# TP 4 - Traitements de vidéos

**1 PARTIE 1 : FONCTIONS SIMPLES ET RAISONNEMENT + PRISES EN MAIN D’OPENCV**

**1.1 RÉCUPÉRATION DES FRAMES DE LA VIDÉO ET AFFICHAGE**

**1.1.1 UTILISATION D’UN VideoCapture OPENCV**

Selon le code:

filename = sys.argv[1] if len(sys.argv) > 1 else '../video.avi'

 cap = cv.VideoCapture(filename)

Il est possible de lire d’autres vidéo en joignant l’argument de commande: other video path. Par exemple, exécutez: *python autoroute.py autoroute2.avi*

**Question 1**

run = cap.isOpened()

# Making sure the capture has opened successfully

if not run:

print("capture opening has failed we cannot do anything :'(")

sys.exit()

......

ret, im = cap.read()

        cv.imshow("Original video", im)

Présentation réussie de la vidéo autoroute2.avi:



**1.2 EXTRACTION DES INFORMATIONS DE LA VIDÉO**

**Pour video.avi.**

**Question 2**

total\_frame= cap.get(cv.CAP\_PROP\_FRAME\_COUNT)

Le nombre d’images qui compose la séquence vidéo N = 614

**Question 3**

# le nombre d’images qui compose la séquence vidéo

    total\_frame= cap.get(cv.CAP\_PROP\_FRAME\_COUNT)

    # Frame rate de la vidéo.

    fps= cap.get(cv.CAP\_PROP\_FPS)

    # La durée d’une vidéo est le nombre total d’images divisé par le taux d’images en secondes.

    total\_time= total\_frame/fps

La durée de la séquence vidéo = 20.466666666666665 s

**Question 4**

# Get video wide and high (resolution)

    video\_width= cap.get(cv.CAP\_PROP\_FRAME\_WIDTH)

    video\_height= cap.get(cv.CAP\_PROP\_FRAME\_HEIGHT)

La résolution de la séquence vidéo: 384 x 288

**1.3 CONVERTIR LA SÉQUENCE VIDÉO EN NIVEAU DE GRIS**

**Question 5**

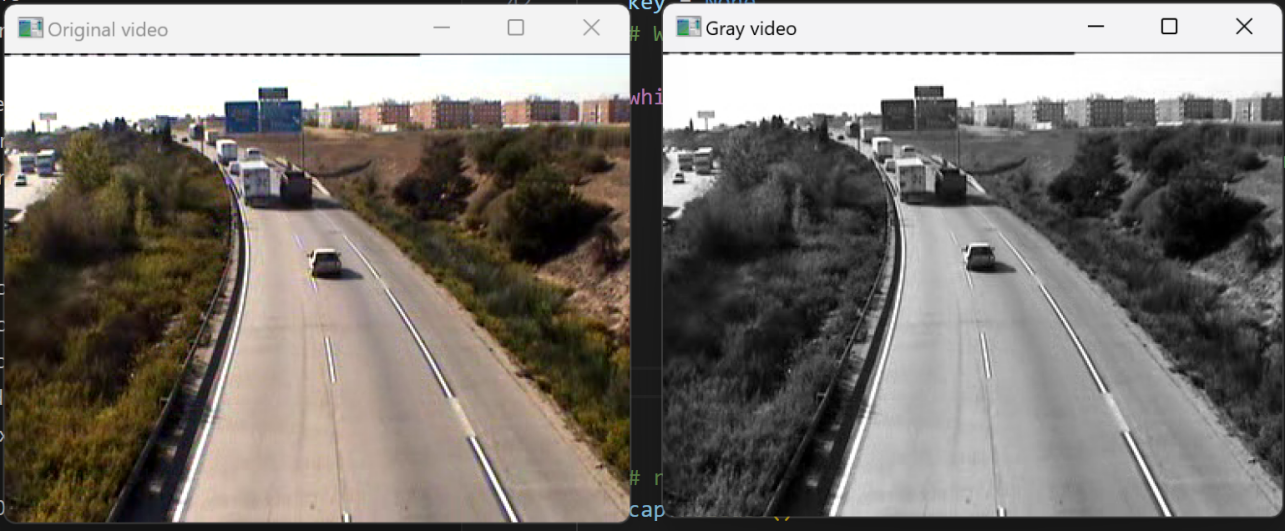
ret, im = cap.read()

        # Turning im into grayscale and storing it in imGray

        imGray = cv.cvtColor(im, cv.COLOR\_BGR2GRAY)

        cv.imshow("Original video", im)

        cv.imshow("Gray video", imGray)



**1.4 ENREGISTREMENT DES FRAMES DE LA VIDÉO DANS UN VECTEUR**

**Question 6**

index = 0

    # Waiting for the user to press ESCAPE before exiting the application

    while key != ESC\_KEY and key!= Q\_KEY:

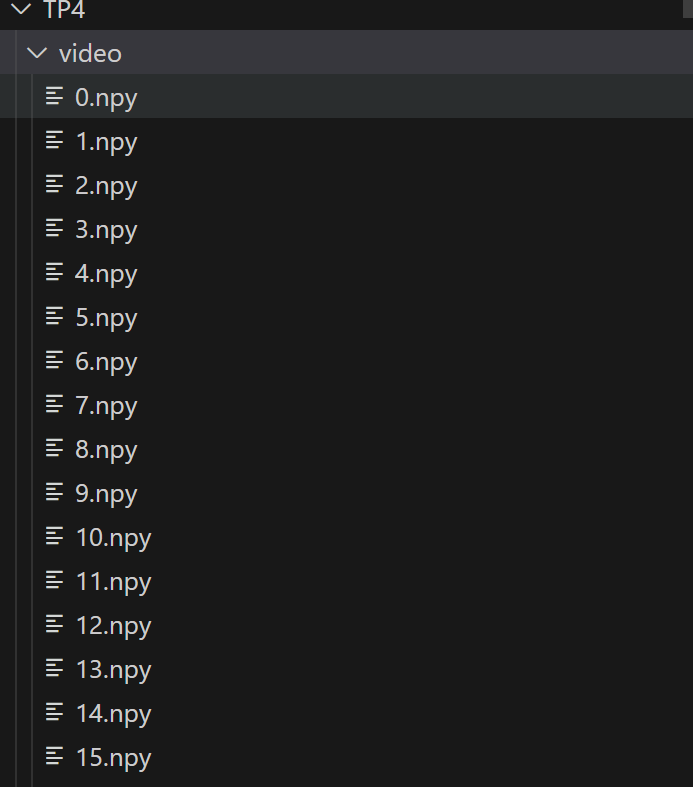
        ret, im = cap.read()

        save\_path = os.path.join(save\_folder, str(index) + ".npy")

        np.save(save\_path, im)

        index += 1

Nous avons mis le numpy.array de chaque image dans un fichier npy.



frames = []

    while key != ESC\_KEY and key!= Q\_KEY:

        ret, im = cap.read()

        frames.append(im)

Enregistre chaque image dans une liste qui peut être lue à partir de la liste si nécessaire. La fonction cap.read() n’est plus nécessaire.

**1.5 EXTRACTION DU FOND DE LA VIDÉO ET DE LA ROUTE**

**Question 7**

def moyenne\_images(images, M):

    # Initialiser une variable pour la somme des images

    somme\_images = np.zeros\_like(images[0], dtype=np.float32)

    # Somme des M premières images

    for i in range(M):

        img = images[i].astype(np.float32)  # Convertir l'image en float32 pour éviter un débordement

        somme\_images += img

    # Calculer la moyenne en divisant la somme par le nombre d'images

    moyenne = somme\_images / M

    # Convertir en type uint8 si nécessaire

    moyenne\_uint8 = np.array(moyenne, dtype=np.uint8)

    return moyenne\_uint8

Par le code:

M = 200

moyenne\_img = moyenne\_images(gray\_frames, M)

On a obtient:

****

**Question 8**

Une valeur trop petite de M peut donner une moyenne avec beaucoup de bruit, tandis qu'une valeur trop grande peut lisser excessivement l'image et entraîner une perte de détails. L’image ci-dessous montre une image moyenne avec M=200 à gauche et M=500 à droite. On peut voir qu’il n’y a pas beaucoup de différence dans cette vidéo.

****

**Question 9**

    moyenne\_img = moyenne\_images(gray\_frames, M)

    cv.imshow("Moyenne image", moyenne\_img)

    thresh, remove\_sky = cv.threshold(moyenne\_img, 210, 255, cv.THRESH\_TOZERO\_INV)

    cv.imshow("Remove sky", remove\_sky)

    thresh, mask = cv.threshold(remove\_sky, 100, 255, cv.THRESH\_BINARY)

    cv.imshow("bilinairy image", mask)

    kernel = np.ones((15,13), np.uint8)

    mask = cv.morphologyEx(mask, cv.MORPH\_OPEN, kernel)

    cv.imshow("morphologyEx Open", mask)

    kernel=cv.getStructuringElement(cv.MORPH\_ELLIPSE,(20,25))

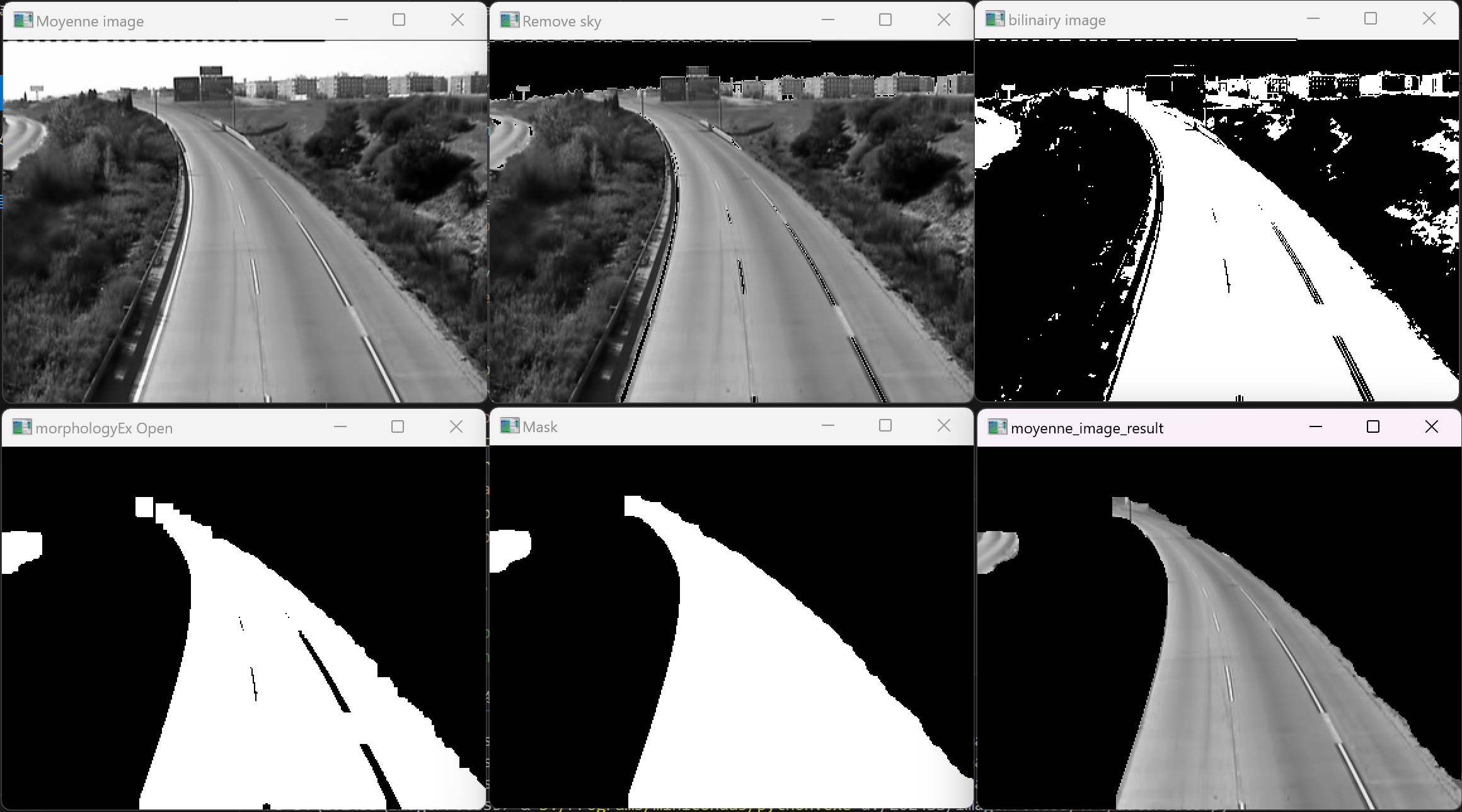
    mask = cv.morphologyEx(mask, cv.MORPH\_CLOSE, kernel)

    cv.imshow("Mask", mask)

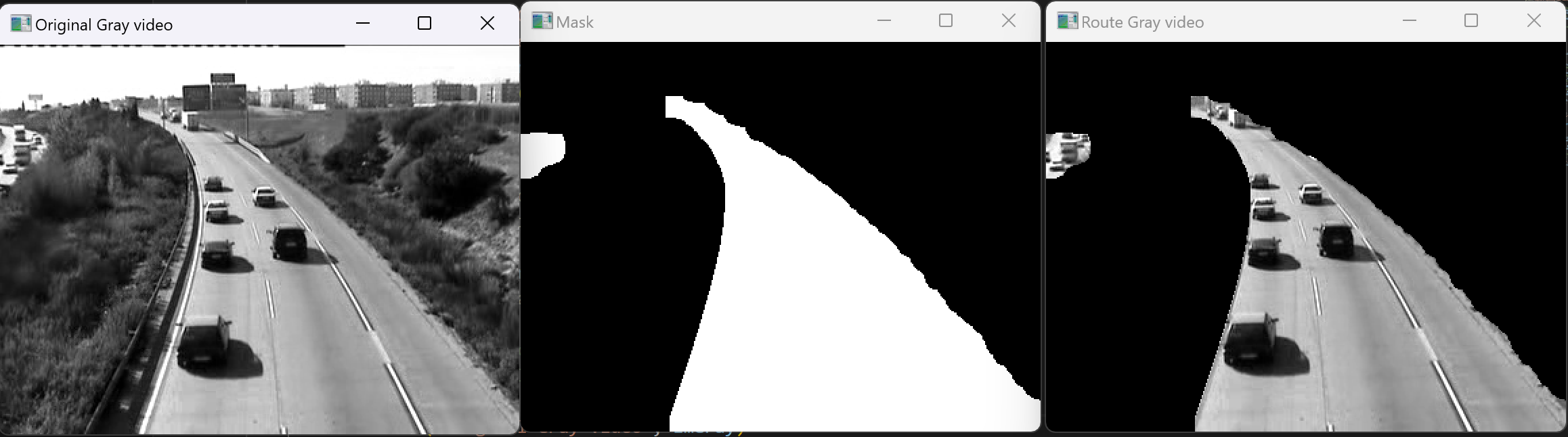
    result = cv.bitwise\_and(moyenne\_img, mask)

    cv.imshow("moyenne\_image\_result", result)

Traiter le dessin moyen pour obtenir la masque. Si l'image est directement transformée en une image binaire, le ciel est trop clair pour les opérations ultérieures. Le ciel est donc d’abord noirci pour obtenir un diagramme binaire. On peut constater que les images ne sont pas idéales, donc les opérations d'ouverture et de fermeture sont effectuées. Ensuite, nous utiliserons l'image résultante comme la masque.



        imGray = cv.bitwise\_and(imGray, mask)

De la droite vers la gauche : image issue de la vidéo en niveaux de gris. “Masque” de la route. Image extraite de la vidéo en niveaux de gris où l’on a gardé que les pixels non nuls du masque.

**Question 10**



        cars = cv.absdiff(imGray, moyenne\_img)

        cars = cv.bitwise\_and(cars, mask)

On utilise cv.absdiff() pour soustraire l'image à la valeur du pixel d'arrière - plan, laissant des pixels non nuls pour former la forme de la voiture.

**Question 11**

Pensez-vous que toute l’image soit nécessaire pour le comptage des voitures?

Je pense que non. En fait, seules les zones de l'image où les voitures sont susceptibles d'apparaître sont pertinentes pour le processus de détection. Cela permet non seulement de réduire la charge de calcul, mais aussi d'améliorer la précision en se concentrant sur les régions d'intérêt.

**3 PARTIE 3 : COMPTAGE DE VOITURES !**

**3.1 COMPOSANTES CONNEXES**

**Question 12**

Utilisez la méthode OpenCV connectedComponents afin de compter grossiè-

rement le nombre de voitures.

        cars = cv.absdiff(imGray, moyenne\_img)

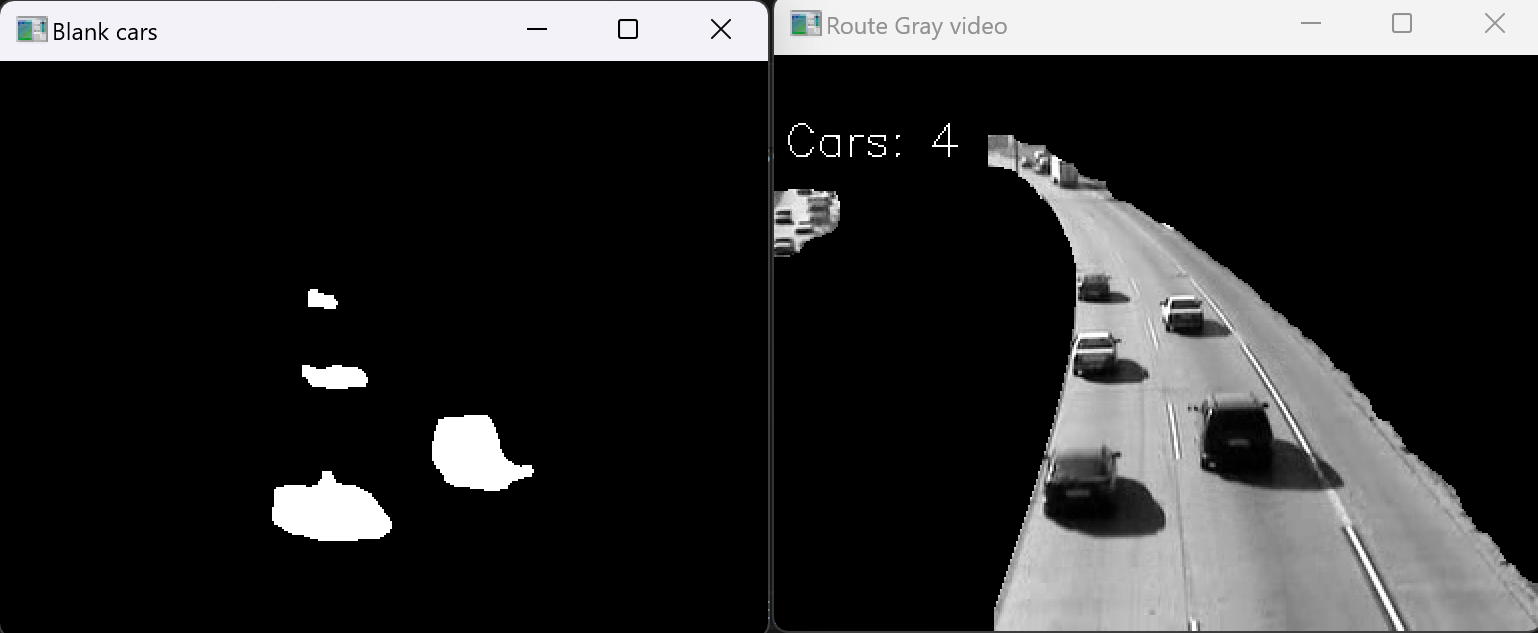
        cars = cv.bitwise\_and(cars, mask)

        \_, car\_mask = cv.threshold(cars, 100, 255, cv.THRESH\_BINARY)

        blank\_cars = cv.morphologyEx(car\_mask, cv.MORPH\_OPEN, kernel\_open)

        blank\_cars = cv.morphologyEx(blank\_cars, cv.MORPH\_CLOSE, kernel\_close)

        num\_cars, labels = cv.connectedComponents(blank\_cars)



**Question 13**

Les limites à l’utilisation de connectedComponents:

La méthode connectedcomponents ne prend en compte que les relations entre les pixels d'un graphique et ne peut pas faire de distinction entre les formes, les tailles, etc., peut identifier autre chose qu'une voiture, ce qui réduit la précision du comptage. Et quand il y a de l'ombre, le nombre de voitures ne peut pas être identifié avec précision.

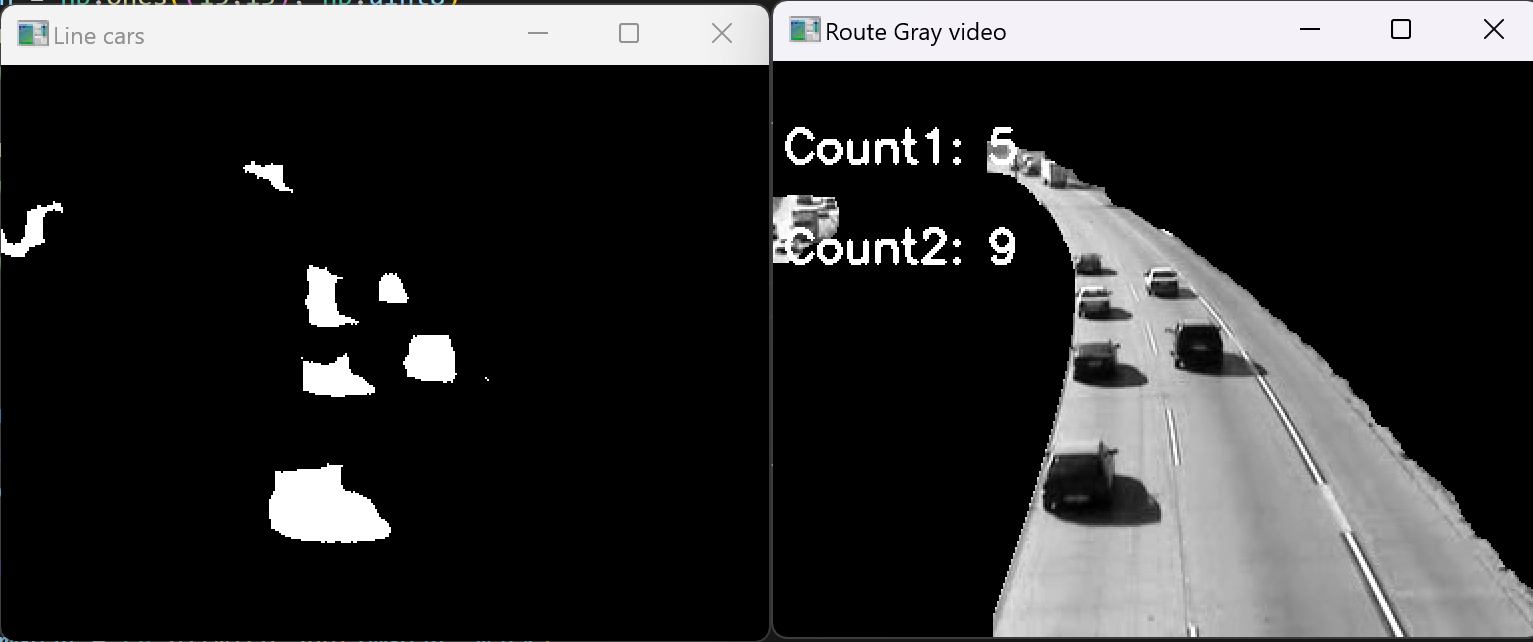
**Question 14**

kernel\_close = cv.getStructuringElement(cv.MORPH\_ELLIPSE,(10,15))

......

        line\_cars = cv.morphologyEx(car\_mask, cv.MORPH\_CLOSE, kernel\_close)

        contours, \_ = cv.findContours(line\_cars, cv.RETR\_EXTERNAL, cv.CHAIN\_APPROX\_SIMPLE)

        num\_cars2 = len(contours)

**Question 15**

# Count cars by other method

        edges = cv.Canny(cars, threshold1=30, threshold2=100)

        edges = cv.morphologyEx(edges, cv.MORPH\_CLOSE, kernel\_close)

        contours, \_ = cv.findContours(edges.copy(), cv.RETR\_EXTERNAL, cv.CHAIN\_APPROX\_SIMPLE)

        num\_cars3 = 0

        for contour in contours:

            # polygonal approximation

            epsilon = 0.02 \* cv.arcLength(contour, True)

            approx = cv.approxPolyDP(contour, epsilon, True)

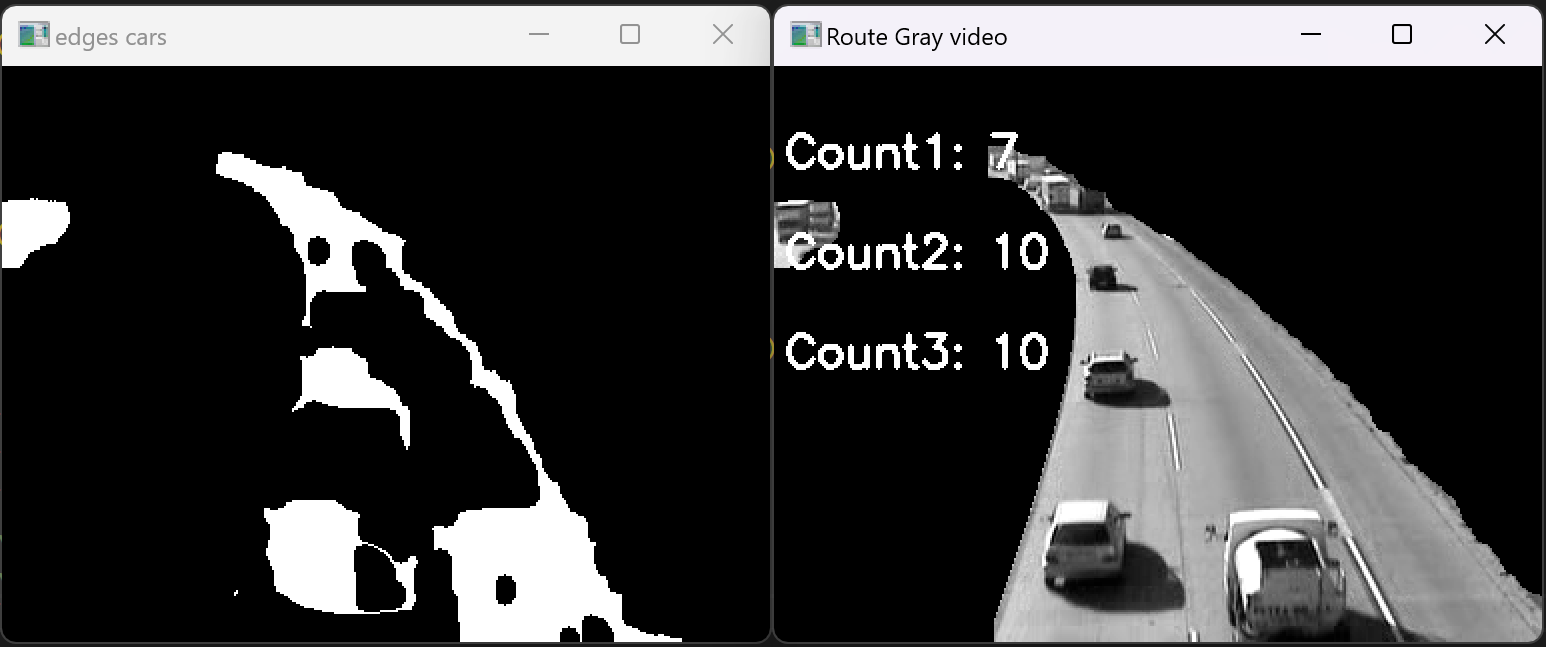
            # Check if it is close to a quadrilateral

            if len(approx) == 4:

                cv.drawContours(cars, [approx], 0, (0, 255, 0), 2)

                num\_cars3 += 1

**Question 16**



**4 PARTIE 4 : IMPLÉMENTATION DE FLOT OPTIQUE POUR LE SUIVI DE MOUVEMENT DANS UNE VIDÉO**

# Initialize for flow track

    feature\_params = dict(maxCorners=100,

                          qualityLevel=0.3,

                          minDistance=7,

                          blockSize=7)

    # Parameters for lucas kanade optical flow

    lk\_params = dict(winSize=(15, 15),

                     maxLevel=2,

                     criteria=(cv.TERM\_CRITERIA\_EPS | cv.TERM\_CRITERIA\_COUNT, 10, 0.03))

    # Create some random colors

    color = np.random.randint(0, 255, (100, 3))

    # Take first frame and find corners in it

    old\_gray = gray\_frames[0]

    p0 = cv.goodFeaturesToTrack(old\_gray, mask=rout\_mask, \*\*feature\_params)

    # Create a mask image for drawing purposes

    mask = np.zeros\_like(frames[0])

    while key != ESC\_KEY and key!= Q\_KEY:

        im = frames[current\_index]

        imGray = gray\_frames[current\_index]

        rout\_imGray = cv.bitwise\_and(imGray, rout\_mask)

        # calculate optical flow

        p1, st, err = cv.calcOpticalFlowPyrLK(old\_gray, rout\_imGray, p0, None, \*\*lk\_params)

        # Select good points

        if p1 is not None:

            good\_new = p1[st == 1]

            good\_old = p0[st == 1]

        # draw the tracks

        for i, (new, old) in enumerate(zip(good\_new, good\_old)):

            a, b = new.ravel()

            c, d = old.ravel()

            mask = cv.line(mask, (int(a), int(b)), (int(c), int(d)), color[i].tolist(), 2)

            im = cv.circle(im, (int(a), int(b)), 5, color[i].tolist(), -1)

            img = cv.add(im, mask)

            cv.imshow("flow", img)

            # Now update the previous frame and previous points

            old\_gray = rout\_imGray.copy()

            p0 = good\_new.reshape(-1, 1, 2)

