TP : Suivi d'image

Dans ce projet, nous allons utiliser des algorithmes de suivi d'image pour, dans un premier temps, monitorer un élément dynamique dans une vidéo. Puis, nous exploiterons ces algorithmes pour rectifier l'image et simuler un objet statique.

I Affichage de l'image

1) Récupérez la séquence d'images à l'adresse suivante: https://github.com/gaysimon/TP_tracking. Vous écrirez ensuite un programme pour charger et afficher la première frame (*img1.png*) à l'écran. (note: les images, bien qu'en noir et blanc, sont bien des images RGB, les canaux R, G et B ayant les mêmes valeurs). Vous écrirez pour cela une fonction avec la signature suivante, qui charge une image d'après son numéro de frame, pour la charger dans une variable globale *image*.

```
void load( int i )
```

Astuce: *- charger une image et la convertir en matrice en Java*

```
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
//////////////////////
BufferedImage image;
try {
        image = ImageIO.read(new File("/path/to/the/image.png"));
} catch (IOException e) {System.out.print(e);}
//////////////////////
map=new int[image.getWidth()][image.getHeight()][3];
for (int i=0;i<image.getWidth();i++){
        for (int j=0;j<image.getHeight();j++){
                map[i][j][0]=(image.getRGB(i, j)>> 16) & 0x000000FF;
                map[i][j][1]=(image.getRGB(i, j)>>  8) & 0x000000FF;
                map[i][j][2]=(image.getRGB(i, j)     ) & 0x000000FF;
        }
}
```

- charger, lire, écrire et afficher une image en Python

```
import matplotlib.image as mpimg
import numpy as np
import matplotlib.pyplot as plt
######################
img = mpimg.imread("my/image.png")
if img.dtype == np.float32: # convert values to [0;255] integer
    img = (img * 255).astype(np.uint8)
######################
r=img[50,50,0]
g=img[50,50,1]
b=img[50,50,2]
######################
img[50,50,0]=r
img[50,50,1]=g
img[50,50,2]=b
######################
plt.imshow(img)
plt.show()                    # show() for blocking display, draw() for non-blocking
```

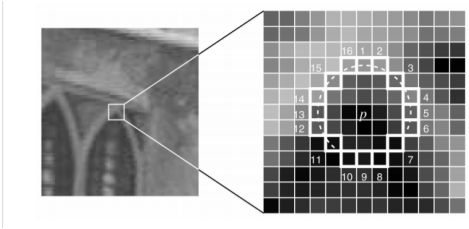
II Détection de points d'intérêt

Le suivi d'image consiste dans un premier temps à trouver des points faciles à suivre. Un bon point d'intérêt est un élément de l'image dont l'intensité lumineuse change si on le déplace dans plusieurs directions. Les coins sont donc d'excellent points d'intérêts car on peut détecter un déplacement dans toutes les directions.

1) *Recherche bibliographique* : dans votre rapport, vous listerez différentes façons de détecter des points d'intérêt, en indiquant leurs avantages et inconvéniants.

Dans ce projet, nous allons utiliser l'un des détecteurs les plus simples et rapides : FAST(Features from Accelerated Segment Test). Cette méthode consiste à tester seulement un petit nombre de pixels autours d'un point considéré, plutôt qu'un pattern 2D autours de ce point, et d'appliquer un critère déterminant si le pixel central peut être utilisé comme point d'intérêt.

Dans la version la plus utilisée, le détecteur FAST utilise 16 pixels formant un cercle autours du point. Le critère est le suivant : on récupère l'intensité du point p, notée I_p, et on défini une valeur seuil t qui détermine à partir de quand un pixel k est considéré comme plus clair ($I_k > I_p + t$) ou plus foncé ($I_k < I_p - t$) que p. Si le nombre de pixel plus clairs ou plus foncés que p est suppérieur à 12, alors p est bien un coin.



détecteur de coins FAST : pour chaque point p, l'algorithme va tester 16 pixels autours de p.

2) Préparez la liste des offsets (ci-dessous) correspondant à un cercle de pixels, que vous enregistrerez dans une matrice *circle* de taille 16x2.

		(-1;3)	(0;3)	(1;3)		
	(-2;2)				(2,2)	
(-3,1)						(3,1)
(-3,0)			p			(3,0)
(-3,-1)						(3,-1)
	(-2;-2)				(2,-2)	
		(-1;-3)	(0;-3)	(1;-3)		

3) Implantez l'algorithme FAST, décrit ci-dessous. Vous utiliserez un seuil de 5. La fonction doit avoir la signature undiquée.

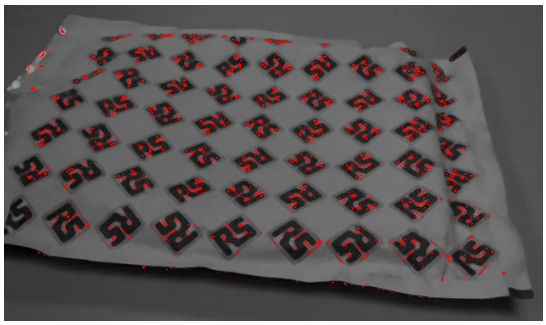
```
boolean fastCorner(int x, int y){
      counter_dark=0
      counter_bright=0

      for each line i in circle do

        if ( image[x][y] <  image[ x+circle[i][0] ][ y+circle[i][1] ]  - 5  do                    // circle's offsets
            counter_bright++
         end if
         if ( image[x][y] >  image[ x+circle[i][0] ][ y+circle[i][1] ]  + 5  do
            counter_bright++
         end if

      end for

      return counter_dark > 12 OR counter_bright > 12
}
```

4) Appliquez l'algorithme FAST sur l'image entière (attention aux marges!), enregistrez les points dans une liste et affichez-les sur l'image.

Astuce *: afficher des points sur une image en python :*

```
plt.plot(j, i, "ro", markersize=2)          # 'ro' indicates a red point
```

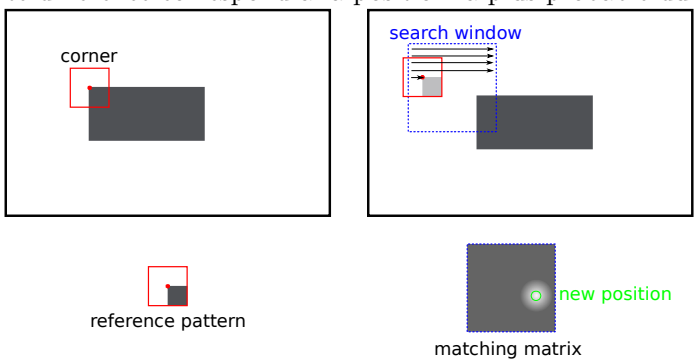
Vous trouverez ici comment générer des points de différentes formes et couleur :
https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.plot.html



Les points rouges sont des coins détectés par FAST.

III Suivi d'image

Nous allons utiliser un algorithme de suivi pour monitorer le mouvement d'un point particulier dans une vidéo. Le suivi de mouvement, ou motion tracking, consiste à définir un point d'une première frame, puis à trouver la position de ce point dans les frames suivantes. Nous allons utiliser une approche simple : le pattern matching. Cette approche consiste à comparer un pattern d'image autours du point à chercher avec les patterns obtenus en différentes positions d'une fenêtre de recherche dans une autre image. La position avec la plus petite différence correspond à la position la plus probable du point recherché.



À gauche, le pattern de référence est défini autours du point d'intérêt. À droite, sur la frame suivante, on compare le pattern de référence en différentes positions de la fenêtre de recherche. LA position avec la plus faible différence correspond à la position du point d'intérêt dans cette nouvelle frame.

Dans cette partie du TP, le but est de permettre à un utilisateur de sélectionner manuellement un point de l'image et d'enregistrer les déplacements de ce point.

1) Ajoutez un *listener* sur votre image pour récupérer la position d'un clic de la souris. Vous chercherez ensuite le point d'intérêt le plus proche de ce point, dans la liste définie précédemment.

***Astuce** : la fonction racine carré est strictement croissante. Vous pouvez ainsi comparer les distances d^2 au lieu de d, en évitant ainsi l'utilisation de la fonction sqrt pour calculer les distances.*

***Astuce** : ce bout de code python permet d'ajouter à un afficheur une fonction qui récupère les coordonnées du pointeur de la souris, les affichent, et ferment l'afficheur :*

```
# This function gets the mouse event, displays coordinates and close the window
def onclick(event):
    print(event.xdata, event.ydata)
    plt.close()

# Attach the listener function to the window
fig,ax = plt.subplots()
fig.canvas.mpl_connect('button_press_event', onclick)

# Display the window
plt.imshow(img)
plt.show()
```

2) Définissez un point du suivi comme une structure contenant, avec la méthode de votre choix (liste de tableaux ou programmation objet) :
- les coordonnées initiales Pi = (xi, yi)
- les coordonnées de la position courante C = (xc, yc)
- un pattern constitué d'une matrice de taille 15 x 15 contenant *image*[(x_i-7;x_i+7)][(y_i-7;y_i+7)] de la première frame.

3) Implantez la fonction de suivi décrite ci-dessous, avec une fenêtre de recherche de 40x40 pixels.
Note: attention aux marges !

```
current position of POI is C=(xc, yc)
size of pattern is (px, py)
size of search window is (wx, wy)

initialize a matrix match of size (wx, wy)

// compute matching matrix
for each position X=(x,y) of search window, x in [-wx/2,wx/2[ , y in [-wy/2, wy/2[ do

        float matching=0

        // compute matching at position ( cx+x, cy+y) of the image
        for each pixel I=(i,j) in pattern, i in [0, px[ , j in [0, py[ do
            matching+= | pattern[i][j] – image[ xc + x + i-px/2] [ yc + y + j-py/2] |
        end for

        // set the matching value in match
        match[x+wx/2][y+wy/2] = matching

    end for

    // get the best matching (lowest value)
    float min=INFINITY
    Xmin=(0,0)

     for each position X=(x,y) in match do
        if (match[x][y] < min) do
            min=match[x][y]
            Xmin=X
        end if
     end for

     // update current position
     C=Xmin
```

4) Modifiez le programme principal pour exécuter les étapes suivantes :

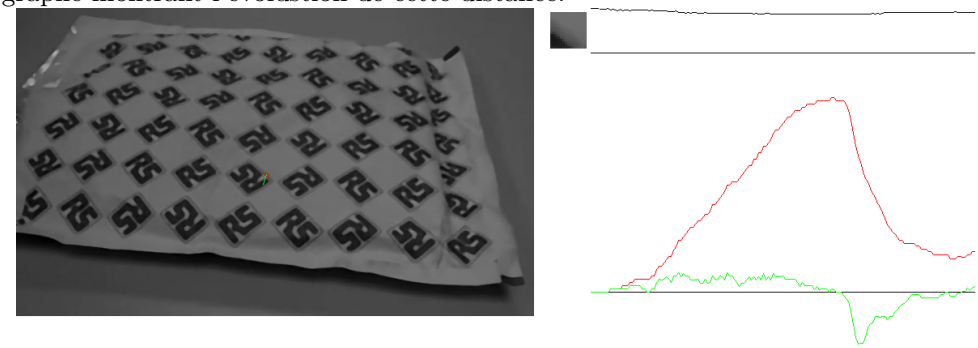
-load image 1
-detect corners
-wait for user selection
-create a tracking point on the selected corner
- for i in [2, 158] do
    - load image i
    - track position

Sur l'image affichée, vous dessinerez la position courante du point, ainsi qu'un trait reliant la position initiale et la position courante.

***Astuce** : fonctions pour dessiner un point et un trait en python :*

```
plt.plot(j, i, "ro", markersize=2)    # point
plt.plot([j1,j2], [i1,i2])            #line
```

5) Nous voulons enregistrer le mouvement du point afin de surveiller l'évolution de l'objet. Enregistrez la distance de déplacement du point et enregistrez les valeurs successives dans un tableau. Puis, vous afficherez le graphe montrant l'évolustion de cette distance.



À gauche, le point suivi (rouge) et le déplacement (trait vert). À droite, l'enregistrement de l'évolution de la distance (en vert) et de l'accélération (en rouge).

IV Stabilisation de l'image

L'objet nécessite une intervention chirurgicale d'urgence, mais celui-ci ne doit pas cesser de fonctionner. Afin d'assister le chirurgien, nous allons développer un algorithme pour stabiliser l'image et donner l'illusion d'une objet statique. Comme nous allons applique l'algorithme de suivi à tous les points d'intérêt, il va falloir réduire leur nombre.

1) Commencez par dupliquer votre projet. Modifiez le nouveau pour réduire le nombre de points collectés par FAST. Pour cela, nous allons simplement éliminer les points trop proches les uns des autres, avec l'algorithme suivant :

```
for i in [0, list.length[ do
    for j in [i+1, list.length[ do
        if dist(list[i], list[j]) < 10 do          // min distance is 10px
            list.remove(j)
            j--                                    // don't forget do decrement
        end if
    end for
end for
```

2) Créez un point de suivi pour chaque point d'intérêt restant. Affichez les points de suivi sur l'image. Vous devriez maintenant pouvoir mesurer le mouvement de l'objet en chaque point.
***Astuce** : si l'exécution est trop lente, vous pouvez réduire la taille de la fenêtre de recherche.*

3) Nous allons maintenant calculer la transformation en chaque pixel de l'image en se basant sur les points de suivi autours du pixel. Le mouvement m_p en p est donné par :

$$m_p = \frac{\sum_{k \in tracklist} w_k \times m_k}{\sum_{k \in tracklist} w_k}$$

Nous proposons de définir le poids w_k avec la fonction *wk = 50-d(p,k) if d(p,k)<50, wk=0 else*. Appliquez l'algorithme suivant pour calculer le mouvement en chaque pixel :

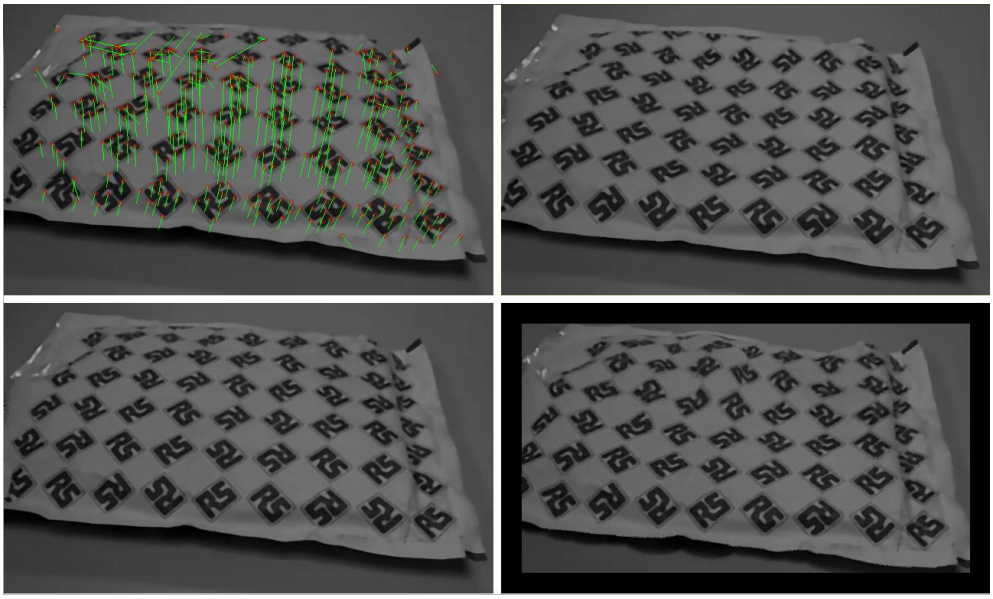
```
initialize distX a matrix of same size than image
initialize distY a matrix of same size than image
initialize weightMap a matrix of same size than image

// get the weighted sum of displacement on X and Y axis on each pixel
for k in trackList do

    dx = k.xc – k.xi                    // displacement of tracking point
    dy = k.yc – k.yi

    for i in [-50;50] do
       for j in [-50, 50] do

            if  i*i+j*j <2500 do                    // 2500=50x50
              d=sqrt(i*i+j*j)
                    distX[ k.xc+i ][ k.yc+j ] += dx * (50-d)
                    distY[ k.xc+i ][ k.yc+j ] += dy * (50-d)
                    weightMap[ k.xc+i ][ k.yc+j ] += 50-d
           end if
     end for
  end for
end for

// divide by sum of weights
for I in [0;image.width] do
   for j in [0;image.height] do
          if weightMap[i][j]>0 do
            distX[i][j] = distX[i][j] / weightMap[i][j]
               distY[i][j] = distY[i][j] / weightMap[i][j]
            end if
      end for
  end for
```

Ainsi, les matrices distX et distY contiennent le mouvement moyen en chaque pixel.

4) Nous pouvons dès à présent rectifier l'image :

```
image2[i][j] = image[ i + distX[i][j] ][ j + distY[i][j] ]
```

Écrivez la fonction qui génère l'image rectifiée et affichez-la.



En bas à gauche, la frame courante. En haut à gauche, les points de suivi. En haut à droite, la première frame. En bas à droite, l'image rectifiée.