Anomaly Detection in Vehicle Tracking System

Overview

This Python script utilizes OpenCV and dlib libraries to detect and track vehicles in a video stream. It also includes anomaly scoring logic to detect unusual vehicle speeds.

Dependencies

1. OpenCV (cv2): OpenCV is a popular library for computer vision tasks. It provides various functionalities for image and video processing, including reading, writing, and manipulation of frames.
2. dlib: dlib is a modern C++ toolkit containing machine learning algorithms and tools for creating complex software in C++ to solve real-world problems.
3. Math: The math module provides mathematical functions for calculations required in the code.
4. Time: The time module is used for timing operations, which can be useful for performance measurement and synchronization.

Initialization

1. Loading Cascade Classifier: carCascade = cv2.CascadeClassifier('vech.xml')
2. This line loads a pre-trained cascade classifier XML file that is used for vehicle detection.
    - Opening Video Stream: video = cv2.VideoCapture('a3.mp4')
3. It initializes a video capture object by specifying the input video file.
    - Setting Frame Dimensions: WIDTH = 1280 and HEIGHT = 720
    - These variables set the desired width and height for resizing frames

Function Definitions

1. '**estimateSpeed(location1, location2**)'
   This function calculates the speed of a vehicle by measuring the distance it travels between two frames and converting it into real-world speed.
    - Inputs: location1 and location2 represent the positions of the vehicle in consecutive frames.

- How it Works:
    - Distance Calculation: It measures the distance travelled by the vehicle in pixels.
    - Pixels per Meter (ppm): A fixed value (ppm) is used to convert pixels into meters, representing real-world distances.
    - Speed Estimation: The distance in meters is divided by the time between frames to determine the speed in meters per second (m/s), then converted to kilometres per hour (km/h) for easier understanding.
    - ppm Usage: In the code, a constant ppm value (e.g., 2) is set to define the relationship between pixels and meters. Adjusting this value affects the speed estimation, as it determines how pixel distances translate to real-world distances.

2. '**trackMultipleObjects()**'

   This function is the heart of the script, responsible for tracking vehicles and detecting anomalies in their speeds within the video stream.

   What it Does:

   - Initialization: The function sets up various variables and parameters required for vehicle tracking and anomaly detection. This includes defining colours for visualizing vehicle bounding boxes and initializing counters and dictionaries to keep track of vehicles.
   - Frame Processing Loop:
     - Reading Frames: The function continuously reads frames from the video stream.
     - Vehicle Detection: It detects vehicles in each frame using a pre-trained cascade classifier. This identifies potential regions where vehicles are located.
     - Vehicle Tracking: Once a vehicle is detected, the function initializes tracking for that vehicle using correlation trackers from the dlib library. These trackers follow the vehicles' positions across consecutive frames, ensuring smooth tracking even if the vehicles change in appearance or position.
     - Speed Estimation: It estimates the speed of each tracked vehicle by comparing its positions in consecutive frames. The estimateSpeed function is utilized for this purpose.
   - Anomaly Detection:
     - The function integrates anomaly scoring logic to identify vehicles with unusual speeds.
     - If a vehicle's speed exceeds a predefined threshold, it's marked as an anomaly. This is achieved by changing the colour of the displayed speed text on the video frame, making it visually distinct.
   - Display and Output:
     - Processed frames, including tracked vehicles and annotated speeds, are displayed in real-time.

- Additionally, the processed frames are written to an output video file, allowing for further analysis or documentation of the detected anomalies.

Anomaly Scoring Logic

1. Anomaly scoring logic is integrated within the trackMultipleObjects() function to detect vehicles with unusual speeds.
2. The speed threshold is set at 40 km/h, beyond which vehicles are considered anomalies.

```
# Anomaly Scoring Logic
for i in carLocation1.keys():
    if frameCounter % 1 == 0:
        [x1, y1, w1, h1] = carLocation1[i]
        [x2, y2, w2, h2] = carLocation2[i]
        carLocation1[i] = [x2, y2, w2, h2]
        if [x1, y1, w1, h1] != [x2, y2, w2, h2]:
            if (speed[i] == None or speed[i] == 0) and y1 >= 275 and y1 <= 285:
                speed[i] = estimateSpeed([x1, y1, w1, h1], [x1, y2, w2, h2])
            if speed[i] != None and y1 >= 180:
                if int(speed[i]) < 40:
                    cv2.putText(resultImage, str(int(speed[i])) + "km/h", (int(x1 + w1 / 2), int(y1- 5)),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.75, (100, 0, 0), 2)  # Blue text for normal speed
                if int(speed[i]) >= 40:
                    cv2.putText(resultImage, str(int(speed[i])) + "km/h", (int(x1 + w1 / 2), int(y1- 5)),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0, 0, 100), 2)  # Red text for anomaly
```

3. When the speed of a vehicle exceeds 40 km/h, the script changes the colour of the displayed speed text to red to indicate an anomaly.
4. For vehicles with speeds below 40 km/h, the text colour remains blue, indicating normal speed.

Conclusion

The provided code demonstrates a comprehensive approach to vehicle detection, tracking, and anomaly detection in a video stream. By combining OpenCV and dlib functionalities, it offers a robust solution for analysing vehicle movements and identifying anomalies in real-time video data.