

Tutoriel OpenCV Python - Traitement d'images - Vision par ordinateur

Deviens un monstre de la vision par ordinateur et du machine learning en 5 à 10 minutes par jour

🕒 22/07/2019 💬 0 Commentaire 📁 Traitement d'images

👤 Gundamotoko

OpenCV est actuellement la référence de la vision par Ordinateur, peu importe dans quel laboratoire, entreprise, université ou vous irez pour faire du traitement et de l'analyse d'image, il est impossible que les gens qui y soit n'aient pas connaissance de l'existence d'[OpenCV](#).

Cela dit, même si elles sont toutes au courant de son existence, elles ne l'utilisent pas forcément, car elles préfèrent parfois des solutions alternatives telles que [MATLAB](#), [Pandore](#) et un peu plus rarement leurs propres solutions propriétaires.

Voici une liste non exhaustive des différentes opérations d'images que permet de faire [OpenCV](#) :


Ouverture, enregistrement et affichage d'une image ou d'une vidéo.



Construction et affichage de l'histogramme d'une image

Localisation et extraction d'objets via des fonctions de segmentation tels que le seuillage d'Otsu.

Opérations d'image usuelles (redimensionnement, rotation, opérateurs morphologiques, filtrage, et bien d'autres encore).

Détection et tracking d'objets

Deviens un  monstre de la vision par ordinateur et du machine learning en 5 à 10 minutes

 En continuant votre navigation sur ce site, vous acceptez l'utilisation des cookies  afin d'assurer le bon déroulement de votre visite et de réaliser des statistiques d'audience.

Dans cet article, nous verrons ensemble :

Pourquoi utiliser **OpenCV**

Comment installer OpenCV avec python

Les fonctions usuelles d'OpenCV (lire, enregistrer, afficher une image)

Les différents espaces de couleurs (Comment convertir une image en niveau de gris, ou encore comment passer un image dans l'espace HSV)

Une méthodes d'extraction d'objets présents dans une image (histoire de faire un truc un peu costaud quand même dans ce tuto ;-D)

Enfin pour les plus courageux, et les plus passionnés, une petite introduction à l'extraction de caractéristiques avec OpenCV.



Logo d'OpenCV

Pourquoi utiliser OpenCV ?

⚠ En continuant votre navigation sur ce site, vous acceptez l'utilisation des cookies **×** afin d'assurer le bon déroulement de votre visite et de réaliser des statistiques d'audience.

Deviens un **×**
monstre de la
vision par
ordinateur et du
machine learning
en 5 à 10 minutes

-OpenCV est simple d'utilisation et puissante

En effet, le nom des fonctions est simple et relate effectivement leur rôle (ex : imread pour lire une image, imwrite pour enregistrer une image. Simple ! Non ? ;-))

-OpenCV est gratuite

ce qui probablement doit plaire à beaucoup de sa communauté

-Le fait d'être gratuite ne l'empêche aucunement d'être puissante, quoi de plus normal pour une bibliothèque énormément utilisée par de gros industriels (Intel, Willow Garage, probablement Google) et de nombreuses universités telles que l'université des Antilles d'où je viens, mais aussi L'UPMC à Paris.


-Ses sources sont modifiables et accessibles.



Ce dernier point est un argument de persuasion majeur pour les maniaques du contrôle qui sont persuadés qu'ils vont modifier la librairie et apporter une modif révolutionnaire, mais qui en vérité ne le feront jamais ! En outre cela permet aux universités et industriels d'adapter la bibliothèque à leurs besoins, mais aussi de proposer des modifications via la célèbre plateforme Github ou l'on peut trouver les sources d'OpenCV : [lien](#).

-Enfin pour finir, cette librairie est disponible dans de nombreux langages,

Python, Java, Matlab, C#, C++, JavaScript, etc.... Attention cependant, cela ne veut pas dire que les binding d'OpenCV que vous trouvez dans ces langages sont officiels. Pour preuve, le binding python, utilisé par toute la communauté, n'est pas développé par [OpenCV.org](#).

J'espère que tout cela vous aura convaincu de la puissance, de la liberté des actions qu'accorde OpenCV, mais également de la portabilité de cette dernière. Je rajouterai à cela, que tous les

Deviens un  monstre de la vision par ordinateur et du machine learning en 5 à 10 minutes

 En continuant votre navigation sur ce site, vous acceptez l'utilisation des cookies  afin d'assurer le bon déroulement de votre visite et de réaliser des statistiques d'audience.

Actuellement le langage python est actuellement le plus simple pour commencer avec OpenCV.

Ceci dit, c'est loin d'être le plus économe, j'avais lu un jour sur [un article de developpez.com](#) que celui-ci était 76 fois si je me trompe pas plus énergivore que le langage C.

Je ne vous cache pas les émissions de CO2 que ça doit faire !

Python le langage le plus polluant de la terre XD.

Je délire un peu :) ! Ceci dit l'info est vraie.

Retournons aux choses sérieuses. Il existe différentes manières d'installer OpenCV, j'ai d'ailleurs commencé un tuto à ce sujet que vous pouvez trouver via ce lien : [Installer OpenCV avec Python 3 - Les différentes méthodes](#). Toutefois, je vais quand même brièvement introduire la question ici avec la méthode la plus simple et directe.

Si vous n'avez pas python, vous vous en doutez il va falloir l'installer :


[Téléchargement python.](#)



Une fois installé, le gestionnaire de paquet de ce dernier "pip" est directement installé avec.

Eh oui, ça se lit "pip", du coup on va pip installer OpenCV. Je ri un peu, pas trop tout de fois.

La commande qui permet d'installer OpenCV est la suivante :

```
pip install opencv
```

Deviens un  monstre de la vision par ordinateur et du machine learning en 5 à 10 minutes

 En continuant votre navigation sur ce site, vous acceptez l'utilisation des cookies  afin d'assurer le bon déroulement de votre visite et de réaliser des statistiques d'audience.

Du coup, à chaque fois que je veux pip installer une bibliothèque je suis obligé de taper ça :

```
python -m pip install opencv
```

Je dis ça pour les gens qui sont sur Windows et qui auraient le même souci que moi.

Comment charger et enregistrer une image avec OpenCV?

Évidemment si tu connais déjà le langage python, ce qui va suivre sera loin de te gêner sinon ben je vais tenter de t'expliquer au mieux.

Pour utiliser OpenCV, il est nécessaire de la charger en mémoire. Cela se fait en tapant la ligne :


```
import cv2
```

Pour charger une image, tu peux utiliser la fonction `cv2.imread(nom_image)` comme cela :



```
image =cv2.imread('fleur.png')
```

Pour sauvegarder une image, c'est la fonction `imwrite(nom_image, image)` qu'il faut utiliser.

```
cv2.imwrite('fleur.png',gray)
```

Deviens un  monstre de la vision par ordinateur et du machine learning en 5 à 10 minutes

Comment afficher une image

 En continuant votre navigation sur ce site, vous acceptez l'utilisation des cookies  afin d'assurer le bon déroulement de votre visite et de réaliser des statistiques d'audience.

graphique.

Puis, il faut utiliser la fonction `imshow` qui prend pour paramètres le nom de la fenêtre et l'image à afficher.

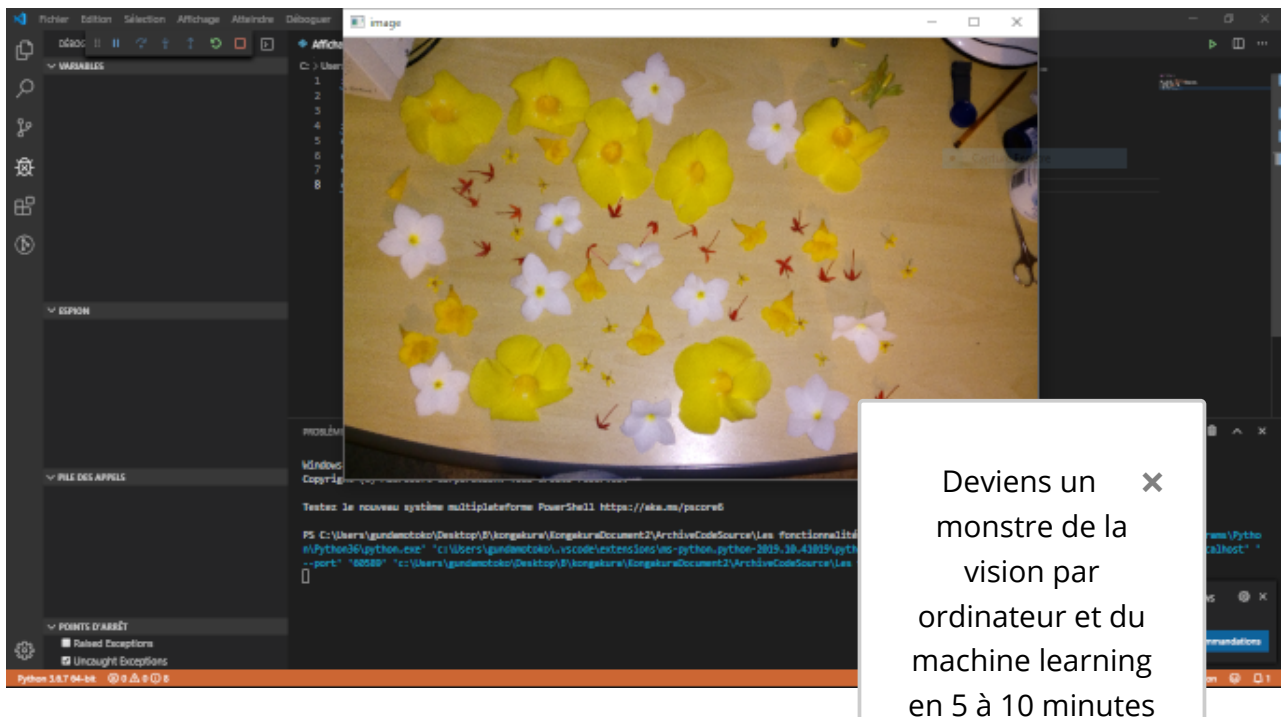
La fonction `waitKey` permettra ensuite de mettre la fenêtre en attente d'un événement.

Pour finir, la fonction `destroyAllWindows` libérera la mémoire occupée par la fenêtre après que celle-ci ait été fermée.

Code complet pour afficher une image :

```
import cv2
img=cv2.imread ("fleur.png")
cv2.namedWindow('image', cv2.WINDOW_NORMAL)
cv2.imshow('image',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Résultat :



⚠ En continuant votre navigation sur ce site, vous acceptez l'utilisation des cookies ✕ afin d'assurer le bon déroulement de votre visite et de réaliser des statistiques d'audience.

```
cv2.namedWindow('image', cv2.WINDOW_NORMAL)
```

Namewindow prend en argument le nom de la fenêtre et un flag qui indique si vous souhaitez que la fenêtre soit redimensionnable ou non. Par défauts, ce second paramètre vaut cv.WINDOW_AUTOSIZE. Cela signifie que la taille de la fenêtre s'adaptera à la taille de l'image et ne sera pas modifiable. Personnellement, je préfère mettre cv2.WINDOW_NORMAL, car si l'image est trop grande ou trop petite vous pourrez modifier manuellement la taille de la fenêtre.

```
cv2.imshow('image', img)
```

Imshow comme je l'ai dit précédemment prend en premier paramètre le nom de la fenêtre et en second paramètre l'image.

```
cv2.waitKey(0)
```


Mets la fenêtre en attente pendant une certaine durée indiquée en milliseconde en paramètre de la fonction. Sinon, jusqu'à ce qu'il y ait un évènement généré par l'utilisateur.



```
cv2.destroyAllWindows()
```

Détruis toutes les fenêtres et libère la mémoire associée à celles-ci. La fonction n'a besoin d'aucun argument. Il est possible de détruire une fenêtre spécifique en passant son nom en argument de la fonction.

Les espaces de couleurs

De manière générale, quand tu vas travailler sur une image, souvent, celle-ci sera au format RGB. C'est le pire format qu'il puisse être pour travailler sur une image, car celui-ci est adapté pour les ordinateurs et non pour les humains.

Deviens un  monstre de la vision par ordinateur et du machine learning en 5 à 10 minutes

 En continuant votre navigation sur ce site, vous acceptez l'utilisation des cookies  afin d'assurer le bon déroulement de votre visite et de réaliser des statistiques d'audience.

Mais pour cela, je vais d'abord t'expliquer le fonctionnement de la rétine humaine, cette petite partie de l'oeil humain qui te permet de voir.

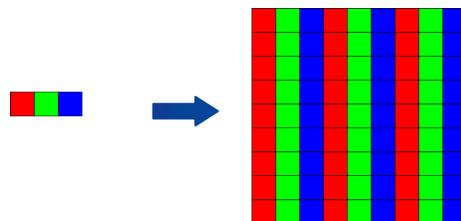
La rétine humaine, si tu l'as un peu étudié sinon je t'invite à lire ce [tuto](#) que j'ai écrit, est constitué de 150 millions de capteurs de lumière qui sont de deux types les « [bâtonnets](#) » et les « [cônes](#) ».


Les [bâtonnets](#) sont chargés de la vision scotopique (en noir et blanc) tandis que les [cônes](#) sont chargés de la vision photopique (en couleurs). les [cônes](#) sont au nombre de 7 millions sur notre rétine et sont divisibles en trois catégories : les [cônes](#) de faible, de moyenne et de forte longueur d'onde. Chacune des catégories étant de longueur d'ondes différentes permet de capter une lumière différente c'est pourquoi les [cônes](#) de faible, de moyenne et de forte longueur d'onde permettront de capter respectivement la lumière bleue, la verte et le rouge.



Les autres couleurs que l'on peut voir telles que le jaune, le magenta ou encore la couleur du bois sont obtenus par combinaison de ses trois couleurs élémentaires.

Pour le capteur des appareils photo, c'est pareil que pour l'oeil humain il s'agit de milliers de [photo-capteurs](#) assemblés ensemble en forme de grille.

Ces photo-capteurs par des processus physiques que je ne peux expliquer puisque ce n'est pas mon domaine d'expertise transforment la lumière en signal électrique. Ce signal continu est par la suite converti en signal numérique qui sera traité par un ordinateur.



Deviens un  monstre de la vision par ordinateur et du machine learning en 5 à 10 minutes

 En continuant votre navigation sur ce site, vous acceptez l'utilisation des cookies  afin d'assurer le bon déroulement de votre visite et de réaliser des statistiques d'audience.

Le système **RGB** est celui utilisé par défaut dans les appareil photo car leur architecture matérielle nous y contraint.

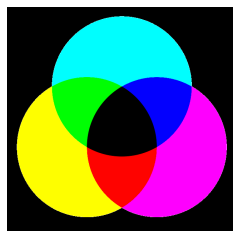
Basiquement, un appareil photo consiste en un assemblage de nano photo-capteur en grille. Chaque capteur n'est capable de capter qu'une seule longueur d'onde parmi le rouge le vert et le bleu, d'où la représentation **RGB**.


Pour être manipulée informatiquement, il est nécessaire que la couleur soit convertie en valeur numérique. C'est pour cette raison que la notion d'espaces de couleurs a été inventée. Il existe énormément de manières de représenter une couleur.



Je ne vais aborder avec toi que les quatre des représentation représentations les plus populaires le RGB, le niveau de gris, le CMYB et le HSV. La CIE Lab est également populaire mais je ne suis pas habitué à l'utiliser.

L'espace CMYB (Cyan Magenta, Yellows, Black)

L'espace de couleurs CMYB (Cyan Magenta, Yellows, Black) est particulièrement utilisé en imprimerie, car l'encre des imprimantes est de couleur cyan, magenta et jaune, le noir parfait en mélangeant ces trois couleurs étant impossible à obtenir une cartouche d'encre noire est rajoutée d'où la composante B.



Deviens un  monstre de la vision par ordinateur et du machine learning en 5 à 10 minutes

 En continuant votre navigation sur ce site, vous acceptez l'utilisation des cookies  afin d'assurer le bon déroulement de votre visite et de réaliser des statistiques d'audience.

teinte de l'objet (le H) quand par exemple nous disons qu'il est rouge ou qu'il est vert.

La saturation (Le S) représente la pureté de la couleur de l'objet, ainsi un objet rouge ayant une forte pureté un nous paraîtra éclatant, tandis qu'avec une faible pureté il aura plutôt un aspect délavé.

Pour finir, l'intensité (Le V) représente la quantité de lumière émise sur un objet, plus elle est forte et plus la couleur est claire, tandis que plus celle-ci est faible, plus la couleur de l'objet est sombre.

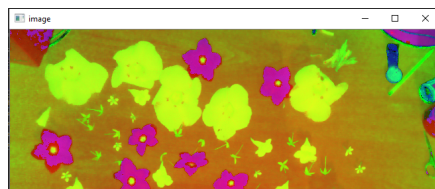
La ligne de code suivante permet de passer de l'espace **RGB** à l'espace **HSV** :


Code :



```
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
```

Exemple :

```
import cv2
img=cv2.imread ("fleur.png")
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
cv2.namedWindow('hsv', cv2.WINDOW_NORMAL)
cv2.imshow('hsv',hsv)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Résultat :

Deviens un  monstre de la vision par ordinateur et du machine learning en 5 à 10 minutes

 En continuant votre navigation sur ce site, vous acceptez l'utilisation des cookies  afin d'assurer le bon déroulement de votre visite et de réaliser des statistiques d'audience.

Transformer une image en niveau de gris

Il est possible de transformer une image couleur en niveau de gris avec la ligne de code suivant :

Code

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

Résultat :



Qu'est ce que c'est la segmentation d'images?


La segmentation permet d'isoler les différents objets présents dans une image.



Il existe différentes méthodes de segmentation : la **classification**, le **clustering**, les **level-set**, **graph-cut**, etc

Mais la plus simple est le seuillage, c'est pourquoi je vais te parler uniquement de celle-ci dans cette seconde partie.

Le seuillage est une opération qui permet de transformer une image en niveau de gris en image binaire (noir et blanc),

l'image obtenue est appelée masque binaire.

Deviens un  monstre de la vision par ordinateur et du machine learning en 5 à 10 minutes

 En continuant votre navigation sur ce site, vous acceptez l'utilisation des cookies  afin d'assurer le bon déroulement de votre visite et de réaliser des statistiques d'audience.

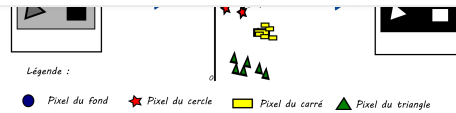


Illustration du seuillage optimal

Il existe 2 méthodes pour seuiller une image, le seuillage manuel et le seuillage automatique.

Seuillage manuel

Sur la l'illustration ci-dessus, trois formes sont présentes dans l'image originale.

Les pixels du rond sont représentés par des étoiles, ceux du carré par des rectangles jaunes, ceux du triangle par des triangles verts et les cercles bleus correspondent au fond.

Dans la figure, nous pouvons remarquer que tous les pixels correspondants du fond (rond bleu) ont une valeur supérieure à 175. Nous en déduisons que le seuil optimal est 175. En faisant cela, nous avons déterminé le seuil optimal manuellement.


Pour utiliser le seuil manuel avec OpenCV, il suffit d'appeler la fonction `threshold`



comme suit :

Code :

```
ret,th=cv2.threshold(img, seuil,couleur, option)
```

Elle prend en paramètre, `img` : l'image à traiter , `seuil` : la valeur du seuil , `couleur` : la couleur que l'on souhaite attribuer à la zone objet et elle retourne `ret` : la valeur du seuil, et `th` : l'image binaire résultat du seuillage. Afin d'illustrer cela, dans le code suivant, j'ai seuillé l'image `fleur.png` préalablement passée en niveau de gris.

Deviens un  monstre de la vision par ordinateur et du machine learning en 5 à 10 minutes

 En continuant votre navigation sur ce site, vous acceptez l'utilisation des cookies  afin d'assurer le bon déroulement de votre visite et de réaliser des statistiques d'audience.

```
gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
ret,th=cv2.threshold(gray,150,255,cv2.THRESH_BINARY)
cv2.namedWindow('image',cv2.WINDOW_NORMAL)
cv2.imshow('image',th)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Résultat:

Nous remarquons que le seuillage n'a pas fonctionné.

Cela est dû au fait que les pixels du fond ont une valeur trop proche de celle des fleurs.

C'est pourquoi afin que ces valeurs soient bien distinctes, il est nécessaire de changer d'espace de couleur.

Nous allons donc passer dans l'espace de couleur **HSV** et observer l'histogramme de l'image !


Construire l'histogramme d'une image



Pour calculer l'histogramme d'une image, il faut faire appel à la fonction `calchist` :

Code :

```
hist= cv2.calcHist([img],[i],[256],[0,256])
```

Elle prend en paramètre l'image, les indices des canaux que l'on souhaite extraire, la taille de l'histogramme, l'intervalle des valeurs de pixel à mesurer. Après avoir utilisé `calchist` pour

Deviens un  monstre de la vision par ordinateur et du machine learning en 5 à 10 minutes

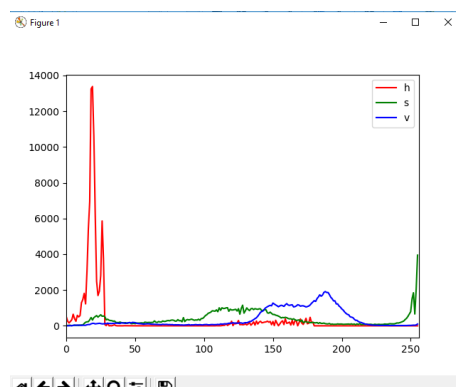
 En continuant votre navigation sur ce site, vous acceptez l'utilisation des cookies  afin d'assurer le bon déroulement de votre visite et de réaliser des statistiques d'audience.


Une fois installée, il faudra appeler les fonction plot, xlim et show pour afficher l'histogramme. Plot servira à afficher le graphique à l'écran, xlim délimitera l'axe des abscisses à 256 niveau, et show créera la fenêtre qui affichera les histogrammes.



```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
img=cv.imread ("fleur.png");
#RGB -> HSV.
hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)
#Déclaration des couleurs des courbes
color = ('r','g','b')
#Déclaration des noms des courbes.
labels = ('h','s','v')
#Pour col allant r à b et pour i allant de 0 au nombre d
e couleurs
for i,col in enumerate(color):
    #Hist prend la valeur de l'histogramme de hsv sur l
a canal i.
    hist = cv.calcHist([hsv],[i],None,[256],[0,256])
    # Plot de hist.
    plt.plot(hist,color = col,label=labels[i])
    plt.xlim([0,256])
#Affichage.
plt.show()
```

Dans le code ci-dessus, **matplotlib** est importé (ligne 3), l'image est mise en mémoire, puis convertis en hsv grâce à **cvtColor** (ligne 6). Ligne 12, une boucle qui va itérer sur chaque canal de l'image est créée. L'histogramme du canal courant est obtenu à la ligne 14 en utilisant **calcHist** puis est affiché à la ligne 18.

Résultat :



Deviens un 
monstre de la
vision par
ordinateur et du
machine learning
en 5 à 10 minutes

 En continuant votre navigation sur ce site, vous acceptez l'utilisation des cookies  afin d'assurer le bon déroulement de votre visite et de réaliser des statistiques d'audience.

Un bon seuil donc sur le canal hue serait un peu après les pics, au alentour de 35-45.

La saturation et la luminosité ne nous apprennent pas grand-chose sur l'image comme nous pouvons le constater. Afin de confirmer cela, nous pouvons seuiller pour voir.

Mais je préfère vous présenter dès à présent la méthode d'Otsu.

Seuillage automatique : Méthode d'Otsu

Il consiste à seuiller l'image courante pour tous les seuils possibles (de 0 à 255 pour une image en niveau de gris).

Un masque binaire pour chacun des seuils est alors créé.

Otsu va ensuite appliquer successivement chaque masque à l'image traitée.

Pour chaque masque, 2 images seront obtenues. Une ne contenant que le fond, et une ne contenant que les objets.


La variance interclasse, entre les pixels du fond et ceux des objets, est calculée et le seuil du masque pour lequel l'on obtient la plus grande variance est retenu comme seuil optimal.



La fonction python permettant d'utiliser cet algorithme est la fonction `threshold`.

Je ne vais pas la présenter, car je l'ai présenté précédemment seuls le second (il passe à 0) et le dernier paramètre changent. `cv.THRESH_OTSU` a été rajouté).

Code :

```
ret2, th2 = cv.threshold(img, 0, 255, cv.THRESH_BINARY+cv.THRESH_OTSU)
```

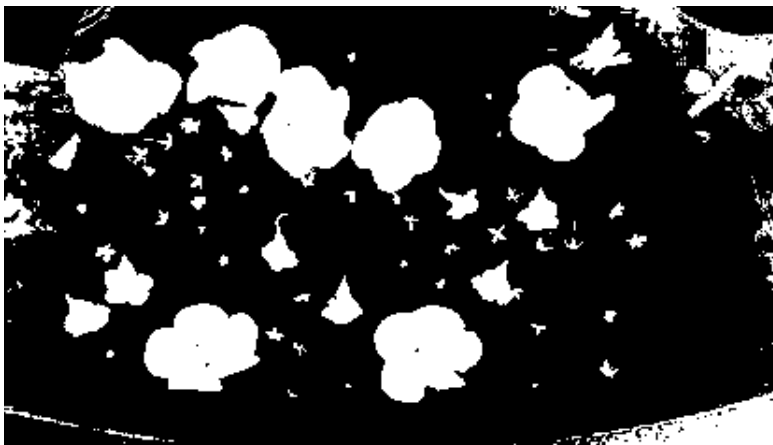
Deviens un  monstre de la vision par ordinateur et du machine learning en 5 à 10 minutes

 En continuant votre navigation sur ce site, vous acceptez l'utilisation des cookies  afin d'assurer le bon déroulement de votre visite et de réaliser des statistiques d'audience.


```
hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)
h,s,v= cv.split(hsv)
ret_h, th_h = cv.threshold(h,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
ret_s, th_s = cv.threshold(s,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
ret_v, th_v = cv.threshold(s,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
cv.imwrite("th_h.png", th_h)
cv.imwrite("th_s.png", th_s)
cv.imwrite("th_v.png", th_v)
```



Résultat :

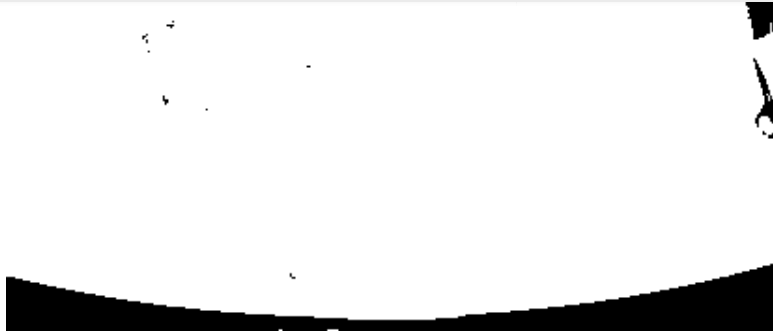
Seuillage d'Otsu sur le canal hue



Seuillage d'Otsu sur le canal Saturation

Deviens un 
monstre de la
vision par
ordinateur et du
machine learning
en 5 à 10 minutes

 En continuant votre navigation sur ce site, vous acceptez l'utilisation des cookies 
afin d'assurer le bon déroulement de votre visite et de réaliser des statistiques
d'audience.



Seuillage d'Otsu sur le canal Luminosité

Les seuils optimaux trouvés par Otsu pour les canaux teinte, saturation et luminosité sont respectivement 75, 168 et 114.

Remarquons que, la segmentation obtenue sur le canal luminosité n'a rien donné d'intéressant. Celle du canal hue a permis d'extraire les fleurs blanches. Enfin celle de la saturation a extrait toutes les fleurs sauf les blanches.

Afin d'obtenir une segmentation parfaite qui sépare toutes les fleurs du fond, il nous est nécessaire de fusionner le résultat obtenu en hue et en saturation.

Cette opération se fera par l'intermédiaire d'opérateurs binaires d'image.


Les Opérateurs binaires d'image :



Les opérateurs binaires d'image permettent d'appliquer aux images des opérations binaire logique

telles que le NON, le OU, le ET, mais aussi, le OU exclusif.

L'opérateur NO

L'opérateur NO passe en noir les pixels qui sont blancs et en blanc les pixels qui sont noirs. Sa fonction OpenCV est

Deviens un  monstre de la vision par ordinateur et du machine learning en 5 à 10 minutes

 En continuant votre navigation sur ce site, vous acceptez l'utilisation des cookies  afin d'assurer le bon déroulement de votre visite et de réaliser des statistiques d'audience.

endroits ou les pixels de l'image 1 et de l'image 2 était simultanément blancs. Sa fonction OpenCV est `bitwise_and` qui prend le paramètre deux images binaires.

```
cv2.bitwise_and(img1, img2)
```

L'opérateur OR

Le "ou" passe les pixels en blanc si les pixels de l'image 1 ou de l'image 2 sont blancs. Sa fonction OpenCV est `bitwise_or`. Elle prend en paramètre deux images binaires.

```
cv2.bitwise_or(img1, img2)
```

L'opérateur XOR


Le "ou exclusif" passe les pixels en blanc si les pixels de l'image 1 ou de l'image 2 sont blancs, mais pas si les deux sont blanc ensemble. Sa fonction OpenCV est `bitwise_xor`. Elle prend en paramètre deux images binaires.



```
cv2.bitwise_xor(img1, img2)
```

Dans notre cas, nous appliquerons le "ou", car nous voulons une image dans laquelle les pixels des fleurs blanches et des autres fleurs sont allumés.

Autrement dit, nous souhaitons les fleurs présentes sur l'image binaire hue ou sur l'image binaire saturation. Nous utiliserons ensuite la fonction `cv2.bitwise_and`

pour appliquer le masque binaire obtenu sur l'image fleur.png. Nous obtiendrons ainsi l'image fleur.png privée de son fond. Le code suivant permet de réaliser ces opérations.

Deviens un  monstre de la vision par ordinateur et du machine learning en 5 à 10 minutes

 En continuant votre navigation sur ce site, vous acceptez l'utilisation des cookies  afin d'assurer le bon déroulement de votre visite et de réaliser des statistiques d'audience.

```
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
h,s,v= cv2.split(hsv)
ret_h, th_h = cv2.threshold(h,0,255,cv2.THRESH_BINARY+cv2.THRESH_O
TSU)
ret_s, th_s = cv2.threshold(s,0,255,cv2.THRESH_BINARY+cv2.THRESH_O
TSU)
#Fusion th_h et th_s
th=cv2.bitwise_or(th_h,th_s)
cv2.imwrite("th.png",th)
```


Aux lignes 6 et 7 les canaux h et s sont seuillés. Les images binaires résultantes sont th_h et th_s, à la ligne 9 l'image binaire finale th est obtenue en fusionnant et th_h et th_s.



Résultat :



Remplissage des régions

Nous pouvons améliorer ce résultat en retirant les trous à l'intérieur des fleurs blanches. Pour ce faire, nous allons remplir les trous présents dans le masque de l'image.

Deviens un  monstre de la vision par ordinateur et du machine learning en 5 à 10 minutes

 En continuant votre navigation sur ce site, vous acceptez l'utilisation des cookies  afin d'assurer le bon déroulement de votre visite et de réaliser des statistiques d'audience.



Cela se fera notamment grâce à la fonction `floodfill` qui permet de remplir des régions.


```
cv2.floodfill(im_floodfill,mask,seedPoint,newVal)
```



Elle prend en paramètre une image source de 1 canal ou 3 canaux, un masque, cet argument indique par des pixels non nuls les zones que la fonction ne doit pas traiter il doit être deux pixels plus larges et deux pixels plus longs que l'image source. Le `seedPoint` qui est le pixel de départ de la fonction. `NewVal` est la valeur que l'on souhaite donner aux régions à remplir.

```
cv2.copyMakeBorder(th, top,bottom,left,right,borderType, value)
```

Nous nous aiderons également de la fonction `copyMakeBorder` qui nous permettra d'ajouter temporairement des bords vides à l'image afin que la fonction `flood fill` ne remplisse pas une mauvaise zone. Elle prend en paramètre une image source, la taille en nombre de pixels de chacun des bords haut, bas, gauche, et droite. Un flag qui indique le type de bord que l'on souhaite. Dans notre cas, nous utiliserons `BORDER_Constant` signifie que nous voulons des bords d'une seule couleur. Et enfin la couleur des bords qui sera dans notre cas du noir.

Code :

Deviens un  monstre de la vision par ordinateur et du machine learning en 5 à 10 minutes

 En continuant votre navigation sur ce site, vous acceptez l'utilisation des cookies  afin d'assurer le bon déroulement de votre visite et de réaliser des statistiques d'audience.

```


hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
h,s,v= cv2.split(hsv)
ret_h, th_h = cv2.threshold(h,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
ret_s, th_s = cv2.threshold(s,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
#Fusion th_h et th_s
th=cv2.bitwise_or(th_h,th_s)
#Ajouts de bord à l'image
bordersize=10
th=cv2.copyMakeBorder(th, top=bordersize, bottom=bordersize, left=bordersize, right=bordersize, borderType= cv2.BORDER_CONSTANT, value=[0,0,0] )
#Remplissage des contours
im_floodfill = th.copy()
h, w = th.shape[:2]
mask = np.zeros((h+2, w+2), np.uint8)
cv2.floodFill(im_floodfill, mask, (0,0), 255)
im_floodfill_inv = cv2.bitwise_not(im_floodfill)
th = th | im_floodfill_inv
#Enlèvement des bord de l'image
th=th[bordersize: len(th)-bordersize,bordersize: len(th)-bordersize]
resultat=cv2.bitwise_and(img,img,mask=th)
cv2.imwrite("im_floodfill.png",im_floodfill)
cv2.imwrite("th.png",th)
cv2.imwrite("resultat.png",resultat)



```

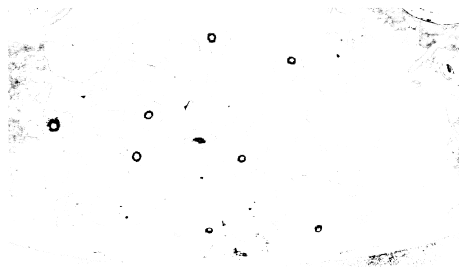
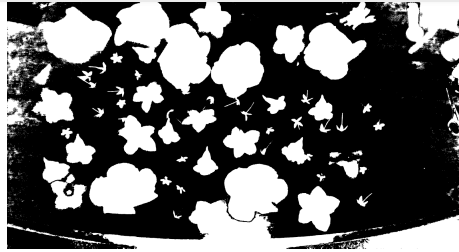
La première partie du code est la même que précédemment. Nous seuillons sur les canaux teinte et saturation, que par la suite nous fusionnons afin d'obtenir le masque binaire de l'image.

Ensuite, nous commençons par ajouter des bords à l'image à la ligne 13, ligne 16, nous créons une image imfloodfill qui contiendra les régions à remplir, le masque est créé à la ligne 19 avec en noir les trous à remplir et en blanc le reste. Nous l'inversons donc à la ligne 20 avant de la fusionner avec le masque de l'image. Nous enlevons les bords que nous avions précédemment mis avant d'appliquer le masque à l'image et d'enregistrer les résultats.

Nous obtenons alors ces trois images avec im floodfill que j'a


Deviens un  monstre de la vision par ordinateur et du machine learning en 5 à 10 minutes



 En continuant votre navigation sur ce site, vous acceptez l'utilisation des cookies  afin d'assurer le bon déroulement de votre visite et de réaliser des statistiques d'audience.



Nous constatons que les trous ont bien disparu sauf pour un... les bords n'étaient pas totalement fermés. Enfin, nous obtenons l'image résultat finale.

Découpage des différents objets

Deviens un  monstre de la vision par ordinateur et du machine learning en 5 à 10 minutes

 En continuant votre navigation sur ce site, vous acceptez l'utilisation des cookies  afin d'assurer le bon déroulement de votre visite et de réaliser des statistiques d'audience.

dans celle-ci. Cela se fait grâce aux fonctions `findContours`, `drawContours` et `boundingRect` d'OpenCV.

La fonction `findContours` nous retourne les contours des objets présents dans une image

binaire, elle prend en paramètre une image binaire, le mode, qui indique à la fonction les contours dont nous souhaitons obtenir cela peut être les contours extérieurs, intérieurs ou les deux. Enfin, l'argument `method` indique comment nous souhaitons que les contours soient représentés dans notre cas, ils seront représentés par une suite de points connectés.


```
contours, hierarchy = cv2.findContours(thresh, mode, method)
```



`DrawContours` va permettre de dessiner un à un sur des images vierges chacun des contours

extraits précédemment avec `find contours`. Elle prend en paramètre, une image sur laquelle la fonction va dessiner les contours, l'indice du contour que l'on souhaite dessiner, la couleur que l'on souhaite donner à ce contour et enfin l'épaisseur que l'on souhaite, -1 si l'on souhaite que le contour soit rempli.

```
cv2.drawContours(image, contours, contourIdx, couleur, thickness)
```

Pour finir `boundingrect` est une fonction qui retourne les coordonnées de la boundingbox d'un contour c'est-à-dire les coordonnées du carré de taille minimum contenant le contour. Cette fonction prend en paramètre le contour et retourne les coordonnées de sa bounding box.


Deviens un  monstre de la vision par ordinateur et du machine learning en 5 à 10 minutes



 En continuant votre navigation sur ce site, vous acceptez l'utilisation des cookies  afin d'assurer le bon déroulement de votre visite et de réaliser des statistiques d'audience.

code :

```
import cv2
import numpy as np
img=cv2.imread ("fleur.png");
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
h,s,v= cv2.split(hsv)
ret_h, th_h = cv2.threshold(h,0,255,cv2.THRESH_BINARY+cv2.
ret_s, th_s = cv2.threshold(s,0,255,cv2.THRESH_BINARY+cv2.
#Fusion th_h et th_s
th=cv2.bitwise_or(th_h,th_s)
#Ajouts de bord à l'image
bordersize=10
th=cv2.copyMakeBorder(th, top=bordersize, bottom=bordersiz
#Remplissage des contours
im_floodfill = th.copy()
h, w = th.shape[:2]
mask = np.zeros((h+2, w+2), np.uint8)
cv2.floodFill(im_floodfill, mask, (0,0), 255)
im_floodfill_inv = cv2.bitwise_not(im_floodfill)
th = th | im_floodfill_inv
#Enlèvement des bord de l'image
th=th[bordersize: len(th)-bordersize,bordersize: len(th[0]
resultat=cv2.bitwise_and(img,img,mask=th)
cv2.imwrite("im_floodfill.png",im_floodfill)
cv2.imwrite("th.png",th)
cv2.imwrite("resultat.png",resultat)
contours, hierarchy = cv2.findContours(th,cv2.RETR_TREE,cv
for i in range (0, len(contours)) :
    mask_BB_i = np.zeros((len(th),len(th[0])), np.uint8)
    x,y,w,h = cv2.boundingRect(contours[i])
    cv2.drawContours(mask_BB_i, contours, i, (255,255,255)
    BB_i=cv2.bitwise_and(img,img,mask=mask_BB_i)
    if h >15 and w>15 :
        BB_i=BB_i[y:y+h,x:x+w]
        cv2.imwrite("BB_"+str(i)+".png",BB_i)
```

Au début, c'est le même code que précédemment, jusqu'à la ligne 26. À la ligne 26, on extrait de l'image binaire th les contours des fleurs, Ret_tree signifie que l'on veut tout les contours et chain_approx_simple que les contours doivent être représentés par une suite de point. À la ligne 27, il y a une boucle qui va itérer sur tous les contours, à la ligne 28 une image vierge est créée, la bounding box du contour courant est extraite à la ligne 29, la zone englobée par le contour courant est dessinée sur l'image vierge qui a été créée au début de la boucle ligne 30.

Deviens un  monstre de la vision par ordinateur et du machine learning en 5 à 10 minutes

 En continuant votre navigation sur ce site, vous acceptez l'utilisation des cookies  afin d'assurer le bon déroulement de votre visite et de réaliser des statistiques d'audience.

qui ne sont pas des fleurs soient extraites.

Résultat :




Nous remarquons que toutes les fleurs ont été extraites de manière individuelle, mais pas que, beaucoup d'objets gênants ont aussi été extraits. Nous verrons prochainement comment automatiquement retirer ses objets.



Comment extraire des informations d'une image ?

La caractérisation d'objet est la deuxième étape primordiale d'un logiciel de reconnaissance d'images. Il s'agit de transformer l'imagette de chaque objet en valeurs quantitatives ou qualitatives afin d'obtenir une information traitable par les algorithmes de classification tels que les réseaux de neurones ou encore les arbres de décision. Cette information, se veut être une synthèse d'une ou plusieurs propriétés de l'objet, par exemple l'aire de l'objet est une synthèse de la taille de l'objet.

Il existe quatre grands types d'attributs permettant de décrire un objet à partir des pixels qui le compose : les attributs de région (aire, moments, etc ...), de contour (périmètre, les coefficients elliptiques de Fourier, etc...), de couleur (moyenne, histogramme, etc...) , et de texture (GLCM, filtre de Gabor, etc). OpenCV et scikit-image mettent à notre disposition un grand nombre de fonctions afin d'extraire chacun de ces attributs.

Pour des raisons de simplicité, dans ce chapitre, nous travaillerons exclusivement avec les 4 fleurs suivantes :

Deviens un  monstre de la vision par ordinateur et du machine learning en 5 à 10 minutes

 En continuant votre navigation sur ce site, vous acceptez l'utilisation des cookies  afin d'assurer le bon déroulement de votre visite et de réaliser des statistiques d'audience.



Les attributs de régions

En analyse d'image, la région d'un objet est définie comme étant l'étendue de pixel qui la compose. La figure ci-dessous illustre cette définition. À côté de chaque fleur, nous pouvons visualiser sa région.

Pour l'analyse des régions, OpenCV propose les différents attributs suivants :

L'aire

Obtenir l'aire d'une région consiste à compter le nombre de pixels qui la compose. La fonction OpenCV permettant de l'extraire est `contourArea(cnt)`, elle prend en paramètre le contour d'une région et retourne son aire.


Code :



```
area = cv2.contourArea(cnt)
```

Le périmètre

Tout comme il est possible d'extraire l'aire d'un objet, il est également possible d'en extraire le périmètre. Pour cela, il faut utiliser la fonction `arcLength`, qui va compter le nombre de pixels entourant l'objet. Elle prend en paramètre le contour de l'objet et un booléen indiquant si l'objet est une courbe où non et retourne son périmètre.

Code :

Deviens un  monstre de la vision par ordinateur et du machine learning en 5 à 10 minutes

 En continuant votre navigation sur ce site, vous acceptez l'utilisation des cookies  afin d'assurer le bon déroulement de votre visite et de réaliser des statistiques d'audience.

si l'objet est rond. Plus l'objet est rond et plus cette mesure sera proche de 1 et inversement moins l'objet sera rond et moins cette mesure sera proche de 1. La circularité s'obtient via la formule suivante :

$$circularity = \frac{4 \pi Area}{Perimeter^2}$$

Code :

```
circularity= 4*area/(perimeter*perimeter)
```

La longueur et la hauteur de la bounding box contenant la région.

Il existe d'autres caractéristiques de région basées sur la bounding box de l'objet par exemple la longueur et la hauteur de la bounding box.

Code :


```
x,y,w,h = cv.boundingRect(cnt)
```



L'aspect Ratio.

C'est le quotient de la longueur divisé par la hauteur.

Code :

```
aspect_ratio=w/h
```

Deviens un  monstre de la vision par ordinateur et du machine learning en 5 à 10 minutes

 En continuant votre navigation sur ce site, vous acceptez l'utilisation des cookies  afin d'assurer le bon déroulement de votre visite et de réaliser des statistiques d'audience.

Code :

```
area = cv.contourArea(cnt)
x,y,w,h = cv.boundingRect(cnt)
rect_area = w*h
extent = float(area)/rect_area
```

La solidité

La solidité est le quotient de l'aire d'un objet par l'aire de sa forme convexe.

$$\text{solidité} = \frac{\text{area}}{\text{hull area}}$$

Code :

```
area = cv.contourArea(cnt)
hull = cv.convexHull(cnt)
hull_area = cv.contourArea(hull)
solidity = float(area)/hull_area
```


Le diamètre équivalent

Le diamètre équivalent est le diamètre du cercle ayant la même aire que le contour.



$$\text{Diamètre équivalent} = \sqrt{4 \text{ Area} / \pi}$$

Code :

```
area = cv.contourArea(cnt)
equi_diameter = np.sqrt(4*area/np.pi)
```

Deviens un  monstre de la vision par ordinateur et du machine learning en 5 à 10 minutes


Exemple :



 En continuant votre navigation sur ce site, vous acceptez l'utilisation des cookies  afin d'assurer le bon déroulement de votre visite et de réaliser des statistiques d'audience.

Code :

```
import cv2
import math
fleurs=[] # chargement des images
fleurs.append(cv2.imread ("fleur1.png"));
fleurs.append(cv2.imread ("fleur2.png"));
fleurs.append(cv2.imread ("fleur3.png"));
fleurs.append(cv2.imread ("fleur4.png"));
gray_fleurs=[]
for i in range (0, len(fleurs)): # transformation en ni
veau de gris
    gray_fleurs.append(cv2.cvtColor(fleurs[i], cv2.COLOR
_BGR2GRAY))
th_fleurs=[]
for i in range (0, len(fleurs)): # seuillage
    ret,th = cv2.threshold(gray_fleurs[i],5,255,cv2.THRE
SH_BINARY)
    th_fleurs.append(th)
contours_fleurs=[] # extraction des contours
for i in range (0, len(fleurs)):
    contours, hierarchy = cv2.findContours(th_fleurs[i],
cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
    contours_fleurs.append(contours)
airs_fleurs=[]
perimetres_fleurs=[]
circularités_fleurs=[]
extent_fleurs=[]
aspect_ratio_fleurs=[]
for i in range (0, len(fleurs)): # extraction des carac
téristiques
    airs_fleurs.append(cv2.contourArea(contours_fleurs
[i][0]))
    perimetres_fleurs.append(cv2.arcLength(contours_fleu
rs[i][0],True))
    circularités_fleurs.append(4*math.pi*airs_fleurs
[i]/(perimetres_fleurs[i]*perimetres_fleurs[i]))
    x,y,w,h = cv2.boundingRect(contours_fleurs[i][0])
    extent_fleurs.append(w/h)
    aspect_ratio_fleurs.append(airs_fleurs[i]/(w*h))
    contours_fleurs.append(contours)
for i in range (0, len(fleurs)): # affichage
    print ("fleur",i," : ", " air :", airs_fleurs[i],
erimetre :", perimetres_fleurs[i], "circularité :",
    circularités_fleurs[i], "extent :", extent_fleurs
[i], "aspect ratio:", aspect_ratio_fleurs[i])
```

Nous chargeons les images de la ligne 10 à 12, nous passons celle-ci en niveau de gris afin de pouvoir de la ligne 14 à 17 les


Deviens un  monstre de la vision par ordinateur et du machine learning en 5 à 10 minutes



 En continuant votre navigation sur ce site, vous acceptez l'utilisation des cookies  afin d'assurer le bon déroulement de votre visite et de réaliser des statistiques d'audience.

Résultat :

fleur 0 : aire : 4663.0 périmètre :
685.8376532793045 circularité :
0.12457549745517907 extent : aspect ratio:
0.26860599078341013
fleur 1 : aire : 97072.5 périmètre :
1532.5209821462631 circularité :
0.5193895650343685 extent : aspect ratio:
0.6075575027382256
fleur 2 : aire : 215627.0 périmètre :
2436.507910966873 circularité :
0.45643333292163246 extent : aspect ratio:
0.7069158265715925
fleur 3 : air : 29071.0 périmètre :
869.5777685642242 circularité :
0.4831177331746388 extent : aspect ratio:
0.4716255678131084

JE M'INSCRIS !

Deviens un 
monstre de la
vision par
ordinateur et du
machine learning
en 5 à 10 minutes

 En continuant votre navigation sur ce site, vous acceptez l'utilisation des cookies 
afin d'assurer le bon déroulement de votre visite et de réaliser des statistiques
d'audience.