Séance 14 :

Suivi de cible par drone

Avant toute chose :

- J'ai fait ce TP deux fois, une à ma manière grâce à la librairie `opencv`, et une en suivant les consignes avec `PIL`. Je ne détaille ici que la deuxième version.
- Voici la strucutre de la séance :

```
>
  > helpers
    > image.py
    > path.py
    > plotter.py

  > ressources
    > im1.png
    > im2.png

  > main.py
```

  - `helpers/image.py` contient toutes les fonctions nécessaires à la réalisation des questions
  - `helpers/path.py` contient un bout de code pour transformer un chemin relatif en chemin absolu :

> helpers/path.py

```python
import os

def r(path):
    dirname = os.path.dirname(__file__)
    return os.path.join(dirname, '..', path)
```

  - `helpers/poltter.py` est un module que j'ai écrit qui permet d'afficher simplement plusieurs images en même temps (C.F. annexe)
  - `helpers/poltter.py` main.py contient tout le code à exécuter

1. Préparatifs

1. ***Importer l'image*** `im1.png`.
2. ***Décomposer l'image en 3 tableaux*** `numpy` ***(*** `R`***,*** `G` ***et*** `B`***) contenant chacun une des trois couleurs Rouge Vert et Bleu.***
3. ***Déterminer la taille en pixel de l'image.***
   On importe le nécessaire :

> main.py

```python
# Imports ---------------------------------------------------------------------
import PIL.Image as im
import numpy as np
import matplotlib.pyplot as plt
from helpers.plotter import Plotter as pltr
from helpers.path import r
```

   On instancie les constantes du fichier :

> main.py

```python
# Contantes -------------------------------------------------------------------
# lien vers images
IMAGE_1_PATH = r('ressources/im1.png')
IMAGE_2_PATH = r('ressources/im2.png')
```

On fait les préparatifs :

> main.py

```
# Preparatifs ------------------------------------------------------------------
# On ouvre l'image
i = np.array(im.open(IMAGE_1_PATH))

# On decompose l'image
R, G, B = decomposeRGB(i)

# On determine sa taille
h, w = R.shape[0], R.shape[1]
```

2. Recherche de la couleur de la voiture

1. ***Faire afficher dans une même fenêtre externe l'image initiale en couleurs et les 3 images*** `R`, `G` ***et*** `B` ***en niveaux de rouge, vert bleu.***

> helpers/image.py

```
import numpy as np

def decomposeRGB(A):
    h, w, p = A.shape
    R = np.zeros_like(A)
    G = np.zeros_like(A)
    B = np.zeros_like(A)
    for i in range(h):
        for j in range(w):
            R[i][j][0] = A[i][j][0]
            G[i][j][1] = A[i][j][1]
            B[i][j][2] = A[i][j][2]
    return R, G, B
```

> main.py

```
# Imports ----------------------------------------------------------------------
from helpers.image import decomposeRGB

# Recherche de la couleur ------------------------------------------------------
# On cree le plot
P = pltr()

# On ajoute les images
P.addSubplot(i)
P.addSubplot(R)
P.addSubplot(G)
P.addSubplot(B)

# On affiche le tout
P.show()
```

Ce qui donne : (j'ai éclaircit l'image pour l'impression)

2. ***En faisant glisser la souris sur la voiture, on peut lire les niveaux de chaque couleur, en déduire 3 fourchettes approximatives dans lesquelles sont comprises les 3 couleurs R, G, B.***
Je trouve :

$$\begin{cases} R &= 132 \\ G &= 25 \\ B &= 47 \end{cases}$$

3. Seuillage

1. ***A partir des fourchettes établies précédemment, transformer l'image en noir et blanc ou seule la voiture sera blanche.***
Toujours pour l'impression, j'ai fait l'inverse au niveau des couleurs :

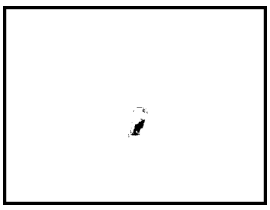
> helpers/image.py

```
1 def seuillage(i, r, g, b, p):
2
3     SEUIL_MAX = [r+p, g+p, b+p]
4     SEUIL_MIN = [r-p, g-p, b-p]
5
6     plus = np.where(np.any(i > SEUIL_MAX, axis=-1), 255, 0)
7     moins = np.where(np.any(i < SEUIL_MIN, axis=-1), 255, 0)
8
9     return np.logical_or(plus, moins)
```

> main.py

```
1 # Seuillage ------------------------------------------------------------------
2 r, g, b, p = 132, 25, 47, 20
3 mask = seuillage(i, r, g, b, p)
4
5 P.addSubplot(mask)
6 P.show()
```

J'obtiens alors :



4. Position de la voiture dans l'image

1. ***Par la méthode de votre choix, écrire une fonction qui retourne les coordonnées (X,Y) en pixels du centre de la voiture par rapport au centre de l'image.***

> helpers/image.py

```
1 def position(mask):
2     Ymax, Xmax = np.max(np.where(mask == 0), axis=-1)
3     Ymin, Xmin = np.min(np.where(mask == 0), axis=-1)
4
5     # Pour tracer le rectangle
6     X = [Xmax, Xmax, Xmin, Xmin, Xmax]
7     Y = [Ymax, Ymin, Ymin, Ymax, Ymax]
8
9     # Pour placer le point par rapport a l'origni
10    x, y = (Xmin + Xmax) / 2, (Ymin + Ymax)/2
11
12    # Point par rapport au centre de l'image
13    h, w = mask.shape
14    xi, yi = x-w/2, y-h/2
15    return X, Y, x, y, xi, yi
```

> main.py

```
# Position ------------------------------------------------------------------
X, Y, x, y, xi, yi = position(mask)

plt.plot(X, Y, 'b')
plt.plot(x, y, 'bx')
plt.imshow(i, alpha=0.3)
plt.show()
```

Remarque : Cette fois-ci j'utilise `plt` et non mon `Plotter` car il ne peut pas afficher des listes de points... Et j'obtiens l'image de gauche que j'ai agrandi à droite.

5. Données de commande pour recentrer l'objectif de la caméra

1. ***Ecrire une fonction qui retourne un quadruplet de booléenscontenant les données L+, L-, H+ ou H- indiquant si la nacelle doit être réorientée suivant la longueur ou la hauteur de l'image et suivant le sens.***

> helpers/image.py

```
def recenter(h, w, x, y, O):
    Lp, Lm, Hp, Hm = False, False, False, False
    if (x > O*w):
        Lp = True
    elif (x < -O*w):
        Lm = True

    if (y > O*h):
        Hp = True
    elif (y < -O*h):
        Hm = True

    return Lp, Lm, Hp, Hm
```

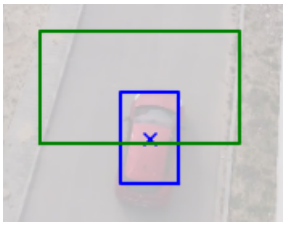
> helpers/image.py

```
# Recentrage ----------------------------------------------------------------
# Pourcentage de l'image
O = 0.25

Lp, Lm, Hp, Hm = recenter(h, w, xi, yi, O)
print(Lp, Lm, Hp, Hm)

# On rajoute le rectangle vert de la 'safezone'
plt.plot([(w/2)*(1-O), (w/2)*(1+O), (w/2)*(1+O), (w/2)*(1-O), (w/2)*(1-O)],
         [(h/2)*(1+O), (h/2)*(1+O), (h/2)*(1-O), (h/2)*(1-O), (h/2)*(1+O)],
         'g')
plt.show()
```

J'obtiens alors :

```
False, False, False, False
```

J'ai effectué en zoom sur la partie intéressante, on voit alors que le centre de la voiture est tout juste dans le rectangle vert, ce qui explique qu'il ne faille pas bouger la caméra.



6. Recadrage de l'image

1. ***Ecrire une fonction qui retourne une image couleur qui a pour dimension la moitié de l'image initiale et qui place le véhicule au centre si l'excentration le permet. Sinon, on montrera l'image au mieux.***
2. ***Faire afficher cette image.***

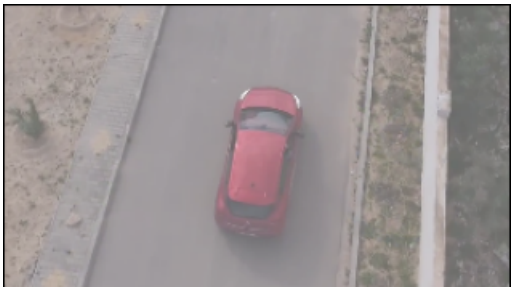
> helpers/image.py

```
def crop(i, xi, yi):
    h, w, p = i.shape

    # coordonnees de la nouvelle origine
    x, y = xi, yi

    # si l'excentration est trop grande, il ne faut pas deborder
    if(xi > w/4):
        x = w/4
    if(xi < -w/4):
        x = -w/4
    if(yi > h/4):
        y = h/4
    if(yi > h/4):
        y = h/4

    a = int(w/4 + x)
    b = int(3*w/4 + x)
    c = int(h/4 + y)
    d = int(3*h/4 + y)

    return i[c:d, a:b]
```

> main.py

```
# Recadrage ------------------------------------------------------------------
B = crop(i, xi, yi)
plt.imshow(B, alpha=0.6)
plt.show()
```

Sans aucun recadrage cette fois-ci :



7. Annexe

Voici le contenu du fichier `helpers/poltter.py`

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


class Plotter:

    def __init__(self, name=''):
        self.imageToPlot = []
        self.name = name
        plt.rcParams['toolbar'] = 'None'

    def addSubplot(self, image, name='', axis=False):
        if type(image) == PIL.JpegImagePlugin.JpegImageFile:
            image = np.array(image)
        cmap = 'gray' if image[0][0].size == 1 else 'viridis'

        self.imageToPlot.append(ImageToPlot(image, cmap, name, axis))

    def show(self):
        plt.suptitle(self.name, fontsize=22, fontweight=4, color='#2c3e50')
        l = len(self.imageToPlot)
        nbOfColumn = int(np.sqrt(l))
        nbOfLines = l // nbOfColumn
        nbOfLines = nbOfLines if l % nbOfColumn == 0 else nbOfLines + 1

        i = 0
        for img in self.imageToPlot:
            i += 1
            plt.subplot(nbOfColumn, nbOfLines, i)
            plt.imshow(img.image, cmap=img.colorMode)
            if not img.axis:
                plt.axis('off')
                plt.title(img.name, fontweight='bold', color='#27ae60')
                plt.margins(0, 0)
                plt.subplots_adjust(top=1, bottom=0, right=1, left=0,
                                    hspace=0, wspace=0)
            pass

        figManager = plt.get_current_fig_manager()
        figManager.window.showMaximized()
        plt.show()


class ImageToPlot:
    def __init__(self, image, colorMode, name, axis):
        self.image = image
        self.colorMode = colorMode
        self.name = name
        self.axis = axis
```