Auto Parking

Abstract—**Development of an Automatic Parking System for a Four-Motor DC Car Integrated with ESP32 and Ultrasonic Sensors**

This project introduces an Automatic Parking System (APS) tailored for a Direct Current (DC) car featuring a unique configuration of four motors, grouped into pairs and controlled by motor drivers, an ESP32 microcontroller, and two ultrasonic sensors. The design includes an ultrasonic sensor positioned at the rear of the vehicle and another sensor mounted on a servo motor located at the car's front.

The APS is engineered to enable seamless autonomous parking maneuvers by leveraging the embedded control systems and sensor technologies. The four motors, divided into pairs, are orchestrated through dedicated motor drivers, facilitating precise and synchronized movement control. The ESP32 microcontroller serves as the central processing unit, managing sensor data integration, decision-making algorithms, and coordination of motor actions for parking operations.

The dual ultrasonic sensor setup enhances the system's perception capabilities, with one sensor strategically positioned at the rear of the car to gauge proximity to obstacles during reversing maneuvers. The second sensor, mounted on a servo motor at the car's front, provides a wide-ranging view, allowing dynamic adjustment of the sensing angle to detect parking spaces and obstacles in the vehicle's vicinity.

The project's primary objective is to develop an efficient and reliable autonomous parking system tailored for DC cars, emphasizing synchronized motor control, sensor fusion, and intelligent decision-making. This APS aims to enhance parking precision and safety, reduce driver intervention, and contribute to the evolution of smart and automated transportation systems.

This endeavor signifies a step forward in the implementation of sophisticated parking assistance systems, leveraging a combination of motor control, ESP32 processing capabilities, and dual ultrasonic sensor deployment to enable precise and autonomous parking maneuvers for DC cars.

Keywords: Automatic Parking System, DC Car, Motor Control, ESP32, Ultrasonic Sensors, Autonomous Driving, Intelligent Transportation Systems.

I. Introduction

The Autonomous Toy Car Parking project, designed as an instructive and fun venture, uses the ESP32 microcontroller to instil autonomy in a small-scale toy car. The project, which aims to inspire curiosity and creativity, teaches users to the principles of robotics, programming, and the potential uses of autonomous systems.

1. Key Objectives:

Autonomous Navigation:

The ESP32 microcontroller, equipped with infrared sensors, guides the toy car through a simulated parking lot. The car autonomously explores its surroundings, identifying obstacles and determining optimal parking spots.

Precise Parking Movements:

Coordinated movements by the ESP32-controlled motors enable the toy car to execute precise parking manoeuvres. This not only adds a layer of realism to play but also introduces users to the concept of precision control in robotics.

User Interaction through Mobile App:

An accompanying mobile application enhances user experience. Users can initiate autonomous parking sequences, monitor the toy car's activities, and even customize parking scenarios through the intuitive app interface.

Educational Exploration:

Beyond play, the project serves as an educational tool, introducing users to the basics of programming, sensor integration, and the principles of autonomous navigation. It encourages hands-on learning and sparks interest in STEM-related fields from an early age.

2. Technical Aspects:

This report provides a detailed exploration of the technical components involved in the project. It delves into the ESP32's programming logic, the integration of infrared sensors for obstacle detection, and the orchestration of motor movements to achieve seamless autonomous parking.

Societal Impact:

While designed for play, the Autonomous Toy Car Parking project has broader implications for educational robotics. It encourages the development of problem-solving skills, logical thinking, and introduces users to the exciting world of robotics and automation, laying the foundation for future technological exploration.

In conclusion, this project goes beyond traditional toys, offering a glimpse into the potential of autonomous systems even at a miniature scale. Through playful exploration, it opens doors to learning and creativity, showcasing the impact of technology in transforming even the simplest of toys into autonomous wonders.

II. BACKGROUND

The hardware components of an automatic parking system for cars can vary depending on the specific implementation and complexity of the system. We use this components:

A. *Microcontroller:*

A central processing unit is required to process data from sensors, cameras, and other sources. It makes decisions based on this information to control the vehicle's movements.

We use microcontroller module called ESP32:

The ESP32 is a powerful microcontroller module that combines both Wi-Fi and Bluetooth capabilities, making it a versatile and popular choice for a wide range of embedded applications. The ESP32 was designed to address the limitations of the ESP8266 and provide enhanced features, making it suitable for more complex and demanding projects.

ESP32 features:

Dual-Core Processor:

The ESP32 is equipped with a dual-core Tensilica LX6 processor, allowing for efficient multitasking and improved performance. This feature is particularly beneficial for applications that require real-time processing.

Wireless Connectivity:

Wi-Fi: The ESP32 supports 2.4 GHz Wi-Fi, making it capable of connecting to local networks and the internet. This feature is crucial for IoT (Internet of Things) applications, where devices need to communicate and exchange data over a network.

Bluetooth: In addition to Wi-Fi, the ESP32 comes with Bluetooth support. This allows for seamless integration with other Bluetooth-enabled devices, making it suitable for applications such as home automation, wearables, and more.

Low Power Consumption:

The ESP32 is designed with power efficiency in mind, offering various low-power modes to conserve energy. This makes it suitable for battery-powered devices and applications where power consumption is a critical factor.

Rich Set of Peripherals:

The ESP32 provides a wide range of peripherals, including GPIO (General Purpose Input/Output) pins, I2C, SPI, UART, and more. This flexibility allows developers to interface with various sensors, actuators, and external devices.

Integrated Security Features:

Security is a significant concern in connected devices, and the ESP32 addresses this by providing features such as secure boot, flash encryption, and hardware-accelerated cryptographic functions.

Open-Source Development:

The ESP32 benefits from a vibrant and active open-source community. Espressif Systems provides a comprehensive development framework called ESP-IDF (Espressif IoT Development Framework), which facilitates the development of applications for the ESP32 using C or C++.

Affordability and Availability:

The ESP32 is cost-effective, making it an attractive choice for both hobbyists and professionals. Its widespread popularity has led to the availability of a variety of development boards and modules from different manufacturers.

B. *Actuators:*

1. ELECTRIC MOTORS:

Motors are used to control the movement of the vehicle, including steering, acceleration, and braking.

We use electrical motor called Direct Current Motor:

DC motors are electrical machines that convert electrical energy into mechanical energy through the interaction of magnetic fields. They operate on the principle of electromagnetic induction and are widely used in various applications due to their simplicity, reliability, and ease of control.

Components of DC Motors:

Stator:

The stator is the stationary part of the motor that generates a magnetic field when electric current flows through it. In most DC motors, the stator consists of coils of wire wound around a core, creating an electromagnet.

Rotator:

The rotor is the rotating part of the motor. It is usually located inside the stator and contains a winding or a set of permanent magnets. The interaction between the stator's magnetic field and the rotor's magnetic elements produces the motor's rotational motion.

Commutator (for Brushed DC Motors):

Brushed DC motors have a commutator, which is a rotary switch that reverses the direction of current flow in the rotor winding. It ensures that the torque produced by the motor is always in the same direction, resulting in continuous rotation.

Brushes (for Brushed DC Motors):

Brushes are conductive components that maintain electrical contact with the rotating commutator. They supply current to the rotor winding, allowing the motor to operate.

Brushless DC Motors (BLDC):

In contrast to brushed DC motors, brushless DC motors use electronic controllers (usually called motor controllers or ESCs) and have permanent magnets on the rotor. The need for brushes and a commutator is eliminated, resulting in reduced maintenance and improved efficiency.

Advantages of DC Motors:

Simple Control:

DC motors are relatively easy to control, allowing for precise speed and direction adjustments.

High Torque at Low Speeds:

DC motors can provide high torque at low speeds, making them suitable for applications requiring both power and control.

Efficiency:

Brushless DC motors, in particular, are known for their high efficiency and reliability due to the absence of brushes, which reduces friction and wear.

Cost-Effectiveness:

DC motors are often cost-effective and widely available, making them a popular choice for various applications.

2. **Servo Motors:**

A servo motor is a type of motor that is designed for precise control of its angular position. Servo motors are commonly used in robotics, remote control systems, and various other applications where precise movement is required. A micro servo is a smaller version of a standard servo motor, often used in applications where size and weight are critical factors.

We use servo motor called Micro Servo MG90 0:180 Degree:

Components of Micro Servo MG90 0:180 Degree:

Motor:

The core of the servo, the motor generates the mechanical power needed for the servo to move. In a Micro Servo MG90, you can expect a small-sized DC motor.

Gear Train:

Servo motors often incorporate a gear train to reduce the high-speed, low-torque output of the motor to a slower, high-torque output at the servo's shaft. The gear train determines the precision and strength of the servo's movements.

Control Circuit:

The control circuit interprets the incoming signals (typically PWM signals) and adjusts the position of the servo motor accordingly. It is responsible for maintaining the desired position and adjusting the motor's speed and direction.

Potentiometer (Position Feedback Device):

Most servo motors, including those in the Micro Servo MG90, use a potentiometer for position feedback. The potentiometer provides information about the current position of the motor shaft to the control circuit. This feedback loop allows the servo to adjust its position accurately.

Amplifier or Driver:

The amplifier or driver circuit amplifies the control signal received by the servo's control circuit to the necessary levels to drive the motor. It ensures that the motor receives the appropriate power to move to the desired position.

Output Shaft:

The output shaft is the part of the servo that rotates and performs the mechanical work. It is usually connected to the load (such as a robotic arm or a steering mechanism) and moves to the specified position based on the control input.

Housing:

The housing encases and protects the internal components of the servo motor. It is designed to be compact, durable, and often includes mounting points for easy integration into various applications.

C. *Sensors:*

Ultrasonic Sensors:
These sensors are used to detect the distance between the car and obstacles. They can provide information about the surroundings and help the car navigate in tight spaces without collisions.
Components of Ultrasonic Sensors:

Transducer:

The transducer is a crucial component that generates and emits ultrasonic waves. In ultrasonic sensors, piezoelectric crystals are commonly used as transducers. When an electric voltage is applied to these crystals, they vibrate and produce ultrasonic waves.

Ultrasonic Waves:

Ultrasonic waves are sound waves with a frequency above the upper limit of human hearing, typically above 20 kHz. Ultrasonic sensors emit these waves, and the sensor's operation is based on measuring the time it takes for the waves to travel to an object and back.

Diaphragm (Receiver):

The sensor's receiver, often a diaphragm or another transducer, detects the ultrasonic waves reflected back from objects in the sensor's field of view.

Time-of-Flight Measurement System:

Ultrasonic sensors operate on the principle of time-of-flight measurement. The sensor measures the time it takes for the ultrasonic waves to travel from the transmitter to the target object and back to the receiver.

Signal Processing Circuitry:

The received signal is processed by internal circuitry, which calculates the time delay between the emitted and received signals. This time delay is used to determine the distance to the object.

Output Interface:

Ultrasonic sensors typically have an output interface to provide distance information to external devices or microcontrollers. Common output types include analog voltage, digital pulse width modulation (PWM), or serial communication.

D. *Power Supply:*

We use 4 rechargeable batteries. The nominal voltage of the battery is 3.7V. This is the average voltage that the battery operates at during most of its discharge cycle.
The total nominal voltage of 4 batteries is 12V.

E. *Motor Driver:*

Motor driver plays a critical role in controlling the movement of motors that drive various components such as wheels or steering mechanisms. The motor driver is an electronic device that interprets signals from a control system and provides the necessary power and direction to the motors.

Role of motor driver:

Motor Control:

The primary function of a motor driver is to control the speed, direction, and rotation of the motors in the car. This includes precise control over the movements required for parking, such as forward and backward motions, as well as steering.

Interface with Control System:

The motor driver acts as an interface between the control system (which may include sensors, cameras, and algorithms for autonomous parking) and the physical motors. It interprets the commands received from the control system and converts them into electrical signals that drive the motors.

Power Amplification:

Motor drivers amplify the low-power control signals from the microcontroller or control system to levels sufficient to drive the high-power motors. This ensures that the motors receive the necessary power to carry out the required movements.

Direction Control:

The motor driver controls the direction of rotation of the motors. In the context of auto parking, this involves precise control of the wheels or steering mechanism to navigate the car into a parking space.

Speed Control:

Motor drivers enable the control system to adjust the speed of the motors. This is crucial for maintaining safe and accurate movements during the parking process.

III. PROPOSED IDEA

The auto parking project is a comprehensive system that achieves autonomous parking by utilizing four motors, ultrasonic sensors, and an ESP microcontroller. The car's efficient maneuvering through parking scenarios is made possible by the seamless integration of hardware components and meticulous control functions.

The system workflow is organized in the following manner:

Forward Movement:

The car advances until the back ultrasonic sensor detects a potential obstacle at a 30 cm threshold.

Right Rotation:

Upon reaching the threshold, the car initiates a right rotation to scan for available parking slots.

Slot Detection:

The servo-mounted ultrasonic sensor rotates 90 degrees to identify an open parking slot.

Forward Movement to Parking Slot:

The car proceeds forward until no obstacles are detected or the ultrasonic sensor reads 4 cm, indicating the presence of a wall or the end of the parking slot.

Control functions govern the car's movements:

Forward:

Initiates forward movement.

Backward:

Executes backward movement.

Left:

Causes a left turn.

Right:

Initiates a right turn.

Controlling Motor Speed Using Enable:

In addition to the core functionalities, the project includes the ability to control motor speed through the use of the enables of the driver [EnA,EnB]. The 'enable' function allows for fine-tuning of motor speeds, giving you more options for adjusting the car's movement dynamics.
This improvement allows for more nuanced control of the vehicle, particularly during complex parking manoeuvres. The project can achieve smoother accelerations and decelerations by adjusting the 'speed' variable and analogWrite function, which contributes to the overall precision of the auto parking mechanism.

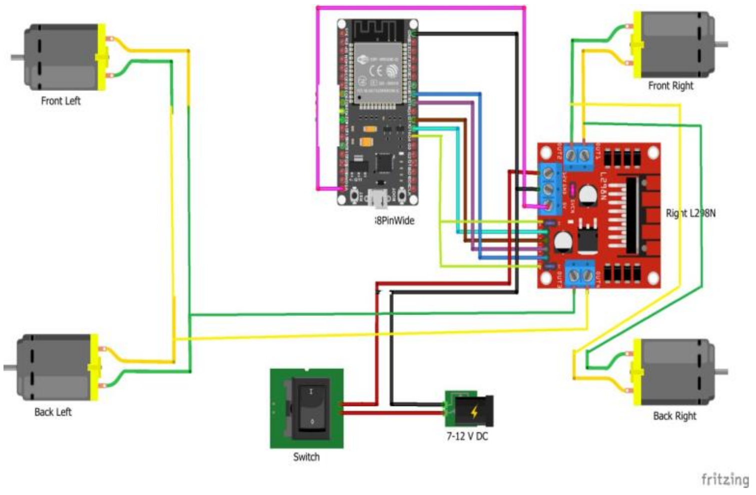
A Specific Stop function is activated upon slot detection, halting all movements to ensure a safe parking process.

The ESP microcontroller acts as the central processing unit, interpreting ultrasonic sensor data and issuing precise commands to motors and the servo. It plays a crucial role in decision-making, orchestrating movements based on real-time distance readings.

Sensor readings provide continuous distance measurements, and the ESP interprets these readings, making decisions based on predefined thresholds. For instance, a 30 cm threshold for obstacle detection and a 4 cm threshold for parking slot detection.
Accounting for environmental factors such as varying lighting conditions and irregular obstacles is one of the project's challenges and considerations. Ultrasonic sensor calibration is critical for accurate measurements, and precise motor movements are required to avoid collisions and facilitate accurate parking.

Finally, the auto parking mechanism project successfully integrates motors, ultrasonic sensors, and an ESP microcontroller to develop an autonomous parking solution. The detailed workflow, control functions, and decision-making processes of the project lay the groundwork for future advances in autonomous vehicle technologies.

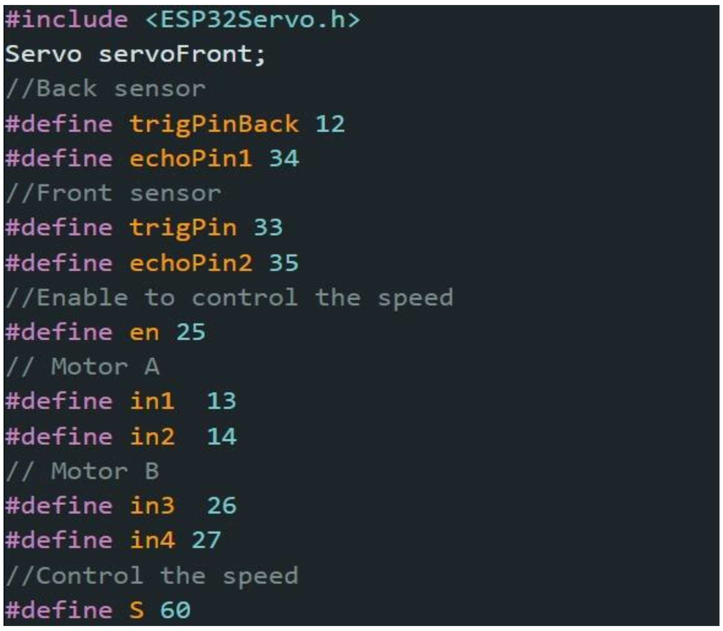
Simulation of the hardware:



IV. SOFTWARE

This Arduino code is designed for an ESP32-based robot equipped with ultrasonic sensors and a servo motor to perform autonomous parking. The robot uses these components to navigate and find a parking slot. Below is a summary and detailed explanation of the key components of the code:

1. Setup and Global Definitions

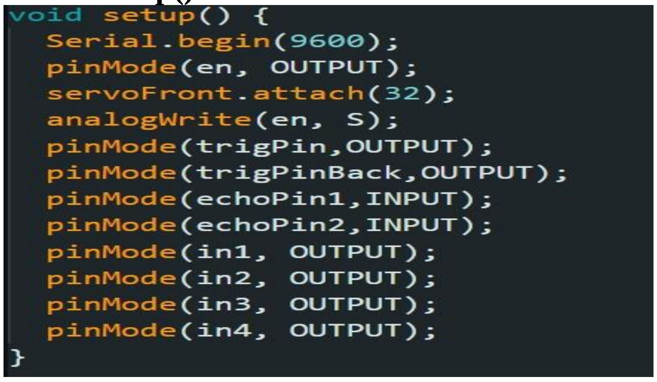


```
#include <ESP32Servo.h>
Servo servoFront;
//Back sensor
#define trigPinBack 12
#define echoPin1 34
//Front sensor
#define trigPin 33
#define echoPin2 35
//Enable to control the speed
#define en 25
// Motor A
#define in1  13
#define in2  14
// Motor B
#define in3  26
#define in4 27
//Control the speed
#define S 60
```

Servo and Pin Definitions:
The script begins by including the ESP32Servo library and defining pins for ultrasonic sensors, motors, and the servo motor. The Servo object servoFront is attached to pin 32.
Motor Control Pins:
Pins for controlling the motors (in1, in2, in3, in4) and the enable pin (en) for speed control are defined.

Speed Constant (S):
The speed for motor movement is set to 60.

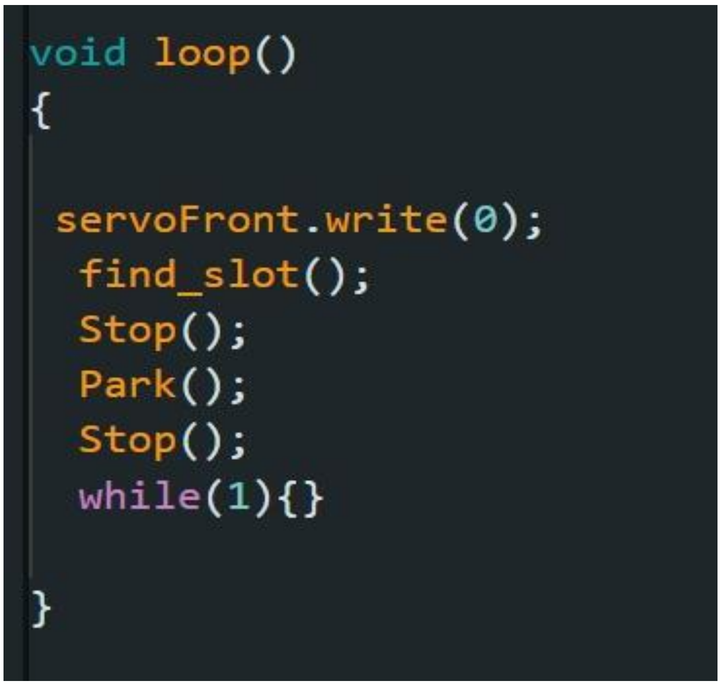
2. setup() Function



```
void setup() {
  Serial.begin(9600);
  pinMode(en, OUTPUT);
  servoFront.attach(32);
  analogWrite(en, S);
  pinMode(trigPin,OUTPUT);
  pinMode(trigPinBack,OUTPUT);
  pinMode(echoPin1,INPUT);
  pinMode(echoPin2,INPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);
}
```

Initializations:
Serial communication starts at 9600 baud, and pins are set as input or output as required.
Servo and Motor Initialization:
The servo is attached, and the motor speed is set using analogWrite().

3. loop() Function

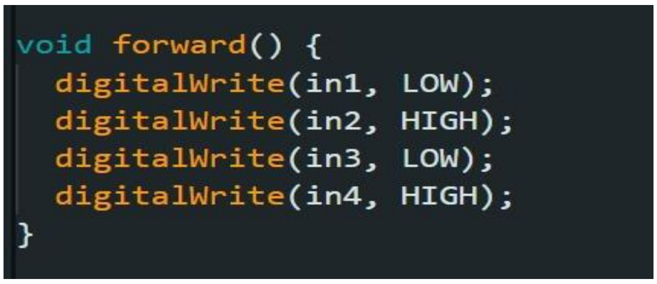


```
void loop()
{

 servoFront.write(0);
  find_slot();
  Stop();
  Park();
  Stop();
  while(1){}

}
```

The main loop sets the servo position, calls the find_slot() function to locate a parking space, executes the Park() function to park, and then enters an infinite loop to stop further actions.

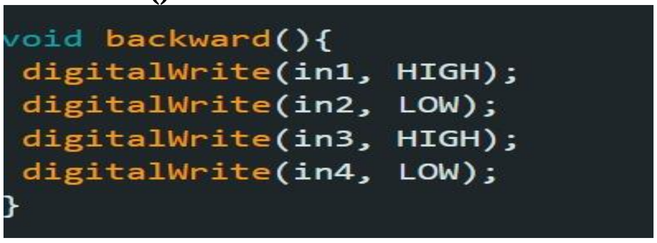
4. Motor Control Functions

forward()



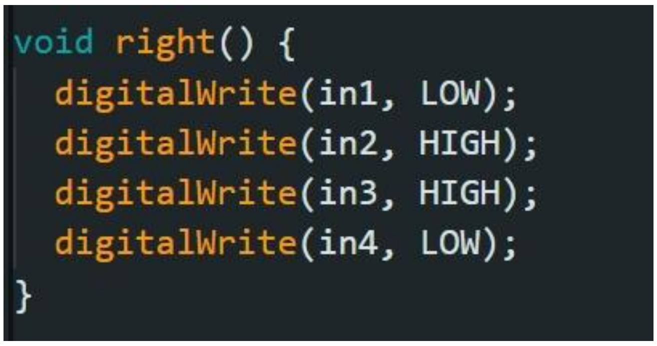
```
void forward() {
  digitalWrite(in1, LOW);
  digitalWrite(in2, HIGH);
  digitalWrite(in3, LOW);
  digitalWrite(in4, HIGH);
}
```

backward()



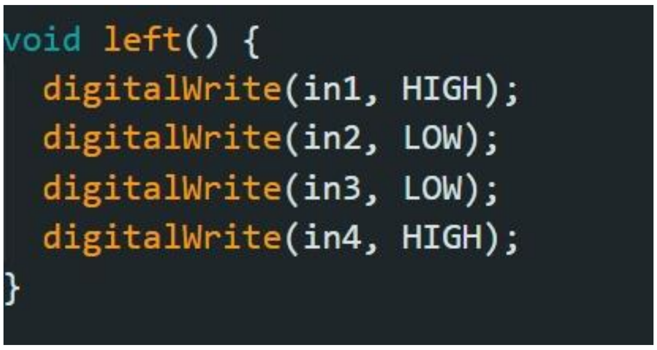
```
void backward(){
 digitalWrite(in1, HIGH);
 digitalWrite(in2, LOW);
 digitalWrite(in3, HIGH);
 digitalWrite(in4, LOW);
}
```

right()



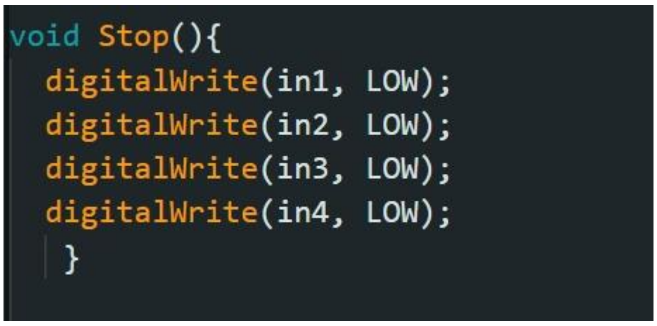
```
void right() {
  digitalWrite(in1, LOW);
  digitalWrite(in2, HIGH);
  digitalWrite(in3, HIGH);
  digitalWrite(in4, LOW);
}
```

left()



```
void left() {
  digitalWrite(in1, HIGH);
  digitalWrite(in2, LOW);
  digitalWrite(in3, LOW);
  digitalWrite(in4, HIGH);
}
```

Stop()

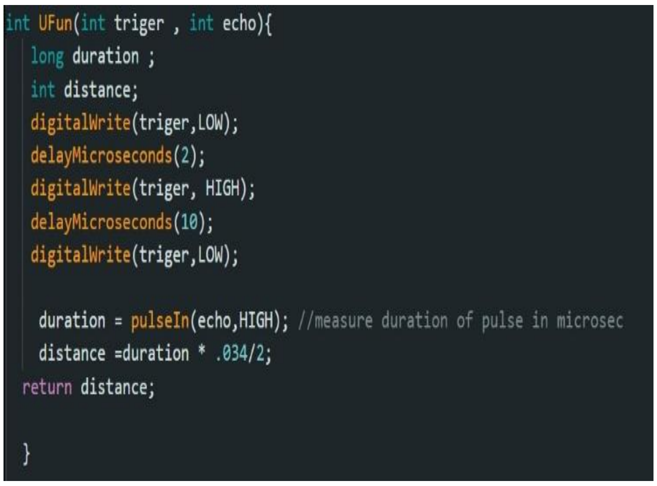


```
void Stop(){
  digitalWrite(in1, LOW);
  digitalWrite(in2, LOW);
  digitalWrite(in3, LOW);
  digitalWrite(in4, LOW);
  }
```

These functions control the motors for moving forward, backward, turning right, turning left, and stopping the robot.

Ultrasonic Sensor Function:

UFun()



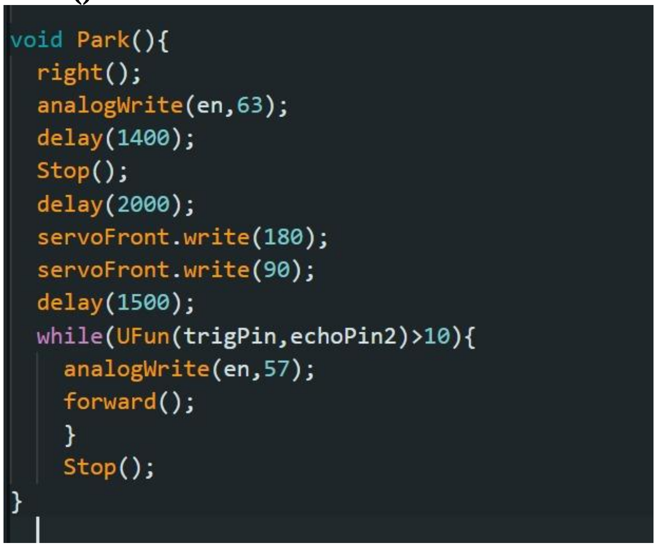
```
int UFun(int triger , int echo){
  long duration ;
  int distance;
  digitalWrite(triger,LOW);
  delayMicroseconds(2);
  digitalWrite(triger, HIGH);
  delayMicroseconds(10);
  digitalWrite(triger,LOW);

   duration = pulseIn(echo,HIGH); //measure duration of pulse in microsec
   distance =duration * .034/2;
 return distance;

 }
```

Distance Measurement:

This function takes trigger and echo pins as arguments and calculates the distance by measuring the time taken for an ultrasonic pulse to return.

5. Parking Functions

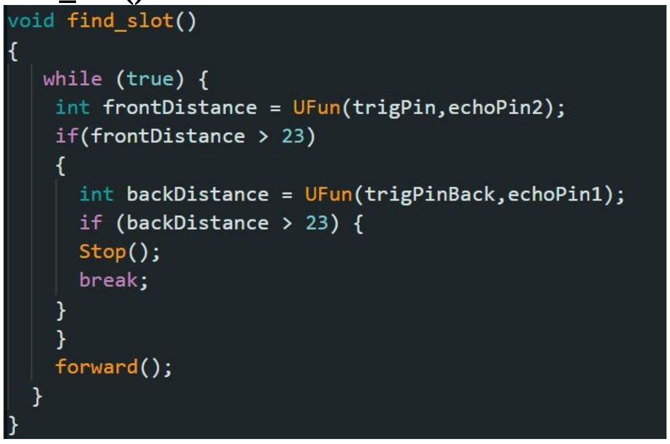
Park():



```
void Park(){
  right();
  analogWrite(en,63);
  delay(1400);
  Stop();
  delay(2000);
  servoFront.write(180);
  servoFront.write(90);
  delay(1500);
  while(UFun(trigPin,echoPin2)>10){
    analogWrite(en,57);
    forward();
    }
    Stop();
}
```

Controls the robot to execute a parking maneuver using the right turn, adjusting the servo, and moving forward until a certain distance is maintained from an obstacle.

find_slot():



```
void find_slot()
{
   while (true) {
    int frontDistance = UFun(trigPin,echoPin2);
    if(frontDistance > 23)
    {
      int backDistance = UFun(trigPinBack,echoPin1);
      if (backDistance > 23) {
      Stop();
      break;
    }
    }
    forward();
  }
}
```

The robot moves forward until it finds a space that is large enough to park (determined by a threshold distance from obstacles both in front and at the back).

Summary

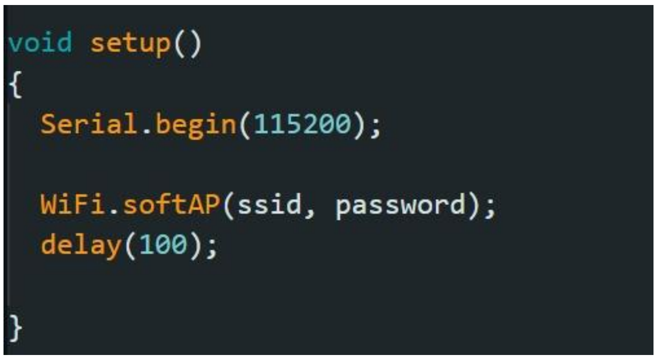
The script is designed for an autonomous parking robot. It uses ultrasonic sensors for distance measurement to detect parking spaces and obstacles. The robot can move in different directions and park itself autonomously in a detected space. The use of functions for each movement and action makes the code modular and easier to understand. However, the script lacks error handling or contingencies for unexpected scenarios, which might be necessary for a robust real-world application.

V. WI-FI CODE

This code is for an ESP32-based Wi-Fi controlled device, possibly a robotic vehicle .It includes functionality for both creating a Wi-Fi access point and handling HTTP requests for remote control, although the HTTP server part is currently commented out.

Key Components of the Code:

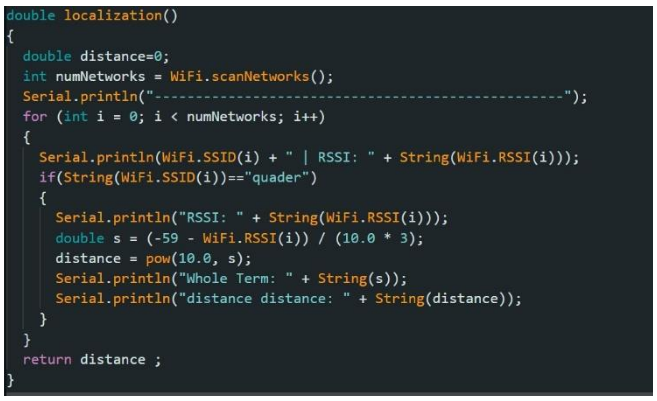
1. **WiFi Setup:**



```
void setup()
{
  Serial.begin(115200);

  WiFi.softAP(ssid, password);
  delay(100);

}
```

The ESP32 is configured to create its own WiFi access point with SSID car_2 and password 12345678.The commented-out section includes code for setting a static IP address for the device but it's not activated in the current setup.

2. **Localization Function:**



```
double localization()
{
  double distance=0;
  int numNetworks = WiFi.scanNetworks();
  Serial.println("----------------------------------------------");
  for (int i = 0; i < numNetworks; i++)
  {
    Serial.println(WiFi.SSID(i) + " | RSSI: " + String(WiFi.RSSI(i)));
    if(String(WiFi.SSID(i))=="quader")
    {
      Serial.println("RSSI: " + String(WiFi.RSSI(i)));
      double s = (-59 - WiFi.RSSI(i)) / (10.0 * 3);
      distance = pow(10.0, s);
      Serial.println("Whole Term: " + String(s));
      Serial.println("distance distance: " + String(distance));
    }
  }
  return distance ;
}
```

The localization function scans for nearby WiFi networks and calculates the distance from a specific network named "quader".It uses the RSSI value to estimate the distance based on a logarithmic path-loss model.The formula used assumes a reference RSSI (signal strength at one meter distance) of -59 dBm and an environmental factor of 3. These values are typical for indoor environments but might need adjustment for accuracy.

Serial Communication:

The code heavily uses Serial.println for debugging and indicating control commands in the console.

Summary:

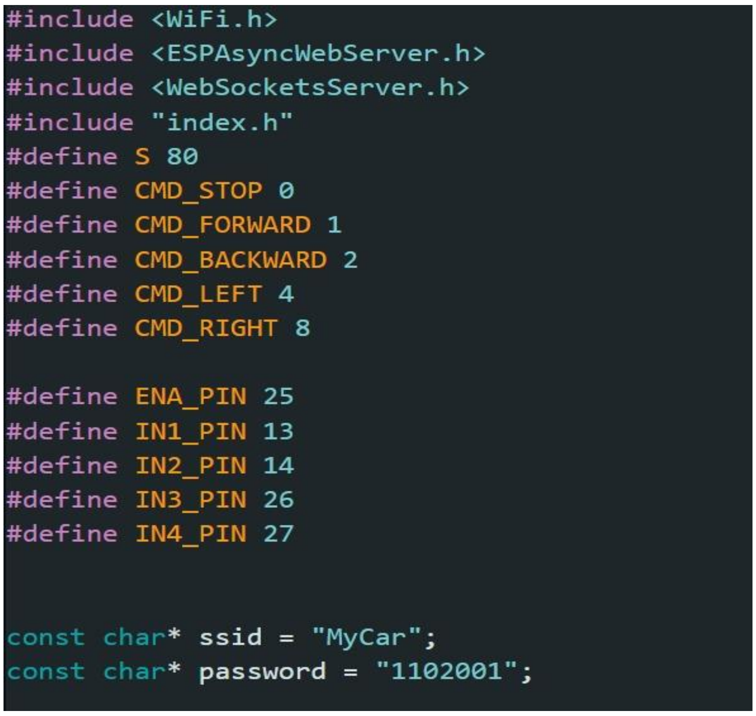
The program sets up an ESP32 as a WiFi access point. It has capabilities (currently commented out) to act as a web server for remote control.The control commands are designed for directional movement but are not implemented beyond serial print statements.

It includes a function to estimate distance to a specific WiFi network using RSSI values, which can be used for rudimentary localization.

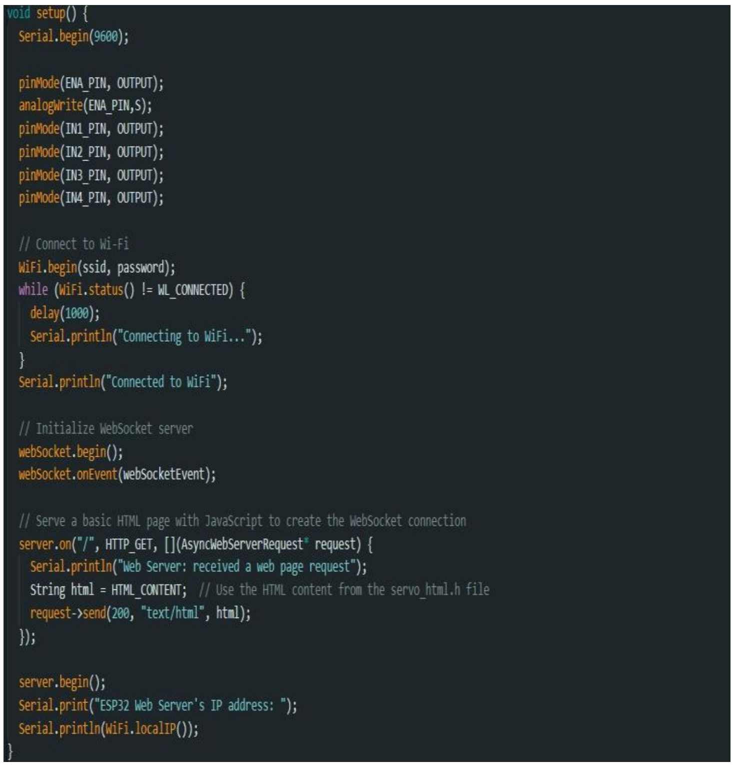
The localization function can be used as is for estimating the distance to the "quader" WiFi network.

VI. HTML CODE TO CONTROL THE ROBOT

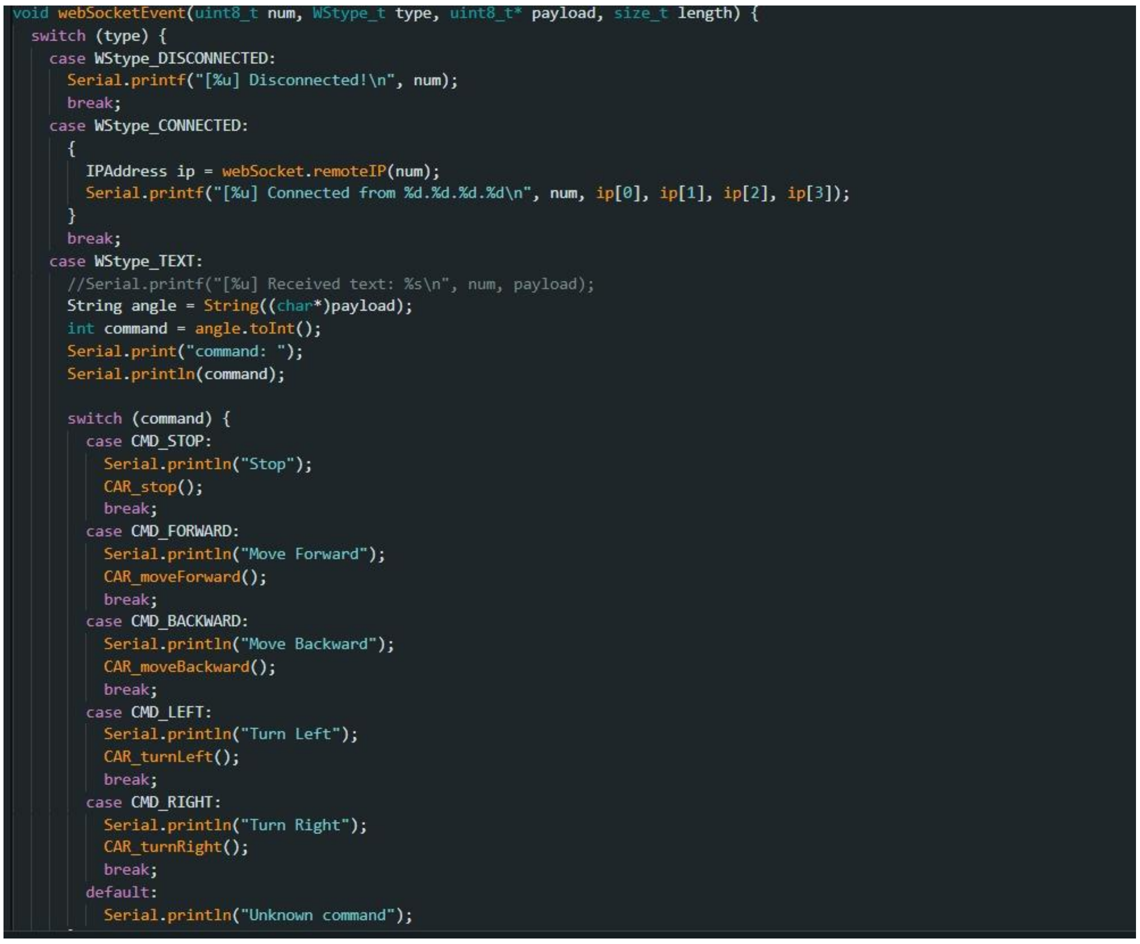
This code is designed for an ESP32 microcontroller to create a Wi-Fi controlled remote car. The ESP32 serves a web page over Wi-Fi, allowing a user to control the car through a browser interface. Here's a detailed breakdown of the program:


```
#include <WiFi.h>
#include <ESPAsyncWebServer.h>
#include <WebSocketsServer.h>
#include "index.h"
#define S 80
#define CMD_STOP 0
#define CMD_FORWARD 1
#define CMD_BACKWARD 2
#define CMD_LEFT 4
#define CMD_RIGHT 8

#define ENA_PIN 25
#define IN1_PIN 13
#define IN2_PIN 14
#define IN3_PIN 26
#define IN4_PIN 27


const char* ssid = "MyCar";
const char* password = "1102001";
```

Library Inclusions:
The code includes necessary libraries for Wi-Fi and server functionality:
WiFi.h:
For Wi-Fi connection.
ESPAsyncWebServer.h:
For handling HTTP requests asynchronously.
WebSocketsServer.h:
For WebSocket communication, enabling real-time bi-directional communication between the server (ESP32) and the client (web browser).
Constants and Pin Definitions:
Defines constants for motor control and WebSocket commands (CMD_STOP, CMD_FORWARD, etc.).
Pin assignments for controlling the car's motors.
Wi-Fi Credentials:
ssid and password variables for connecting to a Wi-Fi network.
Server and WebSocket Initialization:
An AsyncWebServer and WebSocketsServer are instantiated for handling HTTP and WebSocket connections.
WebSocket Event Handling (webSocketEvent function):
Manages different WebSocket events like connection, disconnection, and receiving text.
Parses the received commands and calls corresponding functions to control the car (CAR_moveForward, CAR_moveBackward, etc.).
Setup Function (setup):


```
void setup() {
  Serial.begin(9600);

  pinMode(ENA_PIN, OUTPUT);
  analogWrite(ENA_PIN,S);
  pinMode(IN1_PIN, OUTPUT);
  pinMode(IN2_PIN, OUTPUT);
  pinMode(IN3_PIN, OUTPUT);
  pinMode(IN4_PIN, OUTPUT);

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");

  // Initialize WebSocket server
  webSocket.begin();
  webSocket.onEvent(webSocketEvent);

  // Serve a basic HTML page with JavaScript to create the WebSocket connection
  server.on("/", HTTP_GET, [](AsyncWebServerRequest* request) {
    Serial.println("Web Server: received a web page request");
    String html = HTML_CONTENT;  // Use the HTML content from the servo_html.h file
    request->send(200, "text/html", html);
  });

  server.begin();
  Serial.print("ESP32 Web Server's IP address: ");
  Serial.println(WiFi.localIP());
}
```

Initializes serial communication and sets motor control pins.
Connects to the Wi-Fi network.
Initializes the WebSocket and sets up the event handler.
Configures the web server to serve the HTML content when the root URL ("/") is accessed.
Main Loop (loop function):
Continuously checks for WebSocket events.
The placeholder comment indicates where additional, user-defined code can be added.
Car Control Functions:
Functions like CAR_moveForward, CAR_moveBackward, etc., control the car's motors by setting the digital pins to HIGH or LOW.



```
void webSocketEvent(uint8_t num, WStype_t type, uint8_t* payload, size_t length) {
  switch (type) {
    case WStype_DISCONNECTED:
      Serial.printf("[%u] Disconnected!\n", num);
      break;
    case WStype_CONNECTED:
      {
        IPAddress ip = webSocket.remoteIP(num);
        Serial.printf("[%u] Connected from %d.%d.%d.%d\n", num, ip[0], ip[1], ip[2], ip[3]);
      }
      break;
    case WStype_TEXT:
      //Serial.printf("[%u] Received text: %s\n", num, payload);
      String angle = String((char*)payload);
      int command = angle.toInt();
      Serial.print("command: ");
      Serial.println(command);

      switch (command) {
        case CMD_STOP:
          Serial.println("Stop");
          CAR_stop();
          break;
        case CMD_FORWARD:
          Serial.println("Move Forward");
          CAR_moveForward();
          break;
        case CMD_BACKWARD:
          Serial.println("Move Backward");
          CAR_moveBackward();
          break;
        case CMD_LEFT:
          Serial.println("Turn Left");
          CAR_turnLeft();
          break;
        case CMD_RIGHT:
          Serial.println("Turn Right");
          CAR_turnRight();
          break;
        default:
          Serial.println("Unknown command");
```

HTML Content (HTML_CONTENT):
A multiline string literal containing the HTML, CSS, and JavaScript for the car control web interface.
JavaScript handles button events, sends commands to the ESP32 via WebSocket, and updates the UI based on user interactions.

Web Interface:
Provides a simple control panel with buttons for moving the car forward, backward, left, right, and stopping it.
Uses WebSocket to send real-time control commands to the ESP32.
Automatically tries to connect to the WebSocket server upon loading.
Responsiveness and Styling:
The web interface is styled for usability and responsiveness, making it suitable for both desktop and mobile browsers.

In summary, this program turns an ESP32 into a Wi-Fi-controlled car server. It handles WebSocket connections for real-time control, serves a web-based control interface, and interprets received commands to physically maneuver the car. The use of asynchronous server and WebSocket ensures that the ESP32 can handle multiple connections and commands without significant latency.