UNIVERSITÉ LIBRE DE BRUXELLES

BRUFACE
BRUSSELS FACULTY OF ENGINEERING

VRIJE UNIVERSITEIT BRUSSEL

Computer Vision

PROJECT REPORT: OPTICAL FLOW

Electrical Engineering

Di Bella Leandro

2022-2023

Prof. Dr. Ir. Sahli Hichem
Engineering Sciences

Contents

Introduction

As part of the Computer Vision course, students are asked to realize a variety of projects about computer vision like Stereo Vision, Photometric Stereo. For this project, student had to implement an algorithm capable of estimating an optical flow between consecutives frames.

In this project, Optical flow has been approached as a hand-crafted optimization problem over the space of displacement fields between a pair of images and two methods of the most well-known methods will be illustrated here.

The first one is the Lucas–Kanade method which is a widely used differential method for optical flow estimation developed by Bruce D. Lucas and Takeo Kanade. The second one is called the Horn–Schunck method and estimates optical flow as a global method which introduces a global constraint of smoothness.

A first theoritical reminder of both methods will be adressed. Then, the implementation of the two approaches will be explained. Those two algorithms have been implemented in Python and tested on two differents groups of consecutives images. Lastly, the results will be showed with a discussion.

Chapter 1
Theoritical Reminder

As previoulsy said, the first part consists of a theoritical reminder about Optical Flow in general and the two methods illustrated in this project. The first one will be Lucas-kanade method and followed by Horn-Shunck methond.

1.1 Optical Flow : What is it ?

Optical flow is the task of estimating per-pixel motion between video frames. It is a long-standing vision problem that remains unsolved. The optical flow is the relation of the motion field which returns the 2D projection of the physical movement of points relative to the observer to 2D displacement of pixel patchs on the image plane as we can see on the figure 1.1.

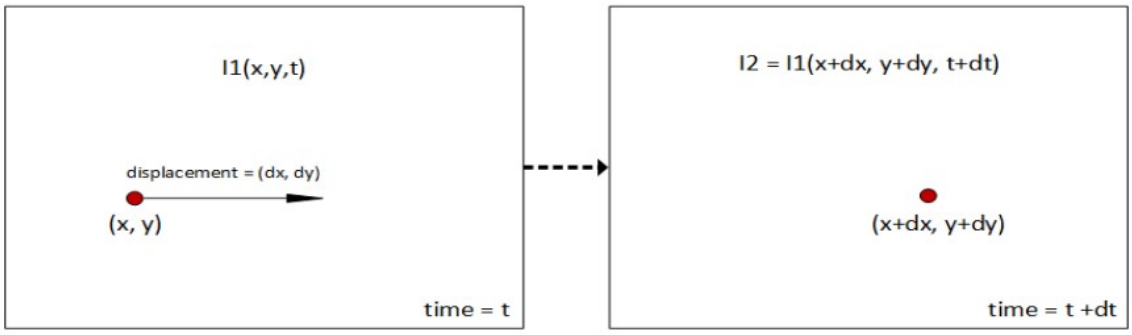

Figure 1.1: Optical Flow Schema

Opical Flow is of two types. Sparse optical flow and dense optical flow which are reprensented on figure 1.2. Sparse optical flow describes the flow vector only for some selected objects' features (e.g., edges, corners), while dense optical flow computes the flow vectors of all pixels from the image. In this projects, the second one has been considered.

Figure 1.2: Optical Flow Schema

1.1.1 Normal Optical Flow

In order to estimate the optical flow of two consecutives images, we will estimate te pixel motion from image I(x,y;t) to image I(x,y;t+δt). For that, two constraints will be established :

- Color constancy/Brightness constancy : a points I(x,y;t) looks "the same" in the images
- Small motion: Points do not move very far

It can be expressed as :

$$I(x, y, t) = I(x + \delta x, y + \delta y, t + \delta t) \tag{1.1}$$

By taking the Taylor Series Approximation :

$$I(x + \delta x, y + \delta y, t + \delta t) = I(x, y, t) + \frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t + ... \tag{1.2}$$

We finally get :

$$\frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t = 0 \tag{1.3}$$

Or divided by t:

$$\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t} = 0 \tag{1.4}$$

This equation described a constraint line illustrated on the figure below (fig 1.3) :

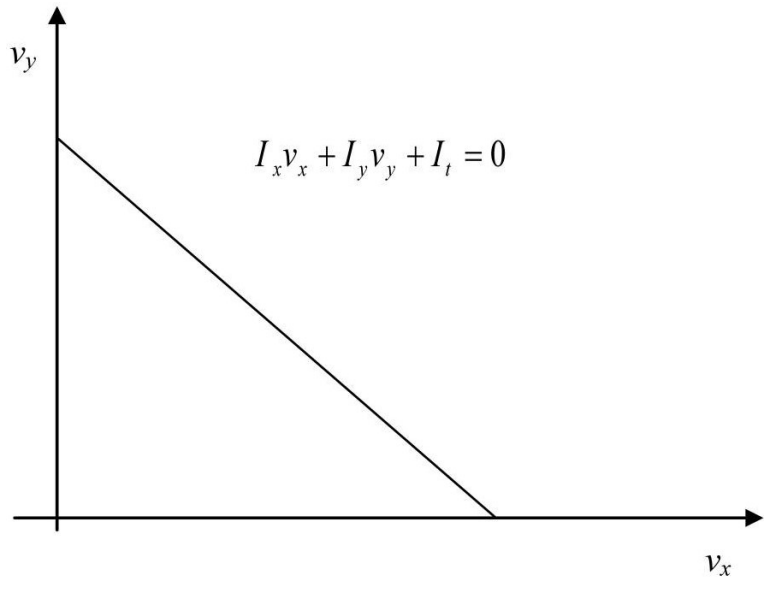

Figure 1.3: Optical Flow constraint Line

This problem cannot be solved as there are two unknowns in one equation, but if a small region is supposed to have the same velocity, the problem has a solution. The optical flow called Normal flow is then described as :

$$u_n = (u, v) = (\frac{-E_x E_t}{E_x^2 + E_y^2}, \frac{-E_y E_t}{E_x^2 + E_y^2})^T \tag{1.5}$$

In order to have better results, one could use more constraints and equations. But where do we get more equations ? That is what Bruce D. Lucas and Takeo Kanade have thought about.

1.2 Lucas-Kanade Method

In order to have more equations and take into account the different direction of the optical flow, the solution is to take a patch of pixels around the pixel of interest. Lucas and Kanade provide constraint by minimizing E over a local neighborhood.
For that, two assumptions have been made :

- Flow is locally smooth
- Neighboring pixels have same displacement

In matrix formulation, Using k*k pixels patch, it gives us m equations where it takes the following form :

$$\begin{bmatrix} I_x(p_1)I_y(p_1) \\ I_x(p_2)I_y(p_2) \\ .. \\ .. \\ .. \\ I_x(p_m)I_y(p_m) \end{bmatrix} \times \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} I_x(p_1) \\ I_x(p_1) \\ . \\ . \\ . \\ I_x(p_m) \end{bmatrix} \tag{1.6}$$

That gives us :

$$Ax = b \tag{1.7}$$

With a least square approximation, we finally obtain the solution expressed below :

$$\begin{bmatrix} \sum_{p\in P} I_x I_x & \sum_{p\in P} I_x I_y \\ \sum_{p\in P} I_y I_x & \sum_{p\in P} I_y I_y \end{bmatrix} \times \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum_{p\in P} I_x I_t \\ \sum_{p\in P} Iyx I_t \end{bmatrix} \tag{1.8}$$

where the summation is over each pixel p in a patch P.

In matrix form :

$$A^{\perp} A\hat{x} = A^{\perp} b \tag{1.9}$$

This method to compute Optical Flow works well when the flow is locally smooth or the neighboring pixels have same displacement especially when there are corners and we can get different gradients. On the other hand, the Lucas-Kanade approach is a local optimization problem that cannot perform properly if the object movements are too large since we are working with small patches of neighboors. The real object motion cannot extend beyond the considered region. It is called the aperture problem.

1.3 Aperture Problem

Each of the tuple (u,v) satisfying the condition in equation 1.4 are part of a straight line. For each point, the tuple (u,v) reprensents the normal flow, flow normal to the true flow. On the other hand, the parallel flow as such cannot be computed. In other word, since the problem owns only one constraints, the optical flow cannot be computed in more than one direction e.g. normal direction. It is called the aperture problem. If you look at the following figure (1.4, whatever the direction of the pannel, a black line will have the same normal optical flow for all three panels even if they have differents movements.

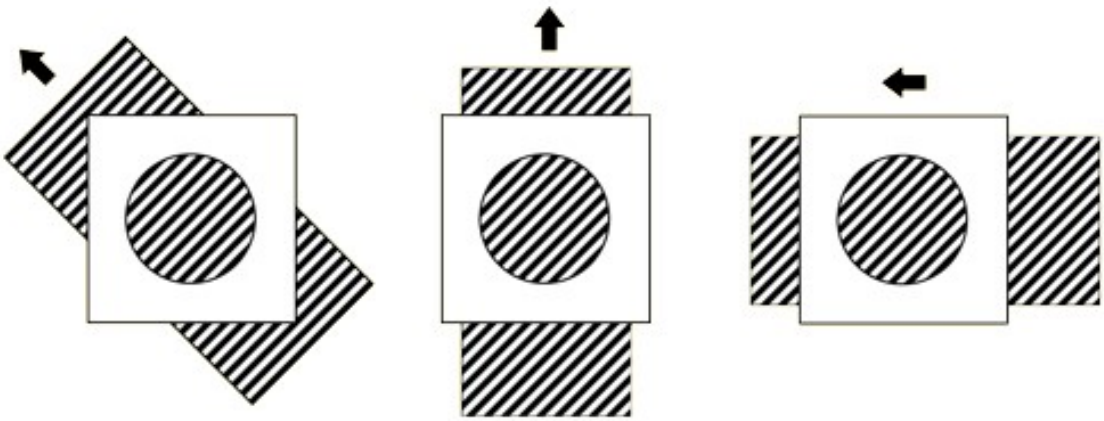

Figure 1.4: Aperture Problem

In order to solve this problem, and for others purposes aswell obviously, Horn–Schunck method has been invented.

1.4 Horn-Shunck Method

This method is different from Lucas-Kanade method in a way that it asks for a smooth flow instead of a constant flow. The constraint is no longer a flow over a small patch that should be the same but a flow of neighboring pixels that should be smooth. This method is a global method and design for dense optical flow.
The figure below 1.5 illustrates well the difference between the two constraints where the right image is more likely to happen. On the left, we have pacthes with a flow constant and a the right, smooth flow everywhere.

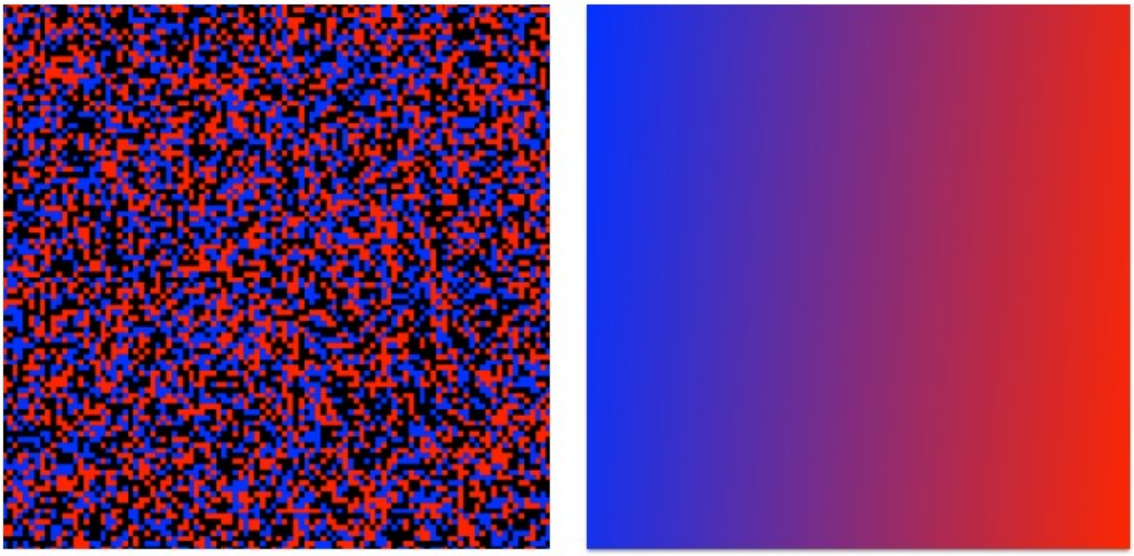

Figure 1.5: Constrained Flow Field

In order to derive the final Horn-Shunck Optical Flow, let's start from the initial objective function, the smoothness constraints :

$$E_d(i,j) = [\, I_x u_{ij} + I_y v_{ij} + I_t \,]^{\,2} \tag{1.10}$$

and the error in Optical Flow Constraint

$$E_s(i,j) = \frac{1}{4}[\,(\,u_{ij} - u_{i+1,j}\,)^{\,2} + (\,u_{ij} - u_{i,j+1}\,)^{\,2} + (\,v_{ij} - v_{i+1,j}\,)^{\,2} + (\,v_{ij} - v_{i+1,j}\,)^{\,2}] \tag{1.11}$$

We can constuct our minimization problem as as :

$$min \sum_{ij} E_d(i,j) + \lambda E_s(i,j) \tag{1.12}$$

By taking the partial derivatives of this whole sum, we get :

$$\frac{\partial E}{\partial u_{kl}} = 2(u_{kl} - \overline{u}_{kl}) + 2\lambda(I_x u_{kl} + I_y v_{kl} + I_t) I_x \tag{1.13}$$

$$\frac{\partial E}{\partial v_{kl}} = 2(v_{kl} - \overline{v}_{kl}) + 2\lambda(I_x u_{kl} + I_y v_{kl} + I_t) I_y \tag{1.14}$$

When we are setting the derivatives to zero and solve for unknowns u and v by re-arranging the terms, we get the following updated equations :

$$\hat{u}_{kl} = \overline{u}_{kl} - \frac{I_x \overline{u}_{kl} + I_y \overline{v}_{kl} + I_t}{1 + \lambda(I_x^2 + I_y^2)} I_x \tag{1.15}$$

$$\hat{v}_{kl} = \overline{v}_{kl} - \frac{I_x \overline{u}_{kl} + I_y \overline{v}_{kl} + I_t}{1 + \lambda(I_x^2 + I_y^2)} I_y \tag{1.16}$$

The Optical Flow Field is updated for each pixel and is iterated until it converges.

Chapter 2
Method Implementation

In this section will be written the implementation of both algorithms and the preprocessing step. The Python Language has been used.

2.1 Preprocessing Step

Lucas-kanade and Horn-SHunck methods both use the gradients in the X-direction and Y-direction of the images. This step consists of computing partial derivatives Ix , Iy , It between two frames. These partial derivatives are obtained by averaging the forward differences between them as described in the code below :

```
1     def computeFeature(self, image_path, index):
2         img_1 = cv2.imread(image_path[index-1], cv2....
      IMREAD_GRAYSCALE)
3         img_2 = cv2.imread(image_path[index], cv2.IMREAD_GRAYSCALE)
4         img1 = img_1#/255
5         img2 = img_2#/255
6         img_1_x = cv2.Sobel(img1, cv2.CV_64F, 1, 0, ksize=3, ...
      borderType=cv.BORDER_DEFAULT)
7         filter_x = np.transpose(np.array([[-1., -1.], [1., 1.]]))
8
9         img_1_y = cv2.Sobel(img1, cv2.CV_64F, 0, 1, ksize=3, ...
      borderType=cv.BORDER_DEFAULT)
10        img_2_x = cv2.Sobel(img2, cv2.CV_64F, 1, 0, ksize=3, ...
      borderType=cv.BORDER_DEFAULT)
11        img_2_y = cv2.Sobel(img2, cv2.CV_64F, 0, 1, ksize=3, ...
      borderType=cv.BORDER_DEFAULT)
12
13        img1 = cv2.GaussianBlur(img1, (5, 5), 0)
14        img2 = cv2.GaussianBlur(img2, (5, 5), 0)
15
16        kernel_t = np.array([[1., 1.], [1., 1.]])
17        Ix =  img_1_x + img_2_x
18        Iy = img_1_y + img_2_y
19
20        mode = 'same'
21        It = signal.convolve2d(img2, kernel_t, boundary='symm', ...
      mode=mode)+ signal.convolve2d(img1, - kernel_t, boundary='symm',...
       mode=mode)
22        return Ix, Iy, It, img_1, img1, img2
```

2.2 Lucas-Kanade Method

As previously said, Lucas-Kanade method computes the Optical Flow for each pixel over a patch of neighbooring pixels. Refering to equation 1.8, the tuple (u,v) will be isolated

and computed by matrix multiplications with patches of 9 pixels.

```
    print('Lucas Kanade Optical Flow')
    u = np.zeros(img1.shape)
    v = np.zeros(img1.shape)
    n = 1
    for i in range(1, u.shape[0]):
        for j in range(1, u.shape[1]):
            Ixpi = np.sum(np.power(Ix[i - n:i+n+1, j - n:j + n + ...
    1],2))
            Iypi = np.sum(np.power(Iy[i - n:i+n+1, j - n:j + n + ...
    1],2))
            IxIy = np.sum(np.multiply(Ix[i - n:i+n+1, j - n:j + n +...
     1], Iy[i - n:i+n+1, j - n:j + n + 1]))
            A = np.array([[Ixpi, IxIy],[IxIy,Iypi]])
            try:
                A_inv = np.linalg.inv(A)
            except:
                A_inv = A
            IxIt = -1*np.sum(np.multiply(Ix[i - n:i+n+1, j - n:j + ...
    n + 1], It[i - n:i+n+1, j - n:j + n + 1]))
            IyIt = -1*np.sum(np.multiply(Iy[i - n:i+n+1, j - n:j + ...
    n + 1], It[i - n:i+n+1, j - n:j + n + 1]))
            P = np.array([[IxIt],[IyIt]])
            uv = np.matmul(A_inv, P)
            u[i, j] = uv[0]
            v[i, j] = uv[1]
```

2.3 Horn-Shunck Method

Lastly, Horn-Shunck method has been implemented. Compared to Lucas-Kanade, the algorithm won't run for each pixels one after the other but for the entiere image at each iteration. As a note, this method uses two hyperparameters which are lamb, a weight and *max_iter* which is the maximum iteration number of our algorithm. The computation of u and v from equation 1.15 and 1.16 are then :

```
    Ix, Iy, It, img_1, img1, img2 = self.computeFeature(image_path,...
     index)

    avg_u,avg_v,u,v, iteration  = 0,0,0,0,0
    avg_u = u
    avg_v = v
    lamb = 10
    max_iter = 1
    while iteration < max_iter:
        u = avg_u - (Ix*avg_u + Iy*avg_v + It)*Ix/(1+ lamb*(Ix ** 2...
     + Iy ** 2))
        v = avg_v - (Ix*avg_u + Iy*avg_v + It)*Iy/(1+ lamb*(Ix ** 2...
     + Iy ** 2))
```

```
11          kernel = np.ones((3,3),np.float32)/9
12          avg_u = cv.filter2D(u,-1,kernel)
13          avg_v = cv.filter2D(v,-1,kernel)
14          iteration += 1
```

Chapter 3
Results and Discussion

The 6 frames used for this project are the following ones.

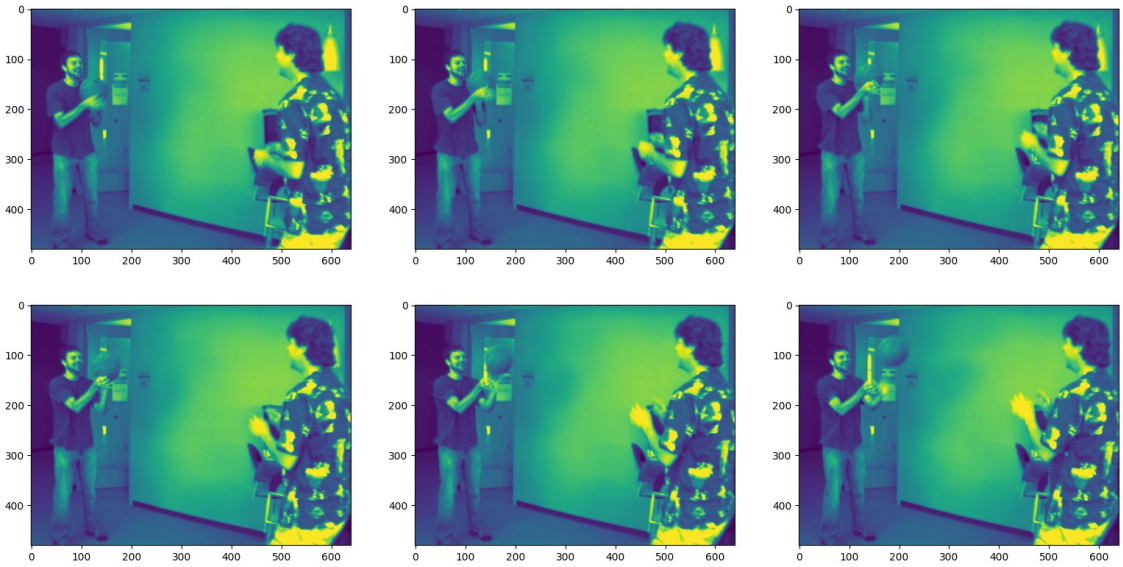

Figure 3.1: Input Images

3.1 Lucas-Kanade Method

In this section, you can find the result we got from the Lucas-Kanade method. Below are the components U and V amplitudes of the optical flow which are very high for all pixels. From those components alone, we cannot distinguish any flow whatsover, something more needs to be done.

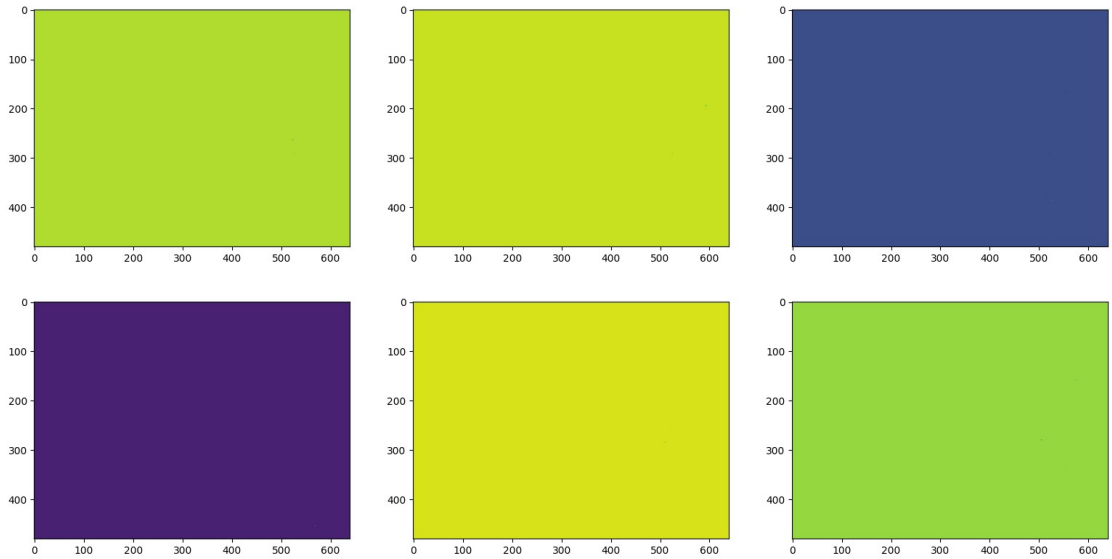

Figure 3.2: V component of Optical Flow

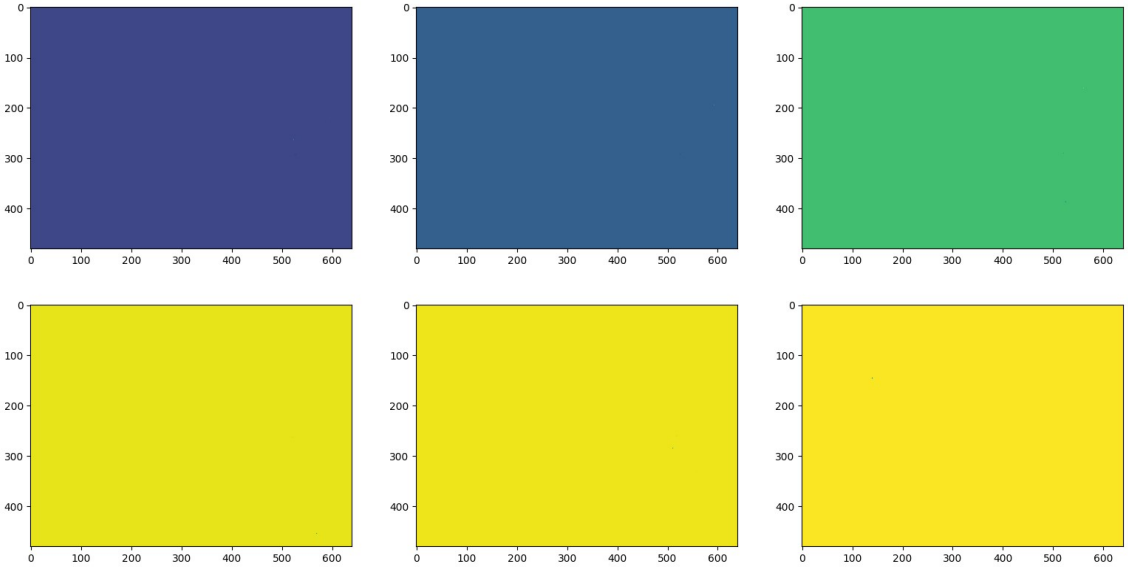

Figure 3.3: U component of Optical Flow

From those two componants, we can extract the real Optical Flow for different images/timestamp (Figure 3.4):

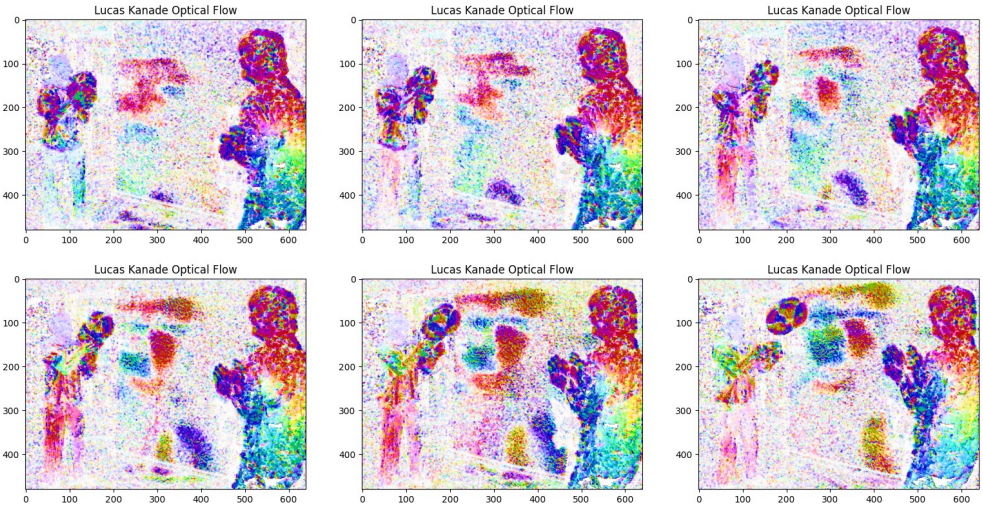

Figure 3.4: Final Optical Flow

We can clearly see that the motion fields are highlighted and have more colors than other region. One can also say that those images are very noisy. As an experience, the amplitudes of U andV have been divided by a weight parameter of 15:

```
u[i, j] = uv[0]/15
v[i, j] = uv[1]/15
```

The resulting U, V components have been improved and a kind of Optical Flow can be slighlty seen :

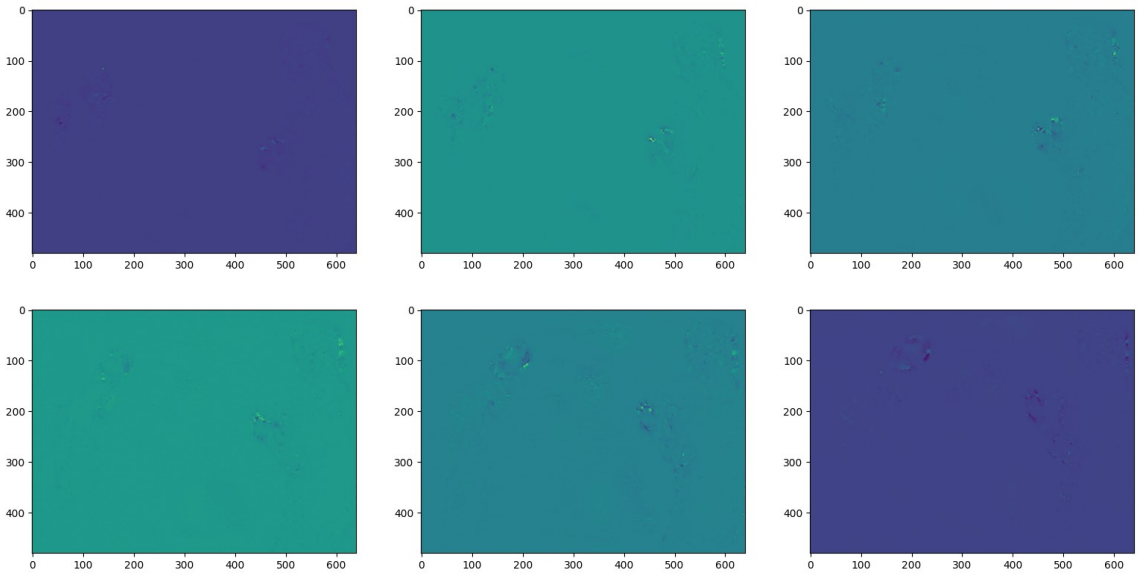

Figure 3.5: V component of Optical Flow with weight

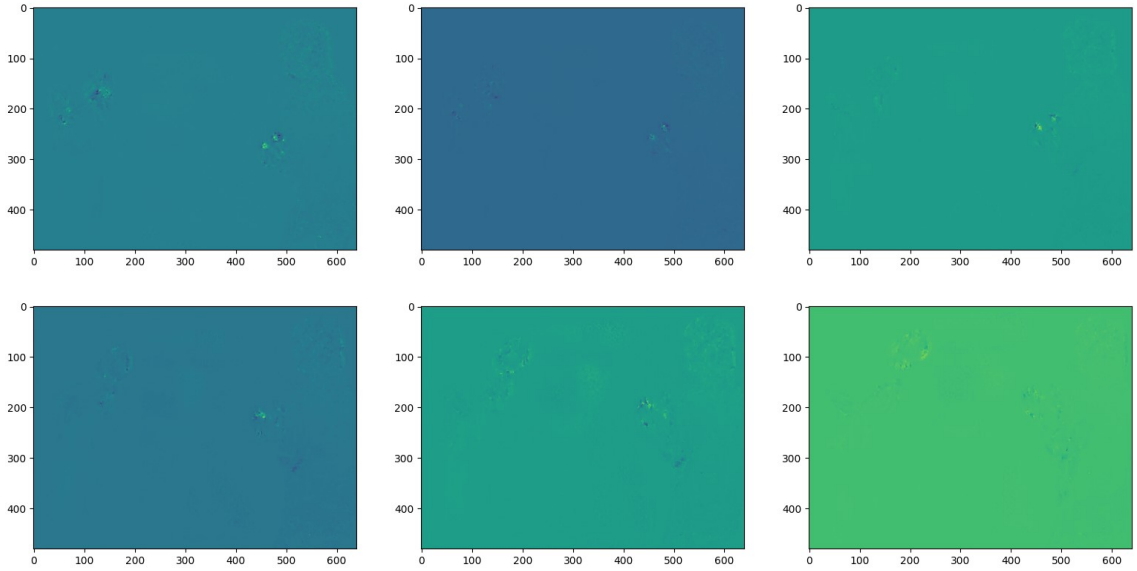

Figure 3.6: U component of Optical Flow with weight

On the other hand, the final Optical Flow is way less noisy and more accurate about what really move onto the images on the figure 3.7, especially the hands and the ball.

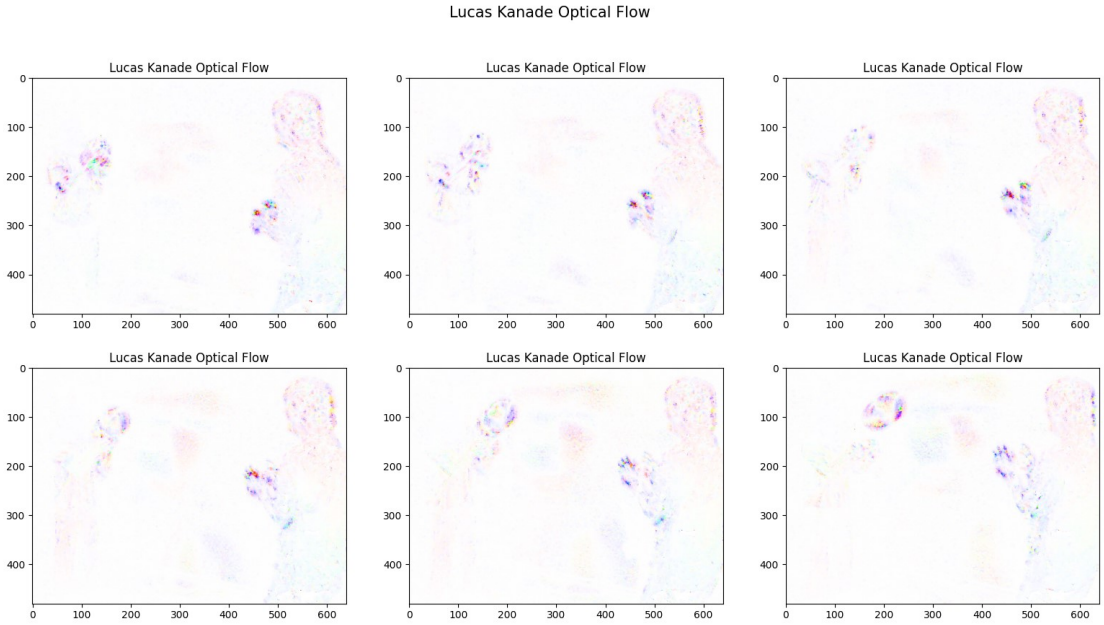

Figure 3.7: Final Optical Flow with weight

3.2 Horn-Shunck Method

The result from Horn-Shunck Method with lambda = 1 and $max\_iter = 1$ is displayed on figure 3.8 :

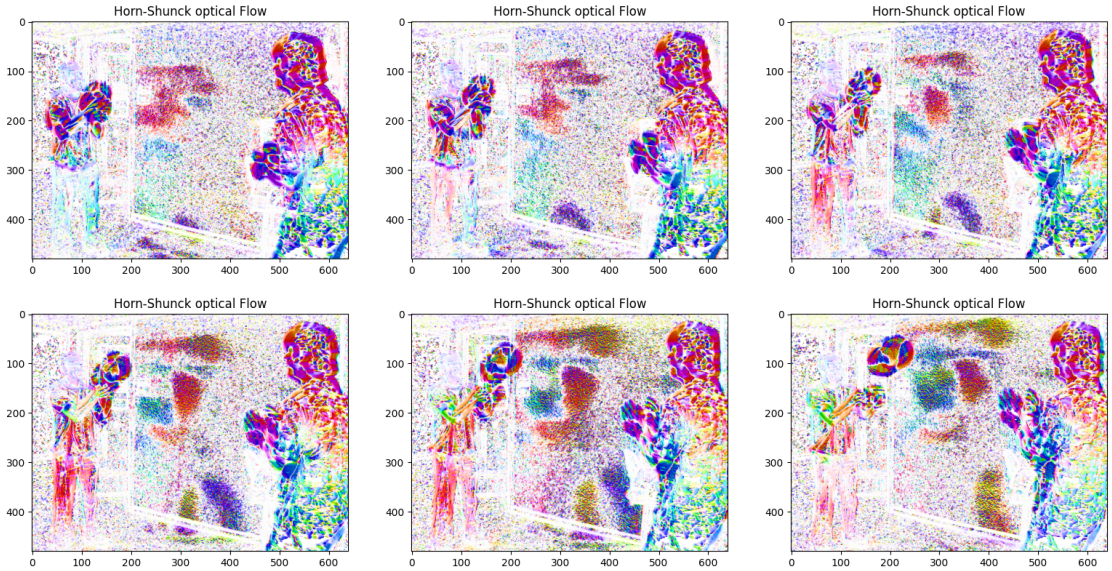

Figure 3.8: Final Optical Flow with weight

As in Lucas-Kanade results, the optical flow is well computed for those few images. The bigger the density of colorpoint is on the image, the bigger the motion field is which can be verified on the input images. But still, those optical flows are very noisy and even steady walls look like they are moving on figure 3.8. Hopefully, Horn-Shunck method owns two hyperparameters and have allowed us to tweak our computation of U and V and adjust our optical flow.

On figure 3.10, we can see that an increasing lambda leads to a more precise optical flow with a smaller density of colorpoints meaning this result is less noisy than the initial one. Altough a lambda too big would lead to a white image, in our case, a lambda in the range of 10-30 would be good.

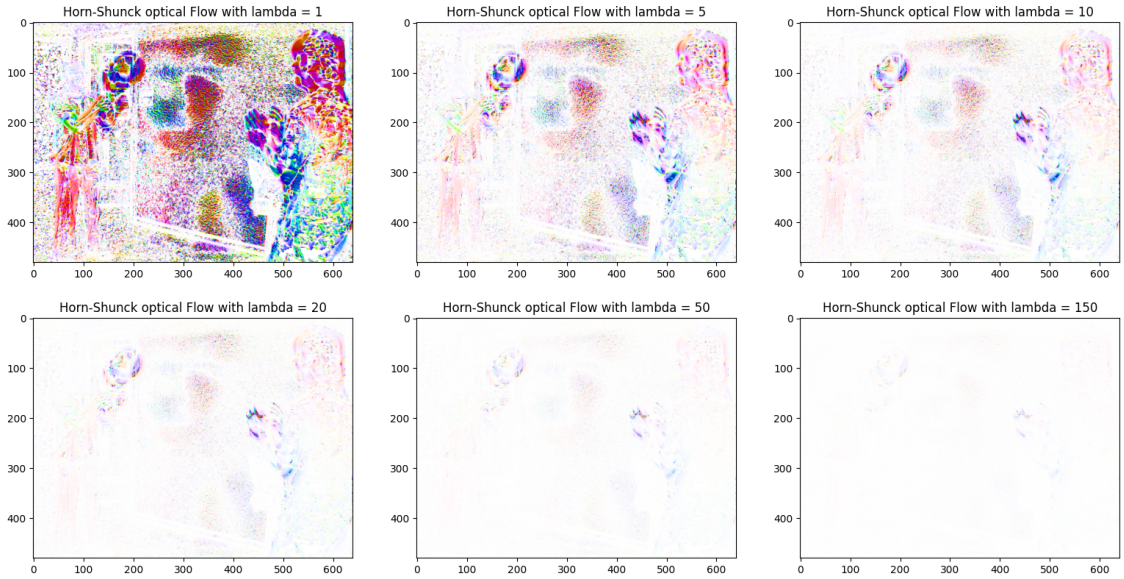

Figure 3.9: Final Optical Flow with weight

The other hyperparamater we could tweak is the number of iteration that would do the algorithm. As the number of iteration increases, the optical flow figure become less and less clear but become stronger. For this case, the perfect iteration would be 5.

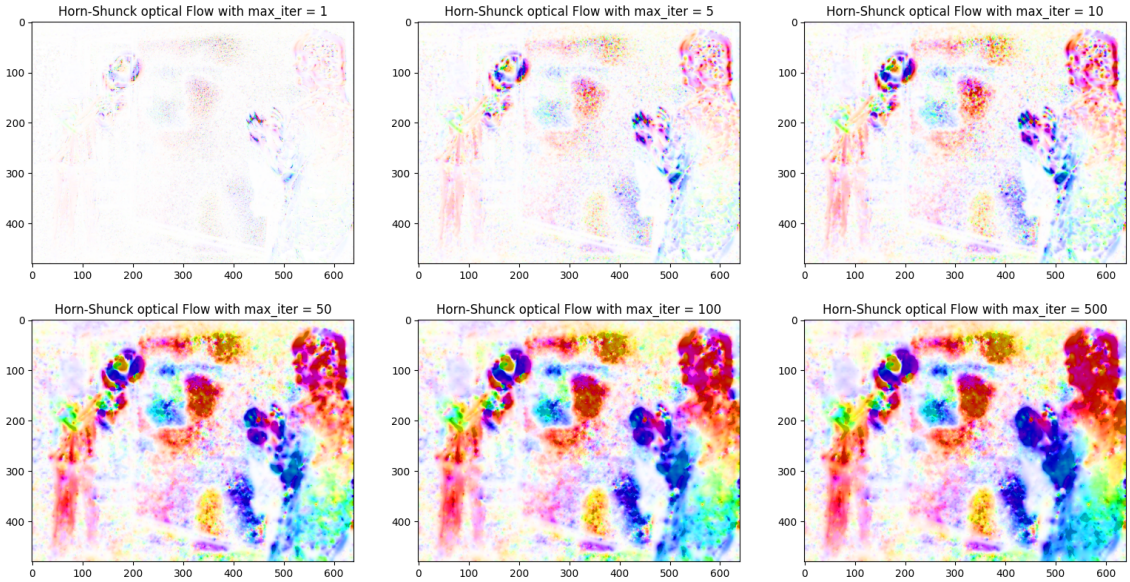

Figure 3.10: Final Optical Flow with weight

3.3 Other example

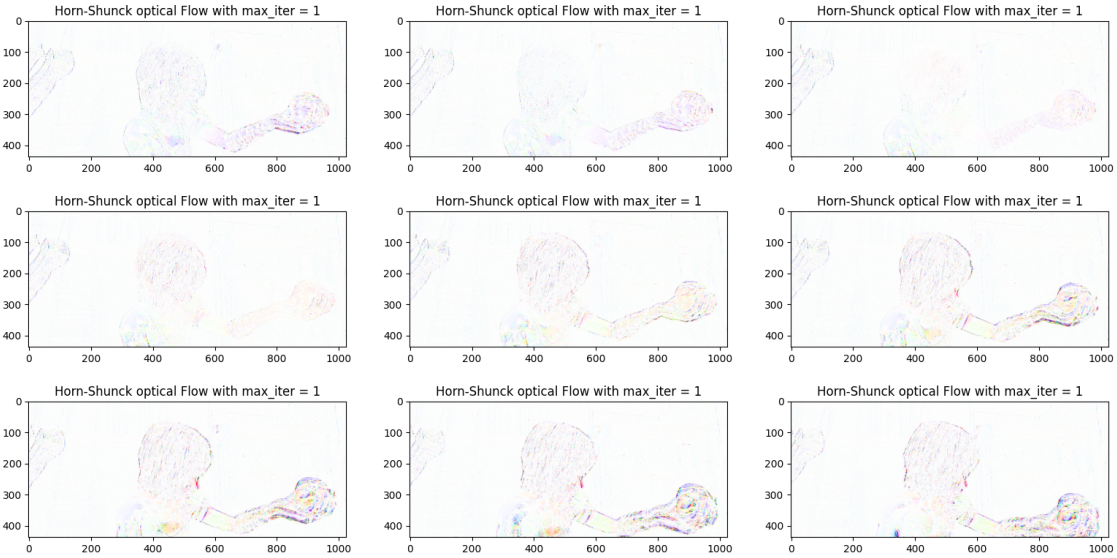

Figure 3.11: Horn-Shunck Optical Flow

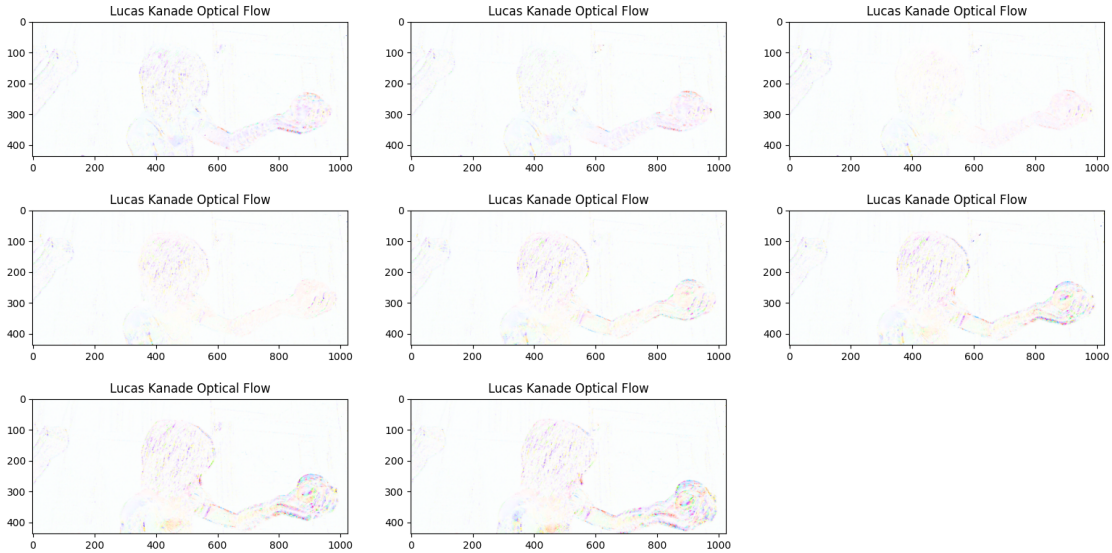

Figure 3.12: Kanade Optical Flow

3.4 Interpretation

Lucas-Kanade and Horn-Shuck methods are equivalent when it comes to small application like we did in this project. For the Lucas-Kanade method, we had to add a weight parameter to get smooth results. Some technique exist to improve the quality of the optical flow but havn't been implemented in this project. We could find a pyramidal alogrithm or add spatial gaussian filter.
On the other hand, Horn-Shunck method was sufficient to himself to get pretty good result at first try. Being able to change parameters such as the weight λ or the number of interation can be interesting in some cases.
λ can be used to improve the smoothness of the output and remove the noise and disturbance. λ cannot obviously be set too high or it will remove the relevant pixel as well as the noise but needs to be set high enough to keep only the dense optical flow. max_iter isn't as important as λ but could still help at improving the result and make a tradeoff between the runtime and the quality of the output. The more you increase max_iter, the better the optical flow is, but the longer the algorithm will run.
It is important to note that Lucas-Kanade method take much more time than Horn-Shunck method to complete for similar result. (With the implementation cited above).

Chapter 4
Deep Learning Method : The future ?

As previously said, Optical flow is the task of estimating per-pixel motion between video frames. It is a long-standing vision problem that remains unsolved. During many years, Optical Flow has been computed by hands with methods such as the two developed above.
Recently, deep learning has been shown as a promising alternative to traditional methods. Deep learning can side-step formulating an optimization problem and train a network to directly predict flow.
In order deep into AI computing Optical Flow on our set of images, the RAFT model which is well-known deep-learning model has been tested. RAFT stands for Recurrent All-Pairs Field Transform.

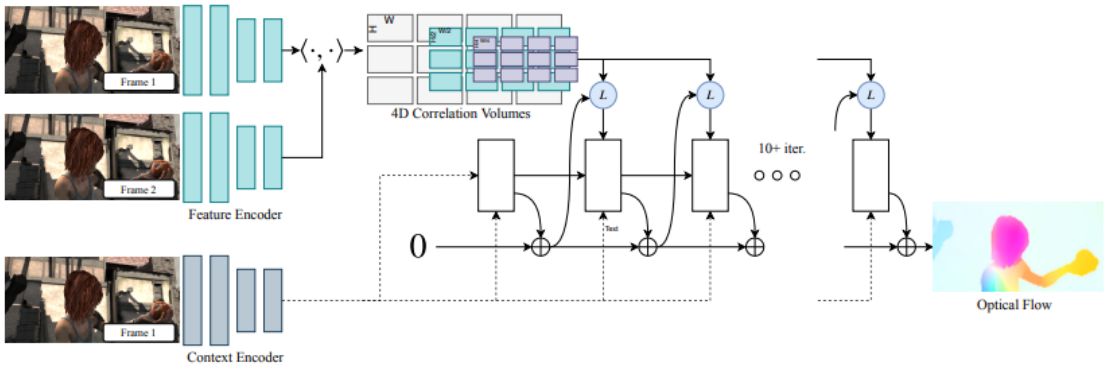

Figure 4.1: RAFT architecture

RAFT consists of three main components: (1) a feature encoder that extracts a feature vector for each pixel; (2) a correlation layer that produces a 4Dvcorrelation volume for all pairs of pixels, with subsequent pooling to produce lower resolution volumes; (3) a recurrent GRU-based update operator that retrieves values from the correlation volumes and iteratively updates a flow field initialized at zero.

The result given by RAFT on our set of images can be found on figure 4.2

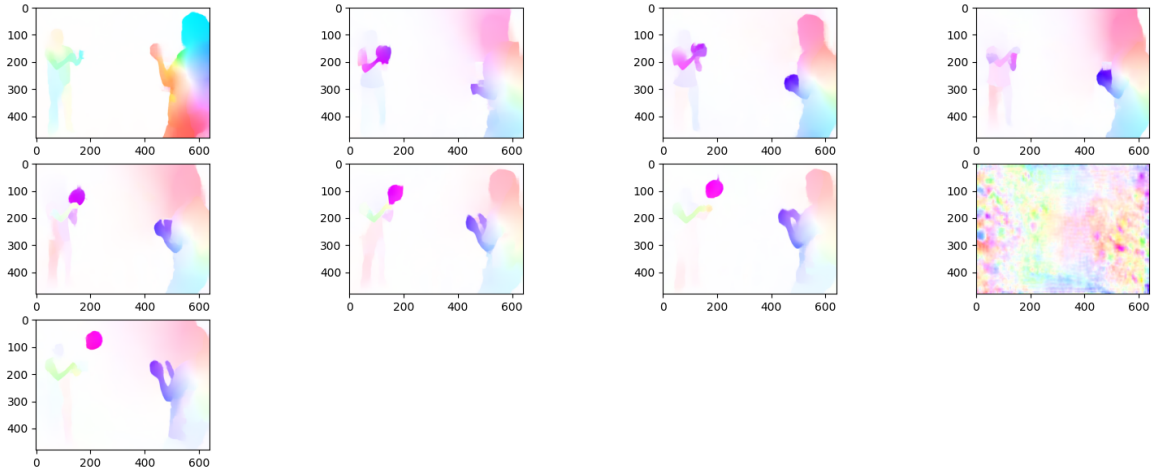

Figure 4.2: RAFT results

For more information on how to run RAFT, refer to the README.md in the ZIP file or in the official repository.

Chapter 5
Conclusion

The goal of this project was to implement hand-crafted methods to compute Optical Flow on a set of given images. Two methods have been tested : Lucas-kanade method and Horn-Shunck method. Both algorithms give similar results but Horn-Shunck allow for more flexibility thanks to his hyperparameters.
Additionnally, a deep learning architecture has been tested and achieve way better quality Optical Flow than simple techniques.

Bibliography

[1] Zachary Teed, Jia Deng *RAFT: Recurrent All-Pairs Field Transforms for Optical Flow*, arXiv preprint arXiv:2003.12039v3 (2020).

[2] Kitani Kris (2021), *Lucas-Kanade Optical Flow*, http://www.cs.cmu.edu/ 16385/s15/lectures/Lecture21.pdf

[3] Sahli Hichem (2022), *Optical Flow*, https://canvas.vub.be/

[4] Zvorișteanu, O.; Caraiman, S.; Manta, V.-I. *Speeding Up Semantic Instance Segmentation by Using Motion Information.* Mathematics 2022, 10, 2365. https://doi.org/10.3390/math10142365

[5] Sharmin N, Brad R. *Optimal Filter Estimation for Lucas-Kanade Optical Flow.* Sensors (Basel). 2012 Sep 17;12(9):12694–709. doi: 10.3390/s120912694. PMCID: PMC3478865.