Informatique

**Matrices de pixels et images**

***Détection et extraction de panneaux***

# Contexte

Nous allons réaliser un algorithme qui permet d’identifier, d’extraire et de redimensionner des panneaux de circulation à partir de photos prises dans la circulation routière. Pour cette étude, nous identifierons les panneaux cerclés de rouge. Voici quelques exemples d’application de l’algorithme à à partir de photos issues d’internet :

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  | Une image contenant texte, signalisation, Panneau de signalisation, plein air  Description générée automatiquement |

Nous verrons en deuxième année (ITC2) un algorithme d’intelligence artificielle (TD 3-1 - Algorithme KNN) capable de les reconnaître automatiquement.



Pour cette application, nous allons utiliser l’image ci-contre nommée « Image\_1.bmp » de format 800x600 au format RGB.

Vous avez en annexe à la fin de ce sujet un extrait de l’aide de Python à propos des modules matplotlib et numpy.

Les images sous python

On ouvre les images avec la fonction imread du module matplotlib.

* 1. Préciser le type des images ainsi ouvertes
  2. Préciser le type de codage des nombres dans ces images
  3. En détaillant le calcul, préciser le nombre d’octets que prend un pixel dans la mémoire de l’ordinateur
  4. Préciser la taille approximative de l’image fournie en Mo arrondie au dixième
  5. Sachant qu’un calcul dans un ordinateur est de l’ordre de la micro seconde, donner une estimation du temps de traitement de cette image avec python

Ouverture de l’image

L’image bmp est placée dans le répertoire où notre fichier Python est stocké.

* 1. Créer une fonction Affiche(fig,im) qui permet d’afficher l’image im (format issu de imread) sur la figure fig
  2. Ecrire les lignes de code créant l’image « Image », transformant en triplets si quadruplets le cas échéant, et l’affichant sur la figure 1

Dans la suite, l’argument im lors de la définition des fonctions sera cette image en couleurs.

Traitement noir et blanc

On souhaite pouvoir utiliser numpy dans la suite en l’appelant np.

* 1. Ecrire l’instruction réalisant cet import

Nous devons sélectionner le rouge dans l’image étudiée. Un simple critère sur les valeurs seules R, G et B des pixels n’est pas suffisant. Dans l’ouvrage collectif coordonné par Philippe FOUCHER de titre « Détection et reconnaissance de la signalisation verticale par analyse d’images », on trouve une référence [18] page 24 au travail de G. DUTILLEUX et P. CHARBONNIER intitulé « Détection de signalisation routière par ajustement de formes prototypes » dont voici un extrait :

Une image contenant texte, Police, nombre, ligne

Description générée automatiquement

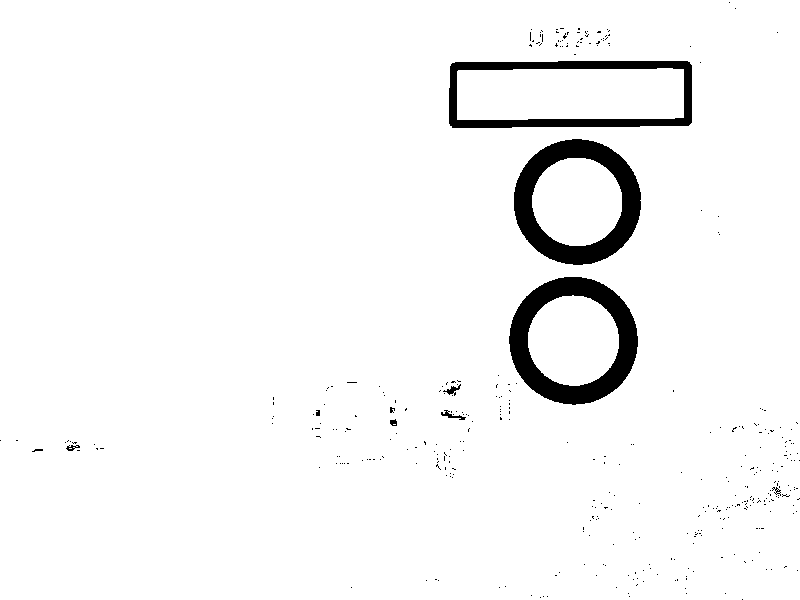
Lorsqu’un pixel est considéré rouge, on le mettra en noir sur l’image noir et blanc, blanc sinon.

* 1. Créer une fonction NB(im,alpha) qui renvoie une nouvelle image (format array de numpy) qui sera la transformation de l’image im en noir et blancs en respectant les critères proposés ci-dessus (im non modifiée)

On choisit .

* 1. Ecrire les lignes de codes créant l’image noir et blanc « Image\_NB » et l’affichant sur la figure 2

Voici l’image obtenue (cadre ombré ajouté dans ce document pour que les bords soient visibles) :



* 1. Préciser la complexité en temps de la fonction NB

Sur l’image obtenue, on remarque beaucoup de points noirs isolés. On souhaite proposer un traitement afin de les faire disparaitre. On donne le code suivant :

|  |
| --- |
| **def** Convolution**(**im**,**K**):**  Nl**,**Nc **=** im**.**shape**[**0**:**2**]**  Dim\_K **=** K**.**shape**[**0**]**  k **=** Dim\_K**//**2  Im\_Conv **=** np**.**copy**(**im**)**  **for** c **in** range**(**k**,**Nc**-**k**):**  **for** l **in** range**(**k**,**Nl**-**k**):**  Im\_Loc **=** im**[**l**-**k**:**l**+**k**+**1**,**c**-**k**:**c**+**k**+**1**]**  R**,**G**,**B **=** 0**,**0**,**0  n **=** K**.**shape**[**0**]**  **for** ll **in** range**(**n**):**  **for** cc **in** range**(**n**):**  tk **=** K**[**ll**,**cc**]**  RM**,**GM**,**BM **=** Im\_Loc**[**ll**,**cc**]**  R **+=** tk**\***RM  G **+=** tk**\***GM  B **+=** tk**\***BM  R**,**G**,**B **=** int**(**R**),**int**(**G**),**int**(**B**)**  R**,**G**,**B **=** max**(**min**(**R**,**255**),**0**),**max**(**min**(**G**,**255**),**0**),**max**(**min**(**B**,**255**),**0**)**  Im\_Conv**[**l**,**c**]** **=** R**,**G**,**B  **return** Im\_Conv |

On souhaite proposer une fonction qui, à partir d’une image en nuances de gris , renvoie une nouvelle image en noir et blanc selon le critère : .

* 1. Proposer une fonction Seuil(imng,k) prenant en argument une image en nuances de gris et un seuil k, et renvoyant une nouvelle image en noir et blanc comme précisé (imng non modifiée)
  2. Proposer un traitement à l’image « Image\_NB » afin d’obtenir le résultat ci-dessous qui sera affiché sur la figure 3

Remarque : l’image Image\_NB sera écrasée et remplacée par le résultat du traitement

Une image contenant noir et blanc, cercle, capture d’écran, croquis

Description générée automatiquement

Dans la suite, l’argument imnb lors de la définition des fonctions sera cette image en noir et blanc.

Etude des zones de l’image

L’image obtenue ci-dessus présente une grande « zone » blanche et un certain nombre de « zones » noires. Nous allons programmer une méthode permettant d’associer un numéro à chacune de ces zones afin de pouvoir les dénombrer et de les étudier.

Une image contenant texte, Police, capture d’écran

Description générée automatiquementPour chaque pixel de l’image étudiée désigné par une liste [l,c] de sa ligne l et sa colonne c, on souhaite déterminer chacun de ses 4 voisins (gauche, droite, bas, haut) dans cet ordre sous la forme d’une liste de 4 listes [li,ci]. Un voisin qui sortirait de l’image sera défini comme égal au pixel [l,c].

* 1. Proposer une fonction Liste\_voisins(l,c,Nl,Nc) prenant en argument la ligne l et colonne c du pixel étudié, les nombres de lignes Nl et colonnes Nc de l’image, et renvoyant la liste de ses 4 voisins comme précisé ci-dessus

On suppose avoir à disposition une table T sous forme d’array à deux dimensions, de mêmes dimensions que l’image étudiée, contenant des entiers codés sur 64 bits. Elle est initialisée à des valeurs de -1 partout. On souhaite que cette table contienne par la suite des entiers positifs croissants tels que toutes les cases contenant l’entier k représentent une « zone » numéro k de l’image noir et blanc.

Ainsi, comme le premier pixel en haut à gauche est blanc, la première zone de numéro k=0 sera l’intégralité du blanc de l’image ne formant qu’une zone. La zone 1 sera une première zone noire, la zone deux une seconde, etc.

Une image contenant texte, horloge

Description générée automatiquement

Voici un exemple de table T associée à l’image de 8x8 pixels ci-contre en réalisant un parcours ligne par ligne (pour chaque ligne, puis pour chaque colonne) de la procédure que nous allons mettre en place :

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  | | |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 2 | | 0 | 3 | 3 | 0 | 0 | 1 | 0 | 0 | | 0 | 3 | 3 | 0 | 0 | 1 | 0 | 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | | 5 | 0 | 6 | 0 | 5 | 0 | 4 | 4 | | 5 | 0 | 0 | 0 | 5 | 0 | 4 | 4 | | 5 | 5 | 5 | 5 | 5 | 0 | 0 | 0 | |

Ce qui donnera, en affichant la table T avec notre fonction Affiche (lorsque ce sont des entiers, et non des triplets, on obtient un dégradé de couleurs de bleu à jaune automatiquement) :

Une image contenant texte, trousse de secours, horloge

Description générée automatiquement

Dans la suite, on dira qu’une case T[l,c] a été **marquée** si sa valeur T[l,c] est différente de -1.

Par ailleurs, on dira par abus de langage « **pixel [l,c]** » pour désigner le pixel à la ligne l et la colonne c et « **case [l,c]** » la valeur de la case à la ligne l et la colonne c de la table T.

Enfin, comme nous travaillerons sur l’image en noir et blanc, on parlera de « **valeur** » 0 ou 255 pour identifier les valeurs R=G=B de tous les pixels.

On souhaite créer une procédure qui, à partir d’un pixel initial [l,c] quelconque de l’image associé à une case T[l,c] initialisée à un entier k dans T, explore tous ses voisins non encore marqués ayant la même valeur, afin de les marquer du même entier k, et procède ainsi pour tous les voisins des voisins (etc.), tant qu’il y en a. Cette procédure va donc permettre de remplir la table T avec le même entier k pour chaque zone de l’image.

Principe de la fonction d’exploration à partir du pixel initial [l,c] :

* Initialiser une pile (liste) avec le couple [l,c] du pixel initial
* Tant que la pile n’est pas vide :
  + Dépiler un pixel [l,c]
  + Affecter l’entier k à la case [l,c]
  + Déterminer tous les voisins du pixel [l,c]
  + Pour chaque voisin non marqué de même valeur :
    - Ajouter ce voisin à la pile

*Attention : Tout n’est volontairement pas dit*

Exemples : L’appel de la fonction en

* [0,0] va créer toute la zone de 0 de la table T
* [0,5] va créer toute la zone de 1 dans la table T
  1. Proposer une fonction Explorer(imnb,l,c,T,k) prenant en argument la ligne l et la colonne c du pixel initial et l’entier k, et réalisant la procédure attendue

Il reste maintenant à appeler cette procédure sur l’intégralité des pixels non marqués de la table T sur l’image en noir et blanc à partir du pixel [0,0].

Principe de la fonction de détermination des zones :

* Initialisation d’une table T d’entiers à -1 (dtype**=**'int64')
* Initialisation de l’entier k à 0
* Parcours de tous les pixels ligne par ligne
* Exploration du pixel s’il n’est pas marqué en lui attribuant l’entier k (création de la zone k)
* Incrémentation de l’entier k

*Attention : Tout n’est volontairement pas dit*

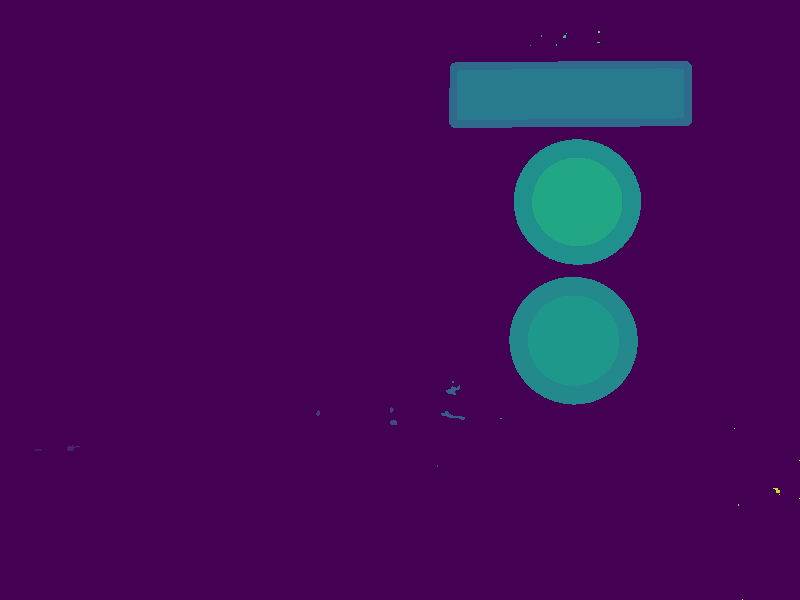
* 1. Créer la fonction Zones(imnb) réalisant cette procédure et renvoyant la table T associée à imnb
  2. Ecrire les lignes de code créant la table « Table » associée à l’image fournie, l’affichant sur la figure 4, et créant et affichant dans la console le nombre de zones trouvées « Nb\_Zones »

Remarque : On pourra utiliser np.amax(A) pour obtenir le maximum d’un array A.

Dans la suite, les arguments n et T lors de la définition des fonctions seront le nombre de zones « Nb\_Zones » et la table « Table ».

Vous devriez obtenir ceci :





On souhaite mettre en place une fonction qui, en un seul parcours de la table T, renvoie une liste LD des données D de chaque zone LD[k]=D=[Taille,Ml,Mc,Col,l\_min,c\_min,l\_max,c\_max] avec :

* Taille : nombre de pixels de la zone k (entier)
* Ml : ligne moyenne de la zone k (flottant)
* Mc : colonne moyenne de la zone k (flottant)
* Col : valeur R du triplet RBG de la zone k (0 : Noir, 255 : Blanc)
* l\_min,c\_min,l\_max,c\_max sont les limites de la zone k (incluses)

Remarque : le point [Ml,Mc] est le centre de la zone k sur l’image. Pour rappel, on obtient le centre (L,C) d’un ensemble de n pixels (Li,Ci) ainsi :

On propose le code suivant :

|  |
| --- |
| **def** Donnees**(**T**,**n**,**imnb**):**  Nl**,**Nc **=** T**.**shape  Taille**,**Ml**,**Mc**,**Col**,**l\_min**,**c\_min**,**l\_max**,**c\_max **=** 0**,**0**,**0**,-**1**,**# ZONE 1 #  LD **=** **[[**Taille**,**Ml**,**Mc**,**Col**,**l\_min**,**c\_min**,**l\_max**,**c\_max**]** **for** k **in** range**(**n**)]**  **for** c **in** range**(**Nc**):**  **for** l **in** range**(**Nl**):**  k **=** # ZONE 2 #  Taille**,**Ml**,**Mc**,**\_**,**l\_min**,**c\_min**,**l\_max**,**c\_max **=** LD**[**k**]**  Ml **=** # ZONE 3 #  Mc **=** # ZONE 4 #  Taille **+=** 1  Col **=** # ZONE 5 #  **if** l **<** l\_min**:**  # ZONE 6 #  **if** l **>** l\_max**:**  # ZONE 7 #  **if** c **<** c\_min**:**  # ZONE 8 #  **if** c **>** c\_max**:**  # ZONE 9 #  LD**[**k**]** **=** **[**Taille**,**Ml**,**Mc**,**Col**,**l\_min**,**c\_min**,**l\_max**,**c\_max**]**  **return** LD |

* 1. Compléter les différentes zones du code proposé et proposer les lignes de code permettant de créer la liste « Donnees\_Zones » contenant les données issues de la fonction Donnees

Dans la suite, l’argument LD lors de la définition des fonctions sera cette liste « Donnees\_Zones ».

Redimensionnement

Dans la suite, nous allons être en mesure d’isoler et d’extraire automatiquement les panneaux de l’image étudiée. Il nous faudra transformer des images rectangulaires en images carrées, que ce soit pour l’identification ou l’extraction de ces panneaux, comme illustré ci-dessous :



Pour cela, nous allons réaliser un algorithme de redimensionnement par interpolation bilinéaire.

On considère 4 points dits « Origine » , , et  dans une image aux lignes et colonnes précisées ci-dessous :

Les 4 points peuvent être placés dans l’image (pas forcément sur les bords), mais :

* et doivent être à la même ligne, de même que et
* et doivent être à la même colonne, de même que et

A chaque point de cette image est associée un n-uplet noté . On souhaite déterminer le n-uplet associé à un point quelconque de l’image par interpolation bilinéaire. Nous avons réalisé cette démarche dans le cours sur l’agrandissement avec les triplets RGB de points encadrants une zone à remplir, je ne rappelle donc ici que la fonction utile pour la suite :

On obtiendra par exemple , et si est le centre des 4 points , .

Soit une image blanche de dimensions dite « origine » et l’image à recadrer dite « cible » de dimensions . En parcourant toute l’image « origine », on va pour chaque point déterminer par interpolation bilinéaire les coordonnées du point de l’image « cible » à recadrer . On appliquera alors au point de l’image origine la couleur du point dans l’image cible si ce point en fait partie.

Autrement dit, en définissant les coordonnées des 4 coins dans l’image à recadrer :

* On calcule les coordonnées du pixel associé dans l’image cible
* A partir d’une image d’origine blanche, on applique la couleur dans l’image origine du pixel si et . Toutefois, dans notre application, il ne sera pas nécessaire de réaliser ces deux tests car les points étant les 4 coins de l’image origine, leurs images seront incluses dans le quadrilatère délimité par les points

On ne travaillera qu’avec des listes sauf pour les valeurs des pixels qui seront des arrays.

On propose les deux fonctions suivantes :

|  |
| --- |
| **def** Prod\_MV**(**M**,**V**):**  Res **=** **[]**  n **=** len**(**M**)**  t **=** len**(**M**[**0**][**0**])**  **for** l **in** range**(**n**):**  Res\_Loc **=** **[**0 **for** \_ **in** range**(**t**)]**  **for** c **in** range**(**n**):**  **for** i **in** range**(**t**):**  Res\_Loc**[**i**]** **+=** V**[**c**]\***M**[**l**][**c**][**i**]**  Res**.**append**(**Res\_Loc**)**  **return** Res  **def** Prod\_VV**(**V1**,**V2**):**  n **=** len**(**V1**)**  t **=** len**(**V1**[**0**])**  Res **=** **[**0 **for** \_ **in** range**(**t**)]**  **for** i **in** range**(**n**):**  **for** j **in** range**(**t**):**  Res**[**j**]** **+=** V1**[**i**][**j**]\***V2**[**i**]**  **return** Res |

* 1. Proposer une fonction LC\_Bilin(l,c,LO,LC) prenant en argument les coordonnées l,c d’un point de l’image origine, les listes LO et LC, et renvoyant les coordonnées L,C associées au couple l,c par interpolation bilinéaire

A partir d’une image au format array, on veut obtenir la liste des listes (li,ci) de ses 4 coins automatiquement et dans l’ordre : haut gauche, haut droit, bas gauche, bas droit. Exemple sur une image 50x100 :

Une image contenant texte, Police, blanc, typographie

Description générée automatiquement

* 1. Proposer une fonction Coins(im) renvoyant la liste des 4 coins de l’image im

Pour rappel, dans notre application, l’image cible présentera la particularité d’être un rectangle (côtés horizontaux et verticaux) à redimensionner en carré.

* 1. Proposer la fonction Resize(im,dim) prenant en argument une image im rectangulaire et le tuple dim contenant les nombres de lignes et colonnes de l’image attendue, et renvoyant une nouvelle une image aux dimensions souhaitées (im non modifiée)

Voici un exemple d’application :



Une image contenant texte, plein air, Panneau de signalisation, signalisation

Description générée automatiquement

Sélection des panneaux



Nous souhaitons dans cette partie identifier automatiquement sur l’image étudiée les panneaux cerclés de rouge. Pour cela, nous avons à disposition dans le répertoire de travail l’image rouge « Cercle.bmp » ci-contre.

Pour comprendre où l’on va, la procédure globale de cette partie est la suivante :

* Transformation de l’image « Cercle » en image noir et blanc « Cercle\_NB »
* Redimensionnement de « Cercle\_NB » en 100x100
* Itérations sur toutes les zones k détectées par l’algorithme de détection de zones :
  + Extraction de la zone k dans l’image en noir et blanc avec une fonction « Extraction\_k » permettant d’extraire la zone par son numéro sans inclure d’autres zones. Ainsi, si d’autres zones sont proches du cercle ou dans le cercle, elles ne seront pas extraites
  + Redimensionnement de ces zones en 100x100
  + Calcul et normalisation de la distance entre chaque zone extraite et « Cercle\_NB » (dn)
  + Stockage des couples [dn,k] des distances normalisées et numéros des zones
* Sélection des zones ayant une « distance » inférieure à un critère « Crit »
* Extraction des zones sélectionnées dans leur totalité (cercle + contenu) à l’aide d’une fonction « Extraction »

|  |  |  |
| --- | --- | --- |
| Cercle\_NB | Zones proches de Cercle\_NB | Panneaux extraits |
|  |  |  |
|  |  |

Nous allons travailler avec des n-uplets (listes de n termes) qui seront associées à chaque image. On suppose que le nombre de termes n de chaque n-uplets est identique dans tout le sujet.

On rappelle que la distance euclidienne entre deux n-uplets et est le résultat du calcul suivant :

* 1. Créer une fonction Distance\_uv(u,v) renvoyant la distance euclidienne entre les deux n-uplets sous forme de listes u et v

Pour chaque image, on crée une liste L\_RGB des valeurs de R, G et B de chacun de ses pixels. Ainsi, avec des images ayant toujours la même dimension de 100x100 pixels, on obtient une liste de 30 000 valeurs pour chacune. Exemple d’application :

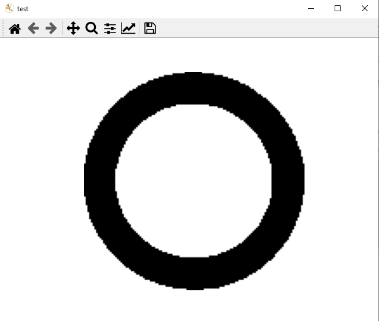
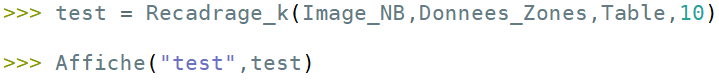
|  |  |
| --- | --- |
| Code | Figure |
| Noir **=** **[**0**,**0**,**0**]**  Blanc **=** **[**255**,**255**,**255**]**  L1 **=** **[**Noir**,**Blanc**,**Noir**]**  L2 **=** **[**Noir**,**Noir**,**Blanc**]**  L3 **=** **[**Noir**,**Blanc**,**Noir**]**  A **=** **[**L1**,**L2**,**L3**]**  Damier **=** np**.**array**(**A**,**dtype**=**'uint8'**)**  Affiche**(**'Damier'**,**Damier**)** |  |
|  | |

* 1. Créer une fonction Analyse(im) qui, à partir d’une image sous forme d’array, renvoie la liste L\_RGB associée – Attention, transformer les R, G et B en flottants, sinon il y aura overflow avec les uint8 lors des calculs de distances



Il nous faut maintenant pouvoir extraire une zone d’indice k dans l’image en noir et blanc. En passant la souris sur l’image de « Table » obtenue précédemment, on trouve le numéro du contour rouge du panneau supérieur : 10.

Nous allons proposer une fonction qui procède à l’extraction de ce contour de la sorte :



* 1. Créer la fonction Recadrage\_k(imnb,LD,T,k) renvoyant une nouvelle image aux dimensions de la zone k, et ne contenant que les pixels de cette zone dans l’image imnb

Remarque : Les panneaux circulaires se projettent et les cercles sont en réalité des ellipses sur Image\_NB. Le recadrage proposé dans ce DS déformera légèrement le contenu des panneaux. Pour aller plus loin, il faudra réaliser un algorithme de détection d’ellipses et un recadrage projectif.

On souhaite proposer une fonction qui réalise la procédure décrite en début de cette partie, c’est-à-dire :

* Initialiser une liste vide Res
* Redimensionner le cercle en 100x100
* Le transformer en noir et blanc
* En crée la liste L\_RGB
* Pour chaque zone k de l’image en noir et blanc
  + Créer l’image noir et blanc de la zone par recadrage
  + Redimensionner la zone en 100x100
  + Créer sa liste L\_RGB
  + Calculer la distance
  + Normaliser la distance en calculant (\*)
  + Ajouter à Res le couple [dn,k]
* Trier en place Res par ordre croissant des distances (utilisation de la méthode sort autorisée)
* Renvoyer Res

(\*) La normalisation consiste à exprimer les distances entre 0 et 1 quelles que soient la dimension des images comparées. Soit le nombre de lignes/colonnes des images (carrées) contenant donc pixels. Les listes L\_RGB contiennent donc valeurs. La distance maximale trouvée par distance euclidienne vaut :

On calculera donc

* 1. Créer la fonction Etude\_Motif(imnb,imc,T,LD,a) prenant en argument l’array imc associé à l’image du cercle, le côté des images redimensionnées a, et renvoyant la liste Res attendue
  2. Proposer le code créant la liste « Distances » issue de la fonction Etude\_Motif

On souhaite proposer une fonction qui renvoie la liste des données contenues dans « Donnees\_Zones » des zones les plus proches au sens de la norme euclidienne de « Cercle » selon un critère « dmax ».

* 1. Selection(Ldst,dmax,LD) renvoyant la liste attendue

Nous avons appliqué l’algorithme développé précédemment sur les 7 images présentées en début de ce sujet et affiché le résultat de la commande suivante :

|  |
| --- |
| **print([**round**(**t**[**0**],**2**)** **for** t **in** Distances**][**0**:**4**])** |

Voici les résultats obtenus:

|  |
| --- |
| Image\_1**:** **[**0.23**,** 0.23**,** 0.59**,** 0.59**]**  Image\_2**:** **[**0.23**,** 0.56**,** 0.56**,** 0.59**]**  Image\_3**:** **[**0.22**,** 0.59**,** 0.59**,** 0.62**]**  Image\_4**:** **[**0.26**,** 0.53**,** 0.59**,** 0.59**]**  Image\_5**:** **[**0.21**,** 0.59**,** 0.59**,** 0.59**]**  Image\_6**:** **[**0.24**,** 0.26**,** 0.59**,** 0.59**]**  Image\_7**:** **[**0.25**,** 0.59**,** 0.59**,** 0.59**]** |

* 1. Proposer la valeur du critère dmax à utiliser par la suite
  2. Proposer le code créant la liste « Zones\_Sel » des données des panneaux de l’image « Image »

Extraction des panneaux

Maintenant que nous avons sélectionné les panneaux, il ne reste plus qu’à les extraire de l’image en couleur et à les redimensionner. Pour récupérer une zone dans une image, il serait simple de récupérer la zone d’intérêt avec des slices : im**[**l\_min**:**l\_max**+**1**,**c\_max**:**c\_max**+**1**]**. Toutefois, nous interdisons les slices dans ce sujet.

* 1. Créer la fonction Recadrage(im,l\_min,l\_max,c\_min,c\_max) renvoyant une nouvelle image contenant la portion de im dans les limites (incluses) désignées par les 4 paramètres l\_min,l\_max,c\_min,c\_max

Pour chaque zone sélectionnée, il ne reste plus qu’à :

* Déterminer ses limites l\_min,l\_max,c\_min,c\_max dans « Image »
* La recadrer
* La redimensionner en 100x100
* L’afficher sur une figure dont le numéro sera 10 pour la première, 11 pour la seconde etc.
* Sauvegarder cette image au format bmp avec imsave pour ne pas la redimensionner à l’enregistrement



* 1. Créer le code attendu

Voici le résultat obtenu :

Il est maintenant temps de mettre en place un algorithme d’IA pour reconnaitre ces panneaux automatiquement !

ANNEXE

|  |  |
| --- | --- |
| **Extraits de l’aide numpy** | |
| shape | shape(a)  Return the shape of an array.  Parameters  ----------  a : array\_like  Input array.  Returns  -------  shape : tuple of ints  The elements of the shape tuple give the lengths of the  corresponding array dimensions. |
| copy | copy(a, order='K', subok=False)  Return an array copy of the given object.  Parameters  ----------  a : array\_like  Input data.  Returns  -------  arr : ndarray  Array interpretation of `a`. |
| ones | ones(shape, dtype=None, order='C', \*, like=None)  Return a new array of given shape and type, filled with ones.  Parameters  ----------  shape : int or sequence of ints  Shape of the new array, e.g., ``(2, 3)`` or ``2``.  dtype : data-type, optional  The desired data-type for the array, e.g., `numpy.int8`. Default is `numpy.float64`.  order : {'C', 'F'}, optional, default: C  Whether to store multi-dimensional data in row-major (C-style) or column-major (Fortran-style) order in memory.  like : array\_like  Reference object to allow the creation of arrays which are not NumPy arrays. If an array-like passed in as ``like`` supports the ``\_\_array\_function\_\_`` protocol, the result will be defined by it. In this case, it ensures the creation of an array object compatible with that passed in via this argument.  .. note::  The ``like`` keyword is an experimental feature pending on acceptance of :ref:`NEP 35 <NEP35>`.  .. versionadded:: 1.20.0  Returns  -------  out : ndarray  Array of ones with the given shape, dtype, and order.  See Also  --------  ones\_like : Return an array of ones with shape and type of input.  empty : Return a new uninitialized array.  zeros : Return a new array setting values to zero.  full : Return a new array of given shape filled with value.  Examples  --------  >>> np.ones(5)  array([1., 1., 1., 1., 1.])  >>> np.ones((5,), dtype=int)  array([1, 1, 1, 1, 1])  >>> np.ones((2, 1))  array([[1.],[1.]])  >>> s = (2,2)  >>> np.ones(s)  array([[1., 1.],[1., 1.]]) |

|  |  |
| --- | --- |
| **Extraits de l’aide matplotlib.pyplot** | |
| imread | imread(fname, format=None)  Read an image from a file into an array.  Parameters  ----------  fname : str or file-like  The image file to read: a filename, a URL or a file-like object opened in read-binary mode.  format : str, optional  The image file format assumed for reading the data. If not given, the format is deduced from the filename. If nothing can be deduced, PNG is tried.  Returns  -------  `numpy.array`  The image data. The returned array has shape  - (M, N) for grayscale images.  - (M, N, 3) for RGB images.  - (M, N, 4) for RGBA images. |
| imshow | imshow(X, cmap=None, norm=None, aspect=None, interpolation=None, alpha=None, vmin=None, vmax=None, origin=None, extent=None, \*, filternorm=True, filterrad=4.0, resample=None, url=None, data=None, \*\*kwargs)  Display data as an image, i.e., on a 2D regular raster.  The input may either be actual RGB(A) data, or 2D scalar data, which will be rendered as a pseudocolor image. For displaying a grayscale image set up the color mapping using the parameters ``cmap='gray', vmin=0, vmax=255``.  The number of pixels used to render an image is set by the axes size and the \*dpi\* of the figure. This can lead to aliasing artifacts when the image is resampled because the displayed image size will usually not match the size of \*X\* (see  :doc:`/gallery/images\_contours\_and\_fields/image\_antialiasing`).  The resampling can be controlled via the \*interpolation\* parameter and/or :rc:`image.interpolation`.  Parameters  ----------  X : array-like or PIL image  The image data. Supported array shapes are:  - (M, N): an image with scalar data. The values are mapped to colors using normalization and a colormap. See parameters \*norm\*,  \*cmap\*, \*vmin\*, \*vmax\*.  - (M, N, 3): an image with RGB values (0-1 float or 0-255 int).  - (M, N, 4): an image with RGBA values (0-1 float or 0-255 int), i.e. including transparency.  The first two dimensions (M, N) define the rows and columns of the image.  Out-of-range RGB(A) values are clipped. |
| figure | figure(num=None, figsize=None, dpi=None, facecolor=None, edgecolor=None, frameon=True, FigureClass=<class 'matplotlib.figure.Figure'>, clear=False, \*\*kwargs)  Create a new figure, or activate an existing figure.  Parameters  ----------  num : int or str, optional  A unique identifier for the figure. |
| axis | axis(\*args, emit=True, \*\*kwargs)  Convenience method to get or set some axis properties.  option : bool or str  If a bool, turns axis lines and labels on or off. If a string, possible values are:  ======== ==========================================================  Value Description  ======== ==========================================================  'on' Turn on axis lines and labels. Same as ``True``.  'off' Turn off axis lines and labels. Same as ``False``. |
| imsave | msave(fname, arr, \*\*kwargs)  Save an array as an image file.  Parameters  ----------  fname : str or path-like or file-like  A path or a file-like object to store the image in. If \*format\* is not set, then the output format is inferred from the extension of \*fname\*, if any, and from :rc:`savefig.format` otherwise. If \*format\* is set, it determines the output format.  arr : array-like  The image data. The shape can be one of MxN (luminance), MxNx3 (RGB) or MxNx4 (RGBA). |