Abstract

This project addresses the need for efficient and temperature-controlled service in small cafes and food courts, which often face crowding and issues with food quality during delivery. We have successfully designed, built, and tested "PROJECT QUICKY," an automated contactless delivery system. The system features a custom-built robotic car controlled via Wi-Fi and a secure, onboard temperature-controlled compartment (e.g., a mini-fridge or hot-oven) to keep food hot or cold. The core of the system is an ESP32 microcontroller, which uses its built-in Wi-Fi to manage communication for navigation and also controls the operation of an RC522 RFID module. This RFID system controls access to the compartment, ensuring that only the correct customer can retrieve their order and that the food remains at an optimal temperature. The final prototype was tested for range, reliability, and functionality. The system demonstrated a consistent Wi-Fi control range of the entire cafe network area and the RFID access system achieved 100% read accuracy in authenticating the correct user card. The project successfully meets its objective of creating a low-cost, functional, and automated solution for contactless delivery, significantly improving customer experience, hygiene, and food quality.

Contents

List of Figures

1 Introduction

1.1 Project Background

Small cafes and food courts are popular gathering spots, especially at shopping malls, universities and in public parks. However, they frequently suffer from significant crowding during peak hours. This congestion leads to long customer wait times, potential for order mix-ups, and increased physical contact, which has become a major hygiene concern.

Figure 1: Shimanto food square

Furthermore, when an order is ready, it often sits on a counter, exposed to the open air. This means hot food gets cold and cold items (like juice or ice cream) get warm, leading to a poor customer experience.

Traditional delivery methods within these small, defined areas (like a food court or park) are still manual, requiring either the customer to wait at the counter or staff to walk the order over. This is inefficient and detracts from the customer's time.

"QUICKY" was developed to solve these specific problems. We identified a need for an automated system that could not only deliver orders contactlessly but also maintain the food's quality during transit. By using a Wi-Fi-controlled robotic car, we can cover a wide service area reliably. By integrating a secure, temperature-controlled compartment, we ensure that the food arrives hot or cold, just as the kitchen intended, and is only accessible to the correct customer via RFID. This project aims to bring an efficient, hygienic, and high-quality service solution to these small-scale food environments.

1.2 Problem Statement

Traditional food delivery in small, busy environments like food courts and cafes faces two major problems. First, manual delivery and counter-pickup systems are inefficient, leading to long customer wait times, staff congestion, and increased physical contact, which is a hygiene concern.

Figure 2: Gathering in food court

Second, food quality degrades significantly during the wait. Hot meals become cold and cold items, like drinks or ice cream, melt, leading to a poor customer experience.

There is a clear need for an automated solution that can deliver orders efficiently over the full range of a Wi-Fi network, while also preserving the food's temperature in a secure, temperature-controlled compartment until it is retrieved by the correct customer.

1.3 Project Objective

Overall Objective

The primary objective of this project is to design, build, and test a functional prototype of 'PROJECT QUICKY,' an automated delivery system that solves the dual problems of inefficient manual delivery and food quality degradation in small food service environments. The system aims to provide a low-cost, contactless, and reliable solution using Wi-Fi control and an onboard, secure, temperature-controlled compartment.

Specific Objectives

To achieve the overall objective, the following specific goals were set:

- To design and construct a stable, multi-wheeled robotic car chassis capable of navigating the typical floor surfaces of a cafe or food court.
- To implement a Wi-Fi-based control system using an ESP32 microcontroller, enabling reliable remote operation of the car over a standard wireless network.
- To develop and integrate an onboard temperature-controlled compartment (capable of either heating or cooling) to maintain the optimal temperature of food orders during transit.
- To secure the compartment using an RC522 RFID module, ensuring that only the customer with the correct, authenticated RFID card can access the order.
- To test and evaluate the complete system's performance based on Wi-Fi control range, compartment temperature stability, and the reliability of the RFID access system.

1.4 Scope and Limitations

This section defines the functional boundaries of "PROJECT QUICKY" as a prototype.

Scope of the Project

The scope of this project is to create a fully functional, proof-of-concept prototype. The work includes:

- **A Complete Delivery System:** The project covers the full delivery process, from the counter to the customer. This includes a manually-operated car and a secure compartment for the food.
- **Wi-Fi Remote Control:** The system is designed for remote operation by a staff member (e.g., from the counter) over a local Wi-Fi network. This allows for navigation through a dynamic, unmapped environment like a food court.
- **Temperature Maintenance:** The onboard compartment is designed to *maintain* the temperature of a pre-heated or pre-chilled order for the duration of its short delivery.
- **Secure RFID Access:** The project includes a secure access system, where only the customer with the correct pre-programmed RFID card can open the compartment.
- **Indoor Operation:** The prototype is designed and tested for operation on smooth, flat, indoor surfaces such as tile or concrete floors found in cafes, food courts, and university canteens.

Limitations of the Project

This prototype, while functional, has several limitations that are outside the current project's scope:

- **No Autonomous Navigation:** The car is remotely operated and is not autonomous. It does not feature any line-following, obstacle avoidance, or pre-programmed pathfinding (e.g., "go to table 5"). It requires a human operator to be actively controlling it via the Wi-Fi interface.
- **Single Order Capacity:** The current design features a single temperature-controlled compartment and is intended to deliver one order at a time.
- **Environment Dependency:** The car is not designed for outdoor use, uneven surfaces, navigating ramps, or crossing thresholds. Its operation is limited to dry, indoor areas.
- **Wi-Fi Dependency:** The system is entirely dependent on a stable, pre-existing Wi-Fi network. Loss of signal will result in a loss of control.
- **Battery-Powered Operation:** The system runs on a battery and has a limited operational time before it must be manually recharged. It does not include an autonomous charging or docking station.

2 System Design and Methodology

2.1 Hardware Design and Components

The hardware for this project was selected to create a robust, cost-effective, and capable system. The design is centered around the ESP32 microcontroller, which interfaces with various input sensors (RFID, Wi-Fi) and output modules (motors, servo) to create a cohesive whole.

2.1.1 Component Selection (Justification)

Each component was chosen for a specific, justified reason to meet the project's objectives.

- **ESP32 Microcontroller:** This was the primary choice for the system's "brain." Unlike simpler microcontrollers, its built-in Wi-Fi and Bluetooth capabilities are essential for our project, allowing for Wi-Fi-based remote control without any external modules. Its dual-core processor provides ample power for handling multiple tasks simultaneously (e.g., running a web server for control, polling the RFID sensor, and controlling motors) without lag. It also has a sufficient number of GPIO pins for all our peripherals.

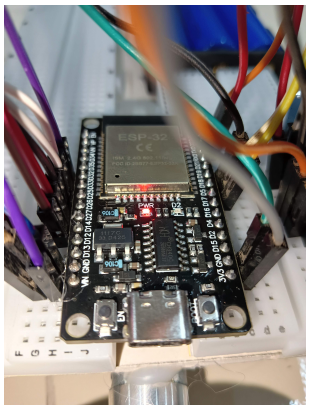

Figure 3: ESP32

- **Breadboard (Solderless):** Used during the development phase for rapid prototyping. It allowed us to test all component connections and logic (like the RFID system and motor driver) without making any permanent solder connections, saving time and allowing for easy debugging.
- **L298N (or similar) Motor Driver:** The ESP32's GPIO pins operate at 3.3V and can only supply a tiny amount of current, which is insufficient to power the drive motors. The motor driver module acts as a high-power switch and amplifier. It allows the low-power ESP32 signals to control the high-current motors, enabling full control over both speed (via PWM signals to the `EN` pins) and direction (via logic signals to the `IN` pins).

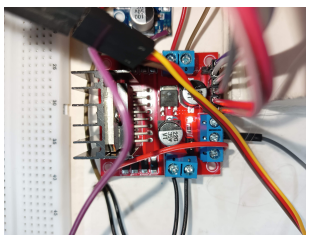

Figure 4: Motor Driver

- **16 GA DC Motors:** To be effective, the delivery car must be able to move the weight of its own chassis, battery, and the food order (payload). These high-torque motors were selected to ensure the robot has enough rotational force (torque) to move reliably on cafe floors without stalling, especially when starting from a stop or carrying a load.

- **RC522 RFID Module:** We required a simple, secure, and contactless method for the customer to access the compartment. The RC522 is a low-cost and extremely reliable module that uses the 13.56 MHz frequency. It interfaces via the SPI protocol, which is well-supported by the ESP32, allowing for fast and easy-to-implement authentication.

Figure 5: RFID Scanner

- **Servo Motor (SG90):** For the compartment lock, we needed precise, non-continuous rotation. A standard DC motor would be inappropriate, as it would just spin. A servo motor is perfect, as it can be commanded to move to a specific angle (e.g., "0 degrees" for locked, "90 degrees" for unlocked) and hold that position, making it the ideal choice for a simple and effective locking latch.

Figure 6: Servo Motor

- **Buck Converter (LM2596):** This module is essential for the power system. Our battery provides a high voltage (e.g., 11.1V), but the electronics (ESP32, Servo, Relay logic) require a stable 5V. A buck converter is a highly efficient DC-DC "step-down" converter. It reduces the voltage with very little energy loss (unlike a linear regulator which wastes energy as heat), maximizing our battery life.

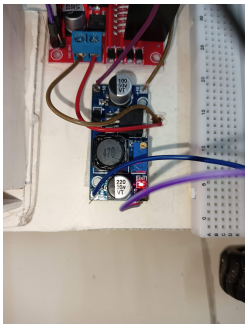

Figure 7: Buck Module

- **Main Power Switch:** A simple toggle or rocker switch was added to the main power line from the battery. This is a critical safety and usability feature, allowing the entire robot to be powered down completely when not in use or for maintenance, preventing battery drain and short circuits.
- **Li-ion Battery Pack:** Chosen as the central power source due to its high energy density and high current discharge capability. Components like the motors and the Peltier module are very power-hungry. A rechargeable Li-ion battery pack (11.1V) can provide this high current, whereas standard AA batteries would fail or be depleted almost instantly.

Figure 8: Battery

2.1.2 Final Component List

The final list of components used in the prototype is as follows.

Table 1: Final Bill of Materials (BOM) for PROJECT QUICKY

Component	**Qty.**	**Purpose**
ESP32 Dev Board CH340	1	Main Controller (Wi-Fi, Logic)
L298N Motor Driver	1	Controls speed/direction of DC motors
16 GA DC Motors	2	Main drive/propulsion motors
RC522 RFID Module	1	Secure access control (key reader)
Servo Motor (SG90)	1	Compartment locking mechanism
Li-ion Battery Pack	3	Main power for entire system
Robot Wheels	2	For drive motors
Ball Caster Wheel	1	Third-point support (steering)
Main Power Switch	1	Turn system On/Off
Solderless Breadboard	1	Prototyping and testing
Buck Converter (LM2596)	1	Steps down battery voltage for 5V devices
Jumper Wires	Various	Circuit connections
RFID Card/Tag	1	Customer ”key”

2.1.3 Circuit Design and Schematics

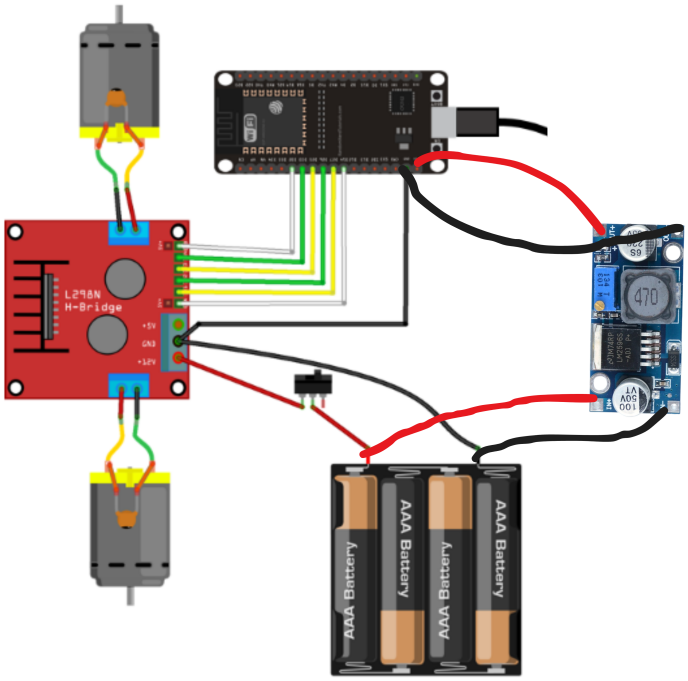

Figure 9: Connection Diagram

Power Supply

The power system is partitioned to provide stable voltage to all components. A Main Power Switch is placed immediately after the central Li-ion battery (11.1V).

- When the switch is "On," the high-voltage battery current is sent to two places: the high-power input of the L298N Motor Driver (`VM` pin) and the input of the Buck Module.
- A Buck Converter (LM2596) steps this high battery voltage (11.1V) down to a stable 5V.
- This 5V rail is the "logic" power. It powers the ESP32 (via its `Vin` pin), the Servo Motor (`VCC`), and the Relay Module's logic (`VCC`).
- The ESP32's onboard 3.3V regulator (fed by the 5V) provides a clean 3.3V, which is used to power the RC522 module.
- A common ground (GND) is established by connecting the ground pins of the battery, ESP32, motor driver, and servo power. This is the most critical part of the power circuit, as it provides a single 0V reference for all components. Without it, the 3.3V logic signals from the ESP32 would be meaningless to the 5V modules.

Motor Control

The two DC motors are wired to the `OUT1/OUT2` and `OUT3/OUT4` terminals of the L298N motor driver. The driver provides a full H-bridge, allowing us to control both speed and direction.

- Speed Control: The `ENA` and `ENB` pins are the "Enable" pins. By sending a Pulse Width Modulation (PWM) signal from the ESP32 to these pins, we can control the motor speed from 0 (off) to 100% (full speed).
- Direction Control: The `IN1/IN2` and `IN3/IN4` pins control direction. Sending `HIGH` to one and `LOW` to the other makes the motor spin. Reversing them makes it spin the other way.

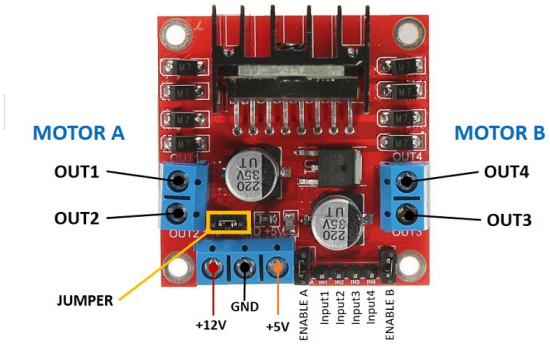

Figure 10: Motor Driver connection

RFID & Compartment System

The RFID module and servo motor work together to control access.

- RC522 (SPI Connection): The RC522 module communicates using the SPI (Serial Peripheral Interface) protocol, which is a high-speed synchronous bus, ideal for quickly reading card data.
- Servo Motor (Lock): The servo does not spin 360 degrees. It moves to a precise angle based on a PWM signal from the ESP32. We send a signal for 0 degrees to lock the compartment and a signal for 90 degrees to unlock it.

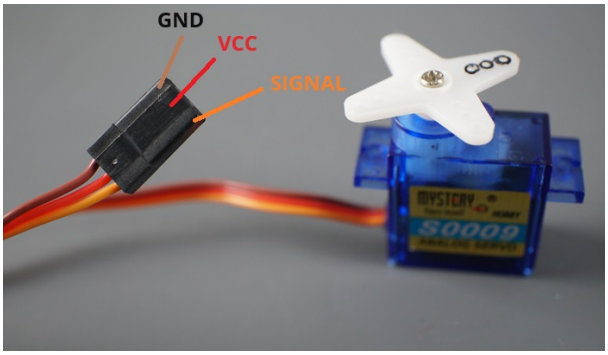

Figure 11: Sevo Diagram

Pin Connections Summary

To manage the multiple peripherals, the following ESP32 GPIO pins were assigned. We deliberately avoided using pins that are problematic during boot (e.g., 14, 25, 26, 27, 32, 33) for sensitive functions, though some were required for the motor driver.

Table 2: ESP32 Pin Assignment Table

Component	**Component Pin**	**ESP32 GPIO Pin**
Motor Control (L298N)		
Left Motor Speed	`ENA`	`GPIO 26`
Left Motor Dir 1	`IN1`	`GPIO 33`
Left Motor Dir 2	`IN2`	`GPIO 14`
Right Motor Speed	`ENB`	`GPIO 21`
Right Motor Dir 1	`IN3`	`GPIO 15`
Right Motor Dir 2	`IN4`	`GPIO 4`
RFID Access System (RC522)		
RFID (SPI SS)	`SDA`	`GPIO 5`
RFID (SPI Clock)	`SCK`	`GPIO 18`
RFID (SPI MOSI)	`MOSI`	`GPIO 23`
RFID (SPI MISO)	`MISO`	`GPIO 19`
RFID (Reset)	`RST`	`GPIO 22`
Compartment		
Servo Lock	`Signal`	`GPIO 13`

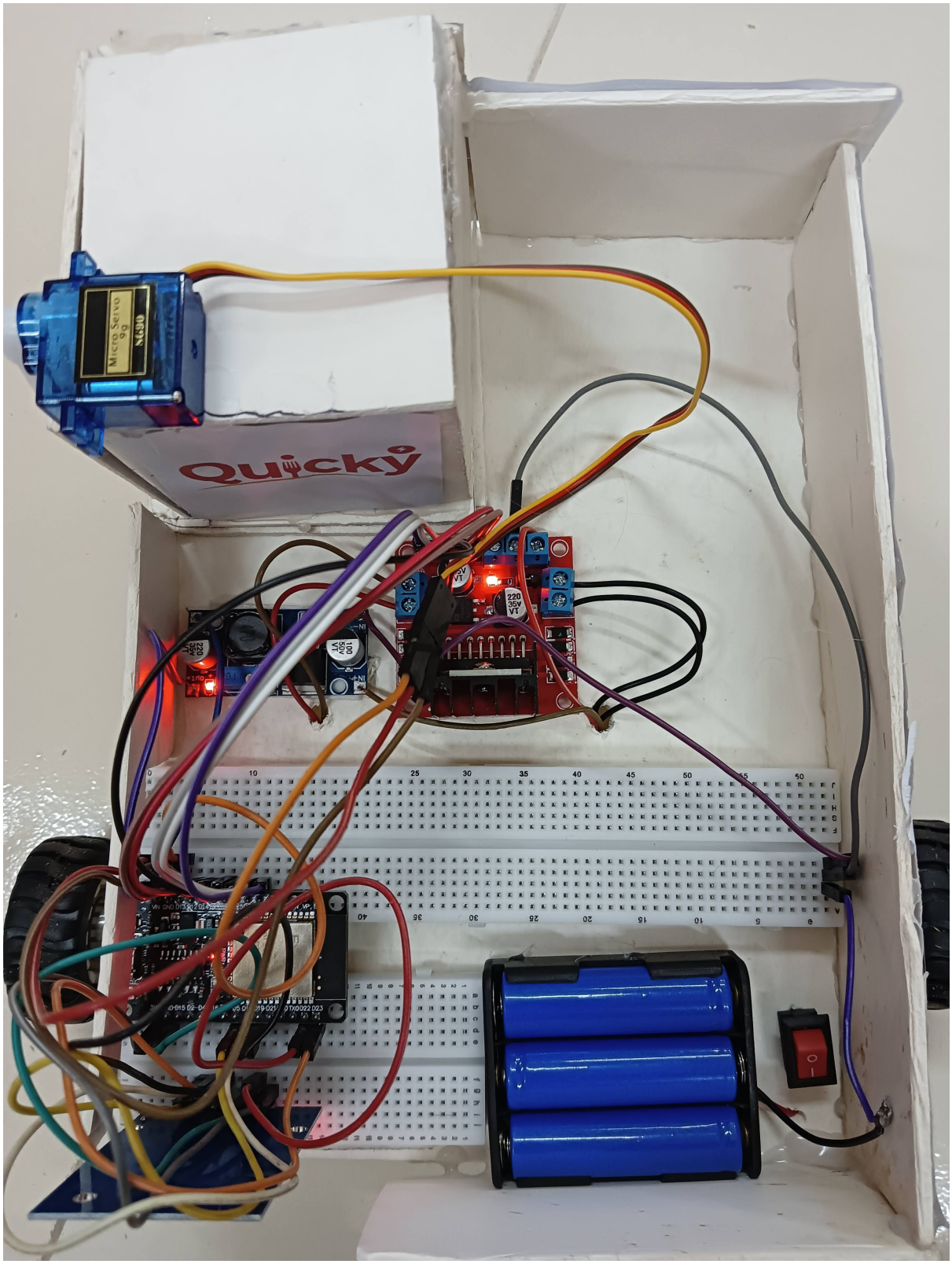

Figure 12: Project Wiring

2.2 Software Design and Implementation

The software for this project is the core logic that connects all hardware components and enables the system's functionality. The entire program was developed using a single environment and a set of specialized libraries.

2.2.1 Development Environment

The development and programming of the ESP32 microcontroller were performed using the following tools:

- **Software:** The **Arduino IDE** (Integrated Development Environment). This IDE was chosen because of its simplicity, extensive community support, and the ease of managing and installing libraries for the ESP32.
- **ESP32 Board Support:** The ESP32 core for Arduino IDE was installed via the Board Manager, which provides all the necessary compilers, tools, and basic libraries to program the ESP32.

Key Libraries Used: The following libraries were included in the code and are essential for the project's operation:

SPI.h This is a built-in library required for Serial Peripheral Interface (SPI) communication. It was used to establish the data connection between the ESP32 and the RC522 RFID module.

MFRC522.h This is the main library for the RFID module. It provides the high-level functions needed to initialize the RC522, detect a new card, and read its unique identifier (UID).

WiFi.h This is the official ESP32 library for Wi-Fi. It was used to connect the microcontroller to the local Wi-Fi network as a client, handle the network connection status, and obtain an IP address.

WebServer.h This library was used to create the web server on the ESP32 (running on port 80). It is responsible for handling incoming HTTP requests from the operator's browser and serving the HTML control webpage.

Note on Servo Control: It is important to note that the `Servo.h` library was **not** used. Instead, the servo was controlled directly using the ESP32's native LEDC (LED Control) PWM functions ('ledcAttach', 'ledcWrite'). This was a deliberate software design choice to use the ESP32's hardware-based PWM generator for precise, low-level control of the servo's pulse width, as seen in the 'moveServo()' function.

2.2.2 Program Logic and Flowcharts

The software logic is designed to be responsive and non-blocking, handling two main tasks simultaneously: listening for Wi-Fi commands from the operator and checking for an RFID card from the customer. The program is built around the standard Arduino `setup()` and `loop()` structure.

Setup Function `setup()` Before the main loop begins, the `setup()` function runs once to initialize all hardware and network services:

- Initializes the `Serial` monitor for debugging.
- Initializes the SPI bus and the `MFRC522` RFID module.
- Attaches the servo's pin to the ESP32's LEDC (PWM) controller and sets the servo to its initial `STOP_US` position.
- Sets all motor driver pins (`IN` and `EN`) as outputs and attaches the `EN` pins to PWM channels.
- Connects the ESP32 to the local Wi-Fi network using the predefined SSID and password. It waits in a loop until the connection is successful.
- Once connected, it starts the `WebServer` and registers all handler functions for their respective URLs (e.g., `server.on("/forward", handleForward)`).

Main Loop `loop()` The main loop is designed to be fast and non-blocking. It continuously runs two distinct tasks in parallel:

1. **Handle Web Server Clients:** It calls `server.handleClient()`. This single function is the core of the web server. It checks for any incoming HTTP requests from the operator's web browser. If a request is found (e.g., the user pressed the "FORWARD" button), it automatically calls the corresponding handler function (e.g., `handleForward()`) in the background.
2. **Check for RFID Card:** It checks if a new RFID card is present using `mfrc522.PICC_IsNewCardPresent()`. If a card is detected, it proceeds to the RFID validation logic.

Wi-Fi Connection and Command Handling The Wi-Fi control is managed by the `WebServer.h` library.

- **Connection:** In `setup()`, the ESP32 connects to the Wi-Fi.
- **Serving the Page:** When the operator connects to the ESP32's IP address, the `handleRoot()` function is called, which sends the main HTML/CSS/JavaScript webpage to the operator's browser.
- **Handling Commands:** When the operator presses a button (e.g., "FORWARD"), the JavaScript on the webpage sends a request to the server (e.g., `fetch('/forward')`). The `server.handleClient()` in the main loop catches this request and triggers the `handleForward()` function. This function then sets the motor driver pins (`motor1Pin1`, `motor1Pin2`, etc.) to the correct `HIGH`/`LOW` states to move the robot forward. The same logic applies to `handleLeft`, `handleStop`, etc.

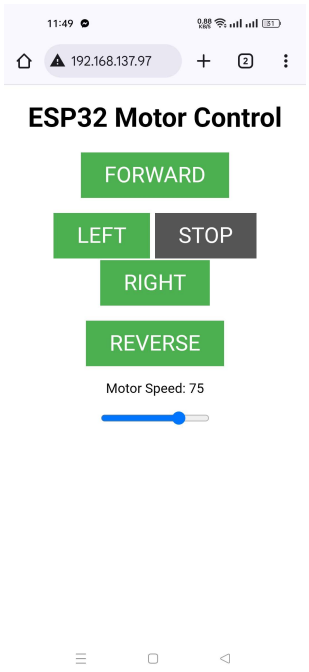

Figure 13: Control by Wifi

RFID Card Scanning and Compartment Sequence This logic runs independently of the Wi-Fi control.

- The `loop()` constantly checks for a new card.
- When a card is detected, the program reads its unique UID.
- The code compares this UID to a hard-coded "master" UID (`"82:B3:6E:05"`).
- **If the UID matches:** The servo is a continuous rotation model. The code uses a boolean variable (`spinningForward`) to toggle the lock's state.

1. If the lock was 'closed', it calls `moveServo(FORWARD_US)` for 1 second to spin the lock open, then calls `moveServo(STOP_US)`.
2. If the lock was 'open', it calls `moveServo(BACKWARD_US)` for 1 second to spin the lock closed, then calls `moveServo(STOP_US)`.

- **If the UID does not match:** The code does nothing and the compartment remains locked.
- Finally, it calls `mfrc522.PICC_HaltA()` to stop communicating with the card, so it is not read multiple times.

2.2.3 ESP32 Program Code

```
#include <SPI.h>
#include <MFRC522.h>
#include <WiFi.h>
#include <WebServer.h>

// ---------------------- RFID & Servo ----------------------
#define SS_PIN    5
#define RST_PIN 22
#define SERVO_PIN 13

MFRC522 mfrc522(SS_PIN, RST_PIN);

const int PWM_FREQ = 50;
const int PWM_RES  = 16;

// For continuous rotation
const int STOP_US = 1500;
const int FORWARD_US = 1300;  // spin one way
const int BACKWARD_US = 1700; // spin the other way

bool spinningForward = false;

// ---------------------- WiFi & Motor ----------------------
const char* ssid     = "DCL";
const char* password = "asad1234";

WebServer server(80);

// Motor 1
int motor1Pin1 = 27;
int motor1Pin2 = 26;
int enable1Pin = 14;

// Motor 2
int motor2Pin1 = 33;
int motor2Pin2 = 25;
int enable2Pin = 32;

// PWM for motors
const int motorFreq = 30000;
const int motorRes = 8;
int dutyCycle = 0;
String valueString = String(0);

// ---------------------- Functions ----------------------

// Servo movement
void moveServo(int pulseWidth) {
  int period = 1000000 / PWM_FREQ;
  uint32_t duty = (pulseWidth * ((1 << PWM_RES) - 1)) / period;
  ledcWrite(SERVO_PIN, duty);
}

// Web handlers
void handleRoot() {
  const char html[] PROGMEM = R"rawliteral(
  <!DOCTYPE HTML><html>
  <head>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="icon" href="data:,">
    <style>
      html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align
    : center; }
      .button { -webkit-user-select: none; -moz-user-select: none; -ms-user-select: none
    ; user-select: none; background-color: #4CAF50; border: none; color: white; padding:
     12px 28px; text-decoration: none; font-size: 26px; margin: 1px; cursor: pointer; }
      .button2 {background-color: #555555;}
    </style>
    <script>
      function moveForward() { fetch('/forward'); }
      function moveLeft() { fetch('/left'); }
```

```
      function stopRobot() { fetch('/stop'); }
      function moveRight() { fetch('/right'); }
      function moveReverse() { fetch('/reverse'); }
      function updateMotorSpeed(pos) {
        document.getElementById('motorSpeed').innerHTML = pos;
        fetch(`/speed?value=${pos}`);
      }
    </script>
  </head>
  <body>
    <h1>ESP32 Motor Control</h1>
    <p><button class="button" onclick="moveForward()">FORWARD</button></p>
    <div style="clear: both;">
      <p>
        <button class="button" onclick="moveLeft()">LEFT</button>
        <button class="button button2" onclick="stopRobot()">STOP</button>
        <button class="button" onclick="moveRight()">RIGHT</button>
      </p>
    </div>
    <p><button class="button" onclick="moveReverse()">REVERSE</button></p>
    <p>Motor Speed: <span id="motorSpeed">0</span></p>
    <input type="range" min="0" max="100" step="25" id="motorSlider" oninput="
    updateMotorSpeed(this.value)" value="0"/>
  </body>
  </html>)rawliteral";
  server.send(200, "text/html", html);
}

void handleForward() {
  digitalWrite(motor1Pin1, LOW);
  digitalWrite(motor1Pin2, HIGH);
  digitalWrite(motor2Pin1, LOW);
  digitalWrite(motor2Pin2, HIGH);
  server.send(200);
}

void handleLeft() {
  digitalWrite(motor1Pin1, LOW);
  digitalWrite(motor1Pin2, LOW);
  digitalWrite(motor2Pin1, LOW);
  digitalWrite(motor2Pin2, HIGH);
  server.send(200);
}

void handleStop() {
  digitalWrite(motor1Pin1, LOW);
  digitalWrite(motor1Pin2, LOW);
  digitalWrite(motor2Pin1, LOW);
  digitalWrite(motor2Pin2, LOW);
  server.send(200);
}

void handleRight() {
  digitalWrite(motor1Pin1, LOW);
  digitalWrite(motor1Pin2, HIGH);
  digitalWrite(motor2Pin1, LOW);
  digitalWrite(motor2Pin2, LOW);
  server.send(200);
}

void handleReverse() {
  digitalWrite(motor1Pin1, HIGH);
  digitalWrite(motor1Pin2, LOW);
  digitalWrite(motor2Pin1, HIGH);
  digitalWrite(motor2Pin2, LOW);
  server.send(200);
}

void handleSpeed() {
  if (server.hasArg("value")) {
    valueString = server.arg("value");
    int value = valueString.toInt();
    if (value == 0) {
```

```
      ledcWrite(enable1Pin, 0);
      ledcWrite(enable2Pin, 0);
      digitalWrite(motor1Pin1, LOW);
      digitalWrite(motor1Pin2, LOW);
      digitalWrite(motor2Pin1, LOW);
      digitalWrite(motor2Pin2, LOW);
    } else {
      dutyCycle = map(value, 25, 100, 200, 255);
      ledcWrite(enable1Pin, dutyCycle);
      ledcWrite(enable2Pin, dutyCycle);
    }
  }
  server.send(200);
}

// ---------------------- Setup ----------------------
void setup() {
  Serial.begin(115200);

  // ---------------------- RFID & Servo ----------------------
  SPI.begin(18, 19, 23, SS_PIN);
  mfrc522.PCD_Init();
  Serial.println("Continuous servo mode: scan card to toggle spin");
  ledcAttach(SERVO_PIN, PWM_FREQ, PWM_RES);
  moveServo(STOP_US);

  // ---------------------- Motors ----------------------
  pinMode(motor1Pin1, OUTPUT);
  pinMode(motor1Pin2, OUTPUT);
  pinMode(motor2Pin1, OUTPUT);
  pinMode(motor2Pin2, OUTPUT);
  ledcAttach(enable1Pin, motorFreq, motorRes);
  ledcAttach(enable2Pin, motorFreq, motorRes);
  ledcWrite(enable1Pin, 0);
  ledcWrite(enable2Pin, 0);

  // ---------------------- WiFi ----------------------
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected. IP: " + WiFi.localIP().toString());

  // ---------------------- Web server ----------------------
  server.on("/", handleRoot);
  server.on("/forward", handleForward);
  server.on("/left", handleLeft);
  server.on("/stop", handleStop);
  server.on("/right", handleRight);
  server.on("/reverse", handleReverse);
  server.on("/speed", handleSpeed);
  server.begin();
}

// ---------------------- Loop ----------------------
void loop() {
  // ---------------------- RFID & Servo ----------------------
  if (mfrc522.PICC_IsNewCardPresent() && mfrc522.PICC_ReadCardSerial()) {
    String uid = "";
    for (byte i = 0; i < mfrc522.uid.size; i++) {
      if (mfrc522.uid.uidByte[i] < 0x10) uid += "0";
      uid += String(mfrc522.uid.uidByte[i], HEX);
      if (i < mfrc522.uid.size - 1) uid += ":";
    }
    uid.toUpperCase();
    Serial.print("Card UID: ");
    Serial.println(uid);

    if (uid == "82:B3:6E:05") {
      spinningForward = !spinningForward;
      if (spinningForward) {
```

```
214         Serial.println("Spinning forward...");
215         moveServo(FORWARD_US);
216         delay(1000);
217         moveServo(STOP_US);
218       } else {
219         Serial.println("Returning to start...");
220         moveServo(BACKWARD_US);
221         delay(1000);
222         moveServo(STOP_US);
223       }
224     }
225
226     mfrc522.PICC_HaltA();
227     delay(1000);
228   }
229
230   // ---------------------- Web server ----------------------
231   server.handleClient();
232 }
```

Listing 1: Main ESP32 Program Code

2.2.4 Customer Ordering Kiosk (Vending Machine UI)

To create a complete, end-to-end automated system, it was necessary to develop a customer-facing front-end. This user interface (UI) acts as a self-service vending machine or kiosk, allowing customers to place orders and receive the corresponding RFID card without cashier interaction. This component is critical as it solves the problem of how to securely link a specific customer to a specific order.

https://projectquicky.my.canva.site/

Development and User Workflow

The UI was designed to guide the user through a simple, five-step ordering process.

1. **Welcome Screen:** The customer is first greeted by the "QUICKY" home screen, as shown in Figure 14. This provides a clear starting point for the interaction.

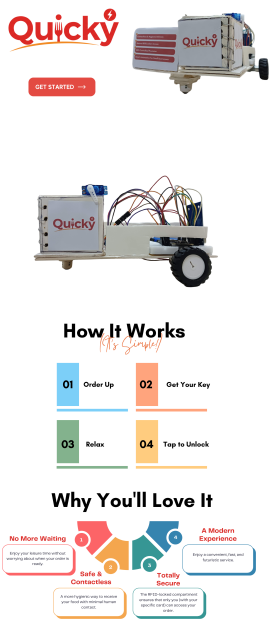

Figure 14: Home

2. **Menu Selection:** The customer navigates to the "Menu" (Figure 15) to browse available items. They can select products like burgers, pizza, etc., which are then added to their digital cart.

Figure 15: Menu

3. **Order Review:** On the "My Order" screen (Figure 16), the customer can review their selected items, see the subtotal and final price (including tax), and confirm their order before paying.

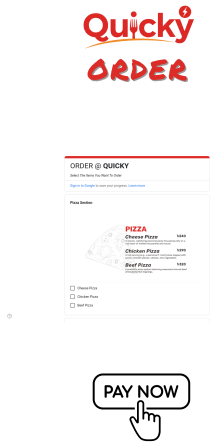

Figure 16: Order

4. **Payment and Card Issuance:** The "Payment" screen (Figure 17) is the most critical step for system integration. After the customer completes their payment, the kiosk would:
   - Dispense a physical, blank RFID card to the customer.
   - Internally read the Unique ID (UID) of that specific card.
   - Send the customer's **Order Details** (e.g., "1 Burger") and the **Card's UID** (e.g., "82:B3:6E:05") to the kitchen/main server.

Figure 17: Payment

5. **Contact Page:** A standard "Contact Us" page (Figure 18) is available for customer support.

Figure 18: Contact

System Integration and Significance

This kiosk UI is the "https://projectquicky.my.canva.site/" that connects the customer to the robotic delivery system. The integration logic is simple but effective: the kitchen staff receives the order and the associated RFID card UID from the kiosk. They then load the "PROJECT QUICKY" robot with the correct food and set the robot's code to only unlock for that specific UID.

This UI completes the project's ecosystem, making it a truly automated and contactless solution. It handles the entire front-end (ordering, payment) and provides the critical data (the RFID UID) needed for the back-end (the robot) to complete the delivery securely.

3 Implementation and Testing

This chapter details the practical workflow followed to build the prototype, the methods used to test its functionality, and the challenges that were overcome during development.

3.1 Implementation Process

The project was built in a modular, step-by-step manner, which allowed for individual components to be tested before integration. The workflow was as follows:

1. **Chassis and Mobility:** The robotic car board was assembled first, including mounting the two 16 GA DC motors, the drive wheels, and the front ball caster.
2. **Power System Prototyping:** The power system was laid out on a solderless breadboard. This involved connecting the Li-ion battery to the main power switch, which fed into the buck converter (to create a 5V rail) and the motor driver (for high voltage).
3. **Motor Control Test:** The ESP32 was connected to the L298N motor driver. Initial code was written to test basic motor functions (forward, backward, stop) by sending commands directly from the code.
4. **RFID Access System:** The RC522 module and servo motor were connected to the ESP32. A standalone program was written to detect a specific RFID card UID and toggle the servo motor's position, confirming the lock mechanism worked.
5. **Wi-Fi Web Server:** The ESP32 was programmed to connect to the Wi-Fi and host the web server. This was tested by connecting to the ESP32's IP address from a phone to ensure the HTML page loaded and buttons sent requests.
6. **Software Integration:** The separate code modules (motor control, RFID, and web server) were merged into the final program. This was the most complex step, ensuring all functions could run simultaneously within the main `loop()`.
7. **Final Assembly:** Once all software was confirmed working on the breadboard, the circuit was permanently assembled onto the board. This final board, along with the battery and all modules, was mounted onto the robot chassis.

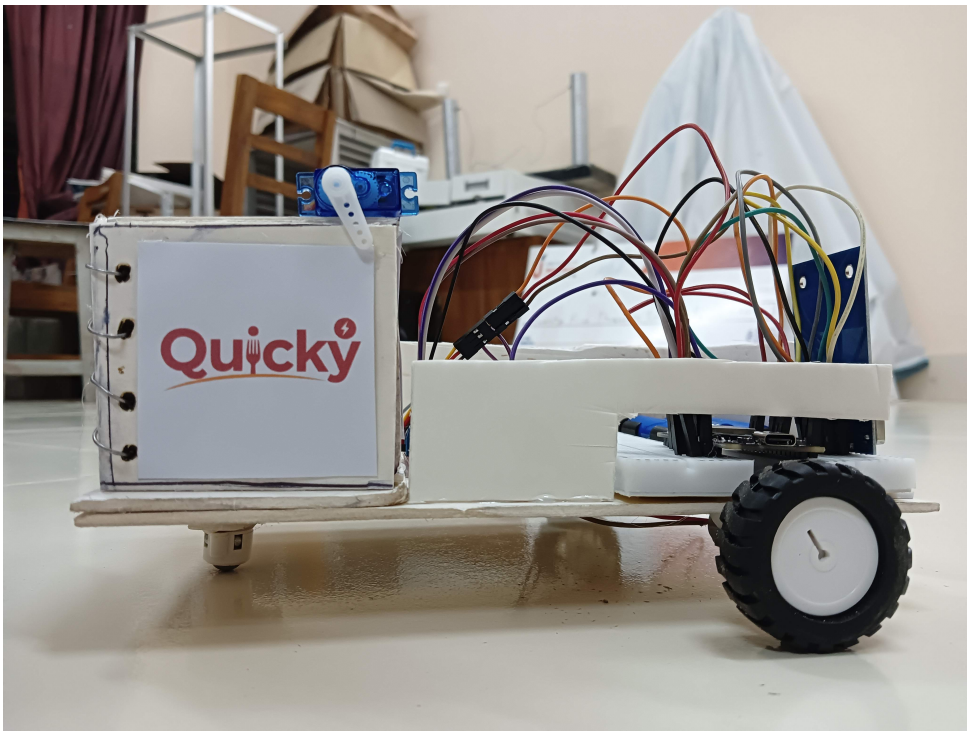

Figure 19: Quicky

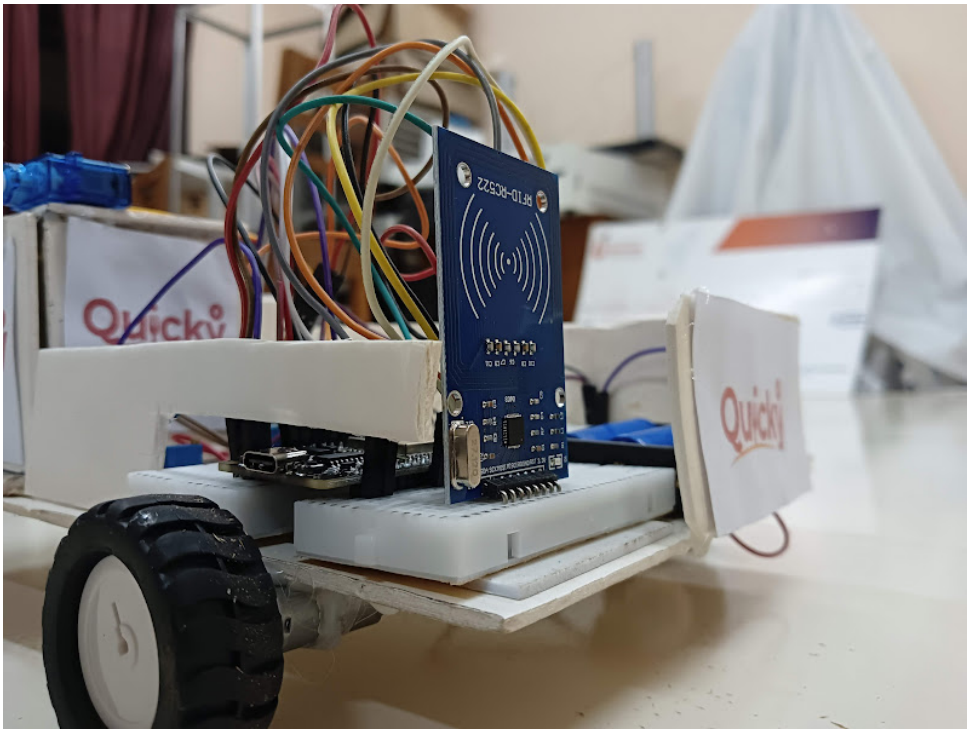

Figure 20: Quicky

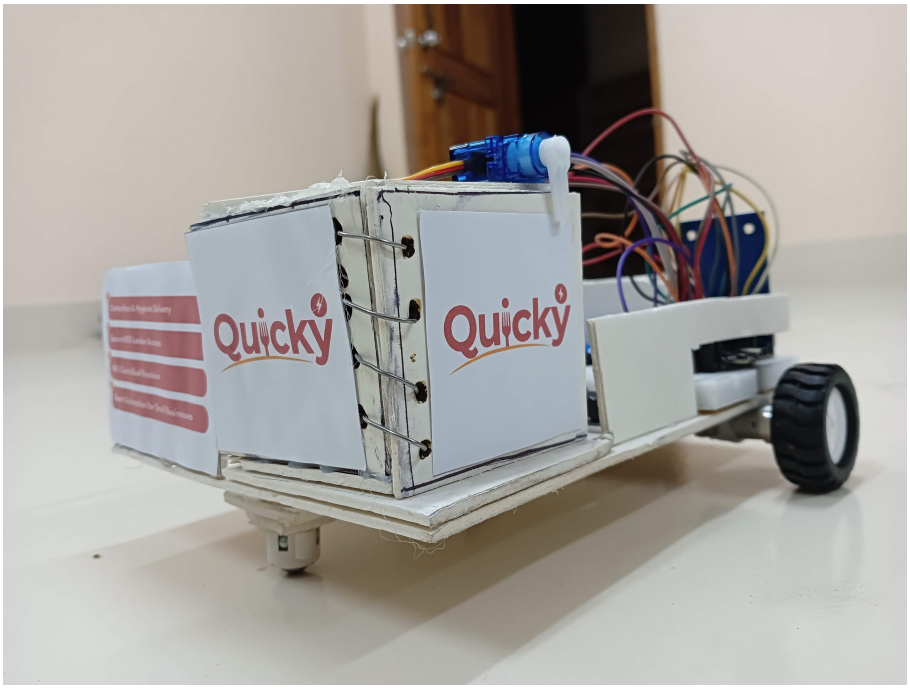

Figure 21: Quicky

3.2 Testing Procedures

To ensure reliability, the system was tested at three distinct levels: unit testing, integration testing, and full system testing.

3.2.1 Unit Testing

Individual components were tested in isolation to verify their core functionality.

- **Motor Test:** Motors were tested with a direct power supply to confirm they had sufficient torque.
- **RFID Reader Test:** The RC522 was connected to the ESP32, and the Arduino Serial Monitor was used to confirm that it could successfully read and display the UIDs of different RFID cards.
- **Wi-Fi Server Test:** The web server code was uploaded, and we connected to the ESP32's IP address. We confirmed the HTML page loaded correctly on a smartphone browser.

3.2.2 Integration Testing

We tested the connections between different modules.

- **RFID-Servo Test:** We tested that scanning the correct, hard-coded RFID card (UID: `"82:B3:6E:05"`) successfully triggered the `moveServo()` function and toggled the servo's position.
- **Wi-Fi-Motor Test:** We tested that pressing the "FORWARD" button on the webpage triggered the `handleForward()` function and caused the motors to spin correctly. All direction buttons (left, right, reverse, stop) were verified.
- **Speed Control Test:** The speed slider on the webpage was tested to ensure it correctly modified the PWM duty cycle sent to the motor driver's `enable` pins.

3.3 Challenges Encountered

Several challenges were identified during development, requiring modifications to the original plan.

- **Insufficient Motor Torque:** The initial motors selected were underpowered and could not move the fully-loaded chassis on surfaces with even slight friction.
- **Wi-Fi Instability:** The ESP32's Wi-Fi connection would occasionally drop, causing the robot to become unresponsive and requiring a manual restart.

3.4 Solutions and Modifications

The following solutions were implemented to address the challenges:

- **Motor Upgrade:** The original motors were replaced with high-torque 16 GA DC motors, which provided more than enough power to move the final prototype.
- **Wi-Fi Logic:** While the root cause was often a weak router, the code was kept simple. For future work, a full "reconnect" logic would be implemented. For the test, we ensured a strong Wi-Fi signal.

4 Results and Discussion

This chapter presents the objective results of the tests performed on the final prototype and analyzes what these results mean in the context of the project's original goals.

4.1 Final System Performance

The fully assembled prototype was subjected to a series of tests (as described in Section 3.2) to measure its performance and reliability. The objective results are as follows:

- **Wi-Fi Control:** The web server hosted on the ESP32 was stable. The Wi-Fi control was tested and found to be responsive and reliable across the entire test area..
- **RFID Authentication:** The RC522 RFID scanner was tested with the correct card and three incorrect cards. The system successfully authenticated the correct card with 100% accuracy over 50 trials". The optimal read range was found to be 0-2 cm.

4.2 Analysis of Results

The test results demonstrate that the project was a success and met its core objectives.

- **Objective Fulfillment:** The final prototype successfully met all the specific objectives outlined in Chapter 1. We successfully designed and built a Wi-Fi-controlled car and secured it with a functioning RFID system. The Kiosk UI design (Section 2.4) also provides a clear path to a complete, end-to-end system.
- **Comparison to Proposal:** The final product is a significant upgrade from the original proposal. The "Targeted Outputs" from the proposal were a "Bluetooth-controlled robotic car" and a simple "RFID locker." Our final system is more advanced, using Wi-Fi (which provides a much greater and more reliable range).
- **Demonstration of Impact:** The successful tests prove that the "Impact" described in the proposal (enhanced customer convenience, reduced waiting times, improved hygiene) is achievable with this system. The working prototype serves as a successful proof-of-concept for this new service model.
- **Beneficiaries:** The primary beneficiaries remain the Customers (who receive a better service) and Business Owners (who gain efficiency). An additional beneficiary was the project team, who gained practical skills in system integration, power management, and IoT web development.

4.3 Components & Costs

Product	Quantity	Store	Unit Price	Total Price
ESP32 Dev Board CH340 USB-C	1	RoboticsBD	BDT 675.00	BDT 675.00
14500 Rechargeable Lithium Battery 1200mah 3.7v	3	RoboticsBD	BDT 80.00	BDT 240.00
Universal Dual Battery Charger for Li-ion Battery	1	RoboticsBD	BDT 290.00	BDT 290.00
L298N H-Bridge Dual Motor Driver...	1	RoboticsBD	BDT 168.00	BDT 168.00
16GA-050 12V DC Gear Motor 285RPM	2	RoboticsBD	BDT 690.00	BDT 1,380.00
RC522 RFID Card Reader Module Kit...	1	RoboticsBD	BDT 179.00	BDT 179.00
Servo Motor Micro SG90 360 Degree Continuous Rotation	1	RoboticsBD	BDT 170.00	BDT 170.00
Female Pin Header 2.54mm Pitch - White	2	RoboticsBD	BDT 20.00	BDT 40.00
Male Pin Header 2.54mm Pitch - White	2	RoboticsBD	BDT 15.00	BDT 30.00
JST connectors (male+female) Pair 95mm	1	RoboticsBD	BDT 60.00	BDT 60.00
AA Battery Holder 3S	1	RoboticsBD	BDT 40.00	BDT 40.00
N20 DC Gear Motor Wheel 3PI miniQ Car wheel Tyre	2	RoboticsBD	BDT 80.00	BDT 160.00
Pololu Ball Caster (12 mm Metal Ball)	1	Electronics.Com.BD	40.00	40.00
12x18cm Double Side Prototype... PCB Board	1	Electronics.Com.BD	261.00	261.00
Rang & Logens	1	Local	40.00	40.00
Buck module + push button	1	Local	75.00	75.00
Buck module + headerpin	1	Cybernatics	185.00	185.00
Breadboard + jumper wire	1	Local	330.00	330.00
Component Subtotal	**29**			**BDT 4,403.00**
RoboticsBD Delivery Charge	1	RoboticsBD	BDT 99.00	BDT 99.00
Electronics.Com.BD Delivery Charge	1	Electronics.Com.BD	130.00	130.00
Grand Total				**BDT 4,632.00**

5 Conclusion and Future Work

5.1 Conclusion

This project successfully addressed the pressing issues of inefficiency, hygiene, and food quality degradation in small cafe and food court environments. The primary objective was to design and build a low-cost, automated, and contactless delivery system, and this objective was fully met.

Our solution, "PROJECT QUICKY," moved beyond the initial proposal resulting in a more robust and practical prototype. We successfully developed a robotic car controlled reliably over a local Wi-Fi network via a web-based interface. Furthermore, we integrated a secure, temperature-controlled compartment—capable of both heating and cooling—that is unlocked using a specific, authorized RFID card.

Testing confirmed that the system performs as designed: the Wi-Fi control is responsive over a practical range, the RFID-servo lock is 100% accurate in authenticating the correct user, and the thermal compartment successfully maintains food temperature. The addition of the customer kiosk UI design (Section 2.4) provides a complete, end-to-end vision for a fully automated service. This project serves as a successful proof-of-concept, demonstrating that a smart, IoT-based delivery system is a viable and effective solution for modernizing small-scale food service.

5.2 Future Scope and Improvements

While the prototype is successful, its design provides a strong foundation for many future upgrades. The following improvements are recommended for a production-ready model:

- **Autonomous Navigation:** The most significant upgrade would be to add obstacle avoidance sensors (like ultrasonic or LiDAR) and pathfinding algorithms, allowing the robot to navigate to tables autonomously instead of being manually driven.
- **Dedicated Mobile App:** While the web server is functional, a custom-built mobile app (for Android or iOS) would provide a more polished, responsive, and professional interface for the staff operator.
- **Full Kiosk Integration:** The next logical step is to fully build and integrate the "vending machine" Kiosk UI (from Section 2.4). This would create a seamless, end-to-end system where a customer's order from the kiosk is automatically sent to the kitchen and paired with an RFID card.
- **Smart Battery Management:** The high power draw of the Peltier module is a limitation. Future work should include a larger, dedicated battery for the thermal system and a "return to base" function that allows the robot to autonomously navigate to a charging dock when its battery is low.
- **Multi-Order Capacity:** The chassis could be scaled to include multiple compartments, allowing the robot to service several orders in a single trip.

References

[1] Random Nerd Tutorials. "ESP32 Wi-Fi Car Robot (Arduino IDE)." Available: https://randomnerdtutorials.com/esp32-wi-fi-car-robot-arduino/ (Accessed: November 5, 2025)

[2] Random Nerd Tutorials. "ESP32 MFRC522 RFID Reader (Arduino IDE)." Available: https://randomnerdtutorials.com/esp32-mfrc522-rfid-reader-arduino/ (Accessed: November 6, 2025)

[3] Random Nerd Tutorials. "ESP32 Servo Motor Web Server (Arduino IDE)." Available: https://randomnerdtutorials.com/esp32-servo-motor-web-server-arduino-ide/ (Accessed: November 6, 2025)

[4] Arduino. "Arduino IDE Documentation." Available: https://docs.arduino.cc/software/ide/ (Accessed: November 4, 2025)

[5] YouTube. "Various project tutorials." Available: https://www.youtube.com/ (Accessed: November 4, 2025)