VEHICLE DETECTION AND SPEED ESTIMATION SYSTEM

Semester project

Session 2023-2027

BS in ARTIFICIAL INTELLIGENGE



Department of Software Engineering

Faculty of Computer Science & Information Technology

The Superior University, Lahore

FALL 2023

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Type (Nature of project) | | | [ ✓ ] **D**evelopment [ ] **R**esearch [ ] **R**&**D** | | |
| Area of specialization | | | Artificial Intelligence | | |
| **Project Group Members** | | | | | |
| Sr.# | Reg. # | Student Name | | Email ID | \*Signature |
| (i) |  | Ayesha Faisal | | ayeshafaisal1311@gmail.com |  |

# Table of Contents

[VEHICLE DETECTION AND SPEED ESTIMATION SYSTEM 1](#_Toc184647589)

[Semester project 1](#_Toc184647590)

[Session 2023-2027 1](#_Toc184647591)

[BS in ARTIFICIAL INTELLIGENGE 1](#_Toc184647592)

[Table of Contents 3](#_Toc184647593)

[List of Figures 4](#_Toc184647594)

[Chapter 1 5](#_Toc184647595)

[Introduction 5](#_Toc184647596)

[Chapter 2 7](#_Toc184647597)

[Tool & Technology 7](#_Toc184647598)

[Chapter 3 9](#_Toc184647599)

[Implementation Code 9](#_Toc184647600)

[1. Tracker.py: 9](#_Toc184647601)

[2. Speed\_detection.py: 12](#_Toc184647602)

[Chapter 4 18](#_Toc184647603)

[Conclusion 18](#_Toc184647604)

# 

# List of Figures

[Figure 3-1‑1Tracker.py 6](#_Toc184647565)

[Figure 3-1‑2.Tracker.py 8](#_Toc184647566)

[Figure 3-2‑1.Speed\_detection.py 9](#_Toc184647567)

[Figure 3-2‑2.Speed\_detction.py 11](#_Toc184647568)

[Figure 3-2‑3.Speed\_detection.py 12](#_Toc184647569)

[Figure 3-2‑4.Speed\_detection.py 13](#_Toc184647570)

[Figure 3-2‑5.Speed\_detection.py 14](#_Toc184647571)

# 

# Chapter 1

# Introduction

**Title: Vehicle Detection and Speed Estimation System**

**Objective**

This project focuses on developing an advanced vehicle detection and tracking system using a combination of computer vision techniques and machine learning. The main goal is to identify vehicles, track their movements, and record relevant data such as their unique IDs, speeds, and directions. This system can be applied in traffic monitoring, vehicle counting, and smart city infrastructure management.

**Overview**

The project utilizes the YOLO (You Only Look Once) object detection model from the Ultralytics library to detect vehicles in real-time from video footage. It also employs tracking algorithms to ensure that the vehicles are consistently identified across multiple frames.

**Key Components**

* **Video Input and Processing:**

The system reads a video stream **using OpenCV** and resizes each frame to ensure uniform processing dimensions (1020 x 637).

* **Object Detection:**

The **YOLO model** identifies bounding boxes around vehicles in each frame. These bounding boxes include coordinates (x1, y1, x2, y2) and a class ID to specify the type of vehicle (e.g., car).

* **Tracking Mechanism:**

Each detected vehicle is assigned a unique ID using a tracking system. The center point of the bounding box is calculated to monitor its movement.

* The proximity between vehicles in successive frames is evaluated to maintain consistent tracking.
* **Speed Calculation:**

The time taken for a vehicle to cross two predefined lines is recorded. Using this time and the known distance between the lines, the speed of the vehicle is calculated.

* **Annotations and Visualizations**:

Each frame is annotated with bounding boxes, unique vehicle IDs, and calculated speeds. Lines and labels are drawn to indicate tracking points and thresholds.

* A circle is drawn at the center of each detected vehicle for visualization.
* **Output Video Creation:**

The processed frames, with all annotations and visualizations, are saved to an output video file named **output.avi** using the XVID codec at 20 frames per second.

# Chapter 2

# Tool & Technology

**1. Software and Libraries**

* **VS Code:** The project development environment. Visual Studio Code provided an intuitive interface and support for Python debugging and development.
* **Python Kernel (3.12.4):** The project was executed using the Python kernel, ensuring compatibility with required libraries and seamless execution.

**2. Programming Language**

**Python:** Python was used for the project due to its extensive libraries, readability, and support for AI-based applications.

**3. Libraries and Frameworks**

* **OpenCV:** Used for image and video processing, including resizing, drawing, and visualization of frames.
* **Pandas:** Utilized for data manipulation and extraction of bounding box coordinates from YOLO model outputs.
* **Math:** Used to calculate Euclidean distances for object tracking.
* **Ultralytics YOLO:** YOLO model was used for vehicle detection and classification.

**Hardware Specifications**

**System Configuration**

* **Processor:** Intel(R) Core(TM) i5-6300U CPU @ 2.40GHz, 2.50GHz. The dual-core processor provided sufficient computational power for running the tracking and detection algorithms.
* **Installed RAM:** 8.00 GB (7.66 GB usable). This memory capacity was adequate for handling video frame processing and YOLO model operations.
* **System Type:** 64-bit Operating System, x64-based Processor. This architecture ensured compatibility with modern Python libraries and tools.

# 

# Chapter 3

# Implementation Code

## Tracker.py:

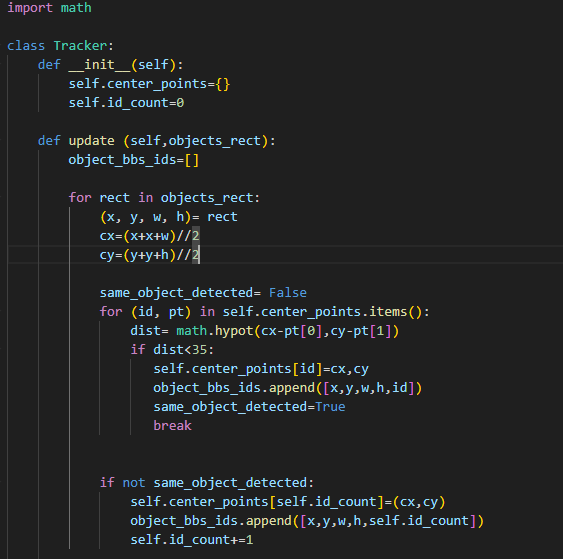
****

Figure 3-‑Tracker.py

**Explanation:**

The “Tracker” class is used for tracking objects in a video by assigning unique IDs and updating their positions across frames. Here's how it works:

**Key Components:**

**1. Attributes:**

- **“center\_points”:** A dictionary storing the center coordinates of tracked objects with their IDs.

- **“id\_count”:** A counter to assign unique IDs to new objects.

**2. Method: “update”**

**- Input:** A list of bounding boxes for detected objects (‘[x, y, w, h]’).

**- Output:** A list of bounding boxes with their assigned IDs.

**How It Works**

**1. Calculate Center Points**: For each bounding box, the center point ‘(cx, cy)’ is computed.

**2. Match Existing Objects:** The method checks if the center point is close (distance < 35) to any previously tracked object using the Euclidean distance formula. If matched, the existing object's center is updated, and its ID is reused.

**3. Assign New IDs:** If no match is found, a new ID is assigned, and the object is added to the tracking list.

**4. Update the List:** The updated bounding box with its assigned ID is added to the output list.

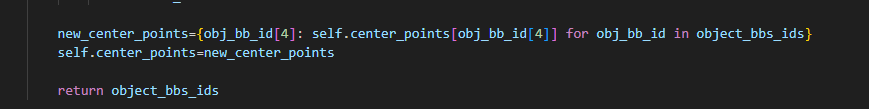


Figure 3-‑.Tracker.py

**Explanation:**

This code updates the “center\_points” dictionary to retain only the objects detected in the current frame, discarding old data. It creates a new dictionary with current object IDs and their center points, replaces the old dictionary, and returns the updated list of bounding boxes with for further processing.

## Speed\_detection.py:

****

Figure 3-‑.Speed\_detection.py

**Explanation:**

1. **Importing Libraries:** Essential libraries like **cv2** for computer vision, pandas for data handling, and **ultralytics for YOLO** object detection are imported. Tracker is a custom module for tracking objects, and time is used for calculating speed.
2. **Loading the YOLO Model:** A pre-trained YOLOv8 model (yolov8s.pt) is loaded to detect objects in video frames.
3. **Mouse Callback Function:** The RGB function captures the mouse position (x, y) whenever it moves over the window named 'RED' and prints the coordinates.
4. **Setting up Video Input:** A video file (highway.mp4) is opened for processing using OpenCV's VideoCapture.
5. **Reading Class Labels**: A text file (classeslist.txt) containing object class labels is read and split into a list for mapping detection outputs to their respective classes.
6. **Initializing the Tracker:** The Tracker class is initialized to track objects between frames.
7. **Region Definition:** Two horizontal lines (cy1 and cy2) are defined to mark regions in the video for monitoring object movement. An offset is used to allow slight leeway when objects cross these lines.
8. **Counters and Dictionaries:**

* **cars\_down and cars\_up:** Store timestamps for vehicles moving down or up, respectively.
* **counter and counter1:** Lists to keep track of unique vehicle IDs that have crossed each line.

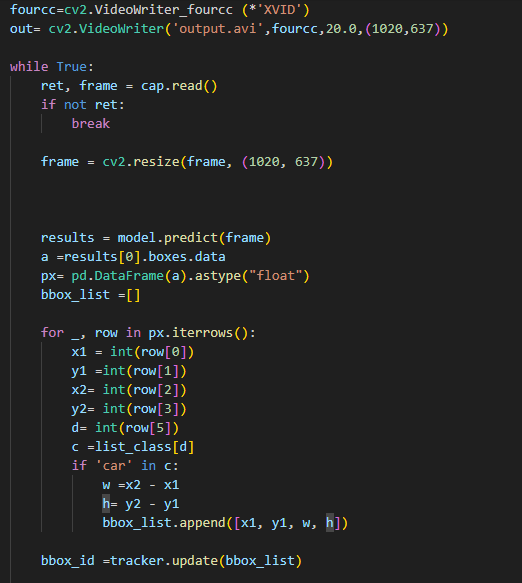


Figure 3-‑.Speed\_detction.py

**Explanation:**

This code processes video frames to detect and track cars. It reads each frame, resizes it, and uses a pre-trained model to detect objects. The bounding boxes of detected cars are extracted, and their dimensions are used for tracking. The tracker assigns unique IDs to the cars and updates their positions across frames. The processed video is saved as an output file.

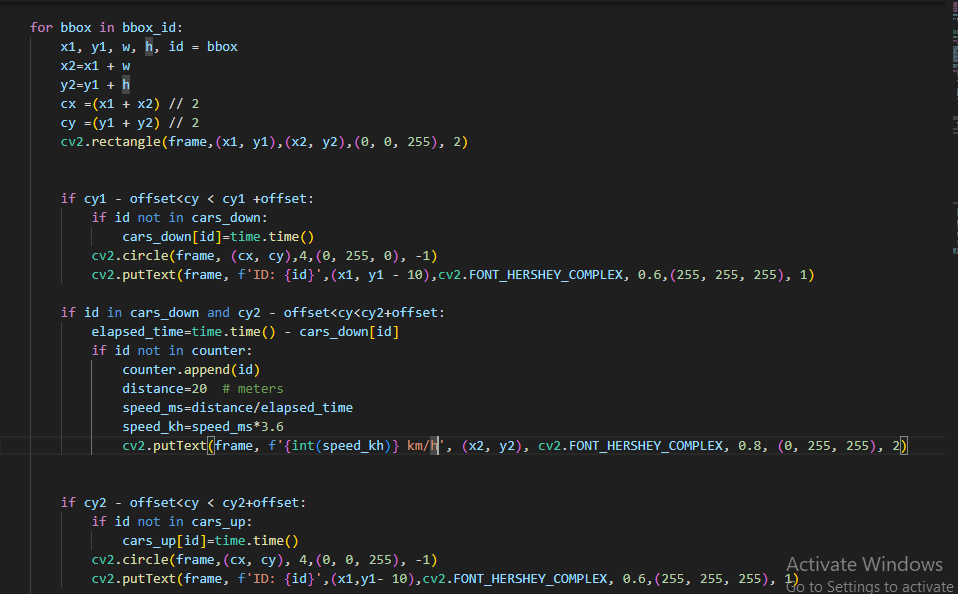


Figure 3-‑.Speed\_detection.py

**Explanation:**

This code tracks cars and calculates their speed based on their movement. For each bounding box, it calculates the center point (cx, cy). It draws a rectangle around the car and checks if the car crosses a specific horizontal line (cy1 or cy2) within a defined offset. If a car crosses the line, it records the time. When a car crosses another line, the code calculates the speed by measuring the time elapsed and the distance traveled. The speed is displayed on the frame in kilometers per hour (km/h).

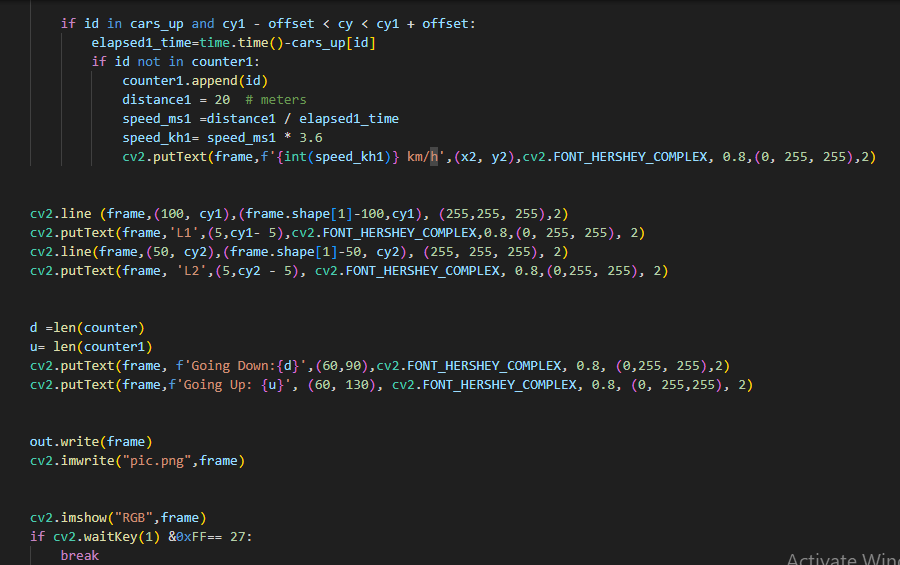


Figure 3-‑.Speed\_detection.py

**Explanation:**

This code tracks cars moving across two horizontal lines (L1 and L2) in a video frame and calculates their speed.

- When a car crosses the second line (L2), its ID and time are recorded.

- If the car then crosses the first line (L1), the elapsed time is used to calculate its speed in km/h, which is displayed on the frame.

- The code also draws the lines (L1, L2) on the frame and labels them.

- It keeps a count of cars moving down (crossing L1 after L2) and cars moving up (crossing L2 after L1), displaying the counts on the frame.

- The processed frame is saved to both a video file (`out.write(frame)`) and an image (`cv2.imwrite("pic.png", frame)`).

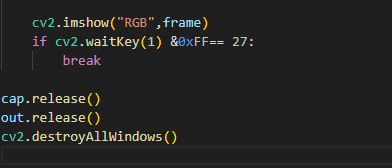


Figure 3-‑.Speed\_detection.py

**Explanation:**

This code displays the current video frame in a window titled "RGB" using cv2.imshow. The program checks if the 'Esc' key (ASCII code 27) is pressed, and if so, it breaks the loop and stops the video. Finally, the code releases the video capture and output objects (cap.release() and out.release()) and closes any open OpenCV windows (cv2.destroyAllWindows()).

# Chapter 4

# Conclusion

**Conclusion:**

The code uses computer vision and deep learning to detect and track vehicles in a video, focusing on calculating their speeds. The YOLO (You Only Look Once) model is used for detecting vehicles, and a tracking algorithm helps follow their movement.

**Key features include:**

- **Object Detection**: YOLO detects vehicles and draws bounding boxes around them.

**- Vehicle Tracking:** A tracking system follows each vehicle, maintaining their identity across frames.

**- Speed Calculation:** The code calculates vehicle speeds by measuring the time it takes to cross specific lines, then converts it to km/h.

**- Data Visualization:** It displays vehicle IDs and speeds on the video, making it easier to analyze their movement in real-time.

# 