VISION: Practical work 2 – advanced optical flow methods

D. Béréziat
Sorbonne Université - Master IMA/DIGIT

January 20, 2026

Start up From the course web page: `http://perso.lip6.fr/Dominique.Bereziat/VISION/`, get the pratical work archive, unzip it in your home directory.

Due date By Tuesday at noon, January 26 via Moodle. Provide an archive containing code sources, a report, or notebook. Do not provide data sets in the archive unless it is new data (see previous practical work for details).

1 Nagel method

This method is similar to Horn and Schunck ones. However it requires second spatial derivatives from images I_1 and I_2: I_{xx}, I_{yy}, and I_{xy}. Theses derivatives can be obtained by applying twice the `gradhorn()` procedure. The iterative scheme (see lecture II, slide 8) also applies spatial derivatives on scalar image f: f_x, f_y, $f_{xy} = \frac{\partial f_x}{\partial y}$. We suggest to use the following operators: $f_x(i,j) \simeq \frac{f(i,j+1)-f(i,j-1)}{2}$ for horizontal derivative, and $f_y(i,j) \simeq \frac{f(i+1,j)-f(i-1,j)}{2}$ for the vertical derivative.

Algorithm (see slides 6, 7, and 8 lecture II)

1. Read two images I_1, I_2 (same size and dimensions)
2. Compute I_x, I_y, I_t, I_{xx}, I_{yy}, and I_{xy}
3. Set N (number of iterations), α (regularization), δ (oriented regularization)
4. $u^0 = v^0 = 0$, two images having same size than I_1
5. For $k = 0$ to $N-1$:
   (a) compute $\tilde{u}^k = \eta(u^k)$, $\tilde{v}^k = \eta(v^k)$ with:
   - η a function applying on a scalar image f such as $\eta(f) = \bar{f} - 2I_xI_yf_{xy} - q\nabla f$
   - $q = \frac{1}{I_x^2+I_y^2+2\delta}\nabla I^T \left[\begin{pmatrix} I_{yy} & -I_{xy} \\ -I_{xy} & I_{xx}\end{pmatrix} + 2\begin{pmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy}\end{pmatrix} V\right]$
   - $V = \frac{1}{I_x^2+I_y^2+2\delta}\begin{pmatrix} I_y^2+\delta & -I_xI_y \\ -I_xI_y & I_x^2+\delta\end{pmatrix}$
   - and $\bar{f}$ is as in Horn and Schunck algorithm step 5.(a)

   (b) compute: for i line index and j row index

$$\begin{aligned} u^{k+1} &= \tilde{u}^k - I_x \frac{I_x\tilde{u}^k + I_y\tilde{v}^k + I_t}{\alpha + I_x^2 + I_y^2} \\ v^{k+1} &= \tilde{v}^k - I_y \frac{I_x\tilde{u}^k + I_y\tilde{v}^k + I_t}{\alpha + I_x^2 + I_y^2} \end{aligned}$$

Evaluate, compare, and discuss the result of Nagel method applied on image sequences used in practical work 1.

2 Use automatic differentiation and optimizer

One can leverage the capability of deep learning framework (such as Pytorch) to automatically determine the gradient of a cost function and optimizer to minimize the cost function.

- You will be able to write in Python a function taking as parameters the velocity field u and v, and the image gradient I_x, I_y, I_t and returning the Horn and Schunck cost function (i.e. $\|I_x u + I_y v + I_t\|_2^2 + \alpha(\|\nabla u\|_2^2 + \|\nabla v\|_2^2)$).
- The optimization procedure is similar to that used in deep learning training:
  - initialize the optimizer, the parameters being the values of images u and v. We **strongly** suggest to use LBFGS (see example below),
  - iterate the following procedure: set to zero the gradient buffer, compute the cost function, retropropagate the gradient, perform a steepest descent.
- Example of steepest descent with Pytorch LBFGS:

```python
optimizer = torch.optim.LBFGS(...)
def closure():
    optimizer.zero_grad() # nullify gradient
    L = costfunction(...) # compute the cost
    L.backward())         # retropropagate and one step of descent direction
optimizer.step(closure)   # perform the optimization steps
```

- Experiment with L_2 norms and compare with Horn and Schunck. Experiment with L_1 and Huber norms and discuss the results.

3 Multiresolution optical flow

We want to code and test the Pyramidal Lucas-Kanade method. This method is based on a multiresolution algorithm as described in lecture II (slides 29–35). We recall the algorithm:

1. Set parameters K (number of resolutions), σ (standard deviation of the Gaussian antialiasing filter), W (window size of Lucas-Kanade method)
2. Read I_1 and I_2 two consecutive images
3. Build the pyramid of resolution:
   - $(I_1^k)_{k=1\cdots K}$ such as:

   $$\begin{aligned} I_1^K &= I_1 \\ I_1^{k-1} &= \downarrow (I_1^k \star g_\sigma) \end{aligned}$$

   - and same process for $(I_2^k)_{k=1\cdots K}$.

   $\downarrow$ is a downsampling operator that resize an image of size $n \times m$ to a image of size $\frac{n}{2} \times \frac{m}{2}$. The convolution with a Gaussian kernel **before** the downsampling allows us to remove aliasing.
4. Initialize $u^0 = v^0 = 0$

5. Iterate $k = 1 \cdots K$:

   (a) Upsample u^{k-1} and v^{k-1} to the size of I_1^k and apply a scale factor of 2 (see lecture II). The upsampling can be done by `skimage.transform.resize()` from the `scikit-image` module. Tips: read the documentation, and choose a bi-linear interpolation. In the following, $\uparrow$ is the upsampling operator: $u^k = \uparrow u^{k-1}, v^k = \uparrow v^{k-1}$.

   (b) Compute the shifted image $I_{shifted} = I_2(x + u^k, y + v^k)$. This can be done by the function `skimage.transform.warp()`. We suggest to use the following Python function:

```python
from skimage.transform import warp
def warp_image(I, u, v):
    nr, nc = I.shape
    row_coords, col_coords = np.meshgrid(np.arange(nr), np.arange(nc),
                                         indexing='ij')
    return warp(I, np.array([row_coords + v, col_coords + u]))
```

   and investigate the parameters `order` and `mode` of `warp()`

   (c) Compute the spatio-temporal gradient between I_1^k and $I_{shifted}(x)$. Tips: you can use the `gradhorn()` function.

   (d) Apply Lucas Kanade solver: $du, dv = LK(Ix, Iy, It)$

   (e) Update velocities: $u^k = u^k + du, v^k = v^k + dv$

Apply this algorithm on data with large displacement (archive data2.zip). Discuss your results. You can also use alternative optical flow solver to LK: for example, Horn-Schunck, Iterative LK (described in slides 27-28 in lecture II), or other methods.