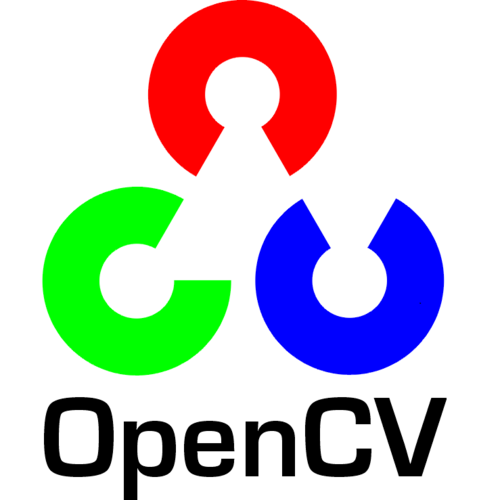


Université de Tours – Département informatique – Antenne de Blois

Année universitaire 2019-2020

Reconnaissance et forme d’analyse d’image

Rapport TP1 : De l’image aux descripteurs



Réalisé par : Ridha Kchouk Supérvisé par : Donatello Conte

Table des matières

[Introduction : 2](#_Toc32722937)

[1.1 Exercice 1 : Prise en main de Python + OpenCV 5](#_Toc32722938)

[1.1.1 Question 1 5](#_Toc32722939)

[1.2 Exercice 2 : Filtrage, convolution, détection de contours 6](#_Toc32722940)

[1.2.1 Question 1 6](#_Toc32722941)

[1.2.2 Question 2 10](#_Toc32722942)

[1.3 Exercice 3 : Images binaires et opérations entre images 13](#_Toc32722943)

[1.3.1 Question 1 13](#_Toc32722944)

[1.3.2 Question 2 19](#_Toc32722945)

[1.4 Exercice 4 : Détection de changement de plan et résumé automatique de vidéo 21](#_Toc32722946)

[1.4.1 Question 1 : Changement de plan 21](#_Toc32722947)

[1.4.2 Question 2 : Création d’images résumés de plan 23](#_Toc32722948)

[1.5 Exercice 5 : Calcul de descripteurs SIFT, appariement de descripteurs et résumé automatique de vidéo 24](#_Toc32722949)

[1.5.1 Question 1 : Création d’images résumés de plan 25](#_Toc32722950)

[1.5.2 Question 2 : Calcul de descripteurs SIFT et appariement 27](#_Toc32722951)

[1.6 Réponse aux questions sur la fin du TP 28](#_Toc32722952)

Table des figures

[Figure 1 : Exercice 1 code commenté 4](#_Toc32727032)

[Figure 2 : Résultat après exécution du code 5](https://d.docs.live.net/9d6588607331eafb/Travail/UFR%202018-2019%20(M1)/RFAI/New%20Year/Rapport/Rapport%20%20TP%201%20-%20De%20l'image%20aux%20descripteurs.docx#_Toc32727033)

[Figure 3 : Code source fonction imgproc 6](#_Toc32727034)

[Figure 4 : Application des filtres 7](#_Toc32727035)

[Figure 5 : Flou gaussien 7](https://d.docs.live.net/9d6588607331eafb/Travail/UFR%202018-2019%20(M1)/RFAI/New%20Year/Rapport/Rapport%20%20TP%201%20-%20De%20l'image%20aux%20descripteurs.docx#_Toc32727036)

[Figure 6 : Image original, aucun filtre 7](#_Toc32727037)

[Figure 7 : Averaging 8](https://d.docs.live.net/9d6588607331eafb/Travail/UFR%202018-2019%20(M1)/RFAI/New%20Year/Rapport/Rapport%20%20TP%201%20-%20De%20l'image%20aux%20descripteurs.docx#_Toc32727038)

[Figure 8 : Flou médian 8](https://d.docs.live.net/9d6588607331eafb/Travail/UFR%202018-2019%20(M1)/RFAI/New%20Year/Rapport/Rapport%20%20TP%201%20-%20De%20l'image%20aux%20descripteurs.docx#_Toc32727039)

[Figure 9 : Filtrage bilatéral 8](https://d.docs.live.net/9d6588607331eafb/Travail/UFR%202018-2019%20(M1)/RFAI/New%20Year/Rapport/Rapport%20%20TP%201%20-%20De%20l'image%20aux%20descripteurs.docx#_Toc32727040)

[Figure 10 : Code source filtrage par gradient 9](#_Toc32727041)

[Figure 11 : Filtrage de type gradient 10](#_Toc32727042)

[Figure 12 : Code source Canny Edges 11](#_Toc32727043)

[Figure 13 : Canny Edge Borders 12](#_Toc32727044)

[Figure 14 : Code source seuillage et binarisation 13](#_Toc32727045)

[Figure 15 : Seuillage simple 14](#_Toc32727046)

[Figure 16 : Seuillage adaptif 15](#_Toc32727047)

[Figure 17 : Binarisation 15](#_Toc32727048)

[Figure 18 : Code source érosion 16](#_Toc32727049)

[Figure 19 : Code source dilatation 16](#_Toc32727050)

[Figure 20 : Image original érosion 17](https://d.docs.live.net/9d6588607331eafb/Travail/UFR%202018-2019%20(M1)/RFAI/New%20Year/Rapport/Rapport%20%20TP%201%20-%20De%20l'image%20aux%20descripteurs.docx#_Toc32727051)

[Figure 21 : Dilatation 17](https://d.docs.live.net/9d6588607331eafb/Travail/UFR%202018-2019%20(M1)/RFAI/New%20Year/Rapport/Rapport%20%20TP%201%20-%20De%20l'image%20aux%20descripteurs.docx#_Toc32727052)

[Figure 22 : Différence encerclé dilatation 17](https://d.docs.live.net/9d6588607331eafb/Travail/UFR%202018-2019%20(M1)/RFAI/New%20Year/Rapport/Rapport%20%20TP%201%20-%20De%20l'image%20aux%20descripteurs.docx#_Toc32727053)

[Figure 23: Image original dilatation 17](https://d.docs.live.net/9d6588607331eafb/Travail/UFR%202018-2019%20(M1)/RFAI/New%20Year/Rapport/Rapport%20%20TP%201%20-%20De%20l'image%20aux%20descripteurs.docx#_Toc32727054)

[Figure 24 : Code source soustraction image 18](#_Toc32727055)

[Figure 25 : Deuxième version de la fonction 19](#_Toc32727056)

[Figure 26 : Différences entre les cartes avec une opération de soustraction 19](#_Toc32727057)

[Figure 27 : Code source changement de plan 20](#_Toc32727058)

[Figure 28 : Changement de plan résultat console 21](#_Toc32727059)

[Figure 29 : Changement de plan détecté sur console 21](#_Toc32727060)

[Figure 30 : Image de référence, image actuelle respectivement 22](#_Toc32727061)

[Figure 31 : Code source résumés de plan 22](#_Toc32727062)

[Figure 32 : Nombre de changement détecté 23](#_Toc32727063)

[Figure 33 : Code source changement de plan 24](#_Toc32727064)

[Figure 35 : Nombre d'image par plan 25](#_Toc32727065)

[Figure 34 : Comptage des images par plan 25](https://d.docs.live.net/9d6588607331eafb/Travail/UFR%202018-2019%20(M1)/RFAI/New%20Year/Rapport/Rapport%20%20TP%201%20-%20De%20l'image%20aux%20descripteurs.docx#_Toc32727066)

[Figure 36 : Points variant 26](#_Toc32727067)

[Figure 37 : Code source detection SIFT 27](#_Toc32727068)

[Figure 38 : Descripteur SIFT 27](#_Toc32727069)

[Figure 39 : Example histogramme simple 28](#_Toc32727070)

## Exercice 1 : Prise en main de Python + OpenCV

Dans ce premier exercice il est demandé d’éditer le fichier source « videoprecessing.py » en ajoutant des commentaires afin d’étudier le fonctionnement du code source pour une bonne première prise en main de OpenCV. Soit la capture d’écran suivante montrant le code source commenté :

### Question 1

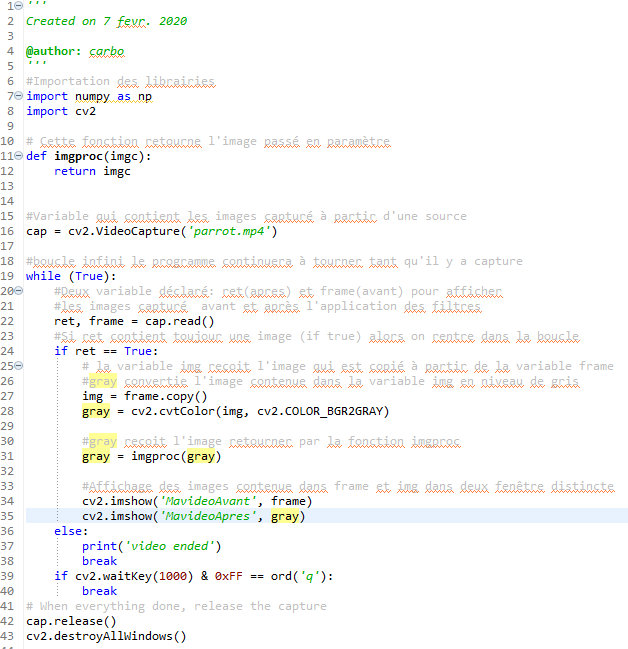


Figure 1 : Exercice 1 code commenté

Pour une bref explication du code, cela consiste à lire une vidéo (capture) et afficher deux mêmes images à la différence qu’une sera en couleur alors que l’autre sera en niveau de gris.

Voici le résultat après exécution du code, deux même image l’une en couleur l’autre en niveau de gris :

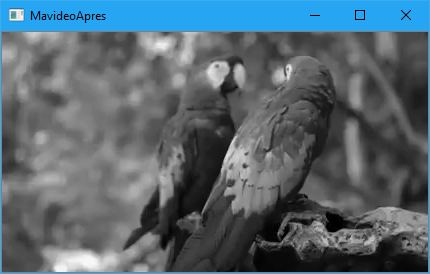
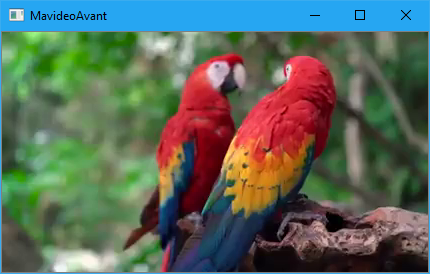


Figure 2 : Résultat après exécution du code

## Exercice 2 : Filtrage, convolution, détection de contours

Le but de cet exercice consiste à modifier le code source de l’exercice 1, particulièrement la fonction « imgproc » afin d’appliquer différents traitement de filtrage (tel que le flou gaussien, masque médian, etc…). Il est également demandé d’appliquer des algorithmes de détection de contours (tel que Sobel, Laplacien, Canny).

### Question 1

Python facilite l’application des filtres sur des images en quelque lignes de code, cela étant dit le mécanisme de ces fonctions cache une complexité tenant compte des fonctions mathématique qui peuvent être appliqué. Dans cette question il est demandé d’appliquer différents traitement de filtrage nous commencerons par traité les filtrages de tel que flou moyen « Averaging », flou gaussien, flou médian, et enfin le filtrage bilatéral respectivement.

Filtrage flou moyen : « Averaging » consiste à prendre simplement la moyenne de tous les pixels sous la zone du noyau et remplace l'élément central. La hauteur et la largeur du noyau doit être doit être positif et impaire. Ceci est fait par la fonction cv2.blur()

Flou gaussien : Dans ce cas, au lieu d'un filtre en boîte, on utilise un noyau gaussien. Cela se fait avec la fonction cv2.GaussianBlur(). Nous devons spécifier la largeur et la hauteur du noyau qui doivent être positives et impaires. Nous devons également spécifier l'écart-type dans les directions X et Y, sigmaX et sigmaY respectivement. Si seul sigmaX est spécifié, sigmaY est considéré comme identique à sigmaX. Si les deux sont donnés sous forme de zéros, ils sont calculés à partir de la taille du noyau. Le flou gaussien est très efficace pour éliminer le bruit gaussien de l'image.

Flou médian : La fonction cv2.medianBlur() qui prend la médiane de tous les pixels sous la zone du noyau et l'élément central est remplacé par cette valeur médiane. Ceci est très efficace contre le bruits dans les images. L'élément central est toujours remplacé par une valeur de pixel dans l'image. Cela permet de réduire efficacement le bruit. La taille de noyau doit être un nombre entier impair positif.

filtrage bilatéral : cv2.bilateralFilter() est très efficace pour éliminer le bruit tout en gardant les bords tranchants. Mais son fonctionnement est plus lent que celui des autres filtres. Nous avons déjà vu que le filtre gaussien prend le voisinage autour du pixel et trouve sa moyenne gaussienne pondérée. Ce filtre gaussien est une fonction de l'espace seul, c'est-à-dire que les pixels proches sont pris en compte lors du filtrage. Il ne tient pas compte du fait que les pixels ont presque la même intensité. Il ne tient pas compte du fait que le pixel soit un pixel de bordure ou non. Il brouille donc aussi les bords, ce que nous ne voulons pas faire.

La capture d’écran ci-dessous montre comment les filtres sont appliquer à l’aide de la fonction imgproc(). Cette fonction prend en paramètre une image et la retourne avec un filtre appliquer dessus

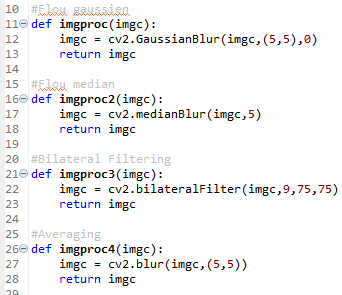


Figure 3 : Code source fonction imgproc

La capture d’écran ci-dessous montre comment est appelé la fonction et comment les images sont affichées par la suite :

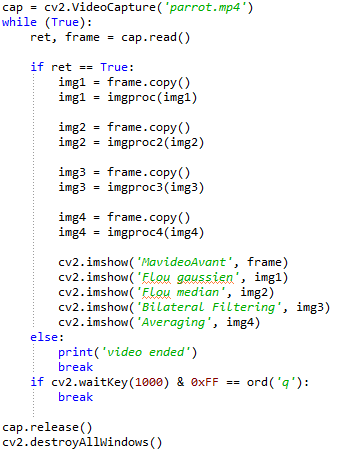


Figure 4 : Application des filtres

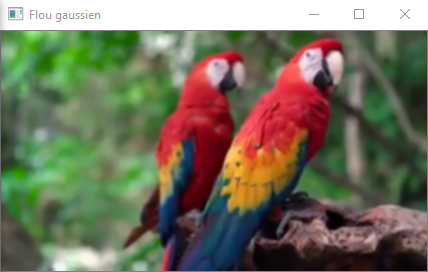
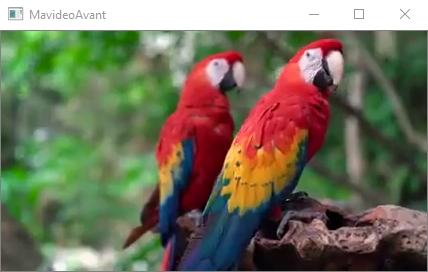
Les captures d’écrans ci-dessous montre les résultats après application de ces différents filtres.

Figure 5 : Flou gaussien

Figure 6 : Image original, aucun filtre

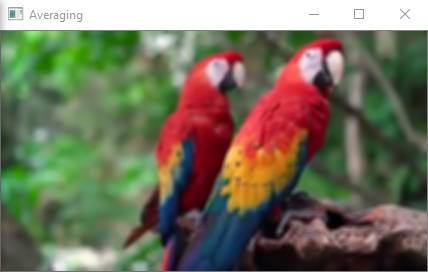
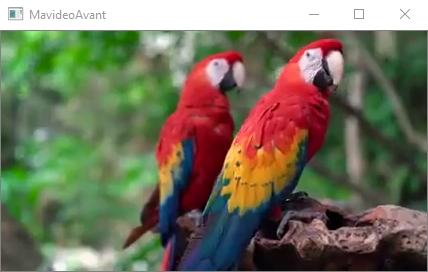


Figure 6 : Image original, aucun filtre

Figure 7 : Averaging

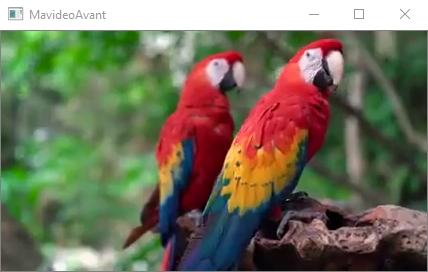
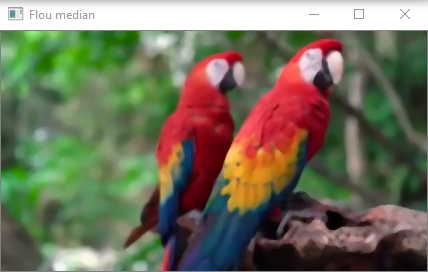


Figure 8 : Flou médian

Figure 6 : Image original, aucun filtre

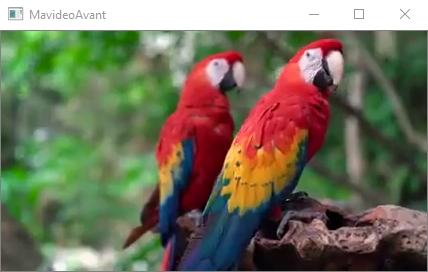
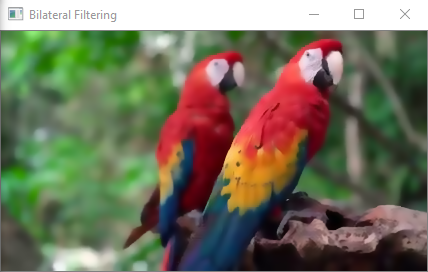


Figure 9 : Filtrage bilatéral

Figure 6 : Image original, aucun filtre

### Question 2

Dans cette question il est demandé d’appliquer des algorithmes de détection des contours au moyen de plusieurs technique tel que Sobel, Laplacien, Canny en complétant le code précédant.

Nous allons commencer premièrement par l’utilisation de trois type de filtrage de gradient qui sont respectivement les dérivées Sobel & Scharr, produits dérivés de Laplacian.

* Les opérateur Sobel est une opération conjointe de lissage et de différenciation gausssienne, donc plus résistante au bruit.
* Alors que le produit de dérivés Laplacian calcule le Laplacien de l'image donnée par la relation, où chaque dérivé est trouvé en utilisant les dérivés de Sobel.

Pour appliquer de tel filtre voici le code source suivant :

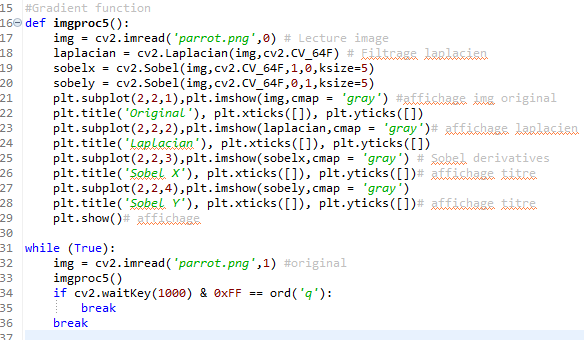


Figure 10 : Code source filtrage par gradient

Voici le résultat après exécution du code source :

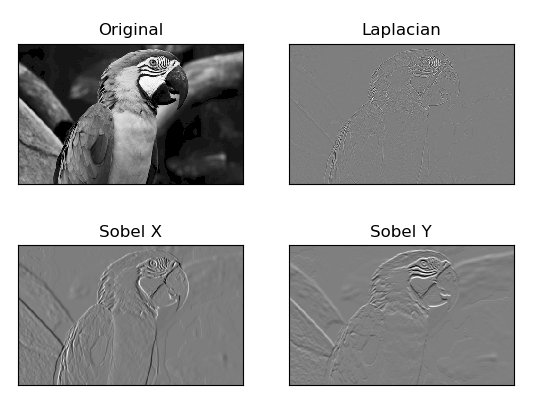


Figure 11 : Filtrage de type gradient

Par la suite nous allons utiliser l’algorithme de détection de contours Canny Edge qui peut être utiliser dans OpenCV à l’aide de la fonction cv2.Canny().

Il s'agit d'un algorithme à plusieurs étapes et nous allons passer en revue chaque étape.

* 1ere étape : réduction du bruit
  + Comme la détection des contours est sensible au bruit dans l'image, la première étape consiste à éliminer le bruit dans l'image avec un filtre gaussien
* 2ème étape : Trouver le gradient d'intensité de l'image
  + L'image lissée est ensuite filtrée avec un noyau de Sobel dans les directions horizontale et verticale pour obtenir une dérivée première dans les directions horizontale et verticale à partir de ces deux images, nous pouvons trouver le gradient de bord et la direction pour chaque pixel
* 3ème étape : Suppression non maximale
  + Après avoir obtenu la magnitude et la direction du gradient, un balayage complet de l'image est effectué pour éliminer les pixels indésirables qui ne constituent pas forcément le bord. Pour cela, à chaque pixel, on vérifie s'il s'agit d'un maximum local dans son voisinage dans la direction du gradient.
* 4ème étape « Hysteresis Thresholding »
  + Cette étape permet de décider quels sont les bords qui sont vraiment des bords et quels sont ceux qui ne le sont pas. Pour cela, nous avons besoin de deux valeurs seuils, minVal et maxVal. Les arêtes dont le gradient d'intensité est supérieur à maxVal sont sûres d'être des arêtes et celles qui sont inférieures à minVal sont sûres de ne pas être des arêtes, donc rejetées. Ceux qui se situent entre ces deux seuils sont classés comme étant des arêtes ou des non-arêtes en fonction de leur connectivité. Si elles sont connectées à des pixels "sûrs", elles sont considérées comme faisant partie des bords. Dans le cas contraire, ils sont également rejetés.

OpenCV met tout ce qui précède dans une seule fonction, cv2.Canny(). Nous allons voir comment l'utiliser. Le premier argument est notre image d'entrée. Les deuxième et troisième arguments sont nos minVal et maxVal respectivement. Le troisième argument est aperture\_size. C'est la taille du noyau Sobel utilisé pour trouver des gradients d'image. Le dernier argument est le gradient qui spécifie l'équation pour trouver la magnitude du gradient.

Soit la capture d’écran suivante correspond à l’application de la fonction de détection des bords avec l’algorithme Canny Edges :

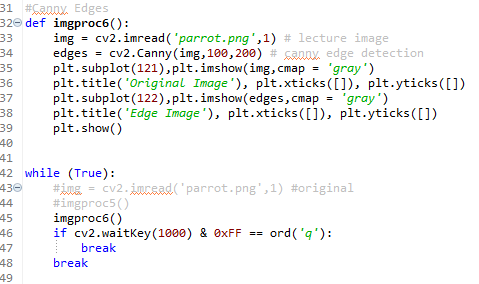


Figure 12 : Code source Canny Edges

Résultat nous pouvons ainsi constaté sur l’image à droite la détection des bords du perroquet avec une précision impressionnante tel que la détection du bord du bec, des plumes sur les ailes.



Figure 13 : Canny Edge Borders

## Exercice 3 : Images binaires et opérations entre images

Le but de cet exercice sera d’appliquer différents algorithmes de seuillage / binarisation (seuillage fixe ou adaptif) suivi d’opérations de morphologie mathématique tel que l’érosion, dilatation suivie par la suite d’opération de soustraction entre deux frames successifs avant affichage.

### Question 1

Afin de répondre à cette question il est demandé premièrement de compléter le code fourni dans l’exercice 1 en modifiant la fonction imgproc() dans laquelle sera implémenter les algorithmes demandé.

* Seuillage simple : Si la valeur d'un pixel est supérieure à une valeur de seuil donné, on lui attribue une valeur (peut être blanche), sinon on lui attribue une autre valeur (peut être noire). La fonction utilisée est cv2.threshold. Le premier argument est l'image source, qui doit être une image en niveaux de gris. Le deuxième argument est la valeur de seuil qui est utilisée pour classer les valeurs des pixels. Le troisième argument est le maxVal qui représente la valeur à donner si la valeur du pixel est supérieure (parfois inférieure) à la valeur de seuil.
* Seuillage adaptif : L'algorithme calcule le seuil pour de petites régions de l'image. Ainsi, nous obtenons des seuils différents pour différentes régions de la même image et cela nous donne de meilleurs résultats pour des images dont l'éclairage varie.
* Binarisation : Le but de l’algorithme consiste à calculer automatiquement une valeur de seuil à partir de l’histogramme de l’image bimodale.

La capture d’écran ci-dessous montre les algorithmes utiliser afin d’appliquer des seuillage et binarisation sur une image.

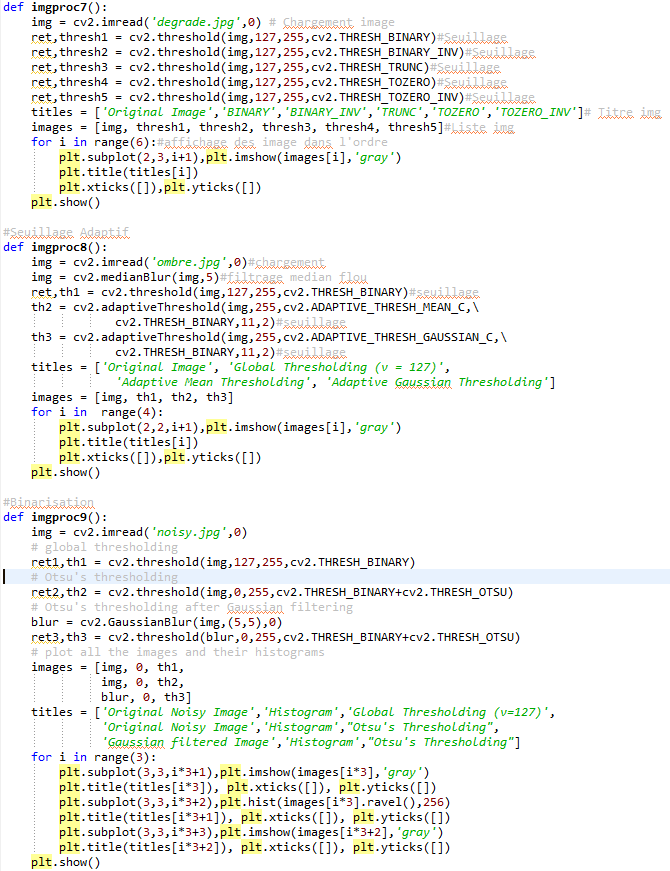


Figure 14 : Code source seuillage et binarisation

Les captures d’écran ci-dessous montre le résultat après exécution du code source :

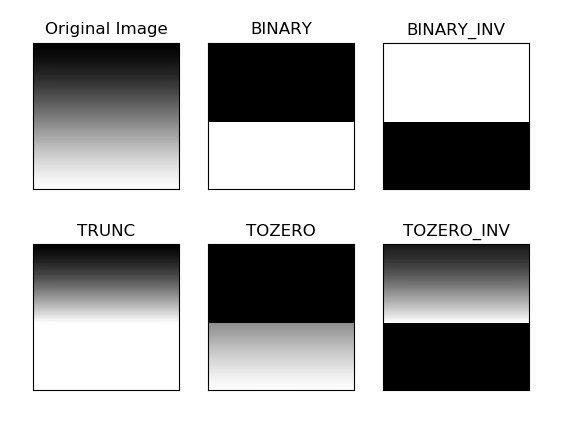


Figure 15 : Seuillage simple

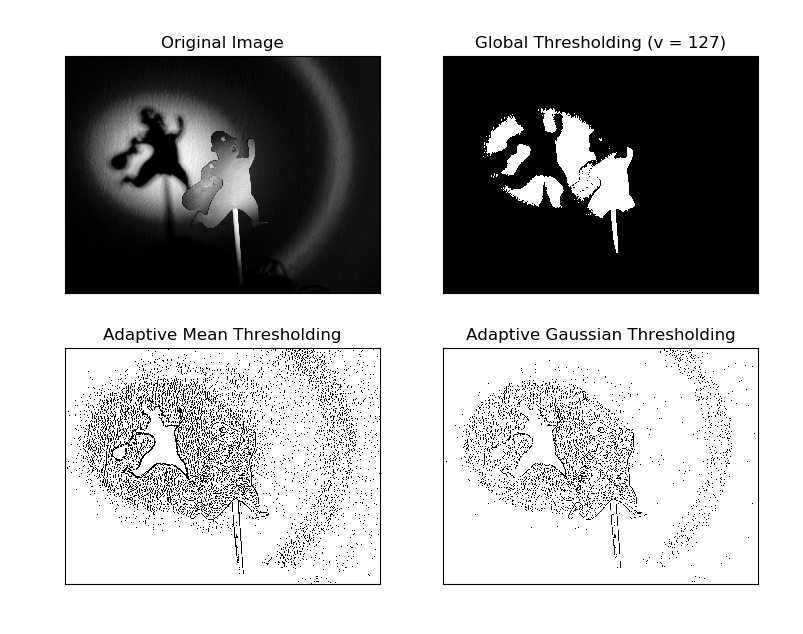


Figure 16 : Seuillage adaptif

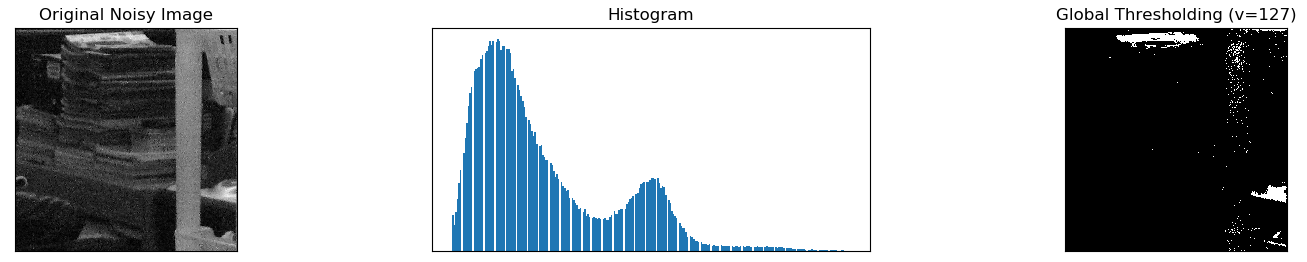


Figure 17 : Binarisation

La seconde partie de la question consistait à traité les forme morphologie mathématique (érosion, dilatation) aux frames courant avant affichage.

Les transformations morphologiques sont des opérations simples basées sur la forme de l'image. Elles sont normalement effectuées sur des images binaires. Elle nécessite deux entrées, l'une est notre image originale, l'autre est appelée « noyau » qui décide de la nature de l'opération. Les deux opérateurs morphologiques de base sont l'érosion et la dilatation. Leurs principes de fonctionnement est presque le même mais à une différence près.

* L’érosion : Un pixel de l'image originale (soit 1 ou 0) ne sera considéré comme 1 que si tous les pixels sous le noyau sont à 1, sinon il est érodé (ramené à zéro).

Ainsi, tous les pixels proches de la limite seront éliminés en fonction de la taille du noyau. Ainsi, l'épaisseur ou la taille de l'objet de premier plan diminue ou simplement la région blanche diminue dans l'image. C'est utile pour supprimer les petits bruits blancs

* La dilatation : C'est tout le contraire de l'érosion. Ici, un élément de pixel est "1" si au moins un pixel sous le noyau est "1". Il augmente donc la région blanche de l'image ou la taille de l'objet de premier plan augmente.

Les captures d’écran suivante contiennent le codes sources de la fonction imgproc() qui permet l’application de tel filtres sur une image :

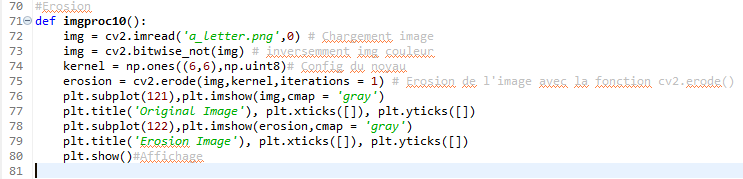


Figure 18 : Code source érosion

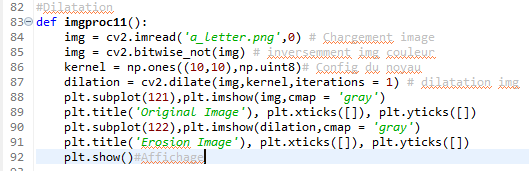


Figure 19 : Code source dilatation

Voici le résultat du code source après exécution de la fonction « imgproc11() » pour l’érosion de l’image :

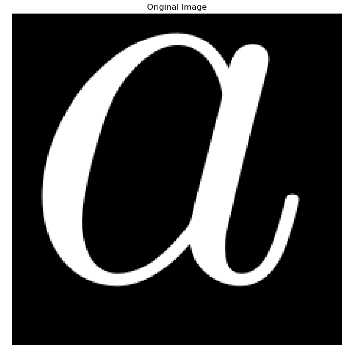
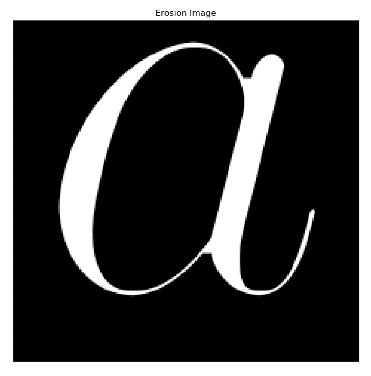
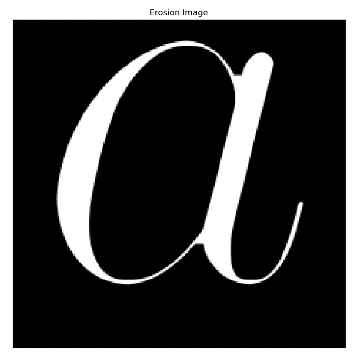


Figure 22 : Différences encerclé érosion

Figure 21 : Érosion

Figure 20 : Image original érosion

Sur ces image, nous pouvons voir sur la figure 22 que certaines différence peuvent être très évidente par rapport à l’image original : « figure 20 ».

La capturé d’écran suivante concerne la dilatation de l’image faite ici avec la fonction « imgporc12() » :

Figure 21 : Dilatation

Figure 22 : Différence encerclé dilatation



Figure 23: Image original dilatation

De même sur la figure 22 les différences sont très visibles, on remarque une dilatation (augmentation de l’épaisseur de la forme) ces paramètres peuvent notamment servir à améliorer la forme de certaines lettres pour faciliter la lecture avec un algorithme de type SVM lorsque les lettres ne sont pas assez épaisses.

### Question 2

Dans cette question il est demandé de compléter le code fourni dans l’Exercice 1 afin d’appliquer une opération de soustraction entre 2 frames successives avant affichage. L’intérêt de ce type d’opération permet de trouver la différence entre deux images pour détecter les changements de plan. Les deux images doivent avoir la même profondeur et le même type.

La soustraction se fait de la façon suivante :

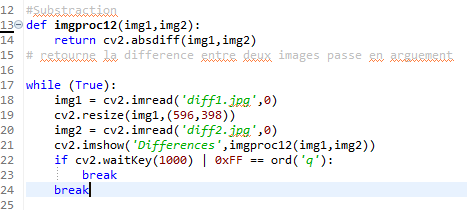


Figure 24 : Code source soustraction image

Sur la figure 24 est représenté de façon simple et basique le fonctionnement de la soustraction sous « OpenCV », cela se fait au moyen de l’opération cv2.absdiff([arg\_img1],[arg\_img2]) qui retourne le résultat de la soustraction entre les images entrée en argument de la fonction.

L’opération de soustraction est réalisé de la façon suivante :

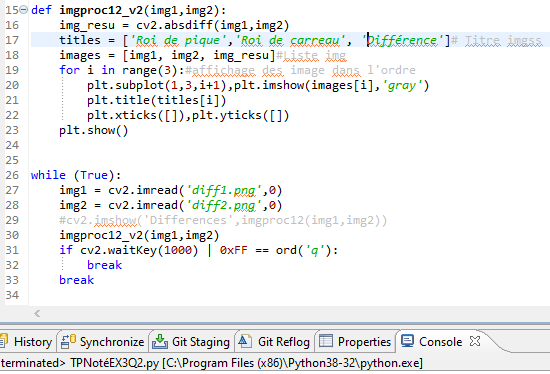


Figure 25 : Deuxième version de la fonction

Sur la figure 25, une seconde version de la fonction précédente à la différence que celle-ci permet de mieux comparer visuellement les cartes. Voici le résultat de l’éxécution de la fonction « imgproc12\_v2([img\_arg1],[img\_arg2]) » :

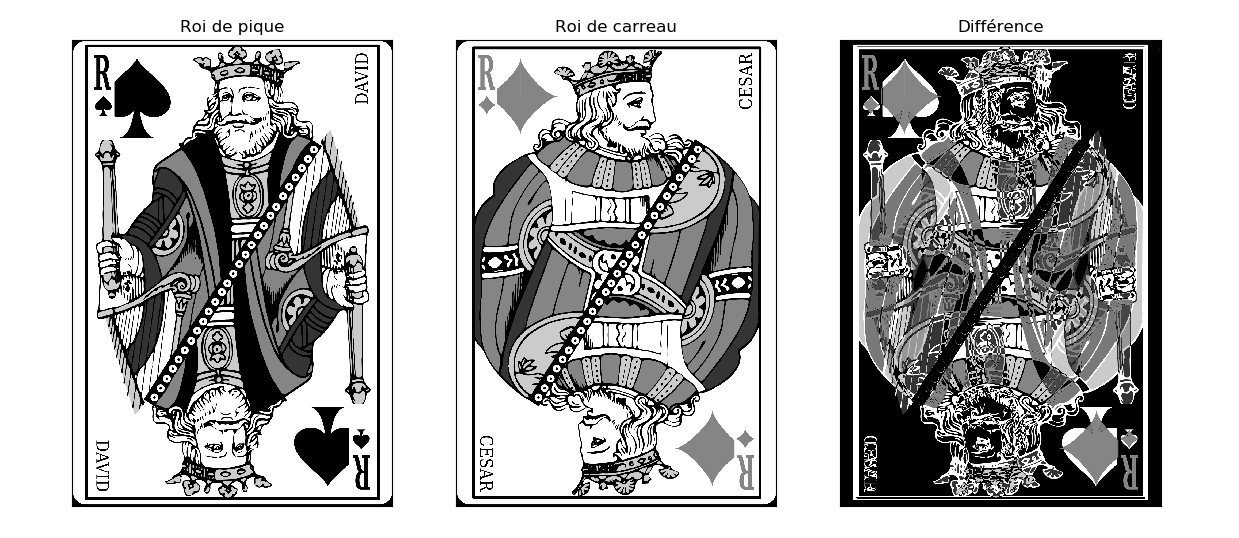


Figure 26 : Différences entre les cartes avec une opération de soustraction

## Exercice 4 : Détection de changement de plan et résumé automatique de vidéo

Le but de cet exercice est de mettre en place un mécanisme de détection automatique de changement de plan dans les vidéos ainsi qu’un mécanisme de génération automatique de résumés de plan dans les vidéos.

### Question 1 : Changement de plan

Le principe de l’algo consiste premièrement à acquérir la première image de la vidéo, cette image servira de référence pour faire des comparaisons afin de déterminer s’il y a changement de plan. Le changement de plan se fera à l’aide de la fonction de soustraction sur laquelle nous allons compter les valeurs différentes de 0 (0 = pas de changement) par rapport à l’image de référence. De plus il nous sera donc possible de fixer un seuil de « différence » afin que l’algorithme puisse reconnaitre un changement de plan.

La figure suivante contient le code source permettant de faire un tel traitement :

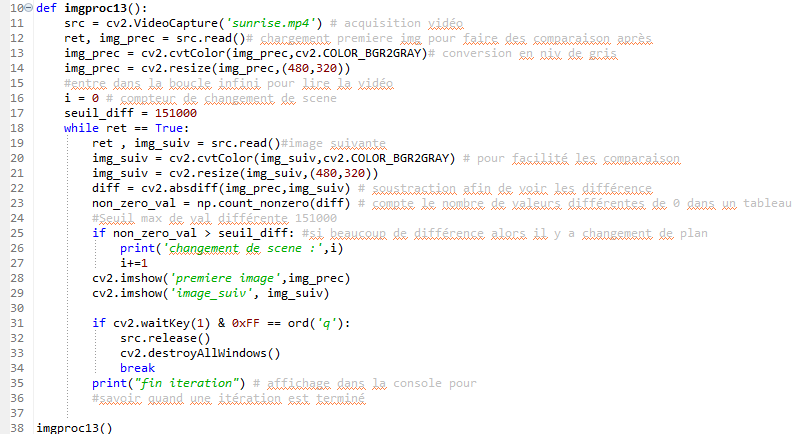


Figure 27 : Code source changement de plan

Après exécution du code source nous pouvons constaté les résultats suivant :

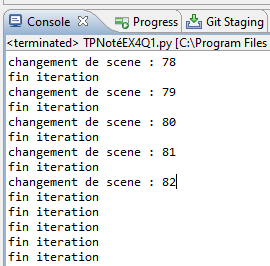


Figure 28 : Changement de plan résultat console

L’algorithme détecte un changement de plan jusqu’à la scène 82, puis plus de changement de plan ici marqué par une suite de mots : « fin iteration »

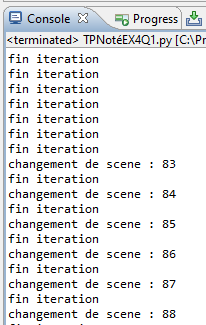


Figure 29 : Changement de plan détecté sur console

Après une suite d’itération l’algorithme redétecte des changements de plan qui ont dépassé le seuil de tolérance au changement de plan entre l’image de référence et l’image actuelle

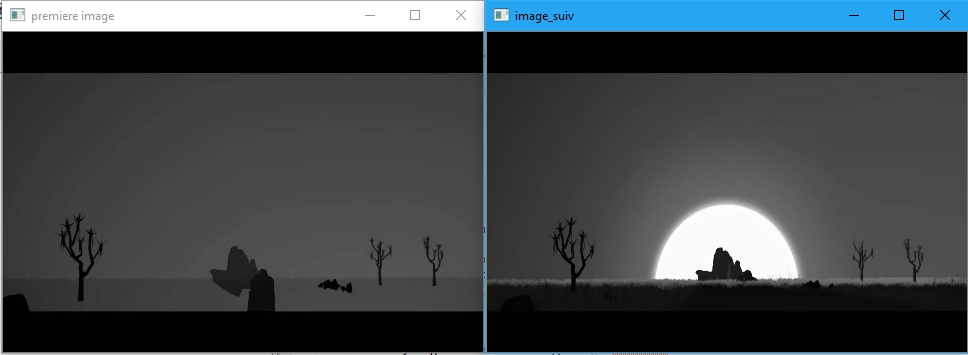


Figure 30 : Image de référence, image actuelle respectivement

### Question 2 : Création d’images résumés de plan

Dans cette question il est demandé de mettre en place un mécanisme de génération automatique de résumés de plans dans les vidéos. Le but de l’algorithme mis en œuvre est de prendre deux image la première servira de référence pour comparaison par rapport à la frame suivante. Si la valeur de la frame suivante est supérieur au seuil cité précédemment alors cet image représente une nouveau plan.

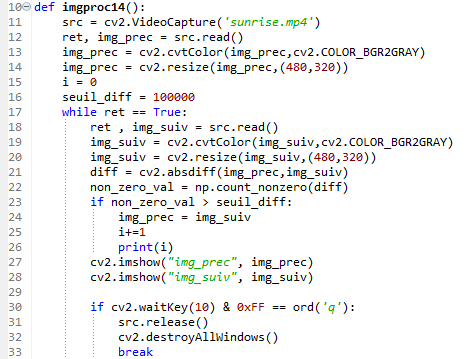
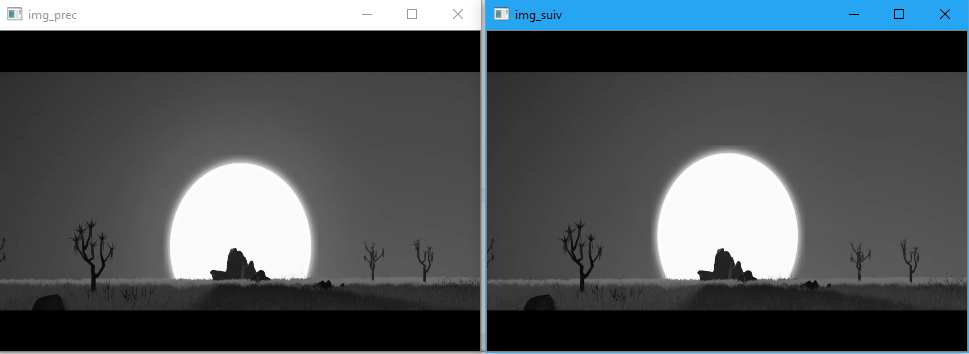


Figure 31 : Code source résumés de plan



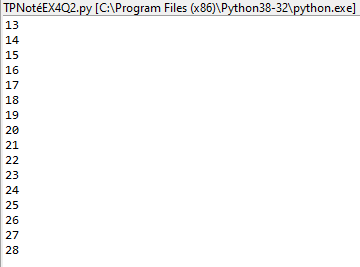


Figure 32 : Nombre de changement détecté

## Exercice 5 : Calcul de descripteurs SIFT, appariement de descripteurs et résumé automatique de vidéo

L’objectif de cet exercice est de mettre en place un ensemble d’algorithme en réutilisant le code source écrit précédemment. Il est demandé premièrement de en place un algorithme qui construit une liste stockant l’ensemble des images correspondant à un seul plan, par la suite il faut créer une fonction qui, étant donnée deux images, calcule les points SIFT des deux images et trouve les correspondances entre ces points (provenant des deux images) tout en ajoutant un paramètre de seuil afin de retenir que les meilleurs correspondances.

Il faut également créer une fonction qui, étant donnée une séquence d’images, fournit l’image qui a le meilleur appariement avec sa précédente dans la séquence. Et enfin exploiter l’ensemble des méthodes vues précédemment pour mettre en place un mécanisme de génération automatique de résumés de plans dans les vidéos.

### Question 1 : Création d’images résumés de plan

Le principe de l’algorithme consiste à capturer les trames et de remarquer un changement de plan en effectuant une opération de soustraction (différence absolue) par la suite le résultat retourner contient une matrice de valeur pour les pixels, dans cette matrice sera compter le nombre de différences entre les pixels si le changement est important par rapport à la trame précédente, alors il s’agit d’un nouveau plan. Si c’est un nouveau plan alors il sera ajouté à la liste de plan suivi d’un comptage d’images jusqu’au nouveau changement de plan.

Voici un extrait du code source permettant un tel traitement :

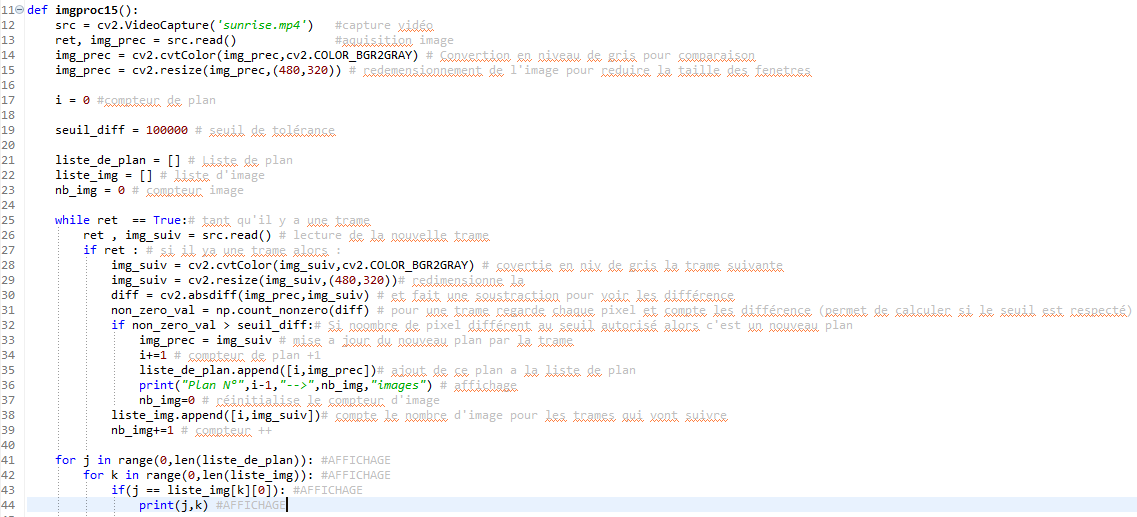
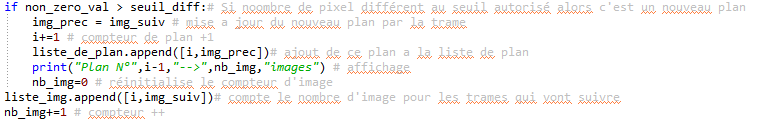


Figure 33 : Code source changement de plan

Le chainage entre les nouveaux plan et le comptage de leurs images se fait de la façon suivante à l’aide de listes l’une contiendra l’id du plan ainsi que la trame de ce changement de plan. Par la suite une seconde liste comptera les nouvelles trames pour ce plan jusqu’au prochain changement de plan.

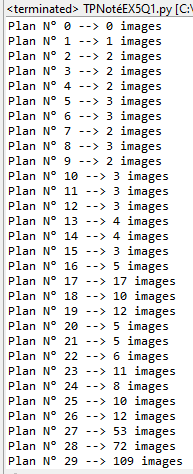
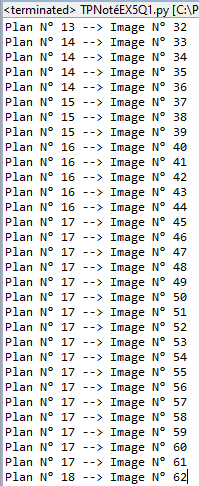
Les listes accueil des tuples de la façon suivante :



Liste de plan 🡪 : (id\_nouveau\_plan, trame\_du\_nouveau\_plan)

Liste d’image par plan 🡪 : (id\_plan, id\_image)

Ainsi « id\_nouveau\_plan » et « id\_plan » joue le rôle de clé afin de garantir une cohérence entre les plans et les trames leurs appartenant. La figure suivante contient le résultat du traitement, il sera expliqué par la suite.



17

16

15

14

Figure 35 : Nombre d'image par plan

Figure 34 : Comptage des images par plan

* Figure 34 : Désigne le plan N° et le nombre d’image qu’il contient. On remarque que le plan N° 14 contient 4 images ; N°15 contient 3 images ; N°16 contient 5 images.
* Figure 35 : Désigne quel sont les trames appartement à quel plan ; ainsi le plan N°14 à 4 images, image numéro [33,34,35,36] ; plan N°15 contient 3 images numéroté [37,38,39] ; N°16 , 5 images [40,41,42,43,44]

Bien que l’algorithme semble séduisant, celui-ci reste faiblement performant en raison de son chainage, en effet celui-ci utilise deux liste chainé les unes aux autres

### Question 2 : Calcul de descripteurs SIFT et appariement

Dans cette question nous allons traiter sur le calcul de descripteur SIFT entre deux images et trouver les correspondances entre ces points.

Il existe plusieurs façons de détecter les bords d’une image, dont l’une est nommé « Harris ». Sauf que cette méthode reste invariante à la rotation, ce qui signifie que, même si l'image est tournée, nous pouvons trouver les mêmes coins. C'est évident, car les coins restent des coins dans l'image tournée également. Mais qu'en est-il de la mise à l'échelle ? Un coin peut ne pas être un coin si l'image est mise à l'échelle. Par exemple, regardez une image simple ci-dessous. Un coin dans une petite image à l'intérieur d'une petite fenêtre est plat lorsqu'il est zoomé dans la même fenêtre. Le coin de Harris n'est donc pas invariant en termes d'échelle.

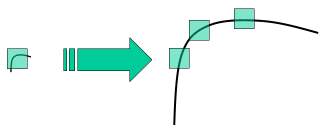


Figure 36 : Points variant

Commençons par la détection des points clés et dessinons-les. Nous devons d'abord construire un objet SIFT. Nous pouvons lui passer différents paramètres qui sont optionnels et qui sont bien expliqués dans la documentation en ligne de OpenCV.

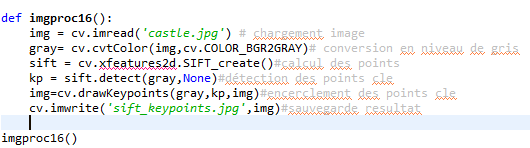


Figure 37 : Code source detection SIFT

La fonction sift.detect() trouve le point clé dans les images. Il est possible de passer un masque si l’on cherche seulement une partie de l'image. Chaque point clé est une structure spéciale qui possède de nombreux attributs comme ses coordonnées (x,y), la taille du voisinage significatif, l'angle qui spécifie son orientation, la réponse qui spécifie la force des points clés, etc.

OpenCV fournit également la fonction cv.drawKeyPoints() qui dessine les petits cercles sur les emplacements des points clés.

En voici les résultat :



Figure 38 : Descripteur SIFT

## 1.6 Réponse aux questions sur la fin du TP

1 - Sur un histogramme, l'axe x représente:   
🡪 un niveau de gris   
L’histogramme d’une image mesure la distribution des niveaux de gris dans l’image

- Sur un histogramme, l'axe y représente:   
- -un nombre de pixels

l’histogramme permet de connaitre la probabilité de tomber sur un pixel de valeur x en tirant un pixel au hasard dans l’image.

Un histogramme est donc représenté de la façon suivante en montrant la distributivité des niveau de gris (class) en fonction du nombre de pixels

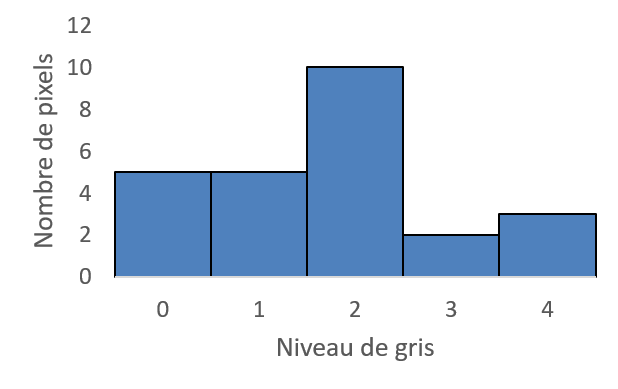


Figure 39 : Example histogramme simple

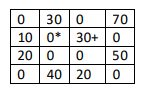
3 - Si on additionne toutes les valeurs de l'histogramme entre 0 et 255, on obtient le nombre de pixels de l'image.  
🡪FAUX

Cette valeur ne représente que des niveaux de gris, les additionné ne donnera pas le nombre de pixels présent sur une image

4 - Lorsqu'on applique un filtrage, 2 pixels de niveaux de gris identiques dans l'image d'origine auront nécessairement des niveaux de gris identiques dans l'image modifiée.  
 🡪 FAUX

Si un filtrage appliqué il est fortement possible que les valeurs de ces deux pixels changent lorsque le filtrage lisse les bords tel que le filtre moyenneur

5 - On applique un filtre 3x3 à une image, dont voici un petit morceau :



Quelle sera la valeur du pixel \* après filtrage médian 3x3?   
🡪 0

Il faut sélectionner l’élément médian après avoir triées les valeurs dans l’ordre croissant

Quelle sera la valeur du pixel \* après filtrage moyenneur 3x3?   
🡪10