Real-Time Video Object Detection using Yolo and Computer Vision with Automated WhatsApp Alerts for Security and Surveillance

A Project Report submitted
in partial fulfillment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE & ENGINEERING

By

21B01A0501 - Adapala Lohitha
21B01A0511 - Ballarapu Glory
21B01A0526 - Bonam Renuka Lakshmi Tejaswini
21B01A0527 - Boorlagadda V N S Asritha
22B05A0501 – Bhupatiraju Harshita

Under the esteemed guidance of
Dr. M. Prasad **Ph. D, MISTE, SMIEEE**
Associate Professor

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
SHRI VISHNU ENGINEERING COLLEGE FOR WOMEN(A)
(Approved by AICTE, Accredited by NBA & NAAC, Affiliated to JNTU Kakinada)
BHIMAVARAM – 534 202
2024 – 2025

SHRI VISHNU ENGINEERING COLLEGE FOR WOMEN(A)
(Approved by AICTE, Accredited by NBA & NAAC, Affiliated to JNTU Kakinada)
BHIMAVARAM – 534 202

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

VISHNU
UNIVERSAL LEARNING

CERTIFICATE

*This is to certify that the project entitled "**REAL-TIME VIDEO OBJECT DETECTION USING YOLO AND COMPUTER VISION WITH AUTOMATED WHATSAPP ALERTS FOR SECURITY AND SURVEILLANCE**", is being submitted by **Adapala Lohitha, Ballarapu Glory, Bonam Renuka Lakshmi Tejaswini, Boorlagadda V N S Asritha, Bhupatiraju Harshita** bearing the **Regd.No's. 21B01A0501, 21B01A0511, 21B01A0526, 21B01A0527, 22B05A0501** in partial fulfillment of the requirements for the award of the degree of "**Bachelor of Technology** in **Computer Science & Engineering**" is a record of bonafide work carried out by them under my guidance and supervision during the academic year 2024–2025 and it has been found worthy of acceptance according to the requirements of the university.*

Internal Guide **Head of the Department**

External Examiner

ACKNOWLEDGMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of the people who made it possible and whose constant encouragement and guidance has been a source of inspiration throughout the course of this seminar. We take this opportunity to express our gratitude to all those who have helped us in this seminar.

We wish to place our deep sense of gratitude to **Sri. K. V. Vishnu Raju, Chairman of SVES**, for his constant support on each and every progressive work of mine.

We wish to place our deep sense of gratitude to **Dr. P. Srinivasa Raju, Our Director, Student Affairs, Admin, SVES-Bhimavaram.**

We wish to express our sincere thanks to **Dr. G. Srinivasa Rao, Principal of SVECW**

for being a source of inspiration and constant encouragement.

We wish to express our sincere thanks to **Prof. P. Venkata Rama Raju, Vice-Principal of SVECW** for being a source of inspirational and constant encouragement.

We wish to place our deep sense of gratitude to **Dr. P. Kiran Sree, Head of the Department of Computer Science & Engineering** for his valuable pieces of advice in completing this successfully.

We are deeply indebted and sincere thanks to our **PRC members Dr. P. R. Sudha Rani, Dr. V. V. R. Maheswara Rao, Dr. J. Veeraraghavan, Mr. G. V. S. S. Prasad Raju** for their valuable advice in completing this project successfully.

We are deeply thankful to our **project coordinator Dr. P. R. Sudha Rani, Professor** for her indispensable guidance and unwavering support throughout our project completion.

Our deep sense of gratitude and sincere thanks to our guide **Dr. M. Prasad, Associate Professor** for his unflinching devotion and valuable suggestions throughout our project work.

Project Team:

1. 21B01A0501 - Adapala Lohitha
2. 21B01A0511 - Ballarapu Glory
3. 21B01A0526 - Bonam Renuka Lakshmi Tejaswini
4. 21B01A0527 - Boorlagadda V N S Asritha
5. 22B05A0501 - Bhupatiraju Harshita

ABSTRACT

This project aims to develop an advanced real-time object detection system that can automatically detect human presence using YOLO (You Only Look Once) and OpenCV (Open Source Computer Vision Library). The primary objective of this project is to enhance security surveillance by reducing manual monitoring efforts and ensuring immediate alerts when an intruder or human is detected.

The proposed system captures live video feeds from a camera and processes them using the YOLOv4 model to identify humans. Once a human is detected, the system takes a screenshot of the detected object and sends a WhatsApp alert to authorized personnel using the Twilio API. Additionally, a MySQL database is implemented to store all detected data and images for future reference.

The main advantage of this project is its ability to provide real-time alerts to prevent potential security threats and reduce manual monitoring efforts. By utilizing Computer Vision and Deep Learning, the system can quickly and accurately detect human presence without false alarms. Furthermore, the implementation of a WhatsApp notification system enhances communication efficiency by ensuring that authorized personnel are informed immediately when suspicious activity is detected.

The motivation behind this project is to minimize human error in surveillance, reduce dependency on manual monitoring, and provide a faster response to potential security threats. Unlike traditional surveillance systems, which rely on human supervision, this system works autonomously and efficiently without any human intervention.

This project can be widely implemented in residential security, commercial buildings, banking sectors, industrial premises, and public spaces. The combination of YOLOv4, OpenCV, Twilio API, and MySQL ensures efficient surveillance, real-time alerts, and secure data storage, making this project a major advancement in modern security systems.

Table Of Contents

Image Index

INTRODUCTION

In today’s world, security and surveillance are essential for protecting homes, businesses, and public places. Traditional security systems, such as CCTV cameras, primarily serve as recording tools rather than real-time detection and response mechanisms. These systems rely heavily on human operators to continuously monitor live video feeds and respond manually to any suspicious activity. However, human fatigue, delayed reaction times, and potential oversight make traditional monitoring systems less effective in critical situations. Additionally, storage limitations, the absence of automated alerts, and the inability to differentiate between real threats and false alarms further reduce their reliability.

To overcome these challenges, our project proposes an intelligent, real-time security system that utilizes deep learning and computer vision techniques for automatic intruder detection and instant alerts via WhatsApp. By integrating YOLOv4 (You Only Look Once), OpenCV, Twilio API, and MySQL for face recognition, we have developed a system that detects human presence, records video evidence, and notifies the owner immediately—eliminating the need for manual monitoring.

Security threats such as unauthorized access, burglary, and vandalism require immediate action rather than a delayed response. In traditional systems, when an intruder is detected, security teams must review recorded footage, which can result in significant time loss before taking action. This delay increases the risk of property loss, security breaches, or harm to individuals. Additionally, manual surveillance systems often require continuous human supervision, which is not cost-effective and prone to human error.

To address these limitations, an AI-powered system capable of detecting intrusions autonomously and notifying concerned authorities instantly is highly beneficial. Our system does this by:

- Processing live video streams using OpenCV
- Detecting human presence with YOLOv4
- Capturing screenshots and recording video when an intruder is detected
- Sending WhatsApp alerts using the Twilio API
- Performing face recognition to identify known vs. unknown individuals
- Allowing users to respond to alerts and decide whether to notify emergency contacts

Our system continuously monitors video feeds from a connected webcam. The YOLOv4 model processes each frame to detect humans in real time. If a human is detected, the system takes a screenshot of the intruder, saves it, and sends a WhatsApp alert to the owner and security contacts. The alert includes:

- A link to the captured image for verification
- An option to trigger an emergency response
- Live video recording for additional security evidence

If the owner or authorized personnel confirm a threat by replying "YES", the system immediately sends an emergency alert to law enforcement or security teams. If all contacts reply "NO", the system recognizes it as a false alarm and does not escalate the alert— reducing unnecessary panic.

To enhance security, our system integrates face recognition using MySQL databases. If the detected person is recognized as an authorized individual, the system does not trigger an alert. However, if the person is unknown, the notification is sent for verification.

Our project combines deep learning, computer vision, and cloud communication services to create a robust security solution. The key technologies include:

- YOLOv4: A deep learning-based real-time object detection algorithm known for its speed and accuracy, ideal for identifying humans in video streams.
- OpenCV: A powerful computer vision library that captures live video, processes frames, and performs image-based operations.
- Twilio API: Enables WhatsApp alerts to notify users when an intruder is detected.
- MySQL: Stores face recognition data to differentiate between authorized and unauthorized individuals.
- Ngrok: Provides a public URL to share media files such as screenshots and recorded videos over the internet.
- XAMPP: Hosts captured images and videos for remote access.

Security is a fundamental need, but not everyone can afford expensive security guards or advanced surveillance infrastructure. Our project aims to create an affordable, efficient, and fully automated surveillance system that can be deployed in homes, businesses, and industries without requiring constant human monitoring. By combining deep learning with real-time alerting mechanisms, we provide a modern, technology-driven solution to prevent security breaches and intrusions before they escalate.
The system can be further expanded to integrate additional security features, such as night vision support, cloud-based monitoring, and multi-camera tracking, making it a future-proof solution for modern surveillance needs.

This introduction lays the foundation for our research by explaining the need for automation in surveillance, the role of deep learning, and the technical aspects of our project. The combination of YOLOv4, OpenCV, and Twilio API ensures that our system not only detects intruders but also provides immediate alerts for timely action. The integration of face recognition and emergency alert decision-making makes it a highly efficient security solution suitable for homes, businesses, and industrial applications.

This project aims to bridge the gap between traditional surveillance and modern AI-driven security systems, providing an automated, efficient, and cost-effective solution for real- time monitoring.

2. SYSTEM ANALYSIS

System analysis plays a crucial role in designing and developing an efficient security surveillance system. This phase involves understanding the limitations of existing security systems, proposing a new and improved approach, and evaluating its feasibility from technical, operational, and economic perspectives. The aim is to create a robust and automated solution that minimizes human intervention while improving security monitoring and response times.

Our project, "Real-Time Video Object Detection Using YOLO and Computer Vision with Automated WhatsApp Alerts," is designed to address the shortcomings of traditional surveillance systems by integrating real-time object detection with an instant alert system. The system uses YOLOv4 (You Only Look Once) for object detection, OpenCV for video stream processing, and Twilio API for sending automated WhatsApp alerts.

2.1 Existing System

Most existing surveillance systems rely on CCTV cameras that continuously record video footage. These systems have been used for decades but come with several drawbacks, including:

▪ Manual Monitoring: Traditional security cameras require continuous human supervision, which is time-consuming and prone to human error. Security personnel may miss crucial moments due to fatigue or distractions.

▪ Delayed Response Time: Since surveillance footage is typically reviewed after an incident occurs, real-time intervention is not possible. This delay can result in security breaches, theft, or unauthorized intrusions going unnoticed for extended periods.

▪ No Automated Alerts: Traditional surveillance systems do not provide instant notifications to users. Security personnel must manually analyze recorded footage, which often leads to delayed decision-making.

▪ High Storage Costs: Continuous recording requires a large amount of storage space, making long-term data retention expensive and impractical for some users.

▪ Inefficient in Low-Light Conditions: Many traditional systems struggle with poor visibility in low-light environments, reducing their effectiveness in night-time surveillance.

Existing Technological Solutions and Their Limitations:

Several modern solutions attempt to improve security monitoring, but they also have limitations:

1. **Motion-Based Detection Systems:**
    - Uses motion sensors to trigger an alert when movement is detected.
    - Often produces false alarms due to wind, small animals, or moving objects like shadows and branches.

2. **Facial Recognition-Based Security:**
    - Uses AI to match faces against a database of known individuals.
    - High computational cost, privacy concerns, and the need for continuous database updates.

Given these limitations, there is a need for a smarter, more efficient, and automated security system that can:

- ■ Instantly notify users through WhatsApp alerts.
- ■ Reduce false alarms by accurately detecting human presence.
- ■ Store evidence efficiently with screenshots and recorded footage.
- ■ Allow user verification to decide whether to trigger emergency alerts.

2.2 Proposed System

To overcome the limitations of traditional surveillance systems, our project integrates real-time object detection with automated alerting mechanisms using deep learning and computer vision technologies. The system is designed to:

- Use YOLOv4 to detect humans in a video feed with high accuracy.
- Capture a screenshot when an intruder is detected.
- Send an instant WhatsApp alert using Twilio API.
- Perform real-time face recognition to distinguish known individuals from intruders.
- Record and store footage while a human is present in the frame.
- Allow the owner to decide whether to notify emergency contacts.

Key Features of the Proposed System:

1. **Real-Time Human Detection (YOLOv4 + OpenCV)**
    - YOLOv4 is a fast and efficient object detection model that processes entire video frames in one pass, making it ideal for real-time applications.

- OpenCV is used to capture and process video streams, ensuring smooth integration with surveillance cameras.

2. **Automated WhatsApp Alerts (Twilio API)**
    - When a human is detected, an alert is sent via WhatsApp to the owner.
    - The message includes a screenshot of the intruder and a decision prompt (Reply "YES" to trigger emergency contact, "NO" to dismiss the alert).

3. **Face Recognition for Owner Verification (MySQL Database)**
    - The system compares detected faces with a stored database.
    - If the detected person is recognized, the system does not trigger an alert.
    - If the person is unknown, an alert is sent to the owner for verification.

4. **Continuous Screen Recording**
    - The system records video footage whenever a human is detected.
    - The recording stops only when the human leaves the frame to save storage.

5. **Emergency Response Decision System**
    - If ANY device replies "YES," an emergency alert is sent to law enforcement or security teams.
    - If ALL devices reply "NO," the alert is dismissed.

Advantages of the Proposed System:

- Fast and Accurate: YOLOv4 ensures real-time object detection with minimal computational overhead.
- Automated Alerts: Eliminates manual monitoring by sending instant notifications to users.
- Reduces False Alarms: Uses face recognition to avoid triggering alerts for known individuals.
- Cost-Effective: Requires only a webcam and a basic computer setup, making it an affordable solution for home and business security.
- Scalable: Can be expanded to support multiple cameras for large-scale deployments.

2.3 Feasibility Study

The feasibility study evaluates whether the proposed system is technically, operationally, and economically viable.

Technical Feasibility:

- Hardware Availability: The system requires a webcam, a computer with a basic GPU, and internet access—all of which are readily available.

Software Compatibility:

- The system is built using Python, OpenCV, YOLOv4, and Twilio API, which are open-source and widely supported.
- AI and Machine Learning Integration: The deep learning models used (YOLOv4) are pre-trained and optimized for real-time execution.

Operational Feasibility:

- User Acceptance: The system is easy to use with WhatsApp alerts as the primary communication method.
- Minimal Human Supervision: The system runs automatically, reducing the need for manual monitoring.
- Integration with Existing Infrastructure: Can be deployed alongside existing CCTV systems without additional costs.

Economic Feasibility:

- Low Implementation Cost: Uses free and open-source technologies, reducing development and deployment expenses.
- No Subscription Fees: Unlike cloud-based security solutions, our system does not require a monthly subscription.
- Energy Efficient: Since the system operates only when motion is detected, it conserves power compared to traditional always-on recording systems.

The proposed system effectively addresses the challenges of traditional surveillance systems by providing a fast, automated, and real-time security monitoring solution. The technical, operational, and economic feasibility studies confirm that the system is efficient, scalable, and practical for real-world deployment.

3. SYSTEM REQUIREMENTS SPECIFICATION

System Requirements Specification (SRS) is a pivotal phase in the software development process, acting as a blueprint that outlines the detailed needs and expectations of the proposed system. This section serves as a comprehensive guide to the software and hardware components, as well as the functional capabilities that the system must possess. Through a detailed analysis of the system's requirements, this phase ensures a clear understanding of what is to be developed and provides a foundation for effective project planning and execution.

3.1 Software Requirements:

Software requirements delineate the essential components and specifications that the software must possess to achieve its intended functionality. In the context of our project, the software requirements will encompass the programming languages, frameworks, libraries, and other software tools necessary for implementing the YOLO v3 model, OpenCV integration, and the voice output module. The software requirements specification will guide the development team in choosing the appropriate technology stack to ensure a seamless and efficient software development process.

1. **Operating System (OS) - Windows 8/10:**
    a. Windows 8/10 provides a stable and widely used platform for software development. It offers compatibility with various libraries and tools required for your project.
2. **Programming Language - Python:**
    a. Python is a versatile and easy-to-learn language, making it an excellent choice for rapid development. It offers extensive libraries and frameworks for computer vision, deep learning, and text-to-speech functionalities, which are crucial for your project.
3. **Deep Learning Model - YOLO v4:**
    a. YOLO (You Only Look Once) is a state-of-the-art object detection algorithm that is fast and accurate. YOLO v4 is particularly suitable for real-time applications like yours, where it can quickly detect objects in live video streams, providing essential information for visually impaired individuals.
4. **Datasets - COCO datasets for training:**
    a. COCO (Common Objects in Context) dataset is a large-scale object detection, segmentation, and captioning dataset. By using COCO datasets for training your YOLO v3 model, you ensure that it has been exposed to a diverse range of objects and scenarios, enhancing its ability to recognize objects accurately in various environments.

5. **Libraries:**
   a. **OpenCV:**
      i. OpenCV (Open Source Computer Vision Library) is a powerful library for real-time computer vision tasks. It provides various functionalities for image and video processing, including object detection, which is essential for your project's object detection component.
   b. **NumPy:**
      i. NumPy is a fundamental package for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently. NumPy complements OpenCV and facilitates various data manipulation tasks in your project.
   c. **twilio:**
      i. Twilio is a cloud communications platform that enables developers to integrate messaging, voice, and video communication into applications using APIs. It provides services for sending SMS, making phone calls, and automating messaging via channels like WhatsApp. Twilio’s scalability and reliability make it ideal for real-time notifications, customer engagement, and security alerts. In this project, Twilio’s WhatsApp API is used to send automated alerts when an object of interest is detected, ensuring instant communication for security and surveillance.

6. **IDE that supports Python development:**
   a. An Integrated Development Environment (IDE) that supports Python development, such as PyCharm, Visual Studio Code, or Jupyter Notebook, provides a convenient environment for writing, debugging, and running Python code. It streamlines the development process and enhances productivity by offering features like code highlighting, auto-completion, and debugging tools.

3.2 Hardware Requirements

Hardware requirements encompass the physical components and specifications necessary for the proper functioning of the system. In our project, considerations will be made for the devices that will run the application, such as cameras for capturing visual data, processors capable of handling real-time image processing, and the memory requirements to support the YOLO v3 model and associated software components. The hardware requirements specification serves as a guideline for selecting suitable devices and ensures that the proposed system can operate effectively in a variety of environments.

- Dual-core processor, clocked at a minimum of 2.0 GHz.
- At least 20 GB of available storage space for the system software, models, and other related files.
- Minimum 8 GB RAM to ensure smooth operation and quick access to necessary data.
- A mobile phone.
- Webcam: External or built-in webcam for live video feed integration and real-time object detection.

3.3 Functional Requirements:

Functional requirements define the core capabilities and features of the system to ensure efficient and reliable performance. In this project, the system integrates YOLO for object detection, OpenCV for image processing, and Twilio's WhatsApp API for real-time alerts. These requirements outline the essential functionalities needed to achieve seamless security and surveillance automation.

1. **Object Detection Using YOLO:**

   - **Real-Time Detection:** The system must detect objects in real-time from live video feeds, ensuring prompt identification of security threats.
   - **Multiclass Detection:** The YOLO model should be capable of detecting and categorizing multiple objects, such as humans, vehicles, and suspicious items.
   - **High Accuracy and Precision:** The detection algorithm should minimize false positives and negatives to ensure reliable surveillance.
   - **Environmental Adaptability:** The model should function effectively in different lighting conditions, backgrounds, and weather variations to provide accurate results.

2. **OpenCV Integration:**

   - **Image and Video Processing:** OpenCV must process frames efficiently, extracting relevant features for YOLO-based object detection.
   - **Preprocessing Techniques:** The system should apply necessary preprocessing steps such as resizing, grayscale conversion, and noise reduction to enhance detection performance.
   - **Camera Compatibility:** The system should support multiple camera types, including CCTV, IP cameras, and webcams, ensuring flexible deployment.

3. WhatsApp Alert System via Twilio:

- **Automated Alerts:** The system must send real-time WhatsApp notifications whenever a specific object is detected.
- **Message Customization:** Alerts should include relevant details such as object type, timestamp, and location for security teams.
- **Multiple Recipient Support:** The system should allow notifications to be sent to multiple predefined users, ensuring wider coverage.
- **Reliability and Speed:** Messages should be delivered instantly without significant

delays to facilitate immediate response.

These functional requirements ensure the system delivers real-time, accurate object detection and automated alerts, making it a valuable tool for enhanced security and surveillance operations.

4. SYSTEM DESIGN

System design is a critical phase in the software development lifecycle where the conceptualization from earlier stages begins to take concrete form. It involves the creation of a detailed blueprint that outlines the architecture, components, modules, and interactions within the system. This phase not only guides the development team but also serves as a communication tool, ensuring a shared understanding of the system's structure and functionality among stakeholders.

4.1 Introduction to Software Design

Software design encompasses the process of transforming user requirements into a well-organized and structured set of instructions for the development team. It involves making strategic decisions about system architecture, data flow, user interfaces, and functionality to ensure that the end product aligns with the project's goals. A well-crafted software design not only facilitates efficient coding but also enhances the system's maintainability, scalability, and overall performance.

System Design:

The system design for this security solution encompasses both the architectural framework and the software design, ensuring that all components work cohesively to deliver a robust and efficient application. The design is structured to facilitate real-time human detection, alert generation, face recognition, and video recording, while also ensuring scalability, maintainability, and security.

Architectural Overview

The architecture of the system is based on a modular approach, allowing for the separation of concerns and the independent development of various components. The key components of the system architecture include:

Camera Module:

This module captures real-time video feeds from a fixed camera positioned in the monitored area. It serves as the primary input source for human detection and face recognition.
Human Detection and Face Recognition Module:

Utilizing OpenCV and face_recognition libraries, this module processes the video feed to detect human presence and recognize faces. It employs deep learning algorithms to ensure accurate detection and recognition, maintaining a database of known faces stored in a MySQL database.

Alert Management Module:

This module is responsible for generating alerts when a human is detected. It integrates with the Twilio API to send messages and media to the owner's devices. The alert management system also handles user responses, determining whether to initiate an emergency alert based on the owner's feedback.

Video Recording Module:

This component manages the recording of video footage while a human is detected. It saves the recordings in a specified directory and generates public URLs for easy access via Ngrok.
Database Module:

The database module interacts with a MySQL database to store and retrieve face encodings and associated metadata. It provides functions for adding new faces, retrieving existing faces, and managing the overall data integrity.

User Interface Module:

This module provides a user-friendly interface for the owner to interact with the system. It displays real-time alerts, video feeds, and allows for user responses to alerts.

Software Design:

The software design focuses on the implementation of the system's functionalities through well-defined classes, functions, and data structures. The design adheres to principles of modularity, encapsulation, and reusability, ensuring that each component can be developed and tested independently.

Modular Design:

Each functional component of the system is encapsulated within its own module (e.g., take_screenshot.py, send_alert.py, register_face.py, etc.). This modularity allows for easier maintenance and updates, as changes to one module do not directly affect others.

Class and Function Structure:

The system employs a combination of classes and functions to organize code logically. For instance, functions for human detection, alert sending, and face recognition are grouped within their respective modules, promoting code clarity and reusability.

Data Flow:

The data flow within the system is designed to be efficient. The camera module captures frames, which are processed by the human detection and face recognition module. Detected events trigger alerts, which are managed by the alert management module. User responses are captured and processed to determine the next course of action.

Error Handling:

Robust error handling mechanisms are implemented throughout the system to manage potential issues, such as camera access failures, database connection errors, and API call failures. This ensures that the system remains operational and can recover gracefully from unexpected situations.

Security Measures:

The software design incorporates security measures to protect sensitive data, including encryption for stored face encodings and secure API communication. Compliance with privacy regulations is also considered in the design to safeguard user information.

Testing and Validation:

The system design includes provisions for testing and validation of each module. Unit tests are implemented to verify the functionality of individual components, while integration tests ensure that the modules work together seamlessly.

By adhering to this structured system and software design, the project aims to deliver a reliable, efficient, and user-friendly security solution that meets the needs of homeowners while addressing contemporary security challenges. The modular architecture and thoughtful software design facilitate future enhancements and scalability, ensuring that the system can adapt to evolving requirements.

4.3 UML Diagrams

Use-oriented techniques are widely used in software requirement analysis and design. Use cases and usage scenarios facilitate system understanding and provide a common language for communication. This paper presents a scenario-based modeling technique and discusses its applications. In this model, scenarios are organized hierarchically and they capture the system functionality at various abstraction levels including scenario groups, scenarios, and sub-scenarios.

Combining scenarios or sub-scenarios can form complex scenarios. Data are also separately identified, organized, and attached to scenarios. This scenario model can be used to cross check with the UML model. It can also direct systematic scenario-based testing including test case generation, test coverage analysis with respect to requirements, and functional regression testing.

An object contains both data and methods that control the data. The data represents the state of the object. A class describes an object and they also form a hierarchy to model the real-world system. The hierarchy is represented as inheritance and the classes can also be associated in different ways as per the requirement. Objects are the real-world entities that exist around us and the basic concepts such as abstraction, encapsulation, inheritance, and polymorphism all can be represented using UML.UML is powerful enough to represent all the concepts that exist in object-oriented analysis and design.

The Primary goals in the design of the UML are as follows:

- Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
- Provide extendibility and specialization mechanisms to extend the core concepts.
- Be independent of particular programming languages and development processes.
- Provide a formal basis for understanding the modeling language.
- Encourage the growth of the OO tools market.
- Support higher level development concepts such as collaborations, frameworks, patterns and components.
- Integrate best practices.

Use case diagram:

A use case diagram in the unified modeling language (uml) is a type of behavioral diagram defined by and created from a use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

Following are the purposes of a use case diagram given below:

- It depicts the external view of the system.
- It recognizes the internal as well as external factors that influence the system.
- It represents the interaction between the actors.

Use case diagrams commonly contains

- Use cases
- Actors
- Dependency, generalization and association relationships.

Use cases: A use case is a software and system engineering term that describes how a user uses a system to accomplish a particular goal.

Actors: An actor is a person, organization or external system that plays a role in one more interaction with the system

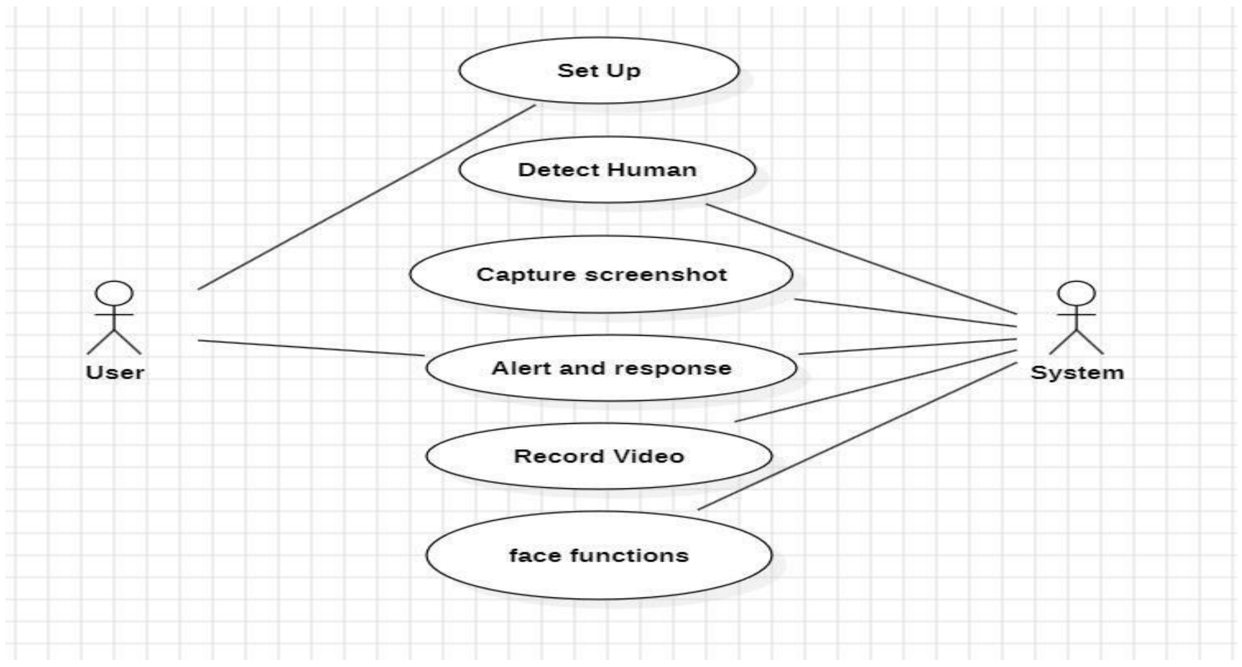


Fig 4.3.1 Use Case diagram for our system

In the context of "Real-time Video Object Detection using YOLO and Computer Vision with automated WhatsApp alerts for security and surveillance," the primary actors are the "User" and the "System." Here's a breakdown of the identified use cases:

Set Up:

- Actor(s): User, System
- Description: User configures camera feed, detection zones, WhatsApp API settings, etc. System initializes YOLO model and environment.

Detect Human (Object Detection):

- Actor: System
- Description: YOLO detects objects (e.g., human, vehicle) in real-time from the video stream.

Capture Screenshot:

- Actor: System
- Description: When a human or object of interest is detected, the system captures a screenshot of the frame.

Alert and Response:

- Actor: User, System
- Description: System sends an automated WhatsApp alert with the image to the user. The user may acknowledge or act on the alert.

Record Video:

- Actor: System
- Description: Continuous or event-triggered video recording is performed and stored for later review.

Face Functions (e.g., Recognition, Blurring):

- Actor: System
- Description: Advanced facial recognition (for known/unknown persons) or privacy functions like face blurring, depending on design.

This Use Case Diagram efficiently outlines the flow of actions between the User and the System, with most operations automated by the system. The YOLO object detection model is central to triggering downstream tasks like screenshot capture, alerts via WhatsApp, and video recording.

Class diagram:

In software engineering, a class diagram in the unified modeling language (uml) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.
It depicts a static view of an application. It represents the types of objects residing in the system and the relationships between them.
A class consists of its objects, and also it may inherit from other classes.
A class diagram is used to visualize, describe, document various different aspects of the system, and also construct executable software code.
It shows the attributes, classes, functions, and relationships to give an overview of the software system. It constitutes class names, attributes, and functions in a separate compartment that helps in software development.
Since it is a collection of classes, interfaces, associations, collaborations, and constraints, it is termed as a structural diagram.

Class diagram contains
• Classes
• Interfaces
• Dependency, generalization and association

Class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application. A collection of class diagrams represent the whole system.
The following points should be remembered while drawing a class diagram –
• The name of the class diagram should be meaningful to describe the aspect of the system.
• Each element and their relationships should be identified in advance.
• Responsibility (attributes and methods) of each class should be clearly identified
• For each class, a minimum number of properties should be specified, as unnecessary properties will make the diagram complicated.
• Use notes whenever required to describe some aspect of the diagram. At the end of the drawing it should be understandable to the developer/coder.
• Finally, before making the final version, the diagram should be drawn on plain paper and reworked as many times as possible to make it correct.

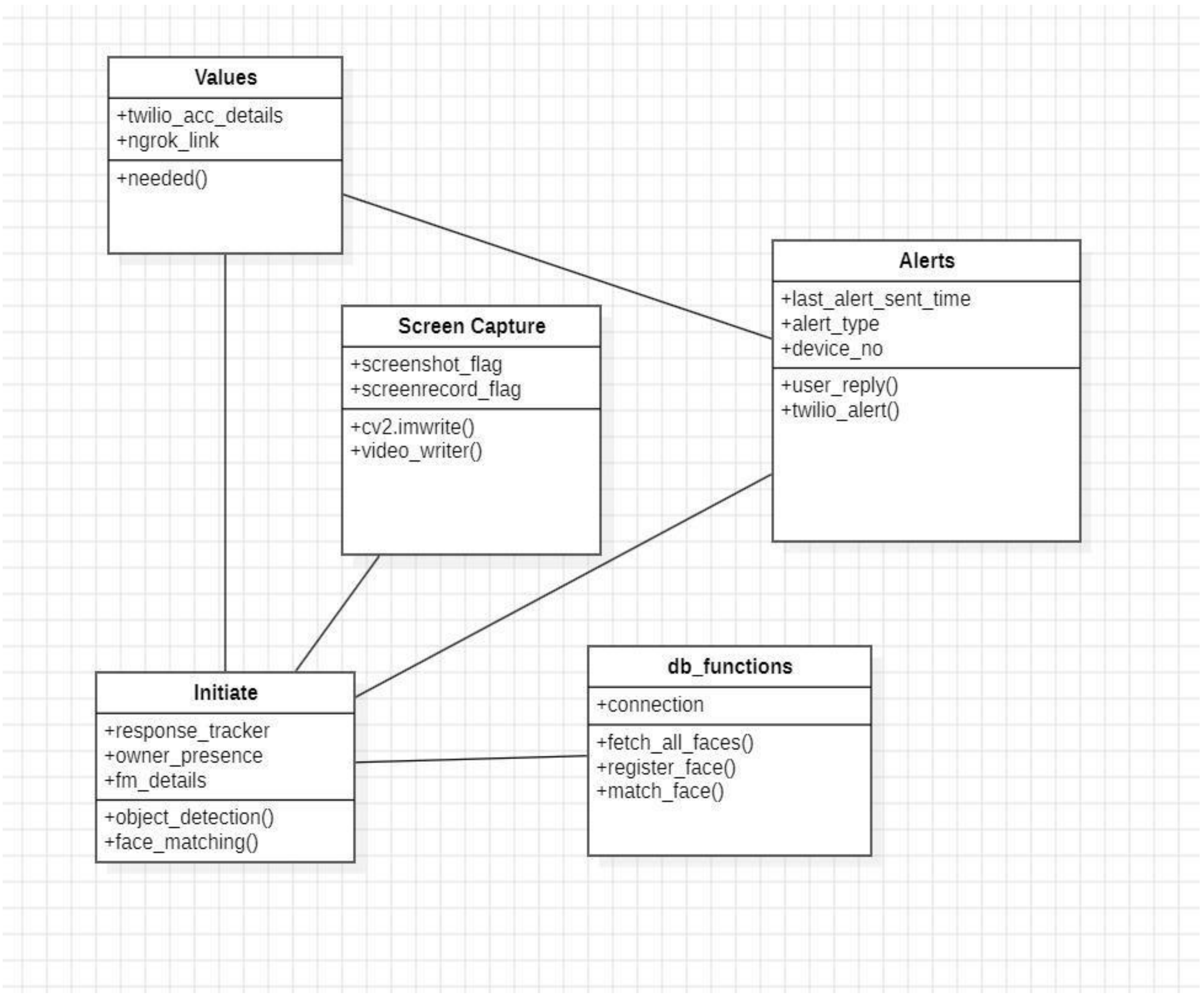


Fig 4.3.2 Class Diagram for our system

This class diagram represents the backend structure of the "Real-time Video Object Detection using YOLO and Computer Vision with automated WhatsApp alerts" system. It illustrates how different classes interact for object detection, face matching, alerting, and capturing visuals (screenshots/videos), while also managing credentials and user data.

- **Values:**
  - **Purpose:**
    - Stores external configuration and credentials.
  - **Attributes:**
    - twilio_acc_details: Stores Twilio account information for sending WhatsApp alerts.
    - ngrok_link: Stores Ngrok public URL for exposing the local server to the internet.
  - **Methods:**
    - needed(): Returns or sets required config values.
- **Screen Capture:**
  - **Purpose:**
    - Handles visual data capturing.
  - **Attributes:**
    - screenshot_flag: Boolean to control screenshot capturing.
    - screenrecord_flag: Boolean to control video recording.
  - **Methods:**
    - cv2.imwrite(): Saves screenshots using OpenCV.
    - video_writer(): Handles video recording and writing frames.
- **Alerts:**
  - **Purpose:**
    - Manages alert generation and user communication.
  - **Attributes:**
    - last_alert_sent_time: Timestamp of the last alert sent.
    - alert_type: Type of alert (e.g., intrusion, unknown face).
    - device_no: Target device number for WhatsApp alert.
  - **Methods:**
    - user_reply(): Handles user response to alerts.
    - twilio_alert(): Sends alert using Twilio API.
- **Initiate:**
  - **Purpose:**
    - Main entry point for detection and matching functionalities.
  - **Attributes:**
    - response_tracker: Tracks user responses or decisions.
    - owner_presence: Flags if the known user is present.
    - fm_details: Face matching details for validation.
  - **Methods:**
    - object_detection(): Detects objects using YOLO or similar model.
    - face_matching(): Matches detected faces with known records.

- **db_functions:**
    - **Purpose:**
        - Interacts with the facial recognition database.
    - **Attributes:**
        - connection: Database connection instance.
    - **Methods:**
        - fetch_all_faces(): Retrieves all registered faces.
        - register_face(): Adds a new face to the database.
        - match_face(): Matches input face with existing records.

Sequence diagram:

A sequence diagram in unified modeling language (uml) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a message sequence chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

It represents the flow of messages in the system and is also termed as an event diagram. It helps in envisioning several dynamic scenarios.

It portrays the communication between any two lifelines as a time-ordered sequence of events, such that these lifelines took part at the run time.

In UML, the lifeline is represented by a vertical bar, whereas the message flow is represented by a vertical dotted line that extends across the bottom of the page. It incorporates the iterations as well as branching.

Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.

Purpose of Sequence Diagram

- Model high-level interaction between active objects in a system.
- Model the interaction between object instances within a collaboration that realizes a use case.
- Model the interaction between objects within a collaboration that realizes an operation
- Either model generic interactions (showing all possible paths through the interaction) or specific instances of an interaction (showing just one path through the interaction) .

This sequence diagram represents a basic scenario where the user starts the video capture, and the YoloObjectDetector continuously captures frames, detects objects, performs non-maximum suppression, draws bounding boxes on the frame, and speaks out the detected objects. The interactions involve method calls and messages between the user, the object detector, and the video capture component.

- The User initiates the security process by sending a command to the System.
- The System enables and starts monitoring via the connected Camera.
- The Camera captures a human presence, sending this information back to the System.
- Upon detection, the System sends an alert to the User for validation or action
- The User responds to the alert (e.g., verifying if it's a known person).
- If no response or in critical situations, the System triggers an Emergency Alert to the Emergency contact.

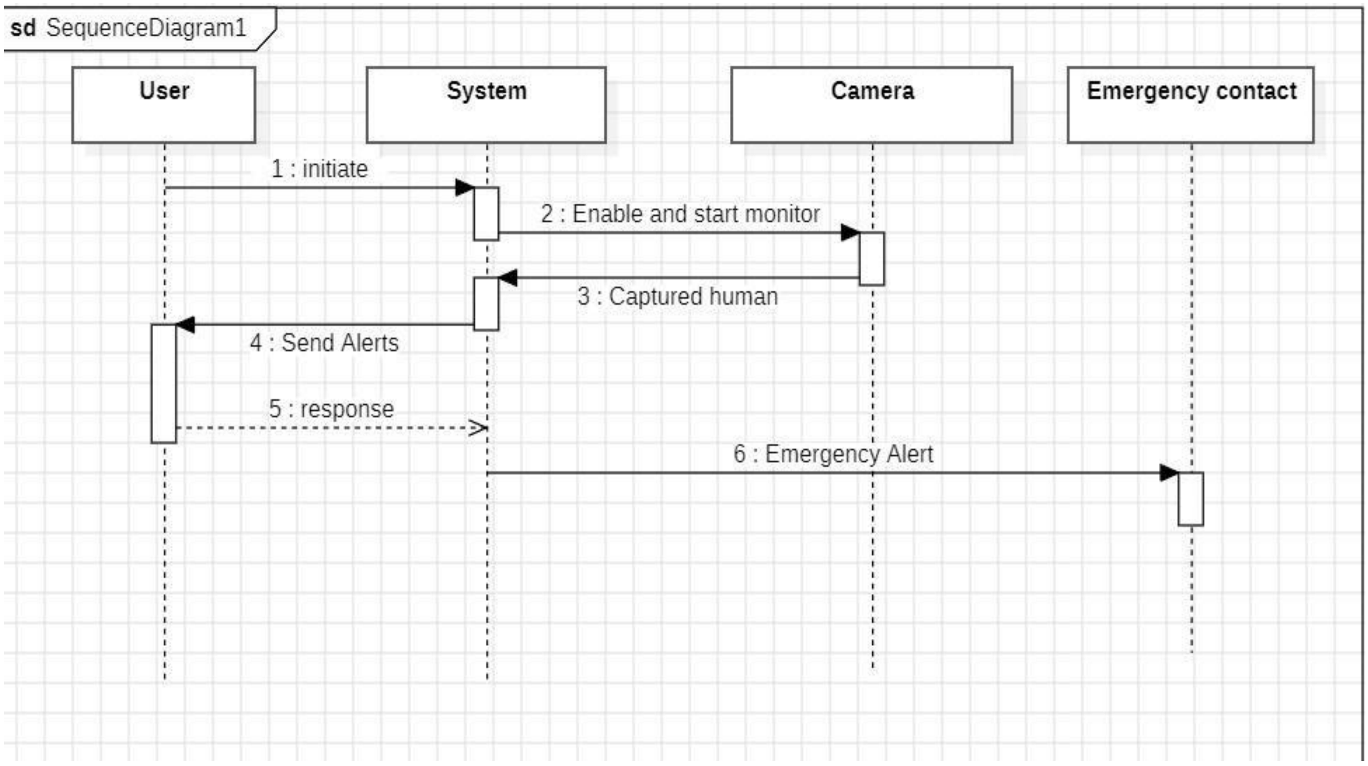


Fig 4.3.3 Sequence Diagram for our system

Deployment diagram:

The deployment diagram visualizes the physical hardware on which the software will be deployed. It portrays the static deployment view of a system.

- It involves the nodes and their relationships.
- It ascertains how software is deployed on the hardware.
- It maps the software architecture created in design to the physical system architecture, where the software will be executed as a node.

Since it involves many nodes, the relationship is shown by utilizing communication paths. The main purpose of the deployment diagram is to represent how software is installed on the hardware component. It depicts in what manner a software interacts with hardware to perform its execution.

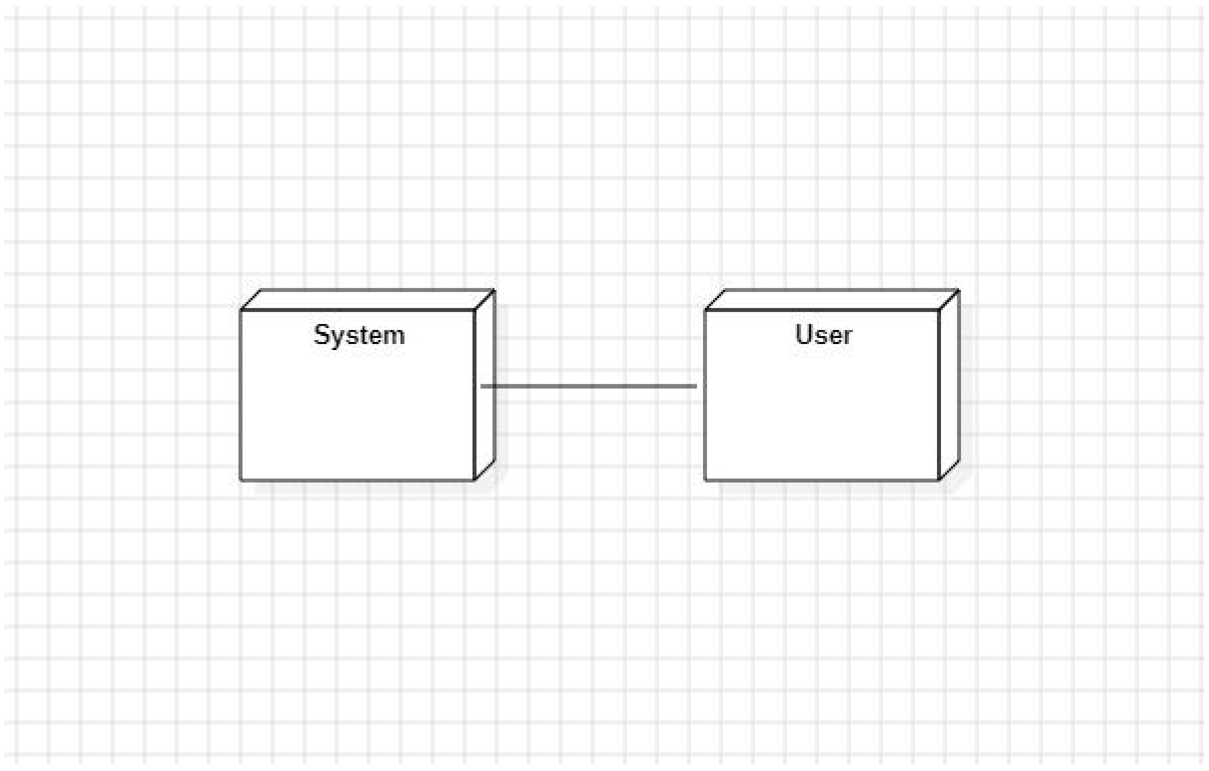


Fig 4.3.4 Deployment Diagram for our system

Component diagram:

A component diagram is a type of structural diagram in the Unified Modeling Language (UML) that depicts the high-level components or modular parts of a system and their interactions. It provides a visual representation of the organization and relationships between components within a system or software application.

At its core, a component diagram consists of various components, each representing a distinct modular unit or building block of the system. These components encapsulate functionality and behavior, and they can be of different types, such as classes, modules, subsystems, or even entire software systems.

The connections between components are represented by lines, called connectors, which illustrate the dependencies and relationships between components. These relationships can include dependencies, associations, aggregations, or compositions, depending on the nature of the interaction between the components. For example, a dependency relationship might indicate that one component relies on the functionality provided by another component.

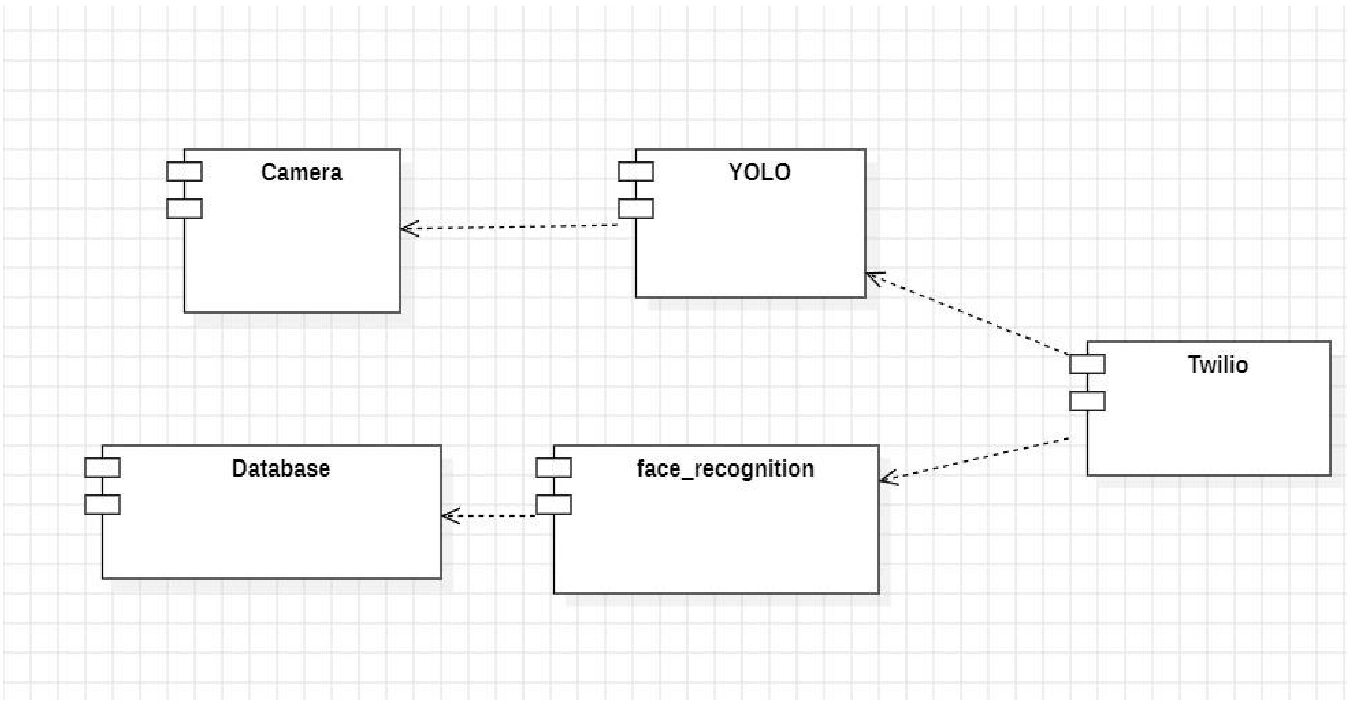


Fig 4.3.5 Component Diagram for our system

5. SYSTEM IMPLEMENTATION

Key Aspects of System Implementation:

1. **Coding and Programming:**
   During the implementation phase, developers write the actual code based on the design specifications. This involves translating the system's architecture, modules, and functionalities into programming languages such as Python, which is commonly used in machine learning and computer vision projects.

2. **Integration of Components:**
   Integrating different components is crucial to ensure that the system operates as a cohesive whole. In our project, this involves bringing together the YOLO v4 model for object detection, the OpenCV library for image and video processing, and the voice output module into a unified application.

3. **Testing:**
   Testing is an integral part of the implementation process. Developers conduct unit testing to ensure individual components work correctly and integration testing to verify the interactions between different modules. In our project, testing involves validating the accuracy of object detection, assessing real-time processing capabilities, and confirming the effectiveness of the voice output module.

4. **Refinement and Optimization:**
   As implementation progresses, developers may identify areas for refinement and optimization. This could involve improving the efficiency of algorithms, optimizing code for better performance, or addressing any issues that arise during testing. The goal is to fine-tune the system for optimal functionality.

5. **Documentation:**
   Comprehensive documentation is created during the implementation phase to provide insights into the codebase, explain algorithms and methodologies, and guide future development or maintenance. Documentation is vital for ensuring the system's sustainability and facilitating collaboration among team members.

The system’s architecture is modular, with each component functioning independently while contributing to the overall objective of real-time surveillance. The live video feed is captured using a standard webcam and processed using OpenCV. The cv2.VideoCapture() function initializes the camera, and frames are continuously retrieved for further analysis. The captured frames are then passed to the YOLOv4 object detection model, which has been pre-trained to recognize multiple objects with high accuracy. The model processes each frame by first converting it into a blob format, then performing a forward pass through the neural network to extract object predictions. Non-Maximum Suppression (NMS) is applied to remove duplicate detections, ensuring that only the most relevant objects remain. The system primarily focuses on detecting humans, as this is the trigger for sending alerts and activating security measures.

When an unauthorized human presence is detected, the system automatically captures a screenshot of the frame. This is achieved using the cv2.imwrite() function, which saves the image in a local directory. To enable remote access to these images, XAMPP is used to host them on a local server, and Ngrok is employed to generate a public URL. This URL is then embedded in the alert message sent to predefined contacts via WhatsApp. The Twilio API facilitates automated messaging, ensuring that security personnel or property owners receive real-time alerts. The alert message contains the detected screenshot and a prompt asking the recipients whether an emergency response should be triggered. If any recipient replies with "Yes," the system escalates the alert, notifying emergency services or additional contacts.

To enhance the system’s reliability and minimize false alarms, a face recognition module is integrated. Using the face_recognition library, the system extracts facial features from detected individuals and compares them against a stored database in MySQL. If the detected face matches an authorized user, the alert is suppressed to avoid unnecessary notifications. Conversely, if the face is unrecognized, a new entry is dynamically registered for future reference. Additionally, the system includes a screen recording module that activates whenever a human is detected. The recording continues until the person exits the frame, ensuring that all activities are documented for security review. These recorded videos are stored securely and can be accessed remotely as needed.

The overall workflow of the system is as follows: the webcam continuously captures video, YOLOv4 processes each frame to detect objects, and if a human is found, a screenshot is taken and analyzed for facial recognition. An alert is then sent to designated contacts with an option to confirm an emergency. If confirmed, an escalation message is triggered, and screen recording is initiated until no humans are detected. Throughout the process, data is logged in a MySQL database for future analysis. This implementation creates a highly efficient and automated surveillance system that significantly enhances security monitoring. By integrating real-time object detection, automated alerting, and facial recognition, the system ensures prompt responses to potential threats. Future improvements could include expanding support for multiple cameras, improving detection under low-light conditions, and implementing cloud storage for greater scalability and accessibility.

The implementation of our project, "Real-Time Video Object Detection Using YOLO and Computer Vision with Automated WhatsApp Alerts," involves integrating multiple components to ensure a seamless, automated, and highly responsive surveillance system. The system is built on a modular architecture where each component performs a specialized function while working together to achieve the main goal: detecting human intrusions in real-time and sending alerts to the concerned individuals. The core technologies used include YOLOv4 for object detection, OpenCV for handling live video streams, Twilio API for sending alerts via WhatsApp, Ngrok for making locally stored images accessible through the internet, MySQL for face recognition data storage, and Python as the primary programming language.

The live video capture module is the first stage of the system, responsible for acquiring a continuous video stream from a webcam. Using OpenCV's cv2.VideoCapture(), the system initializes the camera and retrieves frames in real-time. Each captured frame is then preprocessed before passing it to the YOLOv4 object detection module. This preprocessing involves resizing the frame to match YOLO’s input size, converting it into a blob, and normalizing pixel values. These transformations enhance detection speed and accuracy.

The YOLOv4 detection module is the heart of the system. It processes each frame and detects objects within the scene using a deep neural network trained on the COCO dataset. The model applies Non-Maximum Suppression (NMS) to eliminate redundant bounding boxes and ensures only the most relevant objects are highlighted. Among the many detected objects, the system focuses on humans, as identifying unauthorized persons is the key purpose of this implementation.

Once a person is detected, the system takes immediate security measures. The first action is to capture a screenshot of the intruder, which is stored locally using OpenCV's cv2.imwrite() function. Since the system needs to send this image via WhatsApp, Ngrok is used to create a public URL that enables remote access to the locally saved screenshot. The next step involves automatically sending an alert through WhatsApp using the Twilio API. This alert message is sent to predefined phone numbers and contains the image of the detected person along with a notification that an unknown individual has been spotted.

An important feature of the system is the face recognition module, which ensures that alerts are only sent for actual threats. Using the face_recognition library, the system extracts facial features from detected individuals and cross-references them with a MySQL database of known users. If the person is identified as a trusted individual, the alert is suppressed. However, if the face is unrecognized, the system registers it as a new entry for future comparisons. This mechanism reduces false alarms and makes the surveillance system more intelligent.

The screen recording module further enhances security by documenting any intrusion events. If an unauthorized person is detected, the system automatically starts recording the live feed, storing the footage in a secure local directory. The recording continues until the person exits the frame, ensuring that all movements are logged. This feature provides valuable evidence that can be reviewed later.

The workflow of the system is structured as follows:

- The webcam continuously captures video frames.
- YOLOv4 processes each frame and detects objects.
- If a human is detected, the system:
    - ✓ Takes a screenshot and generates a public URL using Ngrok.
    - ✓ Checks for face recognition in the MySQL database.
    - ✓ Sends a WhatsApp alert using Twilio.
- The user receiving the alert can confirm whether emergency services should be notified.
- If an emergency is confirmed, additional security notifications are sent.
- Screen recording is activated when a person remains in the frame.
- All captured data is stored securely for future reference.

The system implementation ensures that real-time detection and alerting are achieved with minimal delay, providing an efficient solution for security monitoring. This robust approach, combining deep learning-based object detection, face recognition, and automated messaging, significantly enhances surveillance efficiency and response time. Future improvements could include multi-camera support, night vision enhancements, and integration with IoT-based security systems to provide an even more comprehensive security framework.

5.1 Introduction

System implementation marks a significant juncture in our project's journey, where the meticulously crafted design specifications come to life through coding and development. This phase is instrumental in transforming the conceptual framework into a tangible, functional system that holds the potential to empower visually impaired individuals in their daily lives.

Translating Design into Code:

At the heart of the implementation phase lies the translation of design blueprints into actual code. Developers will embark on the task of writing Python code to bring to fruition the envisioned integration of the YOLO v4 model, OpenCV, and the voice output module. The intricacies of the YOLO v4 model, known for its object detection prowess, will be intricately woven into the codebase. Simultaneously, OpenCV's capabilities for image and video processing will be harnessed to ensure seamless interaction with the visual data captured by the system. The implementation phase demands a harmonious collaboration of coding skills, machine learning expertise, and an acute understanding of computer vision techniques.

Integration of Technological Components:

The seamless integration of various technological components is fundamental to the success of our project, *Real-Time Video Object Detection Using YOLO and Computer Vision with Automated WhatsApp Alerts for Security and Surveillance*. This process involves the effective combination of YOLO's object detection capabilities with OpenCV's image processing features while ensuring smooth communication with Twilio's WhatsApp API for real-time notifications. The coordination between these technologies ensures a highly responsive and efficient security solution.

The YOLO model is trained and optimized to detect multiple objects in real-time with high accuracy. OpenCV processes video frames, ensuring they are appropriately preprocessed before being passed to the YOLO model for detection. Once an object of interest is detected, Twilio's API is triggered to send automated WhatsApp alerts containing relevant information, such as timestamps and detected object categories. This integration facilitates instant threat recognition and notification, making the system highly effective in surveillance applications.

Testing and Iterative Refinement:

To ensure robustness and reliability, the system undergoes rigorous testing at multiple levels. Unit testing is conducted on individual components, such as the YOLO model, OpenCV functions, and Twilio messaging API, to verify their standalone performance. Integration testing evaluates the seamless interaction between these components, ensuring data flows correctly from video processing to object detection and automated messaging.

Simulated real-world conditions, such as varying lighting, crowded environments, and different camera angles, are used to validate the system’s performance. Any false detections or delays in messaging are addressed through iterative refinement, improving accuracy, efficiency, and overall responsiveness. Feedback from initial test runs is incorporated into the system, ensuring continuous enhancements before full-scale deployment.

Optimization for Real-World Usage:

For the system to function effectively in real-world surveillance environments, optimization is key. Latency reduction is prioritized to ensure near-instant detection and alert transmission. The system is optimized to process video feeds efficiently, minimizing computational overhead while maintaining high accuracy.

The detection model is fine-tuned to handle diverse environmental conditions, including varying brightness levels, occlusions, and different object sizes. The WhatsApp notification system is also designed to be lightweight and scalable, ensuring real-time message delivery even in high-alert scenarios. By addressing these practical considerations, the system becomes a reliable and scalable solution for security and surveillance applications.

Documentation for Sustainability:

A well-documented system is essential for future maintenance, scalability, and further enhancements. Throughout the implementation process, detailed documentation is maintained, covering:

- **System architecture and workflow**, explaining how different components interact.
- **Model training and optimization techniques**, ensuring future improvements in detection accuracy.
- **API integration guidelines**, detailing the interaction with Twilio's messaging service.
- **Codebase structure and troubleshooting guidelines**, allowing for easy debugging and enhancements.

This documentation ensures that future developers, researchers, or system administrators can understand, modify, and expand the project as needed. As the implementation phase progresses, the project transitions from conceptual design to a fully functional system that delivers real-time security monitoring and automated alerts, significantly enhancing situational awareness in surveillance applications.

As the implementation phase unfolds, the vision outlined in the design phase materializes into a tangible solution that holds the potential to significantly enhance the lives of visually impaired individuals.

5.2 Project Modules

The project's architecture is organized into distinct modules, each serving a specific purpose and contributing to the overall functionality of the system. These modules are intricately designed to collaborate seamlessly, providing a cohesive and effective solution for visually impaired users. Let's explore the key modules that constitute the foundation of our project.

Object Detection Module:

The Object Detection module forms the backbone of the project, leveraging the state-of-the-art YOLO (You Only Look Once) v4 model. This module is responsible for real-time identification and localization of objects within the captured visual data. The YOLO v4 model excels in its ability to swiftly process images and video frames, offering accurate and rapid object detection capabilities. The output of this module serves as the foundation for subsequent interactions within the system.

Image and Video Processing Module (OpenCV):

The Image and Video Processing module, powered by the OpenCV library, plays a crucial role in enhancing and refining the visual data obtained from the Object Detection module. OpenCV facilitates tasks such as image manipulation, filtering, and preprocessing, ensuring that the input data is optimized for accurate object detection. This module acts as a vital intermediary, preparing the visual information before it is presented to the user. The seamless integration of OpenCV enhances the overall efficiency and versatility of the system.

Twilio Module:

The Twilio module plays a crucial role in enabling real-time communication in our system by integrating automated WhatsApp alerts for object detection events. Twilio’s API provides a seamless way to send notifications, ensuring timely updates whenever an object of interest is detected. This enhances the efficiency of security and surveillance applications by providing instant alerts to concerned personnel.

Integration with the System:

The Twilio module is integrated into the system using its Programmable Messaging API, which allows messages to be sent via WhatsApp. The integration follows these key steps:

- *Twilio Account Setup* – A Twilio account is configured with a verified WhatsApp number to send messages.
- *API Authentication* – Secure authentication using Twilio's Account SID and Auth Token ensures safe and authorized communication.
- *Message Formatting* – The system generates a structured message containing details like object type, detection timestamp, and an image capture (optional).
- *Automated Trigger* – Whenever YOLO detects an object, the system triggers the Twilio API to send a real-time WhatsApp alert to predefined contacts.

Key Features and Benefits:

- *Real-time Notifications* – Immediate alerts ensure prompt action in surveillance scenarios.
- *Reliability and Scalability* – Twilio's cloud-based architecture ensures message delivery even under high loads.
- *Secure and Encrypted Communication* – Twilio ensures that messages remain secure through authentication mechanisms.
- *Customizable Messaging* – The content of alerts can be modified to include text, images, or links based on requirements.

By leveraging Twilio's messaging capabilities, our system ensures a responsive and proactive security mechanism, providing real-time awareness and enhancing surveillance efficiency.

OpenCV Module:

The OpenCV (Open-Source Computer Vision) module plays a pivotal role in our project, serving as a dynamic and versatile tool for image and video processing. As a critical component of the implementation phase, the integration of OpenCV augments the system's capabilities, ensuring efficient handling of visual data captured by the device's camera.

Key Functions of the OpenCV Module:

1. Real-time Object Detection:
OpenCV brings to the project the capability for real-time object detection, a fundamental requirement for empowering visually impaired individuals in navigating their surroundings. Leveraging the YOLO (You Only Look Once) v4 model for object detection, OpenCV processes the live video feed captured by the camera, identifying and localizing objects in real-time.

2. Image Processing Algorithms:
The module incorporates a rich set of image processing algorithms that enhance the quality and interpretability of visual data. These algorithms contribute to refining the accuracy of object detection and ensuring that the system can effectively recognize a diverse array of objects in various environments.

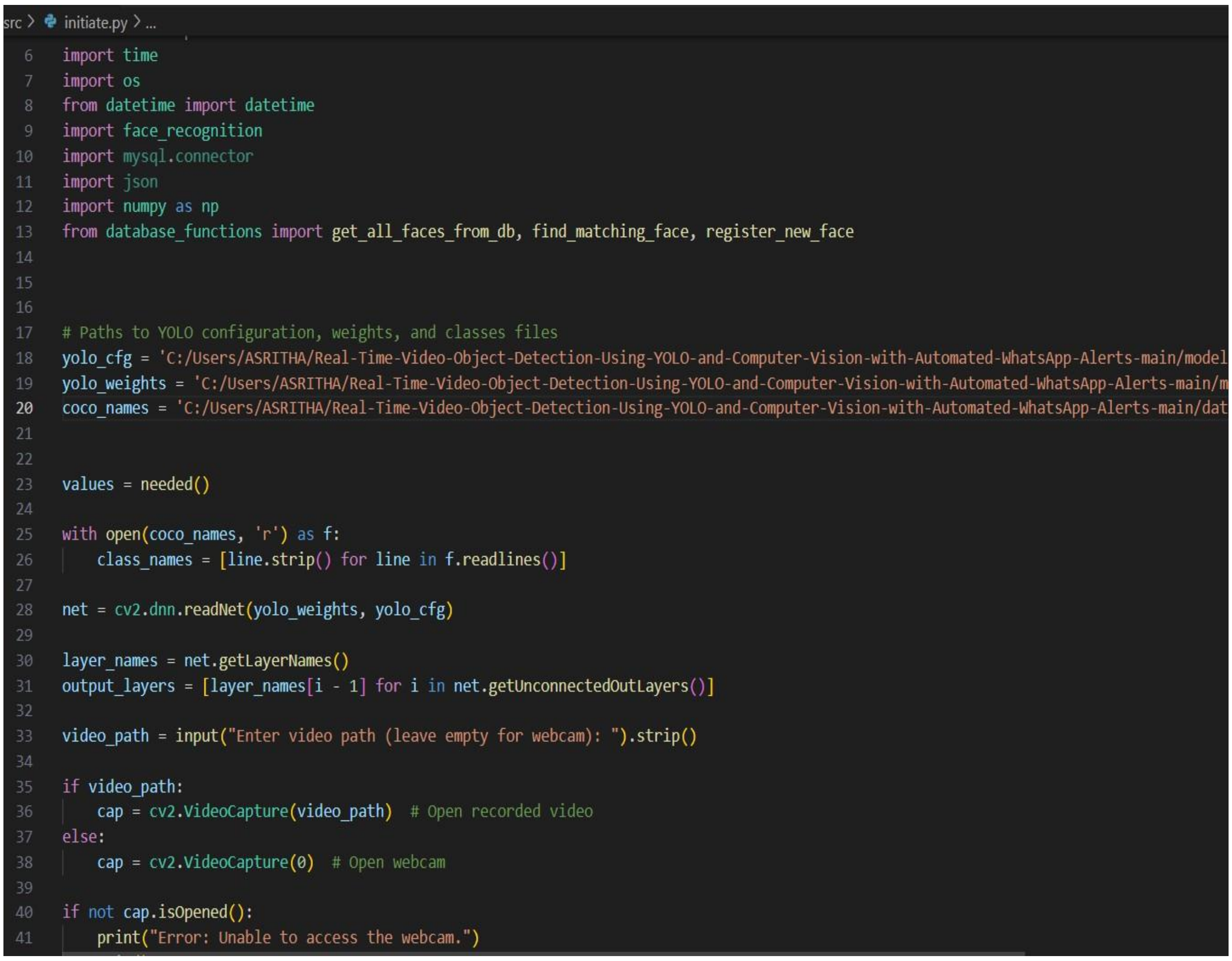
3. Feature Extraction:
OpenCV facilitates the extraction of essential features from images, enabling the system to discern key characteristics of detected objects. This feature extraction is crucial for providing users with meaningful and relevant information about the objects in their vicinity.

4. Adaptability to Dynamic Environments:
An inherent strength of OpenCV lies in its adaptability to dynamic environments. The module equips the system to handle changes in lighting conditions, varying perspectives, and different types of objects, making it a robust solution for real-world scenarios where visually impaired users navigate through diverse surroundings.

5. Integration with YOLO v4 Model:
OpenCV seamlessly integrates with the YOLO v4 model, forming a cohesive unit that capitalizes on the strengths of both technologies. The YOLO v4 model provides accurate and rapid object detection, while OpenCV adds a layer of versatility, allowing for comprehensive image and video processing.

Implementation Considerations:

The integration of the OpenCV module involves configuring it to effectively communicate with the YOLO v4 model and to process the visual data in real-time. The module must be optimized to operate on the selected hardware, ensuring efficient resource utilization and minimal latency in object detection.


src > initiate.py > ...

```python
import time
import os
from datetime import datetime
import face_recognition
import mysql.connector
import json
import numpy as np
from database_functions import get_all_faces_from_db, find_matching_face, register_new_face



# Paths to YOLO configuration, weights, and classes files
yolo_cfg = 'C:/Users/ASRITHA/Real-Time-Video-Object-Detection-Using-YOLO-and-Computer-Vision-with-Automated-WhatsApp-Alerts-main/model
yolo_weights = 'C:/Users/ASRITHA/Real-Time-Video-Object-Detection-Using-YOLO-and-Computer-Vision-with-Automated-WhatsApp-Alerts-main/m
coco_names = 'C:/Users/ASRITHA/Real-Time-Video-Object-Detection-Using-YOLO-and-Computer-Vision-with-Automated-WhatsApp-Alerts-main/dat


values = needed()

with open(coco_names, 'r') as f:
    class_names = [line.strip() for line in f.readlines()]

net = cv2.dnn.readNet(yolo_weights, yolo_cfg)

layer_names = net.getLayerNames()
output_layers = [layer_names[i - 1] for i in net.getUnconnectedOutLayers()]

video_path = input("Enter video path (leave empty for webcam): ").strip()

if video_path:
    cap = cv2.VideoCapture(video_path)  # Open recorded video
else:
    cap = cv2.VideoCapture(0)  # Open webcam

if not cap.isOpened():
    print("Error: Unable to access the webcam.")
```

Fig 5.2.1 OpenCV implementation

User-Friendly Output:

OpenCV contributes to the user-friendly design of the system by enhancing the clarity of visual information. The module, in conjunction with the YOLO v4 model, ensures that the detected objects are accurately identified and that relevant features are extracted to provide meaningful feedback through the voice output module.

In summary, the OpenCV module serves as the backbone of our project's visual processing capabilities. Its integration empowers the system to provide real-time object detection, adapt to dynamic environments, and deliver user-friendly information to visually impaired individuals, aligning perfectly with the project's objective of enhancing accessibility and independence.

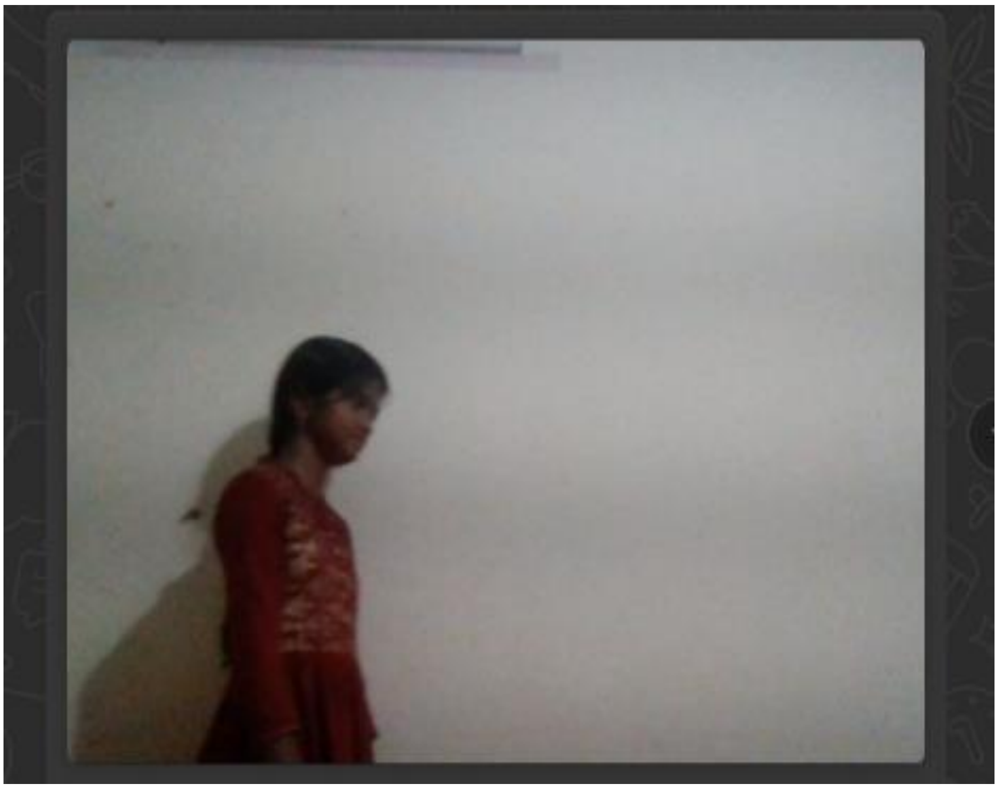


Fig 5.2.2 Connecting the camera using OpenCV

Object Detection Module:

Object Detection using Deep Learning - YOLO v4 Module:

In our project, the utilization of the You Only Look Once (YOLO) v4 model for object detection is a key technological component. YOLO is a state-of-the-art deep learning algorithm known for its efficiency, speed, and accuracy in detecting objects in real-time. Here, we delve into the specifics of how the YOLO v4 module is implemented and its role in providing seamless object detection.

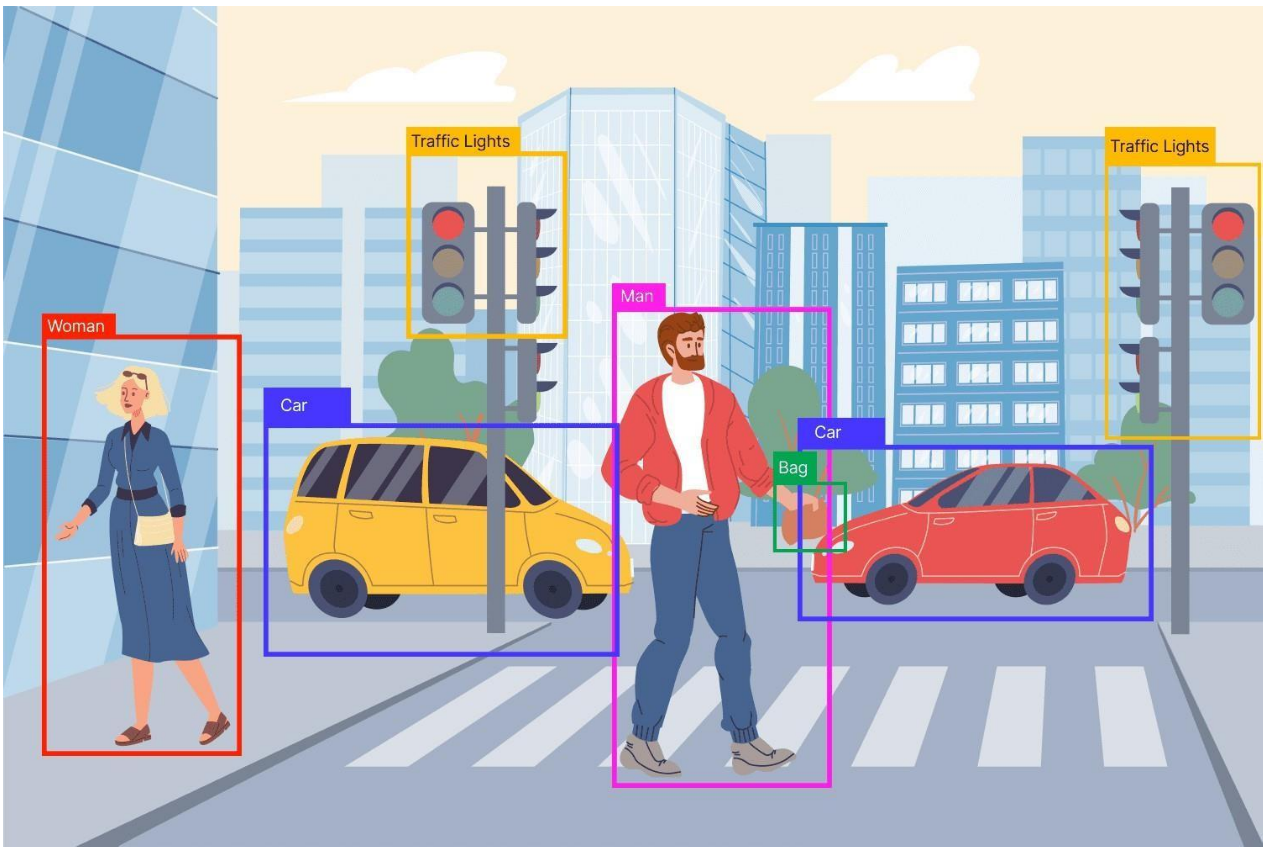


Fig 5.2.3 Object Detection

YOLO v4 Overview:

YOLO is a popular object detection algorithm that divides an image into a grid and assigns bounding boxes and class predictions to objects within each grid cell. YOLO v4, the fourth iteration of the YOLO series, enhances accuracy and extends capabilities. It introduces features such as multi-scale detection, enabling the model to detect objects at different sizes, and utilizes a darknet neural network framework.

Integration into the System:

The YOLO v4 module is integrated into our system to analyze visual data captured in real-time through cameras. The deep neural network architecture of YOLO v4 allows for the simultaneous detection of multiple objects in an image or video frame. Each detected object is associated with a bounding box and a probability score, indicating the confidence of the model in the accuracy of the detection.

Object Detection Process:

1. Input Processing: Visual data from the camera feed is input into the YOLO v4 module.

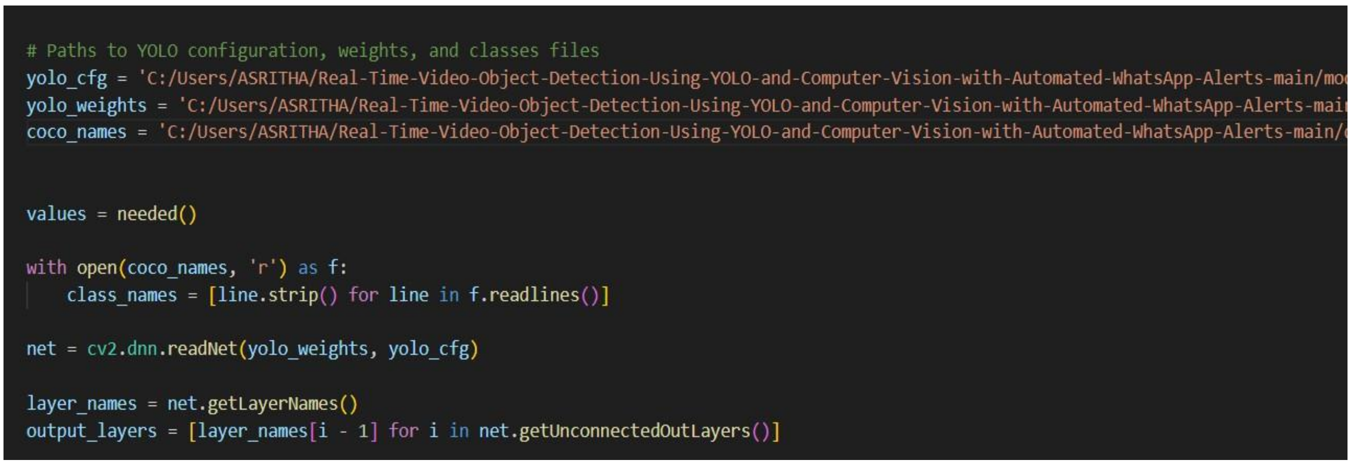
2. Grid Division: The image is divided into a grid, and each grid cell is responsible for predicting objects present within its boundaries.

3. Bounding Box Prediction: YOLO v4 predicts bounding boxes for objects, providing information about their location in the image.

4. Class Prediction: The model assigns a class label to each detected object, indicating the type of object it is (e.g., person, car, or chair).

5. Confidence Scoring: A confidence score is assigned to each prediction, reflecting the model's certainty about the accuracy of the detection.

6. Non-Maximum Suppression: Post-processing involves applying non-maximum suppression to filter out redundant bounding boxes and keep the most confident predictions.



```
# Paths to YOLO configuration, weights, and classes files
yolo_cfg = 'C:/Users/ASRITHA/Real-Time-Video-Object-Detection-Using-YOLO-and-Computer-Vision-with-Automated-WhatsApp-Alerts-main/mo
yolo_weights = 'C:/Users/ASRITHA/Real-Time-Video-Object-Detection-Using-YOLO-and-Computer-Vision-with-Automated-WhatsApp-Alerts-mai
coco_names = 'C:/Users/ASRITHA/Real-Time-Video-Object-Detection-Using-YOLO-and-Computer-Vision-with-Automated-WhatsApp-Alerts-main/


values = needed()

with open(coco_names, 'r') as f:
    class_names = [line.strip() for line in f.readlines()]

net = cv2.dnn.readNet(yolo_weights, yolo_cfg)

layer_names = net.getLayerNames()
output_layers = [layer_names[i - 1] for i in net.getUnconnectedOutLayers()]
```

Fig.5.2.4 Loading Yolo model

Advantages of YOLO v4 for Object Detection:

1. **Real-Time Performance** – YOLO v4 is optimized for speed, achieving high FPS (frames per second), making it ideal for real-time applications like surveillance, autonomous driving, and assistive technologies.

2. **High Accuracy and Precision** – With improvements in feature extraction and CSPDarknet53 as the backbone, YOLO v4 enhances object detection accuracy while reducing false positives and false negatives.

3. **Optimized for Edge Devices** – Unlike heavier models, YOLO v4 is designed to run efficiently on GPUs, TPUs, and even some high-performance CPUs, making it suitable for deployment on embedded systems and mobile devices.

4. **Better Generalization** – YOLO v4 implements Mosaic data augmentation, self- adversarial training, and CIoU loss, leading to better generalization across diverse datasets and environmental conditions.

5. **Improved Detection of Small Objects** – The advanced anchor-based detection mechanism enables better recognition of small and densely packed objects in an image.

6. **Robust Feature Extraction** – The integration of SPP (Spatial Pyramid Pooling) and PANet (Path Aggregation Network) improves object localization and detection performance, especially for small and occluded objects.

7. **Multi-Class Object Detection** – YOLO v4 can detect multiple objects in a single pass, making it highly efficient for applications requiring real-time analysis of multiple entities simultaneously.

8. **Lower Computational Cost** – Compared to other high-accuracy object detection models like Faster R-CNN, YOLO v4 balances speed and accuracy while maintaining lower computational complexity.

Challenges and Considerations:

1. **Hardware and Computational Requirements** – YOLO v4, while optimized for speed, still demands high-performance GPUs for efficient real-time processing. Running the model on low-end hardware may result in latency issues and degraded performance.

2. **Training and Dataset Challenges** – Customizing YOLO v4 for specific applications requires high-quality, annotated datasets. Ensuring proper labeling, handling class

imbalances, and training with diverse images to prevent bias is crucial.

3. **Trade-Off Between Speed and Accuracy** – Although YOLO v4 provides a balance, fine-tuning hyperparameters for optimal speed vs. accuracy can be complex, especially in environments where high precision is required.

4. **Handling Small and Overlapping Objects** – Despite improvements, detecting tiny or closely packed objects remains challenging, as YOLO's grid-based detection may cause errors in classification or localization.

5. **Environmental Variability** – Lighting conditions, occlusions, and background clutter can affect detection accuracy. Ensuring robustness in low-light or high-noise environments is essential for real-world deployment.

6. **Real-Time Voice Feedback Synchronization** – Integrating speech output for detected objects must ensure low-latency response without delays in audio output, especially in applications for visually impaired users.

7. **Optimization for Edge Devices** – While YOLO v4 is efficient, deploying it on low- power devices (e.g., Raspberry Pi, Jetson Nano) requires optimizations like model quantization and pruned architectures to reduce memory footprint.

8. **Security and Privacy Concerns** – If integrated with cloud-based services, data privacy must be addressed, ensuring that sensitive information is not exposed or misused during processing.

9. **Integration with External Modules** – Combining YOLO v4 with OpenCV, Twilio for alerts, and text-to-speech engines requires seamless API integrations, error handling, and smooth data flow management.

10. **Continuous Model Updates and Maintenance** – Object detection models require frequent updates to stay relevant with new datasets and evolving detection scenarios. Managing version compatibility and retraining efforts is an ongoing challenge.

5.3 Algorithm

The success of our project relies heavily on the implementation of sophisticated algorithms that underpin the core functionalities. Here, we provide insights into the primary algorithms employed in our system, focusing on the YOLO v4 algorithm for object detection.

1. YOLO v4 Algorithm:

Overview:

The You Only Look Once (YOLO) v4 algorithm serves as the backbone of our object detection system. YOLO v4 revolutionizes the field of object detection by dividing an image into a grid and simultaneously predicting bounding boxes and class probabilities for objects within each grid cell. This approach enables real-time and accurate detection of multiple objects within a single pass through the neural network.

Multi-Scale Detection:

One of the key advancements in YOLO v4 is its ability to detect objects at different scales. The model incorporates feature maps from multiple layers of the neural network, allowing it to effectively recognize objects of various sizes in an image. This multi-scale detection capability enhances the versatility of the algorithm, making it well-suited for our project's objective of identifying objects in diverse visual environments.

Darknet Neural Network Framework:

YOLO v4 is implemented using the darknet neural network framework, which is open-source and facilitates the training and deployment of deep neural networks. Darknet provides the necessary infrastructure for building, configuring, and fine-tuning the YOLO v4 model, ensuring that it aligns with the specific requirements of our project.

Sure, let's break down the YOLO (You Only Look Once) algorithm and understand how it works.

YOLO Architecture:

Input Image:

YOLO takes an input image and divides it into an S × S grid.

Bounding Box Prediction:

Each grid cell predicts B bounding boxes and their corresponding confidence scores.

A bounding box is represented by (x, y, w, h), where (x, y) is the center of the box, and (w, h) are the width and height.

Class Prediction:

Each grid cell also predicts C conditional class probabilities.

Output:

The final output is a tensor of shape (S, S, B * 5 + C), containing all the bounding box and class predictions.

YOLO Loss Function:

YOLO uses a combination of localization loss (how well it predicts the bounding box) and classification loss (how well it predicts the class) to train the network.

The total loss is the sum of the localization loss, confidence loss (for predicting if an object is present in a box), and class loss.

Yolo Working:

YOLO trains and tests on full images and directly optimizes detection performance. YOLO model has several benefits over other traditional methods of object detection like the following.

- First, YOLO is extremely fast. Since frame detection in YOLO is a regression problem there is no need for a complex pipeline. We can simply run our neural network on any new image at test time to make predictions.
- Second, YOLO sees the entire image during training and testing unlike other sliding window algorithms which require multiple iterations to process a single image.
- Third, YOLO learns generalizable object representations. When trained on real time images and tested, YOLO outperforms top detection methods like DPM and RCNN.

YOLO network uses features from the entire image to predict each bounding box. It also predicts all bounding boxes across all classes for an image simultaneously.
This means our network reasons globally about the full image and all the objects in the image. The YOLO design enables end-to-end training and real time speeds while maintaining high average precision.

- First YOLO divides the input image into an S × S grid as shown in fig.

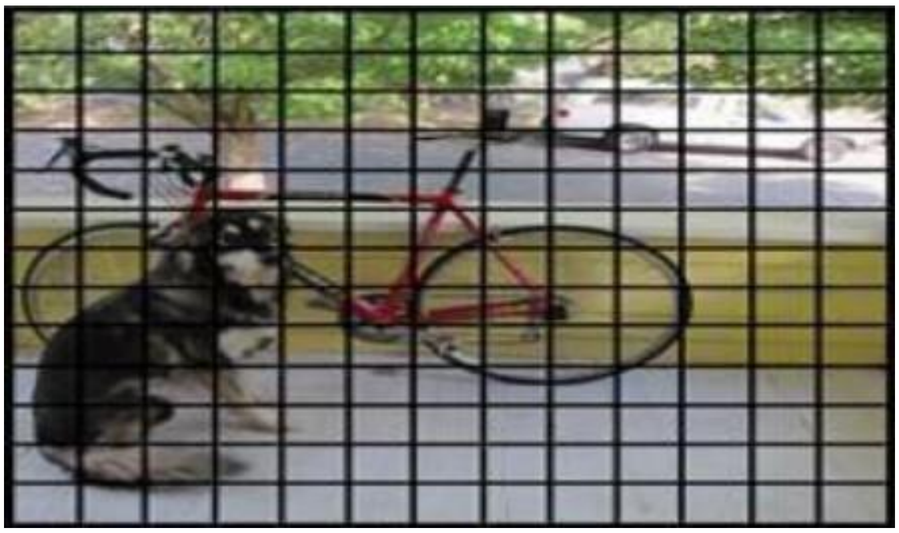


Fig 5.3.1 Divide the image into S × S grid

- If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object.

- Each grid cell predicts B bounding boxes and confidence scores for those boxes as shown in fig.
- This confidence scores reflect how confident the model is that the box contains an object. If no object exists in that cell, the confidence scores should be zero.

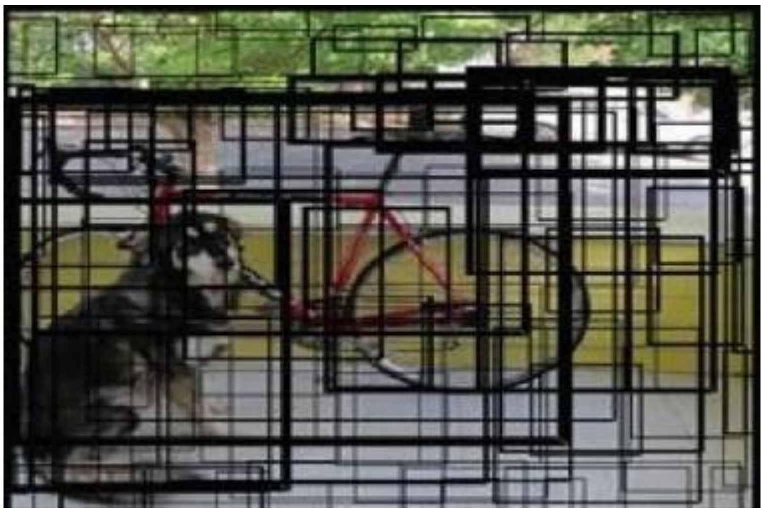


Fig 5.3.2 Calculate Bounding Boxes and confidence score for each box.

- Each grid cell also predicts conditional class probabilities.

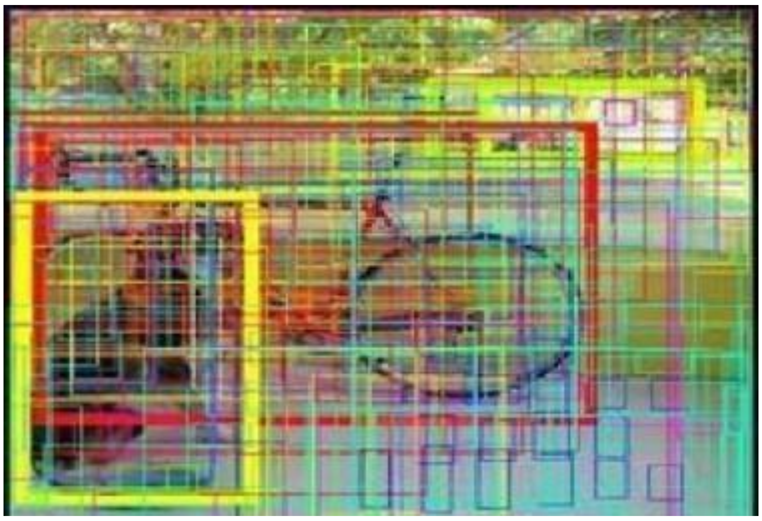


Fig 5.3.3 Multiply Probability and Confidence Scores

- Finally, we multiply the conditional class probabilities as shown in fig. 9 and the individual box confidence predictions which gives us class-specific confidence scores for each box as shown in fig.

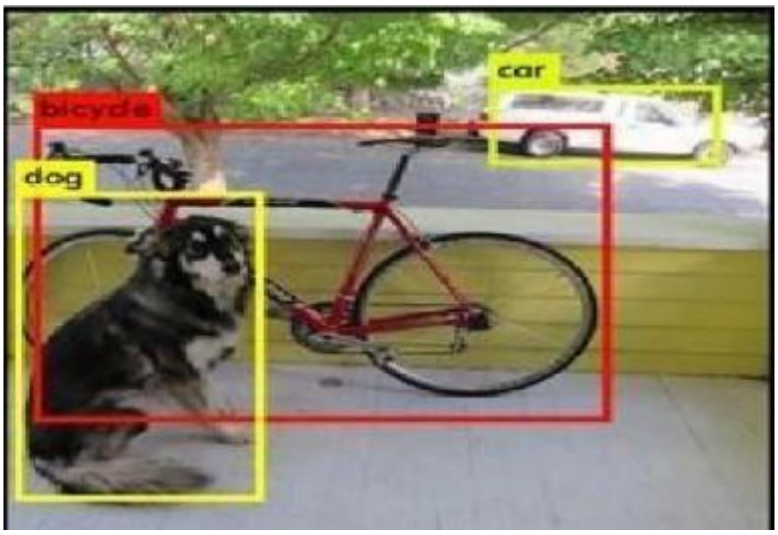


Fig 5.3.4 Final Output

YOLO Working in our project:

1. **Live Camera Feed Input:**
   - The camera continuously streams video frames to the system for analysis.
   - Each frame is captured in real-time and passed to the YOLO model.

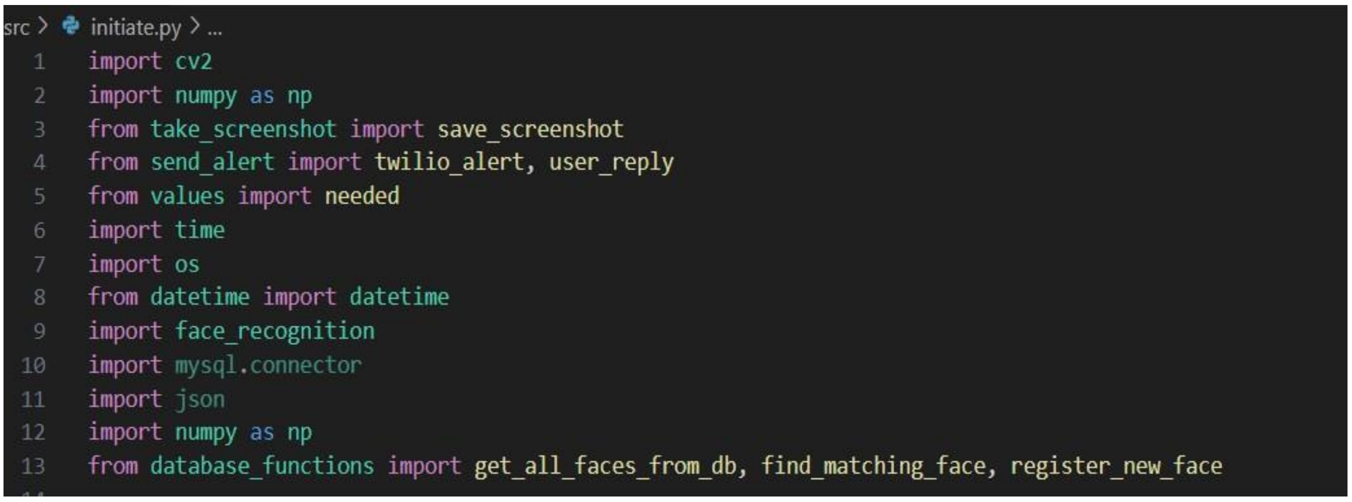
2. **YOLO-Based Object Detection:**
   - YOLO processes each frame once and detects objects or humans in a single forward pass.
   - It divides the image into a grid and predicts:
     - Bounding boxes
     - Class probabilities (e.g., "person", "bag", etc.)
   - Fast and efficient for real-time detection.

3. **Filtering Detected Objects:**
   - Only relevant objects (like humans) are filtered using confidence thresholds.
   - If a human is detected, the system considers this as potential activity.

4. **Triggering Further Actions:**
   - If human detection is successful:
     - The frame is passed to the face recognition module (to check if it's a known person).
     - If unknown, the system captures a screenshot/video.
     - Alerts are sent via Twilio WhatsApp API.

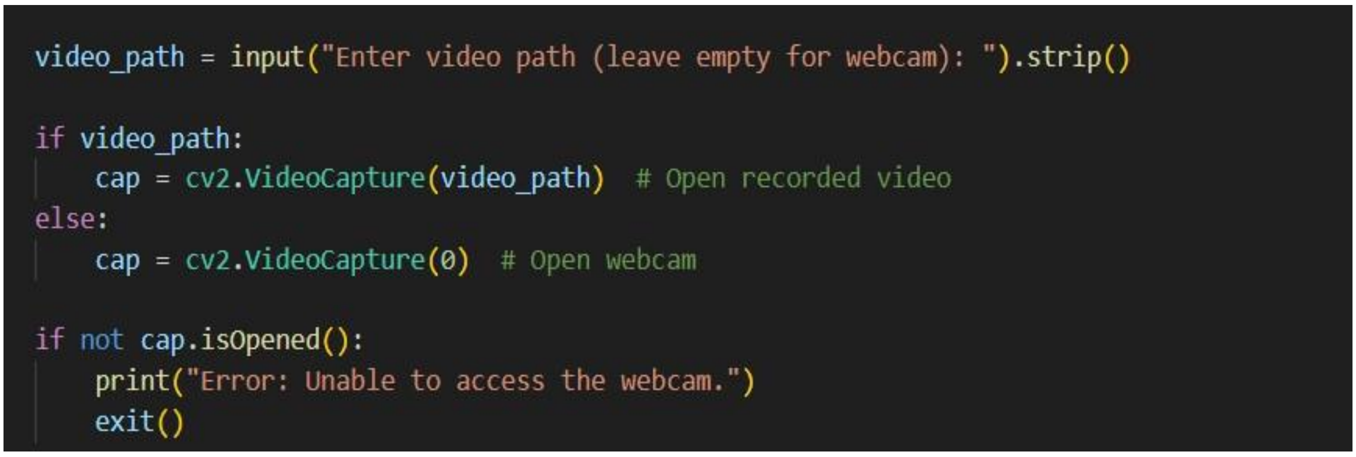
5. **Emergency Handling:**
   - If there's no response from the user or the threat is critical, an emergency alert is sent to pre-configured contacts.

5.4 Screens



```
src > initiate.py > ...
import cv2
import numpy as np
from take_screenshot import save_screenshot
from send_alert import twilio_alert, user_reply
from values import needed
import time
import os
from datetime import datetime
import face_recognition
import mysql.connector
import json
import numpy as np
from database_functions import get_all_faces_from_db, find_matching_face, register_new_face
```

Fig 5.4.1 Importing Required libraries

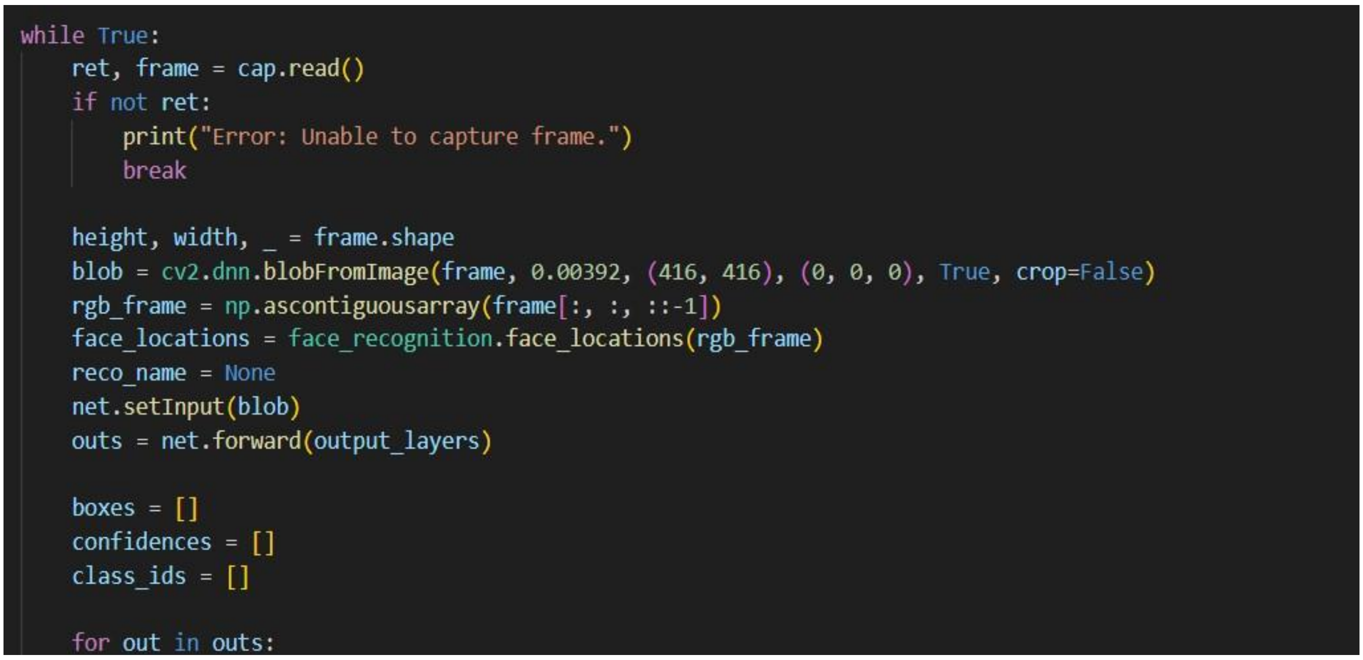


```
video_path = input("Enter video path (leave empty for webcam): ").strip()

if video_path:
    cap = cv2.VideoCapture(video_path)  # Open recorded video
else:
    cap = cv2.VideoCapture(0)  # Open webcam

if not cap.isOpened():
    print("Error: Unable to access the webcam.")
    exit()
```

Fig 5.4.2 Initializing OpenCV object



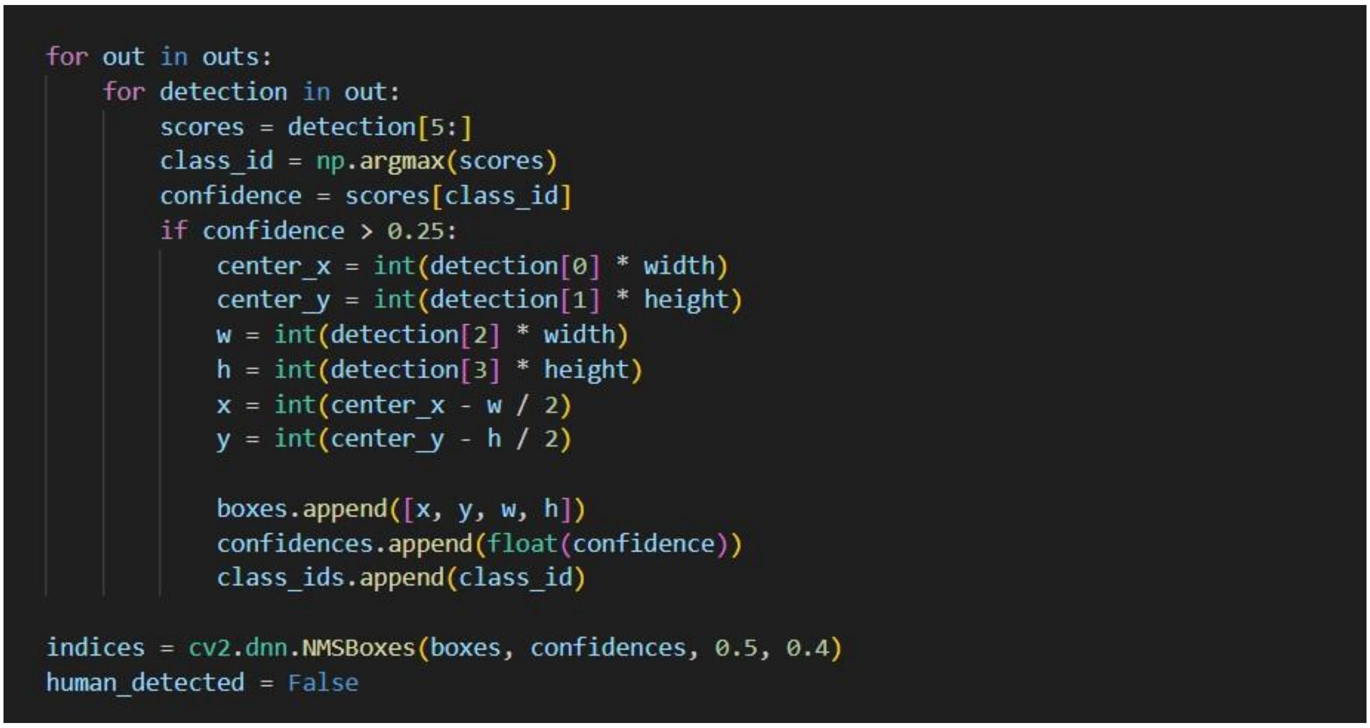
```
while True:
    ret, frame = cap.read()
    if not ret:
        print("Error: Unable to capture frame.")
        break

    height, width, _ = frame.shape
    blob = cv2.dnn.blobFromImage(frame, 0.00392, (416, 416), (0, 0, 0), True, crop=False)
    rgb_frame = np.ascontiguousarray(frame[:, :, ::-1])
    face_locations = face_recognition.face_locations(rgb_frame)
    reco_name = None
    net.setInput(blob)
    outs = net.forward(output_layers)

    boxes = []
    confidences = []
    class_ids = []

    for out in outs:
```

Fig 5.4.3 Capturing frames



```
for out in outs:
    for detection in out:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > 0.25:
            center_x = int(detection[0] * width)
            center_y = int(detection[1] * height)
            w = int(detection[2] * width)
            h = int(detection[3] * height)
            x = int(center_x - w / 2)
            y = int(center_y - h / 2)

            boxes.append([x, y, w, h])
            confidences.append(float(confidence))
            class_ids.append(class_id)

indices = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
human_detected = False
```

Fig 5.4.4 Object detection

6. SYSTEM TESTING

6.1 Introduction

Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, subassemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner.

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner.

There are various types of tests. Each test type addresses a specific testing requirement. Software Testing is evaluation of the software against requirements gathered from users and system specifications. Testing is conducted at the phase level in software development life cycle or at module level in program code. Software testing comprises Validation and Verification.

Testing is a group of techniques to determine the correctness of the application under the predefined script but, testing cannot find all the defects of the application. The main intent of testing is to detect failures of the application so that failures can be discovered and corrected. It does not demonstrate that a product functions properly under all conditions but only that it is not working in some specific conditions.

Testing furnishes comparison that compares the behavior and state of software against mechanisms because the problem can be recognized by the mechanism. The mechanism may include past versions of the same specified product, comparable products, and interfaces of expected purpose, relevant standards, or other criteria but not limited up to these. Testing includes an examination of code and also the execution of code in various environments, conditions as well as all the examining aspects of the code.

In the current scenario of software development, a testing team may be separate from the development team so that Information derived from testing can be used to correct the process of software development. System Testing is basically performed by a testing team that is independent of the development team that helps to test the quality of the system impartially.

Usually, software testing includes:

1. Unit tests: The program is broken down into blocks, and each element (unit) is tested separately.

2. Integration tests: This type of testing observes how multiple components of the program work together. Then, to evaluate the performance of the model in machine learning, we use two sets of data:

- Validation set: Having only a training set and a testing set is not enough. And that can result in overfitting. To avoid that, you can select a small validation data set to evaluate a model. Only after we get maximum accuracy on the validation set, we make the testing set come into the game.
- Test set: Our model might fit the training dataset perfectly well. But where are the guarantees that it will do equally well in real-life? In order to assure that, you select samples for a testing set from our training set, that the machine hasn't seen before.

It is important to remain unbiased during selection and draw samples at random. Also, we should not use the same set many times to avoid training on our test data. Our test set should be large enough to provide statistically meaningful results and be representative of the data set as a whole.

Cross-validation:
Cross-validation is a model evaluation technique that can be performed even on a limited dataset. The training set is divided into small subsets, and the model is trained and validated on each of these samples. Validation: In this method, we perform training on the 50% of the given data-set and the rest 50% is used for the testing purpose. The major drawback of this method is that we perform training on the 50% of the dataset, it may be possible that the remaining 50% of the data contains some important information which we are leaving while training our model i.e higher bias.

Leave One Out Cross Validation:
In this method, we perform training on the whole data-set but leaves only one data-point of the available data-set and then iterates for each data-point. It has some advantages as well as disadvantages also. An advantage of using this method is that we make use of all data points and hence it is low bias. The major drawback of this method is that it leads to higher variation in the testing model as we are testing against one data point. If the data point is an outlier it can lead to higher variation. Another drawback is it takes a lot of execution time as it iterates over 'the number of data points times.

K-fold cross-validation:

The most common cross-validation method is called k-fold cross-validation. To use it, you need to divide the dataset into k subsets (also called folds) and use them k times. For example, by breaking the dataset into 10 subsets, you will perform a 10-fold cross validation. Each subset must be used as the validation set at least once. In Machine Learning model development, the performance of the system is measured in terms of accuracy, precision and recall of the model. It's common to also use validation accuracy and test accuracy as evaluation metrics, which provide a better indication of the model's performance on unseen data.

Accuracy:

Accuracy is a metric for how much of the predictions the model makes are true. The higher the accuracy is, the better. However, it is not the only important metric when you estimate the performance.

$$\text{Accuracy} = TP+TN/TP+FP+FN+TN$$

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not. In simple words, testing is executing a system in order to identify any gaps, errors, or missing requirements in contrast to the actual requirements.

According to ANSI/IEEE 1059 standard, Testing can be defined as - A process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item.

It depends on the process and the associated stakeholders of the project(s). In the IT industry, large companies have a team with responsibilities to evaluate the developed software in context of the given requirements. Moreover, developers also conduct testing which is called Unit Testing. In most cases, the following professionals are involved in testing a system within their respective capacities –

- Software Tester
- Software Developer
- Project Lead/Manager
- End User

Different companies have different designations for people who test the software on the basis of their experience and knowledge such as Software Tester, Software Quality Assurance Engineer, QA Analyst, etc.

It is not possible to test the software at any time during its cycle. The next two sections state when testing should be started and when to end it during the SDLC.

An early start to testing reduces the cost and time to rework and produce error-free software that is delivered to the client. However, in Software Development Life Cycle (SDLC), testing can be started from the Requirements Gathering phase and continued till the deployment of the software.

It also depends on the development model that is being used. For example, in the Waterfall model, formal testing is conducted in the testing phase; but in the incremental model, testing is performed at the end of every increment/iteration and the whole application is tested at the end.

In a software development project, errors can be introduced at any stage during development. Though errors are detected after each phase by techniques like inspections, some errors remain undetected.

Ultimately, these remaining errors are reflected in the code. There are two types of approaches for identifying defects in the software: static and dynamic. In static analysis, the code is not executed but is evaluated through some process or some tools for locating defects. Code inspections, which we discussed in the previous chapter, are one static approach. Another is static analysis of code through the use of tools.

In dynamic analysis, code is executed, and the execution is used for determining defects. Testing is the most common dynamic technique that is employed. Indeed, testing is the most commonly used technique for detecting defects, and performs a very critical role for ensuring quality.

During testing, the software under test (SUT) is executed with a finite set of test cases, and the behavior of the system for these test cases is evaluated to determine if the system is performing as expected.

The basic purpose of testing is to increase the confidence in the functioning of SUT. Clearly, the effectiveness and efficiency of testing depends critically on the test cases selected. Here we focus on: Basic concepts and definitions relating to testing, like error, fault, failure, test case, test suite, test harness, etc.

- The testing process—how testing is planned and how testing of a unit is done.
- Test case selection using black-box testing approaches.
- Test case selection using white-box approaches.
- Some metrics like coverage and reliability that can be employed during testing.

The basic goal of the software development process is to produce software that has no errors or very few errors. We have seen that different levels of testing are needed to detect the defects injected during the various tasks in the project. The testing process for a project consists of three high-level tasks: test planning, test case design, and test execution. We will discuss these in the rest of this section.

1. Test Plan:

In a project, testing commences with a test plan and terminates with successful execution of acceptance testing. A test plan is a general document for the entire project that defines the scope, approach to be taken, and the schedule of testing, as well as identifies the test items for testing and the personnel responsible for the different activities of testing. The test planning can be done well before the actual testing commences and can be done in parallel with the coding and design activities. The inputs for forming the test plan are:

(1) Project plan

(2) Requirements document

(3) Architecture or design document.

2. Test Case Design:

The test plan focuses on how the testing for the project will proceed, which units will be tested, and what approaches (and tools) are to be used during the various stages of testing. However, it does not deal with the details of testing a unit, nor does it specify which test cases are to be used.

3. Test Case Execution:

The test case specifications only specify the set of test cases for the unit to be tested. However, executing the test cases may require construction of driver modules or stubs. It may also require modules to set up the environment as stated in the test plan and test case specifications.

If test frameworks are being used, then the setting of the environment as well as inputs for a test case is already done in the test scripts, and execution is straightforward. During test case execution, defects are found.

6.2 Testing methods

- System Testing includes testing of a fully integrated software system.
- Generally, a computer system is made with the integration of software (any software is only a single element of a computer system).
- The software is developed in units and then interfaced with other software and hardware to create a complete computer system. In other words, a computer system consists of a group of software to perform the various tasks, but only software cannot perform the task; for that software must be interfaced with compatible hardware.
- System testing is a series of different types of tests with the purpose to exercise and examine the full working of an integrated software computer system against requirements.

To check the end-to-end flow of an application or the software as a user is known as System testing. In this, we navigate (go through) all the necessary modules of an application and check if the end features or the end business works fine, and test the product as a whole system. It is end-to-end testing where the testing environment is similar to the production environment.

There are mainly two widely used methods for software testing, one is White box testing which uses internal coding to design test cases and another is black box testing which uses GUI or user perspective to develop test cases.

The following are the Testing Methodologies:

- Unit Testing.
- Integration Testing.
- Validation Testing.
- White box testing.
- Black box testing.

Unit Testing:

Unit testing focuses verification effort on the smallest unit of Software design that is the module. Unit testing exercises specific paths in a module’s control structure to ensure complete coverage and maximum error detection. This test focuses on each module individually, ensuring that it functions properly as a unit. Hence, the naming is Unit Testing. During this testing, each module is tested individually and the module interfaces are verified for the consistency with design specification. All important processing path are tested for the expected results. Unit Testing is a type of software testing where individual units or components of a software are tested. The purpose is to validate that each unit of the software code performs as expected. Unit Testing is done during the development (coding phase) of an application by the developers. Unit Tests isolate a section of code and verify its correctness. A unit may be an individual function, method, procedure, module, or object. In SDLC, STLC, V Model, Unit testing is the first level of testing done before integration testing. Unit testing is a White Box testing technique that is usually performed by the developer. Though, in a practical world due to time crunch or reluctance of developers to test, QA engineers also do unit testing.

• Unit tests help to fix bugs early in the development cycle and save costs.
• It helps the developers to understand the testing code base and enables them to make changes quickly.
• Good unit tests serve as project documentation
• Unit tests help with code reuse. Migrate both your code and your tests to your new project. Tweak the code until the tests run again.

In order to execute Unit Tests, developers write a section of code to test a specific function in a software application. Developers can also isolate this function to test more rigorously which reveals unnecessary dependencies between the function being tested and other units so the dependencies can be eliminated. Developers generally use Unit Test framework to develop automated test cases for unit testing.

Unit Testing is of two types:
• Manual
• Automatic

Unit testing is commonly automated but may still be performed manually. Software Engineering does not favor one over the other but automation is preferred.

Unit testing in your project can help ensure that individual components or functions work as expected. For your object detection, you can consider the following unit testing aspects:

1. Object Detection Unit Test:

- Test the `detect_objects` function: If you have a separate function for object detection, you can test it by providing different input images and checking if the expected objects are detected.
- Test the accuracy of object detection: For a given input image, check if the detected objects match the ground truth annotations.

2. Frame Processing Unit Test:

- Test the frame processing logic: If you have specific logic for processing each frame, ensure that it works correctly. You can use mock input frames and check the output.

3. Performance Testing:

- Evaluate the performance of your system by feeding it with various images and assessing the processing time. Ensure that the frame rate is within acceptable limits and doesn't lead to significant delays.

4. Error Handling:

- Introduce errors deliberately (e.g., provide an invalid image path) and check if your system handles these errors gracefully without crashing. Ensure that appropriate error messages are displayed or logged.

5. Integration Testing:

- Test the integration of different components. For example, check if the object detection results are correctly passed to the face recognition module.

Integration Testing:

Integration testing addresses the issues associated with the dual problems of verification and program construction. After the software has been integrated a set of high order tests are conducted. The main objective in this testing process is to take modules and build a program structure that has been dictated by design. Integration Testing is defined as a type of testing where software modules are integrated logically and tested as a group. A typical software project consists of multiple software modules, coded by different programmers. The purpose of this level of testing is to expose defects in the interaction between these software modules when they are integrated. Integration Testing focuses on checking data communication amongst these modules. Hence it is also termed as 'I&T' (Integration and Testing), 'String Testing' and sometimes 'Thread Testing'.

- A Module, in general, is designed by an individual software developer whose understanding and programming logic may differ from other programmers. Integration Testing becomes necessary to verify the software modules work in unity.
- At the time of module development, there are wide chances of change in requirements by the clients. These new requirements may not be unit tested and hence system integration Testing becomes necessary.
- Interfaces of the software modules with the database could be erroneous.
- External Hardware interfaces, if any, could be erroneous.
- Inadequate exception handling could cause issues.

Integration test case differs from other test cases in the sense it focuses mainly on the interfaces & flow of data/information between the modules. Here priority is to be given for the integrating links rather than the unit functions which are already tested.

Integration test approaches –

There are four types of integration testing approaches. Those approaches are the following:

- Big-Bang Integration Testing
- Bottom-Up Integration Testing
- Top-Down Integration Testing
- Mixed Integration Testing

Big-Bang Integration Testing:

It is the simplest integration testing approach, where all the modules are combining and verifying the functionality after the completion of individual module testing. In simple words, all the modules of the system are simply put together and tested. This approach is practicable only for very small systems. If once an error is found during the integration testing, it is very difficult to localize the error as the error may potentially belong to any of the modules being integrated. So, debugging errors reported during big bang integration testing is very expensive to fix. The Big-bang integration workflow diagram is a process diagram that shows how different parts of a system are integrated. It is typically used to show how different software components are integrated. It is a graphical representation of the software development process that is used in many organizations. It is a process that starts with the idea for a new software project and ends with the delivery of the software to the customer.

Bottom-Up Integration Testing:

Bottom-up Testing is a type of incremental integration testing approach in which testing is done by integrating or joining two or more modules by moving upward from bottom to top through control flow of architecture structure. In these, low-level modules are tested first, and then high-level modules are tested. This type of testing or approach is also known as inductive reasoning and is used as a synthesis synonym in many cases.Bottom-up testing is user-friendly testing and results in an increase in overall software development. This testing results in high success rates with long-lasting results.

Processing:

Following are the steps that are needed to be followed during the processing:

1. Clusters are formed by merging or combining low-level modules or elements. These clusters are also known as builds that are responsible for performing the certain secondary or subsidiary function of a software.
2. It is important to write a control program for testing. These control programs are also known as drivers or high-level modules. It simply coordinates input and output of a test case.
3. Testing is done of the entire build or cluster containing low-level modules.
4. Lastly, control programs or drivers or high level modules are removed and clusters are integrated by moving upward from bottom to top in program structure with help of control flow.

Top-Down Integration Testing:

Top-down testing is a type of incremental integration testing approach in which testing is done by integrating or joining two or more modules by moving down from top to bottom through control flow of architecture structure. In these, high-level modules are tested first, and then low-level modules are tested. Then, finally, integration is done to ensure that the system is working properly. Stubs and drivers are used to carry out this project. This technique is used to increase or stimulate behavior of Modules that are not integrated into a lower level.

Following are the steps that are needed to be followed during processing:

1. Test driver represents the main control module also known as a high-level module and stubs are generally used for all low-level modules that directly subordinate (present or rank below another) to high-level modules.
2. In this, testing takes place from the bottom. So, high-level modules are tested first in isolation.
3. After this, low-level modules or subordinated modules or stubs replace high-level modules one by one at a time. This can be done using methods like depth-first or breadth search.
4. The process is repeated until each module is integrated and tested.
5. Another stub replaces the present real or control module after completion of each set of tests. These stubs act as a temporary replacement for a called module (stubs) and give the same result or output as the actual product gives.
6. To check if there is any defect or any error occurring or present, regression testing is done and it's important to reduce any side effect that might be caused due to errors occurring.

Mixed Integration Testing:

A mixed integration testing is also called sandwiched integration testing. A mixed integration testing follows a combination of top down and bottom-up testing approaches. In a top-down approach, testing can start only after the top-level module has been coded and unit tested. In the bottom-up approach, testing can start only after the bottom level modules are ready. This sandwich or mixed approach overcomes this shortcoming of the top-down and bottom-up approaches. It is also called the hybrid integration testing. Also, stubs and drivers are used in mixed integration testing.Integration testing in the context of this project involves testing the collaboration and interaction between different modules or components. In your case, integration testing can focus on ensuring the seamless coordination between the object detection module, frame processing, and the text-to-speech module. Here are some aspects to consider for integration testing:

1. End-to-End Processing:

- Verify that when the system is running, the entire end-to-end process works as expected. This includes capturing frames from the camera, performing object detection, processing the results, and generating voice output.

2. Communication Between Modules:

- Test the communication pathways between the object detection module and the text-to-speech module. Ensure that relevant information about detected objects is correctly passed from the object detection module to the voice output module.

3. Data Flow:

- Examine the flow of data between different components. Ensure that the data passed between modules is in the expected format and that there are no data-related issues causing errors.

4. Error Handling Across Modules:

- Test how the system handles errors or unexpected situations that may arise during integration. For example, check how the system responds if there is a failure in the object detection process or if there is an issue with the text-to-speech module.

5. Synchronization and Timing:

- Verify that the timing and synchronization between modules are appropriate. Ensure that one module doesn't get ahead or fall behind in processing frames, leading to inconsistencies in the results.

6. Concurrency and Parallelism:

- If your system involves parallel processing or concurrency, ensure that modules are designed to work correctly in such environments. Test the behavior when multiple components are processing data simultaneously.

7. Resource Management:

- Check if the system manages resources efficiently, especially if there are shared resources between different modules. For example, ensure that memory is appropriately allocated and released.

8. Integration with External Libraries:

- If your project utilizes external libraries (such as OpenCV, pyttsx3, etc.), test the integration with these libraries to ensure compatibility and correct usage.

Validating Testing:

The process of evaluating software during the development process or at the end of the development process to determine whether it satisfies specified business requirements. Validation Testing ensures that the product actually meets the client's needs. It can also be defined as to demonstrate that the product fulfills its intended use when deployed in an appropriate environment. Validation checks are performed on the following fields.

Text Field:

The text field can contain only the number of characters lesser than or equal to its size. The text fields are alphanumeric in some tables and alphabetic in other tables. Incorrect entries always flash and error messages.

Numeric Field:

The numeric field can contain only numbers from 0 to 9. An entry of any character flashes an error message. The individual modules are checked for accuracy and what it has to perform. Each module is subjected to a test run along with sample data. The individually tested modules are integrated into a single system. The testing should be planned so that all the requirements are individually tested. A successful test is one that gives out the defects for the inappropriate data and produces an output revealing the errors in the system.

Preparation of Test Data:

Taking various kinds of test data does the above testing. Preparation of test data plays a vital role in the system testing. After preparing the test data the system under study is tested using that test data. While testing the system by using test data errors are again uncovered and corrected by using above testing steps and corrections are also noted for future use.

Using Live Test Data:

Live test data are those that are actually extracted from organization files. After a system is partially constructed, programmers or analysts often ask users to key in a success of data from their normal activities. In other instances, programmers or analysts extract a set of live data from the files and have them entered themselves. It is difficult to obtain live data in sufficient amounts to conduct extensive testing. And, although it is realistic data that will show how the system will perform for the typical processing requirement, assuming that the live data entered are in fact typical, such data generally will not test all combinations of formats that can enter the system. This bias toward typical values does not provide a true systems test and in fact ignores the cases most likely to cause system failure.

Using Artificial Test Data:

Artificial test data are created solely for test purposes, since they can be generated to test all combinations of formats and values. In other words, the artificial data, which can quickly be prepared by a data generating utility program in the information systems department, make possible the testing of all login and control paths through the program. The most effective test programs use artificial test data generated by persons other than those who wrote the programs. Often, an independent team of testers formulates a testing plan, using the system's specifications. The package "Virtual Private Network" has satisfied all the requirements specified as per software requirement specification and was accepted.

White Box Testing:

"White box testing" (also known as clear, glass box or structural testing) is a testing technique which evaluates the code and the internal structure of a program. White box testing involves looking at the structure of the code. When you know the internal structure of a product, tests can be conducted to ensure that the internal operations performed according to the specification. And all internal components have been adequately exercised. We perform white box testing for following reasons:

To ensure:

- That all independent paths within a module have been exercised at least once.
- All logical decisions verified on their true and false values.
- All loops executed at their boundaries and within their operational bounds internal data structures validity.

To discover the following types of bugs:

- Logical error tends to creep into our work when we design and implement functions, conditions or controls that are out of the program.
- The design errors due to the difference between logical flow of the program and the actual implementation.
- Typographical errors and syntax checking.

Steps to Perform White Box Testing:

Step 1 – Understand the functionality of an application through its source code which means that a tester must be well versed with the programming language and the other tools as well techniques used to develop the software.

Step 2– Create the tests and execute them.

The below are some White-box testing techniques:

- **Control-flow testing** - The purpose of the control-flow testing to set up test cases which covers all statements and branch conditions. The branch conditions are tested for both being true and false, so that all statements can be covered.

- **Data-flow testing** - This testing technique emphasizes to cover all the data variables included in the program. It tests where the variables were declared and defined and where they were used or changed.

Black Box Testing:

Black Box Testing is also known as behavioral, opaque-box, closed-box, specification based or eye-to-eye testing.

It is a Software Testing method that analyzes the functionality of a software/application without knowing much about the internal structure/design of the item that is being tested and compares the input value with the output value. The main focus in Black Box Testing is on the functionality of the system as a whole.

The term 'Behavioral Testing' is also used for Black Box Testing.

Behavioral test design is slightly different from the black-box test design because the use of internal knowledge isn't strictly forbidden, but it's still discouraged.

Each testing method has its own advantages and disadvantages. There are some bugs that cannot be found using the only black box or only white box technique.

Majority of the applications are tested by Black Box method. We need to cover the majority of test cases so that most of the bugs will get discovered by a Black-Box method.

In this testing method, the design and structure of the code are not known to the tester, and testing engineers and end users conduct this test on the software.

Black-box testing techniques:

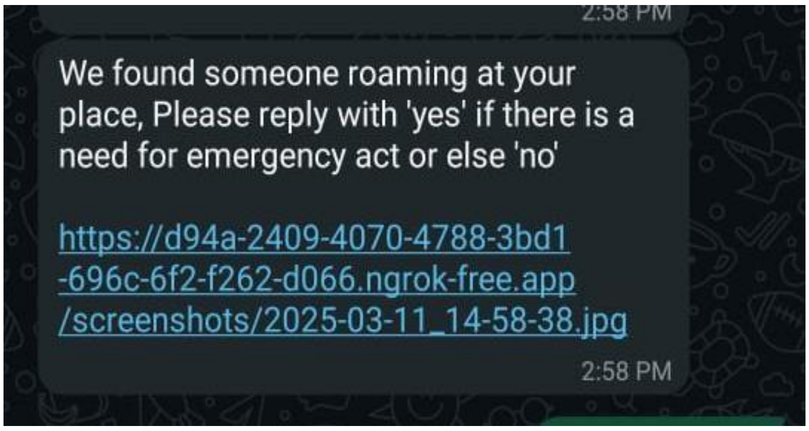
- **Equivalence class** - The input is divided into similar classes. If one element of a class passes the test, it is assumed that all the class is passed.
- **Boundary values** - The input is divided into higher and lower end values. If these values pass the test, it is assumed that all values in between may pass too.
- **Cause-effect graphing** - In both previous methods, only one input value at a time is tested. Cause (input) – Effect (output) is a testing technique where combinations of input values are tested in a systematic way.
- **Pair-wise Testing** - The behavior of software depends on multiple parameters. In pairwise testing, the multiple parameters are tested pair-wise for their different values.
- **State-based testing** - The system changes state on provision of input. These systems are tested based on their states and input.

Generic steps of black box testing:

- The black box test is based on the specification of requirements, so it is examined in the beginning.
- In the second step, the tester creates a positive test scenario and an adverse test scenario by selecting valid and invalid input values to check that the software is processing them correctly or incorrectly.
- In the third step, the tester develops various test cases such as decision table, all pairs test, equivalent division, error estimation, cause-effect graph, etc.
- The fourth phase includes the execution of all test cases.
- In the fifth step, the tester compares the expected output against the actual output.
- In the sixth and final step, if there is any flaw in the software then it is cured and tested again.

6.3 Test Cases

Test Description	**Expected Result**	**Actual Result**
Hardware Compatibility	Hardware components (camera, microphone) integrate properly	All hardware components detected and functional
Software Installation and Configuration	Successful installation of Python, OpenCV, and required libs	Software installed without errors
Camera Connectivity	Camera successfully connects to the system	Camera detected and initialized
Image Capture and Processing	System captures and processes live video frames	Live video feed displayed and processed
Object Detection Accuracy	Detection of predefined objects in test environment	Detected objects with >80% accuracy
Real-Time Processing	Real-time processing of video feed for continuous output	Processing real time video



2:58 PM

We found someone roaming at your place, Please reply with 'yes' if there is a need for emergency act or else 'no'

https://d94a-2409-4070-4788-3bd1-696c-6f2-f262-d066.ngrok-free.app/screenshots/2025-03-11_14-58-38.jpg

2:58 PM

Fig 6.3.1 Sending alert messages if something is detected

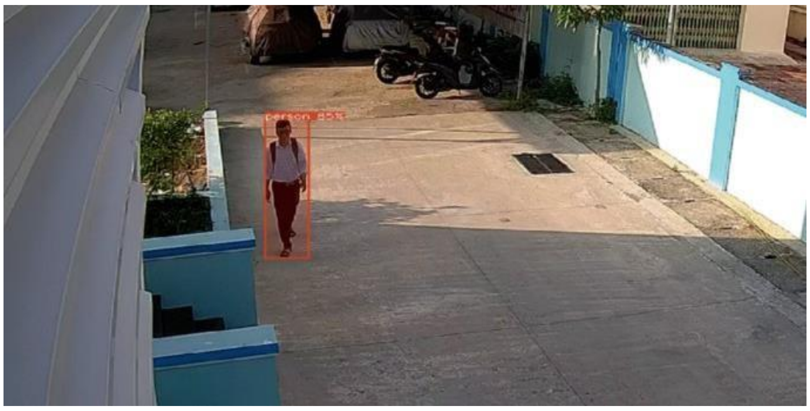
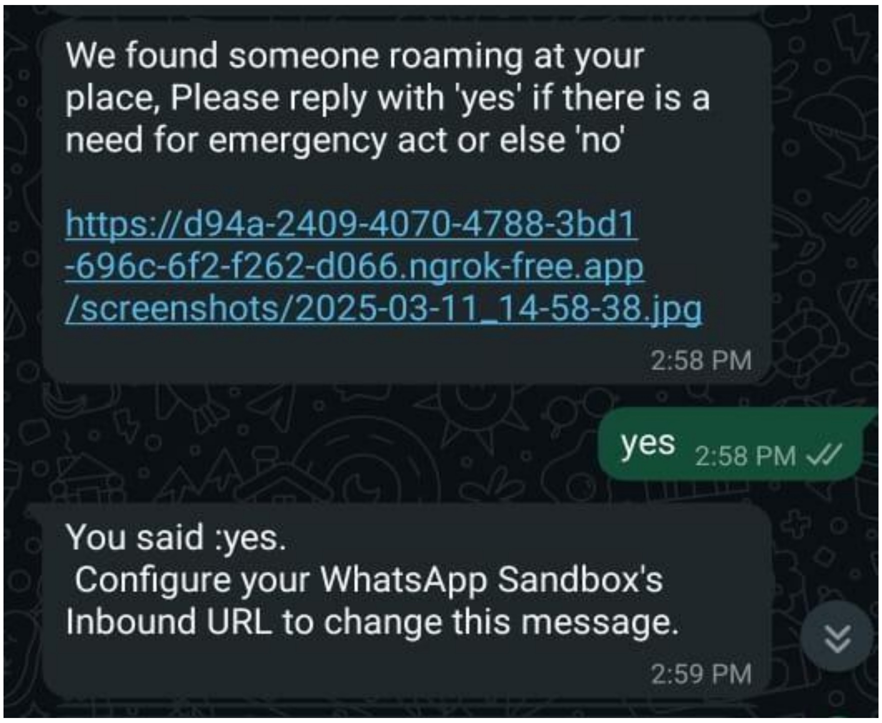


Fig 6.3.2 Detecting persons



We found someone roaming at your place, Please reply with 'yes' if there is a need for emergency act or else 'no'

https://d94a-2409-4070-4788-3bd1-696c-6f2-f262-d066.ngrok-free.app/screenshots/2025-03-11_14-58-38.jpg

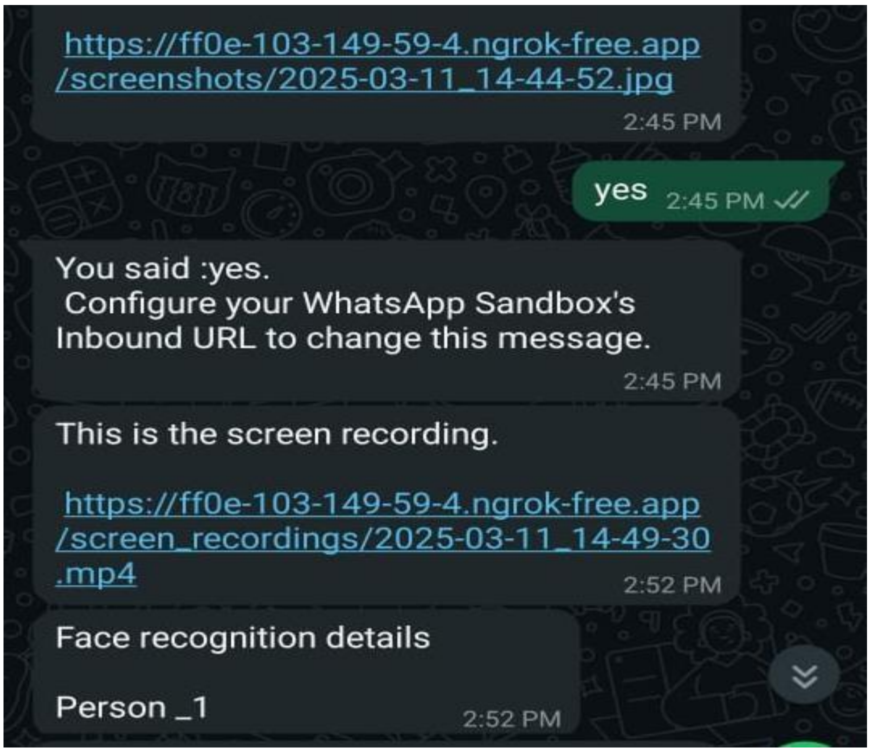
2:58 PM

yes 2:58 PM

You said :yes.
Configure your WhatsApp Sandbox's Inbound URL to change this message.

2:59 PM

Fig 6.3.3 Seeking response from user whether to implement emergency act or not



https://ff0e-103-149-59-4.ngrok-free.app/screenshots/2025-03-11_14-44-52.jpg

2:45 PM

yes 2:45 PM

You said :yes.
Configure your WhatsApp Sandbox's Inbound URL to change this message.

2:45 PM

This is the screen recording.

https://ff0e-103-149-59-4.ngrok-free.app/screen_recordings/2025-03-11_14-49-30.mp4

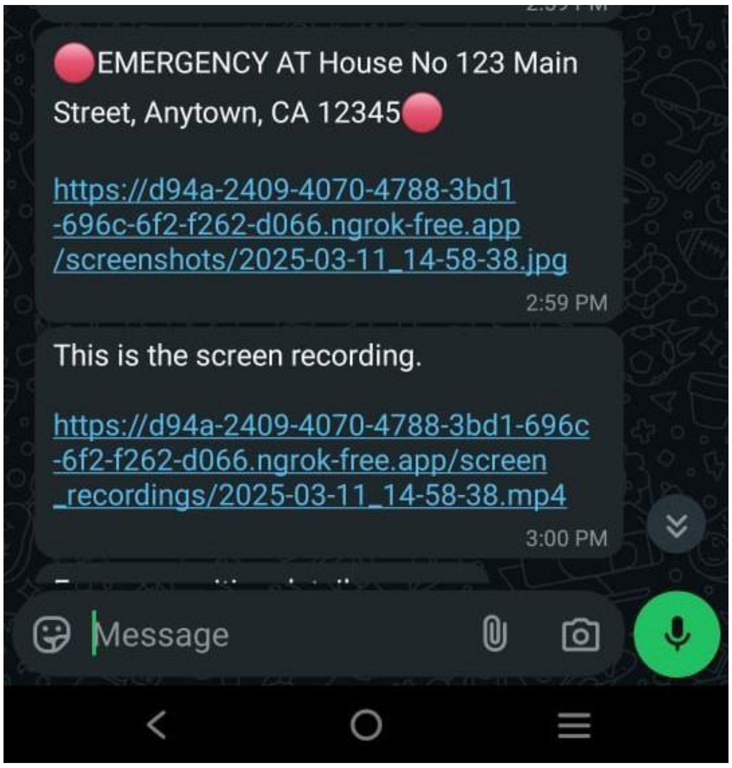
2:52 PM

Face recognition details

Person _1

2:52 PM

Fig 6.3.4 Action taken when user responds “yes”



🔴EMERGENCY AT House No 123 Main Street, Anytown, CA 12345🔴

https://d94a-2409-4070-4788-3bd1-696c-6f2-f262-d066.ngrok-free.app/screenshots/2025-03-11_14-58-38.jpg

2:59 PM

This is the screen recording.

https://d94a-2409-4070-4788-3bd1-696c-6f2-f262-d066.ngrok-free.app/screen_recordings/2025-03-11_14-58-38.mp4

3:00 PM

Fig 6.3.5 Emergency action

7. CONCLUSION

The project “Real-Time Video Object Detection Using YOLO and Computer Vision with Automated WhatsApp Alerts for Security and Surveillance” successfully addresses the need for an efficient and automated surveillance system. By integrating deep learning-based object detection with real-time alerting mechanisms, this system enhances the security infrastructure by minimizing human intervention and providing instant notifications in case of potential threats. The combination of YOLO for object detection, OpenCV for video processing, and Twilio’s WhatsApp API for automated alerts creates a robust and reliable solution that can be deployed in various security-sensitive environments.

Throughout the development process, significant effort was put into optimizing the accuracy and speed of the system. The YOLO model was chosen due to its high-speed detection capability and efficiency in identifying multiple objects within a single frame. OpenCV facilitated real-time video frame extraction and processing, ensuring smooth detection and tracking of objects in live video streams. Additionally, the WhatsApp notification system was implemented to provide immediate alerts to security personnel, reducing response times and improving incident handling. This seamless integration of technologies resulted in a highly responsive security solution that is both effective and scalable.

The project successfully demonstrated several key advantages. The system's ability to operate in real-time makes it ideal for scenarios where quick decision-making is critical, such as monitoring restricted areas, public spaces, or private properties. The automation of security alerts eliminates delays caused by human monitoring and ensures that suspicious activities do not go unnoticed. Furthermore, the implementation is cost-effective compared to traditional security systems, which often require extensive manual surveillance and expensive hardware setups. The flexibility of the solution allows it to be adapted for various applications beyond security, including traffic monitoring, industrial automation, and retail store surveillance.

Despite its achievements, the project encountered certain limitations. One of the primary challenges was handling false positives and false negatives in object detection, which required fine-tuning of the model’s confidence thresholds. Additionally, environmental factors such as poor lighting conditions and occlusions affected detection accuracy, suggesting the need for advanced image enhancement techniques or infrared-based vision systems in future iterations. Another challenge was optimizing performance on resource-constrained hardware, as real-time processing requires significant computational power, particularly when running YOLO on edge devices like Raspberry Pi.

Looking ahead, several enhancements can be made to further improve the system’s efficiency and usability. Integrating the solution with cloud-based AI platforms can enhance scalability and computational performance, allowing for remote monitoring across multiple locations.

Adding facial recognition capabilities can improve threat detection by identifying known or unauthorized individuals, making the system more intelligent and proactive. Additionally, incorporating behavioral analysis through deep learning can enable the system to predict suspicious activities based on movement patterns, further strengthening security measures.

In conclusion, this project provides a solid foundation for next-generation surveillance systems that leverage artificial intelligence, real-time processing, and automated communication. The successful implementation of this solution demonstrates its potential to revolutionize security monitoring by making it faster, more reliable, and highly efficient. With continuous advancements in deep learning and computer vision, such a system can be refined to achieve even greater accuracy and intelligence, ensuring enhanced security across diverse applications.

8. BIBLIOGRAPHY

[1] Hapsari, N.S., Fatman, Y. and Sugiarto, B., Image Extraction in OpenCV Using the Local Binary Pattern Method. Journal of Software Engineering, Information and Communication Technology (SEICT), 5(2), pp.75-86.

[2] Redmon, J. and Farhadi, A., 2018. Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767.

[3] Medida, L.H., Kumar, G.L.N.V.S. and Prasad, M., 2025. Transformative AIoT Applications in Medicine: Real-World Case Studies. In Future Innovations in the Convergence of AI and Internet of Things in Medicine (pp. 367-406). IGI Global Scientific Publishing. https://doi.org/10.4018/979-8-3693-7703-1.ch014.

[4] Santhosh, B., Karthik, P., Kumar, V.K., Akash, C., Reddy, N.J., Rao, N.S. and Kumar, M.V.P., 2025. Using Deep Learning REST APIs, Vid-Sum summarizes videos. International Journal of Engineering Research and Science & Technology, 21(1), pp.611-618.

[5] Tistarelli, M., Dubey, S., Singh, S.K. and Jiang, X., 2023. Computer Vision and Machine Intelligence. London, UK of Great Britain: Springer.

[6] Shreevatsan, S.S., Annambhotla, Y.L., Reji, A. and Rajagopal, S.M., 2025, February. Scheduling Policies in Edge Computing for Real-Time Video Analytics. In 2025 3rd International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT) (pp. 301-306). IEEE.

[7] Sultani, W., Chen, C. and Shah, M., 2018. Real-world anomaly detection in surveillance videos. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 6479-6488).

[8] Hammoudeh, M.A.A., Alsaykhan, M., Alsalameh, R. and Althwaibi, N., 2022. Computer Vision: A Review of Detecting Objects in Videos--Challenges and Techniques. International Journal of Online & Biomedical Engineering, 18(1), 18(01), 15–27. https://doi.org/10.3991/ijoe.v18i01.27577

[9] Ren, S., He, K., Girshick, R. and Sun, J., 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. Advances in neural information processing systems, 28.

[10] Xiao, K., Wang, L., Xu, H., Zhang, P. and Zhang, H., 2025. Research on mechanical part recognition method based on improved mask R-CNN instance segmentation. Robotic Systems and Applications.

[11] Bochkovskiy, A., Wang, C.Y. and Liao, H.Y.M., 2020. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*.

[12] Bertinetto, L., Valmadre, J., Henriques, J.F., Vedaldi, A. and Torr, P.H., 2016. Fully-convolutional siamese networks for object tracking. In Computer vision–ECCV 2016 workshops: Amsterdam, the Netherlands, October 8-10 and 15-16, 2016, proceedings, part II 14 (pp. 850-865). Springer International Publishing.

[13] Viola, P., & Jones, M. J. (2001). Rapid object detection using a boosted cascade of simple features. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), 1, 511-518. https://doi.org/10.1109/CVPR.2001.990517

[14] Prasad, M., Ramadevi, S., Das, G.S., Prasanthi, B.V., Ramesh, A., Sree, P.K. and Lakshmi, K.A., 2024, December. A Study on Fatty Liver Segmentation and Classification as Revealed by CT Scans.
In International Conference on Intelligent Systems and Sustainable Computing (pp. 419-429). Singapore: Springer Nature Singapore. https://doi.org/10.1007/978-981-97-8355-7_36.

[15] Jaber, A. A., Ali, Z. H., Sabti, H. A. A., & Al-Sukeinee, R. J. (2023). Object Detection and Tracking in Real-Time Video Streams Using Convolutional Neural Networks. International Journal of Intelligent Systems and Applications in Engineering, 12(1s), 505–511.

[16] Lee, Ji-Woon, and Hyun-Soo Kang. 2024. "Three-Stage Deep Learning Framework for Video Surveillance" Applied Sciences 14, no. 1: 408. https://doi.org/10.3390/app14010408

[17] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A. and Chen, L.C., 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4510-4520).

[18] Berroukham, A., Housni, K., Lahraichi, M. and Boulfrifi, I., 2023. Deep learning-based methods for anomaly detection in video surveillance: a review. Bulletin of Electrical Engineering and Informatics, 12(1), pp.314-327.

[19] Cai, H., 2022. Application of intelligent real-time image processing in fitness motion detection under internet of things. The Journal of Supercomputing, 78(6), pp.7788-7804.

[20] Tan, M., Pang, R. and Le, Q.V., 2020. Efficientdet: Scalable and efficient object detection. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 10781-10790).

[21] Davies, R.O. and Ostaszewski, A.J., 2019. Optimal forward contract design for inventory: a value-of-waiting analysis. In Ulam Type Stability (pp. 73-96). Cham: Springer International Publishing.

[22] Zhu, P., Wen, L., Bian, X., Ling, H. and Hu, Q., 2018. Vision meets drones: A challenge. arXiv preprint arXiv:1804.07437.

[23] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y. and Berg, A.C., 2016. Ssd: Single shot multibox detector. In Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14 (pp. 21-37). Springer International Publishing.

[24] Twilio, "WhatsApp API Documentation," Available: https://www.twilio.com/whatsapp.

[25] Lin, T.Y., Goyal, P., Girshick, R., He, K. and Dollár, P., 2017. Focal loss for dense object detection. In Proceedings of the IEEE international conference on computer vision (pp. 2980-2988).

9. APPENDIX

9.1 Introduction to Python

Python is a high-level, interpreted programming language known for its simplicity, readability, and versatility. Created by Guido van Rossum and first released in 1991, Python has since become one of the most popular programming languages worldwide, used extensively in various domains, including web development, data science, artificial intelligence, and automation.

Key Features of Python:

1. **Readability:** Python's syntax emphasizes readability and simplicity, making it easy for developers to write and understand code. Its indentation-based block structure eliminates the need for explicit braces or semicolons, enhancing code clarity and reducing cognitive overhead.

2. **Versatility:** Python supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Its extensive standard library provides a wide range of modules and packages for diverse tasks, from file I/O and networking to web development and scientific computing.

3. **Interpreted Nature:** Python is an interpreted language, meaning code is executed line by line by an interpreter, eliminating the need for compilation before execution. This enables rapid prototyping, development, and testing, facilitating a faster development cycle.

4. **Dynamically Typed:** Python is dynamically typed, allowing variables to be assigned values without explicit declaration of data types. This flexibility simplifies code writing and enhances code expressiveness, but it also requires careful attention to variable types to avoid runtime errors.

5. **Strong Community Support:** Python boasts a vibrant and active community of developers, enthusiasts, and contributors who continually enhance and expand its ecosystem. The Python Package Index (PyPI) hosts thousands of third-party libraries and frameworks, providing solutions for virtually any programming task.

Applications of Python:

- **Web Development:** Frameworks like Django and Flask enable rapid development of web applications.
- **Data Science and Machine Learning:** Libraries such as NumPy, Pandas, and scikit-learn facilitate data manipulation, analysis, and machine learning.
- **Artificial Intelligence and Natural Language Processing:** Tools like TensorFlow, PyTorch, and NLTK empower developers to build intelligent systems and language processing applications.

9.2 Introduction to OpenCV:

OpenCV is an open-source library originally developed by Intel in 1999 and later maintained by a community of developers. It offers a comprehensive suite of computer vision algorithms and tools, making it a popular choice for a wide range of applications, including object detection, image segmentation, face recognition, and more.

Preprocessing Ear Images:

In our ear biometrics project, we utilize OpenCV for preprocessing ear images before feeding them into our classification model. This preprocessing pipeline may include tasks such as resizing images to a standardized format, converting images to grayscale, and applying filters to enhance image quality and remove noise.

Feature Extraction and Detection:

OpenCV provides a variety of feature extraction and detection algorithms that we can leverage to identify key characteristics and patterns in ear images. For example, we can use Haar cascades for ear detection, which are pre-trained classifiers capable of locating ear regions within images accurately.

Real-Time Image Processing:

OpenCV offers real-time image processing capabilities, making it suitable for applications that require fast and efficient processing of video streams or live camera feeds. In our project, we can use OpenCV to capture live video from a camera, detect ears in real-time, and perform biometric authentication on the fly.

Integration with Machine Learning Libraries:

OpenCV seamlessly integrates with popular machine learning libraries like TensorFlow and Keras, allowing us to combine the power of computer vision with deep learning techniques. We can preprocess images using OpenCV and then pass them as input to our machine learning models built with TensorFlow or Keras for classification.

Cross-Platform Compatibility:

OpenCV is cross-platform and supports various operating systems, including Windows, macOS, Linux, Android, and iOS. This cross-platform compatibility ensures that our ear biometrics system can run on a wide range of devices and platforms, making it versatile and accessible.

Community Support and Documentation:

OpenCV boasts a vibrant community of developers and researchers who contribute to its ongoing development and maintenance. The extensive documentation, tutorials, and code examples provided by the OpenCV community make it easy for developers to learn and use the library for their projects.

9.3 Introduction to YoloV4:

YOLO is a deep learning-based object detection algorithm designed for high-speed and accurate detection in real-time applications. Unlike traditional object detection methods that process images in multiple stages, YOLO analyzes an entire image in a single pass, significantly reducing computation time.

- Single-Pass Object Detection – Unlike region-based approaches like R-CNN, YOLO detects all objects in an image simultaneously, making it highly efficient.
- Darknet-53 Backbone – Uses a convolutional neural network (CNN) with 53 layers, trained for image classification and feature extraction.
- Multi-Scale Detection – YOLO can detect objects at different scales, improving accuracy for small and large objects in the same frame.
- Anchor Boxes and Bounding Box Regression – Uses predefined anchor boxes to predict object locations more accurately.
- Non-Maximum Suppression (NMS) – Filters out redundant bounding boxes and keeps only the most confident predictions, reducing false positives.

In this project, YOLO is responsible for detecting and classifying objects in the video feed. It analyzes each frame and identifies predefined objects such as humans, vehicles, or suspicious items. When an object of interest is detected, the system triggers an alert mechanism. The use of YOLO ensures that detections are performed in real-time without significant delays, making it suitable for security and surveillance applications.

9.4 Introduction to Twilio API for WhatsApp Alerts:

Twilio API is a cloud-based communication platform that provides programmable messaging services, including SMS, voice, and WhatsApp messaging. It enables automated notifications for real-time communication, making it ideal for alerting users about critical events.

- Integration with Python – Twilio's API allows seamless integration with Python, enabling automated message sending based on event triggers.
- Secure and Reliable Communication – WhatsApp messages sent through Twilio are end-

to-end encrypted, ensuring privacy and data security.

- Support for Multimedia Messaging – Twilio allows sending not just text messages but also images, videos, and documents.
- Immediate Response Mechanism – The system ensures real-time delivery of security alerts to designated personnel.

In this project, Twilio API is used to send automated WhatsApp alerts whenever the YOLO model detects a suspicious object or activity. Once an alert is triggered, a notification containing the detected object's details and timestamp is sent to security personnel. This ensures quick action, reducing response time and improving overall surveillance efficiency.

Appendix-A

Project Repository Details

Project Title: Real-Time Video Object Detection using Yolo and Computer Vision with Automated WhatsApp Alerts for Security and Surveillance

Batch: A4

Batch Members:
1. Adapala Lohitha (21B01A0501)
2. Ballarapu Glory (21B01A0511)
3. Bonam Renuka Lakshmi Tejaswini (21B01A0526)
4. Boorlagadda VNS Asritha (21B01A0527)
5. Bhupatiraju Harshita (22B05A0501)

Department: Computer Science and Engineering

Institution: Shri Vishnu Engineering College for Women

Guide: Dr. M. Prasad

Submission Date: 11/04/2025

Project Repository Link
The complete project files, including source code, documentation, and additional resources, are available at the following GitHub repository.

GitHub Repository: https://github.com/TejaswiniBonam/Real-Time-Video-Object-Detection-Using-YOLO-and-Computer-Vision-with-Automated-WhatsApp-Alerts

For quick access, scan the QR code below: