**VEHICLE DETECTION AND COUNTING USING OPENCV AND PYTHON**

The code for vehicle detection and counting is done using a Python script that uses OpenCV to perform vehicle detection and counting from a video source (in this case, a video file). The detection is based on background subtraction and contour analysis. The code also includes a counting line, and it tracks vehicles that cross this line.

The detailed explanation of the code is given below:

import cv2

import numpy as np

import time

time\_text=""

This part imports the libraries OpenCV, NumPy, and time.

OpenCV, or Open Source Computer Vision Library, is an open-source computer vision and machine learning software library. It provides a wide range of tools and functions for image and video processing, as well as computer vision tasks.

NumPy is a powerful numerical computing library in Python that provides support for large, multi-dimensional arrays and matrices, along with mathematical functions to operate on these arrays. It is a fundamental package for scientific computing with Python and is widely used in various fields such as data science, machine learning, signal processing, and more.

The **time** module in Python provides various time-related functions. It allows you to work with time, measure time intervals, and perform various operations related to time.

MINIMUM\_RECTANGLE\_WIDTH = 80

MINIMUM\_RECTANGLE\_HEIGHT = 80

OFFSET\_ERROR = 6

COUNTING\_LINE\_POSITION = 550

These constants define the minimum width and height for a detected rectangle to be considered a vehicle. **OFFSET\_ERROR** is a tolerance for detecting when a vehicle crosses the counting line. **COUNTING\_LINE\_POSITION** is the y-coordinate of the counting line.

detected\_objects = []

car\_count = 0

time\_text = ""

**detected\_objects** keeps track of the centers of detected vehicles. **car\_count** stores the total count of detected vehicles. **time\_text** is used to display the time taken to detect a certain number of vehicles.

def get\_center(x, y, w, h):

x1 = int(w / 2)

y1 = int(h / 2)

cx = x + x1

cy = y + y1

return cx, cy

This function calculates the center coordinates of a rectangle given its top-left coordinates (**x**, **y**) and its width (**w**) and height (**h**).

cap = cv2.VideoCapture("C:\\Users\\welcome\\Downloads\\video.mp4")

background\_subtractor = cv2.createBackgroundSubtractorMOG2()

This line opens the video file for capturing frames. The file path is specified in the argument.

A background subtractor is created using the MOG2 (Mixture of Gaussians) algorithm to detect moving objects.

while True:

ret, frame = cap.read()

The main loop reads frames from the video capture.

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

blur = cv2.GaussianBlur(gray, (3, 3), 5)

foreground\_mask = background\_subtractor.apply(blur)

dilated = cv2.dilate(foreground\_mask, np.ones((5, 5)))

kernel = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE, (5, 5))

closing = cv2.morphologyEx(dilated, cv2.MORPH\_CLOSE, kernel, iterations=2)

The frame is processed to create a foreground mask using background subtraction. Morphological operations (dilation and closing) are applied to enhance the mask.

contours, \_ = cv2.findContours(closing, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

Contours are found in the processed frame.

cv2.line(frame, (25, COUNTING\_LINE\_POSITION), (1200, COUNTING\_LINE\_POSITION), (255, 127, 0), 3)

A counting line is drawn on the frame.

for i, contour in enumerate(contours):

x, y, w, h = cv2.boundingRect(contour)

```

for center\_x, center\_y in detected\_objects:

if center\_y < (COUNTING\_LINE\_POSITION + OFFSET\_ERROR) and center\_y > (COUNTING\_LINE\_POSITION - OFFSET\_ERROR):

car\_count += 1

It checks if the center of a vehicle crosses the counting line and increments the **car\_count** accordingly.

if car\_count == target\_vehicle\_count:

# ...

If the target vehicle count is reached, it calculates the elapsed time and updates **time\_text**.

cv2.putText(frame, "VEHICLE COUNT : " + str(car\_count), (450, 70), cv2.FONT\_HERSHEY\_SIMPLEX, 2, (0, 0, 255), 5)

The frame is annotated with the current vehicle count.

cv2.imshow("Vehicle Detection", frame)

The annotated frame is displayed.

if cv2.waitKey(1) == 27: # Exit on ESC

break

The loop exits when the 'ESC' key is pressed

cv2.destroyAllWindows()

cap.release()

OpenCV windows are closed, and video capture resources are released.

If the target vehicle count is reached, the code displays additional information about the impact of vehicle density on environmental pollution.

Please note that the code assumes a video file path for input and certain constants. Adjustments may be needed based on the specific requirements of your application.

The code iterates through detected contours, and for each contour, it checks if the bounding rectangle size is above the specified minimum. If so, it draws a rectangle and calculates the center of the vehicle



