Image et Interaction

Cyril Barrelet & Marc Hartley

December 2024

1 Introduction

1.1 Soustraction de fond

Dans de nombreuses situations, nous souhaitons retirer une partie de l'image. Durant ce TP, nous verrons deux situations différentes requierant ce que l'on appelle "la soustraction de fond" (ou "background substraction" en anglais).

Dans un premier temps, on va vouloir, à partir d'une vidéo, conserver uniquement la scène immobile. En ignorant tout les pixels en mouvement, on peut facilement isoler les éléments qui bougent (des passants, des voitures, ...) et les éléments fixes (la rue, le paysage, ...).

Ensuite, on verra comment retirer un fond vert sur une scène avec des acteurs. C'est une étape importante pour pouvoir insérer des effets spéciaux dans les films récents.

1.2 Manipulation de video

Toujours en utilisant OpenCV, on va maintenant voir comment manipuler des videos plutôt que des images.

En informatique, on considère qu'une video n'est qu'une succession d'images (de "frames"). On a alors une image (couleur), qui est un tableau de dimension $N \times M \times C$, qui est prolongée dans une quatrième dimension, le temps. On a alors un tableau de dimension $N \times M \times C \times T$.

Néanmoins, il y a souvent beaucoup trop de frames dans une video pour la stocker en entier dans la RAM. Voyons comment la lire, image par image, avec OpenCV :

Figure 1: Une frame d'une video remplie de voitures

```python
import numpy as np
import cv2

PERIOD = 30

cap = cv2.VideoCapture('vtest.avi')
#   On charge le "reader" de la video
while cap.isOpened(): # Tant que la video n'est pas finie,
    ret, frame = cap.read()
    #   On lit la "frame" suivante dans la vidéo.
    #   Cette fonction renvoie 2 valeurs : est-ce que nous
    #   avons pu lire l'image, ainsi que l'image en elle même

    if not ret:
        print("Oh non... On n'a pas réussi à lire l'image...")
        break

    cv2.imshow('frame', frame)
    if cv2.waitKey(PERIOD) == ord('q'):
        # Un appui sur "q" ferme le programme
        break

cap.release()
# On peut dire au "reader" de lacher le fichier video
cv2.destroyAllWindows()
```

Questions :

- Que réalise ce code ?
- Que contient la variable ”frame” ?
- À quel endroit de ce code allons-nous insérer du code pour manipuler la video ?

Exercices :

- Téléchargez la video `traffic_CCTV.mp4` et utilisez-la dans votre code.
- Affichez la vidéo normalement, en niveau de gris, puis en noir et blanc (vous pouvez utiliser un seuil manuel ou avec Otsu).

1.3 La vidéo moyenne

D’après vous, qu’est-ce que la ”moyenne d’une video”? Peut-être qu’il est plus intéressant de calculer la moyenne sur chaque pixels tout au long de notre video. Changeons légèrement notre programme pour stoker une image moyenne :

```python
cap = cv2.VideoCapture('CCTV.mp4')

ret, frame = cap.read()
# On va juste regarder la première frame pour connaitre la
#   taille de l'image
height, width, channel = ???

nombre_de_frames = 0
frame_moyenne = np.zeros((height, width, channel), dtype=np.uint32)
# On commence avec une image nulle. On utilise un uint32 parce
#   qu'on risque d'avoir des grandes valeurs dedans

while cap.isOpened(): # Tant que la video n'est pas finie,
    ret, frame = cap.read()
    if not ret:
        break

    nombre de frames += 1
    frame_moyenne = ???
    # On ajoute les valeurs sur chaque frame, pour
    #   pouvoir faire la moyenne ensuite

    # cv2.imshow('frame', frame)
    # Sert à rien d'afficher notre video, maintenant
cap.release()
# On peut dire au "reader" de lacher le fichier video

frame_moyenne = ???
# On fait la moyenne des frames

cv2.imshow("Image moyenne", frame_moyenne.astype(np.uint8)
cv2.waitKey(0)

cv2.destroyAllWindows()
```

Questions :

- Selon vous, le résultat serait-il différent avec une médiane plutôt qu'une moyenne?
- Quel est le problème majeur du calcul par la médiane?

Figure 2: Resultat attendu de la moyenne de la vidéo

1.4 Differencier le fond

Super, on a réussi à isoler le fond de notre vidéo. C'est-à-dire ce qui reste immobile tout du long de la vidéo. Maintenant, souvenez-vous de l'objectif de ce TP, c'est le "background substraction". Alors essayons de soustraire le fond!

Dans notre code, on va relancer une nouvelle boucle de vidéo, et voir ce que ça fait sans le fond. Attention, réfléchissez aux possibles problèmes que peut poser la soustraction de deux "uint" (entier non signés) !!

```
# Ne pas oublier la première boucle de lecture video
#   ici pour pouvoir calculer la frame moyenne

cap = cv2.VideoCapture('CCTV.mp4')
while cap.isOpened():  # Tant que la video n'est pas finie,
    ret, frame = cap.read()
    if not ret:
        break

    difference = ???
    cv2.imshow('frame', difference.astype(np.uint8))
    if cv2.waitKey(PERIOD) == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

Questions :

- Que devrions-nous voir ?
- Comment avez-vous géré le fait que les valeurs de pixel puissent être négatives ? Peut-être que la fonction `np.clip` peut vous être utile.
- Quel problème peut-on voir avec la détection de fond vers la fin de la vidéo ?

Voici le code que propose OpenCV pour appliquer la soustraction de fond :

```python
import cv2

PERIOD = 30

backSub = cv2.createBackgroundSubtractorMOG2()

capture = cv2.VideoCapture("CCTV.mp4")
if not capture.isOpened():
    exit(0)

while True:
    ret, frame = capture.read()
    if frame is None:
        break

    fgMask = backSub.apply(frame)

    # cv2.imshow('Frame', frame)
    cv2.imshow('FG Mask', fgMask)

    keyboard = cv2.waitKey(PERIOD)
    if keyboard == 'q' or keyboard == 27:
        break
```

Questions :

- Au niveau du code, quelle est la principale différence avec notre code ?
- Comment pensez-vous qu'il soit possible que la vidéo ne soit lue qu'une unique fois ?
- Quels sont les avantages et quels sont les inconvéniants de leur méthode ? Notez qu'en remplaçant `cv2.waitKey(PERIOD)` par `cv2.waitKey(0)`, vous pouvez avancer la vidéo frame par frame.
- Allez voir la documentation de la fonction `createBackgroundSubtractorMOG2`[1]. Quels sont les paramètres qui peuvent être modifiés, et quelle peut être leur utilité? Quelle autre fonction peut être utilisée à la place ?

[1] `https://docs.opencv.org/4.x/de/de1/group__video__motion.html`

1.5 (Vraiment optionnel) Imiter le fonctionnement de OpenCV

On peut essayer d'appliquer nous même le processus qu'utilise OpenCV. Dans leur cas, le calcul de l'image de fond *évolue* dans le temps. Vous avez peut-être vu le paramètre "history" dans la fonction, elle décrit combien de frames vont être stockées dans un tampon (ou *buffer* en anglais) et seront utilisées pour calculer le fond.

Voilà un morceau de code à compléter :

```
PERIOD = 30
HISTORY = 10 # Nombre de frame a conserver dans le buffer

cap = cv2.VideoCapture('CCTV.mp4')
ret, frame = cap.read()

height, width, channel = frame.shape
nombre_de_frames = 0

# On cree un tampon (de taille 0 au début), en précisant
#   que les éléments auront la taille d'une image normale
buffer = np.zeros((0, height, width, 3), dtype=np.uint8)

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    # On va ajouter la nouvelle frame dans le buffer
    #   (cherchez la documentation de np.append)
    buffer = ???

    # Mais si on a déjà trop de frames, on va devoir
    #   retirer la toute première
    #   (cherchez la documentation de np.delete)
    if nombre_de_frames >= HISTORY:
        buffer = ???

    print(buffer.shape)
    # Verifie que le buffer ne dépasse pas HISTORY

    # Puis on fait la moyenne du buffer
    frame_moyenne = np.average(buffer, axis=0)

    nombre_de_frames += 1

    # Calcul de la difference, comme avant
    difference = np.clip(
        frame.astype(np.int32) - frame_moyenne
```

```
            .astype(np.int32),
        0, 255)
    cv2.imshow('fond', frame_moyenne.astype(np.uint8))
    cv2.imshow('difference', difference.astype(np.uint8))
    if cv2.waitKey(PERIOD) == ord('q'):
        break
cap.release()
```

Vous remarquerez possiblement que le temps de calcul devient bien trop long, même en conservant une cinquantaine de frames dans le buffer. Trois solutions s'offrent à vous : **préférer les fonctions complètement optimisées d'OpenCV** (fortement conseillé), s'intéresser aux estimations de moyennes (notamment par "moyenne incrémentale"), ou modifier la taille de l'image avec `frame = cv2.resize(frame, (200, 100))` (vous comprenez pourquoi les caméra de sécurité ont une résolution très faible?).

1.6 Retirer un fond vert

Au cinema, l'utilisation de fonds verts est très fréquente. Cela permet de détecter automatiquement la partie où il y a un acteur et la partie où il y a le fond, qu'on peut modifier par un fond virtuel.

Pourquoi un fond vert? Parce que on a juste à verifier le canal "vert" de nos images pour l'isoler!

Isolons le canal vert de notre image, et stockons cela dans une image en niveau de gris :

```
img_initiale = cv2.imread('green_screen.jpg')
height, width, channel = img_initiale.shape

canal_vert = np.zeros((height, width), dtype=np.uint8)

for i in range(height):
    for j in range(width):
        # On extrait le canal vert
        canal_vert[i, j] = img_initiale[i, j, ???]
cv2.imshow('Canal vert', canal_vert(np.uint8))
cv2.waitKey() # Attendre une interaction clavier
cv2.destroyAllWindows() # Fermer toutes les fenêtres
```

Questions :

- Est-ce que le fond se distingue bien?
- En passant la souris sur l'image affichée, quelle est approximativement la valeur du canal vert sur le fond?

Maintenant, on va réaliser ce qu'on appelle le "masque binaire", c'est-à-dire binariser le canal vert pour obtenir une image dont les pixels valent "1" s'ils appartiennent au fond ou valent "0" s'ils appartiennent à l'acteur. Plusieurs choix s'offrent à vous : la méthode manuelle, la méthode d'Otsu, ou encore d'autres? Utilisez les fonctions OpenCV (ou les votres) pour tester[2] :

[2] https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html

```
# Binarisation "à la main"
THRESHOLD = ???
_, masque_vert = cv2.threshold(canal_vert, THRESHOLD, 255,
    cv2.THRESH_BINARY)

# Binarisation avec Otsu
_, masque_vert = cv2.threshold(canal_vert, 0, 255,
    cv2.THRESH_BINARY + cv2.THRESH_OTSU)

# Binarisation adaptative proposee par OpenCV
masque_vert = cv2.adaptiveThreshold(canal_vert, 255,
    cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY,11,2)
```

Questions :

- Dans le cas du fond vert, quelle méthode propose le meilleur résultat? Et pourquoi ?

Exercices :

- Améliorez le résultat du masquage en utilisant des opérations morphologiques (érosion, dilatation)[3].

1.7 Remplacer le fond d'une image

On est maintenant capables de savoir quelle partie de notre image doit être remplacée. Mais il faut aussi trouver comment les remplacer!

La solution est simple : on parcourt chaque pixel (i,j) et si le masque vaut "0" à cette position, on affiche l'image initiale, sinon on affiche l'image d'un fond virtuel! Le résultat est une "image composite".

```
# On charge une image pour le fond virtuel
img_fond = cv2.imread(???)
# On la redimensionne pour avoir la même
#   taille que l'image d'origine
img_fond = cv2.resize(???)

# Creation d'une image finale
img_composite = np.zeros((height, width, channel),
    dtype=np.uint8)
```

Et maintenant, plus qu'à parcourir chacun de nos pixels!

[3] https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html

```
for i in range(height):
    for j in range(width):
        if ??? == 0:
            # Si le masque == 0, c'est que c'est l'acteur
            img_composite[i, j] = ???
        else:
            # Sinon, c'est le fond virtuel
            img_composite[i, j] = ???
cv2.imshow('Image finale', img_composite.astype(np.uint8))
cv2.waitKey() # Attendre une interaction clavier
cv2.destroyAllWindows() # Fermer toutes les fenêtres
```

Le résultat attendu devrait ressembler à cela, par exemple :

Figure 3: Image composite

Toujours déçu de ces quelques pixels verts sur les bords de la silhouette, nous rappelant les vieux effets spéciaux des années 70? On va remédier à ça en utilisant un type de convolution maintenant connu, le flou.

Le problème avec ces pixels verts, c'est qu'ils ne se fondent pas bien dans le paysage. Alors on va appliquer un peu de flou sur notre masque. De cette façon, on va "lisser" la silhouette de notre acteur :

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

```
# Appliquer la convolution 2D
masque_flou = scipy.signal.convolve2d(masque_vert, kernel,
        mode='same', boundary='fill', fillvalue=0)
```

Si on regarde notre code pour créer l'image composite, on voit bien que le "if" va forcément causer une transition brutale d'un pixel à l'autre, alors qu'on vient essayer d'avoir un masque "continu". On va utiliser, à la place du "if", une "interpolation linéaire".

Je vous laisse fouiller ce qu'est une interpolation linéaire [4], mais voilà ce que l'on peut retenir : La valeur d'un pixel est un mélange (une moyenne pondérée) entre la couleur A (image Acteur) et la couleur B (image Fond virtuel). Cette pondération est donnée par un "interpolant" t, c'est-à-dire une valeur entre 0 et 1 qui définit si on est plutôt influencé par la couleur A ou la couleur B. La formule pour connaitre la couleur finale devient $resultat = A * (1 - t) + B * t$. Ici on peut utiliser le masque comme valeur de t.

```
masque_normalise = mask[i, j] / 255
img_composite[i, j] = img_initiale[i, j] * (1 - masque_normalise) +
                      img_fond[i, j] * masque_normalise
```

Moui... pas fou, comme résultat, on a l'impression que le vert est encore plus présent... Et oui, les pixels du masque qui étaient "blancs" mais proches de l'acteur sont maintenant "dilués". Essayons de passer le masque en 2 temps : si le pixel du masque est "blanc", on affiche le fond; si le pixel du masque est noir, alors on applique la strategie du flou:

```
masque_flou = np.maximum(     # On prend le maximum entre ...
                    masque_vert,  # Masque binaire
                    scipy.signal.convolve2d(
                                  # Et le masque flouté
                        masque_vert, kernel,
                        mode='same', boundary='fill',
                        fillvalue=0
                    )
            )
```

Le résultat, en utilisant le même code, devrait être amélioré :

[4] https://en.wikipedia.org/wiki/Linear_interpolation

Figure 4: Image composite avec un masque flouté

1.8 Intégrer la balle rebondissante en fond d'image

Reprenez votre code du précédent TP affichant une balle rebondissante, et utilisez le caneva comme image de fond. Bravo, vous avez incrusté, aussi simplement que ça, un acteur dans un jeu vidéo!

Figure 5: Une balle rebondissante derrière un acteur sur fond vert

1.9 Plus de souplesse dans le fond vert

Avoir un fond vert, c'est bien, mais vous n'avez peut-être pas un mur avec un vert "parfait" chez vous. On a alors chercher à isoler les pixels qui se rapprochent d'une couleur quelconque.

On commence par définir la couleur que l'on souhaite isoler (attention, OpenCV utilise la norme Bleu-Vert-Rouge [BGR] plutôt que la version plus commune du Rouge-Vert-Bleu [RGB]):

```
couleur_cible = np.array([bleu, vert, rouge])
```

Et maintenant, créons notre masque. Plutôt que d'extraire simplement le canal vert, on peut regarder la difference entre la couleur d'un pixel et la couleur cible. Attention encore, les pixels ont 3 valeurs (BGR), donc on les considère comme des vecteurs à 3 dimensions. Comment calculer la distance (ou "norme de la difference") entre deux vecteurs? Vous pouvez chercher la documentation de la fonction `np.norm`.

```
# Dans la boucle pour calculer le masque :
masque[i, j] = ???

# Puis en sortie de programme :
cv2.imshow('Masque', masque)
```

```
cv2.waitKey() # Attendre une interaction clavier
cv2.destroyAllWindows() # Fermer toutes les fenêtres
```

Si jamais vous vous attendiez à avoir des pixels blancs au lieu de pixels noirs, c'est qu'il faut inverser la valeur de notre masque. Une formule comme `masque[i, j] = 255 - ???` suffira possiblement!

Pour aller plus loin : Même si vous avez choisi une bonne couleur cible, vous aurez possiblement des surprises si la luminosité de votre vidéo baisse un peu. Pour remédier à cela, on passe généralement par d'autre représentation de la couleur. On a vu le RGB/BGR, mais une autre très utilisée est le HSV (Hue-Saturation-Value).

La fonction `cv2.cvtColor(img, cv2.COLOR_BGR2HSV)` vous permet de changer d'espace couleur (representation de la couleur). Cherchez à utiliser le premier canal (Hue) pour binariser votre masque!

1.10 Bonus pour ceux qui lisent jusqu'au bout

Il est bien connu que le language Python est généralement lent. Pardon, une *mauvaise utilisation* de Python est lente.

Dans notre cas, mesurons la vitesse de calcul avec la fonction `time.time()` :

```
import time
# Les autres imports ...
# Quelques morceaux de codes ...

debut = time.time()

# ICI, le code qu'on veut mesurer

fin = time.time()

print(f"Le code prend : {int((fin - debut) * 1000)} millisecondes")
```

Sur mon PC, en ignorant les `cv2.imread`, ainsi que `cv2.imshow`, mon code prend environ 3000 millisecondes. Pour une vidéo à 30 FPS (frames par seconde), soit environ une image toutes les 30 millisecondes, c'est beaucoup trop élevé! On se retrouve à environ 0.3 FPS.

Le secret, encore une fois, c'est d'utiliser les fonctions OpenCV et éviter les boucles dans notre code. Voilà un exemple qui tourne en moins de 30 millisecondes :

```
import time
import cv2
import numpy as np

img_initiale = cv2.imread('green_screen2.jpg')
height, width, channel = img_initiale.shape
img_fond = cv2.imread("new_york.jpg")
img_fond = cv2.resize(img_fond, (width, height), interpolation=cv2.INTER_LINEAR)

debut = time.time()

canal_vert = img_initiale[:, :, 1]
THRESHOLD = 150
_, masque_vert = cv2.threshold(canal_vert, THRESHOLD, 255, cv2.THRESH_BINARY)
kernel = np.ones((7,7),np.uint8)
masque_vert = cv2.erode(masque_vert, kernel, iterations = 1)
masque_vert = cv2.dilate(masque_vert, kernel, iterations = 1)
masque_vert = masque_vert / 255
masque_vert = np.maximum(masque_vert, cv2.GaussianBlur(masque_vert, (11, 11), 2.0))
masque_vert = masque_vert.reshape((height, width, 1))
img_composite = img_initiale * (1 - masque_vert) + img_fond * masque_vert
```

```
fin = time.time()

print(f"Le code prend : {int((fin - debut) * 1000)} millisecondes")
```