TP8 - Traitement d'images

La première partie de ce document est un point de cours sur les images. Le TP et les questions associées se trouvent en deuxième partie.

I Rappel Traitement d'images

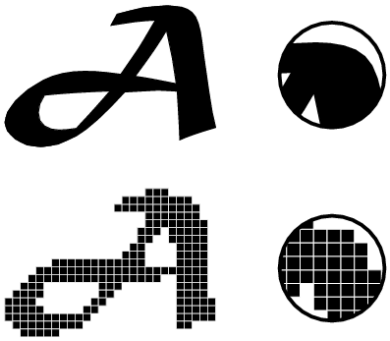
I.1 Codage numérique d'une image

Problème : Un ordinateur traite de l'information sous forme numérique binaire. Comment coder les images à l'aide de codes binaires ?

Il existe deux modes de codage d'une image numérique :

— Le mode matriciel (« bitmap ») : il repose sur le principe d'une grille de pixels.

— Le mode vectoriel (« vector ») : on décrit les propriétés mathématiques des formes de l'image.



Une image vectorielle peut être zoomée sans que cela n'altère la qualité du rendu. Ce mode est adapté aux formes et images qui ne sont pas trop complexes : logos, typographie, plans, cartes, etc.

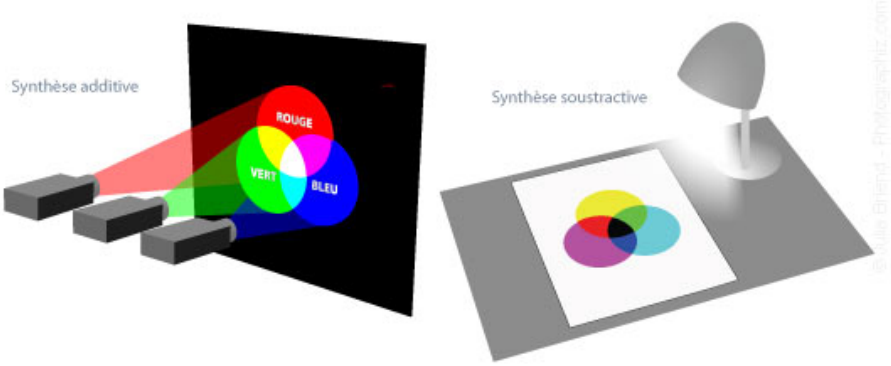
Les formats de fichiers vectoriels sont PostScript, PDF, SVG, etc.

Pour une image matricielle, le nombre de pixels utilisés par unité de longueur donne un rendu plus ou moins précis des formes de l'image. Un zoom trop important fait apparaître la pixellisation de l'image. Les formats de fichiers matriciels sont BMP, GIF, PNG, JPEG, etc.

On s'intéresse par la suite au codage des images par le mode matriciel.

I.2 Notion de calorimétrie

Problème : Comment coder les couleurs à l'aide de codes binaires ?



La synthèse additive est utilisée par les moniteurs et les projecteurs. Elle est constituée des trois lumières de base qui sont le Rouge, le Vert et le Bleu (RVB). Les couleurs secondaires sont plus claires que les primaires, c'est pour cela que leur synthèse est appelée additive.

La synthèse soustractive est utilisée dans l'imprimerie et par les imprimantes. Les couleurs primaires sont le Cyan, le Magenta, le Jaune et le Noir (CMJN). Si on mélange deux couleurs primaires, le résultat sera une couleur secondaire plus sombre qui absorbera donc plus de lumière, c'est pour cette raison que cette synthèse est nommée soustractive.

Un modèle de colorimétrie est une manière de coder des couleurs avec des nombres. Nous ne présenterons ici que le modèle RGB (Rouge Vert Bleu).

Le modèle RVB caractérise une couleur à l'aide de trois paramètres (chacun sur un octet, soit un nombre de 0 à 255) correspondant aux trois couleurs Rouge, Vert et Bleu.

$000000_{(16)}$ = (0,0,0) = noir
$FFFFFF_{(16)}$ = (255,255,255) = blanc
$FF0000_{(16)}$ = (255,0,0) = rouge

I.3 Le mode matriciel « bitmap »

Une matrice de pixels (« picture elements ») est constituée d'un ensemble de points pour former une image. Le pixel représente ainsi le plus petit élément constitutif d'une image numérique.

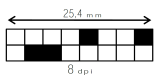
Formats limités à 256 couleurs : GIF, PCX, PGM, etc.
Formats acceptant différentes quantifications BMP, TIFF, TGA, PNG, etc.
Formats limités à 16 millions de couleurs JPEG, etc.

Les images peuvent facilement être créées et stockées dans un tableau de pixels, la lecture et l'écriture d'un pixel sont aisées.

Par contre, les fichiers peuvent être très gros (nécessité de compression).

De plus, il y a des problèmes de changement d'échelle (apparition d'effets de marches d'escalier ou de flou avec interpolation). Enfin, les dimensions de l'image doivent être prévues pour la résolution de l'interface de sortie (écran, imprimante).

La discrétisation d'une image se caractérise par sa définition, c'est-à-dire le nombre de pixels utilisés.



La résolution de l'image établit un lien entre le nombre de pixels d'une image et sa taille réelle. Elle se caractérise par un nombre de pixels par unité de longueur.

La quantification des couleurs est exprimée en bits par pixel (bpp) : 1 (deux couleurs : noir et blanc par exemple), 8 (niveaux de gris entre 0 et 255 par exemple, 256 couleurs), 16 bits, etc.

Exemple 1 : *« 800 x 600 » signifie une largeur de 800 et une hauteur de 600 pixels.*
« 600 dpi » (« ppp » : point par pouce, « dpi » : dots per inch) signifie 600 pixels par pouce (1 pouce = 25,4 mm). En connaissant le nombre de pixels d'une image et la mémoire nécessaire à l'affichage d'un pixel, il est alors possible de définir exactement la taille qu'occupe le fichier.
*Une définition du type 800 * 600 avec un codage sur 24 bits (3 octets) des couleurs, donne un fichier image de taille 1,44 Mo.*

I.4 Traitement des images matricielles

I.4.1 Afficher une image

Nous allons utiliser la bibliothèque image de matplolib.

```
import os # Import des commandes permettant de gerer les repertoires de travail
os.chdir('Repertoire de travail') # on se place dans le repertoire ou se trouve la base de donnees

import matplotlib.image as img      # bibliotheque image
import matplotlib.pyplot as plt
import numpy as np

picture = img.imread("image.png") # stocker l'image dans une variable- format de l'image png
picture = (picture * 255).astype(np.uint8) # les teintes sont sur 8 bits de 0 à 255
picture = picture[:,:, :3].copy() # (R, G, B)
plt.imshow(picture)  # afficher l'image
plt.show()
```

- Que veut dire : `print(picture.shape)` ?

  Sortie
  (825, 845, 3)

  `picture` est une matrice contenant chaque pixel de l'image.
  Le nombre de lignes de cette matrice est 825, le nombre de colonnes 845.
  Ainsi l'image contient 825 x 845 pixels.

- Que veut dire : `print(picture[10, 20])` ?

  Sortie
  [160, 112, 150]

  Le pixel de l'image situé à la 11e ligne et la 21e colonne a pour couleur RGB (160, 112, 150) (couleur parme).

- Que veut dire : `print(picture[10, 20, 2])` ?

  Sortie
  150

  La teinte bleue du pixel de l'image situé à la 11e ligne et la 21e colonne vaut 150.

I.4.2 Manipulation simple sur une image

Les opérations de transformation géométrique sont relativement simples, ne consistant qu'en un déplacement de pixels, sans modification.

I.4.3 Symétrie d'une image

Nous voulons obtenir un effet miroir, c'est à dire une symétrie par rapport à un axe vertical, le bord droit de l'image en l'occurence.
Q1. Compléter le code suivant et faire afficher l'image symétrique.

```
def symetrie(matB) :
  #création d'une matrice contenant que des zeros
    nb_lig, nb_col, nb_coul = matB.shape
    matC = np.zeros((nb_lig, nb_col, nb_coul), np.uint8)


    for i in range(nb_lig) :
        for j in range(nb_col) :
            for k in range(nb_coul) :
                matC[i, j, k] = #A compléter
    return matC
```

I.4.4 Inversion des couleurs d'une image

Inverser les couleurs d'une image ayant deux couleurs « noir » et « blanc » revient à changer les pixels noirs en blancs et inversement.

Pour le modèle RVB, cela revient à changer chaque « valeur initiale » R, V et B en « 255 - valeur initiale ».

Par exemple, si le pixel a pour couleur RGB (10, 0, 155), on lui attribue la couleur (245, 255, 100).
Q2. En s'inspirant de la fonction précédente, écrire une fonction `inversion`, la tester.

I.4.5 Passer en niveau de gris

Une des possibilités pour obtenir une image en niveau de gris à partir d'une image couleur RGB est de remplacer chaque teinte (la rouge, la verte et la bleue) par la moyenne des 3 teintes.

Par exemple si le pixel a pour couleur (100, 10, 10), on lui attribue la couleur (40, 40, 40).
Q3. Compléter la fonction `niveau_de_gris` et la tester.

```
def niveau_de_gris(matB) :
    nb_lig, nb_col, nb_coul = matB.shape
    matC = np.zeros((nb_lig, nb_col, nb_coul), np.uint8)
    for i in range(nb_lig) :
        for j in range(nb_col) :
            #conversion de int sur 8 bits en int pour faire des operations
            r, g, b = int(matB[i, j, 0]), int(matB[i, j, 1]), int(matB[i, j, 2])
            gris = #A completer
            for k in range(nb_coul) :
                matC[i, j, k] = #A completer
    return matC
```

La CIE (Commission Internationale de l'Eclairage) normalise la répartition pour obtenir un niveau de gris correct. . Et dans sa norme 709, Il est indiqué que pour les images naturelles les poids respectifs doivent être 0.2125 * R + 0.7154 * G + 0.0721 * B. Nous allons donc utiliser cette répartition dans notre code.
Q4. Ecrire la fonction `niveau_de_gris_norme` et la tester.

II Photos à la plage !

Récupérer les fichiers du TP8 sur `Prepabellevue.org\PCSI1\Informatique\TP` ou dans l'atelier `ConsoleEleve\TPInfo1A\Bureau\TP8`.

Les décompresser puis placer dans un nouveau répertoire `TP8` le fichier portant votre nom ainsi que le fichier `Plage.png`.

Convertir les images JPG en PNG avec l'application paint.

II.1 Exercice 1

Q5. Charger dans une variable `plage` la photo `Plage.png` puis l' afficher.
Q6. Tester et comprendre les instructions suivantes :

```
plt.figure(2)
plt.imshow(plage[600:, :1000, :])
plt.show()
```

Q7. Afficher la plage en niveau de gris normalisé
Q8. On utilise la fonction `deepcopy` du module `copy`.
Tester et comprendre les instructions suivantes :

```
from copy import deepcopy

nb_lig, nb_col, nb_coul = plage.shape
plage3 = deepcopy(plage)
for i in range(nb_lig) :
    for j in range(nb_col) :
        plage3[i, j, 1] = 0
        plage3[i, j, 2] = 0

plt.figure(4)
plt.imshow(plage3)
plt.show()
```

II.2 Exercice 2

Le but de cet exercice est d'incorporer votre **portrait** à l'image `Plage.png`.

Il faudra pour ceci supprimer le décor, c'est-à-dire le tableau de la classe.

Q9. Récupérer votre photo sous `Python`, l'afficher, stocker sa taille `(nb_lig_po, nb_col_po)`.
Il peut être judicieux de modifier la taille si celle-ci est plus grande que celle de la plage...

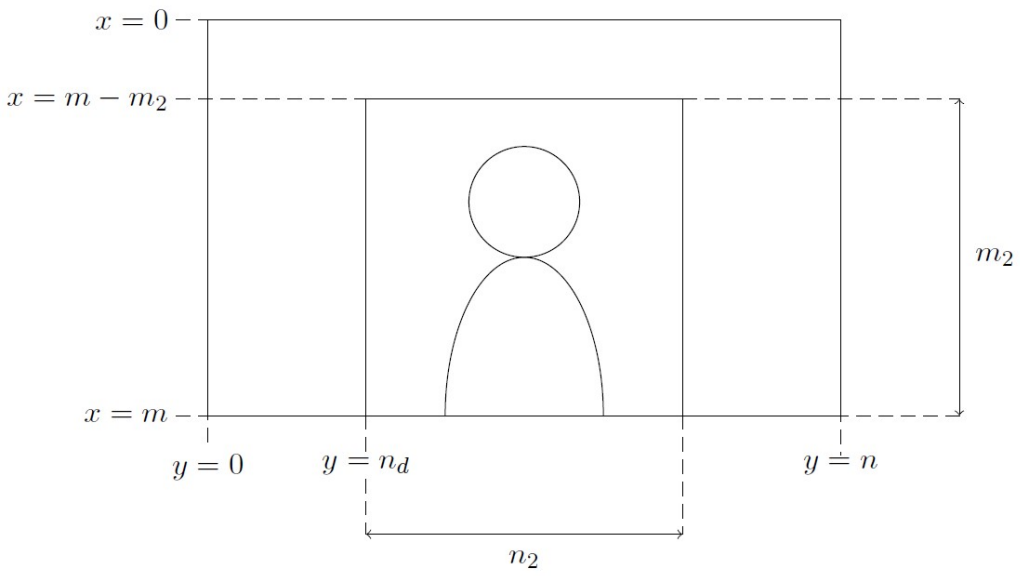
Q10. Créer une fonction `tableau` qui reçoit un triplet d'entiers (r, g, b) et qui renvoie `True` si la couleur correspond à celle du tableau et `False` sinon. *Avec Paint, on peut regarder le codage RGB de la couleur du tableau.*

On pourra tester cette fonction en remplaçant les pixels représentant le tableau sur le portrait par des pixels noirs.

En jouant sur les conditions on peut améliorer le résultat.

Q11. Remplacer les pixels de la matrice contenant la plage par les pixels de votre portrait, en omettant ceux qui représentent le tableau.

On pourra s'aider du schéma ci-dessous.



Afficher le résultat.

III Traitement de l'image (suite)

III.1 Détection de contours

La reconnaissance de formes dans une image est une composante importante de l'analyse d'images. Elle se décompose en plusieurs étapes qui consistent à extraire les contours des objets dans l'image afin de les reconnaitre ou d'en détecter le mouvement. La première de ces étapes est la mise en évidence des contours des objets dans l'image. C'est cette étape que nous allons aborder très succintement.

Un contour définit la limite d'un objet dans une image. Cette limite est caractérisée par un changement dans l'image : un changement de couleur ou de contraste. Ce changement se traduit dans la valeur des pixels qui sont localisés de part et d'autre de la limite. Nous sommes donc à la recherche d'un moyen de détecter et de localiser un changement.

Considérons un pixel p(i,j) dans une image couleur. Ce pixel est-il semblable, de même couleur, que ses voisins ? Si non, quelle est la différence de couleur entre lui et ses voisins ? Est-elle grande, ce qui signifierait qu'il est situé à la limite d'un objet ? Que signifie une "grande" ou une "petite" différence ? Comment la mesurer pratiquement ? C'est ce que nous allons essayer de traduire en algorithme.

Le principe est de récupérer la valeur de chaque pixel avoisinant pour chaque pixel de l'image, Puis de mesurer la différence, la "distance", entre notre pixel de référence et ses voisins en utilisant une fonction de norme standard.

Après avoir calculé la distance entre mon pixel courant et ses voisins, je décide si ce pixel est sur un contour ou pas à l'aide d'un seuillage. S'il est inférieur au seuil, c'est à dire pas très "distant" de ses voisins, je décide qu'il n'est pas élément d'un contour et je trace le en blanc, sinon, je le trace en noir, comme un contour.

```
def contour(matB) :
    nb_lig, nb_col, nb_coul = matB.shape
    matC = np.zeros((nb_lig, nb_col, nb_coul), np.uint8)
    for i in range(2, nb_lig - 2) :
        for j in range(2, nb_col - 2) :
            p1 = int(matB[i-2, j, 0])
            p2 = int(matB[i, j-2, 0])
            p3 = int(matB[i+2, j, 0])
            p4 = int(matB[i, j+2, 0])
            norme = int(np.sqrt((p1-p2) **2 +(p2-p4)**2)/2)
            seuil = 150
            if norme < seuil : p = 255
            else : p = 0
            for k in range(nb_coul) :
                matC[i, j, k] = p
    return matC
```

III.2 Filtres

Le filtrage consiste à appliquer une transformation (appelée filtre) à tout ou partie d'une image numérique en appliquant un opérateur. On distingue généralement les types de filtres suivants :

III.2.1 Définition d'un filtre

Un filtre est une transformation mathématique (appelée produit de convolution) permettant, pour chaque pixel de la zone à laquelle il s'applique, de modifier sa valeur en fonction des valeurs des pixels avoisinants, affectées de coefficients.

Le filtre est représenté par un tableau (matrice), caractérisé par ses dimensions et ses coefficients, dont le centre correspond au pixel concerné. Les coefficients du tableau déterminent les propriétés du filtre.

Prenons le cas d'une matrice 3x3, donc un tableau de neuf nombres représentés ici par a, b, c, ..., i qui sont les paramètres du filtre et que l'on peut donc définir (cf. figure).

Nous allons travailler sur une image en niveau de gris (256 valeurs). Dans l'exemple proposé le pixel de couleur rouge a un niveau de gris égal à 115. En quoi consiste l'application du filtre à ce pixel ?

La valeur 115 est remplacée par la somme de 9 produits :
45.a + 60.b + 81.c + 82.d + 115.e + 133.f + 130.g + 154.h + 147.i
Afin de normaliser ce résultat, bien souvent, on divisera ce résultat par la somme des coefficients du filtre, soit a+b+c+d+e+f+g+h+i.

	0	1	2	3	4	5
0	48	45	60	81	83	65
1	58	82	115	133	104	55
2	99	130	154	147	96	37
3	136	160	163	138	86	39
4	156	158	157	139	89	42
5	154	154	156	145	98	45

Extrait de l'image en niveaux de gris. Lorsqu'on applique le filtre au pixel encadré en rouge, le calcul fait intervenir ce même pixel et ses 8 voisins (zone verte). La zone verte a la même taille que la matrice du filtre

a	b	c
d	e	f
g	h	i

Matrice du filtre 3x3; les valeurs a, b, c, ..., i peuvent être positives, négatives, ou nulles, entières ou non

Qu'en est-il pour un pixel appartenant à la bordure de l'image, car il n'a pas tous ses voisins ?

La solution la plus simple est d'ignorer ces pixels : on balaye l'image de la deuxième ligne à l'avant-dernière, et pour chaque ligne, de la deuxième colonne à l'avant-dernière.

Que se passe-t-il si le résultat du calcul sort de l'intervalle [0 ;255] correspondant aux valeurs extrêmes des composantes RVB ?

— Solution 1 : le résultat (différent pour chaque pixel) est divisé par la somme des coefficients du filtre.

— Solution 2 : On écrête tout résultat supérieur à 255 ou inférieur à 0

— Solution 3 : On ramène l'ensemble des résultats dans l'intervalle [0,255] en appliquant une proportionnalité. Cela nécessite de faire une recherche de la valeur min et de la valeur max dans toute l'image.

-1	-1	-1
-1	9	-1
-1	-1	-1

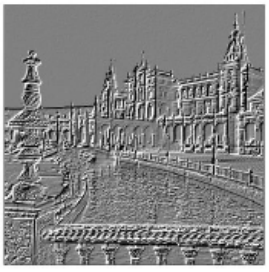
filtre réhausseur de contraste

-1/6	-2/3	-1/6
-2/3	13/3	-2/3
-1/6	-2/3	-1/6

autre filtre réhausseur

-2	-1	0
-1	0	1
0	1	2

filtre d'embossage (effet de relief)



IV Programmation graphique d'un jeu de rugby

IV.1 Conception d'une application sportive

L'application « Rugby Manager » est un jeu destiné aux smartphones et aux tablettes. Il permet de créer une équipe, de l'entraîner et de jouer avec elle. Il est possible de joueur seul (contre l'intelligence artificielle du logiciel) ou en mode multijoueurs. Enfin, l'application est utilisable en ligne ou en Bluetooth.

L'objectif aujourd'hui est de programmer un certain nombre de fonction qui seront utilisées pour créer cette application.

Partie graphique du mode jeu

Dans cette partie nous souhaitons tout d'abord représenter la partie graphique du mode jeu, c'est-à-dire :

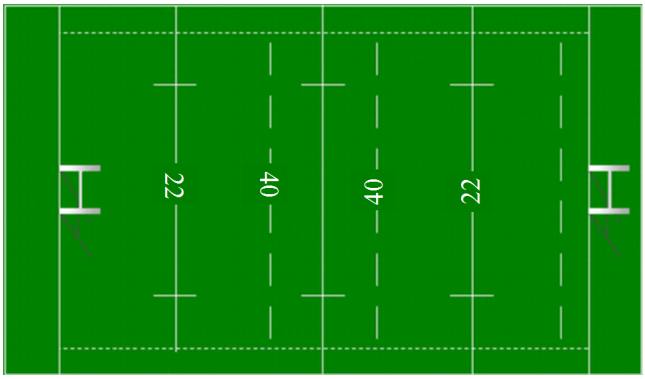
— un terrain ;

— deux équipes ;

— un ballon.

nous allons réaliser une étude partielle de ces fonctionnalités.

On souhaite afficher l'image du terrain de rugby suivant, enregistrée dans le répertoire "C : CCP" sous le nom *stade.bmp* :



Objectif

Les compétences évaluées dans ce TP portent sur le traitement des images. Il s'agit d'utiliser la bibliothèque d'image décrite en annexe pour compléter ou modifier des fonctions d'affichages de l'image du terrain et des pixels des maillots des joueurs. Dans un second temps il s'agit de mettre en place un "floutage" d'une image pour la mettre en arrière plan dans le mode statistique.

Q12. Créer un programme qui permet :

— de se placer dans le dossier où se trouve l'image ;

— d'ouvrir l'image et de la stocker dans une variable *image_terrain* ;

— d'afficher l'image en arrière plan.

il est nécessaire ensuite nécessaire de connaître les dimensions de l'image qui peut être connu grâce aux deux fonctions suivantes

```
1 xmax= picture.shape[1]     #taille du tableau
2 ymax= picture.shape[0]    #taille du tableau
3 print ("taille de l'image : %d x %d" % (xmax,ymax))
```

Pour obtenir la couleur du pixel il suffit donc de choisir le ou les bons indices du tableau.

Q13. Donner les instructions qui permettent de faire cette opération et de stocker le résultat dans deux variables *dim_long* et *dim_larg* respectivement dimension en longueur et en largeur. Afficher ensuite le résultat sous la forme ("longueur x largeur").

On propose dans un premier temps de représenter les joueurs par 4 pixels disposés en carré ayant la couleur du maillot de l'équipe. Le choix des couleurs de maillots doit être laissé libre à l'utilisateur. Cependant, le vert correspondant à la couleur du terrain et le blanc correspondant à la couleur du ballon et des lignes ne pourront pas être utilisés.
Précisons que les lignes font 2 pixels de large, sur une image qui en fait 620, et qu'il est facile de se positionner au centre du terrain. Les lignes pointillées des 40 sont elles à 50 pixels du centre.

Q14. Créer une fonction *coul()* qui permet de connaître la couleur exacte du terrain et de la stocker dans une variable *coul_ter*. L'argument d'entrée de la fonction est la variable *image*, la sortie est la variable *coul_ter*. Attention : on choisira comme pixel de référence un des pixels le plus proche du centre du terrain.

Q15. Quel est le type de la variable *coul_ter*.
Sur combien de bits mémoire est codé un pixel ?

Partie graphique du mode statistiques

IV.1.1 Floutage de l'image

Objectif
Dans cette partie, nous intéressons d'abord au traitement d'image évolué : la mise en place un "floutage" d'une image pour l'arrière plan du mode statistique. Nous étudierons par la suite des fonctions simples de statistiques qui permettent d'alimenter ce mode du jeu.

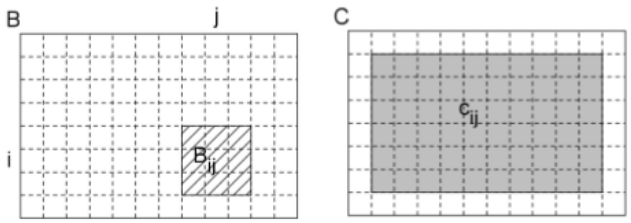
On s'intéresse donc ici à un autre mode du jeu vidéo : le mode statistique. Dans ce mode, une image de jeu floue va être mise en arrière plan. Nous allons donc étudier les fonctions permettant d'effectuer ce floutage.
Pour réaliser un floutage par moyenne simple sur la matrice de pixels, il faut lui appliquer un filtre, que l'on appelle également un masque. Afin de bien comprendre ce principe, nous proposons d'étudier un exemple de filtrage : On considère les matrices

$$A = \begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix}$$

et

$$B = \begin{pmatrix} 5 & 6 & 7 & 8 & 9 & 10 \\ -5 & -6 & -7 & -8 & -9 & -10 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 3 & 3 & 4 & 4 \\ 0 & 0 & 1 & 3 & 3 & 3 \end{pmatrix}.$$

Pour chaque élément *bij* de B, on considère la matrice 3×3 *Bij* qui l'entoure, on calcule le produit de convolution de A par *Bij* **(multiplication classique *A*Bij*)** et on note *cij* la somme des coefficients de la matrice produit obtenue (on pourra utiliser la fonction *np.sum* sur une liste).
Si *bij* est un élément en bordure de B, on posera *cij* = *bij*. On forme ainsi une nouvelle matrice C dont les éléments intérieurs sont les *cij* (et les éléments au bord sont les *bij*).



On dit qu'on a filtré la matrice B par la matrice A, ou qu'on a appliqué le masque (filtre) A sur l'image B.

Q16. Compléter la fonction *filtrer1(filtreA,matB)* qui prend en argument une matrice carrée *filtreA* de dimension *taille*×*taille* (taille est un entier impaire de dimension 3) et une matrice quelconque *matB* de dimensions supérieures, et qui renvoie la matrice C. On remarquera que si *taille* > 3, la bordure devra être plus épaisse.

On souhaite maintenant appliquer le filtre A à une matrice de pixels B, c'est à dire aux 3 tableaux B[:, :,0], B[:, :,1], B[:, :,2] et enregistrer le résultat dans une matrice C de même format que B.

Q17. Créer une fonction *filtrer(filtreA,matB)* qui prend en argument une matrice carrée *filtreA* de dimension *taille*×*taille* et un tableau numpy *matB* de dimensions *np*3,et qui renvoie le tableau C de dimensions identiques. Vous pourrez calculer successivement C[:, :,0], C[:, :,1], C[:, :,2] à l'aide d'une boucle.

Ces matrices réalisent un floutage par moyenne simple (coefficients tous égaux, de somme 1). Plus la taille du filtre est grande, plus le flou sera fort. On peut améliorer cette technique en utilisant un flou gaussien. Son principe est de calculer une moyenne pondérée en accordant plus de poids au pixel central et en diminuant le poids des pixels périphériques.
La matrice servant de filtre est calculée selon le modèle d'une courbe de Gauss (courbe en cloche) à 2 dimensions :
La fonction de Laplace-Gauss est $G(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}$. A deux dimensions on utilise $G(x,y) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2+y^2}{2\sigma^2}}$.

Plus l'écart type σ est grand, plus l'image sera floutée. En pratique, σ et la taille (impaire) du filtre étant fixés, on calcule chaque élément de la matrice filtre par la formule $(x,y) = k.e^{-\frac{x^2+y^2}{2\sigma^2}}$,

- où x et y sont le nombre de lignes et de colonnes qui séparent cet élément du centre,
- et k un coefficient constant tel que la somme de tous les éléments (de type float) ainsi calculés soit 1.

Par exemple, pour *taille=5* et $\sigma = 0.9$, le filtre est proche de :

Q18. Écrire la fonction la fonction *matriceFlouGaussien(taille,sigma)* qui prend en argument la *taille* (impaire) de la matrice de floutage, *sigma* l'écart type de déviation standard et qui retourne la matrice filtre correspondant au niveau gaussien.

Q19. Écrire ensuite la fonction *FloutageGaussien(tabPix,taille,sigma)* qui utilise la fonction *matriceFlouGaussien(taille,sigma)* et *filtrer(filtreA,matB)* et qui renvoie la matrice *tabPix* qui a été floutée grâce au filtre défini dans la fonction *matriceFlouGaussien*.

IV.1.2 Détection d'un joueur

Objectif
Dans cette partie, nous intéressons d'abord au traitement d'image évolué : la rotation de l'image et la détection de joueur sur l'image.

Q20. Écrire la fonction qui permet de faire une rotation de $-\pi/2$ de notre image.

Pour permettre la détection d'un joueur sur le terrain, il faut définir les contours principaux des objets présents sur le terrain.

Q21. Écrire la fonction qui permet de passer en niveau de gris notre image

Q22. Écrire la fonction qui permet de définir les contours.