TP 4 - Traitements de vidéos

0 OBJECTIFS ET ÉVALUATION DES TPS

0.1 CONTEXTE ET ATTENTES

Au cours de ce TP, vous allez travailler sur une séquence vidéo acquise depuis une caméra d'autoroute.

L'idée est de se placer dans un cadre d'application, où par exemple une société d'exploitation routière souhaite savoir la fréquentation d'un de ces tronçons d'autoroute (on peut imaginer de nombreuses applications derrière).

L'objectif sera dans un premier temps de faire une extraction du fond de la vidéo et de détecter les voitures en mouvement, puis dans un second temps de compter le nombre de véhicules qui transitent sur cette portion durant le laps de temps de la vidéo.

0.2 RESSOURCES FOURNIES

Pour vous aider à démarrer le TP, nous vous fournissons les éléments suivants :

- un fichier `main.cpp` ou `autoroute.py` contenant un squelette **minimaliste** de code OpenCV ;
- un fichier `CMakeLists.txt` vous permettant de générer le `Makefile` et donc de compiler votre application en C++ ;
- un fichier `video.avi` de la séquence prise par une caméra d'autoroute ;

0.3 TRAVAIL À RENDRE

Vous présenterez à la fin du tp :

— un compte rendu détaillé regroupant vos démarches et réflexions sur la solution à apporter ;
— les tests et résultats intermédiaires obtenus pour arriver à la solution ;
— votre code principal et fonctions commentées.

Lorsqu'il est demandé une explication dans le texte de ce TP, vous devez détailler cette explication dans votre compte-rendu (en mentionnant la question et/ou le numéro de section).

1 Partie 1 : Fonctions simples et raisonnement + prises en main d'OpenCV

Cette partie décrit les premières étapes à effectuer, sans forcément rentrer dans les détails. C'est volontaire pour vous faire travailler sur OpenCV et vous forcer à utiliser la documentation. Toutefois, le sujet vous indiquera une liste des fonctions OpenCV qui sont potentiellement intéressantes dans ce TP (et dans chaque section).
Le but est également de vous laisser libre de proposer vos solutions (car il n'y a bien évidemment pas qu'une seule solution à ce TP) pour arriver à une solution (même approchée).

1.1 Récupération des frames de la vidéo et Affichage

1.1.1 Utilisation d'un `VideoCapture` OpenCV

Regardez le squelette fourni (dans le fichier `main.cpp` ou `autoroute.py`). Expliquez comment lancer le programme si vous souhaiter utilisez une vidéo dont le nom n'est pas `video.avi`.
La classe OpenCV utilisée pour lire des vidéos (fichier ou flux) est la classe `VideoCapture`. Cette dernière fournit des fonctions permettant d'ouvrir, de lire, de récupérer des informations de la vidéo ou même de modifier certaines informations.
Comme dans la suite de ce TP, nous vous invitions à aller regarder la documentation en ligne ici :
`https://docs.opencv.org/4.x/d8/dfe/classcv_1_1VideoCapture.html`
En particulier, regardez les fonctions :

— `open`
— `isOpened`
— `read`

Nous allons utiliser la méthode `pollKey` au lieu de `waitKey` du package `highgui` (resp. `cv2`) d'OpenCV que nous avions utilisé dans le tp précédent :
`https://docs.opencv.org/4.x/d7/dfc/group__highgui.html`
Expliquez la différence entre ces deux méthodes.

Question 1 : Lecture de la vidéo Modifiez le squelette fourni pour que vous puissiez lire la vidéo fournie. Pour y arriver, il se peut que vous deviez commenter des lignes de code fournies. Illustrez le bon fonctionnement de votre programme.
N. B. À ce moment du programme si votre exécutable plante une fois la vidéo terminée ce n'est pas un problème.

1.2 Extraction des informations de la vidéo.

On considère que la séquence vidéo fournie dure t secondes, avec n images par secondes, ce qui constitue une séquence de N images au total.
En vous basant sur la documentation de la classe `VideoCapture`, en particulier aux méthodes :

- `get`
- `set`

Vous répondrez aux questions suivantes.

Question 2 : Quel est le nombre d'images qui compose la séquence vidéo ? Précisez dans votre rapport le code utilisé.

$$N = ?$$

Question 3 : Quelle est la durée de la séquence vidéo ? Précisez dans votre rapport le code utilisé et comment vous êtes arrivés à ce résultat.

Question 4 : Quelle est la résolution de la séquence vidéo ? Précisez dans votre rapport le code utilisé.

1.3 Convertir la séquence vidéo en niveau de gris

Dans la suite de ce TP, nous allons principalement travailler sur des images en niveaux de gris pour pouvoir appliquer certains traitements inapplicables sur les images couleurs. Ainsi, nous allons devoir transformer les images couleurs en niveaux de gris.
Pour vous aider dans cette tâche, nous vous conseillons d'aller voir la documentation de la fonction `cvtColor` :
`https://docs.opencv.org/4.x/d8/d01/group__imgproc__color__conversions.html`

Question 5 : Modifiez le code fourni afin de pouvoir afficher dans une fenêtre nommée "Gray video" les images de la vidéo en niveaux de gris. Illustrez le code modifié et le résultat en faisant une capture d'écran des deux images côte à côte (cf. Figure 1.1).

1.4 Enregistrement des frames de la vidéo dans un vecteur

Afin de pouvoir travailler plus facilement sur notre vidéo, nous nous proposons de lire une première fois la vidéo et de stocker chaque image (de type `cv::Mat` en c++ et numpy.array en python) dans un vecteur (`std::vector` en c++).

Question 6 : Modifiez le code fourni afin de stocker dans un vecteur chacune des images composant la vidéo. Ainsi, nous pourrons ensuite parcourir ce vecteur avec plus de liberté que si nous utilisions directement la variable de type `VideoCapture`. Illustrez le code mis en place.

FIGURE 1.1 – Vidéo affichée en couleur et en niveaux de gris.

1.5 EXTRACTION DU FOND DE LA VIDÉO ET DE LA ROUTE

Dans cette partie, nous nous proposons d'essayer d'extraire le fond (*background*) de la vidéo. Le fond d'une vidéo est caractérisé par le fait que les pixels le composant sont "très souvent" à l'image et ne sont pas énormément modifiés.
Un des moyens d'extraire le fond d'une vidéo est de calculer la moyenne de la vidéo sur un certain nombre d'images consécutives. On peut au contraire (et dans notre cas de figure bien précis) supposer que les endroits de l'image où il y a beaucoup de variations d'une image à l'autre correspondent à la route!

1.5.1 PRÉCISIONS OPENCV SUR LE TYPE `cv::Mat` EN C++

Les images OpenCV son stockées dans des variables de type `cv::Mat`. La documentation de cette classe est disponible ici :
`https://docs.opencv.org/4.x/d3/d63/classcv_1_1Mat.html`
Chaque matrice possède un nombre de ligne (`rows`) et de colonnes (`cols`), attention toutefois, le nombre de lignes représente la hauteur de l'image alors que le nombre de colonnes représente la largeur!

Chaque `Mat` possède un `type de données` qui correspond à ce qu'il y a de stocké dans la matrice. Les types OpenCV ont une convention de nommage :
`CV_<bit-depth>U|S|FC(<number_of_channels>)` où `U` représente un type de données entier non signé, `S` représente un type entier signé et `F` représente un type flottant.

Notez toutefois une information importante :

— en fonction du type d'image (couleur ou niveau de gris), les pixels des `cv::Mat` ne sont pas du même type !
  — une image en niveau de gris stockera des pixels de type `uchar` (c'est-à-dire `unsigned char`) **sur un seul canal**. Cela correspond au `type` OpenCV `CV_8UC1`
  — une image en couleur stockera des pixels de type `uchar` (c'est-à-dire `unsigned char`) **sur trois canaux** Cela correspond donc au type OpenCV `CV_8UC3`.
— si vous souhaitez un peu plus d'exemples, vous pouvez aller à l'adresse suivante : `https://docs.opencv.org/4.x/d6/d6d/tutorial_mat_the_basic_image_container.html`

Voici un exemple d'accès à un pixel pour une image (`m_im`) en niveau de gris (donc de type `CV_8UC1`) et pour une image en couleurs (donc de type `CV_8UC3`) :

```
int grayVal = m_im.at<uchar>(0,0);
cv::Vec3b col = m_im.at<Vec3b>(0,0);
int greenCol = col[1];
```

1.5.2 Précisions sur les tableaux d'images en Python

Les tableaux d'images en Python sont stockées sous forme de numpy.array.
Voici un exemple d'accès à un pixel pour une image (`m_im`) en niveau de gris (donc de type `uint8`) et pour une image en couleurs (donc de type `ndarray`)

```
m_im[0][0] # renvoie la valeur du niveau de gris du pixel
m_imc[0][0] # renvoie un tableau [B G R] - Blue - Green - Red
# pour trasformer un tableau im en type uint8
im = np.array(im ,dtype=np.uint8)
```

1.5.3 Calculer l'image moyenne de la vidéo sur M images

Dans un premier temps, nous vous demandons de calculer une moyenne sur M images consécutives de la vidéo. Vous trouverez dans la Figure 1.2 un exemple de résultat obtenu sur la vidéo fournie en calculant la moyenne avec $M = 200$ images.
Il est possible de calculer la moyenne d'images en OpenCV de plusieurs manières différentes :
— en faisant des calculs pixel à pixel (cf. premier TP de VSION et l'utilisation de la méthode `at`) ;
— en faisant des calculs sur l'image complète et en utilisant la surcharge d'opérateurs OpenCV.
Les deux solutions aboutissent à des résultats équivalents, vous pouvez donc choisir celle que vous souhaitez.

Question 7 : Écrivez une fonction calculant la moyenne de M images consécutives. Cette fonction doit prendre en paramètre le nombre d'images M sur lesquelles la moyenne doit être calculée. La méthode doit retourner l'image moyennée résultat.

FIGURE 1.2 – Moyenne sur $M = 200$ images de la vidéo.

Dans votre rapport, précisez les éventuelles difficultés rencontrées. Détaillez comment vous les avez résolues et pourquoi elles sont apparues.

Question 8 : Illustrez le calcul de la moyenne avec différentes valeurs de M. Le choix de la valeur de M est-il important ? Pensez-vous qu'il soit simple de déterminer une bonne valeur de M ?

1.5.4 EXTRACTION DE LA ROUTE DE LA SÉQUENCE VIDÉO

Nous souhaitons maintenant être capables d'isoler dans une séquence d'images la route du fond de l'image. Cela nous permettra de nous concentrer sur la partie "intéressante" de la vidéo, là où passent les voitures. Pour arriver à isoler la route du reste de la vidéo, il vous sera probablement nécessaire de procéder par étapes.

En particulier, il vous sera probablement nécessaire de devoir modifier une image à partir d'un "masque" qui vous permet d'isoler la partie de l'image que vous souhaitez, voir Figure 1.3.

Utilisation de "masque" en OpenCV OpenCV vous permet de facilement manipuler des masques pour extraire des parties d'une image, cela est par exemple faisable en faisant appel à la méthode `copyTo` à laquelle on fournit un deuxième paramètre, voir :
`https://docs.opencv.org/4.x/d3/d63/classcv_1_1Mat.html`
Il est donc possible d'extraire une partie de l'image `img`, qui correspond à tous les pixels non nuls d'une image `mask` et de copier le résultat dans une troisième image `res`.
En C++ :

```
img.copyTo(res,mask);
```

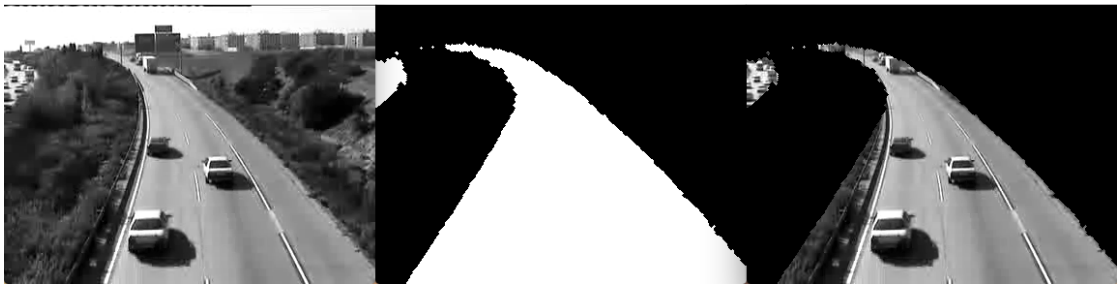

FIGURE 1.3 – De la droite vers la gauche : image issue de la vidéo en niveaux de gris. “Masque” de la route. Image extraite de la vidéo en niveaux de gris où l'on a gardé que les pixels non nuls du masque.

En Python :

```
res = cv.bitwise_and(image, mask)
```

Notez que l'image de masque `mask` **doit être** une image de type `CV_8U` en C++ et `np.uint8` en Python.

Question 9 : Proposez une méthode pour détecter la route dans une séquence d'images vidéo. Détaillez les étapes que vous appliquez pour arriver à détecter la route dans une séquence de M images. Illustrez le résultat obtenu.

1.6 DÉTECTION DES VOITURES

Une fois la route isolée dans la vidéo, nous pouvons essayer de détecter les voitures dans la vidéo en temps réel. Á ce stade, vous devez avoir une image dans laquelle seule la route est visible (cf. image de droite de la Figure 1.3).

Il vous reste à être capables d'isoler les voitures puis à les compter. Nous souhaitons donc être capables d'isoler les voitures dans l'image, puis de les compter.

1.6.1 ISOLATION DES VOITURES

Question 10 : En vous basant sur ce que vous avez fait jusqu'à présent, proposez une méthode pour isoler les voitures dans la séquence vidéo. Illustrez le résultat obtenu.

1.6.2 SÉLECTION D'UNE ZONE DE L'IMAGE ?

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

Question 11 : Si oui, passez à la question suivante. Sinon, détaillez la manière dont vous sélectionnez la zone de l'image qui vous intéresse. Illustrez le résultat obtenu.

2 Partie 2 : En vrac : des fonctions OpenCV qui peuvent vous servir (ou pas) !

Dans cette partie, nous vous listons un ensemble de fonctions OpenCV qui peuvent vous être utiles tout au long de ce TP.

Attention : **ces fonctions ne sont pas données dans un ordre spécifique!** Ne pensez pas que vous allez forcément devoir les utiliser les unes à la suite des autres.

Vous trouverez également une présentation succincte, mais intéressante de quelques fonctions OpenCV ici :
`http://www.cse.unr.edu/~bebis/CS485/Lectures/Intro_OpenCV.pdf`

2.1 Remettre à zéro une image

2.1.1 En C++

En C++ la fonction `cv::Mat::zeros` vous permet de vous assure qu'une matrice de type `cv::Mat` est "vide". Les arguments de cette fonction sont :

- `int rows` : nombre de lignes de votre matrice
- `int cols` : nombre de colonnes de votre matrice
- `int type` : type de données stockées dans la matrice

Un exemple d'utilisation :

```
cv::Mat maMatConvertie;
cv::Mat img;
...
// on suppose img non vide et déjà créée
maMatConvertie = cv::Mat::zeros(img.rows,img.cols,CV_8UC1);
```

2.1.2 En Python

En Python, il suffit de remettre à 0 le tableau numpy qui stoke les informations de l'image :

```
img = np.zeros((img.shape[0],img.shape[1],3), dtype=np.uint8)
```

2.2 Convertir une image d'un type dans un autre

2.2.1 En C++

La fonction : `cv::convertTo` permet de convertir des données d'un type vers un autre. Un exemple d'utilisation :

```
cv::Mat maMatConvertie;
...
// on suppose img de type CV_32FC1 par exemple
img.convertTo(maMatConvertie,CV_8U);
```

2.2.2 En Python

Pour changer le type de données d'une image OpenCv en Python, nous pouvons utiliser la fonction .astype de numpy. Par exemple, si img est une image OpenCv et que je veux la rendre de type "CV_8UC1", il me suffit de taper :

```
img = img.astype('uint8')
```

2.3 Seuillage en OpenCV

OpenCV propose deux grandes fonctions permettant de faire du seuillage d'images :

- `cv::threshold` (resp. cv2.threshold) : pour un seuillage "simple"
- `cv::adaptiveThreshold` (resp. cv2.adaptiveThreshold) : pour un seuillage adaptatif

Ces deux méthodes ont besoin d'un certain nombre de paramètres, dont certains sont expliqués ici pour la version c++ :
https://docs.opencv.org/4.5.5/db/d8e/tutorial_threshold.html
et ici pour la version python :
https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html
Nous vous conseillons d'aller jeter un œil à cette page pour bien comprendre ce que fait le seuillage.

2.4 Opérations morphologiques

OpenCV fournit une méthode (`morphologyEx`) permettant de réaliser des opérations de morphologie mathématique (voir vos cours).
Une page détaillant les différentes opérations en C++ est disponible à l'adresse suivante :
https://docs.opencv.org/4.5.5/db/df6/tutorial_erosion_dilatation.html et https://docs.opencv.org/4.5.5/d3/dbe/tutorial_opening_closing_hats.html
La documentation de la méthode `morphologyEx` se trouve ici :
https://docs.opencv.org/4.5.5/d4/d86/group__imgproc__filter.html#ga67493776e3ad1a3df638838
En python vous avez un tutoriei ici : https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html

2.5 Composantes connexes

OpenCV nous permet de calculer les composantes connexes dans une image. Plusieurs méthodes sont offertes pour se faire :

- depuis la version 3.0, une méthode dédiée `connectedComponents` est proposée qui retourne le nombre de composantes connexes détectées en fonction des paramètres fournis

— toutefois, si vous souhaitez pouvoir faire d'autres choses vous pouvez utiliser les méthodes `findContours`, potentiellement combinées avec `approxPolyDP`, `contourArea`

2.5.1 Connected Components

Vous trouverez plus bas un exemple d'utilisation de la méthode `connectedComponents`.

Exemple en C++ Cette méthode nous retourne le nombre de composantes connexes détectées dans l'image. Les paramètres de cette méthode sont les suivants en C++ :

— `image` : l'image à labéliser
— `labels` : l'image de destination des labels
— `connectivity` : la connectivité utilisée pour détecter les composantes connexes (cette valeur vaut soit 4 soit 8). Voir ici pour quelques détails `https://en.wikipedia.org/wiki/Connected-component_labeling` ou ici `http://www.imageprocessingplace.com/downloads_V3/root_downloads/tutorials/contour_tracing_Abeer_George_Ghuneim/connectivity.html`
— `type` : le type de l'image de destination (soit `CV_32S` soit `CV_16U`)

Plus d'informations ici :
http://docs.opencv.org/3.0-beta/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html#connectedcomponents

```
// On suppose que 'img' est une image binaire
cv::Mat testCC(img.size(), CV_32S);
// on compte le nombre de composantes connexes
int nLabels = connectedComponents(cut, testCC, 8);

// Les lignes suivantes sont pour l'affichage de ces composantes connexes !
std::vector<Vec3b> colors(nLabels);
colors[0] = Vec3b(0, 0, 0);       //background color
// on genere des couleurs aleatoires
for (int label = 1; label < nLabels; ++label) {
        colors[label] = Vec3b((rand() & 255), (rand() & 255), (rand() & 255));
}
// on affiche dans l'image 'dstCC' les composantes connexes en couleur
Mat dstCC(img.size(), CV_8UC3);
for (int r = 0; r < dstCC.rows; ++r) {
        for (int c = 0; c < dstCC.cols; ++c) {
                int label = testCC.at<int>(r, c);
                Vec3b &pixel = dstCC.at<Vec3b>(r, c);
                pixel = colors[label];
                }
}

// on affiche l'image resultat
```

```
imshow("Connected_comps", dstCC );
```

Exemple en Python Le code ci-dessous montre un exemple de l'utilisation de `connectedComponents` en Python

```
import cv2
import numpy as np

img = cv2.imread('Baboon.jpg', cv2.IMREAD_GRAYSCALE)
img = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)[1]  # ensure binary
num_labels, labels_im = cv2.connectedComponents(img)

def imshow_components(labels):
        # Map component labels to hue val
        label_hue = np.uint8(179*labels/np.max(labels))
        blank_ch = 255*np.ones_like(label_hue)
        labeled_img = cv2.merge([label_hue, blank_ch, blank_ch])

        # cvt to BGR for display
        labeled_img = cv2.cvtColor(labeled_img, cv2.COLOR_HSV2BGR)

        # set bg label to black
        labeled_img[label_hue==0] = 0

        cv2.imshow('labeled.png', labeled_img)
        cv2.waitKey()

imshow_components(labels_im)
```

2.5.2 Détection de contours

Une alternative possible et qui permet **plus de contrôle et de faire éventuellement plus de calculs sur les composantes connexes** consiste à combiner plusieurs appels de fonction OpenCV différentes.
Ces opérations peuvent être les suivantes en C++ :

- `findContours` : méthode de détection de contours dans une image, voir détails ici http://docs.opencv.org/3.0-beta/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html#findcontours
- `approxPolyDP` : permet d'approximer une courbe polygonale par une autre courbe polygonale plus simple étant donné une précision à ne pas dépasser. Plus d'informations ici http://docs.opencv.org/3.0-beta/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html#approxpolydp

— contourArea : permet de calculer l'aire d'un contour passé en paramètre. Voir plus d'informations ici http://docs.opencv.org/3.0-beta/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html#contourarea
— drawContour : permet de dessiner des contours pour les visualiser dans une image. Voir ici : http://docs.opencv.org/3.0-beta/modules/imgproc/doc/drawing_functions.html#drawcontours

Voici un exemple d'utilisation de contours et de dessin :

```
// On suppose l'image 'img' étant une image binaire

// L'image 'output' sera utilisée pour le dessin des contours
cv::Mat output = cv::Mat::zeros(img.size(), CV_8UC3);

/// Find contours
vector<vector<Point> > contours;
vector<Vec4i> hierarchy;
RNG rng(12345);

// détection de contours dans 'img'
findContours( img, contours, hierarchy,
CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, Point(0, 0) );

// Draw contours
Mat drawing = Mat::zeros( im.size(), CV_8UC3 );
for( int i = 0; i< contours.size(); i++ )
{
Scalar color = Scalar( rng.uniform(0, 255), rng.uniform(0,255), rng.uniform(0,255) );
drawContours(output, contours_poly, i, color, 1, 8, vector<Vec4i>(), 0, Point());
}
```

Vous trouverez ici : https://docs.opencv.org/4.x/d4/d73/tutorial_py_contours_begin.html des exemples d'utilisation de fonctions équivalentes en Python.

3 Partie 3 : Comptage de voitures !

En utilisant les fonctions présentées dans la partie précédente, nous allons reprendre notre TP de comptage. Nous voilà arrivés à l'issue de la Question 1.6.2 avec une image prête au comptage des voitures.

3.1 Composantes Connexes

Nous vous proposons dans un premier temps d'utiliser les notions d'étiquetage en composantes connexes pour avoir une première estimation du nombre de voitures dans la vidéo.

Question 12 : Utilisez la méthode OpenCV `connectedComponents` afin de compter grossièrement le nombre de voitures. Illustrez le résultat obtenu sur une frame de votre choix avec le nombre de voitures correspondant.

Question 13 : Détaillez les limites à l'utilisation de cette méthode. Quel(s) problème(s) voyez-vous à l'utilisation de cette méthode pour notre problème ?

Question 14 : Utilisez cette fois les méthodes d'OpenCV `findContours` et les méthodes associées présentées plus haut. Illustrez le résultat obtenu sur la même frame qu'à la Question 14. Détaillez bien les traitements additionnels que vous avez rajoutés à l'appel à `findContours`. Comparez vos résultats à ceux obtenus à la Question 14 et expliquez pourquoi ils sont différents.

3.2 Votre propre solution

Question 15 : Proposez une autre solution de votre choix pour le comptage de voiture. Détaillez le raisonnement que vous avez choisi.

Question 16 : Illustrez le résultat obtenu sur la même frame qu'à la Question 14.

3.3 Changeons de vidéo :)

Dans cette section, nous nous proposons de changer la vidéo prise en entrée afin de constater comment se comporte notre programme.

Question 17 : Utilisez la vidéo autoroute2.mp4. Votre méthode fonctionne-t-elle toujours ? Listez les raisons des problèmes que vous constatez. Si vous ne constatez pas de problème, expliquez pourquoi cela fonctionne.

Question 18 : Utilisez la vidéo autoroute3.mp4. Votre méthode fonctionne-t-elle toujours ? Listez les raisons des problèmes que vous constatez. Si vous ne constatez pas de problème, expliquez pourquoi cela fonctionne.

4 Partie 4 : Implémentation de flot optique pour le suivi de mouvement dans une vidéo

Maintenant que nous avons réussi à détecter des voitures dans un flux vidéo, nous pouvons nous intéresser à un autre problème : celui du suivi du mouvement de ces voitures. Une des méthodes permettant d'évaluer les mouvements d'objets dans une vidéo est le **flot optique** (ou **optical flow**) :
`https://en.wikipedia.org/wiki/Optical_flow`

Une des méthodes permettant de calculer le flot optique dans un flux vidéo est la méthode dite de **Lucas-Kanade**, du nom de ses deux inventeurs :
`https://en.wikipedia.org/wiki/Lucas-Kanade_method`
En particulier, OpenCV propose une méthode nommée `calcOpticalFlowPyrLK` qui est une implémentation dite "pyramidale" de cette méthode L-K. L'article décrivant cette implémentation est disponible à l'adresse suivante :
`http://robots.stanford.edu/cs223b04/algo_tracking.pdf`

Et la documentation de la méthode `calcOpticalFlowPyrLK` est disponible ici :
`https://docs.opencv.org/4.5.5/dc/d6b/group__video__track.html#ga473e4b886d0bcc6b65831eb88e`
ainsi qu'un **tutoriel sur le flot optique** en C++ et en Python qui peut vous servir de base de code :
`https://docs.opencv.org/4.x/d4/dee/tutorial_optical_flow.html`

Cette méthode permet d'estimer les mouvements dans une région de l'image (une fenêtre autour du pixel courant traité) en utilisant les gradients calculés dans cette "fenêtre". Dans notre cas, l'idée de l'estimation de mouvement est de trouver la transformation a appliquer entre deux ensembles de points d'intérêt POI_{prev} et POI_{curr} en se basant sur une taille de fenêtre de recherche. Cette fenêtre correspond à la zone où l'on va rechercher chacun des points de l'ensemble POI_{prev} pour l'apparier avec un des points de POI_{curr}. Afin d'appliquer cet algorithme, nous devons donc être capables de détecter des points d'intérêt dans une image.

Nous vous fournissons une implémentation minimale d'un calcul de flot optique avec la méthode pyramidale de Lucas-Kanade, dont voici l'algorithme en pseudo-code :

1. création d'un objet de type `cv::Ptr<cv::FeatureDetector>` qui va nous permettre de détecter les points d'intérêt dans l'image ;
2. appel d'une méthode appliquant le calcul du flot optique pour chaque frame ;
3. récupération de la frame courante et de la précédente (stockées au début de l'algorithme) ;
4. détection des points d'intérêt et stockage dans un vecteur de points d'intérêts `prevPts` ;
5. calcul du flot optique ;
6. stockage le cas échéant des points détectés dans `trackedPts` et dessin à l'écran du flot optique ;
7. stockage des points de `trackedPts` comme nouveaux points "précédents" (`prevPts`).

Pour rappel voici une page détaillant le fonctionnement des principaux détecteurs de points d'intérêt en OpenCV en Python :
`https://docs.opencv.org/3.4/db/d27/tutorial_py_table_of_contents_feature2d.html`

Question 19 : A partir du code du tutoriel, faite votre propre suivi des voitures sur la vidéo video.avi Illustrez le résultat.

Question 20 : En vous rappelant que nous souhaitons détecter le mouvement des voitures dans une vidéo. Quel problème voyez-vous avec la détection de points d'intérêts dans cette vidéo ? En vous basant de ce que vous avez **déjà fait** sur les premières parties de ce TP proposez une solution à ce problème.

Question 21 : Décrivez le comportement d'un point d'intérêt détecté sur un véhicule et suivi ("tracké") par l'algorithme de flot optique. Quel problème cela pose-t-il avec l'implémentation fournie ?

Question 22 : Proposez une solution afin de régler le problème évoqué à la question précédente. Vous joindrez votre code à la réponse. Illustrez le bon fonctionnement sur la vidéo `video.avi`.

Question 23 : Testez maintenant votre solution sur la vidéo `autoroute3.mp4`. À partir de cette vidéo, décrivez à quoi pourrait servir un mécanisme de flot optique sur une caméra autoroutière ?