

SMART PARKING

TEAM MEMBER

960321104027:BLESSY GOLD B.R

PROJECT:PHASE 3:DEVELOPMENT PART 1



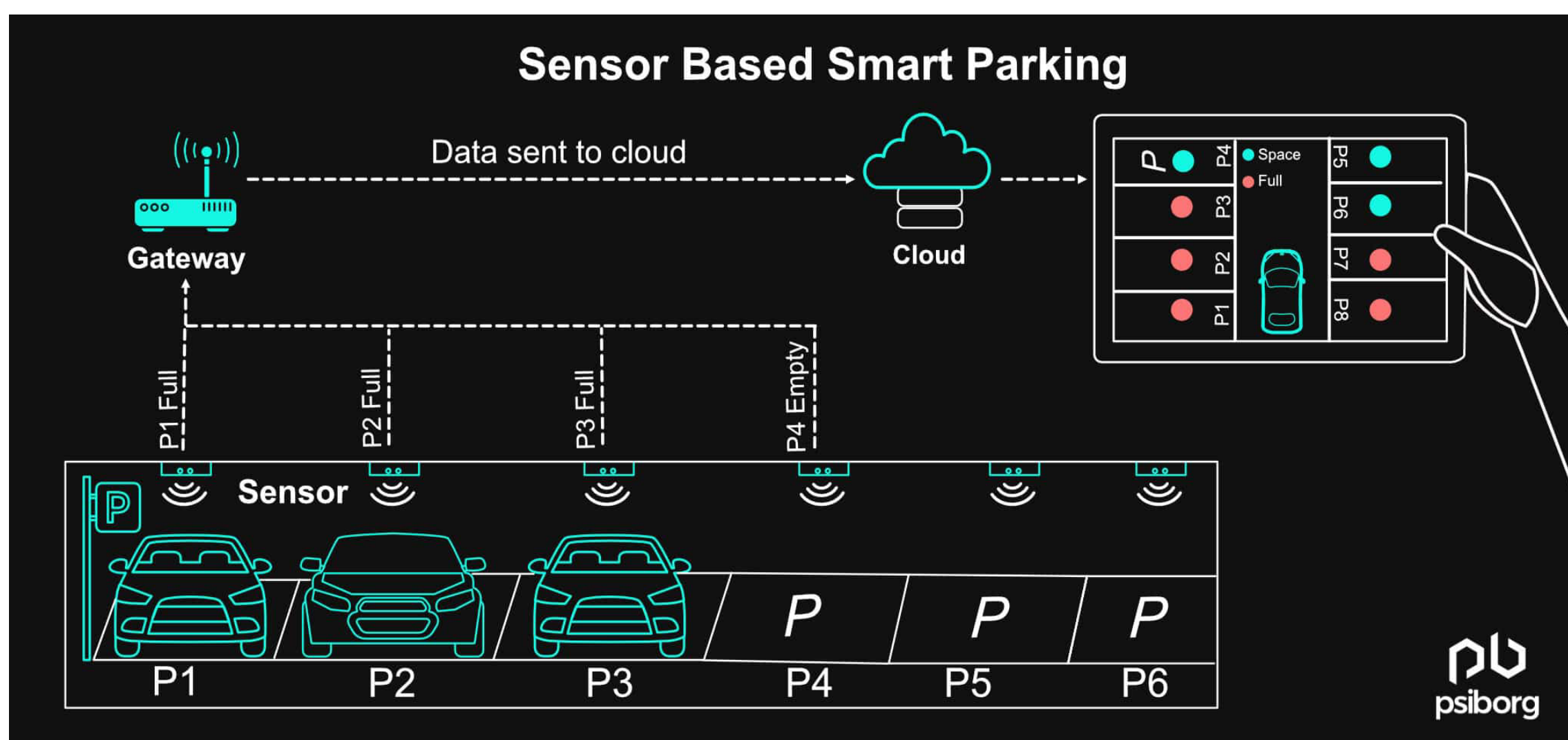
☒ INTRODUCTION:

- Integrating IoT sensors with a Raspberry Pi for smart parking is a forward-thinking solution that promises to revolutionize the way we manage urban parking spaces. In this project, we will explore the exciting world of Internet of Things (IoT) technology and Raspberry Pi to create a cutting-edge smart parking system.
- By harnessing the power of sensors, wireless communication, and a compact, versatile Raspberry Pi, we aim to develop a cost-effective and efficient solution to address the growing challenges of urban parking. This integration has the potential to optimize parking space utilization, reduce congestion, and enhance the overall urban experience..
- Harnessing the capabilities of IoT sensors to detect parking space availability is a transformative step towards achieving efficient and smart parking solutions. In

this endeavor, we will explore the concept of configuring IoT sensors to monitor and relay real-time data about parking space occupancy.

- we aim to revolutionize the way we approach urban parking, making it more convenient and stress-free for everyone. This innovation has the potential to significantly reduce congestion, save time, and enhance the overall urban living experience. Join us on this journey as we unravel the possibilities of configuring IoT sensors for a brighter and more efficient future of smart parking.

☒ BUILD THE IOT SENSOR SYSTEM:



► Building an IoT sensor system for smart parking involves several steps:

► Define Requirements:

- ✓ Clearly define the requirements for your smart parking system.
- ✓ Consider factors like the number of parking spots, sensor types (e.g., ultrasonic, magnetic, infrared), and data transmission methods (e.g., Wi-Fi, LoRa, cellular).

► Select Sensors:

- ✓ Choose appropriate sensors for detecting vehicle presence and status in parking spots. Ultrasonic or magnetic sensors are commonly used for this purpose.

► Microcontrollers:

- ✓ Select microcontrollers (e.g., Arduino, Raspberry Pi) to interface with the sensors and process data. Ensure they have the required connectivity options.

► **Connectivity:**

- ✓ Choose a communication protocol (e.g., MQTT, HTTP) and a network (Wi-Fi, LoRa, cellular) for transmitting data from sensors to a central server or cloud platform.

► **Power Supply:**

- ✓ Plan the power supply for your sensors. Battery-powered sensors may need periodic maintenance, while hardwired sensors can be more reliable but require installation.

► **Data Processing:**

- ✓ Implement data processing logic on the microcontrollers to handle sensor data. This may involve algorithms to detect parking spot occupancy.

► **Cloud Platform:**

- ✓ Choose a cloud platform (e.g., AWS, Azure, Google Cloud) to manage and analyze data. Integrate your IoT devices with the cloud platform using its IoT services.

► **User Interface:**

- ✓ Develop a user interface (web or mobile app) for users to check parking spot availability and make reservations if needed.

► **Security:**

- ✓ Implement security measures to protect data and devices. Use encryption, access controls, and secure authentication.

► **Testing:**

- ✓ Thoroughly test your system to ensure it accurately detects parking spot occupancy and reliably transmits data.

► **Data Analytics:**

- ✓ Use data analytics to gain insights from the collected data, such as parking spot utilization trends.

☒ BUILD THE RASPBERRY PI INTEGRATION:

- ▶ Building a Raspberry Pi-based smart parking system involves several steps, including hardware setup and software development. Here's a high-level overview of the process:

▶ **Hardware Components:**

- ✓ **Raspberry Pi:**

- Use a Raspberry Pi board (e.g., Raspberry Pi 4) as the central controller.

- ✓ **Ultrasonic Sensors:**

- To detect vehicle presence in parking spaces.

- ✓ **LEDs or Displays:**

- To indicate parking space availability.

- ✓ **Power Supply:**

- Ensure a stable power source for the Raspberry Pi and sensors.

▶ **Software Development:**

- ✓ **Raspberry Pi OS Installation:**

- Install a compatible Raspberry Pi OS (e.g., Raspbian) on the Raspberry Pi.

- ✓ **Python Programming:**

- Most Raspberry Pi projects are coded in Python. You'll need to develop Python scripts to control the sensors, camera, and communication.

▶ **OpenCV:**

- If you're using a camera, you may want to use OpenCV for image processing.

▶ **IoT Protocols:**

- **Set up** communication protocols (MQTT, HTTP, etc.) to transmit data to a central server or cloud platform.

► **Database:**

- Store data such as parking spot availability, timestamps, and images in a database on the Raspberry Pi or a remote server.

► **User Interface:**

- Create a simple user interface for displaying parking information, possibly using a web interface.

► **Integration:**

- Integrate your Raspberry Pi with a central system or dashboard where users can access parking availability information.

► **Sensor Integration:**

- Set up the ultrasonic sensors or other sensors to detect vehicle presence in parking spots.
- Write code to interpret sensor data and update the status of each parking spot (e.g., occupied or vacant).

► **Camera Integration:**

- Configure and program the camera module to capture images or video.
- Use OpenCV or other libraries to analyze images, detect license plates, and monitor the parking area.

► **Communication:**

- Establish a connection to the central server or cloud platform for real-time data transmission.
- Ensure that the Raspberry Pi can send data securely and reliably.

► **Data Storage:**

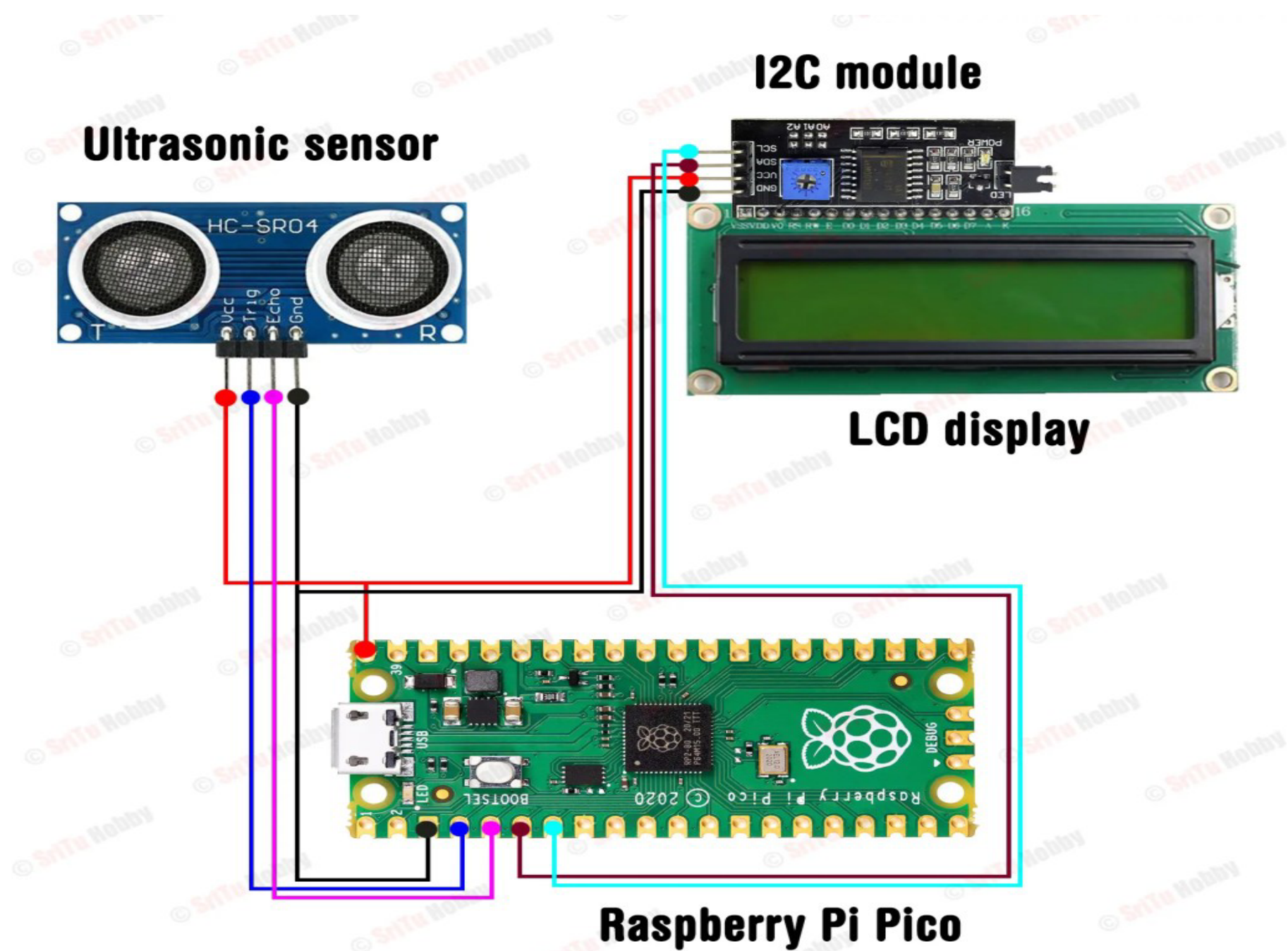
- Store data in a database, either on the Raspberry Pi or a remote server, for historical analysis and reporting.

► **Deployment:**

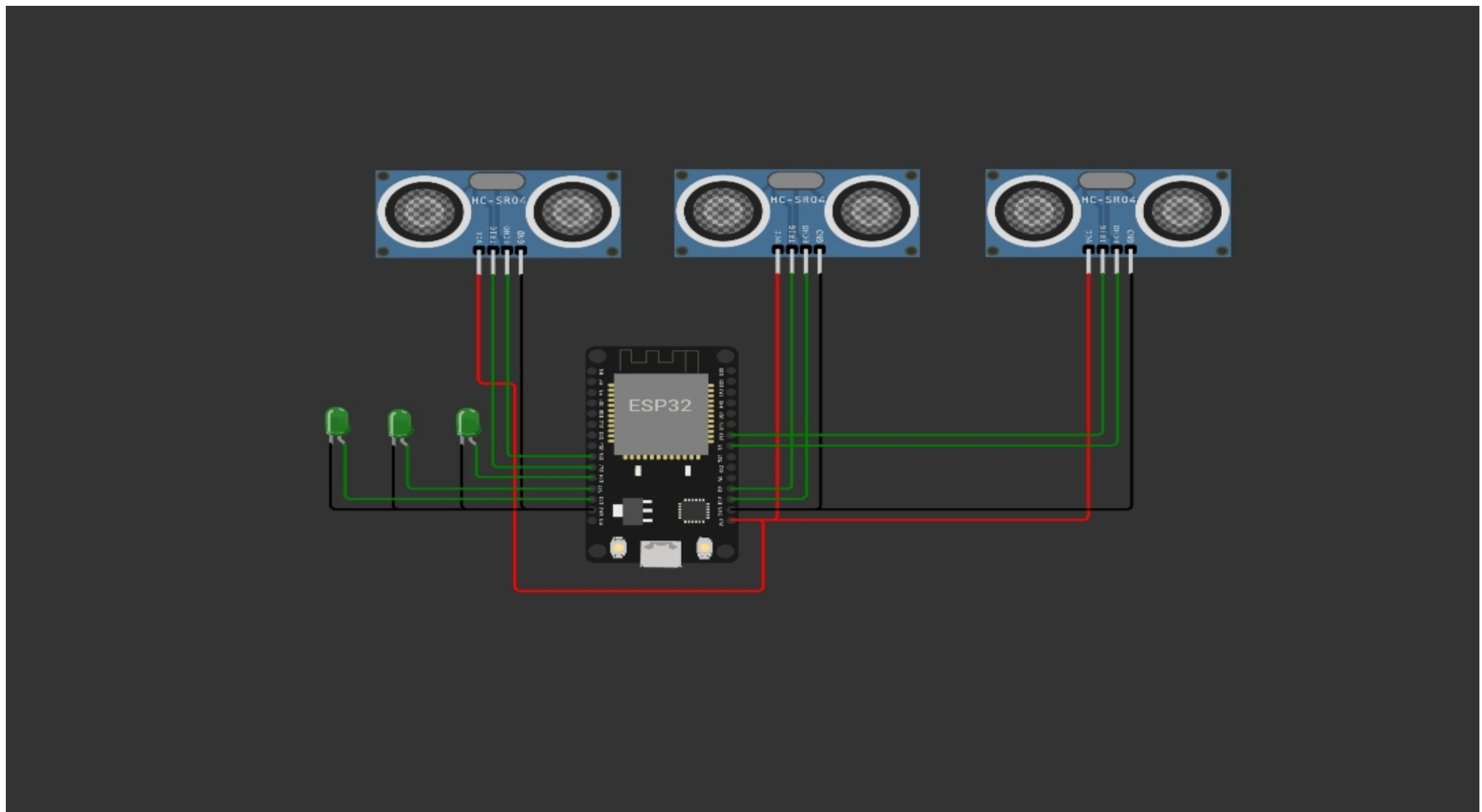
- Install the Raspberry Pi and associated hardware at the parking location.
- Monitor the system and make any necessary adjustments.

► **Scaling:**

- If your project is part of a larger smart parking system, consider how multiple Raspberry Pi devices can be integrated and scaled.



☒ CONFIGURE IOT SENSORS TO DETECT PARKING SPACE AVAILABILITY:



✓ Configuring IoT sensors to detect parking space occupancy for smart parking involves setting up the sensors, connecting them to a central controller (e.g., Raspberry Pi), and developing the necessary software. Here's a step-by-step guide:

► Hardware Components:

✓ **Ultrasonic Sensors:**

- Use ultrasonic sensors to detect the presence of vehicles in parking spaces.

✓ **Central Controller:**

- A Raspberry Pi or similar device to collect and process sensor data.

✓ **Power Supply:**

- Provide power to the sensors and controller.

✓ **Mounting Hardware:**

- Securely install sensors in parking spaces.

► **Software Components:**

✓ **Raspberry Pi OS:**

- Install a compatible operating system (e.g., Raspbian) on the central controller.

✓ **Python Programming:**

- Write Python scripts to interface with the sensors and process data.

► **Configuration Steps:**

✓ **Sensor Placement:**

- Install the ultrasonic sensors in each parking space. Mount them at a height to ensure they can detect the presence of vehicles.

✓ **Wiring:**

- Connect the sensors to the Raspberry Pi using GPIO pins. Refer to the sensor's datasheet and Raspberry Pi pinout for correct connections.

✓ **Raspberry Pi Setup:**

- Power up the Raspberry Pi and make sure it's connected to the internet via Wi-Fi or Ethernet.

✓ **Choose the Right Sensors:**

- Select appropriate sensors for detecting the presence or absence of vehicles. Common options include ultrasonic sensors, infrared sensors, or magnetic sensors.

✓ **Connect Sensors to a Gateway:**

- IoT sensors need to be connected to a central gateway or hub. This can be done using wired or wireless connections (e.g., Wi-Fi, LoRa, or NB-IoT).

✓ **Power Supply:**

- Ensure the sensors have a stable power source. This can be battery-powered, solar-powered, or wired, depending on the location and infrastructure.

✓ **Data Transmission:**

- Configure the sensors to send data to a central server or cloud platform. Ensure they are set up to transmit data at regular intervals or when a change in status is detected.

✓ **Data Processing:**

- Develop software to process the data received from the sensors. This may involve analyzing the sensor data to determine if a parking space is occupied or vacant.

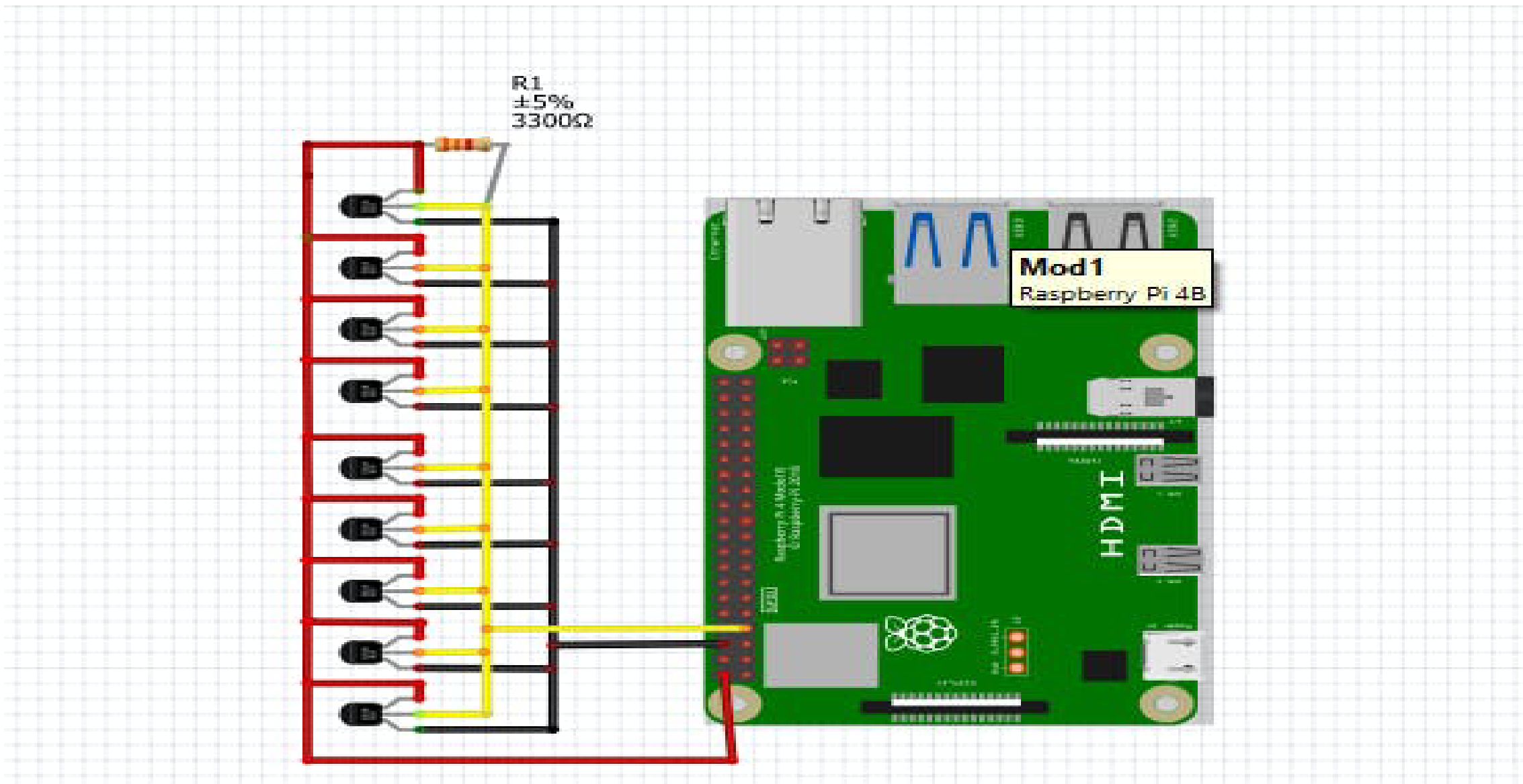
✓ **User Interface:**

- Create a user interface, such as a mobile app or a website, to display parking space availability information to users.

✓ **Notifications:**

- Implement a notification system to alert users when parking spaces become available or are occupied. This can be done through SMS, app notifications, or other means.

☒ **PYTHON SCRIPTS ON RASPBERRY Pi TO COLLECT DATA FROM SENSORS:**



- To collect data from sensors in a smart parking system using a Raspberry Pi, you'll need to write Python scripts that interact with the sensors and manage the data.

✓ **Data Collection:**

- Write Python functions to read data from the sensors. For example, to measure distance with an ultrasonic sensor, you can use the GPIO library to trigger the sensor and measure the time it takes for the echo signal to return.

✓ **Data Processing:**

- Process the data as needed. This may include filtering out noise, calibrating sensor values, and converting them to meaningful units (e.g., distance in meters).

✓ **Automation and Control:**

- You may want to implement logic for controlling parking spot availability or sending alerts when spots are full.

✓ **Scheduled Data Collection:**

- Use Python's scheduling libraries (e.g., cron jobs or APScheduler) to set up regular data collection intervals.

☒ CODING FOR SMART PARKING TO COLLECT DATA FROM SENSORS IN THE WOKWI PLATFORM:

- ▶ Wokwi is a platform for simulating and prototyping electronics projects. To create a smart parking system simulation using Wokwi, you can follow these general steps:

- ✓ **Components Selection:**

- Choose the components you'll need for your project, such as ultrasonic sensors, LEDs, and a microcontroller like Arduino.

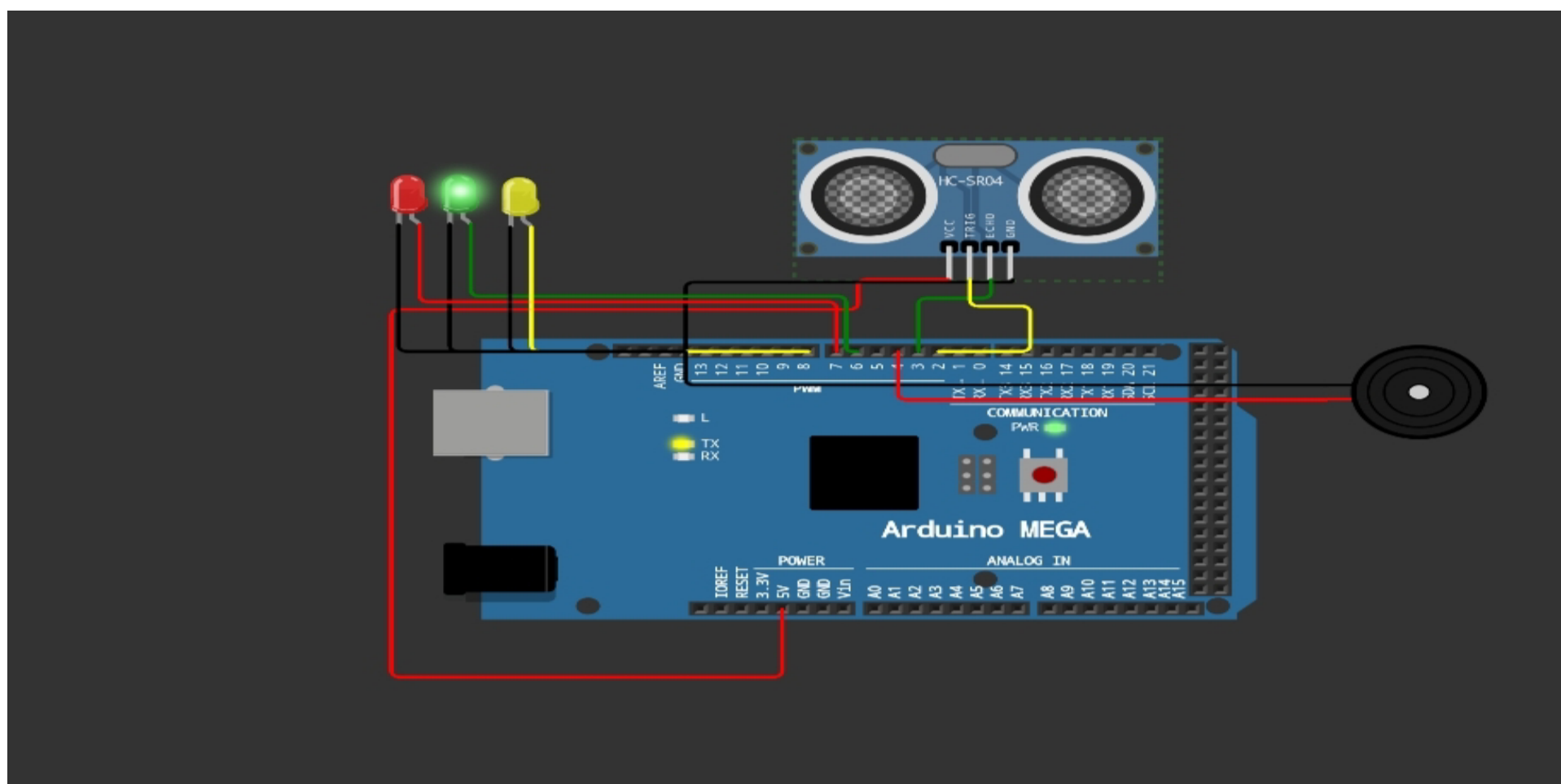
- ✓ **Wokwi Library:**

- Wokwi provides a library of components you can use in your simulation. Make sure to include the necessary components from the library.

- ✓ **Circuit Design:**

- Create a circuit design by connecting the components. For a smart parking system, you'd typically have ultrasonic sensors to detect cars and LEDs to indicate available parking spots.

- ▶ **CIRCUIT DIAGRAM:**



- ▶ **CODE:**

```
#define echoPin 3 // Echo Pin
#define trigPin 2 // Trigger Pin
#define LEDPin 13 // Onboard LED
int LED_EMPTY = 6;
int LED_FULL = 7;
int LED_PENDING = 8;
int BUZZER = 4;
void ULT(void);
int maximumRange = 200; // Maximum range needed
int minimumRange = 0; // Minimum range needed
```

```
long duration, distance; // Duration used to calculate distance
void setup() {
  Serial.begin(115200);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(LEDPin, OUTPUT); // Use LED indicator (if required)
  pinMode(LED_EMPTY, OUTPUT);
  pinMode(LED_FULL, OUTPUT);
  pinMode(LED_PENDING, OUTPUT);
  pinMode(BUZZER, OUTPUT);
}
void loop() {
  ULT();
  Serial.println(distance); // show distance
  /*vacant, out green light*/
  if(distance >= 200){
    digitalWrite(LED_EMPTY, 1);
    digitalWrite(LED_PENDING, 0);
    digitalWrite(LED_FULL, 0);
    Serial.println("Empty Space.");
  }
  /*someone is parking, out yellow light*/
  else if(distance < 200 && distance >= 50){
    digitalWrite(LED_EMPTY, 0);
    digitalWrite(LED_PENDING, 1);
    digitalWrite(LED_FULL, 0);
    tone(BUZZER, 800);
    delay(100);
    digitalWrite(LED_EMPTY, 0);
    digitalWrite(LED_PENDING, 0);
    digitalWrite(LED_FULL, 0);
    noTone(BUZZER);
    delay(500);
    Serial.println("Car is going to park here or going out.");
  }
  /*occupied, out red light*/
```

```
    else{
        digitalWrite(LED_EMPTY,0);
        digitalWrite(LED_PENDING,0);
        digitalWrite(LED_FULL,1);
        Serial.println("Car is Parking.");
    }
}

void ULT(){
    digitalWrite(trigPin,LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin,HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin,LOW);
    duration = pulseIn(echoPin,HIGH);
    //Calculate the distance (in cm) based on the speed of sound.
    distance = duration / 58.2;
}
```

► SIMULATION:

wokwi.com/projects/366586746176717825

Gmail YouTube Maps

WOKWI SAVE SHARE

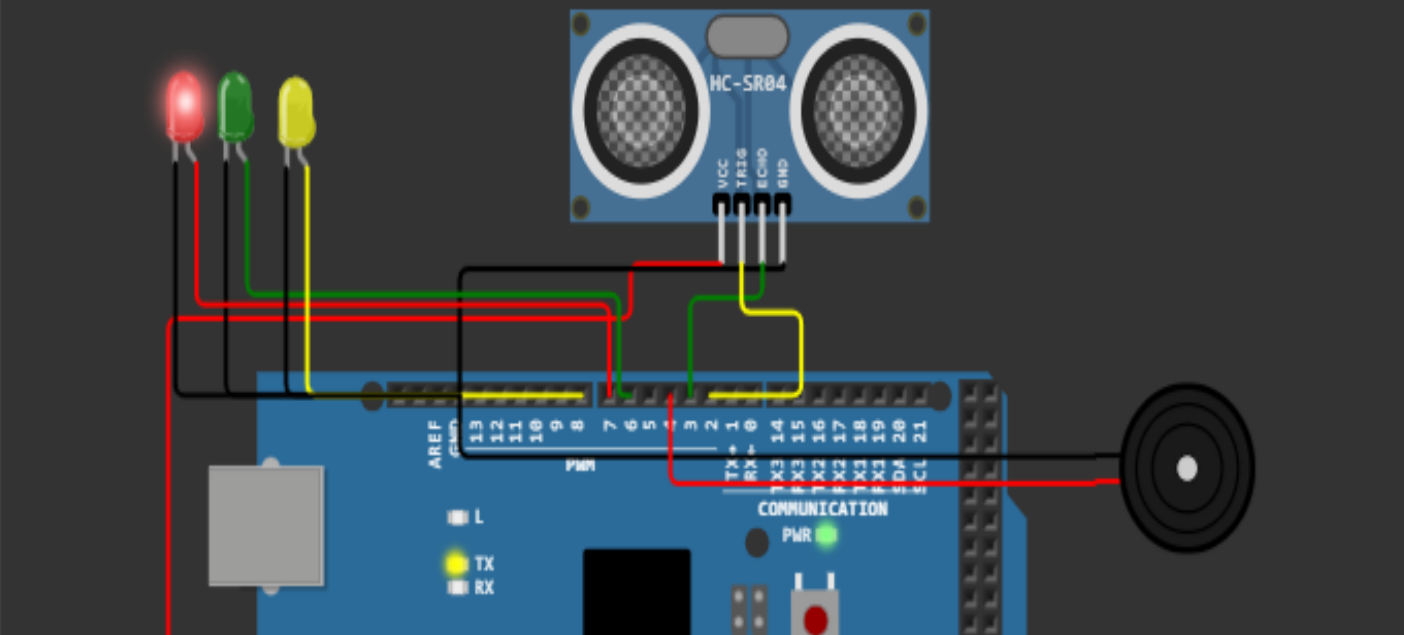
Docs SIGN UP

sketch.ino diagram.json Library Manager

```
21
22 int maximumRange = 200; // Maximum range needed
23 int minimumRange = 0; // Minimum range needed
24 long duration, distance; // Duration used to calculate distance
25
26 void setup() {
27   Serial.begin (115200);
28   pinMode(trigPin, OUTPUT);
29   pinMode(echoPin, INPUT);
30   pinMode(LEDpin, OUTPUT); // Use LED indicator (if required)
31   pinMode(LED_EMPTY, OUTPUT);
32   pinMode(LED_FULL, OUTPUT);
33   pinMode(LED_PENDING, OUTPUT);
34   pinMode(BUZZER, OUTPUT);
35 }
36
37 void loop() {
38   ULT();
39   Serial.println(distance); //show distance
40
41   /*vacant, out green light*/
42   if(distance >= 200){
43     digitalWrite(LED_EMPTY,1);
44     digitalWrite(LED_PENDING,0);
45     digitalWrite(LED_FULL,0);
46     Serial.println("Empty Space.");
47   }
48
49   /*someone is parking_ out yellow light*/
```

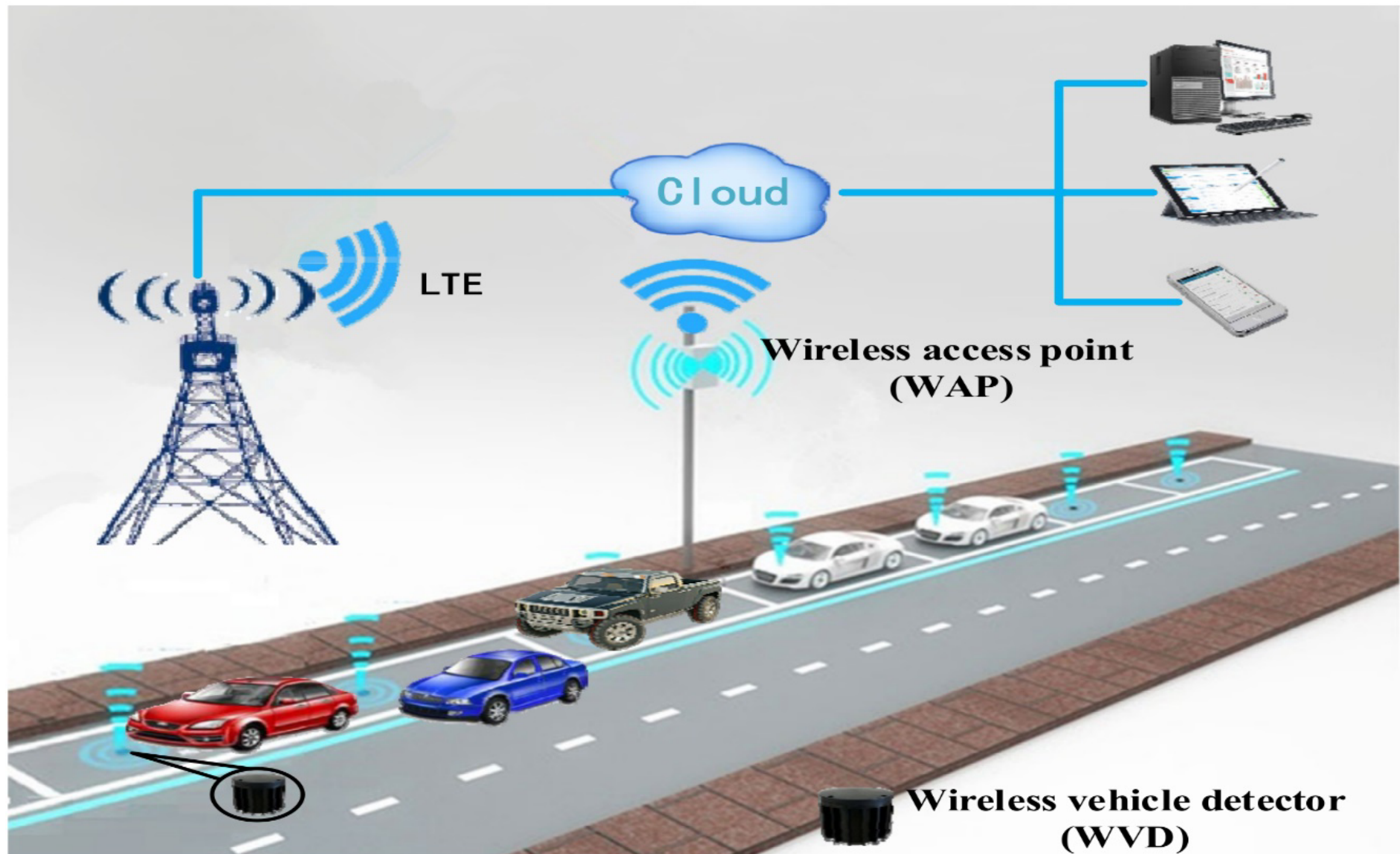
Simulation

00:03.466 28%



Car is Parking.
1
Car is Parking.
1
Car is Parking.
1
Car is Parking.

☒ **SEND DATA TO THE CLOUD OR MOBILE APP SERVER:**



- To send data from sensors to the cloud or a mobile app server in a smart parking system, you can follow these steps:

✓ **Sensor Data Collection:**

- You'll have sensors installed in parking spaces to detect occupancy. These sensors can be ultrasonic, infrared, magnetic, or any other suitable type. They should be configured to collect data about the parking space's status (occupied or vacant).

✓ **Microcontrollers/Edge Devices:**

- You'll need microcontrollers or edge devices (e.g., Arduino, Raspberry Pi) to interface with the sensors and gather data from them. These devices should be connected to the sensors and programmed to read the sensor data.

✓ **Communication Protocols:**

- Choose a communication protocol to transmit the data from the microcontrollers to the cloud or mobile app server. Common protocols include Wi-Fi, cellular (3G/4G/5G), LoRa, or Bluetooth. The choice depends on the range and connectivity options in your parking area.

✓ **Data Formatting:**

- The data from the sensors should be formatted into a structured message, often in JSON or XML format, with relevant information such as parking space ID, status, and timestamp.

✓ **Cloud Service or Mobile App Server:**

- You'll need a cloud service (e.g., AWS, Azure, Google Cloud) or a mobile app server to receive and store the data. Set up an API on the server to handle incoming sensor data.

✓ **Data Processing:**

- The cloud or server can process the incoming data to update the parking status and make it accessible to the mobile app.

✓ **Mobile App Integration:**

- Develop a mobile app for users to access real-time parking information. The app can communicate with the cloud/server API to fetch parking availability data and display it to users.

✓ **User Interface:**

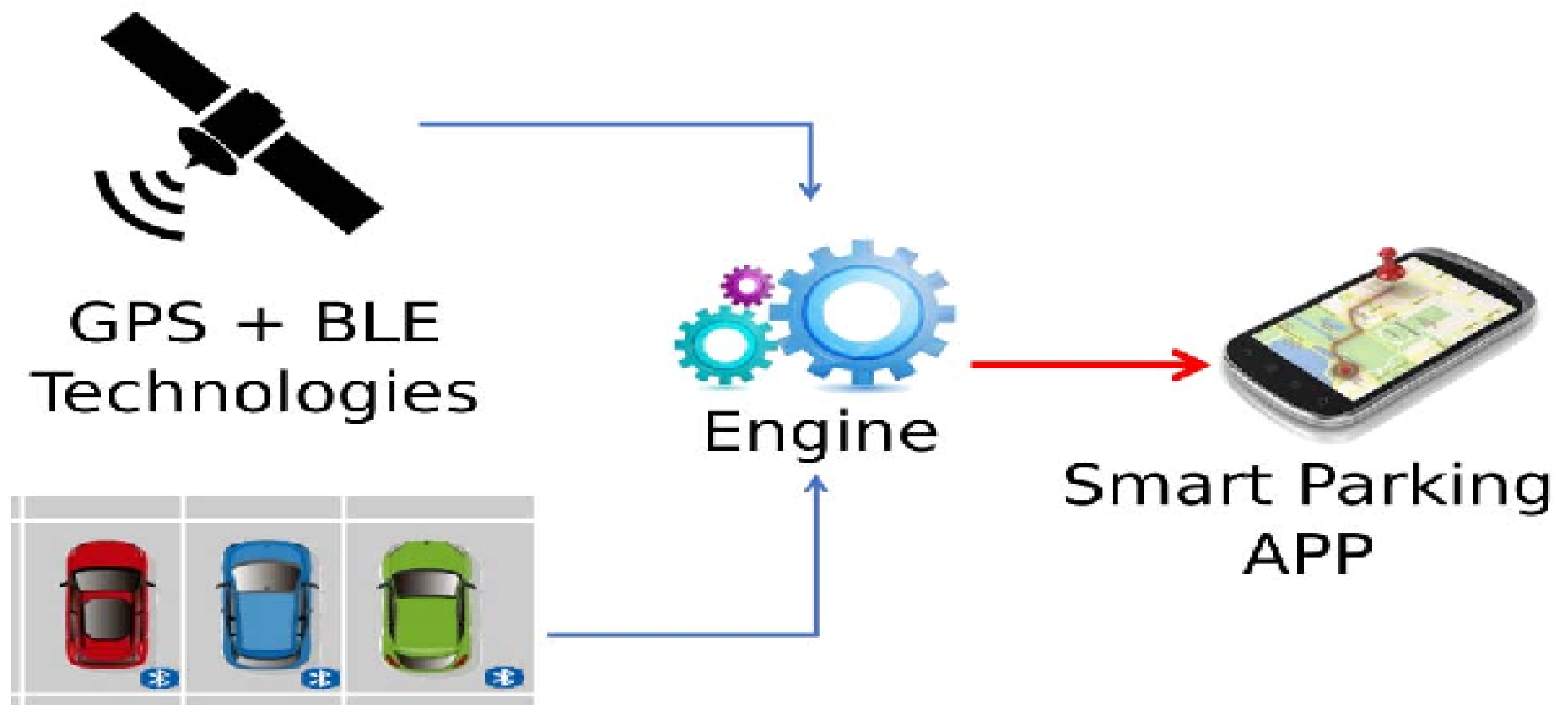
- Create an intuitive user interface in the mobile app to show parking space availability, navigation to vacant spaces, and possibly payment integration.

✓ **Real-time Updates:**

- Ensure that the sensor data is continuously updated in real-time, so users get accurate parking information.

✓ **Maintenance and Monitoring:**

- Regularly monitor and maintain the sensors, microcontrollers, and the server infrastructure to ensure the system's reliability.



❏ CONCLUSION:

- o In conclusion, our journey into building the integration of IoT sensors and Raspberry Pi for smart parking has unveiled a world of innovation and potential. Through the synergy of sensor technology, wireless connectivity, and the computational prowess of the Raspberry Pi, we have created a scalable and intelligent system that can revolutionize urban parking.
- o This project showcases the power of technology in addressing real-world challenges. With the ability to detect parking space availability in real-time, our solution not only enhances convenience for drivers but also has the potential to reduce traffic congestion, lower carbon emissions, and improve the overall urban environment.
- o As we move forward, the possibilities for this integration are limitless. Future developments may include machine learning algorithms for predictive parking, seamless integration with mobile apps, and integration with citywide traffic management systems.
- o In a world where urbanization is on the rise, smart parking solutions like this one will play a pivotal role in creating more efficient, sustainable, and livable cities. Our journey is a testament to the transformative power of technology and its ability to shape a smarter, more connected future for us all.

