OUTLINE OF THE CONTRIBUTION

Abstract—---------- Preethi Billa

Introduction—---------- Satyadivya Maddipudi

Related Work—---------- Satyadivya Maddipudi

Components Used—---------- Satyadivya Maddipudi

Project Architecture—---------- Preethi Billa

Camera Web Server Setup—-- Preethi Billa

Face Recognition Setup—----- Niranjan Reddy Masapeta

Results—---------- Niranjan Reddy Masapeta

Conclusion—---------- Niranjan Reddy Masapeta

References—---------- Teamwork

Face detection and recognition using ESP32, Python and OpenCV

Niranjan Reddy Masapeta[#1], Satya Divya Maddipudi[#2] , Preethi Billa[#3]

[#]*Computer Engineering Department*
San Jose State University
San Jose, California

[1]niranjanreddy.masapeta@sjsu.edu

[2]preethi.billa@sjsu.edu

[3]satyadivya.maddipudi@sjsu.edu

***Abstract*— Face detection will be implemented utilizing the ESP32 microcontroller, Python programming language, and OpenCV library in this project. Face detection is a critical component of many computer vision applications, such as security systems, smart homes, and self-driving cars. In this project, we will take photos using the ESP32 camera module, then analyze them with Python and OpenCV to recognize faces. The collected images will be sent to the computer to be processed through a Wi-Fi network via the ESP32. Setting up the ESP32 and camera module, programming the ESP32 to relay data via Wi-Fi, and developing face detection algorithms in Python using OpenCV are all part of the project. The ultimate result will be a system capable of identifying and recognizing faces in real-time, proving the capability of low-cost, embedded devices to execute complicated computer vision tasks.**

Keywords*— *Facial recognition, ESP32 Sensor, Face detection, OpenCV

I. INTRODUCTION

The Internet of Things (IoT) is a fast-developing topic that entails connecting objects and sensors to the internet and enabling those to interact with one another and with different systems. IoT has several uses, including healthcare, agriculture, transportation, and smart homes. With the development of low-cost embedded systems like the ESP32 microcontroller, it is now simpler than ever to create IoT devices capable of performing complicated tasks. Face identification is a critical job in computer vision that has several practical applications, such as security systems, video surveillance, and human-computer interaction. With the introduction of low-cost embedded systems like the ESP32 microcontroller, complicated computer vision initiatives can now be done on small, low-power devices. Face detection will be done in the following endeavor utilizing the ESP32 microcontroller, Python programming language, and OpenCV library. The ESP32 is a low-cost, low-power microcontroller with built-in Wi-Fi as well as Bluetooth connection that's perfect for IoT and embedded-technology projects. The ESP32 camera module, which is readily interfaced with the ESP32, offers a low-cost and small option for image and video capture. In the computer vision field, OpenCV and Python are commonly used to perform complicated image processing and machine learning methods. OpenCV is an open-source library that supports a variety of image and video processing tasks, including face detection. In this project, we will collect images using the ESP32 camera module, which will then be transferred to the computer through a Wi-Fi network for processing. The Python script running on the computer will process the photos and recognize faces using OpenCV. Faces observed in the image. These are then highlighted or tagged and transmitted back to the ESP32 for display on a connected screen or another output device. Setting up the ESP32 and camera module, programming the ESP32 to relay data via Wi-Fi, and developing face detection algorithms in Python using OpenCV are all part of the project. The ultimate result will be a system capable of detecting and recognizing faces in real time, proving the power of low-cost,

embedded equipment to execute complicated computer vision tasks.

II. RELATED WORK

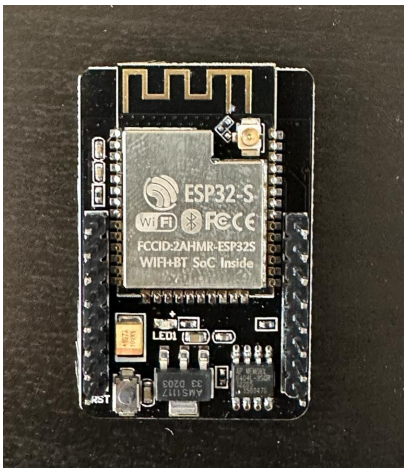
Several papers on face detection utilizing various technologies and platforms have been published. However, relatively few researchers have investigated the use of the ESP32 microcontroller for facial detection. In recent work, a previous study suggested a face identification system using the ESP32 and the TensorFlow Lite library. The system captured photos using a camera module linked to the ESP32, and the TensorFlow Lite software library was used to recognize faces in real-time. The study, however, concentrated on the accuracy of face identification rather than the implementation specifics. In separate research, another study suggested a face recognition system based on the ESP32 and a pre-trained deep learning model. The system captured photos with a camera module linked to the ESP32, which were then analyzed with a pre-trained model to distinguish faces. However, the study did not delve into the usage of OpenCV or the specifics of face detection implementation. Our suggested project, on the other hand, tries to develop facial detection utilizing an ESP32 sensor, Python, and OpenCV. The use of OpenCV enables a large range of methods for image processing, allowing complicated face identification algorithms to be implemented on the ESP32. Furthermore, our study intends to investigate the technical specifics of face detection by applying the ESP32 and Python, making it available to a wider spectrum of developers interested in low-cost, embedded systems.

III. COMPONENTS USED

1. HARDWARE COMPONENTS

1.1. ESP32 CAM

The ESP32 Camera is a development board that combines a powerful microcontroller with a high-resolution camera, making it an ideal platform for developing face recognition systems. While traditional face recognition systems rely on AI and deep learning algorithms to detect and identify faces, there are other techniques that can be used without AI.



One approach to face recognition without AI is to use basic image processing techniques such as edge detection, thresholding, and contour analysis to extract facial features from an image. These features can then be compared to a database of pre-registered faces to determine if there is a match.
To implement this approach using the ESP32 camera, developers can use the OpenCV library, which provides a range of image processing functions that can be used to extract facial features. For example, the Haar Cascade classifier can be used to detect the presence of faces in an image, while the Eigenfaces or Fisherfaces algorithms can be used to extract facial features.

Once the facial features have been extracted, they can be compared to a database of pre-registered faces using techniques such as Euclidean distance or cosine similarity. If a match is found, the system can then perform the desired action, such as unlocking a door or granting access to a system.

While this approach to face recognition is less accurate than using AI-based techniques, it can still be effective for basic applications. It is also much

simpler and less resource-intensive than using deep learning algorithms, which can be a significant advantage for certain applications.

In conclusion, the ESP32 Camera can be used for face recognition without AI by using basic image processing techniques to extract facial features and comparing them to a database of pre-registered faces. While this approach may not be as accurate as AI-based techniques, it can still be effective for basic applications that require simple and efficient face recognition systems.

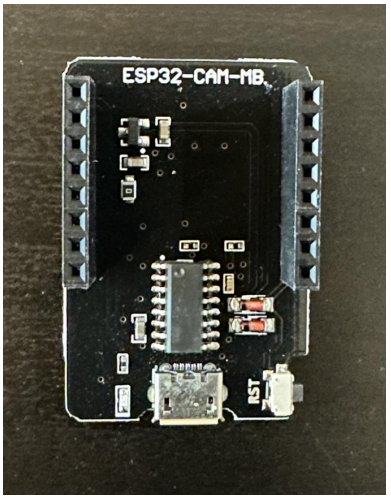
1.2. USB-TTL CONVERTER

A USB TTL converter is a device that can be used to establish a serial communication link between a microcontroller and a computer. In the context of face recognition without AI detection, a USB TTL converter can be used to interface with a microcontroller that is programmed to perform basic face recognition tasks.

The process of face recognition without AI detection typically involves using image processing techniques to extract facial features from an image and then comparing those features to a database of pre-registered faces. To perform these tasks, a microcontroller can be programmed to process the image and extract the relevant features.

The USB TTL converter can be used to connect the microcontroller to a computer, allowing developers to upload code to the microcontroller and monitor its output. The converter typically has a USB interface on one end and a set of pins on the other end that can be connected to the microcontroller.

To perform face recognition using a USB TTL converter, developers can connect the ESP32 Camera to the microcontroller and program it to capture an image of a face. The microcontroller can then process the image using image processing techniques and extract the relevant facial features.



Once the facial features have been extracted, the microcontroller can compare them to a database of pre-registered faces. If a match is found, the system can perform the desired action, such as unlocking a door or granting access to a system.

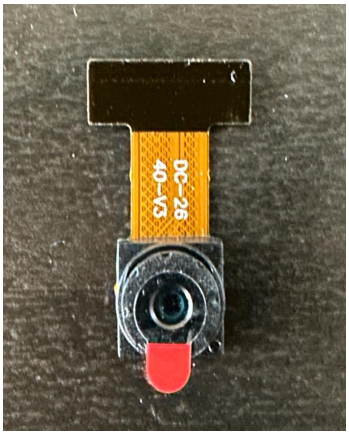
Using a USB TTL converter in this way can provide a simple and efficient way to implement basic face recognition without AI detection. It is also relatively low-cost compared to using AI-based techniques, which can be a significant advantage for certain applications.

In conclusion, a USB TTL converter can be used to interface with a microcontroller that is programmed to perform basic face recognition tasks. By using image processing techniques to extract facial features and comparing them to a database of

pre-registered faces, a USB TTL converter can provide a simple and efficient way to implement face recognition without AI detection.

1.3. OV2640

The OV2640 camera sensor is a popular choice for face recognition systems that don't require AI detection. It captures high-quality images that can be used to extract facial features using image processing techniques like OpenCV. This allows for the development of low-cost, efficient, and effective face recognition systems. The extracted facial features can be compared to a database of pre-registered faces using techniques like Euclidean distance or cosine similarity, allowing the system to perform the desired action, such as granting access or unlocking a door.



The OV2640 camera sensor can be connected to a microcontroller like an ESP32 or an Arduino to create a cost-effective face recognition system. This system can be used in a wide range of applications, including security systems, attendance tracking systems, and access control systems. With its high-quality image capture capabilities and compatibility with various microcontrollers, the OV2640 camera sensor provides a simple and reliable solution for basic face recognition tasks without the need for complex AI algorithms.

2. SOFTWARE COMPONENTS

2.1. PYTHON

Python is a popular programming language that can be used for a wide range of applications, including face recognition without AI detection. Python provides a rich set of libraries and tools for image processing and machine learning that can be used to develop face recognition systems.

The process of face recognition without AI detection typically involves using image processing techniques to extract facial features from an image and then comparing those features to a database of pre-registered faces. To perform these tasks, Python can be used to process the image and extract the relevant features.
Using Python for face recognition without AI detection can provide a simple and efficient way to implement basic face recognition tasks. It also provides the flexibility to customize the system to meet specific requirements and integrate it with other systems.

In conclusion, Python can be used for face recognition without AI detection by using image processing techniques to extract facial features and comparing them to a database of pre-registered faces. The OpenCV library provides a range of functions for image processing that can be used to develop efficient and effective face recognition systems in Python.

2.2. OPENCV

OpenCV is a popular computer vision library that can be used for

face recognition without AI detection. It offers a range of image processing functions that can be used to detect and extract facial features from captured images. These features can then be compared to a database of pre-registered faces to perform the desired action, such as granting access or unlocking a door.

OpenCV can be used in combination with various camera sensors like the OV2640, and microcontrollers like the ESP32 or Arduino to create a low-cost, efficient, and effective face recognition system. With its robust feature detection capabilities and compatibility with various hardware components, OpenCV provides a reliable solution for basic face recognition tasks without the need for complex AI algorithms. It can be used in various applications, including attendance tracking systems, security systems, and access control systems, providing a simple yet powerful tool for face recognition.

2.3. ARDUINO IDE

The Arduino Integrated Development Environment (IDE) is a popular software tool used to program microcontrollers like the Arduino boards. It can be used in conjunction with camera sensors like the OV2640 to develop low-cost face recognition systems without the need for AI detection. The IDE offers a range of libraries and functions that can be used to control the camera, capture and process images, and perform facial feature extraction.

With the Arduino IDE, developers can easily program the microcontroller to control the camera and perform face recognition tasks using OpenCV libraries. The captured images can be processed to extract facial features, which can be compared to a pre-registered database to perform the desired action. This provides a simple and cost-effective solution for basic face recognition tasks in applications like attendance tracking, access control, and security systems. Overall, The Arduino IDE offers a user-friendly platform for developing low-cost face recognition systems without the need for advanced AI algorithms.

2.4. ESP32CAM LIBRARY

The ESP32CAM library is a powerful tool that can be used to control the OV2640 camera module and perform image processing tasks on the ESP32 microcontroller. It can be used to develop face recognition systems without the need for AI detection. The library provides functions for capturing images, processing them using OpenCV libraries, and performing facial feature extraction.

With the ESP32CAM library, developers can create cost-effective face recognition systems for various applications like attendance tracking, access control, and security systems. The captured images can be processed to extract facial features, which can then be compared to a pre-registered database to perform the desired action. The ESP32CAM library provides a simple and efficient platform for developing face recognition systems without the need for complex AI algorithms.

IV. PROJECT ARCHITECTURE

The ESP32 cam module is connected to a computer and programmed via FTDI programmer. The circuit diagram of the connections are given below. The ESP32 board is

powered from VCC and GND pins to FTDI's 5V and GND pins respectively. TX (transfer) and RX (receive) pins from FTDI are connected to UOR and UOT pins of ESP32 board. To upload and flash the program, GPIO0 and GND pins of ESP32 board are connected with a jumper as seen from the schematic diagram below.

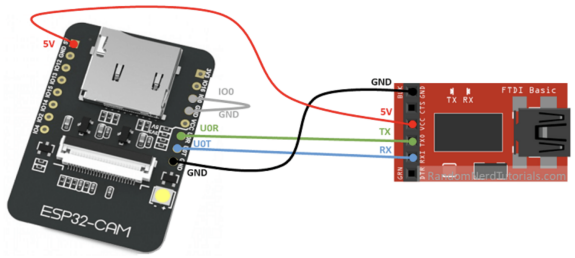
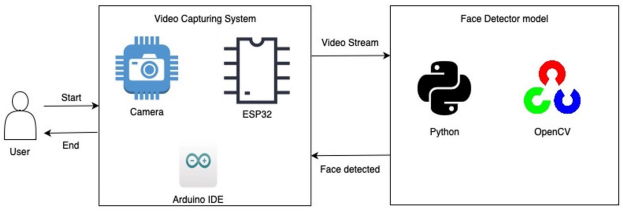
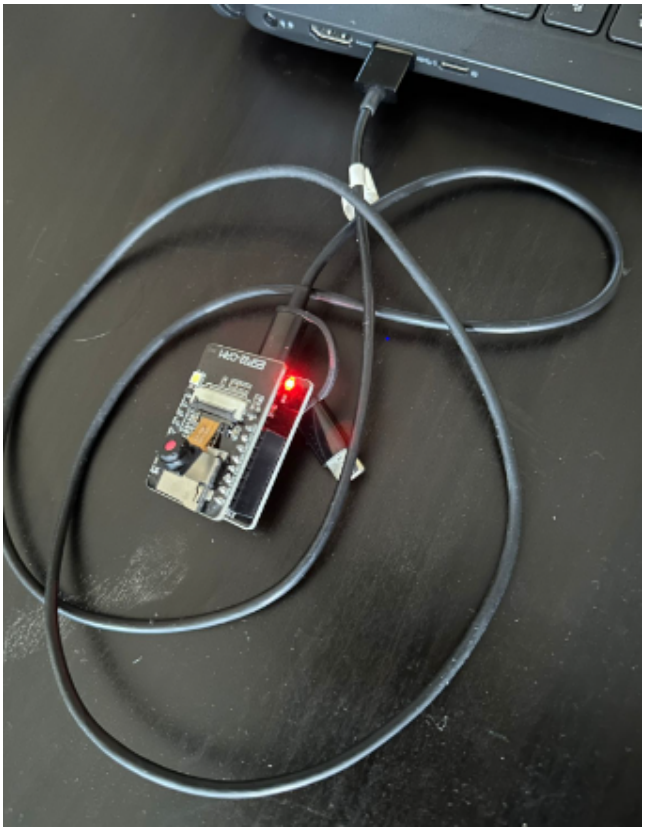


Image Source: randomnerdtutorials.com

The project architecture is given in the diagram below. Our project mainly consists of two components: hardware and software. On the hardware side, we have our main ESP32 microcontroller connected to an OV2640 2 MP camera. The ESP32 board is connected and programmed from a computer using a USB to TTL converter as described in the above image. ESP32 is programmed to connect to the camera web server using Arduino IDE. First step in our project is to set up this hardware and make sure that the camera is connected and we can stream successfully from the camera.



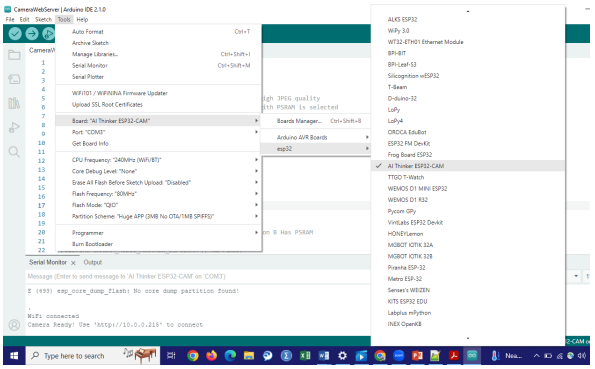


On the software side, to implement face recognition and face detection algorithms, we use python and opencv libraries to set up the face recognition pipeline. We use the camera stream from the ESP32 module to identify and detect faces from the stream to test out the detection model. We then train a face recogniser model from collected face data to recognize the faces. After training the model, we will use that model to identify the person in the stream and display the same.

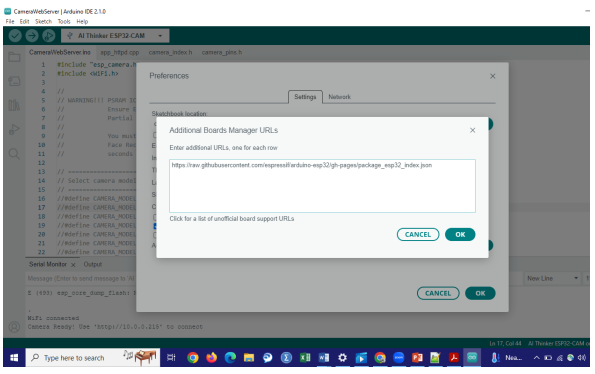
V. CAMERA WEB SERVER SETUP

The ESP32-CAM module is a popular and versatile camera module for various IoT applications, with a powerful ESP32 microcontroller that enables tasks such as image capture, video streaming, and QR code scanning. Below are the guidelines to be followed to successfully integrate and set up the ESP32-CAM module with Arduino IDE version 1.8.11. With careful attention to the steps outlined, you can configure your settings, upload a sample sketch, and test the functionality of the ESP32-CAM module, enabling you to scan QR codes on a streaming web server with ease.

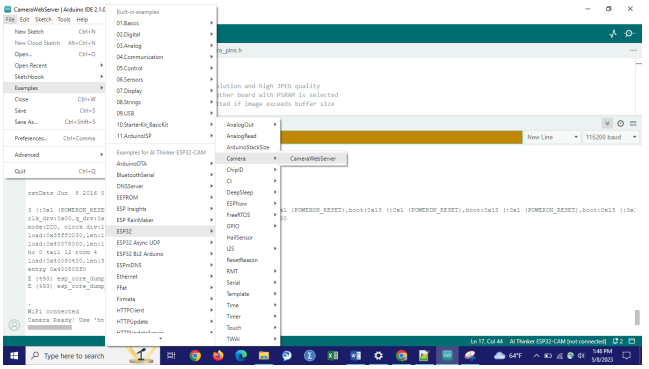
1. To properly integrate the ESP32-CAM with the Arduino IDE 1.8.11, the installation of the ESP32 board definition is necessary. Start by launching the Arduino IDE and then select 'File' and choose 'Preferences.' Here, you will find the 'Additional Boards Manager URLs' field where you should enter the following URL: https://dl.espressif.com/dl/package_esp32_index.json.



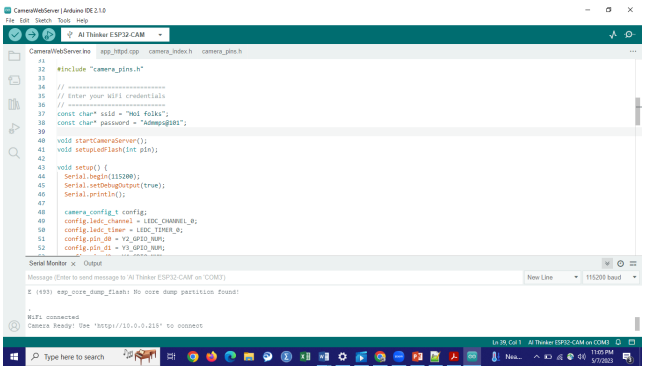
2. After entering the URL, click 'OK' to close the preferences window. Then, go to 'Tools,' select 'Board,' and choose 'Boards Manager.' Use the search bar to look up 'ESP32' and install the package named 'esp32' by Espressif Systems.



3. In the Arduino IDE. Navigate to File->Examples->ESP32->Camera->CameraWebServer. Then open the CameraWebServer file and modify the wifi Credentials before the code is uploaded.



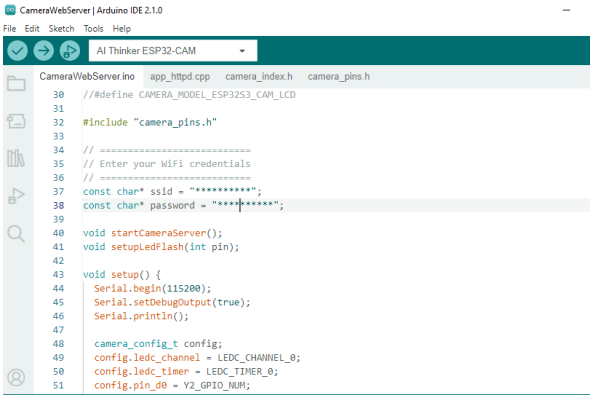
4. Configure the Arduino IDE for the ESP32-CAM



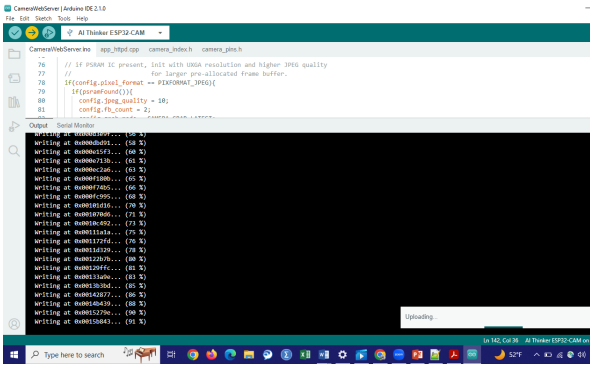
- Choose the appropriate board and COM port : For the ESP32-CAM, you can select either 'AI Thinker ESP32-CAM' or 'ESP32 Wrover Module' from the available board options. Once you have selected the board, you should choose the correct COM port for your FTDI programmer or USB-to-Serial converter under 'Tools' > 'Port.'

- Adjust the Flash Frequency, Flash Mode, and Partition Scheme settings: The recommended settings for the ESP32-CAM are '80 MHz' for Flash Frequency, 'QIO' for Flash Mode, and 'Huge APP (3MB No OTA)' for Partition Scheme.
- Modify the Wi-Fi credentials in the code: Before uploading the sketch, you should modify the Wi-Fi credentials in the code. You can update the two lines of code with your Wi-Fi credentials to

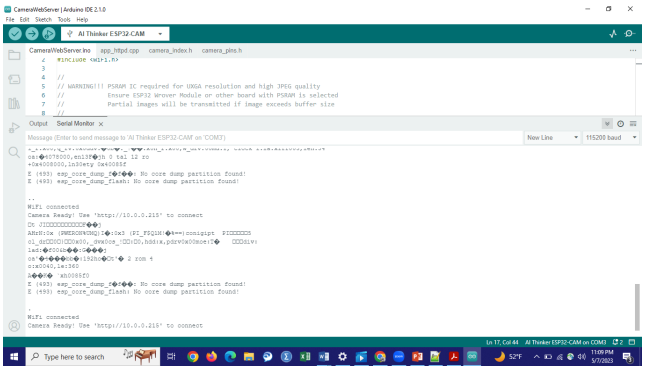
connect the ESP32-CAM to your network and ensure that the sketch is uploaded successfully.



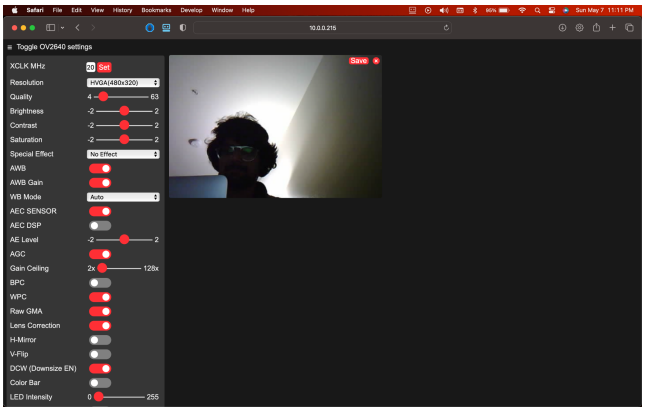
5. To verify that the ESP32-CAM is functioning correctly, you can upload a sample sketch to it. There are numerous sample sketches available online, including the 'CameraWebServer' example provided in the 'esp32' board package. To upload the sketch, first, open it in the Arduino IDE. Then, click the 'Upload' button (or use the keyboard shortcut 'Ctrl' + 'U') to compile and upload the sketch to the ESP32-CAM.
After the upload is complete, disconnect the GPIO0 pin from GND to exit flashing mode. This will enable the ESP32-CAM to execute the newly uploaded sketch.



6. Testing of the camera module : Once you have uploaded the sketch, it's time to test the ESP32-CAM module. Start by opening the Arduino IDE's Serial Monitor under 'Tools' > 'Serial Monitor' and setting the baud rate. If the sketch has been successfully uploaded, the ESP32-CAM should automatically connect to your Wi-Fi network. To view the IP address of the module, open the Serial Monitor in the Arduino IDE. You can see that you get an ip address to access your remote camera server. You can input the ip address within any browser of any device connected in the same network to access the camera stream



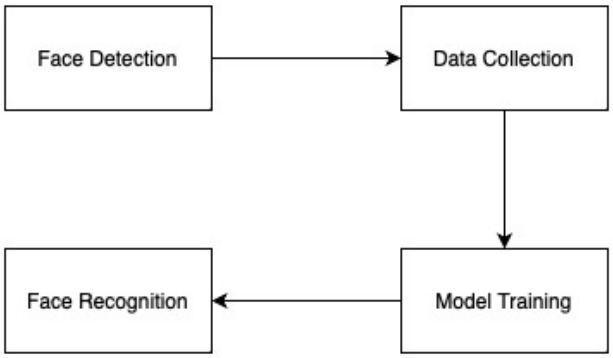
7. Access camera stream from ESP32: Enter the ip address from serial monitor in any browser window to access the camera stream. Here you can change the resolution of the stream and also control brightness, saturation, contrast and other settings of the stream. The tool also gives an inbuilt face detection and face enrollment modules as part of their system. But in this project, we will develop our own robust facial recognition pipeline.



VI. FACE RECOGNITION SETUP

After setting up the ESP32 camera to stream the video, our next step in the project is to build a face recognition model pipeline to identify and

detect people from the camera stream. The pipeline is given below and consists of four components: Face Detection, Data Collection, Model training, and Face Recognition.



Prerequisites: Install all required python dependencies with python package installer (pip). Our required packages for this project are opencv, numpy, and pillow. Any missing module can be installed if you find any module not found error. Create a working directory where we create required project files.

A. Face Detection

1. Create a python file in your working directory with some name like FaceDetection.py.

2. Import required modules in this case opencv (cv2) and numpy. Load a face detector model. We are using the haar cascade xml model from OpenCV builtin library.

```
import numpy as np
import cv2
faceCascade = cv2.CascadeClassifier(cv2.data.haarcascades+'haarcascade_frontalface_default.xml')
```

3. Capture the video stream from ESP32 cam using the ip address provided in the serial monitor. Streams can be captured by opencv's built-in function VideoCapture.

```
URL = "http://10.0.0.215"
cap = cv2.VideoCapture(URL + ":81/stream")
cap.set(3, 640)
cap.set(4, 480)
```

4. Capture the frames from the video stream using a loop. For each frame, we pass it through the faceCascade classifier to detect faces in the image with required parameters.

```
while True:
    ret, img = cap.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.2,
        minNeighbors=5,
        minSize=(20, 20)
    )
```

5. For faces detected in the stream, the next step is to draw a boundary around the face and display it back to the stream. For faces found, we get four parameters to draw a rectangle, (x,y) as a left corner position, h and w as height and width respectively. We create an ROI (Region of Interest) using these parameters and return it back using imshow() function.

```
    for (x,y,w,h) in faces:
        cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = img[y:y+h, x:x+w]
    cv2.imshow('video',img)
    k = cv2.waitKey(30) & 0xff
    if k == 27: # press 'ESC' to quit
        break
```

B. Face Data Collection

In this component, we collect face data to train our face recognition model.

1. Create a python file in the same working directory with name like FaceDataCollection.py
2. Import required models such as cv2 and os (os to save files to directory).
3. Capture the video stream from ESP32 cam using the ip address provided in the serial monitor. Streams can be captured by opencv's built-in function VideoCapture.

```
URL = "http://10.0.0.215"
cam = cv2.VideoCapture(URL + ":81/stream")
cam.set(3, 640) # set video width
cam.set(4, 480) # set video height
face_detector = cv2.CascadeClassifier(cv2.data.haarcascades+'haarcascade_frontalface_default.xml')
```

4. Ask the user to input user id to maintain different ids for different people.

```
face_id = input('\n enter user id end press <return> ==>  ')
print("\n [INFO] Initializing face capture. Look the camera and wait ...")
```

5. Capture the frames from the video stream using a loop. For each frame, we pass it through the faceCascade classifier to detect faces in the image with required parameters.

```
while(True):
    ret, img = cam.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_detector.detectMultiScale(gray, 1.3, 5)
```

6. For each image captured, write that file to a folder called dataset created initially by the user.

```
    for (x,y,w,h) in faces:
        cv2.rectangle(img, (x,y), (x+w,y+h), (255,0,0), 2)
        count += 1
        cv2.imwrite("dataset/User." + str(face_id) + '.' +
                    str(count) + ".jpg", gray[y:y+h,x:x+w])
        cv2.imshow('image', img)
```

7. Once we capture 30 images, we break out from the loop and end the program.

```
    k = cv2.waitKey(100) & 0xff
    if k == 27:
        break
    elif count >= 30:
        break

print("\n [INFO] Exiting Program and cleanup stuff")
```

8. To collect data for multiple people, we have to run the program for each person to collect face data.

C. Face Data Trainer

This component describes the process of training a face recognition model based on the data collected from the previous component.

1. Create a file named FaceTrainer.py to build a face recognition model.
2. Import required python modules such as cv2, numpy and pillow. Pillow is used for dealing with image files in python.

```
import cv2
import numpy as np
from PIL import Image
import os
```

3. Load the haar cascade detector model to detect faces in the image as in component Face Detector. For training the model, we are using LBPH (Local Binary Patterns Histograms) face recognizer which is included in the opencv package.

```
# Path for face image database
path = 'dataset'
recognizer = cv2.face.LBPHFaceRecognizer_create()
detector = cv2.CascadeClassifier(cv2.data.haarcascades+"haarcascade_frontalf
```

4. The function getImagesandLabels() gets the training images stored in the dataset folder with respective ids.

```
def getImagesAndLabels(path):
    imagePaths = [os.path.join(path,f) for f in os.listdir(path)]
    faceSamples=[]
    ids = []
```

5. After loading the images, using pillow's Image library and numpy library, we get the ids and respective face boundaries stored in faces and ids arrays respectively using the haar cascade model.

```
    for imagePath in imagePaths:
        if imagePath == path + '/.DS_Store':
            continue
        PIL_img = Image.open(imagePath).convert('L') # grayscale
        img_numpy = np.array(PIL_img,'uint8')
        id = int(os.path.split(imagePath)[-1].split(".")[1])
        faces = detector.detectMultiScale(img_numpy)
        for (x,y,w,h) in faces:
            faceSamples.append(img_numpy[y:y+h,x:x+w])
            ids.append(id)
    return faceSamples,ids
print ("\n [INFO] Training faces. It will take a few seconds. Wait ...")
faces,ids = getImagesAndLabels(path)
```

6. Using the recognizer model created in step no 2, we generate a trainer.yml model by passing faces and ids as input parameters to the model.

```
recognizer.train(faces, np.array(ids))
# Save the model into trainer/trainer.yml
recognizer.write('trainer/trainer.yml')
# Print the numer of faces trained and end program
print("\n [INFO] {0} faces trained. Exiting Program".format(len(np.unique(ids))))
```

Each time we collect new data in component 2, we need to FaceTrainer.py again to record new people in the system. This is a very important step to remember in the pipeline.

D. Face Recognizer

After creating a trainer.yml model, we can now use this model to recognize faces in the stream.

1. Create a file named FaceRecognizer.py to use the trainer model to identify faces.
2. Import required modules such as cv2, numpy and os.

3. Load the face detector, recognizer functions used in step 3 to detect and recognize faces in images. Load the recognizer function from the trainer.

```
recognizer = cv2.face.LBPHFaceRecognizer_create()
recognizer.read('trainer/trainer.yml')
cascadePath = "haarcascade_frontalface_default.xml"
faceCascade = cv2.CascadeClassifier(cv2.data.haarcascades+cascadePath);
```

4. Capture the video stream from ESP32 cam using the ip address provided in the serial monitor. Streams can be captured by opencv's built-in function VideoCapture.

```
URL = "http://10.0.0.215"
cam = cv2.VideoCapture(URL + ":81/stream")
cam.set(3, 640) # set video widht
cam.set(4, 480) # set video height
```

5. Map the ids you have provided in component 2 to the respective users to display their names while recognizing.

```
id = 0
# names related to ids: example ==> Marcelo: id=1,  etc
names = ['Neeraj']
```

6. Capture the frames from the video stream using a loop. For each frame, we pass it through the faceCascade classifier to detect faces in the image with required parameters.

```
while True:
    ret, img =cam.read()
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor = 1.2,
        minNeighbors = 5,
        minSize = (int(minW), int(minH)),
       )
```

7. Using recognizer.predict() function, we will identify the face of the user and predict the owner of the face.

```
for(x,y,w,h) in faces:
    cv2.rectangle(img, (x,y), (x+w,y+h), (0,255,0), 2)
    id, confidence = recognizer.predict(gray[y:y+h,x:x+w])

    # If confidence is less them 100 ==> "0" : perfect match
    if (confidence < 100):
        id = names[id]
        confidence = "  {0}%".format(round(100 - confidence))
    else:
        id = "unknown"
        confidence = "  {0}%".format(round(100 - confidence))
```

8. We return the image with a text placed on the box indicating the owner of that image back to the stream.

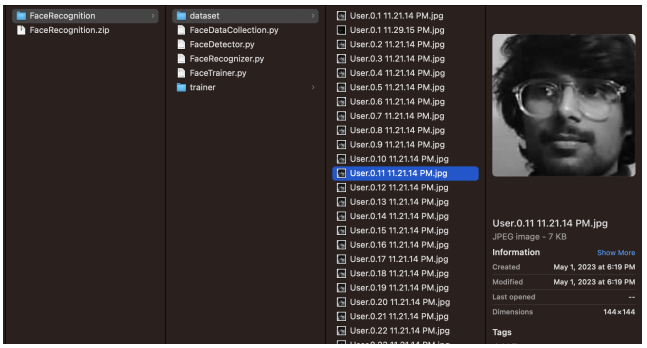
```
        cv2.putText(
                    img,
                    str(id),
                    (x+5,y-5),
                    font,
                    1,
                    (255,255,255),
                    2
                   )
        cv2.putText(
                    img,
                    str(confidence),
                    (x+5,y+h-5),
                    font,
                    1,
                    (255,255,0),
                    1
                   )

    cv2.imshow('camera',img)
```

VII. RESULTS

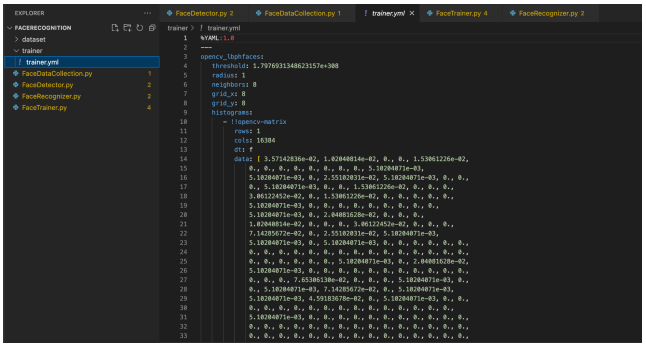
After the successful loading of data collected, Run the python file FaceDataCollection.py. Below screenshot indicates the process of the python run.

```
(base) neerajmssj@Niranjans-MacBook-Air FaceRecognition % python FaceDataCollection.py

enter user id end press <return> ==>  0

[INFO] Initializing face capture. Look the camera and wait ...
```

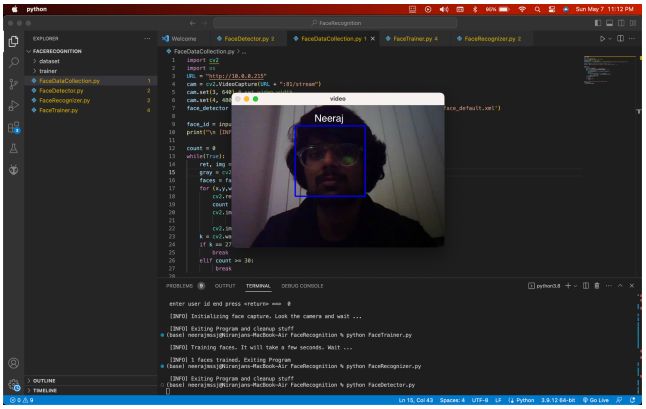
When we run the file , the camera gets started and initializes the face capture. We need to look at the camera and wait. It then captures the 30 images at a time as per the value set by us. All these images get stored in the dataset folder.
The screenshot below shows the images stored and the file location.



Now run the python file FaceTrainer.py. After the file run is completed, it generates the trainer.yml file.



We use the trainer.yml file which has all the data of faces, it detects the face of the person in front of the camera and identifies from the data stored , it then gives the name of the person.



VIII. CONCLUSION

To summarize, face detection and recognition using low-cost embedded systems like ESP32, Python, and OpenCV is a powerful application that can perform complicated computer vision tasks. By interfacing ESP32 with the camera module and programming it to transmit data via Wi-Fi, images can be collected and processed on a computer using Python and OpenCV.

OpenCV, a widely-used open-source library, supports face detection and recognition, enabling the system to detect and recognize faces in real-time. The practical uses of this technology include security systems, video surveillance, and human-computer interaction, highlighting the potential of low-cost embedded equipment for complicated computer vision tasks.

In conclusion, this project showcases the capabilities of low-cost embedded systems, like ESP32 microcontrollers, and their potential for performing complex tasks such as face detection and recognition. With the aid of Python and OpenCV, this technology offers real-time face recognition, which has practical applications in various fields. With the continuous advancement of technology, the possibilities for IoT and embedded technology projects are endless, leading to more innovative applications in the future.

IX. REFERENCES

1. M. Eswar, N. V. Rao, P. Dhanush, M. M. Chowdary and V. H. Deepthi, "Real Time AI Based Face Mask Detector," *2022 International Conference on Emerging Trends in Engineering and Medical Sciences (ICETEMS)*, Nagpur, India, 2022, pp. 206-210, doi: 10.1109/ICETEMS56252.2022.10093259
2. S. Band, R. Javeri, V. Kale, A. Morope and S. Rudrawar, "Military Surveillance System Based on IOT," *2022 Fourth International Conference on Emerging Research in Electronics, Computer Science and Technology (ICERECT)*, Mandya, India, 2022, pp. 1-6, doi: 10.1109/ICERECT56837.2022.10059921.
3. N. N. Mohammed, Z. A. Mohammed and A. N. Faraj, "Line-Following Service Robot Using Arduino with Face Recognition for Offices," *2021 6th International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*, Oita, Japan, 2021, pp. 116-119, doi: 10.1109/ICIIBMS52876.2021.9651634.
4. M. Khan, S. Chakraborty, R. Astya and S. Khepra, "Face Detection and Recognition Using OpenCV," *2019 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, Greater Noida, India, 2019, pp. 116-119, doi: 10.1109/ICCCIS48478.2019.8974493.
5. G. Singh, I. Gupta, J. Singh and N. Kaur, "Face Recognition using Open Source Computer Vision Library (OpenCV) with Python," *2022 10th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, Noida, India, 2022, pp. 1-6, doi: 10.1109/ICRITO56286.2022.9964836.