A project report on

OBJECT IDENTIFICATION & MOTION DETECTION USING DEEP LEARNING

submitted to

MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY

In fulfillment of the academic requirements for the awards of final semester project, Paper code EC-881 in

Bachelor of Technology in Electronics & Communication Engineering

by

MAINAK BHATTACHARJEE____________________________16900320076

SAYAN PACHHAL______________________________________16900320087

KOUSTAV MAJI_______________________________________16900320088

SARBADRITA BHATTACHARJEE________________________16900320091

ABIR BHOWMICK______________________________________16901620026

Under the guidance of

PROF. BASAB KUMAR CHATTERJEE

Department Of Electronics and Communication Engineering

Academy of Technology

Adisaptagram, Hooghly-712121, West Bengal

CERTIFICATE

This is to certify that the project progress report entitled Object Identification and Motion Detection Using Deep Learning is submitted to MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY in the fulfilment of the requirement for the award of the B. TECH degree in ELECTRONICS AND COMMUNICATION ENGINEERING is original work carried out by

Name	**University roll**
MAINAK BHATTACHARJEE	16900320076
SAYAN PACHHAL	16900320087
KOUSTAV MAJI	16900320088
SARBADRITA BHATTACHARJEE	16900320091
ABIR BHOWMICK	16901620026

Under my guidance, the matter embodied in this project progress report is genuine work done by the student. It has not been submitted whether to this University or to any other University/Institute for the fulfilment of the requirement of any course of study.

Prof. Basab Kumar Chatterjee
Department Of Electronics and
Communication Engineering,
Academy of Technology, Aedconagar
Hooghly -712121, West Bengal, India
Date:

Prof. Abhijit Banerjee
Head Of Department,
Department Of Electronics and
Communication Engineering
Academy of Technology. Aedconagar,
Hooghly-712121, West Bengal, India
Date:

ACKNOWLEDGEMENT

We would like to thank our mentor Prof. Basab Kumar Chatterjee for his continues support & guidance for this project. He really explained all the things in such a simple & prominent way & that has become very helpful throughout this project.

- **Signature of the Group Members:**

1.	___________________________	ABIR BHOWMICK (T026)
2.	___________________________	SAYAN PACHHAL (087)
3.	___________________________	SARBADRITA BHATTACHARJEE (091)
4.	___________________________	MAINAK BHATTACHARJEE (076)
5.	___________________________	KOUSTAV MAJI (088)

____

Signature Of Mentor
(Prof. Basab Kumar Chatterjee)

ABSTRACT

Object identification and motion detection in dynamic scenes, especially for humans and vehicles, is one of the current challenging research topics in computer vision. It is a key technology to fight against terrorism, crime, public safety and for efficient management of traffic. In any kind of video surveillance, detection of moving objects from a video is important for object classification, target tracking, activity recognition, and behaviour understanding. Detection of moving objects in video streams is the first relevant step of information and background subtraction is a very popular approach for foreground segmentation.

In this project we will be using highly accurate object detection-algorithms and highly accurate methods like SSD and YOLO. Using these methods and algorithms, based on deep learning which is also based on machine learning require lots of mathematical and deep learning frameworks understanding by using dependencies such as OpenCV and OpenCV library for implementing our project. OpenCV is a huge open-source library for computer vision, machine learning, and image processing. OpenCV supports a wide variety of programming languages like Python, C++, Java, etc. It can process images and videos to identify objects, faces, or even the handwriting of a human.

The aim of this paper is to propose an artificial intelligence-based approach to moving object detection and tracking. Specifically, we adopt an approach to moving object detection based on self-organization through artificial neural networks. Such approach allows to handle scenes containing moving backgrounds and gradual illumination variations, and achieves robust detection for different types of videos taken with stationary cameras. Object identification and motion detection is widely used for face detection, vehicle detection, pedestrian counting, security systems, video surveillance systems and self-driving cars.

CONTENTS

LIST OF FIGURES

Figure Page

LIST OF TABLES

Chapter 1:
INTRODUCTION

1. INTRODUCTION

A few years ago, the creation of the software and hardware image processing systems was mainly limited to the development of the user interface, which most of the programmers of each firm were engaged in. The situation has been significantly changed with the advent of the Windows operating system when the majority of the developers switched to solving the problems of image processing itself. However, this has not yet led to the cardinal progress in solving typical tasks of recognizing faces, car numbers, road signs, analysing remote and medical images, etc. Each of these "eternal" problems is solved by trial and error by the efforts of numerous groups of the engineers and scientists. As modern technical solutions are turn out to be excessively expensive, the task of automating the creation of the software tools for solving intellectual problems is formulated and intensively solved abroad. In the field of image processing, the required tool kit should be supporting the analysis and recognition of images of previously unknown content and ensure the effective development of applications by ordinary programmers. Just as the Windows toolkit supports the creation of interfaces for solving various applied problems.

Object recognition is to describe a collection of related computer vision tasks that involve activities like identifying objects in digital photographs. Image classification involves activities such as predicting the class of one object in an image. Object localization is referring to identifying the location of one or more objects in an image and drawing an abounding box around their extent. Object detection does the work of combines these two tasks and localizes and classifies one or more objects in an image. When a user or practitioner refers to the term “object recognition”, they often mean “object detection”. It may be challenging for beginners to distinguish between different related computer vision tasks.

So, we can distinguish between these three computer vision tasks with this example:

Image Classification: This is done by Predict the type or class of an object in an image.

Input: An image which consists of a single object, such as a photograph.
Output: A class label (Ex: one or more integers that are mapped to class labels).

Object Localization: This is done through, Locate the presence of objects in an image and indicate their location with a bounding box.

Input: An image which consists of one or more objects, such as a photograph.

Output: One or more bounding boxes (e.g. defined by a point, width, and height).

Object Detection: This is done through, Locate the presence of objects with a bounding box and types or classes of the located objects in an image.

Input: An image which consists of one or more objects, such as a photograph.
Output: One or more bounding boxes (e.g. defined by a point, width, and height), and a class label for each bounding box.

One of the further extensions to this breakdown of computer vision tasks is object segmentation, also called "object instance segmentation" or "semantic segmentation," where instances of recognized objects are indicated by highlighting the specific pixels of the object instead of a coarse bounding box. From this breakdown, we can understand that object recognition refers to a suite of challenging computer vision tasks. For example, image classification is simply straight forward, but the differences between object localization and object detection can be confusing, especially when all three tasks may be just as equally referred to as object recognition.

Humans can detect and identify objects present in an image. The human visual system is fast and accurate and can also perform complex tasks like identifying multiple objects and detect obstacles with little conscious thought. The availability of large sets of data, faster GPUs, and better algorithms, we can now easily train computers to detect and classify multiple objects within an image with high accuracy. We need to understand terms such as object detection, object localization, loss function for object detection and localization, and finally explore an object detection algorithm known as "You only look once" (YOLO).

Image classification also involves assigning a class label to an image, whereas object localization involves drawing a bounding box around one or more objects in an image. Object detection is always more challenging and combines these two tasks and draws a bounding box around each object of interest in the image and assigns them a class label. Together, all these problems are referred to as object recognition.

Object recognition refers to a collection of related tasks for identifying objects in digital photographs. Region-based Convolutional Neural Networks, or R-CNNs, is a family of techniques for addressing object localization and recognition tasks, designed for model performance. You Only Look Once, or YOLO is known as the second family of techniques for object recognition designed for speed and real-time use.

Chapter 2:
REVIEW OF PREVIOUS WORK

2. REVIEW OF PREVIOUS WORK

As we know with increasing population there is increase in thefts and robbery and illegal trespassing, there is a need of some kind of security mechanism that can safeguard us from such acts.

So, if someone could keep an eye on all of our belongings then it could solve the matter, but such ideas are far from practical implementations. But since we are living in an era of digital revolution, we can use security cameras to do such kind of activity.

All the existing security surveillance camera works on only providing the visual of the place where it is installed. So, someone has to monitor the feed all the time. But our device will provide the feed as well as the warning mail & will trigger the warning alarm if any movement happens at the proctored area.

So basically, we are combining motion detection and object identification as a unit such that it cannot only detect motion in its vicinity but can also detect the type of the object, that is what kind object it is. Example cat, dog, person etc.

Our device will also try to identify what kind of the object is moving & that will be an add-on benefit.

Now there will be a question that we are providing these facilities so the device will be costly. But the answer will be ‘No’. We are showing the price comparison of a highly used & user recommended surveillance camera with our device –

2.1 Comparison between existing and our product:

Godrej Eve NX Cube Security Wi-Fi Camera	Our Device
1. For Indoor Security Camera Use 2. Night Vision Feature 3. Smart Motion Detection 4. SD Card Capacity: 128 GB 5. HD Quality Video 6. Smartphone Viewing 7. Supports Onvif	1. For Indoor Security Camera Use 2. Night Vision Feature 3. Motion Detection 4. Object Identification 5. External Storage can be added 6. Less power consumption 7. Instant mail alert

Fig 2.1: Existing Market product Godrej Eve NX Cube [Courtesy: https://www.godrej.com/Resources/Support/EVENX Cube_22072315.pdf]

- Regular Price: **₹2,399.00**
- Sale Price: **₹1,699.00**

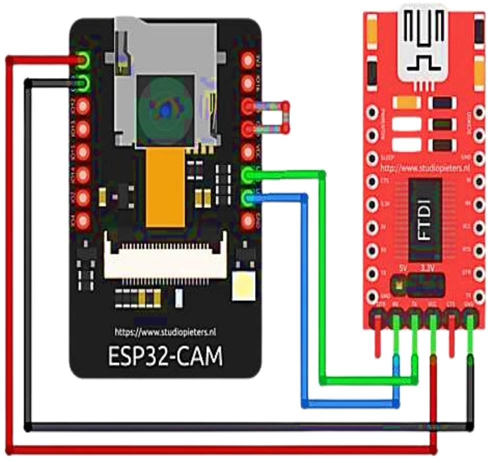


Fig 2.2: Our Device [Courtesy: hackster.io]

- ESP32 CAM module: **₹329.00**
- FTDI: **₹287.00**
- USB Micro-B: **₹149.00**
- Jumper wires: **₹119.00**

Estimated Total Cost: ₹884

Chapter 3:
THEORETICAL BACKGROUND

3. THEORITICAL BACKGROUND

Object identification and motion sensing are two key technologies that have a wide range of applications in fields such as robotics, security, and surveillance. Object identification involves determining the type of object that is present in a given image or video sequence, while motion sensing involves detecting and tracking moving objects.

There are a variety of different approaches to object identification and motion sensing, each with its own advantages and disadvantages. Some common approaches include:

- **Vision-based object identification:** This approach uses cameras to capture images or video sequences, which are then processed by computer vision algorithms to identify objects. Vision-based object identification is a powerful technique that can be used to identify a wide range of objects, but it can be computationally expensive and may not be reliable in all environments.

- **Sensor-based object identification:** This approach uses sensors such as ultrasonic sensors, infrared sensors, and radar sensors to detect objects. Sensor-based object identification is a robust technique that can be used in a variety of environments, but it may not be able to identify objects as accurately as vision-based object identification.

- **Motion sensing:** This approach uses sensors such as cameras, radar sensors, and ultrasonic sensors to detect and track moving objects. Motion sensing can be used in a variety of applications, such as security systems, traffic monitoring systems, and robotics.

3.1 Motion detection Algorithms:

3.1.1 Background Subtraction:

- This method detects motion by comparing each pixel in the current frame with a reference background image . If the pixel value significantly deviates from the background, it is considered part of the moving object.
- Background subtraction is widely used due to its simplicity and effectiveness. However, it can be sensitive to lighting changes and shadows.

3.1.2 **Frame Differencing**:

- Frame differencing involves subtracting the current frame from the previous one. The resulting difference image highlights areas with motion.
- It is a straightforward technique but may suffer from noise and false positives.

3.1.3 **Temporal Differencing**:

- Temporal differencing compares consecutive frames by subtracting pixel values. If the difference exceeds a threshold, it indicates motion.
- This method is robust against gradual illumination changes.

3.1.4 **Optical Flow**:

- Optical flow algorithms estimate the motion of pixels between frames. They compute the displacement vectors for each pixel, representing the direction and magnitude of motion .
- Lucas-Kanade and Horn-Schunck are popular optical flow methods.

3.2 Object Identification Algorithms:

3.2.1 Image Classification: This task involves predicting the class of an object in an image. For instance, given an image of a cat, an image classifier would label it as “cat.”

3.2.2 Object Localization: In this task, we identify the location of one or more objects in an image and draw bounding boxes around them. The bounding boxes indicate the extent of the objects. For example, if there are multiple cats in an image, object localization would find their positions.

3.2.3 Object Detection: Object detection combines both image classification and object localization. It localizes and classifies one or more objects in an image. So, if there are multiple cats in an image, an object detection algorithm would draw bounding boxes around each cat and assign class labels to them.

3.2.4 Region-Based Convolutional Neural Networks (R-CNNs): R-CNNs are designed for model performance. They address object localization and recognition tasks. These models use a region proposal network to identify potential object regions, followed by fine-tuning using a CNN. R-CNNs have been influential in advancing object recognition tasks.

3.2.5 You Only Look Once (YOLO): YOLO is designed for speed and real-time use. Unlike R-CNNs, YOLO processes the entire image in one pass, predicting bounding boxes and class labels simultaneously. It's efficient and widely used in applications like real-time object detection in videos and surveillance systems.

3.3 Literature Survey:

There has been a significant amount of research on object identification and motion sensing in recent years. Some notable advances include:

3.3.1 Deep learning-based object identification: Deep learning algorithms have been shown to be very effective for object identification tasks. Deep learning-based object identification algorithms can learn complex patterns in data, which allows them to identify objects even in challenging environments.

3.3.2 3D object identification: 3D object identification is a challenging problem, but there has been significant progress in recent years. 3D object identification algorithms can now be used to identify objects in real-time from video sequences.

3.3.3 Motion sensing with low-cost sensors: Researchers have developed new motion sensing algorithms that can be used with low-cost sensors such as cameras and ultrasonic sensors. This has made it possible to develop affordable motion sensing systems for a wide range of applications.

3.3.4 UV and IR Motion sensing: Motion sensors and object detectors sometimes fail to act due to poor or no lighting (illumination conditions). Hence researchers went beyond the visibility spectrum with invention of ultraviolet and infrared sensors.

3.3.5 YOLOv8: The YOLO models are famous for two main reasons: their impressive speed and accuracy and their ability to detect objects in images quickly and dependably. YOLOv8 improved performance using a multi-scale feature extraction architecture for better object detection across different scales. As the YOLO framework advanced, the tradeoff between speed and accuracy became more refined.

3.4 Applications:

Object identification and motion sensing have a wide range of applications in fields such as robotics, security, and surveillance. Some specific examples include:

- **Robotics:** Object identification and motion sensing are essential for robots to be able to navigate and interact with their environment. For example, robots can use object identification to identify objects that they need to pick up and place, and they can use motion sensing to avoid obstacles.
- **Security:** Object identification and motion sensing are used in security systems to detect intruders and other potential threats. For example, security cameras can use object identification to identify people and vehicles, and they can use motion sensing to detect people moving around in restricted areas.
- **Surveillance:** Object identification and motion sensing are used in surveillance systems to monitor people and activities. For example, surveillance cameras can use object identification to identify people of interest, and they can use motion sensing to track people's movements.
- **Pedestrian Detection**: Identifying people in images or video streams, crucial for surveillance and safety.
- **Animal Detection**: Detecting animals for wildlife monitoring or conservation efforts.
- **Vehicle Detection**: Used in traffic management, parking systems, and autonomous vehicles.
- **People Counting**: Tracking the number of people in a given area.
- **Face Detection**: Recognizing faces in photos or videos.
- **Text Detection**: Locating and extracting text from images.
- **Pose Detection**: Estimating human body poses.
- **Number Plate Recognition**: Identifying license plates for security or parking systems.
- **Intruder Alarms**: Motion-detection devices like PIR sensors can trigger alarms when unauthorized entry is detected.
- **Automatic Ticket Gates**: These gates use motion detection to allow entry or exit.
- **Entryway Lighting**: Motion sensors can turn on lights when someone enters an area.
- **Security Lighting**: Outdoor lights can be activated based on detected motion.

Chapter 4:
DESIGN

4. DESIGN

The aim of this project is to design and implement a system that can identify and track objects in real-time using a camera and a motion sensor. The system will be able to recognize different types of objects, such as humans, animals, vehicles, etc., and monitor their movements and positions in the environment. The system will also be able to alert the user if any abnormal or suspicious activity is detected, such as intrusions, thefts, accidents, etc.

Our project will go through the following stages:

- Connect ESP32 to FTDI and then to a laptop/desktop using a USB cable. This step establishes the physical connection between your ESP32 and your development environment.
- Write the script in the Arduino IDE for programming the Controller Board (ESP32) and to obtain its IPV4 address in URL form.
- Develop a Python script in any IDE (like PyCharm, VS code etc.) for object identification and motion detection which will use that URL obtained from the Arduino IDE to communicate with ESP32.
- Deploy and test the system in a real-world environment where you want to monitor for movement.
- Implement an alert system in your Python script to send an email notification whenever motion is detected. The email includes e the name of detected object that creating the movement.

4.1 Hardware Requirements:

Here is the list of Hardware Component required for the project:

4.1.1 ESP-32 Camera Module: The ESP32-CAM is a versatile microcontroller module integrating an ESP32 chip and a camera. It combines the power of the ESP32, which offers Wi-Fi and Bluetooth connectivity along with ample processing power, with a camera module capable of capturing images and video. With its small form factor and low power consumption, it's suitable for various IoT and DIY projects, including home automation, surveillance systems, and even robotics. The ESP32-CAM can be programmed using the Arduino IDE, making it accessible to a wide range of developers and hobbyists. Its built-in microSD card slot allows for local storage of images and video footage, while its GPIO pins enable interfacing with other sensors and peripherals. Overall, the ESP32-CAM provides an affordable and compact solution for projects requiring both wireless connectivity and camera capabilities.

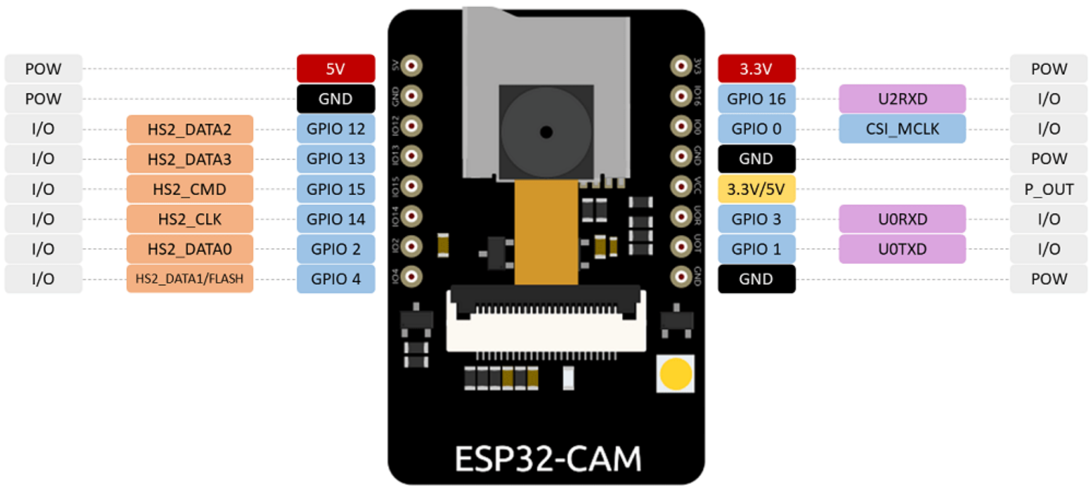


Fig 4.1.1 ESP32-CAM [Courtesy: https://www.espressif.com/]

The functional block diagram of the SoC is shown below:

Block Diagram:

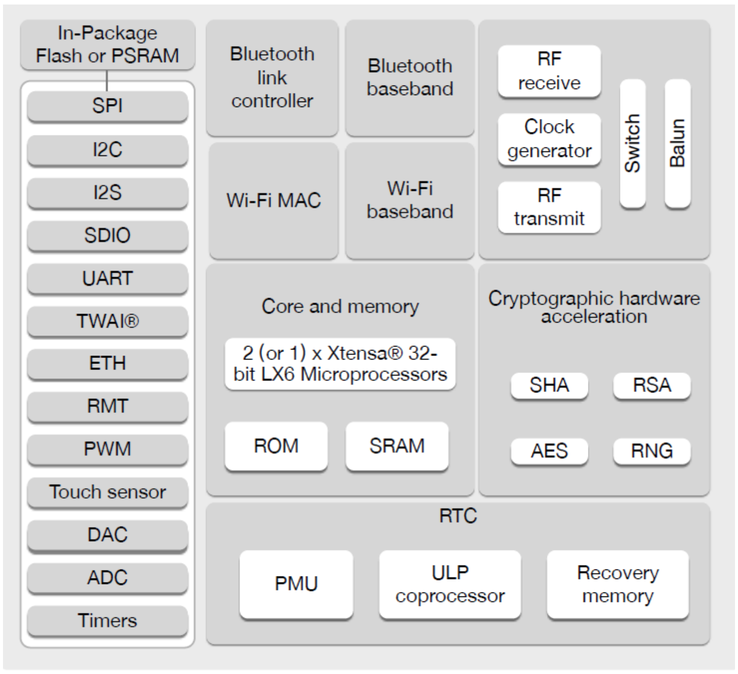


Fig 4.1.2 ESP32 Block Diagram [Courtesy: https://www.espressif.com/]

Pin Diagram:

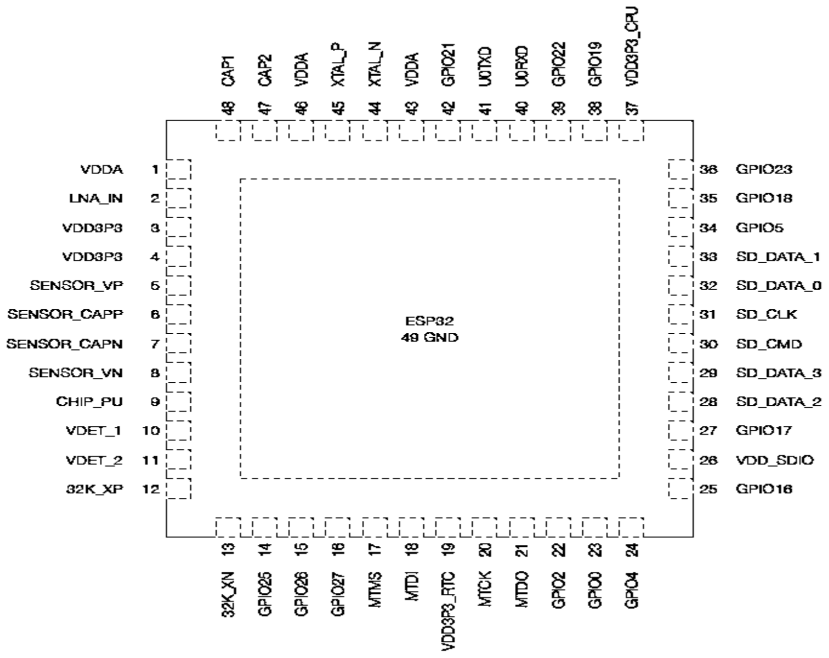


Fig 4.1.3 ESP32 Pin layout [Courtesy: https://www.espressif.com/]

4.1.2 FTDI: FTDI majorly focuses on USB technology, and the USB to TTL interface is their most popular product. These are available as cable as well as a module (like a chip). These products are widely used as an interface in microcontroller development boards like Arduino, ESP-01s, etc. because they require a USB interface. The FTDI interface is the easiest solution to connect the devices with the TTL level interface to USB since it does not require any additional software installation in advance. In simple words, the FTDI adapter module is a complete package in which the FTDI chip is integrated with connectors, voltage regulators, Tx/Rx, and other breakout points. The module thus falls under the category of UART board and is mostly used for TTL serial communication.

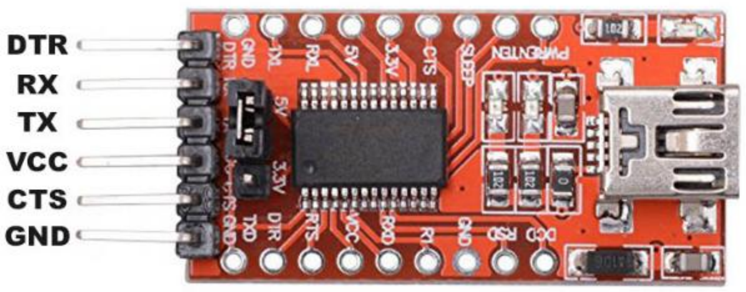


Fig 4.1.4 FTDI [Courtesy: https://www.espressif.com/]

FTDI & ESP-32 CONNECTION

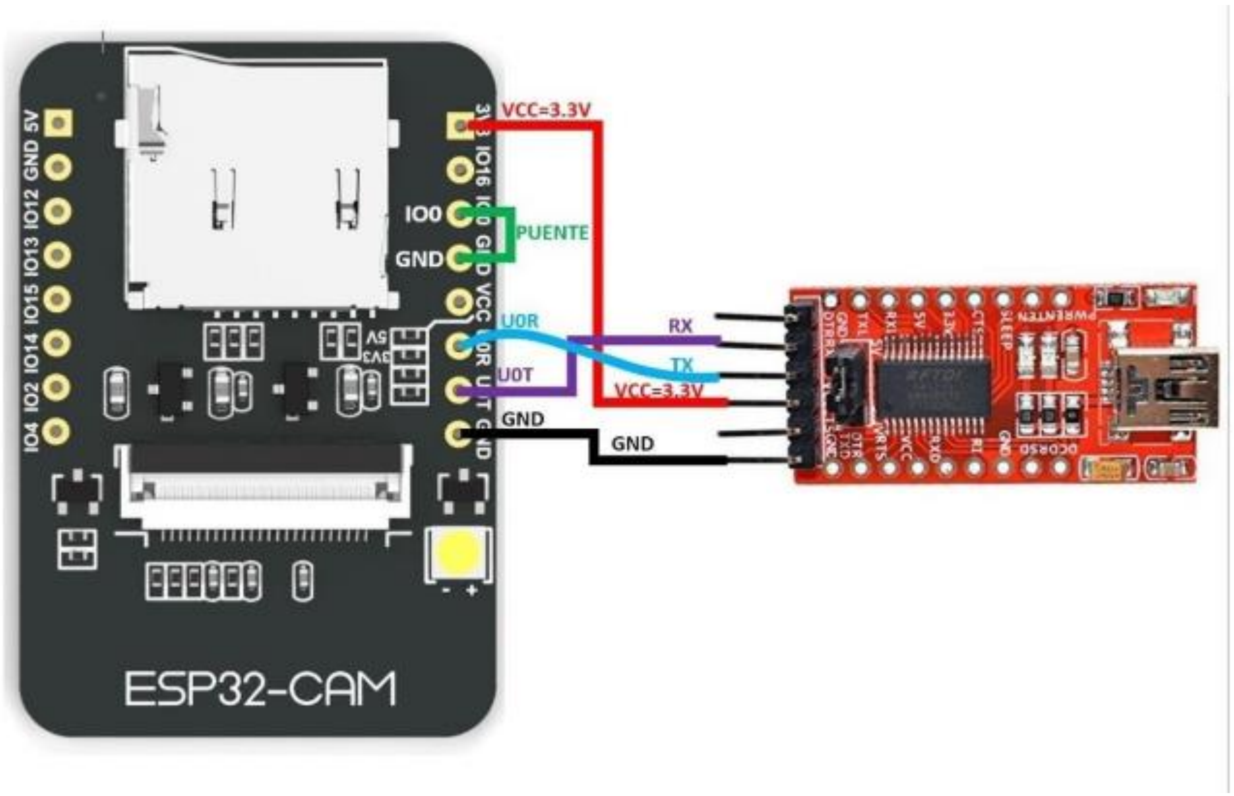


Fig 4.1.5 ESP32 Connection with FTDI [Courtesy: https://www.espressif.com/]

4.1.3 OV2640 Camera:

OV2640 is a 1/4-inch CMOS UXGA (1632 x 1232) image sensor, the sensor is small in size and low in operating voltage, providing the same functions of a single-chip UXGA camera and image processor. The OV2640 is a popular image sensor produced by Omni-Vision Technologies. It's commonly used in various electronic devices, especially in DIY projects, IoT applications, and development boards like Arduino, Raspberry Pi and ESP-32 boards. The OV2640 is known for its relatively low cost, compact size, and decent image quality, making it suitable for applications such as surveillance cameras, robotics, and embedded systems where capturing still images or video is required. It typically interfaces with microcontrollers or microprocessors via protocols like SPI or SCCB (I2C).

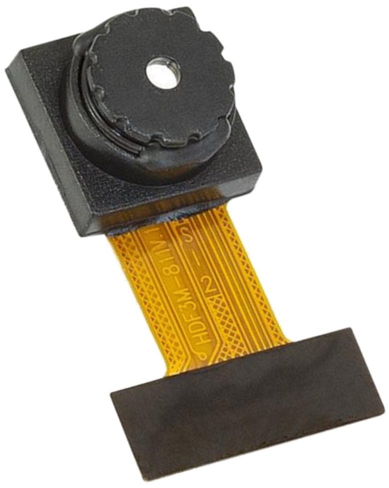


Fig 4.1.6 OV2640 Camera [Courtesy: amazon.in]

4.1.4 USB Cable (Micro-B):

Fig 4.1.7 USB micro-b cable
[Courtesy: amazon.in]

USB Micro-B cables are another type of USB cable commonly used for connecting devices, although they are gradually being phased out in favour of USB-C cables. The Micro-B connector is characterized by its asymmetrical shape, with one side slightly narrower than the other, making it necessary to orient the plug correctly when inserting it into a port.

These cables are often used for charging and data transfer purposes with a variety of devices, including smartphones, tablets, digital cameras, and other peripherals. They typically feature USB 2.0 or USB 3.0 specifications, offering varying data transfer speeds depending on the version.

4.1.5 Jumper Wires: Female to Female jumper wires required in order to connect ESP32 CAM with FTDI module.

4.2 <u>Software Requirements:</u>

Here is the list of required software:

4.2.1 Python (PyCharm): Python is a high-level, interpreted programming language known for its simplicity and readability. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python's extensive standard library and large community make it versatile for various applications, from web development and data analysis to artificial intelligence and scientific computing. Its dynamic typing and automatic memory management simplify development. Python's syntax emphasizes code readability, fostering clean and concise code, making it an excellent choice for beginners and professionals alike.

And PyCharm is a powerful integrated development environment (IDE) specifically designed for Python programming. It offers features like code completion, debugging, version control integration, and support for web development frameworks, making it a popular choice among Python developers for efficient coding workflows.

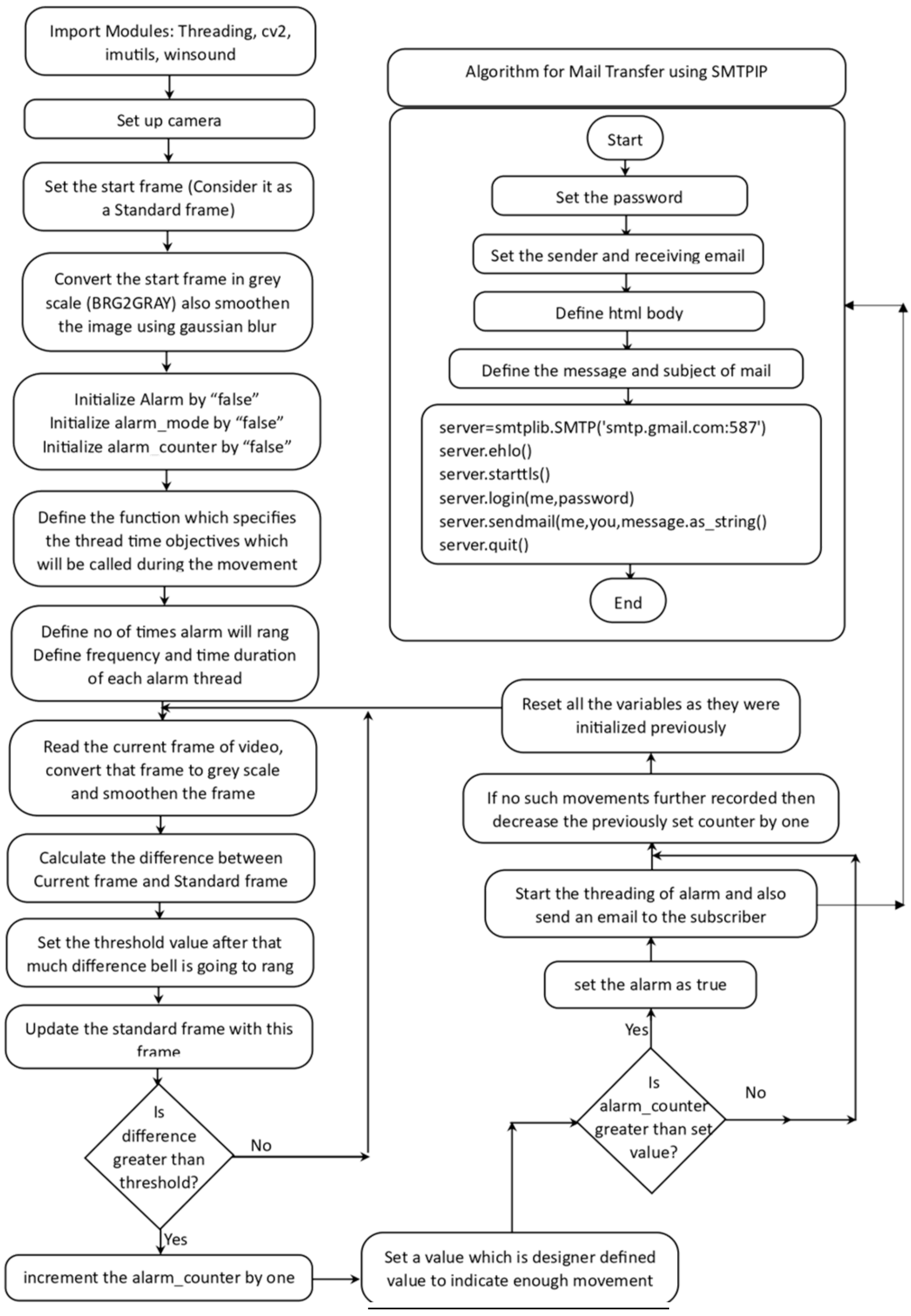
Fig 4.2.1 PyCharm & Python Logo [Courtesy: google.com]

4.2.2 Arduino IDE: The Arduino Integrated Development Environment (IDE) is a software platform for programming Arduino boards. It provides a simple interface for writing, compiling, and uploading code to control various electronic projects. The IDE supports C and C++ programming languages, making it accessible for beginners and experts alike.

Fig 4.2.2 Arduino IDE Logo [Courtesy: google.com]

4.3 Flowchart:

4.3.1 Motion Detection:


Import Modules: Threading, cv2, imutils, winsound
Set up camera
Set the start frame (Consider it as a Standard frame)
Convert the start frame in grey scale (BRG2GRAY) also smoothen the image using gaussian blur
Initialize Alarm by "false" Initialize alarm_mode by "false" Initialize alarm_counter by "false"
Define the function which specifies the thread time objectives which will be called during the movement
Define no of times alarm will rang Define frequency and time duration of each alarm thread
Read the current frame of video, convert that frame to grey scale and smoothen the frame
Calculate the difference between Current frame and Standard frame
Set the threshold value after that much difference bell is going to rang
Update the standard frame with this frame
Is difference greater than threshold?
No
Yes
increment the alarm_counter by one
Set a value which is designer defined value to indicate enough movement
Is alarm_counter greater than set value?
Yes
No
set the alarm as true
Start the threading of alarm and also send an email to the subscriber
If no such movements further recorded then decrease the previously set counter by one
Reset all the variables as they were initialized previously
Algorithm for Mail Transfer using SMTPIP
Start
Set the password
Set the sender and receiving email
Define html body
Define the message and subject of mail
End

```
server=smtplib.SMTP('smtp.gmail.com:587')
server.ehlo()
server.starttls()
server.login(me,password)
server.sendmail(me,you,message.as_string()
server.quit()
```

Fig 4.3.1 Motion Detection Flowchart

4.3.2 Object identification:

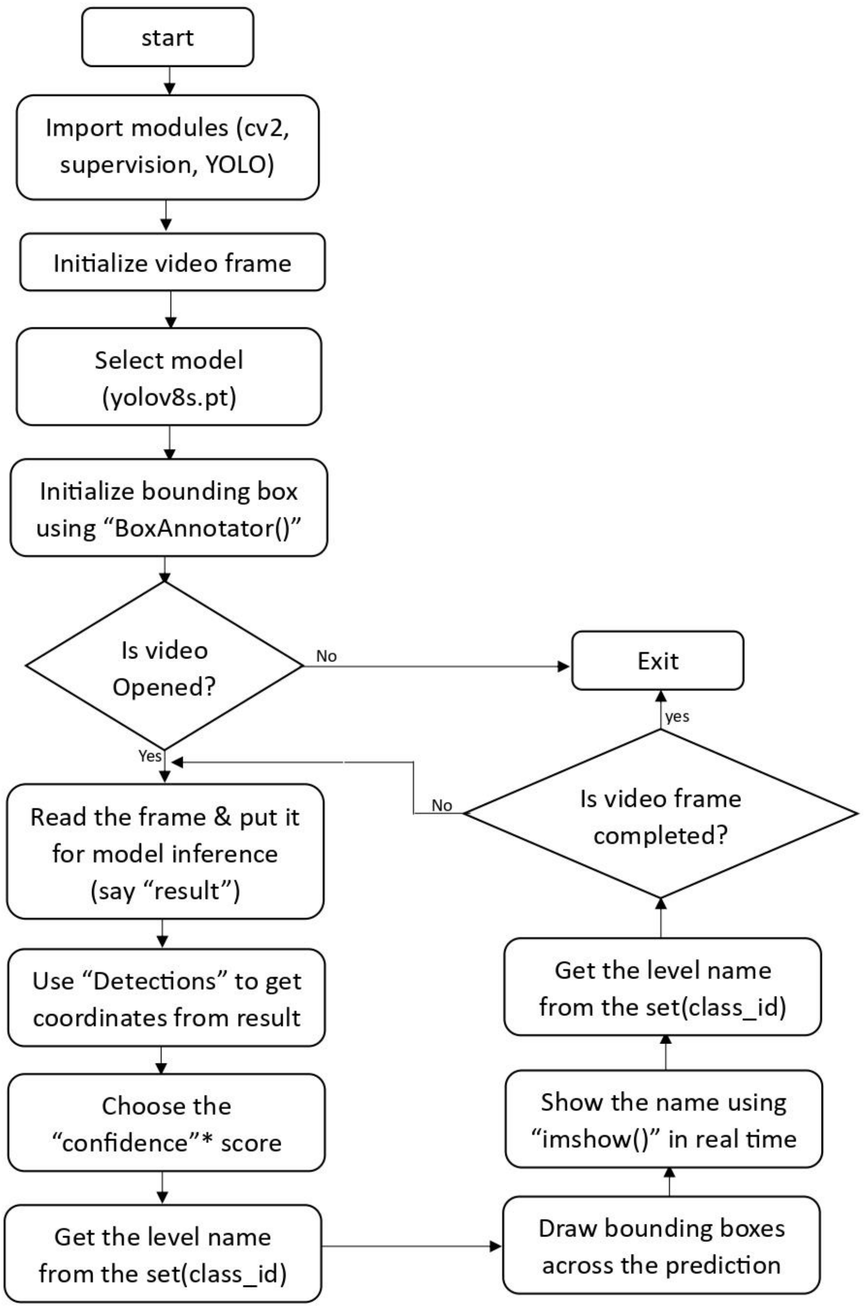

start
Import modules (cv2, supervision, YOLO)
Initialize video frame
Select model (yolov8s.pt)
Initialize bounding box using "BoxAnnotator()"
Is video Opened?
No
Exit
Yes
Read the frame & put it for model inference (say "result")
Use "Detections" to get coordinates from result
Choose the "confidence"* score
Get the level name from the set(class_id)
Draw bounding boxes across the prediction
Show the name using "imshow()" in real time
Get the level name from the set(class_id)
Is video frame completed?
No
yes

Fig 4.3.2 Object Identification Flowchart

4.4 Procedure:

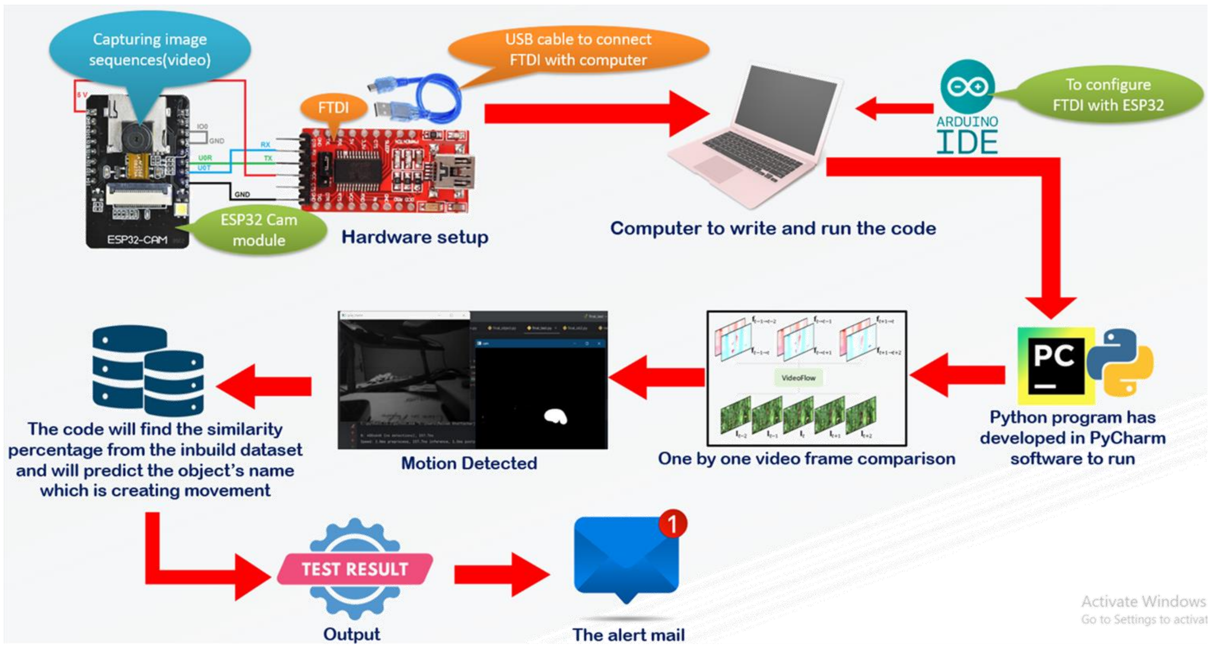

Capturing image sequences(video)
USB cable to connect FTDI with computer
FTDI
To configure FTDI with ESP32
ARDUINO IDE
ESP32 Cam module
ESP32-CAM
Hardware setup
Computer to write and run the code
The code will find the similarity percentage from the inbuild dataset and will predict the object's name which is creating movement
Motion Detected
One by one video frame comparison
Python program has developed in PyCharm software to run
TEST RESULT
Output
The alert mail
Activate Windows
Go to Settings to activat

Fig 4.4.1 Procedural Flow

- **Hardware Setup:**
  - Connect ESP32-s to FTDI and then to a laptop/desktop using a USB cable. This step establishes the physical connection between your ESP32 and your development environment.

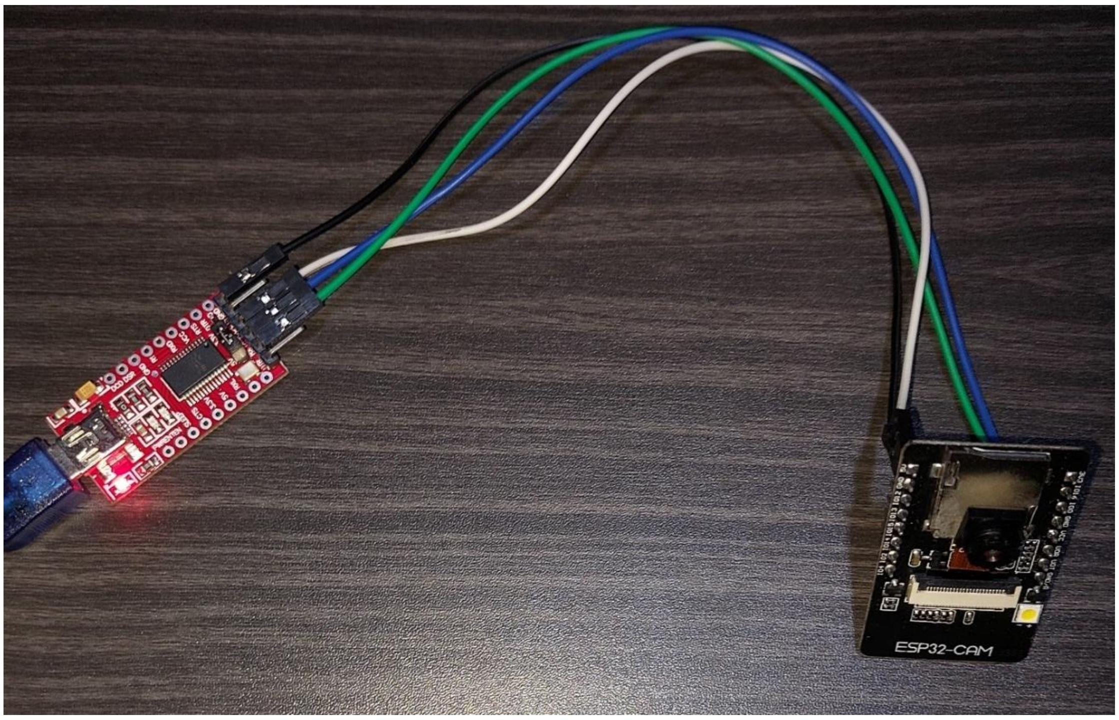


Fig 4.4.2 ESP32 CAM connected with FTDI

- **Software Development:**
  - Write the script in the Arduino IDE for programming the Controller Board (ESP32) to obtain its IPV4 address in URL form.
  - Develop a Python script for object identification and motion detection. This script will use the IPV4 address obtained from the ESP32 to communicate with it.
  - You can use any integrated development environment (IDE), such as PyCharm or VS Code, for writing and running your Python code.

- **Integration and Testing:**
  - Integrate the Arduino and Python scripts. Ensure that the Python script can successfully communicate with the ESP32 and receive the necessary data (such as the IPV4 address).
  - Test the integrated system in a controlled environment to ensure that object identification and motion detection are functioning as expected.

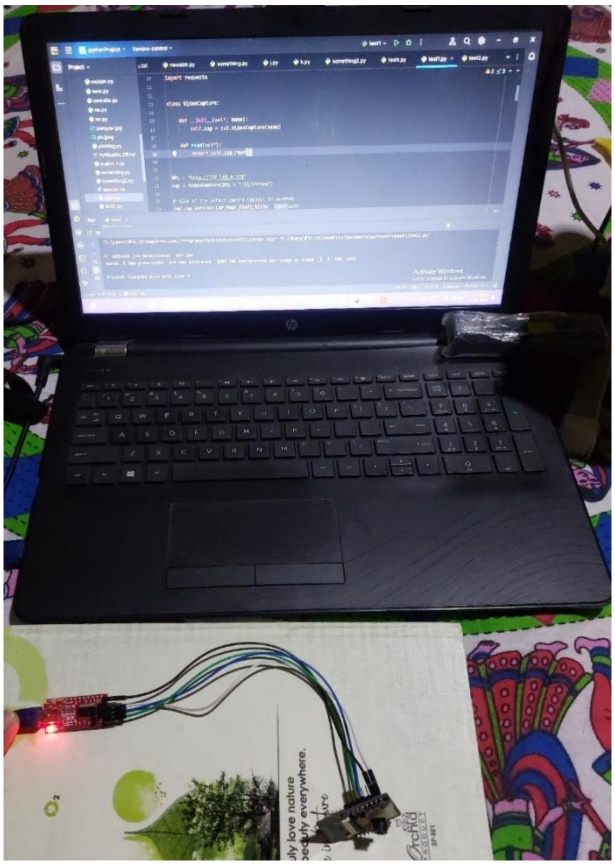


Fig 4.4.3 Connection of FTDI and ESP32 CAM with USB cable and Laptop

- **Real-world Deployment:**
    - Deploy the system in a real-world environment where you want to monitor for movement.
    - Ensure that the ESP32 and the Python script are running and able to communicate with each other in the deployed environment.

- **Alert System:**
    - Implement an alert system in your Python script to send an email notification whenever motion is detected.
    - The email includes the name of the detected object that is creating the movement.

Chapter 5:
RESULT

5. RESULT

5.1 Motion Detection:

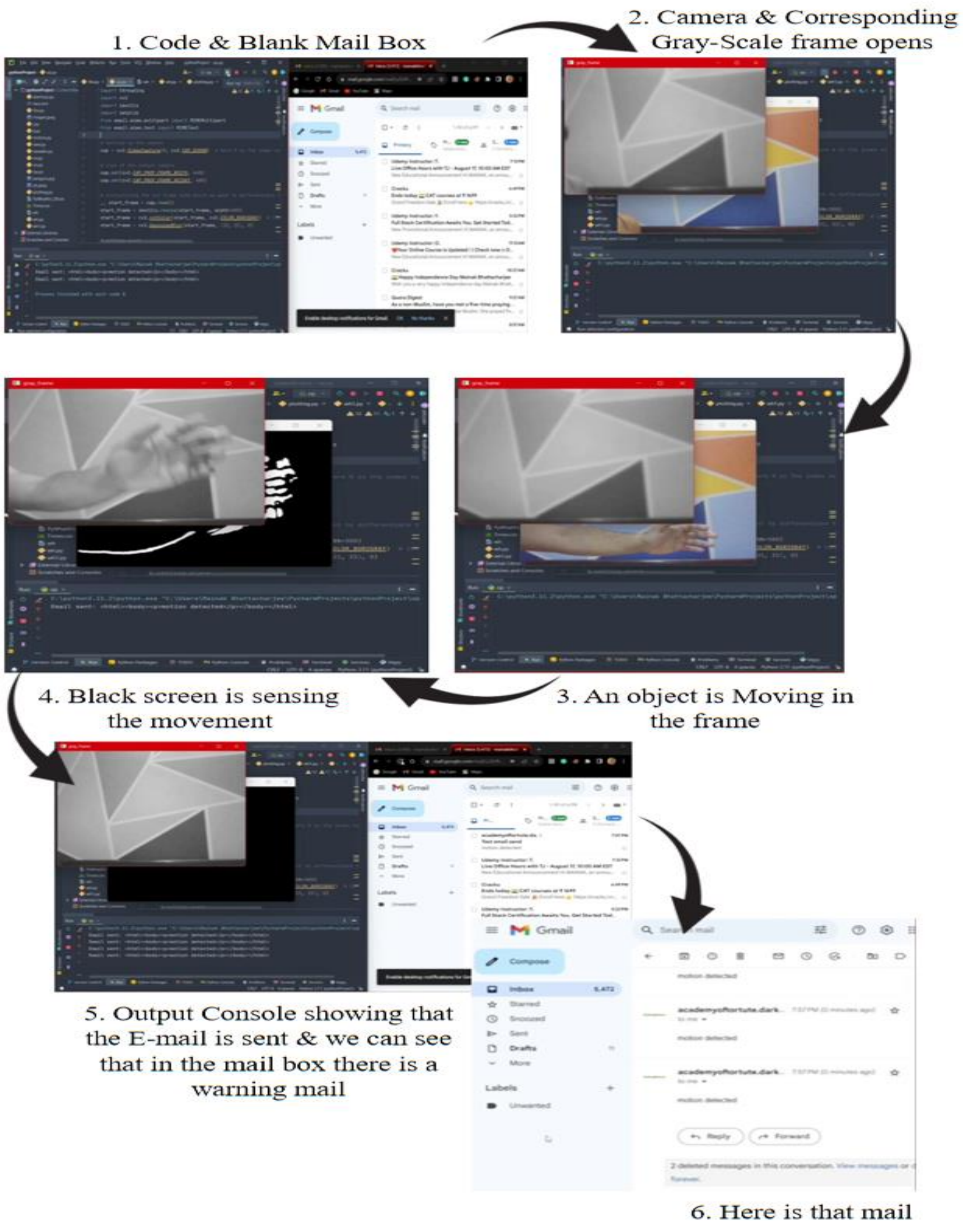
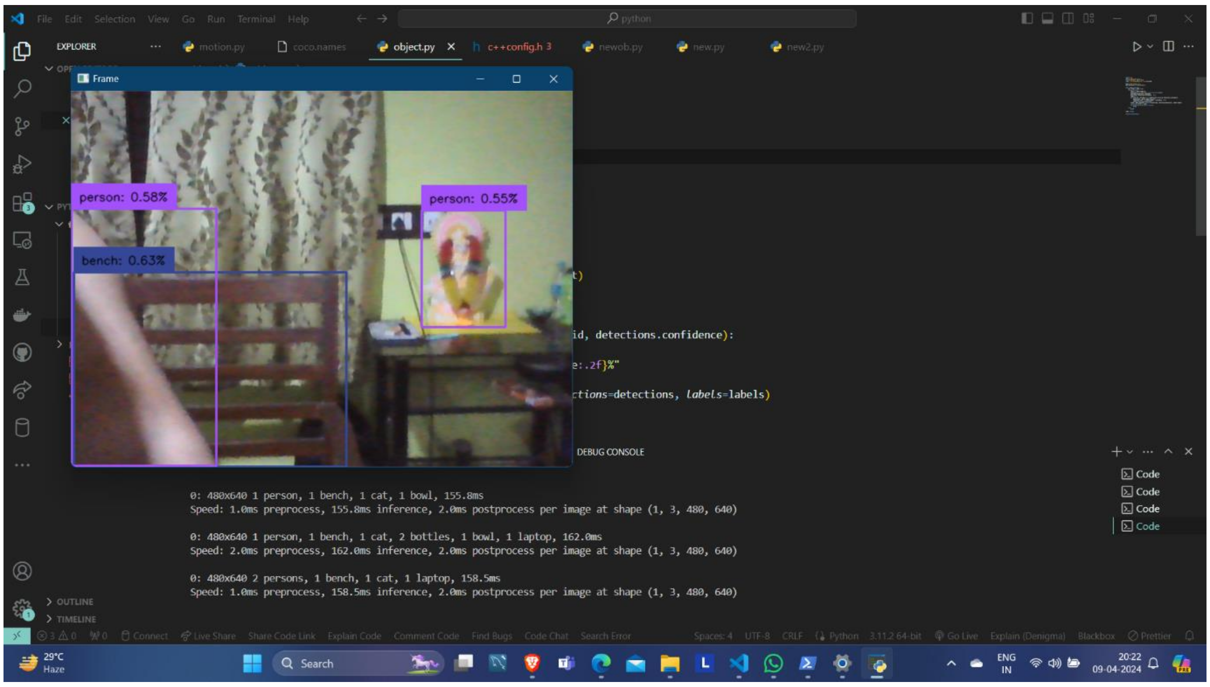

1. Code & Blank Mail Box
2. Camera & Corresponding Gray-Scale frame opens
3. An object is Moving in the frame
4. Black screen is sensing the movement
5. Output Console showing that the E-mail is sent & we can see that in the mail box there is a warning mail
6. Here is that mail

Fig 5.1 Motion Detection Code Output

5.2 Object Identification:

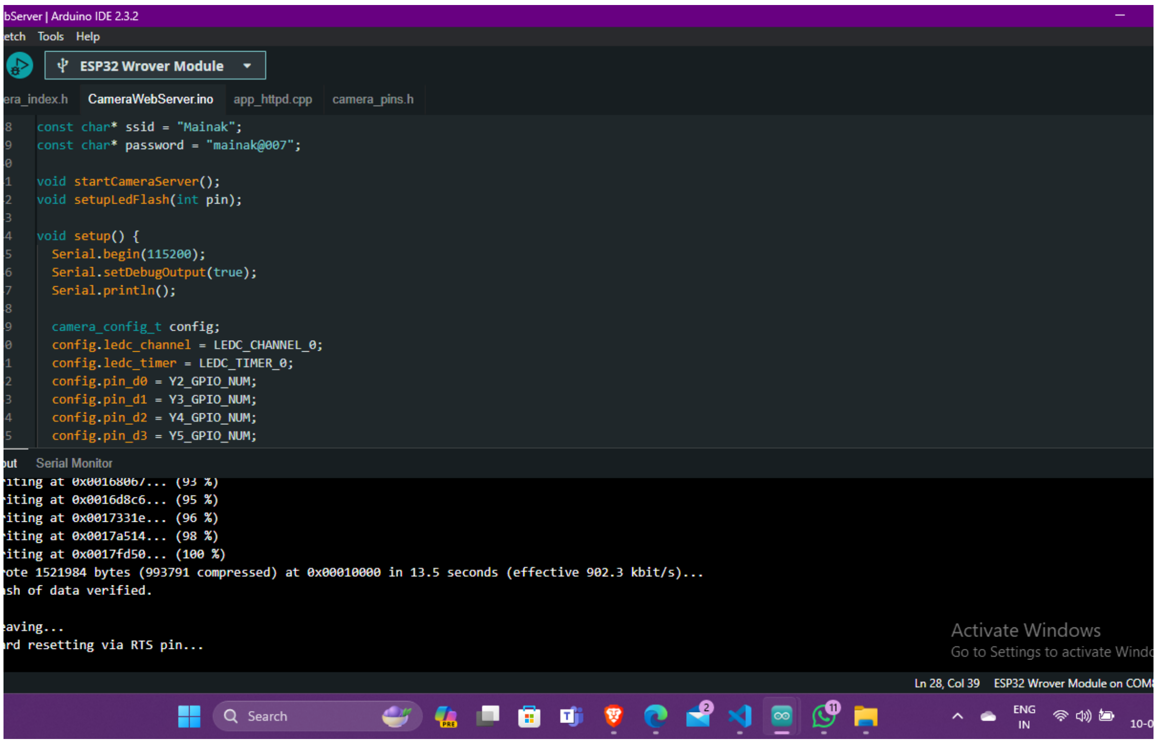


```
t)
id, detections.confidence):
e:.2f}%"
ctions=detections, labels=labels)
```

```
0: 480x640 1 person, 1 bench, 1 cat, 1 bowl, 155.8ms
Speed: 1.0ms preprocess, 155.8ms inference, 2.0ms postprocess per image at shape (1, 3, 480, 640)

0: 480x640 1 person, 1 bench, 1 cat, 2 bottles, 1 bowl, 1 laptop, 162.0ms
Speed: 2.0ms preprocess, 162.0ms inference, 2.0ms postprocess per image at shape (1, 3, 480, 640)

0: 480x640 2 persons, 1 bench, 1 cat, 1 laptop, 158.5ms
Speed: 1.0ms preprocess, 158.5ms inference, 2.0ms postprocess per image at shape (1, 3, 480, 640)
```

Fig 5.2 Object Identification Code Output

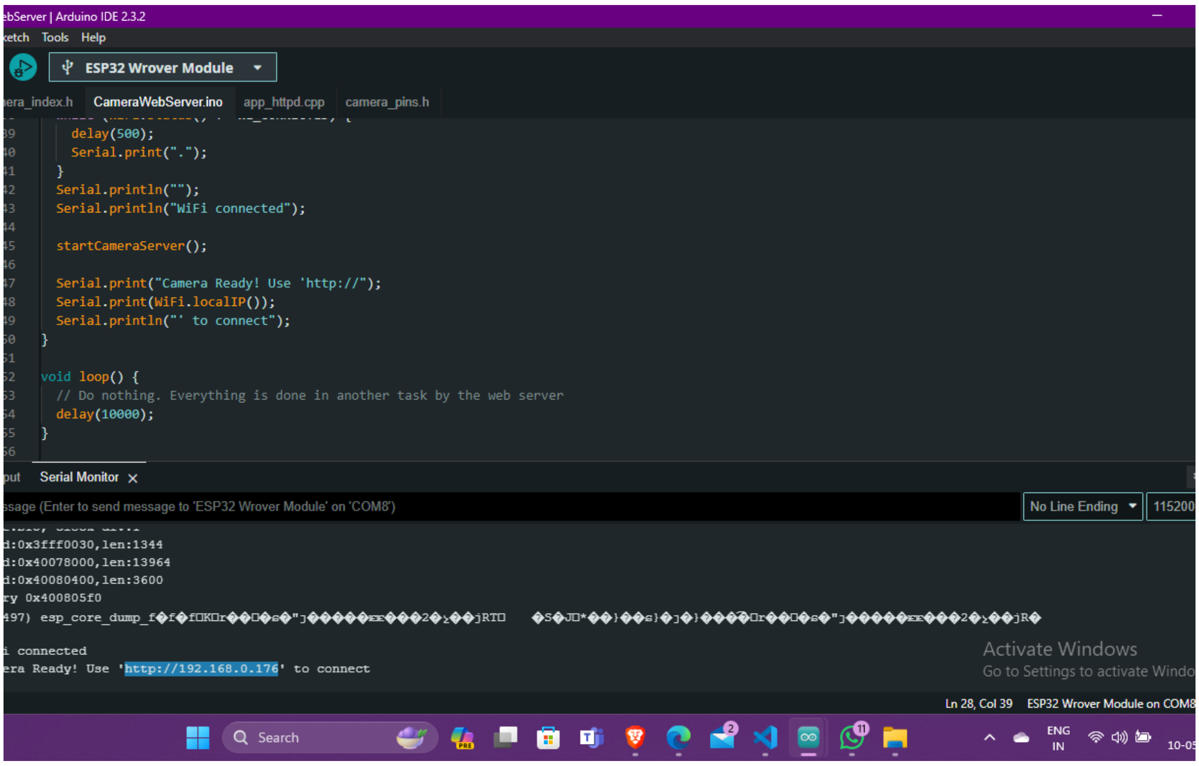
5.3 Final:



```
const char* ssid = "Mainak";
const char* password = "mainak@007";

void startCameraServer();
void setupLedFlash(int pin);

void setup() {
  Serial.begin(115200);
  Serial.setDebugOutput(true);
  Serial.println();

  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
```

```
iting at 0x00168067... (93 %)
iting at 0x0016d8c6... (95 %)
iting at 0x0017331e... (96 %)
iting at 0x0017a514... (98 %)
iting at 0x0017fd50... (100 %)
ote 1521984 bytes (993791 compressed) at 0x00010000 in 13.5 seconds (effective 902.3 kbit/s)...
sh of data verified.

aving...
rd resetting via RTS pin...
```

Fig 5.3.1 Connecting FTDI with ESP32 CAM



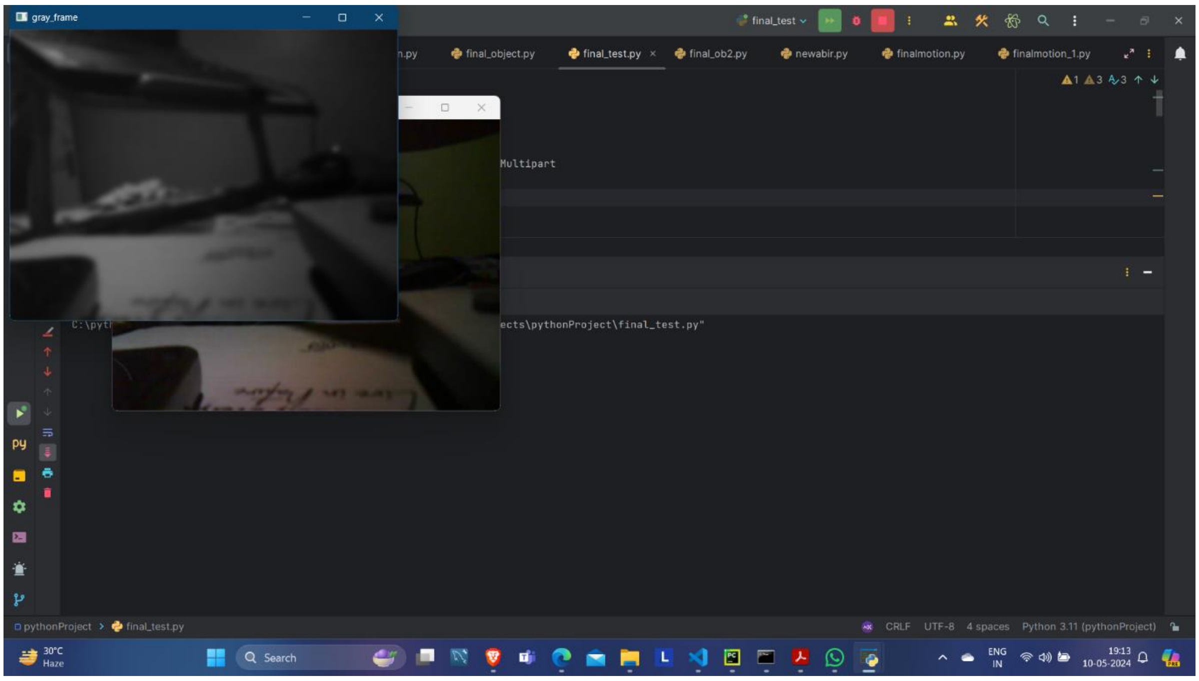
```
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");

  startCameraServer();

  Serial.print("Camera Ready! Use 'http://");
  Serial.print(WiFi.localIP());
  Serial.println("' to connect");
}

void loop() {
  // Do nothing. Everything is done in another task by the web server
  delay(10000);
}
```

ssage (Enter to send message to 'ESP32 Wrover Module' on 'COM8')

```
d:0x3fff0030,len:1344
d:0x40078000,len:13964
d:0x40080400,len:3600
ry 0x400805f0
497) esp_core_dump_f�f�f□K□r��□�s�"j�����ж���2�≥��jRT□   �S�J□*��}��s}�j�}���□r��□�s�"j�����ж���2�≥��jR�

i connected
era Ready! Use 'http://192.168.0.176' to connect
```

Ln 28, Col 39   ESP32 Wrover Module on COM8

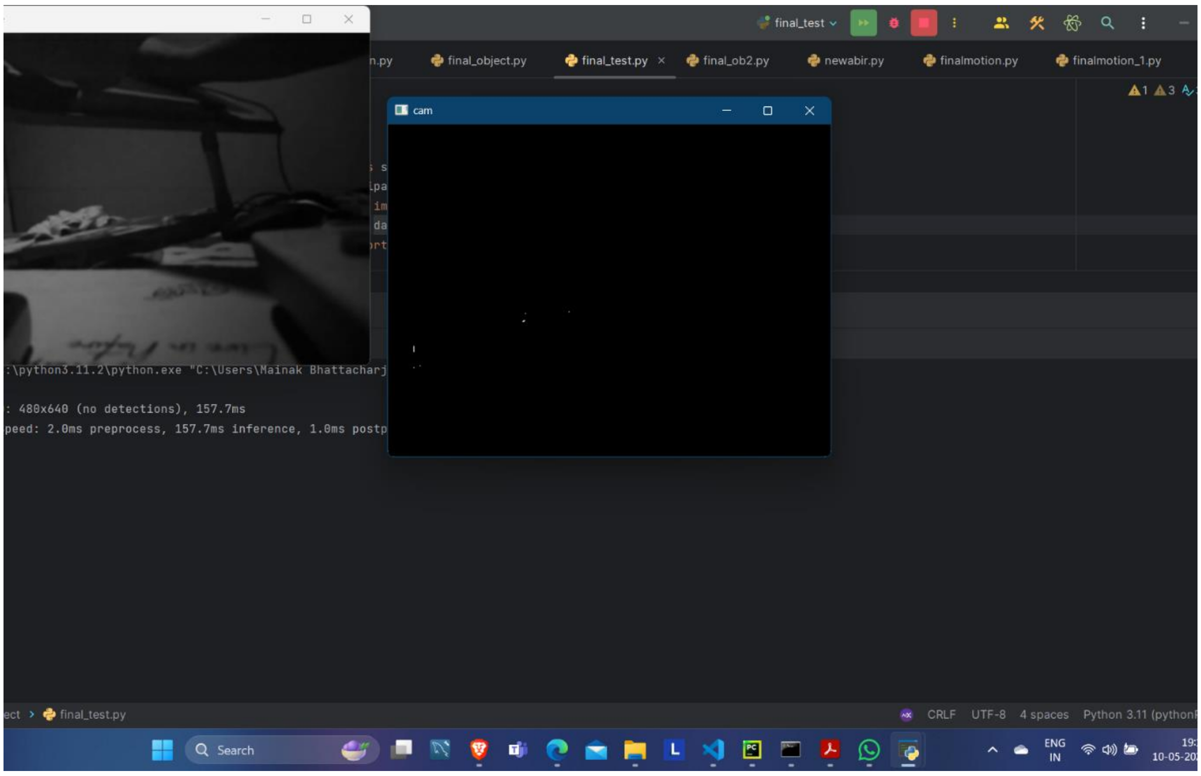
Fig 5.3.2 Getting IPV4 address from Arduino IDE



```
Multipart
```

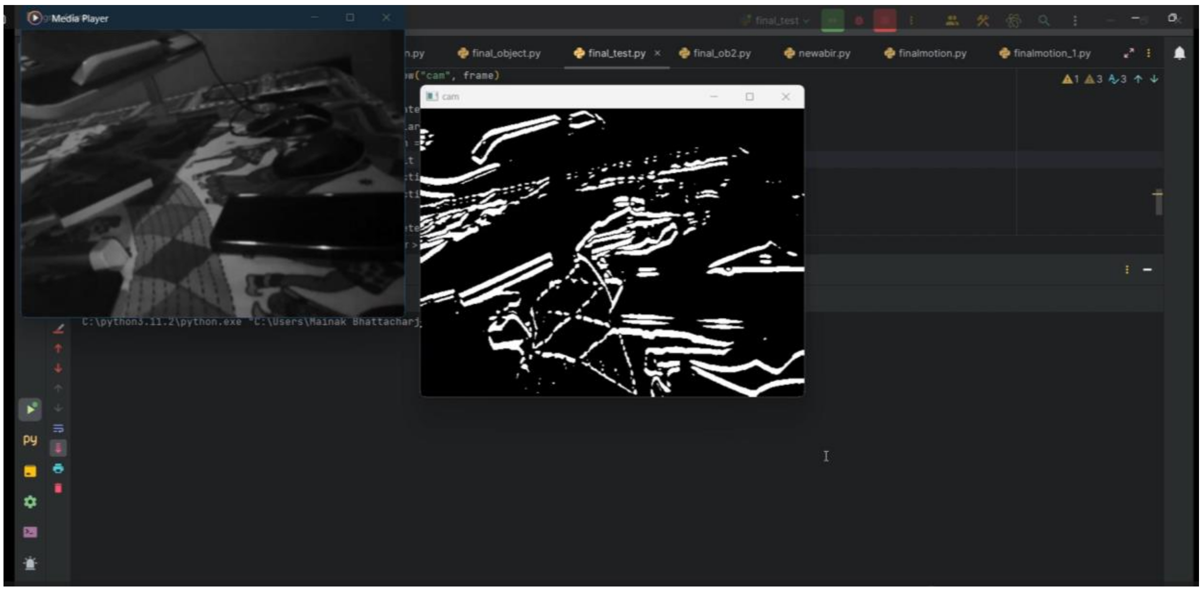
```
C:\pyt                                  ects\pythonProject\final_test.py"
```

Fig 5.3.3 Initializing the camera, converting the frame in grey scale



```
:\python3.11.2\python.exe "C:\Users\Mainak Bhattacharj

: 480x640 (no detections), 157.7ms
peed: 2.0ms preprocess, 157.7ms inference, 1.0ms postp
```

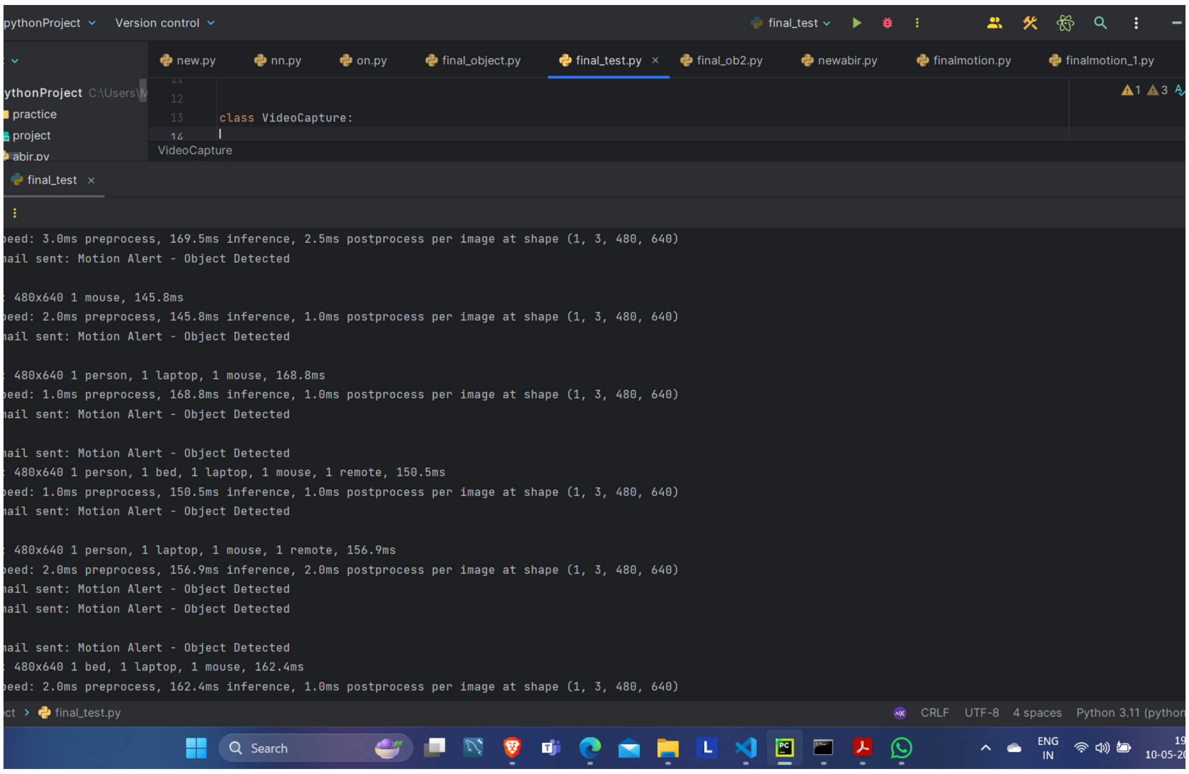
Fig 5.3.4 No difference between two frames so no white spot detected



```
w("cam", frame)
```

```
C:\python3.11.2\python.exe "C:\Users\Mainak Bhattacharj
```

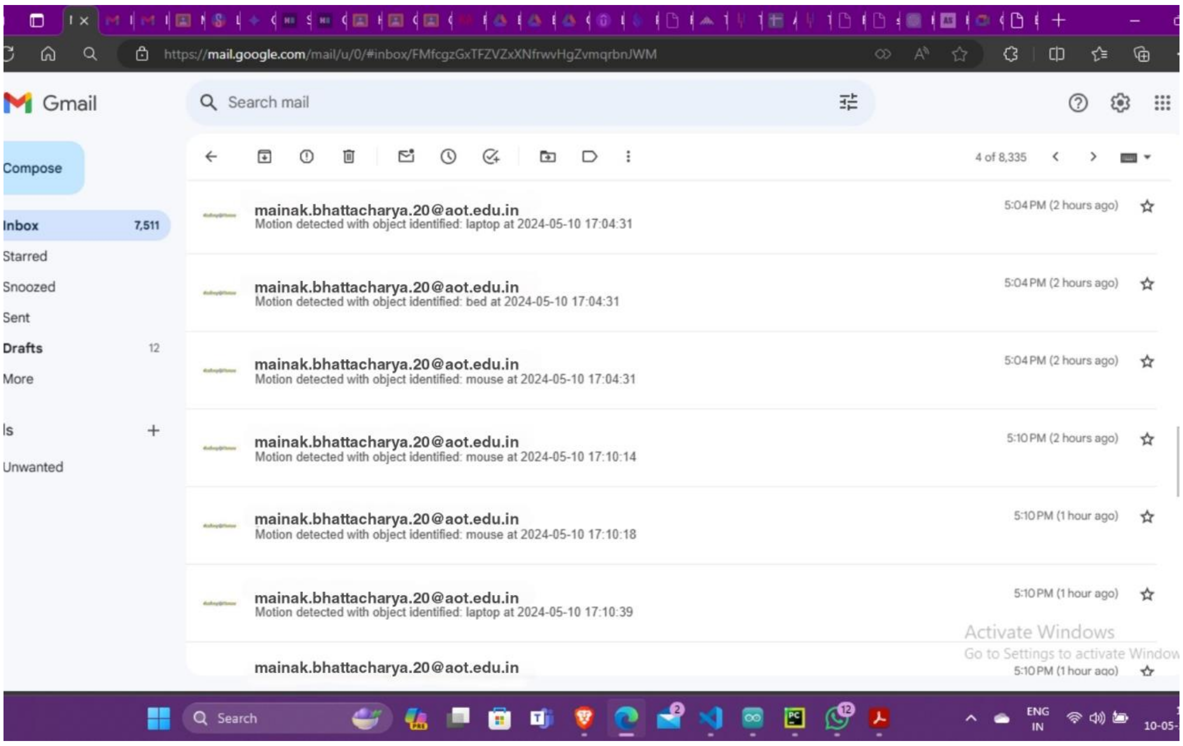
Fig 5.3.5 Tracking of white spot by comparing (i.e. the motion)



```
class VideoCapture:
```

```
eed: 3.0ms preprocess, 169.5ms inference, 2.5ms postprocess per image at shape (1, 3, 480, 640)
ail sent: Motion Alert - Object Detected

 480x640 1 mouse, 145.8ms
eed: 2.0ms preprocess, 145.8ms inference, 1.0ms postprocess per image at shape (1, 3, 480, 640)
ail sent: Motion Alert - Object Detected

 480x640 1 person, 1 laptop, 1 mouse, 168.8ms
eed: 1.0ms preprocess, 168.8ms inference, 1.0ms postprocess per image at shape (1, 3, 480, 640)
ail sent: Motion Alert - Object Detected

ail sent: Motion Alert - Object Detected
 480x640 1 person, 1 bed, 1 laptop, 1 mouse, 1 remote, 150.5ms
eed: 1.0ms preprocess, 150.5ms inference, 1.0ms postprocess per image at shape (1, 3, 480, 640)
ail sent: Motion Alert - Object Detected

 480x640 1 person, 1 laptop, 1 mouse, 1 remote, 156.9ms
eed: 2.0ms preprocess, 156.9ms inference, 2.0ms postprocess per image at shape (1, 3, 480, 640)
ail sent: Motion Alert - Object Detected
ail sent: Motion Alert - Object Detected

ail sent: Motion Alert - Object Detected
 480x640 1 bed, 1 laptop, 1 mouse, 162.4ms
eed: 2.0ms preprocess, 162.4ms inference, 1.0ms postprocess per image at shape (1, 3, 480, 640)
```

Fig 5.3.6 Getting output as the name of moving object



mainak.bhattacharya.20@aot.edu.in
Motion detected with object identified: laptop at 2024-05-10 17:04:31
5:04 PM (2 hours ago)

mainak.bhattacharya.20@aot.edu.in
Motion detected with object identified: bed at 2024-05-10 17:04:31
5:04 PM (2 hours ago)

mainak.bhattacharya.20@aot.edu.in
Motion detected with object identified: mouse at 2024-05-10 17:04:31
5:04 PM (2 hours ago)

mainak.bhattacharya.20@aot.edu.in
Motion detected with object identified: mouse at 2024-05-10 17:10:14
5:10 PM (2 hours ago)

mainak.bhattacharya.20@aot.edu.in
Motion detected with object identified: mouse at 2024-05-10 17:10:18
5:10 PM (1 hour ago)

mainak.bhattacharya.20@aot.edu.in
Motion detected with object identified: laptop at 2024-05-10 17:10:39
5:10 PM (1 hour ago)

mainak.bhattacharya.20@aot.edu.in
5:10 PM (1 hour ago)

Fig 5.3.6 Sending Alert Mail to the subscriber

Chapter 6:
FUTURE WORK & CONCLUSION

6.1 CONCLUSION

We have successfully completed the object identification and motion detection process using deep learning algorithms, Optical Flow (for motion detection) and YOLO (You Only Look Once) v8 (for object identification). We have shown that how the code is working theoretically and have also shown the same via pictorial representation. The dataset we have used is a predefined dataset of objects to compare with. So, the result ultimately depends on a parameter called confidence percentage (that is how much an object is like a data present in that set). But that object may be present in data set but for any reason (say distorted image or same object but different type) algorithm may provide wrong answer. To fix that problem several present-day evolving technologies are there like neural networking, contour mathematics etc. they may increase the accuracy of that project. Also to provide a low-price solution we have used ESP32-CAM which has a very less pixel camera (2 Mega Pixel only), but several boards are there like Arduino and other modern-day microcontrollers which could have a higher Mega Pixel camera and our project can provide more efficient solutions in challenging environment like low light, changing weathers etc. The main motto was to build a device for the security purpose which will not require a continuous monitoring at low cost. This device can be placed where anyone don't want any kind of movement like Safes, Locker etc. We hope that we will complete the remaining part of this project very soon. We are building the prototype but in future if we got any chance to scale up our project then we will take a subscription in a renowned cloud platform (Google, AWS etc.) for using the large amount of dataset & storage.

6.2 FUTURE OF OUR PROJECT

Here are a few directions for future plans for object detector and motion detector projects:

- **Integration with IoT:** One potential area for future development is the integration of object detection and motion detection with Internet of Things (IoT) devices. For example, sensors placed in public spaces could be used to detect motion and objects, providing real-time information for security and crowd management.
- **Stand Alone approach:** We Can upload the code direct in Micro-controller and that would be a stand-alone approach**.**
- **3D object detection:** While current object detection systems are limited to 2D images, there is potential for future development in 3D object detection.
- **Privacy-preserving techniques:** As object detection and motion detection systems become more widespread, there is an increasing need for privacy-preserving techniques.
- **Storing of video:** We can store the anomaly video as a proof. For that we have to provide a memory card with the microcontroller.
- **Web/App approach:** We can provide a web/app approach for better and easier user experience.

REFERENCES & APPENDIX

REFERENCES

1. Title: Object Detection and Tracking in Video Sequences

Link: [Object Detection and Tracking in Video Sequences](https://scholarcommons.sc.edu/cgi/viewcontent.cgi?article=3899&context=etd)

2. Title: Real-Time Object Detection and Motion Tracking Using Deep Learning

Link: [Real-Time Object Detection and Motion Tracking Using Deep Learning](https://repository.lib.ncsu.edu/bitstream/handle/1840.20/35244/etd.pdf)

3. Title: Vision-Based Object Detection and Motion Tracking for Autonomous Vehicles

Link: [Vision-Based Object Detection and Motion Tracking for Autonomous Vehicles](https://digital.library.ryerson.ca/islandora/object/RULA%3A4586/datastream/OBJ/download/Vision-Based_Object_Detection_and_Motion_Tracking_for_Autonomous_Vehicles.pdf)

4. Title: Object Detection and Motion Tracking in Surveillance Videos

Link: [Object Detection and Motion Tracking in Surveillance Videos](https://digitalcommons.wpi.edu/cgi/viewcontent.cgi?article=1275&context=mqp-all)

5. Title: Deep Learning Approaches for Object Detection and Motion Tracking in Smart Environments

Link: [Deep Learning Approaches for Object Detection and Motion Tracking in Smart Environments](https://researchrepository.murdoch.edu.au/id/eprint/57120/1/Deep_Learning_Approaches_for_Object_Detection_and_Motion_Tracking_in_Smart_Environments.pdf)

APPENDIX

6 Code:

- **ESP-32 Connection:**

```
#include "esp_camera.h"
#include <WiFi.h>
//
// WARNING!!! PSRAM IC required for UXGA resolution and high
JPEG quality
//           Ensure ESP32 Wrover Module or other board with PSRAM is
selected
//           Partial images will be transmitted if image exceeds buffer size
//
//           You must select partition scheme from the board menu that has
at least 3MB APP space.
//           Face Recognition is DISABLED for ESP32 and ESP32-S2,
because it takes up from 15
//           seconds to process single frame. Face Detection is ENABLED if
PSRAM is enabled as well

// ===================
// Select camera model
// ===================
//#define CAMERA_MODEL_WROVER_KIT // Has PSRAM
//#define CAMERA_MODEL_ESP_EYE // Has PSRAM
//#define CAMERA_MODEL_ESP32S3_EYE // Has PSRAM
//#define CAMERA_MODEL_M5STACK_PSRAM // Has PSRAM
//#define CAMERA_MODEL_M5STACK_V2_PSRAM // M5Camera
version B Has PSRAM
//#define CAMERA_MODEL_M5STACK_WIDE // Has PSRAM
//#define CAMERA_MODEL_M5STACK_ESP32CAM // No PSRAM
//#define CAMERA_MODEL_M5STACK_UNITCAM // No PSRAM
#define CAMERA_MODEL_AI_THINKER // Has PSRAM
//#define CAMERA_MODEL_TTGO_T_JOURNAL // No PSRAM
//#define CAMERA_MODEL_XIAO_ESP32S3 // Has PSRAM
// * Espressif Internal Boards *
//#define CAMERA_MODEL_ESP32_CAM_BOARD
//#define CAMERA_MODEL_ESP32S2_CAM_BOARD
//#define CAMERA_MODEL_ESP32S3_CAM_LCD
```

```
//#define CAMERA_MODEL_DFRobot_FireBeetle2_ESP32S3 // Has
PSRAM
//#define CAMERA_MODEL_DFRobot_Romeo_ESP32S3 // Has
PSRAM
#include "camera_pins.h"

// ===========================
// Enter your WiFi credentials
// ===========================
const char* ssid = "Mainak";
const char* password = "mainak@007";

void startCameraServer();
void setupLedFlash(int pin);

void setup() {
  Serial.begin(115200);
  Serial.setDebugOutput(true);
  Serial.println();

  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sccb_sda = SIOD_GPIO_NUM;
  config.pin_sccb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.frame_size = FRAMESIZE_UXGA;
  config.pixel_format = PIXFORMAT_JPEG; // for streaming
  //config.pixel_format = PIXFORMAT_RGB565; // for face
detection/recognition
  config.grab_mode = CAMERA_GRAB_WHEN_EMPTY;
  config.fb_location = CAMERA_FB_IN_PSRAM;
  config.jpeg_quality = 12;
  config.fb_count = 1;

  // if PSRAM IC present, init with UXGA resolution and higher JPEG
quality
  //                      for larger pre-allocated frame buffer.
  if(config.pixel_format == PIXFORMAT_JPEG){
    if(psramFound()){
      config.jpeg_quality = 10;
      config.fb_count = 2;
      config.grab_mode = CAMERA_GRAB_LATEST;
    } else {
      // Limit the frame size when PSRAM is not available
      config.frame_size = FRAMESIZE_SVGA;
      config.fb_location = CAMERA_FB_IN_DRAM;
    }
  } else {
    // Best option for face detection/recognition
    config.frame_size = FRAMESIZE_240X240;
#if CONFIG_IDF_TARGET_ESP32S3
    config.fb_count = 2;
#endif
  }

#if defined(CAMERA_MODEL_ESP_EYE)
  pinMode(13, INPUT_PULLUP);
  pinMode(14, INPUT_PULLUP);
#endif

  // camera init
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
  }

  sensor_t * s = esp_camera_sensor_get();
  // initial sensors are flipped vertically and colors are a bit saturated
  if (s->id.PID == OV3660_PID) {
    s->set_vflip(s, 1); // flip it back
    s->set_brightness(s, 1); // up the brightness just a bit
    s->set_saturation(s, -2); // lower the saturation
  }
  // drop down frame size for higher initial frame rate
  if(config.pixel_format == PIXFORMAT_JPEG){
    s->set_framesize(s, FRAMESIZE_QVGA);
  }

#if defined(CAMERA_MODEL_M5STACK_WIDE) ||
defined(CAMERA_MODEL_M5STACK_ESP32CAM)
  s->set_vflip(s, 1);
  s->set_hmirror(s, 1);
#endif

#if defined(CAMERA_MODEL_ESP32S3_EYE)
  s->set_vflip(s, 1);
#endif

// Setup LED FLash if LED pin is defined in camera_pins.h
#if defined(LED_GPIO_NUM)
  setupLedFlash(LED_GPIO_NUM);
#endif

  WiFi.begin(ssid, password);
  WiFi.setSleep(false);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");

  startCameraServer();
```

```
    Serial.print("Camera Ready! Use 'http://");
    Serial.print(WiFi.localIP());
    Serial.println("' to connect");
  }

  void loop() {
    // Do nothing. Everything is done in another task by the web server
    delay(10000);
}
```

- **Motion Detection:**

```
import threading
import cv2
import imutils
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText

# Setting up the camera
cap = cv2.VideoCapture(0, cv2.CAP_DSHOW)  # Here 0 is the index no. of the
camera

# size of the output camera
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)

# initializing the 1st frame with which we want to differentiate the current frame
_, start_frame = cap.read()
start_frame = imutils.resize(start_frame, width=500)
start_frame = cv2.cvtColor(start_frame, cv2.COLOR_BGR2GRAY)  #
Converting the color from BGR to gray scale
start_frame = cv2.GaussianBlur(start_frame, (21, 21), 0)

# setting alarm
alarm = False  # by default the alarm is set to false
alarm_mode = False  # to toggle the alarm
alarm_counter = 0  # how much duration is the movement to ring the alarm

# any msg or hardware device to ring the alarm or send a notification
```

```
def beep_alarm():
global alarm

password="jivqfxmtffvkqvab"
me="academyoftortute.darkfuture@gmail.com"
you= "mainakbhattacharya.2108@gmail.com"
email_body="""<html><body><p>motion detected</p></body></html>"""
message= MIMEMultipart('alternative', None,[MIMEText (email_body, 'html')])
message['Subject'] = 'Test email send'
message['From'] = me
message['To'] = you

try:
server=smtplib.SMTP('smtp.gmail.com:587')
server.ehlo()
server.starttls()
server.login(me,password)
server.sendmail(me,you,message.as_string())
server.quit()
print(f'Email sent: {email_body}')
except Exception as e:
print(f'Error in sending email: {e}')

alarm = False

while True:
_, frame = cap.read()
frame = imutils.resize(frame, width=500)

if alarm_mode:
frame_bw = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
frame_bw = cv2.GaussianBlur(frame_bw, (5, 5), 0)

difference = cv2.absdiff(frame_bw, start_frame)  # finding the differences with
initial frame and current frame
threshold = cv2.threshold(difference, 25, 255, cv2.THRESH_BINARY)[1]  #
setting a threshold value
start_frame = frame_bw  # frame for next iteration
if threshold.sum() > 400000:
```

```
# if we notify any movement more than threshold then we increment the alarm
counter by 1
alarm_counter += 1
else:
# if we don't notify any movement more than threshold then we decrement the
alarm counter by 1
if alarm_counter > 0:
alarm_counter -= 1

cv2.imshow("cam", threshold)  # Gray scale screen
else:
cv2.imshow("cam", frame)   # Gray scale screen
if alarm_counter > 20:
if not alarm:
alarm = True
threading.Thread(target=beep_alarm).start()

key_pressed = cv2.waitKey(30)
if key_pressed == ord("t"):
alarm_mode = not alarm_mode   # press t for gray scale screen
alarm_counter = 0
if key_pressed == ord("q"):
alarm_mode = False       # press q for close
break
cv2.imshow("gray_frame", start_frame)  # original output frame

cap.release()
cv2.destroyAllWindows()
```

- **Object Identification:**

```
import cv2
import supervision as sv
from ultralytics import YOLO
video = cv2.VideoCapture(0, cv2.CAP_DSHOW)

model=YOLO("yolov8s.pt")
bbox_annotator=sv.BoxAnnotator()

while video.isOpened():
   ret, frame = video.read()
```

```
    if ret == True:
        result = model(frame)[0]
        detections = sv.Detections.from_ultralytics(result)
        confidence=detections.confidence
        detections= detections[confidence > 0.5]
        labels = []
        for class_id, confidence in zip(detections.class_id,
detections.confidence):
            object_name = model.names[class_id]
            confidence_text = f"{object_name}: {confidence:.2f}%"
            labels.append(confidence_text)
        frame = bbox_annotator.annotate(scene=frame, detections=detections,
labels=labels)
        cv2.imshow("Frame", frame)
        if cv2.waitKey(1) & 0xFF == ord("q"):
            break
    else:
        break

video.release

cv2.destroyAllWindows
```

- **Final Code:**

```
import threading
import cv2
import imutils
import smtplib
import supervision as sv
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from datetime import datetime
from ultralytics import YOLO
import requests


class VideoCapture:

    def _init_(self, name):
        self.cap = cv2.VideoCapture(name)

    def read(self):
        return self.cap.read()


URL = "http://192.168.0.176"
cap = VideoCapture(URL + ":81/stream")

# Size of the output camera (adjust if needed)
cap.cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
cap.cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)

# Initializing the 1st frame with which we want to differentiate the current
frame
_, start_frame = cap.read()
start_frame = imutils.resize(start_frame, width=500)
start_frame = cv2.cvtColor(start_frame, cv2.COLOR_BGR2GRAY)
start_frame = cv2.GaussianBlur(start_frame, (21, 21), 0)
```

```
# Setting alarm
alarm = False
alarm_mode = False
alarm_counter = 0

# Email configuration (replace with your credentials)
password = "your_password"
sender_email = "your_email@gmail.com"
receiver_email = "recipient_email@gmail.com"


# Function to send email notification
def send_email(subject, body):
    message = MIMEMultipart()
    message["From"] = sender_email
    message["To"] = receiver_email
    message["Subject"] = subject
    message.attach(MIMEText(body, "plain"))

    try:
        server = smtplib.SMTP("smtp.gmail.com:587")
        server.ehlo()
        server.starttls()
        server.login(sender_email, password)
        server.sendmail(sender_email, receiver_email, message.as_string())
        server.quit()
        print(f"Email sent: {subject}")
    except Exception as e:
        print(f"Error sending email: {e}")


# Object detection model (replace with your model path)
model = YOLO("yolov8s.pt")
bbox_annotator = sv.BoxAnnotator()


# Function to send email notification
def beep_alarm(object_name):
    global alarm
    current_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    email_body = (
        f"Motion detected with object identified: {object_name} at
{current_time}"
    )
    send_email("Motion Alert - Object Detected", email_body)
    alarm = False

if _name_ == '_main_':
    requests.get(URL + "/control?var=framesize&val={}".format(8))

while True:
    ret, frame = cap.read()

    if ret:
        frame = imutils.resize(frame, width=500)

        if alarm_mode:
            frame_bw = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            frame_bw = cv2.GaussianBlur(frame_bw, (5, 5), 0)

            difference = cv2.absdiff(frame_bw, start_frame)
            threshold = cv2.threshold(difference, 25, 255,
cv2.THRESH_BINARY)[1]
            start_frame = frame_bw

            if threshold.sum() > 400000:
                alarm_counter += 1
            else:
                if alarm_counter > 0:
                    alarm_counter -= 1

            cv2.imshow("cam", threshold)
        else:
            cv2.imshow("cam", frame)

        if alarm_counter > 20:
            if not alarm:
                alarm = True
                result = model(frame)[0]
                detections = sv.Detections.from_ultralytics(result)
            detections = detections[detections.confidence > 0.5]

            if detections:
               labels = [
                  result.names[class_id] for class_id in detections.class_id
               ]
               for object_name in labels:
                  threading.Thread(target=beep_alarm,
args=(object_name,)).start()

      key_pressed = cv2.waitKey(30)
      if key_pressed == ord("t"):
         alarm_mode = not alarm_mode
         alarm_counter = 0
      if key_pressed == ord("q"):
         alarm_mode = False
         break
      cv2.imshow("gray_frame", start_frame)

cv2.destroyAllWindows()
```