UNIVERSITÉ CÔTE D'AZUR

POLYTECH NICE SOPHIA

Rapport intermédiaire de Projet d'Ingénierie

Construction d'un sujet de TP de mesure de débit par analyses d'image (LSPIV)

Mardi 12 Décembre 2020

Étudiants :

- BERRIANE Aymane
- VIGUIE Antoine
- ALVAREZ LEON Christian
- WANG Meng
- PENG Yu

Encadrants :

- M. BRIGODE Pierre
- M. DELESTRE Olivier

Polytech Nice Sophia Campus SophiaTech
930, Route des Colles - BP145
06903 Sophia-Antipolis Cedex France

SOMMAIRE

Introduction **3**

la préparation de l'image **4**

La correction de distorsion optique 4

l'orthorectification 8

La mise en niveau de gris et la correction du gamma et du contraste. 12

Pour tous les images 14

le traitement PIV **15**

l'analyse des donnée **19**

Résultats non traités 19

Filtrage 22

Substitution 23

Estimation de décharge 26

RÉFERENCES **27**

Introduction

Dans cette annexe , on va présent une méthode de python qui a une fonctionnalité similaire au Fudda-LSPIV . Une petite rappel : La technique LSPIV (Large Scale Particle Image Velocimetry) permet de mesurer les vitesses de surface d'un écoulement par analyse de séquence d'images. La méthode LSPIV complète se compose généralement de trois parties principales:

1. la préparation de l'image qui permet de corriger la distorsion optique , et faire l'ortho rectification.
2. le traitement PIV qui permet de calculer le champ de vitesse de surface.
3. l'analyse des donnée qui permet en déduit le débit en utilisent le profil bathymétrique d'une section en travers et la vitesse moyennée de profondeur .

Quand on utilisent le logiciel Fudda-LSPIV , Certain d'entre nous a rencontré des bug techniques ,Dans cette contexte , on a décidé d'utiliser Python Comme alternative. on s'intéresse par python , parce que le python nous offrit des fonctionnalité qui n'est pas intégré dans le logiciel Fudda-LSPIV , par exemple pour corriger la distorsion optique, c'est pas intégré dans Le Fudda-LSPIV mais c'est un étape essentiel. Et d'ailleurs, python est plus puissant pour analyser et traiter les donnés .

On est surtout inspiré par le rapport Flood wave monitoring using LSPIV de G.H. Gerritsen qui est publié dans Github , dans son rapport , il décit un étude de LSPIV à Chuo Kikuu, Dar es Salaam, Tanzania, Notre étude se base sur cette étude. Les données sont accessible sur Github.

la préparation de l'image

La préparation de l'image est nécessaire pour être en mesure de mieux distinguer les motifs en mouvement de l'imagerie et de supprimer les effets de perspective de l'image. La manipulation d'image pour préparer l'imagerie pour l'analyse LSPIV se composent de plusieurs étapes. Subséquemment, ces étapes sont:

(1) La correction de distorsion optique.

(2) l'orthorectification.

(3) La mise en niveau de gris et la correction du gamma et du contraste.

Les deux premières étapes sont appliquées pour garantir la présence de distances égales dans les images. La troisième étape est utilisée pour améliorer la distinction de la graine de l'arrière-plan et donc s'assurer de la validation de similitude processus est efficace.

Dans cette partie, on va utiliser la libraire de python Opencv3[noa] qui nous permette d'appliquer des options de manipulation d'images et le vidéo. vous pouvez l'installer par cette ligne de commande.

```
pip install opencv-python
```

On a aussi besoin d'autres libraires python numpy, pandas, math pour traiter des données . Et si vous voulez refaire cette étude sur vos ordinateurs, On vous conseille d'utiliser le python installé de site Anaconda et jupyter notebook

La correction de distorsion optique

En raison de la la courbure des lentilles de l'appareil photo, les images peuvent être déformées. Figure 1 présente les deux types de distorsions des lentilles les plus fréquentes,la distorsion en barillet et la distorsion en coussinet [Fryer and Brown] , Ces distorsions géométriques sont liées à des facteurs radiaux. Un troisième type de distorsion est distorsion tangentielle, qui se produit lorsque les lentilles ne sont pas parallèles au plan image. Certaines caméras sont capables de faire face à ces distorsions intérieurement. Cependant, la plupart du temps, une certaine quantité de post-traitement est nécessaire pour ajuster les images.

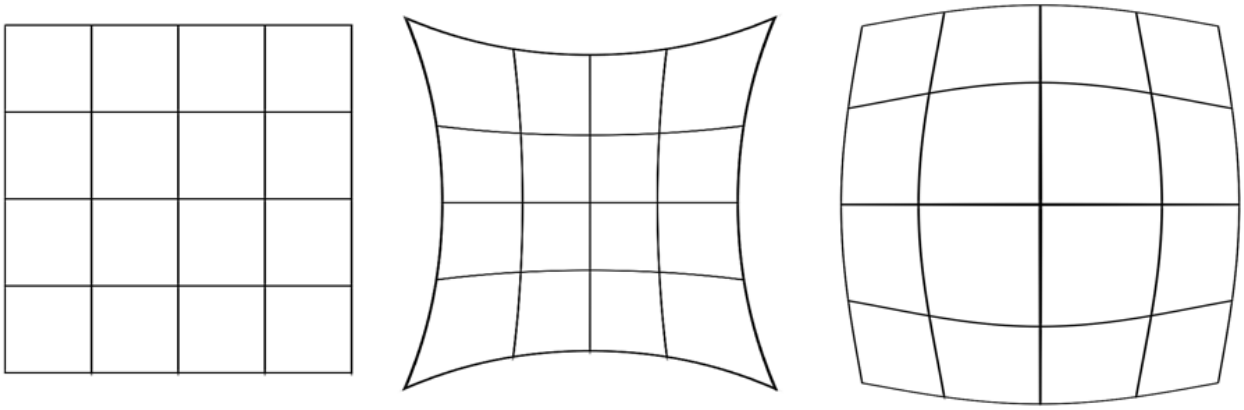


Figure 1: Les differentes types de distortion. Du gauche à la droit : la grille d'origine, la distorsion en barillet et la distorsion en coussinet.

Les formules suivantes sont appliquées pour supprimer la distortion radial (Voir Équation(1)) et la distorsion tangentielle (Voir Équation(2))

$$\begin{aligned} x_{corr} &= x\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) \\ y_{corr} &= y\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) \end{aligned} \tag{1}$$

D'où x et y sont les coordonnées d'origine; x_{corr} et y_{corr} sont les coordonnées corrigés. r est la distance entre le point (x, y) et le centre de la distortion. k_1, k_2, et k_3 sont les coefficients radiales. Pour la distortion en barillet et la distortion en coussinet , k_1 est respectivemen negative et positive. k_2 et k_3 sont négligeables.

$$\begin{aligned} x_{corr} &= x + \left[2p_1 xy + p_2\left(r^2 + 2x^2\right)\right] \\ y_{corr} &= y + \left[p_1\left(r^2 + 2y^2\right) + 2p_2 xy\right] \end{aligned} \tag{2}$$

D'où p_1 et p_2 sont les coefficient de la distortion tangentielle. les différents coefficients sont souvent stockés dans un tableau:

$$C_{dis} = \begin{bmatrix} k_1 & k_2 & p_1 & p_2 & k_3 \end{bmatrix} \tag{3}$$

À part du coefficient de distorsion, pour pouvoir corriger l'imagerie, Une conversion entre les coordonnées de distorsion et la résolution de la caméra est fait. Pour cela, la formule est donnée dans l'équation (4)

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = M_{con} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad \text{d'où} \quad M_{con} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{4}$$

D'où $[x, y, w]$ sont les coordonnées d'image homogène 2D et $[x, y, z]$ sont les coordonnées de caméra 3D . f_x et f_y sont les longueurs locales de la caméra dans le sens x et y ,en genéral, ils sont identiques c_x et c_y sont les coordinations de centre optique de la caméra dans le sens x et y.

```
%matplotlib inline
```

```
import numpy as np
import os
import matplotlib.pyplot as plt
import pandas as pd
import cv2
```

```
dir_video = 'Python/example_video.mp4'
dir_saves = 'Python/frames'
```

```
if not os.path.exists(dir_saves):
    os.mkdir(dir_saves)
```

```
# définition de correction de la distorsion
def lens_corr(img, k1=-10.0e-6, c=2, f=8.0):

    # définition la caractère de l'image
    height, width, __ = img.shape

    # le vecteur de coefficient de distorsion
    dist = np.zeros((5,1),np.float64)
    dist[0,0] = k1

    # la matrice du caméra
    mtx = np.eye(3,dtype=np.float32)

    mtx[0,2] = width/c      # centre x
    mtx[1,2] = height/c     # centre y
    mtx[0,0] = f            # la distance focale x
    mtx[1,1] = f            # la distance focale y

    # la correction d'image par la distorsion
    corr_img = cv2.undistort(img, mtx, dist)

    return corr_img
```

```
# pour localiser les GCPts dans les images:
# importer la vidéo et corriger la première image pour la distorsion de l'objectif
cap = cv2.VideoCapture(dir_video)
if cap.isOpened():
    ret, img = cap.read()
    corr_img = lens_corr(img, k1=-10.0e-6, c=2, f=8.0)
cap.release()

# afficher l'image corrigée et les emplacements des GCP
fig = plt.figure(figsize=(14,5))
plt.subplots_adjust(left=0, bottom=0, right=1, top=1, wspace=0.1, hspace=0)
# [:,:,::-1] BGR => RGB ; Dans opencv c'est BGR
plt.subplot(121)
plt.imshow(img[:,:,::-1])
plt.title('AVANT')
plt.axis('off')
```

```
## (-0.5, 1919.5, 1079.5, -0.5)
```

```
plt.subplot(122)
plt.imshow(corr_img[:,:,::-1])
plt.title('APRÈS')
plt.axis('off')
```

```
## (-0.5, 1919.5, 1079.5, -0.5)
```

```
plt.show()
```

Figure 2: La correction de distorsion optique

l'orthorectification

Pour supprimer les effets de la perspective de l'image ,(où les objets sont plus proches de la caméra semble être plus grande dans l'image) ,l'orthorectification est appliquée. Lors de l'application d'orthorectification,le système de coordonnées de l'imagerie est transféré à une système de coordonnée locale. Pour ce système de coordonnées locales, points de contrôle au sol (GCP) sont mis en place à côté du flux sont utilisés. Pour l'orthorectification processus pour être aussi précis que possible, au moins quatre GCP sont nécessaires, si les images sont capturées perpendiculairement au flux ou lorsque les GCP sont placés au même niveau que le niveau de l'eau. Un minimum de six GCP sont nécessaires lorsque les GCP ne sont pas placés dans le même plan que le niveau d'eau. Sur la Figure 3 les différents sites de jaugeage et les configurations sont affichées.

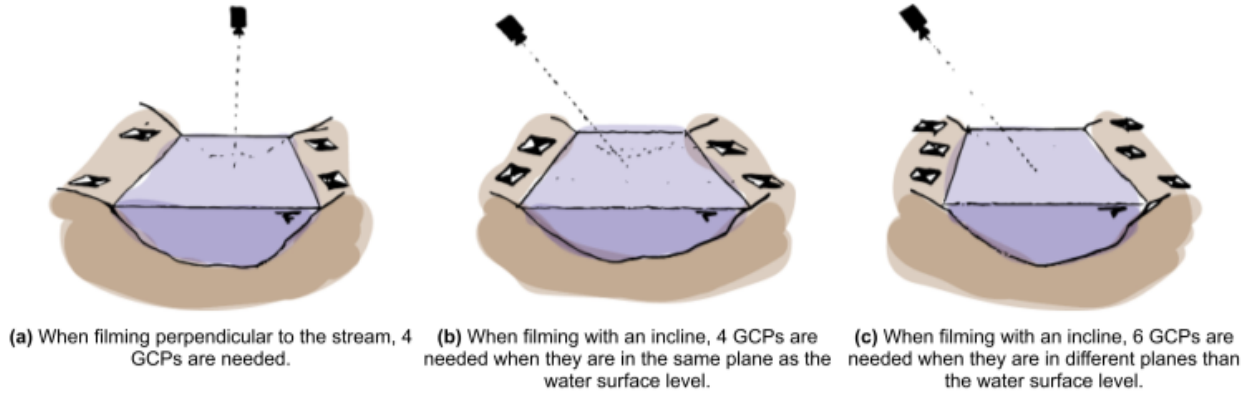


Figure 3: Nombre de points de contrôle au sol (GCP) nécessaires dans différentes circonstances.(source: Flood wave monitoring using LSPIV)

Lors de l'utilisation de quatre GCP, les facteurs d'inversion f_x et f_y sont déterminé en utilisant la formule indiquée dans l'équation (5).

$$p_{loc}(x,y) = p_{img}\left(f_x(x,y), f_y(x,y)\right) \tag{5}$$

D'où $p_{dst}(x,y,z)$ est l'emplacement géographique du point de contrôle au sol dans le système de coordonnées local, généralement en métrique units; $p_{src}(x,y)$ est la coordonnée xy du sol point de contrôle dans l'imagerie, généralement en pixels; et f_x et f_y le facteurs d'inversion.

Simultanément à ce processus, la résolution de l'image peut être réglée en multipliant les coordonnées $GCP_{dst}(x,y)$ par les pixels souhaités par coefficient de mètre.

Lorsque six (ou plus) GCP sont utilisés - parce que les coordonnées en trois dimensions pour les GCP sont nécessaires - un modèle de sténopé peut être utilisé. Cette méthode est expliqué par [Jodeau et al.]

```
# Emplacements des GCPs dans les images (pixels)
data_from = {'x': [992, 1545, 1773,  943],
             'y': [366,  403,  773,  724]}
df_from = pd.DataFrame(data_from)



fig = plt.figure(figsize=(7,4))
plt.subplot(111)
plt.subplots_adjust(left=0, bottom=0, right=1, top=1, wspace=0.1, hspace=0)
plt.imshow(corr_img[:,:,::-1])
plt.plot(df_from.x, df_from.y ,'r.')
plt.axis('off')


## (-0.5, 1919.5, 1079.5, -0.5)
```

```
plt.show()
```

Figure 4: Les localisations des GRPS

```
# Emplacements des GCPs dans le système de coordonnées local (mètres)
data_to = {'x': [ 0.25, 4.25, 4.50, 0.00],
           'y': [ 0.30, 0.20, 3.50, 3.50]}
df_to = pd.DataFrame(data_to)
```

```
def orthorect_param(img, df_from, df_to, PPM=100):

    # définir le valeur de float32(par point)
    pts1 = np.float32(df_from.values)
    pts2 = np.float32(df_to.values * PPM)

    # définir une matrice de transformation basée sur les GCP
    M = cv2.getPerspectiveTransform(pts1, pts2)

    # trouver les emplacements des coins d'image transformés
    height, width, __ = img.shape
    C = np.array([[0, 0, 1],
                  [width, 0, 1],
                  [0, height, 1],
```

```
                    [width, height, 1]])
    C_new = np.array([(np.dot(i, M.T) / np.dot(i, M.T)[2])[:2] for i in C])

    C_new[:, 0] -= min(C_new[:, 0])
    C_new[:, 1] -= min(C_new[:, 1])

    # définir une nouvelle matrice de transformation basée sur les coins de l'image
    # sinon, une partie des images ne sera pas enregistrée
    M_new = cv2.getPerspectiveTransform(np.float32(C[:,:2]), np.float32(C_new))

    return M_new, C_new, df_to
```

```
#déterminer les paramètres d'orthorectification
M, C, __ = orthorect_param(corr_img, df_from, df_to, PPM=100)

# définir la taille de l'image orthorectifiée
cols = int(np.ceil(max(C[:, 0])))
rows = int(np.ceil(max(C[:, 1])))



# orthorectifier l'image

fig = plt.figure(figsize=(7,4))
plt.subplots_adjust(left=0, bottom=0, right=1, top=1, wspace=0.1, hspace=0)

plt.subplot(111)
ortho_img = cv2.warpPerspective(img, M, (cols, rows))
plt.imshow(ortho_img[:,:,::-1])
plt.axis('off')


## (-0.5, 2085.5, 1516.5, -0.5)
```

```
plt.show()
```

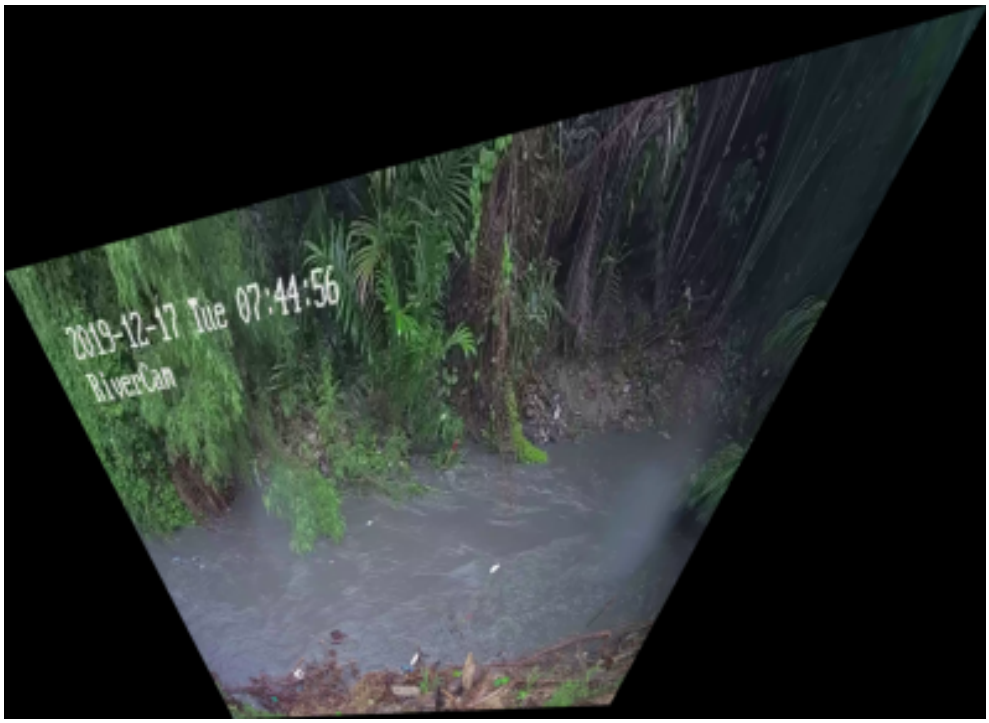


Figure 5: L'orthorectification

La mise en niveau de gris et la correction du gamma et du contraste.

La dernière étape de la préparation de l'image est la conversion du imagerie à une échelle de gris et pour appliquer une correction de contraste et gamma. Une mise à l'échelle des gris est nécessaire pour pouvoir appliquer la validation de similarité entre images vidéo séquentielles. La correction de contraste et gamma est appliquée à améliorer la visibilité des semences. Les corrections de contraste et gamma sont appliqué à l'aide des formules suivantes:

$$O_{constract} = \alpha \cdot I + \beta \tag{6}$$

$$O_{gamma} = \left(\frac{I}{255}\right)^{\frac{1}{\gamma}} \cdot 255 \tag{7}$$

D'où α et β définit la correction du contraste; γ est la correction de gamma; O_n sont les imageries corrigés; et I est l'imagerie d'origine.

```
gamma=0.4
```

```
# transformer l'image en niveaux de gris
Grey_img = cv2.cvtColor(ortho_img, cv2.COLOR_BGR2GRAY)

# appliquer la correction gamma
invGamma = 1./gamma
table = (np.array([((i / 255.0) ** invGamma) * 255
            for i in np.arange(0, 256)]).astype('uint8'))

Gamma_img = cv2.LUT(Grey_img, table)



fig = plt.figure(figsize=(14,4))
plt.subplots_adjust(left=0, bottom=0, right=1, top=1, wspace=0.1, hspace=0)

plt.subplot(121)
plt.imshow(Grey_img,cmap="gray")
plt.title('GRAY SCALE')
plt.axis('off')

## (-0.5, 2085.5, 1516.5, -0.5)

plt.subplot(122)
plt.imshow(Gamma_img, 'gray')
plt.title('GAMMA')
plt.axis('off')

## (-0.5, 2085.5, 1516.5, -0.5)

plt.show()
```

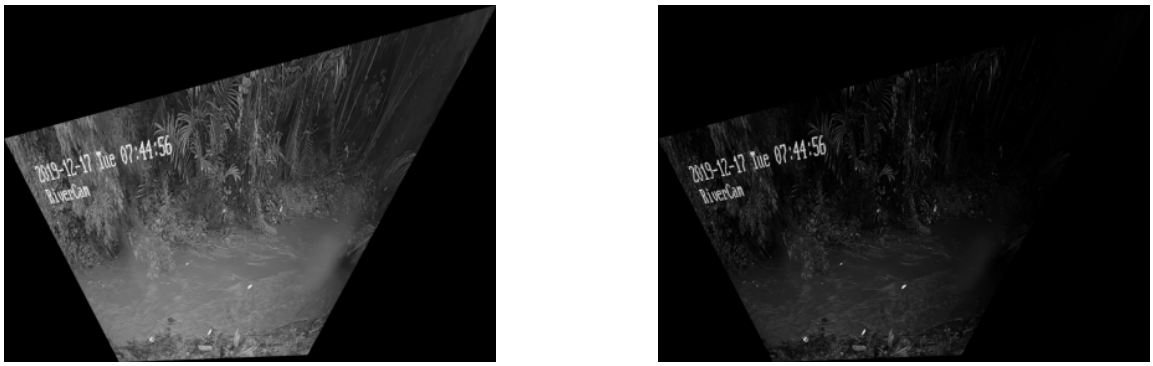


Figure 6: L'orthorectification

Pour tous les images

```
# compteur à utiliser comme nom pour les images individuelles
n = 0
gamma=0.4

# importer une vidéo et lire chaque image individuelle (cadre)
cap = cv2.VideoCapture(dir_video)
while cap.isOpened():
    ret, img = cap.read()
    if ret:

        # appliquer la correction de la distorsion de l'objectif
        img_dist = lens_corr(img, k1=-10.0e-6, c=2, f=8.0)

        # appliquer une orthorectification
        img_orth = cv2.warpPerspective(img_dist, M, (cols, rows))

        # appliquer une échelle de gris, une correction du contraste et du gamma
        img_grey = cv2.cvtColor(img_orth, cv2.COLOR_BGR2GRAY)
        Gamma_img = cv2.LUT(img_grey, table)

        # rogner l'image à une taille plus raisonnable en excluant les limites extérieures
        # l'image pourrait être recadrée dans l'étendue de la zone d'intérêt
        img_out = Gamma_img[max(0, rows-1000):, :min(2000, cols)]
```

```
        # enregistrez l'image avec le nom 'n'
        framename = os.path.join(dir_saves,
                                 'frame_'+str(format(n).zfill(6))+'.jpg')
        cv2.imwrite(framename, img_out)
        n += 1
    else:
        break
cap.release()
```

le traitement PIV

La figure 7 montre les étapes du traitement PIV. Pour deux séquentiels cadres, les images sont divisées en cellules de grille. En déterminant validation de similarité - par exemple, une corrélation croisée ou une rapport signal / bruit [Ran et al., ; Osorio-Cano et al.] - entre les deux cadres de la zone de recherche, les déplacements peuvent être déterminé. Ces déplacements sont ensuite convertis en vitesse d'écoulement vecteurs.

Après avoir appliqué ce processus sur N images, un total de $N - 1$ cartes de vitesse sont créées. Pour chaque carte de vitesse, les résultats peuvent être encore améliorés en appliquant un filtrage supplémentaire basé sur la valeur de similarité dans chaque fenêtre d'interrogation unique, et en remplaçant ces valeurs filtrées par interpolation les cellules de la grille environnantes connues. Ces post-traitements les étapes dépendent du logiciel utilisé ou des résultats requis.

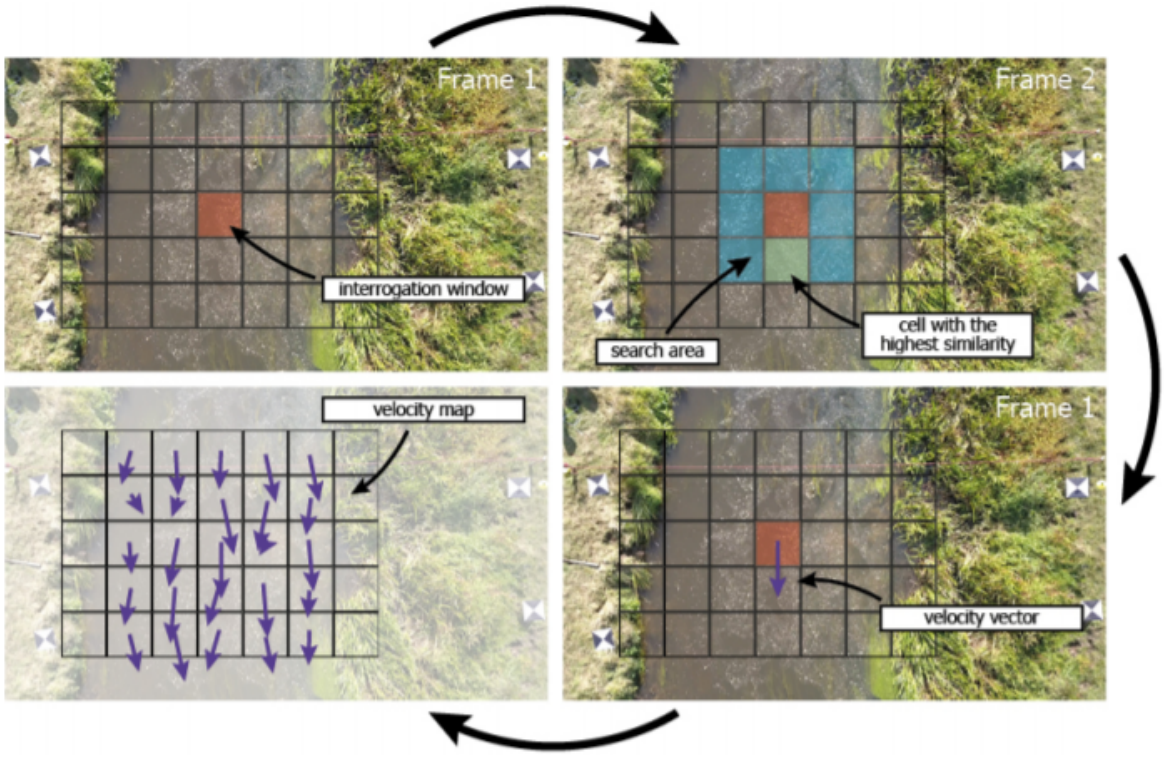


Figure 7: Vue schématique de la méthode LSPIV où une fenêtre d'interrogation est déterminée (la grille est dessinée plus grande dan généralement appliqué) dans la première image et les graines présentes sont comparées à une zone de recherche dans l'image séquentielle 2 à déterminer leurs déplacements. En multipliant le déplacement par la période de temps de trame, la vitesse est déterminée. Lorsque vous appliquez ceci sur toute l'image, une carte de vitesse d'écoulement de surface peut être créée pour chaque image individuelle.

```
import os
import numpy as np
import pandas as pd
from tqdm import tqdm
from openpiv import tools,pyprocess,validation, filters,scaling
import matplotlib.pyplot as plt
```

```
def PIV(
    frame_0,
    frame_1,
    winsize,
    searchsize,
```

```
    overlap,
    frame_rate,
    scaling_factor,
    threshold=1.3,
    output='fil' ):

    # déterminer le pas de temps entre les deux images séquentielles (1 / fps)
    dt = 1./frame_rate

    # estimation des déplacements de graines dans les directions x et y
    # et le rapport signal / bruit correspondant
    u, v, sig2noise = pyprocess.extended_search_area_piv(
                        frame_a = frame_0,
                        frame_b = frame_1,
                        window_size = winsize,
                        overlap = overlap,
                        dt=dt,
                        search_area_size = searchsize,
                        sig2noise_method = 'peak2peak')

    # coordonnées xy du centre de chaque cellule de la grille
    x, y = pyprocess.get_coordinates(image_size=frame_0.shape,
                                     search_area_size=winsize,
                                     overlap=overlap)

    # si la sortie est 'fill' ou 'int':
    # filtrer les cellules de la grille avec un faible rapport signal / bruit
    if output == 'fil' or  output == 'int':
        u, v, mask = validation.sig2noise_val(u,
                                              v,
                                              sig2noise,
                                              threshold=threshold)

    # si la sortie est 'int'
    # remplir les valeurs manquantes par interpolation
    if output == 'int':
        u, v = filters.replace_outliers(u,
                                        v,
                                        method='localmean',
```

```
                                        max_iter=50,
                                        kernel_size=3)

    # mettre à l'échelle les résultats en fonction des pixels par mètre
    x, y, u, v = scaling.uniform(x,
                                 y,
                                 u,
                                 v,
                                 scaling_factor=scaling_factor)

    return x, y, u, v, sig2noise
```

```
dir_frames = 'Python/frames'
dir_saves = 'Python/files'

if not os.path.exists(dir_saves):
    os.mkdir(dir_saves)
```

```
winsize =  60             # pixels, taille de la fenêtre d'interrogation dans l'image A
searchsize = winsize      # pixels, recherche dans l'image B
overlap =  30             # pixels, chevauchement de 50%
frame_rate = 25           # vidéo à fréquence d'images
scaling_factor = 100      # pixels par mètre (PPM)
threshold = 1.3           # rapport signal / bruit auquel les résultats sont filtrés
```

```
# importer les noms et le nombre de cadres
frames = os.listdir(dir_frames)
N = len(frames)-1

for n in tqdm(range(N)):
    # définir les deux trames séquentielles utilisées pour estimer le champ de vitesse
    frame_0 = tools.imread(os.path.join(dir_frames,str(frames[n])))
    frame_1 = tools.imread(os.path.join(dir_frames,str(frames[n+1])))

    # Traitement PIV
    x, y, u, v, sig2noise = PIV(frame_0, frame_1, winsize, searchsize,
                                overlap, frame_rate, scaling_factor,
                                threshold, output='fil')

    # enregistrer les résultats dans un fichier texte
```

```
    tools.save(x,
               y,
               u,
               v,
               sig2noise,
               os.path.join(dir_saves, str(format(n).zfill(6))+'.txt'))
```

l'analyse des donnée

Ce fichier traite deux étapes de post-traitement pour acquérir des vitesses d'écoulement de surface à une section transversale spécifique. Les étapes de post-traitement sont nommées filtrage et substitution . Les deux processus sont expliqués dans les sections suivantes. Pour ces étapes, plusieurs informations supplémentaires sont nécessaires à savoir:

- bathymétrie locale,
- niveau d'eau,
- paramètres de progression verticale des vitesses d'écoulement de surface.

Dans les sections à venir, traiter les sujets suivants: (1) les résultats non traités, (2) le filtrage des vitesses d'écoulement inférieures, et (3) la substitution des vitesses d'écoulement en cas de manque de graines.

```
import os
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy import interpolate
```

```
dir_files = 'Python/files'
bat = r'Python/bathymetry.csv'
```

Résultats non traités

la zone d'intérêt se situe à 1 mètre à gauche et à trois mètres à droite du point central de la bathymétrie (fournie dans le fichier bathymetry.csv). Comme l'écoulement va de gauche à droite, seules les composantes x des vitesses d'écoulement sont utilisées pour estimer les vitesses d'écoulement moyennes.

La largeur du ruisseau est divisée en différentes sections transversales. Pour chaque section, les vitesses d'écoulement sont rassemblées et une valeur moyenne est déterminée.

Tout d'abord, certaines caractéristiques de base sont prouvées. L'emplacement du centre de la bathymétrie dans l'imagerie ($centre_x$ et $centre_y$), les emplacements des berges du cours d'eau à la bathymétrie (y_0 et y_1), et le niveau d'eau (w_l) pendant la vidéo.

```
# localisation de la bathymétrie du centre dans l'imagerie (mètres).
# Utilisé pour aligner les vitesses d'écoulement avec la bathymétrie
centre_x, centre_y = [8.3471579 , 2.01868403]

# emplacements banques de flux
y0, y1 = [-1.397, 3.785]

# niveau d'eau (par rapport au point bathymétrique le plus bas)
wl = 0.9
```

La bathymétrie se trouve dans le fichier **bathymetry.csv**. Pour utiliser cette bathymétrie, les points sont interpolés à l'aide de la fonction scipy.interpolate.interpolate.

```
# importer la bathymétrie locale
df_bat = pd.read_table(bat, sep=';', usecols=['Y', 'H'])
# fonction interpolée de la bathymétrie
func_bat = interpolate.interp1d(df_bat['Y'], df_bat['H'], kind='quadratic')



plt.figure(figsize= (12,4))
plt.scatter(df_bat.Y,df_bat.H,alpha=0.5,label ="Mesures individuelles")
plt.plot(df_bat.Y,func_bat(df_bat.Y),color ="black", alpha=0.3,label= "Bathymétrie interpolée")
plt.xlabel('Y [m]')
plt.ylabel('Z [m]')
plt.legend(loc = 'lower right')
plt.show()
```

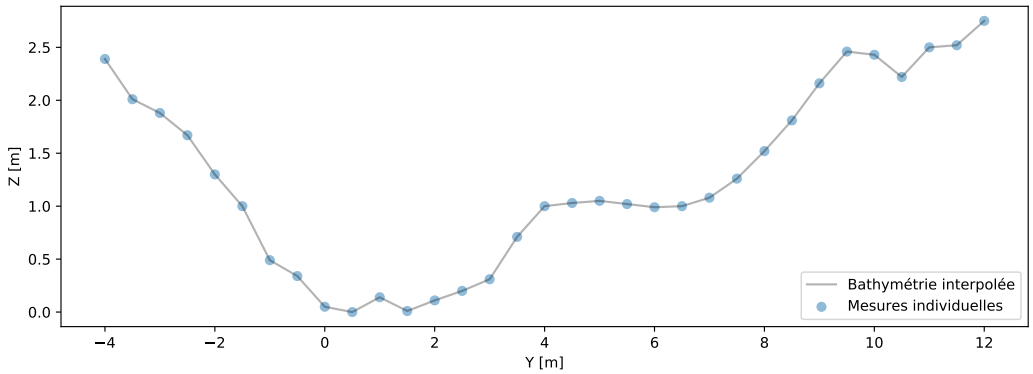


Figure 8: la bathymétrie locale

```
# extraire les noms des fichiers texte
files = os.listdir(dir_files)

# créer un dataframe dans lequel toutes les vitesses de la zone d'intérêt sont stockées
vx_all = pd.DataFrame()

# créer un tableau pour enregistrer les vitesses moyennes
vx_mean_raw = []

for file in files:
    # importer un fichier unique et ajouter des vitesses dans la zone d'intérêt
    # à la trame de données
    df = pd.read_table(os.path.join(dir_files, file),
                       sep='\s+',
                       names=('X', 'Y', 'Vx', 'Vy', 's2n'),
                       header=0,
                       index_col= False)
    df['X']=df['X'].astype('float')
    df['Y']=df['Y'].astype('float')
    df['Vx']=df['Vx'].astype('float')
    df['Vy']=df['Vy'].astype('float')
    df = df[df['X'] > (centre_x - 1)]
    df = df[df['X'] < (centre_x + 3)]
    vx_all = vx_all.append(df)

# définir les différentes coordonnées y (sections de largeur de flux)
```

```
y_unique = np.sort(df.Y.unique())

# corriger les coordonnées y par rapport au 'centre' bathymétrique
y_corrected = y_unique - centre_y
```

Filtrage

Comme la densité d'ensemencement n'est pas assez dense pour avoir des graines à chaque cellule de la grille, il y a des moments où aucune vitesse d'écoulement de surface n'est estimée, ce qui entraîne une large gamme de vitesses d'écoulement. Par conséquent, un filtrage supplémentaire est appliqué pour supprimer les estimations de vitesse d'écoulement inférieure.

Ce filtrage est basé sur la distance d'une mesure par rapport à la vitesse d'écoulement du 95e percentile dans cette section transversale. Si une valeur est plus éloignée que deux fois l'écart type du 95er centile, cette valeur est filtrée. Ou en bref, le filtrage est appliqué lorsque l'instruction suivante est remplie:

$$V_X < V_{q95} - 2 \cdot \sigma_{V_X} \tag{8}$$

Comme cette déclaration peut entraîner un filtrage excessif, une autre instruction (ou seuil) doit être satisfaite avant que le filtrage ne soit appliqué à une section transversale. Le 95e centile doit être supérieur à 2,5 fois l'écart type:

$$V_{q95} > 2.5 \cdot \sigma_{V_X} \tag{9}$$

Les nouvelles estimations des vitesses d'écoulement moyennes peuvent être faites.

```
# tableau pour stocker les vitesses d'écoulement moyennes
vx_mean_fil = []

# pour chaque section transversale
# trouver les vitesses d'écoulement correspondantes et supprimer les valeurs NaN
for yy in y_unique:
    vxi = vx_all.Vx[vx_all.Y == yy]
    vxi = vxi[np.isfinite(vxi)]

    # si toutes les valeurs sont NaN, créez un tableau de longueur 1
    if len(vxi) == 0:
        vxi = [0]
```

```
    # déterminer le 95e quantile et l'écart type
    v_filter = np.quantile(vxi, 0.95)
    std = np.std(vxi)

    # le filtrage est appliqué si le 95e centile> 2,5 * std
    if 2.5*std/v_filter < 1:
        # le filtrage est appliqué sur les valeurs plus éloignées que 2 * std du 95e centile
        vxi_in = vxi[vxi > v_filter - 2*std]
        vxi_out = vxi[vxi < v_filter - 2*std]
        # ajouter une valeur moyenne en fonction des résultats filtrés
        vx_mean_fil.append(np.mean(vxi_in))

    # si le filtrage n'est pas appliqué: déterminer la moyenne sur toutes les valeurs
    else:
      vx_mean_fil.append(np.mean(vxi))
```

Substitution

Comme dans certaines sections transversales aucune graine n'est présente, les vitesses d'écoulement sont à coup sûr sous-estimées à ces endroits. Pour encore faire une estimation éclairée des vitesses d'écoulement à ces endroits, les vitesses d'écoulement à différents stades de l'onde de crue sont estimées et utilisées pour établir une relation entre les vitesses d'écoulement de surface et la profondeur de l'eau. Cette relation est approchée en utilisant la loi logarithmique de Prandtl-von Kármán:

$$V_{x_sub}(h) = \frac{u_\star}{\kappa} \ln\left[\frac{h-d}{h_0}\right] \tag{10}$$

Dans cette formule $V_{x_sub}(h)$ est la vitesse d'écoulement à la profondeur de l'eau h; $u_\star$ est la vitesse de cisaillement;κ est la constante de von Kármán (≈ 0.41); h_0 est la profondeur de rugosité; d est le déplacement dans le plan zéro. Pour l'exemple de vidéo, $u_\star$, h_0 et d sont estimés à 0.235 , 0.054 et 0.15.

Les vitesses d'écoulement de surface sont remplacées si la vitesse d'écoulement de surface est la moitié de la valeur de substitution et lorsque la section se trouve dans les berges du cours d'eau.

```
def vertical_flow(loc_wd, p, d=0.15):
    us = p[0]
    h0 = p[1]
```

```
    return us/0.41 * np.log((loc_wd - d) / h0)

# paramètres de vitesse d'écoulement de surface de progression verticale
param = [0.235, 0.054]
```

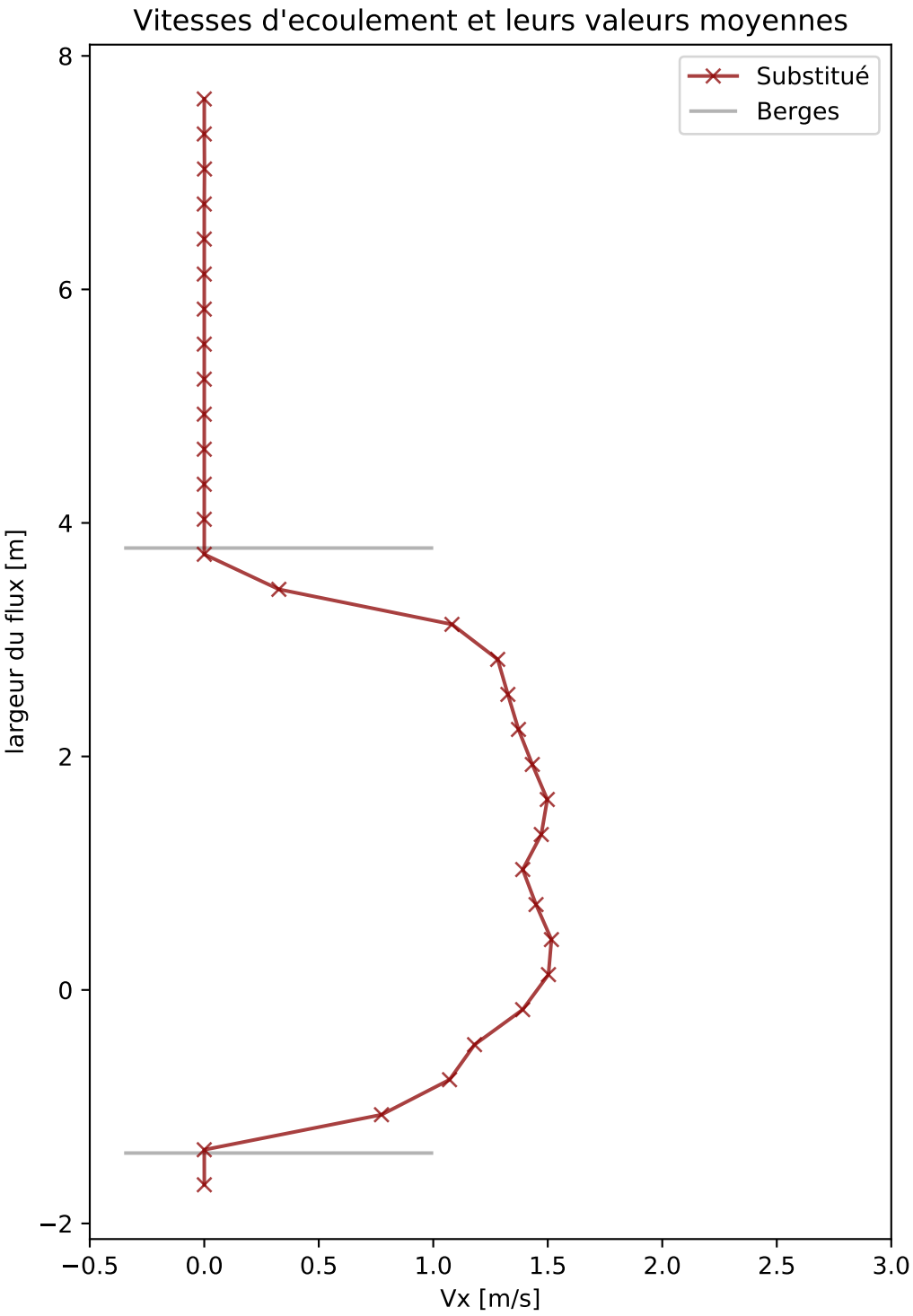
```
# créer une copie des vitesses d'écoulement moyennes filtrées
vx_mean_rep = vx_mean_fil.copy()

# déterminer les profondeurs d'eau à différentes coordonnées y corrigées
# s'il y a de l'eau (wd> 0) et si y est compris entre -3 et 3 (pas de buissons)
# si moyenne * 2 <valeur trouvée en utilisant le profil vertical
wd = wl - func_bat(y_corrected)
for nn, loc_wd in enumerate(wd):
    if loc_wd > 0.15:
        if (y_corrected[nn] > -3.5) and y_corrected[nn] < 3.5:
            # si 2 * la valeur médiane est inférieure à la valeur corrigée: valeur correcte
            # tous les centiles
            if vx_mean_rep[nn] < 0.5*vertical_flow(loc_wd, param):
                    vx_mean_rep[nn] = vertical_flow(loc_wd, param)
```

```
# afficher les vitesses individuelles et les vitesses moyennes des différentes étapes
# de post-traitement
fig = plt.figure(figsize=(6,9))
plt.plot(vx_mean_rep, y_corrected,
         color='darkred', marker='x', zorder=2, alpha=0.75, label='Substitué')
plt.hlines([y0, y1], xmin=-0.35, xmax=1, color='k', alpha=0.3, label='Berges')
plt.title("Vitesses d'ecoulement et leurs valeurs moyennes")
plt.xlabel('Vx [m/s]')
plt.ylabel('largeur du flux [m]')
plt.xlim(-.5, 3)
```

```
## (-0.5, 3.0)
```

```
plt.legend()
plt.show()
```


Vitesses d'ecoulement et leurs valeurs moyennes
Substitué
Berges
largeur du flux [m]
Vx [m/s]
8
6
4
2
0
−2
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0

Estimation de décharge

Les rejets sont estimés en utilisant la méthode de la surface de vitesse, en utilisant la formule suivante:

$$Q = \sum_{n=1}^{N} v_n \cdot d_n \cdot b_n \tag{11}$$

D'où v_n , d_n, et b_n sont respectivement la vitesse d'écoulement, la profondeur et la largeur moyennes en profondeur de la section.

Pour la vitesse d'écoulement moyenne en profondeur, la vitesse d'écoulement de surface est multipliée par un coefficient déterminé empiriquement α. Ce coefficient est généralement entre 0.72 et 0.95. Pour l'exemple de vidéo, le coefficient est estimé à 0.85.

```
# coefficient moyen en profondeur
alpha = 0.85

# sections transversales en largeur
step = y_corrected[1]-y_corrected[0]

# estimation de débit
Q = sum(vx_mean_rep * wd  * alpha * step)

print("Le débit moyen est estimé à {} m³/s".format(round(Q, 2)))
```

```
## Le débit moyen est estimé à 3.74 m³/s
```

RÉFERENCES

OpenCV 3.0. URL https://opencv.org/opencv-3-0/.

John Fryer and Duane Brown. Lens distortion for close-range photogrammetry. 52:51–58.

M. Jodeau, A. Hauet, A. Paquier, J. Le Coz, and G. Dramais. Application and evaluation of LS-PIV technique for the monitoring of river surface velocities in high flow conditions. 19(2):117–127. ISSN 09555986.

Juan David Osorio-Cano, Andrés F. Osorio, and Raul Medina. A method for extracting surface flow velocities and discharge volumes from video images in laboratory. 33:188–196. ISSN 0955-5986. doi: 10.1016/j.flowmeasinst.2013.07.009.

Qi-hua Ran, Wei Li, Qian Liao, Hong-lei Tang, and Meng-yao Wang. Application of an automated LSPIV system in a mountainous stream for continuous flood flow measurements. 30(17):3014–3029. ISSN 1099-1085.