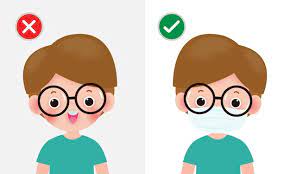
**Mask Detection using Face Detection**

**Introduction:**

A cutting-edge solution created to alleviate the problems caused by the COVID-19 epidemic and increase public health and safety is the "Mask Detection Using Face Detection" initiative. To automatically determine if people are wearing face masks in real time, this project uses computer vision and artificial intelligence capabilities. A critical preventive measure to stop the transmission of respiratory diseases, especially airborne viruses like COVID-19, is to wear a face mask. Monitoring and guaranteeing mask compliance in various public settings, including airports, hospitals, businesses, and retail establishments, can be challenging. Traditional manual monitoring techniques can be time-consuming, ineffective, and dangerous for the persons involved. This project's core competency is detecting human faces in image or video streams. Contemporary face identification algorithms use deep learning methods to recognize facial features and draw bounding boxes around them. This crucial step ensures that only human faces will be analysed after that, increasing the research's effectiveness. The technology uses a classification algorithm to identify faces and assess whether or not a person is wearing a mask. This entails using a dataset including photographs of people wearing masks and those without masks to train a machine-learning model. Based on visual characteristics, the model can distinguish between the two categories. This project aims to deliver a user-friendly, dependable, and in-the-moment solution for tracking mask compliance. Public places and businesses can use this technology to enforce mask-wearing regulations by adding cameras or utilizing current surveillance systems. When mask non-compliance is discovered, suitable measures can be taken, such as alerting the proper parties or staff. It saves time and resources by reducing the need for manual monitoring and intervention. Uses cutting-edge computer vision methods for accurate face and mask classification. Providing real-time information about mask compliance, real-time monitoring enables prompt interventions. Encouraging the use of masks helps with the more significant effort to stop the spread of contagious diseases. For seamless installation, it may be integrated with the existing surveillance infrastructure. A cutting-edge solution to COVID-19-related problems that also improves public health and safety is the "Mask Detection Using Face Detection" program. It uses AI and computer vision to identify the use of a mask in real time automatically. To prevent respiratory disorders like COVID-19, mask use is essential. There are difficulties in ensuring mask compliance in public places, including hospitals, workplaces, and airports. Traditional monitoring techniques are cumbersome and dangerous. The research uses cutting-edge deep learning-based face recognition algorithms to detect human faces in photos and videos. This makes it possible to analyze human faces effectively. The system uses a classification algorithm to detect and rate mask wearers. Based on visual characteristics, a machine learning model that was trained on mask and non-mask photos can distinguish between the two. The project intends to provide a dependable, user-friendly real-time solution for mask compliance enforcement. Public locations can enforce mask laws and take action on non-compliance by installing cameras or using existing surveillance. This combines cutting-edge computer vision to accurately classify faces and masks while reducing the need for manual monitoring and conserving resources. Real-time monitoring offers prompt interventions in line with more significant initiatives to prevent disease. A seamless setup is guaranteed through integration with current surveillance systems.

**Story:**

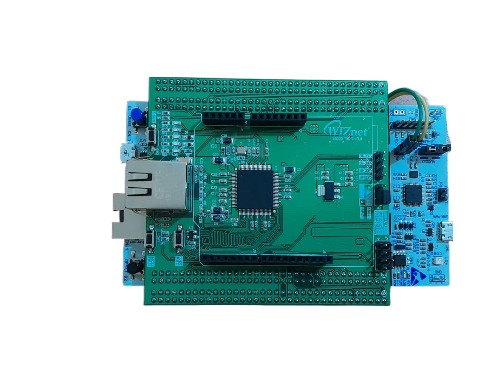
Many people utilize the Internet of Things (IoT) for remote monitoring without human interaction. The topic of mask monitoring in busy institutions is addressed in this study. It tracks entry and directs activities using IoT devices and face recognition driven by AI. During the screening process, the integrated LED blinks as a warning if a person without a mask is discovered. Masks are typically worn to protect others from the virus-containing respiratory droplets of the wearer. By capturing droplets and restricting their release during talking, coughing, or breathing, they also provide minimal protection to the wearer. Since most COVID-19 transmission occurs through these droplets, the chance of other people inhaling the virus decreases. Before symptoms (asymptomatic) or right before signs (pre-symptomatic), carriers might transfer the virus. Masks stop transmission that is unaware. By reducing pathogen propagation, community-wide mask use promotes "herd immunity." This protects people who cannot wear masks and weaker demographics, such as the elderly or people with health concerns. Masks are essential but function best when combined with other practices, including avoiding close contact, washing hands, and immunization. Together, these actions stop the transmission of viruses. Advocate distance and mask use in crowded public spaces, especially in high-transmission locations.



**Things used in this project.**

**Hardware components:**

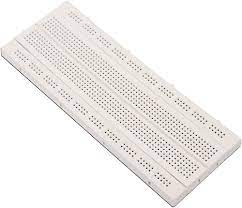
* W5300 TOE Shield with Nucleo-144 (F429ZI)



* Buzzer



* Solderless Breadboard



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Software apps and online services:**   * PyCharm * Arduino IDE * Open CV * Tensor Flow * MQTT | | | | | |
|  |  |  |  |  |  |
| **Components:**  **1. Arduino IDE**  The Arduino Integrated Development Environment (IDE) is a piece of software that includes a text editor, menus, a toolbar with standard function buttons, a message box, and a text console. For communication and uploading of programs, it links to the Arduino hardware. A text editor, message area, text console, toolbar, and menus are all features of the Arduino IDE program. For communication and uploading of programs, it links to the Arduino hardware.  **2.MQTT**  MQTT is a lightweight publish/subscribe transport for distant devices with small code and low network traffic. It is an OASIS standard messaging protocol for the Internet of Things. It is extensively used in oil and gas, manufacturing, telecommunications, and automotive sectors.  **3.Buzzer**  A buzzer sensor is a piece of technology that produces auditory alarms. It produces buzzing or beeping noises by converting electrical signals into sound waves using a tiny transducer. The sensor is wired to an electronic circuit, and when it gets a signal from the circuit, the transducer rapidly vibrates to generate sound waves with a particular frequency. Buzzer sensors are available in various sizes and shapes, and depending on the signals they receive, they can generate varying sound patterns and loudness. They are frequently utilized in alarms, doorbells, timers, and other alerting mechanisms in automation, robotics, security, and IoT devices, providing essential audio input for occurrences or circumstances.  **4.Jumper Wires & Breadboard**  To connect the node MCU board and Pico board.  **Block Diagram:**     * Face mask detection using MQTT, a buzzer, and a Wiznet board is a system that identifies if people are wearing masks. * It employs a camera and a face mask detection model to spot mask violations. * When someone without a mask is detected, the system sends an alert message through MQTT to the Wiznet board. * This board connects to the buzzer, triggering an audible alert to notify others of the violation. This setup is effective for ensuring mask compliance in various settings.   **Workflow:**  **1.Facemask detection using open-CV**  The procedure consists of two primary steps: figuring out whether a person is wearing a mask and detecting human faces and mouths in every input video frame. Data to show whether the mask is on or off is created after looking for facemasks in the video.  **2.Ethernet connection to MQTT**  Ethernet libraries link the Pico board to the internet through Ethernet. Once the connection has been made, an MQTT connection is made to use Ethernet to publish the data collected from the face recognition file (facedetection.py). The data is transmitted as a message once the connection has been established.  **3.Buzzer**  The buzzer is stationed to produce an alert which starts from a message in the MQTT that is passed on to the Arduino.  **Circuit Diagram:**     * Positive pin: Connect to digital pin 15 on the Wiznet. * Negative pin: Connect to the ground on the Wiznet.   **Code Explanation:**  **PyCharm**:   |  | | --- | | import cv2  import numpy as np  import paho.mqtt.client as mqtt  from tensorflow.keras.applications.mobilenet\_v2 import preprocess\_input  from tensorflow.keras.models import load\_model |  * cv2: Use the OpenCV library for computer vision applications, such as processing images and videos. * numpy: NumPy package for array-based numerical calculations. * paho. mqtt.client: Paho MQTT library for MQTT communication. * preprocess input: A Keras (a component of TensorFlow) function for image preparation before feeding them to the MobileNetV2 model. * load model: A Keras function for loading a trained MobileNetV2 model.  |  | | --- | | model = load\_model("model.h5") |  * The algorithm loads a MobileNetV2 model that has already been trained to detect masks. The "model.h5" file for the model needs to be saved in the same directory.  |  | | --- | | mqtt\_broker = "10.21.70.16"  mqtt\_port = 1883  mqtt\_topic = "mask\_alert" |  * The code details the MQTT broker that it will use to transmit notifications. It establishes the broker's port (1883) and address ("broker.emqx.io"). * Additionally, it designates the MQTT subject ("mask\_alert") for message publication.  |  | | --- | | # MQTT client setup  mqtt\_client = mqtt.Client()  def on\_connect(client, userdata, flags, rc):  print("Connected with result code " + str(rc))  client.subscribe(mqtt\_topic)  def on\_message(client, userdata, msg):  print(msg.topic + " " + str(msg.payload))  mqtt\_client.on\_connect = on\_connect  mqtt\_client.on\_message = on\_message |  * Using the Paho MQTT library, the code sets up a MQTT client. On\_connect and on\_message callback routines are set up. * on\_connect: This function is invoked when the client connects to the MQTT broker. It subscribes to the above-mentioned topic. * on\_message: When the client receives a message, this function is invoked. Now, it merely prints the message's topic and payload.  |  | | --- | | mqtt\_client.connect(mqtt\_broker, mqtt\_port, 60) |  * The code links the MQTT client to the broker that was previously mentioned.  |  | | --- | | video\_capture = cv2.VideoCapture(0) |  * To access the laptop camera, the code initializes the video capture object.  |  | | --- | | face\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_frontalface\_default.xml') |  * The code loads the Haar Cascade classifier for face detection. The OpenCV data directory should contain the classifier file.  |  | | --- | | while True:  # Capture video from laptop camera  ret, frame = video\_capture.read()  # Convert the frame to grayscale for face detection  gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  # Detect faces in the frame  faces = face\_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30)) |   The script enters an infinite loop to continually take frames from the camera, perform face detection, and categorize faces for mask detection.  For each frame captured, it converts the frame to grayscale for face detection.   |  | | --- | | # Iterate through detected faces  for (x, y, w, h) in faces:  face\_roi = frame[y:y + h, x:x + w]  face\_input = cv2.resize(face\_roi, (224, 224))  face\_input = np.expand\_dims(face\_input, axis=0)  face\_input = preprocess\_input(face\_input)  # Perform mask detection using the pre-trained model  predictions = model.predict(face\_input)  mask\_probability = predictions[0][0]  no\_mask\_probability = predictions[0][1]  # Draw rectangles around the detected faces  cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)  # If the probability of not wearing a mask is higher than wearing a mask, send an alert to MQTT server  if no\_mask\_probability > mask\_probability:  mqtt\_client.publish(mqtt\_topic, "Mask not detected!") |  * The face region is cropped, resized to (224, 224) (as required by MobileNetV2), and pre-processed for feeding into the model for each detected face. * The presence or absence of a mask on a face is determined using the pre-trained MobileNetV2 model.  |  | | --- | | # Display the frame with detected faces  cv2.imshow('Mask Detection', frame)  # Exit the loop when 'q' is pressed  if cv2.waitKey(1000) & 0xFF == ord('q'):  break |  * It sends an alert message to the MQTT server if the likelihood of not wearing a mask is greater than the likelihood of donning one. * To highlight the area of interest, rectangles are additionally drawn around the recognized faces on the frame. * The processed frame with identified faces is shown in an OpenCV window titled "Mask Detection." * Until the user taps the 'q' key, the loop continues.  |  | | --- | | # Release the video capture and close the window  video\_capture.release()  cv2.destroyAllWindows() |  * The video capture object is released after the loop stops (pressing "q"), and the OpenCV window is closed.   **Code Explanation**:  **Arduino:**   |  | | --- | | #include <Ethernet.h>  #include <Wire.h>  #include <PubSubClient.h>  #include <ArduinoJson.h>  #include "HardwareSerial.h" |  * These lines import the required libraries for the project, including Ethernet communication, I2C communication, MQTT communication, JSON parsing, and hardware serial communication.  |  | | --- | | const char\* device\_unique\_id = "MaskDetection\_" \_\_DATE\_\_ "\_" \_\_TIME\_\_; |  * This line creates a unique identifier for the device using a combination of "MaskDetection," the date, and the time of compilation.  |  | | --- | | #define SERIAL\_OUTPUT  #ifdef SERIAL\_OUTPUT  #define PRINT(x) Serial.print(x)  #define PRINTLN(x) Serial.println(x)  #else  #define PRINT(x)  #define PRINTLN(x)  #endif |  * These macros allow conditional printing to the serial monitor based on whether SERIAL\_OUTPUT is defined. This helps in debugging and monitoring the device's behavior.  |  | | --- | | byte mac[] = { 0xDE, 0xAD, 0xBE, 38, 0x77, 0xF7 };  IPAddress ip(192, 168, 0, 77); |  * These lines define the MAC address and IP address for the Ethernet connection. The IP address can be configured using DHCP or statically assigned based on network availability.  |  | | --- | | IPAddress mqtt\_server(10, 21, 70, 16); | | | | | | |
|  |  |  |  |  |  |

* The IP address of the MQTT broker server is specified, allowing the device to connect to the broker for MQTT communication.

|  |
| --- |
| const char\* mqtt\_topic = "mask\_alert";  EthernetClient mqttClient;  PubSubClient mqtt\_client(mqttClient); |

* These lines define the MQTT broker's IP address and the topic the device subscribes to. An instance of the PubSubClient class is created for MQTT communication.

|  |
| --- |
| void callback(char\* topic, byte\* payload, unsigned int length) {    String message = "";    for (int i = 0; i < length; i++) {      message += (char)payload[i];    }    Serial.println(message);    if (message.equals("Mask not detected!")) {      Serial.println("Activating buzzer.");      digitalWrite(buzzerPin, HIGH);      delay(1000);      digitalWrite(buzzerPin, LOW);      Serial.println("Buzzer deactivated.");    }  } |

* The callback function processes incoming MQTT messages. If the message is "Mask not detected!", the buzzer is activated for a second.

|  |
| --- |
| void reconnect() {    // Loop until we're reconnected    while (!mqtt\_client.connected()) {      PRINT("MQTT: Attempting MQTT connection...");      // Attempt to connect      if (mqtt\_client.connect(device\_unique\_id)) {        PRINTLN("MQTT: connected");        // ... and resubscribe        mqtt\_client.subscribe(mqtt\_topic);      } else {        PRINT("MQTT: failed, rc=");        PRINT(mqtt\_client.state());        PRINTLN(" MQTT: try again in 5 seconds");        // Wait 5 seconds before retrying        delay(2000);      }    }  } |

* The reconnect function attempts to establish an MQTT connection. It keeps trying until successful and handles re-subscribing to the MQTT topic.

|  |
| --- |
| void setup() {      // Open serial communications and wait for port to open:    Serial3.setRx(PC11);    Serial3.setTx(PC10);    delay(50);      // Open serial communications and wait for port to open:    Serial.begin(9600);      while (!Serial) {      ; // wait for serial port to connect. Needed for native USB port only    }    pinMode(buzzerPin, OUTPUT);     // start the Ethernet connection:    PRINTLN("Initialize Ethernet with DHCP:");    if (Ethernet.begin(mac) == 0) {      PRINTLN("Failed to configure Ethernet using DHCP");      // Check for Ethernet hardware present      if (Ethernet.hardwareStatus() == EthernetNoHardware) {        PRINTLN("Ethernet shield was not found.  Sorry, can't run without hardware. :(");        while (true) {          delay(1); // do nothing, no point running without Ethernet hardware        }      }      if (Ethernet.linkStatus() == LinkOFF) {        PRINTLN("Ethernet cable is not connected.");      }      // try to congifure using IP address instead of DHCP:      Ethernet.begin(mac, ip, myDns);    } else {      PRINT("  DHCP assigned IP ");      PRINTLN(Ethernet.localIP());    }      // give the Ethernet shield a second to initialize:    delay(1000);    mqtt\_client.setServer(mqtt\_server, 1883);    mqtt\_client.setCallback(callback);    // give the Ethernet shield a second to initialize:    delay(1000);  } |

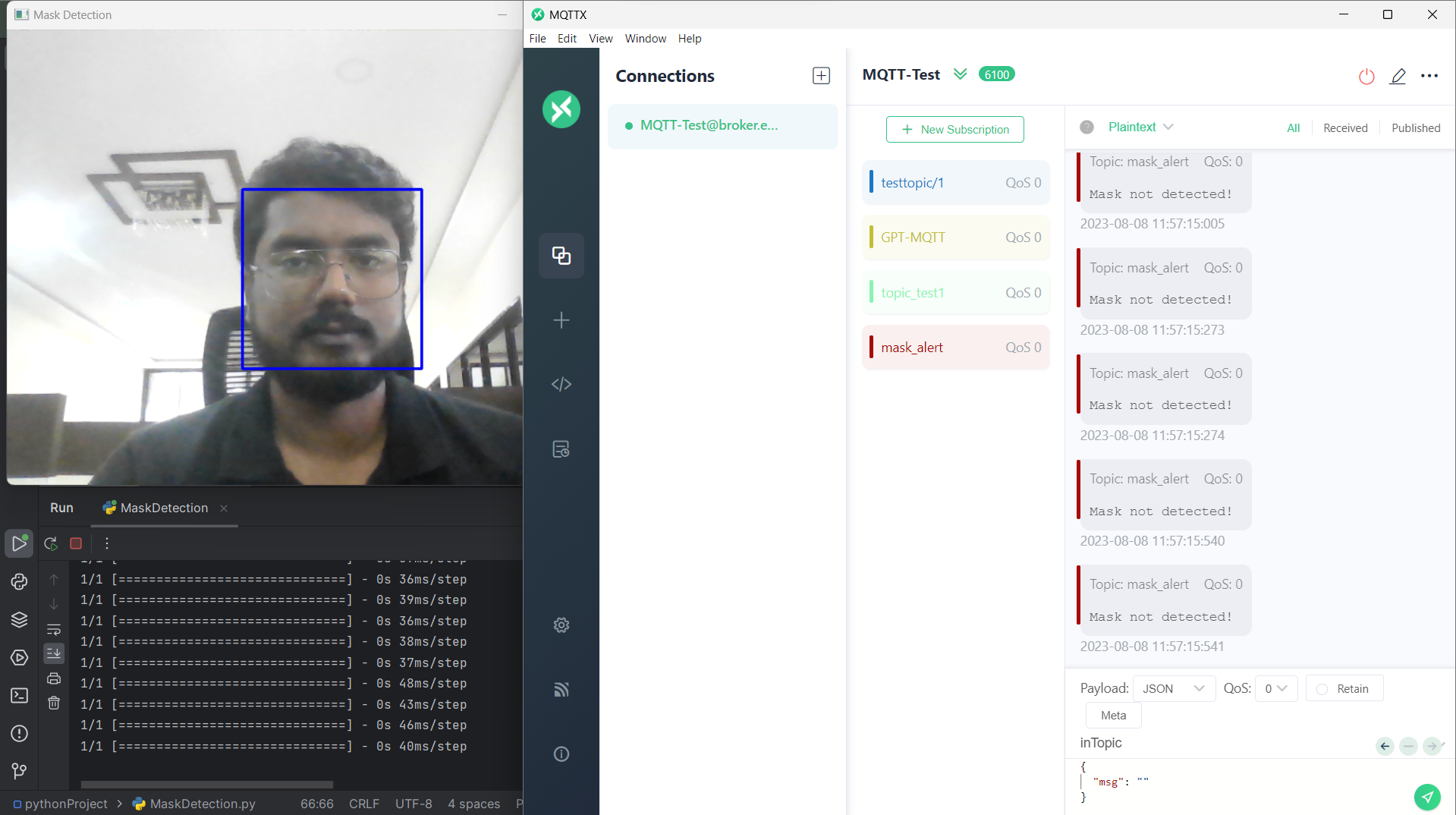
* In the setup function, serial communication is set up, the buzzer pin is configured as an output, Ethernet connection is initialized, and MQTT client is configured and initialized.

|  |
| --- |
| void loop() {    if (!mqtt\_client.connected()) {      reconnect();    }    mqtt\_client.loop();  } |

* The loop function checks and handles MQTT connection status. If not connected, it calls the reconnect function, and it also handles MQTT message reception.

**Output:**

The output when no mask is detected. It includes Buzzer alert too.



The output when mask is detected.

