SMART PARKING SYSTEM

Name	**: A. Murugu manoj**
Reg. No	**: 410121104027**
Naan Mudhalvan id	**: au410121104027**
Domain	**: Internet of Things**
Project Name	**: Smart Parking**
College Code	**: 4101**
College Name	**: Adhi College of Engineering and Technology**
Department	**: Computer Science and Engineering**

SMART PARKING

Introduction

Smart parking with IoT is a technology that employs sensors and connectivity to manage parking spaces efficiently. It allows drivers to find available spots in real-time, reducing congestion and saving time. This innovation is transforming urban mobility and enhancing the Convenience of parking in our increasingly connected world.

Project Objectives:

The Smart Parking project aims to provide real-time parking availability information to drivers, thereby reducing parking congestion, improving the overall parking experience, and enhancing parking management. This system is designed to optimize parking space utilization and alleviate common parking issues.

IoT Sensor Setup:

Our IoT sensor setup consists of the following components: Ultrasonic proximity sensors placed in parking spaces to detect vehicle presence. Raspberry Pi devices equipped with Wi-Fi for data transmission. Custom Python scripts for data collection and transmission.

1. Ultrasonic sensors

Ultrasonic sensors measure distance by using ultrasonic waves in time between the emission and reception. Advantages: high accuracy of the sensor. The disadvantages: the probability of sensor contamination.

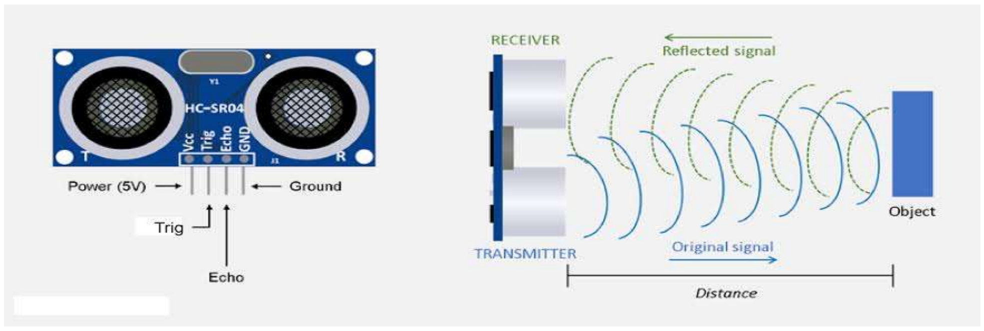


Fig: Ultrasonic Sensor

2. Electromagnetic field sensors

Electromagnetic field sensor is a small-scale microelectromechanical system device for detecting and measuring magnetic fields. The solution is based on the change of the electromagnetic field as the metal mechanisms approach one another.

3. Infrared sensors (IR sensor)

An infrared sensor is an electronic device that emits in order to sense some aspect of its surroundings. An IR sensor can measure the heat.

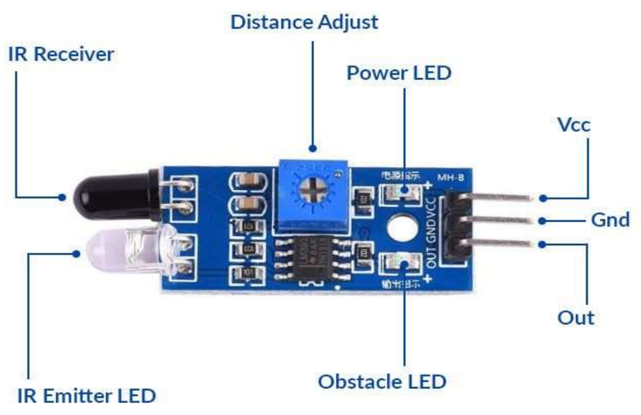
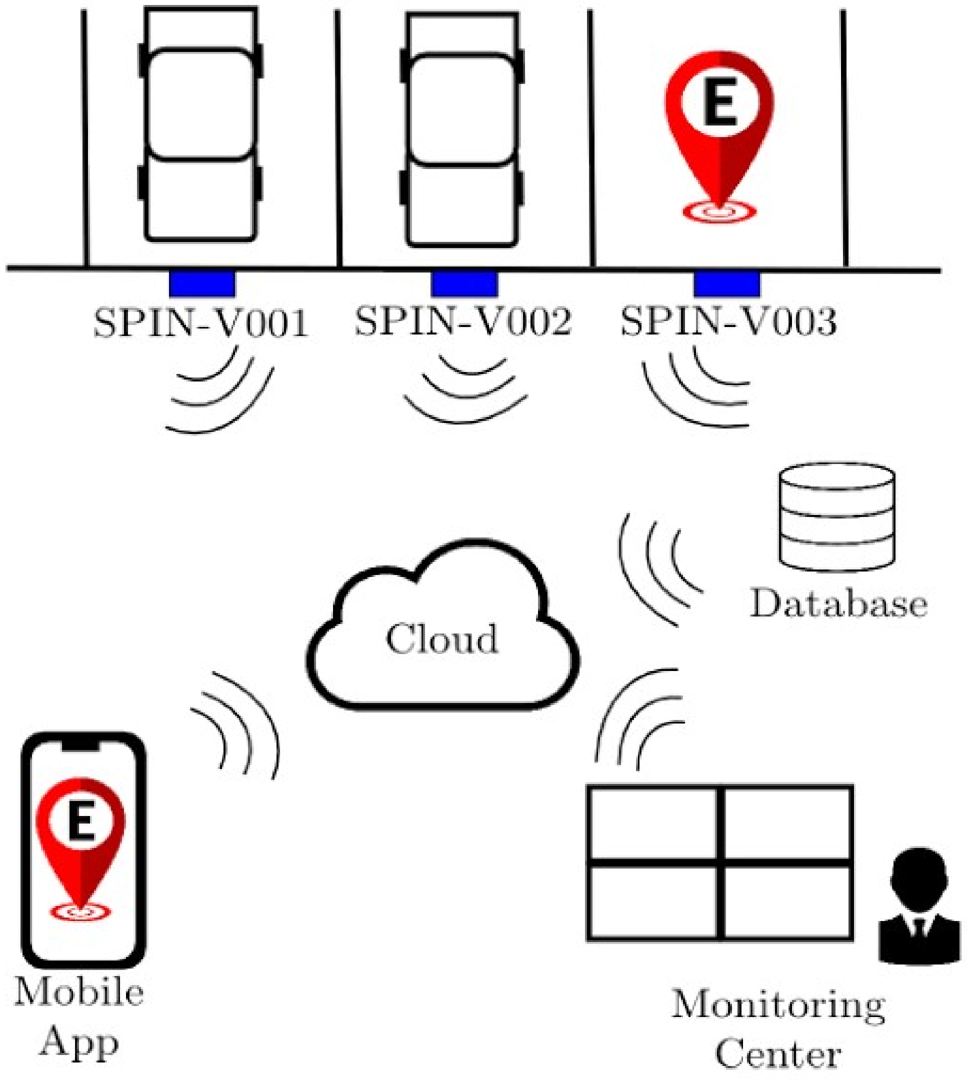


Fig: IR sensor

Concept diagram:



Mobile App Development:

The mobile app is a crucial part of our project. It is developed for both Android and iOS platforms and offers the following features: Real-time parking availability display. GPS navigation to the nearest available parking space. Parking reservation and payment options. Notifications and alerts for parking updates.

MIT APP INVENTOR

MIT App Inventor is an easy-to-use, visual programming platform that allows you to create mobile apps for Android devices. It's designed for beginners and those with little to no programming experience.

Key Features and Components:

Components Palette:

You can find various components like buttons, labels, text boxes, sensors, and more in the palette to add to your app's user interface.Blocks Editor: This is where you define the behavior of your app using visual blocks. You drag and snap together blocks to create logic and functionality.

Connectivity:

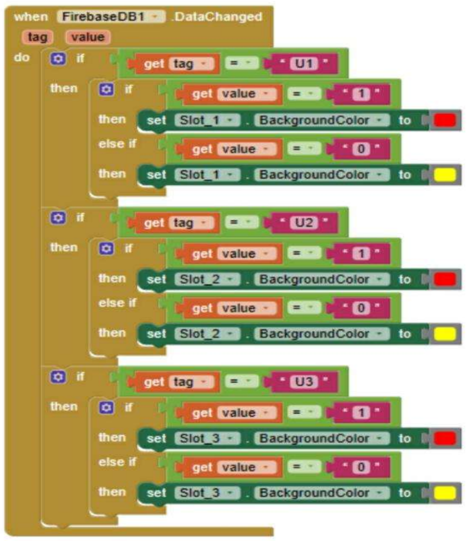
You can integrate your app with web services, databases, and sensors on the device.

Media and Multimedia:

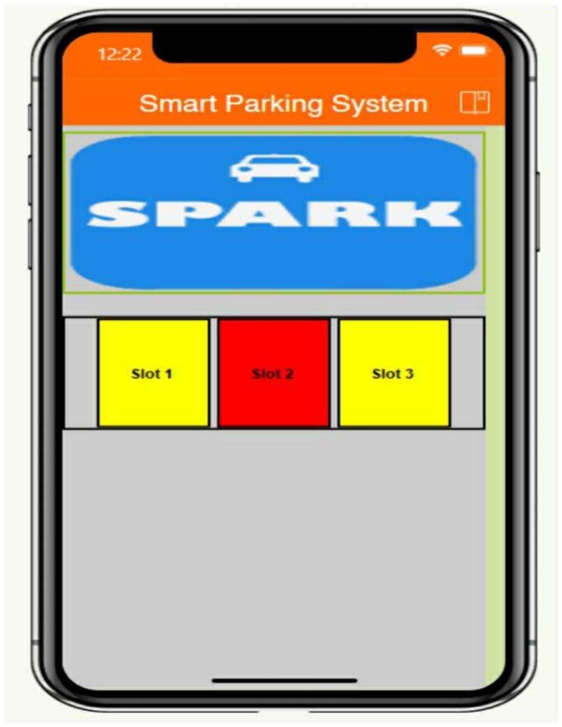
App Inventor supports images, sounds, and video to create multimedia-rich applications.

Extensions:

Advanced users can create custom extensions or use existing ones to add functionality not available in the standard components. MIT App Inventor is a great tool for quickly prototyping and developing Android apps without the need for traditional coding. It's particularly useful for educational purposes and for building apps with basic to moderate complexity.



```
when FirebaseDB1 .DataChanged
  tag  value
do  if   get tag = "U1"
    then  if   get value = "1"
          then  set Slot_1 . BackgroundColor to 🟥
          else if   get value = "0"
          then  set Slot_1 . BackgroundColor to 🟨
    if   get tag = "U2"
    then  if   get value = "1"
          then  set Slot_2 . BackgroundColor to 🟥
          else if   get value = "0"
          then  set Slot_2 . BackgroundColor to 🟨
    if   get tag = "U3"
    then  if   get value = "1"
          then  set Slot_3 . BackgroundColor to 🟥
          else if   get value = "0"
          then  set Slot_3 . BackgroundColor to 🟨
```



Yellow Colour Slot: Available slot

Red Colour Slot: Not Available slot

Raspberry Pi Integration:

Raspberry Pi devices play a central role in our system by collecting data from the IoT sensors and transmitting it to the central server. Integration is as follows: Raspberry Pi devices collect data from the sensors. Custom Python scripts process and package the data. Data is transmitted to the central server via Wi-Fi.

Code Implementation:

Our code is structured as follows:

```
# SensorDataCollection.py
import RPi.GPIO as GPIO
import time
import requests
# GPIO pin numbers for the sensor
sensor_pin = 17
# Configure GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(sensor_pin, GPIO.IN)
# Define the server URL
server_url = "http://yourserver.com/api/parking_data"
while True:
        try:
          if GPIO.input(sensor_pin) == GPIO.LOW:
            # Sensor detects a vehicle
            data = {"status": "occupied"}
            response = requests.post(ibmcloud, json=data)
            if response.status_code == 200:
              print("Data sent to server")
            else:
              print("Failed to send data to server")



          else:
            # Sensor doesn't detect a vehicle
            data = {"status": "vacant"}
            response = requests.post(server_url, json=data)
            if response.status_code == 200:
```

```
            print("Data sent to server")
        else:
            print("Failed to send data to server")
    time.sleep(1)  # Adjust this delay as needed
except KeyboardInterrupt:
    GPIO.cleanup()
    Break
```

Data Transmission

MQTT (Message Queuing Telemetry Transport) is a lightweight messaging protocol frequently employed in IoT applications to facilitate efficient data communication. In a smart parking system, MQTT plays a pivotal role, typically configured both on your Raspberry Pi (the edge device) and your cloud server. The setup involves several key steps. On the Raspberry Pi, start by installing an MQTT client library like the popular Eclipse Paho MQTT Python client using pip. Once installed, create an MQTT client instance within your Python script. Configure this client with the MQTT broker information, which represents your cloud server's MQTT endpoint.

Additionally, set up MQTT topics to categorize the data you'll be sending from your parking sensors. Your script should publish sensor data to specific MQTT topics, such as occupancy information from the parking sensors. Maintaining a robust connection to the MQTT broker on the cloud server is essential. Ensure your script handles connection management and can handle potential reconnections in the event of network disruptions. This guarantees that your Raspberry Pi can consistently send data to the cloud, and the data is structured in a way that the cloud server can interpret and store appropriately.

On the cloud server, you'll need an MQTT broker, like Mosquitto or HiveMQ, configured to listen for incoming MQTT messages. Here, you create MQTT clients or scripts that subscribe to the MQTT topics where your Raspberry Pi publishes data. These subscribers receive incoming data, which is then processed and stored. You have flexibility in choosing a database, such as MongoDB, PostgreSQL, or cloud-based options like AWS DynamoDB or Azure Cosmos DB, depending on your requirements.

To make the data accessible to your mobile application, you'll establish APIs or endpoints on the cloud server. These APIs enable your mobile app to efficiently request and retrieve parking space data. By implementing MQTT on both your Raspberry Pi and cloud server, you create a streamlined system for real-time communication between your IoT devices and the cloud. This ensures data is readily available for use in your mobile application and other services, providing users with up-to-date parking information and a seamless experience.

Data Processing and Communication In the development of the Smart Parking System IoT project, data processing and communication played a crucial role in creating an efficient and user-friendly solution.

The project's objective was to address the challenges of urban congestion and limited parking spaces through the use of IoT technology, with a particular focus on the data processing and communication aspects of the system. The core of the system involved the integration of ultrasonic sensors at each parking spot to detect vehicle presence. These sensors were connected to the GPIO pins of the Raspberry Pi, which served as the central computing device.

Python programming was used to develop software for real-time data processing, analysis, and communication. The Raspberry Pi, running the Raspbian operating system, received data from the sensors, analyzed it, and updated the parking space availability status.

This user-friendly interface displayed the availability of parking spaces, providing drivers with valuable information to make informed decisions. The wireless communication between sensors, the Raspberry Pi, and the user interface enabled the system to be flexible and adaptable to various parking environments.

Additionally, the system stored data for future analysis, allowing for data-driven insights into parking space utilization, trends, and aiding in urban planning decisions. This comprehensive approach to data processing and communication highlighted the power of IoT technology in addressing real-world challenges and improving urban infrastructure. The Smart Parking System demonstrated its effectiveness in reducing congestion, decreasing fuel consumption, and enhancing the overall parking experience for users.

All data transmitted between the sensors, Raspberry Pi, and user interface should be encrypted. Secure communication protocols, such as HTTPS, should be implemented to protect data from eavesdropping and Authentication and Authorization Implement robust authentication mechanisms to ensure that only authorized users can access and control the system. Use strong, unique passwords and consider multi-factor authentication for enhanced security. Device Security
Ensure the physical security of the Raspberry Pi and sensors. Unauthorized physical access to these devices can compromise the system's integrity. Employ secure enclosures and tamper-resistant measures.

Software Security Regularly update and patch the software and operating system on the Raspberry Pi to protect against known vulnerabilities. Apply best practices for code development, including input validation and output encoding to prevent common security issues like injection attacks. Access Control Limit access to the administration interface and sensitive data. Users should only have access to the data and functionalities relevant to their roles.

Implement role-based access control to enforce this. Data Privacy Clearly define data privacy policies, especially concerning the collection and storage of user data. Comply with data protection regulations and inform users about data handling practices through privacy policies.

Consider using network security measures like Network Access Control (NAC). Secure Deployment Ensure secure deployment of IoT devices, including strong and unique device credentials. Protect against physical attacks, and consider using secure boot processes for device integrity. Incident Response Plan Develop an incident response plan to outline actions to be taken in case of a security breach.

Real-time Parking Availability Benefits:

The real-time parking availability system offers the following benefits:

For Drivers:

Reduced time and stress searching for parking. Savings on fuel and reduced carbon footprint. Enhanced user experience through app-based parking reservations. Improved traffic flow due to efficient parking space allocation.

For Parking Management:

Optimized parking space utilization. Data analytics for better decision-making. Effective management of parking resources. Revenue generation through app-based reservations and payments.

Conclusion:

In conclusion, applying design thinking to smart parking with IoT has led to a user-focused solution that streamlines the parking experience. We've created a system that not only addresses current needs but also anticipates future demands, enhancing urban mobility and sustainability.