|  |  |
| --- | --- |
| **Name** | **ID** |
| Nabil Sherif Nabil Ibrahim | 20200583 |
| Lilian Stephanos Younan | 20200404 |
| Aly Eyad Aly | 20200331 |
| John Gamil Samir | 20200132 |
| Mohamed Nasser Yosry | 20200480 |

**Team Members (Group 10)**

**(Auto Parking)**

**Abstract**

This report details the design, development, and successful implementation of a dynamic auto parking system for a small-scale vehicle. The system integrates an ESP32 microcontroller, L298 DC motor driver, ultrasonic sensors, and servo motors, allowing the vehicle to autonomously navigate and park in dynamic environments.

The ESP32 microcontroller processes real-time data from ultrasonic sensors, executes precise control algorithms, and communicates with the L298 motor driver and servo motors, achieving highly accurate parking manoeuvres. Notably, the system also features remote control functionality, enabling manual intervention when necessary.

The experimental results demonstrate the versatility of the design in dynamic scenarios, showcasing its reliability, adaptability, and the added convenience of remote control. The combination of the ESP32 microcontroller and L298 motor driver presents a cost-effective and scalable solution for enhancing parking automation, with potential applications in intelligent transportation systems.

**Background**

1. **ESP32 (Wifi + Bluetooth):**

The ESP32 is a highly versatile microcontroller module renowned for its integrated Wi-Fi and Bluetooth capabilities. This compact yet powerful module is extensively used in embedded systems and Internet of Things (IoT) applications. Its key features include a dual-core processor, enabling efficient multitasking, and low power consumption, making it suitable for a range of energy-efficient devices. With a plethora of I/O interfaces, the ESP32 provides flexibility for connecting to various sensors, actuators, and displays.

Its seamless integration with Wi-Fi enables internet connectivity, while the built-in Bluetooth functionality facilitates wireless communication with other devices. Popular for its adaptability, the ESP32 finds applications in smart home devices, IoT sensors, industrial automation, and more. The support of an active community ensures a wealth of resources, making it accessible for developers of different skill levels. In summary, the ESP32 is a robust microcontroller offering a combination of Wi-Fi and Bluetooth capabilities, making it a preferred choice for diverse electronic projects.

1. **L298N to Motor driver:**

The L298N is a dual H-bridge motor driver widely used in robotics and electronics. With the ability to control two DC motors independently, it allows users to determine the direction and speed of each motor.

Its versatility and ease of use make it popular in projects such as small robots, RC cars, and automation systems. The L298N accepts control inputs from microcontrollers and can handle moderate to high voltage and current levels.

Its built-in diodes protect against voltage spikes, and it can be used in parallel for additional motor control or increased current capacity.

1. **Ultra sonic sensor:**

Ultrasonic sensors are devices that use high-frequency sound waves (ultrasonic waves) for distance measurement and object detection. They consist of a transducer that emits ultrasonic waves and detects their reflections. The time taken for the waves to bounce back is measured to calculate the distance to an object.

These sensors find widespread use in applications such as obstacle avoidance in robotics, liquid level measurement, and proximity sensing in industrial automation. Ultrasonic sensors are valued for their accuracy, versatility, and ease of integration with microcontrollers, making them essential in various fields including automation, robotics, and automotive systems.

1. **Breadboard 400 points:**

The Breadboard 400 points is a versatile prototyping tool in electronics, featuring a grid of 400 tie points arranged in rows and columns. It allows for temporary circuit construction without soldering, enabling easy experimentation and testing of electronic components. With power rails, bus strips, and reusable design, this breadboard is suitable for small to moderately complex projects.

It serves as an educational tool, commonly used in learning electronics, and is widely employed by engineers and hobbyists for quick and temporary circuit assembly during the prototyping phase.

1. **Battery Case Holder 2 cells 18650:**

The "Battery Case Holder 2 cells 18650" is a compact and durable device designed to hold two 18650 lithium-ion batteries in series. It features positive and negative terminals, allowing for easy electrical connections. Commonly used in DIY projects, this holder is often employed to create portable power banks or rechargeable battery packs.

Its versatility, standardized design, and compatibility with widely available 18650 batteries make it a popular choice for various electronic applications requiring a reliable and rechargeable power source.

1. **ON/OFF Switch 2 Pins Dim: 17x11mm:**

The "ON/OFF Switch 2 Pins Dim: 17x11mm" is a compact toggle switch with two pins, measuring 17x11mm. It operates in an ON/OFF fashion, completing or interrupting the circuit by toggling between two positions.

Commonly used in electronics and DIY projects, this switch is versatile, easy to mount on PCBs or panels, and provides a straightforward solution for controlling power in various applications.

1. **Rechargeable Li-ion Battery 18650 (3.7V, 2400mAh) Full Charge 4.2V:**

The "Rechargeable Li-ion Battery 18650 (3.7V, 2400mAh) Full Charge 4.2V" is a commonly used lithium-ion battery with a nominal voltage of 3.7 volts and a capacity of 2400mAh.

Its 18650 form factor makes it widely applicable in various devices, from flashlights to DIY electronics. It is rechargeable, reaching a full charge of 4.2 volts, and is popular for its balance between capacity and size. This battery is a versatile power source in electronics, providing a reliable and rechargeable energy solution.

1. **Wires 20 Cm “Male To Male” Pins:**

Wires 20 cm Male to Male Pins" are 20 cm long jumper wires with male connectors on both ends. These versatile cables are widely used in electronics for quick and temporary connections between components on a breadboard or in DIY projects.

With color-coding for organization, flexibility, and durability, they are essential tools for prototyping and experimentation, making circuit assembly more efficient and convenient.

1. **” Wires 20 Cm“ Female To Female Pins:**

Wires 20 cm Female to Female Pins" are 20 cm long jumper wires with female connectors on both ends. These versatile cables are used in electronics for making temporary connections between components, particularly those with male headers.

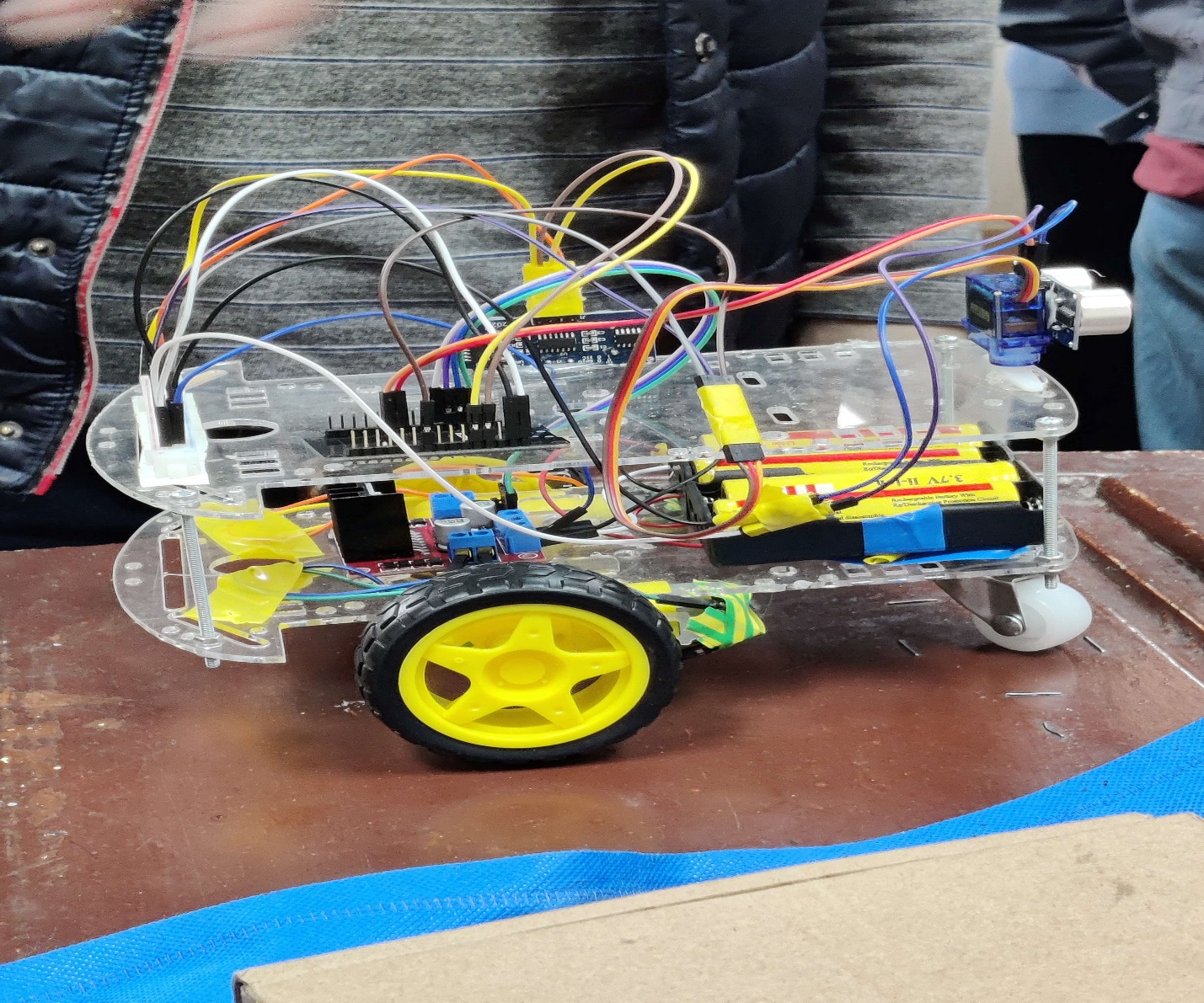
With flexibility, durability, and color-coded options for organization, these wires are essential in prototyping, breadboarding, and various DIY electronics projects, providing a convenient and efficient means of linking components in a circuit.

1. **Micro Servo MG90 0:180 Degree:**

The "Micro Servo MG90 0-180 Degree" is a compact servo motor with precise positional control within a 180-degree range.

Known for its small size and compatibility with microcontrollers, it is commonly used in robotics, RC vehicles, and DIY electronics projects where precise movement and space efficiency are essential.

**Proposed Idea**



The proposed auto-parking car mechanism integrates a novel design to enhance the efficiency of parking maneuvers. The prototype features a three-wheel configuration, with two wheels positioned at the front of the car and a dummy wheel at the rear. This configuration optimizes the maneuverability of the vehicle, allowing for precise movements during parking operations.

## **Hardware Components**

1. **Wheels and Motors:**
   * Two front wheels with attached motors for propulsion.
   * Dummy wheel at the rear for stability and directional control.
2. **Motor Driver (H-Bridge):**
   * The motors are connected to an H-Bridge motor driver, facilitating control over speed and direction.
3. **Micro-controller (ESP):**
   * The H-Bridge is interfaced with an ESP micro-controller responsible for overall system control.
   * The ESP micro-controller manages motor speed, direction, and interfaces with various sensors.
4. **Ultrasonic Sensors:**
   * Two ultrasonic sensors play a crucial role in parking detection.
   * Front-right ultrasonic sensor for obstacle detection in the car's path and determining if a parking slot is suitable for the car's size.
   * Rear-top ultrasonic sensor mounted on a servo motor for versatile scanning of available parking spaces during both forward and backward movements.
5. **Servo Motor:**
   * The servo motor attached to the rear-top ultrasonic sensor enables controlled rotation for comprehensive parking space scanning.

## **Working Principle**

1. **Front Obstacle Detection:**
   * The front-right ultrasonic sensor detects obstacles in the car's path, enabling obstacle avoidance during forward movement and determining slot suitability.
2. **Parking Space Detection:**
   * The rear-top ultrasonic sensor, mounted on a servo motor, scans the surroundings to identify available parking spaces.
3. **Parking Maneuver:**
   * The car moves forward until it reaches the end of the parking slot indicated by the rear-top ultrasonic sensor.
   * The servo motor rotates to measure the available space, allowing the car to reverse and align with the parking slot.
4. **Backward Movement Safety:**
   * During backward movement into the parking slot, the rear-top ultrasonic sensor calculates the distance to the adjacent car, preventing collisions.
5. **Localization:**
   * Utilizing the strength of signals from nearby cars with known coordinates, the ESP micro-controller calculates the distance between the parked cars.
   * By triangulating signals from multiple known points, the car's position is localized on a predefined map.

## 

## **Advantages**

1. **Enhanced Manoeuvrability:**
   * The three-wheel configuration ensures optimal maneuverability, especially in tight parking spaces.
2. **Comprehensive Parking Detection:**
   * The scanning capabilities of the rear-top ultrasonic sensor, coupled with servo motor rotation, provide a comprehensive view of available parking spaces.
3. **Backward Movement Safety:**
   * The rear-top ultrasonic sensor contributes to safe backward movement, calculating distances to avoid collisions with adjacent cars in the parking slot.

The proposed mechanism integrates innovative design elements to address parking challenges efficiently, providing a foundation for a robust auto-parking system with enhanced localization capabilities. The three-wheel configuration and advanced sensor integration contribute to the project's uniqueness, aiming to revolutionize parking solutions.

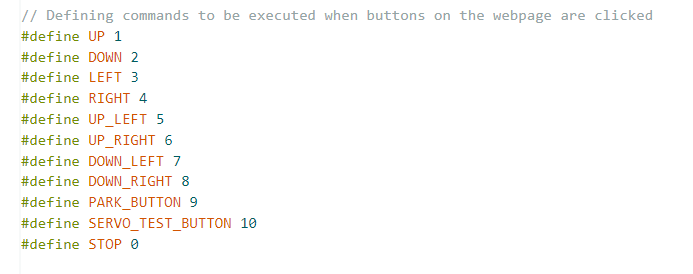
.

**Software**

**Command Definitions**

This section defines commands that correspond to different directions and actions:

* **UP, DOWN, LEFT, RIGHT, UP\_LEFT, UP\_RIGHT, DOWN\_LEFT, DOWN\_RIGHT**: Numeric representations for different movement directions.
* **PARK\_BUTTON**: Command to trigger the parking action.
* **SERVO\_TEST\_BUTTON**: Command to test the servo motor.
* **STOP**: Command to halt the movement.



**Ultrasonic Sensors Configuration**

* **ECHO1, TRIGGER1**: Pins for the first ultrasonic sensor mounted on a servo motor.
* **ECHO2, TRIGGER2**: Pins for the second ultrasonic sensor fixed on the back of the car.

**Servo Motor & Pins Configuration**

* **SERVOPIN**: Pin used to control the servo motor's angle.
* **ultrasonicServo**: An instance of the Servo library used to control the servo motor's movement.

**Variables for Distance Calculation**

* **onetime**: Variable to store the time for distance
* **measurement.dist1, dist2**: Variables to store distance measurements from the ultrasonic sensors.
* **distance\_threshold**: Threshold distance to detect the wall the car is moving next to.

**Motor Control Pins**

* Pins for controlling the left and right motors (IN1, IN2, enA, IN3, IN4, enB).

A white background with black and white text

Description automatically generated

The code snippet provides initializations for the various components and parameters required for an auto parking system utilizing ultrasonic sensors, a servo motor, and motor control pins. It sets up the groundwork for controlling the vehicle's movements, sensing distances, and managing the servo motor's rotation.

**stopMotors()**

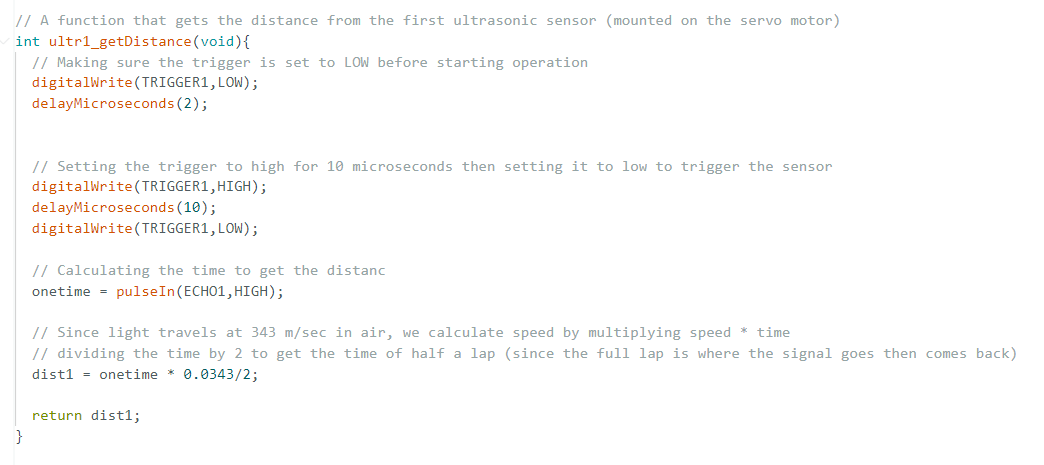
* **Function**: Stops both motors by setting the IN pins of each motor to LOW.
* **Explanation**: This function is responsible for halting the movement of the vehicle. It ensures that both motors come to a stop by setting the control pins (`IN1`, `IN2` for one motor and `IN3`, `IN4` for the other) to a low voltage, effectively turning off the power supply to the motors.

A white background with blue text

Description automatically generated

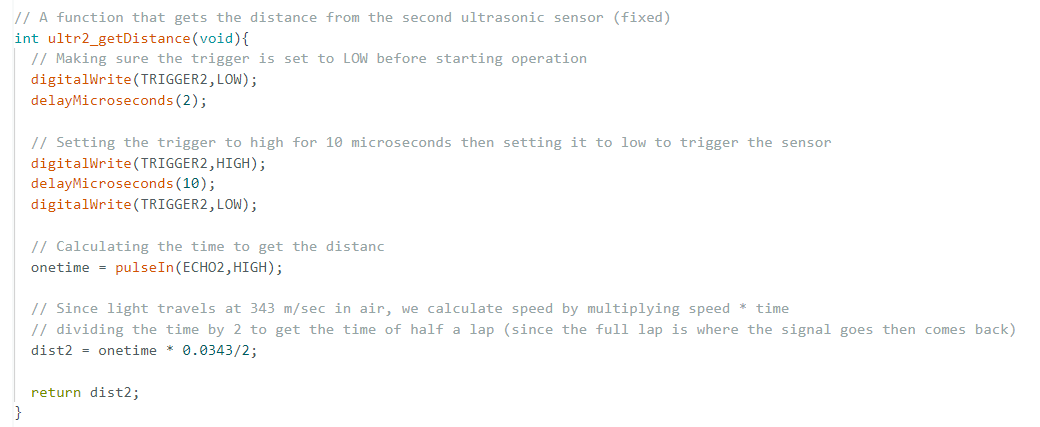
**ultr1\_getDistance()**

* **Function**: Measures the distance from the first ultrasonic sensor (mounted on the servo motor).
* **Explanation**: This function triggers the ultrasonic sensor by sending a brief high signal followed by a low signal. It measures the time taken for the signal to bounce back (echo) from an obstacle and calculates the distance based on the speed of sound in air. This provides the distance between the car and an obstacle in front of it.

****

**ultr2\_getDistance()**

* **Function**: Measures the distance from the second ultrasonic sensor (fixed at the back of the car).
* **Explanation**: Similar to ultr1\_getDistance(), this function triggers the ultrasonic sensor, measures the echo time, and calculates the distance based on the speed of sound. However, this sensor is positioned at the rear of the car, allowing measurement of the distance between the back of the car and an obstacle.

****

**Movement Control Functions:**

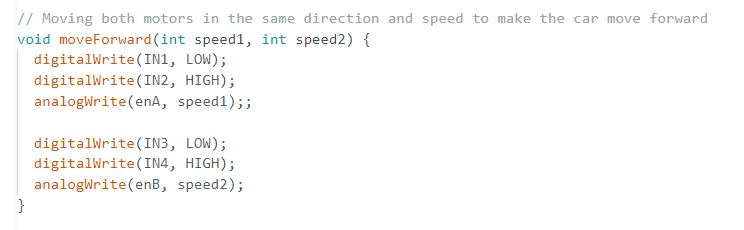
These functions control the movement of the car by manipulating the motor control pins based on the desired direction and speed:

* **moveBackward(speed1, speed2)**
  + **Explanation**: Makes the car move backward by setting the motors in reverse. It sets specific motor control pins (IN1, IN2, IN3, IN4) to certain voltage levels to achieve backward motion, and the speed is adjusted using analogWrite() for PWM-based speed control.

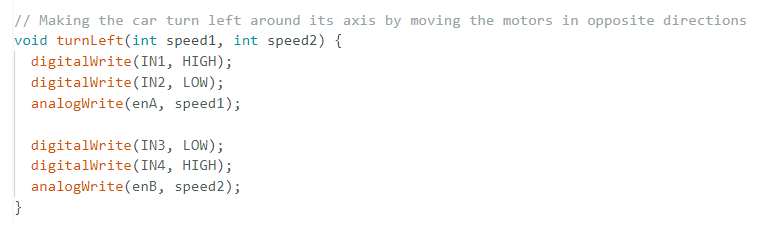
**A white background with black text

Description automatically generated**

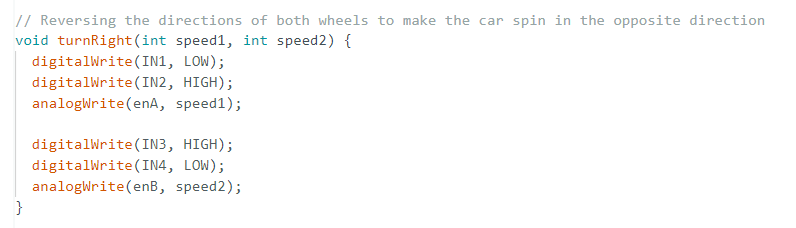
* **moveForward(speed1, speed2)**
  + **Explanation**: Initiates the car to move forward by setting the motors to rotate in the same direction. Similar to moveBackward(), it configures the motor control pins for forward motion and controls the speed using PWM.

****

* **turnLeft(speed1, speed2)**
  + **Explanation**: Causes the car to turn left by moving the motors in opposite directions. This function adjusts the motor control pins to create opposing movements between the left and right wheels, enabling the vehicle to turn left.

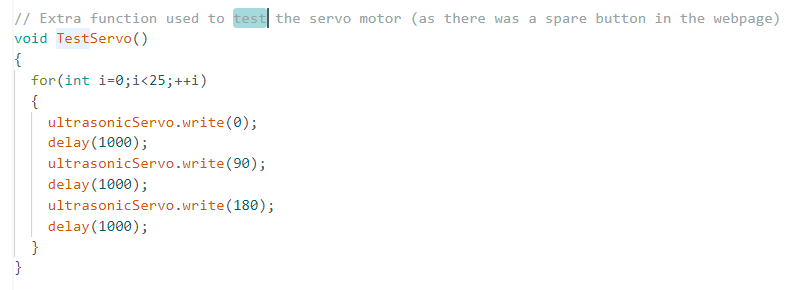


* **turnRight(speed1, speed2)**
  + **Explanation**: Induces a right turn by reversing the movement of the wheels in opposite directions. Similar to `turnLeft()`, it configures the motor control pins to make the car turn right by applying opposing motions to the left and right wheels.



**TestServo()**

* **Function**: Tests the servo motor by rotating it to three different angles (0, 90, 180 degrees) repeatedly for 25 iterations with one-second delays between rotations.
* **Explanation**: This function serves as a verification tool to test the functionality and range of motion of the servo motor. It iterates through three angles (0, 90, 180 degrees) and delays for one second at each angle to confirm the servo's operational capability and movement range.



Each of these functions is crucial for controlling and managing different aspects of the auto parking system, such as motion control, distance measurement, and servo motor testing, contributing to the overall functionality of the system.

**Explanation of the park() Function:**

**Description:**

The park() function is called upon pressing the designated button on the webpage. It orchestrates a sequence of actions for the auto parking system to autonomously park the car in a designated parking spot. Each step of the parking algorithm is outlined in comments within the code.

**Steps:**

1. **Ultrasonic Sensor Alignment**:
   1. The mounted ultrasonic sensor rotates to face the wall (ultrasonicServo.write(180)), allowing distance measurement between the car and the wall.
2. **Approaching the Parking Slot**:
   1. The car moves next to the wall until both ultrasonic sensors (ultr1\_getDistance() and `ultr2\_getDistance()`) register a distance larger than the specified threshold (`distance\_threshold`), indicating that the car is within the parking slot.
   2. During this step, the car moves forward (`moveForward(70,70)`) while continually checking the distances until both sensors detect the sufficient distance from the wall.
3. **Alignment to Parking Spot**:
   1. The car turns left about its axis (`turnLeft(80,80)`) to become perpendicular to the parking spot.
   2. This action ensures the car's alignment with the parking spot for proper entry.
4. **Backward Movement and Distance Measurement**:
   1. The mounted ultrasonic sensor rotates to face the back of the car (ultrasonicServo.write(90)) to measure the distance behind the car as it moves backward.
   2. The car then moves backward into the parking spot (moveBackward(70,70)`) for a certain duration (`delay(300)`) before stopping.
5. **Final Alignment**:
   1. The car turns right about its axis (`turnRight(80,80)`) to become parallel to the parking spot, ensuring proper alignment.

**Function Breakdown:**

* **Delay Control**: delay() is used between actions to ensure proper timing and synchronization of movements.
* **Distance Checks**: ultr1\_getDistance() and ultr2\_getDistance() are repeatedly called within a while loop to continuously monitor distances until both sensors indicate the car is appropriately positioned.
* **Motor Control**: Functions like moveForward(), moveBackward(), turnLeft(), and turnRight() are used to control the motion of the car by setting the motor control pins and adjusting speed.

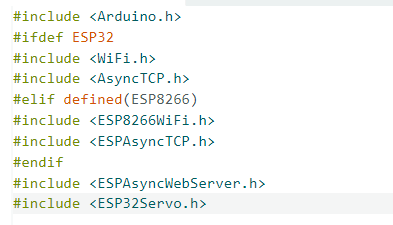


The park() function encapsulates the algorithm for autonomous parking by sequentially executing specific movements and sensor readings. It coordinates the actions required to position the car accurately within a parking spot, making use of the ultrasonic sensors, servo motor, and motor control functions to accomplish the parking sequence autonomously.

**Explanation of WiFi Connection and Localization Code:**

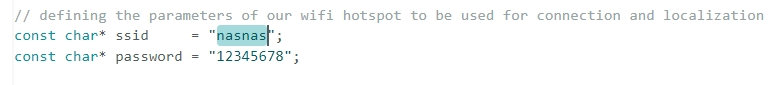
**Libraries Used:**

* The code utilizes libraries specific to the ESP32/ESP8266 platform for WiFi connectivity (WiFi.h or ESP8266WiFi.h), handling asynchronous TCP connections (AsyncTCP.h or ESPAsyncTCP.h), operating an asynchronous web server (ESPAsyncWebServer.h), and controlling the servo motor (ESP32Servo.h).



**WiFi Connection Parameters**

* ssid and password variables store the credentials of the WiFi hotspot to establish a connection.

****

**Localization Function (double localization()):**

* **Purpose**: The localization() function scans nearby WiFi networks and calculates estimated distances based on signal strengths received from specific SSIDs.
* **Steps**:
  + **WiFi Scan**: Scans nearby WiFi networks using WiFi.scanNetworks() and stores the count in numNetwork.
  + **Loop through Scanned Networks**: Iterates through each scanned network and checks if their SSID matches the predefined SSIDs stored in ssidArray[].
  + **Distance Calculation**:
    - If a matching SSID is found, the code calculates the estimated distance using the received signal strength (RSSI) by applying a logarithmic formula to convert RSSI to distance.
    - The formula used: distance = pow(10.0, (-59 - WiFi.RSSI(i)) / (10.0 \* 3)).
  + **Display and Return**: Prints the SSID, its RSSI, and calculated distance.
  + **Return Value**: Returns the estimated distance.

A screenshot of a computer code

Description automatically generated

This section of code establishes WiFi connectivity using predefined credentials and implements a localization function that estimates distances based on signal strengths received from specific SSIDs. The function scans nearby networks, retrieves RSSI values, and converts them into estimated distances for positioning or localization purposes.

**Explanation of HTML Webpage Code:**

This code represents an HTML page designed to control a car wirelessly through a Wi-Fi-connected interface. Here's a breakdown of its structure and functionality:

**HTML Structure:**

* The HTML code includes a structured layout to create a webpage containing various control elements for the car.

**Styling:**

* It includes CSS styling to define the appearance of the control elements, such as buttons representing different directional controls.

**Control Table:**

* The webpage consists of a table structure (<table>) divided into rows and columns, housing the control buttons.

**Control Buttons:**

* The control buttons are represented by table cells (<td>) containing Unicode arrow symbols representing different directions (up, down, left, right, etc.).
* The ontouchstart and ontouchend attributes of each button trigger JavaScript functions (onTouchStartAndEnd(value)) when the user touches and releases the button on a touch-enabled device.

**WebSocket and JavaScript:**

* The JavaScript section at the bottom establishes a WebSocket connection to communicate with the server.
* initWebSocket() function initializes the WebSocket connection when the page loads.
* onTouchStartAndEnd(value) function sends the corresponding control value (e.g., direction commands) through the WebSocket when a control button is touched.
* document.getElementById("mainTable").addEventListener("touchend", function(event){}) prevents the default touch event behavior to improve user experience.





This HTML page sets up a user interface with directional control buttons that, when touched, send specific control values through a WebSocket connection to interact with the car. The page is designed to enable remote control of the car's movements (forward, backward, left, right, etc.) through a touch-enabled interface.

**Explanation of the processCarMovement() Function:**

**Purpose:**

The processCarMovement(String inputValue) function is responsible for handling the input values received from the webpage control buttons. It maps these input values to specific actions or functions related to controlling the car's movements.

**Parameter:**

* **inputValue**: Represents the input value received from the webpage via WebSocket communication.

**Functionality:**

* **Switch Case:**
  + The function uses a switch statement to handle different input values received from the webpage, converting the inputValue string to an integer using inputValue.toInt().
  + Based on the converted integer value, the function performs different actions:
    - **Directional Movements:**
      * UP, DOWN, LEFT, RIGH: Calls functions like moveForward(), moveBackward(), turnLeft(), turnRight() with specific speed values (e.g., moveForward(80,80) for forward movement).
    - **Diagonal Movements:**
      * UP\_LEFT, UP\_RIGHT, DOWN\_LEFT, DOWN\_RIGHT: Initiates diagonal movements by calling moveForward() or moveBackward() with varying speed configurations to control the car's diagonal motion.
    - **Special Actions:**
      * PARK\_BUTTON: Invokes the park() function to initiate the autonomous parking sequence.
      * SERVO\_TEST\_BUTTON: Calls the TestServo() function to test the functionality of the servo motor.
      * STOP: Stops the motors by setting all motor control pins to LOW.
    - **Default Action:**
      * If an unrecognized or default value is received, it stops the motors to ensure no continuous movement in unintended scenarios.

**Serial Output:**

* The function also prints diagnostic information to the Serial Monitor (Serial.printf()) for debugging purposes, displaying the received input value and its converted integer equivalent.

**A screenshot of a computer

Description automatically generated**

This function acts as a dispatcher, interpreting input values received from the webpage control buttons and executing corresponding actions/functions to control the car's movements, perform special actions (like parking or servo testing), or halt the motors as needed.

**Explanation of WebSocket Event Handling Functions:**

**handleRoot() Function:**

* **Purpose:** Handles the request for the home webpage.
* **Functionality:** Sends the home webpage (htmlHomePage) as a response when the root URL is requested (`/`).

**handleNotFound() Function:**

* **Purpose:** Handles the situation when a requested webpage is not found.
* **Functionality:** Sends a 404 - "File Not Found" response if the requested webpage is not available.

**onWebSocketEvent() Function:**

* **Purpose:** Handles WebSocket events (connections, disconnections, data transmission).
* **Functionality:**
  + WebSocket Connection Event (WS\_EVT\_CONNECT):
    - Prints a message when a WebSocket client connects, displaying the client's IP address.
  + WebSocket Disconnection Event (WS\_EVT\_DISCONNECT):
    - Prints a message when a WebSocket client disconnects and stops the car's movement by calling processCarMovement("0").
  + WebSocket Data Event (WS\_EVT\_DATA):
    - Processes data received from a WebSocket client (button press data).
    - Parses the received data and invokes processCarMovement() to handle the control inputs sent from the webpage buttons.
  + Other WebSocket Events (WS\_EVT\_PONG, WS\_EVT\_ERROR):
    - These cases are not explicitly handled in the code.



**setup() Function:**

* **Purpose**: Initializes and sets up various components, configurations, and connections required for the proper functioning of the car control system.

**Serial.begin(9600):**

* Initializes serial communication at a baud rate of 9600.

**WiFi Initialization (WiFi.softAP(ssid, password)):**

* **Purpose:** Initiates a WiFi hotspot for the car control system.
* **Functionality:**
  + Uses WiFi.softAP() to create a hotspot with the provided SSID (ssid) and password (password).

**Setting Up Web Server (server):**

* **Purpose**: Establishes a server to handle webpage requests and WebSocket communication.
* **Functionality**:
  + server.on("/", HTTP\_GET, handleRoot): Associates the `/` path with the `handleRoot() function to serve the home webpage when requested.
  + server.onNotFound(handleNotFound): Handles situations when requested webpages are not found.
  + ws.onEvent(onWebSocketEvent): Assigns the onWebSocketEvent() function to handle WebSocket events.
  + server.addHandler(&ws): Adds WebSocket functionality to the server.
  + server.begin(): Starts the web server to handle incoming requests.

**Motor, Ultrasonic, and Servo Setup:**

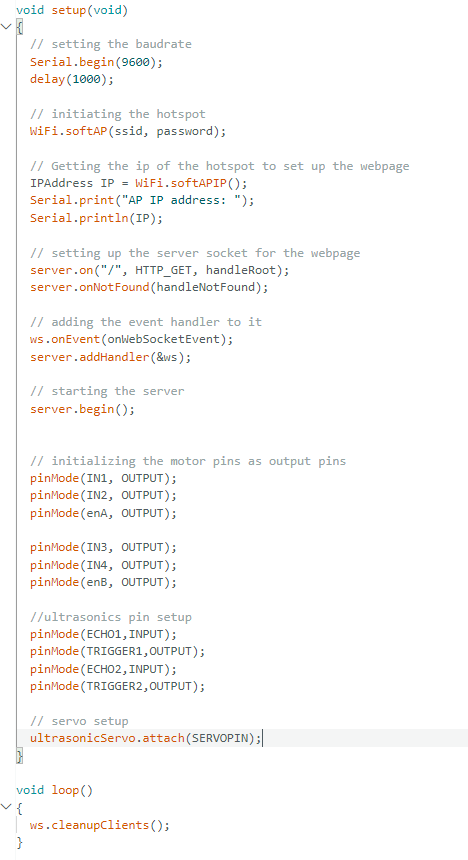
* Configures various pins used for motor control (IN1, IN2, enA, IN3, IN4, enB), ultrasonic sensors (ECHO1, TRIGGER1, ECHO2, TRIGGER2), and the servo motor (SERVOPIN).
* Utilizes pinMode() to set these pins as either INPUT or OUTPUT.

**ultrasonicServo.attach(SERVOPIN):**

* Attaches the servo motor to the specified pin (SERVOPIN) for control.

**loop() Function (ws.cleanupClients()):**

* The loop() function is simple, calling ws.cleanupClients() to manage and clean up WebSocket clients, ensuring proper handling of connections.

****

The setup() function initializes the car control system by setting up WiFi for communication, configuring a web server to handle webpage requests and WebSocket events, and setting pins for motor control, ultrasonic sensors, and the servo motor. It ensures that all necessary components and connections are properly initiated for the system to function as intended. The loop() function focuses on maintaining WebSocket client connections.