CourseName: Computer Vision Lab **Course Code:** CSP-422

Experiment:3.1

Aim:

Write a program to determine the effectiveness of incorporating optical flow analysis into object tracking algorithms.

Software Required:

Any IDE (Jupyter Notebook, Pycharm, Google Colab).

Relevance Of the Experiment:

This experiment aims to assess the effectiveness of incorporating optical flow analysis into object tracking algorithms. Optical flow analysis is a computer vision technique that estimates the motion of objects in a sequence of images. By integrating optical flow analysis into object tracking algorithms, we can enhance the accuracy and robustness of tracking moving objects in videos or real-time camera feeds. This experiment helps in understanding the impact of optical flow analysis on object tracking and provides insights into its potential applications in various fields such as surveillance, robotics, and augmented reality. Experiment aims to assess the effectiveness of incorporating optical flow analysis into object tracking algorithms. Optical flow analysis is a computer vision technique that estimates the motion of objects in a sequence of images. By integrating optical flow analysis into object tracking algorithms, we can enhance the accuracy and robustness of tracking moving objects in videos or real-time camera feeds. This experiment helps in understanding the impact of optical flow analysis on object tracking and provides insights into its potential applications in various fields such as surveillance, robotics, and augmented reality.

Description:

Optical flow analysis is commonly integrated into object tracking algorithms to estimate the motion of objects in a sequence of frames. By leveraging optical flow, object tracking algorithms can track objects across frames and estimate their movement accurately. Optical flow analysis is typically incorporated into object tracking algorithms in following manner:

- Optical Flow Estimation: Optical flow estimation calculates the dense motion vectors representing the apparent motion of pixels between consecutive frames. This can be achieved using various techniques, such as Lucas-Kanade, Horn-Schunck, or more advanced approaches like FlowNet or PWC-Net. Optical flow provides a per-pixel displacement field, indicating the direction and magnitude of motion.

- Feature Selection and Tracking Initialization: Object tracking algorithms typically rely on key features or points to track objects. Using the optical flow field, salient features or points of interest are selected, such as corners or distinctive regions, in the initial frame. These features serve as tracking targets, and their motion is estimated using optical flow.
- Motion Estimation and Propagation: Once the initial features are selected, their motion vectors obtained from the optical flow field are used to estimate the overall motion of the tracked object. This can involve various methods, such as averaging the motion vectors or estimating a dominant motion direction. The estimated motion is then propagated to subsequent frames.
- Feature Tracking and Updating: In each subsequent frame, the selected features are tracked by searching for their corresponding positions based on the estimated motion from the previous frame's optical flow. The features' positions are updated, and their motion vectors are re-estimated using optical flow analysis. This allows the algorithm to track the object's movement over time.
- Occlusion Handling and Re-detection: Optical flow-based tracking algorithms need to handle occlusions where objects may be partially or fully hidden by other objects. Techniques such as motion segmentation or appearance modeling can be employed to handle occlusions and ensure accurate object tracking. If an object is completely lost or occluded, re-detection techniques can be used to re-initialize the tracking process.
- Robustness and Adaptation: Object tracking algorithms utilizing optical flow often incorporate mechanisms to handle challenges such as illumination changes, scale variations, or camera motion. These can include adaptive parameter settings, feature selection strategies, or online model updates to ensure robust and accurate tracking performance.

<u>Pseudo code/Algorithms:</u>

```
1. Read the input video or capture frames from a camera feed.
2. Initialize an object tracker using a suitable algorithm, such as the correlation-based
   tracker or the Kalman filter.
3. Iterate through each frame in the video sequence:
      a. Perform optical flow analysis on the current frame and the previous frame to
```

estimate the motion vectors of key points or regions.

b. Update the object tracker with the new frame and motion information.
c. Draw bounding boxes around the tracked objects in the frame.
d. Display the frame with the tracked objects.

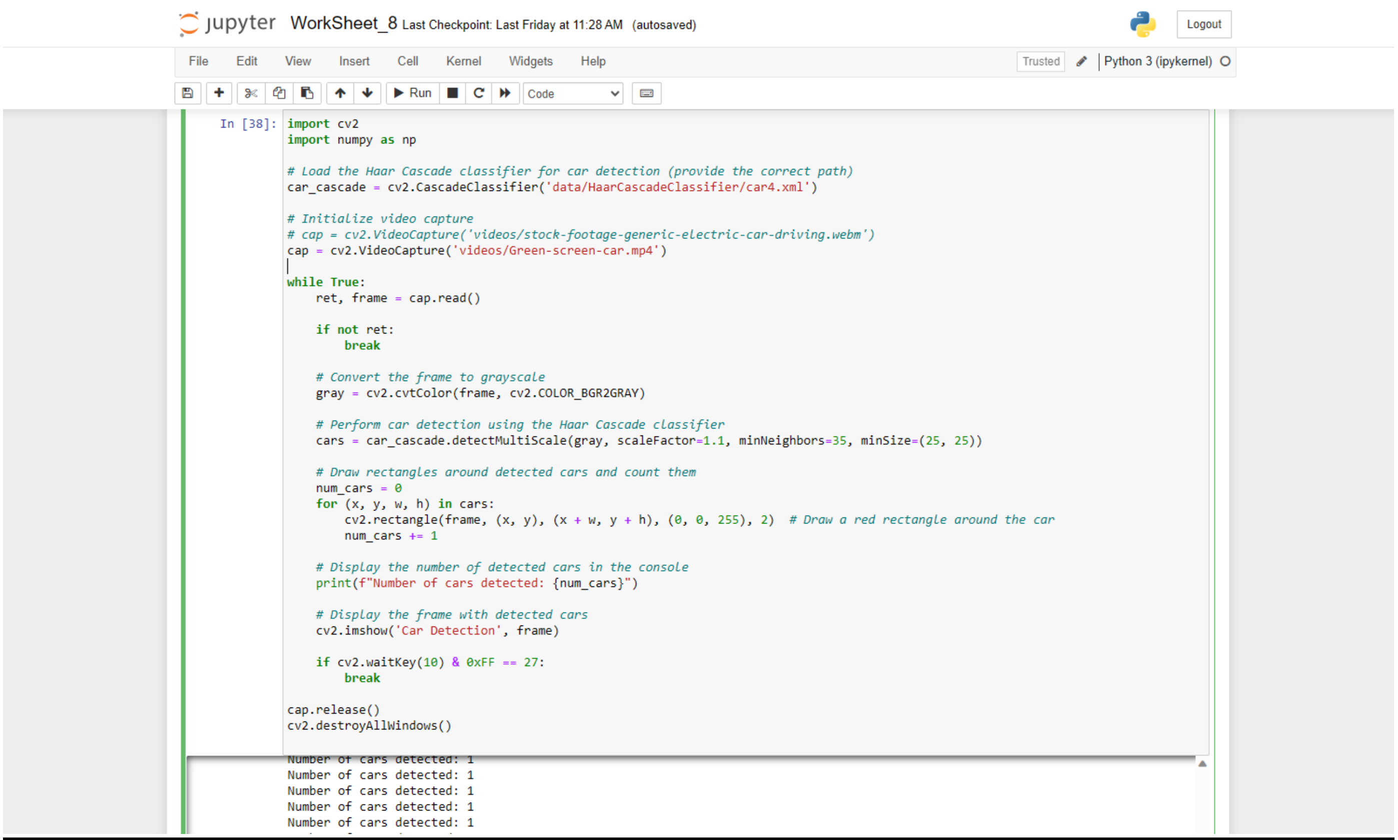
4. Calculate the tracking accuracy by comparing the tracked object's position with the ground truth data, if available.
5. Evaluate the performance of the object tracking algorithm with and without incorporating optical flow analysis by analyzing the tracking accuracy, robustness, and computational efficiency.

Flowchart/Steps:

Implementation:

```
import cv2

import numpy as np


# Load the Haar Cascade classifier for car detection (provide the correct path)

car_cascade = cv2.CascadeClassifier('data/HaarCascadeClassifier/car4.xml')


# Initialize video capture

# cap = cv2.VideoCapture('videos/stock-footage-generic-electric-car-driving.webm')

cap = cv2.VideoCapture('videos/Green-screen-car.mp4')


while True:

    ret, frame = cap.read()
    if not ret:
        break

    # Convert the frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Perform car detection using the Haar Cascade classifier
    cars = car_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=35,
minSize=(25, 25))

    # Draw rectangles around detected cars and count them
    num_cars = 0
    for (x, y, w, h) in cars:
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 2)  # Draw a red rectangle around
the car
        num_cars += 1

    # Display the number of detected cars in the console
    print(f"Number of cars detected: {num_cars}")

    # Display the frame with detected cars
    cv2.imshow('Car Detection', frame)
    if cv2.waitKey(10) & 0xFF == 27:
        break


cap.release()
cv2.destroyAllWindows()
```

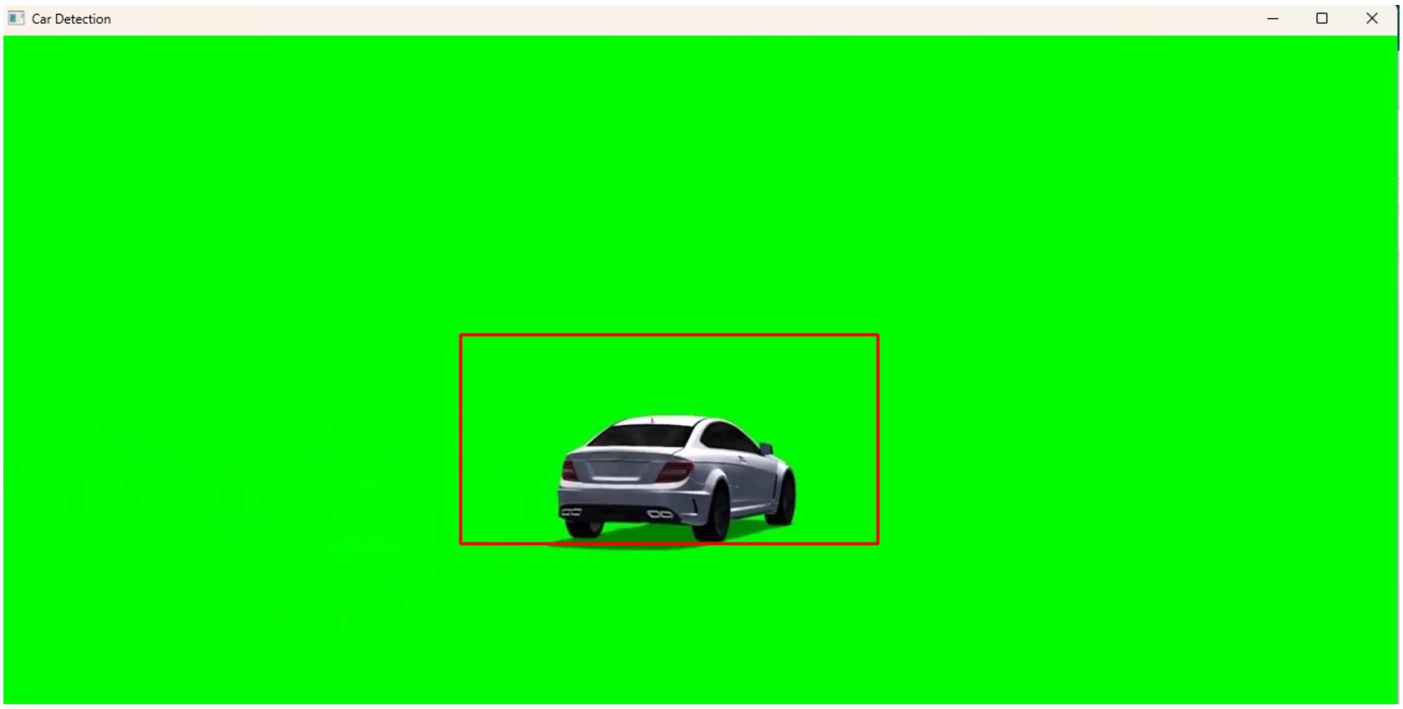
Output:

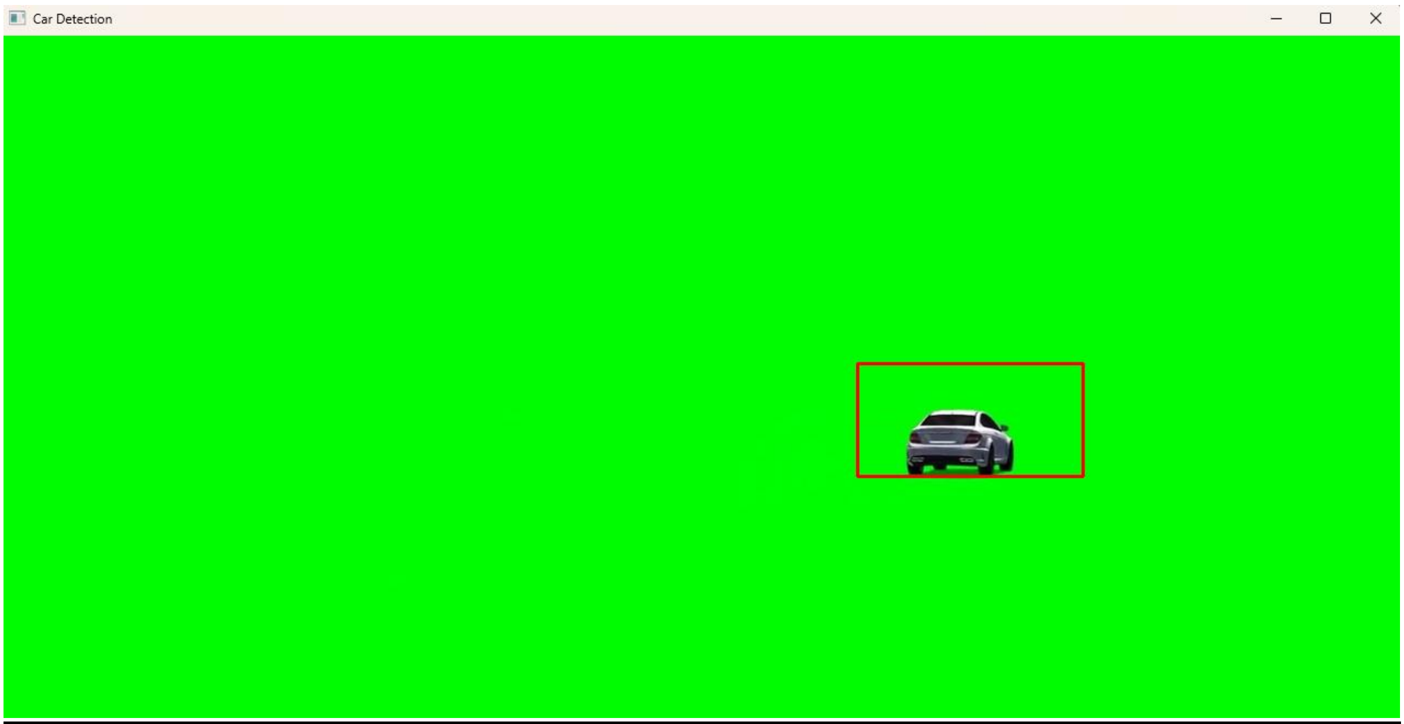

jupyter WorkSheet_8 Last Checkpoint: Last Friday at 11:28 AM (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code

```
In [38]: import cv2
         import numpy as np

         # Load the Haar Cascade classifier for car detection (provide the correct path)
         car_cascade = cv2.CascadeClassifier('data/HaarCascadeClassifier/car4.xml')

         # Initialize video capture
         # cap = cv2.VideoCapture('videos/stock-footage-generic-electric-car-driving.webm')
         cap = cv2.VideoCapture('videos/Green-screen-car.mp4')

         while True:
             ret, frame = cap.read()

             if not ret:
                 break

             # Convert the frame to grayscale
             gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

             # Perform car detection using the Haar Cascade classifier
             cars = car_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=35, minSize=(25, 25))

             # Draw rectangles around detected cars and count them
             num_cars = 0
             for (x, y, w, h) in cars:
                 cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 2)  # Draw a red rectangle around the car
                 num_cars += 1

             # Display the number of detected cars in the console
             print(f"Number of cars detected: {num_cars}")

             # Display the frame with detected cars
             cv2.imshow('Car Detection', frame)

             if cv2.waitKey(10) & 0xFF == 27:
                 break

         cap.release()
         cv2.destroyAllWindows()
```

```
Number of cars detected: 1
Number of cars detected: 1
Number of cars detected: 1
Number of cars detected: 1
Number of cars detected: 1
```

CourseName: Computer Vision Lab **Course Code:** CSP-422


Car Detection


Car Detection