Vehicle Detection & Tracking Solution

Overview

This project provides an optimized framework for detecting and tracking vehicles (cars and bikes) in video streams using the YOLO (You Only Look Once) object detection model, OpenCV for visualization, and custom tracking logic implemented in Python. It enables real-time monitoring, counting, and statistical analysis of vehicles in a video file or webcam feed.

Key Strengths

Robust Tracking:

Your VehicleTracker class efficiently manages object IDs, matches detections across frames by combining IoU (Intersection over Union) and spatial proximity, and validates that vehicles have been seen for multiple frames before "confirming" them. This reduces false positives and ensures accurate counting.

Confidence & Confirmation Logic:

Vehicles are only counted as "confirmed" after being detected for 3 consecutive frames, which helps suppress duplicate IDs and spurious detections.

Comprehensive Counting:

The system differentiates between current visible vehicles, maximum simultaneously tracked, and total unique (confirmed) vehicles. This provides granular insights into vehicle flow and density.

User Interaction:

The UI supports live preview, pausing, screen capture, and on-the-fly reset of tracking, making it user-friendly for analysis and debugging.

Code Structure:

The code is modular and organized, with clear separation between detection, tracking, and visualization.

Performance Optimization:

Efficient use of Numpy operations and structure to process frame-by-frame analysis at (near) real time.

Potential Improvements

Code Formatting:

Your code has several instances of mashed-together lines (e.g., import cv2from ultralytics ...). Separating them and ensuring indentation consistency will improve readability and maintainability.

Generalization:

Currently, the tracker is hardcoded for "cars" and "bike." Consider generalizing the logic so it can dynamically adapt to any class from the model.

Documentation:

While your code includes docstrings, consider adding more inline comments, especially for core tracking logic and utility functions.

Resource Release:

Double-check that all video and window resources are closed (though you handle this in most cases).

Code Explanation (Key Parts)

1. Detector & Tracker Setup

Model Loading:

Loads a trained YOLO model (assumed to be fine-tuned on vehicle data). If loading fails, an error is printed.

Video Input:

User can choose between video file or webcam. The code extracts video properties (fps, dimensions) for further processing.

2. The VehicleTracker Class

Initialization:

Stores tracked vehicles (currently visible), all vehicles ever seen, unique ID counter, and configurable parameters for maximum allowed distance between track updates and maximum missing frames.

Distance and IoU Functions:

calculate_distance: Finds the Euclidean distance between the centers of two bounding boxes.

calculate_iou: Standard formula for IoU to assess overlapping area between bounding boxes.

Update Logic:

First Pass: Matches current detections with previously tracked vehicles using combined score (distance and IoU) and minimum thresholds.

Second Pass: Unmatched detections are instantiated as new "unconfirmed" vehicles.

Third Pass: Maintains vehicles that may be temporarily missed for a few frames to prevent track loss.

Fourth Pass: Marks vehicles as "confirmed" if detected consistently; updates their long-term record.

Statistics Functions:

get_current_counts: Cars and bikes visible *now*.

get_total_unique_vehicles: Unique, confirmed over the entire video.

get_total_by_class: Unique, confirmed per class.

3. Detection Loop

Detection Execution:

Runs YOLO object detection on each frame.

Only considers detection results for "cars" and "bike" (can expand if desired).

Each detection triggers an update in the tracker.

Results Visualization:

Bounding boxes drawn for each tracked vehicle, with different colors for "confirmed" and "pending."

Overlay section summarizes stats: current counts, max seen, unique totals, etc.

User controls allow for pausing, screenshotting, and resetting tracking.

Statistical Output:

At the end, the code prints a detection summary: total unique vehicles, per-class counts, max simultaneous, and a timeline of detections.

4. User Interface

CLI Prompts: Asks the user to pick the input source, and adjusts accordingly. Robust to invalid input.

Notable Aspects

Real-Time Capability: Designed to process frames at video frame rates (if hardware is sufficient).

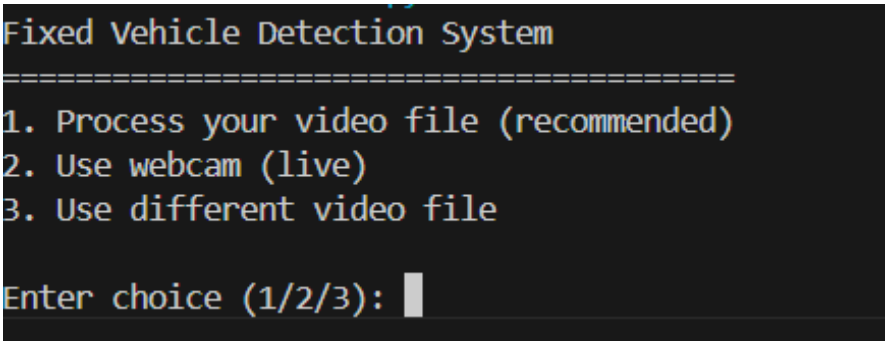
Extensibility: The approach can be tweaked for multi-class tracking, tracking across multiple video streams, or expanding to other object categories.

Application Areas: Useful in traffic analysis, parking management, surveillance, and city planning.

Summary Table

Component	Function
YOLO Detection	Identifies vehicles in each video frame
VehicleTracker	Assigns unique IDs, tracks across frames, filters out spurious detections
Visualization	Real-time drawing of bounding boxes, counters, and overlays
User Controls	Pause, quit, screenshot, reset
Statistics Reporting	Current, maximum, and total unique vehicles, plus per-class breakdowns

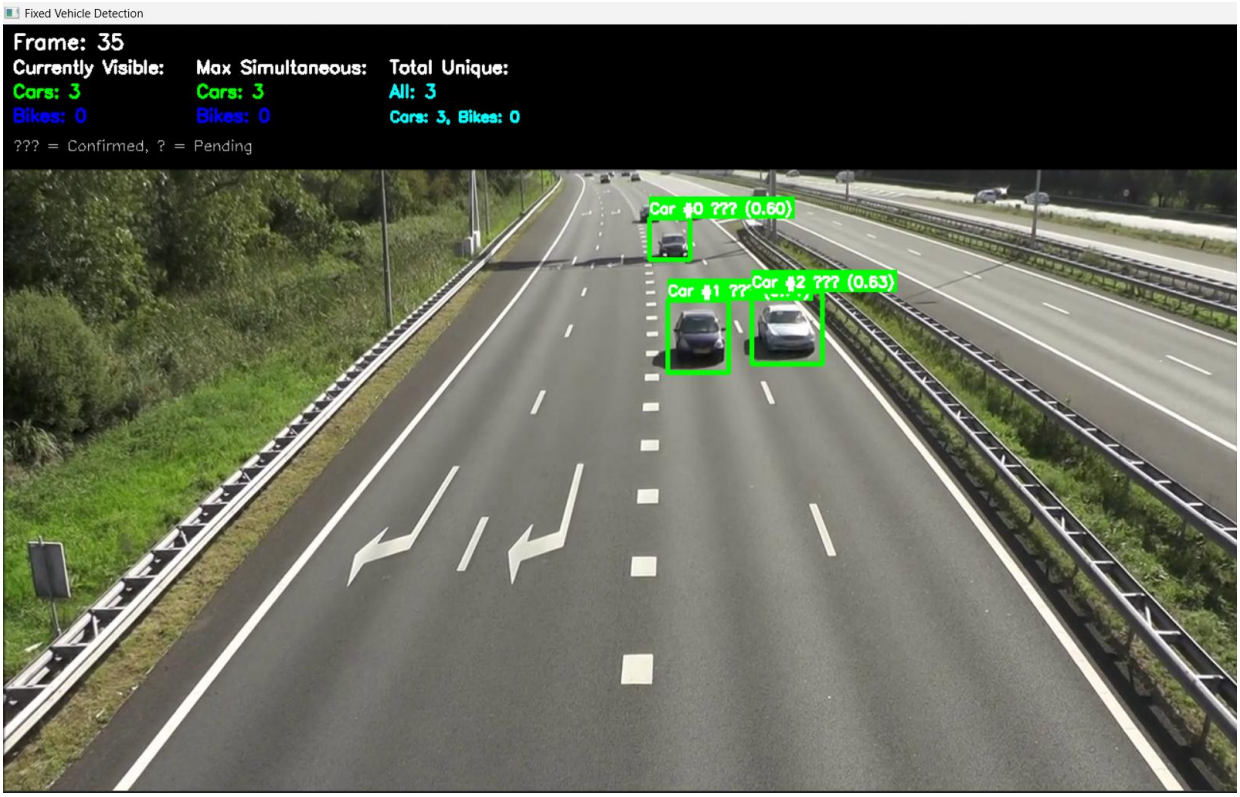
Output



This will be the output.

We can choose which option ,

1. Is for working with already given link in the code
2. This mode we can use live video from webcam, as the input,
3. This feature can use our own file location,with out(“ “).
   (C:\Users\sahal\Downloads\video.mp4)


Fixed Vehicle Detection
Frame: 35
Currently Visible:
Cars: 3
Bikes: 0
Max Simultaneous:
Cars: 3
Bikes: 0
Total Unique:
All: 3
Cars: 3, Bikes: 0
??? = Confirmed, ? = Pending
Car #0 ??? (0.60)
Car #1 ???
Car #2 ??? (0.63)