PART 4: The Challenge

Background: Smart Parking Barrier System.

A local university wants to implement a smart parking barrier system at its main gate.

The system should:

* Automatically open the barrier for registered staff and students.
* Count the number of cars in the parking lot.
* Display the number of available spots.
* Trigger an alert if someone tries to enter when the parking is full.

Step 1: Understand and Define the Problem (Analyse)

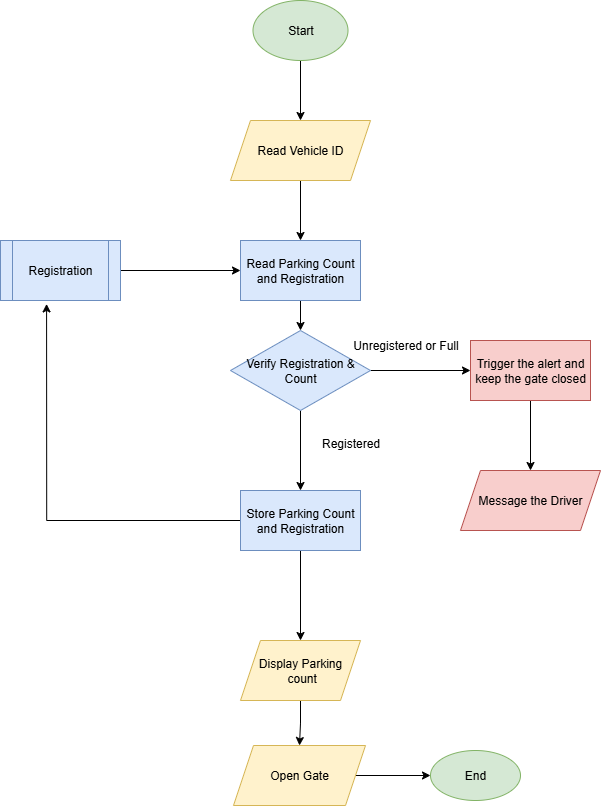
A local university plans to implement a smart parking barrier system at its main gate, designed to automatically open for registered staff and students. Count the number of cars in the parking lot, display the number of available spots, trigger an alert if someone tries to enter when the parking is full. The objective is to simulate and implement this systems logic using the integrated problem-solving process.

Step 2: Organize and Describe the Data

* Features: automatic access, vehicle count, alert for full capacity.
* Inputs: RFID/ID of user, sensor for car entry/exit, max capacity (e.g., 100).
* Outputs: barrier control (open/close), display of free spots, alert system.
* Assumptions: Each vehicle has a unique ID, sensors are reliable, capacity is fixed.
* Expected outcome: Only registered users can enter; system keeps track of space
* Sketch: Block diagram showing RFID reader → Logic → Barrier + Display +Alert

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Example** |
| Max Capacity | Constant | 100 Vehicles |
| Current Count | Variable | 75 vehicles |
| Vehicle ID Input | Input | A12345 |
| Is Registered | Decision | Yes/No |
| Barrier Control | output | Open/close |
| Display | output | 25 spaces available |
| Alert Trigger | output | On/off |

**Step 3: Plan the Solution (Design the Algorithm)**

****

**Step 4: Implement the Solution (Word Coding)**

* Read vehicle ID (input)
* Check if the vehicle is registered.
* If not registered, slam the boom gate over the bumper of the car (that is, do not open the gate).
* If registered but carpark is full ,do not open the gate and gently message the driver the number of spots.
* If registered and there are enough spots, open the gate.
* Note that all transactions are being stored in a database called "Registrations".

**Step 5: Test and Refine the Solution (Debug and Verify)**

• Manual Test Cases:

Some assumptions:

• Max number of cars allowed: 100

• Registrations database is updated (actual car park spots availability)

The Tests:

• Registered user with 95 cars → Allow entry.

• Registered user with 101 cars → Alert + Entry Locked + Message.

• Unregistered user → Alert + Entry Locked + Message

* Some suggested improvements:

• Add logging feature for entries.

• Integrate a web dashboard for monitoring.

• Add override for emergency vehicles

Explore real-world implementation: Use Copilot to discuss how your system could be built using actual hardware (e.g., Arduino, Raspberry Pi).

| **Component** | **Purpose** |
| --- | --- |
| **Arduino Uno / Mega** | Microcontroller to control sensors and actuators |
| **Raspberry Pi** | Optional for advanced tasks like image processing or cloud connectivity |
| **RFID Reader / LPR Camera** | To identify vehicles via RFID tags or license plate recognition |
| **Ultrasonic Sensors / IR Sensors** | Detect vehicle presence and count available spots |
| **Servo Motor / Relay** | Operate the barrier gate |
| **LCD / LED Display** | Show parking count and messages |
| **Wi-Fi / GSM Module** | Send alerts or messages to drivers |
| **Push Buttons / Keypad** | Manual registration or override input |
| **Power Supply** | Provide stable voltage to all components |

**System Architecture**

**1. Vehicle Detection & Identification**

* **RFID tags** on vehicles read by an **RFID reader** connected to Arduino.
* Alternatively, use a **camera + Raspberry Pi** for license plate recognition (LPR).

**2. Parking Count Monitoring**

* **Ultrasonic sensors** placed at each spot detect occupancy.
* Arduino aggregates data to calculate available spots.

**3. Registration Verification**

* Store registered IDs in EEPROM or external database.
* Raspberry Pi can handle cloud-based verification for scalability.

Today

hi, I have drawn a flow chart for smart car parking barrier system , and here it is,Generate alternative solutions: Prompt Copilot to propose different ways to solve the problem or enhance your flowchart logic.

Thanks for sharing your flowchart, Abrarul! It's a solid foundation for a smart car parking barrier system—clear, logical, and ethically sound in its control of access. Let’s explore some **alternative solutions and enhancements** that could improve efficiency, user experience, and system robustness.

**🚀 Alternative Solutions & Enhancements**

**1. Pre-Entry Reservation System**

* **Logic Addition**: Allow users to reserve a parking spot via an app or web portal before arrival.
* **Flowchart Impact**: Add a “Check Reservation” node before “Read Vehicle ID.”
* **Benefit**: Reduces congestion and ensures availability for high-demand areas.

**2. Dynamic Pricing Based on Availability**

* **Logic Addition**: Introduce a “Calculate Parking Fee” node based on current demand.
* **Flowchart Impact**: Insert after “Verify Registration & Count.”
* **Benefit**: Encourages turnover and optimizes space usage.

**3. License Plate Recognition (LPR) Integration**

* **Logic Addition**: Replace manual ID reading with automated LPR.
* **Flowchart Impact**: Modify “Read Vehicle ID” to “Scan License Plate.”
* **Benefit**: Faster processing and reduced human error.

**4. Real-Time Occupancy Monitoring**

* **Logic Addition**: Use IoT sensors to track actual vehicle presence.
* **Flowchart Impact**: Add a “Verify Physical Occupancy” node after “Display Parking Count.”
* **Benefit**: Prevents false counts due to system errors or misuse.

**5. Unregistered Vehicle Handling**

* **Logic Addition**: Offer temporary access or redirect to visitor parking.
* **Flowchart Impact**: Add a decision branch after “Unregistered Vehicle” alert.
* **Benefit**: Improves inclusivity and flexibility.

**6. Emergency Override Protocol**

* **Logic Addition**: Allow emergency vehicles to bypass standard checks.
* **Flowchart Impact**: Insert “Emergency Vehicle?” decision before “Verify Registration.”
* **Benefit**: Ensures safety and compliance with emergency access needs.

**7. User Feedback Loop**

* **Logic Addition**: Collect driver feedback after parking.
* **Flowchart Impact**: Add “Prompt Feedback” node after “End.”
* **Benefit**: Continuous improvement and user satisfaction.

Would you like help visualizing one of these enhanced flowcharts or integrating a specific feature into your current design?

yes

Great! Let’s pick one enhancement and integrate it into your existing flowchart. Since you're an engineering student focused on ethical and technical dimensions, I suggest we start with something that balances **efficiency, inclusivity, and responsible design**.

**🔧 Selected Enhancement: Unregistered Vehicle Handling with Visitor Access**

This enhancement adds flexibility by allowing unregistered vehicles to be redirected to a visitor parking area or offered temporary access, rather than outright denial.

**🧠 Updated Flowchart Logic (Text Version)**

plaintext

Start

↓

Scan Vehicle ID

↓

Read Parking Count & Registration

↓

Is Vehicle Registered?

├── No → Is Visitor Parking Available?

│ ├── No → Alert + Gate Closed → Message Driver → End

│ └── Yes → Assign Visitor Spot → Update Count → Display Count → Open Gate → End

└── Yes → Is Parking Full?

├── Yes → Alert + Gate Closed → Message Driver → End

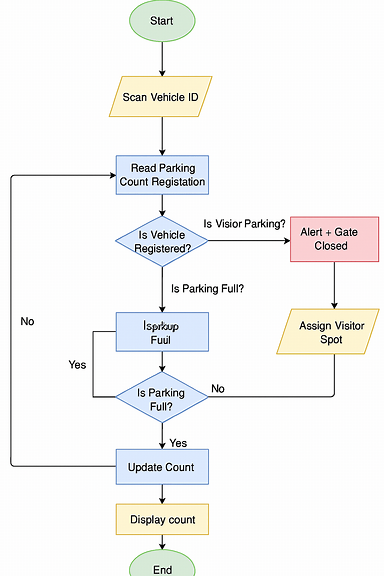
└── No → Update Count → Display Count → Open Gate → End

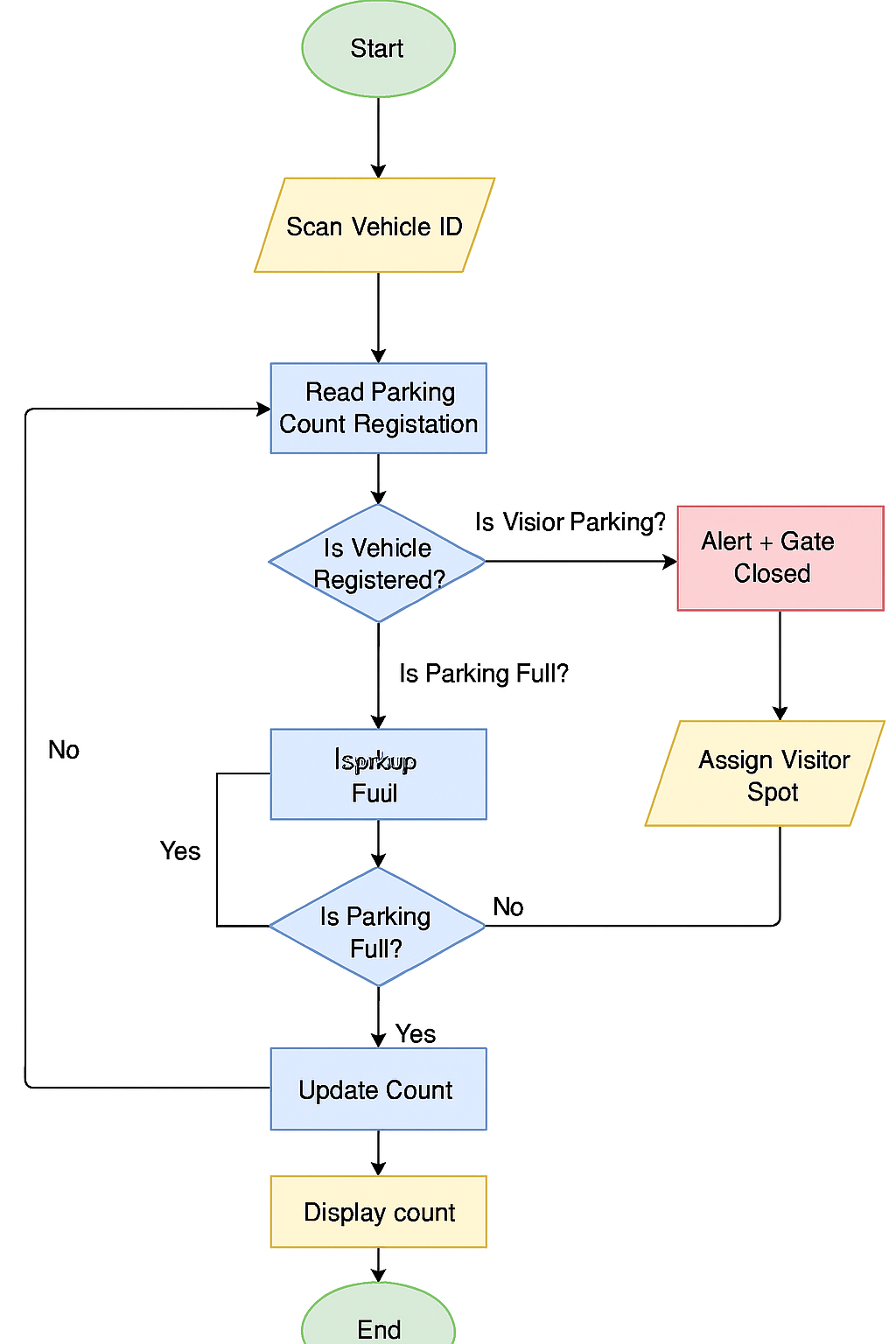
**✅ Benefits of This Enhancement**

* **Ethical Access**: Avoids exclusion of legitimate visitors.
* **System Robustness**: Adds fallback logic for edge cases.
* **User Experience**: Reduces frustration and improves public perception.

Would you like me to sketch this updated flowchart visually for you? Or would you prefer to explore another enhancement like emergency override or real-time occupancy tracking?

Can you illustrate this updated flowchart visually?





Here’s the visual illustration of your **enhanced smart car parking barrier system** flowchart with visitor access logic. <Image>

A flowchart depicts smart car parking barrier logic, with ovals, rectangles, and diamonds representing different steps and decision points, connected by arrows indicating the workflow. The chart uses different shapes and colors - ovals in green for "Start" and "End" steps, rectangles in light yellowish-orange for actions, and diamonds in light blue for decision points, while a light yellowish-orange rectangle specifically denotes assigning visitor spots.

The flowchart begins with a green oval labeled "Start" at the top. From it, a black arrow points downward to a light yellowish-orange rectangle labeled "Scan Vehicle ID." Another black arrow leads downward to a light blue rectangle labeled "Read Parking Count & Registration."

A light blue diamond below this rectangle, labeled "Is Vehicle Registered?", forms the first decision point, branching into two paths. A black arrow labeled "No" goes to the right, connecting to another light blue diamond labeled "Is Visitor Parking Available?". This diamond also branches into two paths: one black arrow labeled "No" points right to a light red rectangle labeled "Alert + Gate Closed," while a second black arrow labeled "Yes" points downward to a light yellowish-orange rectangle labeled "Assign Visitor Spot." Following the light red "Alert + Gate Closed" rectangle to the right, there’s an arrow leading downward to a light red rectangle labeled "Message Driver," which connects to the green "End" oval through another arrow.

From the "Assign Visitor Spot" rectangle, a black arrow leads downward to another light blue rectangle labeled "Update Count," which connects via another black arrow to a light yellowish-orange rectangle labeled "Display Count." This rectangle then links downward to another light yellowish-orange rectangle labeled "Open Gate," which ultimately connects to the green "End" oval at the bottom through another black arrow.

Returning to the "Is Vehicle Registered?" decision point, a black arrow labeled "Yes" leads downward to another light blue diamond labeled "Is Parking Full?", which also branches into two paths. A black arrow labeled "Yes" points right to the same light red rectangle labeled "Alert + Gate Closed," and a black arrow labeled "No" leads downward to the light blue "Update Count" rectangle, which continues to the remaining steps.

The flowchart uses black outlines for each shape and black text labels. The colors include green for the start and end, light yellowish-orange for actions, light blue for decision points, and light red for alerting actions.

</Image>

Would you like to explore how this system could be implemented using sensors, microcontrollers, or software logic next?

Explore real-world implementation: Use Copilot to discuss how your system could be built using actual hardware (e.g., Arduino, Raspberry Pi).

Absolutely, Abrarul! Let’s explore how your **Smart Car Parking Barrier System** could be implemented in the real world using hardware like **Arduino** or **Raspberry Pi**, integrating sensors, actuators, and communication modules.

**🧰 Hardware Components Overview**

| **Component** | **Purpose** |
| --- | --- |
| **Arduino Uno / Mega** | Microcontroller to control sensors and actuators |
| **Raspberry Pi** | Optional for advanced tasks like image processing or cloud connectivity |
| **RFID Reader / LPR Camera** | To identify vehicles via RFID tags or license plate recognition |
| **Ultrasonic Sensors / IR Sensors** | Detect vehicle presence and count available spots |
| **Servo Motor / Relay** | Operate the barrier gate |
| **LCD / LED Display** | Show parking count and messages |
| **Wi-Fi / GSM Module** | Send alerts or messages to drivers |
| **Push Buttons / Keypad** | Manual registration or override input |
| **Power Supply** | Provide stable voltage to all components |

**🔌 System Architecture**

**1. Vehicle Detection & Identification**

* **RFID tags** on vehicles read by an **RFID reader** connected to Arduino.
* Alternatively, use a **camera + Raspberry Pi** for license plate recognition (LPR).

**2. Parking Count Monitoring**

* **Ultrasonic sensors** placed at each spot detect occupancy.
* Arduino aggregates data to calculate available spots.

**3. Registration Verification**

* Store registered IDs in EEPROM or external database.
* Raspberry Pi can handle cloud-based verification for scalability.

**4. Decision Logic**

* Arduino processes inputs:
  + If registered and space available → trigger gate open.
  + If unregistered → check visitor spot availability.
  + If full → send alert via GSM or Wi-Fi.

**5. Gate Control**

* **Servo motor** or **DC motor with relay** opens/closes the barrier.
* Controlled via Arduino based on logic outcome.

**6. User Interface**

* **LCD display** shows parking count and status.
* Optional **touchscreen or keypad** for manual registration.

Today

hi, I have drawn a flow chart for smart car parking barrier system , and here it is,Generate alternative solutions: Prompt Copilot to propose different ways to solve the problem or enhance your flowchart logic.

Thanks for sharing your flowchart, Abrarul! It's a solid foundation for a smart car parking barrier system—clear, logical, and ethically sound in its control of access. Let’s explore some **alternative solutions and enhancements** that could improve efficiency, user experience, and system robustness.

**🚀 Alternative Solutions & Enhancements**

**1. Pre-Entry Reservation System**

* **Logic Addition**: Allow users to reserve a parking spot via an app or web portal before arrival.
* **Flowchart Impact**: Add a “Check Reservation” node before “Read Vehicle ID.”
* **Benefit**: Reduces congestion and ensures availability for high-demand areas.

**2. Dynamic Pricing Based on Availability**

* **Logic Addition**: Introduce a “Calculate Parking Fee” node based on current demand.
* **Flowchart Impact**: Insert after “Verify Registration & Count.”
* **Benefit**: Encourages turnover and optimizes space usage.

**3. License Plate Recognition (LPR) Integration**

* **Logic Addition**: Replace manual ID reading with automated LPR.
* **Flowchart Impact**: Modify “Read Vehicle ID” to “Scan License Plate.”
* **Benefit**: Faster processing and reduced human error.

**4. Real-Time Occupancy Monitoring**

* **Logic Addition**: Use IoT sensors to track actual vehicle presence.
* **Flowchart Impact**: Add a “Verify Physical Occupancy” node after “Display Parking Count.”
* **Benefit**: Prevents false counts due to system errors or misuse.

**5. Unregistered Vehicle Handling**

* **Logic Addition**: Offer temporary access or redirect to visitor parking.
* **Flowchart Impact**: Add a decision branch after “Unregistered Vehicle” alert.
* **Benefit**: Improves inclusivity and flexibility.

**6. Emergency Override Protocol**

* **Logic Addition**: Allow emergency vehicles to bypass standard checks.
* **Flowchart Impact**: Insert “Emergency Vehicle?” decision before “Verify Registration.”
* **Benefit**: Ensures safety and compliance with emergency access needs.

**7. User Feedback Loop**

* **Logic Addition**: Collect driver feedback after parking.
* **Flowchart Impact**: Add “Prompt Feedback” node after “End.”
* **Benefit**: Continuous improvement and user satisfaction.

Would you like help visualizing one of these enhanced flowcharts or integrating a specific feature into your current design?

yes

Great! Let’s pick one enhancement and integrate it into your existing flowchart. Since you're an engineering student focused on ethical and technical dimensions, I suggest we start with something that balances **efficiency, inclusivity, and responsible design**.

**🔧 Selected Enhancement: Unregistered Vehicle Handling with Visitor Access**

This enhancement adds flexibility by allowing unregistered vehicles to be redirected to a visitor parking area or offered temporary access, rather than outright denial.

**🧠 Updated Flowchart Logic (Text Version)**

plaintext

Start

↓

Scan Vehicle ID

↓

Read Parking Count & Registration

↓

Is Vehicle Registered?

├── No → Is Visitor Parking Available?

│ ├── No → Alert + Gate Closed → Message Driver → End

│ └── Yes → Assign Visitor Spot → Update Count → Display Count → Open Gate → End

└── Yes → Is Parking Full?

├── Yes → Alert + Gate Closed → Message Driver → End

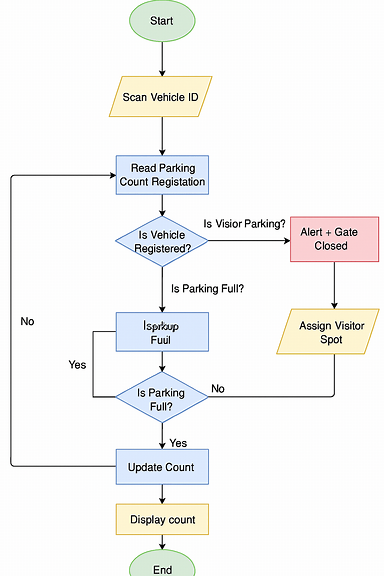
└── No → Update Count → Display Count → Open Gate → End

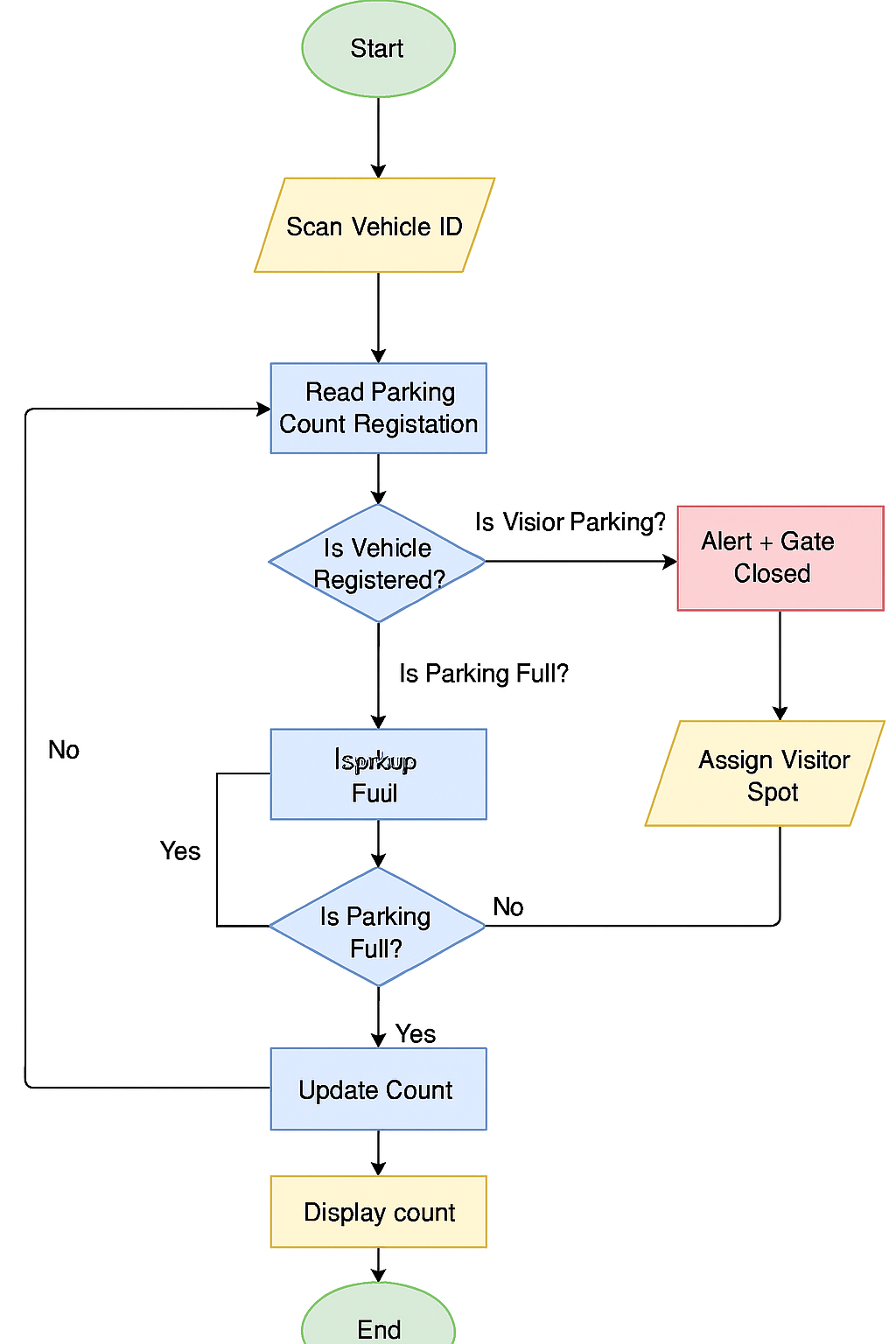
**✅ Benefits of This Enhancement**

* **Ethical Access**: Avoids exclusion of legitimate visitors.
* **System Robustness**: Adds fallback logic for edge cases.
* **User Experience**: Reduces frustration and improves public perception.

Would you like me to sketch this updated flowchart visually for you? Or would you prefer to explore another enhancement like emergency override or real-time occupancy tracking?

Can you illustrate this updated flowchart visually?





Here’s the visual illustration of your **enhanced smart car parking barrier system** flowchart with visitor access logic. <Image>

A flowchart depicts smart car parking barrier logic, with ovals, rectangles, and diamonds representing different steps and decision points, connected by arrows indicating the workflow. The chart uses different shapes and colors - ovals in green for "Start" and "End" steps, rectangles in light yellowish-orange for actions, and diamonds in light blue for decision points, while a light yellowish-orange rectangle specifically denotes assigning visitor spots.

The flowchart begins with a green oval labeled "Start" at the top. From it, a black arrow points downward to a light yellowish-orange rectangle labeled "Scan Vehicle ID." Another black arrow leads downward to a light blue rectangle labeled "Read Parking Count & Registration."

A light blue diamond below this rectangle, labeled "Is Vehicle Registered?", forms the first decision point, branching into two paths. A black arrow labeled "No" goes to the right, connecting to another light blue diamond labeled "Is Visitor Parking Available?". This diamond also branches into two paths: one black arrow labeled "No" points right to a light red rectangle labeled "Alert + Gate Closed," while a second black arrow labeled "Yes" points downward to a light yellowish-orange rectangle labeled "Assign Visitor Spot." Following the light red "Alert + Gate Closed" rectangle to the right, there’s an arrow leading downward to a light red rectangle labeled "Message Driver," which connects to the green "End" oval through another arrow.

From the "Assign Visitor Spot" rectangle, a black arrow leads downward to another light blue rectangle labeled "Update Count," which connects via another black arrow to a light yellowish-orange rectangle labeled "Display Count." This rectangle then links downward to another light yellowish-orange rectangle labeled "Open Gate," which ultimately connects to the green "End" oval at the bottom through another black arrow.

Returning to the "Is Vehicle Registered?" decision point, a black arrow labeled "Yes" leads downward to another light blue diamond labeled "Is Parking Full?", which also branches into two paths. A black arrow labeled "Yes" points right to the same light red rectangle labeled "Alert + Gate Closed," and a black arrow labeled "No" leads downward to the light blue "Update Count" rectangle, which continues to the remaining steps.

The flowchart uses black outlines for each shape and black text labels. The colors include green for the start and end, light yellowish-orange for actions, light blue for decision points, and light red for alerting actions.

</Image>

Would you like to explore how this system could be implemented using sensors, microcontrollers, or software logic next?

* Generate alternative solutions: Prompt Copilot to propose different ways to solve the problem or enhance your flowchart logic.

1. Pre-Entry Reservation System

Logic Addition: Allow users to reserve a parking spot via an app or web portal before arrival.

Flowchart Impact: Add a “Check Reservation” node before “Read Vehicle ID.”

Benefit: Reduces congestion and ensures availability for high-demand areas.

2. Dynamic Pricing Based on Availability

Logic Addition: Introduce a “Calculate Parking Fee” node based on current demand.

Flowchart Impact: Insert after “Verify Registration & Count.”

Benefit: Encourages turnover and optimizes space usage.

Today

hi, I have drawn a flow chart for smart car parking barrier system , and here it is,Generate alternative solutions: Prompt Copilot to propose different ways to solve the problem or enhance your flowchart logic.

Thanks for sharing your flowchart, Abrarul! It's a solid foundation for a smart car parking barrier system—clear, logical, and ethically sound in its control of access. Let’s explore some alternative solutions and enhancements that could improve efficiency, user experience, and system robustness.

🚀 Alternative Solutions & Enhancements

1. Pre-Entry Reservation System

Logic Addition: Allow users to reserve a parking spot via an app or web portal before arrival.

Flowchart Impact: Add a “Check Reservation” node before “Read Vehicle ID.”

Benefit: Reduces congestion and ensures availability for high-demand areas.

2. Dynamic Pricing Based on Availability

Logic Addition: Introduce a “Calculate Parking Fee” node based on current demand.

Flowchart Impact: Insert after “Verify Registration & Count.”

Benefit: Encourages turnover and optimizes space usage.

3. License Plate Recognition (LPR) Integration

Logic Addition: Replace manual ID reading with automated LPR.

Flowchart Impact: Modify “Read Vehicle ID” to “Scan License Plate.”

Benefit: Faster processing and reduced human error.

4. Real-Time Occupancy Monitoring

Logic Addition: Use IoT sensors to track actual vehicle presence.

Flowchart Impact: Add a “Verify Physical Occupancy” node after “Display Parking Count.”

Benefit: Prevents false counts due to system errors or misuse.

5. Unregistered Vehicle Handling

Logic Addition: Offer temporary access or redirect to visitor parking.

Flowchart Impact: Add a decision branch after “Unregistered Vehicle” alert.

Benefit: Improves inclusivity and flexibility.

6. Emergency Override Protocol

Logic Addition: Allow emergency vehicles to bypass standard checks.

Flowchart Impact: Insert “Emergency Vehicle?” decision before “Verify Registration.”

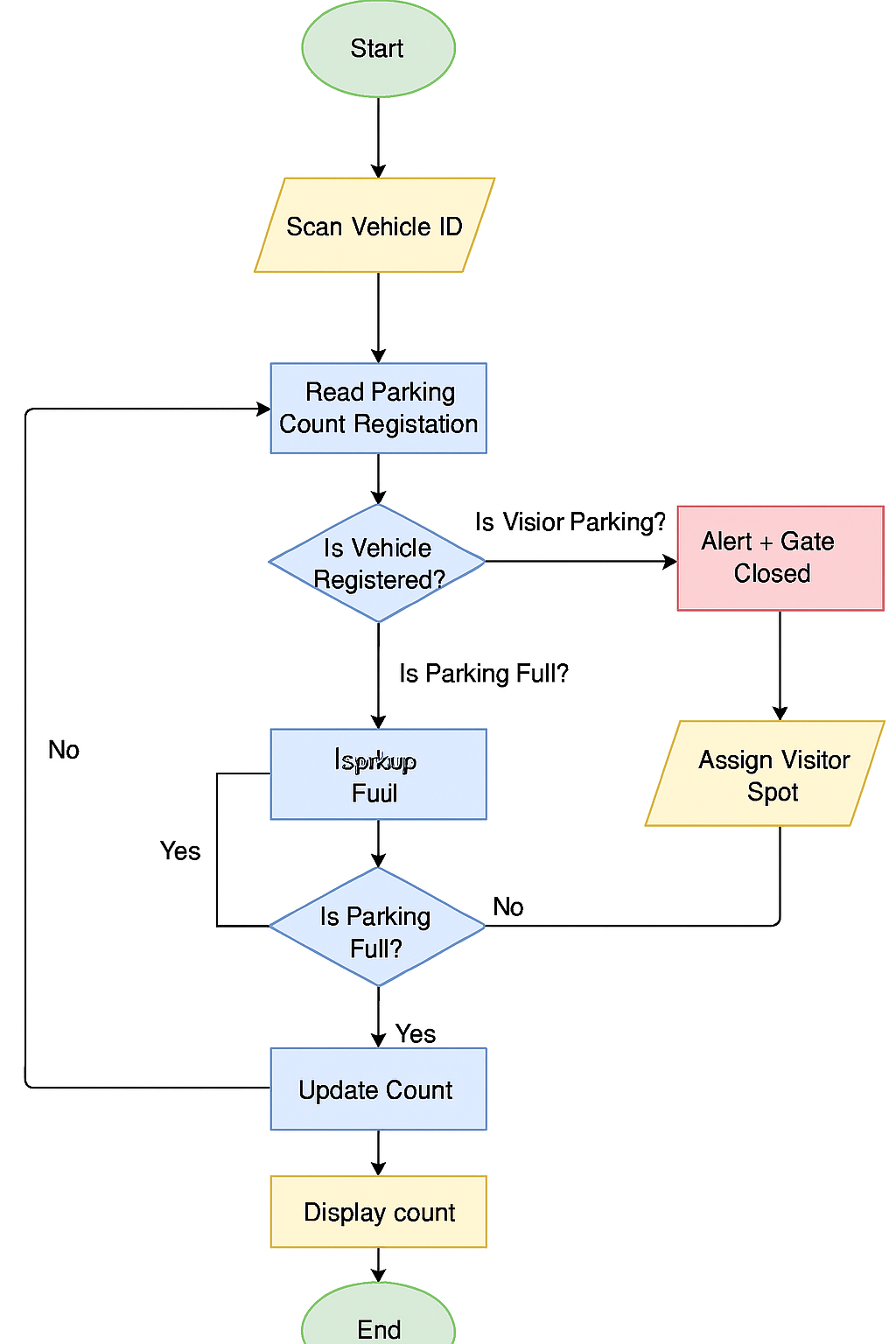
Benefit: Ensures safety and compliance with emergency access needs.

7. User Feedback Loop

Logic Addition: Collect driver feedback after parking.

Flowchart Impact: Add “Prompt Feedback” node after “End.”

Benefit: Continuous improvement and user satisfaction.



* Explore real-world implementation: Use Copilot to discuss how your system could be built using actual hardware (e.g., Arduino, Raspberry Pi).

**🧰 Hardware Components Overview**

| **Component** | **Purpose** |
| --- | --- |
| **Arduino Uno / Mega** | Microcontroller to control sensors and actuators |
| **Raspberry Pi** | Optional for advanced tasks like image processing or cloud connectivity |
| **RFID Reader / LPR Camera** | To identify vehicles via RFID tags or license plate recognition |
| **Ultrasonic Sensors / IR Sensors** | Detect vehicle presence and count available spots |
| **Servo Motor / Relay** | Operate the barrier gate |
| **LCD / LED Display** | Show parking count and messages |
| **Wi-Fi / GSM Module** | Send alerts or messages to drivers |
| **Push Buttons / Keypad** | Manual registration or override input |
| **Power Supply** | Provide stable voltage to all components |

**🔌 System Architecture**

**1. Vehicle Detection & Identification**

* **RFID tags** on vehicles read by an **RFID reader** connected to Arduino.
* Alternatively, use a **camera + Raspberry Pi** for license plate recognition (LPR).

**2. Parking Count Monitoring**

* **Ultrasonic sensors** placed at each spot detect occupancy.
* Arduino aggregates data to calculate available spots.

**3. Registration Verification**

* Store registered IDs in EEPROM or external database.
* Raspberry Pi can handle cloud-based verification for scalability.

**4. Decision Logic**

* Arduino processes inputs:
  + If registered and space available → trigger gate open.
  + If unregistered → check visitor spot availability.
  + If full → send alert via GSM or Wi-Fi.

**5. Gate Control**

* **Servo motor** or **DC motor with relay** opens/closes the barrier.
* Controlled via Arduino based on logic outcome.

**6. User Interface**

* **LCD display** shows parking count and status.
* Optional **touchscreen or keypad** for manual registration.

**7. Communication**

* **Wi-Fi module (ESP8266)** or **GSM module** sends SMS or app notifications.
* Raspberry Pi can host a web dashboard for admin control.

Sample Arduino Logic Snippet:

if (isRegistered(vehicleID) && parkingAvailable()) {

openGate();

updateCount();

displayCount();

} else if (!isRegistered(vehicleID) && visitorSpotAvailable()) {

assignVisitorSpot();

openGate();

updateCount();

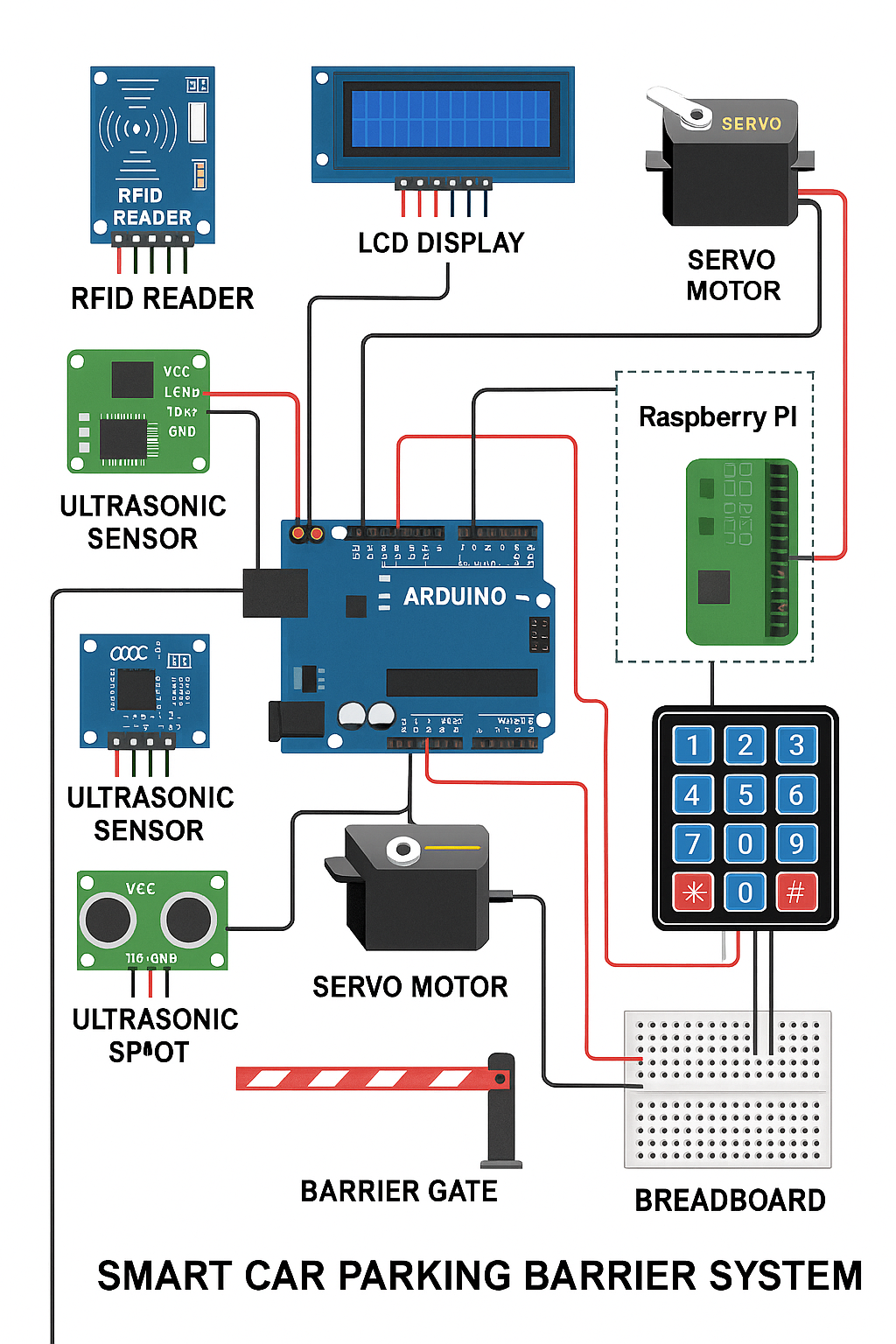
displayCount();

} else {

sendAlert(vehicleID);

keepGateClosed();

}



Here’s a visual schematic of your smart car parking barrier system using Arduino and Raspberry Pi: <Image>A detailed schematic diagram of a smart car parking barrier system illustrates connections between various components including an Arduino, Raspberry Pi, sensors, and actuators. The Arduino board is blue, rectangular, and labeled "ARDUINO" in white letters. It has digital pins (0 to 13), analog pins (A0 to A5), power pins (3.3V, 5V, GND, VIN), and communication pins (SCL, SDA). The Raspberry Pi board is green with a 40-pin header labeled on its right side, and a white dashed box surrounds it labeled "Raspberry Pi".

On the left side of the diagram, an "RFID Reader" is depicted as a rectangular blue module with a white antenna symbol emitting waves. It has four pins labeled "VCC," "MOSI," "SCK," and "RST," which are connected to the Arduino. Below it is an "Ultrasonic Sensor," represented by a green rectangular module with two circular sensors. The pins are labeled "VCC," "ECHO," "TRIG," and "GND," and they are connected to the Arduino.

At the top, a small blue LCD display labeled "LCD DISPLAY" has sixteen rectangles representing characters. It is connected to the Arduino through wires. On the right side, a "GSM / Wi-Fi Module" is represented as a green PCB with an antenna symbol and SMA connector, and has four pins labeled "VCC," "TX," "RX," and "GND," connected to the Arduino.

A servo motor, depicted as a gray rectangular box with a yellow circular horn, is connected to the Arduino through three wires. Below it, there is a barrier gate described as a red and white striped horizontal arm pivoting on a black support. To the right of the Raspberry Pi, a breadboard with a 3x4 keypad is shown. The keypad has twelve buttons arranged in a 3x4 grid with numbers 1-9, an asterisk (\*), zero (0), and hash (#). The keypad has pins connected to the Raspberry Pi through the breadboard.

The diagram with solid lines shows red lines representing power connections, black for ground, and other colors for data or control connections. Labels for each component and module are in uppercase black letters. The RFID Reader, Ultrasonic Sensor, LCD Display, GSM / Wi-Fi Module, Servo Motor, Barrier Gate, Raspberry Pi, Breadboard, and Keypad are clearly labeled. The color-coded lines and the arrangement of the components provide a clear visual representation of the system's functionality.

Camera: straight-on, two-dimensional schematic view shot.

Lighting: clear and even.

Depth of field: deep (all components and connections sharp and legible).

Color balance: distinct colors for different connections (red, black, blue, green). Blue for Arduino, green for Raspberry Pi and other modules.

Image quality: excellent.

Focal points: Arduino and Raspberry Pi.</Image>