AN PRESENTATION ON AIR QUALITY MONITORING

SUBMITTED BY

K.Badri Prasath

P.Ganapathi

K.Anjali

K.Karthika

P.Indhumathi

SMART HOME AUTOMATION

Abstract

In this paper, an IoT-based indoor air quality monitoring platform, consisting of an air quality-sensing device called “Smart-Air” and a web server, is demonstrated.

This platform relies on an IoT and a cloud computing technology to monitor indoor air quality in anywhere and anytime. Smart-Air has been developed based on the IoT technology to efficiently monitor the air quality and transmit the data to a web server via LTE in real time.

The device is composed of a microcontroller, pollutant detection sensors, and LTE modem. In the research, the device was designed to measure a concentration of aerosol, VOC, CO, CO2, and temperature-humidity to monitor the air quality.

Introduction

Atmospheric conditions continue to deteriorate each year due to the growth of civilization and increasing unclean emissions from industries and automobiles.

Although air is an indispensable resource for life, many people are indifferent to the severity of air pollution or have only recently recognized the problem [1–3]. Among various types of pollutants such as water, soil, thermal, and noise, air pollution is the most dangerous and severe, causing climate change and life-threatening diseases.

According to the World Health Organization (WHO), 90 percent of the population now breathes polluted air, and air pollution is the cause of death for 7 million people every year [4, 5].

Real-time monitoring and a rapid alert system produce an efficient platform for improving indoor air quality. Major contributions of the proposed study are as follows:

(i) We propose the use of the Smart-Air for the precise monitoring of indoor air quality

(ii) We propose the utilization of an IoT for efficient monitoring of real-time data

(iii) We propose the adoption of cloud computing for real-time analysis of indoor air quality

(iv) We originally developed a mobile application to make the proposed IoT system with features of anytime, anywhere

(v) The device has been tested for reliability of the data and the platform has been implemented in a building to test its feasibility

Smart-Air

An accurate data measurement of indoor air quality is the most important factor for the platform.

Thus, Smart-Air was developed to collect

air quality, experimental efforts have focused on verifying the reliability of the sensors.

The sensors were tested for the reliability according to the protocols from the Korea Testing Laboratory that was approved by the Ministry of Environment.

The VOC and CO sensors required calibration before the Smart-Air reliability test and were calibrated in a $1\,\mathrm{m} \times 1\,\mathrm{m} \times 1\,\mathrm{m}$-sized acryl chamber.

Laser Dust Sensor

The laser dust sensor was precalibrated in the factory, so only a reliability test was required to verify the data from Smart-Air.

To test the accuracy of the laser dust sensor installed in the device, two types of experiments were performed based on aerosol concentration.

This method is known to be the most reliable for detection due to the factory calibration.

In the experiment, data from three Smart-Air devices were measured and compared to the data obtained from GRIMM 1109.

Temperature-Humidity Sensor

The temperature-humidity sensor was precalibrated in a factory instead of in a laboratory to produce greater accuracy and reliability.

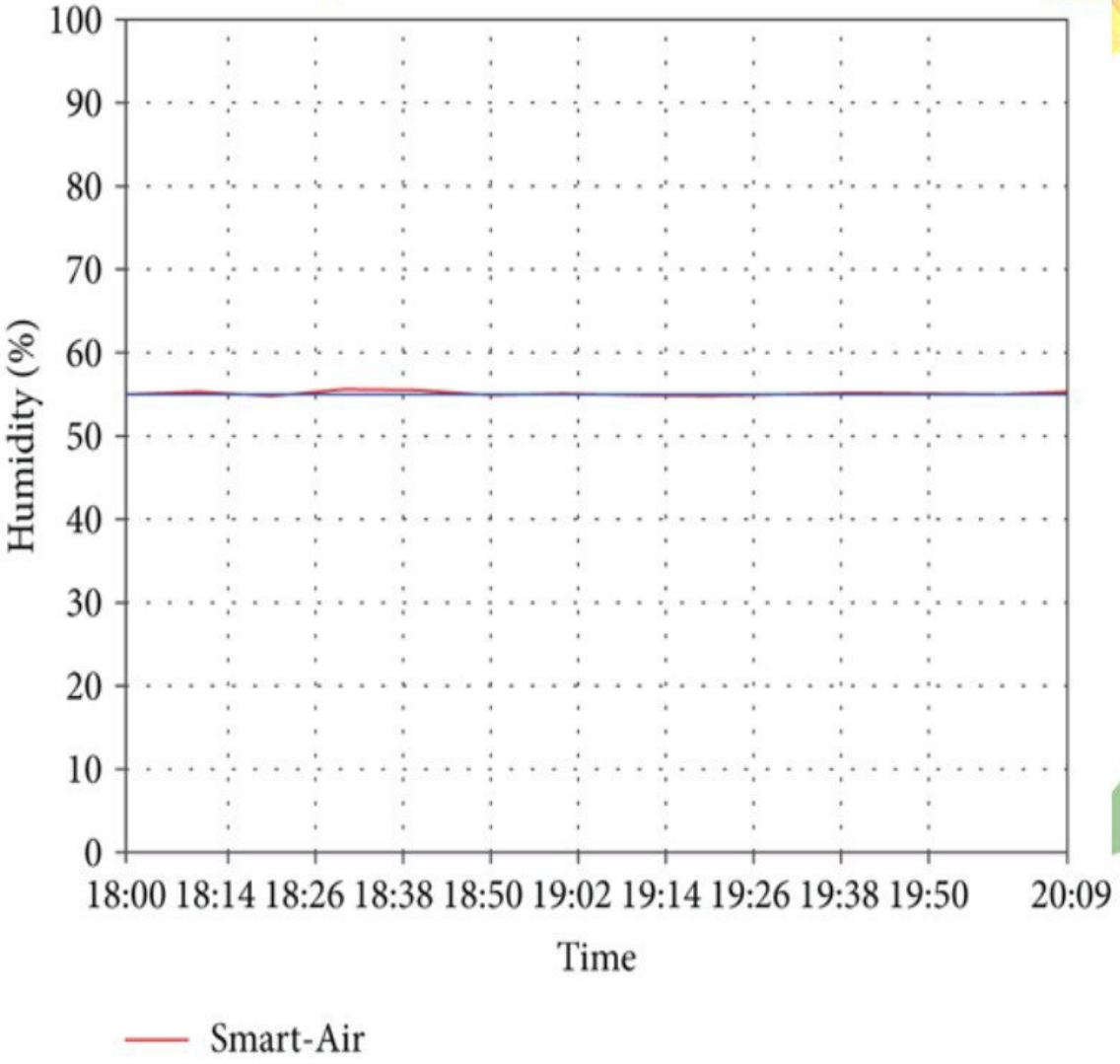
Although additional calibration of the sensor was not required, a reliability test was performed.

Thus, Smart-Air was placed in the chamber for 2 hours with temperature and humidity set points of 19°C and 55%, respectively.

The sensed temperature and humidity were compared to the initial set values for testing the accuracy of the sensor.

The chamber used in the experiment independently maintained specific humidity level and temperature of 19°C and 55%, respectively.

Smart-Air presented measurements as accurate as the set values, verifying the high reliability of the sensor and showing that it did not need extra calibration.

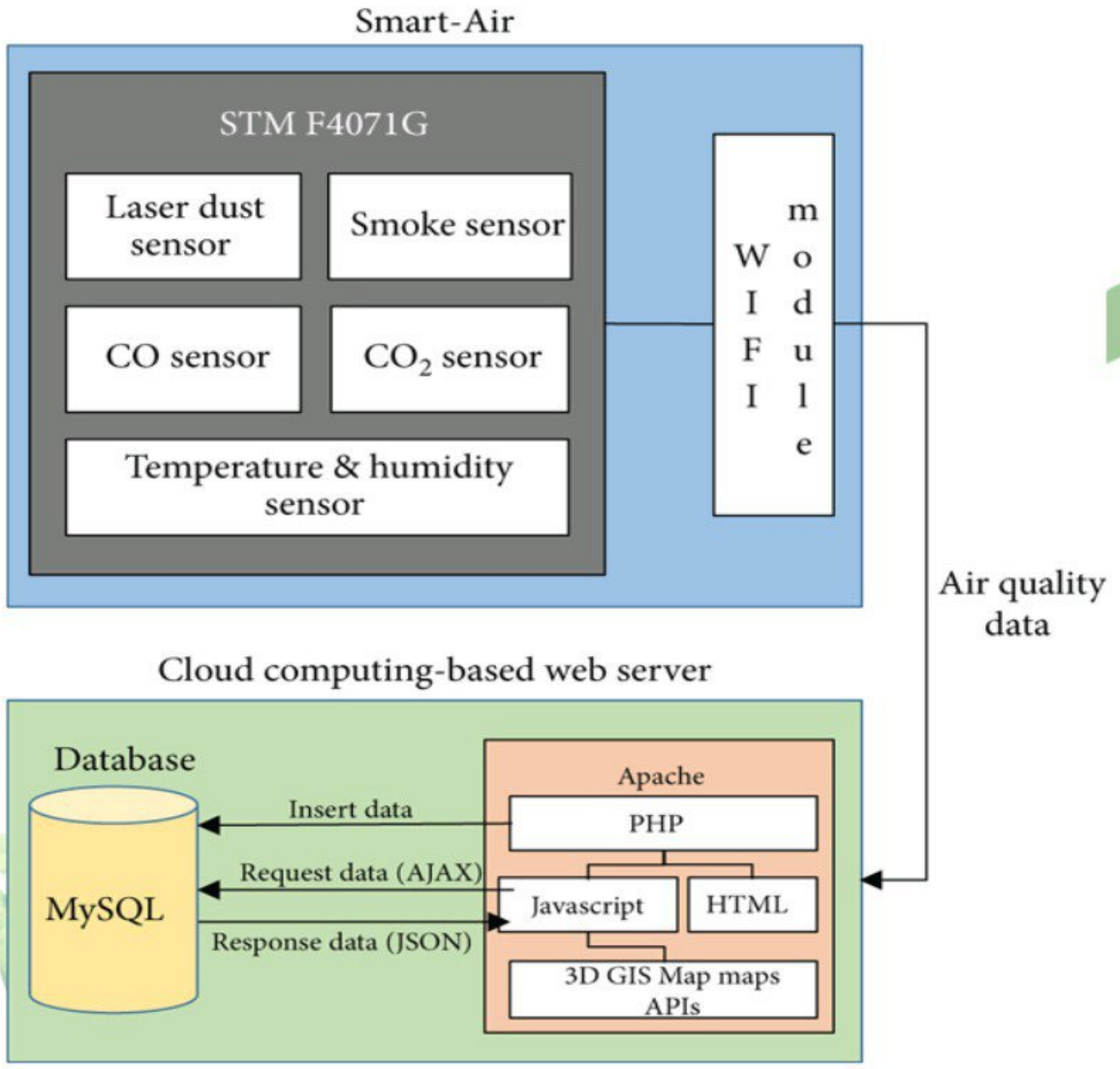


An IoT-Based Air Quality Monitoring Platform

The IoT-based air quality monitoring platform is primarily divided the Smart-Air and the web server. The set of sensing devices necessary to collect the data to analyze air quality comprised a laser dust sensor, a CO sensor, a CO2 sensor, a VOC sensor, and a temperature and humidity sensor.

Each device transmitted data to the web server via the LTE module to determine air quality and visualize the result.

Furthermore, a mobile application was developed for the system to visualize air quality with the web server “anywhere, anytime” in real time.



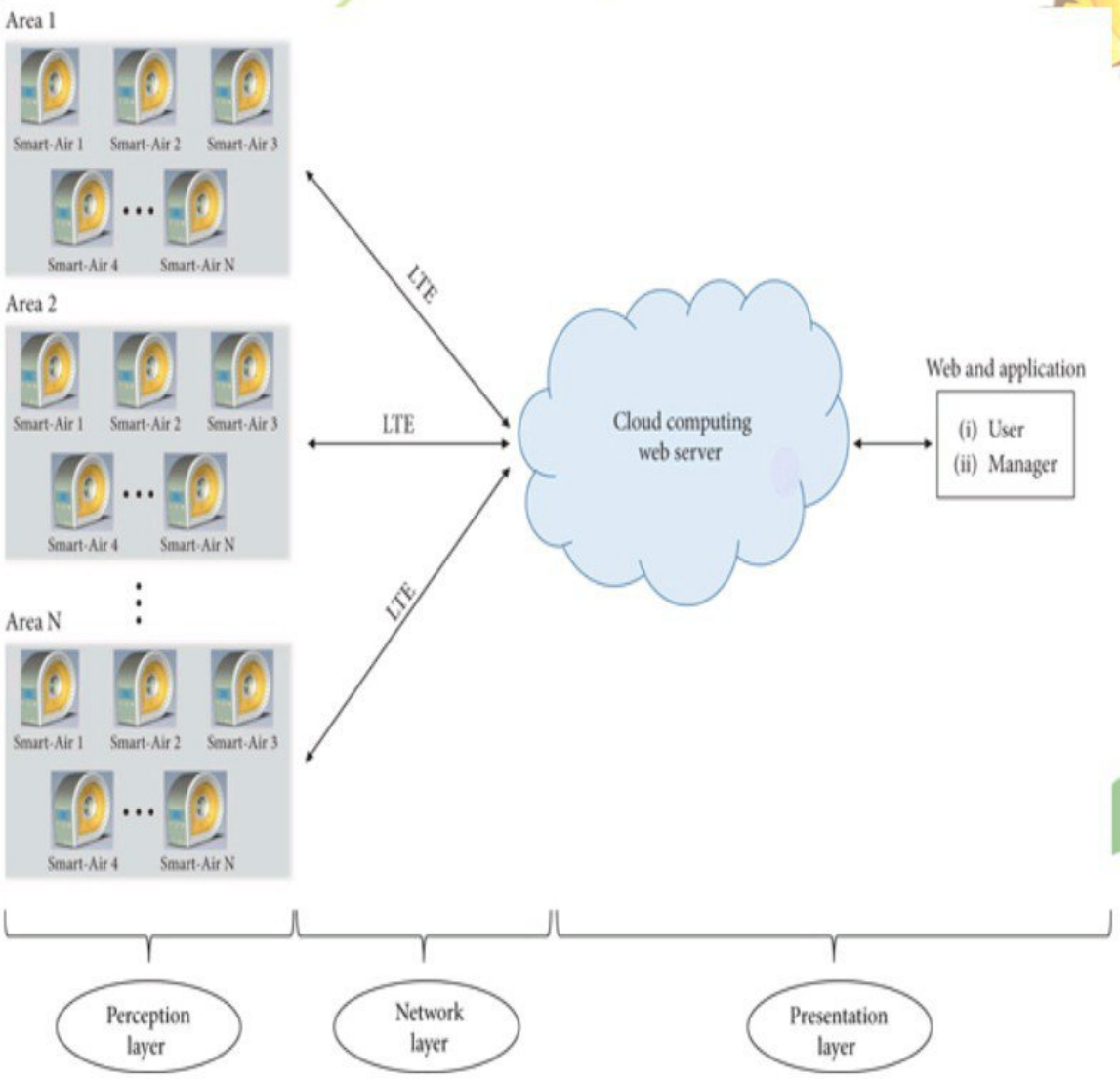
The platform is designed based on an architecture of IoT platform that is mainly comprised of three

components: (i) perception layer, (ii) a network layer, and (iii) presentation layer.

The perception layer is the sensing component to collect data using sensors or any measuring devices.

A block diagram of the IoT-based indoor air quality monitoring platform is shown in diagram 9. For the perception layer of the platform, multiple Smart-Air devices are used for detecting the data needed to analyze the air quality.

Therefore, they can react immediately to improve the air quality.



CODE

```
# Define a class for Air Quality Sensor
class AirQualitySensor:
    def __init__(self, location):
```

```
        self.location = location

    def measure_pollutants(self):
        # Simulated measurements for PM2.5, PM10, CO2, CO, NO2, VOCs
        measurements = {
            'PM2.5': 15,
            'PM10': 20,
            'CO2': 400,
            'CO': 0.5,
            'NO2': 0.02,
            'VOCs': 0.1
        }
        return measurements

# Define a class for Smart Home Automation
class SmartHomeAutomation:
    def __init__(self, sensor):
        self.sensor = sensor
```

```
    def monitor_air_quality(self):
        pollutants =
self.sensor.measure_pollutants()
        return pollutants


    def air_purification(self, pollutants):
        # Simulated air purification based on
pollutant levels
        if pollutants['PM2.5'] > 10 or
pollutants['CO'] > 0.3:
            return "Air purifiers activated"
        else:
            return "Air quality within safe
limits"


# Create an instance of Air Quality Sensor
sensor = AirQualitySensor(location="Living
Room")


# Create an instance of Smart Home
```

Automation

```
home_automation = SmartHomeAutomation(sensor)

# Monitor air quality
pollutants_data = home_automation.monitor_air_quality()
print("Current Air Quality Data:", pollutants_data)

# Activate air purification
purification_result = home_automation.air_purification(pollutants_data)
print(purification_result)
```

Output:

```
Current Air Quality Data: {'PM2.5': 15, 'PM10': 20, 'CO2': 400, 'CO': 0.5, 'NO2': 0.02, 'VOCs': 0.1}
Air purifiers activated
```

Please note that this code is a simplified simulation and does not involve actual IoT communication. In a real-world scenario, you would use dedicated IoT platforms, protocols, and hardware to deploy a functioning system. This example serves as a starting point for understanding the concept.

Smart-Air

When a monitoring area has been determined, the specific types of air pollutants present must be considered.

As mentioned above, Smart-Air has an expandable interface such that multiple sensors can be added to the microcontroller.

Furthermore, the platform can monitor a large area or many areas simultaneously using multiple Smart-Air devices. Then, each device is classified by area to visualize the data.

Each Smart-Air device transmits air quality data to the web server via LTE and automatically indicates the air quality for the specific area by LED color.

displayed by color according to the web server. Additionally, when the specific types of air pollutants in the main page were selected, detailed monitoring of the pollutants was available based on a real-time graph as shown in diagram 13(b). Furthermore, the application alerts the user through a pop-up message when the condition of the air pollutant was moderate or poor.

Results

The goal of the experiment was to perform an initial implementation of the platform to monitor indoor air quality.

Smart-Air wirelessly transmitted the detected data to the web server, which successfully classified the condition of indoor air quality and displayed it via both the web and the application.

Also, the data were saved in the database of the web server as designed such that further studies can be performed on trends of air quality.

Thus, the manager of the building was able to monitor the air quality of the building ubiquitously

and take steps to improve the air quality.

Conclusions

In this paper, the development of an IoT-based indoor air quality monitoring platform is presented. Experiments were performed to verify the air quality measurement device used in the platform based a method suggested by the Ministry of Environment.

We verified the accuracy of indoor air quality monitoring and the desirable performance of the device.

Future work will involve further testing of the device and the platform. In this paper, the experiment focused on testing the reliability of the device and implementing the platform, where more tests are necessary to ensure data accuracy for long time periods.

In addition, ventilation system can be connected to the platform. Thus, the system can be automatically operated to improve the air quality whenever the air quality is not good.

Air Quality Trends Show Clean Air Progress

Nationally, concentrations of air pollutants have dropped significantly since 1990:

Carbon Monoxide (CO) 8-Hour, 79%
Lead (Pb) 3-Month Average, 85% (from 2010)
Nitrogen Dioxide (NO2) Annual, 61%
Nitrogen Dioxide (NO2) 1-Hour, 54%
Ozone (O3) 8-Hour, 21%
Particulate Matter 10 microns (PM10) 24-Hour, 32%
Particulate Matter 2.5 microns (PM2.5) Annual, 37% (from 2000)
Particulate Matter 2.5 microns (PM2.5) 24-Hour, 33% (from 2000)
Sulfur Dioxide (SO2) 1-Hour, 91%
Numerous air toxics have declined with percentages varying by pollutant
Despite increases in air concentrations of

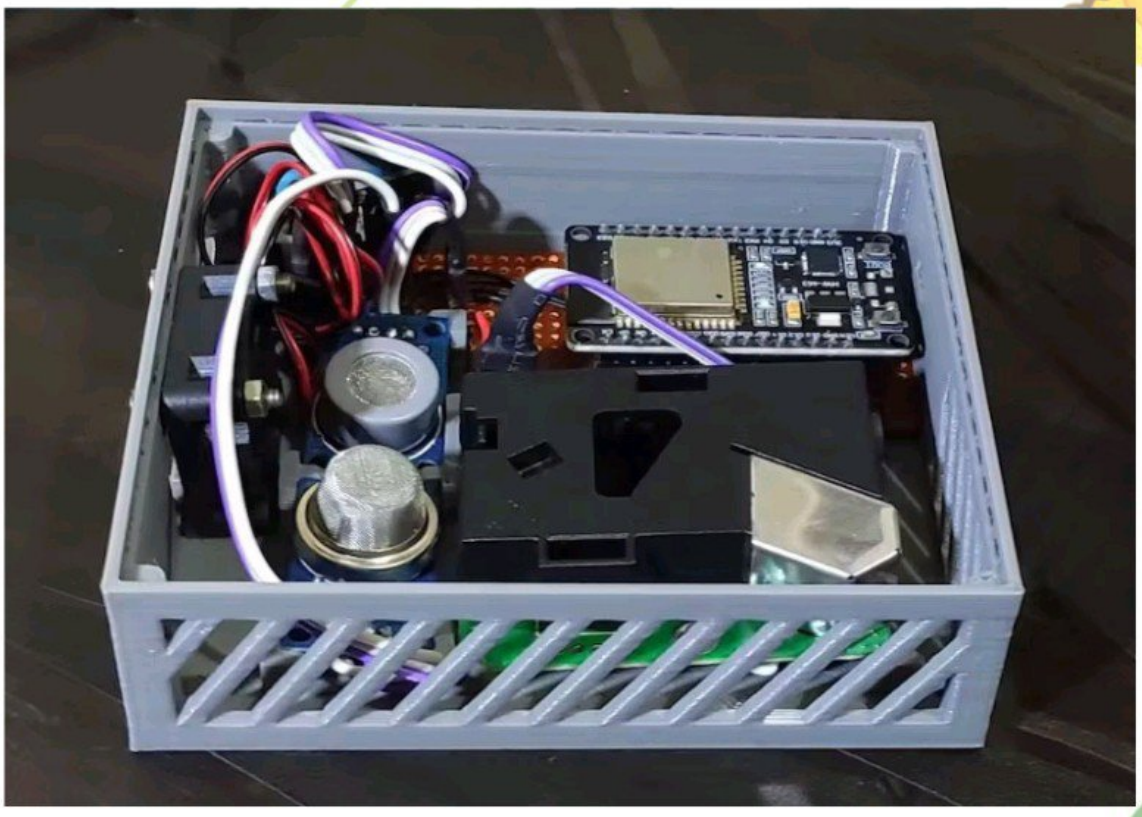
Points on the map indicate the long-term statistical trend direction: decreasing, increasing and no trend. There is insufficient data to determine a trend at sites depicted in gray.

Conclusion

This is how you can build a small scale air quality monitoring system of your own and in the process, learn about Arduino, ESP32, environmental impacts of pollutants etc. on the go.

Air Quality Monitoring System (AQMS)

A compact and portable device for monitoring air quality in your surrounding, even for indoor air quality check.



AIR QUALITY MONITORING USING ESP32

Objective

To measure and monitor the air pollution levels

To detect highly polluted conditions

To alert user as and when the levels cross a specified limit

Components you'll require

-ESP32 system on a chip microcontroller

-MQ135 gas sensor

Software Used

Arduino IDE

Working

Make the connections as per the circuit diagram above.

Connect the ESP32 with your computer/laptop. Select a suitable port and Esp32 board and hit 'Upload' to upload the code into the ESP32.

(I've used the MQ135 gas sensor here which is basically a Air Quality sensor capable of sensing gases like NH3, NOx, alcohol, Benzene, smoke, CO2 and other harmful gases. MQ135 will sense the gases and we will get the Pollution/Air quality level in PPM (parts per million).

The MQ135 sensor will give us output in form of voltage levels and we convert it into PPM (our code and the module libraries will take care of that).

For this project, I've set good air quality levels upto 800 PPM, poor levels in between 800 to 2000 and alert us if it exceeds 2000 PPM.)

concentration). The AQI calculations include the eight pollutants that are PM10, PM2.5, Nitrogen Dioxide (NO2), Sulphur Dioxide (SO2), Carbon Monoxide (CO), ground-level ozone (O3), Ammonia (NH3), and Lead (Pb). However, all of the pollutants are not measured at every location.

Based on the measured 24-hour ambient concentrations of a pollutant, a sub-index is calculated, which is a linear function of concentration (e.g. the sub-index for PM2.5 will be 51 at concentration 31 µg/m3, 100 at concentration 60 µg/m3, and 75 at a concentration of 45 µg/m3). The worst sub-index (or maximum of all parameters) determines the overall AQI.

Code (For ESP32):

```
#include <WiFi.h>
#include <PubSubClient.h>

const char* ssid = "YOUR_WIFI_SSID";
const char* password = "YOUR_WIFI_PASSWORD";
```

```
const char* mqtt_server = "MQTT_BROKER_IP";

WiFiClient espClient;
PubSubClient client(espClient);

int gasSensorPin = A0;

void setup_wifi() {
  // Connect to Wi-Fi
}

void reconnect() {
  // Reconnect to MQTT broker
}

void setup() {
  Serial.begin(115200);

  // Connect to Wi-Fi and MQTT broker
  setup_wifi();
  client.setServer(mqtt_server, 1883);
```

```
    // Initialize gas sensor
    pinMode(gasSensorPin, INPUT);
}

void loop() {
    if (!client.connected()) {
        reconnect();
    }
    client.loop();

    // Read gas sensor data
    int gasValue = analogRead(gasSensorPin);

    // Publish data to MQTT topic
    char msg[10];
    sprintf(msg, "%d", gasValue);
    client.publish("air_quality", msg);

    delay(10000); // Adjust the delay based on your
monitoring frequency
}
```

Output:

1.Sensor Readings: The gas sensor reads the concentration of the target gas (e.g., CO, NO2) and converts it to an analog value.

2.MQTT Data Transmission: The ESP32 publishes the gas concentration data to the MQTT topic "air_quality" with a specified delay.

Setting up the MQTT Broker:

You'll need an MQTT broker to handle the communication between the ESP32 and any clients that want to receive the air quality data. You can use an existing broker service or set up your own using software like Mosquitto.

Remember to replace placeholders (YOUR_WIFI_SSID, YOUR_WIFI_PASSWORD, MQTT_BROKER_IP) with your specific information.

For the server-side code, you'll need to set up an MQTT broker and a subscriber to receive the data published by the ESP32.

Please note that this is a simplified example. For a complete and production-ready system, you may

need to implement additional features like error handling, data validation, and security measures. Additionally, you may want to integrate a database for storing historical data and a web interface for visualization.

AIR QUALITY MONITORING USING RASPBERRY PI AND PYTHON

INTRODUCTION

Air is the very stuff we breathe. It's about 78% nitrogen, 21% oxygen, and 1% argon, and then there's the assorted 'other' bits and pieces – many of which have been spewed out by humans and our related machinery.

Carbon dioxide is obviously an important polluter for climate change, but there are other bits we should be concerned about for our health, including particulate matter.

Creating a full-fledged IoT project with complete source code and output is beyond the scope of a

single response. However, I can guide you through the process and provide a simplified example code for a basic air quality monitoring system using a Raspberry Pi and a PM2.5 sensor.

Components Needed

- Raspberry Pi
- PM2.5 sensor (e.g., SDS011)
- Jumper wires
- Internet connection

Getting started

We picked the SDS011 sensor for our project (see 'Picking a sensor' below for details on why). This sends output via a binary data format on a serial port.

You can read this serial connection directly if you're using a controller with a UART, but the sensors also usually come with a USB-to-serial connector, allowing you to plug it into any modern computer and read the data.

First, you'll need a Raspberry Pi (any version) that's

CODE

```python
import serial
import time

ser = serial.Serial('/dev/ttyUSB0')    # Adjust port if necessary

def read_sensor_data():
    while True:
        data = ser.read(10)
        if data[0] == 66 and data[1] == 77:
            pm25 = (data[3] * 256 + data[2]) / 10
            pm10 = (data[5] * 256 + data[4]) / 10
            return pm25, pm10

if __name__ == '__main__':
    try:
        while True:
            pm25, pm10 = read_sensor_data()
```

```
                print(f'PM2.5: {pm25} µg/m³, PM10: {pm10} µg/m³')
                time.sleep(10)  # Adjust interval as needed

        except KeyboardInterrupt:
            ser.close()
```

Run the Script

In the terminal, navigate to the directory containing your Python script and run it:

```
python air_quality_monitor.py
```

Output

The script will continuously read data from the PM2.5 sensor and print the PM2.5 and PM10 readings in micrograms per cubic meter (µg/m³) every 10 seconds.

leading to a wellgrounded system. Datasheets available on the dashboard of IBM Bluemix account will help in framing good policies against the increasing level of pollution to ensure healthful environment.

Air quality monitoring system can be more advantageous if pollutants like Sulfur dioxide, nitrogen dioxide, ground level ozone etc. are also monitored.

Introduction to Cloud

Definition:

Cloud computing refers to the delivery of computing services, including servers, storage, databases, networking, analytics, and software, over the internet. Instead of owning and maintaining physical hardware and software, users can access these resources through cloud providers on a pay-as-you-go basis.

Key Concepts:

1.On-Demand Availability:

- Cloud resources are available to users on-demand. They can be provisioned and de-provisioned quickly, allowing for flexibility and scalability.

2.Resource Pooling:

- Cloud providers maintain a large pool of computing resources that are shared among multiple users. Resources are dynamically allocated and reassigned based on demand.

3.Elasticity:

- Cloud services can be scaled up or down based on workload requirements. This allows businesses to easily adapt to changing needs without significant upfront investment.

4.Self-Service:

- Users can provision and manage resources through a web-based dashboard or API without the need for human intervention from the service

clouds, allowing data and applications to be shared between them. Provides greater flexibility and optimization of existing infrastructure.

4.Multi-Cloud:

- Involves using services from multiple cloud providers. Offers redundancy, cost optimization, and the ability to choose the best services from different providers.

Cloud computing has revolutionized the way businesses and individuals access and manage their computing resources, enabling greater flexibility, efficiency, and cost-effectiveness.
It plays a crucial role in modern IT infrastructure and continues to drive innovation in technology.

IoT Based Air Quality Monitoring with Node Red Services with ESP8266

In this project we are going to make an IoT Based Air Pollution/Quality Monitoring System with ESP8266, PM2.5 Particulate Matter Sensor, MQ-135 Air Quality Sensor & BME280 Barometric Pressure Sensor.

Abstract

This paper present an implementation of MQTT based air quality monitoring system. The air quality measurement device is a hardware using ESP8266 NodeMCU that connects to sensors for measuring temperature, humidity, concentration of carbon monoxide (CO), ozone gas (O3), and PM2.5.

The firmware of device makes the device act as a publisher that reads the sensor data and sends them to MQTT Broker. Node-RED is used to be a subscriber that subscribes to receive data from MQTT Broker. With Node-RED, we can easily make a flow to manage and handle received data.

Then, Node-RED will send data to the air quality monitoring dashboard which is a responsive web

Components Name

1 NodeMCU ESP8266 Board -1

2 PMS5003 PM2.5/PM10 Sensor -1

3 MQ-135 Air Quality Sensor -1

4 BME280 Sensor -1

5 Connecting Wires -10

6 Breadboard -1

MQ-135 Air Quality Sensor

The MQ-135 gas sensor senses the gases like ammonia nitrogen, oxygen, alcohols, aromatic compounds, sulfide and smoke.

BME280 Barometric Pressure Sensor

Bosch BME280 Humidity, Temperature & Pressure Sensor is an integrated environmental sensor which is very small-sized with low power consumption.This BME280 Atmospheric Sensor Breakout is the easy way to measure barometric pressure, humidity, and temperature readings all without taking up too much space.

CODE

```cpp
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

const char* ssid = "your_SSID";
const char* password = "your_PASSWORD";
const char* mqttServer = "your_MQTT_broker_IP";
const int mqttPort = 1883;
const char* topic = "air_quality";

WiFiClient espClient;
PubSubClient client(espClient);

void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
```

```
    client.setServer(mqttServer, mqttPort);
}

void loop() {
    if (!client.connected()) {
        reconnect();
    }

    int gasValue = readGasSensor(); // Replace with
your actual sensor reading code
    client.publish(topic, String(gasValue).c_str());
    delay(10000); // Adjust as per your requirements
}

void reconnect() {
    while (!client.connected()) {
        Serial.println("Connecting to MQTT...");
        if (client.connect("NodeMCU_Client")) {
            Serial.println("Connected to MQTT");
        } else {
            delay(5000);
        }
```

```
  }
}

int readGasSensor() {
  // Code to read gas data from the sensor
  // Replace this with your actual sensor reading code
}
```

Creating Node-RED Flow

- In Node-RED, create a flow that subscribes to the MQTT topic where the NodeMCU is publishing data. Process the data as needed (e.g., store it in a database, visualize it, etc.).

Data Visualization

- Use Node-RED dashboard nodes or other visualization nodes to display the air quality data.

Output:

- When you run the NodeMCU code, it will

connect to the WiFi network, read gas data from the MQ-135 sensor, and publish it to the MQTT broker. Node-RED will receive this data and process it according to your flow, displaying it in the chosen visualization.

Remember, this is a simplified example. Depending on your specific sensor and requirements, you'll need to adjust the code accordingly.

For a complete project with all the details, including source code and output, I recommend looking for detailed tutorials or guides online, or considering a more in-depth course or resource on IoT projects with Node-RED and the MQ-135 gas sensor.

MOBILE APPLICATION DEVELOPMENT USING MIT APP INVENTOR

Abstract

In present times, it is apparent that weather andclimatic circumstances have grown variable and unpredictable,not just in our country but also in other parts of the world,potentially wreaking havoc on agricultural productivity.

ESP8266 board.Write the Arduino code to read data from the sensor and send it to a server (e.g., ThingSpeak).

Here's a simplified Arduino code snippet:

CODE

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>

const char* ssid = "YOUR_SSID";
const char* password = "YOUR_PASSWORD";
const String apiKey = "YOUR_API_KEY";
const String server = "api.thingspeak.com";

void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
```

```
}

void loop() {
  // Read sensor data (PM2.5, CO2, etc.)
  int pm25Value = getPM25Value(); // Function to get PM2.5 value

  // Send data to ThingSpeak
  HTTPClient http;
  http.begin("http://" + server + "/update?api_key=" + apiKey + "&field1=" + String(pm25Value));
  int httpCode = http.GET();
  http.end();

  delay(30000); // Send data every 30 seconds (adjust as needed)
}
```

Setting up ThingSpeak

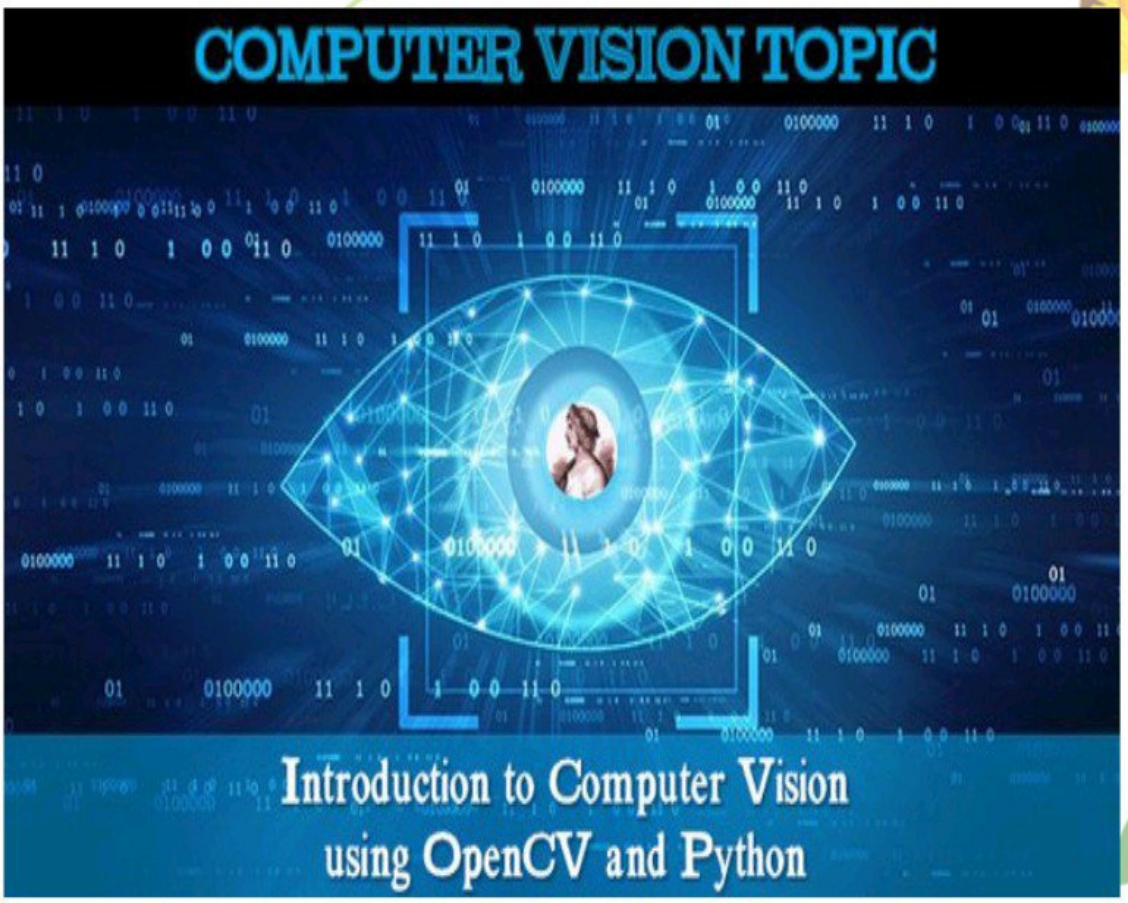
Create an account on ThingSpeak (if you haven't already).Create a new channel and set up a field to store PM2.5 data.

Different sensor functions and working procedureswere discussed in the proposed architecture.

A few additional sensors will be added in the future to sense wind speed and direction, and the prototype will be deployed onto an agricultural field to monitor weather parameters, transforming it into a full fledged Precision Weather Station that may act as an important element of the system.

As a result, we'll be able to see the problem early on and take steps to protect the next generation.

Introduction to Computer Vision using OpenCV and Python



In this guide, we will introduce a brief overview of Deep Learning.

Then, we will discuss the purpose of Computer Vision in Python. After that, we' ll be taught the basics of dealing with data using OpenCV libraries by creating and displaying images.

The fundamental tasks of Computer Vision such as object recognition and semantic segmentation will be explained.

We will also cover the process of feature extraction,

edge and face detection and object classification.

Prerequisites

Before starting this guide, it is essential to be familiar with the basics of Python programming and Image Processing concepts.

Guide map

We will provide a structured content according to the following map:

1.Introduction
2.A brief introduction to Deep Learning
3.Computer vision tasks
4.Computer Vision Systems
5.Python libraries for Computer Vision
6.OpenCV library on Windows and Ubuntu
7.Processing images with OpenCV
8.Use cases for Computer Vision
9.Conclusion.

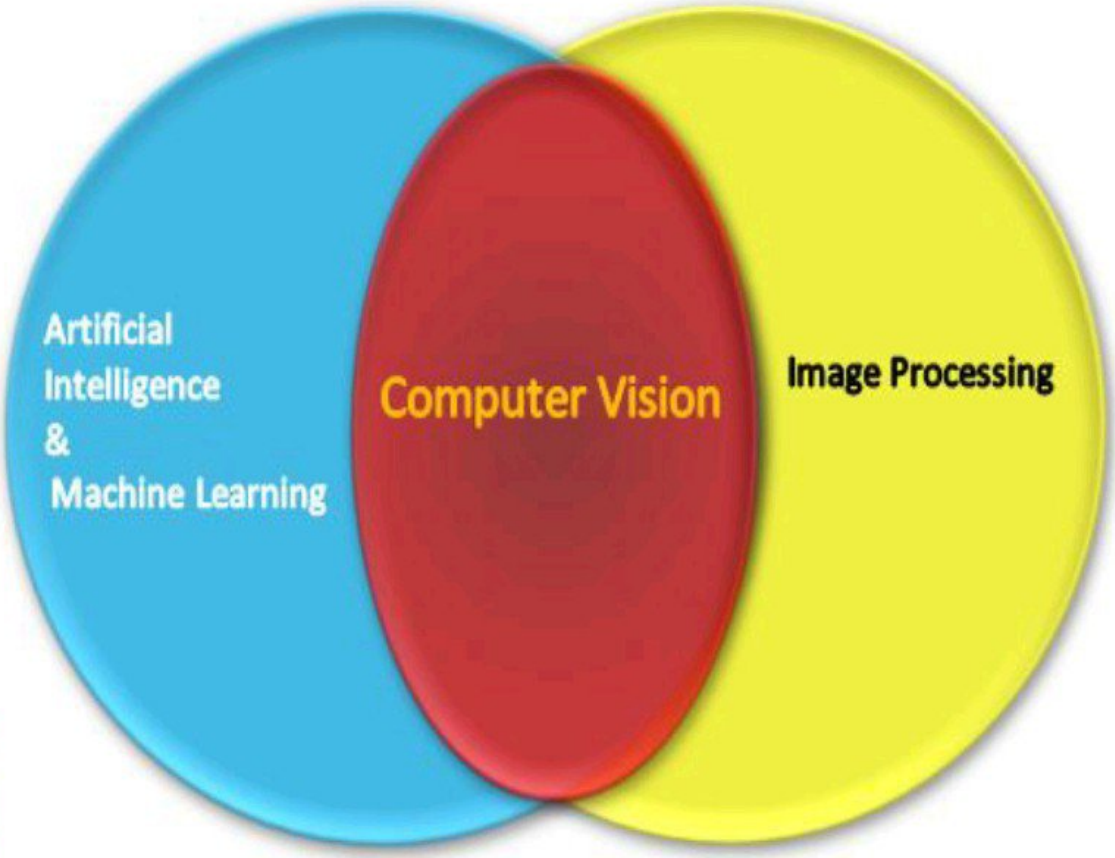
Introduction:

Computer Vision is a branch of Computer Science,

which aims to build up intelligent systems that can understand the content in images as they are perceived by humans.

The data may be presented in different modalities such as sequential (video) images from multiple sensors (cameras) or multidimensional data from a biomedical camera, and so on.

It is the discipline that integrates the methods of acquiring, processing, analyzing and understanding large-scale images from the real world.



Some examples of Computer Vision applications:

-Any application that can recognize objects or humans in an image;

-Automatic control applications (industrial robots, vehicles);

-Object construction models (industrial inspection, medical image analysis);

-Applications make it possible to track a moving object.

Useful books for learning various aspects of Computer Vision: Multiple View Geometry in Computer Vision (opens new window), Computer Vision: Algorithms and Applications(opens new window)

An introduction to Deep Learning

Deep Learning is an Machine Learning strategy that has greatly enhanced performance in many fields

such as Computer Vision, Speech Recognition, Machine Tanslation, and so on.

The use of deep learning techniques, through raw data, allows many challenges to be solved in many economic sectors such as health, transport, finance, etc.

The favourable conditions that allowed the rise of Deep Learning:

-Availability of very large spatio-temporal datasets (Big Data);

-Availability of high-performance computing (GPU);

-Flexibility of new training models (Deep Neural Networks).

Computer vision tasks:

In this section, we will successively examine some tasks of Computer Vision, in particular Image Recognition, Semantic Segmentation, Image Retrieval, Image Restoration, Object Recognition, Video Tracking, and so on.

The only way to solve this issue is to find the best solutions to match certain features (edges, shapes, etc), and in some cases only, often with specific lighting conditions, a background and a certain position for the camera.

Types of recognition:

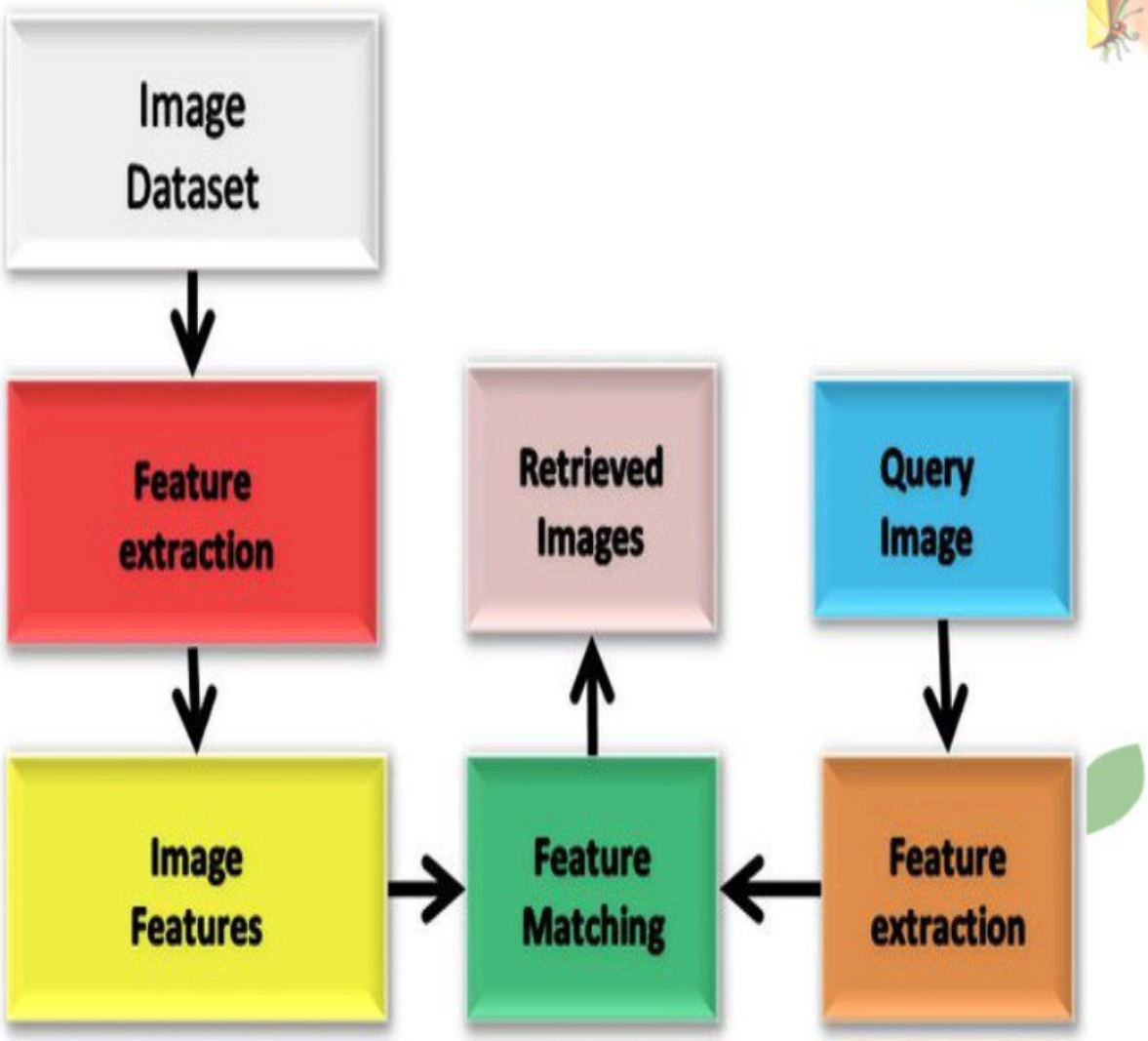
A - Identification: Predefined objects are often identified from different viewpoints of the camera in their different locations.

B - Selection: Define a unique identifier in the shape. For example: identify a person's face or identify the specific type of a person or car.

C - Examination: Image data is treated for a specific object. For example: check for the presence of diseased cells in medical form, check if a car is present on a highway.

Fast MPN-COV: here (opens new window);

Fine-Grained Representation Learning and



You will find here some projects and scripts based on Image Retrieval tasks using a Deep Learning paradigm:

-Deep Local Feature (DeLF): here (opens new window)

-MILDNet: here (opens new window)

detection of a particular object in an image or video. Humans can recognize many objects in images with little effort, although the image may differ slightly from different aspects, such as variations, or even when they are moved or rotated.

Although humans can recognize objects when they are partially hidden, this task remains a challenge for computer vision systems.

Unlike Machine Learning, the Deep Learning paradigm consists of an end-to-end feature representation learning from raw data without any prior data processing steps.

You will find here some projects and scripts based on Object Recognition task using a Deep Learning paradigm:

-TF-slim: here (opens new window)
-DenseNet: here (opens new window)
-DeepBeliefSDK: here (opens new window)

Semantic Segmentation

Semantic segmentation is a Deep Learning

confirm that its coordinate system is correct.

2.Reduce the interference to ensure that the sensor does not provide inaccurate information.

3.Increase the variance in order to ensure that the required information will be available.

Features extraction

Visual data features are extracted at different levels of abstraction from data raw. These benchmarks are categorized into:

1.Global features such as color and shape.

2.Local features such as edges and points. More complex features related to colors and patterns can be obtained.

Segmentation

All zones of the image can be recognized as important locations for subsequent operations.

For example: select a set of key points, divide one or more images that contain the region of interest. 3.5.

High-level processing operations At this stage, the input data consists of a small set of data, such as a set of points or a portion of the image that is suspected to contain the interest object. The other operations are:

1.Ensure that the collected data are consistent with the hypotheses of the intended application.

2.Evaluate the transaction values assigned to the request, such as steering or shape size.

3.Classify the recognized objects into several classes

Python libraries for Computer Vision

The main toolkits for image processing in python are OpenCV, scikit-image and Pillow. The most general Python libraries (Numpy and

operations on images are purely mathematical operations.

But we can't say that programmers will do all these operations every time they use images, hence the development of OpenCV library, which includes functions that perform the most necessary operations in the images.

CODE

```
import cv2

def detect_smoke(image_path):
    # Load the image
    image = cv2.imread(image_path)

    # Convert the image to grayscale
    gray = cv2.cvtColor(image,
cv2.COLOR_BGR2GRAY)
```

```
    # Apply a Gaussian blur to reduce
noise
    blurred = cv2.GaussianBlur(gray, (21,
21), 0)


    # Detect edges using Canny edge
detection
    edges = cv2.Canny(blurred, 50, 150)


    # Find contours in the edged image
    contours,                    _
=
cv2.findContours(edges.copy(),
cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)


    # Loop over the contours
    for contour in contours:
        # Approximate the contour
        approx                       =
```

```
cv2.approxPolyDP(contour,       0.02
cv2.arcLength(contour, True), True)


            # If the contour has 4 vertices, it
is likely a rectangular object (smoke)
            if len(approx) == 4:
                cv2.drawContours(image,
[approx], 0, (0, 255, 0), 5)


        # Display the output image
        cv2.imshow("Output", image)
        cv2.waitKey(0)


# Example usage
detect_smoke("smoke_image.jpg")
```

Conclusions

In this guide, we discussed the topic of Computer Vision using OpenCV and Python. We presented some fundamental tasks of

Computer Vision such as Object Recognition and Semantic Segmentation. We also examined some case studies about the process of edge and face detection, feature extraction and object classification.

THANK YOU