Projet de Traitement d'Images
MASTER 1

Andrea Cherubini

Table des matières

1 Introduction

L'objectif de ce projet est de vous familiariser avec les notions de traitement d'images que vous avez vu en cours. Vous pourrez travailler aussi bien sur les différentes images présentes sur moodle, que sur des images prises avec votre téléphone ou téléchargées depuis internet.

Vous aborderez les notions suivantes :

— conversion du format RGB vers le format HSV,
— histogramme d'image,
— égalisation d'histogramme,
— binarisation avec le seuil d'Otsu,
— opérations morphologiques (erosion/dilatation),
— convolution par filtrage Gaussien,
— convolution pour dériver l'image en x et y,
— extraction des points de Harris.

Ces huits étapes constitueront les paliers notés du projet. A chaque fois que vous atteignez un des huits paliers, appelez l'encadrant et montrez lui le résultat obtenu, en expliquant comment vous avez procédé.

Vous travaillerez de la manière suivante :

1. dans un premier temps (Sect. 2) vous allez préparer les fonctions fondamentales pour le travail,
2. ensuite vous allez réaliser, sur une image, la binarisation en niveau de gris et veiller à la fermeture des zones foncées (Sect. 3),
3. enfin vous allez réaliser, sur une autre image, la détection des points de Harris (Sect. 4).

Il est important de noter que sur chaque image vous obtiendrez des résultats plus ou moins satisfaisant pour une opération donnée.

2 Fonctions de base

Vous utiliserez principalement les fonctions de la bibliothèque `numpy`. Vous utiliserez aussi - pour la sauvegarde et l'affichage des images, la bibliothèque `PIL`, et pour l'affichage des histogrammes de luminance, `matplotlib`. Pour vérifier si l'installation s'est bien déroulée, vous pouvez tester le programme squelette ci-dessous.

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

rgbImg = np.array(Image.open('lena.jpg'))
rawimg = Image.fromarray(rgbImg)
rawimg.show()
```

Ce programme affiche l'image `lena.jpg` et la convertit en array du type numpy.

Vous pouvez voir les caractéristiques de l'image avec les lignes suivantes :

```
print(type(rgbImg))
print(rgbImg.dtype)
print(rgbImg.shape)
```

L'ordre des canaux sera Rouge, Vert, Bleu. Tout au cours du projet, il est opportun de travailler avec des arrays de type double. Pour cela, il va falloir que vous construisiez chaque nouvelle image avec la commande :

```
newImg = np.zeros((rows, cols), np.dtype('d'))
```

Ensuite, pour la sauvegarde et (éventuellement) l'affichage de l'image, vous ferez recours à la classe `Image` de `PIL`, en appelant la fonction suivante (à rajouter dans votre programme) :

```
def saveAndShowGrey(npImg,string):
        img = Image.fromarray(npImg)
        img = img.convert("L")
        img.save(string)
        img.show()
        return img
```

La première étape consistera à visualiser les images de gris et de teinte correspondantes à l'image originale. Pour cela il s'agira – en suivant la logique évoquée ci-dessus, de procéder comme suit :

```
rows = rgbImg.shape[0]
cols = rgbImg.shape[1]
greyImg = np.zeros((rows, cols), np.dtype('d'))
# faire les calculs opportuns
# pour chaque pixel de l'image
# donc dans deux boucles for
saveAndShowGrey(greyImg, '1-greyImg.png')
```

Par exemple, pour l'image de l'arc en ciel, vous devriez obtenir des images comme celles de la Fig. 1.

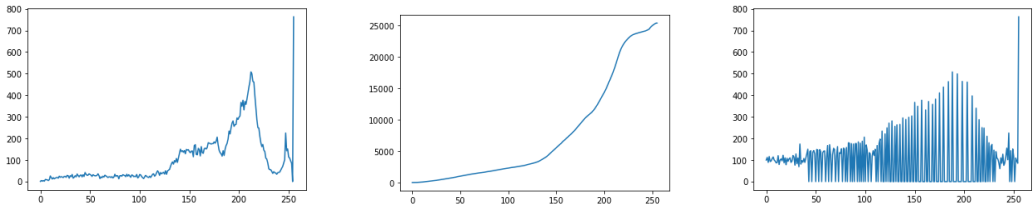
FIGURE 1 – De gauche à droite : images RGB, de gris, et de teinte.

L'étape suivante consistera à écrire une fonction python permettant de calculer et de tracer l'histogramme d'une image à un canal, avec 256 bacs. Cette fonction prend en entrée une image, et à partir de ses luminances maximale et minimale, détermine la fonction permettant de convertir la gamme de luminance de l'image vers l'indice (de 0 à 255) du bac correspondant. Ensuite, chaque pixel est visité pour incrémenter le bac correspondant de l'histogramme. Enfin, la fonction retourne l'histogramme et l'affiche. Pour vous inspirer :

```
def getAndDrawHisto(gImg):
        histo = np.zeros(256, dtype=np.uint)
        # faire des choses...
        plt.plot(histo)
        plt.show()
        return(histo)
```

Par exemple, pour l'image de gris de l'outil, vous devriez obtenir un histogramme comme celui de la Fig. 2a.

Vous êtes maintenant prêts pour la partie suivante.



(a) (b) (c)

FIGURE 2 – De gauche à droite : histogramme de l'image de gris de l'outil, histogramme cumulé, histogramme égalisé.

3 Extraction de blobs et fermeture

Dorénavant, vous ferez tous les traitements sur une image à un seul canal (par exemple sur l'image de gris).

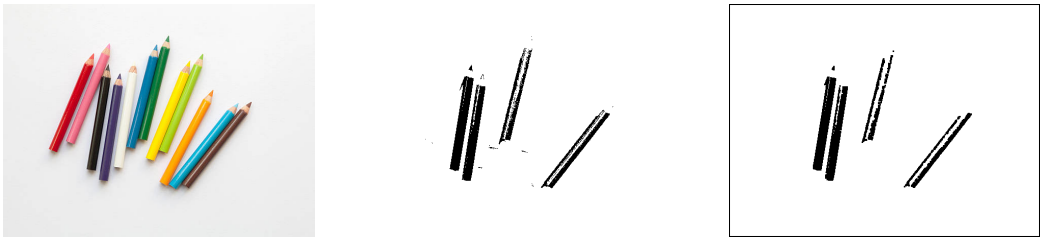
Premièrement, il s'agira d'égaliser l'image. Pour cela, construisez l'histogramme cumulé. Puis, transformez la luminance de chaque pixel de l'image avec la fonction vue en cours. Pour cela prenez soin – de nouveau – à trouver le bac (entre 0 et 255) correspondant à la luminance du pixel. L'image de gris originale et celle égalisée, pour l'image de Lena, sont montrées en Fig. 3

(a) (b)

FIGURE 3 – Image de gris de Lena, avant et après l'égalisation.

A l'aide de la fonction réalisée en Sec. 2, réalisez l'histogramme égalisé de cette image. Par exemple, pour l'image de gris de l'outil, vous devriez obtenir les histogrammes des Figures 2b et 2c.

Procédez maintenant à la binarisation ; vous pourrez faire cela avec l'image brute ou bien avec l'image égalisée. Commencez par appliquer un seuil constant, et à transformer en noir (respectivement, blanc) tout pixel avec une luminance inférieure (supérieure) à ce seuil. Ensuite, écrivez une fonction permettant, à partir d'un histogramme, de calculer le seuil de Otsu, et appliquez ce seuil à l'image. Dans l'exemple des crayons, vous devriez trouver le seuil à 121. Puis, binarisez l'image comme montré en Fig. 4b.



(a) (b) (c)

FIGURE 4 – Image des crayons, de gauche à droite : en couleur, après binarisation et après fermeture.

Enfin, vous allez procéder à la fermeture de l'image. Pour cela, vous aurez besoin de programmer deux fonctions : une pour effectuer une dilatation, et une pour effectuer une érosion. La fermeture consiste à répéter plusieurs boucles composées d'une dilatation, suivie d'une érosion. Vous devrez obtenir une image semblable à Fig. 4c, où les pixels noirs résidus ont disparu. Vous pouvez aussi essayer de réaliser une ouverture (boucles composées d'une érosion, suivie d'une dilatation).

4 Extraction de points de Harris

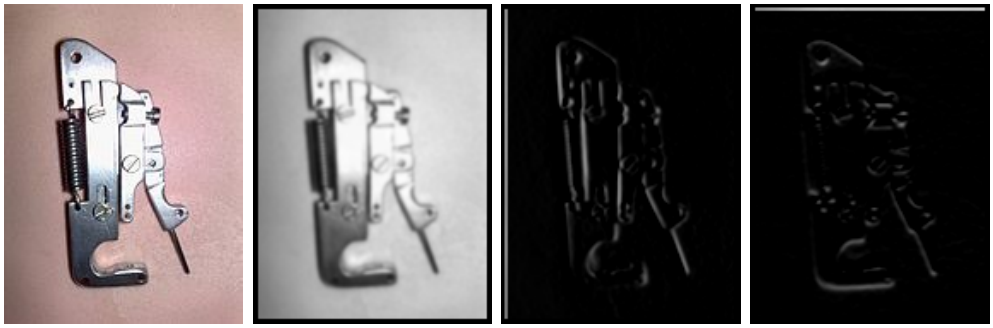
La deuxième partie du projet consiste à détecter les point de Harris dans une image de gris.

Il faudra réaliser plusieurs convolutions, avec des noyaux de tailles et de formes variées (tableau pour le filtrage Gaussien, ligne/colonne pour dériver l'image en x et en y). La première étape consistera donc à écrire une fonction permettant de réaliser une convolution générique, quelque chose comme :

```
def convolution(mask , inp):
        out = np.zeros((inp.shape[0], inp.shape[1]),
                                        np.dtype('d'))
        # calculer out à partir de inp et de mask
        return (out)
```

Attention aux bords de l'image !

Maintenant, vous passerez sur l'image un filtre Gaussien assez grand (tableau de coté 5 ou 7). Il faudra écrire une fonction pour calculer automatiquement les éléments du tableau, et garantir que leur somme est unitaire. Une fois l'image filtrée, vous pourrez obtenir l'image dérivée en x (notée $\mathbf{I}_x$) et dérivée en y (notée $\mathbf{I}_y$), avec les noyaux de convolution correspondants. Les résultats pour l'image de l'outil sont montrés en Fig. 5.



(a) (b) (c) (d)

FIGURE 5 – De gauche à droite : image brute de l'outil, outil après filtrage Gaussien, image dérivée en x (notée $\mathbf{I}_x$) et image dérivée en y (notée $\mathbf{I}_y$).

Pour trouver les points de Harris, il vous faut maintenant calculer les trois produits entre les dérivées partielles en x et y, soit $\mathbf{I}_x\mathbf{I}_x$, $\mathbf{I}_y\mathbf{I}_y$ et $\mathbf{I}_x\mathbf{I}_y$. Ces trois images devront aussi être passées dans le filtre Gaussien, puis être utilisées calculer l'image de Harris. Enfin, en appliquant un seuil (par exemple celui de Otsu) à cette image, vous obtiendrez les points de Harris. La Fig. 6 montre les résultats obtenus pour les images de l'outil et des bâtiments.

(a) (b)

FIGURE 6 – Point de Harris sur les images de l'outil et des bâtiments.

5 Pour aller plus loin

Combien de lignes fait votre code ?
Pouvez vous tout faire en moins de 200 lignes ?
Les mêmes opérations sont implémentées dans la bibliothèque `opencv`.
Esssayez de répeter le projet ou en faire un nouveau en utilisant `opencv`.