Image et Interaction

Cyril Barrelet & Marc Hartley

December 2024

1 Introduction

1.1 Définition

Une image est une représentation visuelle composée d'un ensemble de points élémentaires appelés "pixels". Chaque pixel possède une intensité qui détermine sa luminosité ou sa couleur.

Dans le cas d'une image en niveaux de gris, chaque pixel est représenté par une valeur numérique allant de 0 (noir, absence totale de lumière) à 255 (blanc, luminosité maximale). Pour une image en couleurs, chaque pixel est défini par une combinaison de trois canaux (Rouge, Vert et Bleu - RGB), chacun variant également entre 0 et 255.

1.2 Premiers pas...

Nous allons ici utiliser OpenCV, une bibliothèque très populaire pour le traitement et la manipulation des images.
Tout d'abord, installez OpenCV via la commande :

```
pip install opencv-contrib-python
```

OpenCV repose sur la bibliothèque NumPy, ce qui permet de manipuler les images comme des tableaux multidimensionnels. Par exemple, il est possible de créer une image noire ou blanche très simplement de la manière suivante :

```python
import cv2
import numpy as np

height, width, channel = 512, 512, 1
black_img = np.zeros([height, width, channel],
    dtype=np.uint8)  # Image noire
white_img = np.ones([height, width, channel],
    dtype=np.uint8)  # Image blanche
white_img *= 255  # Convertir les pixels à 255 (blanc)
```

Notez le type de la matrice (`np.uint8`). Un pixel est en effet codé sur un entier non signé en 8 bits, soit une plage de 0 à 255. Attention donc aux dépassements si vous réalisez des opérations sur les images.

OpenCV permet également de charger une image existante et de l'afficher à l'écran :

```
img = cv2.imread('image.jpg')  # Chargement de l'image
cv2.imshow('Image', img)       # Affichage de l'image
cv2.waitKey()                # Attendre une interaction clavier
cv2.destroyAllWindows()        # Fermer toutes les fenêtres
```

Comme mentionné précédemment, une image RGB est composée de trois matrices, chacune représentant l'un des canaux de couleur (Rouge, Vert et Bleu). Cependant, dans OpenCV, les images sont par défaut en ordre BGR (Bleu, Vert, Rouge).
Vous pouvez récupérer les dimensions de l'image (hauteur, largeur) ainsi que le nombre de canaux de la manière suivante :

```
height, width, channel = img.shape
```

Il est parfois nécessaire de redimensionner une image, soit pour l'agrandir, soit pour la réduire. La fonction `resize` de la bibliothèque OpenCV permet de réaliser cette opération. Voici un exemple :

```
down_width = int(width / 8)
down_height = int(height / 8)

downscaled_img = cv2.resize(img, (down_width, down_height),
    interpolation=cv2.INTER_LINEAR)
resized_img = cv2.resize(downscaled_img, (width, height),
    interpolation=cv2.INTER_LINEAR)
```

Questions :

- Que réalise ce code ?
- Quelle est la différence entre les images `img` et `resized_img` ?

Pour accéder à la valeur de chaque pixel d'une image RGB, il suffit de parcourir chaque ligne et chaque colonne tout en extrayant les valeurs des canaux Bleu, Vert et Rouge à l'aide de leurs indices respectifs :

```
for i in range(height):
    for j in range(width):
        value_B = img[i, j, 0]  # Valeur du canal Bleu
        value_G = img[i, j, 1]  # Valeur du canal Vert
        value_R = img[i, j, 2]  # Valeur du canal Rouge
```

Questions :

- Créez une nouvelle image avec les mêmes dimensions que l'image d'origine, mais comportant un seul canal.
- Remplissez cette image en calculant la moyenne des valeurs des canaux RGB pour chaque pixel. Attention aux dépassement ! Pensez à convertir l'image en `np.uint32` avant de réaliser les calculs et la reconvertir en `np.uint8` ultérieurement. Que remarquez-vous ?
- Proposez une solution basée sur le calcul matriciel pour simplifier et accélérer ce processus.
- Utilisez la fonction `cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)` pour obtenir la même conversion automatiquement. Appelez l'enseignant.

1.3 Binarisation

Pour des raisons pratiques, il est parfois nécessaire de binariser une image. Cela consiste à convertir une image RGB en une image contenant uniquement deux valeurs distinctes : 0 (noir) et 255 (blanc). Ce processus est également appelé segmentation binaire.

L'objectif de la binarisation est de déterminer un seuil permettant de distinguer les pixels en fonction de leur intensité en niveaux de gris. Les pixels dont l'intensité est inférieure au seuil sont attribués à la valeur 0, tandis que ceux dont l'intensité est supérieure ou égale au seuil sont attribués à la valeur 255.

Afin de faciliter la détermination de ce seuil, on peut construire un histogramme représentant la distribution des intensités des pixels. Cet histogramme indique l'occurrence de chaque niveau de gris, ce qui permet d'identifier visuellement des regroupements de pixels ou des pics significatifs correspondant à différentes régions de l'image.

Figure 1: Water_coins.jpg

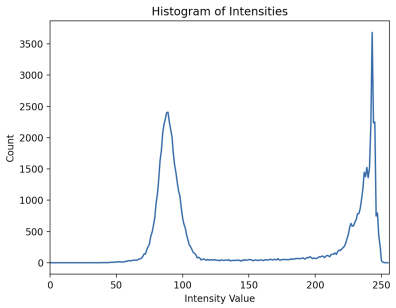

Figure 2: Histogramme d'intensité

Questions :
Nous allons essayer de trouver un seuil permettant de segmenter l'image appelée `water_coins.jpg`. Voici à quoi devrait ressembler votre code :

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

img = cv2.imread('water_coins.jpg', cv2.IMREAD_GRAYSCALE)

# Calcul de l'histogramme ici
histogram = np.zeros(255)

manual_threshold = 0
_, manual_img = cv2.threshold(img, manual_threshold, 255,
    cv2.THRESH_BINARY)

# Méthode d'Otsu ici
otsu_threshold = 0
print(f"Valeur de seuil trouvée avec Otsu : {otsu_threshold}")

cv2.imshow('Original Image', img)
cv2.imshow('Manual Thresholding', manual_img)
cv2.imshow('Otsu Thresholding', otsu_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- Parcourez l'image pixel par pixel et construisez son histogramme d'intensités en comptant l'occurrence de chaque niveau de gris. Affichez le avec matplotlib. Que représentent les deux pics ?

- Déterminez manuellement une valeur seuil qui segmente au mieux l'image en identifiant les regroupements dans l'histogramme.
- Utilisez la méthode d'Otsu[1] pour calculer automatiquement cette valeur seuil et comparez-la à la valeur déterminée manuellement.
  Appelez l'enseignant.

En condition réelle, les images sont souvent bruitées dû à la qualité de la caméra. On peut simuler un bruit gaussien sur notre image comme ceci :

```
img = cv2.imread('water_coins.jpg', cv2.IMREAD_GRAYSCALE)
height, width = img.shape

mean = 0
std_dev = 40
gaussian_noise = np.random.normal(mean, std_dev,
    (height, width)).astype(np.int8)
noisy_img = cv2.add(img, gaussian_noise, dtype=cv2.CV_8UC1)

cv2.imshow('Noisy Image', noisy_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Questions :

- Segmentez l'image bruitée manuellement puis avec la méthode Otsu. Que remarquez-vous ?
- À l'aide de transformations morphologiques[2], essayez d'améliorer votre segmentation. Appelez l'enseignant.

1.4 Convolution et détection de contours

La détection de Canny[3] permet de détecter les contours d'une image en niveau de gris. Voici les étapes de cette méthode de détection :

1. Lisser l'image avec un filtre gaussien pour réduire le bruit.
2. Calculer les gradients horizontaux et verticaux de l'image.
3. Calculer la magnitude des gradients à chaque point de l'image.

[1] https://docs.opencv.org/3.4/d7/d4d/tutorial_py_thresholding.html
[2] https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html
[3] https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html

4. Appliquer un filtrage des non-maximaux pour amincir les bords.
5. Utiliser la double seuillage pour identifier les bords forts, faibles, et supprimer les non-pertinents.

Afin de réduire le bruit d'une image, on peut la convoluer avec un noyau de flou :

```python
import numpy as np
from scipy.signal import convolve2d
import matplotlib.pyplot as plt
import cv2

img = cv2.imread('water_coins.jpg', cv2.IMREAD_GRAYSCALE)

# Noyau de flou
kernel = np.array([[1, 1, 1],
                   [1, 1, 1],
                   [1, 1, 1]]) / 9

# Appliquer la convolution 2D
blurred_img = convolve2d(image, kernel, mode='same',
    boundary='fill', fillvalue=0)
# Convertir l'image en uint8 pour OpenCV
blurred_img = np.uint8(blurred_img)

plt.figure(figsize=(10, 5))
plt.subplot(121)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.subplot(122)
plt.imshow(result, cmap='gray')
plt.title('After Convolution')
plt.show()
```

Les gradients horizontaux G_x et verticaux G_y d'une image peuvent être obtenus grâce aux opérateurs de Sobel, qui utilisent deux noyaux distincts pour filtrer l'image. Le gradient horizontal G_x détecte les changements horizontaux et est calculé avec le noyau suivant :

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * A \tag{1}$$

où A représente l'intensité des pixels de l'image. De manière similaire, le gradient vertical G_y détecte les changements verticaux et utilise le noyau :

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * A \tag{2}$$

L'amplitude des gradients peut quant à elle s'écrire :

$$I = \sqrt{{G_x}^2 + {G_y}^2} \tag{3}$$

Questions :
A partir du code donné ci-dessous :

- Lissez l'image.
- Calculez G_x, G_y et I.
- Comparez l'amplitude des gradients à la fonction `cv2.Canny`. Appelez l'enseignant.

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

img = cv2.imread('water_coins.jpg', cv2.IMREAD_GRAYSCALE)

blurred_img = None
sobel_x = None
sobel_y = None
Gx = None
Gy = None
I =  None

edges = cv2.Canny(blurred_img, 100, 200)

plt.figure(figsize=(12, 4))
plt.subplot(131)
plt.imshow(img, cmap='gray')
plt.title('Original Image')

plt.subplot(132)
plt.imshow(I, cmap='gray')
plt.title('Gradient Magnitude')

plt.subplot(133)
plt.imshow(edges, cmap='gray')
plt.title('Edge Image')
```

1.5 Détection de formes

La transformée de Hough[4] est un outil puissant pour détecter des formes dans des images. Par exemple, cette vidéo montre comment piloter automatiquement un véhicule dans le jeu GTA V en utilisant la détection de lignes. Dans notre cas, nous allons utiliser la transformée de Hough pour détecter des cercles :

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

img = cv2.imread('water_coins.jpg', cv2.IMREAD_GRAYSCALE)
_, binary_img = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY
    + cv2.THRESH_OTSU)
blurred_img = cv2.GaussianBlur(binary_img, (5,5), 2)

circles = cv2.HoughCircles(
    blurred_img,
    cv2.HOUGH_GRADIENT,
    dp=1.2,          # Résolution de l'accumulateur
    minDist=15,      # Distance minimale entre les centres
    param1=30,       # Seuil supérieur pour le Canny
    param2=30,       # Seuil de l'accumulateur
    minRadius=10,    # Rayon minimal des cercles détectés
    maxRadius=50     # Rayon maximal des cercles détectés
)

output_img = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)

if circles is not None:
    circles = np.uint16(np.around(circles))
    for circle in circles[0, :]:
        center = (circle[0], circle[1])
        radius = circle[2]
        cv2.circle(output_img, center, radius, (0, 255, 0), 2)
        cv2.circle(output_img, center, 2, (0, 0, 255), 3)

cv2.imshow('Detected Circles', output_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Questions :

[4] https://fr.wikipedia.org/wiki/Transforme_de_Hough

- Jouez un peu avec les paramètres pour trouver une configuration qui vous convient.
- Est-ce que cette méthode vous semble adaptée pour un jeu interactif ? (La bonne réponse est non !)

1.6 Pour finir, un peu d'animation

Voici un bout de code permettant de déplacer un cercle sur une image :

```
import cv2
import numpy as np

HEIGHT, WIDTH = 500, 500   # Taille de la fenêtre
RADIUS = 10      # Rayon du cercle
PERIOD = 30      # Intervalle entre chaque update en ms

def update_pos(x, y, x_step, y_step):
    if x + RADIUS + x_step < WIDTH:
        x += x_step
    else:
        x = RADIUS

    if y + RADIUS + y_step < HEIGHT:
        y += y_step
    else:
        y = RADIUS

    return x, y, x_step, y_step

canevas = np.ones((HEIGHT, WIDTH, 3), dtype=np.uint8) * 255
center_x, center_y = 0, 0  # Position initiale au centre
x_step, y_step = 2, 6      # Vitesse initiale du cercle

while True:
    # Créer une copie du canevas original pour les dessins
    new_canevas = canevas.copy()

    # Mise à jour de la position du cercle
    center_x, center_y, x_step, y_step = update_pos(center_x,
        center_y, x_step, y_step)

    # Dessiner un cercle rouge sur le nouveau canevas
    cv2.circle(new_canevas, (center_x, center_y), RADIUS,
        (0, 0, 255), -1)  # -1 pour remplir le cercle
```

```
    # Afficher le canevas mis à jour
    cv2.imshow('Game', new_canevas)

    # Attendre un délai et quitter avec la touche 'q'
    if cv2.waitKey(PERIOD) & 0xFF == ord('q'):
        break

cv2.destroyAllWindows()
```

Questions :

- Testez le code.
- Modifiez la fonction `update_pos` pour que le cercle rebondisse lorsqu'il rencontre le bord de l'image.
- Modifiez le code pour ajoutez une balle.
- Modifiez la couleur de la seconde balle en fonction de sa distance avec la première. Plus elle est proche, plus elle est rouge ; plus elle est loin, plus elle est bleue. Appelez l'enseignant.
- **BONUS** Arrêtez les deux balles lorsqu'elles se rencontrent.