# SMART PARKING SYSTEM USING IOT

Smart parking systems using IoT typically incorporate various key elements and technologies to efficiently manage parking spaces and improve the overall parking experience. Some of the commonly used elements in smart parking systems using IoT include:

**IoT Sensors:** IoT sensors are placed in parking spaces to detect vehicle presence. These sensors can be ultrasonic, magnetic, infrared, or video cameras. They collect real-time data about parking space occupancy and relay this information to a central system.

**Communication Networks:** IoT sensors transmit data to a central server or cloud platform using communication protocols like Wi-Fi, LoRa, Zigbee, or cellular networks. These networks enable remote monitoring and control.

**Cloud Computing:** Parking data is typically processed and stored in the cloud. Cloud platforms allow for data analytics, remote access, and scalability. The data can be analysed to derive insights and optimize parking operations.

**Mobile Applications:** Users can access parking information through dedicated mobile apps. These apps can provide real-time availability information, reservations, and navigation to available parking spaces.

**Web Interface:** A web-based interface can be used for both administrative purposes and user access. It allows parking operators to manage the system and users to check parking availability and make reservations.

**Automated Payment Systems:** IoT-enabled smart parking systems often feature automated payment systems. Users can make payments through mobile apps, credit cards, or contactless payment methods.

**Smart Parking Meters:** Traditional parking meters are replaced with smart meters that accept payments digitally, provide information on parking availability, and send alerts for parking violations.

**LED Displays and Signage:** Dynamic LED displays and digital signage boards at parking entrances display real-time parking availability and guide drivers to available parking spots.

**Machine Learning and AI:** Machine learning and artificial intelligence are used to predict parking spot availability, analyse traffic patterns, and optimize parking operations. This can help in resource allocation and congestion management.

**Geolocation Services:** GPS and geolocation services are used to navigate drivers to available parking spots. These services can be integrated into mobile apps and in-vehicle navigation systems.

**Security and Surveillance:** Security cameras and surveillance systems are often integrated into smart parking solutions to enhance safety and monitor parking areas.

**QR Codes and RFID:** QR codes and RFID tags are used for access control and to identify registered users. They can be scanned for entry and exit from parking facilities.

**Environmental Sensors:** In some cases, environmental sensors can be used to monitor air quality, temperature, and humidity in parking facilities, providing a more holistic view of the parking environment.

**Data Analytics and Reporting:** Data collected from IoT sensors can be analyzed to generate reports, trends, and insights. This information is valuable for decision-making and improving parking operations.

**Electric Vehicle Charging Infrastructure:** For electric vehicle (EV) parking, smart parking systems often include EV charging stations. These stations can be monitored and managed remotely.

**License Plate Recognition (LPR):** LPR technology is used to automate entry and exit processes, manage access, and identify vehicles in parking facilities.

**Energy Management:** Energy-efficient lighting and HVAC systems can be integrated into parking facilities, controlled based on occupancy, and monitored for energy savings.

**Emergency and Safety Features:** Smart parking systems may include features such as emergency call buttons, lighting in deserted areas, and assistance services for users in distress.

These elements work together to create an efficient and user-friendly smart parking system that optimizes space utilization, reduces traffic congestion, and enhances the overall parking experience.

Creating a complete Python script for a smart parking system based on IoT sensors is a substantial project. In this example, I'll provide a simplified Python script for simulating IoT sensors for parking space occupancy detection using Python's random module. This script will simulate sensor data and send it to a server. Note that this is a basic example and does not involve real IoT hardware.

**Python Script for Simulated IoT Sensors:**

```
python
import random
import requests
import time
# Server configuration
server_url = "http://localhost:5000"   # Replace with the URL of your server
sensor_id = 1   # Unique ID for this sensor (for multiple sensors)

# Simulate IoT sensor data
def simulate_sensor_data ():
   while True:
      # Simulate parking space occupancy (0: vacant, 1: occupied)
      occupancy = random. Choice ([0, 1])
     #Send sensor data to the server
      send_sensor_data(occupancy)
     # Simulate sensor reading every 5 seconds
      time.sleep(5)
    # Send sensor data to the server
     def send_sensor_data(occupancy):
   data = {
      "sensor_id": sensor_id,
```

```
        "occupancy": occupancy
    }
 try:
        response = requests.post(f"{server_url}/update_sensor", json=data)
        if  response.status_code == 200:
            printf("Sensor {sensor_id} - Occupancy: {occupancy} - Data sent
successfully")
        else:
            printf("Sensor {sensor_id} - Error sending data to the server")
    except requests.exceptions.RequestException as e:
        printf("Sensor {sensor_id} - Request error: {e}")
if _name_ == "_main_":
    simulate_sensor_data ()
```

This script simulates an IoT sensor sending data to a server. It generates random occupancy status (0 for vacant and 1 for occupied) and sends it to the server every 5 seconds. Please note that you need to replace server_url with the actual URL of your server, and ensure that your server can handle incoming sensor data.

Additionally, on the server side, you should have an endpoint to receive sensor data. In the case of a simple Flask server, the endpoint might look like this:

```
python
from flask import Flask, request, jsonify
app = Flask(__name)
sensor_data = {}
@app.route('/update_sensor', methods=['POST'])
def update_sensor ():
```

```
    data = request.get_json ()
    sensor_id = data.get("sensor_id")
    occupancy = data.get("occupancy")
    sensor_data[sensor_id] = occupancy
    return "Sensor data received", 200

if _name_ == '_main_':
    app.run(debug=True, port=5000)
```

Make sure you adapt the server code to your specific requirements for processing and using the sensor data.

To create a smart parking system using Arduino and IoT (Internet of Things) with Python as the backend, you will need an Arduino board with Wi-Fi capabilities (such as an ESP8266 or ESP32), IoT sensors (such as ultrasonic or infrared sensors), and a Python server running on a computer or cloud platform. In this example, we'll use an Arduino ESP8266 with an ultrasonic sensor and Python for the server.

Here's a simple example of Arduino code to detect parking space occupancy using an ultrasonic sensor and send data to a Python server:

**Arduino Code (C/C++):**

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
const char* ssid = "YourWiFiSSID";
const char* password = "YourWiFiPassword";
const char* serverAddress = "YourServerAddress"; // Replace with your server address
int trigPin = D1;  // Trigger pin of the ultrasonic sensor
```

```
int echoPin = D2;   // Echo pin of the ultrasonic sensor
void setup() {
  Serial.begin(115200);
  pinMode (trigPin, OUTPUT);
  pinMode (echoPin, INPUT);
  WiFi.begin(ssid, password);
  while (WiFi.status ()  != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");
}
void loop() {
  long duration;
  float distance;
  // Send a trigger pulse to the ultrasonic sensor
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  // Read the time taken for the ultrasonic pulse to return
  duration = pulseIn(echoPin, HIGH);
    // Convert the time to distance in centimeters
  distance = (duration / 2) / 29.1;
// Send distance data to the server
  sendToServer(distance);
```

```
  delay(1000);
}
void sendToServer(float distance) {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;
       String url = serverAddress + "/update"; // Replace with your server endpoint
    url += "?distance=" + String(distance);
    http.begin(url);
    int httpResponseCode = http.GET();
    if (httpResponseCode > 0) {
      String response = http.getString();
      Serial.println("HTTP Response Code: " + String(httpResponseCode));
      Serial.println("Server Response: " + response);
    } else {
      Serial.println("Error sending data to server");
    }
    http.end();
  }
}
```

**Python Server Code:**

You will need a Python server to receive data from the Arduino. You can use a web framework like Flask or Django to create a server. Here's a simple example using Flask:

**python**

```
from flask import Flask, request
app = Flask(__name)
@app.route('/update', methods=['GET'])
def update():
    distance = request.args.get('distance')
    if distance is not None:
        printf("Parking space distance: {distance} cm")
        # Add your logic to handle parking space occupancy here
        # You can update a database, trigger notifications, etc.
        return "Data received"
    else:
        return "Invalid data"
if _name_ == '__main':
    app.run(host='0.0.0.0', port=5000)
```

In this Python server, the /update endpoint receives the distance data from the Arduino and allows you to handle parking space occupancy data as needed.

Please note that this is a simplified example. In a real-world scenario, you would implement security measures and data processing according to your specific requirements.