REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE DE SAÏDA DR MOULAY TAHAR

Faculté de technologie

Département d'électronique

MEMOIRE DE FIN D'ETUDES EN VUE DE L'OBTENTION

DU DIPLOME DE MASTER EN ELECTRONIQUE

OPTION : INSTRUMENTATION

THEME :

SYSTÈME DE PRESENCE BASÉ SUR LA RECONNAISSANCE FACIALE UTILISANT UNE CARTE ESP32-CAM

Présenté par :

ZEROUALI Belkacem El Mahdi Noureddine

CHABANE Chaouch Yahia Samir

Soutenu le 19 juin 2023

Devant le jury composé de :

BOUKHALFA Malika	Maître de conférences à l'Université de Saida	Présidente
BERBER Redouane	Maître de conférences à l'Université de Saida	Examinateur
MAACHOU Fatima	Maître de conférences à l'Université de Saida	Encadrante

Année Universitaire : 2022 - 2023

DEMOCRATIC AND POPULAR ALGERIAN REPUBLIC

MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

UNIVERSITY OF SAIDA DR MOULAY TAHAR

Faculty of Technology

Department of electronics

Final thesis for the attainment of a Master's degree in Electronics

OPTION : INSTRUMENTATION

THEME :

ATTENDANCE SYSTEM BASED ON FACIAL RECOGNITION USING AN ESP32-CAM CARD

Presented by :

ZEROUALI Belkacem El Mahdi Noureddine

CHABANE Chaouch Yahia Samir

Defended on june 28, 2023

BOUKHALFA Malika	Lecturer at the University of Saida	President
BERBER Redouane	Lecturer at the University of Saida	Reviewer
MAACHOU Fatima	Lecturer at the University of Saida	Supervisor

Academic Year: 2022 - 2023

Abstract

This work proposes a facial recognition-based attendance system using the ESP32-Camera module to enhance attendance management in universities. By integrating facial recognition algorithms with the ESP32-Camera module, the system automates identification and attendance tracking of students. It employs face detection, feature extraction, and face recognition techniques to accurately recognize and record attendance. The system offers advantages such as reduced administrative efforts, real-time monitoring, and data-driven decision making. Evaluation of the system's performance demonstrates its reliability and scalability. The proposed system improves attendance management and streamlines administrative processes in universities.
Keywords : facial recognition, ESP32-CAM, Python, OpenCv, Pie Charts, attendance system.

Résumé

Ce travail propose un système de présence basé sur la reconnaissance faciale utilisant le module ESP32-Camera pour améliorer la gestion des présences dans les universités. En intégrant des algorithmes de reconnaissance faciale au module de caméra ESP32, le système automatise l'identification et le suivi des présences des étudiants. Il utilise des techniques de détection de visage, d'extraction de caractéristiques et de reconnaissance faciale pour reconnaître et enregistrer avec précision les présences. Le système offre des avantages tels que des efforts administratifs réduits, une surveillance en temps réel et une prise de décision basée sur les données. L'évaluation des performances du système démontre sa fiabilité et son évolutivité. Le système proposé améliore la gestion des présences et rationalise les processus administratifs dans les universités.
Mots-clés : reconnaissance faciale, ESP32-CAM, Python, OpenCv, Pie Charts, Système de présence.

الملخص :

يقترح هذا العمل نظام حضور قائم على التعرف على الوجه باستخدام وحدة كاميرا ESP 32-CAM لتعزيز إدارة الحضور في الجامعات. من خلال دمج خوارزميات التعرف على الوجه مع وحدة كاميرا ESP 32-CAM ، يقوم النظام بأتمتة تحديد وتتبع الحضور للطلاب. توظف تقنيات اكتشاف الوجه واستخراج الميزات والتعرف على الوجوه للتعرف بدقة على الحضور وتسجيله. يوفر النظام مزايا مثل تقليل الجهود الإدارية ، والمراقبة في الوقت الفعلي ، واتخاذ القرارات القائمة على البيانات. يوضح تقييم أداء النظام موثوقيته وقابليته للتوسع. يعمل النظام المقترح على تحسين إدارة الحضور وتبسيط العمليات الإدارية في الجامعات.

كلمات البحث : التعرف على الوجه ، ESP32-CAM، Python، OpenCv، Pie Charts، نظام الحضور.

Dedication

I dedicate this modest work:

To my dear parents

To my brothers and sisters

To my partner

To my friends

And to anyone who knows me...

- Zerouali Belkacem

Dedication

I dedicate this modest work:

To my mother and father

To my sisters

To my partner

To my friends

And to anyone who knows me or have helped me in anyway...

- CHABANE Chaouch Yahia Samir

Acknowledgements

We would like to express our deepest thanks to our supervisor Mrs. MAACHOU Fatima for her support and her availability throughout this dissertation through the work sessions organized.

We also extend our thanks to Mrs. Azizi Amina, Dr. in English Literature and Linguistics researcher at University of Jordan for her guidance and mentoring through this work.

We thank the members of the jury for their interest in our work. Our thanks also go to all of our teachers who have contributed to our training.

We reserve a special place here to sincerely thank our parents for their affection and their continuous support and to all those who, in one way or another, have helped and encouraged us in the realization of this modest work.

Contents

List of Figures

List of Tables

Abbreviations

IDE	Integrated Development Environment
USB	Universal Serial Bus
LSB	Least Significant Bit
CPU	Central Processing Unit
Wi-Fi	Wireless Fidelity
IoT	Internet of Things
AI-Thinker	Manufacturer of ESP32 modules
VGA	Video Graphics Array
SPI	Serial Peripheral Interface
ESP-IDF	Espressif IoT Development Framework
APIs	Application Programming Interface
PlatformIO	PlatformIO is an open-source ecosystem for IoT development that provides a unified development platform for embedded systems
SoC	System-on-Chip
PCB	Printed Circuit Board
RAM	Random Access Memory
SRAM	Static Random Access Memory
PSRAM	Pseudo-Static RAM
GitHub	GitHub is a web-based platform for version control and collaboration that allows developers to host, review, and manage code repositories

microSD	microSD is a type of removable flash memory card commonly used for storage in portable devices such as smartphones, tablets, and cameras
u.FL	Ultra-Small Connectors, FL is a series name
ULP	Ultra-Low Power
802.11	Wi-Fi: 802.11 b/g/n – 802.11 is a set of IEEE standards for wireless local area networks (WLANs). The b/g/n standards specify the communication protocols and data rates for Wi-Fi networks
BT	Bluetooth: v4.2 BR/EDR and BLE
BR	Bluetooth Basic Rate
EDR	Enhanced Data Rate
BLE	Bluetooth Low Energy
UXGA	Ultra Extended Graphics Array
IEEE 802.11	IEEE 802.11 standard security features all supported
OTP	One-Time Programmable
CHW	Cryptographic Hardware Acceleration
ECC	Elliptic Curve Cryptography
RNG	Random Number Generator
RTC	Real-Time Clock
GPIO	General Purpose Input/Output
VCC	VCC is a term commonly used in electronics to represent the positive power supply voltage
UART	Universal Asynchronous Receiver-Transmitter
ADC	Analog-to-Digital Converter
PWM	Pulse Width Modulation
LED	Light-Emitting Diode

FTDI	FTDI refers to the Family of USB-to-Serial Interface Integrated Circuit chips manufactured by Future Technology Devices International
AI	Artificial Intelligence
ML	Machine Learning
ANN	Artificial Neural Network
OpenCV	Open Source Computer Vision
URL	Uniform Resource Locator
GET	Hypertext Transfer Protocol GET method
HTTP	Hypertext Transfer Protocol
PIL	Python Imaging Library
NumPy	Numerical Python
os	Operating System
Python	No acronym
getcwd()	Get Current Working Directory
chdir()	Change Directory
datetime	No acronym
now()	No acronym
CSV	Comma-Separated Values
Excel	No acronym
SQL	Structured Query Language
pathlib	No acronym
matplotlib	No acronym
C++	C Plus Plus
OS	Operating System
VSCode	Visual Studio Code

Abbreviations

Git	Global Information Tracker
PyInstaller	Python Installer
CMake	Cross-platform Make
GUI	Graphical User Interface
JPEG	Joint Photographic Experts Group
SSD	Solid State Drive
IP	Internet Protocol

Introduction

This thesis introduces a facial recognition-based attendance system for universities, utilizing the ESP32-Camera module. Traditional attendance management methods in universities often suffer from inefficiency and errors. To address these challenges, the proposed system automates attendance tracking using facial recognition technology.

By integrating facial recognition algorithms with the ESP32-Camera module, the system captures and processes images in real-time, enabling automated identification and recording of student attendance. It follows a three-step process: face detection, feature extraction, and face recognition.

The system offers several advantages, including reduced administrative efforts, real-time monitoring, and data-driven decision making. It enhances attendance management by providing accurate attendance records and generating comprehensive reports for academic planning and resource allocation.

Evaluation of the system's performance includes metrics such as recognition accuracy, processing time, and scalability. User feedback and usability testing assess the system's practicality.

In summary, this thesis aims to develop and evaluate a facial recognition-based attendance system using the ESP32-Camera module, providing an efficient solution to improve attendance management in universities.

Chapter 1

The ESP32-CAM

1.1 Introduction

The history of the ESP32 can be traced back to Espressif Systems' earlier microcontroller, the ESP8266, which gained considerable popularity due to its affordable price and built-in Wi-Fi capabilities. The ESP8266 opened up new possibilities for IoT projects, enabling easy connectivity to the internet and remote control of devices. However, as demands for more advanced features and increased processing power grew, Espressif recognized the need to develop a successor that could address these requirements. In 2016, Espressif Systems unveiled the ESP32, building upon the success and lessons learned from the ESP8266. The ESP32 introduced significant enhancements and expanded capabilities compared to its predecessor. It featured a dual-core processor, higher clock speeds, more memory, and a richer set of peripherals. These improvements aimed to provide developers with a more powerful and versatile microcontroller for IoT applications. Since its release, the ESP32 has gained traction in the IoT community and beyond. Its exceptional performance, extensive wireless connectivity options, and comprehensive development ecosystem have made it a favored choice for a wide range of projects. The ESP32 has found applications in smart home automation, industrial monitoring systems, wearable devices, robotics, and much more. Its popularity can be attributed to both its technical capabilities and the active community that has grown around it, continually contributing to its development and expanding its potential applications.[1]

1.2 ESP32-Camera Module

The ESP32-Camera module serves as an indispensable extension board for the ESP32 microcontroller, offering developers the ability to incorporate camera functionality into their IoT projects Figure 1.1. With a rich history of collaboration between Espressif Systems and Ai-Thinker, this module provides an integrated and efficient solution for capturing images and videos. In this article, we will explore the key facts and features of the ESP32-Camera module, highlighting its purpose, camera sensor capabilities, communication and integration methods, development support, and a range of practical applications.[1]

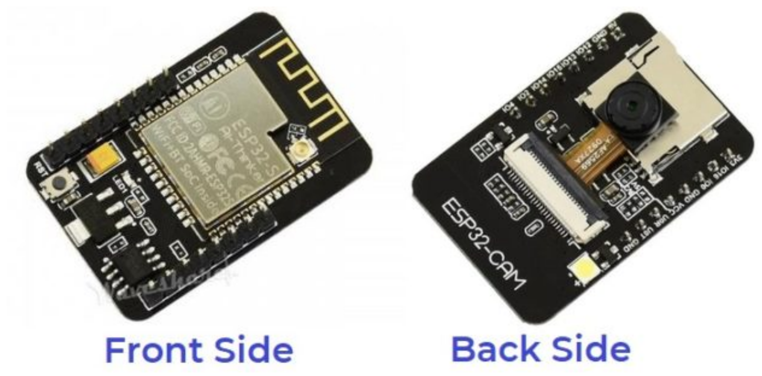


Figure 1.1: The ESP32-CAM

1.2.1 Purpose and Significance

The ESP32-Camera module emerged as a valuable addition to the ESP32 microcontroller, addressing the growing demand for visual data acquisition in IoT applications. By seamlessly integrating a camera sensor into the ESP32 ecosystem, this module enables developers to effortlessly incorporate image and video capturing capabilities into their projects. This integration holds immense potential for IoT projects, home automation systems, attendance systems, surveillance systems, robotics, and various applications that require visual data analysis and interaction.[2]

1.2.2 Collaboration with Ai-Thinker

The development of the ESP32-Camera module is the result of a fruitful collaboration between two industry-leading entities, Espressif Systems and Ai-Thinker. Espressif Systems brings its expertise in microcontroller technology, while Ai-Thinker contributes its extensive knowledge in wireless and camera technologies. This collaboration ensures a high-quality and seamlessly integrated camera module that complements the ESP32 microcontroller's capabilities.[2]

1.2.3 Camera Sensor Capabilities

The ESP32-Camera module leverages a camera sensor to capture high-quality images and videos. The specific camera sensor employed may vary based on the module variant or version being used. Prominent sensors utilized in the ESP32-Camera module include the OV2640 and OV7725, renowned for their affordability and respectable image quality. These camera sensors provide a reliable foundation for capturing visual data in a variety of applications.[2]

1.2.4 Flexible Camera Resolutions

The ESP32-Camera module offers a range of resolutions to suit diverse project requirements. Developers can select resolutions that span from VGA (640x480) for lower-quality images, to higher resolutions such as 2 megapixels (1600x1200) or more, depending on the specific module variant. This flexibility allows developers to strike a balance between image quality, resource utilization, and available processing power within the ESP32 microcontroller.[2]

1.2.5 Seamless Communication and Integration

Designed to seamlessly integrate with the ESP32 microcontroller, the ESP32-Camera module leverages various communication and integration methods. It establishes a connection with the ESP32 via interfaces such as the serial peripheral interface (SPI) or other suitable interfaces. Captured image or video data can then be transmitted to a host device, such as a computer, server, or mobile device, over Wi-Fi or other communication protocols supported by the ESP32.[2]

1.2.6 Development Support and Libraries

To streamline the development process, Espressif provides a dedicated camera library within the ESP-IDF (Espressif IoT Development Framework). This library equips developers with functions and APIs for effortless camera initialization, configuration, and capturing processes. Additionally, it facilitates essential image processing tasks like color correction, resizing, and encoding. The ESP32-Camera module is compatible with popular development environments, including the Arduino IDE and PlatformIO, which enables a vast community of developers to harness its features and capabilities.[2]

1.3 The ESP32 Camera Hardware Overview

The heart of the ESP32-CAM is an ESP32-S System-on-Chip (SoC) from Ai-Thinker. Being an SoC, the ESP32-S chip contains an entire computer—the microprocessor, RAM, storage, and peripherals—on a single chip. While the chip's capabilities are quite impressive, the ESP32-CAM development board adds even more features to the mix. Let's take a look at each component one by one.[3]

1.3.1 The ESP32's Camera

The OV2640 camera sensor on the ESP32-CAM(Figure 1.2) is what sets it apart from other ESP32 development boards and makes it ideal for use in video projects.

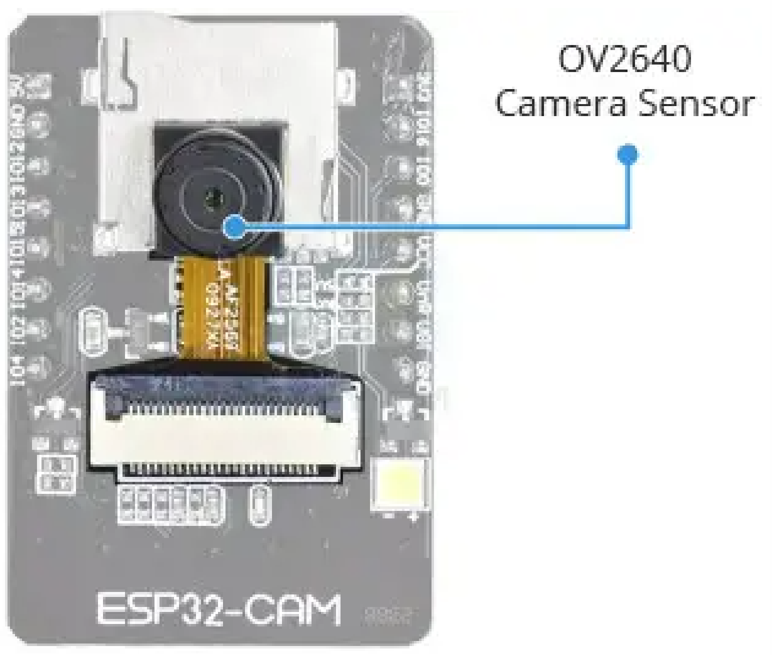


Figure 1.2: The ESP32-CAM Camera sensor

The OV2640 camera has a resolution of 2 megapixels, which translates to a maximum of 1600×1200 pixels, which is sufficient for many surveillance applications. The ESP32-CAM is compatible with a wide variety of camera sensors, as listed on its official libraries website.[2]

1.3.2 The ESP32 Camera Processor

The ESP32-CAM equips the ESP32-S surface-mount(Figure 1.3) printed circuit board module from Ai-Thinker.

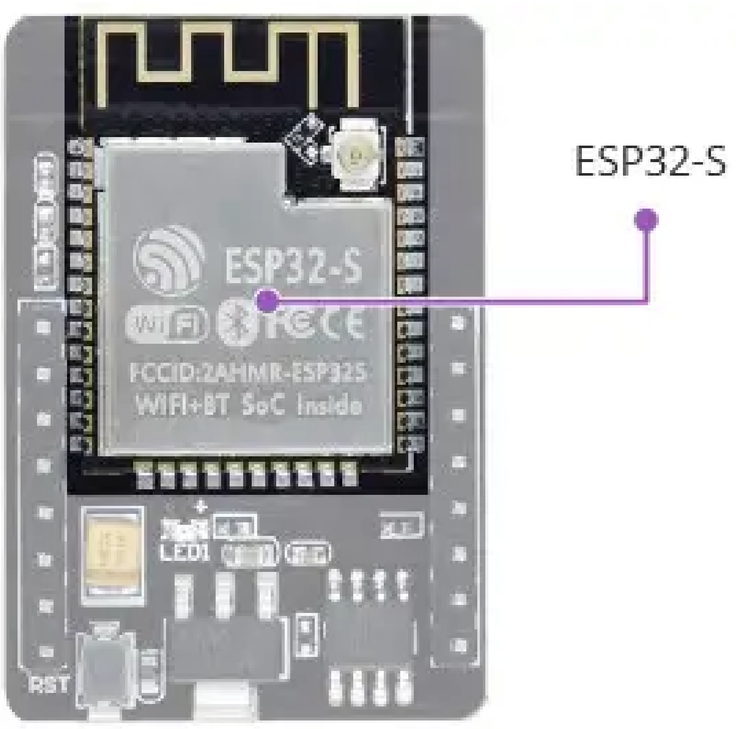


Figure 1.3: The ESP32-CAM Processor

The ESP32-S contains a Tensilica Xtensa® LX6 microprocessor with two 32-bit cores operating at a staggering 240 MHz. This is what makes the ESP32-S suitable for intensive tasks like video processing, facial recognition, and even artificial intelligence.[2][3]

1.3.3 The ESP32 Camera Memory

Memory is paramount for complex tasks, so the ESP32-S has a full 520 kilobytes of internal RAM(Figure 1.4), which resides on the same side as the rest of the chip's components.

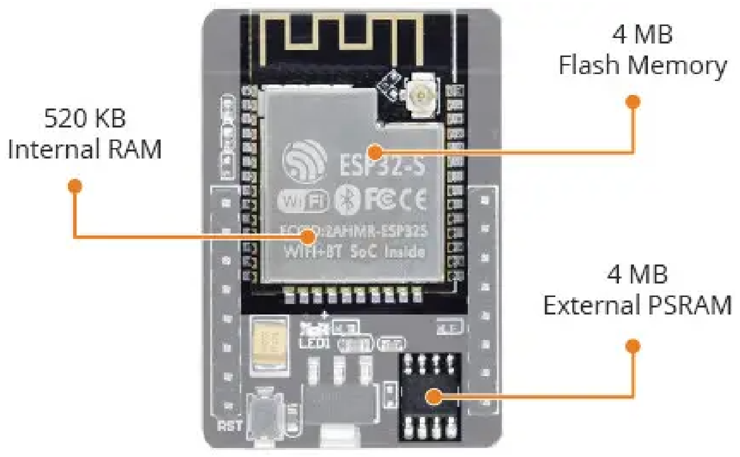


Figure 1.4: The ESP32-CAM Memory

It may be inadequate for RAM-intensive tasks, so ESP32-CAM includes 4 MB of external PSRAM to expand the memory capacity. This is plenty of RAM, especially for intensive audio or graphics processing. All these features amount to nothing if you don't have enough storage for your programs and data. The ESP32-S chip shines here as well, as it contains 4 MB of on-chip flash memory.[3][1]

1.3.4 The Storage

The addition of a microSD card slot(Figure 1.5) on the ESP32-CAM is a nice bonus. This allows for limitless expansion, making it a great little board for data loggers or image capture.[1]

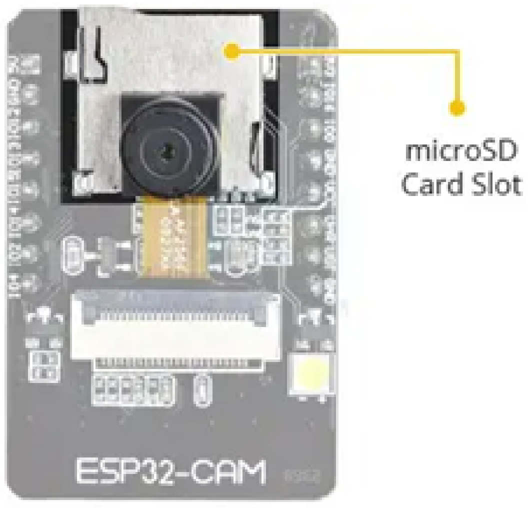


Figure 1.5: The ESP32-CAM Storage

1.3.5 The Antenna

The ESP32-CAM comes with an on-board PCB trace antenna as well as a u.FL connector(Figure 1.6) for connecting an external antenna. An Antenna Selection jumper (zero-ohm resistor) allows you to choose between the two options.[1]

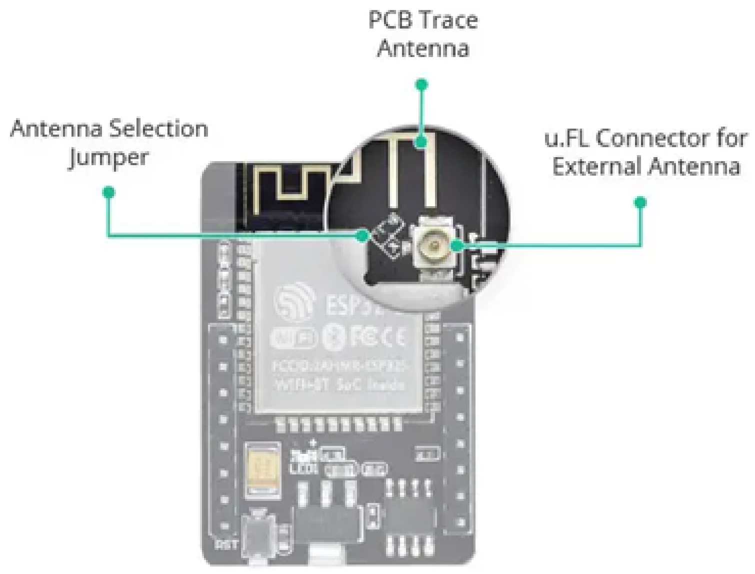


Figure 1.6: The ESP32-CAM Antenna

1.3.6 LEDs

The ESP32-CAM has a white square LED(Figure 1.7). It is intended to be used as a camera flash, but it can also be used for general illumination.[1]

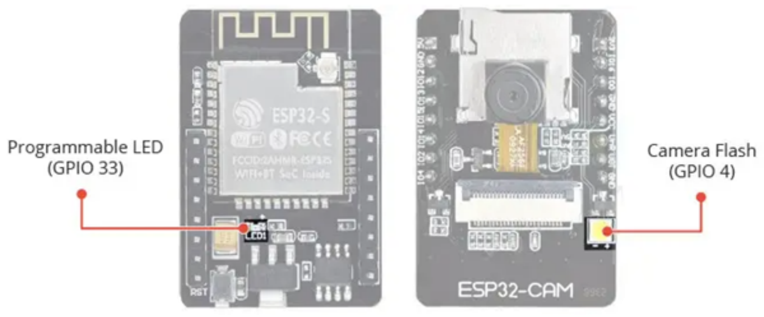


Figure 1.7: The ESP32-CAM LEDs

There is a small red LED on the back that can be used as a status indicator. It is user-programmable and connected to GPIO33.

1.4 Technical Specifications

To summarize, the ESP32-CAM has the following specifications[1]:

Processors: o CPU: Xtensa dual-core 32-bit LX6 microprocessor, operating at 240 MHz and performing at up to 600 DMIPS o Ultra-low power (ULP) co-processor Memory: o 520 KB SRAM o 4MB External PSRAM o 4MB internal flash memory

Wireless connectivity: o Wi-Fi: 802.11 b/g/n o Bluetooth: v4.2 BR/EDR and BLE (shares the radio with Wi-Fi)

Camera: o 2 Megapixel OV2640 sensor o Array size UXGA 1622×1200 o Output formats include YUV422, YUV420, RGB565, RGB555 and 8-bit compressed data o Image transfer rate of 15 to 60 fps o Built-in flash LED o Support many camera sensors

• Supports microSD card o Built-in microSD card slot • Security: o IEEE 802.11 standard security features all supported, including WFA, WPA/WPA2 and WAPI o Secure boot o Flash encryption o 1024-bit OTP, up to 768-bit for customers o Cryptographic hardware acceleration: AES, SHA-2, RSA, elliptic curve cryptography (ECC), random number generator (RNG)

• Power management: o Internal low-dropout regulator o Individual power domain for RTC o 5uA deep sleep current o Wake up from GPIO interrupt, timer, ADC measurements, capacitive touch sensor interrupt

1.5 ESP32 Camera parts Schematics

1.5.1 ESP-32S Module Schematic

ESP-32S Module Schematic(Figure 1.8).

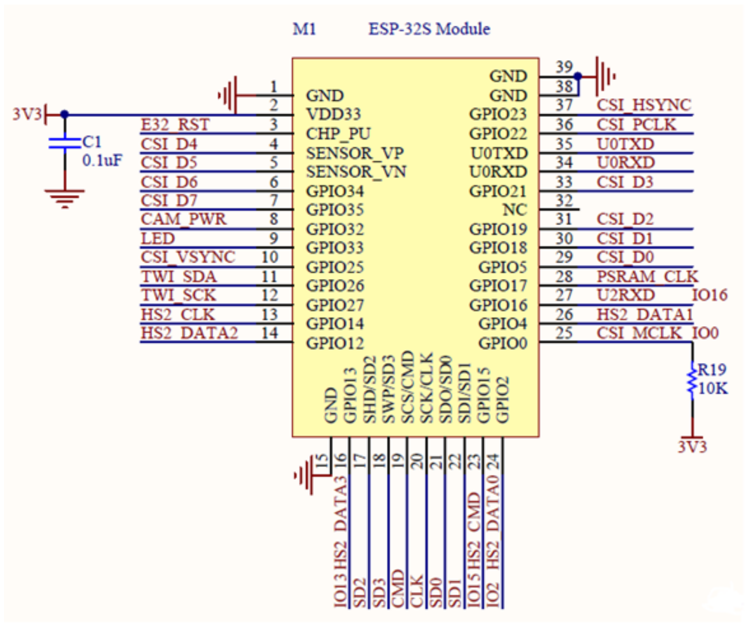


Figure 1.8: The ESP32-S Schematic

1.5.2 PSRAM Schematic

PSRAM Schematic(Figure 1.9).

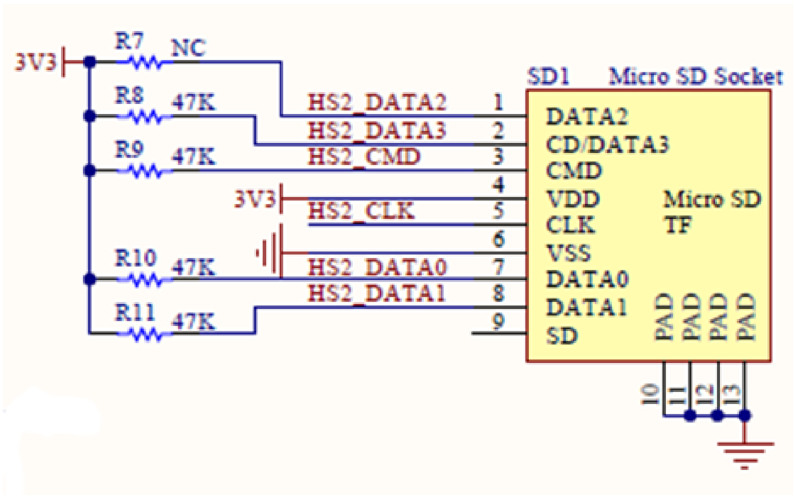


Figure 1.9: The ESP32-CAM PSRAM Schematic

1.5.3 MicroSD Socket Schematic

MicroSD Socket Schematic(Figure 1.10).

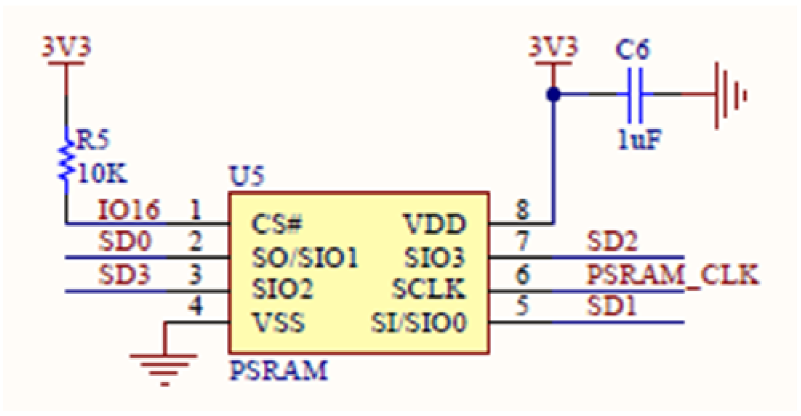


Figure 1.10: The ESP32-CAM MicroSD Socket Schematic

1.5.4 Camera module Schematic

Camera module Schematic(Figure 1.11).

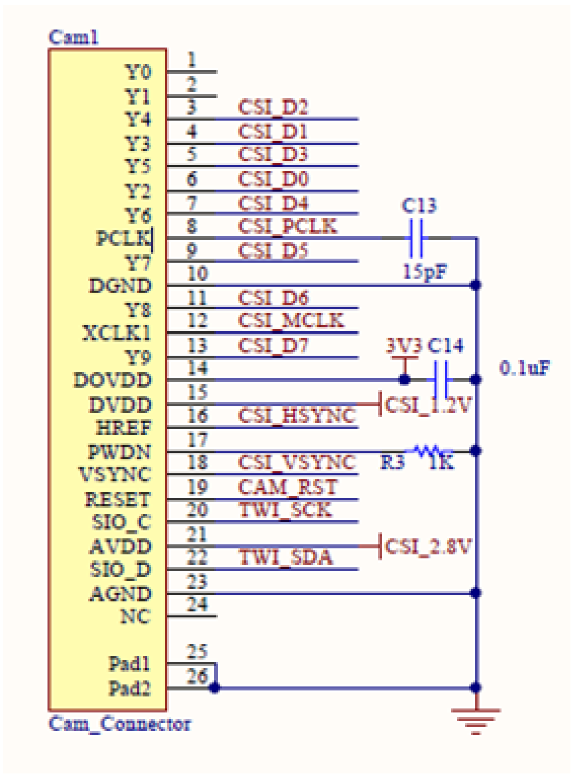


Figure 1.11: The ESP32-CAM Camera sensor Schematic

1.6 ESP32-CAM Power Consumption

The power consumption of the ESP32-CAM varies depending on what you're using it for. It ranges from 80 mAh when not streaming video to around 100 160 mAh when streaming video; with the flash on, it can reach 270 mAh.[1]

Operation mode	**Power Consumption (mAh)**
Standby	80
In streaming	100–160
In streaming with flash	270

Table 1.1: Power consumption in different operation modes

1.6.1 ESP32-CAM Pin-outs

The ESP32-CAM has 16 pins in total. For convenience, pins with similar functionality are grouped together. The pinout is as follows(Figure 1.12).[2][1]

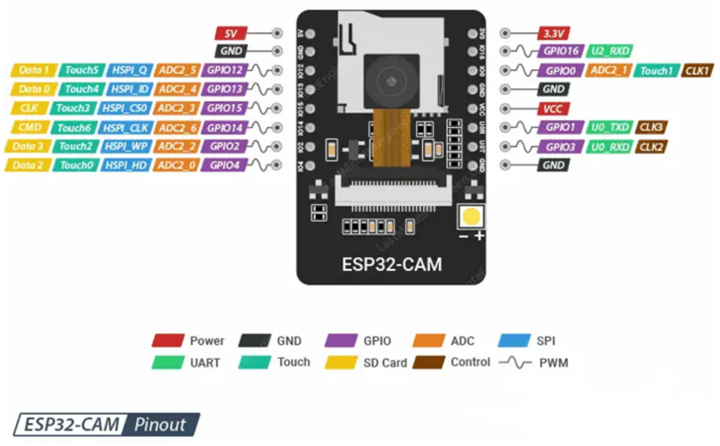

5V
GND
Data 1 Touch5 HSPI_Q ADC2_5 GPIO12
Data 0 Touch4 HSPI_ID ADC2_4 GPIO13
CLK Touch3 HSPI_CS0 ADC2_3 GPIO15
CMD Touch6 HSPI_CLK ADC2_6 GPIO14
Data 3 Touch2 HSPI_WP ADC2_2 GPIO2
Data 2 Touch0 HSPI_HD ADC2_0 GPIO4
3.3V
GPIO16 U2_RXD
GPIO0 ADC2_1 Touch1 CLK1
GND
VCC
GPIO1 U0_TXD CLK3
GPIO3 U0_RXD CLK2
GND
ESP32-CAM
Power
GND
GPIO
ADC
SPI
UART
Touch
SD Card
Control
PWM
ESP32-CAM Pinout

Figure 1.12: The ESP32-CAM Pin-outs

- **Power Pins** :

  There are two power pins: 5V and 3V3. The ESP32-CAM can be powered via the 3.3V or 5V pins. Since many users have reported problems when powering the device with 3.3V, it is advised that the ESP32-CAM always be powered via the 5V pin. The VCC pin normally outputs 3.3V from the on-board voltage regulator. It can, however, be configured to output 5V by using the Zero-ohm link near the VCC pin.

- GND is the ground pin.

- GPIO Pins:

  The ESP32-S chip has 32 GPIO pins in total, but because many of them are used internally for the camera and the PSRAM, the ESP32-CAM only has 10 GPIO pins available. These pins can be assigned a variety of peripheral duties, such as UART, SPI, ADC, and Touch.

- UART Pins:

The ESP32-S chip actually has two UART interfaces, UART0 and UART2. However, only the RX pin (GPIO 16) of UART2 is broken out, making UART0 the only usable UART on the ESP32-CAM (GPIO 1 and GPIO 3). Also, because the ESP32-CAM lacks a USB port, these pins must be used for flashing as well as connecting to UART-devices such as GPS, fingerprint sensors, distance sensors, and so on.

- MicroSD Card Pins:

  They are used for interfacing the microSD card. If you aren't using a microSD card, you can use these pins as regular inputs and outputs.

- ADC Pins :

  On the ESP32-CAM, only ADC2 pins are broken out. However, because ADC2 pins are used internally by the WiFi driver, they cannot be used when Wi-Fi is enabled.

- Touch Pins :

  Touch Pins The ESP32-CAM has 7 capacitive touch-sensing GPIOs. When a capacitive load (such as a human finger) is in close proximity to the GPIO, the ESP32 detects the change in capacitance.

- SPI Pins :

  The ESP32-CAM features only one SPI (VSPI) in slave and master modes.

- PWM Pins :

  The ESP32-CAM has 10 channels (all GPIO pins) of PWM pins controlled by a PWM controller. The PWM output can be used for driving digital motors and LEDs.

For more information, refer to our comprehensive ESP32-CAM pinout reference guide. This guide also explains which ESP32-CAM GPIO pins are safe to use and which pins should be used with caution.

1.7 Programming the ESP32-CAM

Programming the ESP32-CAM can be a bit of a pain as it lacks a built-in USB port. Because of that design decision, users require additional hardware in order to upload programs from the Arduino IDE. None of that is terribly complex, but it is inconvenient. To program this device, you'll need either a USB-to-serial adapter (an FTDI adapter) or an ESP32-CAM-MB programmer adapter.

1.7.1 Using the FTDI Adapter

If you've decided to use the FTDI adapter, here's how you connect it to the ESP32-CAM module(Figure 1.13).

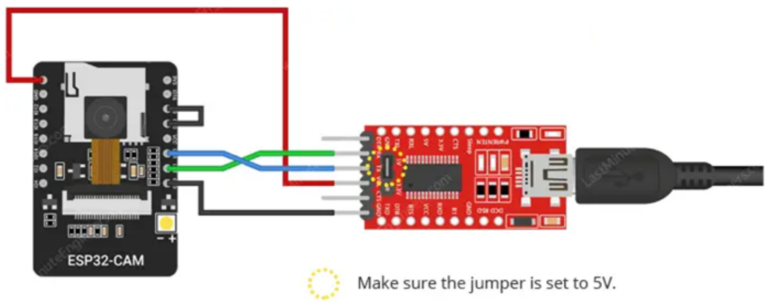


Figure 1.13: The ESP32-CAM FTDI Adapter

Many FTDI programmers have a jumper that lets you choose between 3.3V and 5V. As we are powering the ESP32-CAM with 5V, make sure the jumper is set to 5V.[2]

1.7.2 Using the ESP32-CAM-MB Adapter

Using the FTDI Adapter to program the ESP32-CAM is a bit of a hassle. This is why many vendors now sell the ESP32-CAM board along with a small add-on daughterboard called the ESP32-CAM-MB.

You stack the ESP32-CAM on the daughterboard, attach a micro USB cable, and click the Upload button to program your board. It's that simple.(Figure 1.14).[2]

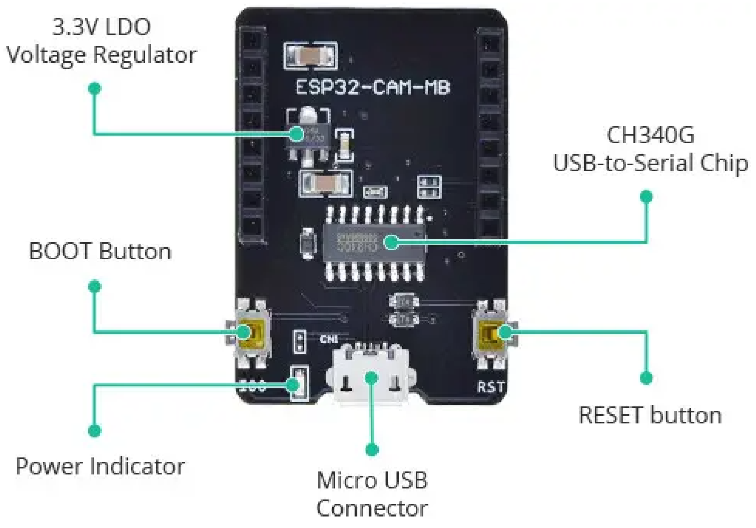


Figure 1.14: The ESP32-CAM -MB

The highlight of this board is the CH340G USB-to-Serial converter. That's what translates data between our computer and the ESP32-CAM. There's also a RESET button, a BOOT button, a power indicator LED, and a voltage regulator to supply the ESP32-CAM with plenty of power.

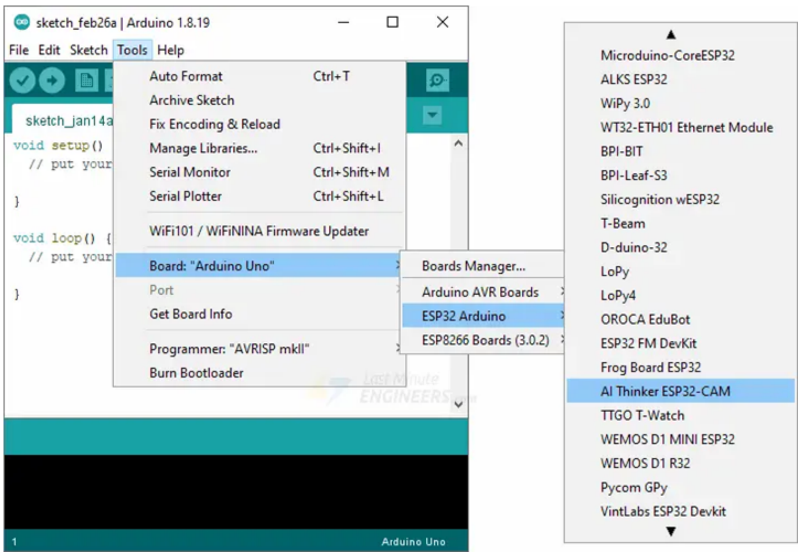
1.7.3 Setting Up the Arduino IDE

Installing the ESP32 Board

To use the ESP32-CAM, or any ESP32, with the Arduino IDE, you must first install the ESP32 board (also known as the ESP32 Arduino Core) via the Arduino Board Manager.

Selecting the Board and Port

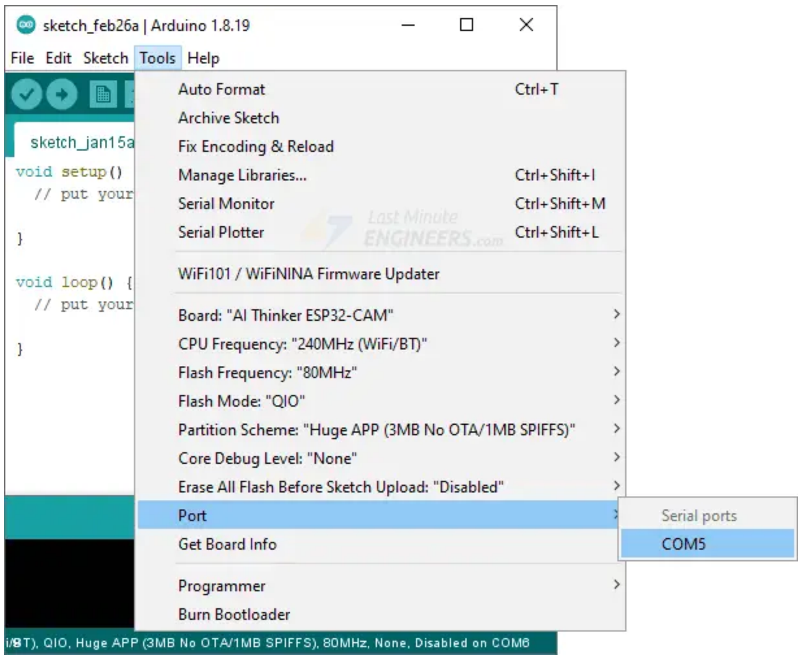
After installing the ESP32 Arduino Core, restart your Arduino IDE and navigate to Tools ¿ Board ¿ ESP32 Arduino and select AI-Thinker ESP32-CAM.(Figure 1.15).[2][3][1]



```
sketch_jan14a
void setup() {
  // put your
}

void loop() {
  // put your
}
```

Figure 1.15: Selecting the Board and Port

Now connect the ESP32-CAM to your computer using a USB cable. Then, navigate to Tools ¿ Port and choose the COM port to which the ESP32-CAM is connected.(Figure 1.16).


sketch_feb26a | Arduino 1.8.19
File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Port
Get Board Info
Programmer
Burn Bootloader
Serial ports
COM5
sketch_jan15a
Last Minute ENGINEERS.com

Board: "AI Thinker ESP32-CAM"
CPU Frequency: "240MHz (WiFi/BT)"
Flash Frequency: "80MHz"
Flash Mode: "QIO"
Partition Scheme: "Huge APP (3MB No OTA/1MB SPIFFS)"
Core Debug Level: "None"
Erase All Flash Before Sketch Upload: "Disabled"

```
void setup() {
  // put your
}

void loop() {
  // put your
}
```

i/BT), QIO, Huge APP (3MB No OTA/1MB SPIFFS), 80MHz, None, Disabled on COM6

Figure 1.16: Choosing the COM port

That’s it; the Arduino IDE is now set up for the ESP32-CAM!

1.8 Considerations

When deploying the ESP32-CAM for facial recognition, factors such as processing power limitations and camera resolution should be carefully considered, particularly in scenarios involving a significant number of students.

1.9 Ethical and Privacy Considerations

Ensuring adherence to ethical standards and privacy regulations is paramount. Obtaining informed consent from students and implementing robust data security measures are essential to protect individual privacy rights.

1.10 Conclusion

The ESP32-Camera module is one of the most powerful to date device in its field, allowing the realisation of many creative ideas and projects that it has successfully and without a doubt revolutionized the world of IoT. By following the steps in this chapter, the user will be able to take advantage of this incredible device and its features with ease, making sure to address the processing power constraints, optimizing camera resolution, and upholding ethical practices, the full potential of this technology can be realized, benefiting both administrators and students alike.

Chapter 2

Facial Recognition Technology

2.1 Introduction

Facial recognition is a category of bio-metric software that maps an individual's facial features mathematically and stores the data as a face-print. The software uses deep learning algorithms to compare a live capture or digital image to the stored face-print in order to verify an individual's identity. The origins of facial recognition technology can be traced back to the 1960s when researchers first ventured into automated face recognition. However, the nascent attempts were limited by the computational power and technology available at the time. In the 1990s, notable breakthroughs in face detection and recognition algorithms laid the foundation for more robust systems. The introduction of 3D facial recognition technology, capable of capturing intricate facial depth information, further enhanced accuracy and resilience.[4]

2.2 Facial recognition process

In order to achieve such technology, there will be many steps the go through, those are as follows :

2.2.1 The image capture process

The image capture process in facial recognition technology involves taking a photograph or video of a person's face using a camera or other image capture device, which in our case, we are going to be using the ESP32-Camera module for that. the figure(Figure 2.1) bellow shows an example on how a face image should be taken for the facial recognition process.[5]

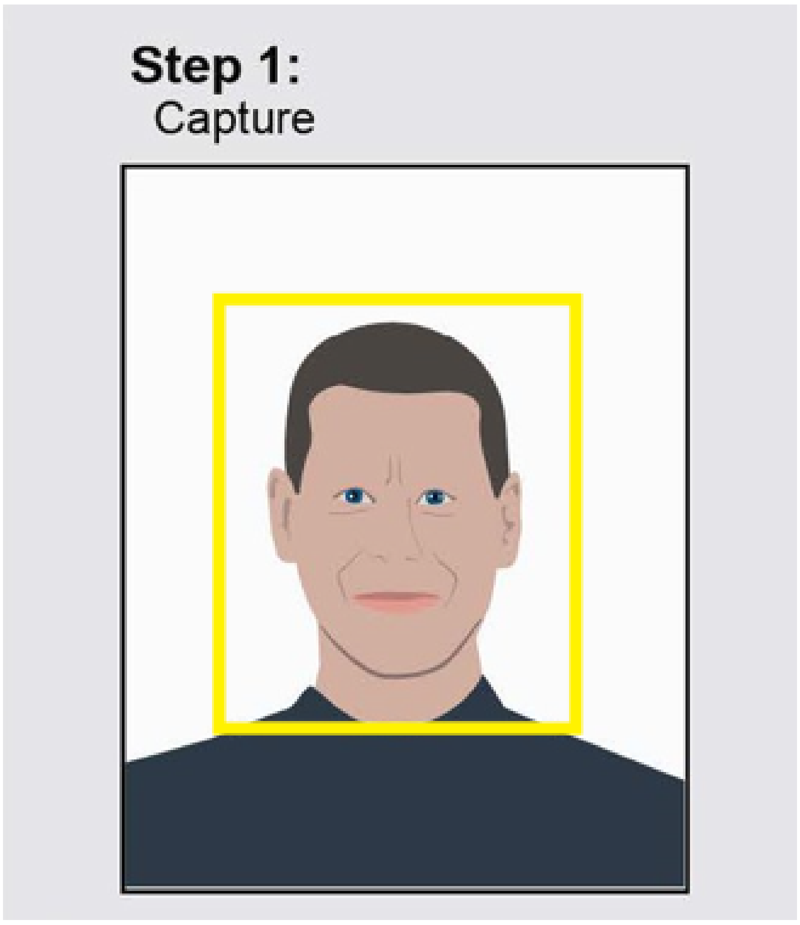


Figure 2.1: The image capture process

The ESP32 camera should be positioned at the appropriate distance and angle to capture the person's face, ideally with the face filling a large portion of the frame. Depending on the setting and the application, the image may be captured in a controlled environment, such as at a security checkpoint, or in an uncontrolled environment, such as in a university.

2.2.2 Face detection process

Face detection, also called facial detection, is an artificial intelligence (AI)-based computer technology used to find and identify human faces in digital images and video. Face detection uses machine learning (ML) and artificial neural network (ANN) technology, and plays an important role in face tracking, face analysis and facial recognition. In face analysis, face detection uses facial expressions to identify which parts of an image or video should be focused on. In a facial recognition system, face detection data is required to generate a face-print and match it with other stored face-prints. Face detection algorithms typically start by searching for human eyes, one of the easiest features to detect. They then try to detect facial landmarks, such as eyebrows, mouth, nOSe, nOStrils and irises. Once the algorithm concludes that it has found a facial region, it does additional tests to confirm that it has detected a face. The next figure(Figure 2.2) show an example on how the process is applied on the captured face image.[6] [5]

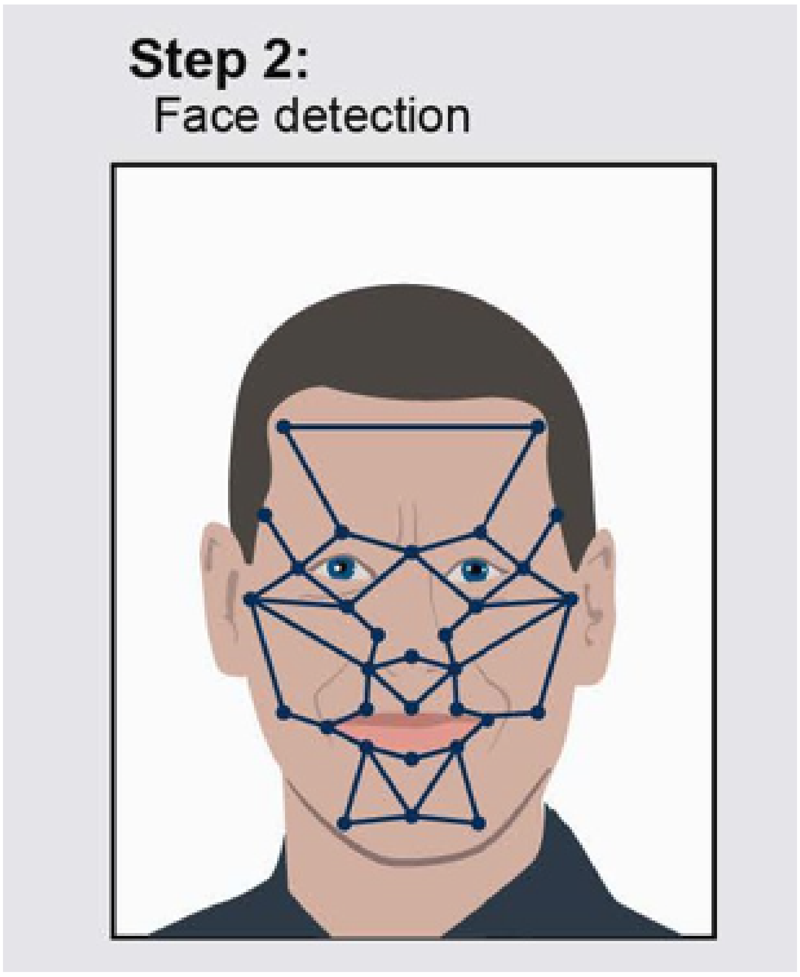


Figure 2.2: The face detection process

To ensure accuracy, the algorithms are trained on large data sets that incorporate hundreds of thousands of positive and negative images. The training improves the algorithms' ability to determine whether there are faces in an image and where they are.

2.2.3 Facial Alignment process

Once faces are detected, the process of facial alignment begins. This crucial step aims to standardize the position and orientation of the detected faces. By normalizing variations caused by different camera angles or facial poses, facial alignment ensures consistency in subsequent analysis and comparisons. The figure(Figure 2.3) bellow represent an example on how the alignment process of the face is done.

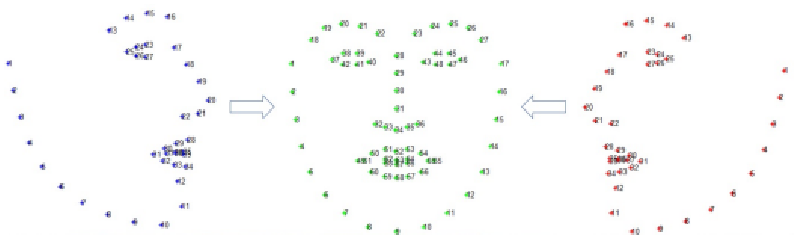


Figure 2.3: The face alignment process

With faces aligned, the system proceeds to extract unique facial features. Complex algorithms meticulously analyze and measure attributes like the size, shape, and texture of various facial components.[5]

2.2.4 Creation of face templates process

Based on the extracted features, the system generates individual face templates. These templates serve as digital reference points, encapsulating the numerical representation of an individual's facial attributes. The face templates are stored in a database for future comparisons and identification purposes. The following figure(Figure 2.4) illustrates the process of the detail conversion.[5]

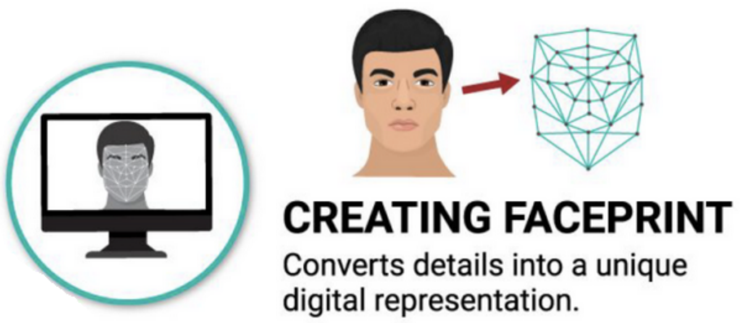


Figure 2.4: Digital reference points creationStudents enrollment process

2.2.5 The enrollment process

During the enrollment phase, individuals' faces are captured, and their face templates are created and stored. This process establishes a database of known identities, allowing the system to recognize and verify students in subsequent encounters. The next figure(Figure 2.5) shows students taking turns in-front of the camera to establish their attendance for their class.[5]

Figure 2.5: Students enrollment process

When facial recognition is performed, the system compares the facial features of an input face with the stored face templates. Complex algorithms calculate the similarity or dissimilarity between the features, determining potential matches or discrepancies.

2.2.6 Decision and Authentication process

Based on the comparison results, a decision is made regarding the identity of the input face. If a high-confidence match is found, the system can authenticate the person's identity. This authentication can be utilized for various purposes, such as enrolling presence in an attendance system or even granting access to secure areas, verifying identities, or enabling personalized experiences. The figure(Figure 2.6) bellow show a face image of a freshly enrolled individual being compared to the ones already in the database so that it will be listed as present in the attendance system if a match is found, if not than the individual will not be listed.

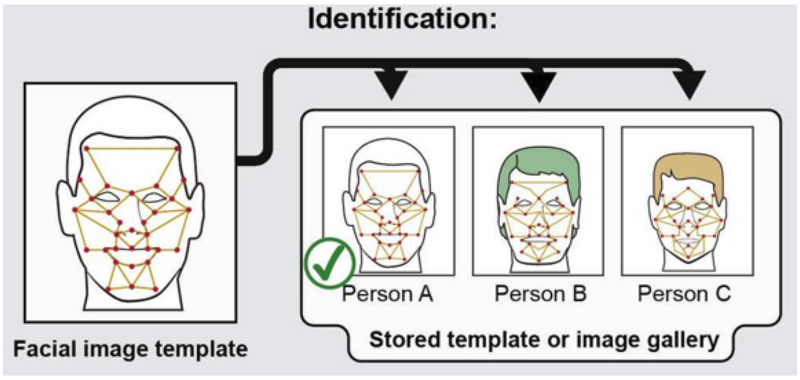


Figure 2.6: Students identity authentication

2.2.7 Ethical Considerations and Privacy

While facial recognition technology offers significant benefits, it also raises important ethical considerations. Privacy concerns, potential biases, and misuse of the technology need to be addressed through appropriate regulations and responsible deployment.[6][5]

2.3 facial recognition implementation using the Python programming language

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cOSt of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.[7] The figure(Figure 2.7) below is the official Python programming language logo.

Figure 2.7: Python programming language logo

2.3.1 Install Python

Visit the official Python website (https://www.python.org/) and download the Python installation package for your operating system. Follow the installation instructions to set up Python on your machine.[**python-tutorials**]

2.3.2 Python Libraries

In order for python to perform the tasks in hand, it relies on different kind of libraries, A library is a collection of code that makes everyday tasks more efficient. Using OpenCV, for example, you can generate visualizations with just one line of code. To create a chart from an object, you'd have to write a lot of code without a library like this. Python is a popular choice for data analysis because of its extensive library of tools for manipulating, visualizing, and training machine learning models.

The Open-CV Library

OpenCV (Open Source Computer Vision) is a popular open-source library for computer vision and image processing tasks. It provides a wide range of functions and algorithms for image and video analysis, including face recognition. In this walkthrough, I will provide you with an overview and a step-by-step guide on how to perform face recognition using OpenCV in Python.[8]

Install OpenCV

To begin, you need to install the OpenCV library in your Python environment. You can install it using pip by running the following command shown in the figure(Figure 2.8) bellow :

```
pip install opencv-python
```

Figure 2.8: OpenCV library install Python command

Import the necessary libraries: After installing OpenCV, import the required libraries in your Python script as it is demonstrated in the next figure(Figure 2.9) :

```
import cv2
import os
```

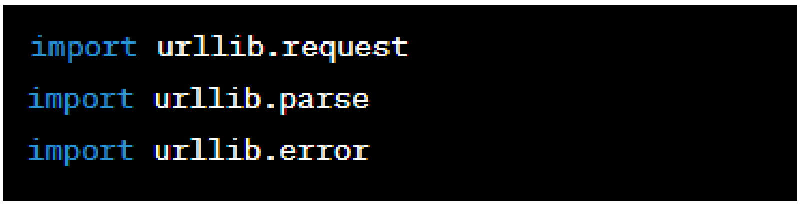
Figure 2.9: Import libraries in Python script

The urllib library

The urllib library in Python is a standard module that provides a set of functions and classes for handling URLs (Uniform Resource Locators). It allows you to interact with web resources by making HTTP requests, handling cookies, managing authentication, and more. The urllib library is part of the Python standard library, so there's no need to install it separately.[9]

Import the necessary modules

Begin by importing the required modules from the urllib library. the figure(Figure 2.10) bellow shows how to Import the necessary modules :

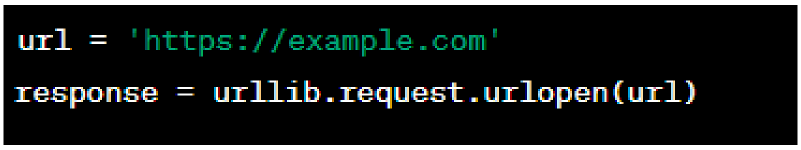


```python
import urllib.request
import urllib.parse
import urllib.error
```

Figure 2.10: urllib library import in Python script

Make a GET request To retrieve data from a URL using a GET request, use the urllib.request.urlopen() function.

This function returns a file-like object that you can use to read the response data. the figure(Figure 2.11) bellow gives a demonstration of this :



```python
url = 'https://example.com'
response = urllib.request.urlopen(url)
```

Figure 2.11: URL data retrieve using GET request in Python script

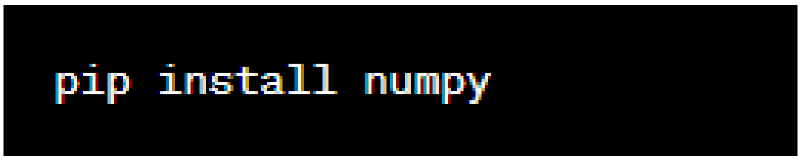
You can read the response data from the file-like object using methods such as read() or readlines().

The NumPy library

The NumPy library in Python is a powerful numerical computing library that provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently. It is a fundamental library for data manipulation and is commonly used in scientific and numerical computations.[10][11]

Install NumPy:

Begin by installing the NumPy library in your Python environment. The figure(Figure 2.12) below shows how to install it using the pip command:

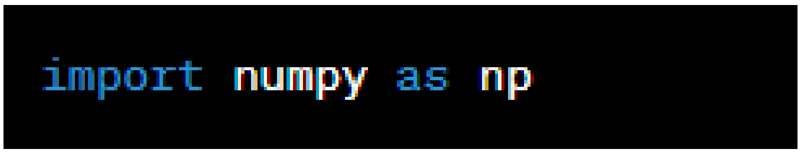


```
pip install numpy
```

Figure 2.12: NumPy library install Python command

Import the necessary module

After installing NumPy, import the module in your Python script like shown in the figure(Figure 2.13) below:



```
import numpy as np
```

Figure 2.13: NumPy library import in Python script

Load and preprocess face images:

Start by loading the face images you want to perform face recognition on. These images can be grayscale or color images. You can use any image loading library, such as OpenCV or PIL, to load the images.

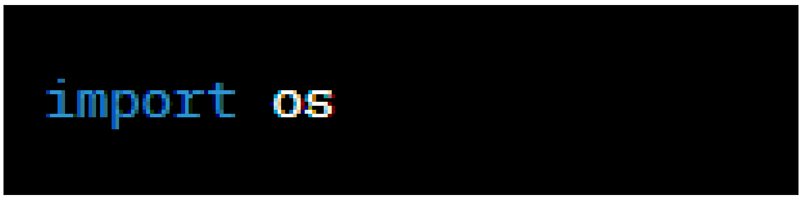
Preprocess the images, if needed, by resizing, normalizing, or converting them to grayscale.

The OS library

The OS library in Python is a standard library module that provides a way to interact with the operating system. It offers functions for working with files and directories, executing system commands, accessing environment variables, and more. The OS module is part of the Python standard library, so there's no need to install it separately.[12]

Import the necessary module

Start by importing the OS module in your Python script as shown in the figure(Figure 2.14) bellow :



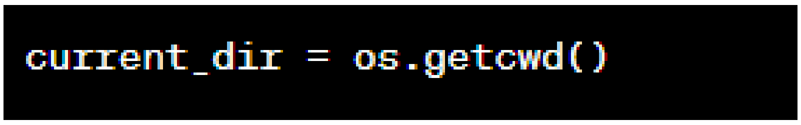
```
import os
```

Figure 2.14: OS library import in Python script

Working with files and directories:

• Get the current working directory:

Use the OS.getcwd() function to retrieve the current working directory like the next figure(Figure 2.15) shows:

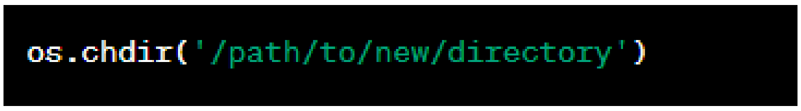


```
current_dir = os.getcwd()
```

Figure 2.15: OS library working directory

• Change the current working directory

You can change the current working directory using the OS.chdir() function, the figure(Figure 2.16) bellow gives an example:



```
os.chdir('/path/to/new/directory')
```

Figure 2.16: OS library directory change

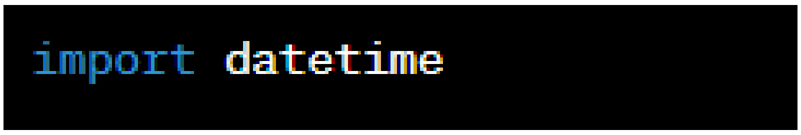
The OS has many functions such as we have seen so we will suffice with these few examples.

The datetime library

The datetime library in Python is a standard library module that provides classes for manipulating dates and times. It allows you to create, manipulate, and format date and time objects. The datetime module is part of the Python standard library, so there's no need to install it separately.[13]

1. Import the necessary module:

Start by importing the datetime module in your Python script. This can be done like shown in the next figure(Figure 2.17)

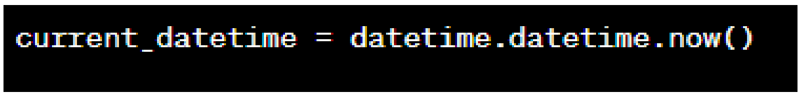


```
import datetime
```

Figure 2.17: Import the datetime module in Python script

Working with dates and times:

• Get the current date and time: Use the datetime.datetime.now() function to retrieve the current date and time, here is an example in the next figure(Figure 2.18)

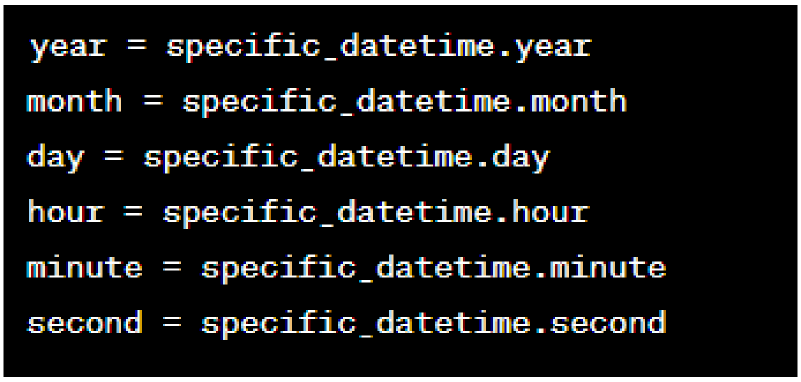


```
current_datetime = datetime.datetime.now()
```

Figure 2.18: Get the current date and time Python function

Access individual components of a datetime object:

You can access the year, month, day, hour, minute, and second components of a datetime object using their respective attributes as shown in the next figure(Figure 2.19).



```
year = specific_datetime.year
month = specific_datetime.month
day = specific_datetime.day
hour = specific_datetime.hour
minute = specific_datetime.minute
second = specific_datetime.second
```

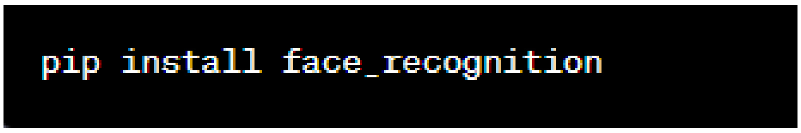
Figure 2.19: individual components of a datetime object

As it is showing, this library plays an instrumental part especially that we are creating an attendance system which relies heavily on time tracking.

The face recognition library

The face-recognition library in Python is a powerful and easy-to-use library for face recognition and facial feature detection. It is built on top of dlib, a C++ library renowned for its state-of-the-art face recognition algorithms. The face-recognition library provides a high-level API that simplifies the process of face detection, face recognition, facial landmark detection, and face encoding.[6][5]

1. Install the library: Begin by installing the face-recognition library in your Python environment. You can install it using pip by running the following command, this is shown in the figure(Figure 2.20).

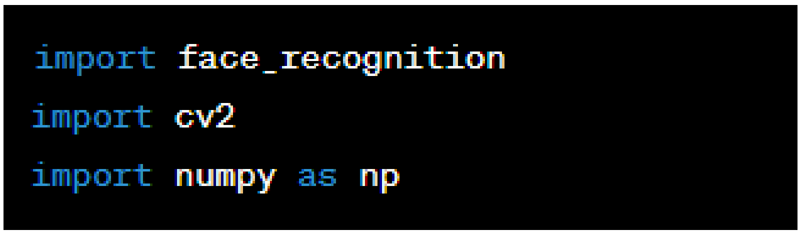


```
pip install face_recognition
```

Figure 2.20: installing the face-recognition library

Import the necessary modules

After installing face-recognition, import the required modules in your Python script. this is demonstrated in the next figure(Figure 2.21).



```
import face_recognition
import cv2
import numpy as np
```

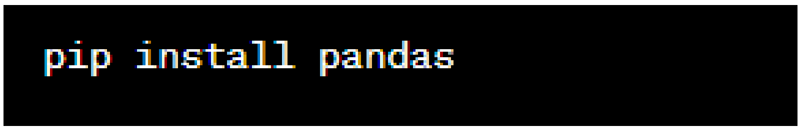
Figure 2.21: import face-recognition modules in Python script

Load and preprocess face images: Start by loading the face images you want to perform face recognition on. You can use any image loading library, such as OpenCV or PIL, to load the images. Preprocess the images, if needed, by resizing, normalizing, or converting them to grayscale.

The pandas Library

The pandas library in Python is a powerful and widely used open-source data manipulation and analysis tool. It provides data structures and functions for efficiently handling structured data, such as tables or spreadsheets, and performing various operations like filtering, sorting, grouping, and aggregating. Although pandas is not directly related to facial recognition, it can be useful for organizing and manipulating data related to facial recognition tasks.[14][14]

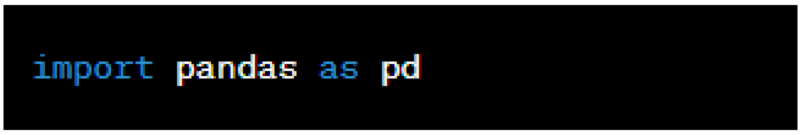
1. Install the library: Begin by installing the pandas library in your Python environment. You can install it using pip by running the following command that is shown in the figure(Figure 2.22).



```
pip install pandas
```

Figure 2.22: Install the pandas Library

Import the necessary module: After installing pandas, import the module in your Python script. the figure(Figure 2.23).



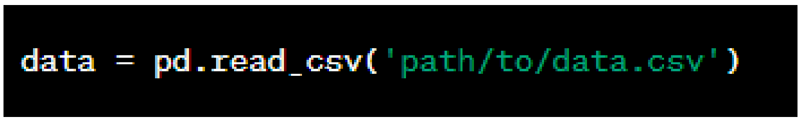
```
import pandas as pd
```

Figure 2.23: Import the pandas module

Loading and organizing data:

- Load data from a file:

pandas provides functions like read-csv(), read-excel(), and read-sql() to load data from different file formats or data sources. For facial recognition, you may have data such as image paths, labels, or facial features stored in a CSV or Excel file. Use the appropriate function to load the data into a DataFrame. this is shown in the next figure(Figure 2.24).



```
data = pd.read_csv('path/to/data.csv')
```

Figure 2.24: Loading data from a file

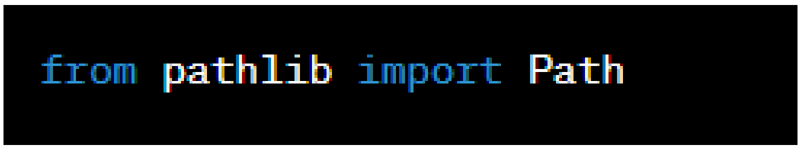
And many other features such as :

• Create a DataFrame • Accessing and manipulating data • Saving data to a file And so on.

The pathlib library

The pathlib library in Python provides an object-oriented approach to working with file system paths. It offers a high-level, platform-independent API for manipulating and accessing files and directories. While pathlib itself does not have specific functionality for facial recognition, it can be useful for handling file paths and organizing the data related to facial recognition tasks.[15]

1. Import the necessary module: Start by importing the pathlib module in your Python script, the figure(Figure 2.25) bellow will show how to do this task.



```
from pathlib import Path
```

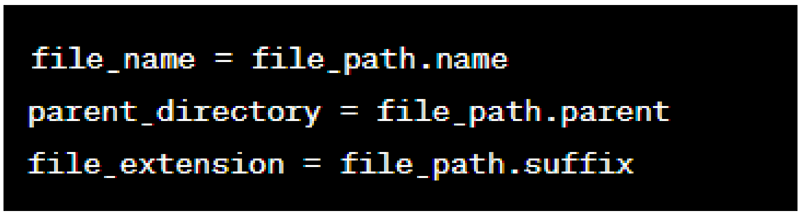
Figure 2.25: Importing the pathlib module in your Python script

Working with file paths:

Create a Path object: Use the Path class to create a Path object representing a file or directory path. You can initialize it with a string representing the path.

Access path components:

The Path object provides properties and methods to access different components of the path, such as the file name, parent directory, file extension, etc. The next figure(Figure 2.26) will help demonstrate this.



```
file_name = file_path.name
parent_directory = file_path.parent
file_extension = file_path.suffix
```

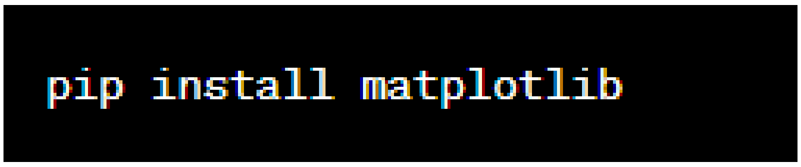
Figure 2.26: Path components access Python script

And many other option that make this library of a great value.

The matplotlib library

The matplotlib library in Python is a powerful and widely used data visualization tool. It provides a comprehensive set of functions for creating a wide range of plots, including line plots, scatter plots, bar plots, histograms, pie charts, and more. With matplotlib, you can create visually appealing and informative visualizations to explore and communicate your data.[16][17]

Install the library: Begin by installing the matplotlib library in your Python environment. You can install it using pip by running the following command show in the figure(Figure 2.27).

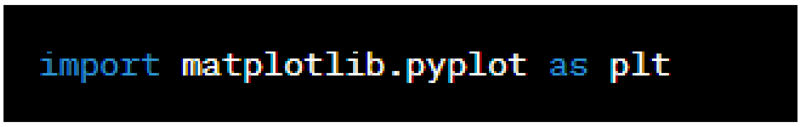


```
pip install matplotlib
```

Figure 2.27: installing the matplotlib library

Import the necessary module:

After installing matplotlib, import the module in your Python script just like in the next figure(Figure 2.28) .



```
import matplotlib.pyplot as plt
```

Figure 2.28: import the matplotlib module in Python script

And as it will be demonstrated later in our code, it gives as much more options to enrich our code.

2.4 Microsoft Visual Studio

Microsoft Visual Studio is an integrated development environment (IDE) developed by Microsoft. It provides a comprehensive set of tools and features for software development, including support for various programming languages such as Python. Visual Studio offers a rich set of capabilities that can greatly enhance the development process for your face recognition project.

In our project, we will be specificly using the Microsoft Visual Studio Community because there are certain dependencies that we will be required, to install the libraries.[18]

Microsoft Visual Studio Installation

Start by downloading and installing Microsoft Visual Studio Community from the official website (https://visualstudio.microsoft.com/vs/community/). Make sure to select the version that suits your operating system and requirements.

Project setup

• Create a new project: Launch Visual Studio Community and create a new project by selecting the appropriate project template. For a Python face recognition project, you can choose the Python Application template.

• Configure project settings: Set the project name, location, and choose the desired Python interpreter. You can configure additional project settings based on your specific requirements.

Coding and development

• Add Python files:

Within the project, you can add Python files to organize your code. Right-click on the project in the Solution Explorer panel, select "Add" and then choose "New Item" to create a new Python file.

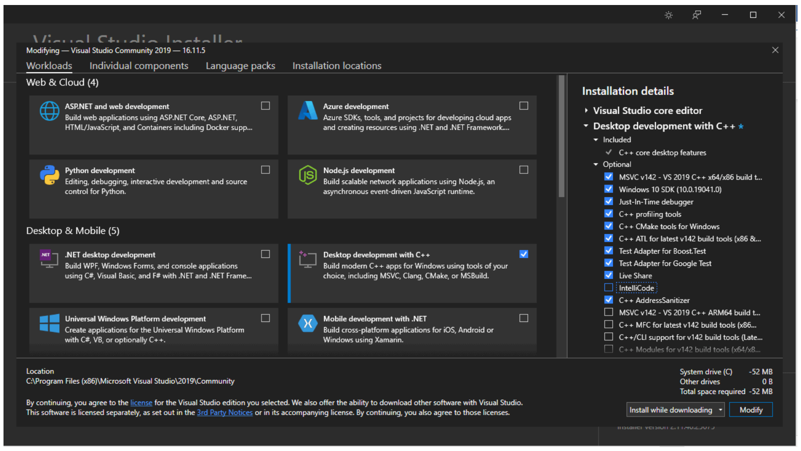
• Write face recognition code:

Write your face recognition code using the libraries and algorithms of your choice. This may include importing and utilizing libraries such as OpenCV, NumPy, and the face-recognition library. Implement functions for tasks such as face detection, feature extraction, and recognition.

• Utilize Visual Studio Community features:

Visual Studio Community offers several features that can improve your development experience, such as code editing and debugging capabilities. Take advantage of features like IntelliSense (code autocompletion and suggestions), code navigation, and the built-in Python debugger to simplify development and troubleshooting. Visual Studio Community provides a robust development environment with numerous features to enhance productivity and streamline the development process. It is recommended to explore the official documentation and tutorials provided by Microsoft for more detailed information on specific features and functionalities. In our project specifically, we will be relying on it only for its desktop development with C++ building tools, as it is a Microsoft product and we are using a windows OS, it is the absolute to go for choice in order to avoid encountering any troubles while working on our program script.

The next figure(Figure 2.29) will show the process mentioned above.


Modifying — Visual Studio Community 2019 — 16.11.5

Web & Cloud (4)

- ☐ **ASP.NET and web development** Build web applications using ASP.NET Core, ASP.NET, HTML/JavaScript, and Containers including Docker supp...
- ☐ **Azure development** Azure SDKs, tools, and projects for developing cloud apps and creating resources using .NET and .NET Framework....
- ☐ **Python development** Editing, debugging, interactive development and source control for Python.
- ☐ **Node.js development** Build scalable network applications using Node.js, an asynchronous event-driven JavaScript runtime.

Desktop & Mobile (5)

- ☐ **.NET desktop development** Build WPF, Windows Forms, and console applications using C#, Visual Basic, and F# with .NET and .NET Frame...
- ✓ **Desktop development with C++** Build modern C++ apps for Windows using tools of your choice, including MSVC, Clang, CMake, or MSBuild.
- ☐ **Universal Windows Platform development** Create applications for the Universal Windows Platform with C#, VB, or optionally C++.
- ☐ **Mobile development with .NET** Build cross-platform applications for iOS, Android or Windows using Xamarin.

Location
C:\Program Files (x86)\Microsoft Visual Studio\2019\Community

By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the 3rd Party Notices or in its accompanying license. By continuing, you also agree to those licenses.

Installation details

- Visual Studio core editor
- Desktop development with C++ *
  - Included
    - ✓ C++ core desktop features
  - Optional
    - ✓ MSVC v142 - VS 2019 C++ x64/x86 build t...
    - ✓ Windows 10 SDK (10.0.19041.0)
    - ✓ Just-In-Time debugger
    - ✓ C++ profiling tools
    - ✓ C++ CMake tools for Windows
    - ✓ C++ ATL for latest v142 build tools (x86 &...
    - ✓ Test Adapter for Boost.Test
    - ✓ Test Adapter for Google Test
    - ✓ Live Share
    - ☐ IntelliCode
    - ✓ C++ AddressSanitizer
    - ☐ MSVC v142 - VS 2019 C++ ARM64 build t...
    - ☐ C++ MFC for latest v142 build tools (x86...
    - ☐ C++/CLI support for v142 build tools (Late...
    - ☐ C++ Modules for v142 build tools (x64/x8...

System drive (C)	-52 MB
Other drives	0 B
Total space required	-52 MB

Figure 2.29: Visual Studio Community required features installation

2.5 CMake

CMake is an open-source, cross-platform build system that allows you to control the compilation process of your software project. It provides a platform-independent way to specify the build configuration, dependencies, and compilation steps for your project, making it easier to build and distribute your face recognition project across different operating systems. Basically without this, our project would not be user friendly and not compatible for the most part, hence non functional.[19] [20][21]

Installation

Start by downloading and installing CMake from the official website (https://cmake.org/download/
Choose the version that corresponds to your operating system. In our case, we are operating on the latest windows 10 version.

Document dependencies

Document the dependencies and requirements of your face recognition project, including the required version of Python, libraries, and any other prerequisites. CMake provides a flexible and platform-independent build system that simplifies the build process and enhances portability. It is recommended to refer to the official CMake documentation and tutorials for more in-depth information on using CMake for your specific project requirements.

2.6 Visual Studio Code (VSCode)

Visual Studio Code (VSCode) is a free and open-source code editor developed by Microsoft. It is a lightweight and highly customizable IDE that provides a wide range of features and extensions to support Python development for your face recognition project. VSCode will be our main IDE in this project.[22][23] [24]

Installation

Start by downloading and installing VSCode from the official website (https://code.visualstudio.co
Choose the version that corresponds to your operating system.

Install Python extension

Launch VSCode and install the Python extension to enable Python development within the editor. Open the Extensions view, search for ”Python”, and click the Install button next to the Python extension.

Project setup

• Create a new project folder:

Create a new folder on your computer to hold your face recognition project files.

• Open the project folder in VSCode:

Open VSCode and select ”Open Folder” from the File menu. Navigate to the project folder you created and select it to open it in VSCode.

• Create Python files:

Within the project folder, create Python files to write your face recognition code. Right-click on the folder in the Explorer panel, select ”New File”, and give it a Python file extension (e.g., .py).

Coding and development

• Write face recognition code:

Open the Python file in VSCode and write your face recognition code using the libraries and algorithms of your choice. Import necessary libraries like OpenCV, NumPy, and the face-recognition library. Implement functions for tasks such as face detection, feature extraction, and recognition.

• Utilize VSCode features:

VSCode provides a range of features to enhance your development experience. Take advantage of features like IntelliSense (code autocompletion and suggestions), code formatting, code navigation, and integrated terminal for running Python scripts.

Debugging

• Set breakpoints:

Use the built-in debugger in VSCode to set breakpoints in your code. Click in the gutter area next to a line of code to set a breakpoint, which will pause the execution of your program at that point.

• Debug configuration:

Create a debug configuration file (launch.json) in the .vscode folder of your project. Configure the Python interpreter path and set the appropriate launch options for debugging.

• Start debugging:

Press F5 or use the Debug menu to start debugging your code. The execution will pause at breakpoints, allowing you to inspect variables, step through code, and identify and fix issues.

Install extensions

• Install relevant extensions:

VSCode has a vast ecosystem of extensions that can enhance the functionality for face recognition projects. Explore the extensions marketplace to find extensions related to image processing, computer vision, and facial recognition. Some popular extensions include OpenCV, Pillow, and NumPy extensions.

Collaboration and version control

• Git integration:

VSCode has built-in Git integration, enabling version control for your face recognition project. Initialize a Git repository, commit changes, and collaborate with team members using features like branch management and pull requests.

• Live Share:

VSCode's Live Share extension allows real-time collaboration with other developers. You can share your workspace, collaborate on code, and debug together remotely.

Deployment and release

• Package and distribute:

Once your face recognition project is complete, you can package it for distribution. Use tools like PyInstaller to create standalone executables or create a distribution package using setup.py with setuptools. • Publish to cloud services:

If you plan to deploy your face recognition project to the cloud, follow the respective platform's guidelines for deployment. Popular cloud platforms like Azure and AWS provide documentation on deploying Python applications.

VSCode offers a lightweight yet powerful development environment with extensive customization options through extensions. It is recommended to explore the official documentation and tutorials for VSCode, as well as the documentation for specific extensions, to fully leverage its capabilities for your project.

2.7 Conclusion

In conclusion, implementing facial recognition technologies using the Python environment correctly and achieving results is somewhat challenging as each and every single component requires attention to detail and time, that being said, operating this on a windows os add to that challenge when compared to other OS platform such as Linux or Mac that are acclimated for such work environments. in overall, when taking the necessary time and going through every single and each step of this chapter you will be able to successfully implement the facial recognition technology without much trouble. This now allows us to move to the next important part of our project successfully.

Chapter 3

A facial recognition based attendance system using the esp32 camera module

3.1 Introduction

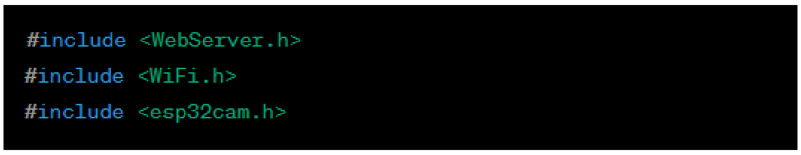
In this chapter, both the previous chapter 1 and 2 contents will come to realization, The Python code that will be used for programming will be working together accordingly with the programming of the ESP32-Cam module in purpose of achieving functionality and performance all while maintaining reliability.[25]

3.2 Setting the ESP32-Camera Module

In order to go ahead and use the esp32-cam for facial recognition, first we need to establish a real time live stream that will be feeding us with the data needed to accomplish the facial recognition process. The ESP32-Cam will be programmed with a certain code that will let us achieve live stream function, in most cases, esp32-cam live stream process is not intended for direct incorporated technologies such as facial recognition, so going ahead with the esp32-cam programming we need to keep that in mind as if else, we will encounter many streaming issues and inconveniences that will render out esp32-cam module not very reliable.

3.2.1 Arduino Code Breakdown

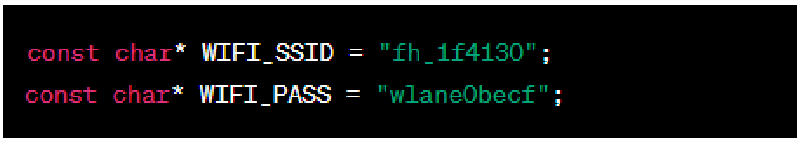
These lines include the necessary libraries for the program: Web-Server, WiFi, and esp32cam (Figure 3.1).



```
#include <WebServer.h>
#include <WiFi.h>
#include <esp32cam.h>
```

Figure 3.1: Arduino employed ESP32-Cam libraries

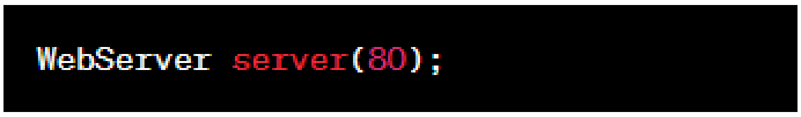
These lines define the Wi-Fi network credentials (SSID and password) that the ESP32-CAM module will connect to (Figure 3.2).



```
const char* WIFI_SSID = "fh_1f4130";
const char* WIFI_PASS = "wlane0becf";
```

Figure 3.2: setting the wireless connection server for the ESP32-Cam

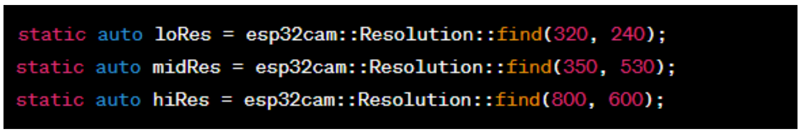
This line creates an instance of the Web-Server class and assigns it to the server object. The 80 parameter indicates that the server will listen on port 80, which is the default port for HTTP communication (Figure 3.3).



```
WebServer server(80);
```

Figure 3.3: HTTP Web-Server communication creation

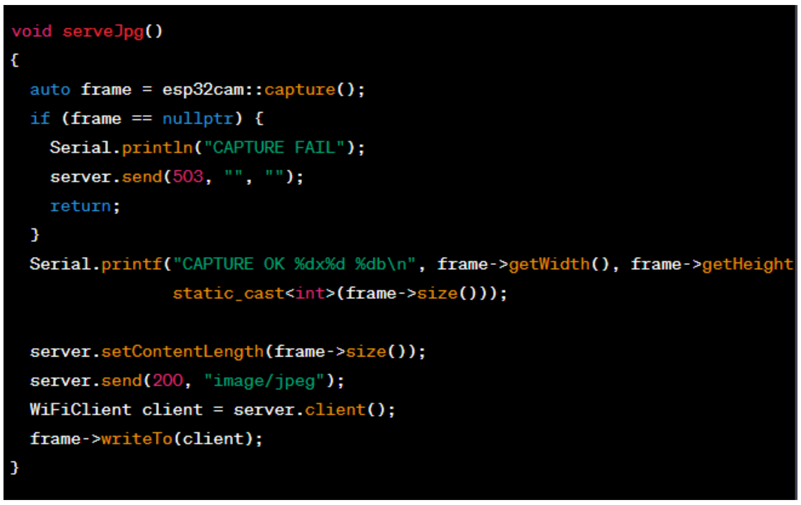
These lines define three different resolutions (loRes, midRes, and hiRes) using the esp32cam::Resolution::find() function. Each resolution is represented by a pair of width and height values.(Figure 3.4)



```
static auto loRes = esp32cam::Resolution::find(320, 240);
static auto midRes = esp32cam::Resolution::find(350, 530);
static auto hiRes = esp32cam::Resolution::find(800, 600);
```

Figure 3.4: ESP32-Cam Resolution setting

This function, serveJpg(), is responsible for capturing an image using the ESP32-CAM module and serving it as a JPEG image. It first captures a frame using esp32cam::capture(). If the frame is null (indicating a capture failure), it prints an error message, sends a 503 status code (Service Unavailable) to the client, and returns.(Figure 3.5)

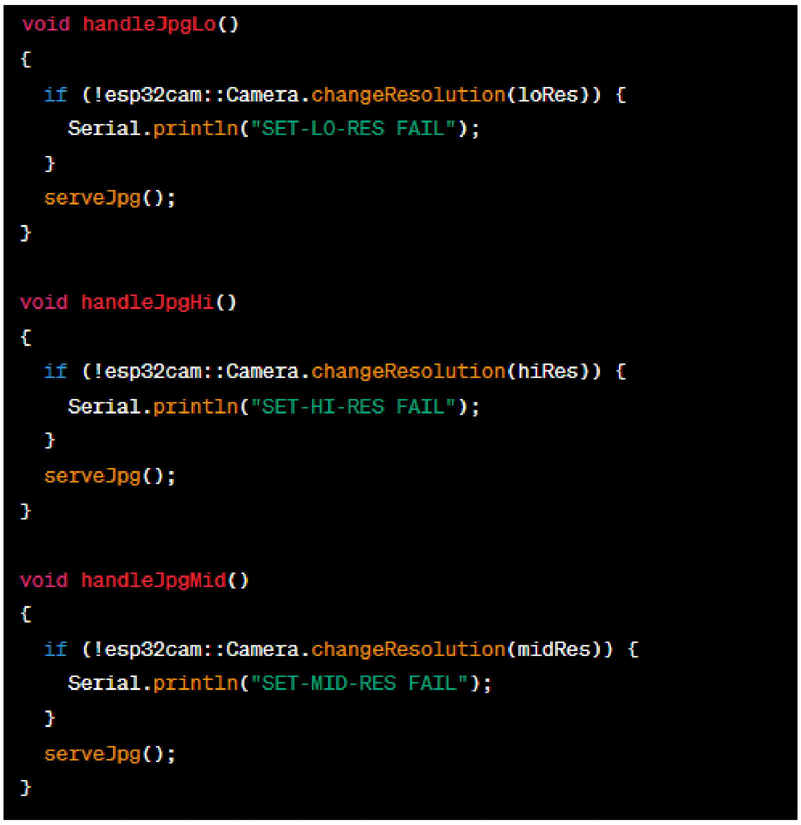


```
void serveJpg()
{
  auto frame = esp32cam::capture();
  if (frame == nullptr) {
    Serial.println("CAPTURE FAIL");
    server.send(503, "", "");
    return;
  }
  Serial.printf("CAPTURE OK %dx%d %db\n", frame->getWidth(), frame->getHeight
                static_cast<int>(frame->size()));

  server.setContentLength(frame->size());
  server.send(200, "image/jpeg");
  WiFiClient client = server.client();
  frame->writeTo(client);
}
```

Figure 3.5: ESP32-Cam image capture process setting

If the capture is successful, it prints the capture details (width, height, and size) to the serial monitor. Then, it sets the content length in the server's response, sends a 200 status code (OK) along with the content type as "image/jpeg", and obtains the client connection. Finally, it writes the frame data to the client.

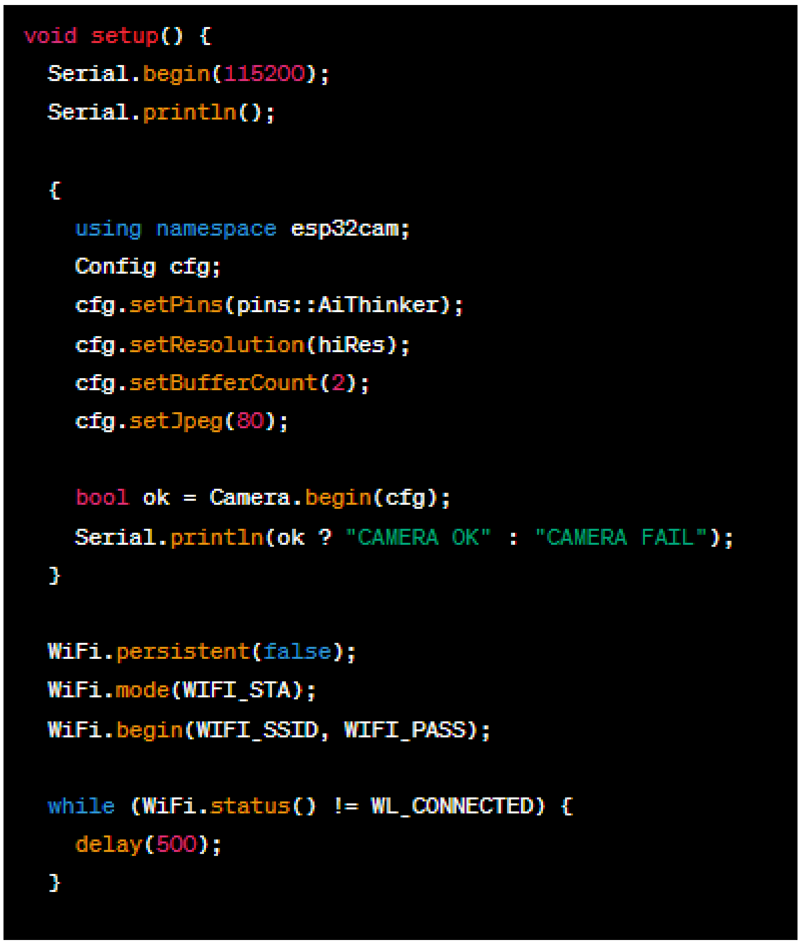
These three functions (handleJpgLo(), handleJpgHi(), and handleJpgMid()) handle HTTP requests for specific JPEG image resolutions. They change the resolution of the camera using esp32cam::Camera.changeResolution() and then call the serveJpg() function to capture and serve the image.(Figure 3.6)



```
void handleJpgLo()
{
  if (!esp32cam::Camera.changeResolution(loRes)) {
    Serial.println("SET-LO-RES FAIL");
  }
  serveJpg();
}

void handleJpgHi()
{
  if (!esp32cam::Camera.changeResolution(hiRes)) {
    Serial.println("SET-HI-RES FAIL");
  }
  serveJpg();
}

void handleJpgMid()
{
  if (!esp32cam::Camera.changeResolution(midRes)) {
    Serial.println("SET-MID-RES FAIL");
  }
  serveJpg();
}
```

Figure 3.6: HTTP requests handling for specific resolutions

The setup() function is the initial setup routine that runs once when the board is powered on or reset. It begins by starting the serial communication and printing some initial messages. Inside the setup() function, the camera configuration is set up using the esp32cam::Config class. It sets the camera pins to those of the Ai-Thinker module, the resolution to hiRes defined earlier, the buffer count to 2, and the JPEG quality to 80. The Camera.begin(cfg) function initializes the camera with the specified configuration.(Figure 3.7)

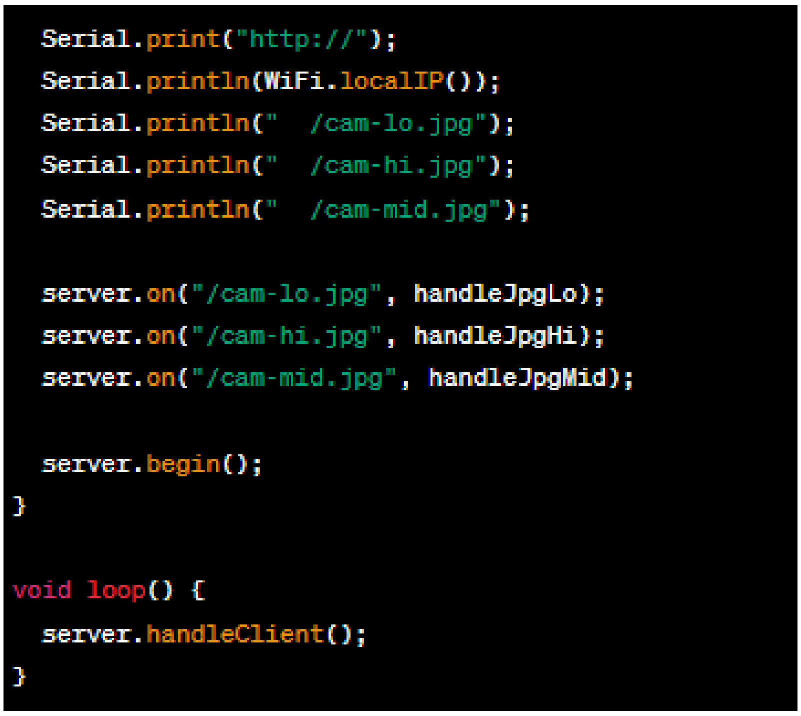


```
void setup() {
  Serial.begin(115200);
  Serial.println();

  {
    using namespace esp32cam;
    Config cfg;
    cfg.setPins(pins::AiThinker);
    cfg.setResolution(hiRes);
    cfg.setBufferCount(2);
    cfg.setJpeg(80);

    bool ok = Camera.begin(cfg);
    Serial.println(ok ? "CAMERA OK" : "CAMERA FAIL");
  }

  WiFi.persistent(false);
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASS);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
  }
```

Figure 3.7: ESP32-Cam initial setup

After that, Wi-Fi is configured to connect to the specified network using the provided credentials. The code waits in a loop until the Wi-Fi connection is established. Once connected, it prints the IP address of the ESP32-CAM module and the available endpoints for accessing the different image resolutions.

The server.on() function sets up the HTTP request handlers for the /cam-lo.jpg, /cam-hi.jpg, and /cam-mid.jpg endpoints, associating them with the corresponding functions handleJpgLo(), handleJpgHi(), and handleJpgMid().(Figure 3.8)



```
  Serial.print("http://");
  Serial.println(WiFi.localIP());
  Serial.println("  /cam-lo.jpg");
  Serial.println("  /cam-hi.jpg");
  Serial.println("  /cam-mid.jpg");

  server.on("/cam-lo.jpg", handleJpgLo);
  server.on("/cam-hi.jpg", handleJpgHi);
  server.on("/cam-mid.jpg", handleJpgMid);

  server.begin();
}

void loop() {
  server.handleClient();
}
```

Figure 3.8: The server.on() function Python code

Finally, the server.begin() function starts the HTTP server, and the loop() function is responsible for handling client requests by calling server.handleClient() repeatedly.

The complete final Arduino code used for the ESP32-Camera live stream capture is at provided as follows :

```
// Include the necessary libraries
#include <Web-Server.h>
#include <WiFi.h>
#include <esp32cam.h>

const char* WIFI_SSID = "fh_1f4130";
const char* WIFI_PASS = "wlane0becf";

Web-Server server(80);


static auto loRes = esp32cam::Resolution::find(320, 240);
static auto midRes = esp32cam::Resolution::find(350, 530);
static auto hiRes = esp32cam::Resolution::find(800, 600);
void serveJpg()
{
  auto frame = esp32cam::capture();
  if (frame == nullptr) {
    Serial.println("CAPTURE FAIL");
    server.send(503, "", "");
    return;
  }
  Serial.printf("CAPTURE OK %dx%d %db\n", frame->getWidth(), frame
     ->getHeight(),
                static_cast<int>(frame->size()));

  server.setContentLength(frame->size());
  server.send(200, "image/jpeg");
  WiFiClient client = server.client();
  frame->writeTo(client);
}

void handleJpgLo()
{
  if (!esp32cam::Camera.changeResolution(loRes)) {
    Serial.println("SET-LO-RES FAIL");
  }
  serveJpg();
}

void handleJpgHi()
{
  if (!esp32cam::Camera.changeResolution(hiRes)) {
    Serial.println("SET-HI-RES FAIL");
  }
  serveJpg();
}

void handleJpgMid()
{
  if (!esp32cam::Camera.changeResolution(midRes)) {
    Serial.println("SET-MID-RES FAIL");
  }
  serveJpg();
}


void  setup(){
  Serial.begin(115200);
  Serial.println();
  {
    using namespace esp32cam;
    Config cfg;
    cfg.setPins(pins::AiThinker);
    cfg.setResolution(hiRes);
    cfg.setBufferCount(2);
    cfg.setJpeg(80);

    bool ok = Camera.begin(cfg);
    Serial.println(ok ? "CAMERA OK" : "CAMERA FAIL");
  }
  WiFi.persistent(false);
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
  }
  Serial.print("http://");
  Serial.println(WiFi.localIP());
  Serial.println("  /cam-lo.jpg");
  Serial.println("  /cam-hi.jpg");
  Serial.println("  /cam-mid.jpg");

  server.on("/cam-lo.jpg", handleJpgLo);
  server.on("/cam-hi.jpg", handleJpgHi);
  server.on("/cam-mid.jpg", handleJpgMid);

  server.begin();
}

void loop()
{
  server.handleClient();
}
```

Listing 3.1: Arduino code

upon executing this code, the following process is set on the ESP32-Camera module :

1. The code includes the necessary libraries for the web server, Wi-Fi connectivity, and the camera module.

2. The Wi-Fi network name (SSID) and password are defined as constants.

3. An instance of the WebServer class is created, which will listen on port 80 for incoming HTTP requests.

4. Three different resolutions for the camera module are defined: low resolution, medium resolution, and high resolution.

5. The serveJpg() function is responsible for capturing an image using the camera module and serving it as a response to an HTTP request. It checks if the capture is successful and sends the appropriate response to the client.

6. Three handler functions (handleJpgLo(), handleJpgHi(), and handleJpgMid()) are defined to handle the HTTP requests for low-resolution, medium-resolution, and high-resolution JPEG images. Each function tries to change the camera resolution

to the specified resolution and calls serveJpg() to capture and serve the image.

7. In the setup() function:

• Serial communication is initialized.

• The camera module is configured with the desired settings (pins, resolution, buffer count, and JPEG quality).

• The camera module is initialized, and the success status is printed to the serial monitor.

• Wi-Fi is configured to work in station mode, and the board attempts to connect to the specified network using the provided SSID and password.

• The local IP address of the board and the available image paths are printed to the serial monitor.

• The server registers the request handlers for the image paths with their corresponding functions.

• The server starts listening for incoming requests. 8. In the loop() function, the server.handleClient() function is called repeatedly to handle incoming client requests.

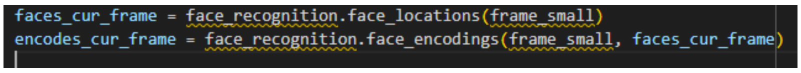
Overall, this code sets up an ESP32-based Arduino board as a web server with a camera module. It connects to a Wi-Fi network, initializes the camera module with the desired settings, and allows clients to request JPEG images in different resolutions. The server captures the images and sends the corresponding responses to the clients.

3.3 Setting up the facial recognition function using Python programming

For the facial recognition process to be included in our ESP32-Cam provided live stream, we are going to use the Python programming language. Our used Python code achieves this process through successfully functioning many tools and libraries that works simultaneously in accordance in order for it to make this possible, and not only that, but many other functionalities that will allows us further customizations and options.

3.3.1 Python code breakdown for facial recognition

In this part, the face recognition library is used for face detection and recognition.(Figure 3.9)

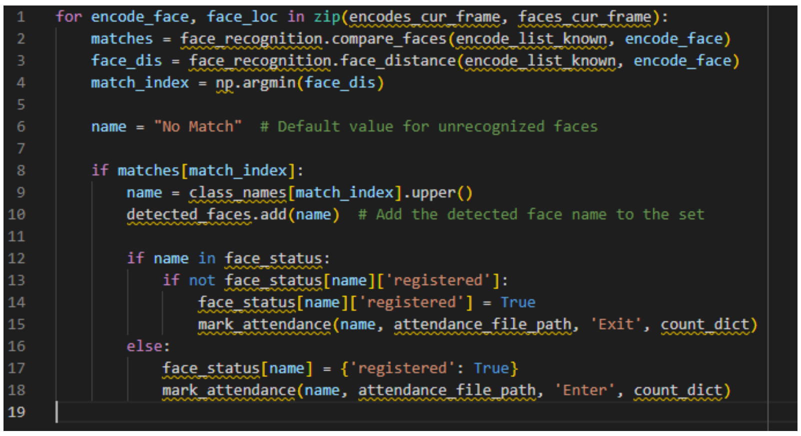


```python
faces_cur_frame = face_recognition.face_locations(frame_small)
encodes_cur_frame = face_recognition.face_encodings(frame_small, faces_cur_frame)
```

Figure 3.9: face recognition library for face detection and recognition

The face locations function is called with the frame small as input. It returns a list of face locations (bounding box coordinates) in the current frame. Each face location consists of four coordinates (top, right, bottom, left), specifying the top-left and bottom-right corners of the bounding box around the face. These face locations are stored in the faces cur frame list. Next, the face encodings function is called with frame small and faces cur frame as inputs. This function calculates the face encodings (a numerical representation of a face) for each face in the frame small. The resulting face encodings are stored in the encodes cur frame list.

This section iterates over each face encoding (encode face) and face location (face loc) in parallel using the zip function.(Figure 3.10)



```python
for encode_face, face_loc in zip(encodes_cur_frame, faces_cur_frame):
    matches = face_recognition.compare_faces(encode_list_known, encode_face)
    face_dis = face_recognition.face_distance(encode_list_known, encode_face)
    match_index = np.argmin(face_dis)

    name = "No Match"  # Default value for unrecognized faces

    if matches[match_index]:
        name = class_names[match_index].upper()
        detected_faces.add(name)  # Add the detected face name to the set

        if name in face_status:
            if not face_status[name]['registered']:
                face_status[name]['registered'] = True
                mark_attendance(name, attendance_file_path, 'Exit', count_dict)
        else:
            face_status[name] = {'registered': True}
            mark_attendance(name, attendance_file_path, 'Enter', count_dict)
```

Figure 3.10: face encoding and location

This section iterates over each face encoding (encode face) and face location (face loc) in parallel using the zip function. For each face, the code compares the face encoding with the list of known face encodings (encode list known) using face recognition.compare faces. This function returns a list of boolean values indicating whether each known face encoding matches the current face encoding. The matches list stores the results. The code also calculates the face distance between the current face encoding and each known face encoding using face recognition.face distance. The face distance represents how similar or dissimilar the current face is to each known face. The face dis list stores the results. The np.argmin function is used to find the index of the smallest face distance in the face dis list. This index corresponds to the best match for the current face among the known faces. The match index variable stores this index. A default name of ”No Match” is assigned to the name variable, indicating that the current face is unrecognized. If matches[match index] is True, it means that the best match for the current face is considered a match. In this case, the name variable is updated with the corresponding class name from the class names list. The class names are assumed to be uppercase strings. The detected face name (name) is added to the detected faces set, indicating that this face has been recognized in the current frame.

The code then checks if the detected face name already exists in the face status dictionary. If it does, it means the face has been registered before. In this case, the code checks if the face is currently not marked as "registered" (face status[name]['registered'] is False). If it is not registered, it updates the registered status to True and calls the mark attendance function with the face name, attendance file path, "Exit" parameter, and count dict as arguments. This function is responsible for marking the attendance of the face. If the detected face name does not exist in the face status dictionary, it means the face is detected for the first time. In this case, a new entry is added to the face status dictionary with the detected face name as the key and a dictionary with a single key-value pair 'registered': True as the value. This indicates that the face is now registered. Overall, this part of the code performs face detection using face locations, face recognition by comparing face encodings with known face encodings, and updates the status of recognized faces in the face status dictionary.

This concludes the Python implementation part of our facial recognition chosen method.

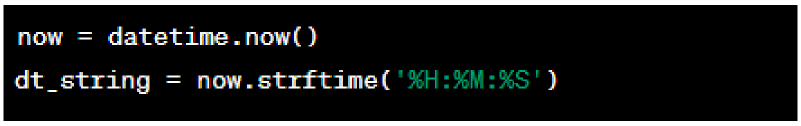
Now for the rest of the code, it doesn't only include the facial recognition process but it also provides multiple necessary functions, one of those functions is the attendance system and it also completely relies on Python programming.

3.3.2 The attendance system Python programming and structure

Marking Attendance

which involves getting the current time, reading the attendance file, creating a new entry DataFrame, checking the status, updating attendance accordingly, and saving the updated attendance back to the file .

- Getting the Current Time.(Figure 3.11)



```
now = datetime.now()
dt_string = now.strftime('%H:%M:%S')
```

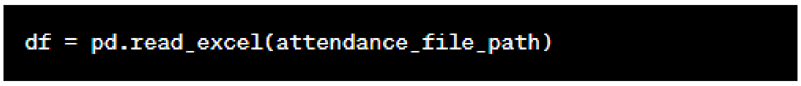
Figure 3.11: Current Time function

- The datetime.now() function retrieves the current date and time.

- The returned value is stored in the now variable, representing the current date and time.

- The strftime('H:M:S') method is used to format the now datetime object into a string with the format "hour:minute:second". - The formatted string is stored in the dt string variable.

• Reading the Attendance File.(Figure 3.12)



```
df = pd.read_excel(attendance_file_path)
```

Figure 3.12: Reading the Attendance File function

- This line reads the existing attendance file specified by attendance file path using the pd.read excel function.

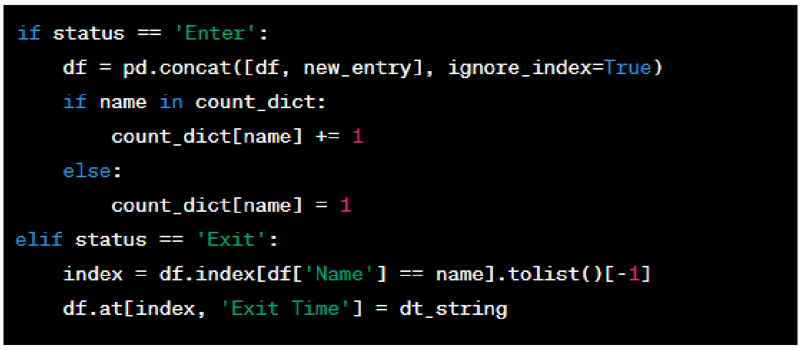
- It stores the contents of the file in a DataFrame named df. The DataFrame will have columns "Name", "Enter Time", and "Exit Time".

• Creating a New Entry DataFrame

This line creates a new DataFrame new entry with a single row containing the name and enter time of the person.

The values are specified as a list [[name, dt string]], and the column names are provided as ["Name", "Enter Time"].

• Checking the Status and Updating Attendance.(Figure 3.13)



```
if status == 'Enter':
    df = pd.concat([df, new_entry], ignore_index=True)
    if name in count_dict:
        count_dict[name] += 1
    else:
        count_dict[name] = 1
elif status == 'Exit':
    index = df.index[df['Name'] == name].tolist()[-1]
    df.at[index, 'Exit Time'] = dt_string
```

Figure 3.13: Status and Attendance Update

- If the status is 'Enter', the code block under the if statement is executed.

- It concatenates the new entry DataFrame with the existing df DataFrame using pd.concat.

- The ignore index=True parameter ensures that the index is reset after concatenation.

- If the person's name already exists in the count dict, it increments the count by 1.

- If the person's name is not present in the count dict, it initializes the count to 1.

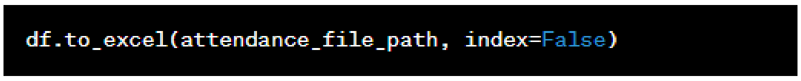
- If the status is 'Exit', the code block under the elif statement is executed.

- It finds the index of the last occurrence of the person's name in the DataFrame using df.index[df['Name'] == name].tolist()[-1].

- The -1 index is used to get the last occurrence of the name if there are multiple entries for the same person.

- It updates the corresponding 'Exit Time' column with the current time using df.at[index, 'Exit Time'] = dt string.

• Saving Updated Attendance.(Figure 3.14)



```
df.to_excel(attendance_file_path, index=False)
```

Figure 3.14: Saving Attendance Update

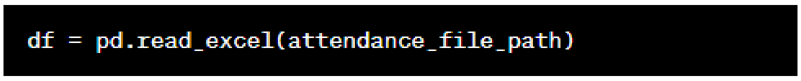
- This line saves the updated DataFrame df to the attendance file specified by attendance file path using df.to excel.

- The index=False parameter ensures that the index is not included in the saved file.

3.3.3 Processing Frames and Face Recognition for Attendance purposes

This step involves reading the attendance file, creating a new entry, updating attendance for 'Enter' and 'Exit' statuses, and saving the updated attendance back to the file.

• Reading the Attendance File.(Figure 3.15)

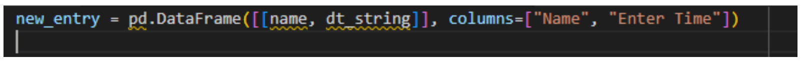


```
df = pd.read_excel(attendance_file_path)
```

Figure 3.15: Attendance File Read

- This line reads the existing attendance file specified by attendance file path using the pd.read-excel function. - It stores the contents of the file in a DataFrame named df. The DataFrame will have columns "Name", "Enter Time", and "Exit Time".

• Creating a New Entry.(Figure 3.16)



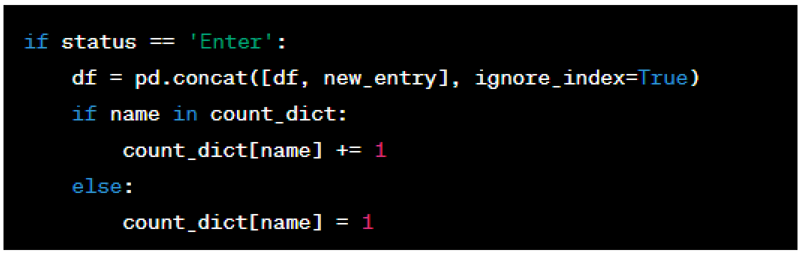
```
new_entry = pd.DataFrame([[name, dt_string]], columns=["Name", "Enter Time"])
```

Figure 3.16: New Entry creation

- This line creates a new DataFrame new-entry with a single row containing the name and enter time of the person. - The values are specified as a list [[name, dt-string]], and the column names are provided as [”Name”, ”Enter Time”].

Updating Attendance for ’Enter’ Status

The Code section(Figure 3.17).



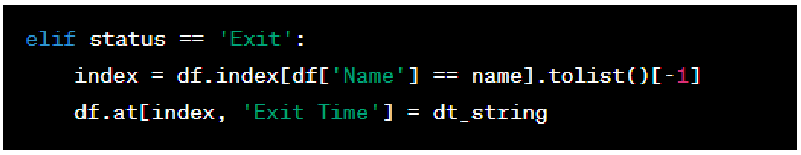
```
if status == 'Enter':
    df = pd.concat([df, new_entry], ignore_index=True)
    if name in count_dict:
        count_dict[name] += 1
    else:
        count_dict[name] = 1
```

Figure 3.17: Attendance for ’Enter’ Status Python code

- This section is executed when the status is ’Enter’. - It concatenates the new entry DataFrame with the existing df DataFrame using pd.concat. - The ignore index=True parameter ensures that the index is reset after concatenation. - If the person’s name already exists in the count dict, it increments the count by 1. - If the person’s name is not present in the count dict, it initializes the count to 1.

Updating Attendance for ’Exit’ Status

The Code section(Figure 3.18).

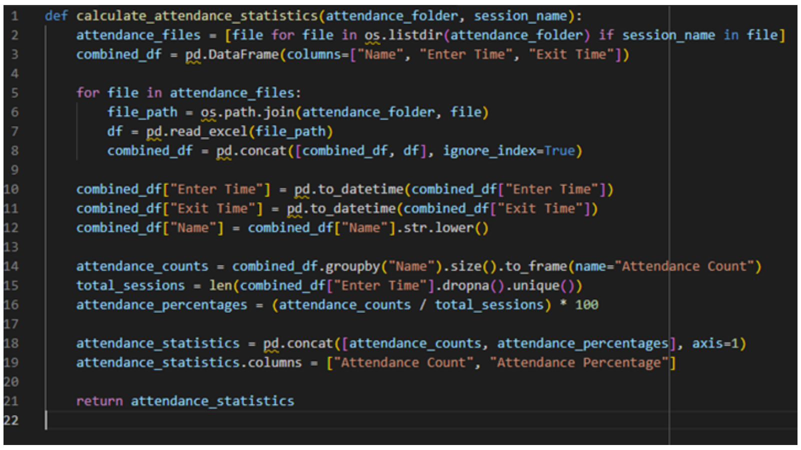


```
elif status == 'Exit':
    index = df.index[df['Name'] == name].tolist()[-1]
    df.at[index, 'Exit Time'] = dt_string
```

Figure 3.18: Attendance for ’Exit’ Status

Calculating Students attendance statistics

The Code section(Figure 3.19).

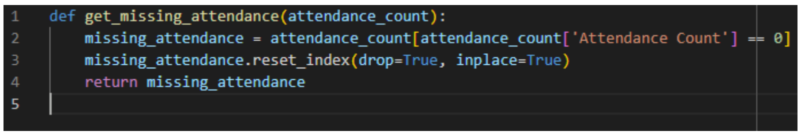


```python
def calculate_attendance_statistics(attendance_folder, session_name):
    attendance_files = [file for file in os.listdir(attendance_folder) if session_name in file]
    combined_df = pd.DataFrame(columns=["Name", "Enter Time", "Exit Time"])

    for file in attendance_files:
        file_path = os.path.join(attendance_folder, file)
        df = pd.read_excel(file_path)
        combined_df = pd.concat([combined_df, df], ignore_index=True)

    combined_df["Enter Time"] = pd.to_datetime(combined_df["Enter Time"])
    combined_df["Exit Time"] = pd.to_datetime(combined_df["Exit Time"])
    combined_df["Name"] = combined_df["Name"].str.lower()

    attendance_counts = combined_df.groupby("Name").size().to_frame(name="Attendance Count")
    total_sessions = len(combined_df["Enter Time"].dropna().unique())
    attendance_percentages = (attendance_counts / total_sessions) * 100

    attendance_statistics = pd.concat([attendance_counts, attendance_percentages], axis=1)
    attendance_statistics.columns = ["Attendance Count", "Attendance Percentage"]

    return attendance_statistics
```

Figure 3.19: Students attendance statistics Python code

The students absence calculation parameters

The Code section(Figure 3.20).



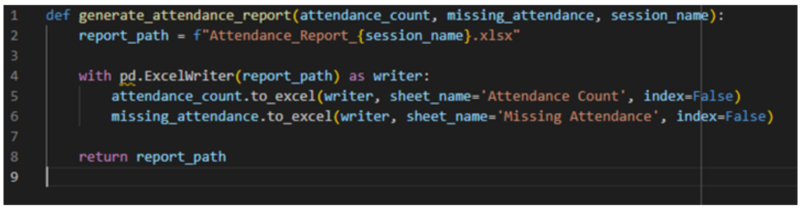
```python
def get_missing_attendance(attendance_count):
    missing_attendance = attendance_count[attendance_count['Attendance Count'] == 0]
    missing_attendance.reset_index(drop=True, inplace=True)
    return missing_attendance
```

Figure 3.20: students absence calculations Python code

- The function takes the attendance-count DataFrame as input. - It filters the attendance-count DataFrame to select rows where the ”Attendance Count” is zero using attendance-count[attendance-count['Attendance= 0]. Count'] = - The index of the resulting DataFrame is reset using reset-index(drop=True, inplace=True) to reassign consecutive indices starting from zero. - Finally, the function returns the missing-attendance DataFrame containing the students with missing attendance records.

Attendance report generating

The Code section(Figure 3.21).



```python
def generate_attendance_report(attendance_count, missing_attendance, session_name):
    report_path = f"Attendance_Report_{session_name}.xlsx"

    with pd.ExcelWriter(report_path) as writer:
        attendance_count.to_excel(writer, sheet_name='Attendance Count', index=False)
        missing_attendance.to_excel(writer, sheet_name='Missing Attendance', index=False)

    return report_path
```

Figure 3.21: Attendance report generating Python code

Explanation:

The generate-attendance-report function is responsible for generating an attendance report containing the attendance count and the list of students with missing attendance records.

- The function takes the attendance-count DataFrame, missing-attendance DataFrame, and session-name as input.

- It defines the path for the attendance report file by concatenating the session name with the file name using f"Attendance-Report-session-name.xlsx".

- A context manager pd.ExcelWriter(report-path) is used to open an Excel writer for writing the report.

- The attendance-count DataFrame is written to the Excel file with the sheet name 'Attendance Count' and without including the index column, using attendance-count.to-excel(writer, sheet-name='Attendance Count', index=False).

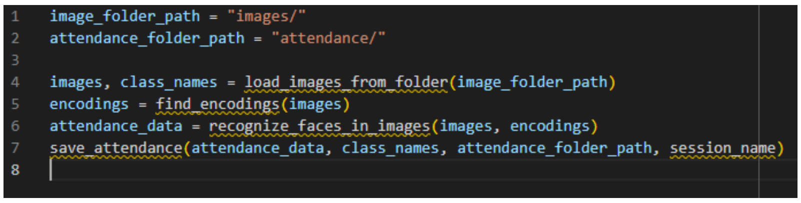
- The missing-attendance DataFrame is written to the Excel file with the sheet name 'Missing Attendance' and without including the index column, using missing-attendance.to-excel(writer, sheet-name='Missing Attendance', index=False).

- The Excel writer is automatically closed when exiting the context manager.

- Finally, the function returns the path of the generated attendance report file.

attendance recognition process

The Code section(Figure 3.22).



```python
image_folder_path = "images/"
attendance_folder_path = "attendance/"

images, class_names = load_images_from_folder(image_folder_path)
encodings = find_encodings(images)
attendance_data = recognize_faces_in_images(images, encodings)
save_attendance(attendance_data, class_names, attendance_folder_path, session_name)
```

Figure 3.22: attendance recognition process Python code

Explanation:

This section of code demonstrates the invocation of the defined functions to perform the attendance recognition process.

- The image-folder-path and attendance-folder-path variables are set to the paths of the image folder and attendance folder, respectively.

- The load-images-from-folder function is called with image-folder-path as an argument to load the images from the specified folder. The returned images and class-names lists are stored in the respective variables.

- The find-encodings function is called with images as an argument to obtain the face encodings for the loaded images. The resulting encodings list is stored in the encodings variable.

- The recognize-faces-in-images function is called with images and encodings as arguments to perform face recognition on the images and retrieve the attendance data. The attendance data is stored in the attendance-data variable.

- The save-attendance function is called with attendance-data, class-names, attendance-folder-path, and session-name as arguments to save the attendance data to the specified attendance folder.

Pie charts function

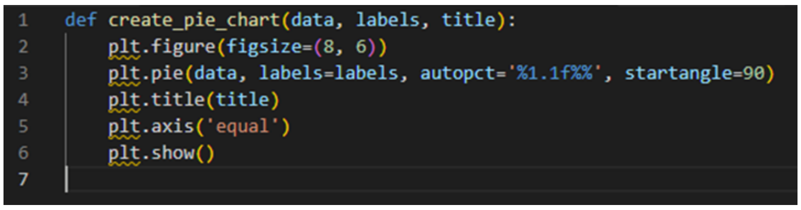
an explanation of the idea behind creating the pie charts and how it works in the code:

In the code, pie charts are created to represent the attendance statistics in a visual format. Pie charts are useful for illustrating the distribution or composition of a whole by dividing it into proportional segments.

Here's an overview of the pie chart creation process in the code:

1. create-pie-chart function :

Section in the code(Figure 3.23).



```python
def create_pie_chart(data, labels, title):
    plt.figure(figsize=(8, 6))
    plt.pie(data, labels=labels, autopct='%1.1f%%', startangle=90)
    plt.title(title)
    plt.axis('equal')
    plt.show()

```

Figure 3.23: pie chart creation Python code

Explanation:

The create-pie-chart function is responsible for creating a pie chart given the data, labels, and title.

• The function takes data, labels, and title as input.

• It sets the figure size using plt.figure(figsize=(8, 6)) to define the dimensions of the chart.

• plt.pie(data, labels=labels, autopct='1.1f', startangle=90) is used to create the pie chart. Here's what each parameter does:

data: The data to be plotted, represented as a list of values.

labels: The labels corresponding to each segment of the pie, represented as a list of strings.

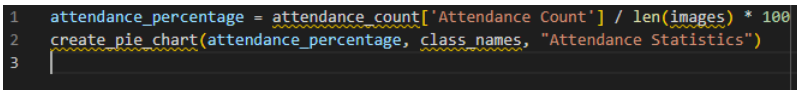
autopct='1.1f': Specifies the format of the percentage values displayed on each pie slice. It formats the values with one decimal place followed by a percentage sign.

startangle=90: Sets the angle at which the first pie slice starts. In this case, it starts from the 90-degree angle (the top).

plt.title(title) sets the title of the pie chart.
plt.axis('equal') ensures that the pie chart is displayed as a circle.
Finally, plt.show() is called to display the pie chart.

Creating pie charts for attendance statistics

Section in the code(Figure 3.24).



```python
attendance_percentage = attendance_count['Attendance Count'] / len(images) * 100
create_pie_chart(attendance_percentage, class_names, "Attendance Statistics")
```

Figure 3.24: pie charts for attendance statistics Python code

Explanation: In this section of code, a pie chart is created to represent the attendance statistics.

• attendance-percentage is calculated by dividing the "Attendance Count" in the attendance-count DataFrame by the total number of images and multiplying it by 100 to obtain the attendance percentage for each class.

• The create-pie-chart function is called with attendance-percentage, class-names, and "Attendance Statistics" as arguments. This generates the pie chart, with attendance-percentage as the data, class-names as the labels, and "Attendance Statistics" as the title.

This process is repeated for each class to create separate pie charts for their attendance statistics.

Overall, the creation of pie charts provides a visual representation of attendance statistics, allowing for an easy understanding of the distribution of attendance across different classes.

3.4 The Complete Python code will look like this

```python
import cv2
import urllib.request
import numpy as np
import os
from datetime import datetime
import face_recognition
import pandas as pd
from pathlib import Path
import plotly.express as px
import plotly.graph_objects as go

def load_images_from_folder(folder_path):
    images = []
    class_names = []
    for file_name in os.listdir(folder_path):
        if file_name.endswith(('.jpg', '.jpeg', '.png')):
            image_path = os.path.join(folder_path, file_name)
            cur_img = cv2.imread(image_path)
            images.append(cur_img)
            class_names.append(os.path.splitext(file_name)[0])
    return images, class_names

def find_encodings(images):
    encode_list = []
    for img in images:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        encode = face_recognition.face_encodings(img)[0]
        encode_list.append(encode)
    return encode_list

def mark_attendance(name, attendance_file_path, status, count_dict)
    :
    now = datetime.now()
    dt_string = now.strftime('%H:%M:%S')

    df = pd.read_excel(attendance_file_path)
    new_entry = pd.DataFrame([[name, dt_string]], columns=["Name",
        "Enter Time"])

    if status == 'Enter':
        df = pd.concat([df, new_entry], ignore_index=True)
        if name in count_dict:
            count_dict[name] += 1
        else:
            count_dict[name] = 1
    elif status == 'Exit':
        index = df.index[df['Name'] == name].tolist()[-1]
        df.at[index, 'Exit Time'] = dt_string

    df.to_excel(attendance_file_path, index=False)



def create_attendance_file(attendance_folder, session_name):
    now = datetime.now()
    date_string = now.strftime('%Y-%m-%d')
    attendance_file_name = f"Attendance_{session_name}_{date_string
        }.xlsx"
    attendance_file_path = os.path.join(attendance_folder, attendance_file_name)

    if os.path.isfile(attendance_file_path):
        return attendance_file_path

    df = pd.DataFrame(columns=["Name", "Enter Time", "Exit Time"])
    df.to_excel(attendance_file_path, index=False)
    return attendance_file_path

def process_frame(frame, encode_list_known, class_names, attendance_file_path, face_status, count_dict):
    try:
        frame_small = cv2.resize(frame, (0, 0), None, 0.25, 0.25)
        frame_small = cv2.cvtColor(frame_small, cv2.COLOR_BGR2RGB)

        faces_cur_frame = face_recognition.face_locations(frame_small)
        encodes_cur_frame = face_recognition.face_encodings(frame_small, faces_cur_frame)

        # Create a set to store the names of the detected faces in the current frame
        detected_faces = set()

        for encode_face, face_loc in zip(encodes_cur_frame, faces_cur_frame):
            matches = face_recognition.compare_faces(encode_list_known, encode_face)
            face_dis = face_recognition.face_distance(encode_list_known, encode_face)
            match_index = np.argmin(face_dis)

            name = "No Match"  # Default value for unrecognized faces

            if matches[match_index]:
                name = class_names[match_index].upper()
                detected_faces.add(name)  # Add the detected face name to the set

                if name in face_status:
                    if not face_status[name]['registered']:
                        face_status[name]['registered'] = True
                        mark_attendance(name, attendance_file_path, 'Exit', count_dict)
                else:
```

```python
                    face_status[name] = {'registered': True}
                    mark_attendance(name, attendance_file_path, 'Enter', count_dict)

            # Draw rectangle around the face
            y1, x2, y2, x1 = face_loc
            y1, x2, y2, x1 = y1 * 4, x2 * 4, y2 * 4, x1 * 4
            cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)

            # Display the name centered below the rectangle
            text_width, text_height = cv2.getTextSize(name, cv2.FONT_HERSHEY_COMPLEX, 1, 2)[0]
            text_x = x1 + (x2 - x1 - text_width) // 2
            text_y = y2 + text_height + 6

            color = (0, 255, 0) if name != 'No Match' else (0, 0, 255)
            cv2.putText(frame, name, (text_x, text_y), cv2.FONT_HERSHEY_COMPLEX, 1, color, 2)

        # Update the registered flag for faces that are no longer detected
        for name, status in list(face_status.items()):
            if status['registered'] and name not in detected_faces:
                status['registered'] = False
                mark_attendance(name, attendance_file_path, 'Exit', count_dict)

    except UnicodeDecodeError:
        pass


def calculate_attendance_statistics(attendance_folder, session_name):
    attendance_files = list(attendance_folder.glob(f"Attendance_{session_name}_*.xlsx"))
    student_attendance = pd.DataFrame(columns=["Name", "Enter Time", "Exit Time"])

    for file_path in attendance_files:
        df = pd.read_excel(file_path)
        student_attendance = pd.concat([student_attendance, df], ignore_index=True)

    student_attendance['Enter Time'] = pd.to_datetime(student_attendance['Enter Time'])
    student_attendance['Exit Time'] = pd.to_datetime(student_attendance['Exit Time'])

    student_attendance['Name'] = student_attendance['Name'].str.lower()  # Convert names to lowercase

    attendance_count = student_attendance.groupby('Name')['Enter Time'].count().reset_index()
    attendance_count.rename(columns={'Enter Time': 'Attendance Count'}, inplace=True)

    total_classes = student_attendance['Enter Time'].nunique()

    all_students = load_images_from_folder(image_folder_path)[1]
    all_students = [student.lower() for student in all_students]  # Convert all_students to lowercase
    missing_students = list(set(all_students) - set(attendance_count['Name']))

    missing_attendance = [{'Name': student, 'Attendance Count': 0} for student in missing_students]
    missing_attendance_df = pd.DataFrame(missing_attendance)

    attendance_count = pd.concat([attendance_count, missing_attendance_df], ignore_index=True)

    attendance_count['Total Classes'] = total_classes
    attendance_count['Attendance Percentage'] = (
        (attendance_count['Attendance Count'] / total_classes) * 100
    ).round(2)

    return attendance_count




def save_attendance_statistics(statistics, file_path):
    statistics.to_excel(file_path, index=False)

def display_attendance_statistics_pie_charts(statistics, save_folder):
    for _, row in statistics.iterrows():
        student = row['Name']
        attendance_count = row['Attendance Count']
        absent_count = row['Total Classes'] - attendance_count

        labels = ['Attendance', 'Absent']
        sizes = [attendance_count, absent_count]
        colors = ['#2ecc71', '#e74c3c']

        fig = px.pie(names=labels, values=sizes, title=f'{student}
            Attendance')
        fig.update_traces(marker=dict(colors=colors))

        chart_save_path = os.path.join(save_folder, f'{student}
            _Attendance_Chart.html')
        fig.write_html(chart_save_path)

def create_main_folder(session_name):
    now = datetime.now()
    folder_name = now.strftime('%Y-%m-%d')
    main_folder = Path(os.getcwd()) / f"{folder_name}_{session_name
        }"

    if main_folder.exists():
        return main_folder

    main_folder.mkdir(parents=True, exist_ok=True)

    return main_folder

def authenticate(username, password):
    # Replace this with your own authentication logic
    valid_username = "user"
    valid_password = "1234"

    return username == valid_username and password ==
        valid_password

session_name = input("Enter the name of the session: ")
main_folder = create_main_folder(session_name)
attendance_folder = main_folder / 'Attendance'
statistics_folder = main_folder / 'Statistics'
pie_chart_save_folder = main_folder / 'PieCharts'

attendance_folder.mkdir(parents=True, exist_ok=True)
statistics_folder.mkdir(parents=True, exist_ok=True)
pie_chart_save_folder.mkdir(parents=True, exist_ok=True)

image_folder_path = Path(r'C:\Users\Administrator\Desktop\TRYattF\
    students faces')
url = 'http://192.168.1.2/cam-hi.jpg'

```

```python
images, class_names = load_images_from_folder(image_folder_path)
encode_list_known = find_encodings(images)
print('Encoding Complete')

attendance_file_path = create_attendance_file(attendance_folder,
    session_name)

face_status = {}

count_dict = {}

# Ask for username and password
username = input("Username: ")
password = input("Password: ")

# Authenticate the user
if authenticate(username, password):
    while True:
        img_arr = np.array(bytearray(urllib.request.urlopen(url).
            read()), dtype=np.uint8)
        frame = cv2.imdecode(img_arr, -1)

        process_frame(frame, encode_list_known, class_names,
            attendance_file_path, face_status, count_dict)

        cv2.imshow('Webcam', frame)
        if cv2.waitKey(1) == 13:  # Press 'Enter' key to exit
            break
else:
    print("Invalid username or password")

cv2.destroyAllWindows()

statistics = calculate_attendance_statistics(attendance_folder,
    session_name)

statistics_file_path = statistics_folder / f'Attendance_Statistics_
    {session_name}.xlsx'

save_attendance_statistics(statistics, statistics_file_path)

display_attendance_statistics_pie_charts(statistics,
    pie_chart_save_folder)
```

Listing 3.2: Python code

upon executing this code, the following functions and operations are set:

1. User Input and Folder Creation: • The user is prompted to enter the name of

the session, which could be a class, meeting, or any other event.

• A main folder is created to store attendance records, statistics, and pie charts for the session. The folder name is based on the session name and the current date.

• Within the main folder, three subfolders are created: "Attendance" to store attendance records, "Statistics" to store attendance statistics, and "PieCharts" to store visualizations.

2. Loading Known Images and Encodings:

• The code specifies the path to the folder containing known images of students (or individuals).

• The images are loaded from the folder, and their face encodings are computed using the face-recognition library.

• The resulting list of face encodings is stored for later use.

3. Creating Attendance File:

• An attendance file is created for the current session using the session name and the current date.

• If an attendance file already exists for the given session, the existing file path is returned; otherwise, a new file is created.

4. Face Recognition and Attendance Tracking:

• The code prompts the user to enter a username and password for authentication purposes.

• If the authentication is successful, the code enters an infinite loop to continuously fetch frames from the webcam stream (provided by the ESP32-CAM).

• Each frame is processed, which involves detecting faces, comparing them with the known face encodings, updating the attendance file, and keeping track of the face status (registered or not registered) and count.

• The processed frame, with face rectangles and names, is displayed in a window.

• The loop terminates when the Enter key is pressed.

5. Calculating and Saving Attendance Statistics:

• After the face recognition loop ends, the attendance statistics are calculated based on the attendance records stored in the "Attendance" folder.

• The statistics include the attendance count, total number of classes, and attendance percentage for each student (or individual).

• The statistics are stored in a DataFrame.

6. Generating and Saving Pie Charts:

• Pie charts are generated to visualize the attendance of each student (or individual) based on the attendance statistics.

• Each pie chart represents the attendance percentage of a student.

• The pie charts are saved as HTML files in the "PieCharts" folder.

Overall, the code sets up a face recognition system using the ESP32-CAM for streaming video, tracks attendance by comparing detected faces with known face encodings, and generates attendance statistics and visualizations for the session.

3.5 Results

Upon successfully enrolling the students faces for this experiment, the desired results are as follows

3.5.1 Students recognition

an example is show in the next figures(Figure 3.25)(Figure 3.26)(Figure 3.27).

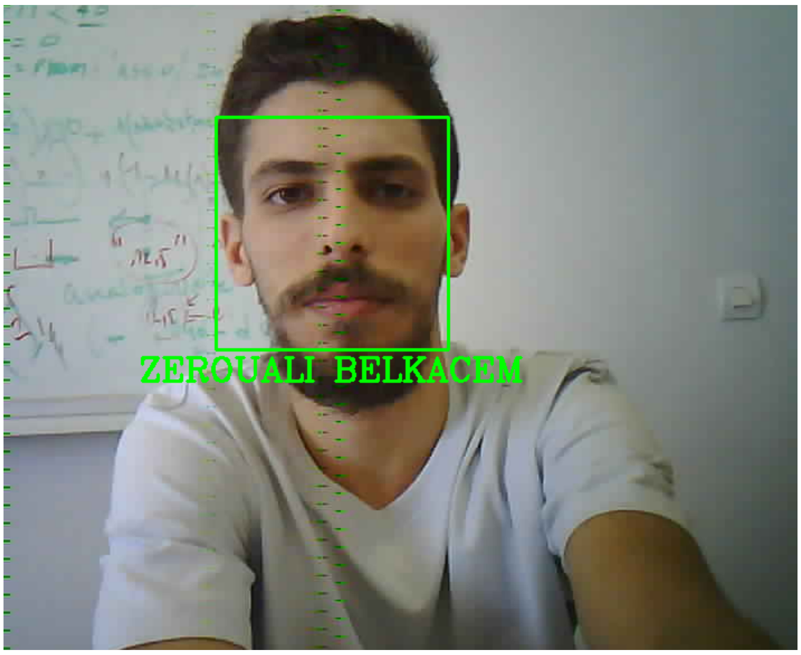


Figure 3.25: Zerouali Belkacem student presence

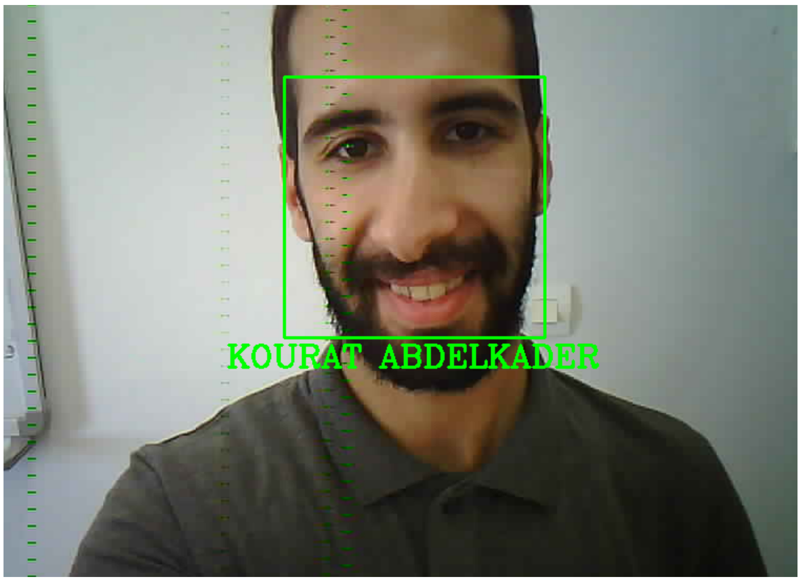


Figure 3.26: Kourat Abdelkader student presence

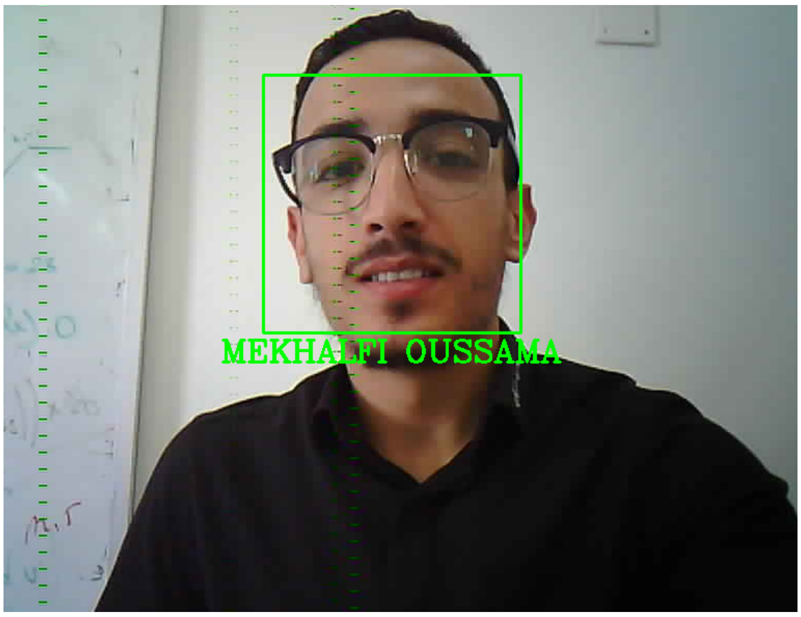
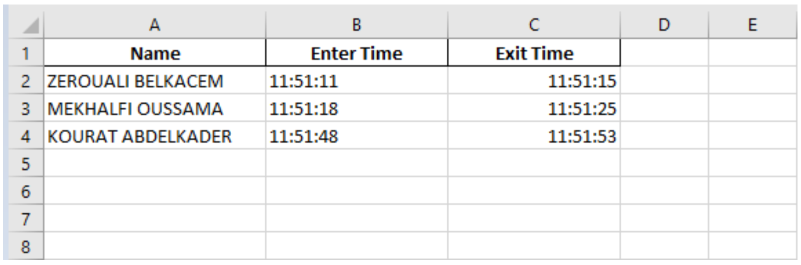


Figure 3.27: Mekhalfi Oussama student presence

3.5.2 Students Attendance Report

This report will contain the time of the student appearance in front of the camera and save these logs in a Excel file. The figures(Figure 3.28) bellows give an example of these students :

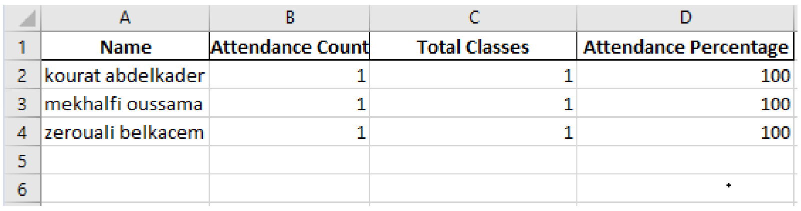


	A	B	C	D	E
1	Name	Enter Time	Exit Time		
2	ZEROUALI BELKACEM	11:51:11	11:51:15		
3	MEKHALFI OUSSAMA	11:51:18	11:51:25		
4	KOURAT ABDELKADER	11:51:48	11:51:53		
5					
6					
7					
8					

Figure 3.28: Students presence log information file

3.5.3 Students statistics

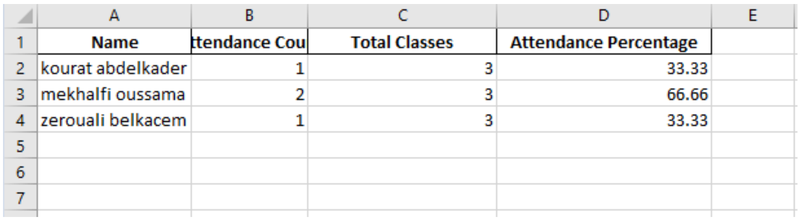
In an Excel file format, students statistics will be calculated and saved. the figure(Figure 3.29) bellow give an example of this :



	A	B	C	D
1	Name	Attendance Count	Total Classes	Attendance Percentage
2	kourat abdelkader	1	1	100
3	mekhalfi oussama	1	1	100
4	zerouali belkacem	1	1	100
5				
6				

Figure 3.29: Students presence statistics information file

Absence is also taken in consideration as shown in the next figure(Figure 3.30):



	A	B	C	D	E
1	Name	ttendance Cou	Total Classes	Attendance Percentage	
2	kourat abdelkader	1	3	33.33	
3	mekhalfi oussama	2	3	66.66	
4	zerouali belkacem	1	3	33.33	
5					
6					
7					

Figure 3.30: Students presence and absence statistics information file

3.5.4 Students Pie charts

There will be a Pie chart for visual representation of each student statistics saved. This figure(Figure 3.31) bellow show an example of a student that has not missed a class in the semester period :

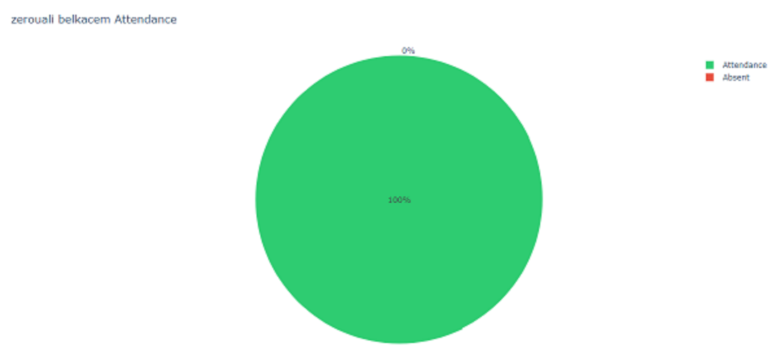


Figure 3.31: Students Pie chart statistics

This figure(Figure 3.32) show if the student has a history of absence in the class :

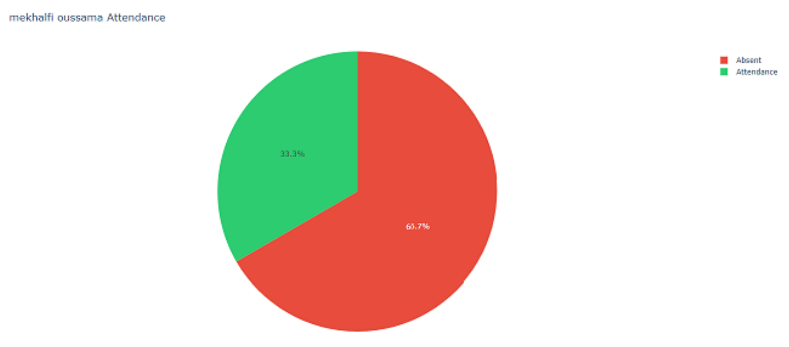


Figure 3.32: Students Pie chart statistics with absence history

3.6 Conclusion

In conclusion, the project combines the capabilities of the Arduino IDE and the Python programming language to create a system for face recognition-based attendance using the ESP32-CAM module.

The Arduino IDE is used to program the ESP32-CAM module, configuring it to capture video and stream it over a local network. The ESP32-CAM module acts as a webcam, providing a continuous video stream that can be accessed by other devices on the same network.

The Python program utilizes the OpenCV and face-recognition libraries to process the video stream from the ESP32-CAM module. It performs face detection and recognition on each frame of the video stream using known face encodings. When a recognized face is detected, the program updates an attendance file, marking the entry and exit times of the individual.

The system also calculates attendance statistics based on the collected data, including the total number of classes conducted and the attendance count for each student. Additionally, it generates pie charts to visualize the attendance percentages of individual students.

By combining the ESP32-CAM's video streaming capabilities with face recognition algorithms in Python, the project provides an automated and efficient solution for tracking attendance. It eliminates the need for manual attendance marking and provides accurate records. The system can be applied in various settings such as classrooms, meetings, or any event where face recognition-based attendance is required.

Conclusion

In conclusion, this thesis aims to develop and evaluate a facial recognition-based attendance system utilizing the ESP32-Camera module for universities. By leveraging the advancements in facial recognition technology and the capabilities of the ESP32-Camera module, the system seeks to automate attendance management, enhance administrative processes, and provide valuable insights for universities. The subsequent chapters will delve into the system's implementation, evaluation, and findings, ultimately contributing to improved attendance tracking in higher education institutions.

References

1. *The Internet of Things with ESP32* http://esp32.net/.
2. Engineers, L. M. *Getting Started With ESP32: A Beginner's Guide* https://lastminuteengineers.com/getting-started-with-esp32/.
3. Deng, J., Trigeorgis, G., Zhou, Y. & Zafeiriou, S. *Joint Multi-View Face Alignment in the Wild* http://arxiv.org/abs/1708.06023.
4. *Facial Recognition History* Thales Group. https://www.thalesgroup.com/en/markets/digital-identity-and-security/government/inspired/history-of-facial-recognition.
5. *Facial Recognition Technology - Innovatrics - How It Works* https://www.innovatrics.com/facial-recognition-technology/.
6. *Welcome to Face Recognition's Documentation! – Face Recognition 1.4.0 Documentation* https://face-recognition.readthedocs.io/en/latest/.
7. *What Is Face Detection and How Does It Work?* Enterprise AI. https://www.techtarget.com/searchenterpriseai/definition/face-detection.
8. *OpenCV: OpenCV-Python Tutorials* https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html.
9. *Urllib – URL Handling Modules* https://docs.python.org/3/library/urllib.html.
10. *NumPy Documentation* https://numpy.org/doc/.
11. *NumPy Quickstart – NumPy v1.24 Manual* https://numpy.org/doc/stable/user/quickstart.html.
12. *Os – Miscellaneous Operating System Interfaces* https://docs.python.org/3/library/os.html.
13. *Datetime – Basic Date and Time Types* https://docs.python.org/3/library/datetime.html.
14. *Getting Started Tutorials – Pandas 2.0.2 Documentation* https://pandas.pydata.org/docs/getting_started/intro_tutorials/.

15. *Pathlib – Object-Oriented Filesystem Paths* https://docs.python.org/3/library/pathlib.html.

16. *Tutorials – Matplotlib 3.7.1 Documentation* https://matplotlib.org/stable/tutorials/index.html.

17. *Users Guide – Matplotlib 3.7.1 Documentation* https://matplotlib.org/stable/users/index.html.

18. *Visual Studio 2022 Community Edition – Download Latest Free Version* https://visualstudio.microsoft.com/vs/community/.

19. *Documentation — CMake* https://cmake.org/documentation/.

20. *CMake Reference Documentation – CMake 3.27.0-Rc1 Documentation* https://cmake.org/cmake/help/latest/.

21. *CMake Tutorial – CMake 3.27.0-Rc1 Documentation* https://cmake.org/cmake/help/latest/guide/tutorial/index.html.

22. *Visual Studio Documentation* https://learn.microsoft.com/en-us/visualstudio/windows/.

23. *Documentation for Visual Studio Code* https://code.visualstudio.com/docs.

24. *Python in Visual Studio Code* https://code.visualstudio.com/docs/languages/python.

25. *GitHub - Ageitgey/Face_recognition: The World's Simplest Facial Recognition API for Python and the Command Line* https://github.com/ageitgey/face_recognition.

26. Legrand. *Déclenchement caméra* https://www.f-legrand.fr/scidoc/docmml/sciphys/arduino/synchrobasler/synchrobasler.html (2023).

27. *Biblatex citation styles* https://fr.overleaf.com/learn/latex/Biblatex_citation_styles (2023).

28. *Zotero — Your personal research assistant* https://www.zotero.org/start (2023).

29. *L298 Dual H-Bridge Motor Driver* https://www.dzduino.com/l298-dual-h-bridge-motor-driver-fr (2023).

30. *CMake Tutorial — CMake* https://cmake.org/cmake-tutorial/.

31. *Visual Studio Product Family Documentation* https://learn.microsoft.com/en-us/visualstudio/.

32. Lin, S.-H. *An Introduction to Face Recognition Technology*

33. *OpenCV Documentation Index* https://docs.opencv.org/.

34. *Pandas Documentation – Pandas 2.0.2 Documentation* https://pandas.pydata.org/docs/.

35. Son, N. T. *et al. Implementing CCTV-Based Attendance Taking Support System Using Deep Face Recognition: A Case Study at FPT Polytechnic College*

Appendix A

Datasheet of Camera OV2640

Advanced Information
Preliminary Datasheet

OV2640 Color CMOS UXGA (2.0 MegaPixel) CameraChip^{TM} with OmniPixel2^{TM} Technology

General Description

The OV2640 CameraChip^{TM} is a low voltage CMOS image sensor that provides the full functionality of a single-chip UXGA (1632x1232) camera and image processor in a small footprint package. The OV2640 provides full-frame, sub-sampled, scaled or windowed 8-bit/10-bit images in a wide range of formats, controlled through the Serial Camera Control Bus (SCCB) interface.

This product has an image array capable of operating at up to 15 frames per second (fps) in UXGA resolution with complete user control over image quality, formatting and output data transfer. All required image processing functions, including exposure control, gamma, white balance, color saturation, hue control, white pixel canceling, noise canceling, and more, are also programmable through the SCCB interface. The OV2640 also includes a compression engine for increased processing power. In addition, OmniVision CameraChips use proprietary sensor technology to improve image quality by reducing or eliminating common lighting/electrical sources of image contamination, such as fixed pattern noise, smearing, etc., to produce a clean, fully stable color image.

Note: The OV2640 uses a lead-free package.

Features

- High sensitivity for low-light operation
- Low operating voltage for embedded portable apps
- Standard SCCB interface
- Output support for Raw RGB, RGB (RGB565/555), GRB422, YUV (422/420) and YCbCr (4:2:2) formats
- Supports image sizes: UXGA, SXGA, SVGA, and any size scaling down from SXGA to 40x30
- VarioPixel® method for sub-sampling
- Automatic image control functions including Automatic Exposure Control (AEC), Automatic Gain Control (AGC), Automatic White Balance (AWB), Automatic Band Filter (ABF), and Automatic Black-Level Calibration (ABLC)
- Image quality controls including color saturation, gamma, sharpness (edge enhancement), lens correction, white pixel canceling, noise canceling, and 50/60 Hz luminance detection
- Line optical black level output capability
- Video or snapshot operation
- Zooming, panning, and windowing functions
- Internal/external frame synchronization
- Variable frame rate control
- Supports LED and flash strobe mode
- Supports scaling
- Supports compression
- Embedded microcontroller

Ordering Information

Product	Package
OV02640-VL9A (Color, lead-free)	38-pin CSP2

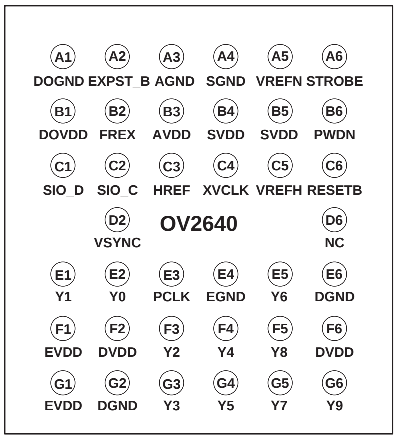
Applications

- Cellular and Camera Phones
- Toys
- PC Multimedia
- Digital Still Cameras

Key Specifications

Array Size	**UXGA**	1600 x 1200
Power Supply	**Core**	1.2VDC ± 5%
	Analog	2.5 ~ 3.0VDC
	I/O	1.7V to 3.3V
Power Requirements	**Active**	125 mW (for 15 fps, UXGA YUV mode)
		140 mW (for 15 fps, UXGA compressed mode)
	Standby	600 µA
Temperature Range	**Operation**	-30°C to 70°C
	Stable Image	0°C to 50°C
Output Formats (8-bit)		• YUV(422/420)/YCbCr422 • RGB565/555 • 8-bit compressed data • 8-/10-bit Raw RGB data
Lens Size		1/4"
Chief Ray Angle		25° non-linear
Maximum Image Transfer Rate	**UXGA/SXGA**	15 fps
	SVGA	30 fps
	CIF	60 fps
Sensitivity		0.6 V/Lux-sec
S/N Ratio		40 dB
Dynamic Range		50 dB
Scan Mode		Progressive
Maximum Exposure Interval		1247 x t_{ROW}
Gamma Correction		Programmable
Pixel Size		2.2 µm x 2.2 µm
Dark Current		15 mV/s at 60°C
Well Capacity		12 Ke
Fixed Pattern Noise		<1% of $V_{PEAK-TO-PEAK}$
Image Area		3590 µm x 2684 µm
Package Dimensions		5725 µm x 6285 µm

Figure 1 OV2640 Pin Diagram (Top View)



Functional Description

Figure 2 shows the functional block diagram of the OV2640 image sensor. The OV2640 includes:

- Image Sensor Array (1632 x 1232 total image array)
- Analog Signal Processor
- 10-Bit A/D Converters
- Digital Signal Processor (DSP)
- Output Formatter
- Compression Engine
- Microcontroller
- SCCB Interface
- Digital Video Port

Figure 2 Functional Block Diagram

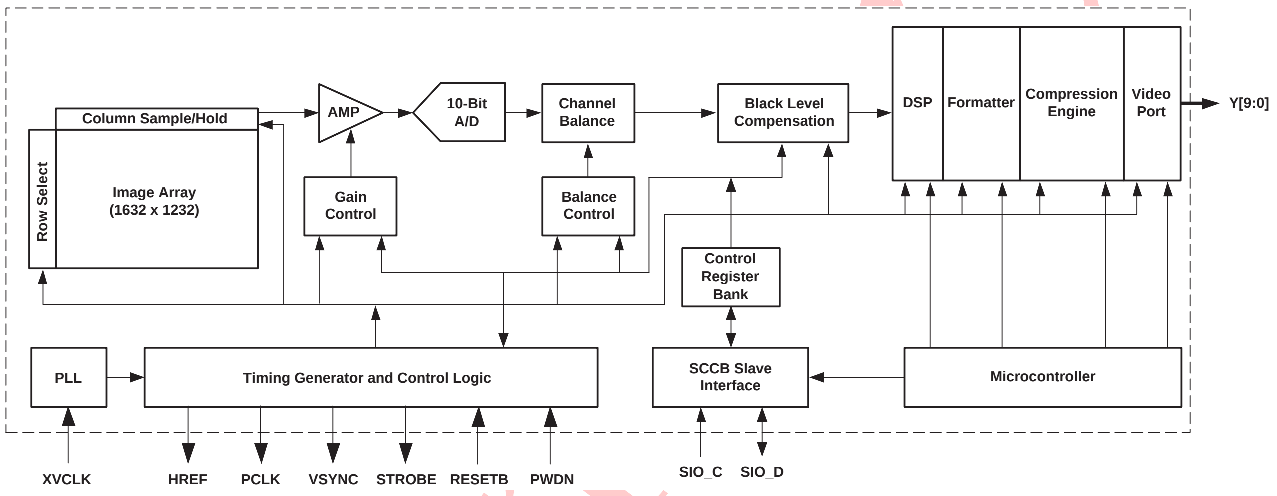
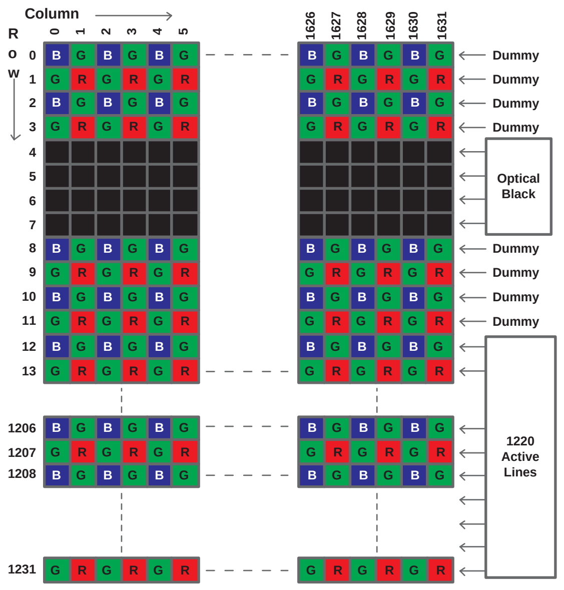


Image Sensor Array

The OV2640 sensor has an image array of 1632 columns by 1232 rows (2,010,624 pixels). Figure 3 shows a cross-section of the image sensor array.

Figure 3 Sensor Array Region Color Filter Layout



The color filters are arranged in a Bayer pattern. The primary color BG/GR array is arranged in line-alternating fashion. Of the 2,010,624 pixels, 1,991,040 (1632x1220) are active. The other pixels are used for black level calibration and interpolation.

The sensor array design is based on a field integration read-out system with line-by-line transfer and an electronic shutter with a synchronous pixel read-out scheme.

Analog Amplifier

When the column sample/hold circuit has sampled one row of pixels, the pixel data will shift out one-by-one into an analog amplifier.

Gain Control

The amplifier gain can either be programmed by the user or controlled by the internal automatic gain control circuit (AGC).

10-Bit A/D Converters

After the analog amplifier, the bayer pattern Raw signal is fed to two 10-bit analog-to-digital (A/D) converters, one for G channel and one shared by the BR channels. These A/D converters operate at speeds up to 20 MHz and are fully synchronous to the pixel rate (actual conversion rate is related to the frame rate).

Channel Balance

The amplified signals are then balanced with a channel balance block. In this block, the Red/Blue channel gain is increased or decreased to match Green channel luminance level.

Balance Control

Channel Balance can be done manually by the user or by the internal automatic white balance (AWB) controller.

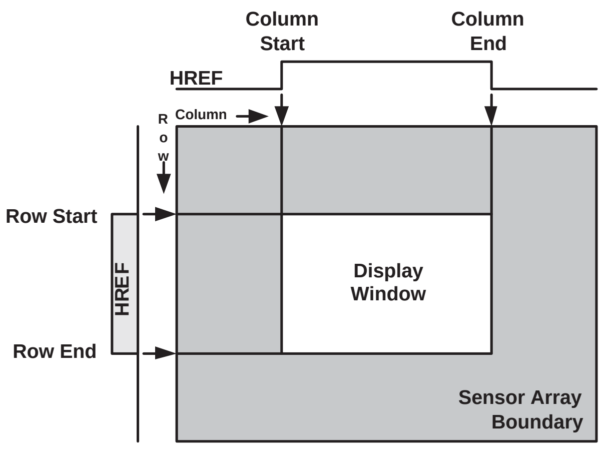
Black Level Compensation

After the pixel data has been digitized, black level calibration can be applied before the data is output. The black level calibration block subtracts the average signal level of optical black pixels to compensate for the dark current in the pixel output. The user can disable black level calibration.

Windowing

The OV2640 allows the user to define window size or region of interest (ROI), as required by the application. Window size setting (in pixels) ranges from 2 x 4 to 1632 x 1220 (UXGA) or 2 x 2 to 818 x 610 (SVGA), and 408 x 304 (CIF), and can be anywhere inside the 1632 x 1220 boundary. Note that modifying window size or window position does not alter the frame or pixel rate. The windowing control merely alters the assertion of the HREF signal to be consistent with the programmed horizontal and vertical ROI. The default window size is 1600 x 1200. Refer to Figure 4 and registers HREFST, HREFEND, REG32, VSTRT, VEND, and COM1 for details.

Figure 4 Windowing



Zooming and Panning Mode

The OV2640 provides zooming and panning modes. The user can select this mode under SVGA/CIF mode timing. The related zoom ratios will be 2:1 of UXGA for SVGA and 4:1 of UXGA for CIF. Registers ZOOMS[7:0] (0x49) and COM19[1:0] (0x48) define the vertical line start point. Register ARCOM2[2] (0x34) defines the horizontal start point.

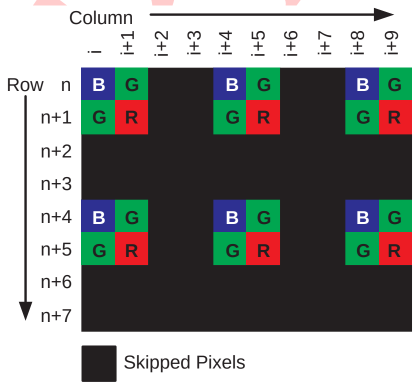
Sub-sampling Mode

The OV2640 supports two sub-sampling modes. Each sub-sampling mode has different resolution and maximum frame rate. These modes are described in the following sections.

SVGA mode

The OV2640 can be programmed to output 800 x 600 (SVGA) sized images for applications where higher resolution image capture is not required. In this mode, both horizontal and vertical pixels will be sub-sampled with an aspect ratio of 4:2 as shown in Figure 5.

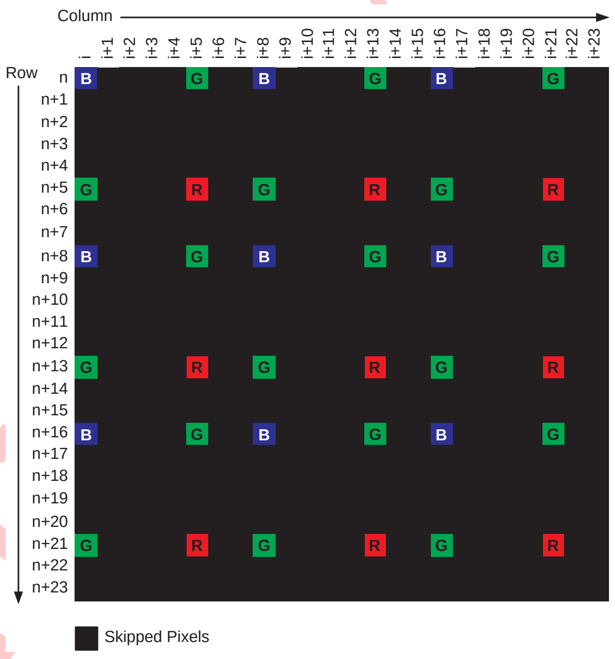
Figure 5 SVGA Sub-Sampling Mode



CIF Mode

The OV2640 can also operate at a higher frame rate to output 400 x 296 sized images. Figure 6 shows the sub-sampling diagram in both horizontal and vertical directions for CIF mode.

Figure 6 CIF Sub-Sampling Mode



Timing Generator and Control Logic

In general, the timing generator controls the following:

- Frame Exposure Mode Timing
- Frame Rate Adjust
- Frame Rate Timing

Frame Exposure Mode Timing

The OV2640 supports frame exposure mode. Typically, the frame exposure mode must work with the aid of an external shutter.

The frame exposure pin, FREX (pin B2), is the frame exposure mode enable pin and the EXPST_B pin (pin A2) serves as the sensor's exposure start trigger. When the external master device asserts the FREX pin high, the sensor array is quickly pre-charged and stays in reset mode until the EXPST_B pin goes low (sensor exposure time can be defined as the period between EXPST_B low and shutter close). After the FREX pin is pulled low, the video data stream is then clocked to the output port in a line-by-line manner. After completing one frame of data output, the OV2640 will output continuous live video data unless in single frame transfer mode. Figure 18 and Figure 19 show the detailed timing and Table 11 shows the timing specifications for this mode.

Frame Rate Adjust

The OV2640 offers three methods for frame rate adjustment:

- Clock prescaler: (see "CLKRC" on page 23)
  By changing the system clock divide ratio and PLL, the frame rate and pixel rate will change together. This method can be used for dividing the frame/pixel rate by: 1/2, 1/3, 1/4 … 1/64 of the input clock rate.
- Line adjustment: (see "REG2A" on page 26 and "FRARL" on page 26)
  By adding a dummy pixel timing in each line (between HSYNC and pixel data out), the frame rate can be changed while leaving the pixel rate as is.
- Vertical sync adjustment:
  By adding dummy line periods to the vertical sync period (see "ADDVSL" on page 26 and "ADDVSH" on page 26 or see "FLL" on page 27 and "FLH" on page 27), the frame rate can be altered while the pixel rate remains the same.

Frame Rate Timing

Default frame timing is illustrated in Figure 15, Figure 16, and Figure 17. Refer to Table 1 for the actual pixel rate at different frame rates.

Table 1 Frame/Pixel Rates in UXGA Mode

Frame Rate (fps)	15	7.5	2.5	1.25
PCLK (MHz)	36	18	6	3

Digital Signal Processor (DSP)

This block controls the interpolation from Raw data to RGB and some image quality control.

- Edge enhancement (a two-dimensional high pass filter)
- Color space converter (can change Raw data to RGB or YUV/YCbCr)
- RGB matrix to eliminate color cross talk
- Hue and saturation control
- Programmable gamma control
- Transfer 10-bit data to 8-bit
- White pixel canceling
- De-noise

Output Formatter

This block controls all output and data formatting required prior to sending the image out.

Scaling Image Output

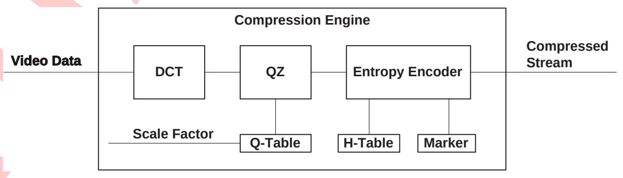
The OV2640 is capable of scaling down the image size from CIF to 40x30. By using SCCB registers, the user can output the desired image size. At certain image sizes, HREF is not consistent in a frame.

Compression Engine

As shown in Figure 7, the Compression Engine consists of three major blocks:

- DCT
- QZ
- Entropy Encoder

Figure 7 Compression Engine Block Diagram



Microcontroller

The OV2640 embeds an 8-bit microcontroller with 512-byte data memory and 4 KB program memory. It provides the flexibility of decoding protocol commands from the host for controlling the system, as well as the ability to fine tune image quality.

SCCB Interface

The Serial Camera Control Bus (SCCB) interface controls the CameraChip operation. Refer to *OmniVision Technologies Serial Camera Control Bus (SCCB) Specification* for detailed usage of the serial control port.

Slave Operation Mode

The OV2640 can be programmed to operate in slave mode (default is master mode).

When used as a slave device, COM7[3] (0x12), CLKRC[6] (0x11), and COM2[2] (0x09) register bits should be set to