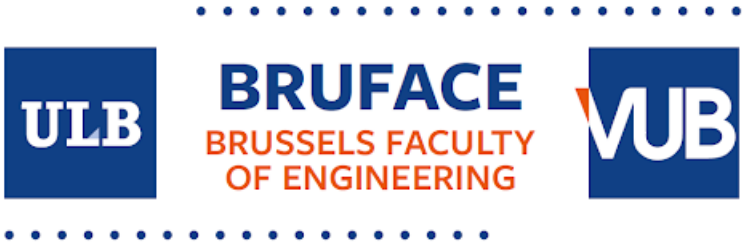

COMPUTER VISION

ELEC-Y512

WPO2 : Optical flow estimation

Authors:
Maxime BOLLENGIER

Professor:
Sahli HICHEM

Assistant
Berenguer ABEL DIAZ

Academic Year: 2021-2022

Contents

1 Introduction

In this report, we will discuss the implementation of the Horn and Schunck method [1] in order to estimate the optical flow between two frames.

2 The Horn and Schunck Method

The classical Horn and Scrunck method [1] consists of adding optical flow constraints to formulate an energy function $J(h)$. The aim is to find a vector field $h = (u, v)$ which minimizes this function in order to estimate the optical flow (u, v). Following [2], the energy function $J(h)$ is written as :

$$J(h) = \int_{\omega} (I_x u + I_y v + I_t)^2 + \alpha^2 (\|\nabla u\|^2 + \|\nabla v\|^2) \tag{1}$$

where I_x , I_y , I_t are the partial derivative between the two frames along respectively the x, y and t axis. α acts like a regulator to balance the weight between the constraints. By representing the minimization problem as an Euler-Lagrange equation [2] and by approximating the Laplacian as follows :

$$\begin{aligned} div(\nabla u) &\approx (\overline{u} - u) \\ div(\nabla v) &\approx (\overline{v} - v) \end{aligned} \tag{2}$$

where $(\overline{u}, \overline{v})$ are local averages of (u, v), we finally obtain the following system of equations [2]:

$$\begin{aligned} (\alpha^2 + I_x^2 + I_y^2)(u - \overline{u}) &= -I_x(I_x\overline{u} + I_y\overline{v} + I_t) \\ (\alpha^2 + I_x^2 + I_y^2)(v - \overline{v}) &= -I_y(I_x\overline{u} + I_y\overline{v} + I_t) \end{aligned} \tag{3}$$

By writing (3) for each pixel will lead to an iterative procedure described in section 3.2.

3 Implementation details

3.1 Step 1 : Compute the gradient between 2 frames

This step consists of computing partial derivatives I_x , I_y , I_t between two frames. These partial derivatives are obtained by averaging the forward differences between them as described in the code below :

```
1
2  def compute_partial_derivative(img1, img2):
3      x_kernel = np.array([[-1, 1], [-1, 1]]) * 0.25
4      y_kernel = np.array([[-1, -1], [1, 1]]) * 0.25
5      t_kernel = np.ones((2, 2)) * 0.25
6
7      I_x = conv2(img1, x_kernel) + conv2(img2, x_kernel)
8      I_y = conv2(img1, y_kernel) + conv2(img2, y_kernel)
9      I_t = conv2(img1, -t_kernel) + conv2(img2, t_kernel)
10
11     return [I_x, I_y, I_t]
```

Listing 1: Gradient function

3.2 Step 2 : Iterative procedure

In order to minimize the loss function (1), an iterative procedure can be obtained by the Jacobi method for solving (3). This iterative process is described by equation (4) and implemented in Listing (2).

$$u^{n+1} := \overline{u}^n - I_x \frac{I_x \overline{u}^n + I_y \overline{v}^n + I_t}{\alpha^2 + I_x^2 + I_y^2}$$

$$v^{n+1} := \overline{v}^n - I_y \frac{I_x \overline{u}^n + I_y \overline{v}^n + I_t}{\alpha^2 + I_x^2 + I_y^2} \tag{4}$$

To implement this iterative procedure, we first set u and v to zero. Next, we create a local average kernel 3×3 to get $(\overline{u}, \overline{v})$ at each iteration. Following [2] this kernel is like :

$$\begin{bmatrix} 1/12 & 1/6 & 1/12 \\ 1/6 & 0 & 1/6 \\ 1/12 & 1/6 & 1/12 \end{bmatrix} \tag{5}$$

Having the average kernel (5) and I_x , I_y , I_t from step 1, we can now begin the iterative process for a fixed value of α. To stop this process, we add two parameters N_s and ϵ. The first one is simply a number of maximum iterations performed. The second one corresponds to an arbitrary small value based on the solution variation between two iterative process h^n, h^{n+1}. Following [2], this small ϵ can be defined as follows :

$$\frac{1}{N} \sum_{i,j} (u_{i,j}^{n+1} - u_{i,j}^n)^2 + (v_{i,j}^{n+1} - v_{i,j}^n)^2 \leq \epsilon^2 \tag{6}$$

Then, the code of the the Horn and Schunck Method for optical flow estimation is shown below :

```
def compute_HS(img1, img2, alpha, epsilon, Ns):
    I_1 = img1
    I_2 = img2

    # set up initial values
    u = np.zeros((I_1.shape[0], I_1.shape[1]))
    v = np.zeros((I_1.shape[0], I_1.shape[1]))
    I_x, I_y, I_z = compute_partial_derivative(I_1, I_2)

    # set up local average kernel
    local_avg_kernel = np.array([[1 / 12, 1 / 6, 1 / 12],
                                 [1 / 6, 0, 1 / 6],
                                 [1 / 12, 1 / 6, 1 / 12]], float)

    # set the stopping values
    iter_counter = 0
    diff = 1
    while diff > epsilon and iter_counter < max_it:

        # apply local average
        u_avg = conv2(u, local_avg_kernel)
        v_avg = conv2(v, local_avg_kernel)

        # copy precedent value
        u_past = u
        v_past = v

        # apply the solution for (u,v)
        u = u_avg - I_x * (I_x * u_avg + I_y * v_avg + I_z/ alpha ** 2 + I_x ** 2 + I_y**2)
        v = v_avg - I_y * (I_x * u_avg + I_y * v_avg + I_z / alpha ** 2 + I_x ** 2 + I_y**2)

        # update stopping values
        diff = np.sum(np.square(u-u_past) + np.square(v - v_past))/len(u)
```

```
34          iter_counter += 1
35      return [u, v]
```

Listing 2: HS algorithm function

3.3 Step 3 : Visualize

To visualize the results, there exist some python libraries like : Pystep [1] (motionplot, quiver plot, stream plot), Matplotlib[2] (quiver plot, vector field plot, etc...), etc .. It's also possible to design own method. By testing all of the above solutions, the result gave not a good looking visual result. So, I found an implementation on the following article [3] on `https://github.com/tomrunia/OpticalFlow_Visualization` which gave me a good visual result.

[1] `https://pysteps.github.io/`
[2] `https://matplotlib.org/`

4 Results and discussion

4.1 Visual results

4.1.1 Optical flows for various alpha

Figure 1: Optical flow between frame 1 and frame 2 for $\alpha = 5$ and stop after $N_s = 5$ iterations

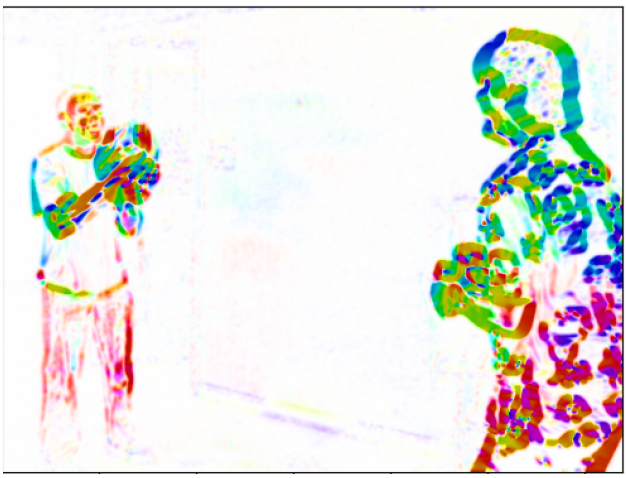

Figure 2: Optical flow between frame 1 and frame 2 for $\alpha = 15$ and stop after $N_s = 5$ iterations

Figure 3: Optical flow between frame 1 and frame 2 for $\alpha = 50$ and stop after $N_s = 5$ iterations

4.1.2 Optical flows for different precisions

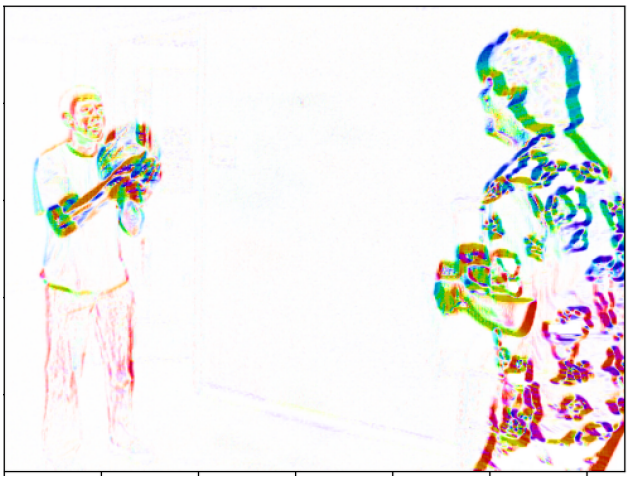

Figure 4: Optical flow between frame 1 and frame 2 for $\alpha = 15$ and stop after $N_s = 1$ iterations

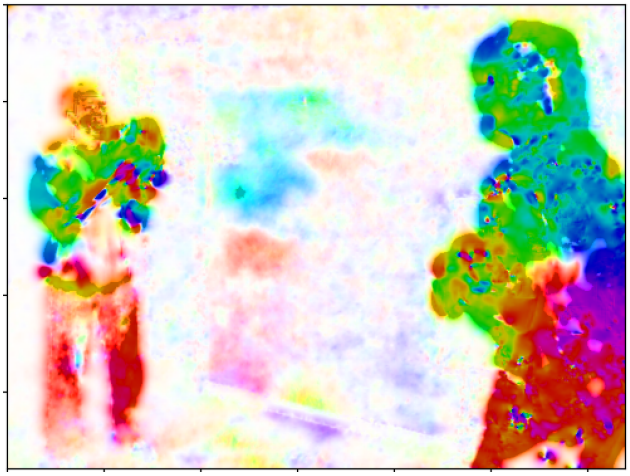

Figure 5: Optical flow between frame 1 and frame 2 for $\alpha = 15$ and stop after $N_s = 100$ iterations

4.2 Discussion on the results

By setting a few values of α it can be observed (on Figures 1, 2 and 3) that it changes the smoothness of the output. Setting alpha too high will lead to a white image without any pixel in motion. In opposite, by setting α too low leads to a noisy result with too much flow as expected. As a conclusion, we can interpret that alpha acts like a noise filter.

N_s and ϵ act like a stabilizer in order to make a trade-off between run-time and precision during the iterative method. If we want a more precise result, we have to decrease ϵ or/and increase N_s with the shortcoming to have a bigger run-time.

It's important to note that α, N_s and ϵ are correlated. If we want to get more precise results, we have to set small ϵ and/or set high vales N_s but we have to increase the value of α to remove the contribution of additional noise which is taken into account by the increasing precision as shown on Figures 4 and 5.

Bibliography

References

[1] B. K. Horn and B. G. Schunck, "Determining optical flow," *Artificial Intelligence*, vol. 17, no. 1, pp. 185–203, 1981, ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(81)90024-2. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0004370281900242.

[2] E. Meinhardt-Llopis, J. Sánchez Pérez, and D. Kondermann, "Horn-Schunck Optical Flow with a Multi-Scale Strategy," *Image Processing On Line*, vol. 3, pp. 151–172, 2013, https://doi.org/10.5201/ipol.2013.20.

[3] S. Baker, S. Roth, D. Scharstein, M. J. Black, J. Lewis, and R. Szeliski, "A database and evaluation methodology for optical flow," in *2007 IEEE 11th International Conference on Computer Vision*, 2007, pp. 1–8. DOI: 10.1109/ICCV.2007.4408903.