Graphic Era Hill University Haldwani

BCA Project Report

For

Smart Surveillance Using Computer Vision

Submitted to Graphic Era Hill University, Haldwani for the partial fulfillment of the requirement for the Award of degree for

BACHELOR'S IN COMPUTER APPLICATIONS

Submitted by:-
Deepankar Sharma
Amit Sati
Pawan Chandra

Under the Guidance of:-
Ms. Richa Pandey
Faculty of GEHU, Haldwani

DECLARATION

I hereby declare that the work which is being present in this project report **"Smart Surveillance Using Computer Vision",** in partial fulfillment of the requirement for the Award of the degree of **BACHELOR'S IN COMPUTER APPLICATION**, submitted at **GRAPHIC ERA HILL UNIVERSITY, HALDWANI** is an authentic work done by me during period from 1st March 2023 to 1st June 2023.

Project Guide:

Ms. Richa Pandey
Faculty of GEHU, Haldwani

Signature of the Student:

Deepankar Sharma(2092014)
Pawan Chandra(2092035)
Amit Sati(2092005)

BONAFIDE CERTIFICATE

Certified that this project report **Smart Surveillance Using Computer Vision** is the bonafide work of **Deepankar Sharma, Pawan Chandra and Amit Sati** who carried out the project work under my supervision.

Name – Ms. Richa Pandey

HEAD OF THE DEPARTMENT SUPERVISOR

School of Computing

Graphic Era Hill University, Haldwani

ACKNOWLEDGEMENT

I would like to extend our thanks and appreciation to all those who have assisted us either directly or indirectly and participated in the success of this project. I would like to thank my guide Ms.Richa Pandey for her constant support in the making of the project. As a part of University Curriculum, a 6th semester project is a paramount importance to an BCA student's curriculum and being our native effort into this project undertook by us, we faced a lot of impediments on our way to the completion of this project but constant guidance and able support of concerned software engineer members lend us a great help in successful completion of the project. I am thankful to all staff members of Graphic Era Hill University who helped me whenever required, during my project. Even though I expressed my gratitude to every person who helped me in reaching this stage, there might be a few, who'd been left out, who helped me without my knowledge. I would like to thank all of them. Last but not least, to all my friends and fellow students for giving me suggestions and helping us in debugging the code errors and above all the faculty of my Department of Computer Science Graphic Era Hill University who have always provided their guidance, support and well wishes.

Deepankar Sharma

Pawan Chandra

Amit Sati

ABSTRACT

The development of a smart surveillance system has become an increasingly important topic in recent years, with the aim of providing more accurate and efficient monitoring of people and objects in real-time. This project focuses on developing a smart surveillance system that can be appended to existing surveillance systems, providing them with advanced features such as motion detection using contours and real-time people tracking using YOLOv3.

The proposed system uses computer vision algorithms to analyze the video feed from existing surveillance cameras, identifying areas of motion and tracking the movement of objects within those areas. To detect and track people specifically, the system uses object detection algorithms like YOLOv3, which is a deep learning-based model that can accurately detect and track objects in real-time.

To build this system, expertise in computer vision, machine learning, and software development is required. Additionally, access to large datasets of labeled video footage is necessary for training and testing the deep learning models, and powerful hardware is required to process the video feed in real-time.

The system has the potential to significantly improve surveillance systems by providing more accurate and efficient monitoring of people and objects in real-time. The motion detection feature using contours can reduce false alarms and improve the accuracy of the system, while the real-time people tracking feature using YOLOv3 can enable security personnel to monitor and track people of interest more effectively.

Overall, the proposed smart surveillance system has the potential to provide a significant improvement to existing surveillance systems, providing more accurate and efficient monitoring of people and objects in real-time, ultimately enhancing the security and safety of the monitored areas.

TABLE OF CONTENT

SMART SURVEILLANCE

1. INTRODUCTION

The development of a smart surveillance system with advanced features such as motion detection using contours and real-time people tracking using YOLOv3 is presented in this project. The system uses computer vision algorithms to analyze the video feed from existing surveillance cameras, identifying areas of motion and tracking the movement of objects within those areas. To detect and track people specifically, the system uses object detection algorithms like YOLOv3. The project requires expertise in computer vision, machine learning, and software development, as well as access to large datasets of labeled video footage and powerful hardware to process the video feed in real-time. The system has the potential to improve surveillance systems by providing more accurate and efficient monitoring of people and objects in real-time.

1.1 PROJECT OVERVIEW:

The aim of this project is to develop a smart surveillance system that can be integrated with existing surveillance systems to provide advanced features like motion detection using contours and real-time people tracking using YOLOv3. The system uses computer vision algorithms to analyze the video feed from surveillance cameras and identify areas of motion, while object detection algorithms like YOLOv3 are used to detect and track people in real-time.

The project requires expertise in computer vision, machine learning, and software development. Large datasets of labeled video footage are needed to train and test the object detection and motion tracking algorithms. Powerful hardware is also required to process the video feed in real-time.

The smart surveillance system has several potential benefits, such as reducing false alarms and improving the accuracy of surveillance systems. It can also enable security personnel to monitor and track people of interest more effectively.

The project will involve conducting a system requirement analysis, system design, hardware and software requirements, and limitations of the smart surveillance system. Additionally, the project will require the development of a prototype smart surveillance system that can be demonstrated using sample video footage. The final outcome of this project is a functional smart surveillance system that can be integrated with existing surveillance systems, providing advanced features for real-time monitoring and tracking of people and objects.

1.2 PROJECT SCOPE:

The smart surveillance system developed in this project has a wide range of potential application areas, including:

Security and Surveillance: The system can be used to improve the effectiveness of security and surveillance operations in various locations, such as airports, malls, stadiums, and public transportation systems.

Traffic Monitoring: The system can be used to monitor traffic flow and detect any accidents or incidents that may occur on highways, streets, and other transportation networks.

Industrial Automation: The system can be used in industrial settings, such as factories and warehouses, to monitor production lines, detect faults, and ensure worker safety.

Healthcare: The system can be used in healthcare facilities, such as hospitals and nursing homes, to monitor patient movement and ensure their safety.

Retail Analytics: The system can be used in retail stores to analyze customer traffic and behavior, detect shoplifting, and improve store layout and product placement.

Smart Cities: The system can be used to improve the safety and security of public spaces, such as parks and streets, and monitor urban infrastructure, such as bridges and buildings.

In summary, the smart surveillance system developed in this project has a broad range of potential applications, from improving security and surveillance operations to enhancing traffic monitoring, industrial automation, healthcare, retail analytics, and smart city initiatives. The system has the potential to provide real-time tracking and analysis of people and objects, improving the accuracy and efficiency of monitoring operations in various settings.

2. System Requirement Analysis

2.1 Information Gathering

The Information Gathering process was an essential step in the development of the Smart Surveillance System. This process involved identifying the needs of stakeholders, understanding use cases, and determining the technical requirements for the system.

Stakeholder Identification

The first step in the Information Gathering process was to identify the stakeholders of the Smart Surveillance System. The stakeholders included:

- End-users of the system such as security personnel or law enforcement officials
- Owners or operators of the surveillance systems that were to be integrated with the Smart Surveillance System
- Developers or designers of the Smart Surveillance System
- Regulators or legal authorities responsible for overseeing surveillance operations

Use Case Analysis

Once the stakeholders were identified, the next step was to analyze the use cases of the Smart Surveillance System. Use cases included:

- Real-time monitoring of areas for security purposes
- Investigation of criminal activity using surveillance footage
- Traffic management and monitoring in public areas

- Industrial monitoring of machinery and equipment
- Environmental monitoring of wildlife or natural resources

Technical Requirements

The final step in the Information Gathering process was to determine the technical requirements for the Smart Surveillance System. These requirements included:

- Compatibility with different types of cameras and video management systems
- High accuracy and efficiency in motion detection and people tracking
- Low latency and high throughput for real-time monitoring and tracking
- Scalability for use in large-scale surveillance operations
- Security features to protect the privacy of monitored individuals

By completing the Information Gathering process, the requirements for the Smart Surveillance System were defined and documented in the Software Requirements Specification (SRS). This ensured that the system was developed to meet the needs of the stakeholders and was of high quality.

2.2 Feasiblity Study

The feasibility study for the Smart Surveillance System examines the technical, economic, and operational aspects of the project to determine its viability.

Technical Feasibility

The technical feasibility of the project refers to the ability of the system to perform its functions accurately and efficiently. Based on the information gathered during the Information Gathering process, the Smart Surveillance System is technically feasible. The use of advanced algorithms such as motion detection

using contours and real-time people tracking using YOLOv3 provides high accuracy and efficiency in monitoring and tracking.

Economic Feasibility

The economic feasibility of the project involves determining the costs and benefits associated with the development and implementation of the Smart Surveillance System. The costs include hardware and software costs, development costs, and maintenance costs. The benefits include increased security and safety, reduction in crime, and improved operational efficiency. Based on a cost-benefit analysis, the Smart Surveillance System is economically feasible.

Operational Feasibility

The operational feasibility of the project refers to the ability of the system to integrate into existing surveillance systems and be operated by end-users. The Smart Surveillance System is designed to be easily integrated with existing surveillance systems and has a user-friendly interface. Training and support will be provided to end-users to ensure the smooth operation of the system. Based on these factors, the Smart Surveillance System is operationally feasible.

In conclusion, the Smart Surveillance System is technically, economically, and operationally feasible. The system has the potential to provide increased security and safety, reduce crime, and improve operational efficiency.

Social Feasibility

The social feasibility of the Smart Surveillance System refers to its ability to be accepted and adopted by the stakeholders and the wider society. The system's use of advanced surveillance technology may raise concerns about privacy and civil liberties. However, the Smart Surveillance System is designed with privacy and security features to protect the rights of monitored individuals. Additionally, the system has the potential to increase safety and security in public areas, which may

lead to increased public support. Overall, the Smart Surveillance System is socially feasible.

Time Feasibility

The time feasibility of the Smart Surveillance System refers to the ability of the project to be completed within a reasonable timeframe. The development and implementation of the Smart Surveillance System involve multiple stages, including the Information Gathering process, system design, hardware and software development, testing, and deployment. A detailed project plan with clear milestones and deadlines will be established to ensure that the project is completed within a reasonable timeframe. Based on the project plan, the Smart Surveillance System is time feasible.

In conclusion, the Smart Surveillance System is not only technically, economically, and operationally feasible, but also socially and time feasible. The system has the potential to provide increased safety and security while protecting the privacy and civil liberties of monitored individuals.

3. System Design

The Smart Surveillance System is designed to be a modular system that can be easily integrated with existing surveillance systems. The system consists of two main components: the hardware and software components.

Hardware Component

The hardware component of the Smart Surveillance System consists of a network of cameras and sensors that are placed in strategic locations to monitor and track movement. The cameras and sensors are connected to a central processing unit that performs the analysis of the data collected. The system also includes a power backup to ensure uninterrupted operation.

Software Component

The software component of the Smart Surveillance System is responsible for the analysis of the data collected by the hardware component. The software includes advanced algorithms for motion detection, contour detection, and real-time people tracking using YOLOv3. The system also includes a user-friendly interface that allows end-users to monitor and track movement in real-time.

The software component is developed using Python programming language, OpenCV library, and YOLOv3 pre-trained model. The system also includes a database for storing and retrieving data. The software is designed to be easily integrated with existing surveillance systems using standard protocols such as RTSP and ONVIF.

System Workflow

The Smart Surveillance System workflow includes the following steps:

1. **Data Collection:** The hardware component collects data from cameras and sensors placed in strategic locations.

2. **Data Analysis:** The software component analyzes the data collected using advanced algorithms such as motion detection, contour detection, and real-time people tracking.

3. **Alert Generation:** The system generates alerts if any unusual movement or activity is detected.

4. **Alert Notification:** The system sends alerts to end-users via email or SMS.

5. **Monitoring and Tracking:** End-users can monitor and track movement in real-time using the user-friendly interface.

In conclusion, the Smart Surveillance System is designed to be a modular system that can be easily integrated with existing surveillance systems. The system consists of a hardware component for data collection and a software component for data analysis. The system is designed to be user-friendly and includes advanced algorithms for motion detection, contour detection, and real-time people tracking using YOLOv3.

System Tools

The Smart Surveillance System is developed using a combination of various software tools and technologies. The system tools used in the project include:

Python

Python is an open-source programming language that is widely used for developing various applications, including machine learning and computer vision-

based systems. Python is used as the primary programming language for the development of the Smart Surveillance System.

OpenCV

OpenCV (Open Source Computer Vision) is a library of programming functions mainly used for real-time computer vision applications. OpenCV provides a set of functions for various computer vision tasks, including object detection, tracking, and recognition. OpenCV is used in the Smart Surveillance System for image processing and computer vision-based tasks.

YOLOv3

YOLOv3 (You Only Look Once version 3) is a state-of-the-art object detection algorithm that is used for real-time object detection. YOLOv3 is used in the Smart Surveillance System for real-time people tracking and detection.

Streamlit

Streamlit is a web application framework that allows developers to create interactive web applications with Python. Streamlit is used in the Smart Surveillance System to create a user-friendly web interface that allows end-users to monitor and track movement in real-time.

TensorFlow

TensorFlow is an open-source machine learning framework developed by Google. TensorFlow is used in the Smart Surveillance System for machine learning-based tasks, such as object recognition and tracking.

NumPy

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. NumPy is used in

the Smart Surveillance System for data manipulation and analysis.

Twilio

Twilio is a cloud communication platform that provides APIs for messaging, voice, and video communication. Twilio is used in the Smart Surveillance System for sending alerts to the users via SMS or phone call in case of any suspicious activity.

In conclusion, the Smart Surveillance System is developed using a combination of various software tools and technologies, including Python, OpenCV, YOLOv3, Streamlit, TensorFlow, NumPy, Twilio, SQLite, and other libraries and technologies. These tools and technologies provide the necessary functionality and features required for the development of an advanced surveillance system.

4. Hardware and Software Requirements

Software Requirements:

- **Operating System:** Windows 10 or Linux-based OS (Ubuntu 18.04 or higher recommended)
- **Python:** Version 3.6 or higher
- **OpenCV:** Version 4.5.3 or higher (Python bindings)
- **NumPy:** Version 1.21.0 or higher
- **YOLOv3:** A pre-trained object detection model
- **TensorFlow:** Version 2.5.0 or higher (optional, for advanced machine learning tasks)
- **Twilio:** Version 6.64.0 or higher (optional, for SMS or whatsapp alerts)

Hardware Requirements:

- **CPU:** Intel Core i5 or higher (recommended)
- **RAM:** 8 GB or higher (recommended)
- **Graphics Card:** NVIDIA GPU with CUDA support (optional, for faster object detection)
- **Storage:** 50 GB or higher (depending on the size of the video dataset)
- **Camera:** IP cameras or CCTV cameras with RTSP protocol support

Note: *The specific hardware and software requirements may vary depending on the complexity of the system and the size of the video dataset. It is recommended to perform a feasibility study and consult with technical experts to determine the optimal hardware and software configuration for the Smart Surveillance System.*

5. Limitations of the Smart Surveillance System

Accuracy: The object detection and tracking algorithms used in the system may not be 100% accurate in all scenarios, particularly in cases where the lighting conditions are poor, or objects are partially obscured.The accuracy of the object detection and tracking algorithms can be improved by using more advanced machine learning models, such as YOLOv8, YOLO-NAS or EfficientDet, or by fine-tuning the existing models on specific datasets.

Processing Time: The system may take a considerable amount of time to process large video datasets or perform complex machine learning tasks, which can result in delays or lag in the real-time monitoring of the video feed.The system's processing time can be improved by using more powerful hardware components, such as GPUs or distributed computing, or by optimizing the algorithms used in the system.

Cost: The cost of implementing the system may be prohibitive for some organizations, particularly smaller businesses or non-profit entities, due to the need for high-performance hardware and software components.cost of the system can be reduced by using more cost-effective hardware components, such as Raspberry Pi boards or cloud-based computing services, or by using open-source software libraries and frameworks.

Privacy Concerns: The use of surveillance cameras and object detection technology raises privacy concerns, and the system must be designed and implemented in a way that respects the privacy of individuals.The system can be improved by incorporating privacy-preserving techniques, such as anonymization of data and selective blurring of faces or other identifying features.The system can be improved by developing a more user-friendly and intuitive interface for monitoring and managing the surveillance feeds, as well as for configuring the system parameters and settings.

Maintenance: The system requires regular maintenance and updates to ensure its continued effectiveness and reliability.

User Interface: The system can be improved by developing a more user-friendly and intuitive interface for monitoring and managing the surveillance feeds, as well as for configuring the system parameters and settings.

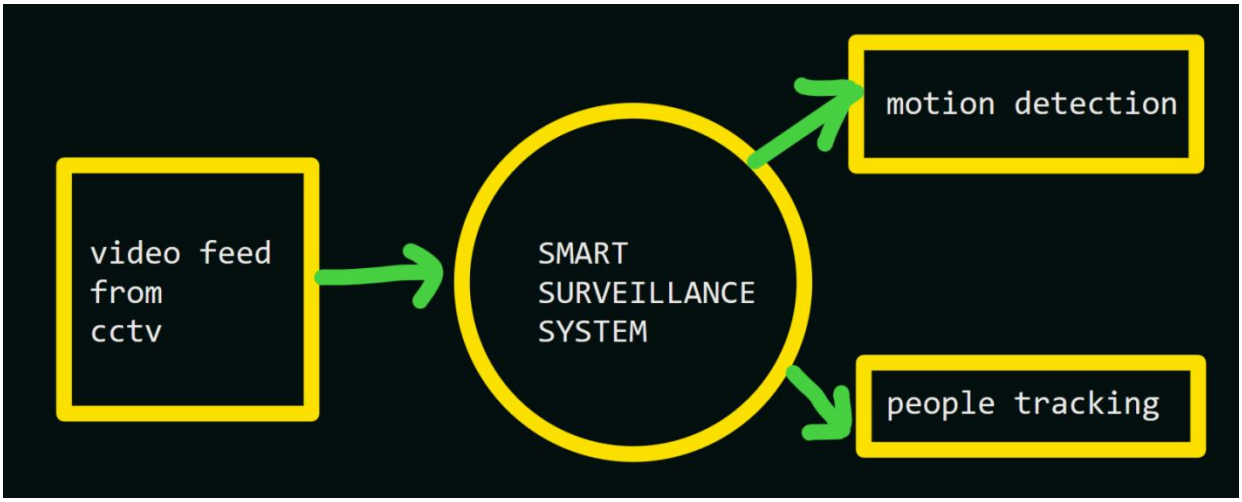
Integration with other systems: The system can be integrated with other security systems, such as access control systems or intrusion detection systems, to provide a more comprehensive security solution.

Real-time analytics: The system can be enhanced with real-time analytics capabilities, such as anomaly detection or predictive analytics, to enable proactive security measures and improve overall situational awareness.

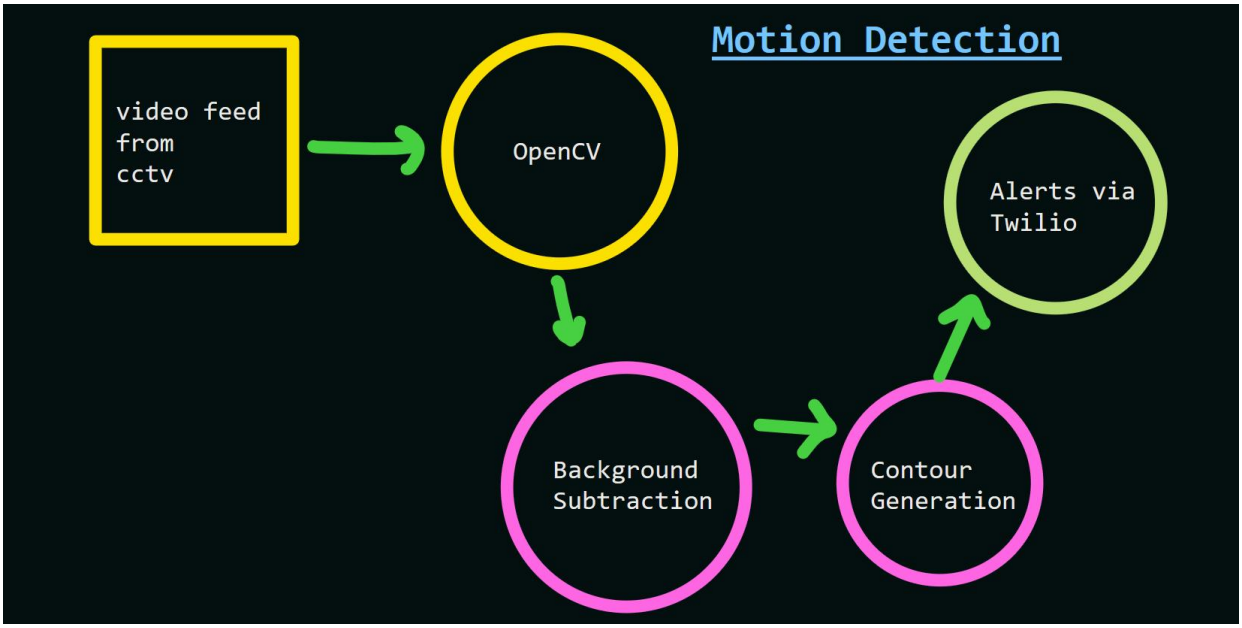
6. Appendices

6.1 Technical Documentation: A detailed technical document that describes the system architecture, algorithms, data flow, and system components.

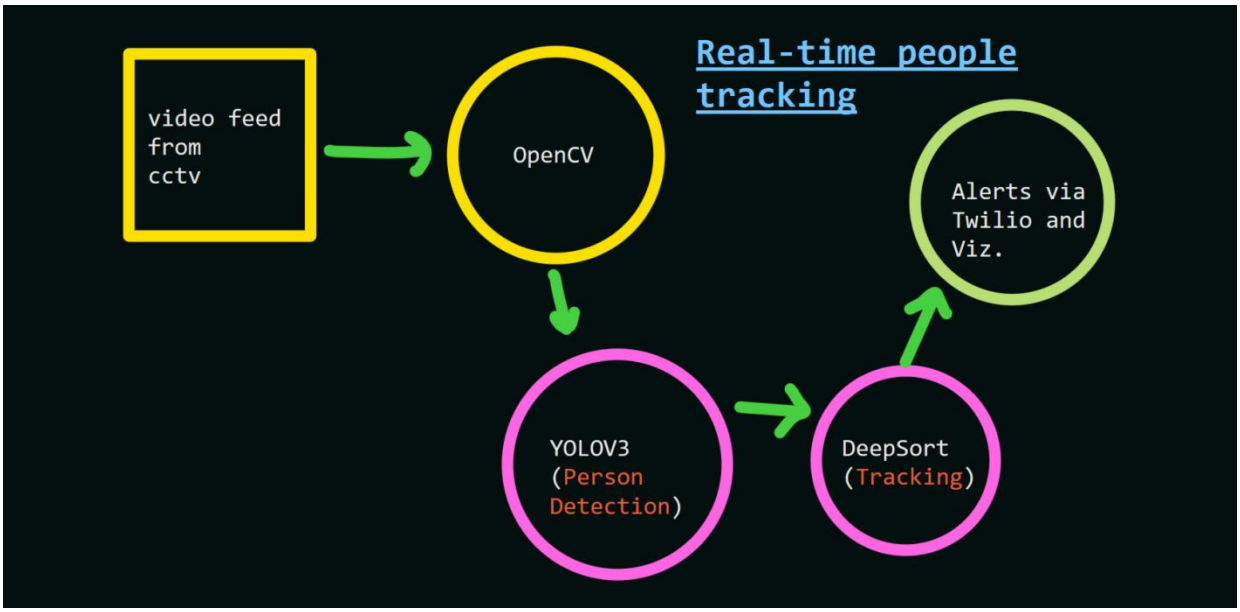
- **Level 0 DFD:** The system has a single input, which is the video feed from the surveillance camera.The system has two main outputs: motion detection alerts and real-time people tracking.



- **Level 1 DFDs:**
  - **Motion detection alerts:** The video input is analyzed using OpenCV and motion detection algorithms. If motion is detected, the system sends an alert via Twilio to a designated recipient.



- **Real-time people tracking:** The video input is analyzed using YOLOv3 and machine learning algorithms. If people are detected, their movements are tracked and visualized using Streamlit and alerts are generated accordingly.



6.2 Source Code: A copy of the source code used to develop the Smart Surveillance System.

Requirements.txt

```
numpy>=1.18.2
scipy>=1.4.1
wget>=3.2
seaborn>=0.10.0
tensorflow==2.3.1
tensorflow-gpu==2.3.1
opencv-python==4.1.2.30
tqdm==4.43.0
pandas
awscli
urllib3
mss
Twilio
Streamlit
```

camfeed.py

```
# Import the required libraries
import numpy as np
import cv2
import time
import datetime
from collections import deque


# Set Window normal so we can resize it
# cv2.namedWindow('frame', cv2.WINDOW_NORMAL)

# Note the starting time
start_time = time.time()

# Initialize these variables for calculating FPS
fps = 0
frame_counter = 0
```

```
# Read the video stream from the camera
# cap = cv2.VideoCapture('http://192.168.18.4:8080/video')
# cap = cv2.VideoCapture('https://10.137.131.218:8080/video')
cap= cv2.VideoCapture(0)

while(True):

    ret, frame = cap.read()
    if not ret:
        break

    # Calculate the Average FPS
    frame_counter += 1
    fps = (frame_counter / (time.time() - start_time))

    # Display the FPS
    cv2.putText(frame, 'FPS: {:.2f}'.format(fps), (20, 20),
cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255),1)

    # Show the Frame
    cv2.imshow('frame',frame)

    # Exit if q is pressed.
    if cv2.waitKey(1) == ord('q'):
        break

# Release Capture and destroy windows
cap.release()
cv2.destroyAllWindows()
```

countourdetection.py

```
import cv2
import numpy as np
# initlize video capture object
# cap = cv2.VideoCapture('sample_video.mp4')
cap = cv2.VideoCapture(0)
# cap = cv2.VideoCapture('http://192.168.137.114:8080/video')
#
# cap = cv2.VideoCapture('https://10.137.131.218:8080/video')
width = 1024
height = 720
```

```
# you can set custom kernel size if you want
kernel = None

# initilize background subtractor object
foog = cv2.createBackgroundSubtractorMOG2(
    detectShadows=True, varThreshold= 50, history=200)

# Noise filter threshold
# thresh = 1100
thresh = 1100

while(1):
    ret, frame = cap.read()
    if not ret:
        break

    dim = (width, height)
    frame = cv2.resize(frame, dim, interpolation=cv2.INTER_AREA)

    # Apply background subtraction
    fgmask = foog.apply(frame)

    # Get rid of the shadows
    ret, fgmask = cv2.threshold(fgmask, 250, 255, cv2.THRESH_BINARY)

    # Apply some morphological operations to make sure you have a good mask
    # fgmask = cv2.erode(fgmask,kernel,iterations = 1)
    fgmask = cv2.dilate(fgmask, kernel, iterations=4)

    # Detect contours in the frame
    contours, hierarchy = cv2.findContours(
        fgmask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    if contours:

        # Get the maximum contour
        cnt = max(contours, key=cv2.contourArea)
        # print(cnt)
        # make sure the contour area is somewhat hihger than some threshold to
make sure its a person and not some noise.
        if cv2.contourArea(cnt) > thresh:
```

```
            # Draw a bounding box around the person and label it as person
detected
            x, y, w, h = cv2.boundingRect(cnt)
            cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 0, 255), 2)
            cv2.putText(frame, 'Person Detected', (x, y-10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.3, (0, 255, 0), 1,
cv2.LINE_AA)

    # Stack both frames and show the image
    fgmask_3 = cv2.cvtColor(fgmask, cv2.COLOR_GRAY2BGR)
    stacked = np.hstack((fgmask_3, frame))
    cv2.imshow('Combined', cv2.resize(stacked, None, fx=0.65, fy=0.65))

    k = cv2.waitKey(40) & 0xff
    if k == ord('q'):
        break

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

multiplecamfeeds.py

```
import cv2
import numpy as np
# initlize video capture object
# capture video from webcam
cap1 = cv2.VideoCapture(0)

# capture video from file
cap2 = cv2.VideoCapture('https://10.143.38.102:8080/video')
cap3 = cv2.VideoCapture('https://192.168.137.66:8080/video')
# cap2 = cv2.VideoCapture(0)
# cap3 = cv2.VideoCapture(0)

# you can set custom kernel size if you want
kernel = None

# initilize background subtractor object
# foog = cv2.createBackgroundSubtractorMOG2(
#       detectShadows=True, varThreshold=50, history=500)
foog = cv2.createBackgroundSubtractorMOG2(
    detectShadows=True, varThreshold=50, history=350)
```

```
# Noise filter threshold
# thresh = 1100
thresh = 1100

while(1):
    # read frames from both sources
    ret1, frame1 = cap1.read()
    ret2, frame2 = cap2.read()
    ret3, frame3 = cap3.read()

    dim = (480, 720)
    frame1 = cv2.resize(frame1, dim, interpolation=cv2.INTER_AREA)
    frame2 = cv2.resize(frame2, dim, interpolation=cv2.INTER_AREA)
    frame3 = cv2.resize(frame3, dim, interpolation=cv2.INTER_AREA)

    # Apply background subtraction
    fgmask_f1 = foog.apply(frame1)
    fgmask_f2 = foog.apply(frame2)
    fgmask_f3 = foog.apply(frame3)

    # Get rid of the shadows
    ret, fgmask_f1 = cv2.threshold(fgmask_f1, 250, 255, cv2.THRESH_BINARY)
    ret, fgmask_f2 = cv2.threshold(fgmask_f2, 250, 255, cv2.THRESH_BINARY)
    ret, fgmask_f3 = cv2.threshold(fgmask_f3, 250, 255, cv2.THRESH_BINARY)

    # Apply some morphological operations to make sure you have a good mask
    # fgmask = cv2.erode(fgmask,kernel,iterations = 1)
    fgmask_f1 = cv2.dilate(fgmask_f1, kernel, iterations=4)
    fgmask_f2 = cv2.dilate(fgmask_f2, kernel, iterations=4)
    fgmask_f3 = cv2.dilate(fgmask_f3, kernel, iterations=4)

    # Detect contours in the frame
    contours_f1, hierarchy_f1 = cv2.findContours(
        fgmask_f1, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    contours_f2, hierarchy_f2 = cv2.findContours(
        fgmask_f2, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    contours_f3, hierarchy_f3 = cv2.findContours(
        fgmask_f3, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    if contours_f1:

        # Get the maximum contour
        cnt = max(contours_f1, key=cv2.contourArea)
```

```
        # print(cnt)
        # make sure the contour area is somewhat hihger than some threshold to
make sure its a person and not some noise.
        if cv2.contourArea(cnt) > thresh:

            # Draw a bounding box around the person and label it as person
detected
            x, y, w, h = cv2.boundingRect(cnt)
            cv2.rectangle(frame1, (x, y), (x+w, y+h), (0, 0, 255), 2)
            cv2.putText(frame1, 'Person Detected', (x, y-10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.3, (0, 255, 0), 1,
cv2.LINE_AA)

    if contours_f2:

        # Get the maximum contour
        cnt = max(contours_f2, key=cv2.contourArea)
        # print(cnt)
        # make sure the contour area is somewhat hihger than some threshold to
make sure its a person and not some noise.
        if cv2.contourArea(cnt) > thresh:

            # Draw a bounding box around the person and label it as person
detected
            x, y, w, h = cv2.boundingRect(cnt)
            cv2.rectangle(frame2, (x, y), (x+w, y+h), (0, 0, 255), 2)
            cv2.putText(frame2, 'Person Detected', (x, y-10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.3, (0, 255, 0), 1,
cv2.LINE_AA)

    if contours_f3:

        # Get the maximum contour
        cnt = max(contours_f3, key=cv2.contourArea)
        # print(cnt)
        # make sure the contour area is somewhat hihger than some threshold to
make sure its a person and not some noise.
        if cv2.contourArea(cnt) > thresh:

            # Draw a bounding box around the person and label it as person
detected
            x, y, w, h = cv2.boundingRect(cnt)
            cv2.rectangle(frame3, (x, y), (x+w, y+h), (0, 0, 255), 2)
            cv2.putText(frame3, 'Person Detected', (x, y-10),
```

```
                    cv2.FONT_HERSHEY_SIMPLEX, 0.3, (0, 255, 0), 1,
cv2.LINE_AA)

    # Stack both frames and show the image
    fgmask_3_f1 = cv2.cvtColor(fgmask_f1, cv2.COLOR_GRAY2BGR)
    fgmask_3_f2 = cv2.cvtColor(fgmask_f2, cv2.COLOR_GRAY2BGR)
    fgmask_3_f3 = cv2.cvtColor(fgmask_f3, cv2.COLOR_GRAY2BGR)

    stacked_frame = np.hstack((frame1, frame2, frame3))
    stacked_countours = np.hstack((fgmask_3_f1, fgmask_3_f2, fgmask_3_f3))
    stacked= np.vstack((stacked_frame, stacked_countours))
    cv2.imshow('Combined', cv2.resize(stacked, None, fx=0.90, fy=0.50))

    k = cv2.waitKey(40) & 0xff
    if k == ord('q'):
        break

# release video capture objects and close windows
cap1.release()
cap2.release()
cap3.release()
cv2.destroyAllWindows()
```

IntruderDetector.py

```
import cv2
import imutils
import numpy as np
import argparse


# model
HOGCV = cv2.HOGDescriptor()
HOGCV.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())


def detect(frame):
    bounding_box_cordinates, weights = HOGCV.detectMultiScale(
        frame, winStride=(4, 4), padding=(8, 8), scale=1.03)

    person = 1
    for x, y, w, h in bounding_box_cordinates:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
```

```
            cv2.putText(frame, f'person {person}', (x, y),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1)
            person += 1

        cv2.putText(frame, 'Status : Detecting ', (40, 40),
                    cv2.FONT_HERSHEY_DUPLEX, 0.8, (255, 0, 0), 2)
        cv2.putText(frame, f'Total Persons : {person-1}',
                    (40, 70), cv2.FONT_HERSHEY_DUPLEX, 0.8, (255, 0, 0), 2)
        cv2.imshow('output', frame)

        return frame



# to detect human

def humanDetector(args):
    image_path = args["image"]
    video_path = args['video']
    if str(args["camera"]) == 'true':
        camera = True
    else:
        camera = False

    writer = None
    if args['output'] is not None and image_path is None:
        writer = cv2.VideoWriter(
            args['output'], cv2.VideoWriter_fourcc(*'MJPG'), 10, (600, 600))


    print('[INFO] Opening Web Cam.')
    detectByCamera('outputs-cv/feed.mp4', writer)



def detectByCamera(path, writer):
    video = cv2.VideoCapture(0)
    print('Detecting people...')

    while True:
        check, frame = video.read()

        frame = detect(frame)
        if writer is not None:
            writer.write(frame)
```

```
        key = cv2.waitKey(1)
        if key == ord('q'):
            break

    video.release()
    cv2.destroyAllWindows()




def argsParser():
    arg_parse = argparse.ArgumentParser()
    arg_parse.add_argument("-v", "--video", default=None, help="path to Video File ")
    arg_parse.add_argument("-i", "--image", default=None, help="path to Image File ")
    arg_parse.add_argument("-c", "--camera", default=False, help="Set true if you want to use the camera.")
    arg_parse.add_argument("-o", "--output", type=str, help="path to optional output video file")
    args = vars(arg_parse.parse_args())

    return args


if __name__ == "__main__":
    HOGCV = cv2.HOGDescriptor()
    HOGCV.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())

    args = argsParser()
    humanDetector(args)
```

PersonCounter.py

```
from imutils.video import VideoStream
from imutils.video import FPS
import argparse
import imutils
```

```
import time
import cv2
from datetime import datetime, time
import numpy as np
import time as time2




ap = argparse.ArgumentParser()
ap.add_argument("-v", "--video", help="path to the video file")
ap.add_argument("-a", "--min-area", type=int, default=500, help="minimum area
size")
ap.add_argument("-t", "--tracker", type=str, default="csrt", help="OpenCV object
tracker type")
args = vars(ap.parse_args())



# extract the OpenCV version info
(major, minor) = cv2.__version__.split(".")[:2]
# if we are using OpenCV 3.2 or an earlier version, we can use a special factory
# function to create the entity that tracks objects
if int(major) == 3 and int(minor) < 3:
    tracker = cv2.Tracker_create(args["tracker"].upper())
    #tracker = cv2.TrackerGOTURN_create()
# otherwise, for OpenCV 3.3 or newer,
# we need to explicity call the respective constructor that contains the tracker
object:
else:
    # initialize a dictionary that maps strings to their corresponding
    # OpenCV object tracker implementations
    OPENCV_OBJECT_TRACKERS = {
        "csrt": cv2.TrackerCSRT_create,
        "kcf": cv2.TrackerKCF_create,
        "boosting": cv2.legacy.TrackerBoosting_create,
        "mil": cv2.TrackerMIL_create,
        "tld": cv2.legacy.TrackerTLD_create,
        "medianflow": cv2.legacy.TrackerMedianFlow_create,
        "mosse": cv2.legacy.TrackerMOSSE_create
    }
# grab the appropriate object tracker using our dictionary of
# OpenCV object tracker objects
    tracker = OPENCV_OBJECT_TRACKERS[args["tracker"]]()
```

```
    #tracker = cv2.TrackerGOTURN_create()
# if the video argument is None, then the code will read from webcam (work in
progress)
if args.get("video", None) is None:
    vs = VideoStream(src=0).start()
    time2.sleep(2.0)
# otherwise, we are reading from a video file
else:
    vs = cv2.VideoCapture(args["video"])




# loop over the frames of the video, and store corresponding information from
each frame
firstFrame = None
initBB2 = None
fps = None
differ = None
now = ''
framecounter = 0
trackeron = 0

while True:
    frame = vs.read()
    frame = frame if args.get("video", None) is None else frame[1]
    # if the frame can not be grabbed, then we have reached the end of the video
    if frame is None:
            break

    # resize the frame to 500
    frame = imutils.resize(frame, width=500)

    framecounter = framecounter+1
    if framecounter > 1:

        (H, W) = frame.shape[:2]
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        gray = cv2.GaussianBlur(gray, (21, 21), 0)

        # if the first frame is None, initialize it
        if firstFrame is None:
```

```
            firstFrame = gray
            continue

        # compute the absolute difference between the current frame and first
frame
        frameDelta = cv2.absdiff(firstFrame, gray)
        thresh = cv2.threshold(frameDelta, 25, 255, cv2.THRESH_BINARY)[1]

        # dilate the thresholded image to fill in holes, then find contours on
thresholded image
        thresh = cv2.dilate(thresh, None, iterations=2)
        # cnts = cv2.findContours(thresh.copy(),
cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
        # cnts = cnts[0] if imutils.is_cv2() else cnts[1]
        contours, heirarchy = cv2.findContours(thresh.copy(),
cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
        cnts = max(contours, key=cv2.contourArea)


        # loop over the contours identified
        contourcount = 0
        for c in cnts:
            contourcount =   contourcount + 1

           # if the contour is too small, ignore it
            if cv2.contourArea(c) < args["min_area"]:
                continue

            # compute the bounding box for the contour, draw it on the frame,
            (x, y, w, h) = cv2.boundingRect(c)
            initBB2 =(x,y,w,h)


            prott1 = r'MobileNetSSD_deploy.prototxt'
            prott2 = r'mobilenet_iter_73000.caffemodel'
            net = cv2.dnn.readNetFromCaffe(prott1, prott2)

            CLASSES = ["person"]
            COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))

            trackbox = frame[y:y+h, x:x+w]
            trackbox = cv2.resize(trackbox, (224, 224))
            cv2.imshow('image',trackbox)
            blob = cv2.dnn.blobFromImage(cv2.resize(trackbox, (300,
300)),0.007843, (300, 300), 127.5)
```

```
            net.setInput(blob)
            detections = net.forward()

            for i in np.arange(0, detections.shape[2]):
                confidence = detections[0, 0, i, 2]

                confidence_level = 0.7

                if confidence > confidence_level:
                    # extract the index of the class label from the `detections`, then
compute the (x, y)-coordinates of
                    # the bounding box for the object
                    idx = int(detections[0, 0, i, 1])
                    box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
                    (startX, startY, endX, endY) = box.astype("int")
                    # draw the prediction on the frame
                    label = "{}: {:.2f}%".format(CLASSES[idx],
                                                 confidence * 100)
                    cv2.rectangle(frame, (startX, startY), (endX, endY),
                                  COLORS[idx], 2)
                    y = startY - 15 if startY - 15 > 15 else startY + 15
                    cv2.putText(frame, label, (startX, y),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5,
COLORS[idx], 2)

            cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 255, 0), 2)
            # Start tracker
            now = datetime.now()
            if differ == None or differ > 9:
                tracker.init(frame, initBB2)
                fps = FPS().start()

    # check to see if we are currently tracking an object, if so, ignore other boxes
    # this code is relevant if we want to identify particular persons (section 2 of
this tutorial)
    if initBB2 is not None:

        # grab the new bounding box coordinates of the object
        (success, box) = tracker.update(frame)

        # check to see if the tracking was a success
        differ = 10
        if success:
            (x, y, w, h) = [int(v) for v in box]
```

```
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
            differ = abs(initBB2[0]-box[0]) + abs(initBB2[1]-box[1])
            i = tracker.update(lastframe)
            if i[0] != True:
                time2.sleep(4000)
        else:
            trackeron = 1

        # update the FPS counter
        fps.update()
        fps.stop()

        # initialize the set of information we'll be displaying on
        # the frame
        info = [
            ("Success", "Yes" if success else "No"),
            ("FPS", "{:.2f}".format(fps.fps())),
        ]

        # loop over the info tuples and draw them on our frame
        for (i, (k, v)) in enumerate(info):
            text = "{}: {}".format(k, v)
            cv2.putText(frame, text, (10, H - ((i * 20) + 20)),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)

        # draw the text and timestamp on the frame
        now2 = datetime.now()
        time_passed_seconds = str((now2-now).seconds)
        cv2.putText(frame, 'Detecting persons', (10, 20),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)

    # show the frame and record if the user presses a key
    cv2.imshow("Video stream", frame)
    key = cv2.waitKey(1) & 0xFF

    # if the `q` key is pressed, break from the lop
    if key == ord("q"):
        break
    if key == ord("d"):
        firstFrame = None
    lastframe = frame

# finally, stop the camera/stream and close any open windows
vs.stop() if args.get("video", None) is None else vs.release()
```

```
cv2.destroyAllWindows()
```

Twilio_api.py

```
from twilio.rest import Client
# Lucky@1234567891011
# deepankarsharma2003@gmail.com
# Your Account SID from twilio.com/console
# account_sid = "AC0436791453c88f23bb818240cbd471a2"
# Your Auth Token from twilio.com/console
# auth_token   = "33dfa5eaaf0c90be139db142df619323"

# Read text from the credentials file and store in data variable
with open('credentials.txt', 'r') as myfile:
  data = myfile.read()

# Convert data variable into dictionary
info_dict = eval(data)


account_sid = 'AC0436791453c88f23bb818240cbd471a2'
# auth_token = '[Redacted]'

# Your Auth Token from twilio.com/console
auth_token = info_dict['auth_token']
client = Client(account_sid, auth_token)

message = client.messages.create(
    from_='whatsapp:+14155238886',
    body='Bade achhe lagte hai\n ye dharti\nye nadiya\nye raina\naur tum...',
    to='whatsapp:+919639102301'
)

print(message.sid)
```

BackgroundRemoval.py

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

```
# cap = cv2.VideoCapture('sample_video.mp4')
cap = cv2.VideoCapture(0)
# cap = cv2.VideoCapture('https://10.137.131.218:8080/video')
#
#
# Create the background subtractor object
foog = cv2.createBackgroundSubtractorMOG2(
    detectShadows=False, varThreshold=40, history=150)


# history ----> jitni jyada history , utna slow motions k liye robust hoga

while(1):

    ret, frame = cap.read()
    if not ret:
        break

    # Apply the background object on each frame
    fgmask = foog.apply(frame)

    # Get rid of the shadows
    ret, fgmask = cv2.threshold(fgmask, 250, 255, cv2.THRESH_BINARY)

    # Show the background subtraction frame.
    # cv2.imshow('All three', fgmask) # original

    fgmask = cv2.cvtColor(fgmask, cv2.COLOR_GRAY2BGR)
    cv2.imshow('Stacked frame', np.hstack((frame, fgmask)))
    k = cv2.waitKey(10)
    if k == ord('q'):
        break

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

CompleteSystem.py

```
# Import the required libraries
```

```
import numpy as np
import cv2
import time
import datetime
from collections import deque
from twilio.rest import Client



def is_person_present(frame, thresh=1100):
    global foog
    # Apply background subtraction
    fgmask = foog.apply(frame)
    # Get rid of the shadows
    ret, fgmask = cv2.threshold(fgmask, 250, 255, cv2.THRESH_BINARY)
    # Apply some morphological operations to make sure you have a good mask
    fgmask = cv2.dilate(fgmask, kernel=None, iterations=4)
    # Detect contours in the frame
    contours, hierarchy = cv2.findContours(
        fgmask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    # Check if there was a contour and the area is somewhat higher than some
threshold so we know its a person and not noise
    if contours and cv2.contourArea(max(contours, key=cv2.contourArea)) >
thresh:
        # Get the max contour
        cnt = max(contours, key=cv2.contourArea)
        # Draw a bounding box around the person and label it as person detected
        # x, y, w, h = cv2.boundingRect(cnt)
        # cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 0, 255), 2)
        # cv2.putText(frame, 'Person Detected', (x, y-10),
        #                 cv2.FONT_HERSHEY_SIMPLEX, 0.3, (0, 255, 0), 1,
cv2.LINE_AA)
        return True, frame
    # Otherwise report there was no one present
    else:
        return False, frame

def send_message(body, info_dict):

    # Your Account SID from twilio.com/console
    account_sid = 'AC0436791453c88f23bb818240cbd471a2'

    # Your Auth Token from twilio.com/console
    auth_token = info_dict['auth_token']
```

```
    client = Client(account_sid, auth_token)
    message = client.messages.create(
        from_='whatsapp:+14155238886',
        body='Alert, Ghar m chor hai !!!!!',
        to='whatsapp:+919639102301'
    )
    print(message)




#time.sleep(15)

# Set Window normal so we can resize it
cv2.namedWindow('frame', cv2.WINDOW_AUTOSIZE)

# This is a test video
# cap = cv2.VideoCapture('sample_video.mp4')
ip1 = input('Enter the ip of the cam1: ')
ip1 = 'https://' + ip1 + ':8080/video'
print('ip1: ', ip1)
ip2 = input('Enter the ip of the cam2: ')
ip2 = 'https://' + ip2 + ':8080/video'
print('ip2: ', ip2)


cap = cv2.VideoCapture(ip1)
cap2 = cv2.VideoCapture(ip2)

# Read the video steram from the camera
#cap = cv2.VideoCapture('http://192.168.18.4:8080/video')

# Get width and height of the frame
# width = int(cap.get(3))
# height = int(cap.get(4))
width = 720
height = 480

# Read and store the credentials information in a dict
with open('credentials.txt', 'r') as myfile:
  data = myfile.read()

info_dict = eval(data)
```

```
# Initialize the background Subtractor
foog = cv2.createBackgroundSubtractorMOG2(
    detectShadows=True, varThreshold=100, history=2000)

# Status is True when person is present and False when the person is not present.
status = False
status2 = False

# After the person disapears from view, wait atleast 7 seconds before making the
status False
patience = 7
patience2 = 7

# We don't consider an initial detection unless its detected 15 times, this gets rid
of false positives
detection_thresh = 15

# Initial time for calculating if patience time is up
initial_time = None
initial_time2 = None

# We are creating a deque object of length detection_thresh and will store
individual detection statuses here
de = deque([False] * detection_thresh, maxlen=detection_thresh)
de2 = deque([False] * detection_thresh, maxlen=detection_thresh)

# Initialize these variables for calculating FPS
fps = 0
fps2 = 0
frame_counter = 0
frame_counter2 = 0
start_time = time.time()
start_time2 = time.time()


while(True):

    ret, frame = cap.read()
    ret2, frame2 = cap2.read()
    if not ret or not ret2:
        break
```

```
    # This function will return a boolean variable telling if someone was present
or not, it will also draw boxes if it
    # finds someone
    dim = (width, height)
    frame = cv2.resize(frame, dim, interpolation=cv2.INTER_AREA)
    frame2 = cv2.resize(frame2, dim, interpolation=cv2.INTER_AREA)
    # frame= np.hstack((frame, frame2))
    detected, annotated_image = is_person_present(frame)
    detected2, annotated_image2 = is_person_present(frame2)

    # Register the current detection status on our deque object
    de.appendleft(detected)
    de2.appendleft(detected2)

    # If we have consectutively detected a person 15 times then we are sure that
soemone is present
    # We also make this is the first time that this person has been detected so we
only initialize the videowriter once
    if sum(de) == detection_thresh and not status:
        status = True
        entry_time = datetime.datetime.now().strftime("%A, %I-%M-
%S %p %d %B %Y")
        # out = cv2.VideoWriter('outputs/{}.mp4'.format(entry_time),
        #                       cv2.VideoWriter_fourcc(*'XVID'), 15.0, (width,
height))


    if sum(de2) == detection_thresh and not status2:
        status2 = True
        entry_time2 = datetime.datetime.now().strftime("%A, %I-%M-
%S %p %d %B %Y")
        # out = cv2.VideoWriter('outputs/{}.mp4'.format(entry_time),
        #                       cv2.VideoWriter_fourcc(*'XVID'), 15.0, (width,
height))




    # If status is True but the person is not in the current frame
    if status and not detected:

        # Restart the patience timer only if the person has not been detected for a
few frames so we are sure it was'nt a
        # False positive
```

```
        if sum(de) > (detection_thresh/2):

            if initial_time is None:
                initial_time = time.time()

        elif initial_time is not None:

            # If the patience has run out and the person is still not detected then set
the status to False
            # Also save the video by releasing the video writer and send a text
message.
            if time.time() - initial_time >= patience:
                status = False
                exit_time =
datetime.datetime.now().strftime("%A, %I:%M:%S %p %d %B %Y")
                # out.release()
                initial_time = None

                body = "Alert: n A Person Entered the Room at {} n Left the room
at {}".format(
                    entry_time, exit_time)
                print(body)
                send_message(body, info_dict)

    # If significant amount of detections (more than half of detection_thresh) has
occured then we reset the Initial Time.
    elif status and sum(de) > (detection_thresh/2):
        initial_time = None

    # Get the current time in the required format
    current_time =
datetime.datetime.now().strftime("%A, %I:%M:%S %p %d %B %Y")

    # Display the FPS
    cv2.putText(annotated_image, 'FPS: {:.2f}'.format(
        fps), (510, 450), cv2.FONT_HERSHEY_COMPLEX, 0.6, (255, 40, 155),
2)

    # Display Time
    cv2.putText(annotated_image, current_time, (310, 20),
                cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 0, 255), 1)

    # Display the Room Status
```

```
    cv2.putText(annotated_image, 'Room Occupied: {}'.format(str(status)), (10,
20), cv2.FONT_HERSHEY_SIMPLEX, 0.6,
                (200, 10, 150), 2)

    # Show the patience Value
    if initial_time is None:
        text = 'Patience: {}'.format(patience)
    else:
        text = 'Patience: {:.2f}'.format(
            max(0, patience - (time.time() - initial_time)))

    cv2.putText(annotated_image, text, (10, 450),
                cv2.FONT_HERSHEY_COMPLEX, 0.6, (255, 40, 155), 2)

    # If status is true save the frame
    # if status:
    #       out.write(annotated_image)
    # If status is True but the person is not in the current frame
    if status2 and not detected2:

        # Restart the patience timer only if the person has not been detected for a
few frames so we are sure it was'nt a
        # False positive
        if sum(de2) > (detection_thresh/2):

            if initial_time2 is None:
                initial_time2 = time.time()

        elif initial_time2 is not None:

            # If the patience has run out and the person is still not detected then set
the status to False
            # Also save the video by releasing the video writer and send a text
message.
            if time.time() - initial_time2 >= patience2:
                status2 = False
                exit_time2 =
datetime.datetime.now().strftime("%A, %I:%M:%S %p %d %B %Y")
                # out.release()
                initial_time2 = None

                body2 = "Alert: n A Person Entered the Room at {} n Left the
room2 at {}".format(
                    entry_time2, exit_time2)
```

```
                print(body2)
                send_message(body2, info_dict)

    # If significant amount of detections (more than half of detection_thresh) has
occured then we reset the Initial Time.
    elif status2 and sum(de2) > (detection_thresh/2):
        initial_time2 = None

    # Get the current time in the required format
    current_time2 =
datetime.datetime.now().strftime("%A, %I:%M:%S %p %d %B %Y")

    # Display the FPS
    cv2.putText(annotated_image2, 'FPS: {:.2f}'.format(
        fps2), (510, 450), cv2.FONT_HERSHEY_COMPLEX, 0.6, (255, 40, 155),
2)

    # Display Time
    cv2.putText(annotated_image2, current_time2, (310, 20),
                cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 0, 255), 1)

    # Display the Room Status
    cv2.putText(annotated_image2, 'Room Occupied: {}'.format(str(status2)), (10,
20), cv2.FONT_HERSHEY_SIMPLEX, 0.6,
                (200, 10, 150), 2)

    # Show the patience Value
    if initial_time2 is None:
        text = 'Patience: {}'.format(patience2)
    else:
        text = 'Patience: {:.2f}'.format(
            max(0, patience2 - (time.time() - initial_time2)))

    cv2.putText(annotated_image2, text, (10, 450),
                cv2.FONT_HERSHEY_COMPLEX, 0.6, (255, 40, 155), 2)

    # If status is true save the frame
    # if status:
    #       out.write(annotated_image)

    frame= np.hstack((frame, frame2))
    # Show the Frame
    cv2.imshow('frame', frame)
```

```
        # Calculate the Average FPS
        frame_counter += 1
        fps = (frame_counter / (time.time() - start_time))

        frame_counter2 += 1
        fps2 = (frame_counter2 / (time.time() - start_time2))

        # Exit if q is pressed.
        if cv2.waitKey(30) == ord('q'):
            break

# Release Capture and destroy windows
cap.release()
cap2.release()

cv2.destroyAllWindows()
# out.release()
```

Camfeed-with-ROI cropping

```
# Import the required libraries
import numpy as np
import cv2
import time
import datetime
from collections import deque

# Set Window normal so we can resize it
# cv2.namedWindow('frame', cv2.WINDOW_NORMAL)

# Note the starting time
start_time = time.time()

# Initialize these variables for calculating FPS
fps = 0
frame_counter = 0

classes = None
with open('coco.names', 'r') as f:
    classes = [line.strip() for line in f.readlines()]

net = cv2.dnn.readNet('yolov3-tiny.weights', 'yolov3-tiny.cfg')
```

```
layer_names = net.getLayerNames()

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



ip= input('Enter the ip of the cam: ')
ip = 'https://' + ip + ':8080/video'
print(ip)

# Read the video stream from the camera
# cap = cv2.VideoCapture('http://192.168.46.101:8080/video')
# cap = cv2.VideoCapture('https://192.168.205.234:8080/video')
cap = cv2.VideoCapture(ip)

# cap = cv2.VideoCapture('https://10.133.173.57:8080/video')
# cap = cv2.VideoCapture(0)

skip= 1
while(True):

    if skip==1:
        skip=2
    elif skip == 2:
        skip = 3
        continue
    elif skip==3:
        skip=4
        continue
    elif skip==4:
        skip=5
        continue
    elif skip==5:
        skip=6
        continue
    elif skip==6:
        skip=7
        continue
    else:
        skip=1
        continue

    ret, frame = cap.read()
```

```
    if not ret:
        break

    # dim = (1024, 720)
    dim = (720, 480)
    frame = cv2.resize(frame, dim, interpolation=cv2.INTER_AREA)

    # vertices = np.array(
    #       [[(0, 0), (0, 200), (200, 200), (200, 0)]], dtype=np.int32)
    # vertices = np.array(
    #       [[(50, 50), (50, 50+300), (50+300, 50+300), (50+300, 50)]],
dtype=np.int32)
    vertices = np.array(
        [[(250, 50), (250, 50+300), (250+300, 50+300), (250+300, 50)]],
dtype=np.int32)
    mask = np.zeros_like(frame)

    # # cv2.fillPoly(mask, vertices, (255, 255, 255))
    cv2.fillPoly(mask, vertices, (255, 255, 255))   # BGR
    # cv2.imshow('mask', mask)
    masked_frame= frame.copy()
    frame = cv2.bitwise_and(frame, mask)
    # frame = cv2.resize(frame, dim, interpolation=cv2.INTER_AREA)
    frame= cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    frame= cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    # frame= cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # frame= cv2.cvtColor(frame, cv2.COLOR_GRAY2RGB)

    # Calculate the Average FPS
    frame_counter += 1
    fps = (frame_counter / (time.time() - start_time))

    # Display the FPS
    cv2.putText(frame, 'FPS: {:.2f}'.format(
        fps), (20, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 1)


    image= frame

    net.setInput(cv2.dnn.blobFromImage(image, 0.00392,
                 (416, 416), (0, 0, 0), True, crop=False))
    outs = net.forward(output_layers)
```

```
    class_ids = []
    confidences = []
    boxes = []
    Width = image.shape[1]
    Height = image.shape[0]
    for out in outs:
        for detection in out:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            # if confidence > 0.1:
            if confidence > 0.15:
                center_x = int(detection[0] * Width)
                center_y = int(detection[1] * Height)
                w = int(detection[2] * Width)
                h = int(detection[3] * Height)
                x = center_x - w / 2
                y = center_y - h / 2
                class_ids.append(class_id)
                confidences.append(float(confidence))
                boxes.append([x, y, w, h])

    indices = cv2.dnn.NMSBoxes(boxes, confidences, 0.1, 0.1)
    #check if is people detection
    count= 0
    for i in indices:
        # i = i[0]
        box = boxes[i]
        # if class_ids[i] == 0 or class_ids[i]==56:
        if class_ids[i] == 0:
            count+=1
            # label = str(classes[class_id])
            label = str(classes[class_ids[i]])

            cv2.rectangle(image, (round(box[0]), round(box[1])), (round(
                box[0]+box[2]), round(box[1]+box[3])), (200, 10, 10), 5)
            cv2.putText(image, label, (round(
                box[0])-10, round(box[1])-10), cv2.FONT_HERSHEY_SIMPLEX,
0.9, (200, 10, 150), 2)

    print(f'{count} people detected !!!')

    # Show the Frame
    cv2.imshow('frame', image)
```

```
    # Exit if q is pressed.
    if cv2.waitKey(1) == ord('q'):
        break

# Release Capture and destroy windows
cap.release()
cv2.destroyAllWindows()
```

DeepSort Tracking:

```
from deep_sort import generate_detections as gdet
from deep_sort.tracker import Tracker
from deep_sort.detection import Detection
from deep_sort import nn_matching
import time
from yolov3.configs import *
from yolov3.utils import Load_Yolo_model, image_preprocess,
postprocess_boxes, nms, draw_bbox, read_class_names
import tensorflow as tf
import numpy as np
import cv2
import os
os.environ['CUDA_VISIBLE_DEVICES'] = '0'



def Object_tracking(Yolo, input_size=416, show=False,
CLASSES=YOLO_COCO_CLASSES, score_threshold=0.3, iou_threshold=0.45,
rectangle_colors='', Track_only=[]):
    # Definition of the parameters
    max_cosine_distance = 0.7
    nn_budget = None

    #initialize deep sort object
    model_filename = 'model_data/mars-small128.pb'
    encoder = gdet.create_box_encoder(model_filename, batch_size=1)
    metric = nn_matching.NearestNeighborDistanceMetric(
        "cosine", max_cosine_distance, nn_budget)
    tracker = Tracker(metric)

    times, times_2 = [], []
```

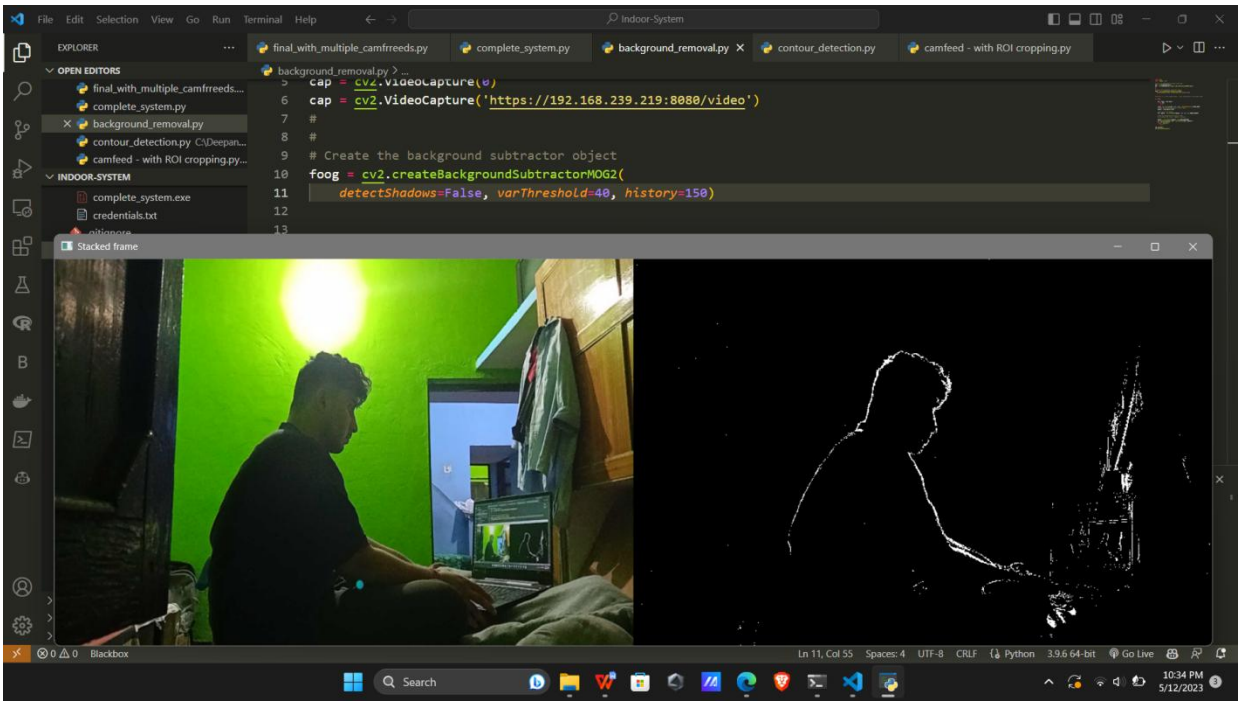
```
    # vid = cv2.VideoCapture(0)   # detect from webcam
    vid = cv2.VideoCapture('https://192.168.253.163:8080/video') # detect from
mobile feed

    # by default VideoCapture returns float instead of int
    width = int(vid.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(vid.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fps = int(vid.get(cv2.CAP_PROP_FPS))
    # codec = cv2.VideoWriter_fourcc(*'XVID')
    # # output_path must be .mp4
    # out = cv2.VideoWriter(output_path, codec, fps, (width, height))

    NUM_CLASS = read_class_names(CLASSES)
    key_list = list(NUM_CLASS.keys())
    val_list = list(NUM_CLASS.values())
    while True:
        _, frame = vid.read()

        try:
            original_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            original_frame = cv2.cvtColor(original_frame,
cv2.COLOR_BGR2RGB)
        except:
            break

        image_data = image_preprocess(np.copy(original_frame), [
                                          input_size, input_size])
        #image_data = tf.expand_dims(image_data, 0)
        image_data = image_data[np.newaxis, ...].astype(np.float32)

        t1 = time.time()
        if YOLO_FRAMEWORK == "tf":
            pred_bbox = Yolo.predict(image_data)
        elif YOLO_FRAMEWORK == "trt":
            batched_input = tf.constant(image_data)
            result = Yolo(batched_input)
            pred_bbox = []
            for key, value in result.items():
                value = value.numpy()
                pred_bbox.append(value)

        #t1 = time.time()
        #pred_bbox = Yolo.predict(image_data)
```

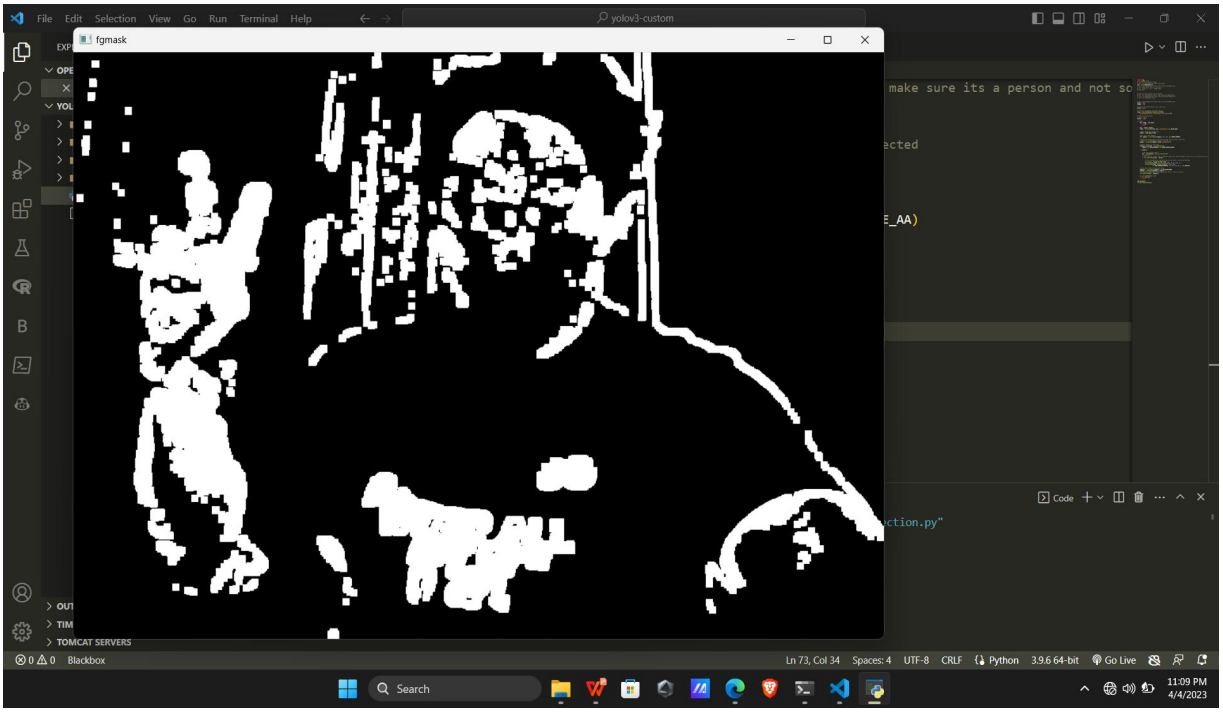
```
        t2 = time.time()

        pred_bbox = [tf.reshape(x, (-1, tf.shape(x)[-1])) for x in pred_bbox]
        pred_bbox = tf.concat(pred_bbox, axis=0)

        bboxes = postprocess_boxes(
            pred_bbox, original_frame, input_size, score_threshold)
        bboxes = nms(bboxes, iou_threshold, method='nms')

        # extract bboxes to boxes (x, y, width, height), scores and names
        boxes, scores, names = [], [], []
        for bbox in bboxes:
            if len(Track_only) != 0 and NUM_CLASS[int(bbox[5])] in Track_only
or len(Track_only) == 0:
                boxes.append([bbox[0].astype(int), bbox[1].astype(int),
bbox[2].astype(
                    int)-bbox[0].astype(int), bbox[3].astype(int)-
bbox[1].astype(int)])
                scores.append(bbox[4])
                names.append(NUM_CLASS[int(bbox[5])])

        # Obtain all the detections for the given frame.
        boxes = np.array(boxes)
        names = np.array(names)
        scores = np.array(scores)
        features = np.array(encoder(original_frame, boxes))
        detections = [Detection(bbox, score, class_name, feature) for bbox,
                      score, class_name, feature in zip(boxes, scores, names,
features)]

        # Pass detections to the deepsort object and obtain the track information.
        tracker.predict()
        tracker.update(detections)

        # Obtain info from the tracks
        tracked_bboxes = []
        for track in tracker.tracks:
            if not track.is_confirmed() or track.time_since_update > 5:
                continue
            bbox = track.to_tlbr()   # Get the corrected/predicted bounding box
            class_name = track.get_class()   # Get the class name of particular
object
            tracking_id = track.track_id   # Get the ID for the particular track
            # Get predicted object index by object name
```

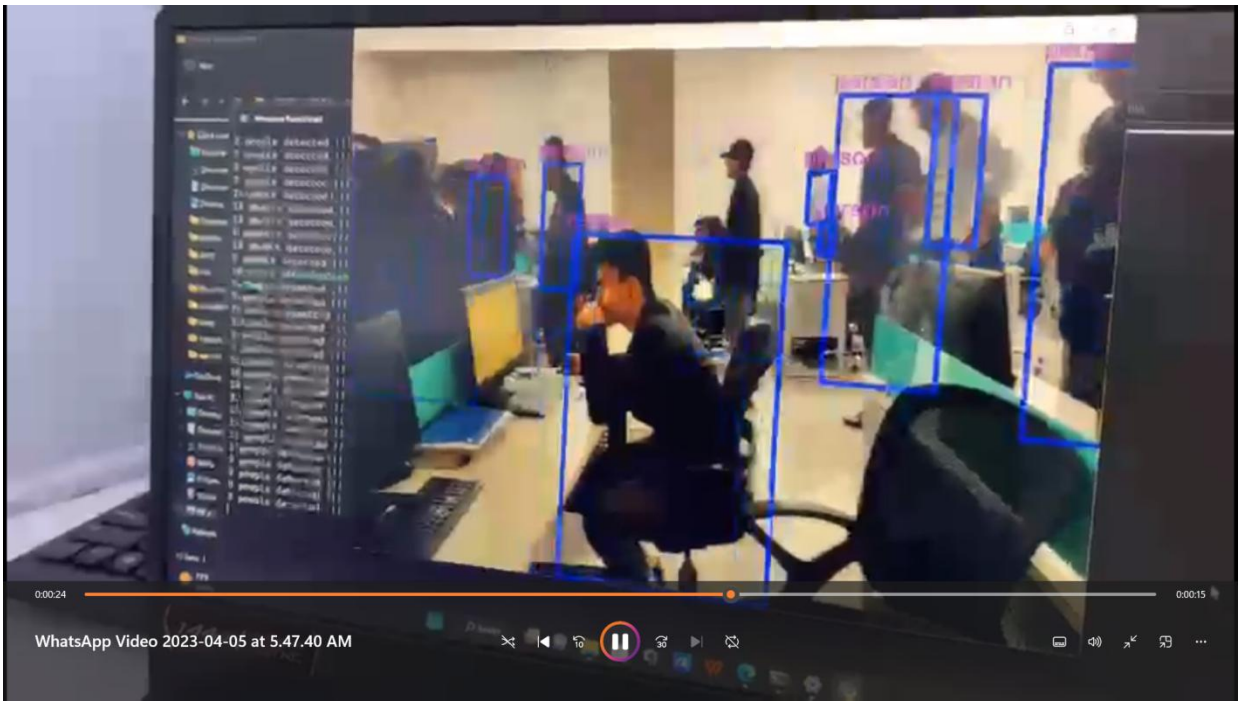
```
                index = key_list[val_list.index(class_name)]
                # Structure data, that we could use it with our draw_bbox function
                tracked_bboxes.append(bbox.tolist() + [tracking_id, index])

        # draw detection on frame
        image = draw_bbox(original_frame, tracked_bboxes,
                          CLASSES=CLASSES, tracking=True)

        t3 = time.time()
        times.append(t2-t1)
        times_2.append(t3-t1)

        times = times[-20:]
        times_2 = times_2[-20:]

        ms = sum(times)/len(times)*1000
        fps = 1000 / ms
        fps2 = 1000 / (sum(times_2)/len(times_2)*1000)

        image = cv2.putText(image, "Time: {:.1f} FPS".format(
            fps), (0, 30), cv2.FONT_HERSHEY_COMPLEX_SMALL, 1, (0, 0,
255), 2)

        # draw original yolo detection
        #image = draw_bbox(image, bboxes, CLASSES=CLASSES,
show_label=False, rectangle_colors=rectangle_colors, tracking=True)

        print("Time: {:.2f}ms, Detection FPS: {:.1f}, total FPS: {:.1f}".format(
            ms, fps, fps2))
        # if output_path != '':
        #       out.write(image)
        if show:
            cv2.imshow('output', image)

            if cv2.waitKey(25) & 0xFF == ord("q"):
                cv2.destroyAllWindows()
                break

    cv2.destroyAllWindows()

yolo = Load_Yolo_model()
# Object_tracking(yolo, video_path, "track.mp4",
input_size=YOLO_INPUT_SIZE, show=False, iou_threshold=0.1,
rectangle_colors=(255,0,0), Track_only = ["person"])
```

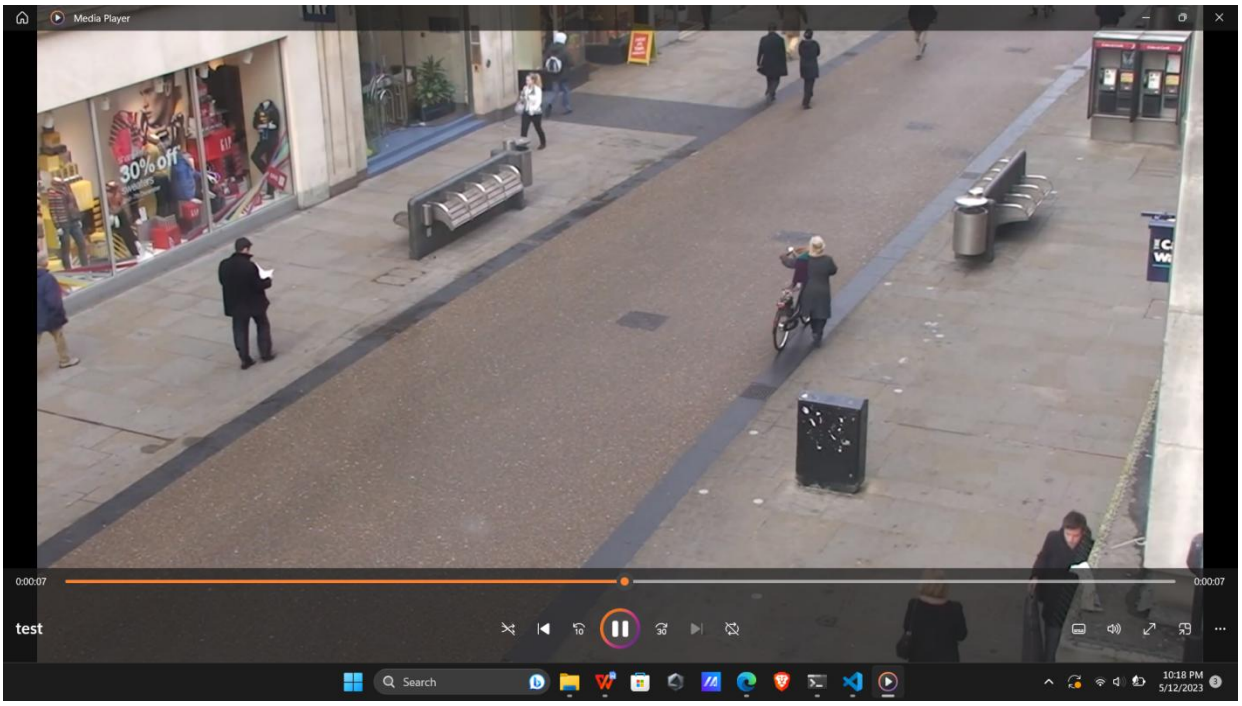
```
Object_tracking(yolo, input_size=YOLO_INPUT_SIZE, show=True,
                iou_threshold=0.1, rectangle_colors=(255, 0, 0),
Track_only=["person"])
```

6.3 Test Data: A set of sample videos and images used to test the system's object detection and tracking capabilities.

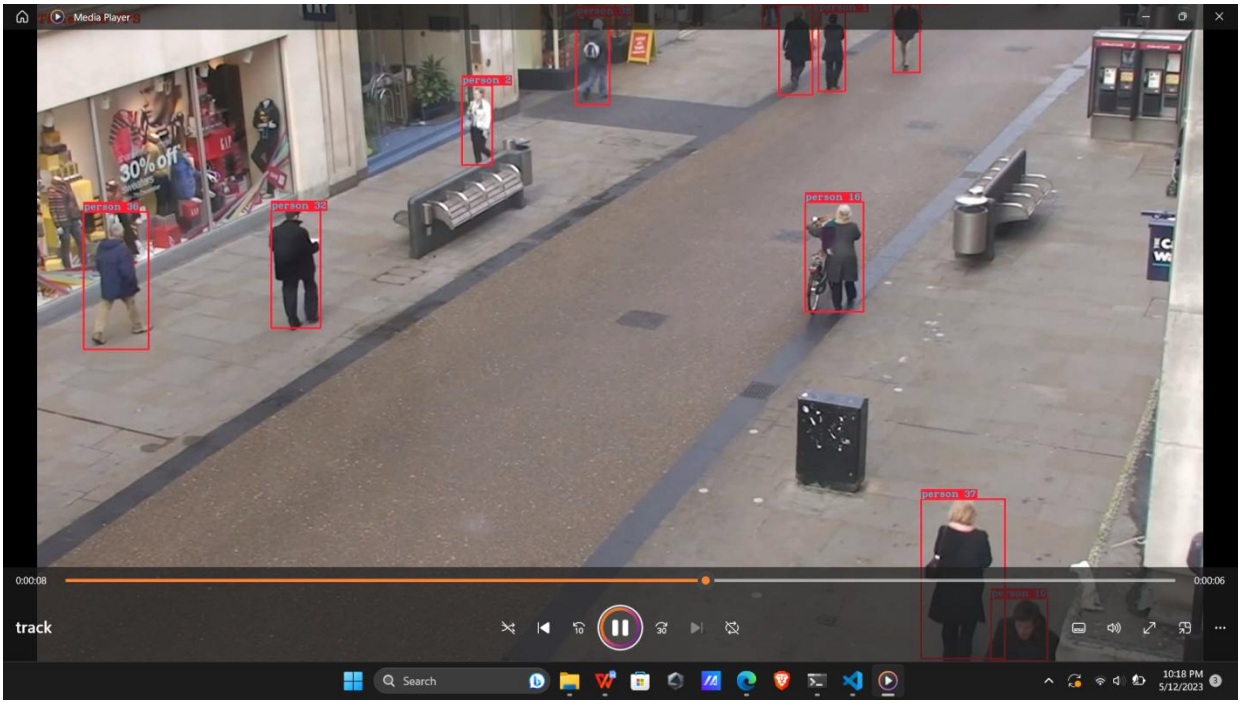


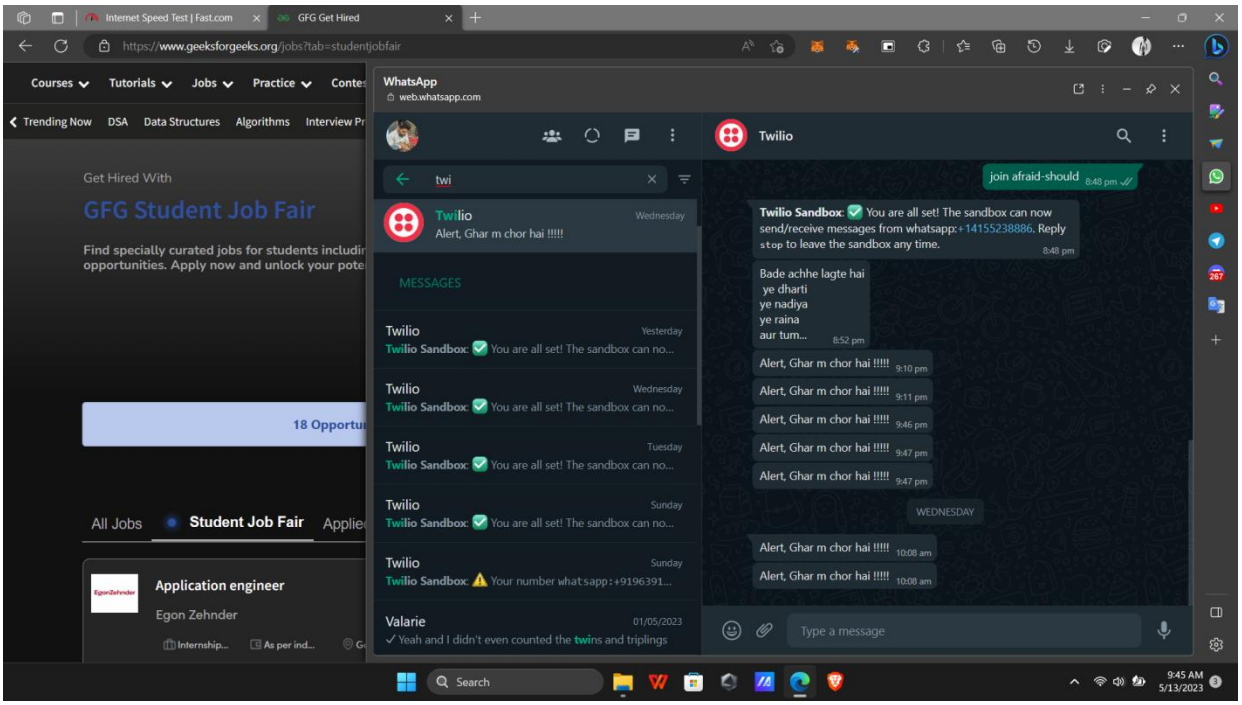


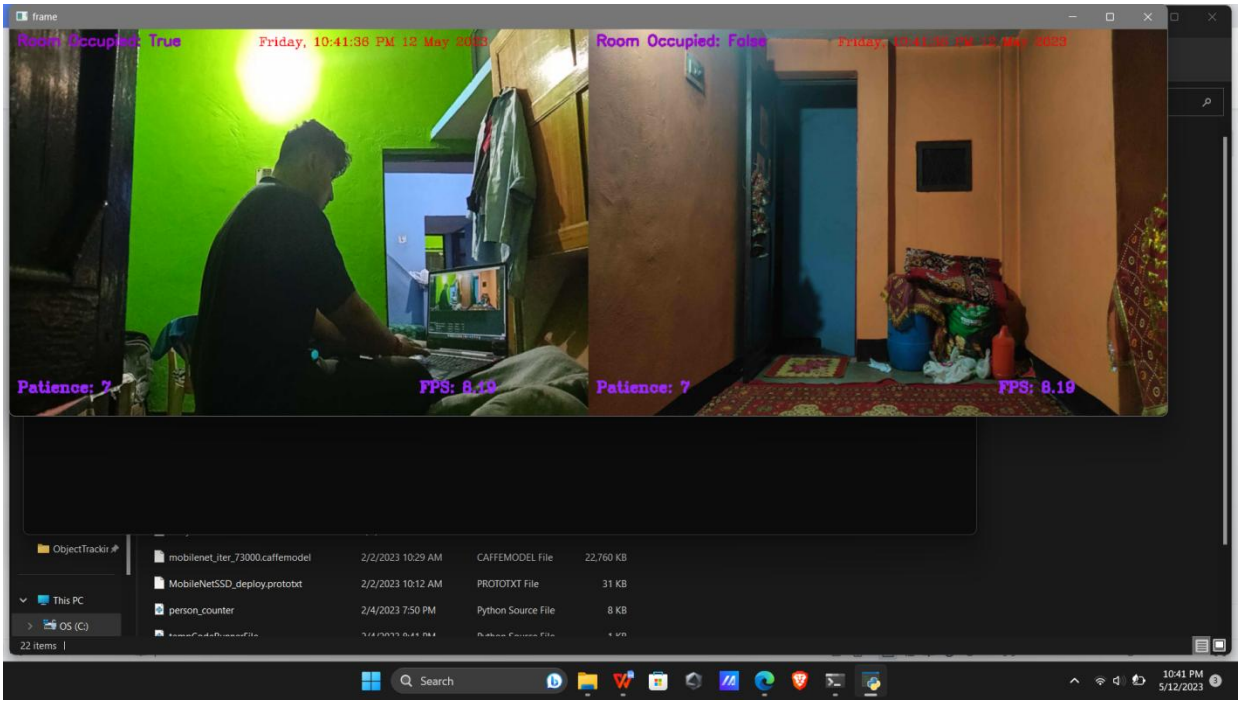

0:00:24
0:00:15
WhatsApp Video 2023-04-05 at 5.47.40 AM

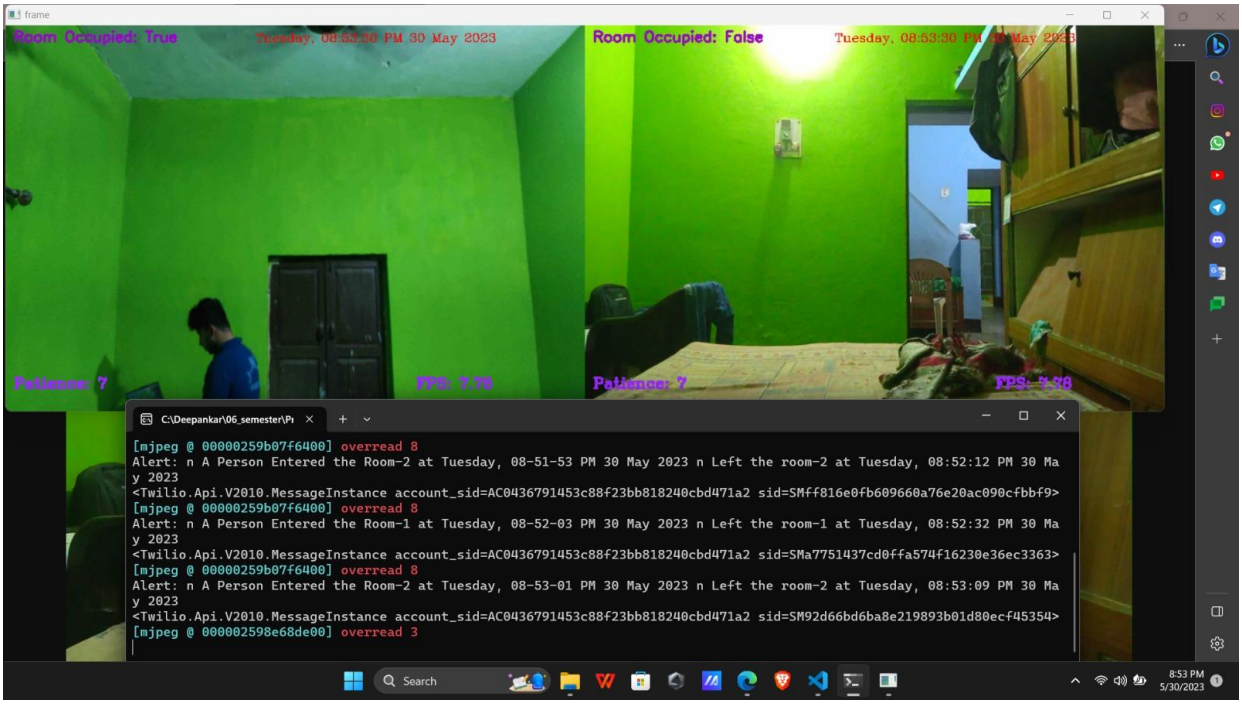

Media Player
30% off
0:00:07
0:00:07
test
Search
10:18 PM
5/12/2023

L


Media Player
person 3
person 36
person 32
person 16
person 37
0:00:08
0:00:06
track
Search
10:18 PM
5/12/2023


Internet Speed Test | Fast.com
GFG Get Hired
https://www.geeksforgeeks.org/jobs?tab=studentjobfair
Courses
Tutorials
Jobs
Practice
Trending Now
DSA
Data Structures
Algorithms
Get Hired With
GFG Student Job Fair
Find specially curated jobs for students includir
opportunities. Apply now and unlock your pote
All Jobs
Student Job Fair
Application engineer
Egon Zehnder
WhatsApp
web.whatsapp.com
twi
Twilio
Alert, Ghar m chor hai !!!!!
MESSAGES
Twilio Sandbox: You are all set! The sandbox can no...
Twilio Sandbox: Your number whatsapp:+9196391...
Valarie
01/05/2023
Yeah and I didn't even counted the twins and triplings
join afraid-should
Twilio Sandbox: You are all set! The sandbox can now send/receive messages from whatsapp:+14155238886. Reply stop to leave the sandbox any time.
Bade achhe lagte hai
ye dharti
ye nadiya
ye raina
aur tum...
WEDNESDAY
Type a message
Search
9:45 AM
5/13/2023





6.4 Risk Assessment: A report on the potential risks and vulnerabilities associated with the Smart Surveillance System, along with mitigation strategies.

- **Data privacy and security:** The system collects and stores sensitive personal information, which could be vulnerable to unauthorized access or breach. To mitigate this risk, the system should incorporate strong encryption and access controls to protect the data, and adhere to relevant data privacy regulations.

- **False positives and false negatives:** The system could generate false positives

(i.e. detecting activity as a security threat when it is not) or false negatives (i.e. failing to detect a real security threat). To mitigate this risk, the system should incorporate robust machine learning algorithms and be continually trained on real-world data to improve accuracy.

- **System downtime:** The system could experience downtime due to hardware failures, network outages, or software bugs. To mitigate this risk, the system should be designed with redundancy and failover mechanisms to ensure continuity of surveillance operations.

- **Integration with existing systems:** The system may face challenges in integrating with existing security systems or infrastructure. To mitigate this risk, the system should be designed with flexibility and modularity, allowing for easy integration and customization.

6.5 Future Work: A section detailing future work and potential enhancements to the system, such as incorporating new algorithms or integrating with other security systems.

- **Integration with other sensors:** The system can be expanded to incorporate other types of sensors, such as audio or environmental sensors, to provide a more comprehensive view of the surveillance environment.

- **Multi-camera tracking:** The system can be enhanced to track individuals across multiple camera feeds, enabling more accurate and comprehensive monitoring of their movements and behavior.

- **Automated threat assessment:** The system can be improved with machine learning algorithms to automatically assess the threat level of an observed activity, enabling more rapid and targeted responses to potential security threats.

- **Facial recognition:** The system can be enhanced with facial recognition technology to identify individuals and track their movements, providing more granular control over access to secure areas.

- **Cloud-based storage and processing:** The system can be adapted to store and process surveillance data in the cloud, enabling more scalable and cost-effective deployments for larger or more complex security environments.

- **Edge computing:** The system can be enhanced with edge computing capabilities, enabling the processing and analysis of surveillance data to be performed closer to the source, reducing latency and improving overall

performance.

- **Improved user interface:** The system can be improved with a more intuitive and user-friendly interface for security personnel, enabling them to more effectively monitor and respond to potential security threats.

6.6 Glossary: A list of technical terms and acronyms used in the project and their definitions.

- **Object Detection:** The process of identifying and locating objects within an image or video feed.
- **Motion Detection:** The process of detecting movement within an image or video feed, typically used to trigger alerts or notifications.
- **YOLOv3:** A deep learning algorithm used for object detection, which stands for "You Only Look Once" version 3.
- **OpenCV:** An open-source computer vision library used for image and video processing.
- **Streamlit:** A Python library used for building interactive web applications for data science and machine learning.
- **Twilio:** A cloud communications platform used for sending SMS messages and making voice calls.
- **TensorFlow:** An open-source machine learning library developed by Google, used for building and training deep learning models.
- **GPU:** A Graphics Processing Unit, a specialized hardware component used for accelerating the processing of large amounts of data, commonly used in machine learning and other computationally intensive tasks.
- **Raspberry Pi:** A small, low-cost computer used for building hardware prototypes and small-scale projects.
- **Real-time Analytics:** The process of analyzing data as it is generated in real-time, typically used for detecting anomalies, predicting future events, or making decisions based on the current situation.

7. Conclusion

The Smart Surveillance System presented in this project makes use of advanced technologies such as YOLOv3, OpenCV, NumPy, and TensorFlow to detect and track objects in real-time. The system provides a reliable and efficient solution for enhancing the security of various premises.

The system's core functionality includes real-time object detection and tracking, motion detection, and generating alerts and notifications. The integration of Twilio APIs enables the system to send SMS and email notifications and even Whats-app alerts to the relevant authorities, improving the system's overall responsiveness and reliability.

Further advancements to the system could include the integration of additional machine learning algorithms, such as deep neural networks, to improve the accuracy and precision of object detection. The system could also incorporate edge computing capabilities to process video feeds on local devices, improving the system's response time and reducing network bandwidth requirements.

Additionally, the system could integrate with existing surveillance infrastructure, such as CCTV cameras and access control systems, to provide a comprehensive security solution. The system could also incorporate advanced analytics and reporting capabilities, providing valuable insights into the security and surveillance of the premises.

Overall, the Smart Surveillance System presented in this project is a highly effective and efficient solution for enhancing the security of various premises. The system's versatility and flexibility make it a valuable asset for organizations seeking to improve their security and surveillance capabilities.

8. Bibliography

- YOLO: Real-Time Object Detection (pjreddie.com)
- Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. arXiv preprint arXiv:1804.02767.
- Bradski, G. (2000). The OpenCV Library. Dr. Dobb's Journal of Software Tools, 25-33.
- Rehman, M. Z., & Saleem, A. (2018). Streamlit: Build custom web applications for machine learning and data science.
- Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... & Zheng, X. (2016). TensorFlow: Large-scale machine learning on heterogeneous systems. arXiv preprint arXiv:1603.04467.
- Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. (2020). Array programming with NumPy. Nature, 585(7825), 357-362.
- Twilio API Documentation. https://www.twilio.com/docs/api. Accessed on 7th April 2023.
- SQLite Documentation. https://www.sqlite.org/docs.html. Accessed on 7th April 2023.
- Matplotlib Documentation. https://matplotlib.org/stable/contents.html. Accessed on 7th April 2023.
- Flask Documentation. https://flask.palletsprojects.com/en/2.1.x/. Accessed on 7th April 2023.
- Docker Documentation. https://docs.docker.com/. Accessed on 7th April 2023.

These references were used as a guide to implement various components of the Smart Surveillance System and provided insights into the best practices for completing the project successfully.