CGC ASSIGNMENT – 3

Name: RAHUL VARMA

Roll No: S20200010212

Motion Detection and Traffic Detection:

Task of this assignment:

In the previous assignment CLAHE is added to improve the quality of the frames.

- ➔ CLAHE

and, when the image captures many rectangles the send back the rectangle to the processing unit and the reject the smaller rectangles.

Now, for this assignment we should detect the vehicles which are closed to each other, that mean when two or more vehicles are comes together then we should detect it as a traffic which I represented in Red rectangle.

And the motion detection I represented in the Green Rectangle.

More Detailed Explanation in the end of pdf.

Before Detection:

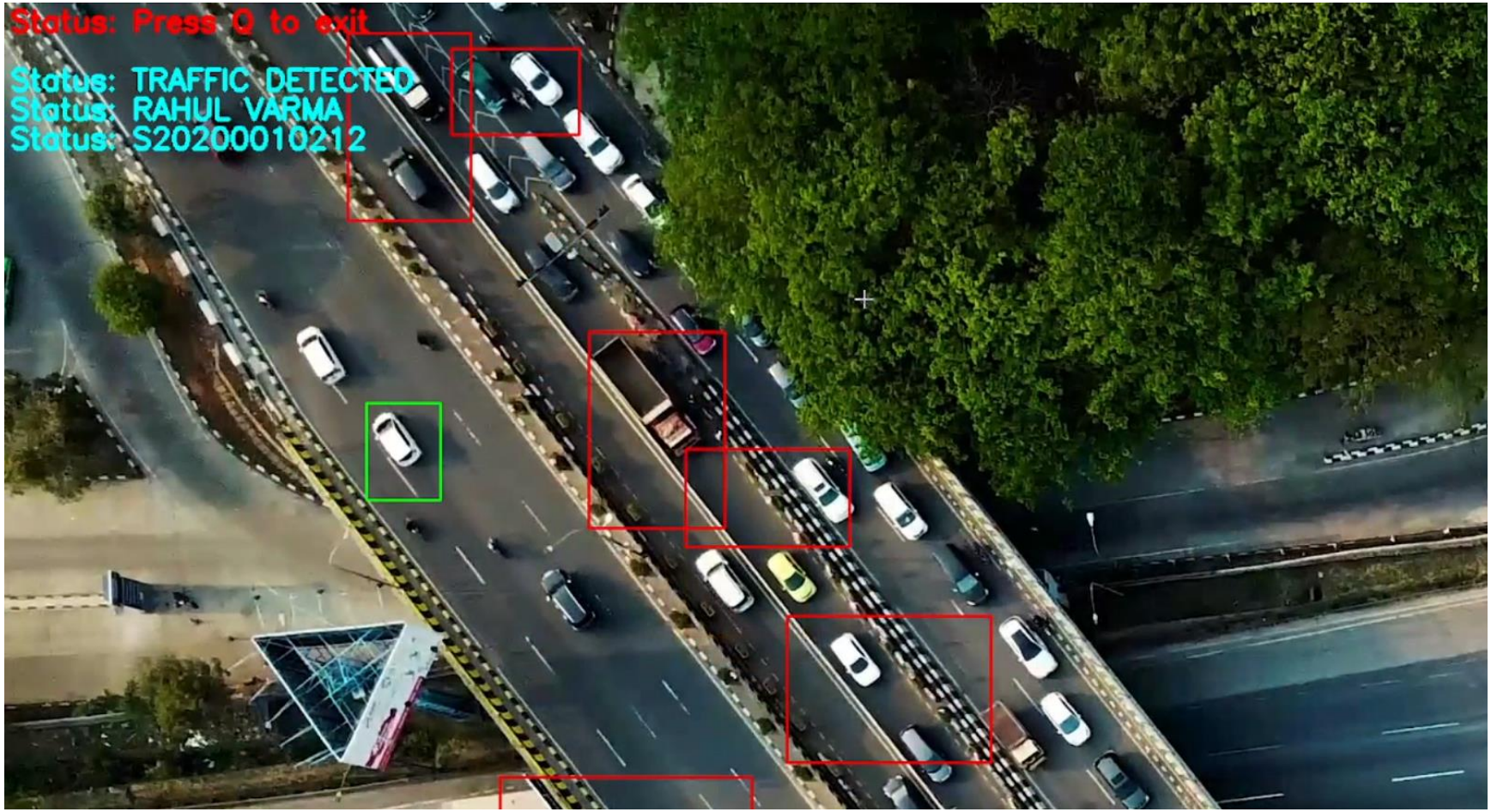
Object Moving is detected in Green Color Rectangle.

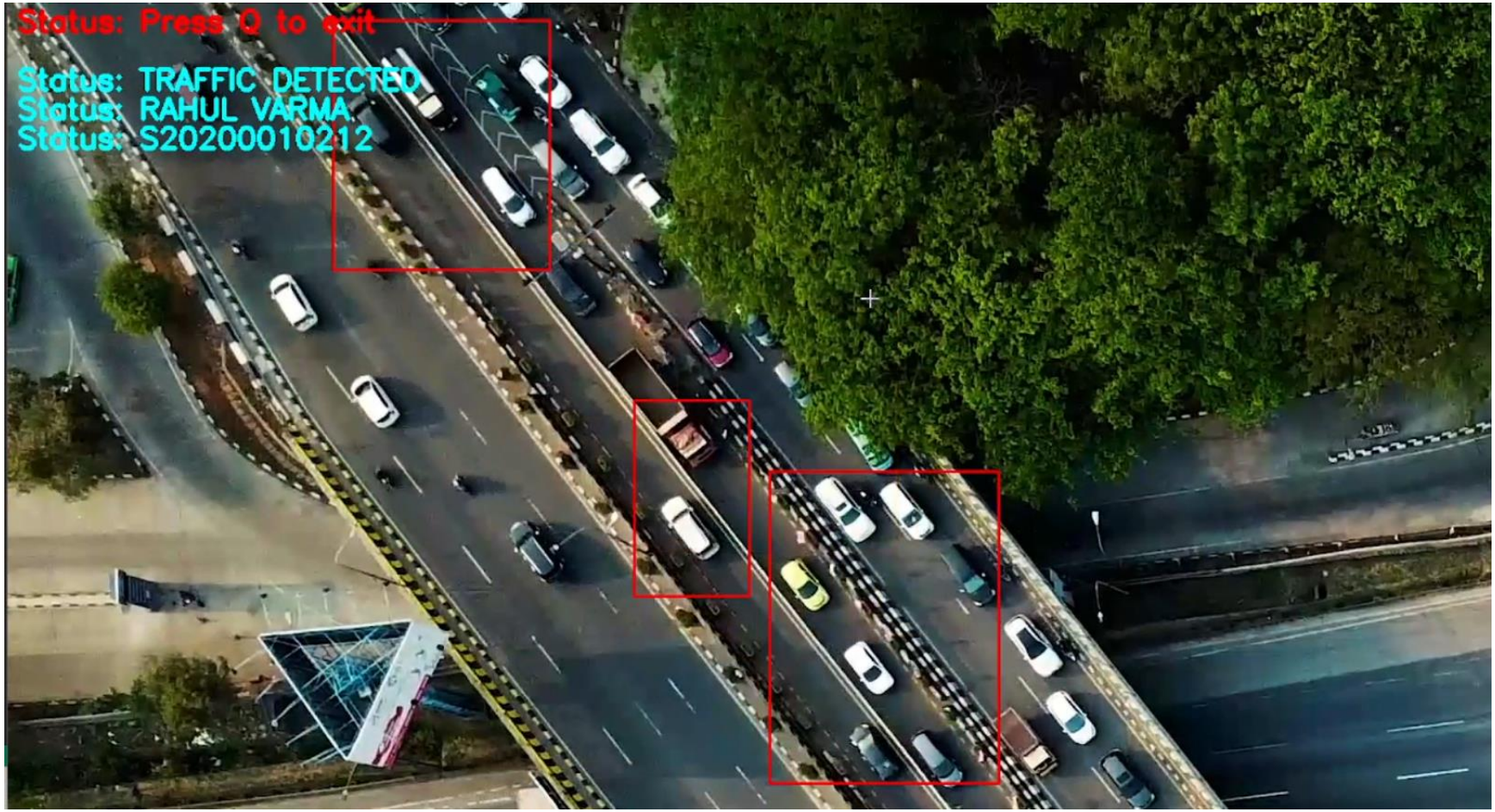
And the traffic is detected in the Red color Rectangle.

Motion and Traffic Detected:

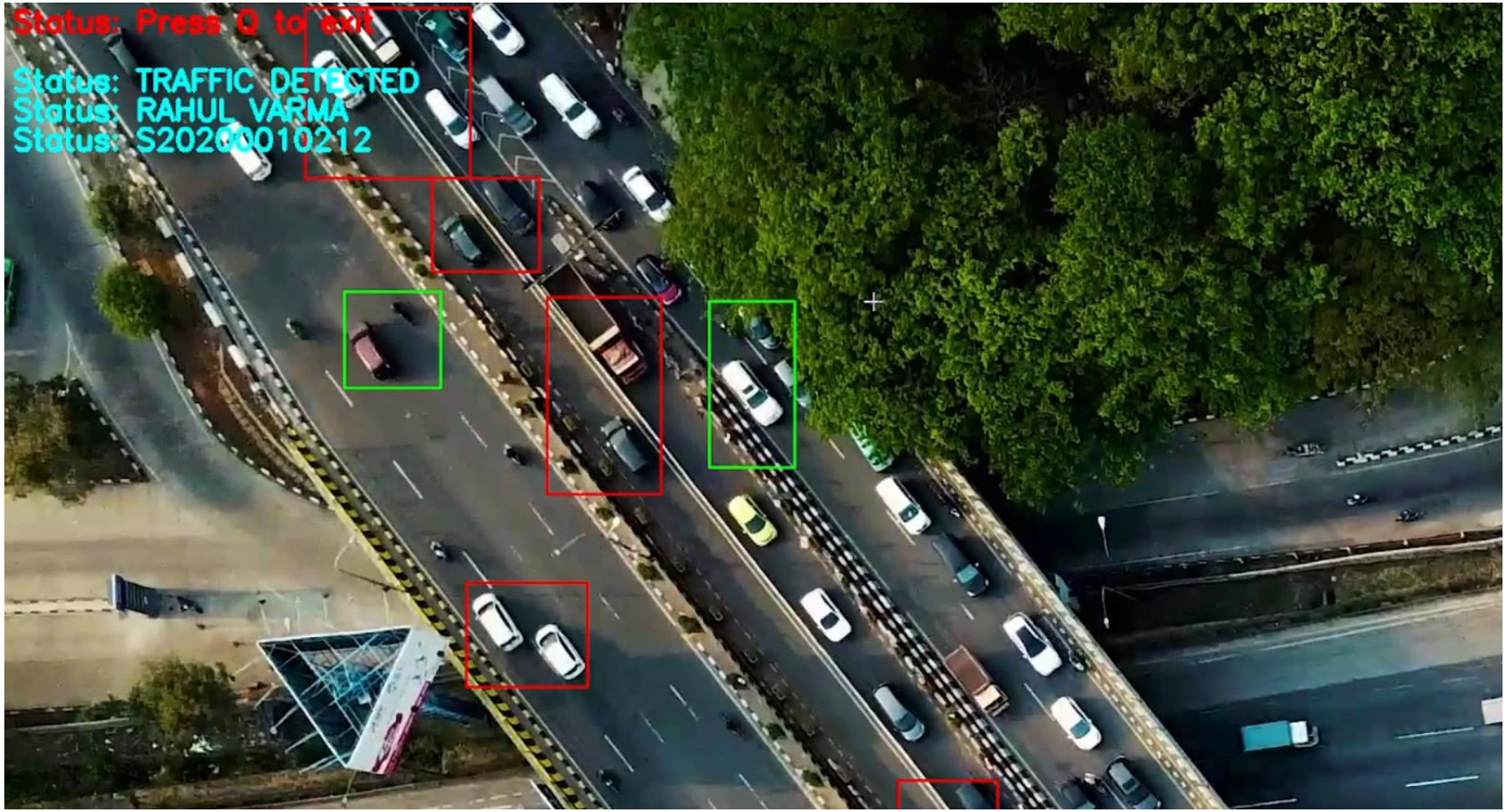
Different Scenarios Screen Shot is Attached:

If two or more vehicles together then there is red color rectangle.




Status: Press Q to exit
Status: TRAFFIC DETECTED
Status: RAHUL VARMA
Status: S20200010212

Single Object Moving is detected in Green Color:



Code:

```python
import cv2
cap = cv2.VideoCapture('video.mp4')
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
while True:
    ret, frame1 = cap.read()
    gray1 = cv2.cvtColor(frame1, cv2.COLOR_BGR2GRAY)
    gray1 = clahe.apply(gray1)
    ret, frame2 = cap.read()
    gray2 = cv2.cvtColor(frame2, cv2.COLOR_BGR2GRAY)
    gray2 = clahe.apply(gray2)
    diff = cv2.absdiff(gray1, gray2)
    thresh = cv2.threshold(diff, 25, 255, cv2.THRESH_BINARY)[1]
    dilated = cv2.dilate(thresh, None, iterations=2)
    cnts, _ = cv2.findContours(dilated, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    grouped_cnts = []
    for c in cnts:
        if cv2.contourArea(c) < 3000:
            continue
        (x, y, w, h) = cv2.boundingRect(c)
        if len(grouped_cnts) == 0:
            grouped_cnts.append([x, y, w, h])
        else:
            found_group = False
            for group in grouped_cnts:
                if abs(x - group[0]) < 100 and abs(y - group[1]) < 100:
                    group[0] = min(group[0], x)
                    group[1] = min(group[1], y)
                    group[2] = max(group[2], x + w - group[0])
                    group[3] = max(group[3], y + h - group[1])
                    found_group = True
                    break
            if not found_group:
                grouped_cnts.append([x, y, w, h])
    for group in grouped_cnts:
        x, y, w, h = group
        if w > 100:
            if len(grouped_cnts) > 2:
                cv2.rectangle(frame1, (x, y), (x + w, y + h), (0, 0, 255), 2)
            else:
                cv2.rectangle(frame1, (x, y), (x + w, y + h), (0, 255, 0), 2)
            cv2.putText(frame1, "Status: {}".format('TRAFFIC DETECTED'), (10, 90), cv2.FONT_HERSHEY_SIMPLEX, 1, (255,255,0), 3)
            cv2.putText(frame1, "Status: {}".format('RAHUL VARMA'), (10, 120), cv2.FONT_HERSHEY_SIMPLEX, 1, (255,255,0), 3)
            cv2.putText(frame1, "Status: {}".format('S20200010212'), (10, 150), cv2.FONT_HERSHEY_SIMPLEX, 1, (255,255,0), 3)
        else:
            cv2.rectangle(frame1, (x, y), (x + w, y + h), (0, 255, 0), 2)
    cv2.putText(frame1, "Status: {}".format('Press Q to exit'), (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 3)
    cv2.imshow('frame', frame1)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

```python
import cv2
# Video input
cap = cv2.VideoCapture('video.mp4')

# Using Clahe from the previous assignment
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))

while True:
    # First frame
    ret, frame1 = cap.read()
    gray1 = cv2.cvtColor(frame1, cv2.COLOR_BGR2GRAY)

    # CLAHE
    gray1 = clahe.apply(gray1)

    # Next frame
    ret, frame2 = cap.read()
    gray2 = cv2.cvtColor(frame2, cv2.COLOR_BGR2GRAY)
    gray2 = clahe.apply(gray2)

    # Calculate difference between frames
    diff = cv2.absdiff(gray1, gray2)

    # Threshold the difference to identify motion
    thresh = cv2.threshold(diff, 25, 255, cv2.THRESH_BINARY)[1]

    # Dilate the thresholded image to fill in holes
    dilated = cv2.dilate(thresh, None, iterations=2)

    # Find contours in the dilated image
```

```python
    cnts, _ = cv2.findContours(dilated, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Group nearby contours
    grouped_cnts = []
    for c in cnts:
        if cv2.contourArea(c) < 3000:
            continue
        (x, y, w, h) = cv2.boundingRect(c)
        if len(grouped_cnts) == 0:
            grouped_cnts.append([x, y, w, h])
        else:
            found_group = False
            for group in grouped_cnts:
                if abs(x - group[0]) < 100 and abs(y - group[1]) < 100:
                    group[0] = min(group[0], x)
                    group[1] = min(group[1], y)
                    group[2] = max(group[2], x + w - group[0])
                    group[3] = max(group[3], y + h - group[1])
                    found_group = True
                    break
            if not found_group:
                grouped_cnts.append([x, y, w, h])

    # Draw rectangles on the original frame
    for group in grouped_cnts:
        x, y, w, h = group
        if w > 100:
            # Draw red rectangle for groups of more than two vehicles
            if len(grouped_cnts) > 2:
                cv2.rectangle(frame1, (x, y), (x + w, y + h), (0, 0, 255), 2)
            else:
                cv2.rectangle(frame1, (x, y), (x + w, y + h), (0, 255, 0), 2)
```

```
            cv2.putText(frame1, "Status: {}".format('TRAFFIC DETECTED'), (10, 90), cv2.FONT_HERSHEY_SIMPLEX, 1,
(255,255,0), 3)
            cv2.putText(frame1, "Status: {}".format('RAHUL VARMA'), (10, 120), cv2.FONT_HERSHEY_SIMPLEX, 1,
(255,255,0), 3)
            cv2.putText(frame1, "Status: {}".format('S20200010212'), (10, 150), cv2.FONT_HERSHEY_SIMPLEX, 1,
(255,255,0), 3)
        else:
            cv2.rectangle(frame1, (x, y), (x + w, y + h), (0, 255, 0), 2)

    cv2.putText(frame1, "Status: {}".format('Press Q to exit'), (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0,
255), 3)


    # Show the frame
    cv2.imshow('frame', frame1)

    # Press Q to exit
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release the capture and destroy the windows
cap.release()
cv2.destroyAllWindows()
```

Detailed Explanation:

- The above code is an implementation of motion detection in a video stream using OpenCV and Python. The goal of motion detection is to
- identify and track changes in the position of objects in the video stream over time. This can be used in a variety of applications,
- such as security systems or traffic monitoring.

- The code begins by importing the necessary libraries, including OpenCV and NumPy. It then loads the video file using OpenCV's
- VideoCapture() function and creates a Contrast Limited Adaptive Histogram Equalization (CLAHE) object to enhance the contrast
- in the grayscale images that will be processed later.

- The core of the motion detection algorithm is implemented in the while loop. In each iteration of the loop, the code reads two
- consecutive frames from the video stream and converts them to grayscale. It then applies CLAHE to each grayscale frame to
- enhance the contrast and improve the detection of subtle changes in brightness.

- The absolute difference between the two grayscale frames is then calculated to produce a difference image, which highlights
- areas where motion has occurred. A thresholding operation is applied to this image to produce a binary image, which is further
- processed with dilation to fill in any holes in the detected regions.

- The code then identifies contours in the dilated binary image and groups nearby contours that likely correspond to the same object
- in the video stream. Each group of contours is then used to draw a rectangle around the detected object in the original color frame.
- The rectangles are drawn in green if their width is less than 100 pixels (indicating that the detected object is likely too small
- to be of interest) and in red if their width is greater than 100 pixels (indicating that a significant object has been detected).
- Finally, the code displays the resulting frame in a window and waits for the user to exit the program.

- In summary, the code demonstrates how to use OpenCV to implement a simple motion detection algorithm that can be used in a variety
- of applications. It provides a good starting point for developing more sophisticated motion detection systems that can be tailored
- to specific use cases.