SMART TRAFFIC MANAGEMENT SYSTEM

Components

IoT Device:

This could be a Raspberry Pi, Arduino, ESP8266, or another micro controllerwith internet connectivity capabilities.

Distance Sensors:

Ultrasonic or infrared sensors to detect the presence of vehicles in parkingspots.

Internet Connectivity:

The IoT device should be able to connect to the internet, either through Wi-Fi or Ethernet.

IoT Platform:

You'll need an IoT platform (e.g., AWS IoT, Google Cloud IoT, or Azure IoT)to receive and store sensor data.

Python Script:

Develop a Python script to read data from the sensors and send it to your IoT platform.

Steps:

1. Import Required Libraries:

Import necessary Python libraries for working with sensors, IoT, and datahandling.

2. IoT Configuration:

Set up IoT configuration parameters. This includes the IoT endpoint, certificates, and client ID.

3. Sensor Setup:

Configure the distance sensors for detecting vehicle presence in parking spots.

4. IoT Client Setup:

Configure and initialize the AWS IoT MQTT client or the IoT client for your chosen platform.

5. Connect to IoT:

Connect to your IoT platform using the configured IoT client.

6. Data Collection and Sending:

In the main loop (infinite loop), do the following:

a. Read Distance Sensors:

Read data from the distance sensors to detect vehicle presence.

b. Create a JSON Message:

Create a JSON message containing the parking spot number, vehicle presencestatus, and any other relevant data.

c. Send Data to IoT:

Publish the JSON message to an IoT topic dedicated to the Smart Parking system using the MQTT or other protocols supported by your IoT platform.

d. Error Handling:

Implement error-handling mechanisms to catch any exceptions that might occur during data collection or sending.

e. Sleep Between Readings:

Add a sleep period between readings. The interval can vary depending on your requirements. For example, you can collect and send data every few seconds or minute

Here's a code snippet outline for Smart Parking using Raspberry Pi andAWS IoT:

```
import paho.mqtt.client as mqtt
import time

# MQTT broker address and port
BROKER_ADDRESS = "mqtt.example.com"
PORT = 1883

# MQTT topic and message for Doppler sensor
TOPIC = "doppler/sensor"
MESSAGE = "Motion detected by Doppler sensor!"

# Callback function when the client connects to the MQTT broker
def on_connect(client, userdata, flags, rc):
    print("Connected to MQTT broker with result code "+str(rc))

# Callback function when the message is published

def on_publish(client, userdata, mid):
    print("Message Published")


client = mqtt.Client()
```

```
# Set up callback functions
client.on_connect = on_connect
client.on_publish = on_publish

# Connect to the MQTT broker
client.connect(BROKER_ADDRESS, PORT, 60)

try:
    while True:
        # Publish a message to the specified topic
        client.publish(TOPIC, MESSAGE)
        print(f"Published message: '{MESSAGE}' to topic: '{TOPIC}'")
        time.sleep(5)  # Publish message every 5 seconds (adjust as needed)

except KeyboardInterrupt:
    print("Publishing stopped by the user.")

finally:
    # Disconnect from the MQTT broker
    client.disconnect()
```

Explanation:

Sensor used in this project is doppler type.

A Doppler sensor, in the context of motion detection, is a type of sensor that uses the Doppler effect to detect movement. The Doppler effect is a phenomenon in physics where the frequency of waves (such as sound or light waves) changes when the source of the waves is moving relative to an observer. In the case of Doppler motion sensors, they use the Doppler effect to detect motion in their surroundings.

Here's how a Doppler motion sensor works:

Transmission of Waves: The Doppler sensor emits radio waves or microwaves into its surroundings.

Reflection of Waves: When these waves encounter a moving object, such as a person or a vehicle, the frequency of the reflected waves changes due to the Doppler effect. If the object is moving toward the sensor, the frequency increases; if it's moving away, the frequency decreases.

Detection of Frequency Change: The sensor detects these frequency changes in the reflected waves.

Motion Detection: By analyzing the changes in frequency, the sensor can determine if there is motion in its field of view. If there is a significant change in frequency, the sensor registers motion.

Doppler sensors are commonly used in this applications, including:

Traffic Systems: Doppler sensors are used in traffic management systems to detect the movement of vehicles on roads. They can be employed in traffic lights or to monitor traffic flow.

Code Explanation:

1.import paho.mqtt.client as mqtt

import time

MQTT broker address and port

BROKER_ADDRESS = "mqtt.example.com"

PORT = 1883

In this code, the paho.mqtt.client module is imported to use the MQTT client functionalities. The time module is imported to handle time-related operations such as adding delays between message publications.

2. TOPIC = "doppler/sensor"
MESSAGE = "Motion detected by Doppler sensor!"

# Callback function when the client connects to the MQTT broker
def on_connect(client, userdata, flags, rc):
    print("Connected to MQTT broker with result code "+str(rc))
def on_publish(client, userdata, mid):
    print("Message Published")

The on_connect function is a callback that gets triggered when the MQTT client successfully connects to the broker. It takes four parameters:

client: The MQTT client instance.
userdata: User data that can be set optionally when initiating the MQTT client.
flags: Flags associated with the connection.
rc: The connection result code. A value of 0 indicates a successful connection.
In this function, it prints a message indicating that the client has successfully connected to the MQTT broker and displays the result code for reference.

```
3. client = mqtt.Client()

# Set up callback functions
client.on_connect = on_connect
client.on_publish = on_publish

# Connect to the MQTT broker
client.connect(BROKER_ADDRESS, PORT, 60)

try:
    while True:
        # Publish a message to the specified topic
        client.publish(TOPIC, MESSAGE)
        print(f"Published message: '{MESSAGE}' to topic: '{TOPIC}'")
        time.sleep(5)  # Publish message every 5 seconds (adjust as needed)

except KeyboardInterrupt:
    print("Publishing stopped by the user.")

finally:
    # Disconnect from the MQTT broker
    client.disconnect()
```

The connect() method is called on the client to establish a connection with the MQTT broker. It takes three parameters:

BROKER_ADDRESS: The address of the MQTT broker to which the client should connect (e.g., "mqtt.example.com"). Replace this with your actual broker address.

PORT: The port number on which the MQTT broker is running. The default MQTT port is 1883.

60: This is the keep-alive time in seconds. It specifies how long the client and the broker should maintain the connection in the absence of communication. A value of 60 seconds is common