# VISION: Pratical work – implementation of two optical flow methods

D. Béréziat
Sorbonne Université - Master IMA/DIGIT

January 17, 2023

**Start up** From the course web page: `http://pequan.lip6.fr/~bereziat/cours/master/vision/`, get the pratical work archive, unzip it in your home directory.

**Due date** By Monsday, January 23 via Moodle. Provide an archive containing the following files in a directory having your name (or several names if it is a jointed work):

- `horn.py`, `lucas.py`, `gradhorn.py`, `horn_xxx.py`, `lucas_xxx.py`, where `xxx` stand for one of seven datasets to process.
- you can also provide Jupyter notebooks or Matlab files (if you prefer Matlab)
- A short report where you discuss the results (do not detail the methods), and hyper-parameters. Each experiment would be illustrated by figures and a table of statistics when the ground truth is available. It also would be useful to compare the two methods between them.
- Do not provide data sets in the archive unless it is new data.

## 1 Horn-Schunck method

### Algorithm

1. Read two images $I_1$, $I_2$ (same size and dimensions)
2. Compute $I_x$, $I_y$ et $I_t$ (see next subsection)
3. Choose $N$ (number of iterations) and $\alpha$ (regularization)
4. $u^0 = v^0 = 0$, two images having same size than $I_1$
5. For $k = 0$ to $N - 1$:
   (a) compute $\bar{u}^k = A \star u^k$, $\bar{v}^k = A \star v^k$ with $A = \begin{pmatrix} \frac{1}{12} & \frac{1}{6} & \frac{1}{12} \\ \frac{1}{6} & 0 & \frac{1}{6} \\ \frac{1}{12} & \frac{1}{6} & \frac{1}{12} \end{pmatrix}$
   (b) compute: for $i$ line index and $j$ row index

$$u^{k+1}(i,j) = \bar{u}^k(i,j) - I_x(i,j)\frac{I_x(i,j)\bar{u}^k(i,j) + I_y(i,j)\bar{v}^k(i,j) + I_t(i,j)}{\alpha + I_x^2(i,j) + I_y^2(i,j)}$$

$$v^{k+1}(i,j) = \bar{v}^k(i,j) - I_y(i,j)\frac{I_x(i,j)\bar{u}^k(i,j) + I_y(i,j)\bar{v}^k(i,j) + I_t(i,j)}{\alpha + I_x^2(i,j) + I_y^2(i,j)}$$

6. Visualization with function `flowToColor()`

7. If available: read the ground truth with the function readFlowFile() and compare with $(u^N, v^N)$ (see next subsection).

## Determination of image gradient

Horn and Schunck proposed the following filter:

$$\begin{aligned}
I_x(i,j) &\simeq \frac{1}{4}[I_1(i,j+1) - I_1(i,j) + I_1(i+1,j+1) - I_1(i+1,j) \\
&\quad +I_2(i,j+1) - I_2(i,j) + I_2(i+1,j+1) - I_2(i+1,j)] \\
I_y(i,j) &\simeq \frac{1}{4}[I_1(i+1,j) - I_1(i,j) + I_1(i+1,j+1) - I_1(i,j+1) \\
&\quad +I_2(i+1,j) - I_2(i,j) + I_2(i+1,j+1) - I_2(i,j+1)] \\
I_t(i,j) &\simeq \frac{1}{4}[I_2(i,j) - I_1(i,j) + I_2(i+1,j) - I_1(i+1,j) \\
&\quad +I_2(i,j+1) - I_1(i,j+1) + I_2(i+1,j+1) - I_1(i+1,j+1)]
\end{aligned}$$

## Using provided functions

Functions computeColor() and readflow() take as parameter or return an array of dimension 3. The two first dimensions are the image coordinates, the third dimension is of size 2, and describe the 2 components of the velocity vector. Here an example in Python that reads an image and display the image gradient map with computeColor():

```
from PIL import Image
nasa = Image.open('nasa9.png')
from numpy import gradient
Ix,Iy = gradient(nasa)
import matplotlib.pyplot as plt
from middlebury import computeColor
plt.imshow(computeColor(Ix,Iy)
plt.show()
```

Or in Matlab:

```
nasa = imread('nasa9.png');
[Ix,Iy] = gradient(nasa);
w(:,:,1) = Ix;
w(:,:,2) = Iy;
imagesc(FlowToColor(w))
```

## Expected work

1. Write the function `gradhorn(I1,I2)` that takes two images as parameter and returns the spatio-temporal gradient of this pair of images.

2. Write the function `horn(I1,I2,alpha,N)` that returns compute the velocity map between images I1 and I2 according to the Horn and Schunck method.

3. Test your function on available data. For each data, you would provide a specific script that typically loads the data, computes the optical flow, displays the velocity map with `computeColor()`. Moreover,

if a ground truth is available, you would compute several statistics (mean, standard deviation) of EPEE, angular error, norm error. Angular error (in space-time, see lecture) at pixel $(i, j)$ is given by:

$$\theta(i,j) = \arccos\left(\frac{1 + w_r(i,j).w_e(i,j)}{\sqrt{1+\|w_r(i,j)\|^2}\sqrt{1+\|w_e(i,j)\|^2}}\right)$$

where $w_r$ is the ground truth (reference) and $w_e$ is the estimated map.

**Tips:**

- First, verify that the function `gradhorn()` is correct (check on data `square` for instance)
- Test la fonction `horn()` on data `mysine`. Find the optimal value `alpha` minimizing the angular error or EPE.
- Verify the velocity map obtained by your algorithm is visually coherent with the change between $I_1$ and $I_2$

4. **Bonus:** with the help of function `quiver()` visualize the velocity map as an under-sampled field of vectors (for instance, display 1 vector among 10). A scaling parameter can also be applied if the vectors has a low magnitude. Another data are available on web site mentioned in the lecture.

# 2 Lucas-Kanade method

## Algorithm

1. Read two images $I_1$ and $I_2$
2. Determine $I_x$, $I_y$ and $I_t$ (with the Horn-Schunck gradient filter)
3. Choose a window size $n$
4. For each pixel $p$ in the image domain (excepted pixels on domain border, i.e. those at distance $\frac{n}{2}$ from the border), do:
   (a) Center a squared window $W$, of size $n \times n$, on pixel $p$
   (b) Form the column vector $B = (-I_t(p'))_{p' \in W}$ (of length $n^2$)
   (c) Form the matrix $A = \begin{pmatrix}(I_x(p'))_{p' \in W} & (I_y(p'))_{p' \in W}\end{pmatrix}$ (of dimension $n^2 \times 2$)
   (d) Compute velocity at pixel $p$: $w = \left(A^T A\right)^{-1} A^T B$

## Expected work

Repeat the questions from the previous section, but considering Lucas-Kanade method. For each experiment, try to find the best parameter ($n$) and compare to the best run of Horn-Schunck. Test alternative neighboring system, for instance a Gaussian window $W$.

The same tips given in the previous section also apply here.