Smart parking

Phase 3:

Certainly, building an IOT sensor system with Raspberry Pi integration to detect parking space occupancy involves several steps. Here's an outline of the process:

IOT Sensor Parking System

In this part you will begin building your project.

Start building the IOT sensor system and Raspberry Pi integration.

Configure IOT sensors (e.g., ultrasonic sensors) to detect parking space occupancy.

Certainly, building an IOT sensor system with Raspberry Pi integration to detect parking space occupancy involves several steps. Here's an outline of the process:

Select IOT Sensors:

Choose appropriate sensors for parking space occupancy detection. Ultrasonic sensors are a good choice for this purpose, as they can measure distances accurately.

Hardware Setup:

Connect the ultrasonic sensors to your Raspberry Pi following the manufacturer's instructions. Ensure your Raspberry Pi is set up with the required peripherals and a stable internet connection.

Install Necessary Libraries:

Install any Python libraries required for sensor communication. For ultrasonic sensors, you might use libraries like RPi.GPIO or specific ultrasonic sensor libraries.

Python Script Development:

Write Python scripts to interact with the sensors.

Read sensor data and calculate distance, which can be used to determine parking space occupancy.

Implement logic to decide whether a parking space is occupied or vacant based on the sensor readings.

Data Transmission:

If you want to send data to the cloud, consider using platforms like AWS IOT, Azure IOT, or Google Cloud IOT.

Implement code to send sensor data to the selected cloud platform using MQTT or HTTP protocols.

Mobile App Integration :

If you want to display parking space status on a mobile app, develop a mobile app that can communicate with the cloud platform. Consider using tools like AWS Amplify or Firebase for mobile app development.

Data Visualization:

Set up data visualization tools or dashboards on the cloud platform to display real-time parking space occupancy status.

Testing and Deployment:

Test your system thoroughly to ensure the sensors are accurately detecting parking space occupancy.

Deploy the Raspberry Pi and sensors at the desired parking location.

Remember to handle security considerations, data storage, and access control if you're transmitting data to the cloud. Additionally, make sure your Raspberry Pi and sensors are powered and connected reliably to maintain continuous operation.

Program:

```
import RPi.GPIO as GPIO
import time

# Set the GPIO pins for the ultrasonic sensor
TRIG = 23
ECHO = 24

# Initialize GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(TRIG, GPIO.OUT)
```

```
GPIO.setup(ECHO, GPIO.IN)

try:
    while True:
        # Trigger the ultrasonic sensor
        GPIO.output(TRIG, True)
        time.sleep(0.00001)
        GPIO.output(TRIG, False)

        pulse_start = time.time()
        pulse_end = time.time()

        # Wait for the ECHO pin to go high
        while GPIO.input(ECHO) == 0:
            pulse_start = time.time()

        # Wait for the ECHO pin to go low
        while GPIO.input(ECHO) == 1:
            pulse_end = time.time()

        # Calculate the distance
        pulse_duration = pulse_end - pulse_start
        distance = (pulse_duration * 34300) / 2

        print(f"Distance: {distance:.2f} cm")
```

```
        # You can add your occupancy logic here based on distance

        time.sleep(1)  # Sleep for 1 second before the next measurement

except KeyboardInterrupt:
    GPIO.cleanup()
```

Program 2:

```
import RPi.GPIO as GPIO
import time
import AWSIoTPythonSDK.MQTTLib as AWSIoTMQTTClient

# Set up GPIO pins for ultrasonic sensors
TRIG = 23
ECHO = 24

# Set up AWS IoT client
# You'll need to configure your AWS IoT credentials and endpoint
myMQTTClient = AWSIoTMQTTClient("smart-parking")
myMQTTClient.configureEndpoint("your-iot-endpoint.amazonaws.com", 8883)
myMQTTClient.configureCredentials("root-CA.pem", "private-key.pem",
"certificate.pem")
```

```
# Connect to AWS IoT
myMQTTClient.connect()

def get_distance():
    GPIO.output(TRIG, True)
    time.sleep(0.00001)
    GPIO.output(TRIG, False)

    pulse_start = time.time()
    pulse_end = time.time()

    while GPIO.input(ECHO) == 0:
        pulse_start = time.time()

    while GPIO.input(ECHO) == 1:
        pulse_end = time.time()

    pulse_duration = pulse_end - pulse_start
    distance = (pulse_duration * 34300) / 2

    return distance

try:
    while True:
        distance = get_distance()
```

```
        print(f"Distance: {distance:.2f} cm")

        # Implement your logic to determine parking space occupancy
        if distance < 10:  # Adjust this threshold based on your setup
            status = "Occupied"
        else:
            status = "Vacant"

        # Send the status to the cloud (AWS IoT in this example)
        myMQTTClient.publish("smart-parking/status", json.dumps({"status": status}), 1)

        time.sleep(1)  # Sleep before the next measurement

except KeyboardInterrupt:
    GPIO.cleanup()
    myMQTTClient.disconnect()
```