

# Utilisation du script de détection

Lucas Reding

École des Mines de Saint-Étienne. Saint-Étienne, 42000, France.

Email: lucas.reding@univ-rouen.fr

## Introduction

L'objectif de l'algorithme ci-dessous est de détecter les "dunes" et d'en déduire leur vitesse en partant d'un diagramme spatio-temporel fourni en paramètre. L'analyse d'image se fait en 3 étapes :

- On transforme l'image initiale en une image binaire à l'aide d'un seuillage.
- On détecte les bords montant et descendant de chaque "dune" et on les groupe ensemble.
- On détermine la droite de régression des bords montant et descendant et on retourne son coefficient directeur.

La détection des bords montant s'effectue en caractérisant ces derniers comme étant un pixel noir suivi de  $n$  pixels blancs (où  $n > 0$  est un paramètre à régler) selon l'axe vertical. De même un bord descendant est caractérisé par un pixel noir précédé de  $n$  pixels blancs.

```
thr_pre = 84

bin_dw = pixels < thr_pre
bin_up = pixels < thr_pre

pixels = pixels >= thr_pre

for i in range(1,n+1):
    bin_dw = bin_dw & np.roll(pixels,i, axis=0)

for i in range(1,n+1):
    bin_up = bin_up & np.roll(pixels,-i, axis=0)
```

L'étape de regroupement consiste à élargir les ensembles ainsi obtenus au travers d'une dilatation morphologique.

```
bin_up = ndimage.morphology.binary_dilation(bin_up)
bin_dw = ndimage.morphology.binary_dilation(bin_dw)
binary = bin_up | bin_dw
```

Puis on élimine le bruit.

```
num, lbls, stats, centroids = cv2.connectedComponentsWithStats(np.int8(binary))
idx = np.where(stats[:,4] > thr_nb)
idx = np.delete(idx[0],0)

for i in tqdm(idx):
    mask = np.nonzero((np.array(lbls)-i) == 0)
    res = res + ((np.array(lbls)-i) == 0)

    imax = np.where(mask[0] == np.max(mask[0]))[0]
    l = (np.array([mask[1],mask[0]]).T)[imax]
    xmax,ymax = np.sum(l, axis=0)/np.size(l, axis=0)

    imin = np.where(mask[0] == np.min(mask[0]))[0]
    l = (np.array([mask[1],mask[0]]).T)[imin]
    xmin, ymin = np.sum(l, axis=0)/np.size(l, axis=0)

    if xmax != xmin:
        coeff = (ymax - ymin)/(xmax-xmin)
    else:
        coeff = 0

    if coeff > coeff_min and coeff < coeff_max:
        data[i] = [[xmax,ymax],[xmin,ymin],i]
        j = j + 1
```

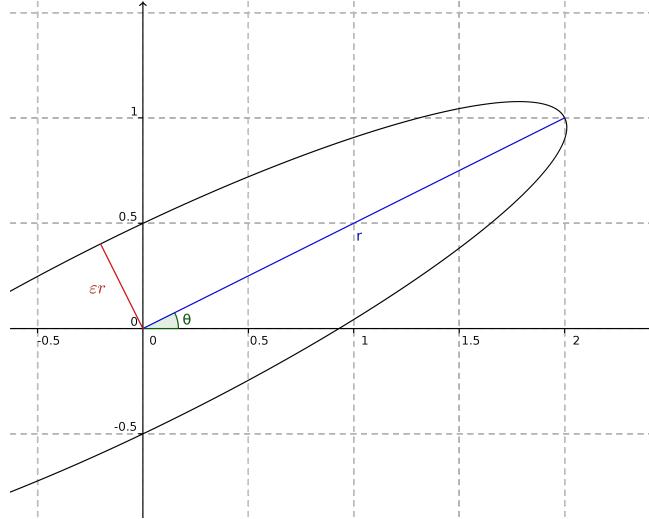


Figure 1: Ellipse  $\mathcal{E}(r, \theta, \epsilon)$

Enfin on regroupe ensemble le bord montant et le bord descendant de chaque dune. Remarquons qu'en raison du bruit initial de l'image, il est probable que chaque bord (montant ou descendant) soit détecter en plusieurs morceaux. On commence alors par regrouper les différentes parties de chaque bord. Pour cela, on commence par définir une distance  $d(x, y) = \sqrt{\langle x, y \rangle}$  with  $\langle x, y \rangle$  the scalar product of  $x$  and  $y$  associated to the ellipse  $\mathcal{E}(r, \theta, \epsilon)$  defined by the semi-major axis length  $r$  and its support line (itself determine by the angle  $\theta$ ) as well as the semi-minor axis length  $\epsilon r$  (see Figure 1). On remarque qu'alors

$$\langle \begin{pmatrix} x \\ y \end{pmatrix}, \begin{pmatrix} x \\ y \end{pmatrix} \rangle = x^2 \left( \cos^2 \theta + \frac{\sin^2 \theta}{\epsilon^2} \right) + xy \left( \cos \theta \sin \theta - \frac{\cos \theta \sin \theta}{\epsilon^2} \right) + y^2 \left( \sin^2 \theta + \frac{\cos^2 \theta}{\epsilon^2} \right)$$

et on détermine alors le voisin le plus proche se trouvant à une distance inférieure ou égale à  $r$ , s'il existe, pour chaque composante connexe.

```
def neighbors(d, r, theta, epsilon):
    N = {}
    for i in d.keys():
        for j in d.keys():
            if i!=j and dist(d[i][0], d[j][1], theta, epsilon) <= r:
                if d[i][2] not in N.keys() or (d[i][2] in N.keys() and dist(d[i][0], d[N[d[i][2]]][1], theta, epsilon) <= dist(d[i][0], d[N[d[i][2]]][1], theta, epsilon)):
                    N[d[i][2]] = d[j][2]
    return N
```

On identifie alors tous les composantes connexes voisines.

```
def update_color(i, N, c, data):
    data[i][2] = c
    if i in N.keys():
        update_color(N[i], N, c, data)
```

Ensuite, on fusionne le bord montant et descendant de chaque "dune". Afin de les regrouper correctement, on parcourt verticalement les pixels situés au dessus des points extrémaux du bord descendant ainsi que du milieu de celui jusqu'à ce l'un de ces pixels fasse partie d'un bord montant (voir Figure 2). Puis on indentifie alors les deux bords.

```
for i in tqdm(data.keys()):
    d = data[i]
    xmax = int(d[0][0])
    ymax = int(d[0][1])
    xmin = int(d[1][0])
    ymin = int(d[1][1])
    xmed = int((xmax + xmin)/2)
    ymed = int((ymax + ymin)/2) - n

    while ymed > 0 and binary[ymed, xmed] == False:
        ymed = ymed-1

    if bin_up[ymed, xmed] == True and lbls[ymed, xmed] in data.keys():
        data[i][2] = data[lbls[ymed, xmed]][2]
    else:
```

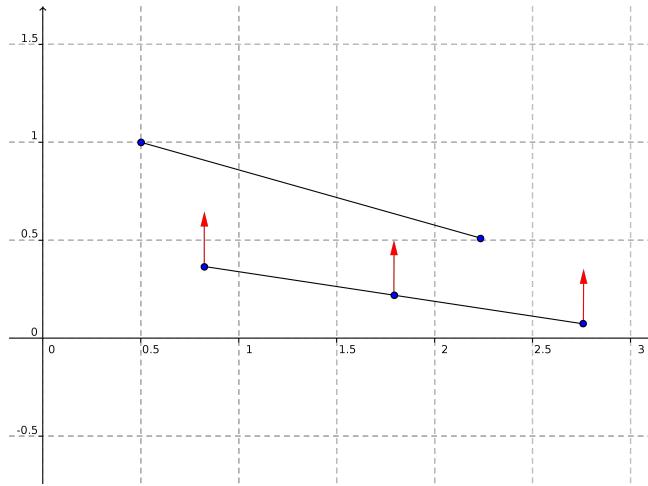


Figure 2: Processus de fusion

```

while bin_up[ymax,xmax] == False and bin_up[ymin,xmin] == False and ymin >= 0:
    ymax = ymax -1
    ymin = ymin -1
if bin_up[ymin,xmin] == True and lbls[ymin,xmin] in data.keys():
    data[i][2] = data[lbls[ymin,xmin]][2]
elif bin_up[ymax,xmax] == True and lbls[ymax,xmax] in data.keys():
    data[i][2] = data[lbls[ymax,xmax]][2]

```

## Algorithme

### Paramètres

Comme nous l'avons vu, l'algorithme contient de nombreux paramètres. Dans cette section, nous nous intéressons aux choix de ces derniers. Les différents paramètres sont réunis au début du fichier source.

```

thr_pre = 84
approx = 0.46
theta = np.arctan(approx)
epsilon = 0.2
thr = 100
thr_nb = 200
coeff_min = 0.25
coeff_max = 0.75
n = 5

```

Paramètre	Valeur
thr_pre	96
approx	0.46.
epsilon	0.1
thr	200
thr_nb	200
coeff_min	0
coeff_max	0.75
n	5

Paramètres pour le fichier "SpatioTemporal\_1\_19737.JPG"

Paramètre	Valeur
thr_pre	84
approx	0.46.
epsilon	0.2
thr	100
thr_nb	200
coeff_min	0.25
coeff_max	0.75
n	5

Paramètres pour le fichier "SpatioTemporal\_1mL2.JPG"

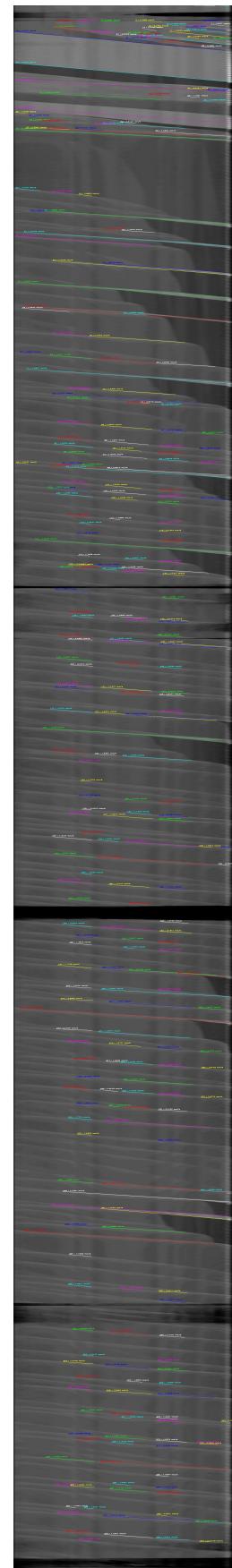
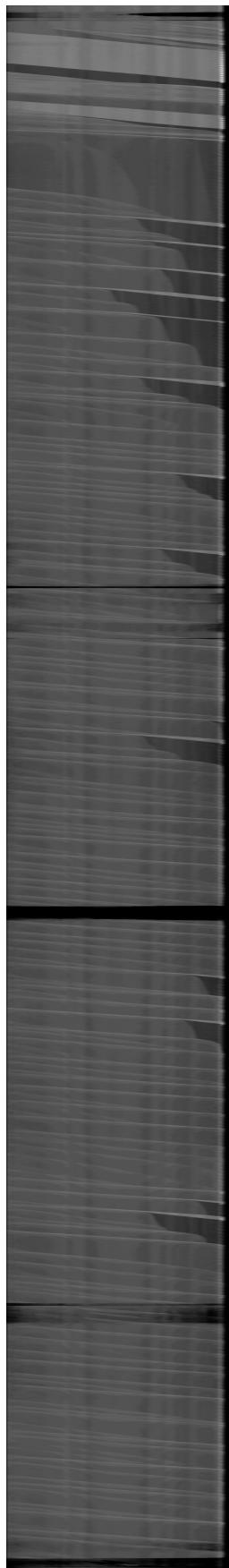
Paramètre	Description
thr_pre	Niveau de seuil de l'image : un pixel ayant une valeur de gris supérieure à cette limite sera considéré comme significatif (i.e. possiblement faisant partie d'une "dune"). Une valeur plus grande permet d'éliminer plus de bruit mais risque d'éliminer des "dunes". Afin de choisir cette valeur, on peut utiliser un logiciel de graphisme tel que Gimp et effectuer une transformation "seuil" jusqu'à trouver une valeur appropriée (on cherche à ce que chaque dune forme une ou plusieurs composante connexe sans que deux "dunes" soient connectées).
approx	Permet d'optimiser la détection des bords montant et descendant autour de la droite d'équation $y = approx \times x$ (ce qui correspond à une vitesse de 0,4 mm/s).
theta	Correspond à l'angle formé par la droite d'équation $y = approx \times x$ et l'horizontale. Ce paramètre est entièrement dépendant du paramètre approx. Il ne fait donc pas partie des paramètres à déterminer.
epsilon	Longueur du demi petit axe de l'ellipse $\mathcal{E}(1, theta, epsilon)$ . Une valeur plus grande permet de limiter les erreurs de multiple détections d'un bord mais une valeur trop grande causera la détection d'un bord descendant en un bord montant.
thr	Distance maximale (pour $d$ ) pour la détection du voisin le plus proche d'une composante connexe. Plus la valeur est grande plus on réduira l'erreur de multiple détection de bord mais une valeur trop grande causera la détection d'un bord descendant en un bord montant. A mettre en relation avec epsilon.
thr_nb	Nombre minimum de pixels pour qu'une composante connexe soit considérée comme une "dune". Une valeur plus grande permet d'éliminer plus de bruit mais risque d'éliminer des "dunes".
coeff_min	Valeur minimale pour le coefficient directeur de la droite de régression. Permet d'éliminer les faux positifs. Si la valeur est trop grande, on risque d'éliminer des données valides.
coeff_max	Valeur maximale pour le coefficient directeur de la droite de régression. Permet d'éliminer les faux positifs. Si la valeur est trop petite, on risque d'éliminer des données valides.
n	Nombre de pixels blancs (après l'étape de thresholding) qu'il faut avoir à la suite afin de détecter un bord montant (resp. descendant). Plus la valeur est haute, moins il y aura de bruit; cependant une valeur plus basse permet de mieux détecter les bords.

## Exemples

Nous illustrons l'utilisation de ce script au travers de deux exemples : "SpatioTemporal\_1\_19737.JPG" et "SpatioTemporal\_1mL2.JPG". Pour chaque relevé spatiotemporel, il faut choisir des paramètres adaptés. Le tableau donne les coefficients directeurs estimés pour chaque dune en fonction de l'indice qui apparaît sur le fichier "detection.jpg". Les résultats de l'algorithme avec les paramètres adaptés pour chaque exemple se situent dans les annexes.



## Annexe 1 : Diagramme Spatiotemporel "SpatioTemporal\_1\_19737.JPG"



## Annexe 2 : Diagramme Spatiotemporel "SpatioTemporal\_1mL2.JPG"

