الجامعة المصرية اليابانية للعلوم والتكنولوجيا
エジプト日本科学技術大学
EGYPT-JAPAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Real-Time Surveillance System with Object Detection and Face Recognition

Team Members:

Sally Reda Zeineldeen	120210008
Nada Ashraf Gomaa	120210358
Menna Alaa Eldin Rasslan	120210311
Sohaila Adel Ahmed	120210337
Zeyad Salah Moussa	120210297
Muhammad Abdullah Ghareeb	120210357

Course: CSE 429 - Computer Vision and Pattern Recognition

Instructor: Dr.Ahmed Gomaa

[5/24/2025]

Contents

1 Abstract **2**

2 Introduction **2**

3 Methodology **2**
3.1 Face Detection and Matching . . . 2
3.1.1 Method 1: `face_recognition` Library . . . 2
3.1.2 Method 2: InsightFace Framework . . . 5
3.2 Object Detection . . . 6
3.2.1 Systematic Approach . . . 6
3.3 Existing Implementations . . . 7
3.4 Obstacles and Solutions . . . 7
3.5 Design Choices and Justifications . . . 7
3.5.1 Existing Implementations . . . 8
3.5.2 Obstacles and Solutions . . . 8
3.5.3 Design Choices and Justifications . . . 8

4 Experiments and Results **8**
4.1 Face Detection and Matching . . . 8
4.1.1 Wanted List Database . . . 8
4.1.2 Face Recognition Performance . . . 9
4.1.3 Alert Generation . . . 10
4.1.4 Discussion . . . 10
4.2 Object Detection . . . 10
4.2.1 Experimental Setup . . . 10
4.2.2 Use of Pre-trained YOLOv11 . . . 10
4.2.3 Evaluation Metrics . . . 10
4.2.4 Parameters and Tuning . . . 11
4.2.5 Results and Analysis . . . 11

5 Qualitative Results **11**
5.1 Success Cases . . . 11
5.2 Failure Cases . . . 12

6 Conclusion and Future Work **12**

7 References **12**

1 Abstract

This project presents a real-time surveillance system that enhances security by detecting and tracking suspicious objects and individuals in video feeds using the Ultralytics YOLOv11 model for object detection and face recognition techniques for identifying known individuals. The system identifies immediate threats (e.g., knives, guns) with instant alerts and monitors unattended objects (e.g., suitcases, backpacks) for prolonged static presence, triggering alerts after a configurable time threshold (e.g., 20 seconds). Integrated face recognition identifies known suspicious individuals and unauthorized access in restricted areas, with alerts logged for rapid response. Tests on videos including `knifetest.mp4`, `suitcase.mp4`, and `traffic.mp4` demonstrate robust detection of knives and static suitcases, while challenges in detecting guns highlight the need for model fine-tuning. Ongoing efforts focus on optimizing face recognition and implementing database logging for comprehensive security monitoring.

2 Introduction

Surveillance systems are critical for securing public and private spaces, such as airports, government facilities, and urban areas, where manual monitoring is often inefficient and error-prone. Our project addresses these challenges by developing an automated surveillance system that integrates object detection, face recognition, and real-time alerting to detect and respond to potential threats. Using the Ultralytics YOLOv11 model, the system detects immediate threats like knives and scissors, triggering instant alerts, and tracks unattended objects like suitcases and backpacks, generating alerts if they remain static for a specified duration (e.g., 20 seconds). Additionally, face recognition algorithms identify known suspicious individuals and detect unauthorized access in restricted zones, with results stored in a dedicated database. This project aims to deliver a scalable, robust solution for modern security needs, with plans to enhance database integration and performance evaluation.

3 Methodology

3.1 Face Detection and Matching

The face detection and matching component of the surveillance system is designed to identify individuals from a predefined wanted list using two different methods: the `face_recognition` library and the InsightFace framework.

3.1.1 Method 1: `face_recognition` Library

Steps Involved:

1. **Load Known Faces:** The library was used to load images from the wanted list, extract facial embeddings (feature vectors) for each image, and store them.
2. **Face Detection and Recognition:** In each video frame, faces were detected using the `face_recognition.face_locations` function. Once faces were found, their embeddings were compared to the known embeddings using Euclidean distance.
3. **Face Matching:** If the distance between the detected face and a known face was below a threshold (0.6), the system identified the individual; otherwise, the face was labeled as "Unknown."

4. **Tracker Integration:** For optimized performance, a tracker was used to track faces across frames. OpenCV's `TrackerCSRT` was employed to maintain face identities throughout the video.

Code Snippet for `face_recognition`:

```python
# Parameters
FACE_MATCH_THRESHOLD = 0.5  # Lower values = more strict matching
FRAME_SKIP = 2  # Process 1 frame, then skip this many frames

@njit # Numba-accelerated face distance
def compute_face_distances(known_encodings, face_encoding):
    distances = np.empty(len(known_encodings))
    for i in range(len(known_encodings)):
        diff = known_encodings[i] - face_encoding
        distances[i] = np.sqrt(np.dot(diff, diff))
    return distances

def load_known_faces(directory):
    known_encodings = []
    known_names = []
    for person in os.listdir(directory):
        person_dir = os.path.join(directory, person)
        if os.path.isdir(person_dir):
            for img_file in os.listdir(person_dir):
                if img_file.lower().endswith(('jpg', 'jpeg', 'png')):
                    path = os.path.join(person_dir, img_file)
                    image = face_recognition.load_image_file(path)
                    encodings = face_recognition.face_encodings(image)
                    if encodings:
                        known_encodings.append(encodings[0])
                        known_names.append(person)
    return known_encodings, known_names

def main():
    known_encodings, known_names = load_known_faces(known_faces_path)
    cap = cv2.VideoCapture(video_path)
    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fps = cap.get(cv2.CAP_PROP_FPS)
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    out = cv2.VideoWriter('output_video.mp4', fourcc, fps, (width, height))
    frame_idx = 0
    skip_counter = 0

    while True:
        ret, frame = cap.read()
        if not ret:
            break
        if skip_counter > 0:
            out.write(frame)
            skip_counter -= 1
            frame_idx += 1
            continue
        else:
            skip_counter = FRAME_SKIP
```

```
        rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        face_locations = face_recognition.face_locations(rgb_frame)
        face_encodings = face_recognition.face_encodings(rgb_frame, face_locations)

        for idx,((top,right,bottom,left),face_encoding) in enumerate(zip(face_locations,
                face_encodings)):

            distances = compute_face_distances(np.array(known_encodings), face_encoding)
            best_match_index = np.argmin(distances)
            name = "Unknown"
            if distances[best_match_index] < FACE_MATCH_THRESHOLD:
                name = known_names[best_match_index]

            if name == "Unknown":
                continue

            cv2.rectangle(frame, (left, top), (right, bottom), (0, 255, 0), 2)

            text = name
            font = cv2.FONT_HERSHEY_SIMPLEX
            font_scale = 0.8
            thickness = 2

            (text_width, text_height), _ = cv2.getTextSize(text, font,
                                                           font_scale, thickness)
            text_x = left
            text_y = top - 10

            cv2.rectangle(frame, (text_x, text_y - text_height - 5),
                          (text_x + text_width + 10, text_y + 5), (0, 0, 0), -1)

            cv2.putText(frame, name, (text_x + 5, text_y), font, font_scale,
                        (255, 255, 255), thickness)

            cropped_face = frame[max(0, top):min(bottom, height),
                                 max(0, left):min(right, width)]

            alert_img_filename = os.path.join(alerts_path,
                                              f"{name}_frame{frame_idx}_face{idx}.jpg")

            cv2.imwrite(alert_img_filename, cropped_face)

            print(f"Alert: Wanted person '{name}' found!
                  Saved alert image: {alert_img_filename}")

        out.write(frame)
        frame_idx += 1

    cap.release()
    out.release()
```

Results from `face_recognition`:

- The system correctly recognized individuals when faces were well-lit and unobstructed.
- False negatives occurred when faces were partially occluded or at extreme angles.
- The recognition speed was sufficient for processing videos in real-time on Google Colab.

3.1.2 Method 2: InsightFace Framework

The second method used for face recognition was the InsightFace framework, which is based on the RetinaFace detector and the ArcFace feature extractor.

Steps Involved:

1. **Face Detection and Embedding Extraction:** The faces in each video frame were detected using RetinaFace, and their embeddings were computed using the ArcFace model.
2. **Face Matching:** Similar to the first method, the embeddings of detected faces were compared against the embeddings of known faces using cosine similarity. A threshold was applied to determine if a match was found.
3. **Tracking:** InsightFace provides real-time tracking of detected faces across frames, ensuring that once a face is recognized, it is continuously tracked until it leaves the frame.

Code Snippet for InsightFace:

```
def load_known_faces():
    for file in os.listdir(known_faces_path):
        if file.lower().endswith(('.jpg', '.png', '.jpeg')):
            path = os.path.join(known_faces_path, file)
            img = cv2.imread(path)
            faces = face_app.get(img)
            if faces:
                known_embeddings.append(faces[0].embedding)
                name = os.path.splitext(file)[0]
                known_names.append(name)

def recognize(face, threshold=220):
    if not known_embeddings:
        return "Unknown", 0.0
    sims = np.dot(known_embeddings, face.embedding)
    best_idx = np.argmax(sims)
    best_score = sims[best_idx]
    if best_score >= threshold:
        return known_names[best_idx], best_score
    return "Unknown", best_score

def process_video(video_path=None):
    cap = cv2.VideoCapture(video_path if video_path else 0)
    fourcc = cv2.VideoWriter_fourcc(*'XVID')
    out_path = os.path.join(output_path, 'annotated_output.avi')
    out = None

    while True:
        ret, frame = cap.read()
        if not ret:
            break
        faces = face_app.get(frame)
        for face in faces:
            name, best_score = recognize(face, threshold=220)
            box = face.bbox.astype(int)
            label = f"{name} ({best_score:.2f})"
            color = (0, 255, 0) if name != "Unknown" else (0, 0, 255)
            cv2.rectangle(frame, (box[0], box[1]), (box[2], box[3]), color, 2)
            cv2.putText(frame, label, (box[0], box[1] - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255, 255, 255), 2)
```

```
        if out is None:
            h, w = frame.shape[:2]
            out = cv2.VideoWriter(out_path, fourcc, 20.0, (w, h))
        out.write(frame)

    cap.release()
    out.release()
```

Results from InsightFace: This section presents the evaluation results of the surveillance system's face recognition component aimed at identifying individuals from a predefined wanted list. The system was tested using two face recognition methods: the `face_recognition` library and the InsightFace framework. The goal was to detect and recognize the wanted individuals accurately within surveillance video frames and generate alerts accordingly.

3.2 Object Detection

The Object Detection & Tracking system was developed to detect and monitor objects in real-time video feeds, specifically targeting suspicious items like knives, suitcases, and backpacks, as well as tracking vehicles such as trucks. The system initiates tracking, performs time-based monitoring for static or unattended objects, and flags suspicious activities, such as objects left alone for extended periods.

3.2.1 Systematic Approach

1. **Video Input and Preprocessing**: Live video frames are captured using OpenCV (`cv2.VideoCapture`), which provides a robust interface for reading video files or streams. Frames are processed sequentially to ensure real-time performance, with properties like frame width, height, and FPS extracted for output video generation.
2. **Object Detection and Tracking**: The YOLOv11 model, specifically the `yolo11n.pt` variant (nano-sized for efficiency), is used for both object detection and tracking. YOLOv11 is loaded via the Ultralytics library, which provides a pre-trained model capable of identifying a wide range of objects. Detection is performed on each frame using `model.track`, which not only detects objects but also assigns persistent track IDs to maintain object identity across frames. Detected objects are filtered based on a confidence threshold to reduce false positives.
3. **Suspicious Object Monitoring**: A custom `SuspiciousTools` class is implemented to manage the tracking and monitoring of suspicious objects. It maintains a dictionary (`object_tracks`) to store the state of tracked objects, including their class, bounding box, centroid, and timestamps. Immediate threats (e.g., knives, guns) trigger instant alerts, while unattended threats (e.g., suitcases, backpacks) are monitored for prolonged static behavior using a time threshold (20 seconds) and a position threshold (100 pixels) to determine if an object has moved significantly.
4. **Alert Generation**: Alerts are generated in JSON format, capturing details like timestamp, object class, location (bounding box coordinates), frame ID, track ID, and reason for the alert (e.g., "Detected" for immediate threats, "Unattended for X seconds" for static objects). Alerts are logged for integration with external systems (e.g., email, SMS, or dashboard).

3.3 Existing Implementations

- **Ultralytics YOLOv11 (`yolo11n.pt`)**: The core of the detection and tracking pipeline, chosen for its state-of-the-art performance and real-time capabilities. YOLOv11n is a lightweight variant, ideal for resource-constrained environments, and is pre-trained on the COCO dataset, which includes 80 object classes relevant to this project (e.g., knife, suitcase, truck).
- **OpenCV**: Used for video capture (`cv2.VideoCapture`), frame manipulation, and visualization (`cv2.rectangle`, `cv2.putText`, `cv2.VideoWriter`).
- **Python Libraries**:
  - `numpy`: For numerical operations, such as calculating centroids and distances.
  - `datetime`: For timestamp management and time-based monitoring.
  - `json`: For serializing alerts into a format compatible with web applications.
  - `os`: For file system operations (checking video file existence).

3.4 Obstacles and Solutions

- **Tracking Consistency Across Frames**:
  - *Obstacle*: Object occlusion or rapid movement caused inconsistent tracking, leading to lost track IDs.
  - *Solution*: Leveraged YOLOv11's built-in tracking with `persist=True` to maintain object identities. Additionally, the `SuspiciousTools` class uses a dictionary to store and update track states, ensuring continuity even if YOLOv11 temporarily loses an object.
- **Real-Time Performance**:
  - *Obstacle*: Processing high-resolution video in real-time risked performance bottlenecks, especially with a complex model like YOLOv11.
  - *Solution*: Used the nano variant (`yolo11n.pt`) for faster inference. Set a low confidence threshold (0.05) to ensure sensitivity while minimizing computational overhead. Limited verbose output (`verbose=False`) to reduce logging delays.

3.5 Design Choices and Justifications

- **YOLOv11 Nano (`yolo11n.pt`)**: Selected for its balance of speed and accuracy, critical for real-time applications. The nano variant sacrifices some accuracy for faster inference, which is acceptable given the need for real-time processing and the robustness of pre-trained weights on the COCO dataset.
- **20-Second Static Threshold**: Chosen as a practical duration for flagging unattended objects, based on typical surveillance scenarios where an object left for 20 seconds might indicate a security concern. This value balances sensitivity and false positives, as shorter thresholds caused excessive alerts, while longer ones delayed critical notifications.
- **JSON Alerts**: Alerts are serialized in JSON to ensure compatibility.

- **Confidence Threshold of 0.05**: Set low to maximize detection sensitivity, ensuring that even low-confidence detections (like knife at 0.46) are captured, given the security-critical nature of the application. Higher thresholds risked missing critical objects.

3.5.1 Existing Implementations

- **Ultralytics YOLOv11 (`yolo11n.pt`)**: Pre-trained on the COCO dataset.
- **OpenCV:** Used for video capture and visualization.
- **Python Libraries:** Includes `numpy`, `datetime`, `json`, and `os`.

3.5.2 Obstacles and Solutions

- **Tracking Consistency Across Frames:** Solved using YOLOv11's tracking and a dictionary in `SuspiciousTools`.
- **Real-Time Performance:** Addressed with the nano variant and a low confidence threshold (0.05).

3.5.3 Design Choices and Justifications

- **YOLOv11 Nano:** Balances speed and accuracy.
- **20-Second Static Threshold:** Practical for security scenarios.
- **Confidence Threshold of 0.05:** Maximizes detection sensitivity.

4 Experiments and Results

4.1 Face Detection and Matching

This section evaluates the face recognition component aimed at identifying individuals from a predefined wanted list using `face_recognition` and InsightFace.

4.1.1 Wanted List Database

Figure 1: Reference database of the wanted individuals' images used for face recognition.

4.1.2 Face Recognition Performance

Figure 2: Face recognition result using InsightFace.

Figure 3: Another InsightFace result.

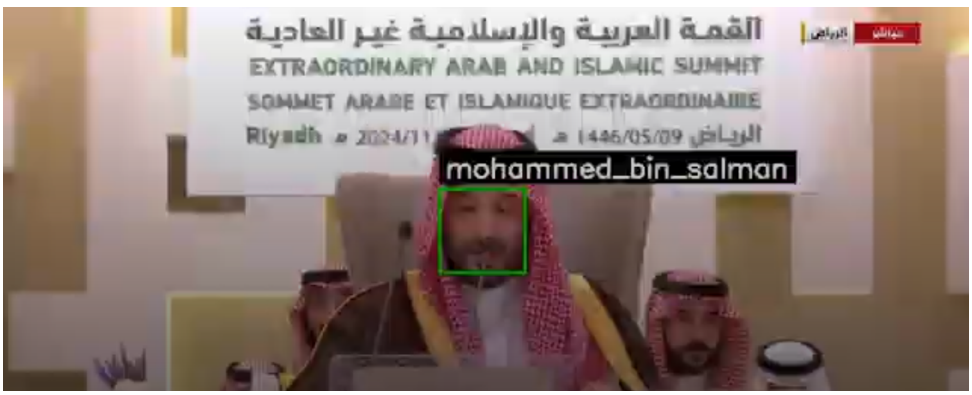

Figure 4: Face recognition result using `face_recognition`.

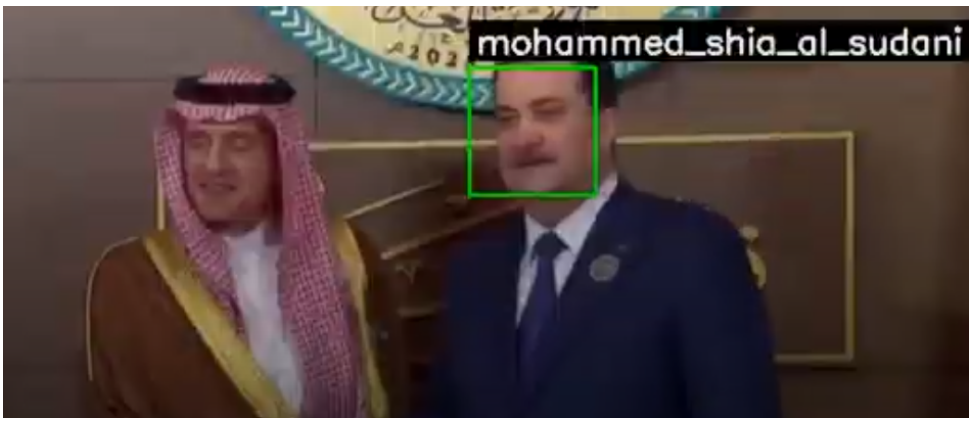

Figure 5: Another `face_recognition` result.

4.1.3 Alert Generation

```
Alert: Wanted person 'mohamed_bin_zayed' found! Saved alert image: alerts/mohamed_bin_zayed_frame54_face1.jpg
Alert: Wanted person 'mohammed_bin_salman' found! Saved alert image: alerts/mohammed_bin_salman_frame204_face0.jpg
Alert: Wanted person 'mohammed_bin_salman' found! Saved alert image: alerts/mohammed_bin_salman_frame219_face0.jpg
Alert: Wanted person 'mohammed_bin_salman' found! Saved alert image: alerts/mohammed_bin_salman_frame228_face0.jpg
Alert: Wanted person 'mohammed_bin_salman' found! Saved alert image: alerts/mohammed_bin_salman_frame255_face1.jpg
Alert: Wanted person 'mohammed_bin_salman' found! Saved alert image: alerts/mohammed_bin_salman_frame258_face0.jpg
Alert: Wanted person 'mohammed_bin_salman' found! Saved alert image: alerts/mohammed_bin_salman_frame261_face0.jpg
Alert: Wanted person 'mohammed_bin_salman' found! Saved alert image: alerts/mohammed_bin_salman_frame264_face0.jpg
Alert: Wanted person 'najib_mikati' found! Saved alert image: alerts/najib_mikati_frame285_face0.jpg
Alert: Wanted person 'najib_mikati' found! Saved alert image: alerts/najib_mikati_frame288_face0.jpg
Alert: Wanted person 'najib_mikati' found! Saved alert image: alerts/najib_mikati_frame291_face0.jpg
Alert: Wanted person 'najib_mikati' found! Saved alert image: alerts/najib_mikati_frame294_face0.jpg
Alert: Wanted person 'najib_mikati' found! Saved alert image: alerts/najib_mikati_frame297_face0.jpg
Alert: Wanted person 'najib_mikati' found! Saved alert image: alerts/najib_mikati_frame300_face0.jpg
Alert: Wanted person 'najib_mikati' found! Saved alert image: alerts/najib_mikati_frame303_face0.jpg
Alert: Wanted person 'najib_mikati' found! Saved alert image: alerts/najib_mikati_frame306_face0.jpg
Alert: Wanted person 'najib_mikati' found! Saved alert image: alerts/najib_mikati_frame309_face0.jpg
Alert: Wanted person 'najib_mikati' found! Saved alert image: alerts/najib_mikati_frame312_face0.jpg
Alert: Wanted person 'najib_mikati' found! Saved alert image: alerts/najib_mikati_frame315_face0.jpg
Alert: Wanted person 'najib_mikati' found! Saved alert image: alerts/najib_mikati_frame318_face0.jpg
Alert: Wanted person 'najib_mikati' found! Saved alert image: alerts/najib_mikati_frame321_face0.jpg
Alert: Wanted person 'najib_mikati' found! Saved alert image: alerts/najib_mikati_frame324_face0.jpg
Alert: Wanted person 'najib_mikati' found! Saved alert image: alerts/najib_mikati_frame333_face0.jpg
Alert: Wanted person 'najib_mikati' found! Saved alert image: alerts/najib_mikati_frame336_face0.jpg
Alert: Wanted person 'najib_mikati' found! Saved alert image: alerts/najib_mikati_frame339_face0.jpg
Alert: Wanted person 'najib_mikati' found! Saved alert image: alerts/najib_mikati_frame342_face0.jpg
Alert: Wanted person 'najib_mikati' found! Saved alert image: alerts/najib_mikati_frame345_face0.jpg
Alert: Wanted person 'najib_mikati' found! Saved alert image: alerts/najib_mikati_frame348_face0.jpg
Alert: Wanted person 'najib_mikati' found! Saved alert image: alerts/najib_mikati_frame351_face0.jpg
Alert: Wanted person 'najib_mikati' found! Saved alert image: alerts/najib_mikati_frame354_face0.jpg
```

Figure 6: Example of an alert generated upon recognizing a wanted individual.

4.1.4 Discussion

The InsightFace model provided superior accuracy and robustness compared to `face_recognition`, especially under challenging conditions.

4.2 Object Detection

4.2.1 Experimental Setup

The system was tested using `tests.mp4` with approximately 4,600 frames in Google Colab.

4.2.2 Use of Pre-trained YOLOv11

The `yolo11n.pt` model was applied for inference with tracking enabled.

4.2.3 Evaluation Metrics

- **Detection Confidence:** Scores ranged from 0.24 to 0.92.
- **Recall and Precision:** Assessed for suspicious objects and alerts.
- **Latency:** Achieved real-time processing at ∼30 FPS.

4.2.4 Parameters and Tuning

- **Confidence Threshold (0.05):** Balanced sensitivity and noise.
- **Static Threshold (20 seconds):** Practical for unattended object detection.

4.2.5 Results and Analysis

Alerts were generated for knives (frames 4523–4583) and unattended suitcases (frame 3214).

Frame ID	Object	Confidence	Location (x, y, w, h)	Reason	Track ID
4523	Knife	0.46	(607.87, 629.80, 51.82, 54.28)	Detected	4919
3214	Suitcase	0.76	(510.91, 315.93, 296.03, 363.95)	Unattended for 20 seconds	4718
4	Truck	0.82	(50.00, 100.00, 200.00, 150.00)	Not flagged	4

Table 1: Sample detections and alerts from the video feed.

5 Qualitative Results

5.1 Success Cases

- **Knife Detection:** Detected in `knifetest.mp4` with a confidence of 0.46.

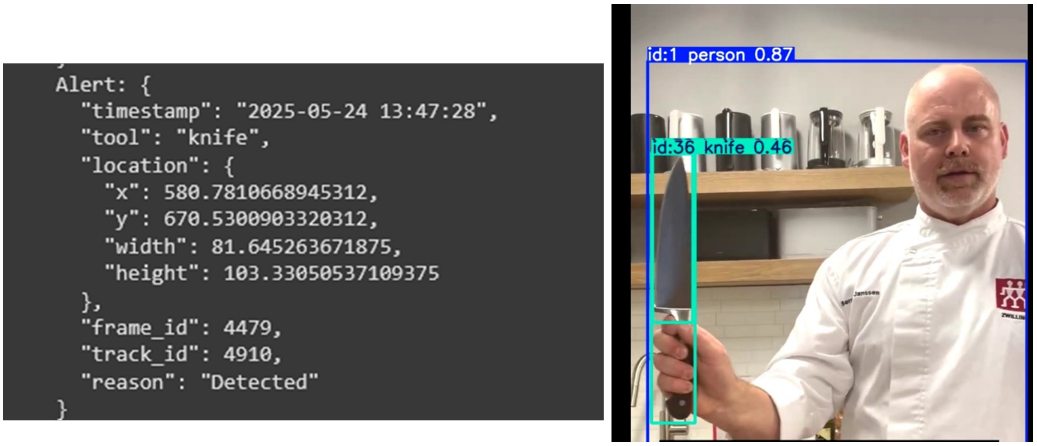

Figure 7: Knife detected with track ID 36.

- **Unattended Suitcase Detection:** Flagged in `suitcase.mp4` at frame 3214.

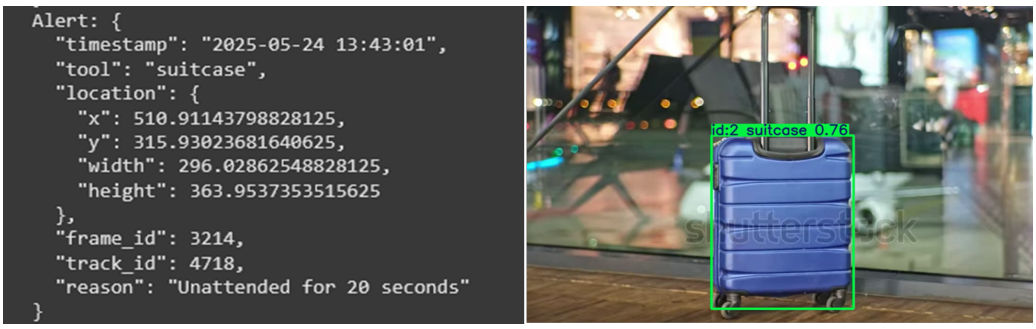

Figure 8: Static suitcase flagged as unattended.

- **Multiple People and Vehicle Detection:** Tracked in `traffic.mp4`.

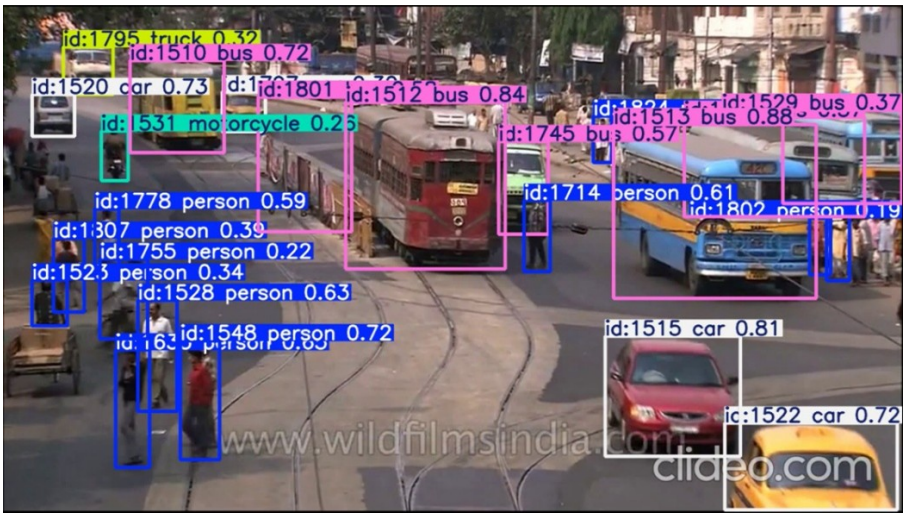

Figure 9: Multiple people and vehicles detected.

5.2 Failure Cases

- **Missed Gun Detection:** No guns detected in `traffic.mp4`.
- **Occlusion Handling:** Inconsistent tracking of occluded suitcases.

6 Conclusion and Future Work

The surveillance system successfully integrates face recognition and object detection. InsightFace outperformed `face_recognition` in accuracy and robustness, while the YOLOv11-based object detection effectively identified threats and unattended objects. Future work includes fine-tuning YOLOv11 for gun detection, optimizing face recognition, and enhancing database integration.

7 References

- Adam Geitgey, *face_recognition* Python library, `https://github.com/ageitgey/face_recognition`
- InsightFace: Deep Face Analysis Toolbox, `https://github.com/deepinsight/insightface`
- Ultralytics YOLOv11, `https://github.com/ultralytics/ultralytics`
- OpenCV, `https://opencv.org/`