

Phase – 5

Documentation : IOT Smart Parking Using python



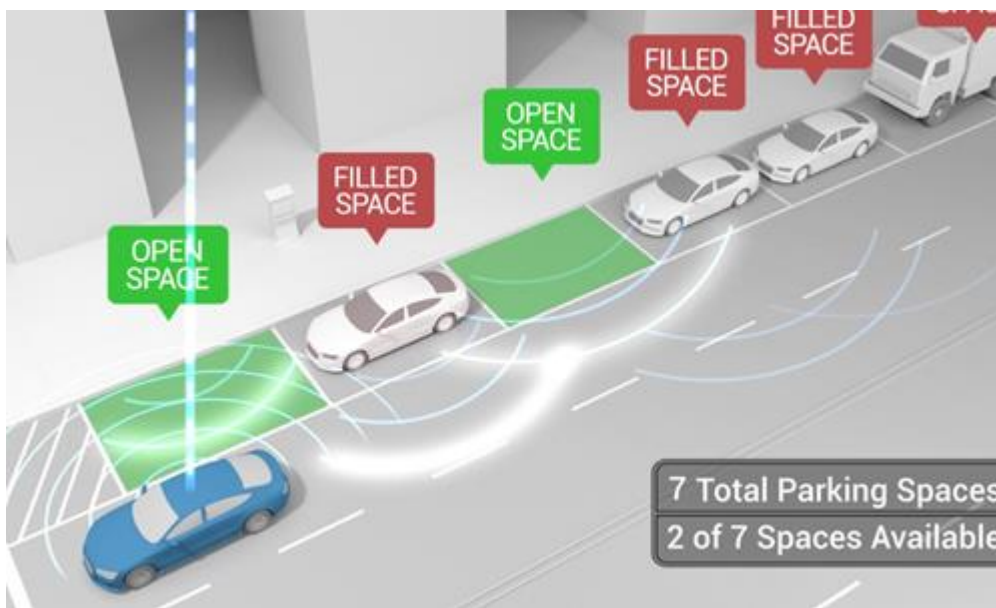
Team Members

M. Mohamed Thaslim	821021104031
R. Vadivel	821021104051
S. Subash Chandrabose	821021104046
M. Dhanush	821021104016

Smart Parking Project's Objectives:

A Smart Parking System using IoT (Internet of Things) is designed to improve parking management, enhance user experience, and optimize space utilization in parking facilities. The project objectives for such a system can include:

1. Real-time Parking Availability: Implement sensors and cameras to monitor parking spaces and provide real-time information to drivers about available parking spots. The system should reduce the time and frustration associated with finding a parking space.



2. Mobile Application Integration: Develop a user-friendly mobile app that allows drivers to check parking availability, reserve spots, and make payments from their smartphones. The app should also provide navigation assistance to guide users to their reserved parking spaces.



3. **Cost-Effective Operation:** Optimize the usage of parking facilities by efficiently allocating parking spaces. Reduce operational costs and maximize revenue by ensuring that all parking spaces are utilized effectively.



4. **Environmental Impact:** Reduce the environmental impact by minimizing the time and fuel wasted while searching for parking. By directing users to available spaces, the system can help reduce congestion and emissions.



5. **Security and Surveillance:** Enhance security by integrating surveillance cameras into the system, which can monitor the parking area for safety and deter illegal activities.



6. Payment and Billing: Enable cashless transactions by allowing users to make payments through the mobile app, credit/debit cards, or other electronic means. Implement flexible billing options, including hourly rates or monthly subscriptions.



7. **User Feedback and Ratings:** Collect user feedback and ratings to continuously improve the parking system. Gather data on user satisfaction and use this information to make enhancements.

8. **Data Analytics:** Utilize data collected from the system to gain insights into parking patterns, peak hours, and user behavior. This data can be used for future planning and to optimize parking facility design.



9. **Sustainability:** Explore ways to reduce energy consumption in the parking facility by using energy-efficient lighting and HVAC systems. Implement solutions to reduce water usage and waste.

10. **Scalability:** Design the system to be scalable, allowing for easy expansion to accommodate more parking spaces or additional parking facilities in the future.

11. **Accessibility and Inclusivity:** Ensure that the system is accessible to all, including individuals with disabilities, by providing appropriate accommodations and features.

12. **Public-Private Partnerships:** Explore partnerships with local businesses or government agencies to promote the adoption and integration of the smart parking system within the community.

13. **Traffic Management:** Integrate the smart parking system with traffic management authorities to help alleviate traffic congestion and promote smoother traffic flow by directing vehicles to available parking spaces.

14. **Maintenance and Reliability:** Develop a proactive maintenance plan to ensure the system's sensors, cameras, and other components are functioning reliably. Implement remote monitoring and alerts for system issues.

15. **Privacy and Data Security:** Prioritize the privacy and security of user data collected by the system, adhering to relevant data protection regulations.

By setting clear objectives for your Smart Parking System project, you can create a well-defined roadmap for its development and ensure that it meets the needs of both parking facility operators and users.

Smart Parking Device Setup :

Creating a smart parking system using IoT devices involves a combination of hardware and software components to monitor and manage parking spaces efficiently. Here is a step-by-step guide to setting up a basic smart parking system:

Hardware Components:

Parking Sensors: You'll need proximity sensors (e.g., ultrasonic or infrared) to detect the presence of vehicles in parking spaces. These sensors will be placed in each parking spot.



Infrared



Ultrasonic

Microcontrollers: Each parking sensor should be connected to a microcontroller (e.g., Arduino, Raspberry Pi, or specialized IoT microcontrollers like ESP8266 or ESP32). These microcontrollers will collect data from the sensors and communicate it to a central system.



ESP8266 and ESP32



Communication Modules: The microcontrollers should have communication modules (e.g., Wi-Fi, Bluetooth, LoRa, or cellular) to send data to the central system.



Central Server: You need a central server that collects data from all the sensors, processes it, and provides a user interface for monitoring and managing parking spaces. This server can be hosted in the cloud or on-premises.

User Interface: Create a user-friendly interface for users to check parking space availability, reserve parking spots, and pay for parking. This interface can be a mobile app or a web application.

Power Supply: Ensure a stable power supply for the sensors and microcontrollers, either through batteries or a wired power source.

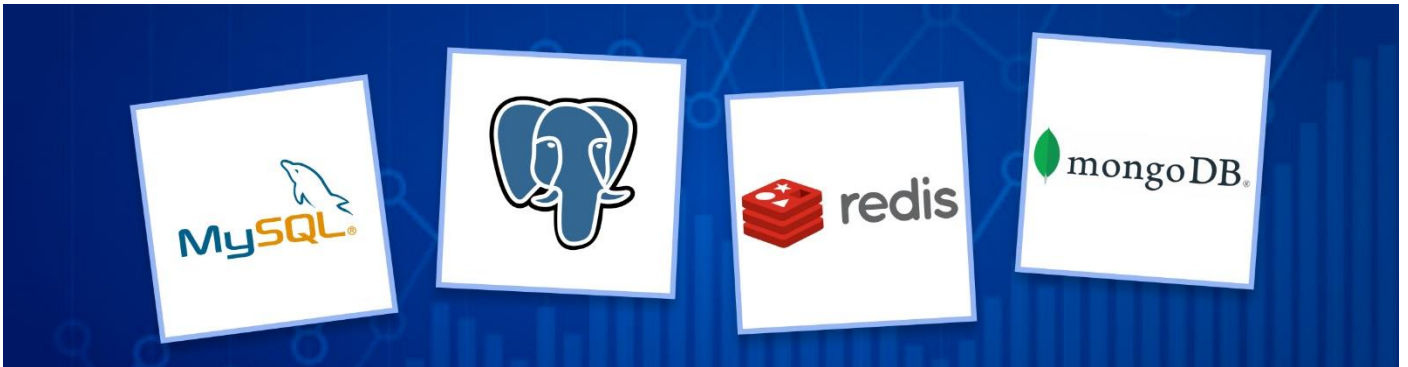
Software Components:

Sensor Data Processing: Develop firmware for the microcontrollers to process sensor data and send it to the central server. Use programming languages suitable for your microcontroller, such as C/C++ for Arduino or Python for Raspberry Pi.

Central Server Application: Develop the central server application to receive data from the sensors, store it in a database, and manage parking space information. You can use languages like Python, Node.js, or Java for server-side development.



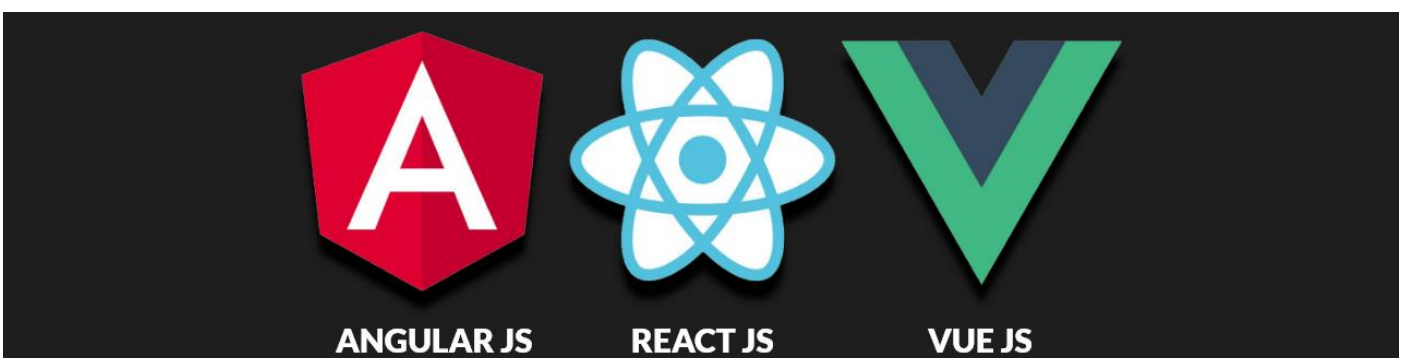
Database: Implement a database system to store information about parking spaces, reservations, and user data. Common choices include MySQL, PostgreSQL, or NoSQL databases like MongoDB.



Real-time Data Processing: Implement real-time processing for parking space availability. You can use tools like Apache Kafka or MQTT to handle real-time data streams.



User Interface Development: Create a user-friendly interface for both parking administrators and users. You can use web development frameworks like React, Angular, or Vue.js for the front-end.



Payment Integration: If your system includes paid parking, integrate payment gateways (e.g., PayPal, Stripe) into the user interface.



Security Considerations:

Secure communication between sensors, microcontrollers, and the central server using encryption and authentication.

Implement user authentication and authorization to protect user data and ensure secure transactions.

Regularly update and patch the software components to address security vulnerabilities.

Testing and Deployment:

1. Thoroughly test the system in a controlled environment to ensure all components work as expected.
2. Deploy the sensors in the parking spaces, connect them to the microcontrollers, and set up the central server.
3. Monitor the system's performance and make necessary adjustments.
4. Train parking attendants and users on how to use the smart parking system.

By following these steps, you can set up a basic smart parking system using IoT devices. Depending on your requirements and budget, you can expand the system's features and capabilities.

Smart Parking Platform Development :

Developing a smart parking system using an IoT (Internet of Things) platform involves integrating various hardware and software components to

efficiently manage and monitor parking spaces. Below are the key steps to develop a smart parking system using an IoT platform:

Define Requirements:

Clearly define the requirements and objectives of your smart parking system, including the number of parking spaces, the level of automation required, and the user interface.

Hardware Components:

Choose the appropriate IoT hardware components such as sensors, cameras, RFID readers, and barrier gates to monitor and control parking spaces. Common sensor types for parking systems include ultrasonic sensors, magnetic sensors, and infrared sensors.

IoT Platform Selection:

Select an IoT platform or a cloud service provider that offers IoT services, such as AWS IoT, Azure IoT, Google Cloud IoT, or a dedicated IoT platform like ThingSpeak, Ubidots, or Particle. Ensure it provides features like data storage, device management, and real-time data processing.

Sensor Deployment:

Install sensors in each parking space to detect vehicle presence and occupancy. Sensors should communicate with the IoT platform to transmit data.

Data Communication:

Establish a reliable communication network, such as Wi-Fi, LoRa, or cellular, for the sensors to transmit data to the IoT platform. Ensure data security and encryption for communication.

Data Processing:

Set up rules and algorithms on the IoT platform to process the sensor data. This can include detecting available parking spaces, predicting parking space availability, and managing reservations.

User Interface:

Develop a user-friendly mobile app or web interface for both administrators and parking users. Users can check the availability of parking spaces, make reservations, and pay for parking.

Payment Integration:

Integrate payment gateways to allow users to pay for parking through the app or website. You can also incorporate options like contactless payments, mobile wallets, or credit card payments.

Parking Guidance System:

Implement a parking guidance system to direct drivers to available parking spaces using LED displays, signage, or mobile notifications.

Mobile App Integration:

Create a mobile app for end-users to access parking information, make reservations, and navigate to available spots.

Security and Access Control:

Implement security measures to prevent unauthorized access or misuse of the parking system. Use RFID cards, license plate recognition, or mobile apps for access control.

Reporting and Analytics:

Collect and analyze data to generate reports on parking space utilization, revenue, and trends. This information can help optimize the system's performance.

Maintenance and Support:

Plan for regular maintenance of sensors and other hardware components, as well as providing customer support for users of the system.

Compliance and Regulations:

Ensure that your smart parking system complies with local regulations and privacy laws, especially concerning data collection and storage.

Testing and Optimization:

Thoroughly test the system to ensure its functionality and reliability. Continuously optimize the system based on user feedback and data analytics.

Scaling:

Consider future scalability and expansion of the system to accommodate more parking spaces or additional features.

Developing a smart parking system using an IoT platform is a complex project that requires a multidisciplinary team with expertise in IoT, software development, hardware integration, and user experience design. It also requires continuous monitoring and maintenance to ensure optimal performance and user satisfaction.

Code Implementation :

Creating a smart parking system using IoT involves several components, including sensors, a microcontroller or IoT platform, and a user interface. Below, I'll provide a high-level code implementation outline in Python for a simple IoT-based smart parking system using a Raspberry Pi and ultrasonic sensors.

Developing a smart parking system using an IoT (Internet of Things) platform involves integrating various hardware and software components to efficiently manage and monitor parking spaces. Below are the key steps to develop a smart parking system using an IoT platform

1.Hardware Setup:

1. Raspberry Pi (or any other IoT platform)
2. Ultrasonic sensors (HC-SR04 or similar)
3. LEDs (for indicating parking status)
4. Breadboard and jumper wires

2. Python Code Implementation:

```
import RPi.GPIO as GPIO

import time

# Set up GPIO pins for ultrasonic sensor and LEDs
TRIG_PIN = 23
ECHO_PIN = 24
LED_GREEN = 17
LED_RED = 27


GPIO.setmode(GPIO.BCM)
GPIO.setup(TRIG_PIN, GPIO.OUT)
GPIO.setup(ECHO_PIN, GPIO.IN)
GPIO.setup(LED_GREEN, GPIO.OUT)
GPIO.setup(LED_RED, GPIO.OUT)


# Function to measure distance using ultrasonic sensor
def measure_distance():
    GPIO.output(TRIG_PIN, GPIO.LOW)
    time.sleep(2) # To stabilize sensor
    GPIO.output(TRIG_PIN, GPIO.HIGH)
    time.sleep(0.00001)
    GPIO.output(TRIG_PIN, GPIO.LOW)


    while GPIO.input(ECHO_PIN) == 0:
        pulse_start = time.time()
```

```
while GPIO.input(ECHO_PIN) == 1:
    pulse_end = time.time()

    pulse_duration = pulse_end - pulse_start
    distance = pulse_duration * 34300 / 2 # Speed of sound is 343 m/s

    return distance

try:
    while True:
        distance = measure_distance()

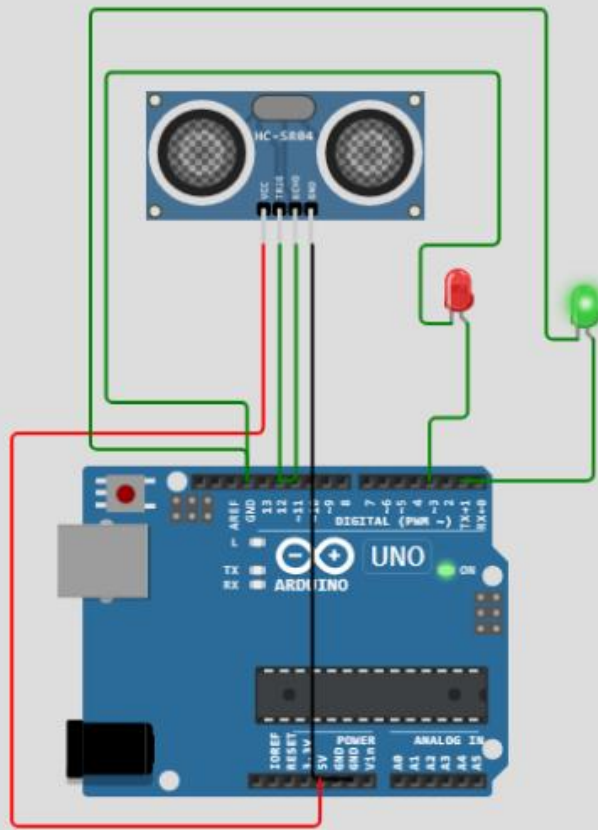
        if distance < 10: # Adjust this threshold for your parking space
            print("Parking spot occupied")
            GPIO.output(LED_GREEN, GPIO.LOW)
            GPIO.output(LED_RED, GPIO.HIGH)
        else:
            print("Parking spot vacant")
            GPIO.output(LED_GREEN, GPIO.HIGH)
            GPIO.output(LED_RED, GPIO.LOW)

        time.sleep(2) # Delay between measurements

except KeyboardInterrupt:
    GPIO.cleanup()
```

Simulation

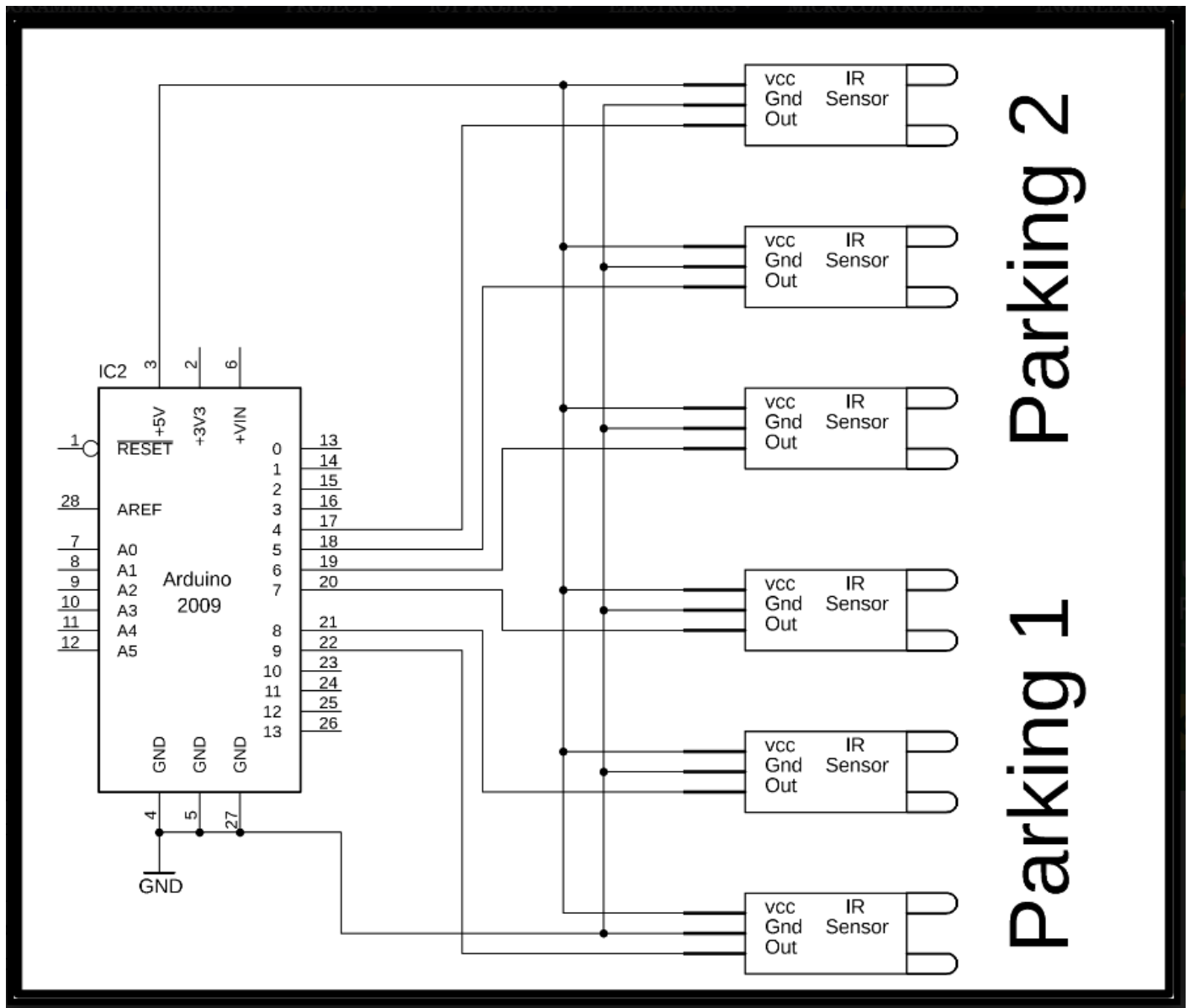
00:06.266 31%



This code will set up a Raspberry Pi to monitor the distance using an ultrasonic sensor. If the distance falls below a certain threshold, it will indicate that the parking spot is occupied using an LED. You can expand this system by sending data to a cloud platform or a mobile app for remote monitoring and notifications.

Remember to adjust the threshold distance and GPIO pin numbers as needed for your specific hardware setup. Additionally, you can integrate this code with a web-based or mobile app for remote access and notifications to create a complete smart parking system.

Parking Circuit Diagram :



This is the complete circuit diagram of the car parking monitoring system. The circuit diagram is designed in cadsoft eagle version 9.1.0. These are the six infrared sensors. Each infrared sensor represents a Slot.

Vcc of all the infrared sensors are connected together and are connected with Arduino's 5v. Similarly, the ground pins of all the infrared sensors are

connected together and are connected with the Arduino's ground. While the out pins of the infrared sensors are connected.

Smart Parking Code Implementation :

```
const int TRIG_PIN_1 = 4;
const int ECHO_PIN_1 = 12;
const int TRIG_PIN_2 = 2;
const int ECHO_PIN_2 = 5;

const int RED_PIN_1 = 27;
const int GREEN_PIN_1 = 26;

const int RED_PIN_2 = 33;
const int GREEN_PIN_2 = 32;
int counter = 0;

float duration_us_1, duration_us_2, distance_cm_1, distance_cm_2;

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

    pinMode(TRIG_PIN_1, OUTPUT);
    pinMode(ECHO_PIN_1, INPUT);
    pinMode(TRIG_PIN_2, OUTPUT);
    pinMode(ECHO_PIN_2, INPUT);

    pinMode(RED_PIN_1, OUTPUT);
    pinMode(GREEN_PIN_1, OUTPUT);

    pinMode(RED_PIN_2, OUTPUT);
    pinMode(GREEN_PIN_2, OUTPUT);
}

void loop()
{
    counter = 0;

    ultrasonic_1();
    ultrasonic_2();

    Serial.print("1: ");
    Serial.println(distance_cm_1);
    Serial.print("2: ");
    Serial.println(distance_cm_2);

    Serial.print("Counter: ");
    Serial.println(counter);
    Serial.println("");
}
```



```

void ultrasonic_1(){
    digitalWrite(TRIG_PIN_1, HIGH);
    delayMicroseconds(10);
    digitalWrite(TRIG_PIN_1, LOW);

    // measure duration of pulse from ECHO pin
    duration_us_1 = pulseIn(ECHO_PIN_1, HIGH);

    // calculate the distance
    distance_cm_1 = 0.017 * duration_us_1;

    if(distance_cm_1 < 50) {
        red_1();
        Serial.println("Slot 1 Terisi");
        counter++;
    } else {
        green_1();
    }
}

void ultrasonic_2(){
    digitalWrite(TRIG_PIN_2, HIGH);
    delayMicroseconds(10);
    digitalWrite(TRIG_PIN_2, LOW);

    // measure duration of pulse from ECHO pin
    duration_us_2 = pulseIn(ECHO_PIN_2, HIGH);

    // calculate the distance
    distance_cm_2 = 0.017 * duration_us_2;

    if(distance_cm_2 < 50) {
        red_2();
        Serial.println("Slot 2 Terisi");
        counter++;
    } else {
        green_2();
    }
}

void red_1(){
    digitalWrite(REDA_PIN_1, HIGH);
    digitalWrite(GREEN_PIN_1, LOW);
    delay(1000);
}

void red_2(){
    digitalWrite(REDA_PIN_2, HIGH);
    digitalWrite(GREEN_PIN_2, LOW);
    delay(1000);
}

void green_1(){
    digitalWrite(REDA_PIN_1, LOW);

```

```

digitalWrite(GREEN_PIN_1, HIGH);
delay(1000);
}

void green_2(){
  digitalWrite(RED_PIN_2, LOW);
  digitalWrite(GREEN_PIN_2, HIGH);
  delay(1000);
}

```

Screenshots Of Output:

The screenshot shows the Wokwi simulation environment. The code editor on the left contains the following code:

```

1  const int TRIG_PIN_1 = 4;
2  const int ECHO_PIN_1 = 12;
3  const int TRIG_PIN_2 = 2;
4  const int ECHO_PIN_2 = 5;
5
6  const int RED_PIN_1 = 27;
7  const int GREEN_PIN_1 = 26;
8
9  const int RED_PIN_2 = 33;
10 const int GREEN_PIN_2 = 32;
11 int counter = 0;
12
13 float duration_us_1, duration_us_2, distance_cm_1, distance_cm_2;
14
15
16 void setup()
17 {
18   Serial.begin(9600);
19
20   pinMode(TRIG_PIN_1, OUTPUT);
21   pinMode(ECHO_PIN_1, INPUT);
22   pinMode(TRIG_PIN_2, OUTPUT);
23   pinMode(ECHO_PIN_2, INPUT);
24
25   pinMode(RED_PIN_1, OUTPUT);
26   pinMode(GREEN_PIN_1, OUTPUT);
27
28   pinMode(RED_PIN_2, OUTPUT);
29   pinMode(GREEN_PIN_2, OUTPUT);
30 }
31
32 void loop()
33 {
34   counter = 0;
35
36   ultrasonic_1();
37   ultrasonic_2();
38
39   Serial.print("1: ");
40   Serial.println(distance_cm_1);
41   Serial.print("2: ");
42   Serial.println(distance_cm_2);

```

The simulation window on the right shows the hardware setup: an ESP32 microcontroller connected to two HC-SR04 ultrasonic sensors and two LEDs. The serial monitor at the bottom displays the initial distance readings for both slots:

```

2: 399.96
Counter: 0

1: 399.96
2: 399.96
Counter: 0

```

Parking slot one :

The screenshot shows the Wokwi simulation environment. The code editor on the left contains the same code as the previous screenshot. The simulation window on the right shows the same hardware setup: an ESP32 microcontroller connected to two HC-SR04 ultrasonic sensors and two LEDs. The serial monitor at the bottom displays the updated distance readings for both slots:

```

Slot 1 Parked
1: 1.99
2: 399.92
Counter: 1

Slot 1 Parked

```

Parking slot Full:

Repository search results

Inbox (160) - thunderhaslim@

Smart Parking System Objective

diagram smart parking system

Smart Parking - Wokwi ES

New Arduino Uno Project - W

wokwi.com/projects/379733262975564801

WOKWI

SAVE

SHARE

Docs

SIGN IN

sketch

Simulation

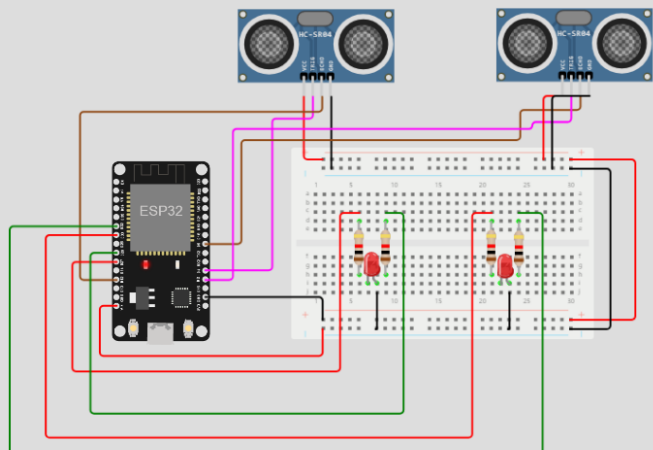
diag

Lib

Var

00:05.533

38%



Slot 1 Parked

Slot 2 Parked

1: 1.99

2: 1.99

Counter: 2

Slot 1 Parked

Wokwi logo

Pause

Stop

Thank You