# Day 4: Python Boot Camp

## Introduction to Motion Tracking using Optical Flow

**Please make your own copy of this notebook before making changes or beginning the exercises!**

## Agenda

1. **2D Optical Flow overview**
2. **2D Optical Flow on synthetic data**
3. **2D Optical Flow on real microscopic data**
4. **Limitations**
5. **Exercises (playing with the parameters and use your own data)**

## Installing and importing libraries

```
# !pip install IPython
# !pip install opencv-python-headless opencv-contrib-python-headless ipywidgets tifffile

from IPython.display import Image
import matplotlib.pyplot as plt
from matplotlib import animation
from tqdm import tqdm
import os; import glob; import cv2
import numpy as np
import scipy as sp
import matplotlib
from scipy import ndimage
from scipy import misc
import scipy.io as sio
import gdown
import imageio
import skimage.io as io
from matplotlib import rc
rc('animation', html='jshtml')
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)     # ignore warnings for specific matplotlib commands which w
from moviepy.video.io.bindings import mplfig_to_npimage
matplotlib.rcParams['animation.embed_limit'] = 2**128            # to set a higher threshold for maximum .gif size that we c
from IPython.display import display, Image
import time
import tifffile as tiff
from natsort import natsorted
from ipywidgets import interactive
from skimage import filters, feature
from multiprocessing import TimeoutError
from multiprocessing.pool import ThreadPool as Pool
from functools import partial
warnings.filterwarnings("ignore")


from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

**HTML link to visualize the raw taxi sequence:**

Credits: Sandipan Dey

```
mv_link = r"https://sandipanweb.files.wordpress.com/2018/02/taxi.gif?w=555&zoom=2"
display(Image(url=mv_link,width=600,height=600))
```

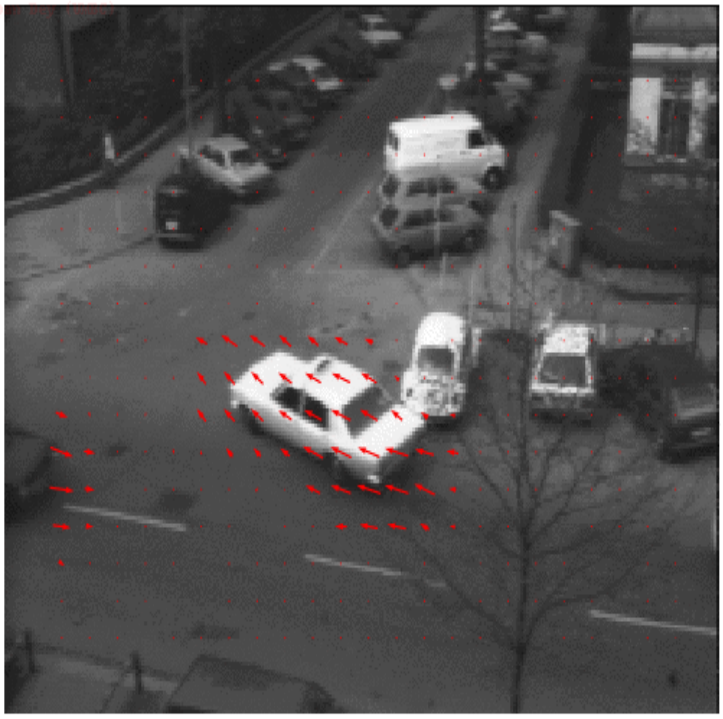
**HTML link to download some biological data (crayfish gut motion)**
Drive link

## 1. 2D Optical Flow overview

**Q: What is Optical Flow?**
A: Optical flow is the pattern of **apparent motion of image objects** between **two consecutive frames** caused by the movement of object or camera. It can also be defined as the distribution of apparent velocities of movement of brightness pattern in an image.

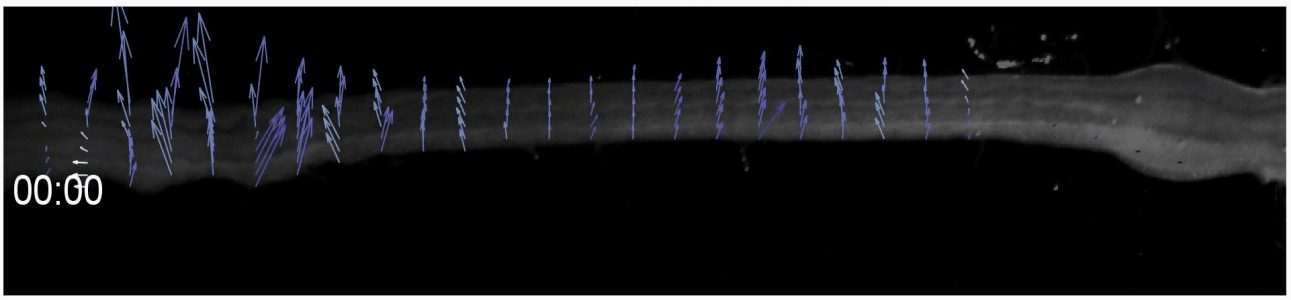
**Example-1: Traffic cam footage called '*Hamburg taxi sequence*'**
Image Credits: Sandipan Dey

```
mv_link = r"https://sandipanweb.files.wordpress.com/2018/02/taxi_opt.gif?w=555&zoom=2"
display(Image(url=mv_link,width=600,height=600))
```



**Example-2: A crayfish gut moving while outside the body**

```
mv_link = r"https://github.com/pathakspandan/Gut_Dynamics-/blob/main/movie_timed_v2.gif?raw=true"
display(Image(url=mv_link,width=1100,height=300))

```




Blue and red represents upward and downward movements respectively. The length of the vectors indicate the magnitude of motion.

## Optical Flow methods

- **Lucas-Kanade algorithm** : Assumes that the **displacement** of the image contents between two nearby instants (frames) is **small** and **approximately constant** within a neighborhood of the point.
- **Horn-Schunck algorithm** : Assumes **smoothness in the flow** over the **whole image**. Thus, it tries to minimize distortions in flow and prefers solutions which show more smoothness.
- **Buxton and Buxton's algorithm** : Estimates optical flow is based on a model of the **motion of edges** in image sequences.
- **Black-Jepson algorithm** : Computes a **maximum likelihood estimate** for the various motion parameters within a patch.

Most of these algorithms are based on the '**equation of brightness constancy**'. It assumes that at each pixel location of an image, any fluctuation in intensity between successive frames is solely due to object motion.

In other words, a point at location $(x, y)$ and time $t$ will have moved by $(\Delta x, \Delta y)$ in the short time interval $\Delta t$ and its brightness $I$ remains the same, i.e.

$$\mathbf{I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t)}$$

Using Taylor's expansion we can obtain the following equation with 2 unknowns ($u$ and $v$) for two velocity components:

```
mv_link = r"https://github.com/pathakspandan/Gut_Dynamics-/blob/main/optical_flow1.png?raw=true"
display(Image(url=mv_link,width=800,height=350))
```

$$I_x u + I_y v + I_t = 0$$

$I_x = \frac{\partial I}{\partial x} \quad I_y = \frac{\partial I}{\partial y}$ **spatial derivative**

$u = \frac{dx}{dt} \quad v = \frac{dy}{dt}$ **optical flow**

$I_t = \frac{\partial I}{\partial t}$ **temporal derivative**

We are trying to solve for $u$ and $v$ here for each pixel location. Therefore we have an **underdetermined** system at hand! (fewer equations, more unknowns)

Solution: For **Lucas-Kanade method**, we will assume that **flow is locally smooth**. Hence, the neighborhood pixels will have the same displacement.

For example, assuming uniform flow in a $5 \times 5$ image patch will give us $25$ equations and $2$ unknowns for the two velocity components. The last step is to solve this **overdetermined system** (more equations, fewer unknowns).

## 2. 2D Optical Flow on synthetic data

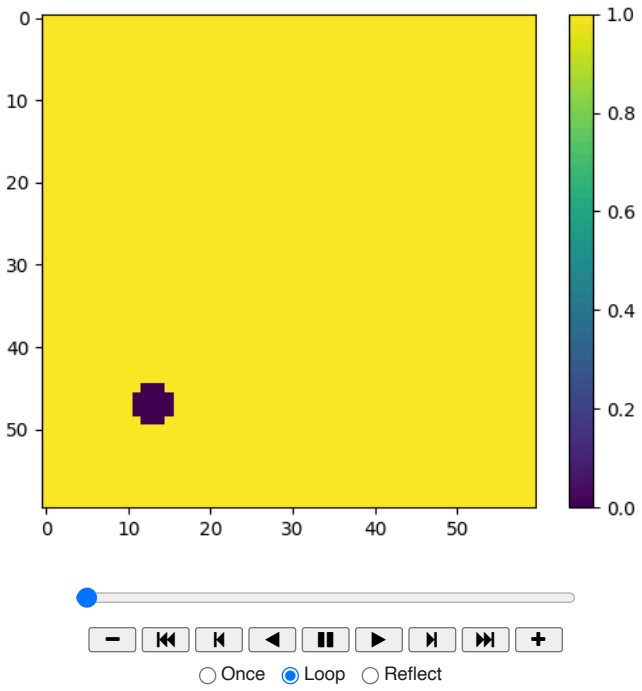
**Creating movies with synthetic data:**
***Example-1: Single particle linear motion***

```
sx_in = 17; sy_in = -17;    # Initial location of the particle
sig = .25;
n_fr = 50       # Number of frames in the movie
data = np.zeros([60,60,n_fr])   # Memory allocation for the synthetic data

for i in tqdm(range(n_fr)):
    sx = sx_in - i;           # shift the particle in x-direction by 1
    sy = sy_in + i;           # shift the particle in y-direction by 1

    [x,y] = np.meshgrid(np.arange(-30+sx,30+sx,1), np.arange(-30+sy,30+sy,1)) # Creates a 2D grid
    r = (x**2 + y**2)**(.5)/sig      # Defines a circle

    data[:,:,i] = r > 10           # Defines a boundary mask for the particle
```

```
100%|██████████| 50/50 [00:00<00:00, 651.74it/s]
```

**Visualizing the data**

```
fig, ax = plt.subplots()            # create plotting object (higher dpi value creates a larger image)
im = ax.imshow(np.squeeze(data[:, :, 0]))

cbar = fig.colorbar(im, ax=ax)             # creates a colorbar

def update(i):                             # update function that animates the movie taking successive frames
    im.set_data(data[:, :, i])

ani = animation.FuncAnimation(fig, update, frames=data.shape[2], interval=40)
plt.close(fig)
ani
```




**Optical Flow function (Lucas-Kanade method):**

Reference: Lucas BD, Kanade T. An iterative image registration technique with an application to stereo vision | Proceedings of the 7th international joint conference on Artificial intelligence - Volume 2. Proceedings of the 7th International Joint Conference on Artificial Intelligence. 1981:pp 674-9.

```
# INPUT:
# img1, img2: Successive time frames (img2 is the latter frame)
# sig: Size of the Gaussian filter for smoothing the image gradients
# thresh: Reliability score threshold (higher value would indicate less flow vectors would satisfy the criteria)


# OUTPUT:
# vx: x-component of velocity
# vy: y-component of velocity
# reliabMat: Reliability score for velocity at each pixel location


def LKxOptFlow(img1,img2,sig,thresh):

    # Obtaining spatio-temporal image gradients
    ddy = np.gradient(img1,axis=0,edge_order=2)
    ddx = np.gradient(img1,axis=1,edge_order=2)
    dt = img2 - img1

    wdx2 = sp.ndimage.gaussian_filter(ddx**2, sig, mode='nearest')
    wdy2 = sp.ndimage.gaussian_filter(ddy**2, sig, mode='nearest')
    wdxy = sp.ndimage.gaussian_filter(ddx*ddy, sig, mode='nearest')
    wdtx = sp.ndimage.gaussian_filter(ddx*dt, sig, mode='nearest')
    wdty = sp.ndimage.gaussian_filter(ddy*dt, sig, mode='nearest')

```

```
    trace = wdx2 + wdy2; determinant=(wdx2*wdy2)-(wdxy**2); eps = np.finfo(float).eps

    e1=(trace + np.sqrt(eps + trace**2 - 4*determinant))/2;
    e2=(trace - np.sqrt(eps + trace**2 - 4*determinant))/2;

    # Ensuring roots are real (due to properties of S.P.D matrix)
    # if (np.amin(eps + trace**2 - 4*determinant) < 0):
    #     print ('Error!',np.amin(eps + trace**2 - 4*determinant))

    reliabMat = np.minimum(e1,e2);    # Realiabilty scores for each flow vector

    # Obtaining the x and y component of velocities
    vx = ((determinant + eps)**(-1))*((wdxy*wdty)-(wdy2*wdtx));
    vy = ((determinant + eps)**(-1))*((wdxy*wdtx)-(wdx2*wdty));

    # Only selecting flow vectors above the reliability threshold
    vx = vx*(reliabMat > thresh);
    vy = vy*(reliabMat > thresh);
    return vx, vy, reliabMat
```

**Decluttering the vector field (for visualization purpose)**

This gets rid of a lot of excess flow vectors. Otherwise the figure becomes too congested!

```
# INPUT:
# vx: x-component of velocity field
# vy: y-component of velocity field
# s: Number of successive vectors to skip (for visualization)


# OUTPUT:
# vxn: decluttered x velocity field
# vyn: decluttered y velocity field

def declutter_quiver(vx,vy,s):
    nx,ny = vx.shape
    vxn = np.copy(vx); vyn = np.copy(vy)
    vxn[np.mod(np.arange(0,nx,1),s)!=0,:] = 0
    vyn[np.mod(np.arange(0,nx,1),s)!=0,:] = 0
    vxn[:,np.mod(np.arange(0,ny,1),s)!=0] = 0
    vyn[:,np.mod(np.arange(0,ny,1),s)!=0] = 0
    return vxn,vyn
```

**Performing Optical Flow on all time-frames**

```
leng = len(data)

# Memory allocation for storing optical flow results
vx_data = np.zeros([leng,leng,n_fr-1])
vy_data = np.zeros([leng,leng,n_fr-1])
vel_data = np.zeros([leng,leng,n_fr-1])
rel_data = np.zeros([leng,leng,n_fr-1])

sigma = 1.5        # smoothing parameter for the inbuilt Gaussian filter
rel_fac = 0.02     # Reliability score threshold

for k in tqdm(range(n_fr-1)):

    I1 = data[:,:,k]         # loading the initial frame of a pair
    I2 = data[:,:,k+1]       # loading the next frame of a pair

    vx,vy,rel = LKxOptFlow(I1,I2,sigma,rel_fac)     # obtaining the vector fields

    vx_data[:,:,k] = vx; vy_data[:,:,k] = vy; rel_data[:,:,k] = rel;
    vel_data[:,:,k] = np.sqrt(vx**2 + vy**2)        # define magnitude of motion
```

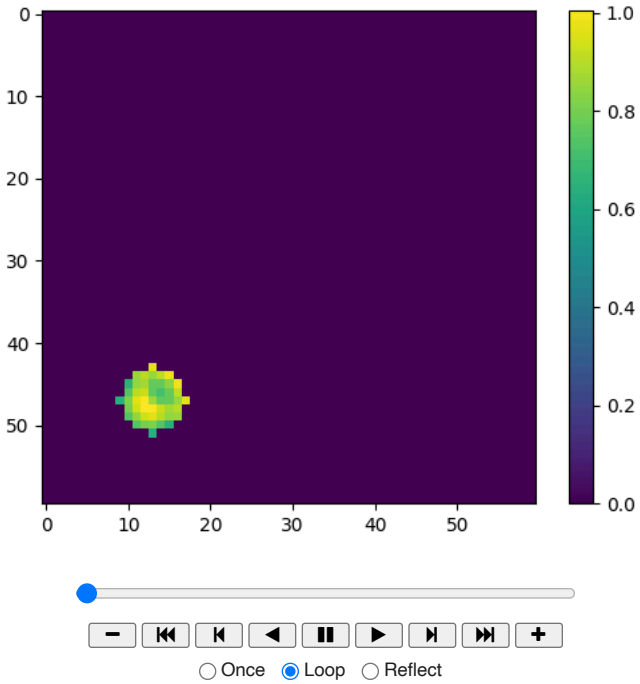
```
100%|██████████| 49/49 [00:00<00:00, 600.03it/s]
```

**Visualizing the velocity magnitudes from Optical Flow results**

Units are in #pixels/frame.

```
fig, ax = plt.subplots()

```

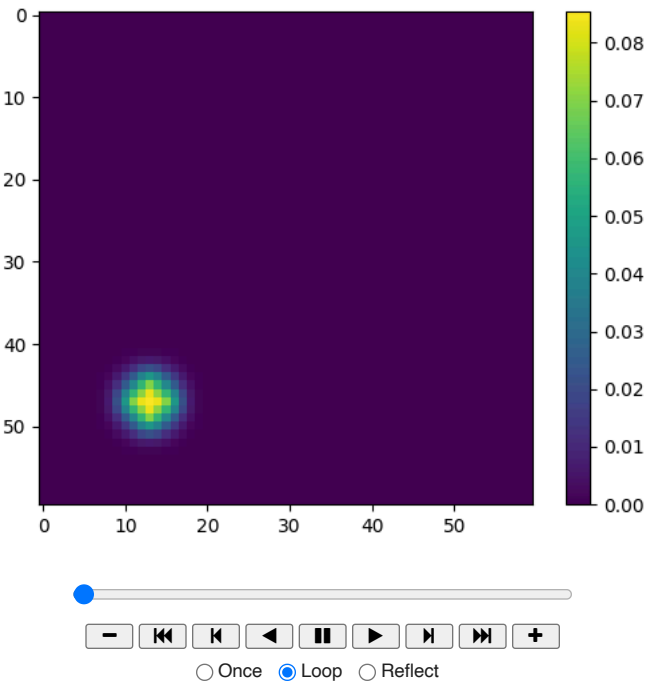
```
im = ax.imshow(np.squeeze(vel_data[:, :, 0]))

cbar = fig.colorbar(im, ax=ax)   # creates a colorbar

def update(i):                    # update function that animates the movie taking successive frames
    im.set_data(vel_data[:, :, i])

ani = animation.FuncAnimation(fig, update, frames = vel_data.shape[2], interval=40)
plt.close(fig)
ani                              # starts the animation
```




**Visualizing the reliability score from Optical Flow results (good sanity check)**

Ideally, it should overlap with the location of the movement. If not, we have a problem here!

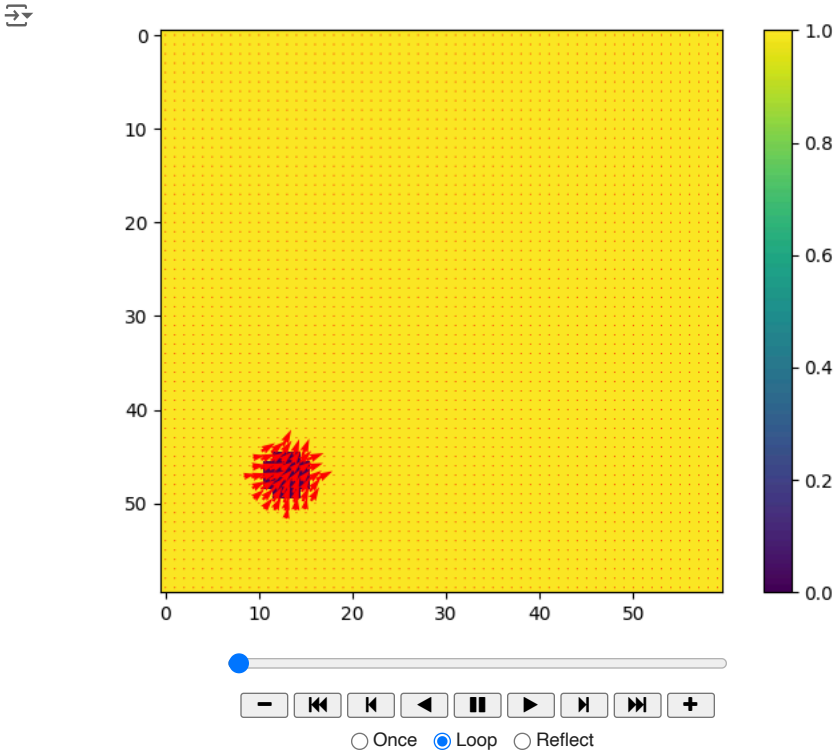
```
fig, ax = plt.subplots()
im = ax.imshow(np.squeeze(rel_data[:, :, 0]))

cbar = fig.colorbar(im, ax=ax)              # creates a colorbar

def update(i):                              # update function that animates the movie taking successive frames
    im.set_data(rel_data[:, :, i])

ani = animation.FuncAnimation(fig, update, frames = rel_data.shape[2], interval=40)
plt.close(fig)
ani
```




**Visualizing the vector field over all time frames**

We want to overlay the vector field on top of the raw images.

```
[X,Y] = np.meshgrid(np.arange(0,60,1), np.arange(0,60,1))             # Creates a 2D grid
U = np.squeeze(vx_data[:,:,0]); V = np.squeeze(vy_data[:,:,0])        # reshape the vector fields into 2D (in case they had th

fig, ax = plt.subplots()
c = ax.imshow(np.squeeze(data[:, :, 0]), origin='upper')              # shows the raw image
im = ax.quiver(X,Y,U,V,angles='xy',color='red',minshaft=1,minlength = 0.5, \
           pivot='middle',scale=30,width=0.005)                       # plots the vector field

cbar = fig.colorbar(c, ax=ax)

def update_quiver(num, im, X, Y):                                     # update function for the animation
    U = np.squeeze(vx_data[:,:,num]);
    V = np.squeeze(vy_data[:,:,num])
    c.set_data(data[:, :, num])
    im.set_UVC(U,V)
    return im,

anim = animation.FuncAnimation(fig, update_quiver, fargs=(im, X, Y),
                               interval=30.0, frames=n_fr-1, blit=False)
fig.tight_layout()
plt.close(fig)
anim                                                                  # starts the animation
```




**Decluttering the vector fields**

To reduce the number of vectors overlaid on top of the particle.

```
# Memory allocation for storing modified vector fields

vx_dc_data = np.zeros([60,60,n_fr-1]);
vy_dc_data = np.zeros([60,60,n_fr-1]);

n_jump = 2;     # Number of vectors to skip on each row and column

for i in tqdm(range(n_fr-1)):
    vx_frame = vx_data[:,:,i]; vy_frame = vy_data[:,:,i];
    [vx_dc,vy_dc] = declutter_quiver(vx_frame,vy_frame,n_jump)
    vx_dc_data[:,:,i] = vx_dc; vy_dc_data[:,:,i] = vy_dc;
```

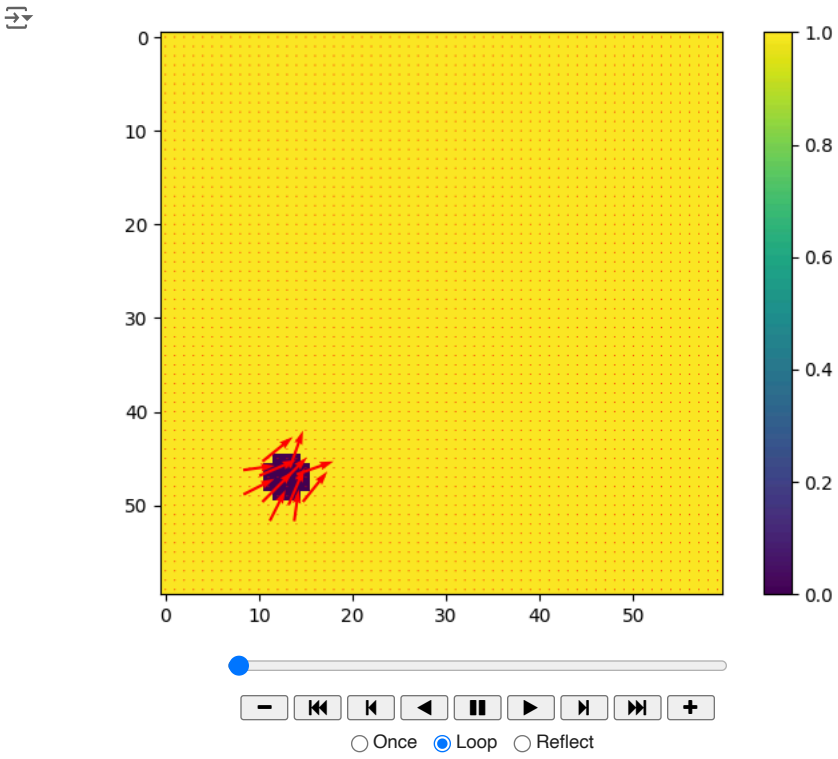
```
100%|██████████| 49/49 [00:00<00:00, 2372.81it/s]
```

**Visualizing decluttered vector fields**

Note: We can also go for 3 or 4 vectors for each instance.

```
[X,Y] = np.meshgrid(np.arange(0,60,1), np.arange(0,60,1))
U = np.squeeze(vx_dc_data[:,:,0]); V = np.squeeze(vy_dc_data[:,:,0])

fig, ax = plt.subplots()
c = ax.imshow(np.squeeze(data[:, :, 0]), origin='upper')    # shows the raw image
im = ax.quiver(X,Y,U,V,angles='xy',color='red',minshaft=1,minlength=0.5, \
           pivot='middle',scale=13,width=0.005)             # plots the vector field


## Most important QUIVER parameters

# minlength: arrows below this length would be shows as a tiny dot
# scale: scale for the arrows (smaller number would indicate much larger arrows)
# width: width of the arrows (higher number would mean thicker arrows)

cbar = fig.colorbar(c, ax=ax)

def update_quiver(num, im, X, Y):            # Update function for the animation
    U = np.squeeze(vx_dc_data[:,:,num]);
    V = np.squeeze(vy_dc_data[:,:,num]);

    c.set_data(data[:, :, num])
```

```
    im.set_UVC(U,V)
    return im,

anim = animation.FuncAnimation(fig, update_quiver, fargs=(im, X, Y),
                               interval=30, frames=n_fr-1, blit=False)
fig.tight_layout()
plt.close (fig)
anim
```

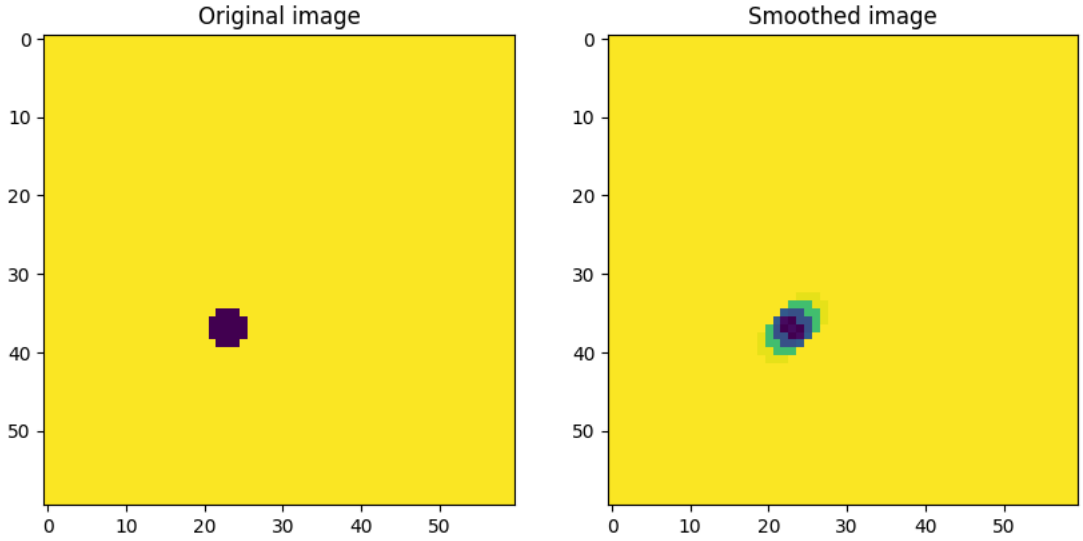



- **Check the effect of Temporal Smoothing**

+ Code + Text

```
start = time.time()

w1 = 0.036; w2 = 0.249; w3 = 0.431;

# smoothing the image frames
smo_data = w1*data[:,:,0:-4] + w2*data[:,:,1:-3] + w3*data[:,:,2:-2] + w2*data[:,:,3:-1] + w1*data[:,:,4:]
fin_im = np.concatenate((data[:,:,0:1],data[:,:,1:2],smo_data,data[:,:,-2:-1],data[:,:,-1:]),axis = 2)   # padding the borderi

end = time.time(); print('Run time = '+str(end-start)+' s')

# check out the a frame of the smoothened movie
fig, ax = plt.subplots(1,2,figsize=(10,5))
fr_id = 10;
ax[0].imshow(data[:,:,fr_id])
ax[0].set_title('Original image')

ax[1].imshow(fin_im[:,:,fr_id])
ax[1].set_title('Smoothed image')
plt.show()
```

```
Run time = 0.008054733276367188 s
```




- **Peforming and visualizing vector fields from the smoothed data**

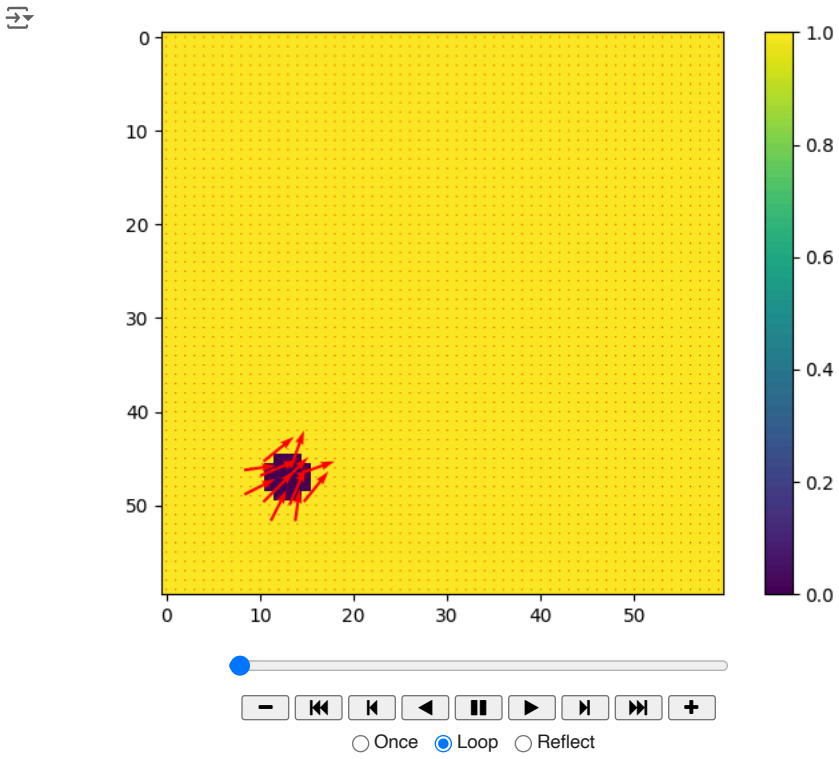
```
leng = len(fin_im)

# Memory allocation for storing optical flow results
vx_smo_data = np.zeros([leng,leng,n_fr-1])
vy_smo_data = np.zeros([leng,leng,n_fr-1])
vel_smo_data = np.zeros([leng,leng,n_fr-1])
rel_smo_data = np.zeros([leng,leng,n_fr-1])

sigma = 1.5        # smoothing parameter for the inbuilt Gaussian filter
rel_fac = 0.02     # Reliability score threshold

for k in tqdm(range(n_fr-1)):

    I1 = fin_im[:,:,k]         # loading the initial frame of a pair
    I2 = fin_im[:,:,k+1]       # loading the next frame of a pair

    vx,vy,rel = LKxOptFlow(I1,I2,sigma,rel_fac)     # obtaining the vector fields

    vx_smo_data[:,:,k] = vx; vy_smo_data[:,:,k] = vy; rel_smo_data[:,:,k] = rel;
    vel_smo_data[:,:,k] = np.sqrt(vx**2 + vy**2)        # define magnitude of motion
```

```
100%|██████████| 49/49 [00:00<00:00, 663.01it/s]
```

```
# Memory allocation for storing modified vector fields

vx_dc_smo_data = np.zeros([60,60,n_fr-1]);
vy_dc_smo_data = np.zeros([60,60,n_fr-1]);

n_jump = 2;     # Number of vectors to skip on each row and column

for i in range(n_fr-1):
    vx_frame = vx_smo_data[:,:,i]; vy_frame = vy_smo_data[:,:,i];
    [vx_dc,vy_dc] = declutter_quiver(vx_frame,vy_frame,n_jump)
    vx_dc_smo_data[:,:,i] = vx_dc; vy_dc_smo_data[:,:,i] = vy_dc;
```

```
[X,Y] = np.meshgrid(np.arange(0,60,1), np.arange(0,60,1))
U = np.squeeze(vx_dc_smo_data[:,:,0]); V = np.squeeze(vy_dc_smo_data[:,:,0])

fig, ax = plt.subplots()
c = ax.imshow(np.squeeze(data[:, :, 0]), origin='upper')    # shows the raw image
im = ax.quiver(X,Y,U,V,angles='xy',color='red',minshaft=1,minlength=0.5, \
           pivot='middle',scale=13,width=0.005)             # plots the vector field


cbar = fig.colorbar(c, ax=ax)

def update_quiver(num, im, X, Y):            # Update function for the animation
    U = np.squeeze(vx_dc_smo_data[:,:,num]);
```

```
    V = np.squeeze(vy_dc_smo_data[:,:,num]);

    c.set_data(data[:, :, num])
    im.set_UVC(U,V)
    return im,

anim = animation.FuncAnimation(fig, update_quiver, fargs=(im, X, Y),
                               interval=30, frames=n_fr-1, blit=False)
fig.tight_layout()
plt.close (fig)
anim
```




**Example-2:** ***Three particles rotating in a circular orbit***

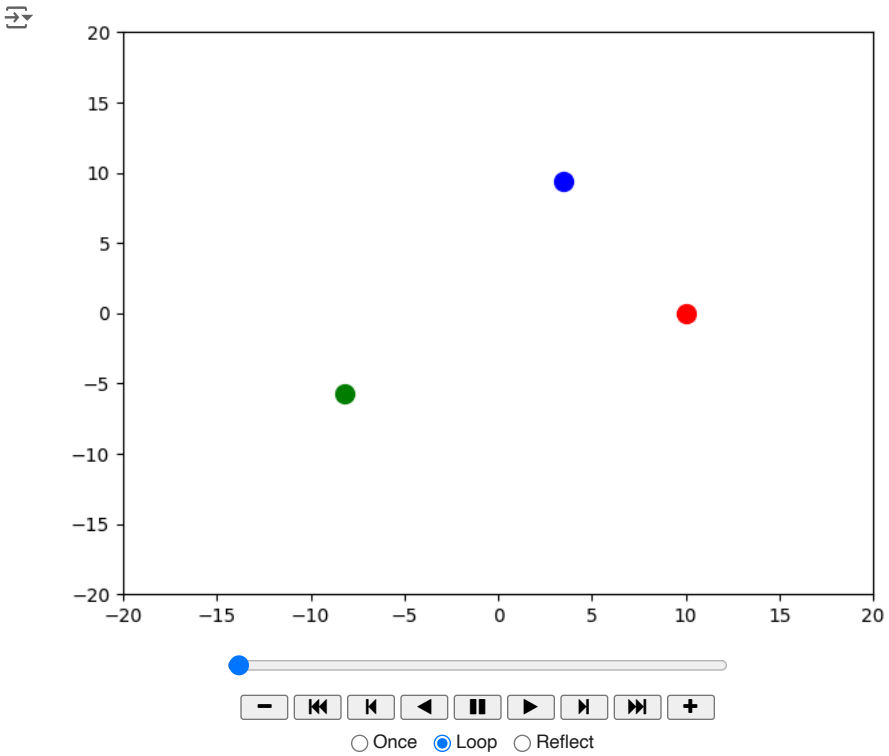
**Creating the particle trajectories**

```
w = 0.05;               # Angular frequency
n_fr = 150;             # Number of frames
x_in = 10; y_in = 10    # Controls the starting locations of the particles

# Memory allocation for x and y locations of the particles
x1_arr = np.zeros(n_fr); y1_arr = np.zeros(n_fr)
x2_arr = np.zeros(n_fr); y2_arr = np.zeros(n_fr)
x3_arr = np.zeros(n_fr); y3_arr = np.zeros(n_fr)

for i in tqdm(range(n_fr)):               # Obtaining the trajectories
    x1_arr[i] = x_in*np.cos(w*i); y1_arr[i] = y_in*np.sin(w*i)
    x2_arr[i] = x_in*np.cos(w*(n_fr+i)); y2_arr[i] = y_in*np.sin(w*(n_fr+i))
    x3_arr[i] = x_in*np.cos(w*(n_fr/2+i)); y3_arr[i] = y_in*np.sin(w*(n_fr/2+i))
```

**Visualizing the trajectories of the particles**

```
fig, ax = plt.subplots()

im1, = ax.plot(x1_arr[0],y1_arr[0],color='red',marker='o',markersize=10)
im2, = ax.plot(x2_arr[0],y2_arr[0],color='blue',marker='o',markersize=10)
im3, = ax.plot(x3_arr[0],y3_arr[0],color='green',marker='o',markersize=10)

ax.set_xlim(-20,20); ax.set_ylim(-20,20)         # set the x and y-bounds of the plot

in_fig = mplfig_to_npimage(fig); in_fig_comp = in_fig[:,:,0]
y_len = np.shape(in_fig_comp)[0]; x_len = np.shape(in_fig_comp)[1];

data = np.zeros([y_len,x_len,n_fr])    # Memory allocation for saving all the figures
```

```
13
14 def update_plot(num, im1, im2, im3):   # update function for animating the figure
15     im1.set_xdata(x1_arr[num]);  im1.set_ydata(y1_arr[num]);
16     im2.set_xdata(x2_arr[num]);  im2.set_ydata(y2_arr[num]);
17     im3.set_xdata(x3_arr[num]);  im3.set_ydata(y3_arr[num]);
18
19 anim = animation.FuncAnimation(fig, update_plot, fargs=(im1, im2, im3), frames = n_fr, interval=25)
20 fig.tight_layout()
21 plt.close(fig)
22 anim
```




**Saving the trajectories as individual images (and then matrices)**

```
 1 start = time.time()
 2
 3 for i in tqdm(range(n_fr)):
 4
 5     fig = plt.figure(dpi=100)
 6
 7     # Plotting the 3 particle trajectories
 8
 9     plt.plot(x1_arr[i],y1_arr[i],color='red',marker='o',markersize=10)
10     plt.plot(x2_arr[i],y2_arr[i],color='blue',marker='o',markersize=10)
11     plt.plot(x3_arr[i],y3_arr[i],color='green',marker='o',markersize=10)
12
13     plt.xlim(-20,20); plt.ylim(-20,20)                    # set the x and y-bounds of the plot
14
15     np_fig = mplfig_to_npimage(fig);                      # save the current figure as a numpy matrix
16     gray = cv2.cvtColor(np_fig, cv2.COLOR_BGR2GRAY)       # convert the RGB image to grayscale
17     data[:,:,i] = gray                                    # storing the grayscale image as our synthetic data
18
19     plt.close(fig)
20
21 end = time.time(); print('Run time = '+str(end-start)+' s')
22
```

```
Run time = 23.76055073738098 s
```

**Performing Optical Flow by calling individual time-frames (stored as matrices)**

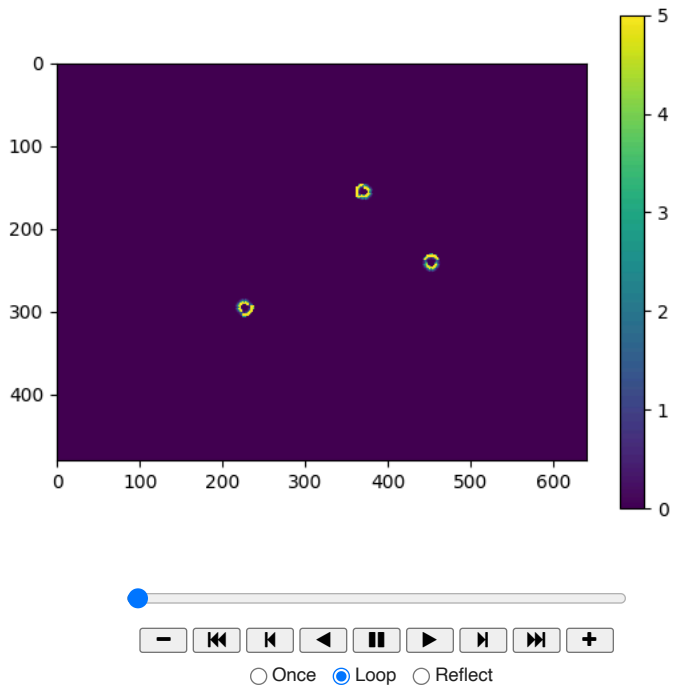
```
 1 # Memory allocation for storing optical flow results
 2 vx_data = np.zeros([y_len,x_len,n_fr])
 3 vy_data = np.zeros([y_len,x_len,n_fr])
```

```
vel_data = np.zeros([y_len,x_len,n_fr])
rel_data = np.zeros([y_len,x_len,n_fr])

sigma = 0.5        # smoothing parameter for the inbuilt Gaussian filter
rel_fac = 0.02     # Reliability score threshold

for k in tqdm(range(n_fr-1)):

    I1 = data[:,:,k]
    I2 = data[:,:,k+1]
    vx,vy,rel = LKxOptFlow(I1,I2,sigma,rel_fac)

    vx_data[:,:,k] = vx; vy_data[:,:,k] = vy; rel_data[:,:,k] = rel;
    vel_data[:,:,k] = np.sqrt(vx**2 + vy**2)
```

**Visualizing the velocity magnitudes as a function of time.**

Units are in #pixels/frame.

```
# @title
fig, ax = plt.subplots()

im = ax.imshow(np.squeeze(vel_data[:, :, 0]),vmin=0,vmax=5)
cbar = fig.colorbar(im, ax=ax)

def update(i):
    im.set_data(vel_data[:, :, i])

ani = animation.FuncAnimation(fig, update, frames = vel_data.shape[2], interval=40)
plt.close(fig)
ani
```




**Define a function that takes in existing vector fields and returns a smoothed and sparser version**

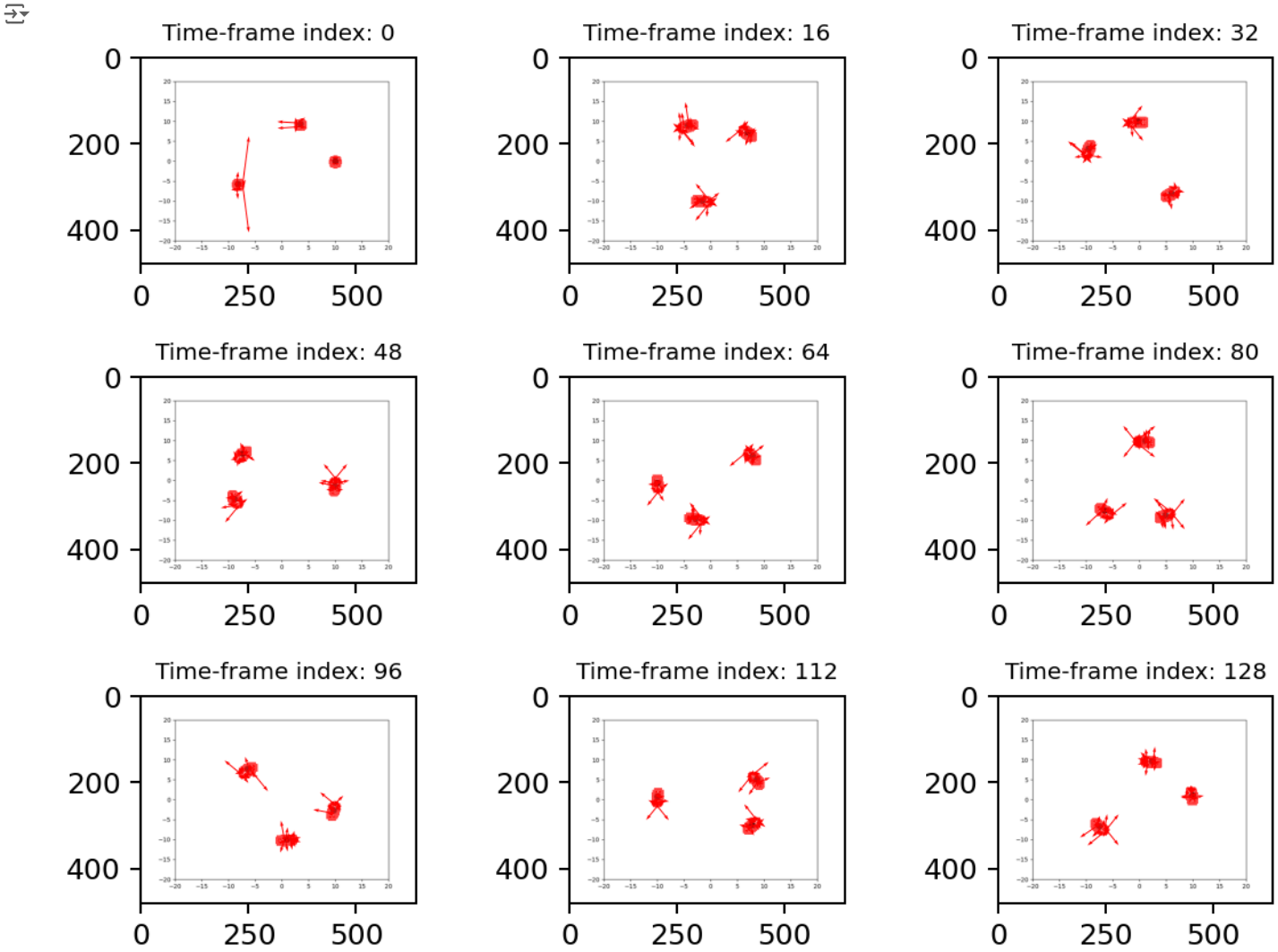
```
# INPUT:
# vx: x-component of velocity field
# vy: y-component of velocity field
# sigma: smoothing parameter used for the Gaussian filter
# jump: Number of successive vectors to skip (for visualization)


# OUTPUT:
# Xn: x-location mask of the non-zero vectors
# Yn: y-location mask of the non-zero vectors
```

```
# Un: x-component of the masked vectors
# Vn: y-component of the masked vectors

def quiver_processing(vx,vy,sigm,jump):

    y_len = np.shape(vx)[0]; x_len = np.shape(vx)[1]
    [X,Y] = np.meshgrid(np.arange(0,x_len,1), np.arange(0,y_len,1))

    # Gaussian smoothing of the vector field (to diminish the effect of spurious vectors)
    vx = sp.ndimage.gaussian_filter(vx, sigm, mode='nearest');
    vy = sp.ndimage.gaussian_filter(vy, sigm, mode='nearest');

    [vx,vy] = declutter_quiver(vx,vy,jump)    # declutter the exsiting vector field

    # only choose vectors with non-zero velocities

    vel = np.sqrt(vx**2+vy**2)
    mask = (vel !=0)      # mask with non-zero vectors

    # compute positions and lengths of all non-zero vectors
    Xn = np.squeeze(X*[mask])
    Yn = np.squeeze(Y*[mask])
    Un = np.squeeze(U*[mask])
    Vn = np.squeeze(V*[mask])

    return Xn,Yn,Un,Vn
```

**Visualizing the vector fields at different time-points.**

Animating all the time-frames with the post-processing of vector fields takes a lot of time.

```
gr_sz  = 3;     # grid size in only one axis (could be generalized further)
# In this case, we will generate a 3X3 image grid.

fig, ax = plt.subplots(gr_sz,gr_sz,dpi=180)   # Higher dpi value will create a larger figure.

for i in tqdm(range(gr_sz)):
    for j in range(gr_sz):

        jp = round((n_fr-1)/gr_sz**2)-1     # defining the number of time-frames to skip for visualization
        num = (gr_sz*i+j)*jp

        U = np.squeeze(vx_data[:,:,num]);
        V = np.squeeze(vy_data[:,:,num])
        [Xn,Yn,Un,Vn] = quiver_processing(U,V,1.0,2)        # post-processing of vector fields

        ax[i,j].imshow(data[:,:,num],'gray')
        ax[i,j].quiver(Xn, Yn, Un, Vn, scale_units='xy', scale = 1.5, width = 0.004, angles='xy', color='red')
        ax[i,j].set_title(f'Time-frame index: {num:d}', fontsize=8)    # sets a title to each subplot

fig.tight_layout()       # IMPORTANT. Otherwise, axes might overlap into each other!
plt.show()
```




## 3. 2D Optical Flow on real microscopic data

**Sparse Optical Flow**

- **Example-1: Crayfish Gut Motility**
  Download the TIFF stack from the shared Google Drive link

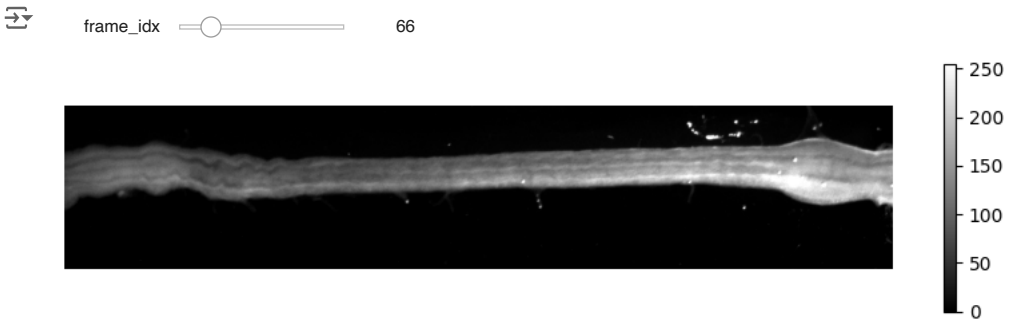
```
url = 'https://drive.google.com/uc?id=143g2f3W-RGRW7jXYnFx_Y7PS_3K9iNIe'
output = '/content/tiff_stack.tif'  # Path to save the downloaded file

gdown.download(url, output, quiet=False)

# Load the TIFF stack
tif_stack = imageio.volread(output)
```

```
Downloading...
From (original): https://drive.google.com/uc?id=143g2f3W-RGRW7jXYnFx_Y7PS_3K9iNIe
From (redirected): https://drive.google.com/uc?id=143g2f3W-RGRW7jXYnFx_Y7PS_3K9iNIe&confirm=t&uuid=16b537e0-4c66-449d-863f-1
To: /content/tiff_stack.tif
100%|██████████| 109M/109M [00:01<00:00, 62.7MB/s]
```

- Visualize the data

```
# Create a function to display a specific frame
def display_frame(frame_idx):
    plt.figure(figsize=(10, 6))
```

```
    plt.imshow(tif_stack[frame_idx], cmap='gray')
    plt.colorbar(shrink = 0.4)
    plt.axis('off')
    plt.show()

# Create an interactive widget
interactive_plot = interactive(display_frame, frame_idx=(0, len(tif_stack)-1))
display(interactive_plot)
```

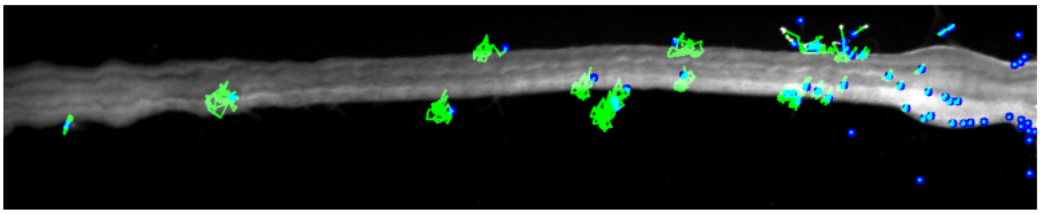



- Identify important features and track only those

```
video_frames = tif_stack

# Parameters for Lucas-Kanade optical flow
lk_params = dict(winSize=(15, 15),
                 maxLevel=2,
                 criteria=(cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03))

# Parameters for Shi-Tomasi corner detection
feature_params = dict(maxCorners=1000,qualityLevel=0.01,minDistance=7,blockSize=7)

# Read the first frame and find corners in it
old_frame = video_frames[0]
old_gray = old_frame.astype(np.uint8)
p0 = cv2.goodFeaturesToTrack(old_gray, mask=None, **feature_params)

# Create a mask image for drawing purposes
mask = np.zeros_like(cv2.cvtColor(old_gray, cv2.COLOR_GRAY2BGR))

frames = []
for i in tqdm(range(1, len(video_frames))):
    frame = video_frames[i]
    frame_gray = frame.astype(np.uint8)

    # Calculate optical flow
    p1, st, err = cv2.calcOpticalFlowPyrLK(old_gray, frame_gray, p0, None, **lk_params)

    # Select good points
    good_new = p1[st == 1]
    good_old = p0[st == 1]

    # Draw the tracks
    frame_with_vectors = cv2.cvtColor(frame_gray, cv2.COLOR_GRAY2BGR)
    for (new, old) in zip(good_new, good_old):
        a, b = new.ravel()
        c, d = old.ravel()
        mask = cv2.line(mask, (int(a), int(b)), (int(c), int(d)), (0, 255, 0), 2)
        frame_with_vectors = cv2.circle(frame_with_vectors, (int(a), int(b)), 5, (0, 0, 255), -1)

    img = cv2.add(frame_with_vectors, mask)
    frames.append(img)

    old_gray = frame_gray.copy()
    p0 = good_new.reshape(-1, 1, 2)

# Create a function to update the frame
def update_frame(frame_idx):
    plt.figure(figsize=(10, 10))
    plt.imshow(frames[frame_idx])
    plt.axis('off')
    plt.show()
```

```
# Create interactive widget
interactive_plot = interactive(update_frame, frame_idx=(0, len(frames)-1))
display(interactive_plot)
```

frame_idx 58



- **Example-2: Actin Dynamics in an Astrocyte**

  Download the TIFF stack from the shared Google Drive link

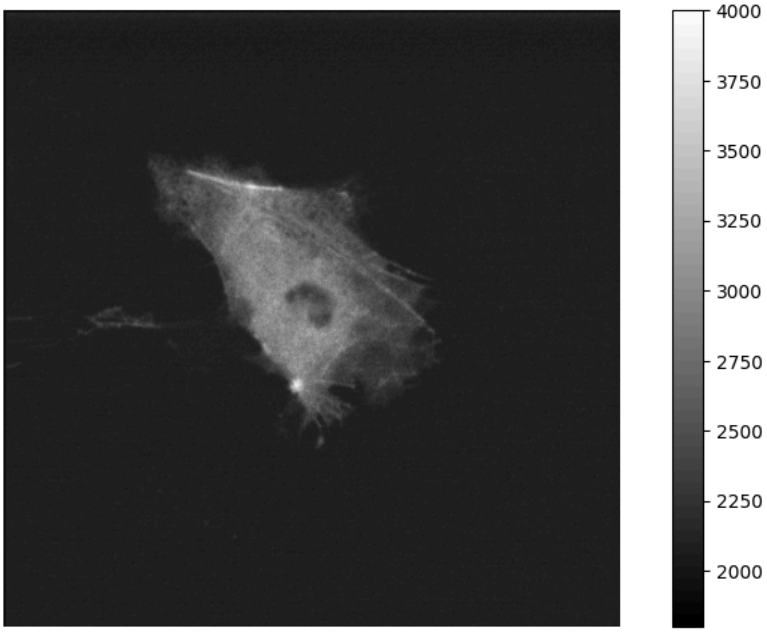
```
url = 'https://drive.google.com/uc?id=1NfrE1nRe9IUs9iS2LiEnQxytbP2Nsj1n'
output = '/content/tiff_stack.tif'  # Path to save the downloaded file

gdown.download(url, output, quiet=False)

# Load the TIFF stack
tif_stack = imageio.volread(output)
```

```
Downloading...
From: https://drive.google.com/uc?id=1NfrE1nRe9IUs9iS2LiEnQxytbP2Nsj1n
To: /content/tiff_stack.tif
100%|██████████| 79.2M/79.2M [00:00<00:00, 79.5MB/s]
```

- Check how the raw movie looks like

```
# Create a function to display a specific frame
def display_frame(frame_idx):
    plt.figure(figsize=(10, 6))
    plt.imshow(tif_stack[frame_idx], cmap='gray', vmin=1900, vmax = 4000)
    plt.colorbar()
    plt.axis('off')
    plt.show()

# Create an interactive widget
interactive_plot = interactive(display_frame, frame_idx=(0, len(tif_stack)-1))
display(interactive_plot)
```

frame_idx 150

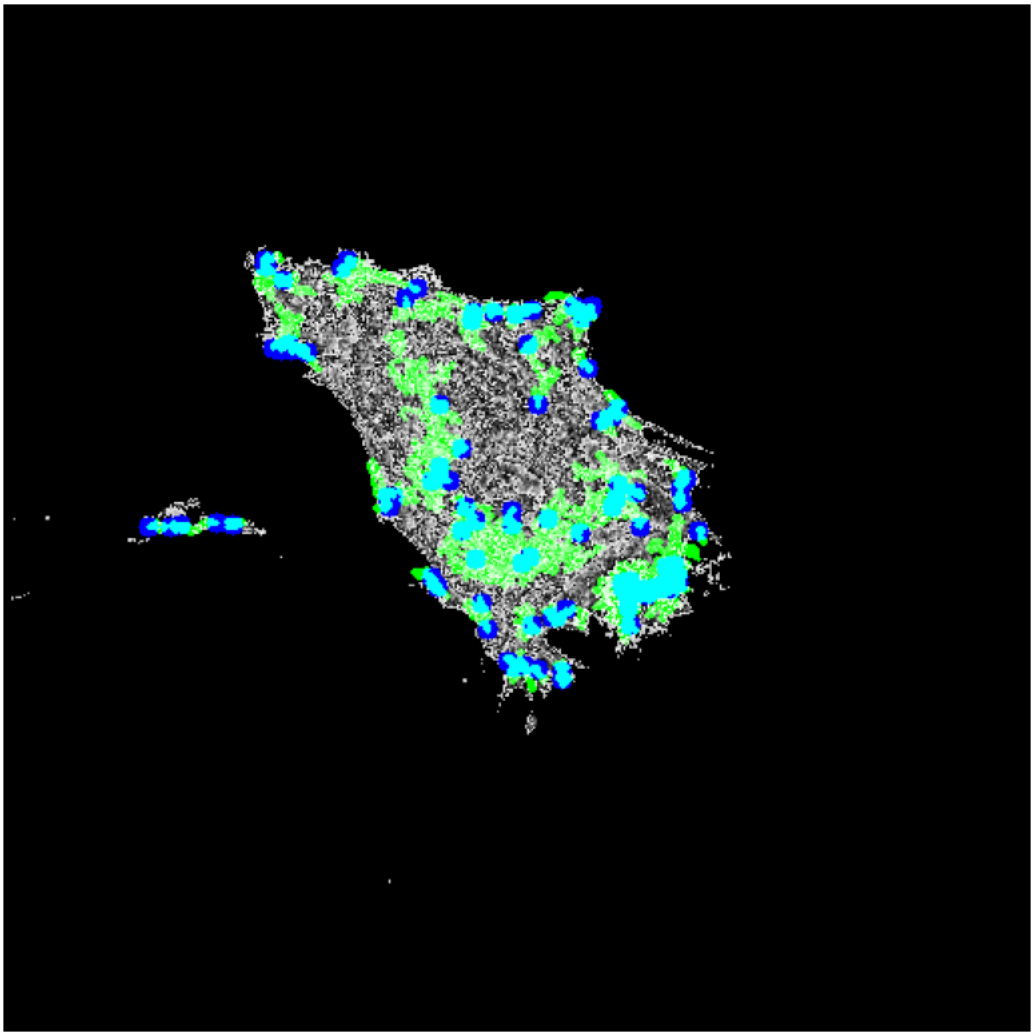



- Identify important features and track only those

```
# video_frames = np.zeros(np.shape(tif_stack))
# for i in range(len(video_frames)):
#     image_8bit = tif_stack[i].astype('uint8')

#     # Apply Otsu's thresholding
#     otsu_thresh_value = filters.threshold_otsu(image_8bit)
#     _, otsu_thresh = cv2.threshold(image_8bit, otsu_thresh_value, 255, cv2.THRESH_BINARY)
#     video_frames[i] = otsu_thresh

video_frames = tif_stack
video_frames[video_frames < np.percentile(video_frames,90)] = 0

# Parameters for Lucas-Kanade optical flow
lk_params = dict(winSize=(15, 15),
                 maxLevel=2,
                 criteria=(cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03))

# Parameters for Shi-Tomasi corner detection
feature_params = dict(maxCorners=100,qualityLevel=0.01,minDistance=7,blockSize=7)

# Read the first frame and find corners in it
old_frame = video_frames[0]
old_gray = old_frame.astype(np.uint8)
p0 = cv2.goodFeaturesToTrack(old_gray, mask=None, **feature_params)

# Create a mask image for drawing purposes
mask = np.zeros_like(cv2.cvtColor(old_gray, cv2.COLOR_GRAY2BGR))

frames = []
for i in tqdm(range(1, len(video_frames))):
    frame = video_frames[i]
    frame_gray = frame.astype(np.uint8)

    # Calculate optical flow
    p1, st, err = cv2.calcOpticalFlowPyrLK(old_gray, frame_gray, p0, None, **lk_params)

    # Select good points
    good_new = p1[st == 1]
    good_old = p0[st == 1]

    # Draw the tracks
    frame_with_vectors = cv2.cvtColor(frame_gray, cv2.COLOR_GRAY2BGR)
    for (new, old) in zip(good_new, good_old):
        a, b = new.ravel()
        c, d = old.ravel()
```

```
        mask = cv2.line(mask, (int(a), int(b)), (int(c), int(d)), (0, 255, 0), 2)
        frame_with_vectors = cv2.circle(frame_with_vectors, (int(a), int(b)), 5, (0, 0, 255), -1)

    img = cv2.add(frame_with_vectors, mask)
    frames.append(img)

    old_gray = frame_gray.copy()
    p0 = good_new.reshape(-1, 1, 2)

# Create a function to update the frame
def update_frame(frame_idx):
    plt.figure(figsize=(10, 10))
    plt.imshow(frames[frame_idx])
    plt.axis('off')
    plt.show()

# Create interactive widget
interactive_plot = interactive(update_frame, frame_idx=(0, len(frames)-1))
display(interactive_plot)

```

frame_idx 149



- **Example-3: Actin Dynamics in a Giant Dicty cell**

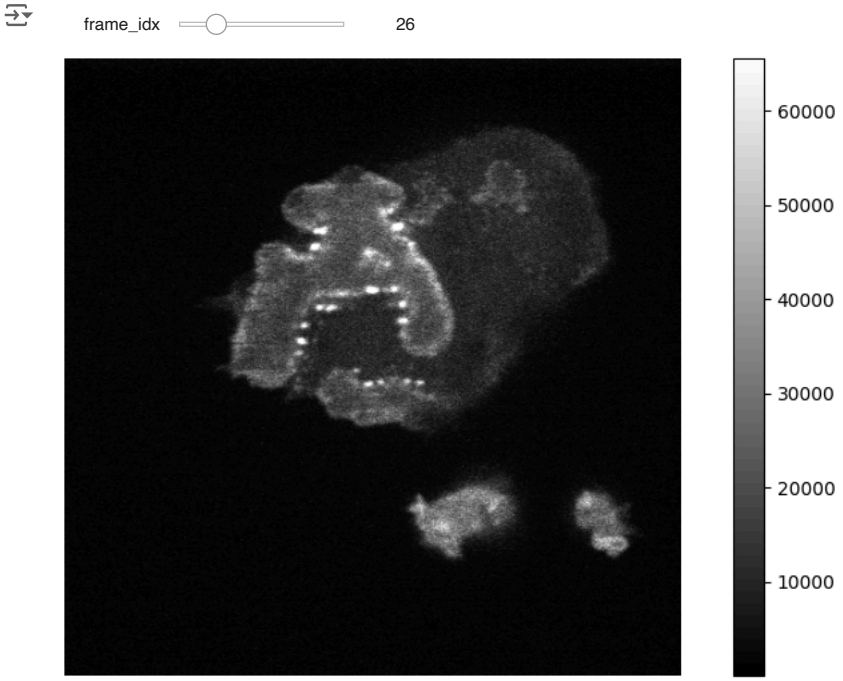
  Download the TIFF stack from the shared Google Drive link

```
url = 'https://drive.google.com/uc?id=1HrdyfbwczLXTqzAtzvCa08gSVR_qA25x'
output = '/content/tiff_stack.tif'  # Path to save the downloaded file

gdown.download(url, output, quiet=False)

```

```
# Load the TIFF stack
tif_stack = imageio.volread(output)
```

```
Downloading...
From: https://drive.google.com/uc?id=1HrdyfbwczLXTqzAtzvCa08gSVR_qA25x
To: /content/tiff_stack.tif
100%|██████████| 63.5M/63.5M [00:01<00:00, 61.9MB/s]
```

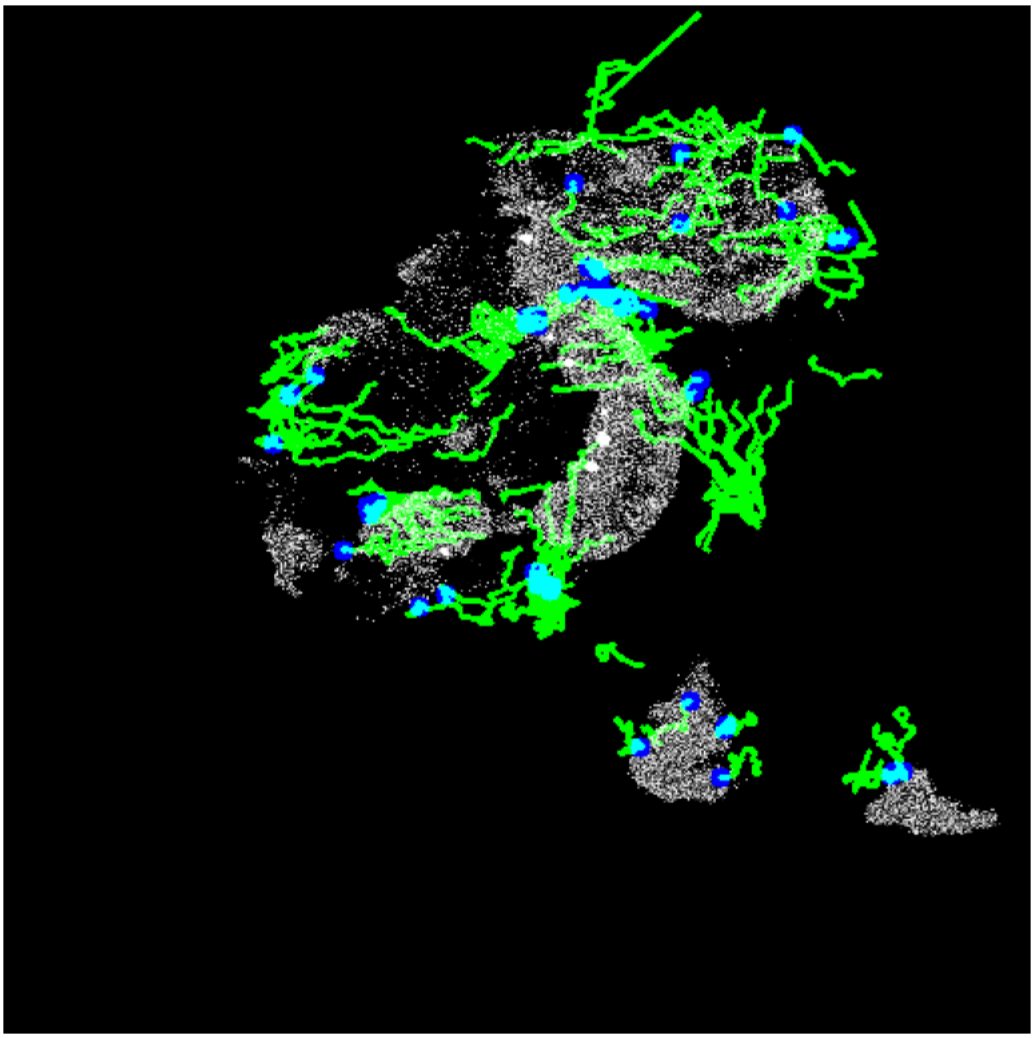
```
# Create a function to display a specific frame
def display_frame(frame_idx):
    plt.figure(figsize=(10, 6))
    plt.imshow(tif_stack[frame_idx], cmap='gray')
    plt.colorbar()
    plt.axis('off')
    plt.show()

# Create an interactive widget
interactive_plot = interactive(display_frame, frame_idx=(0, len(tif_stack)-1))
display(interactive_plot)
```




```
# video_frames = np.zeros(np.shape(tif_stack))
# for i in range(len(video_frames)):
#     image_8bit = tif_stack[i].astype('uint8')

#     # Apply Otsu's thresholding
#     otsu_thresh_value = filters.threshold_otsu(image_8bit)
#     _, otsu_thresh = cv2.threshold(image_8bit, otsu_thresh_value, 255, cv2.THRESH_BINARY)
#     video_frames[i] = otsu_thresh

video_frames = tif_stack
video_frames[video_frames < np.percentile(video_frames,90)] = 0

# Parameters for Lucas-Kanade optical flow
lk_params = dict(winSize=(15, 15),
                 maxLevel=2,
                 criteria=(cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03))

# Parameters for Shi-Tomasi corner detection
feature_params = dict(maxCorners=100,qualityLevel=0.01,minDistance=7,blockSize=7)

# Read the first frame and find corners in it
old_frame = video_frames[0]
old_gray = old_frame.astype(np.uint8)
p0 = cv2.goodFeaturesToTrack(old_gray, mask=None, **feature_params)

# Create a mask image for drawing purposes
mask = np.zeros_like(cv2.cvtColor(old_gray, cv2.COLOR_GRAY2BGR))

```

```
frames = []
for i in range(1, len(video_frames)):
    frame = video_frames[i]
    frame_gray = frame.astype(np.uint8)

    # Calculate optical flow
    p1, st, err = cv2.calcOpticalFlowPyrLK(old_gray, frame_gray, p0, None, **lk_params)

    # Select good points
    good_new = p1[st == 1]
    good_old = p0[st == 1]

    # Draw the tracks
    frame_with_vectors = cv2.cvtColor(frame_gray, cv2.COLOR_GRAY2BGR)
    for (new, old) in zip(good_new, good_old):
        a, b = new.ravel()
        c, d = old.ravel()
        mask = cv2.line(mask, (int(a), int(b)), (int(c), int(d)), (0, 255, 0), 2)
        frame_with_vectors = cv2.circle(frame_with_vectors, (int(a), int(b)), 5, (0, 0, 255), -1)

    img = cv2.add(frame_with_vectors, mask)
    frames.append(img)

    old_gray = frame_gray.copy()
    p0 = good_new.reshape(-1, 1, 2)

# Create a function to update the frame
def update_frame(frame_idx):
    plt.figure(figsize=(10, 10))
    plt.imshow(frames[frame_idx])
    plt.axis('off')
    plt.show()

# Create interactive widget
interactive_plot = interactive(update_frame, frame_idx=(0, len(frames)-1))
display(interactive_plot)

```

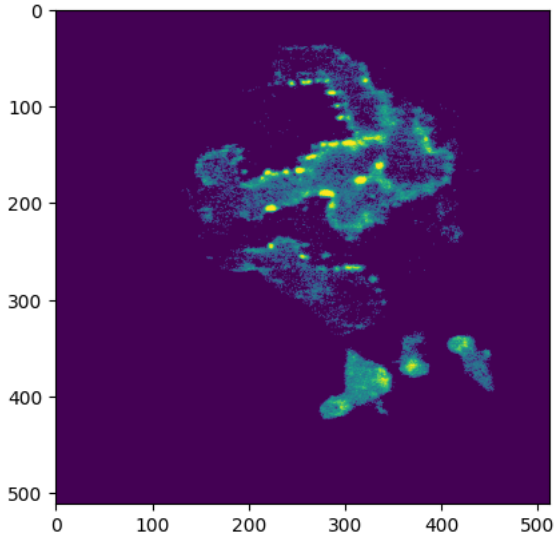
frame_idx 37



**Dense Optical Flow:**

```
# smoothing the raw movie in time
w1 = 0.036; w2 = 0.249; w3 = 0.431;   # smoothing kernel weights
img = tif_stack

start = time.time()

# smoothing the image frames
smo_im = w1*img[0:-4] + w2*img[1:-3] + w3*img[2:-2] + w2*img[3:-1] + w1*img[4:]
fin_im = np.concatenate((img[0:1],img[1:2],smo_im,img[-2:-1],img[-1:]),axis = 0)   # padding the bordering time-frames

end = time.time(); print('Run time = '+str(end-start)+' s')

n_fr,y_len,x_len = np.shape(img)    # save the shape of the movie
plt.imshow(fin_im[0,:,:])    # check out the first frame of the smoothened movie
```

```
Run time = 0.6736276149749756 s
<matplotlib.image.AxesImage at 0x7fd964081ba0>
```




```
# Memory allocation for storing optical flow results
vx_data = np.zeros([n_fr,y_len,x_len])
vy_data = np.zeros([n_fr,y_len,x_len])
vel_data = np.zeros([n_fr,y_len,x_len])
rel_data = np.zeros([n_fr,y_len,x_len])

sigma = 1.0        # smoothing parameter for the inbuilt Gaussian filter
rel_fac = 0.1     # Reliability score threshold

for k in tqdm(range(n_fr-1)):

    I1 = fin_im[k,:,:]
    I2 = fin_im[k+1,:,:]
    vx,vy,rel = LKxOptFlow(I1,I2,sigma,rel_fac)

    vx_data[k,:,:] = vx; vy_data[k,:,:] = vy; rel_data[k,:,:] = rel;
    vel_data[k,:,:] = np.sqrt(vx**2 + vy**2)
```

```
100%|██████████| 120/120 [00:06<00:00, 17.18it/s]
```

Decluttering the vector fields

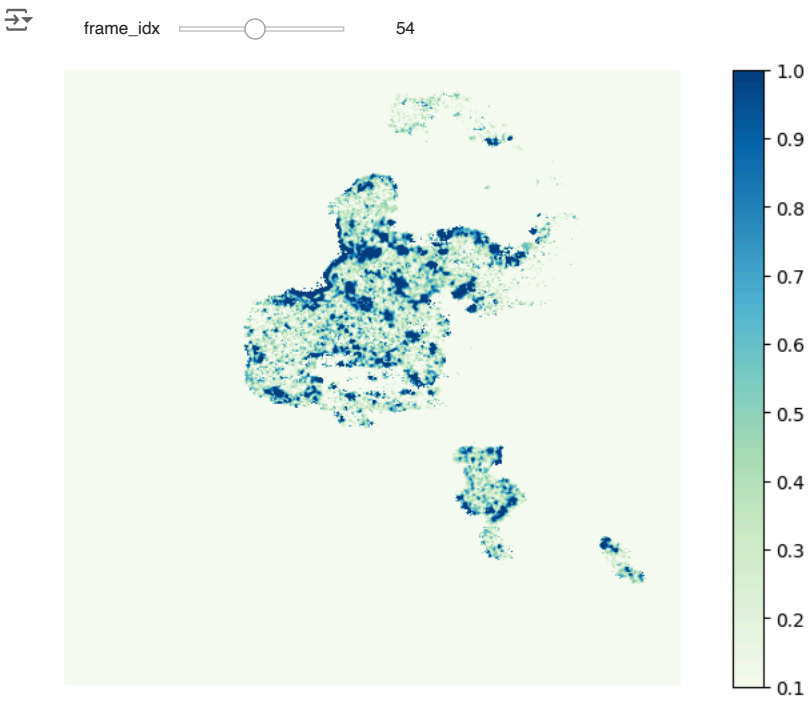
```
# Memory allocation for storing modified vector fields

vx_dc_data = np.zeros([n_fr,y_len,x_len]);
vy_dc_data = np.zeros([n_fr,y_len,x_len]);

n_jump = 5;     # Number of vectors to skip on each row and column

for i in tqdm(range(n_fr-1)):
    vx_frame = vx_data[:,:,i]; vy_frame = vy_data[:,:,i];
    [vx_dc,vy_dc] = declutter_quiver(vx_frame,vy_frame,n_jump)
    vx_dc_data[:,:,i] = vx_dc; vy_dc_data[:,:,i] = vy_dc;
```

```
100%|██████████| 120/120 [00:01<00:00, 113.82it/s]
```

- Show speed heatmap

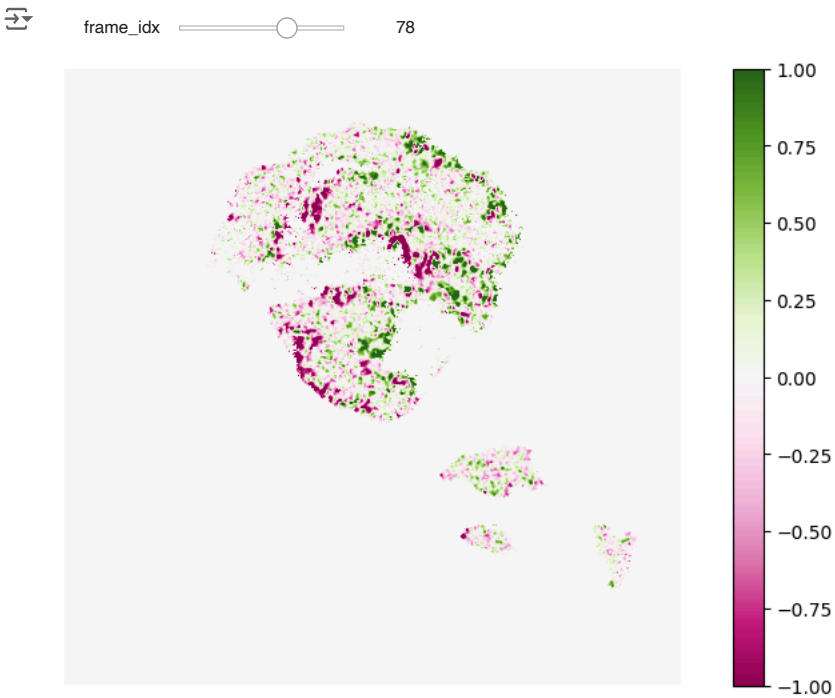
```
# Create a function to display the heatmap of speed values (for a specific frame)

def display_frame(frame_idx):
    plt.figure(figsize=(10, 6))
    plt.imshow(vel_data[frame_idx]*(img[frame_idx]>np.percentile(img,80)), vmin=0.1, vmax=1, cmap='GnBu')
    plt.colorbar()
    plt.axis('off')
    plt.show()

# Create an interactive widget
```

```
11 interactive_plot = interactive(display_frame, frame_idx=(0, len(tif_stack)-1))
12 display(interactive_plot)
```



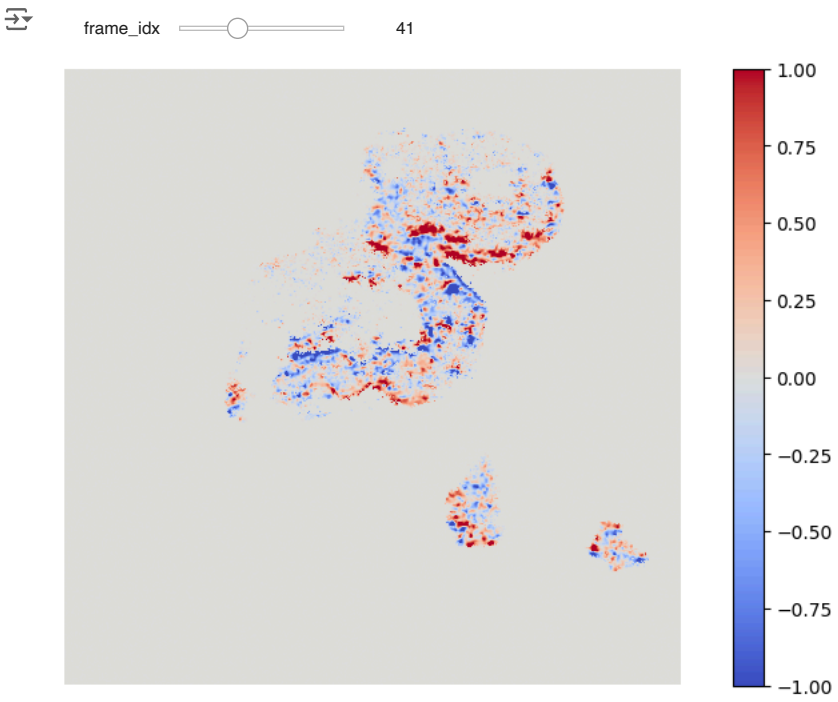

- Show x-velocity heatmap

```
 1 # Create a function to display the heatmap of vx values (for a specific frame)
 2
 3 def display_frame(frame_idx):
 4     plt.figure(figsize=(10, 6))
 5     plt.imshow(vx_data[frame_idx]*(img[frame_idx]>np.percentile(img,80)), vmin=-1, vmax=1, cmap='PiYG')
 6     plt.colorbar()
 7     plt.axis('off')
 8     plt.show()
 9
10 # Create an interactive widget
11 interactive_plot = interactive(display_frame, frame_idx=(0, len(tif_stack)-1))
12 display(interactive_plot)
```



- Show y-velocity heatmap

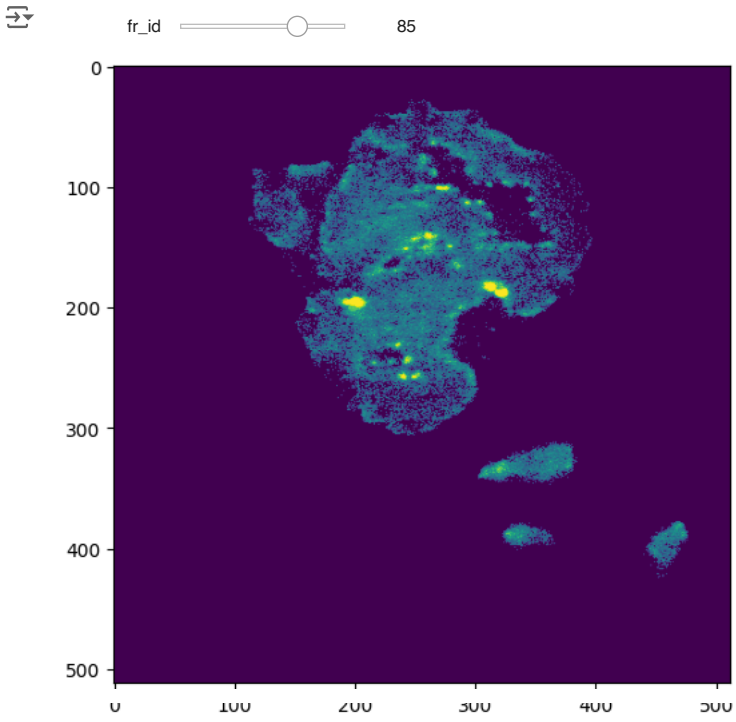
```
# Create a function to display the heatmap of vy values (for a specific frame)

def display_frame(frame_idx):
    plt.figure(figsize=(10, 6))
    plt.imshow(vy_data[frame_idx]*(img[frame_idx]>np.percentile(img,80)), vmin=-1, vmax=1, cmap='coolwarm')
    plt.colorbar()
    plt.axis('off')
    plt.show()

# Create an interactive widget
interactive_plot = interactive(display_frame, frame_idx=(0, len(tif_stack)-1))
display(interactive_plot)
```




```
y_len, x_len = np.shape(img)[1],np.shape(img)[2]
[X,Y] = np.meshgrid(np.arange(0,x_len,1), np.arange(0,y_len,1))
```

```
# Create a function to display the quiver plots

def display_frame(fr_id):

    plt.figure(figsize=(10, 6))


    vx = vx_data[fr_id,:,:]*(img[fr_id]>np.percentile(img,80))
    vy = vy_data[fr_id,:,:]*(img[fr_id]>np.percentile(img,80))

    # Gaussian smoothing of the vector field (to diminish the effect of spurious vectors)
    vx = sp.ndimage.gaussian_filter(vx, 3, mode='nearest');
    vy = sp.ndimage.gaussian_filter(vy, 3, mode='nearest');

    [U,V] = declutter_quiver(vx=vx,vy=vy,s=10)    # declutter the existing vector field

    # only choose vectors with non-zero speed values

    vel = np.sqrt(vx**2+vy**2)
    mask = (vel > 1.0)      # mask with non-zero vectors

    # del Xn,Yn,Un,Vn
    # compute positions and lengths of all non-zero vectors
    Xn = X*mask
    Yn = Y*mask
    Un = U*mask
    Vn = V*mask
```

```
28
29     plt.imshow(img[fr_id])
30     plt.quiver(Xn,Yn,Un,Vn,scale=0.03,angles='xy',scale_units='xy',width = 0.004,minlength=0.0001,headlength=5,headwidth=3,mi
31     plt.show()
32
33
34 # Create an interactive widget
35 interactive_plot = interactive(display_frame, fr_id =(0, len(tif_stack)-1))
36 display(interactive_plot)
```




## 4. Optical Flow limitations

1. Optical flow algorithms (specially Lucas-Kanade method) **assume smoothness of flow** to different extents which **might not be valid** always.

- 'Jerky' motion leads to spurious flow vectors.
- Movies might need to be smoothed in time during data preprocessing. If need be, spatial smoothing parameter within the algorithm has to be checked further.
- Also, one needs to choose the right F.P.S value (frames per second) while imaging.

.

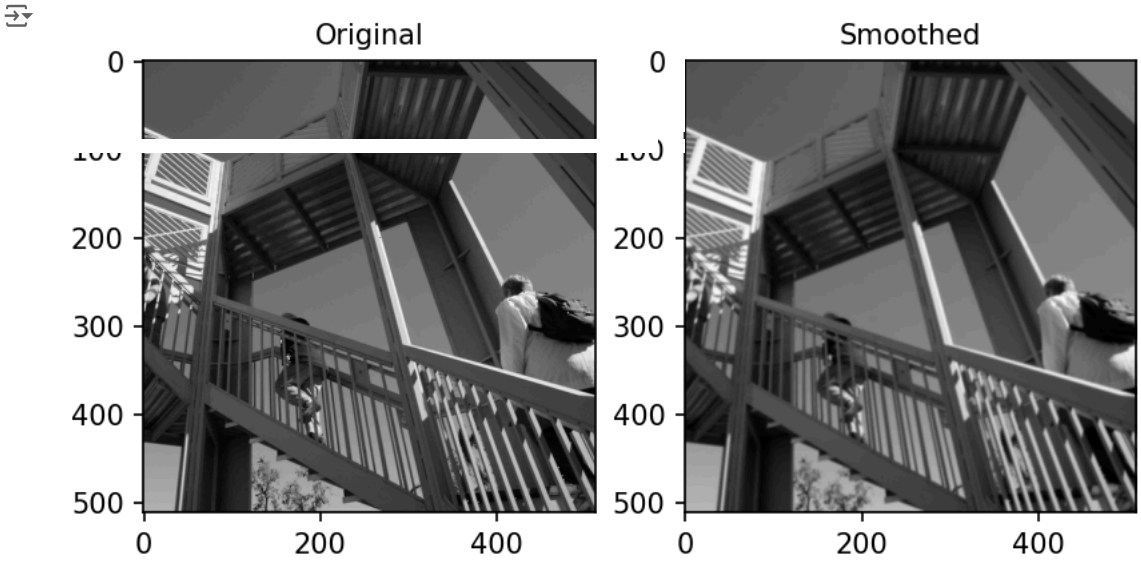
```
1 ## Example smoothing syntax
2
3 sigma = [0,0,1];
4 smo_data = sp.ndimage.gaussian_filter(data, sigma, mode='nearest')         # assuming time is along the third dimension of th
5
```

The first two dimensions are usually y and x. Here we did not perform any spatial smoothing (hence the first two entries of the smoothing kernel is zero).

**Example of spatial smoothing on an image:**

```
1 from scipy import datasets
2
3 fig = plt.figure(dpi=150)
4
5 plt.gray()  # show the filtered result in grayscale
6 ax1 = fig.add_subplot(121)  # left side
```

```
ax2 = fig.add_subplot(122)  # right side

ascent = datasets.ascent()
result = sp.ndimage.gaussian_filter(ascent, sigma=1)

ax1.imshow(ascent); ax1.set_title('Original', fontsize=10
ax2.imshow(result); ax2.set_title('Smoothed', fontsize=10
plt.show()
```




2. **Irregular illumination** of image causes wrong estimation of movement. (here fluctuation of pixel intensity is not necessarily caused by object motion)