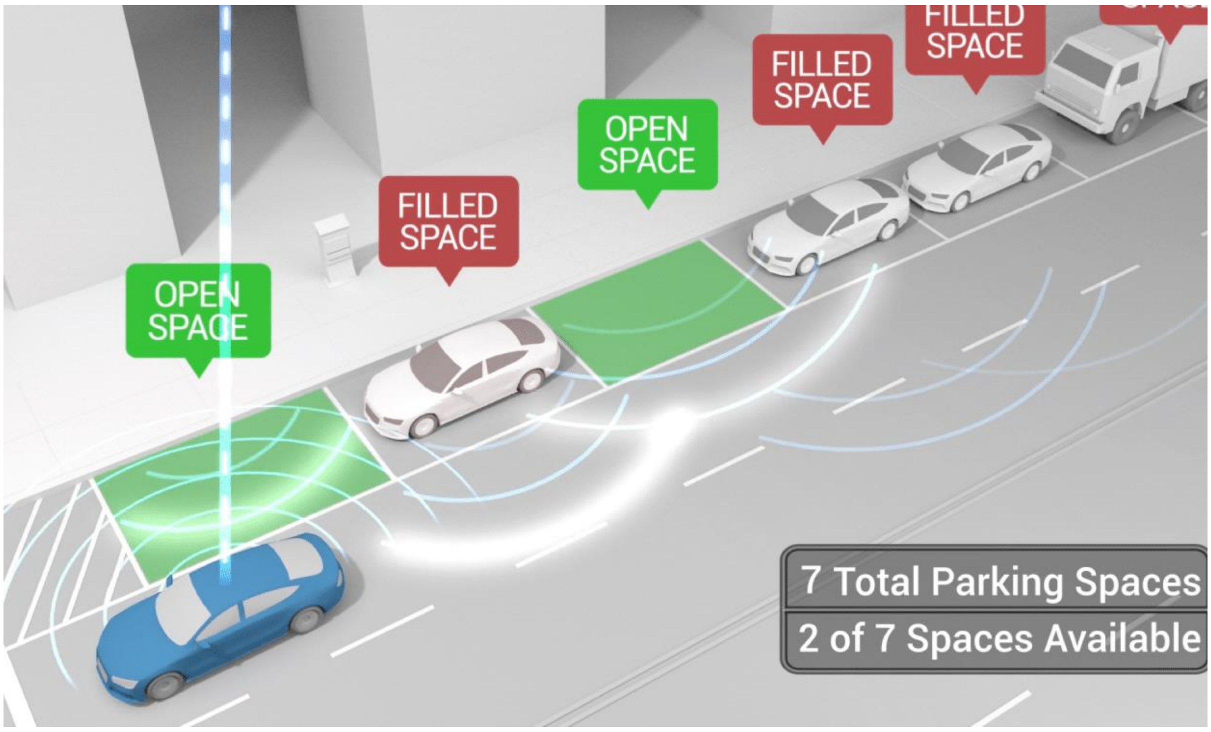
Smart Parking System using IoT



Introduction:

This innovative system leverage the power of the Internet of Things (IoT) and the versatility of Raspberry Pi to revolutionize the way we find and monitor parking space occupancy. By seamlessly integrating IoT sensors with Raspberry Pi, these systems offer a transformative solution that enhances parking management, optimizes space utilization, and ultimately improves the urban living experience.

Hardware Components:

- *Raspberry Pi:* You can use any model with Wi-Fi capability, such as Raspberry Pi 3 or 4.
- *Ultrasonic Sensors:* These sensors are used to detect the presence of vehicles in parking spaces. Common models include HC-SR04 or similar ultrasonic distance sensors.
- *Breadboard and Jumper Wires:* For connecting the sensors to the Raspberry Pi GPIO pins.
- *Power Supply:* A power source for the Raspberry Pi to keep it running.
- *Enclosure (Optional):* An enclosure for the Raspberry Pi and sensors to protect them from environmental factors.

Software and Services:

- *Raspberry Pi OS:* The latest Raspberry Pi OS on your Raspberry Pi and ensure it's connected to the internet.

- *Python Libraries:* The necessary Python libraries on your Raspberry Pi:
    - ♦ *RPi.GPIO:* For GPIO control on the Raspberry Pi.
    - ♦ requests: For making HTTP requests to send data to the cloud or server.
- *Cloud Server or Mobile App Server:* You will need a server to store and manage the data collected from the Raspberry Pi. This server should have an API endpoint to receive and store the data. You can use cloud platforms like AWS, Google Cloud, or services like Firebase for mobile apps.

Making Steps:

- ***Setting Up Raspberry Pi:***
    - Install the Raspberry Pi OS, connect it to the internet, and enable the GPIO pins.
- ***Connect Ultrasonic Sensors:***
    - Wire the ultrasonic sensors to the Raspberry Pi using jumper wires. Connect the trigger (TRIG) and echo (ECHO) pins of the sensors to GPIO pins on the Raspberry Pi.
    - Provide power (VCC) and ground (GND) connections to the sensors.
- ***Python Code:***

```
import RPi.GPIO as GPIO
import time

# Define GPIO pins
TRIG_PIN = 23
ECHO_PIN = 24

# Set up GPIO mode
GPIO.setmode(GPIO.BCM)
GPIO.setup(TRIG_PIN, GPIO.OUT)
GPIO.setup(ECHO_PIN, GPIO.IN)

def measure_distance():
    # Send a trigger pulse
    GPIO.output(TRIG_PIN, True)
    time.sleep(0.00001)
    GPIO.output(TRIG_PIN, False)

    # Measure the time taken for the echo
    while GPIO.input(ECHO_PIN) == 0:
        pulse_start = time.time()
    while GPIO.input(ECHO_PIN) == 1:
        pulse_end = time.time()

    # Calculate distance in centimeters
    pulse_duration = pulse_end - pulse_start
    distance = (pulse_duration * 34300) / 2  # Speed of sound = 343 m/s
    return round(distance, 2)

try:
    while True:
        dist = measure_distance()
        if dist < 10:  # Adjust this threshold based on your parking space configuration
            print("Parking space occupied")
        else:
            print("Parking space available")
        time.sleep(1)  # Adjust the delay as needed

except KeyboardInterrupt:
    GPIO.cleanup()
```

  - Use the script template from the earlier response to collect distance measurements using the ultrasonic sensors.
- *Test Sensors:*
  - Running the Python script on the Raspberry Pi to ensure that the sensors are correctly wired and provide accurate distance measurements.
- ***Data Processing and Transmission:***
  - Modify the Python script to process the data and send it to the cloud or mobile app server. You'll need to replace the "YOUR_API_ENDPOINT" with the actual endpoint for your server.
- ***Modified Python Code:***

```
import RPi.GPIO as GPIO
import time
import requests

# Define GPIO pins
TRIG_PIN = 23
ECHO_PIN = 24

# Set up GPIO mode
GPIO.setmode(GPIO.BCM)
GPIO.setup(TRIG_PIN, GPIO.OUT)
GPIO.setup(ECHO_PIN, GPIO.IN)

# Cloud server API endpoint
API_ENDPOINT = "YOUR_API_ENDPOINT"  # Replace with the actual endpoint

def measure_distance():
    # Send a trigger pulse
    GPIO.output(TRIG_PIN, True)
    time.sleep(0.00001)
    GPIO.output(TRIG_PIN, False)

    # Measure the time taken for the echo
    while GPIO.input(ECHO_PIN) == 0:
        pulse_start = time.time()
    while GPIO.input(ECHO_PIN) == 1:
        pulse_end = time.time()

    # Calculate distance in centimeters
    pulse_duration = pulse_end - pulse_start
    distance = (pulse_duration * 34300) / 2  # Speed of sound = 343 m/s
    return round(distance, 2)

try:
    while True:
        dist = measure_distance()
        print(f"Distance: {dist} cm")

        # Send data to the cloud server
        data = {"distance": dist, "timestamp": int(time.time())}

        response = requests.post(API_ENDPOINT, json=data)

        if response.status_code == 200:
            print("Data sent to the cloud server successfully.")
        else:
            print("Failed to send data to the cloud server.")

        time.sleep(1)  # Adjust the delay as needed

except KeyboardInterrupt:
    GPIO.cleanup()
```

- ***Power Supply for Sensors:***
    - Depending on the sensor's requirements, they are powered from the Raspberry Pi's GPIO pins or use an external power supply.
- ***Monitoring and Maintenance:***
    - Regularly monitor the system for sensor accuracy and functionality. Implement remote monitoring and alerting systems to be aware of any issues that may arise.
- ***Data Security and Privacy:***
    - Ensure that data collected from sensors is secured and follows privacy guidelines. Implement encryption and access controls.

Conclusion:

In conclusion, the Smart Parking System leverages cutting-edge technology, combining Raspberry Pi, ultrasonic sensors, and a cloud or mobile app server, to tackle the challenges of parking space management in urban environments. This system provides real-time data on parking space occupancy, enhancing user convenience, reducing congestion, and contributing to more efficient urban living. The documented system setup, hardware connections, and Python code serve as essential references for system maintenance and troubleshooting.