

SMART PARKING

**IOT SENSOR SYSTEM AND
RASPBERRY Pi INTEGRATION**

SMART PARKING MANAGEMENT SYSTEM USING IOT

Purpose:

The IoT Smart Parking System is an innovative project aimed at revolutionizing urban parking management. By integrating IoT technology with a Python-based control system, this project addresses the growing challenges of urban congestion and parking inefficiencies.

Program Overview:

The Python program plays a pivotal role in processing and interpreting data received from the sensors. It provides a user-friendly interface for accessing parking availability information in real time. Moreover, it incorporates intelligent algorithms to optimize parking space allocation based on occupancy trends.

Objectives:

The primary objectives of this project are to:

1. Provide accurate and real-time information on parking space availability.
2. Optimize parking space allocation to reduce congestion and enhance user convenience.
3. Facilitate seamless interaction between users and the smart parking system.

Key Features:

- Real-time monitoring of parking space occupancy.
- Dynamic allocation of parking spaces based on user demand.
- Integration with mobile applications for user accessibility.
- Data analytics for trend analysis and optimization.

Design Principles:

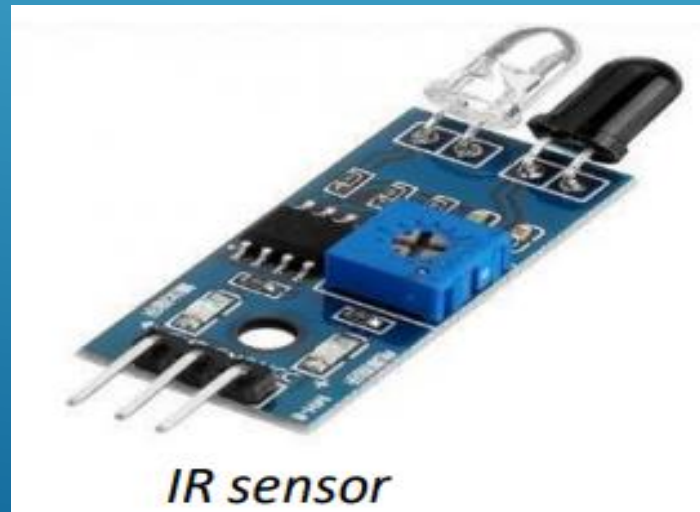
The Python program follows a modular design, allowing for easy scalability and maintenance. It adheres to coding best practices to ensure efficiency and reliability in a real-world urban environment. Design For Smart Parking Management

Design For Smart Parking Management:

Components Required:

Hardware:

- Node MCU ESP8266
- IR Sensor- 5 nos
- Servo Motor-2nos




CODE:

```
import time from machine
import Pin, Servo, I2C
from ntptime import settime from
umqtt.robust import MQTTClient
# WiFi and MQTT settings
WIFI_SSID = "YourWiFiSSID"
WIFI_PASSWORD = "YourWiFiPassword"
MQTT_BROKER = "io.adafruit.com"
MQTT_PORT = 1883
MQTT_NAME = "aschoudhary"
MQTT_PASS = "1ac95cb8580b4271bbb6d9f75d0668f1"
# Entry and exit sensors
carEnter = Pin(4, Pin.IN)
carExited = Pin(5, Pin.IN)
entrysensor = False exitsensor=False
```

```
# Servo configuration servo_gate = Servo(Pin(16))
# Servo for gate
# Parking slots
s1 = Pin(13, Pin.IN)
s2 = Pin(12, Pin.IN)
s3 = Pin(0, Pin.IN)
s1_occupied = False
s2_occupied = False
s3_occupied = False
# Initialize I2C for OLED display
(if used) i2c = I2C(-1, scl=Pin(22), sda=Pin(21))
# Initialize OLED display and write functions to display data (not shown here)
# MQTT topics
entry_gate_topic = "{}f/EntryGate".format(MQTT_NAME)
exit_gate_topic = "{}f/ExitGate".format(MQTT_NAME)
cars_parked_topic = "{}f/CarsParked".format(MQTT_NAME)
# MQTT callback functions (not shown here)
# Initialize MQTT
client mqtt_client = MQTTClient(MQTT_NAME, MQTT_BROKER, port=MQTT_PORT,
user=MQTT_NAME, password=MQTT_PASS) mqtt_client.set_callback(sub_cb)
# Main loop while True:
# Check entry and exit sensors
```

```
entrysensor = not carEnter.value()
exitsensor = not carExited.value()
if entrysensor:
    # Car entered
    # Increment count and open gate
    count += 1
    servo_gate.angle(0)
    time.sleep(3)
    servo_gate.angle(80)
if exitsensor:
    # Car exited
    # Decrement count and open gate
    count -= 1
    servo_gate.angle(0)
    time.sleep(3)
    servo_gate.angle(80)
# Publish the number of parked cars to the MQTT topic
mqtt_client.publish(cars_parked_topic,
str(count))
# Check parking slots if s1.value() and not s1_occupied: # Slot 1 is occupied
s1_occupied = True # Publish entry time to the MQTT topic
mqtt_client.publish(entry_slot1_topic, get_current_time()) if not s1.value() and
s1_occupied:
```

```
# Slot 1 is available
s1_occupied = False
# Publish exit time to the MQTT topic
mqtt_client.publish(exit_slot1_topic, get_current_time())
# Repeat the same logic for other parking slots (s2 and s3)
# Handle MQTT subscriptions and messages
mqtt_client.check_msg()
# Delay for a moment to avoid excessive message publishing
time.sleep(1)
```

Several white lines of varying lengths and slopes are positioned in the bottom right corner of the slide, creating a modern, abstract graphic element.

User Interaction: Users can access the system through a dedicated mobile application. The intuitive interface displays real-time parking availability and allows users to reserve parking spaces in advance.

Expected Outputs: Upon successful execution, the Python program provides users with up-to-date information on available parking spaces. It also dynamically updates the allocation of spaces as vehicles enter and exit the parking area.

Testing and Validation: Extensive testing was conducted to validate the accuracy and reliability of the system. Simulated scenarios were used to mimic various parking situations, and the program consistently provided accurate results.

Challenges Faced: One of the main challenges encountered was optimizing the communication between the microcontroller and Python program to ensure real-time data processing. Additionally, fine-tuning the algorithms for efficient parking space allocation was a critical aspect of the development process.

OUTPUT :

After finishing the procedures the final output looks like given below.

