**Intrusion Detection System**

**Using Real-Time Computer Vision**

By:

Balaji Sai Jayanth – 225890282

Agrima Goyal – 225890378

Aarushi Garg - 225890388

**Index**

1. Introduction
2. Objective
3. Technologies Used
4. System Architecture
5. Step 1: Movement Detection
6. Step 2: Object Detection and Recognition (Face Recognition)
7. Step 3: Object Tracking
8. Outputs
9. Future Enhancements
10. Conclusion

**1. Introduction**

This project aims to create a real-time intrusion detection system using computer vision. The system integrates movement detection, face recognition, and object tracking to build an intelligent CCTV-style security solution.

It is capable of distinguishing between authorized employees and unknown individuals and provides alerts based on the detection.

**2. Objective**

To develop an integrated real-time security system that:

* Detects movement in surveillance footage
* Recognizes known employees through face recognition
* Tracks detected individuals in a predefined Region of Interest (ROI)
* Generates a warning when an unknown person enters the secure area

**3. Technologies Used**

|  |  |
| --- | --- |
| **Tool/Library** | **Purpose** |
| Python | Core programming language |
| OpenCV | Image processing and webcam interaction |
| face\_recognition | Facial feature detection and encoding |
| Ultralytics YOLOv8 | Object detection and classification |
| NumPy, Pandas | Numerical and tabular data processing |
| Matplotlib, Seaborn | Data visualization |
| FFmpeg | Video compression and encoding |
| tqdm, os, datetime | Auxiliary functionalities |

**4. System Architecture**

Each module can work independently, but together they create a complete real-time surveillance pipeline.

**5. Step 1: Movement Detection**

**Purpose**: Detect moving objects using background subtraction and contour merging.

**Methodology**:

* Background subtraction with MOG2
* Morphological filtering to reduce noise
* Merging of fragmented contours

**Key Code Snippets**:

fgbg = cv2.createBackgroundSubtractorMOG2(history=500, detectShadows=True)

fgmask = fgbg.apply(frame)

kernel = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE, (5, 5))

fgmask = cv2.morphologyEx(fgmask, cv2.MORPH\_CLOSE, kernel)

fgmask = cv2.morphologyEx(fgmask, cv2.MORPH\_OPEN, kernel)

def merge\_contours(contours, max\_distance=50):

mask = np.zeros((720, 1280), dtype=np.uint8)

for cnt in contours:

cv2.drawContours(mask, [cnt], -1, 255, -1)

kernel = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE, (max\_distance//2, max\_distance//2))

mask = cv2.dilate(mask, kernel, iterations=2)

merged\_contours, \_ = cv2.findContours(mask, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

return merged\_contours

**Output**:

A person with his hand up

AI-generated content may be incorrect.

**Performance**:

* 15-25 FPS on 1280x720 resolution
* <100ms latency

**Limitations**:

* Merges nearby people
* Less effective for fast motion

**6. Step 2: Object Detection & Recognition (Face Recognition)**

**Purpose**: Identify known individuals using facial features and mark attendance or trigger alerts.

**Pipeline**:

* Load training images and extract 128-D encodings
* Match real-time frames with known encodings
* Log attendance in a CSV

**Key Code Snippets**:

import face\_recognition

for file in os.listdir('Images'):

img = cv2.imread(f'Images/{file}')

encodings = face\_recognition.face\_encodings(cv2.cvtColor(img, cv2.COLOR\_BGR2RGB))

known\_faces.append(encodings[0])

known\_names.append(os.path.splitext(file)[0])

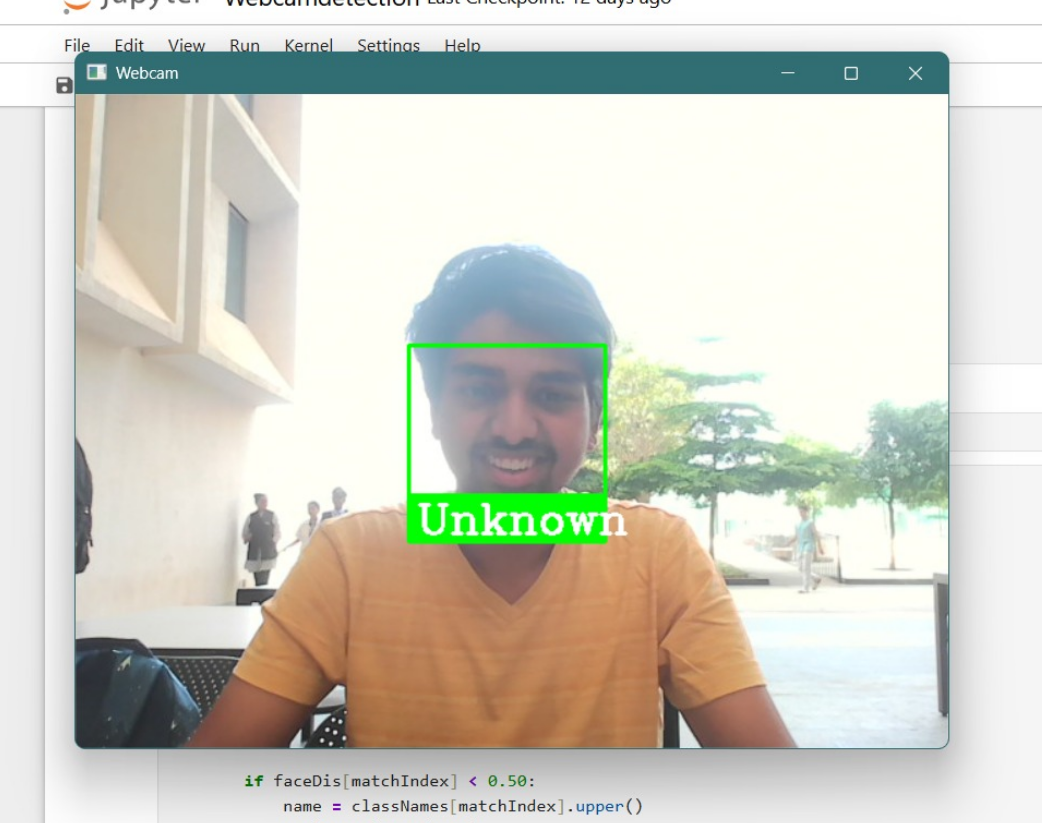
# In live feed

matches = face\_recognition.compare\_faces(known\_faces, current\_encoding)

**Output**:

A screenshot of a computer

AI-generated content may be incorrect.



**Strengths**:

* Contactless, real-time
* Easy integration with CSV-based logs

**Limitations**:

* Spoofing with photos
* Range of the algorithm is less

**7. Step 3: Object Tracking**

**Purpose**: Track moving objects within a Region of Interest (ROI), count intrusions, and visually annotate frames.

**Steps**:

* Load video and YOLOv8 model
* Detect objects within ROI
* Track using bounding boxes and object centers
* Annotate ID and count people

**Key Code Snippets**:

model = YOLO('yolov8n.pt')

y\_hat = model.predict(ROI, conf=0.3, classes=[0], device=0)

for ix, row in positions\_frame.iterrows():

xmin, ymin, xmax, ymax = map(int, row[:4])

center\_x, center\_y = (xmax + xmin) // 2, (ymax + ymin) // 2

id\_obj = assign\_tracking\_id(center\_x, center\_y)

cv2.rectangle(ROI, (xmin, ymin), (xmax, ymax), (0,0,255), 2)

cv2.putText(ROI, f"{id\_obj}", (xmin, ymin - 10), ...)

**Output**:

* Bounding boxes + ID
* People count in ROI

A group of people walking in a large room

AI-generated content may be incorrect.

**Limitations**:

* Fixed ROI only
* Doesn't use predictive filters like Kalman

**8. Final Outputs**

* Live annotated video showing bounding boxes, names/IDs, and people counts
* Logged CSV file for recognized individuals
* Final MP4 video with overlaid alerts and ROI

**9. Future Enhancements**

|  |  |
| --- | --- |
| **Feature** | **Benefit** |
| GUI using Tkinter or PyQt | Easier user interaction |
| Cloud Integration | Sync logs to Google Sheets or Firebase |
| Spoof Detection | Avoid image-based intrusions |
| Advanced Face Models (ArcFace) | Better face matching accuracy |
| Kalman Filter Integration | Smoother object tracking |
| Email/SMS Alerts | Notify admins of unauthorized access |
| Masked Face Detection | Increase reliability in COVID-era setups |

**10. Conclusion**

This project demonstrates a modular yet integrated real-time intrusion detection system using computer vision. The combination of motion detection, face recognition, and YOLO-based tracking forms a complete surveillance pipeline. With future enhancements and improved robustness, it can be deployed in real-world security scenarios for office campuses, labs, and restricted areas.