DM3. Reconnaissance d'image pour suivi de cible par drone

Ce devoir est à réaliser sur un environnement python de votre choix (Spyder, Pyzo, ...) à installer sur votre ordinateur personnel. Le travail est à rendre sous la forme d'un script.

INSTRUCTIONS

- Votre fichier python s'appellera `nom_prenom_dm3.py` (tout en minuscule et sans accent).
- Vos remarques seront inscrites **dans le script** sous forme de commentaires avec un #.
- Les tests réalisés doivent être visibles **dans le script** du programme.
- Les noms de variables donnés dans l'énoncé doivent être strictement respectés.
- Télécharger les images brutes nécessaire au DM `im1.png` et `im2.png` ainsi que le fichier `detection_centre.py` sur le site `https://ptsilamartin.github.io`.
- Les parties 6 et 7 sont facultatives.
- Les fonctions prédéfinies `max`, `min` et `sum` ne sont pas autorisées.
- les erreurs de syntaxe au lancement de votre script doivent être corrigées. Le script doit s'exécuter sans erreur. Commenter les lignes qui posent problème si nécessaire.

Date limite d'envoi du fichier : Jeudi 6 mars au soir

Pour rendre le DM déposer votre script sur l'ENT, dans la zone Moodle de votre classe.
(Solution de repli : `https://nuage03.apps.education.fr/index.php/s/CfHKx2a6JaPE3ki`).

1 Présentation

L'utilisation de drones pour filmer une scène (événements sportifs, publicité, actions militaires...) est de plus en plus répandue car elle permet d'avoir des angles de vues plus intéressants et au plus proche de l'action, en s'affranchissant d'éventuels obstacles (foule, etc).

La fonctionnalité « **suivi de cible** » permet de placer systématiquement et automatiquement le sujet à filmer (par exemple le véhicule en tête d'une course automobile) au centre de l'image. Le retour d'image fourni par la caméra permet de connaitre les mouvements de la cible dans l'image et d'en déduire les déplacements à imposer à la caméra afin de recentrer automatiquement la cible sur l'image.

2 Objectif

Divers algorithmes de tracking sont utilisés pour arriver à détecter les mouvements de la cible à partir de l'image filmée (algorithme KLT [Kanade Lucas 1992], RMRm [Obodez 1995],...).
On s'intéressera à un algorithme simplifié basé sur une technique de traitement d'image. Pour l'utiliser, il faut que l'objet à reconnaître ait un caractère particulier (forme, couleur, ...) de façon à le différencier du fond : dans l'exemple ci-dessous l'objet à détecter est le véhicule, caractérisé par sa couleur rouge.

2.a Seuillage

Dans un premier temps l'image est seuillée : les pixels dont la couleur s'approche du rouge sont différenciés des autres comme le montre l'image ci-dessous :


C'est cette étape qui sera traitée dans ce DM.

2.b Détection du centre

Il suffit alors de détecter le centre de l'objet à partir de l'image seuillée en effectuant un calcul de barycentre par exemple.

3 Prise en main des images à tester et de quelques fonctions utiles

Cette partie vous permet de prendre en main les différents outils utiles à ce DM. Aucune rédaction particulière n'est demandée dans votre script pour cette partie.

3.a Import des bibliothèques

Utiliser les lignes suivantes pour importer les bibliothèques nécessaires au DM :

PYTHON
```
import matplotlib.pyplot as plt          #tracé de courbes
import matplotlib.image as mpimg         #visualisation des images
import numpy as np
```

3.b Lecture de l'image test

Il est important de placer les fichiers à lire (ici les fichiers `im1.png` et `im2.png`) dans le même répertoire que votre script python.
Si vous utilisez l'environnement **Pyzo**, il faut - au moment d'exécuter - effectuer un clic droit sur l'onglet de votre fichier et choisir `"Run file as script" ou "Exécuter en tant que script"`.

Prendre connaissance de l'image brute stockée dans le fichier `"im1.png"`. Vous pouvez utiliser les instructions suivantes pour lire une image à partir d'un fichier :

PYTHON
```
#lecture de l'image
img1=mpimg.imread('im1.png')

# affichage de l'image
plt.figure()
plt.imshow(img1)
plt.show()
```

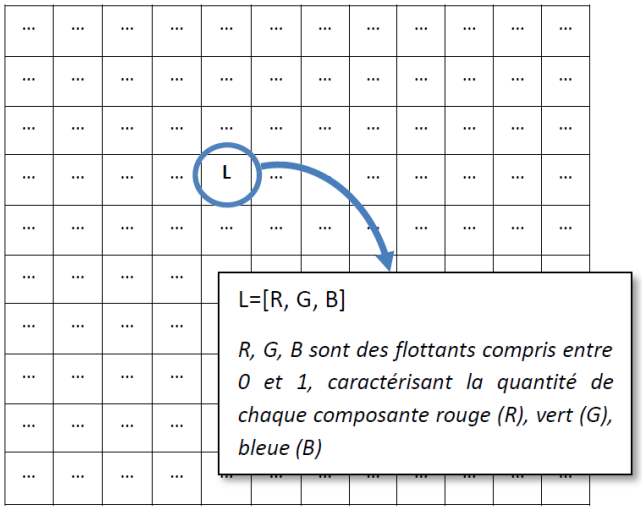
Vous remarquerez le sens des axes sur l'image affichée.

3.c Analyse de la structure de l'image

Les images avec l'extension `png` *« portable network graphics »* sont enregistrées sous forme matricielle avec différents formats.
Le format des images utilisées pour le DM est RGB float32 qui code sur 32 bits chaque couleur.

PYTHON
```
array([[[ 0.29411766,  0.27058825,
0.27058825],[ 0.32156864,  0.29411766,
0.30588236],...,[ 0.22352941,  0.16862746,
0.16862746]]], dtype=float32)
```

Les 3 valeurs de chaque liste correspondent aux trois couleurs **RGB** (Red, Green, Blue) de chaque pixel de l'image.



Structure de la matrice image seuillée

Les valeurs des trois couleurs sont comprises entre 0 et 1.

	R	G	B	Couleur obtenue
[0,0,0]	0	0	0	Noir
[1,1,1]	1	1	1	Blanc
[1,0,0]	1	0	0	Rouge
[0.3,0.3,0.3]	0.3	0.3	0.3	Gris

On trouve aussi le codage **RGBa** qui ajoute une quatrième composante associée à la transparence.
Vérifier cette structure et tester la méthode `shape` :

PYTHON
```
print (img1[0,0])
print (img1[200,300])
print (img1.shape)
```

4 Seuillage à partir d'une image au format RGB

L'image seuillée est une image en noir et blanc, obtenue par seuillage de l'image brute. Ce seuillage permet de distinguer la forme à reconnaître du reste de l'image.
Ici, l'objet à reconnaître est caractérisé par sa couleur. Tous les pixels situés dans une plage de couleur définie deviennent des pixels blancs sur l'image seuillée. Les autres sont noirs.
La qualité de la détection du centre de la forme est fortement conditionnée par la qualité de l'image seuillée. Sur l'image seuillée, la forme à reconnaître doit apparaître clairement en blanc.

4.a Définition des seuils

Pour définir les seuils de couleurs, on propose d'utiliser une liste de 6 éléments :

`seuilsRGB` = [Rmin, Rmax, Gmin, Gmax, Bmin, Bmax]

Les pixels vérifiant les conditions suivantes sur leurs composantes R, G et B, seront considérés comme appartenant à l'objet à détecter et auront leurs valeurs fixées à 1 sur l'image seuillée :

$$R \in [Rmin, Rmax] \; ; \; G \in [Gmin, Gmax] \; ; \; G \in [Bmin, Bmax]$$

Les autres pixels seront noirs (valeurs fixées à 0).

Question 1. L'objet à détecter est de couleur rouge, on propose de conserver les pixels contenant un niveau de rouge supérieur à 0.5. Aucune condition n'est imposée sur les autres composantes (G et B). Définir la liste `seuilsRGB` correspondante.

Question 2. Écrire une fonction `seuillage_pixel(pixelRGB:np.ndarray,seuil:list)` qui prend pour argument un tableau de 3 valeurs définissant la couleur d'un pixel et une liste de seuils. Cette fonction renvoie une liste `[0,0,0]` ou `[1,1,1]` suivant la stratégie définie ci-dessus.

Question 3. Créer un pixel avec l'instruction pixel=np.array(`[a,b,c]`) où a, b et c sont des flottants pris dans l'intervalle `[0, 1]`. Tester votre fonction `seuillage_pixel(pixelRGB, seuil)` et copier votre test dans votre script.

Question 4. Écrire une fonction `seuillage(img:np.ndarray,seuil:list)` qui prend pour argument une matrice image brute `img` et une liste de seuils et qui renvoie l'image seuillée en noir et blanc sous forme matricielle. **L'image brute ne sera pas modifiée**. Vous pourrez utiliser la fonction `seuillage_pixel(pixelRGB:np.ndarray,seuil:list)`.

Indication : Pour créer l'image seuillée sans modifier l'image donnée en argument, une des solutions est de commencer par créer une nouvelle image noire ayant les mêmes dimensions que `img` comme indiqué ci-dessous :

PYTHON
```
# les dimensions de l'image matricielle sont obtenues avec shape
nbligne,nbcolonne,pixel=img.shape

im_s=np.zeros((nbligne,nbcolonne,pixel))
# im_s est une matrice de 0 de dimensions nbligne × nbcolonne × pixel
```

Question 5. Tester cette fonction sur l'image `img1` fournie. Afficher l'image obtenue et analyser la qualité du seuillage. Indiquer en commentaire dans votre script en quelques lignes pourquoi le seuillage à partir d'une image au format RGB ne fonctionne pas bien. Sauvegarder votre image sous le nom de fichier `seuillage1.png`.

L'instruction `mpimg.imsave('fichier.png', img)` permet de sauvegarder une image dans un fichier.

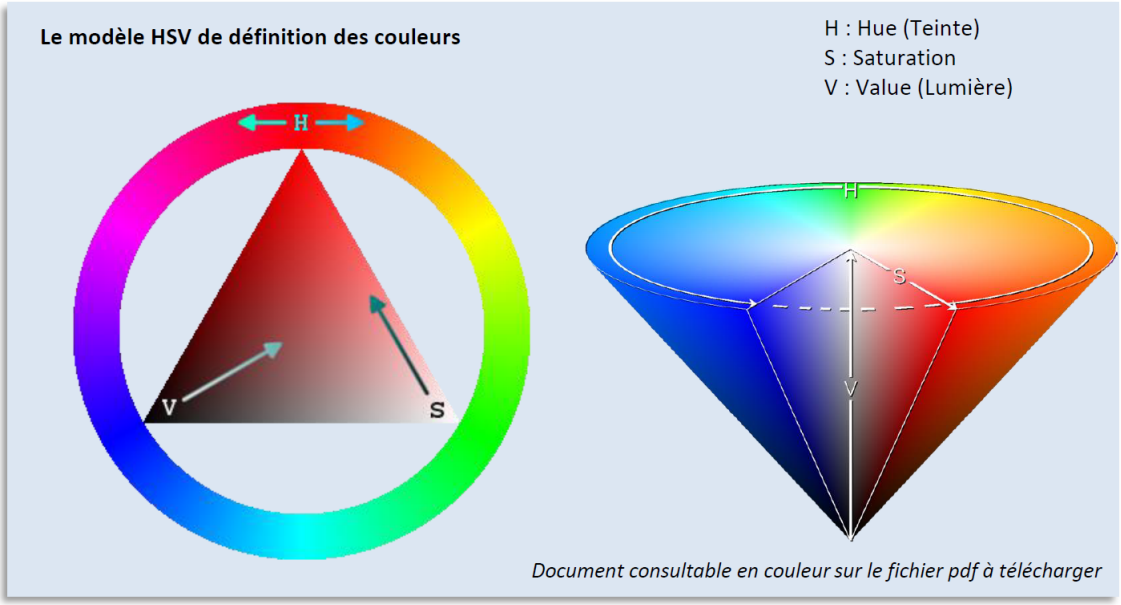
5 Seuillage à partir d'une image au format HSV

5.a Format HSV

Le modèle HSV (Hue, Saturation, Value) ou THV (Teinte, Saturation, Lumière) est un modèle plus proche de la perception humaine.
La couleur de chaque pixel toujours codée sous la forme d'une liste contenant 3 flottants compris entre 0 et 1 (ou 0 et 255 dans certains codages). Ces 3 flottants, notés H, S, V représentent :

- **H : Hue** (ou teinte) : elle caractérise la teinte (sur le cercle chromatique), comme illustré ci-dessous

  La fermeture du cercle chromatique a pour conséquence que les rouges se trouvent à la fois proche de zéro et de 1.
- **S : Saturation** : cette composante représente l'intensité de la coloration, en ce qu'elle se distingue du gris. Les gris ont une valeur de saturation proche de zéro.
- **V : Value** (lumière) : cette composante exprime l'impression de clarté, de brillance de la couleur.



	H	S	V	Couleur obtenue
[0,0,0]	0	0	0	Noir
[0,0,1]	0	0	1	Blanc
[0,1,1]	0	1	1	Rouge vif
[0,0,0.3]	0	0	0.3	Gris

5.b Conversion d'une image au format RGB en image au format HSV

Une méthode de conversion d'une image au format RGB en une image au format HSV est proposée ci-dessous :

On calcule les nombres Δ et C_{max} :

$$\Delta = \max(R,G,B) - \min(R,G,B)$$

$$C_{max} = \max(R,G,B)$$

Dans le cas où $\Delta = 0$, H est défini nul par convention.

Puis, on détermine les valeurs H,S,V, comme ci-dessous :

$$H = \begin{cases} \frac{1}{6}\left(\frac{G-B}{\Delta} modulo(6)\right), si\ R = C_{max} \\ \frac{1}{6}\left(2 + \frac{B-R}{\Delta}\right), si\ G = C_{max} \\ \frac{1}{6}\left(4 + \frac{R-G}{\Delta}\right), si\ B = C_{max} \end{cases}$$

$$S = \begin{cases} 0\ si\ C_{max} = 0 \\ \frac{\Delta}{C_{max}} si\ C_{max} \neq 0 \end{cases}$$

$$V = C_{max}$$

On rappelle que le modulo s'obtient en Python avec l'opérateur %.

Question 6. Écrire les fonctions `mini(L)` et `maxi(L)`, qui renvoient le minimum (respectivement le maximum) d'une liste L de flottants passée en argument. Tester vos fonctions avec une liste de votre choix, copier vos tests dans votre script.

Question 7. Écrire la fonction `RGB_to_HSV(pixelRGB:np.ndarray)` qui prend pour argument un tableau `pixelRGB` contenant, dans l'ordre, les 3 composantes [R,G,B] et renvoie le tableau des 3 composantes [H,S,V] correspondantes. Vous utiliserez les fonctions `mini(L)` et `maxi(L)`.

A RETENIR

Pour tester l'égalité de deux flottants, on procédera de la façon suivante :

$$X \text{ sera considéré comme étant égal à } Y, \text{ si : } Y - \epsilon < X < Y + \epsilon$$

ϵ étant un flottant suffisamment petit mais supérieur à la précision de calcul de l'ordinateur. On pourra prendre par exemple $\epsilon = 10^{-8}$.
C'est le même principe pour tester l'égalité d'un nombre flottant à zéro.

Question 8. Tester votre fonction `RGB_to_HSV(pixelRGB:np.ndarray)`.
Si `pixelRGB` est `[0.6,1,1]`, vous devez obtenir le tableau [H,S,V] : [0.5, 0.4, 1.] ;
Si `pixelRGB` est `[0.9,0.01,0.92]`, vous aurez [0.82967033, 0.98913043, 0.92].

Question 9. Écrire la fonction `conversion_HSV(imgRGB:np.ndarray)` qui convertit une image au format RGB en image au format HSV. **L'image brute ne sera pas modifiée**.

Pour ne conserver que les rouges tirant vers le bordeaux, on choisit de ne sélectionner que les valeurs de teinte (H) comprises entre 0.7 et 1. Pour éviter les gris, on ne garde que les couleurs à forte saturation (S) : saturations comprises entre 0.6 et 1.
La liste de seuils a le format suivant : seuilsHSV= `[Hmin, Hmax, Smin, Smax, Vmin, Vmax]`.

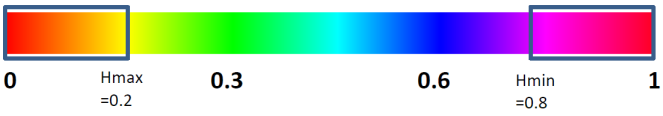
Question 10. Définir la liste de seuils `seuilsHSV` associée à notre problématique.

Question 11. En utilisant les fonctions `conversion_HSV` et `seuillage`, tester cette fonction sur l'image `img1`. Afficher l'image seuillée obtenue et analyser la qualité du seuillage. Sauvegarder l'image seuillée sous le nom de fichier `seuillage2.png`.

On peut tester aussi le processus de seuillage sur l'image `img2` et observer que cela est moins efficace car la nuance de rouge est un peu différente.

6 Amélioration de la fonction seuillage (partie facultative)

Pour une image au format HSV que l'on veut seuiller en ne conservant que les couleurs autour du rouge primaire, le seuillage peut conduire à ne conserver que les pixels de composante H dans l'intervalle `[0.8, 1]` ainsi que ceux de l'intervalle `[0, 0.2]`.



Pour gérer ce cas, on peut donner la possibilité de fournir une valeur de `Hmin` supérieure à celle de `Hmax`.

Question 12. Proposer fonction `seuillage2` version améliorée de votre fonction `seuillage` qui prenne en compte cette nouvelle contrainte lorsque l'utilisateur entre une valeur de `Hmin` supérieure à celle de `Hmax`. Tester cette nouvelle fonction sur l'image `im2.png`.

7 Détection du centre de la forme (partie facultative)

Question 13. Prendre connaissance de la fonction `cherche_centre` fournie dans le fichier `detection_centre.py`, et copier cette fonction dans votre script.

Question 14. Tester cette fonction en plaçant un rond vert au centre de la forme détectée et vérifier que cette technique permet bien de détecter le centre de l'objet.

Indication : La commande suivante permet de placer un rond vert au point de coordonnée x, y :
`plt.plot([x],[y],'go')`