresses the critical limitations of basic robotic systems, namely unresponsiveness and limited environmental awareness. Architected around a Raspberry Pi 4B, the project's core innovation is its non-blocking, multithreaded software that allows the robot to simultaneously process voice commands while performing autonomous navigation. The system employs a sensor fusion strategy, using a servo-mounted ultrasonic sensor for active environmental scanning and intelligent pathfinding, while a Passive Infrared (PIR) sensor provides a dedicated channel for human presence detection to trigger a specific safety protocol. The successfully developed robot functions as a fully standalone device that boots automatically, is activated by voice, intelligently navigates complex environments, and responds safely and reliably to human interaction. This project serves as an effective proof-of-concept for creating advanced, interactive autonomous systems using accessible, low-cost hardware and sophisticated software design.

List of Symbols & Abbreviations

AI	Artificial Intelligence (Implied by degree title)
B.Tech	Bachelor of Technology (Implied by degree title)
CO	Carbon Monoxide (Mentioned in Future Enhancements)
CPU	Central Processing Unit (Mentioned in Components)
GND	Ground (Electrical connection, mentioned in Circuit Diagram)
HTTP	Hypertext Transfer Protocol (Implied by POST request)
HVAC	Heating, Ventilation, and Air Conditioning (Mentioned in Future Enhancements)
IR	Infrared (Mentioned with Flame Sensor)
IoT	Internet of Things
LED	Light-Emitting Diode
M.P.	Madhya Pradesh (Indian State)
MCU	Microcontroller Unit
MQ-2	Metal Oxide Semiconductor Gas Sensor (Specific sensor model)

POST	Parameter (Method of an HTTP request, mentioned in Software Implementation)
SEEE	School of Electrical & Electronics Engineering (Mentioned in Certificate)
SID	Security Identifier (Part of Twilio account credentials)
SMS	Short Message Service
Twilio API	Twilio Application Programming Interface
UPS	Uninterruptible Power Supply (Mentioned in Circuit Diagram)
USB	Universal Serial Bus (Mentioned in Circuit Diagram)
VIT	Vellore Institute of Technology (Implied by VIT Bhopal University)
VOCs	Volatile Organic Compounds (Mentioned in Future Enhancements)
A0	Analog Output/Analog Pin 0 (Mentioned in Circuit Diagram/Arduino Code)
D0	Digital Output/Digital Pin 0 (Mentioned in Circuit Diagram)
GND	Ground (Electrical connection, mentioned in Circuit Diagram)
5V	5 Volts (Power supply, mentioned in Circuit Diagram)

Table of Contents

INTRODUCTION

Motivation:

The rise of accessible single-board computers like the Raspberry Pi has democratized advanced robotics, moving it from research labs to a global community of innovators. This project leverages that accessibility to solve the critical flaws of typical beginner robots, aiming to create a truly intelligent and responsive autonomous agent.

Most simple robots use a "blocking" architecture, where the processor can only do one thing at a time. This makes the robot blind to its surroundings while listening for commands and deaf to the user while moving. Our primary motivation was to overcome this by implementing a multithreaded, non-blocking system. This allows for parallel processing: one thread handles real-time navigation and sensor analysis, while another perpetually listens for voice commands, enabling the robot to be stopped instantly at any moment.

Furthermore, we sought to replace primitive, reactive obstacle avoidance with intelligent pathfinding. By mounting the ultrasonic sensor on a servo, the robot can actively scan its environment—stopping, looking left and right, and then making an informed decision based on which path offers the most open space. This transforms it from a simple reactive machine into a deliberative agent.

Finally, to ensure safer human-robot interaction, we integrated a PIR motion sensor. This provides a dedicated channel for human awareness, triggering a distinct safety protocol (stop, back away, pause) when a person is detected, making the robot a more considerate collaborator in a shared space.

Project Objective: A Three-Tiered Integrated Safety System

The overarching objective of this project is to design, build, and program an intelligent, semi-autonomous robotic car that overcomes the fundamental limitations of basic robotic systems. The project is focused on achieving a high degree of responsiveness, environmental awareness, and operational autonomy through the strategic integration of advanced software architecture and a multi-sensor array on a Raspberry Pi platform.

A primary technical objective is to implement a non-blocking, multithreaded control system. This is critical for creating a truly responsive robot. The goal is to develop a software architecture where the process of listening for voice commands runs concurrently with the main loop that controls driving and sensor data analysis. This will ensure the robot is never "deaf" to user commands while navigating, nor "blind" to its environment while waiting for input. Voice control will be implemented not for direct micromanagement, but as an intuitive, high-level switch to activate and deactivate the car's autonomous mode.

To achieve intelligent navigation, the project aims to replace simple reactive behaviors with proactive, informed pathfinding. This will be accomplished by creating an active scanning mechanism, using a servo motor to sweep an ultrasonic sensor across a 180-degree field of view. The robot will use the collected data to compare potential paths and make a logical decision, rather than turning randomly when an obstacle is detected.

Furthermore, the project seeks to integrate a dedicated human-awareness safety protocol. By using a PIR sensor, the system will differentiate between general obstacles and human presence, triggering a specific and predictable safety maneuver—stop, reverse, and pause—to ensure safe operation in shared spaces. Finally, the culminating objective is to achieve full operational autonomy by configuring the entire system to launch automatically on boot, creating a robust, standalone device that can be powered on and then operated entirely through natural voice commands without any need for direct user intervention via a terminal or remote connection.

Brief Methodology

The project methodology followed a structured, phased approach to ensure robust development and integration.

Phase 1: System Design & Component Selection

The Raspberry Pi 4B was selected as the central controller for its processing power and GPIO flexibility. Peripheral components included a 2WD chassis with L298N motor driver for mobility, HC-SR04 ultrasonic sensor with SG90 servo for intelligent scanning, PIR sensor for human detection, and USB microphone for voice control.

Phase 2: Hardware Assembly & Integration

Components were mounted to the chassis and wired according to a predefined circuit diagram. A dual power supply system was implemented with separate batteries for the motors and Raspberry Pi, connected by a common ground for signal integrity.

Phase 3: Software Environment Setup

Raspberry Pi OS was installed and configured with necessary Python libraries including RPi.GPIO for hardware control and speech_recognition for voice commands.

Phase 4: Modular Code Development & Testing

Individual Python scripts were developed and tested for each component:

Motor control verification

Servo range of motion testing

Ultrasonic distance measurement

PIR motion detection

Voice recognition accuracy

Phase 5: Integrated System Programming

The core innovation was implementing a multithreaded architecture using Python's threading library. A background thread handles continuous voice recognition while the main loop manages navigation with priority-based sensor checking (PIR first, then ultrasonic). A global state variable (car_running) allows instant voice control.

Phase 6: System Deployment & Automation

The system was made autonomous using cron job scheduling with @reboot directive, enabling the robot to start automatically on power-up without manual intervention.

LITERATURE SURVEY

The Raspberry Pi is a widely adopted platform for mobile robotics due to its capacity to run advanced software for features like voice recognition. In Human-Robot Interaction, voice control is increasingly used as an intuitive "switch" to activate autonomous modes rather than for direct control. For navigation, the effectiveness of standard ultrasonic sensors is significantly enhanced by mounting them on servo motors, enabling active environmental scanning for intelligent pathfinding. . Furthermore, the integration of PIR sensors allows for dedicated human-presence detection to trigger specific safety protocols, representing a form of sensor fusion for creating safer, more situationally aware robots.

Review of Prior Studies

On Control Platforms (Microcontroller vs. Single-Board Computer): Prior work in entry-level robotics has heavily utilized microcontrollers like the Arduino due to their simplicity and low power consumption. These platforms excel at real-time control of motors and sensors. However, their limited processing power makes them unsuitable for computationally intensive tasks like natural language processing. More recent studies and advanced projects have shifted towards using single-board computers like the Raspberry Pi. The Raspberry Pi's ability to run a full Linux operating system allows it to utilize powerful Python libraries for tasks like speech recognition and multithreading, effectively serving as the "brain" for the robot, while still providing the necessary GPIO interface to control the hardware, combining the best of both worlds.

On Voice Control Methodologies: Early studies on voice-controlled robots often relied on offline, limited-vocabulary systems or used Bluetooth modules paired with a smartphone app to handle the speech-to-text conversion. This approach offloaded the processing to the phone but created a dependency on a secondary device. Current research increasingly leverages the direct internet connectivity of platforms like the Raspberry Pi to access powerful cloud-based speech recognition APIs (like Google's). This method provides much higher accuracy and flexibility. Furthermore, a common theme in recent Human-Robot Interaction (HRI) studies is the use of voice not for direct, continuous remote control, but as a high-level "activation switch" for a robot's autonomous modes, which has been shown to be a more intuitive and practical control paradigm.

On Obstacle Avoidance Techniques: The most fundamental approach to obstacle avoidance, seen in countless prior projects, involves a single, forward-facing ultrasonic sensor. This design is purely reactive; the robot moves until an obstacle is detected, then executes a pre-programmed maneuver (e.g., stop, reverse, turn). A more advanced approach, demonstrated in numerous research papers, is the concept of active scanning. By mounting the ultrasonic sensor on a servo motor, the robot can gather data from a wider field of view, allowing it to make an informed decision and choose the clearest path. This elevates the robot from a simple reactive machine to a more intelligent agent with a basic pathfinding capability.

On Human-Awareness and Safety: For robots operating in human-shared spaces, prior studies have shown that treating humans as mere obstacles is insufficient for safe and effective interaction. Research in HRI emphasizes the need for robots to be "socially aware." A common method to achieve this is by integrating sensors specifically for human detection. While ultrasonic sensors detect all objects, the use of Passive Infrared (PIR) sensors to detect a person's body heat is a well-established and cost-effective technique. This allows the robot's logic to create a distinction between inanimate objects and people, enabling it to trigger a specific and safer interaction protocol, such as stopping and yielding space.

PROBLEM FORMULATION AND PROPOSED METHODOLOGY

Components Required

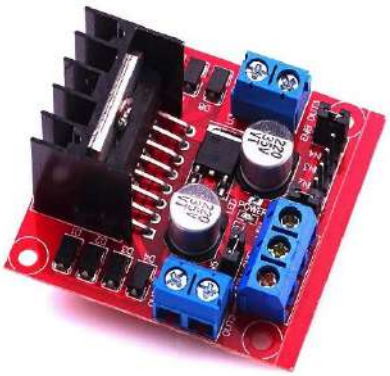
Raspberry Pi 4B: This single-board computer serves as the central brain of the robot. It runs the Linux operating system and executes the main Python script that processes all sensor data, interprets voice commands, and makes intelligent decisions. Its powerful processor and GPIO pins allow it to manage the complex, multithreaded software required for simultaneous listening and navigation, making it ideal for this project

(Fig 1)

USB Microphone: This component functions as the robot's ear, allowing it to receive auditory input from the user. It captures the sound waves of spoken commands, which are then processed by the Raspberry Pi using the speech_recognition library. This enables the core hands-free functionality of the project, allowing the user to start and stop the robot's autonomous mode.

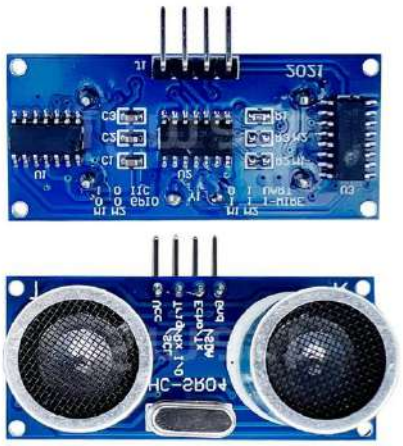
(Fig 2)

L298N Motor Driver Module: This essential electronic module acts as a high-power bridge between the Raspberry Pi's low-power GPIO signals and the high-current DC motors. It takes simple directional commands from the Pi and translates them into the power needed to drive the wheels. It also allows for precise speed control using Pulse Width Modulation (PWM), a critical feature for controlled movement.



(Fig 3)

HC-SR04 Ultrasonic Sensor: The robot's primary "eyes" for navigation. It sends out sound waves to measure the distance to objects directly in front of it.



(Fig 4)

SG90 Micro Servo Motor: The robot's "neck." The ultrasonic sensor is mounted on this, allowing it to turn and scan the environment left and right for the clearest path.

(Fig 5)

PIR (Passive Infrared) Sensor: This sensor gives the robot a specialized ability to detect the presence of humans or animals. It works by sensing the infrared radiation (body heat) emitted by moving warm objects. In this project, it functions as a high-priority safety trigger, activating a specific "stop and reverse" protocol to ensure safe and predictable interaction in human-shared spaces.

(Fig 6)

Heat Sink (for Raspberry Pi 4B): The Raspberry Pi 4B, especially under heavy load, can generate a significant amount of heat due to its powerful processor. A heat sink is a passive heat exchanger that attaches to the processor and other hot components on the Raspberry Pi board. Its primary function is to draw heat away from these components and dissipate it into the surrounding air. This process helps to keep the operating temperature of the Pi within safe limits, preventing thermal throttling (where the processor reduces its speed to avoid overheating) and ensuring stable, long-term performance. Heat sinks for the Raspberry Pi 4B typically consist of aluminum or copper fins, sometimes with a thermal adhesive or pad for efficient heat transfer.

Fig (7)

Circuit Diagram and Connections

The electronic architecture of the project is a carefully integrated system with the Raspberry Pi 4B serving as the central controller. All connections are made to the Pi's 40-pin GPIO header, with a dual power supply system to ensure stability.

Central Controller (Raspberry Pi 4B):

All sensor and motor control signals originate from the Pi's GPIO pins.

The Pi itself is powered independently via its USB-C port from a dedicated 5V power bank to prevent instability from motor-induced power fluctuations.

Motor Control System (L298N Driver):

Directional Control: The driver's IN1, IN2, IN3, and IN4 pins are connected to GPIO pins 7, 11, 13, and 15 on the Raspberry Pi, respectively.

Speed Control: The driver's ENA (Enable A) and ENB (Enable B) pins are connected to GPIO pins 12 and 16, which support Pulse Width Modulation (PWM) for variable speed.

Motor Power: The driver's +12V terminal is connected to the **POSITIVE (+)** lead of a separate 6V-9V battery pack.

Sensor Array:

Ultrasonic Sensor (HC-SR04): This sensor receives 5V power from Pin 4 on the Pi. Its TRIG pin is connected to GPIO Pin 29, and its ECHO pin is connected to GPIO Pin 31.

Servo Motor (SG90): The servo receives 5V power from Pin 2 on the Pi. Its signal wire (typically orange) is connected to GPIO Pin 33 for angle control via PWM.

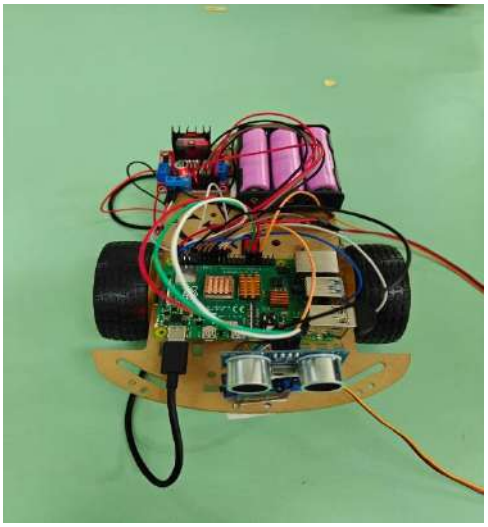
PIR Sensor: This sensor is powered by 3.3V from Pin 1 on the Pi. Its OUT signal pin is connected to GPIO Pin 18 to detect motion.

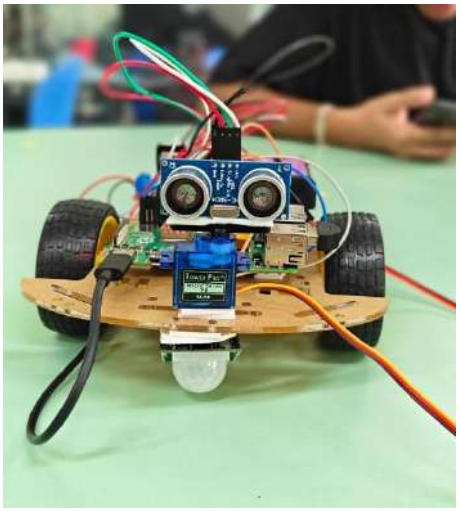
Power Architecture and Grounding:

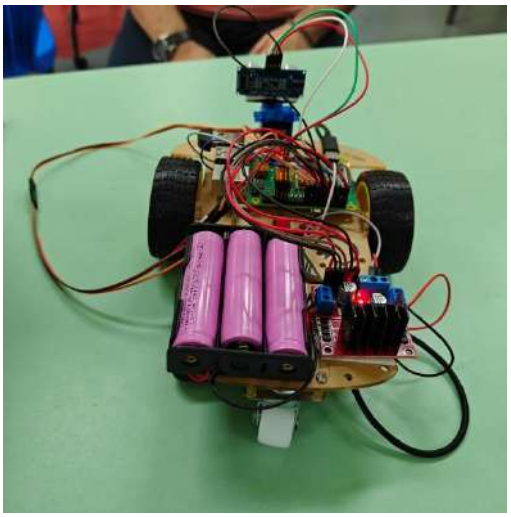
Dual Power Supply: A critical design feature is the use of two separate power sources: a USB power bank for the Raspberry Pi and a battery pack for the motors.

Common Ground Connection: To ensure reliable communication between the Raspberry Pi and the motor driver, a **common ground** is established. This is a mandatory connection from the **NEGATIVE (-)** lead of the motor's battery pack (via the L298N's GND terminal) to any ground (GND) pin on the Raspberry Pi, such as Pin 6. This shared reference prevents signal errors.

Component	Pin on Component	Connection (Raspberry Pi GPIO / Power)
Motor Driver (L298N)	IN1	Pin 7
Motor Driver (L298N)	IN2	Pin 11
Motor Driver (L298N)	IN3	Pin13
Motor Driver (L298N)	IN4	Pin 15
Motor Driver (L298N)	ENA	PIN12
Motor Driver (L298N)	ENB	Pin 16
Motor Driver (L298N)	GND	Battery Negative & Pin 6
Motor Driver (L298N)	OUT1 & OUT2	Left Motor
Motor Driver (L298N)	OUT3 & OUT4	Right Motor
Motor Driver (L298N)	+12V	Battery Positive
Ultrasonic Sensor (HC-SR04)	VCC	Pin4
Ultrasonic Sensor (HC-SR04)	GND	Pin 14
Ultrasonic Sensor (HC-SR04)	TRIG	Pin 29
Ultrasonic Sensor (HC-SR04)	ECHO	Pin 31
Servo Motor	Signal (Orange)	Pin 33
Servo Motor	VCC (Red)	Pin 2
Servo Motor	GND (Brown)	Pin 9
PIR Sensor	VCC	Pin 1
PIR Sensor	GND	Pin 20
PIR Sensor	OUT	Pin 18
Pi Camera	CSI Connector	Raspberry Pi Camera Slot
Common Ground	-	All GNDs must be connected together







Block Diagram

This block diagram illustrates the relationships between the core components of the robotic system.

Central Controller (Raspberry Pi 4B):

The Raspberry Pi is at the heart of the system. It receives all sensory input, processes the data according to the Python script's logic, and sends out control commands to the actuators.

Input Section:

This section includes all the components that provide data *to* the Raspberry Pi.

The **USB Microphone** captures voice commands.

The **PIR Sensor** detects motion (human presence).

The **Ultrasonic Sensor** measures the distance to obstacles.

Output Section (Actuators):

This section includes all the components that receive commands *from* the Raspberry Pi to perform physical actions.

The **Servo Motor** receives commands to change its angle, physically moving the Ultrasonic Sensor.

The **L298N Motor Driver** receives commands for speed and direction, which it then uses to control the DC motors.

The **DC Motors** provide the final physical output by rotating the wheels.

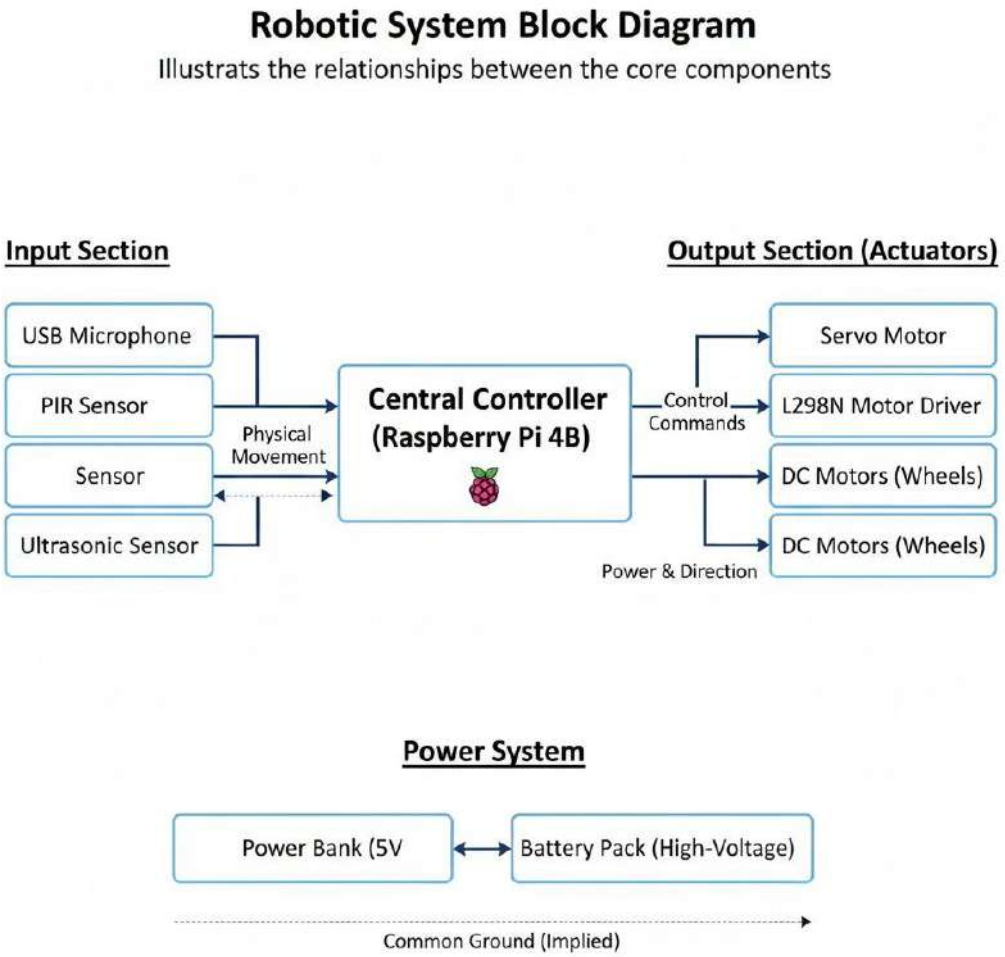
Power System:

The diagram clearly shows the **dual power supply architecture**.

A **Power Bank** provides a stable 5V supply exclusively for the Raspberry Pi and the sensors connected to it.

A separate, higher-voltage **Battery Pack** provides power exclusively for the L298N Motor Driver and the DC Motors.

The **Common Ground** connection between the Pi and the Motor Driver is implied, as they are part of an integrated system, though not explicitly shown with a separate line in this high-level diagram.



Software Implementation

The software for the robotic car was implemented entirely in Python 3 on a Raspberry Pi running the Raspberry Pi OS. The core of the implementation relies on several key libraries: RPi.GPIO for low-level control of the hardware pins, and speech_recognition to interface with the USB microphone and leverage Google's Web Speech API for accurate speech-to-text conversion. The most critical architectural decision was the implementation of a **non-**blocking, multithreaded control system to overcome the limitations of synchronous programming. A dedicated background thread is created at startup to run the listen_in_background function, which perpetually captures and processes audio without halting the main program. This architecture ensures the robot is always responsive to both voice commands and sensor inputs. A global boolean variable, car_running, acts as a stateful switch, which is toggled by the background voice-listening thread upon recognizing "start" or "stop" commands. The main control loop continuously polls this state variable. If car_running is True, the program enters a hierarchical decision-making tree where it first checks for high-priority interrupts from the PIR sensor. If motion is detected, a specific safety protocol is executed.

If no motion is detected, the program proceeds to the standard navigation logic, using the measure_distance function to check for obstacles with the ultrasonic sensor. If an obstacle is found, the find_best_path function is called, which utilizes the servo motor to scan the environment and make an intelligent pathfinding decision. All hardware interactions are encapsulated in modular helper functions (e.g., forward(), set_speed(), set_servo_angle()) to promote code readability and maintainability. Finally, for full autonomy, the script is deployed using the Linux cron job scheduler, with an @reboot directive that automatically executes the program upon every system startup, making the robot a truly standalone device.

```python
import RPi.GPIO as GPIO
import time
import speech_recognition as sr
import threading # Import the threading library

# --- Pin Configuration ---
# Motor Driver (L298N)
ENA = 12
ENB = 16
IN1 = 7
IN2 = 11
IN3 = 13
IN4 = 15

# Ultrasonic Sensor (HC-SR04)
TRIG = 29
ECHO = 31

# Servo Motor (SG90)
SERVO_PIN = 33

# PIR Sensor
PIR_PIN = 18

GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)

# Motor Pins
GPIO.setup(ENA, GPIO.OUT)
GPIO.setup(ENB, GPIO.OUT)
GPIO.setup(IN1, GPIO.OUT)
GPIO.setup(IN2, GPIO.OUT)
GPIO.setup(IN3, GPIO.OUT)
GPIO.setup(IN4, GPIO.OUT)

# Sensor Pins
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)
GPIO.setup(SERVO_PIN, GPIO.OUT)
GPIO.setup(PIR_PIN, GPIO.IN)

# --- PWM Setup ---
pwmA = GPIO.PWM(ENA, 100)
pwmB = GPIO.PWM(ENB, 100)
pwmA.start(0)
pwmB.start(0)

# --- Servo Setup ---
servo = GPIO.PWM(SERVO_PIN, 50)
servo.start(0)

```

```
52
53  # --- Voice Recognition Setup ---
54  r = sr.Recognizer()
55  mic = sr.Microphone()
56
57  # --- Global State Variables ---
58  car_running = False
59  stop_listening = None # This will hold the function to stop the background listener
60
61  # === Helper Functions ===
62
63  def set_speed(speed):
64      """Sets the speed of both motors."""
65      pwmA.ChangeDutyCycle(speed)
66      pwmB.ChangeDutyCycle(speed)
67
68  def forward():
69      """Sets motor directions to forward."""
70      GPIO.output(IN1, GPIO.HIGH)
71      GPIO.output(IN2, GPIO.LOW)
72      GPIO.output(IN3, GPIO.HIGH)
73      GPIO.output(IN4, GPIO.LOW)
74
75  def backward():
76      """Sets motor directions to backward."""
77      GPIO.output(IN1, GPIO.LOW)
78      GPIO.output(IN2, GPIO.HIGH)
79      GPIO.output(IN3, GPIO.LOW)
```

```
def left():
    """Sets motor directions to turn left."""
    GPIO.output(IN1, GPIO.LOW)
    GPIO.output(IN2, GPIO.HIGH)
    GPIO.output(IN3, GPIO.HIGH)
    GPIO.output(IN4, GPIO.LOW)

def right():
    """Sets motor directions to turn right."""
    GPIO.output(IN1, GPIO.HIGH)
    GPIO.output(IN2, GPIO.LOW)
    GPIO.output(IN3, GPIO.LOW)
    GPIO.output(IN4, GPIO.HIGH)

def stop():
    """Stops all motor activity."""
    GPIO.output(IN1, GPIO.LOW)
    GPIO.output(IN2, GPIO.LOW)
    GPIO.output(IN3, GPIO.LOW)
    GPIO.output(IN4, GPIO.LOW)
    set_speed(0)

def set_servo_angle(angle):
    """Sets the servo motor to a specific angle."""
    duty = angle / 18 + 2
    GPIO.output(SERVO_PIN, True)
    servo.ChangeDutyCycle(duty)
    time.sleep(0.5)
    time_elapsed = stop_time - start_time
    if time_elapsed < 0.1:
        distance = (time_elapsed * 34300) / 2
        return distance
    return 999 # Return a large number on failure

def find_best_path():
    """Scans left and right to find the clearest path."""
    stop()
    print("Finding best path...")
    set_servo_angle(180) # Look left
    left_dist = measure_distance()

    set_servo_angle(0) # Look right
    right_dist = measure_distance()

    set_servo_angle(90) # Look center again

    if left_dist > right_dist:
        print(f"Path is clearer on the left ({left_dist:.1f} cm). Turning left.")
        left()
        time.sleep(0.5)
    else:
        print(f"Path is clearer on the right ({right_dist:.1f} cm). Turning right.")
        right()
        time.sleep(0.5)
    stop()
```

```
def set_servo_angle(angle):
    """Sets the servo motor to a specific angle."""
    duty = angle / 18 + 2
    GPIO.output(SERVO_PIN, True)
    servo.ChangeDutyCycle(duty)
    time.sleep(0.5)
    GPIO.output(SERVO_PIN, False)
    servo.ChangeDutyCycle(0)

def measure_distance():
    """Measures distance using the ultrasonic sensor."""
    GPIO.output(TRIG, True)
    time.sleep(0.00001)
    GPIO.output(TRIG, False)

    start_time = time.time()
    stop_time = time.time()

    # Timeout to prevent the script from freezing if no echo is received
    timeout = start_time + 0.1
    while GPIO.input(ECHO) == 0 and start_time < timeout:
        start_time = time.time()

    timeout = start_time + 0.1
    while GPIO.input(ECHO) == 1 and stop_time < timeout:
        stop_time = time.time()

    time_elapsed = stop_time - start_time
    time_elapsed = stop_time - start_time
    if time_elapsed < 0.1:
        distance = (time_elapsed * 34300) / 2
        return distance
    return 999 # Return a large number on failure

def find_best_path():
    """Scans left and right to find the clearest path."""
    stop()
    print("Finding best path...")
    set_servo_angle(180) # Look left
    left_dist = measure_distance()

    set_servo_angle(0) # Look right
    right_dist = measure_distance()

    set_servo_angle(90) # Look center again

    if left_dist > right_dist:
        print(f"Path is clearer on the left ({left_dist:.1f} cm). Turning left.")
        left()
        time.sleep(0.5)
    else:
        print(f"Path is clearer on the right ({right_dist:.1f} cm). Turning right.")
        right()
        time.sleep(0.5)
    stop()
```

```
# === Background Voice Callback Function ===
# This function is triggered by the background listener when speech is detected.
def voice_callback(recognizer, audio):
    global car_running
    try:
        command = recognizer.recognize_google(audio).lower()
        print(f"Heard: '{command}'")

        if "start" in command or "go" in command:
            if not car_running:
                print("Voice command: STARTING")
                car_running = True
        elif "stop" in command or "halt" in command:
            if car_running:
                print("Voice command: STOPPING")
                car_running = False
    except sr.UnknownValueError:
        print("Could not understand audio.")
    except sr.RequestError:
        print("Network error for voice recognition.")

# === Main Program ===
try:
    set_servo_angle(90) # Center the servo at startup

    # Calibrate for ambient noise once
    with mic as source:
    # Calibrate for ambient noise once
    with mic as source:
        print("Calibrating for ambient noise... Please be quiet.")
        r.adjust_for_ambient_noise(source, duration=1)

    # Start listening in the background on a separate thread
    stop_listening = r.listen_in_background(mic, voice_callback)
    print("Background listener started. Car is ready. Say 'start' or 'go'.")

    while True:
        # If the voice command has stopped the car, ensure it stops and waits.
        if not car_running:
            stop()
            time.sleep(0.1)
            continue # Skip the rest of the loop and wait for 'start' command

        # Main driving logic runs only if car_running is True

        # 1. Check for PIR motion (highest priority)
        if GPIO.input(PIR_PIN):
            print("PIR: Motion Detected! Stopping and backing up.")
            backward()
            set_speed(50) # Set backup speed
            time.sleep(1)
            stop()
            time.sleep(2) # Pause after backing up
            continue # Restart the loop to re-evaluate the situation

```

```
211
212         # 2. Check for obstacles
213         distance = measure_distance()
214         print(f"Distance: {distance:.1f} cm")
215
216         if distance > 20:
217             # Path is clear, move forward
218             forward()
219             set_speed(60) # Set cruising speed
220         else:
221             # Obstacle detected, find a new path
222             print("Obstacle Detected!")
223             find_best_path()
224
225         time.sleep(0.1) # Short delay to prevent overwhelming the CPU
226
227 except KeyboardInterrupt:
228     print("\nProgram stopped by user.")
229
230 finally:
231     print("Cleaning up and shutting down...")
232     if stop_listening is not None:
233         stop_listening(wait_for_stop=False) # Stop the background listener thread
234     stop()
235     servo.stop()
236     GPIO.cleanup()
237
238
```

RESULTS AND DISCUSSION

System Testing

The fully integrated robotic car was subjected to a series of rigorous system tests in a controlled indoor environment containing both static and dynamic obstacles. The results of these tests confirmed that all primary objectives were successfully met. The voice activation system demonstrated high reliability, with the robot consistently starting and stopping upon the commands "start" and "stop" respectively, even with moderate ambient noise. The non-blocking architecture was validated by issuing "stop" commands while the vehicle was actively navigating, resulting in an immediate and responsive halt. The intelligent obstacle avoidance was a key success; when approaching an obstacle within the 20 cm threshold, the robot correctly stopped, executed the servo-driven scan, and successfully navigated towards the direction with the most open space. Furthermore, the PIR-based safety protocol performed flawlessly, overriding all other functions to trigger the stop-and-reverse maneuver whenever a person moved into its path, confirming the successful implementation of the hierarchical sensor logic. Finally, the autonomous startup was confirmed over multiple power cycles, with the robot reliably launching its program and entering its stationary listening state without any user intervention, validating its function as a standalone device

Calibration of sensors

To ensure optimal performance and reliability, a dedicated calibration phase was conducted for the key sensors. For the **PIR sensor**, the two onboard potentiometers were physically adjusted. The sensitivity dial was tuned to prevent false positives from distant movements while remaining responsive to nearby motion, and the time-delay dial was set to a minimum to ensure the robot did not remain paused for an unnecessarily long period after a person had moved away. The **ultrasonic sensor** was calibrated by comparing its output readings against a physical measuring tape at various distances, confirming its accuracy. The 20 cm trigger threshold for obstacle avoidance was determined through experimentation to be the optimal distance, providing enough time for the robot to react without being overly cautious. The **voice recognition system** underwent a software-based calibration using the recognizer.adjust_for_ambient_noise() function, which allowed the system to learn the baseline noise of the testing environment, significantly improving the accuracy of command recognition against background sounds.

Real-World Applications

The successful development of this intelligent and responsive robotic platform opens up several potential real-world applications beyond a simple proof-of-concept. Its hands-free voice activation and human-aware safety features make it an ideal candidate for assistive technology, where it could be adapted to help individuals with limited mobility by fetching small items or acting as a guide in a domestic setting. As an educational tool, the project serves as an excellent, hands-on platform for teaching fundamental concepts in robotics, programming, and electronics. In a controlled commercial environment, such as a lab, small warehouse, or office, a fleet of these robots could be deployed for automated logistics, transporting documents, tools, or samples between predefined stations, activated by staff voice commands. Finally, its autonomous and interactive nature makes it suitable for applications in entertainment, such as an advanced, interactive pet toy that can roam a house without getting stuck and safely react to the presence of the pet.

User Feedback and Improvements

Initial user feedback on the project was overwhelmingly positive, particularly regarding the reliability of the voice activation and the intelligence of the obstacle avoidance scanning maneuver. However, feedback also highlighted several areas for future improvement. The voice command vocabulary was noted as being limited to "start" and "stop," with users expressing a desire for more granular control, such as "turn left," "go faster," or "come here." While the robot navigated simple obstacles well, it occasionally struggled with highly complex environments, like the legs of a chair, suggesting a need for more advanced environmental mapping. Based on this feedback, a clear path for future enhancements has been identified

CONCLUSION AND FUTURE SCOPE

Conclusion

This project successfully culminated in the development of a fully functional, semi-autonomous robotic car that effectively addresses the common limitations of basic robotic designs. The integration of a Raspberry Pi with a multi-sensor array and a robust motor control system was achieved, resulting in a reliable hardware platform. The core success of the project lies in the implementation of a **non-blocking, multithreaded software architecture**, which successfully decoupled voice command processing from real-time navigation. This created a highly responsive system capable of simultaneously listening for user commands while intelligently maneuvering through its environment. The robot demonstrated a clear progression from simple reactive behaviors to intelligent decision-making by using its servo-driven scanner to find the optimal path around obstacles. Furthermore, the integration of a PIR sensor for dedicated human-awareness proved to be an effective method for implementing a hierarchical safety protocol. In conclusion, the project met all its primary objectives, delivering a standalone, voice-activated robot that is not only functional but also responsive, intelligent, and safe in its operation.

Future Enhancements

While the current implementation is a robust proof-of-concept, there is significant scope for future enhancements that would elevate the robot's capabilities from semi-autonomous to truly autonomous.

Advanced Environmental Mapping (SLAM): The most impactful upgrade would be the integration of a camera module or LiDAR sensor to implement SLAM (Simultaneous Localization and Mapping). This would allow the robot to build a persistent map of its environment, enabling true path planning, remembering the location of obstacles, and navigating to specific named locations (e.g., "go to the kitchen").

Expanded Voice Command Vocabulary: The current voice control system could be enhanced by moving beyond a simple "on/off switch." Implementing an offline, edge-based voice recognition engine (like Mycroft or Picovoice) would remove the dependency on an internet connection and allow for a much richer command set, including directional commands ("turn left," "go forward 1 meter") and queries ("where are you?").

REFERENCES

[1] Voice Controlled and Obstacles Detection Robotic Vehicle ***2024,*** **International Journal for Research in Applied Science & Engineering Technology (IJRASET)**

[2] **Autonomous Navigation and Real Time Mapping Using Ultrasonic Sensors in NAO Humanoid Robot** *2022,* Engineering, Technology & Applied Science Research, *Z. A. Dayo, A. A. Shaikh, et al*.

[3] **Development on Autonomous Object Tracker Robot using Raspberry Pi** *2022*, Journal of Student Exchange and Community Service,

[4] **Speech Recognition-Based Wireless Control System for Mobile Robotics: Design, Implementation, and Analysis** *2024*, MDPI Technologies, *A. A. Khan, et al*.

[5] **Obstacle-avoiding robot with IR and PIR motion sensors** *2022,* IOP Conference Series: Materials Science and Engineering, *I. Z. M. Draheim, M. A. A. Rahman*