Abstract

This project focuses on the automated detection of over-speeding vehicles in video feeds using state-of-the-art computer vision techniques. It leverages YOLOv8 for accurate vehicle detection and DeepSORT for real-time object tracking. By calculating the pixel displacement of tracked vehicles and applying a calibrated scale factor, it estimates their real-world speeds. Vehicles exceeding a predefined speed limit are flagged as over-speeding, with visual alerts displayed on the video frames. The system is implemented using Python, OpenCV for video processing, and YOLOv8’s pretrained model weights. This project aims to provide a practical solution for traffic monitoring, enhancing road safety and supporting law enforcement. The pipeline can be extended to real-time feeds for live surveillance and integrated with dashboards for detailed analytics. The final implementation demonstrates the power of AI in automating traffic enforcement with minimal human intervention.

Furthermore, the project includes robust handling of varying lighting conditions and camera perspectives, ensuring accurate performance across diverse video sources. It also highlights the critical role of calibration and testing in real-world deployments, underscoring the importance of precise distance measurement and frame rate calculations for reliable speed estimation. In addition, the modular design of the system allows for easy updates and future integration with cloud-based storage systems and big data analytics platforms for long-term traffic analysis and pattern recognition.

The project also emphasizes the need for privacy-preserving data handling practices, as vehicle monitoring often involves sensitive information. Techniques such as anonymization of license plates or restricting data retention to only flagged events can be incorporated to comply with privacy laws and public concerns.

Beyond static video feeds, the system’s flexible architecture makes it suitable for deployment in live surveillance scenarios, such as integration with city CCTV networks or roadside cameras. This real-time capability opens new possibilities for automated traffic violation ticketing systems and data-driven urban planning strategies. By analyzing collected data over time, authorities can identify high-risk areas prone to speeding, design more effective road safety measures, and improve overall urban mobility.

The project also explores avenues for incorporating advanced AI techniques like behavior prediction of moving vehicles for comprehensive traffic violation detection. This comprehensive approach demonstrates how a focused application of AI can create powerful tools for smarter cities, safer roads, and efficient law enforcement.

Top of Form

Bottom of Form

Chapter 1  
 Introduction

The domain of this mini project is computer vision and video analytics for traffic monitoring. Over-speeding detection is an essential application in intelligent transportation systems, where the goal is to ensure road safety and regulate traffic behavior. Traditional methods involve physical speed detectors or radar guns, which can be expensive and require manual operation. In contrast, this project uses AI-driven techniques to automate the detection process, eliminating the need for constant human supervision and increasing accuracy and efficiency.

The project is implemented in Python using popular libraries:

* OpenCV for video processing and manipulation.
* Ultralytics YOLOv8 for vehicle detection.
* DeepSORT for robust object tracking.
* NumPy for efficient numerical calculations.

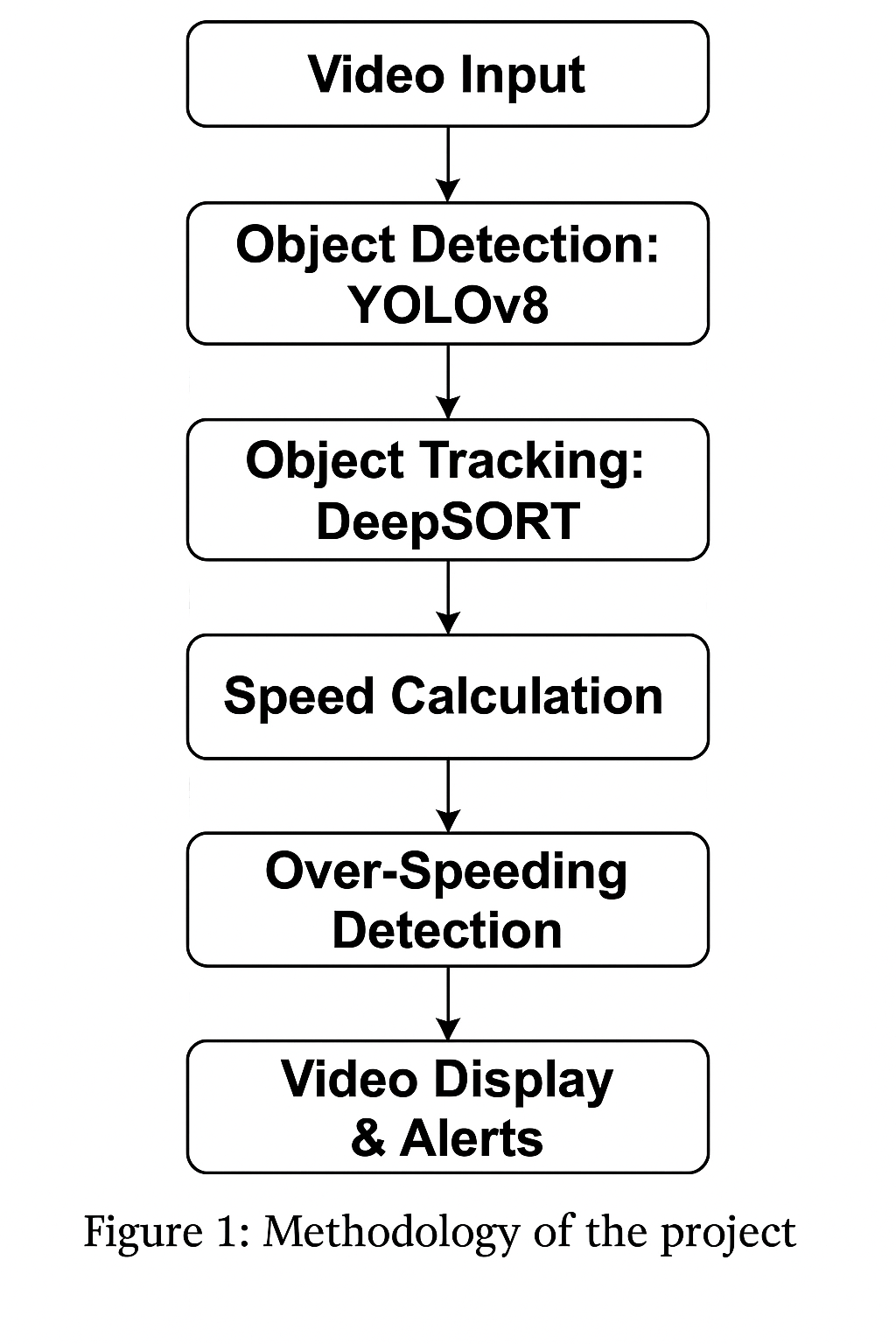
The development was done using Visual Studio Code as the Integrated Development Environment (IDE) on the Windows platform. The project can also be adapted to Linux and Mac environments. This work highlights how advanced deep learning models and tracking algorithms can replace manual processes in traffic monitoring and law enforcement.A key advantage of this approach is its scalability and flexibility. By leveraging YOLOv8’s pretrained model, the system can recognize various types of vehicles across different lighting and weather conditions. DeepSORT provides reliable tracking even in crowded scenes with multiple overlapping vehicles. Together, these models enable continuous monitoring and real-time alerts, ensuring immediate action can be taken when violations occur.

Another significant benefit is the ability to process video feeds from multiple cameras simultaneously, offering a comprehensive view of traffic patterns and speeding trends. This data can be used to inform decision-making by city planners and law enforcement agencies, leading to better road infrastructure and improved public safety.

Furthermore, the modular design of the system ensures that new features or vehicle classes can be added with minimal changes. For example, future enhancements might include automatic number plate recognition (ANPR) for more precise vehicle identification, or integration with traffic light control systems for dynamic traffic management.

Chapter 2  
 Methodology

The system architecture is depicted below, showing the flow of data from video input to final results.

****

This methodology provides a systematic approach to addressing the problem of over-speeding detection in traffic surveillance videos. By combining YOLOv8’s high-precision vehicle detection with DeepSORT’s robust multi-object tracking, the system ensures accurate identification and continuous tracking of vehicles. The incorporation of speed calculation using real-world scale factors bridges the gap between video-based pixel data and actionable, real-world metrics.

1. Video Input:  
The system begins by receiving a video feed. This can be a pre-recorded video file or a live feed from a camera monitoring traffic. The video serves as the input for subsequent processing stages.

2. Object Detection: YOLOv8  
Each frame of the video is processed by the YOLOv8 model (You Only Look Once, version 8). YOLOv8 is a real-time object detection algorithm known for its speed and accuracy. It identifies and classifies objects in the frame, focusing on vehicle classes such as cars, motorcycles, buses, and trucks.

3. Object Tracking: DeepSORT  
After detection, DeepSORT (Deep Simple Online and Realtime Tracking) is used to track the detected vehicles across frames. DeepSORT assigns a unique ID to each vehicle and maintains its identity even as it moves across the scene. This step ensures that the system can calculate the displacement of each vehicle over time accurately.

4. Speed Calculation  
For each tracked vehicle, the system calculates the distance it has moved in the video between consecutive frames. Using a predefined scale factor (0.05 meters/pixel in this case) and the video’s frame rate (FPS), the pixel displacement is converted into real-world speed (km/h). This enables the estimation of how fast each vehicle is moving in the actual scene.

5. Over-Speeding Detection  
The calculated speeds are compared to a pre-set speed limit (60 km/h by default). If a vehicle’s speed exceeds this limit, it is flagged as “Over-Speeding”. This step helps identify traffic violations in real time.

6. Video Display & Alerts  
The final stage involves displaying the processed video frames with additional visual information:

* Vehicle IDs and bounding boxes are overlaid on each vehicle.
* The real-time speed (in km/h) is displayed near each vehicle.
* For vehicles detected as over-speeding, a clear “Over-Speeding!” alert in red is displayed below them.  
  This output can be shown on a monitor in traffic control centers or stored for future analysis.

Chapter 3  
 Code Snippets

import cv2

import numpy as np

from ultralytics import YOLO

from deep\_sort\_realtime.deepsort\_tracker import DeepSort

# --- SETTINGS ---

video\_path = r'video\_samples\abcd.mp4'

scale\_factor = 0.05 # Meters per pixel (calibrate for real-world accuracy)

speed\_limit = 60 # Speed limit in km/h for overspeeding alert

# --- INIT MODELS ---

model = YOLO('yolov8n.pt')

tracker = DeepSort(max\_age=30)

# --- OPEN VIDEO ---

cap = cv2.VideoCapture(video\_path)

if not cap.isOpened():

print(f"Error: Could not open video file: {video\_path}")

exit()

print("Video opened successfully!")

# --- GET FPS ---

fps = cap.get(cv2.CAP\_PROP\_FPS)

if fps == 0 or np.isnan(fps):

fps = 30 # Fallback value

print(f"Video FPS: {fps}")

# --- TRACKING ---

prev\_positions = { }

# --- MAIN LOOP ---

while True:

ret, frame = cap.read( )

if not ret:

print("End of video or read error.")

break

# --- YOLO DETECTION ---

results = model(frame, verbose=False)[0]

detections = [ ]

# Filter vehicle classes: car(2), motorcycle(3), bus(5), truck(7)

for r in results.boxes.data.tolist():

x1, y1, x2, y2, conf, cls = r

if int(cls) in [2, 3, 5, 7]:

detections.append(([x1, y1, x2, y2], conf, 'vehicle'))

# --- TRACK OBJECTS ---

tracks = tracker.update\_tracks(detections, frame=frame)

for track in tracks:

if not track.is\_confirmed():

continue

track\_id = track.track\_id

l, t, r, b = track.to\_ltrb()

cx, cy = (l + r) // 2, (t + b) // 2

curr\_pos = (cx, cy)

# --- SPEED CALCULATION ---

if track\_id in prev\_positions:

dx = curr\_pos[0] - prev\_positions[track\_id][0]

dy = curr\_pos[1] - prev\_positions[track\_id][1]

pixel\_distance = np.sqrt(dx\*\*2 + dy\*\*2)

meter\_distance = pixel\_distance \* scale\_factor

speed\_mps = meter\_distance \* fps

speed\_kph = speed\_mps \* 3.6

# --- PRINT TO TERMINAL ---

print(f"Track ID: {track\_id} | Speed: {speed\_kph:.2f} km/h | "

f"Pixel Distance: {pixel\_distance:.2f} px | FPS: {fps:.2f}")

# --- OVERLAY ON VIDEO ---

cv2.putText(frame, f"{speed\_kph:.1f} km/h", (int(l), int(t) - 10),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (0, 255, 255), 2)

if speed\_kph > speed\_limit:

cv2.putText(frame, "Over-Speeding!", (int(l), int(b) + 20),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 255), 2)

# Update previous position

prev\_positions[track\_id] = curr\_pos

# --- DRAW BOUNDING BOX & ID ---

cv2.rectangle(frame, (int(l), int(t)), (int(r), int(b)), (0, 255, 0), 2)

cv2.putText(frame, f"ID: {track\_id}", (int(l), int(t) - 25),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 255, 0), 1)

# --- DISPLAY FRAME ---

cv2.imshow("Over-Speeding Detection", frame)

# Exit on 'q' key

if cv2.waitKey(30) & 0xFF == ord('q'):

print("Exit requested by user.")

break

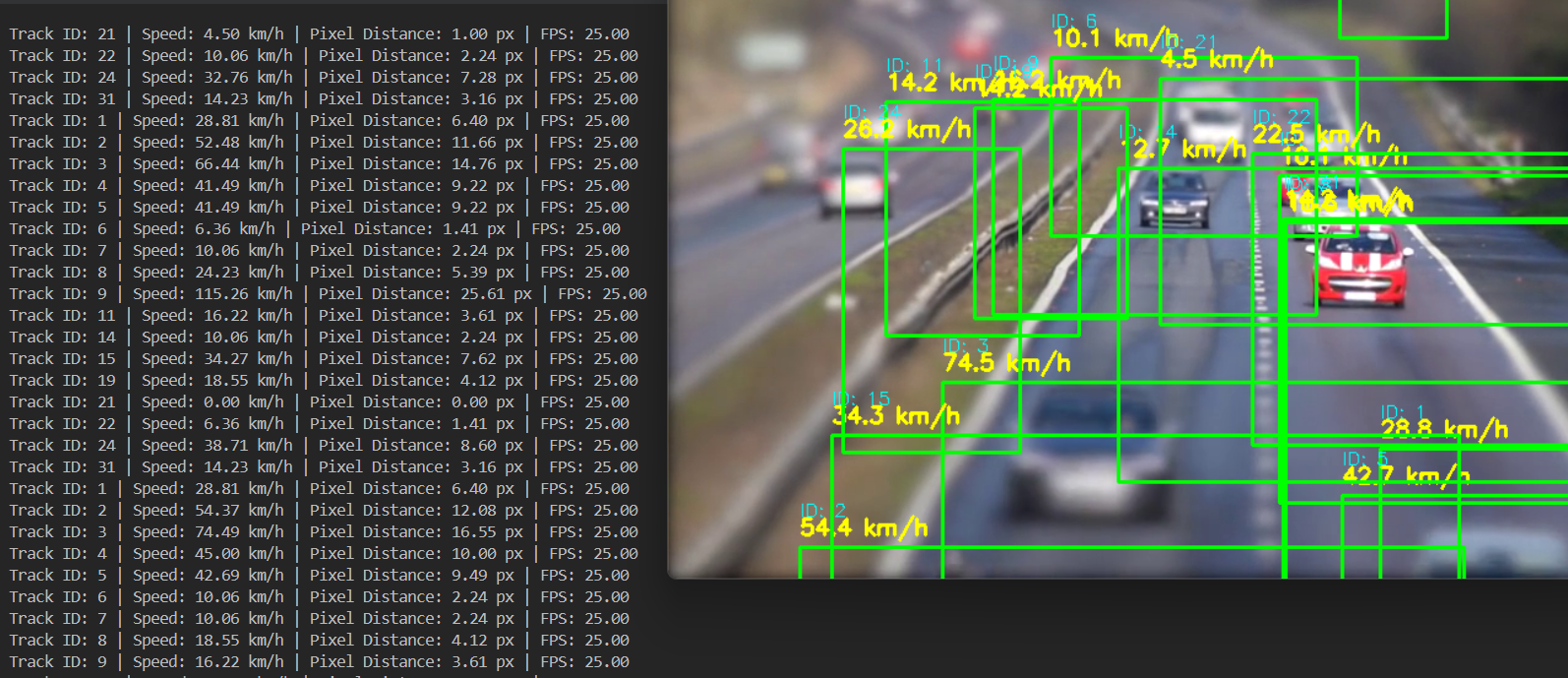
# --- CLEANUP ---

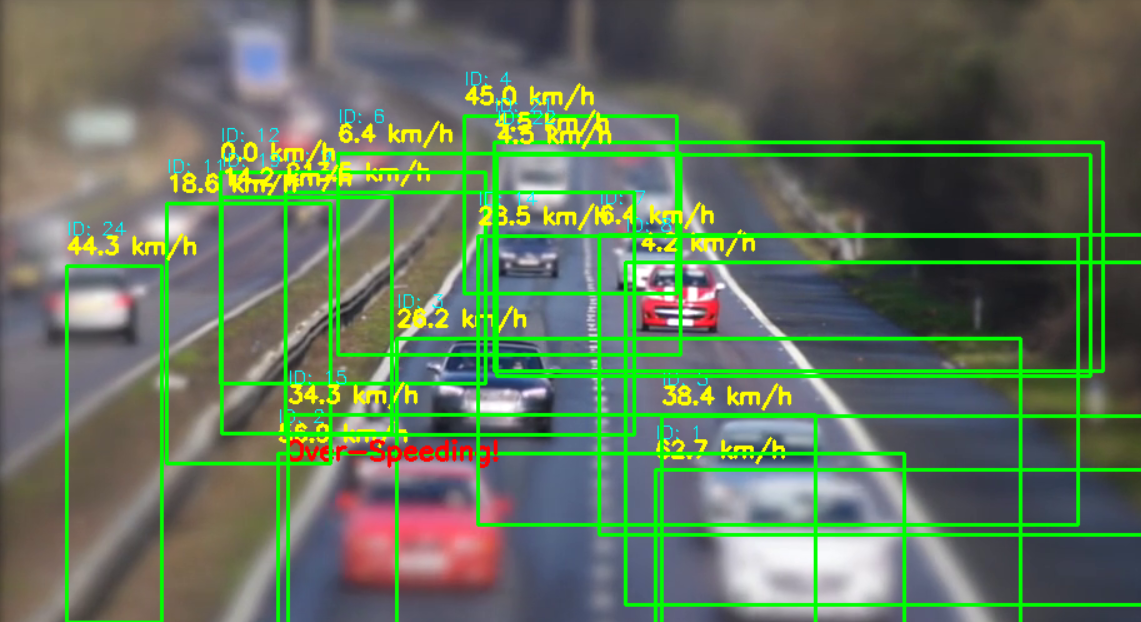
cap.release()

cv2.destroyAllWindows()

print("Program finished.")

Chapter 4  
 Results and Discussion

The system processes the video and overlays real-time information. Screenshots from sample outputs show:  
- Vehicle Detection: Vehicles are identified with bounding boxes and unique IDs.  
- Speed Annotation: Each vehicle’s speed (in km/h) is shown above its bounding box.  
- Over-Speeding Alerts: Vehicles exceeding the speed limit have a red “Over-Speeding!” warning displayed.  
The tracking and speed estimation remain consistent for well-lit and moderately crowded scenes. Accurate speed estimation depends on the correct calibration of the scale factor.



Chapter 5  
 Conclusion

This mini project successfully implements an automated over-speeding detection system using YOLOv8 and DeepSORT. It demonstrates the power of deep learning and tracking algorithms in real-world traffic surveillance applications.  
  
What I learned: This project provided hands-on experience with advanced computer vision techniques. I learned how to integrate object detection with tracking, convert pixel-level information to real-world measurements, and overlay informative alerts on video feeds. This work deepened my understanding of deploying AI models in practical scenarios.