SY32 – TD Vision 01 : Stéganographie couleur

Introduction : NumPy et matplotlib

Pour l'installation de `python` et des bibliothèques utilisées en TD, suivre ou s'inspirer des instructions données dans la page web de l'UV : `https://vision.uv.utc.fr/doku.php?id=setup-python`.

Pour bien démarrer, il est nécessaire de savoir manipuler un minimum les outils des libraires NumPy (`https://numpy.org/`) et matplotlib (`https://matplotlib.org/`).
Dans le cas contraire, commencer par étudier ces librairies d'études scientifiques à l'aide d'un cours tel que `https://nbhosting.inria.fr/builds/ue12-p22-python-numerique/handouts/latest/0-00-intro.html`.

Exercice 1 : Synthèse d'image en Python et NumPy

Synthétiser l'image de la Figure 1, représentant 3 disques rouge-vert-bleu et leurs intersections en synthèse additive des couleurs.

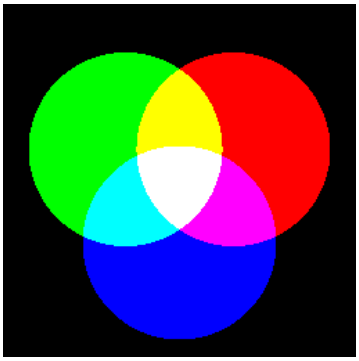


FIGURE 1 – Synthèse additive.

Caractéristiques demandées :

- Taille de l'image : 256×256 pixels
- Rayon des disques : $r = 70$ pixels
- Distance entre le centre des images et le centre des trois disques : $d = 45$ pixels
- En prenant le centre de l'image comme origine, l'angle entre la droite qui passe par le centre du disque rouge et le centre de l'image et l'axe horizontal vaut : $\pi/6$.
  Il en est de même pour le disque vert.

1. Importer la bilbiothèque `numpy` et le module `pyplot` de `matplotlib` :

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

   Définir les variables r et d.
   Les images sont définies avec l'origine en haut à gauche, x orienté vers la droite et y vers le bas.
   En prenant le centre de l'image $(x_0, y_0) = (128, 128)$ comme référence, calculer les coordonnées (x_c, y_c) des centres des 3 disques, où c peut être r pour le disque rouge, g pour le vert ou b pour le bleu.
   Appliquer une translation aux coordonnées ainsi calculées pour les situer dans le repère image.
2. Considérer l'un des disques de couleur. Utiliser l'équation d'un disque pour calculer le masque du disque : une image M à valeurs scalaires, de dimensions 256×256 et qui vaut la valeur 255 pour chaque pixel à l'intérieur du disque, et la valeur 0 ailleurs.
   Pour cela, on pourra définir une matrice D intermédiaire de taille 256×256 qui représente la distance au centre du disque.
   Utiliser une boucle explicite (exemple donné pour le disque rouge, centré en (x_r, y_r)) :

```
D = np.zeros((256,256),dtype='float32')
M = np.zeros((256,256),dtype='uint8')

for i in range(256):
    for j in range(256):
        D[i,j] = np.sqrt( (j-xr)**2 + (i-yr)**2 )
```

Noter l'opération $a**b$ pour lever les valeurs de a à la puissance b.
Calculer le masque M à partir de D et valider le résultat visuellement.
Pour afficher les distances et le masque (attention aux types de données uint8/double ou float) :

```
fig, ax = plt.subplots(1,2)
ax[0].imshow(D)
ax[0].set_title('D')
ax[1].imshow(M, cmap='gray')
ax[1].set_title('M')
plt.show()
```

3. Effectuer le même calcul en utilisant les boucles implicites (toujours l'exemple du disque rouge) :

```
D = np.zeros((256,256),dtype='float32')
M = np.zeros((256,256),dtype='uint8')
x = np.arange(256)
y = np.arange(256)
xx, yy = np.meshgrid(x, y)

D = np.sqrt( (xx-xr)**2 + (yy-yr)**2 )
```

Noter l'utilisation de `meshgrid` pour fabriquer une matrice xx contenant les valeurs de i et une matrice yy contenant les valeurs de j.

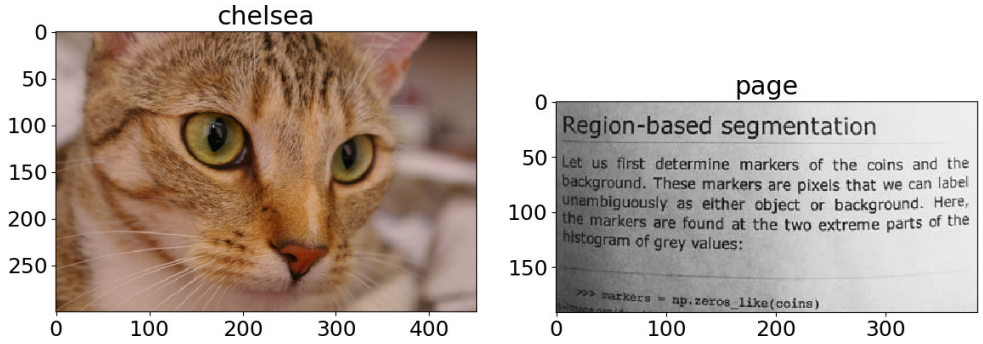
4. Générer un tel masque pour chaque disque, et en déduire les matrices R, G et B prenant des valeurs entre 0 et 255 correspondant à chaque canal de couleur.
5. Combiner les 3 canaux R, G et B en une matrice $256 \times 256 \times 3$ en les concaténant selon la 3^{e} dimension et afficher l'image couleur qui en résulte.

```
I = np.dstack((R,G,B))
```

Exercice 2 : Essais en stéganographie

Définition Larousse :
stéganographie
(du grec *steganos*, caché, et *graphein*, écrire)
Ensemble de techniques permettant de transmettre une information en la dissimulant au sein d'une autre information (photo, vidéo, texte, etc.) sans rapport avec la première et le plus souvent anodine.

Par technique de stéganographie, nous allons cacher l'image « page », contenant du texte, dans l'image « chelsea », le portrait d'un chat. Ces deux images sont affichées en Figure 2.



(a) Chelsea. (b) Page.

FIGURE 2 – Images à utiliser pour pratiquer la stéganographie de sorte à cacher (b) dans (a).

1. Charger les deux images et afficher :
   - Leurs dimensions
   - Le nombre de canaux
   - Le type de données utilisé pour coder les intensités
   - Leurs valeurs mini et maxi
   - Afficher les images

```
import numpy as np
import matplotlib.pyplot as plt
from skimage import data

%matplotlib auto

Ivis = data.chelsea()
Isecret = data.page()

print(Ivis.shape)
print(Ivis.dtype)
print(Ivis.min(), Ivis.max())

print(Isecret.shape)
print(Isecret.dtype)
print(Isecret.min(), Isecret.max())

fig, axes = plt.subplots(1, 2)
axes[0].imshow(Ivis)
axes[0].set_title("Chelsea")
axes[1].imshow(Isecret, cmap='gray')
axes[1].set_title("Page")
plt.show()
```

2. Visualiser indépendamment les trois composantes Rouge, Vert, Bleu de l'image « chelsea », comme des images en niveaux de gris (1 seul canal).
3. Sur quelle composante les intensités sont-elles les plus élevées ? C'est à priori la composante où l'on pourrait cacher les données de l'image « page » avec l'impact le moins visible.
4. Ajouter une bordure blanche (=255) dans l'image « page » de telle sorte à ce qu'elle soit aux mêmes dimensions que « chelsea » (agrandir le canevas), comme dans la Figure 3.
5. Inverser les intensités des pixels de l'image « page » ainsi étendue, comme dans la Figure 4.

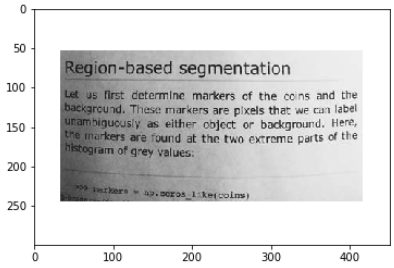


FIGURE 3 – Image Page avec canevas étendu.

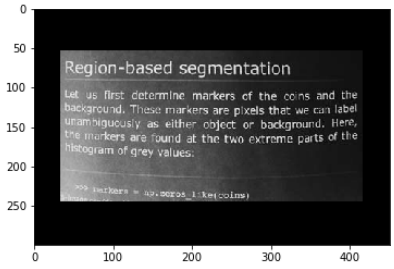


FIGURE 4 – Image Page avec canevas étendu, inversée.

6. Mélanger le canal rouge de « chelsea » avec l'image « page » aux intensités inversées, de telle sorte à garder la valeur maximale entre les deux.
   Afficher le canal résultat.
7. Recomposer l'image du chat en remplaçant le canal rouge par le canal modifié à la question précédente et afficher côte à côte l'image « chelsea » d'origine et la nouvelle image couleur.
   Commenter. Comment améliorer la procédure ?
   Expérimenter certaines propositions.

8. **Technique LSB** (*Least Significant Bit*) : les couleurs sont codées sur 8 bits tel que décrit dans le Tableau 1.

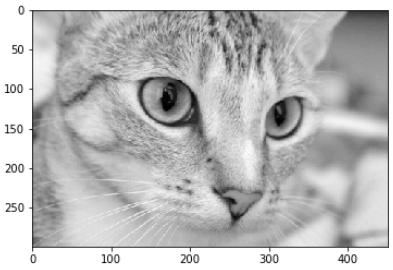
Poids fort	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Poids faible
	128	64	32	16	8	4	2	1	

TABLE 1 – Codage des intensités d'une image sur un octet. Chaque bit (ligne du haut) code dans cet ordre la valeur base 10 correspondante lorsqu'il vaut 1 (ligne du bas).

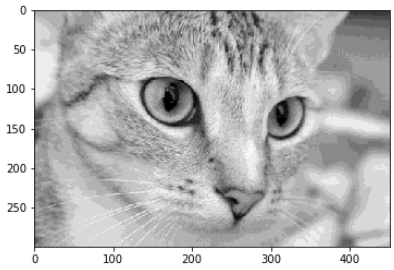
Changer de valeur un bit de poids faible est peu visible (voire imperceptible), tandis que modifier un bit de poids fort change de manière importante l'intensité du pixel.
Le codage LSB consiste à utiliser la moitié des bits, ceux ayant les poids les plus faibles pour coder l'information de l'image à cacher. Cela implique de coder l'information des canaux à fusionner sur 4 bits au lieu de 8.

Créer une version tronquée du canal rouge de « chelsea » dont les 4 bits de poids faible sont mis à zéro. Faire de même pour l'image « page » (comportant les bordures pour étendre le canevas). Les résultats sont illustrés en Figure 5
Afficher leurs valeurs minimales et maximales. Afficher les anciens et nouveaux canaux.

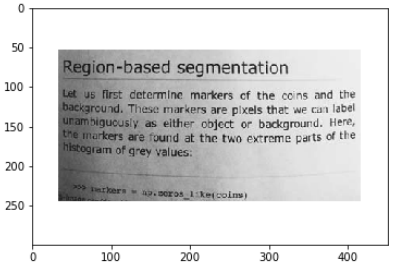
```
truncR = IvisR & int('11110000',2)
print(truncR.min(), truncR.max())
truncSecret = Isecret2 & int('11110000',2)
print(truncSecret.min(), truncSecret.max())
```



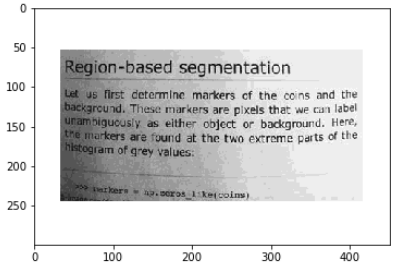
(a) Canal rouge de Chelsea.



(b) Canal rouge de Chelsea, tronqué.



(c) Page.



(d) Page, tronqué.

FIGURE 5 – Comparaison des images (a) et (c) avec leurs versions dont les bits de poids faible ont été tronqués, respectivement (b) et (d).

9. À présent, décaler les bits de l'image « page » tronquée de ses 4 bits de poids faible vers la droite, de telle sorte à ce que les informations soient codées sur les 4 bits de poids faible au lieu des bits de poids fort. L'opérateur `>>` permet de décaler les bits vers la droite.
Afficher les valeurs mini et maxi en résultant.
Ensuite, assembler les bits de poids forts de « chelsea » avec ces bits « de poids faible » (peut se faire via une opération binaire OU).
Afficher les valeurs mini et maxi du canal issu de cet assemblage. Afficher le canal résultant de cette opération. Le résultat doit être tel que la Figure 6.

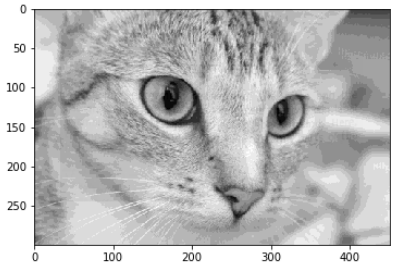


FIGURE 6 – Image Page cachée dans le canal rouge de Chelsea par technique LSB.

10. Recomposer l'image du chat en couleurs en remplaçant le canal rouge par le canal modifié à la question précédente et afficher côte à côte l'image « chelsea » d'origine et la nouvelle image couleur.
    Commenter.
11. Pour constater que l'image modifiée est bien différente de l'originale, afficher l'image des différences entre les intensités du canal rouge du chat d'origine avec le canal rouge du chat contenant les données cachées.
12. Une astuce permettant de voir le type de manipulation effectuée consiste à visualiser l'histogramme des canaux de l'image « chelsea » originale et de l'image fusionnée.
13. Proposer le code permettant de récupérer l'image cachée. Afficher l'image récupérée à côté de l'image « page » originale.
14. Afficher côte-à-côte l'image « chelsea » originale et le résultat de la fusion si on utilise alternativement chaque canal de couleur R, V et B.
    Commenter.
15. Cacher une image secrète en couleur. Cacher l'image « astronaut », visible dans la Figure 7, dans l'image « chelsea ». L'image « astronaut » étant plus grande, il ne faut en prendre qu'une portion de dimensions égales à « chelsea ».

```
Isecret = Isecret[:Ivis.shape[0],:Ivis.shape[1],:]
```

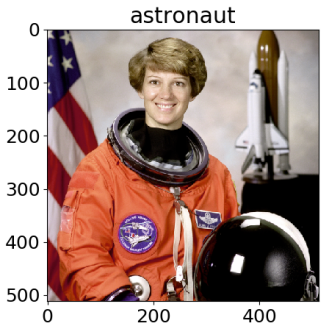


FIGURE 7 – Image couleur Astronaut à cacher.

Liste de fonctions utiles :

- `numpy.ones` et `numpy.ones_like`
- `numpy.zeros` et `numpy.zeros_like`
- `numpy.full` et `numpy.full_like`
- `numpy.stack`
- `numpy.vstack`
- `numpy.hstack`
- `numpy.dstack`
- `numpy.maximum`
- `numpy.copy`
- `numpy.exposure.histogram`