SMART PARKING SYSTEM USING IOT

NAME: MATHANKUMAR H

EMAIL: wwwmathanrasig55@gmail.com

AU:721221106302

Creating a smart parking system using a Raspberry Pi and a control application can be a fun and useful project. This system can help users find available parking spots and reserve them. Here's a high-level overview of how you can build such a system:

Hardware Components:

Raspberry Pi: You'll need a Raspberry Pi to act as the central controller for the smart parking system.

IR Sensors: These sensors will be used to detect the presence of a vehicle in each parking spot.

Camera (optional): You can use a camera to capture images of the parking lot and provide visual information to users.

LEDs or Displays: To indicate the status of each parking spot (e.g., available, occupied, reserved).

Internet Connectivity: The Raspberry Pi needs to be connected to the internet to communicate with the control application.

Software Components:

Raspberry Pi Software: Develop software for the Raspberry Pi to control the sensors, process data, and communicate with the control application. You can use Python for this purpose.

Control Application: Create a mobile or web application that allows users to check the parking lot's status, reserve a parking spot, and receive notifications.

Here's a step-by-step guide to create a simple smart parking system:

Step 1: Set Up the Hardware

Connect ultrasonic sensors to the Raspberry Pi. You'll need one sensor per parking spot.

Connect LEDs or displays to the Raspberry Pi to indicate the status of each spot.

If you're using a camera, set it up and ensure it's properly connected to the Raspberry Pi.

Step 2: Install and Configure the Raspberry Pi

Install the Raspbian operating system on your Raspberry Pi.

Set up Wi-Fi or Ethernet connectivity.

Install Python and any necessary libraries for sensor and camera control.

Step 3: Develop Raspberry Pi Software

Write Python scripts to read data from the ultrasonic sensors to detect vehicle presence.

Implement logic to update the parking spot status (available, occupied, reserved) based on sensor data.

If you're using a camera, capture and process images to provide visual information.

Create a web server on the Raspberry Pi to host a simple API for communication with the control application.

Step 4: Create the Control Application

Develop a mobile or web application for users to access the parking system.

Integrate with the Raspberry Pi's API to check parking spot availability and make reservations.

Add user authentication and registration features.

Implement a notification system to inform users about their reservations and parking spot availability.

Step 5: Test and Deploy

Test the system to ensure sensors are working accurately, and the control application functions as expected.

Deploy the system in a real parking lot.

Step 6: Continuous Monitoring and Maintenance

Regularly monitor and maintain the system to ensure it operates smoothly.

Perform updates and improvements as needed based on user feedback and changing requirements.

This is a simplified overview of building a smart parking system using a Raspberry Pi and a control application. You can expand and customize the system according to your requirements and budget, potentially adding features like payment processing, security, and advanced reservation management.

PROGRAM:

```
import RPi.GPIO as GPIO
import time
from flask import Flask, jsonify

app = Flask(__name__)

# GPIO pin setup
TRIG = 23
ECHO = 24
GPIO.setmode(GPIO.BCM)
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)

# Parking spot status (initialized as available)
parking_spots = {
   'A1': 'available',
   'A2': 'available',
   'A3': 'available',
}

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

   # Measure the time taken for the echo to return
   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  # Speed of sound = 343 m/s
    return distance

@app.route('/get_spot_status/<spot_id>')
def get_spot_status(spot_id):
    status = parking_spots.get(spot_id, 'invalid')
    return jsonify({'spot_id': spot_id, 'status': status})

def update_parking_status():
    while True:
        for spot_id in parking_spots:
            distance = measure_distance()
            if distance < 10:  # Adjust this threshold for your parking lot
                parking_spots[spot_id] = 'occupied'
            else:
                parking_spots[spot_id] = 'available'
        time.sleep(5)  # Check the status periodically

if __name__ == '__main__':
    try:
        update_parking_status()
    except KeyboardInterrupt:
        GPIO.cleanup()
```

FINAL CIRCUIT: