# DS informatique

## I.1 Mesure du contraste

**Question 1** Préciser l’espace mémoire nécessaire pour stocker la valeur d’une composante, puis celle d’un pixel et enfin celle d’une image en Mo (= 1000ko) ou Mio (= 1024kio).

Une composante de couleur est un nombre compris entre 0 et 255. Il se stocke sur 1 octet.

Un pixel stocké au format RVB est composé de 3 nombres compris entre 0 et 255. Un pixel nécessite donc 3 octets de stockage.

Une image de 48 MPixels utilisera donc 3 *×* 48 *×* 106 octets, c’est à dire 144Mo.

**Question 2** Ecrire une fonction Clinear(val), qui prend en argument une valeur de l’espace non linéaire et qui renvoie la valeur linéarisée.

|  |
| --- |
| **def** Clinear(val):  **if** val<=0.04045:  Clin=val/12.92  **else**:  Clin=((val+0.055)/1.055)\*\*2.4  **return** Clin |

**Question 3** Ecrire une fonction Y(pix) qui prend en argument une liste de trois valeurs correspondant à un pixel au format RVB et qui renvoie la valeur *Y* du niveau de gris dans l’espace non linéaire.

|  |
| --- |
| **def** Y(pix):  Ylin = 0.2126\*Clinear(pix[0]) + 0.7152\* Clinear(pix[1]) + 0.0722\* Clinear(pix[2])  **if** Ylin<=0.0031308:  Y=12.92\*Ylin  **else**:  Y=1.055\*Ylin\*\*(1/2.4)-0.055  **return** Y |

**Question 4** Ecrire une fonction NiveauxGris(I) qui prend en argument une image *I* au format RVB et qui renvoie une image de même dimension en niveau de gris.

|  |
| --- |
| **def** NiveauxGris(I):  taillex,tailley,nb\_couleurs = np.shape(I) *# taille de l’image initiale* Igris=np.zeros(taillex,tailley) *# on crée l’image en niveaux de gris*  **for** x **in** range(taillex):  **for** y **in** range(tailley):  Igris[x,y]=Y(I[x,y])  **return** Igris |

**Question 5** Ecrire une fonction convolution(A,B) prenant en argument deux matrices de taille 3x3 et qui renvoie la valeur du produit de convolution.

|  |
| --- |
| **def** convolution(A,B):  conv=0 *# on initialise*  **for** i **in** range(3):  **for** j **in** range(3):  conv = conv + A[i,j]\*B[i,j]  **return** conv |

**Question 6** Ecrire une fonction contraste\_pixel(I,i,j) prenant en argument une image I au format niveaux de gris et les coordonnées du pixel (*i,j*) qui envoie la valeur du contraste défini précédemment par la quantité *c*.

|  |
| --- |
| **def** contraste\_pixel(I,i,j):  Gx=np.array([[ -1, 0, 1],  [ -2, 0, 2],  [ -1, 0, 1]])  Gy=np.array([[ -1, -2, -1],  [ 0, 0, 0],  [ 1, 2, 1]])  Ie=np.array([[0, 0, 0],  [0, 0, 0],  [0, 0, 0]])  **for** k **in** range(3):  **for** l **in** range(3):  Ie[k,l]=I[k-1+i,l-1+j]  conv1=convolution(Ie,Gx)  conv2=convolution(Ie,Gy) c=int(np.sqrt(conv1\*\*2+conv2\*\*2))  **return** c |

**Question 7** Ecrire une fonction contraste(I) prenant en argument une image I au format niveaux de gris et qui renvoie la valeur du contraste de référence *cref*.

|  |
| --- |
| **def** contraste(I):  taillex,tailley = np.shape(I) *# taille de l’image*  cref=0  **for** i **in** range(1,taillex-1):  **for** j **in** range(1,tailley-1): *# on retire les pixels des bords*  cref = cref + contraste\_pixel(I,i,j)  cref = cref / ((taillex-2)\*(tailley-2)) *# on fait la moyenne*  **return** cref |

**Question 8** Ecrire une fonction reglage, dont les arguments et les valeurs de retour sont à définir, répondant au comportement décrit en partant de la position 0 en pas. On supposera que le maximum de contraste existe.

|  |
| --- |
| **def** reglage(I,val):  ’’’  I : image initiale  val : position de l’objectif ’’’  cref=contraste(I) *# contraste de l’image initiale*  sens=1 *# sens de déplacement de l’objectif (1 : sens + ; -1 : sens -)*  position\_objectif(val+sens) *# première itération pour savoir dans quel sens il faut avancer*  Img1=prise() c=contraste(NiveauxGris((Img1))  **if** c<cref: *# si on est dans le mauvais sens, on repart de I* sens = -sens c=cref  **while** c<=cref:   cref=c  val = val+sens  position\_objectif(val)  Img1=prise()  c=contraste(Img1)  **return** cref |

Remarque : Cet algorithme est simpliste car il ne cherche le sens de déplacement de l’objectif qu’une fois, et uniquement entre 2 prises. Pour l’améliorer, il faudrait faire plusieurs mesures pour s’assurer du bon sens de déplacement.

## I.2 Détection de phase

**Question 9** Ecrire une fonction extraction(L1,L2,dec) prenant en argument deux listes L1 et L2 ainsi qu’une valeur entière de décalage et qui renvoie deux sous-listes à comparer de longueur len(L1)-dec, conformément aux règles présentées ci-dessus. On supposera que les deux listes L1 et L2 sont de même taille.

|  |
| --- |
| **def** extraction(L1,L2,dec):  **if** dec==0:  **return** L1,L2  **elif** dec>0:  **return** L1[:-dec],L2[dec:]  **else**:  **return** L1[-dec:],L2[:dec] |

**Question 10** Ecrire une fonction comparaison(L1,L2) (n’utilisant pas la comparaison interne de Python entre les listes) prenant en argument deux listes L1 et L2 qui renvoie True si les listes sont identiques et False sinon.

On peut prévoir le cas où les deux listes ne sont pas de même longueur (elles sont dans ce cas-là forcément différentes), même si cette fonction sera utilisée dans ce problème avec des listes de même longueur.

|  |
| --- |
| **def** comparaison(L1,L2):  *# Cas des longueurs différentes*  **if** len(L1) != len(L2):  **return False**  *# Cas des longueurs identiques*  **for** i **in** range(len(L1)):  **if** L1[i] != L2[i]:  **return False**  **return True** |

**Question 11** Ecrire une fonction recherche\_decalage(L1,L2) prenant en argument deux listes L1 et L2 et qui renvoie la valeur du décalage ou None s’il n’existe pas.

|  |
| --- |
| **def** recherche\_decalage(L1,L2):  dec=-80  **for** dec in range (-80 ;80): *# décalage compris entre -80 et 80 inclus*  L1e, L2e = extraction(L1,L2,dec)  **if** comparaison(L1e, L2e):  **return** dec  **return None** |

**Question 12** Evaluer la complexité de la fonction recherche\_decalage(L1,L2) en prenant en compte le nombre de comparaisons en fonction de *n* la taille des listes et de *m* le nombre de décalages maximal à prendre en compte (161 dans notre exemple) dans le meilleur et dans le pire des cas.

L’extraction se fait en *O*(*n − |dec|*) opérations élémentaires, tout comme la comparaison des listes extraites. Chaque itération de la boucle while a donc une complexité en *O*(*n − |dec|*) opérations élémentaires, soit en *O*(*n*) opérations élémentaires (en négligeant *dec* devant *n*).

Dans le pire cas, il n’existe pas de décalage. La boucle while sera donc itérée *m* fois : la complexité totale sera en *O*(*n × m*) opérations élémentaires. Dans le meilleur cas, le décalage est de -80, une seule itération de la boucle while sera nécessaire. La complexité totale sera en *O*(*n*) opérations élémentaires.

|  |
| --- |
| **def** erreur(L1,L2):  err=0  n=len(L1)  **for** i **in** range(n):  err = err + (L1[i]-L2[i])\*\*2  err = err/n  **return** err |

**Question 13** Ecrire une fonction erreur(L1,L2) qui retourne l’erreur quadratique définie ci-dessus.

**Question 14** Ecrire une fonction recherche\_decalage\_2(L1,L2) qui cherche le minimum de l’erreur pour des décalages de *−*80 à 80 et qui retourne la valeur de ce décalage. Evaluer la complexité de cette fonction.

|  |
| --- |
| **def** recherche\_decalage\_2(L1,L2):  *# Initialisation de la recherche du minimum* decalage\_min = 0  valeur\_min = -1  *# Valeur aberrante pour l’initialisation (l’erreur quadratique ne peut pas être négative)*  **for** dec **in** range (-80, 80+1): *# de -80 à +80 inclus*  L1e, L2e = extraction(L1,L2,dec)   err=erreur(L1e,L2e)  **if** valeur\_min > err **or** valeur\_min < 0: *# On retient le premier minimum* decalage\_min = dec  valeur\_min = err  dec = dec+1  **return** decalage\_min |

La fonction erreur est de complexité O(n) et on réalise avec cette fonction m fois le calcul de l’erreur. Donc la complexité est d’ordre O(mn).

## I.3 Comparaison des deux méthodes

**Question 15** Décrire en 5 lignes maximum les avantages et les inconvénients de ces deux méthodes et laquelle vous semble la plus pertinente.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | **Mesure du contraste** | | |
| **Avantages** | | Simplicité de programmation. | | |
| **Inconvénients** | | Calculs itératifs donc position finale plus lente à trouver. On suppose que le contraste possède un minimum. | | |
|  | | **Détection de phase** |
| **Avantages** | | Détermination directe du décalage nécessaire. |
| **Inconvénients** | | Risque d’instabilité plus grand. |

## I.4 Commande du moteur pas à pas

**Question 16** Ecrire une fonction faire\_un\_pas\_positif(pas\_actuel) qui prend en argument la valeur du pas courant qui modifie l’état des sorties IN1 à IN4 et qui renvoie la nouvelle valeur du pas. On veillera à ne changer l’état que des sorties nécessaires.

|  |
| --- |
| **def** faire\_un\_pas\_positif(pas\_actuel):  pas=(pas\_actuel+1) % 8  **if** pas==0:  modif\_sortie(IN4,**False**)  **elif** pas==1:   modif\_sortie(IN3,**True**)  **elif** pas==2:  modif\_sortie(IN1,**False**)  **elif** pas==3:  modif\_sortie(IN2,**True**)  **elif** pas==4:  modif\_sortie(IN3,**False**)  **elif** pas==5:  modif\_sortie(IN4,**True**)  **elif** pas==6:  modif\_sortie(IN2,**False**)  **else**:  modif\_sortie(IN1,**True**)  **return**  pas\_actuel+1 |

On peut compacter ce code en relevant quelques règles :

* Si le pas à réaliser est pair, alors il faut passer une sortie à 0, sinon à 1.
* L’ordre des sorties concernées par des pas est : 4,3,1,2,3,4,2,1.

|  |
| --- |
| **def** faire\_un\_pas\_positif(pas\_actuel):  pas=(pas\_actuel+1) % 8  valeur = (pas % 2 == 1)  sortie = [IN4,IN3,IN1,IN2,IN3,IN4,IN2,IN1][pas] modif\_sortie(sortie,valeur)  **return**  pas\_actuel+1 |

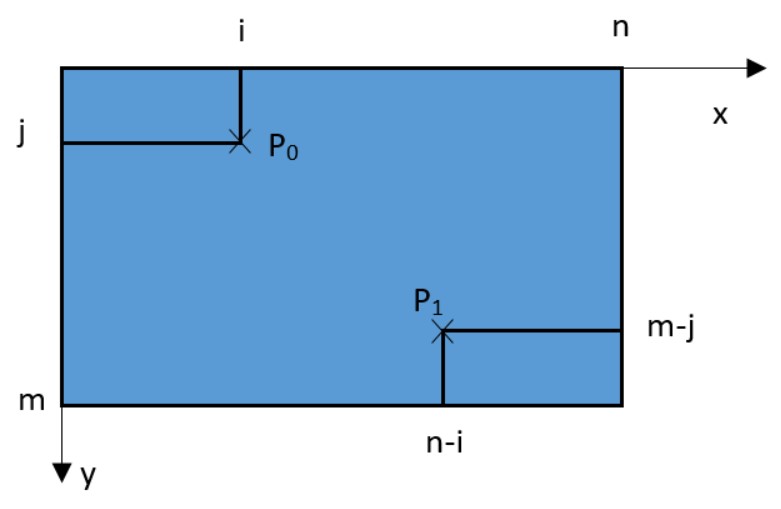
**Question 17** Ecrire une fonction position\_objectif(pas) qui prend en argument la position en pas à atteindre à partir de la position courante réalisant le déplacement demandé pas à pas. Il peut y avoir plus d’un pas à effectuer. On vérifiera que le pas demandé est atteignable (compris entre 0 et 1000).

|  |
| --- |
| **def** position\_objectif(pas):  **if** pas>=0 **and** pas<=1000:  **if** pas>pas\_courant:  **for** i **in** range(pas-pas\_courant):  faire\_un\_pas\_positif(pas\_courant+i)  **else**:  **for** i **in** range(pas\_courant-pas):  faire\_un\_pas\_negatif(pas\_courant-i) |

# II Gestion des photographies

**Question 18** Illustrer sur un schéma les nouvelles coordonnées d’un pixel de coordonnées (*i,j*) après une rotation de 180 degrés. Ecrire une fonction rotation\_180(image) qui prend en argument une image au format RVB et qui renvoie une nouvelle image pivotée de 180°.

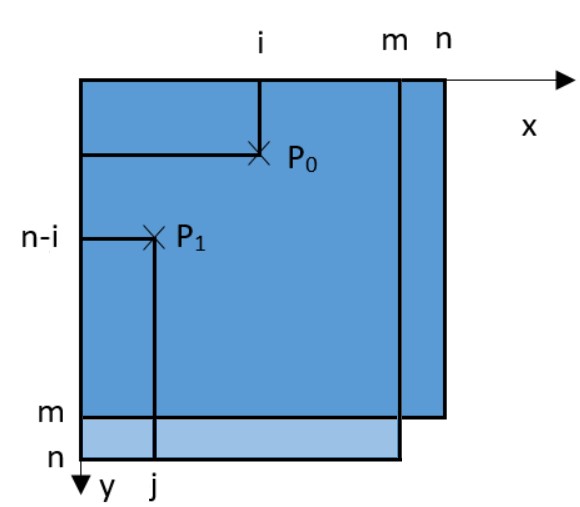
Soit une image de dimensions *n × m*. Un point de coordonnées (*i,j*) se retrouvera en position (*n − i,m − j*).



|  |
| --- |
| **def** rotation\_180(image):  taillex,tailley,nb\_couleurs = np.shape(image) nouvelle\_image=np.zeros(taillex,tailley,nb\_couleurs)  **for** x **in** range(taillex):  **for** y **in** range(tailley):  **for** c **in** range(nb\_couleurs):  nouvelle\_image[x,y,c]=image[taillex-x,tailley-y,c]  **return** nouvelle\_image |

**Question 19** Illustrer sur un schéma les nouvelles coordonnées d’un pixel de coordonnées (*i,j*) après une rotation de 90 degrés. Ecrire une fonction rotation\_90(image) qui prend en argument une image au format RVB et qui renvoie une nouvelle image pivotée de 90° dans le sens trigonométrique.

Soit une image de dimensions *n × m*. Un point de coordonnées (*i,j*) se retrouvera en position (*j,n − i*).



|  |
| --- |
| **def** rotation\_90(image):  taillex,tailley,nb\_couleurs = np.shape(image) nouvelle\_image=np.zeros(tailley,taillex,nb\_couleurs)  **for** x **in** range(taillex):  **for** y **in** range(tailley):  **for** c **in** range(nb\_couleurs):  nouvelle\_image[x,y,c]=image[y,taillex-x,c]  **return** nouvelle\_image |