SMART PARKING PROJECT

-TEAM NAME:Proj_224689_Team_2

TEAM MEMBERS:

1.R. SATHANA

2.N.C. SRI VAIBAVA LAKSHMI

3.S. HIMRITHA

4.P. MAHALAKSHMI

5.N. KAVITHA

PROJECT REPORT: SMART PARKING SYSTEM WITH IOT SENSORS (DEVELOPMENT PART 1 & 2)

ABSTRACT

The "Smart Parking System with IoT Sensors" project is dedicated to developing an intelligent and efficient parking management solution using IoT sensors and the Raspberry Pi platform. This report documents the progress achieved in Development Part 1, emphasizing sensor selection, Raspberry Pi integration, Python script development for data collection, and secure data transmission to a cloud or mobile app server.

PROBLEM STATEMENT

In urban environments, parking management is a significant challenge. Drivers often struggle to find available parking spaces, leading to congestion and wasted time. Furthermore, parking facilities face difficulties in monitoring space occupancy efficiently. Our project addresses these issues by creating a system that provides real-time parking space availability information to both drivers and facility operators.

INTRODUCTION

Development Part 1 of the "Smart Parking System with IoT Sensors" project focuses on laying the foundation for a sophisticated parking management solution. This report outlines the milestones achieved in this initial phase.

DEVELOPMENT PART 1:

TARGET AND METHODOLOGY

Target:

1. Develop a reliable system for detecting parking space occupancy.
2. Create a user-friendly mobile app for real-time parking space availability updates.
3. Implement data transmission to a designated cloud or mobile app server for real-time monitoring.
4. Ensure data security and privacy during transmission.

Methodology:

Milestone 1 : IoT Sensor Selection and Setup

- **Sensor Selection**: HC-SR04 ultrasonic sensors were chosen for their accuracy and cost-effectiveness.
- **Sensor Configuration**: Ultrasonic sensors were connected to a Raspberry Pi, configured to measure distances and detect objects, particularly vehicles in parking spaces.

Milestone 2: Raspberry Pi Integration

- **Role of Raspberry Pi**: The Raspberry Pi serves as the central control unit, receiving data from the sensors.
- **GPIO Interface**: The GPIO pins of the Raspberry Pi were configured to trigger measurements and collect data.

Milestone 3: Python Scripts for Data Collection and Data Transmission

- **Data Collection and Sensor Interface**: Python scripts were developed to collect data from the ultrasonic sensors accurately.
- **Data Transmission**: Data collected from the sensors was transmitted to a designated cloud platform or mobile app server for real-time monitoring.

SOLUTION AND DISCUSSION

In Development Part 1, we successfully addressed the initial project milestones:

1. **Sensor Selection**: HC-SR04 ultrasonic sensors were chosen for their precision and cost-effectiveness. These sensors have a wide detection range and are well-suited for accurate distance measurements in parking spaces. The choice of sensors ensures that the system can reliably detect vehicle presence.
2. **Raspberry Pi Integration**: The Raspberry Pi plays a central role in our solution. It serves as the core controller, responsible for orchestrating data collection from the ultrasonic sensors and facilitating communication with the designated cloud platform or mobile app server. The Raspberry Pi's versatility and GPIO capabilities make it an ideal choice for this task.
3. **Data Collection and Sensor Interface**: The development of Python scripts to handle data collection from the ultrasonic sensors is a critical aspect of our solution. These scripts create a seamless interface between the Raspberry Pi and the sensors,

allowing for efficient and precise data capture. They are designed to:

- Trigger the sensors to initiate distance measurements by sending ultrasonic signals.
- Calculate precise distance measurements based on the time it takes for ultrasonic waves to bounce back from objects, particularly vehicles in parking spaces.
- Capture data from the sensors at regular intervals and store it for further processing. This ensures a continuous flow of information about parking space occupancy.

The use of Python as the programming language simplifies script development and provides the flexibility to make real-time adjustments and enhancements.

4. **Data Transmission**: The Python scripts were extended to include data transmission capabilities. This addition allows the system to share parking space occupancy data with a designated cloud platform or mobile app server in real-time. The implementation of data transmission ensures that users have access to up-to-the-minute information about available parking spaces.
    - **Security and Efficiency**: Security and efficiency were paramount in the data transmission process. Robust encryption measures and data integrity checks were implemented to safeguard user data during transmission. Additionally, the transmission process was optimized for efficiency, ensuring that data is transferred seamlessly to the designated server.

- **Continuous Monitoring**: The main loop within the Python scripts enables continuous data collection and transmission. This loop ensures that parking space occupancy data remains current and relevant. Users can trust the system to provide real-time updates, contributing to an improved parking experience.

FUTURE DEVELOPMENT (PART 2)

Development Part 1 has laid a solid foundation for the project, and we are eager to move forward with Part 2, which will focus on:

1. **User-Friendly Mobile App**: We plan to create a user-friendly mobile app that enables drivers to easily access real-time parking space availability information. The app will feature an intuitive interface and provide location-based services, making it a valuable tool for urban commuters.
2. **Advanced Detection Algorithms**: To further enhance the system's capabilities, we will implement advanced machine learning algorithms. These algorithms will improve detection accuracy, enabling the system to differentiate between various types of vehicles and even detect anomalies or obstructions in parking spaces.
3. **Security and Scalability**: As the system gains more users, we will concentrate on enhancing its security and scalability. Measures will be taken to fortify data encryption and access controls. Scalability improvements will ensure the system's performance and responsiveness as it accommodates a growing user base.

DATA COLLECTION AND SENSOR INTERFACE:

(Also attached the code file in github account)

******Data Collection and Sensor Interface******

```
import RPi.GPIO as GPIO
import time

# Set GPIO mode to BCM
GPIO.setmode(GPIO.BCM)

# Define GPIO pins for the sensor
TRIGGER_PIN = 18
ECHO_PIN = 24

# Set up GPIO pins
GPIO.setup(TRIGGER_PIN, GPIO.OUT)
GPIO.setup(ECHO_PIN, GPIO.IN)

# Function to trigger a distance measurement
def trigger_measurement():
    GPIO.output(TRIGGER_PIN, True)
```

```
    time.sleep(0.00001)
    GPIO.output(TRIGGER_PIN, False)

# Function to measure the distance
def measure_distance():
    pulse_start = time.time()
    pulse_end = time.time()

    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
```

******Data Transmission to the Cloud/Mobile App Server******

```
import requests
# Define the server endpoint for data transmission
SERVER_ENDPOINT = "http://example.com/api/parking"
# Main loop for data collection and transmission
try:
    while True:
        # Trigger a distance measurement
        trigger_measurement()

        # Measure the distance
        distance = measure_distance()

        # Send data to the server
        data = {"distance": distance}
        response = requests.post(SERVER_ENDPOINT, json=data)

        if response.status_code == 200:
            print(f"Data sent: Distance - {distance} cm")
        else:
```

```
            print("Data transmission failed")

        # Adjust the data collection and transmission interval
        time.sleep(5)
except KeyboardInterrupt:
    print("Data collection stopped by the user")
finally:
    # Clean up GPIO settings
    GPIO.cleanup()
```

CONCLUSION:

Development Part 1 has established the foundation for a comprehensive and efficient parking management system. With the selection of reliable sensors, integration with the Raspberry Pi, the development of Python scripts for data collection and transmission, and the emphasis on data security, we are poised to provide a solution that addresses the challenges of urban parking.

Our commitment to real-time data, user-friendly interfaces, and advanced technologies positions the "Smart Parking System with IoT Sensors" project as a valuable addition to smart city infrastructure. It not only benefits drivers by simplifying parking but also contributes to the optimization of parking space utilization, reducing congestion and improving urban living.