02_MEC

November 13, 2016

```
In [5]: from IPython.display import SVG, display, Image
        import matplotlib.pyplot as pl
        from pylab import *
        from scipy.ndimage import convolve
        import cv2
        %pylab inline
        rcParams['figure.figsize'] = (10,4)
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
/usr/local/lib/python2.7/dist-packages/IPython/core/magics/pylab.py:161: UserWarnir
`%matplotlib` prevents importing * from pylab and numpy
  "\n`%matplotlib` prevents importing * from pylab and numpy"
```

```
In [2]: from IPython.display import display, Math, Latex
```

0.1 Mineur Robotique

0.2 Perception pour la robotique

0.2.1 Cours 2 : traitement d'images et mise en correspondance

Aurélien Plyer
aplyer.esiea@gmail.com

0.3 Commencons par un peut de traitement d'images

```
In [3]: def imagesc(I):
            '''
                Petite fonction pour afficher une image en niveaux de gris
            '''
            _ = imshow(I,'gray')
            axis('off')

        def spill(filename):
            '''
                Populate the workspace from saved data
```

```
    '''
    f = np.load(filename)
    for key, val in f.iteritems():
        globals()[key] = val
    f.close()
```

0.3.1 Qu'est-ce qu'une image?

```
In [15]: import cv2 # On importe le module OpenCV

         I = cv2.imread('img/onera_batn.jpg', cv2.IMREAD_GRAYSCALE)
         imagesc(I)
```

```
In [5]: print('type de I : '+type(I).__name__)

type de I : ndarray


In [6]: print(I)

[[ 45  35  21 ...,  95 116 138]
 [ 40  39  37 ..., 114 126 141]
 [ 38  48  60 ..., 141 145 151]
 ...,
 [215 215 215 ..., 215 215 215]
 [215 215 215 ..., 215 214 214]
 [215 215 215 ..., 214 214 213]]


In [7]: print(I.shape)
```

```
(1000, 2000)
```

0.3.2 Du coup on peut manipuler les images comme on manipule des matrices numpy

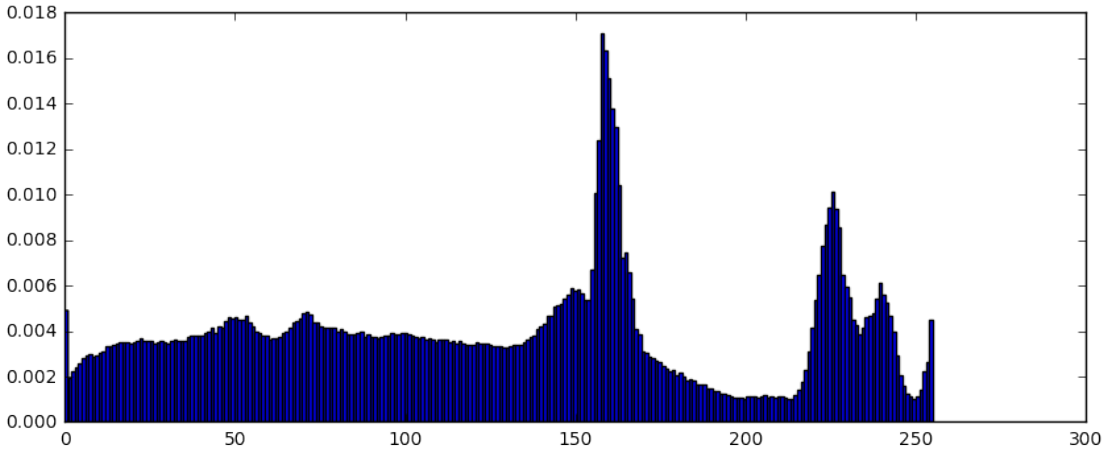
```
In [16]: imagesc(I[::4, :])
```

```
In [17]: imagesc(I[::-2,::-2])
```

0.4 Et on fait quoi avec une image?

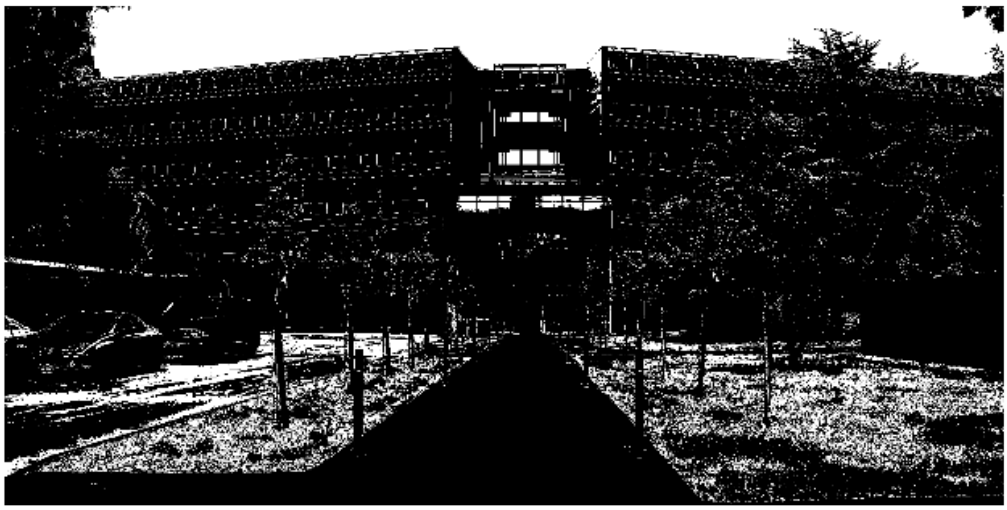
on calcul un histogramme :

```
In [18]: from scipy import histogram
         n, bins, patches = hist(I.ravel(), 256, normed=True, facecolor='blue')
```



un pic entre 150 et 180... visualisons cela avec un masque de sélection :

```
In [19]: Mask = np.zeros(I.shape)
         Mask[(150 < I)*(I < 180)] = 1
         imagesc(Mask)
```



Ok c'est donc le niveau de gris du ciel et de la pelouse.

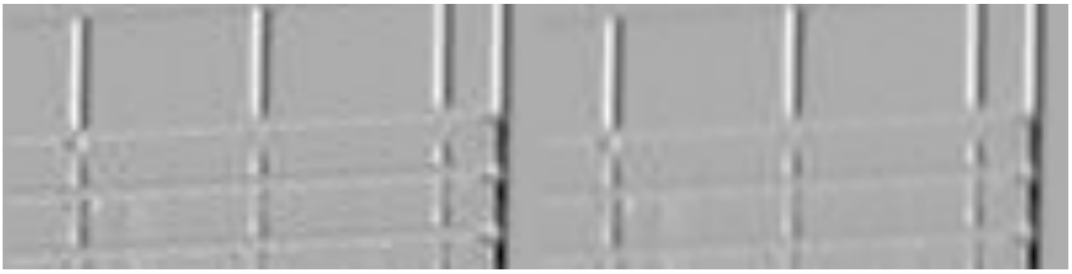
0.5 Comment on calcul les gradients de I?

On va se concentrer sur un petit zoom de notre batiment

```
In [20]: Crop = I[200:250,800:900].astype(np.float32)
         imagesc(Crop)
```

Pour cela on a le choix de plusieurs oppérateurs : - opérateur backward $\begin{bmatrix} -1 & 1 & 0 \end{bmatrix}$ - opérateur forward $\begin{bmatrix} 0 & -1 & 1 \end{bmatrix}$ - opérateur centré $\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$ - Prewitt $\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$ - Sobel $\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$

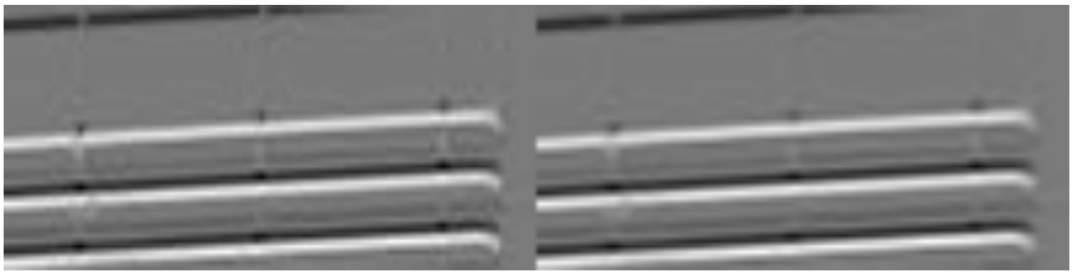
```
In [27]: from scipy.ndimage import convolve
         Icx = convolve(Crop, np.array([[-1,0,1]]))
         Isx = convolve(Crop, np.array([[-1,0,1],[-2,0,2],[-1,0,1]])/4.)
         imagesc(np.concatenate([Icx, Isx],axis = 1))
```



0.6 En x c'est toujours facile comment on fait en y?

```
In [28]: from scipy.ndimage import convolve
         Icy = convolve(Crop, np.array([[-1,0,1]]).T )
```

```
Isy = convolve(Crop, np.array([[-1,0,1],[-2,0,2],[-1,0,1]]).T/4. )
imagesc(np.concatenate([Icy, Isy],axis = 1))
```



Bon ok, j'ai rien dit...

0.6.1 Détectons les contours

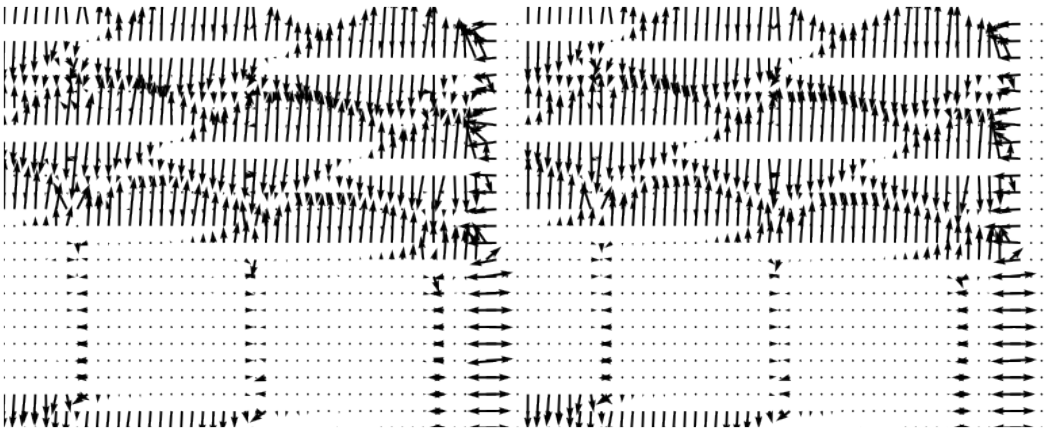
```
In [35]: seuil = 100
```

```
N = np.sqrt(Icy**2+Icx**2)
edges = np.zeros(N.shape)
edges[N>seuil] = 1
imagesc(edges)
```

0.6.2 Et si on faisait mieux pour le seuillage?

```
In [40]: seuil_bas, seuil_haut = 40, 80

         N = np.sqrt(Icy**2+Icx**2)
         edges = np.zeros(N.shape)
         bas   = edges.copy()
         edges[N>seuil_haut] = 1
         bas[N > seuil_bas] = 1

In [41]: tmp = convolve(edges, np.ones([3,3]))
         edges[tmp * bas > 0] = 1

         imagesc(edges)
```

```
In [23]: cols, rows = Crop.shape[1], Crop.shape[0]
         X, Y = meshgrid(range(cols),range(rows))
         X = np.concatenate([X,X], axis = 1)
         Y = np.concatenate([Y,Y], axis = 1)
         Iy = np.concatenate([Icy,Isy], axis = 1)
         _= quiver(Ix[::2,::2],Iy[::2,::2])
         _ = axis('off')
```



0.6.3 Détection de contour par seuillage

La différenciation des contours sur la carte générée se fait par seuillage à hysteresis.

Cela nécessite deux seuils, un haut et un bas; qui seront comparés à l'intensité du gradient de chaque point. Le critère de décision est le suivant. Pour chaque point, si l'intensité de son gradient est :

```
Inférieur au seuil bas, le point est rejeté ;
Supérieur au seuil haut, le point est accepté comme formant un contour ;
Entre le seuil bas et le seuil haut, le point est accepté s'il est connecté à un po
```

Une fois ceci réalisé, l'image obtenue est binaire avec d'un côté les pixels appartenant aux contours et les autres.

0.7 Qu'est-ce qu'on peut faire avec des gradients?

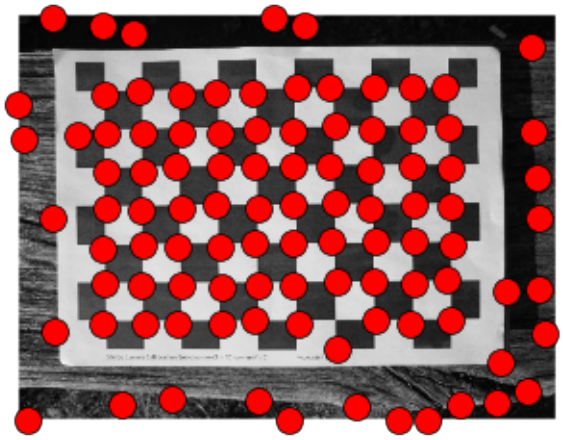
0.7.1 un extracteur de Harris? ca vous dit?

en 10 lignes de codes?

```
In [42]: from scipy.ndimage import filters
         def harris_points(I, alpha, windows,local_win):
             Ix = convolve(I, np.array([[-1.,0.,1.]]))
             Iy = convolve(I, np.array([[-1.,0.,1.]]).T)
             Ix2 = convolve(Ix * Ix, windows)
             Iy2 = convolve(Iy * Iy, windows)
             Ixy = convolve(Ix * Iy, windows)
             harris = Ix2*Iy2 - Ixy**2 - alpha * ((Ix2+Iy2)**2)
             harris_max = filters.maximum_filter(harris, footprint = local_win)
             return reversed(np.where(harris_max == harris))
```

In [52]:
```
I2 = cv2.imread('img/checkerboard.jpg', cv2.IMREAD_GRAYSCALE).astype(np.fl
```

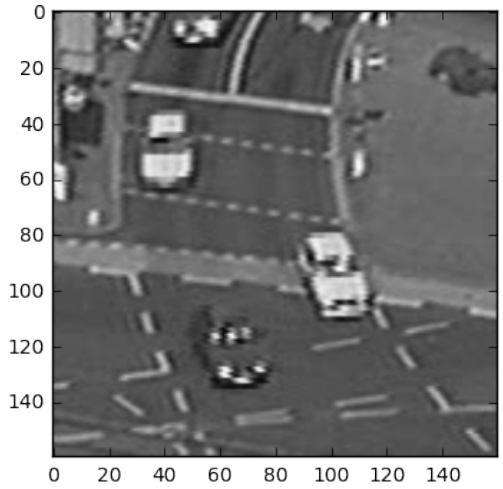
In [53]:
```
x, y = harris_points(I2, 0.05,np.ones([7,7]), np.ones([50,50]))
imagesc(I2)
_ = plot(x, y,'ro',ms=10)
```

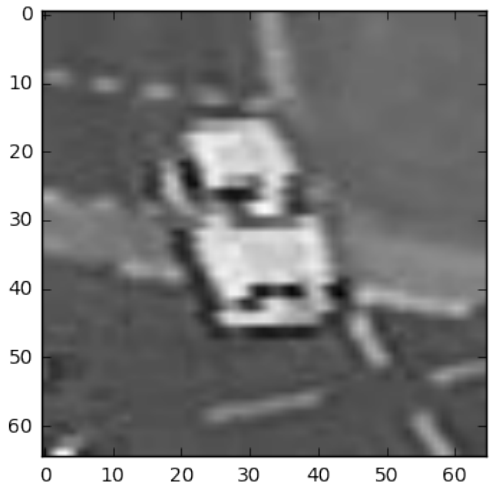


0.7.2 et en plus ca maarche! ø/

0.8 Et pour le mouvement de la voiture?

In [56]:
```
I = cv2.imread('./img/00000016.png', cv2.IMREAD_GRAYSCALE).astype(np.float
P = cv2.imread('./img/car.png', cv2.IMREAD_GRAYSCALE).astype(np.float)
subplot(1,2,1)
_ = imshow(I,'gray')
subplot(1,2,2)
_ = imshow(P,'gray')
```





$$SSD(x) = \sum_{(dx)inW(x)} ||I(x+dx) - P(dx)||^2$$

calculer une SSD ca a l'aire compliqué...

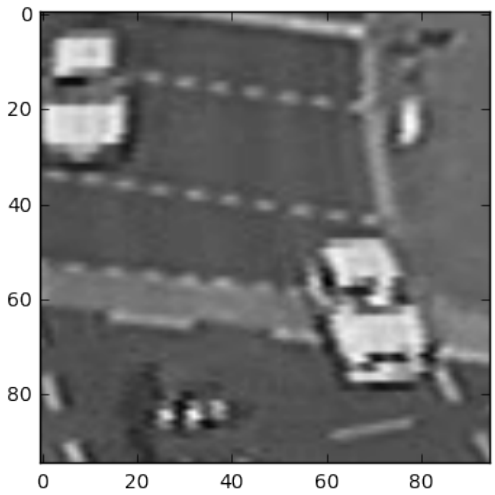
0.8.1 En une ligne de code?

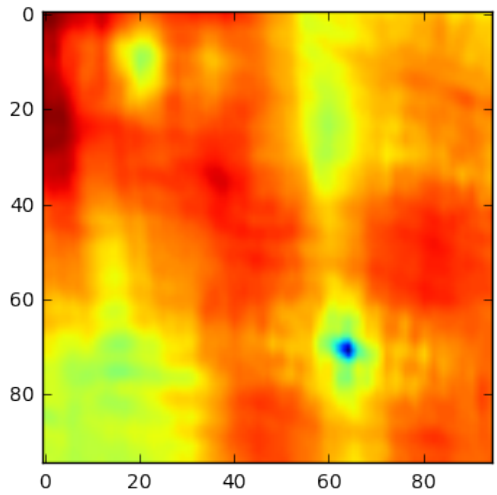
deal!

```
In [57]: r = P.shape[0]/2
         row, col = I.shape
         score = np.sqrt( np.array( [np.sum((P-I[y-r:y+r+1,x-r:x+r+1])**2)
                                     for x in range(r,col-r-1)
                                     for y in range(r,row-r-1)])
                        ).reshape([row-2*r-1, col-2*r-1])
```

Bon ok, un peut longue... mais avec un écran 4K ca tiens!

```
In [58]: subplot(1,2,1)
         _ = imshow(I[r:-r-1,r:-r-1], 'gray')
         subplot(1,2,2)
         _ = imshow(score)
```



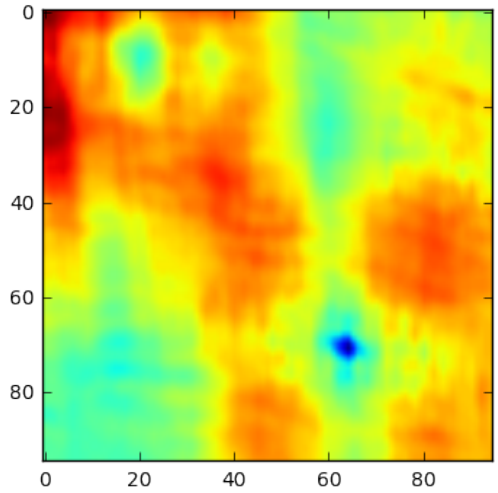


Si on décompose un peut ca donne :

```
In [59]: def scoreSSD(X,Y):
             return np.sum((X-Y)**2)

In [60]: dist = [scoreSSD(I[y-r:y+r+1,x-r:x+r+1], P) for x in range(r,col-r-1)
                                                     for y in range(r,row-r-1)]
         print type(dist)
         print dist[0]

<type 'list'>
16451717.0


In [61]: dist = np.array(dist)
         print dist
         print dist.shape

[ 16451717.  16440806.  16286897. ...,  12106325.  12205315.  12304704.]
(9025,)
```

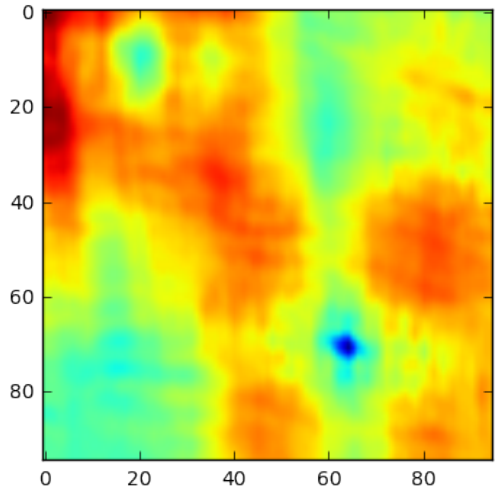
Il ne reste plus qu'a réordonner le vecteur pour en faire une image :

```
In [62]: SSD = dist.reshape([row-2*r-1, col-2*r-1])
         subplot(1,2,1)
         imagesc(I[r:-r-1,r:-r-1])
         subplot(1,2,2)
         _ = imshow(SSD)
```



0.8.2 en écrivant les bouclce for en mode C cela donne :

```
In [68]: dist = np.zeros([row-2*r-1, col-2*r-1])
         for x in range(r,col-r-1):
                 for y in range(r,row-r-1):
                     dist[y-r,x-r] = scoreSSD(I[y-r:y+r+1,x-r:x+r+1], P)
         subplot(1,2,1)
         imagesc(I[r:-r-1,r:-r-1])
         subplot(1,2,2)
         _ = imshow(SSD)
```



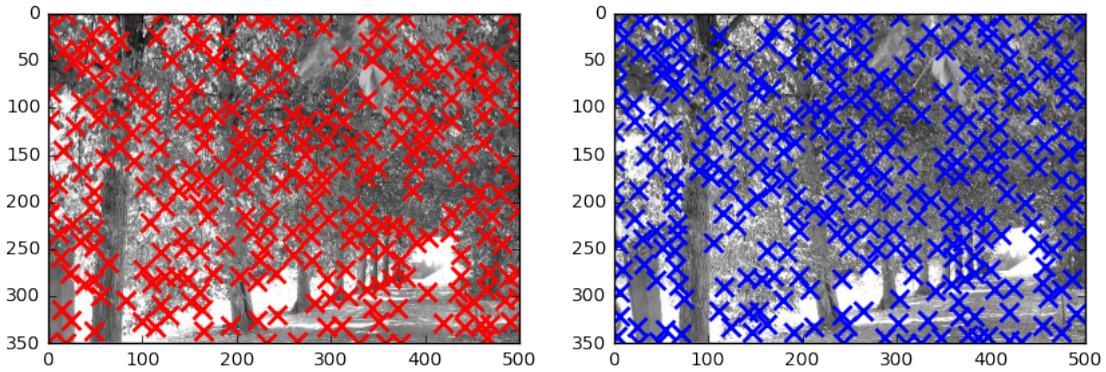
0.9 Calcul de mise en correspondance

On commence par charger deux images :

```
In [7]: I1 = cv2.imread('./img/img1.png', cv2.IMREAD_GRAYSCALE).astype(np.float) [:
        I2 = cv2.imread('./img/img2.png', cv2.IMREAD_GRAYSCALE).astype(np.float) [:
        pl.subplot(1,2,1), imagesc(I1)
        _= pl.subplot(1,2,2), imagesc(I2)
```

On extrait nos points de harris :

```
In [72]: x1, y1 = harris_points(I1, 0.05,np.ones([7,7]), np.ones([20,20]) )
         x2, y2 = harris_points(I2, 0.05,np.ones([7,7]), np.ones([20,20]) )
         subplot(1,2,1)
         _ = imshow(I1, 'gray')
         _ = plot(x1, y1,'rx',ms=10,mew=2)
         _ = axis((0,500,350,0))
         subplot(1,2,2)
         _ = imshow(I2, 'gray')
         _ = plot(x2, y2,'bx',ms=10, mew=2)
         _ = axis((0,500,350,0))
```



On supprime les points trop *borderline*

```
In [73]: rad = 15
         row, col = I1.shape
         pts1 = [(x, y) for x,y in zip(x1,y1) if x >= rad and x < col-rad
                                                 and y >= rad and y < row-rad]
         pts2 = [(x, y) for x,y in zip(x2,y2) if x >= rad and x < col-rad
                                                 and y >= rad and y < row-rad]
```

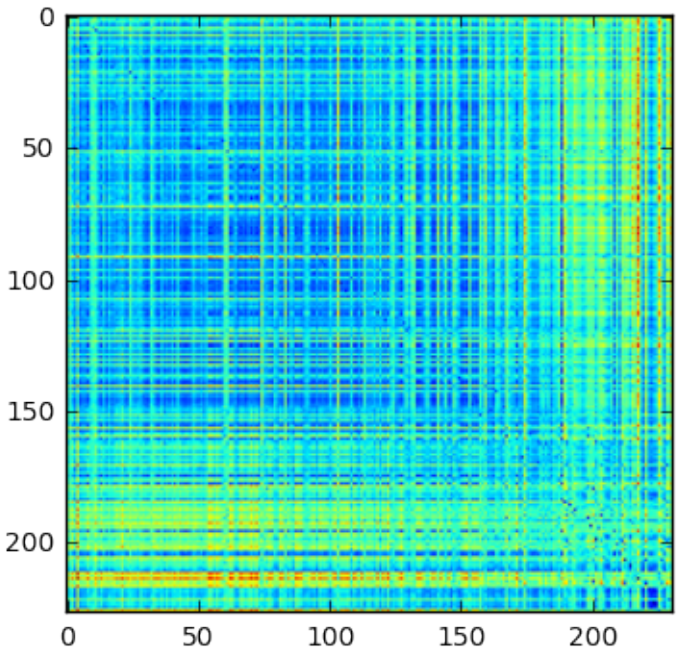
On va utiliser note fonction score SSD pour calculer les score et une petite fonction pour récupérer les patchs images

```
In [74]: def scoreSSD(X,Y):
             return np.sqrt(np.sum((X-Y)**2))

         def getPatch(I, pos, rad):
             return I[pos[1]-rad:pos[1]+rad+1, pos[0]-rad:pos[0]+rad+1]
```

notre score final sera stoqué dans une matrice **A**

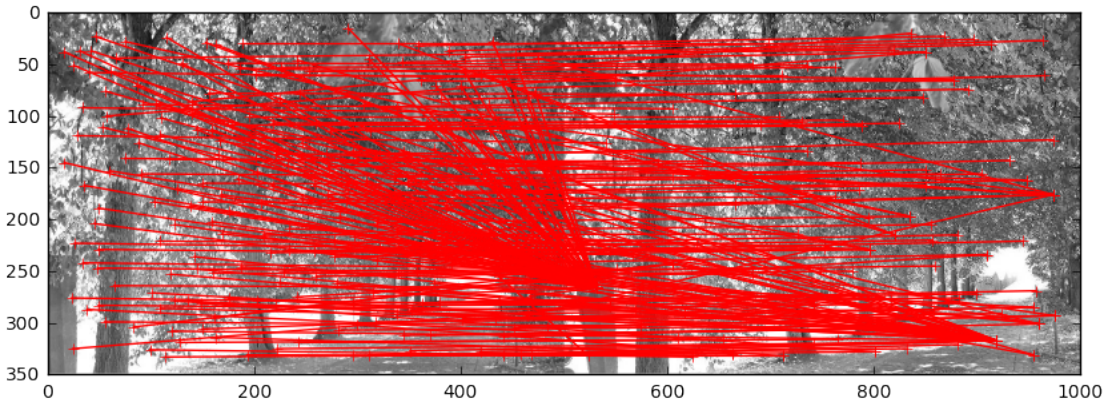
```
In [75]: def computeSSDMat(I1,I2,pts1,pts2):
             A = np.zeros([len(pts1), len(pts2)])
             for i1, p1 in enumerate(pts1):
                 for i2, p2 in enumerate(pts2):
                     A[i1,i2] = scoreSSD(getPatch(I1,p1,rad),getPatch(I2,p2,rad))
             return A
```

```
In [76]: Aff = computeSSDMat(I1,I2,pts1,pts2)
         _ = imshow(Aff)
```



Pour en extraire les affectation, on a deux solution : - de **1** vers **2** (*forward*) - de **2** vers **1** (*backward*)

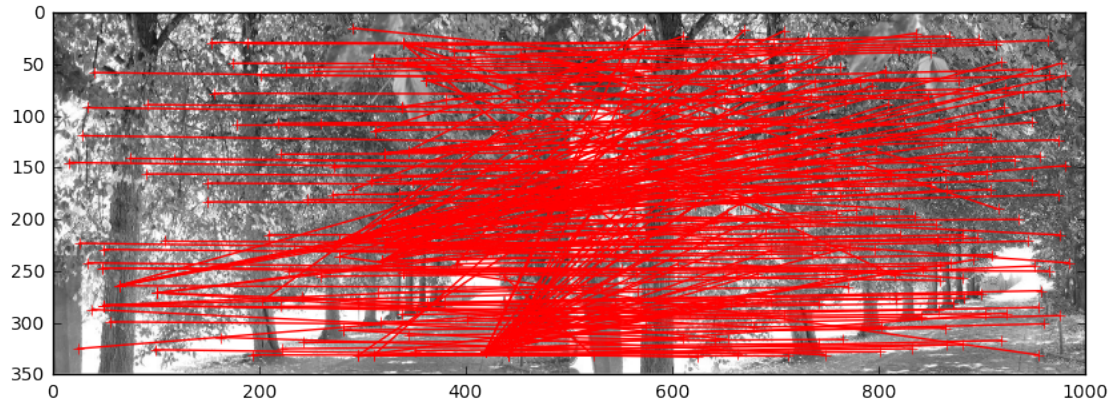
```
In [77]: def showCorrespondance(I0,I1,corr, color = 'r'):
             _ = imshow(np.concatenate([I0,I1], axis = 1), 'gray')
             for aff in corr:
                 m = np.array(aff)
                 m[1,0] += I0.shape[1]
                 _ = plot(m[:,0],m[:,1],color+'+-')
             axis((0,2*I0.shape[1],I0.shape[0],0))

In [78]: def forwardExtract(A, pts1, pts2):
             idx2 = np.argmin(A, axis = 1)
             return [(pts1[i], pts2[idx]) for i, idx in enumerate(idx2)]

In [79]: forward = forwardExtract(Aff,pts1,pts2)
         showCorrespondance(I1,I2,forward)
```



Pas terrible…

0.9.1 Et pour le backward?

```
In [80]: def backwardExtract(A, pts1, pts2):
             idx1 = np.argmin(A, axis = 0)
             return [(pts1[idx], pts2[i]) for i, idx in enumerate(idx1)]

In [81]: backward = backwardExtract(Aff,pts1,pts2)
         showCorrespondance(I1,I2,backward)
```

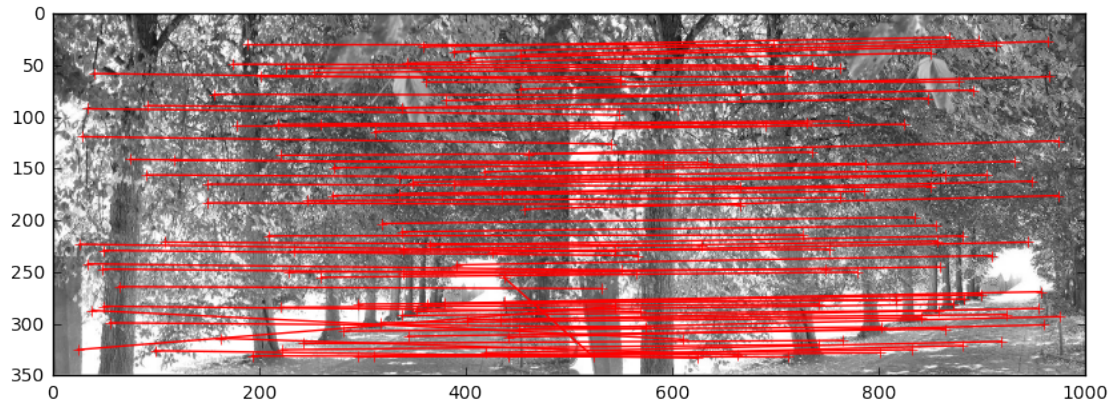


C'est pas mieux…

0.9.2 Et si on utilisait les deux?

On prend les affectations que si elles sont dans les deux…

```
In [82]: def forwardBackwardExtract(Aff, pts1, pts2):
             idx1 = np.argmin(Aff, axis = 0)
             idx2 = np.argmin(Aff, axis = 1)
             return [(pts1[idx], pts2[i]) for i, idx in enumerate(idx1)
                                                  if idx2[idx] == i]

In [83]: ar = forwardBackwardExtract(Aff, pts1, pts2)
         showCorrespondance(I1,I2,ar)
```

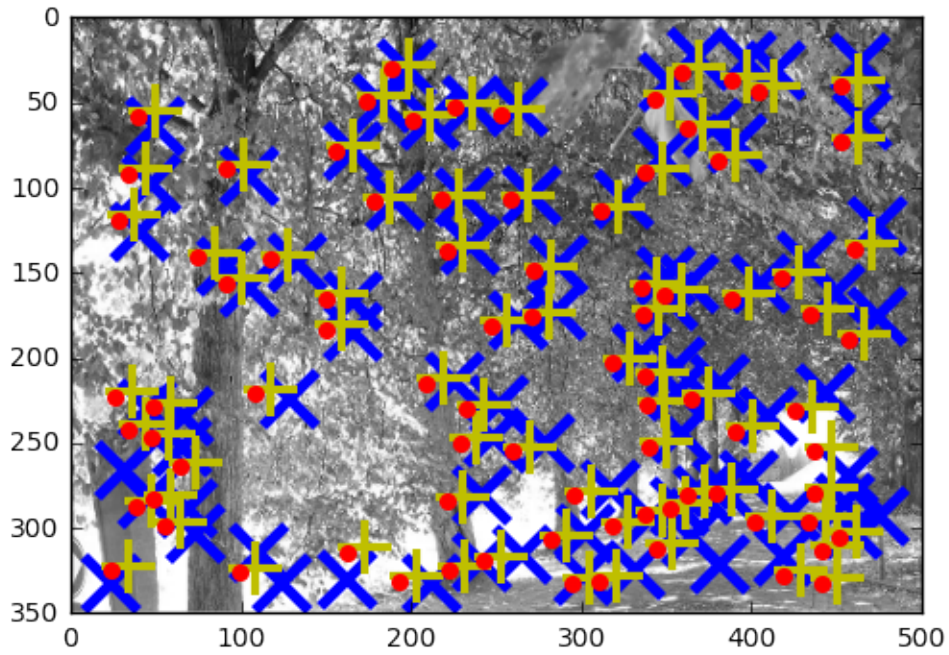


0.9.3 Bon avec des correspondance on peut faire quoi?

estimer une translation par exemple :

$$\hat{t} = \frac{1}{N}\sum_{i-0}^{N}(X_2(i) - X_1(i))$$

```
In [84]: X = np.array(ar).reshape([len(ar),4])

In [85]: X1 = X[:,:2]
         X2 = X[:,-2:]
         delta =  X2 - X1
         t = np.mean(delta,axis = 0)
         print 'translation : ', t

translation :  [ 9.83529412 -2.77647059]


In [86]: _ = axis((0, I1.shape[1], I1.shape[0], 0))
         _ = imshow(I2, 'gray')
         _ = plot(X2[:,0], X2[:,1], 'bx', mew = 4, ms = 20)
         _ = plot(X1[:,0]+t[0], X1[:,1]+t[1], 'y+', mew = 3, ms = 20)
         _ = plot(X1[:,0], X1[:,1], 'r.', mew = 4, ms = 5)
```



On s'est rapproché... mais c'est pas flagrant!

0.10 Sur l'importance de regarder les données

0.10.1 Le quatret d'Ascombe

```
In [19]: asc = np.loadtxt('data/ascomb.txt')
         print(np.mean(asc[:,:2],axis =0))
         print(np.mean(asc[:,2:4],axis =0))
         print(np.mean(asc[:,4:6],axis =0))
         print(np.mean(asc[:,6:8],axis =0))
```

```
[ 9.          7.50090909]
[ 9.          7.50090909]
[ 9.   7.5]
[ 9.          7.50090909]
```

0.10.2 Regardons la variance :

```
In [20]: print(np.std(asc[:,:2],axis =0))
         print(np.std(asc[:,2:4],axis =0))
         print(np.std(asc[:,4:6],axis =0))
         print(np.std(asc[:,6:8],axis =0))
```

```
[ 3.16227766  1.93702422]
[ 3.16227766  1.93710869]
[ 3.16227766  1.93593294]
[ 3.16227766  1.93608065]
```

0.10.3 Le coefficient de corrélation :

```
In [28]: print(np.corrcoef(asc[:,0], asc[:,1])[0,1])
         print(np.corrcoef(asc[:,2], asc[:,3])[0,1])
         print(np.corrcoef(asc[:,4], asc[:,5])[0,1])
         print(np.corrcoef(asc[:,6], asc[:,7])[0,1])
```

```
0.816420516345
0.816236506
0.81628673949
0.816521436889
```

0.10.4 La régression linéaire :

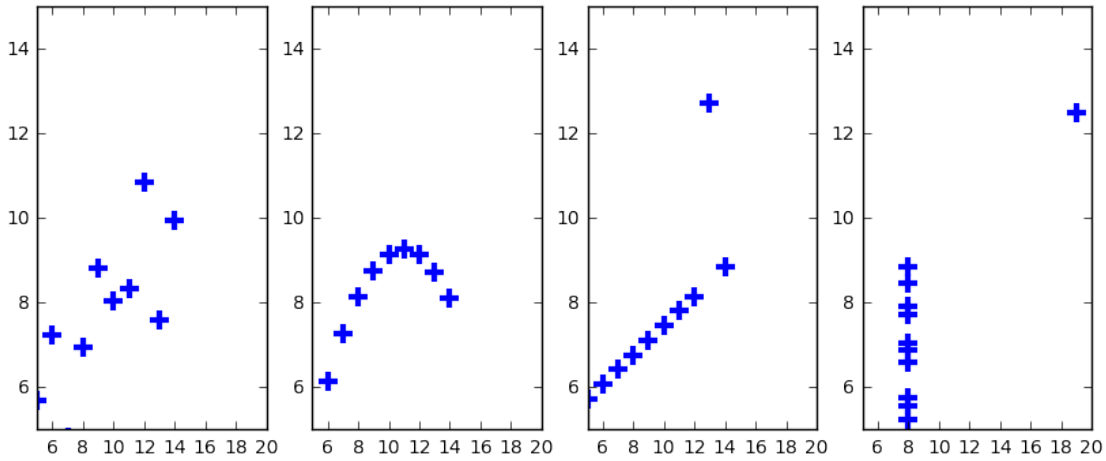
```
In [27]: from scipy.stats import linregress
         a, b, _, _, _ = linregress(asc[:,0], asc[:,1])
         print(a,b)
         a, b, _, _, _ = linregress(asc[:,2], asc[:,3])
         print(a,b)
         a, b, _, _, _ = linregress(asc[:,4], asc[:,5])
```

```
        print(a,b)
        a, b, _, _, _ = linregress(asc[:,6], asc[:,7])
        print(a,b)
```

```
(0.50009090909090914, 3.0000909090909103)
(0.50000000000000011, 3.0009090909090892)
(0.49972727272727291, 3.0024545454545439)
(0.49990909090909091, 3.0017272727272726)
```

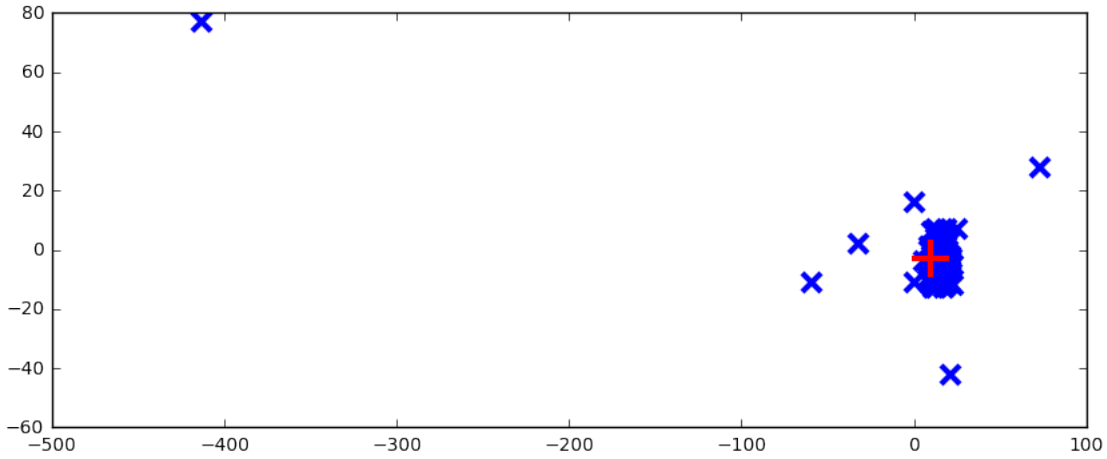
0.10.5 Regardons nos données !

```
In [36]: pl.figure()
         pl.subplot(1,4,1)
         pl.plot(asc[:,0], asc[:,1],'+', ms = 10, mew = 3)
         pl.axis((5,20,5,15))
         pl.subplot(1,4,2)
         pl.plot(asc[:,2], asc[:,3],'+', ms = 10, mew = 3)
         pl.axis((5,20,5,15))
         pl.subplot(1,4,3)
         pl.plot(asc[:,4], asc[:,5],'+', ms = 10, mew = 3)
         pl.axis((5,20,5,15))
         pl.subplot(1,4,4)
         pl.plot(asc[:,6], asc[:,7],'+', ms = 10, mew = 3)
         _ = pl.axis((5,20,5,15))
```



0.10.6 revenons a notre assosiation

```
In [99]: _ = plot(delta[:,0], delta[:,1],'bx', ms = 10, mew = 3)
         _ = plot(t[0], t[1], 'r+', ms = 20, mew = 3)
```



Comment améliorer tout ca?

0.10.7 Et si on cherchait une affinité?

petit rappel :

$$x_2 = a_1 * x_1 + a_2 * y_1 + a_3$$

$$y_2 = a_4 * x_1 + a_5 * y_1 + a_6$$

si on veut écrire sous la forme de la minimisation de moindre carrés :

$$\hat{x} = argmin(E(x))$$

où :

$$E(x) = (Ax - b)^2$$

et :

$$x = [a_1, a_2, a_3, a_4, a_5, a_6]^t$$

La solution du problème est alors :

$$\hat{x} = (A^t A)^{-1} A^t b$$

0.10.8 Comment on écrit A et b?

Ici A et b sont les *données* (les coordonnés **(x,y)** de nos points) :

$$A = \begin{pmatrix} \underline{x_1} & \underline{y_1} & \underline{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & \underline{x_1} & \underline{y_1} & \underline{1} \end{pmatrix}$$

et pour b :

$$b = \begin{pmatrix} \underline{x_2} \\ \underline{y_2} \end{pmatrix}$$

Les paramètres qu'on cherche à estimer sont les coefficients $a_{i,j}$ de la matrice d'affinité
Ca donne quoi en pyton?

```
In [100]: vone  = np.ones([X1.shape[0], 1])
          bzero = np.zeros([X1.shape[0], 3])

          tmp1 = np.concatenate([X1,    vone, bzero], axis = 1)
          tmp2 = np.concatenate([bzero, X1,   vone], axis = 1)

          A = np.concatenate([tmp1, tmp2], axis = 0)
          b = np.concatenate([X2[:,0], X2[:,1]], axis = 0).reshape(A.shape[0],1)
```

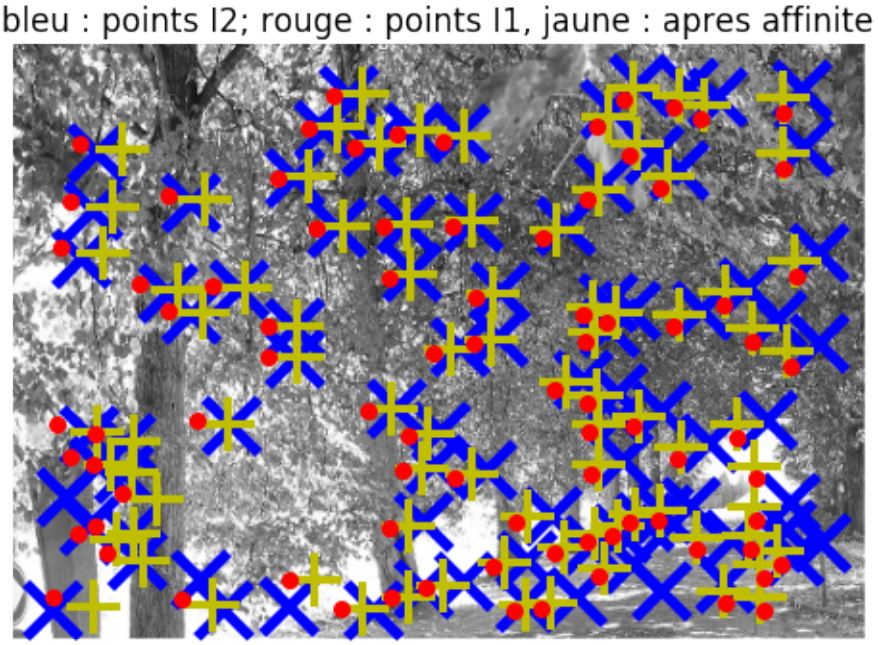
0.10.9 La solution est alors directe (en utilisant les fonction numpy) :

```
In [101]: affine = np.dot( np.linalg.inv(np.dot(A.T,A)), np.dot(A.T, b)).reshape([2
          print affine
```

```
[[  9.39972304e-01  -1.03160118e-02   2.77284091e+01]
 [ -2.94969226e-02   1.00503029e+00   4.01452495e+00]]
```

Ok mais ca donne quoi?

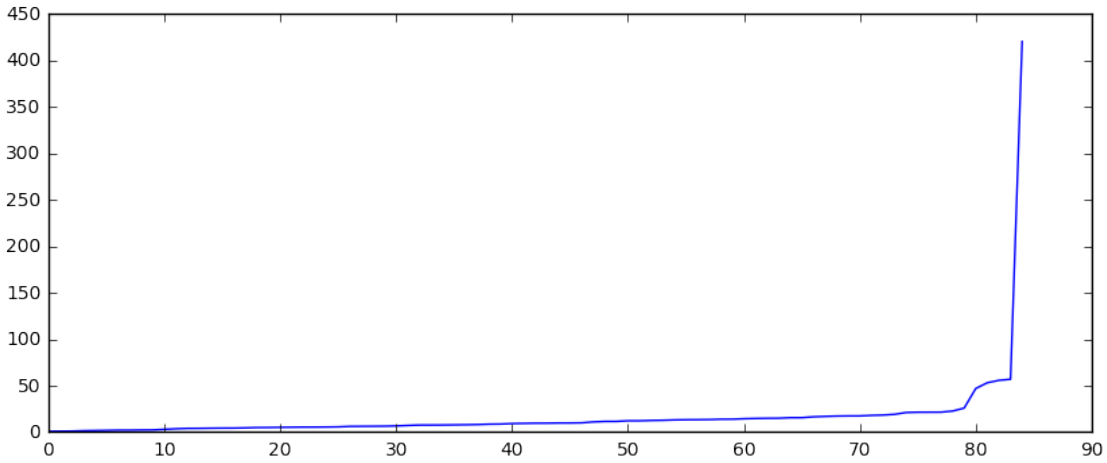
```
In [104]: X1_2 = np.dot(affine[:,:2], X1.T).T + np.dot(np.ones([X1.shape[0],1]),aff
          _ = axis((0, I1.shape[1], I1.shape[0], 0))
          imagesc(I2)
          _ = plot(X2[:,0], X2[:,1], 'bx', mew = 4, ms = 20)
          _ = plot(X1_2[:,0], X1_2[:,1], 'y+', mew = 3, ms = 20)
          _ = plot(X1[:,0], X1[:,1], 'r.', mew = 4, ms = 5)
          _ = pl.title('bleu : points I2; rouge : points I1, jaune : apres affinite
```



Pas super génial non plus...

0.10.10 Cherchons a comprendre : regardons le résidus

```
In [105]: residus = np.sqrt(np.sum((X2-X1_2)**2 , axis = 1))
          _ = plot(sort(residus))
```



ok, on a quelques association qui sont bien douteuse!

0.10.11 Ok donc on va filtrer les données abérante

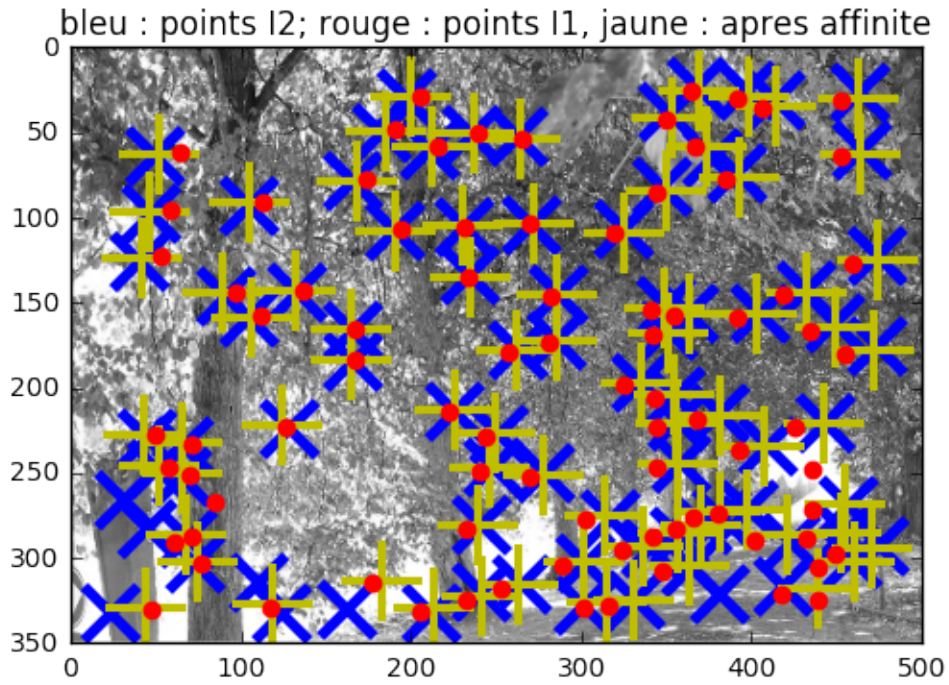
```
In [106]: def estimeAffine(X1,X2):
              vone = np.ones([X1.shape[0], 1])
              bzero = np.zeros([X1.shape[0], 3])
              tmp1 = np.concatenate([X1,    vone, bzero], axis = 1)
              tmp2 = np.concatenate([bzero, X1,   vone], axis = 1)
              A = np.concatenate([tmp1, tmp2], axis = 0)
              b = np.concatenate([X2[:,0], X2[:,1]], axis = 0)
              b = b.reshape(A.shape[0],1)
              affine = np.dot( np.linalg.inv(np.dot(A.T,A)), np.dot(A.T, b))
              affine = affine.reshape([2,3])
              X1_2 = np.dot(affine[:,:2], X1.T).T
              X1_2 += np.dot(np.ones([X1.shape[0],1]),affine[:,2].reshape([1,2]))
              return affine, X1_2
```

On se fixe un seuil de 50 pixel de résidus et on relance la machine!

```
In [107]: valid = (residus < 50)
          affine2, newpts = estimeAffine( X1[valid,:], X2[valid,:])
```

0.10.12 Et ca donne quoi ?

```
In [109]: _ = axis((0, I1.shape[1], I1.shape[0], 0))
          _ = imshow(I2, 'gray')
          _ = plot(X2[:,0], X2[:,1], 'bx', mew = 4, ms = 20)
          _ = plot(newpts[:,0], newpts[:,1], 'y+', mew = 3, ms = 30)
          _ = plot(X1_2[:,0], X1_2[:,1], 'r.', mew = 4, ms = 6)
          _ = pl.title('bleu : points I2; rouge : points I1, jaune : apres affinite
```



0.10.13 ø/